Centro Federal de Educação Tecnológica de Minas Gerais
Campus Divinópolis
Graduação em Engenharia Mecatrônica

Túlio César Borges

Condução Colaborativa para Carros Autônomos Utilizando Sistemas Multi-Agentes

Divinópolis
2018

Túlio César Borges

Condução Colaborativa para Carros Autônomos Utilizando Sistemas Multi-Agentes

Monografia de Trabalho de Conclusão de Curso apresentada ao Colegiado de Graduação em Engenharia Mecatrônica como parte dos requisitos exigidos para a obtenção do título de Engenhario Mecatrônico.
Eixo de Formação: Computação e Controle.

Orientador: Prof. Dr. Luís Filipe Pereira Silva

Divinópolis
2018

Agradecimentos

Agradeço,

à meus Pais pelo amor, incentivo e o esforço valioso de dar sempre a melhor educação possível, dentro de casa e na escola.

ao meu irmão e minha irmã pelo apoio incondicional.

aos amigos que me acompanharam nessa caminhada e fizeram esses anos nessa Instituição muito mais agradáveis.

a todos os professores pelo conhecimento e sabedoria proporcionados.

ao Google pelo papel fundamental desempenhado na minha formação.

a todos que de alguma forma contribuíram com o meu progresso como aluno e como Ser.

Você tem que ser são para pensar claramente,
mas pode pensar profundamente e ser insano.

Nikola Tesla

Resumo

Carros inteligentes estão se tornando uma realidade. Grandes empresas, como Google, Apple e Uber, possuem linhas de pesquisas direcionadas nessa área. A Tesla, inclusive, já lançou uma linha de carros comerciais autônomos. No entanto, essas tecnologias esbarram em questões legais que tratam sobre a confiabilidade do sistema. A condução colaborativa é uma alternativa que vem sendo estudada e pode ser a solução para que os carros autônomos se tornem mais seguros. Sistemas Multi-Agente é uma área da Inteligência Computacional preocupada com o desenvolvimento de soluções distribuídas para problemas complexos, e vem sendo utilizada recentemente com sucesso em diversos estudos que tratam da comunicação entre entidades distintas. A proposta de trabalho aqui exposta objetiva aplicar técnicas de sistemas multi-agente em um problema de condução colaborativa focado em uma situação cotidiana de trânsito, em que os veículos devem cooperar entre si para resolver esse problema local. Este projeto engloba duas áreas de interesse; controle e computação. Tendo isso em vista, é proposto para este projeto a revisão da literatura existente sobre dinâmica veicular e sobre agentes, modelagem matemática de um carro e subsequente controle desse, desenvolvimento do algoritmo dos agentes e aplicação desses algoritmos em um modelo matemático de um cruzamento não sinalizado. Este projeto apresenta, então, o diferencial de aplicar técnicas de inteligência computacional na resolução de problemas, algo que não é abordado no curso e que promete uma simplificação significante desses problemas.

Palavras-chave: Controle, Inteligência computacional, Sistemas multi-agente.

Abstract

Intelligent cars are becoming a reality. Large companies, e.g. Google, Apple and Uber, do have research lines focused on it. Tesla have had, since October 2016, a commercial line of autonomous vehicles. These technologies, however, stumble on legal issues concerning the reliability of the autonomous systems. Collaborative driving is an alternative that is being studied on the academic environment and can be a solution so that autonomous vehicles become safer. Multiagent Systems is a subfield of Computational Intelligence that provides tools for the development of distributed solutions for complex problems and is being used with success in several studies concerning communication between different parties. This work proposal's goal is to implement multiagent system techniques on a collaborative driving problem focused on an everyday traffic situation, in which the vehicles must collaborate to solve this local problem. This work contemplates two fields of interest; control and computer science. It is proposed for this project a review of the existing literature on vehicle dynamics and agent systems, the modeling of the steering dynamics of a car and development of a control system for this model, development and implementation of an agent algorithm on a virtual model of a traffic intersection without traffic lights. This project differentiates from other in using Computational Intelligence to solve problems, something which is not approached in the mechatronics course and promises a significant simplification of these problems.

Key-words: Control. Computational Intelligence. Multiagent Systems.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Quadros

Lista de Acrônimos e Notação

NHTSA	*National Highway Traffic Safety Administration*
VIAC	*VisLab Intercontinental Autonomous Challenge*
PID	Proporcional Integral Derivativo
MAVS	*Multi-Agent & Visualization Systems*
MATISSE	*Multi-Agent based TraffIc Safety Simulation systEm*
GM	*General Motors*
RCA	*Radio Corporation of America*
CMU	*Carnegie Mellon University*
IC	Inteligência Computacional
IAD	Inteligência Artificial Distribuída
SDP	Solução Distribuída de Problemas
SMA	Sistema Multi-Agente
SAU	Sistema de Agente Único
FIPA	*Foundation for Intelligent Physical Agents*
IEEE	*Institute of Electrical and Electronics Engineers*
AID	*Agent IDentification*
GUID	*Globally Unique IDentification*
HAP	*Home Agent Platform*
AMS	*Agent Management System*
DF	*Directory Facilitator*
MTS	*Message Transport System*
ACL	*Agent Communication Language*
JADE	*JAVA Agent DEvelopment Framework*

$\mathbf{r}, \mathbf{F}$	Vetores físicos são representados por letras em negrito caixa baixa. Tensores e matrizes por letras em negrito caixa alta.
$\dot{\mathbf{r}}, \ddot{\mathbf{r}}$	Pontos sobre uma variável representam as derivadas temporais dessa.
$\mathbf{r}_v$	O índice inferior direita representa o corpo ou sistema de coordenada da quantidade representada.
${}_h\mathbf{r}_v$	O índice inferior esquerdo representa o ponto, corpo ou sistema de coordenadas de referência da quantidade representada. Se o campo estiver vazio entende-se o sistema de coordenadas inercial como referência.
${}^V_V\boldsymbol{r}_v$	O índice superior esquerdo indica o sistema de coordenadas no qual a quantidade indicada está representado. Se o campo estiver vazio entende-se que a quantidade é um vetor físico.

Capítulo 1

Introdução

Desde os anos de 1920 a aplicação de técnicas de controle foram estudadas para promover melhorias de desempenho, conforto e segurança de veículos. Os resultados desses estudos podem ser vistos em muitas tecnologias, que hoje são comuns no vocabulário até dos mais leigos quanto a esse assunto. Alguns exemplos são; controle ativo de suspensão, controle de estabilidade e assistente de direção.

Nos anos mais recentes o foco das pesquisas na área da automação veicular se encontra em obter veículos totalmente autônomos, fazendo uso das tecnologias citadas anteriormente. Diversas grandes empresas se envolveram nesses estudos, como por exemplo a Google [Markoff 2010], a Apple [Taylor 2016] e a Uber [Hawkins 2016], produzindo protótipos autônomos. Já a Tesla [The Tesla Team 2016] vem comercializando todos os seus carros produzidos desde Outubro de 2016 com a funcionalidade de auto-pilotagem desbloqueada.

O grande problema que envolve essa área é a segurança. Esses sistemas de auto-pilotagem ainda não tem garantia de segurança suficiente para poderem tomar controle de um carro sem supervisão humana. Muitos estados nos EUA promulgaram legislações a respeito do tráfego de veículos autônomos, que podem ser conferidas em NCSL 2017, exigindo supervisão humana durante a execução desses sistemas. Apesar de parecer que há um esforço contrário por parte dos governos em relação a essas tecnologias, a NHTSA (*National Highway Traffic Safety Administration*) publicou em setembro de 2016 um guia incentivando o desenvolvimento e testes de veículos autônomos [NHTSA 2016].

O primeiro acidente fatal envolvendo carros autônomos ocorreu em maio de 2016, quando o Tesla, em modo de auto-pilotagem, de Joshua Brown bateu na lateral de uma carreta, levando-o ao óbito. Tal fato colocou em duvida a efetividade desses sistemas. Investigações pela NHTSA não encontraram uma tendência defeituosa com a tecnologia e portanto não justificando a continuação das investigações. Além disso, uma estatística feita pela NHTSA aponta que os carros da Tesla em modo de auto-pilotagem se envolveram em 40% menos acidentes do que aqueles pilotados por pessoas. "We want to promote these technologies,they will save lives and cut crashes dramatically, but innovation is a bumpy road."[1] disse Bryan Thomas, diretor de comunicações da NHTSA [Stewart 2017].

[1]"Nós queremos promover essas tecnologias, elas vão salvar vidas e diminuir drasticamente o numero de acidentes, porém a inovação é ua estrada esburacada.", tradução livre.

Como apontado em Lin e Maxemchuk 2012, a próxima geração de carros autônomos tende a usar métodos de comunicação para melhorar a segurança dessas tecnologias, compartilhando leituras de sensores, informações de operação ou até mesmo colaborando entre si para alcançar um objetivo. Nos últimos anos, muitos estudos estiveram focados na direção colaborativa podendo ser esse o fator decisivo para a entrada definitiva dos veículos autônomos no mercado, simbolizando assim uma redução nos acidentes de trânsito, congestionamentos urbanos e rodoviários, e melhor utilização das vias públicas.

Nota-se, porém, que apesar de haver um grande investimento na pesquisa e produção de carros autônomos por parte das empresas citadas anteriormente, as pesquisas no ramo de condução colaborativa para veículos autônomos permanecem a princípio concentradas, ou somente divulgadas, no meio acadêmico. Um dos possíveis motivos para esse fato é que as tecnologias envolvidas podem se tratar de segredos industriais e, portanto, não divulgados abertamente pelas empresas.

Dadas as informações apresentadas acima percebe-se a relevância de se fazer um estudo sobre direção colaborativa. Neste trabalho será aplicado conceitos de sistemas multi-agente ao problema da direção colaborativa em uma situação cotidiana, tal como um cruzamento não sinalizado, e analisado o comportamento da técnica utilizada.

1.1 Definição do Problema

Pode-se perceber nos sistemas de carros autônomos comerciais atuais que estes operam de forma independente dos outros veículos. Não foi identificado, até então, ações das grandes empresas envolvidas nas pesquisas de carros autônomos com relação ao estudo e desenvolvimento de sistemas colaborativos. Esse fato pode ser devido a possíveis segredos industriais por parte dessas. A divulgação de pesquisas sobre sistemas colaborativos têm-se mantido restrita ao meio acadêmico até o presente momento. Porém até neste meio os trabalhos publicados sobre condução colaborativa são escassos, gerando uma baixa diversidades de metodologias e topologias propostas para trabalhar com sistemas multi-agente em situações de condução colaborativa.

1.2 Motivação

A motivação principal da realização deste trabalho é a razão social. A condução colaborativa é uma tecnologia ainda em estudo e desenvolvimento, e que poderá ser uma solução para a elevada quantidade de acidentes e congestionamentos que pode-se observar hoje em dia. A solução desses problemas proporcionará uma diminuição do número de fatalidades no trânsito, bem como uma redução de engarrafamentos.

1.3 Objetivos de pesquisa

São descritos aqui o objetivo geral do projeto de pesquisa assim como os objetivos específicos. Espera-se que com a realização dos objetivos específicos, o objetivo geral seja alcançado.

1.3.1 Objetivo Geral

Aplicar os conceitos de agentes ao problema da condução colaborativa, propondo uma topologia de projeto de sistemas multi-agente para as malhas de trânsito urbanas.

1.3.2 Objetivos Específicos

Estes a seguir são os objetivos específicos desta pesquisa:

- Adquirir conhecimentos sobre a dinâmica de direção de carros e desenvolver um modelo matemático representativo dessa;
- Propor um sistema de controle para o modelo do carro, bem como uma plataforma de testes;
- Entender as metodologias de desenvolvimento orientado a agentes e técnicas de inteligência computacional;
- Desenvolver os algoritmos dos agentes para a situação proposta; e
- Testar os algoritmos desenvolvidos ao modelo controlado, observando o comportamento desses na situação proposta.

1.4 Organização do Documento

O restante deste trabalho esta dividido em quatro capítulos mais um apêndice. No segundo capítulo discute-se sobre os principais acontecimentos envolvendo os assuntos que compõe este trabalho. Posteriormente são tratados os conceitos necessários para o bom entendimento deste texto.

No terceiro capítulo é mostrada a formulação final dos modelos que serão utilizados no restante do trabalho e apresentadas as simulações realizadas em cima destes modelos.

No quarto capítulo são apresentados a topologia proposta para projeto de sistemas multi-agente aplicado à condução colaborativa e os passos realizados no desenvolvimento dos códigos.

Por fim no quinto capítulo é realizada uma conclusão sobre o trabalho, bem como uma discussão de propostas de melhorias e continuidade para este trabalho.

O apêndice contem todos os códigos utilizados nas simulações do sistema-multi-agente.

Capítulo 2

Revisão Bibliográfica

2.1 Estado da Arte

Nos últimos anos, tem-se percebido um aumento considerável no desenvolvimento de projetos de carros autônomos. Em 2010, o carro autônomo Audi TTS percorreu o trajeto da montanha Pikes Peak, Colorado, EUA, de 20 km em 27 minutos [Funke et al. 2012], sendo que o recorde humano nesse trajeto era de 17 minutos. Esse acontecimento rompeu uma barreira importante, pois, pela primeira vez na história dos veículos autônomos, foi mostrado o quão próximo carros auto-pilotados estão dos melhores pilotos humanos.

Figura 2.1: Trajeto percorrido durante a VIAC. Fonte: M. Bertozzi et al. 2011

Figura 2.2: Carro usado na VIAC. Fonte: M. Bertozzi et al. 2011

Os desenvolvimentos das tecnologias nessa área possibilitaram o acontecimento de competições de carros autônomos, das quais a VIAC (VisLab Intercontinental Autonomous Challenge), também em 2010, foi a mais importante [M. Bertozzi et al. 2011; A. Broggi et al. 2010]. Essa competição consistiu em uma viagem de 13.000 km, de Parma, Itália, até Shanghai, China, conforme mostrado na Figura 2.1, sendo que ela foi composta por quatro minivans, como mostra a Figura 2.2, carregados com bens e com influencia humana desprezável.

Vale destacar que diversas grandes empresas também estão realizando estudos nessa área, como, por exemplo, a Google [Markoff 2010], a Apple [Taylor 2016] e a Uber [Hawkins 2016]. Já a Tesla passou a comercializar todos os seus carros produzidos desde Outubro de 2016 com o *hardware* necessário para as tecnologias de auto-pilotagem, como mostra a Figura 2.3 [The Tesla Team 2016]. No entanto, apesar do investimento em pesquisa e produção de carros autônomos pelas empresas supracitadas, nota-se as pesquisas no ramo de condução colaborativa para veículos autônomos permanecem concentradas no meio acadêmico.

Figura 2.3: Model S, um dos modelos da Tesla já produzidos com sistema de auto-pilotagem. Fonte: `https://www.tesla.com/`

Em 2009, Xavier e Pan propuseram um esquema prático e simples de um controlador PID (Proporcional Integral Derivativo) para execução de uma ultrapassagem em uma pista reta, obtendo sucesso em simulações. Na situação proposta, grupos de 3 carros colaboram para que a manobra seja executada e mantendo a distância entre os veículos.

Também em 2009,, então pesquisadores no laboratório de Sistemas de Visualização e Multi-Agentes (MAVS)[1] na Universidade do Texas em Dallas, apresentaram a primeira versão do MATISSE[2]. O MATISSE é um aplicativo desenvolvido para auxiliar no estudo de Sistemas Multi-Agente aplicado à controle de tráfego, que incorpora simulação completa de dinâmica veicular e um ambiente de visualização 3D e 2D. Duas outras versões do MATISSE foram desenvolvidas [Torabi, Al-Zinati e R. Z. Wenkstern 2018; Al-Zinati e R. Wenkstern 2015], e uma imagem da versão 3.0 pode ser vista na Figura 2.4

Motivado pela crescente complexidade que os protocolos de sistemas de condução colaborativa estavam exibindo, Lin e Maxemchuk propuseram em 2012 uma nova arquitetura para esses sistemas. Essa arquitetura visava reduzir a complexidade dos protocolos de condução colaborativa utilizados de forma a não sobrecarregar os sistemas de comunicação existentes.

Já em 2014, Acikmese e Bayard aplicaram o método *probabilistic guidance*[3] à um caso colaborativo de *swarm robots*[4]. Nesse estudo membros de sub-grupos do *swarm* colaboram para definir o objetivo comum do seu grupo.

2.2 Revisão de Literatura

Os carros autônomos vem sendo estudados desde o início do século 20, quando, em 1925, um carro controlado a rádio foi demonstrado em Nova Iorque. O Linrrican Wonder, como foi chamado, era um carro equipado com uma antena transmissora e era operado por um segundo

[1] do inglês, Multi-Agent & Visualization Systems

[2] Multi-Agent based TraffIc Safety Simulation systEm, tradução livre: Sistema de Simulação de Segurança no Transito baseado em Multi-Agente

[3] Direção probabilística, tradução livre.

[4] Grupo de unidades robóticas simples que atuam em conjunto para desempenhar tarefas complexas.

Figura 2.4: Visualização 3D de uma simulação no MATISSE 3.0. Fonte: [Torabi, Al-Zinati e R. Z. Wenkstern 2018]

veículo que o seguia transmitindo impulsos de rádio, os quais eram então enviados a disjuntores que operavam pequenos motores que direcionavam o carro [Bimbraw 2015].

Em 1939, a exibição Futurama de Norman Bel Geddes, patrocinada pela GM (General Motors), contava com carros elétricos controlados também via rádio por circuitos embutidos na pista [Bimbraw 2015].

Foi produzido o primeiro progresso desde o Linrrican Wonder em 1953, na RCA Labs (Radio Corporation of America): um carro miniatura guiado e controlado por cabos espalhados pelo chão do laboratório. Leland Hancock e L. N. Ress levaram a ideia ao nível dos carros comerciais, testando o sistema em uma rodovia nas margens de Lincoln, Nebraska, em 1958 [Bimbraw 2015].

Durante a década de 1960, o Transport and Road Reshearch Laboratory [5] do Reino Unido testou um carro auto-pilotado, modelo Citroen DS, que era guiado por cabos magnéticos enterrados na pista. O carro conseguia trafegar pela pista de testes a 130 km/h sem desviar do caminho nem variar a velocidade [Cardew 1970].

Em 1995, o projeto do Navlab, CMU (Carnegie Mellon University), alcançou semi autonomia em uma viagem de 5.000 km através dos Estados Unidos. O carro usava redes neurais para controlar o volante, porém os pedais de aceleração e frenagem eram controlados por pessoas [Pomerleau 1993; Thorpe et al. 1991].

Em 1998, surgiu o projeto de Alberto Broggi, da Universidade de Parma, ARGO, que consistia em fazer um Lancia Thema seguir as marcas das autopistas. O projeto alcançou sucesso ao fazer uma viagem de 1.900 km pelo norte da Itália com velocidade média de 90 km/h. O veículo possuía duas câmeras de baixo custo e fez 94% do percurso em modo automático [Alberto Broggi, Massimo Bertozzi e Fascioli 2000].

Os sistemas de automação veicular progrediram então até o ponto em que estes conseguem detectar o comportamento de veículos adjacentes e reagirem a esta informação. A próxima

[5] Laboratório de Pesquisa em Estradas e Transportes, tradução livre.

geração de veículos autônomos progrediria então para a comunicação entre veículos próximos e compartilhamento de informações de operação e leitura de sensores internos [Lin e Maxemchuk 2012].

Apesar do protocolo IEEE 802.11 ter sido lançado em 1997 [Crow et al. 1997], possibilitando comunicação sem fio local padronizada, não houve ação de grandes empresas voltadas a pesquisas na área de condução colaborativa para veículos autônomos. As pesquisas nessa área concentraram-se no meio acadêmico, sendo [Gaubert et al. 2003] e [Halle, Laumonier e Chaib-Draa 2004] alguns dos primeiros trabalhos realizados nesta área.

Além da comunicação entre veículos há necessidade também de implementar sistemas capazes de fazer bom uso das informações obtidas por esse meio. A IC (Inteligência Computacional) é um ramo da ciência da computação que possui técnicas diversas para fazer tratamento de dados e atuar sobre estes de forma a completar objetivos pré-estabelecidos [Russell e Norvig 2010].

Pôde-se perceber nas ultimas décadas uma grande adoção de técnicas da IC para a solução de problemas em diversas áreas do conhecimento. As pesquisas em automação veicular também estão adotando essas técnicas para diversos fins, como por exemplo: obter controladores com capacidade de aprendizado, fazer reconhecimento de imagens da pista, modelar o comportamento de motoristas, entre outros [Khan e Parker 2016; Li, Chen e Lim 2010; Musoles 2016; Yan et al. 2016].

Um conceito de IC que ganha cada vez mais atenção da comunidade acadêmica em geral é o de Sistema Multi-Agente. Esse sistema possui algumas características que fazem dele uma opção interessante na solução de muitos problemas, sendo algumas dessas: a robustez garantida em implementações com agentes redundantes, velocidade de operação devido ao paralelismo, além disso a modularidade desses sistemas os torna escaláveis e pode levar a uma codificação mais fácil [Peter Stone e Veloso 2000].

Esses sistemas vêm sendo usados em inúmeros casos. Em Decker 1996 foi realizada uma série de estudos de casos sobre o uso de sistemas multi-agentes em diversas situações, tais como no agendamento de recursos em um hospital e coleta de informações na internet. Já em Ansola et al. 2014 foi feito um estudo na otimização do agendamento de recursos no *Ground Handling Operations*[6] de um aeroporto.

Como feito em Dresner e P. Stone 2004 e Sabetghadam et al. 2012, sistemas multi-agente podem ser utilizados para controle e gestão de tráfego urbano. Já em Kristensen e Smith 2015 foi realizado um estudo sobre a utilização de multi-agentes na simulação de uma interseção de trânsito com e sem semáforo, utilizando um algoritmo de seleção para sequenciar os veículos na ultima.

Em Liu 2017a,b vem sendo realizados estudos no uso de sistemas multi-agente em problemas de controle preditivo. Já em Ranjbar-Sahraei et al. 2012 foram utilizadas técnicas de controle *fuzzy* e multiagentes em conjunto para controle de *swarm robots*.

Levando em consideração os trabalhos citados acima, pode-se perceber que, tanto a automação veicular quanto a IC, são de vital importância para os atuais ambientes econômico e acadêmico, e o quão fortemente interligados esses assuntos estão.

[6]Termo utilizado para designar o conjunto de operações de apoio às aeronaves e passageiros, assim como movimentação de bagagens, cargas e correios.

2.3 Metodologia

Para este trabalho foram primeiramente revidados alguns conceitos importantes que já haviam sido estudados anteriormente durante o curso. Esses assuntos foram discutidos nas seções 2.4.1, 2.4.2.1 e 2.4.3. Além disso foram estudados assuntos adicionais que se fizeram necessários para entender e adaptar o modelo encontrado em Schramm, Hiller e Bardini 2014; esses estudos foram sumarizados na seção 2.4.2.2. Foi também feito um estudo sobre Inteligência Computacional, Sistemas Multi-Agente e as normatizações FIPA, que foram aplicados no desenvolvimento do trabalho.

Além disso foram feitas simulações em malha aberta com o modelo que será utilizado durante o desenvolvimento do trabalho. A formulação final do modelo e as simulações podem ser encontradas no Capítulo 3. Neste capítulo encontram-se também o desenvolvimento de um modelo de velocidade de cruzeiro e os testes realizados com este modelo.

Também foi realizado o desenvolvimento de um Sistema Multi-Agente e uma plataforma de testes, ambos descritos no Capítulo 4. Os testes e resultados obtidos com esse sistema também pode ser visualizado no mesmo capítulo.

2.4 Fundamentação Teórica

São apresentados a seguir conceitos de temas que se mostraram relevantes para o desenvolvimento do presente trabalho, contribuindo desta forma para o entendimento do mesmo.

2.4.1 Modelagem Matemática de Sistemas

A modelagem matemática da dinâmica de processos é uma parte importante quando se trata de sistemas de controle. Um modelo matemático é definido como um grupo de equações que representam, com certa precisão, a dinâmica de um processo, seja ele físico, químico, etc.

O modelo de um sistema pode ser obtido por diversos métodos. Alguns desses é a modelagem a partir das leis físicas que governam a dinâmica do processo estudado, como as Leis de Newton para os sistemas mecânicos ou as Leis da Termodinâmica para os térmicos [Ogata 1982], e a identificação de sistemas.

2.4.1.1 Sistemas não lineares

Apesar de muitos sistemas físicos serem representados de forma linear, na prática a maioria desses são na verdade não lineares. E mesmo aqueles que são considerados lineares, são tal apenas em determinadas faixas de operação. Um sistema não linear pode ser considerado como tal se o princípio da superposição não se aplica a ele. Esse diz que para um sistema linear, a resposta à múltiplas entradas pode ser calculada resolvendo o problema para cada entrada separadamente e posteriormente combinando os resultados [Ogata 1982].

Existem diferentes tipos de não linearidades que podem existir entre as variáveis de um processo. Alguns exemplos dessas são a saturação, as zonas-mortas e os termos não lineares (senoidais, quadráticos, etc.).

2.4.1.2 Equações de Estado

Um modo de se representar um modelo matemático é por meio das equações de estado, sendo estas dadas, genericamente, como mostra a equação 2.1:

$$\begin{aligned} \dot{x}_1 &= f_1(t, x_1, \ldots, x_n, u_1, \ldots, u_p) \\ \dot{x}_2 &= f_2(t, x_1, \ldots, x_n, u_1, \ldots, u_p) \\ &\vdots \quad \vdots \\ \dot{x}_n &= f_n(t, x_1, \ldots, x_n, u_1, \ldots, u_p) \end{aligned} \tag{2.1}$$

Em que $x_1, \ldots, x_n$ são as variáveis de estado e $u_1, \ldots, u_p$ são as entradas do sistema. Essas equações podem ser dadas também em forma vetorial, em que tem-se

$$x = [x_1, x_2, \ldots, x_n]^T \tag{2.2}$$
$$u = [u_1, u_2, \ldots, u_p]^T \tag{2.3}$$
$$f(t, x, u) = [f_1(t, x, u), f_2(t, x, u), \ldots, f_n(t, x, u)]^T \tag{2.4}$$
$$\tag{2.5}$$

e então pode-se substituir as n equações de estado por uma equação de estado de dimensão n [Khalil 1996]

$$\dot{x} = f(t, x, u). \tag{2.6}$$

Pode-se ainda definir a seguinte relação

$$y = [y_1, y_2, \ldots, y_m]^T \tag{2.7}$$
$$h(t, x, u) = [h_1(t, x, u), h_2(t, x, u), \ldots, h_m(t, x, u)]^T \tag{2.8}$$
$$y = h(t, x, u) \tag{2.9}$$

onde y é denominado vetor de saída e engloba variáveis de interesse na análise do sistema. Observe que os termos de y podem ser estados independentes, e combinações lineares e não lineares desses. A Equação 2.6 em conjunto com a Equação 2.9 define o que é chamado de modelo em espaço de estados [Khalil 1996].

A Equação de Estados 2.6 pode também assumir as duas seguintes formas

$$\dot{x} = f(t, x) \tag{2.10}$$
$$\dot{x} = f(x, u) \tag{2.11}$$

onde a primeira é chamada de equação de estados não-forçada[7] e a segunda, invariante no tempo. Observe que no primeiro caso as entradas do sistema não precisam ser necessariamente nulas, essas podem simplesmente estarem descritas como uma função do tempo $u = \gamma(t)$, uma realimentação de estados $u = \gamma(x)$ (ver Seção 2.4.4), ou ambas $u = \gamma(t, x)$[Khalil 1996].

[7] do inglês, *unforced state equation*, tradução livre.

2.4.1.3 Espaço de Estados

Caso o modelo representado seja linear pode-se, ainda, representá-lo no seguinte formato de Espaço de Estados

$$\dot{x} = \mathbf{A}x + \mathbf{B}u, \qquad x \in \Re^n, u \in \Re^p, \tag{2.12}$$

$$y = \mathbf{C}x + \mathbf{D}u, \qquad y \in \Re^m \tag{2.13}$$

em que $\mathbf{A} \in \Re^{n\times n}$, $\mathbf{B} \in \Re^{n\times p}$, $\mathbf{C} \in \Re^{m\times n}$ e $\mathbf{D} \in \Re^{m\times p}$ [Hespanha 2009]. Observe que neste formato os auto-valores, λ (Equação (2.14)), da matriz de dinâmicas, A, representam a dinâmica do sistema modelado [Ogata 1982], e são as raízes da Equação Característica 2.14.

$$|\lambda\mathbf{I} - \mathbf{A}| = 0 \tag{2.14}$$

2.4.1.4 Pontos de Equilíbrio

Um conceito de fundamental importância quando se trabalha com espaço de estados é o de pontos de equilíbrio. A seguinte definição vale para sistemas não-forçados [Khalil 1996].

Definição 2.4.1. (Equilíbrio) Um ponto $x = x^{eq}$ no espaço de estados é chamado de ponto de equilíbrio de 2.10 se o sistema permanece em x^{eq} por todo tempo futuro sempre que ele começa em x^{eq}

Para sistemas invariantes no tempo a seguinte definição vale [Hespanha 2009].

Definição 2.4.2. (Equilíbrio) Um par $(x^{eq}, u^{eq}) \in \Re^n \times \Re^p$ é dito ser um ponto de equilíbrio de 2.11 se $f(x^{eq}, u^{eq}) = 0$. Nesse caso

$$u(t) = u^{eq}, \qquad x(t) = x^{eq}, \qquad y(t) = y^{eq} := h(x^{eq}, u^{eq})$$

é uma solução de 2.11

2.4.1.5 Discretização de Modelos em Espaço de Estados

Suponha o modelo apresentado na seção 2.4.1.3 onde $x = x(t)$ e $u = u(t)$. Supondo que esse modelo seja amostrado em intervalos T regulares, então deseja-se poder escrever esse sistema na forma de um Espaço de Estados discreto

$$x[kT + T] = \mathbf{A_d}x[kT] + \mathbf{B_d}u[kT] \tag{2.15}$$

$$y[kT] = \mathbf{C}x[kT] + \mathbf{D}u[kT] \tag{2.16}$$

sendo necessário obter uma relação entre as matrizes de estado contínua com as discretas [Levine 2010b].

Partindo da solução no tempo para os estados do sistema contínuo:

$$x(t) = e^{\mathbf{A}t}x(0) + \int_0^t e^{\mathbf{A}t-\tau}\mathbf{B}u(\tau)d\tau \tag{2.17}$$

com $t = kT, k = 0, 1, 2...$, a solução se torna:

$$x(kT) = e^{\mathbf{A}kT}x(0) + \int_0^{kT} e^{\mathbf{A}(kT-\tau)}\mathbf{B}u(\tau)d\tau \tag{2.18}$$

A solução dos estados para a amostragem $k+1$ pode, então, ser expressada da seguinte forma:

$$\begin{aligned}
x(kT+T) &= e^{\mathbf{A}(kT+T)}x(0) + \int_0^{kT+T} e^{\mathbf{A}(kT+T-\tau)}\mathbf{B}u(\tau)d\tau \\
&= e^{\mathbf{A}T}e^{\mathbf{A}kT}x(0) + \int_0^{kT} e^{\mathbf{A}(kT+T-\tau)}\mathbf{B}u(\tau)d\tau + \int_{kT}^{kT+T} e^{\mathbf{A}(kT+T-\tau)}\mathbf{B}u(\tau)d\tau \\
&= e^{\mathbf{A}T}\left[e^{\mathbf{A}kT}x(0) + \int_0^{kT} e^{\mathbf{A}(kT-\tau)}\mathbf{B}u(\tau)d\tau\right] + \int_{kT}^{kT+T} e^{\mathbf{A}(kT+T-\tau)}\mathbf{B}u(\tau)d\tau \\
&= e^{\mathbf{A}T}x(kT) + \int_{kT}^{kT+T} e^{\mathbf{A}(kT+T-\tau)}\mathbf{B}u(\tau)d\tau
\end{aligned}$$

ou seja,

$$\mathbf{A_d} = e^{\mathbf{A}T} \tag{2.19}$$

Dado que a entrada $u(t)$ seja constante entre cada amostragem, como é para casos de sistemas implementados com *zero-order hold* (ZOH), então:

$$x(kT+T) = e^{\mathbf{A}T}x(kT) + \left[\int_{kT}^{kT+T} e^{\mathbf{A}(kT+T-\tau)}d\tau\right]\mathbf{B}u(kT)$$

ou seja,

$$\mathbf{B_d} = \left[\int_{kT}^{kT+T} e^{\mathbf{A}(kT+T-\tau)}d\tau\right]\mathbf{B}$$

Seja, $\gamma = kT + T - \tau$ e $d\gamma = d\tau$, então:

$$\mathbf{B_d} = \left[\int_0^{T} e^{\mathbf{A}\gamma}d\gamma\right]\mathbf{B}$$

Expandindo $e^{A\gamma}$ em uma série de potências, tem-se:

$$\begin{aligned}
\mathbf{B_d} &= \left[\int_0^{T}\left(I + \frac{\mathbf{A}\gamma}{1!} + \frac{\mathbf{A}^2\gamma^2}{2!} + ... + \frac{\mathbf{A}^i\gamma^i}{i!} + ...\right)d\gamma\right]\mathbf{B} \\
&= \left[IT + \frac{\mathbf{A}T^2}{2!} + \frac{\mathbf{A}^2T^3}{3!} + ... + \frac{\mathbf{A}^iT^(i+1)}{(i+1)!} + ...\right]\mathbf{B}
\end{aligned}$$

multiplicando ambos os termos por $\mathbf{A}$,

$$\begin{aligned}
\mathbf{AB_d} &= \left[\frac{\mathbf{A}T}{1!} + \frac{\mathbf{A}^2T^2}{2!} + \frac{\mathbf{A}^3T^3}{3!} + ... + \frac{\mathbf{A}^(i+1)T^(i+1)}{(i+1)!} + ...\right]\mathbf{B} \\
&= (e^{\mathbf{A}T} - I)\mathbf{B}
\end{aligned}$$

e caso $\mathbf{A}$ seja invertível,

$$\mathbf{B_d} = \mathbf{A}^{-1}(e^{\mathbf{A}T} - I)\mathbf{B} = (e^{\mathbf{A}T} - I)\mathbf{A^{-1}B} \tag{2.20}$$

2.4.2 Modelo não linear *Single-Track* de um Carro

Para a obtenção do modelo da dinâmica de um carro é necessário primeiro observar alguns fenômenos que influenciam essa dinâmica. A seguir são descritos alguns conceitos e equações que, em seguida, são utilizados na obtenção do modelo final.

2.4.2.1 Resistência do Ar

Um corpo em movimento em um fluido, sofre ação deste na direção oposta do seu movimento. Essa ação é denominada arrasto e vem de duas fontes: atrito e pressão. A primeira acontece devido à força de atrito viscoso que gera uma tensão cisalhante na superfície do corpo, já a segunda é resultado da queda de pressão no sentido contrário do movimento causada pelo atrito ou por fatores geométricos do corpo em movimento, resultando em uma pressão superior na frente desse corpo. Essas duas fontes de arrasto podem ser modeladas como uma única força em forma vetorial como

$$\boldsymbol{F}_W = \frac{1}{2} c_W \rho_L A \dot{\boldsymbol{r}} |\dot{\boldsymbol{r}}| \tag{2.21}$$

em que c_W é o coeficiente de resistência aerodinâmica, ρ_L é a densidade do meio e, A e $\dot{\boldsymbol{r}}$ são respectivamente a área frontal e a velocidade linear do corpo [Moran 2003; Schramm, Hiller e Bardini 2014]. Esta equação também pode ser escrita em em coordenadas inerciais da seguinte forma

$$\begin{bmatrix} F_{W,x} \\ F_{W,y} \\ F_{W,z} \end{bmatrix} = \begin{bmatrix} \frac{1}{2} c_W \rho_L A \dot{x} \sqrt{\dot{x}^2 + \dot{y}^2 + \dot{z}^2} \\ \frac{1}{2} c_W \rho_L A \dot{y} \sqrt{\dot{x}^2 + \dot{y}^2 + \dot{z}^2} \\ \frac{1}{2} c_W \rho_L A \dot{z} \sqrt{\dot{x}^2 + \dot{y}^2 + \dot{z}^2} \end{bmatrix}.$$

2.4.2.2 Forças que Atuam nos Pneus

Com exceção das forças aerodinâmicas e das produzidas pela gravidade todas as outras forças que afetam a dirigibilidade de um carro vêm dos pneus. Como tal é importante que a modelagem das forças que atuam no contato pneu-pista seja suficientemente precisa, especialmente para situações em que ocorrem grandes acelerações laterais [Borrelli et al. 2005; Falcone et al. 2007].

Para o entendimento dos modelos dessas forças é necessário primeiramente entender dois conceitos: os escorregamentos[8] longitudinal e lateral. Porém antes de abordar esses dois conceitos é necessário definir uma quantidade de grande importância para a dinâmica veicular.

Raio Dinâmico do Pneu Um pneu de construção radial roda significativamente diferente de um pneu diagonal, que já não tem mais uso relevante em carros de passageiro modernos [Schramm, Hiller e Bardini 2014]. Devido à característica não-elástica da armadura metálica do pneu radial esse apresenta, mesmo em condições extremas de carregamento, um alongamento da banda de rolagem mínimo (abaixo de 1%) [Schieschke e Gnadler 1987]. Portanto, é sensato afirmar que a circunferência de rolagem U é constante em condições normais de operação do pneu. Essa circunferência é a medida de distância percorrida por um ponto na superfície do pneu durante uma revolução e é normalmente medida com o pneu rolando a $60\,\mathrm{km\,h^{-1}}$ sem

[8] do inglês, *slip*.

Tabela 2.1: Exemplo de raios dinâmicos e estáticos de pneus comerciais.

Pneus	r_0 [mm]	r_{stat} [mm]	U [mm]	r_{dyn} [mm]
185/65R15	311	284	1895	302
195/65R15	318	290	1935	308
205/65R15	324	294	1975	315

aceleração ou frenagem [DIN 1986]. A circunferência de rolagem descreve o raio dinâmico do pneu

$$r_{dyn} = \frac{U}{2\pi}. \tag{2.22}$$

Como a armadura metálica rígida não está na superfície da carcaça, mas sim abaixo desta, o raio dinâmico do pneu é menor que o raio de fabricação r_0, porém maior que o raio estático r_{stat}. A relação desses raios pode ser observada na Figura 2.5, e os valores dos raios de alguns modelos de pneus comerciais encontram-se expostos na Tabela 2.1.

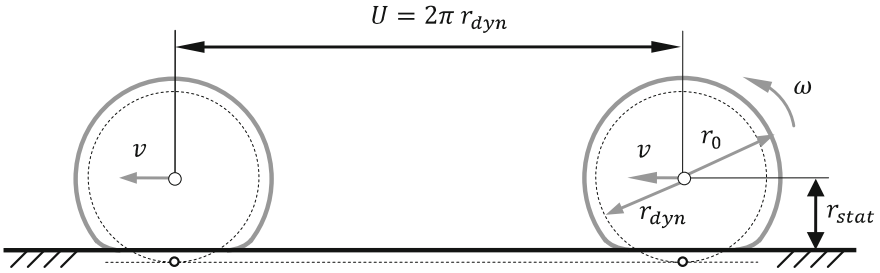


Figura 2.5: Definição do raio dinâmico do pneu.
Fonte: Schramm, Hiller e Bardini 2014

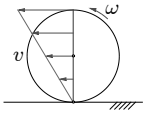
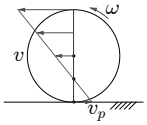
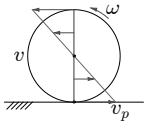
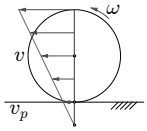
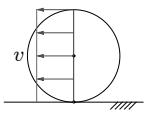
No resto deste texto, quando for mencionado o raio da roda (ou pneu) de um veículo, o leitor deve assumir que está sendo falado do raio dinâmico.

Escorregamento Longitudinal: Essa é uma quantidade cinemática que descreve o estado do movimento de uma roda, seja essa conduzida, frenada ou rolando livre. Assume-se para essa análise que a roda seja um corpo rígido. Existem dois estados do movimento que uma roda rígida pode assumir em uma movimentação planar[Schramm, Hiller e Bardini 2014]:

- rolamento cinemático puro sem deslizamento; e
- rolamento combinado com deslizamento.

No estudo de dinâmica veicular, o segundo caso é relevante e acontece em todos os casos em que o veículo está sob tração motora ou em frenagem. Para a análise do escorregamento longitudinal, deve-se considerar quatro variáveis: o raio r da roda, a velocidade v_x do centro da roda no sentido longitudinal (eixo x no sistema de coordenada fixo da roda), a velocidade imaginária v_p do contato do pneu com o chão e a velocidade angular da roda.

Tabela 2.2: Escorregamento de corpo rígido. Fonte: [Schramm, Hiller e Bardini 2014]

$v_x = r\omega$	$v_x < r\omega$		$v_x > r\omega$	
Roda rolando	Roda conduzida	Roda deslizando	Roda frenada	Roda travada
				
Sem escorregamento	Escorregamento de aceleração		Escorregamento de frenagem	
$s_A = 0$	$s_A = \frac{v_p}{r\omega}$	$s_A = 1$	$s_F = \frac{v_p}{v}$	$s_F = 1$
$s_F = 0$	$= \frac{r\omega - v_x}{r\omega}$		$= \frac{v_x - r\omega}{v}$	

Idealmente, a velocidade v_p no contato com o chão seria nula, porém em uma situação real isso não acontece. Essa velocidade é a diferença entra as velocidades v e $r\omega$ do pneu, dada com relação a maior delas. É em função dessa velocidade imaginária que os escorregamentos de aceleração e frenagem são dados. Esse primeiro

$$s_A = \frac{v_p}{r\omega} = \frac{r\omega - v_x}{r\omega} \tag{2.23}$$

é referente às situações em que o veículo está sendo conduzido ($v < r\omega$), e o segundo

$$s_F = \frac{v_p}{v_x} = \frac{v_x - r\omega}{v_x} \tag{2.24}$$

é usado quanto o carro está sendo frenado ($v > r\omega$). Pode-se então definir o escorregamento em termos gerais como:

$$s = \frac{|v_p|}{\max(|v_x|, |r\omega|)} = \frac{|v_x - r\omega|}{\max(|v_x|, |r\omega|)} \tag{2.25}$$

Essa definição garante que o escorregamento em condições extremas, roda travada e deslizando, assuma o valor 1[Schramm, Hiller e Bardini 2014]. Na Tabela 2.2 pode-se observar os diferentes estados possíveis e o valor do escorregamento para cada uma das condições.

Escorregamento lateral: Quando uma roda rolando livremente é atuada por uma força lateral, esta apresenta uma velocidade na direção da força em adição a seu movimento longitudinal. Deste modo, a roda se desloca em uma direção diferente daquela para qual a roda está direcionada. Pode-se então definir o angulo de escorregamento, sendo esse o angulo entre as direções da circunferência da roda e do movimento, como pode ser observado na Figura 2.6. Em termos matemáticos, pode-se definir esse ângulo por

$$\alpha = \arctan -\frac{v_y}{v_x}, \tag{2.26}$$

em que v_y é a velocidade lateral (no eixo y do sistema de coordenada fixo da roda).

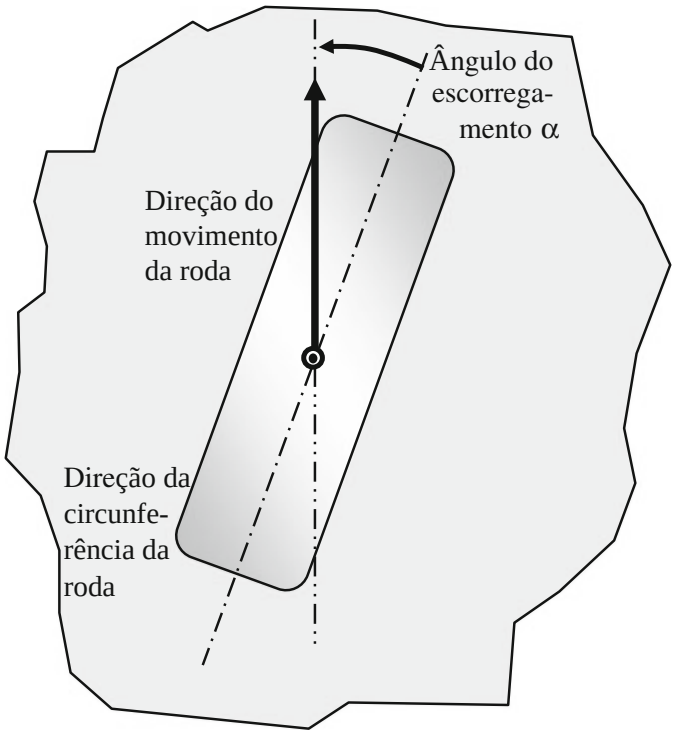


Figura 2.6: Definição do ângulo de escorregamento.
Fonte: Harrer e Pfeffer 2016

Esse escorregamento pode acontecer por dois motivos: o primeiro é a elasticidade da borracha que compõe o pneu permitindo que este se deforme de tal forma que o trecho de borracha que faz contato com o chão esteja alinhado com a direção do movimento e o segundo é o deslizamento da borracha no sentido da força aplicada preservando sua rotação e movimento longitudinal [Harrer e Pfeffer 2016; Schramm, Hiller e Bardini 2014]. Em uma situação real sempre vai haver tanto deformação quanto deslizamento da borracha do pneu, fenômeno que pode ser observado na Figura 2.7.

Fórmula Mágica: A fórmula mágica [Bakker, Nyborg e H. B. Pacejka 1987], também conhecida como modelo de Pacejka, é provavelmente o modelo estacionário de pneu mais utilizado [Levine 2010a]. Esse é um modelo não linear semi-empírico complexo que leva em consideração as forças longitudinais e laterais em ação combinada de frenagem (ou aceleração) e curvas [Falcone et al. 2007]. Os requisitos dessa fórmula mágica são que essa descreva todas as características estacionárias do pneu com precisão, facilidade de acesso aos dados necessários para a computação das forças e possibilidade de interpretação física das relações. Uma das possíveis funções usadas para descrever essas relações é, por exemplo, a seguinte [H. Pacejka 2006]:

$$y(x) = D \sin\left(C \arctan\left(Bx - E\left(Bx - \arctan Bx\right)\right)\right) \tag{2.27}$$

Figura 2.7: Comparativo da roda em rolamento livre e sob ação de uma força lateral. Fonte: Clark 1981

em que, $y(x)$ representa as forças longitudinais e laterais, e x é o escorregamento longitudinal s ou o angulo de escorregamento α. D é a força máxima, B é o coeficiente de dureza do pneu e, C e E são parâmetros de ajuste da curva no eixo x. Adicionalmente, o valor de arctan(BCD) é a inclinação das curvas características na origem [Schramm, Hiller e Bardini 2014]. Essas equações podem também serem expandidas para contabilizar deslocamentos horizontais e verticais nas curvas. Um exemplo dessas curvas pode ser visto na Figura 2.8.

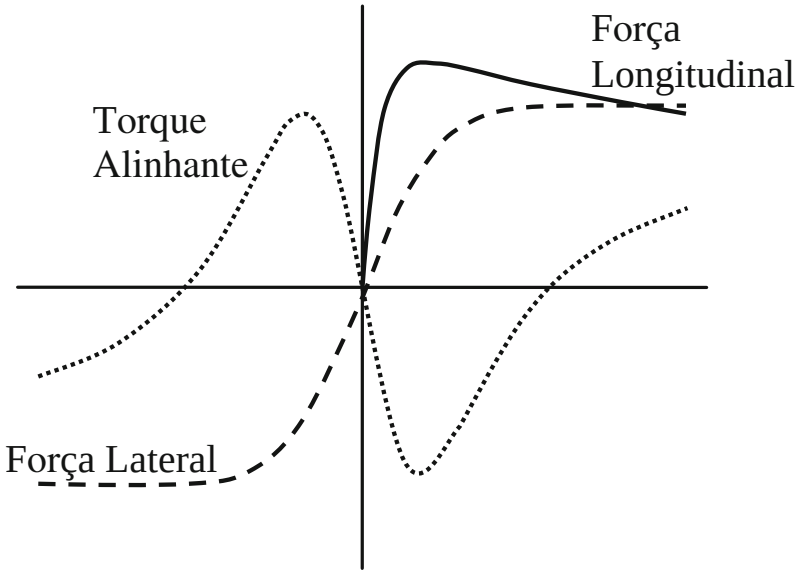


Figura 2.8: Exemplo de curva característica das forças que atuam no pneu. Fonte: Schramm, Hiller e Bardini 2014

É sensato admitir que as forças longitudinais e laterais não são independentes. Para representar essa dependência é em geral utilizado curvas ponderadas que efetua essa interdependência das forças. Um conjunto de equações que pode ser utilizado para modelar as dependência de forças é apresentado a seguir.

Utilizando uma versão simplificada da fórmula mágica, tem-se as forças estacionárias que

atuam nos pneus,

$$\begin{bmatrix} F_{x,stat} \\ F_{y,stat} \end{bmatrix} = F_{z,eff} \begin{bmatrix} \mu_x \sin\left(c_x \arctan\left(b_x \frac{s_a}{\mu_x}\right)\right) \\ \mu_y \sin\left(c_y \arctan\left(b_y \frac{s_a}{\mu_y}\right)\right) \end{bmatrix} \tag{2.28}$$

em que $F_{z,eff}$ é a carga efetiva nos pneus e μ é o coeficiente de atrito estático entre os materiais do pneu e da pista. Essa força pode ser calculada utilizando parâmetros variáveis e considerando a dependência degressiva das cargas nas forças horizontais. Para efeito de simplicidade, nesse trabalho é considerada cargas estáticas e obtidas por equilíbrio de forças, que está descrito na Seção 3.1.

A variável de entrada nessa formulação da fórmula mágica é o escorregamento normalizado

$$s_a = \sqrt{s^2 + \tan^2 \alpha}, \tag{2.29}$$

em que o escorregamento longitudinal s e o ângulo do escorregamento α são dados pelas equações 2.25 e 2.26.

A força resultante pode ser então calculada, com magnitude

$$F_\psi = \sqrt{\frac{s^2 F_{x,stat}^2 + \alpha^2 F_{y,stat}^2}{s_a^2}} \tag{2.30}$$

e angulo de ação

$$\psi = \arctan \frac{\alpha}{s}, \tag{2.31}$$

resultando em

$$\begin{bmatrix} {}^v F_x \\ {}^v F_y \end{bmatrix} = F_\psi(s_a) \begin{bmatrix} \cos \psi_v \\ \sin \psi_v \end{bmatrix} = \frac{1}{s_a} F_\psi(s_a) \begin{bmatrix} s \\ \alpha \end{bmatrix} \tag{2.32}$$

Adicionalmente pode-se acrescentar uma dinâmica de primeira ordem (veja Ogata 1982, pág. 161–162) na equação da força resultante. Isso é importante em situações onde há mudanças rápidas de percurso ou velocidade para considerar o tempo de acomodação da borracha do pneu. Nesses casos deve-se escolher uma constante de tempo adequada para as equações diferenciais. Neste trabalho a dinâmica de acomodação do pneu será desconsiderada.

2.4.3 Sistemas de Controle

Controle automático de processos é fundamental em qualquer área das ciências e engenharias, sendo parte integral de diversos sistemas veiculares, robóticos, industriais e de manufatura [Ogata 1982]. A maioria das técnicas de análise e desenvolvimento de sistemas de controle vêm da teoria de sistemas lineares, que assume uma relação linear entre a entrada e a saída de cada parte do sistema, e como tal esses podem ser representados por blocos.

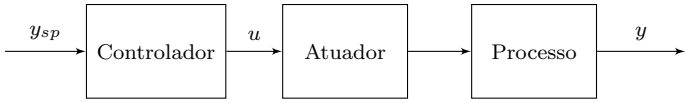


Figura 2.9: Sistema em malha aberta.

Um sistema de controle é um arranjo de componentes interligados em uma configuração que resultará na resposta desejada do sistema. Esse arranjo pode ser em malha aberta, em que o controlador não recebe informação da saída do processo, como mostrado na Figura 2.9, ou em malha fechada, no qual a saída é realimentada no controlador, como exibido na Figura 2.10[Dorf e Bishop 2011].

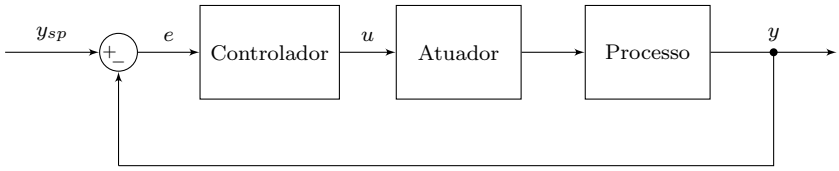


Figura 2.10: Sistema em malha fechada.

Nesses sistemas de controle geralmente controla-se o valor de uma quantidade de interesse, a variável controlada ou para a maioria dos casos a saída do sistema y, por meio da variação de outra quantidade, a variável manipulada, também chamada de sinal de controle u. É função do controlador calcular o sinal de controle com base no valor desejado y_{sp} (*set-point*), para sistemas em malha aberta, ou no erro do sistema e, que é a diferença entre o valor desejado e a variável controlada, para sistemas em malha fechada [Ogata 1982].

Como mencionado anteriormente, grande parte das ferramentas de analise e projeto de controladores vêm da teoria de sistemas lineares. Em um sistema de controle linear, não linearidades, como as descritas na Seção 2.4.1.1, seriam linearizadas nas proximidades de um ponto de operação e ferramentas de desenvolvimento lineares seriam utilizadas para propor um controlador e garantir a estabilidade do sistema. Essas ferramentas não podem ser usadas, porém, para desenvolver controladores não lineares e analisar a estabilidade desses sistemas. Para tal, utiliza-se ferramentas não lineares, como as descritas em [Khalil 1996; Tanaka e H. O. Wang 2001], algumas das quais são apresentadas posteriormente neste texto.

2.4.4 Técnicas de Controle

Como dito na Seção 2.4.3, a maioria das técnicas de desenvolvimento de controladores disponíveis atualmente vêm da teoria de sistemas lineares e, portanto, são capazes de projetar apenas controladores lineares. Vem havendo, porém, nas últimas décadas o desenvolvimento de técnicas de projeto de controladores não lineares.

2.4.4.1 Realimentação de Estados:

Considerando um sistema linear representado na forma de Espaço de Estados, equações (2.12) e (2.13), pode-se escolher a seguinte lei de controle

$$u = -\mathbf{K}x, \qquad \mathbf{K} \in \Re^{p\times n} \tag{2.33}$$

em que $\mathbf{K}$ é a matriz de ganhos de realimentação, transformando então a equação de estado (2.12) em

$$\dot{x} = (\mathbf{A} - \mathbf{BK})x. \tag{2.34}$$

Note que os auto-valores da matriz $(\mathbf{A} - \mathbf{BK})$ determinam a dinâmica em malha fechada do modelo e, portanto, pode-se escolher $\mathbf{K}$ de forma que o sistema controlado tenha uma determinada dinâmica [Hespanha 2009; Levine 2010b; Ogata 1982]. É importante salientar que caso os estados do sistema não sejam fáceis de mensurar é necessário utilizar um observador de estados e avaliar o impacto que este dá à dinâmica do sistema controlado.

2.4.4.2 Realimentação de Estados com Integrador

Considerando agora o caso em que deseja-se adicionar uma ação integral à um sistema controle por realimentação de estados discreto[9], de modo que o controle passe funcionar com erro de estado estacionário igual à zero. A topologia de controle deve ser modificada de acordo com o visto na Figura 2.11.

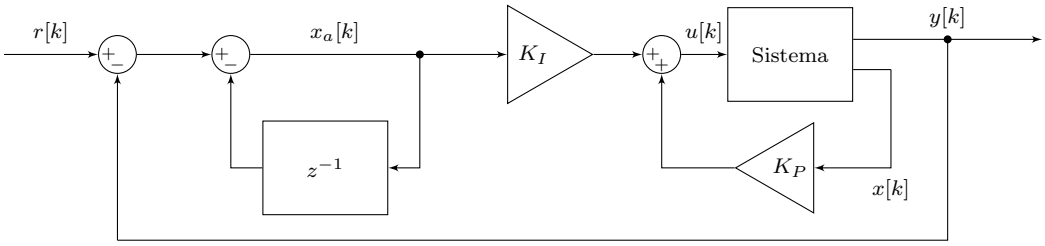


Figura 2.11: Sistema de controle discreto por realimentação de estados com ação integral. Fonte: Adaptado de [Lopes, Leite e Silva 2018]

Sistema Aumentado Esse sistema pode ser reorganizado para a topologia mostrada na Figura 2.12, onde tem-se então uma formulação igual ao da realimentação de estados sem ação integral e, portanto, pode-se utilizar as mesmas técnicas para projetar o ganho de realimentação.

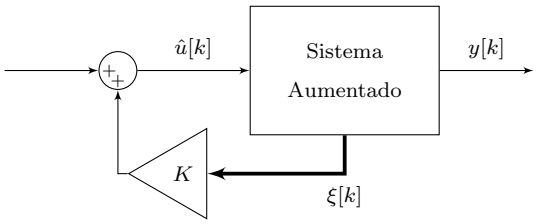


Figura 2.12: Sistema de controle discreto por realimentação de estados com ação integral na forma aumentada. Fonte: Adaptado de [Lopes, Leite e Silva 2018]

Observe que nessa topologia o Sistema Aumentado equivale à [Lopes, Leite e Silva 2018]:

$$\mathbf{A_{ad}} = \begin{bmatrix} \mathbf{A_d} & 0 \\ -\mathbf{C_d}\mathbf{A_d} & \mathbf{I} \end{bmatrix}, \quad \mathbf{B_{ad}} = \begin{bmatrix} \mathbf{B_d} \\ -\mathbf{C_d}\mathbf{B_d} \end{bmatrix} \tag{2.35}$$

e o ganho da realimentação:

$$K = \begin{bmatrix} K_P & K_I \end{bmatrix} \tag{2.36}$$

[9]Os passos para a discretização de um modelo em Espaço de Estados pode ser visto na Seção 2.4.1.5

Quando é necessário utilizar um controlador não linear, esse pode ser projetado com diversas técnicas que estão disponíveis na literatura, uma dessas é a Realimentação Linearizante [Khalil 1996].

2.4.5 Inteligência Computacional

A IC é um campo das ciências que possui muitas definições. O Quadro 2.1 apresenta oito definições de IC organizadas em quatro seções. As duas superiores estão preocupadas com o processo do pensamento, enquanto que as inferiores com o comportamento. As definições à esquerda avaliam o sucesso em quesitos de fidelidade à performance humana, já aquelas à direita o fazem por meio da performance ideal, um sistema é racional se este faz as escolhas certas dado o que este sistema sabe.

Quadro 2.1: Algumas definições de Inteligência Computacional. Adaptado de Russell e Norvig 2010[10]

Pensar Humanamente	**Pensar Racionalmente**
"O emocionante esforço de fazer computadores pensar... *máquinas com mentes*, no sentido completo e literal."[Haugeland 1985] "[A automação de] atividades que associamos com pensamento humano, atividades tais como tomada de decisão, solução de problemas, aprendizado..."[Bellman 1978]	"O estudo das faculdades mentais por meio de modelos computacionais."[Charniak e McDermott 1985] "O estudo das computações que tornam possível perceber, raciocinar e agir."[Winston 1992]
Agir Humanamente	**Agir Racionalmente**
"A arte de criar máquinas que realizam tarefas, que quando realizadas por pessoas, requerem inteligencia."[Kurzweil 1990] "A ciência de como fazer computadores realizarem coisas nas quais, no presente momento, humanos são melhores."[Rich e Knight 1991]	"Inteligência Computacional é a ciência do projeto de agentes inteligentes."[Poole, Mackworth e Goebel 1998] "IA... está interessada em comportamento inteligente em artefatos."[Nilsson 1998]

Ao longo da história da Inteligência Computacional todas as quatro abordagens foram seguidas, cada uma por pesquisadores diferentes e com métodos distintos.

2.4.5.1 Inteligência Computacional Distribuída

A maioria dos pesquisadores em inteligência computacional investiga como um agente pode apresentar comportamento inteligente tal como solução de problemas, planejamento, aprendizado, etc.. Alguns desenvolvimentos recentes provocaram, conjuntamente, aumento no interesse em concorrência e distribuição na IC: o desenvolvimento de poderosos computadores concorrentes, a proliferação de redes de computadores multi-nó e o reconhecimento que a maioria das

[10]Definições traduzidas do inglês, tradução livre. Para ver os textos originais ver referências inclusas ou [Russell e Norvig 2010, pág. 2]

atividades humanas, incluindo resolução de problemas, envolve grupos de pessoas. A Inteligência Computacional Distribuída (ICD) é o campo interessado em concorrência nas computações da IC[Bond e Gasser 2014].

Tradicionalmente a ICD é dividida em dois sub-grupos: Solução Distribuída de Problemas (SDP)[11] e Sistema Multi-Agente (SMA) [Peter Stone e Veloso 2000]. SDP estuda como a resolução de problemas pode ser dividida em módulos que compartilham informações e conhecimentos sobre o problema e a solução sendo desenvolvida. Já as pesquisas em SMA se preocupa em coordenar o comportamento de um conjunto de agentes e como esses cooperam para agir. Agentes em um SMA podem tanto trabalhar em prol de um único objetivo quanto objetivos individuais que se interceptam. Em SMA's os agentes, assim como módulos em SDP, devem compartilhar conhecimentos sobre o problema e a solução, porém além disso eles devem raciocinar o processo de cooperação entre agentes[Bond e Gasser 2014]. Em SMA, a tarefa de coordenação pode se tornar difícil, visto que pode ocorrer situações (Sistemas Abertos) em que não há conhecimento globalmente consistente, critérios de sucesso globais ou até mesmo representação universal do sistema[Hewitt 1986, 1985].

Considere um problema de satisfação de restrições[12]. Em SDP esses problemas seriam divididos em sub-problemas, não necessariamente independentes, que após resolvidos seriam usados para sintetizar a solução do problema original. Já em SMA, agentes distintos com seus próprios interesses e objetivos seriam alocados para resolver subproblemas, resolvendo, assim, o problema geral [Peter Stone e Veloso 2000].

2.4.6 Sistemas Multi-Agente

O termo agente foi mencionado diversas vezes anteriormente neste trabalho, sua definição, no entanto, foi evitada até este momento. Embora não haja um acordo sobre a definição de agente na comunidade de IC [Russell e Norvig 2010], em geral assume-se que agente seja uma entidade que opera independente de influência humana e em um ambiente no qual outros processos e agentes atuam [Shoham 1990]. Em um sistema multi-agente, utiliza-se de diversos agentes para resolver partes de um problema complexo independentemente, de forma que em uma escala maior o problema original seja resolvido.

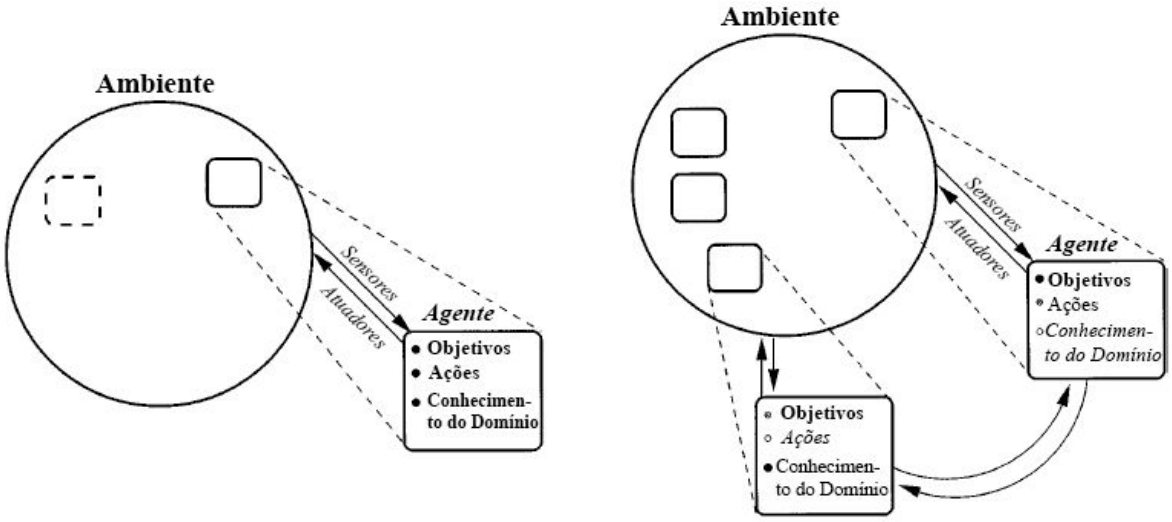
Como já foi expresso na Seção 2.2 os sistemas multi-agente possui certas características que o tornam uma opção válida para resolver diversos problemas. As características mais notáveis dessas são: paralelismo, robustez, escalabilidade e programação mais simples.

2.4.6.1 Agente Único vs. Multi-Agente

A alternativa mais óbvia a ser utilizada no lugar de um SMA é um Sistema de Agente Único (SAU). Esses sistemas possuem um único agente que toma todas as decisões. Esse tipo de sistema pode ter múltiplas entidades, como, por exemplo, diversos atuadores ou até mesmo múltiplos

[11] Do inglês, *Distribute Problem Solving*, tradução livre.

[12] Classe de problemas onde há um conjunto de variáveis $X_1, \ldots, X_n$, domínios $D_1, \ldots, D_n$, um para cada variável, e restrições $C_1, \ldots, C_m$. Nesses problemas tem-se o objetivo de satisfazer todas as restrições de modo que cada variável X_i esteja dentro de seu domínio D_i. Fazem parte dessa classe problemas clássicos, tais como o Sudoku e o problema das oito rainhas [Russell e Norvig 2010]



(a) SAU (b) SMA

Figura 2.13: Exemplo de Sistema de Agente Único e Sistema Multi-Agente.
Fonte: Adaptado de Peter Stone e Veloso 2000

componentes fisicamente separados. Se, no entanto, cada entidade envia suas percepções e recebe ordens de um único processo central, então há nesse sistema apenas um agente.

Sistema de Agente Único: Em um SAU, o agente modela a si mesmo, o ambiente e suas interações. O agente é uma entidade independente, com seus próprios objetivos, ações e conhecimento. Em um SAU nenhuma outra entidade desse tipo é considerada pelo agente. Embora outros agentes possam existir no meio, o agente não os reconhece como tal, e portanto modela esses como parte do ambiente [Peter Stone e Veloso 2000]. Um exemplo das interações em um SAU pode ser vista na Figura 2.13a.

Sistema Multi-Agente A diferença entre SMA e SAU é que em Sistemas Multi-Agente, diversos agentes existem que reconhecem a si e a outros como tal e modelam seus objetivos e ações. Em um caso geral de SMA, pode haver interação direta (comunicação) entre os agentes.

A principal diferença, do ponto de vista de um agente individual, entre SAU e SMA, é que no segundo, além da incerteza, que pode ser intrínseca do ambiente, as dinâmicas do meio podem ser alteradas intencionalmente, de modo imprevisível, por outros agentes. Portanto pode-se considerar que todo SMA possui um ambiente dinâmico [Peter Stone e Veloso 2000].

A Figura 2.13b demonstra como seriam as interações entre agentes de um SMA. Em um sistema pode existir qualquer quantidade de agentes, com diferentes graus de heterogeneidade, podendo, ou não, se comunicar. O modelo que cada agente desenvolve de outros também possui graus de incerteza, demonstrado na figura pelos diferentes estilos de fontes.

Caso o leitor queira estudar exemplos de SMA, recomenda-se o artigo de Peter Stone e Veloso 2000, que exemplifica SMA's aplicados ao Domínio Presa/Predador, com diferentes tipos de sistemas.

2.4.7 FIPA

A FIPA [13](acrônimo de, *Foundation for Intelligent Physical Agents*) é uma organização normatizadora da Sociedade de Computação da IEEE (Instituto dos Engenheiros Eletricistas e Eletrônicos)[14] focada em promover tecnologias de agentes e a interoperabilidade dos seus padrões com outras tecnologias.

As normatizações FIPA são suportadas por diversos frameworks e ambientes de desenvolvimento de agentes físicos e virtuais. A importância dessas normas é evidenciada pela habilidade de agentes desenvolvidos em conformidade com esse padrão conseguirem interagir entre si, mesmo que tenham sido programados por pessoas e em frameworks diferentes.

2.4.8 Plataforma de Agentes

Uma AP (*Agent Platform*) pode ser compreendida como uma instalação física onde os agentes existem e pode realizar suas funcionalidades. A AP é responsável por gerir as mudanças de estado do agente, seu ciclo de vida além de prover recursos para auxiliar os agentes, como o AMS (*Agent Management System*), o DF (*Directory Facilitator*) e o MTS (*Message Transport System*)[*FIPA Agent Management Specification* 2004]. Uma exemplificação da topologia de uma AP pode ser encontrada na Figura 2.14.

Agente O Agente é um processo computacional que implementa as funcionalidades comunicativas e autônomas de uma aplicação ou entidade [*FIPA Abstract Architecture Specification* 2002]. Segundo as especificações FIPA todo agente físico deve residir dentro de uma AP, e esta pode conter múltiplos agentes registrados nela. Todo agente deve possuir uma *Agent IDentification* (AID), que deve incluir, entre outras informações opcionais, um GUID (*Globally Unique IDentification*[15]) que geralmente é composto pelo seu nome local seguido pelo símbolo @ e a identificação da AP em que ele reside. Segue abaixo um exemplo dessa normatização do nome de um agente.

$$\overbrace{\underbrace{\text{Agent-1}}_{\text{Nome local}} @ \underbrace{192.168.1.18{:}1099/\text{JADE}}_{\text{Nome da AP}}}^{\text{GUID}}$$

Os agentes têm um ciclo de vida que é gerido pela AP. Esse ciclo pode ser visto na Figura 2.15.

Sistema Gestor de Agentes O AMS (*Agent Management System*) é um componente fundamental da AP e deve ser único, ou seja, pode existir apenas um AMS por AP. O AMS é o responsável por realizar as funções administrativas da AP, como criar e deletar agentes, e auxiliar na transferências de um Agente para outra AP.

[13]Informações sobre a FIPA e suas especificações podem ser encontradas no endereço online: `http://www.fipa.org`

[14]Do inglês, Institute of Electrical and Electronics Engineers

[15]do inglês, Identificador Único Global

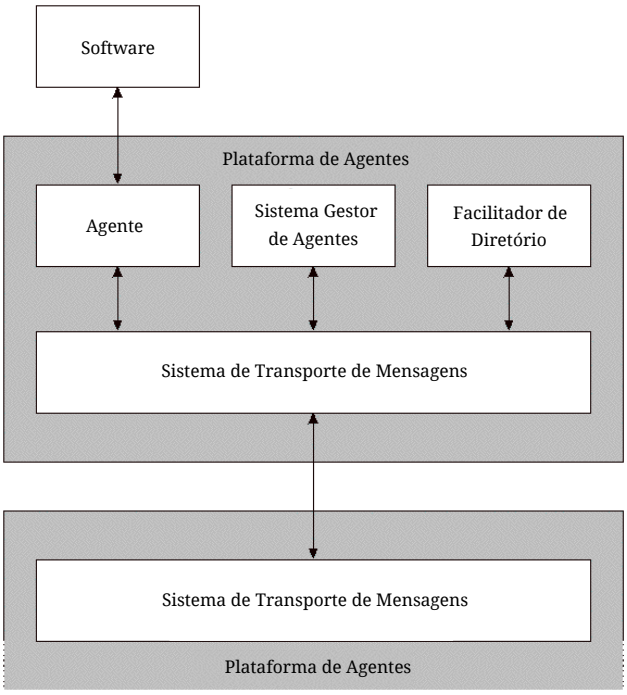


Figura 2.14: Representação de uma AP segundo as especificações FIPA. Fonte: Adaptado de *FIPA Agent Management Specification* 2004

Como parte das especificações FIPA, todo Agente deve ser registrado com o AMS da sua HAP[16]. Devido a esse fato o AMS também deve oferecer serviços de *White Pages*[17], fornecendo informações sobre todos os agentes registrados com ela por meio de buscas que qualquer agente pode fazer, mesmo estando em outra AP[*FIPA Agent Management Specification* 2004].

Facilitador de Diretório O DF (*Directory Facilitator*) é um componente opcional da AP que deve, caso implementado, fornecer serviços de *Yellow Pages*[18]. Agentes podem registrar seus serviços com o DF para que outros Agentes possam então buscar o DF por serviços que estes precisem[*FIPA Abstract Architecture Specification* 2002; *FIPA Agent Management Specification* 2004].

Sistema de Transporte de Mensagens O MTS (*Message Transport System*) provê o serviço de passagem de mensagem entre agentes de uma mesma AP ou de APs diferentes. As mensagens são passadas utilizando a Linguagem de Comunicação de Agentes[19] (ACL) [*FIPA Abstract*

[16] *Home Agent Platform*, tradução livre AP mãe.
[17] Expressão do inglês que se refere as páginas brancas de uma lista telefônica.
[18] Expressão do inglês que se refere as páginas amarelas de uma lista telefônica.
[19] em inglês, *Agent Comunication Language*

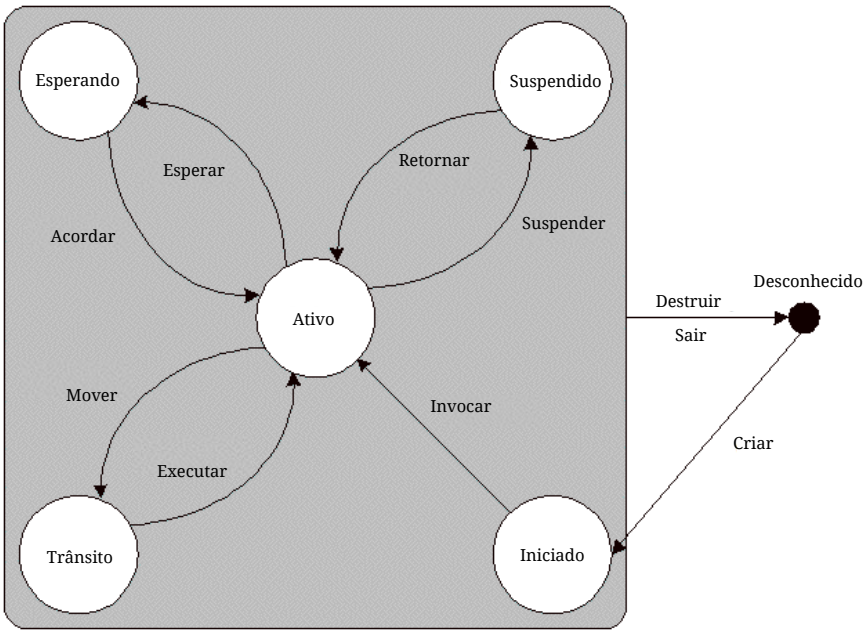


Figura 2.15: Ciclo de vida de um Agente. Fonte: Adaptado de *FIPA Agent Management Specification* 2004

Architecture Specification 2002; *FIPA Agent Message Transport Service Specification* 2002].

Linguagem ACL A ACL é a linguagem utilizada por Agentes FIPA para a transmissão de mensagens para outros Agentes. A linguagem é baseada no conceito de *Speech Acts*[20] introduzido por Austin e posteriormente retrabalhado por Searle. A ACL utiliza uma abordagem dos *Speech Acts* Searliana, utilizando uma semântica voltada à abstrações psicológicas, apesar de haver um movimento de pesquisadores [Chopra et al. 2012; Singh 1991] no sentido de fazer uma transição para uma abordagem Austiniana, com semântica voltada à abstrações sociais.

A Linguagem ACL provida pela FIPA possui 22 performativas que podem ser utilizadas pelos Agentes para interagir com outros Agentes. Essas performativas são [Bellifemine, Caire e Greenwood 2007; *FIPA Communicative Act Library Specification* 2002]:

- `ACCEPT_PROPOSAL`: Resposta afirmativa à um `PROPOSE`;
- `AGREE`: Resposta afirmativa à um `REQUEST`;
- `CANCEL`: Ação de cancelar um `REQUEST` prévio;

[20]Conceito abordado na *Speech Act Theory* em que os Atos de Fala (tradução livre) são enunciados com funções performativas, ou seja, os Atos de Fala representam ações realizadas (performadas) com palavras

- `CFP`: É um chamado à apresentação de propostas, do inglês *Call For Proposals*;
- `CONFIRM`: Afirma a veracidade de uma informação, quando sabe-se que o destinatário está incerto sobre a informação;
- `DISCONFIRM`: Afirma a falsidade de uma informação, quando sabe-se que o destinatário acredita ou acha provável que a informação é verdadeira;
- `FAILURE`: Informa falha ao executar uma ação requerida por um `REQUEST`;
- `INFORM`: Informa o destinatário sobre algo;
- `INFORM_IF`: Similar ao `CONFIRM`/`DISCONFIRM`, porém essa é uma ação macro e não pode ser performada diretamente;
- `INFORM_REF`: Similar ao `INFORM_IF`, porém requisitando o valor de uma expressão;
- `NOT_UNDERSTOOD`: Replica que o Agente não entendeu a mensagem recebida;
- `PROPAGATE`: Pedido de propagação da mensagem enviada à todos os agentes que o destinatário conhece;
- `PROPOSE`: Resposta à uma CFP;
- `PROXY`: Semelhante ao `PROPAGATE`, porém destinado à uma lista de Agentes anexada à mensagem;
- `QUERY_IF`: Utilizado quando é desejado confirmar a validade de uma informação ao invés de requerer a realização de uma tarefa;
- `QUERY_REF`: Similar ao `QUERY_IF`, porém requisitando o valor de uma expressão;
- `REFUSE`: Resposta negativa à um `REQUEST`;
- `REJECT_PROPOSAL`: Resposta negativa à um `PROPOSE`;
- `REQUEST`: Requerimento da execução de uma ação;
- `REQUEST_WHEN`: Requerimento da execução de uma ação quando uma determinada condição se tornar verdadeira;
- `REQUEST_WHENEVER`: Requerimento da execução de uma ação sempre que uma determinada condição se tornar verdadeira;
- `SUBSCRIBE`: Requerimento de uma intensão persistente de notificação sobre o valor de uma determinada referencia.

2.4.9 JAVA Agent DEvelopment Framework

O JADE é um *Framework* completamente desenvolvido em JAVA, produzido para simplificar o desenvolvimento de SMA em conformidade com as especificações FIPA.

Arquitetura JADE A arquitetura de uma AP em JADE difere levemente do padrão FIPA, pois ela adiciona uma nova abstração: os contêineres. Esses são os processos Java que fornecem as funcionalidades de runtime necessárias para hospedar e executar agentes.

Um grupo de contêineres centrados em torno de um Contêiner Principal (CP) forma uma AP. Para iniciar a AP é necessário lançar o Contêiner Principal e depois é possível ir adicionando Contêineres ao CP [Bellifemine, Caire e Greenwood 2007].

2.5 Conclusão do Capítulo

Neste capítulo foi visto uma breve exposição acerca do estado da arte nas principais áreas de pesquisa desse trabalho, com informações interessantes, tal como o avanço que a Tesla vêm promovendo na questão dos veículos autônomos, bem como o MATISSE, que é a ferramenta mais atual disponível no mundo quando se trata de aplicação de agentes em um contexto de condução colaborativa.

Além disto foi exposto uma breve revisão de literatura expondo os principais avanços que ocorreram ao longo da história nas áreas de veículos autônomos e programação orientada a agentes.

Por fim foi apresentado a fundamentação teórica que serviu de base para o desenvolvimento deste trabalho, abordando temas interessantes como modelagem matemática, projeto de controladores, inteligência computacional e sistemas multi-agentes.

Capítulo 3

Modelagem

Está presente na literatura uma grande variedade de modelos *single-track* lineares de carros. Esses modelos permitem ter uma previsão de comportamento típico da dinâmica desses veículos. Eles não permitem, porém, avaliar essa dinâmica em situações de maior angulação da roda ou acelerações laterais grandes. Por esses motivos foi decidido utilizar neste trabalho um modelo não linear para obter uma descrição mais precisa da dinâmica do carro.

3.1 Cinética do Modelo *Single-Track*

O modelo proposto para este trabalho pode ser visto em perspectivas superior e lateral nas figuras 3.1 e 3.2 respectivamente, e possui as seguintes características:

- Chassi rígido com três graus de liberdade, dois translacionais em x_V e y_V, e um rotacional ψ_V em torno do eixo vertical;
- Duas rodas imaginárias, uma dianteira e uma traseira, de índices v e h respectivamente, representativas das duas rodas de cada eixo. As rodas são caracterizadas por um vetor velocidade e nelas atuam uma determinada força;
- O ângulo da roda do eixo dianteiro, representado por δ;
- Forças de resistência aerodinâmica caracterizadas por F_W.

Aplicando a segunda lei de Newton no centro de gravidade do chassi, obtém-se a seguinte relação:

$$m\ddot{\mathbf{r}}_V = \mathbf{F}_v + \mathbf{F}_h + \mathbf{F}_W + \mathbf{F}_G \tag{3.1}$$

A aceleração $\ddot{\mathbf{r}}_V$ do carro pode ser obtida derivando duas vezes no tempo o vetor posição $\mathbf{r}_V$ do centro de gravidade do veículo.

$$\mathbf{r}_V = \begin{bmatrix} x_V \\ y_V \\ z_V \end{bmatrix}, \qquad \dot{\mathbf{r}}_V = \begin{bmatrix} \dot{x}_V \\ \dot{y}_V \\ 0 \end{bmatrix} \qquad \ddot{\mathbf{r}}_V = \begin{bmatrix} \ddot{x}_V \\ \ddot{y}_V \\ 0 \end{bmatrix} \tag{3.2}$$

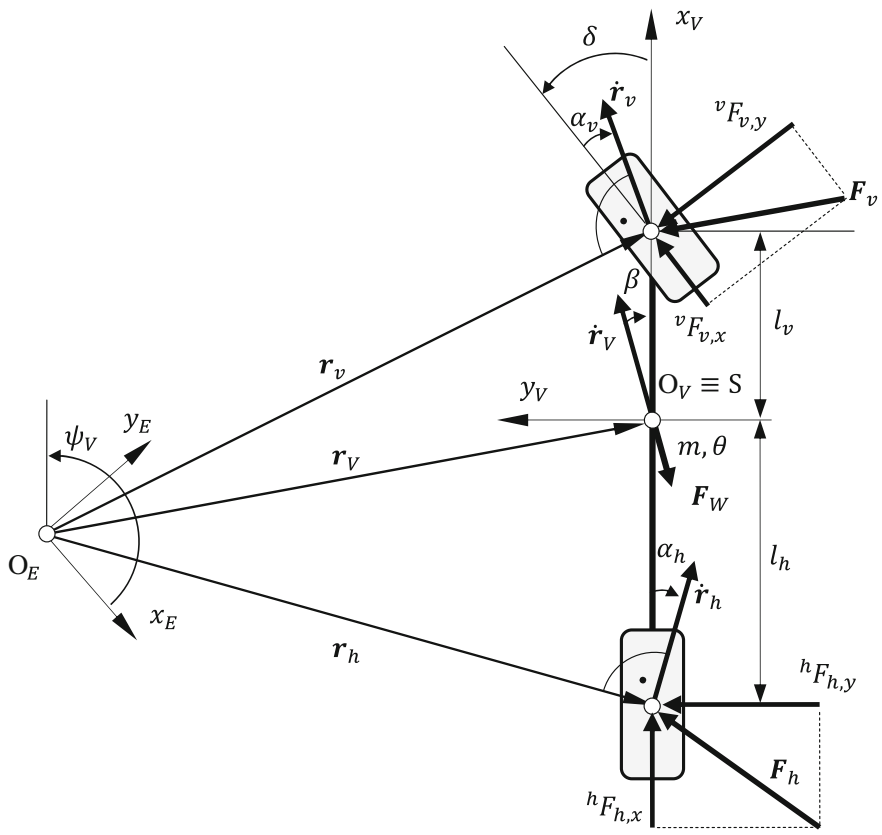


Figura 3.1: Vista superior do modelo *single-track*. Fonte: [Schramm, Hiller e Bardini 2014]

As forças que atuam nas rodas e aquelas devido à gravidade podem ser expressas no sistema de coordenadas inercial como:

$$\mathbf{F}_v = \begin{bmatrix} F_{v,x} \\ F_{v,y} \\ F_{v,z} \end{bmatrix}, \qquad \mathbf{F}_h = \begin{bmatrix} F_{h,x} \\ F_{h,y} \\ F_{h,z} \end{bmatrix}, \qquad \mathbf{F}_G = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} \tag{3.3}$$

E como foi explicado na seção 2.4.2.1, o arrasto pode ser expresso em forma vetorial e em coordenadas inerciais da seguinte forma:

$$\mathbf{F}_W = \frac{1}{2} c_W \rho_L A \dot{\boldsymbol{r}}_V |\dot{\boldsymbol{r}}_V| = \begin{bmatrix} F_{W,x} \\ F_{W,y} \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} c_W \rho_L A \dot{x}_V \sqrt{\dot{x}_V^2 + \dot{y}_V^2} \\ \frac{1}{2} c_W \rho_L A \dot{y}_V \sqrt{\dot{x}_V^2 + \dot{y}_V^2} \\ 0 \end{bmatrix} \tag{3.4}$$

A equação 3.1 pode ser então reescrita na seguinte forma:

$$\begin{bmatrix} m\ddot{x}_V \\ m\ddot{y}_V \\ 0 \end{bmatrix} = \begin{bmatrix} F_{v,x} + F_{h,x} - F_{w,x} \\ F_{v,y} + F_{h,y} - F_{w,y} \\ F_{v,z} + F_{h,z} - mg \end{bmatrix} \tag{3.5}$$

De modo similar, pode-se utilizar o princípio do momento angular para obter a equação 3.6 em relação ao centro de gravidade do veículo.

$$\boldsymbol{\Theta}_V \dot{\boldsymbol{\omega}}_V + \boldsymbol{\omega}_V \times (\boldsymbol{\Theta}_V \boldsymbol{\omega}_V) =_V^V \boldsymbol{r}_v \times \boldsymbol{F}_v +_V^V \boldsymbol{r}_h \times \boldsymbol{F}_h \tag{3.6}$$

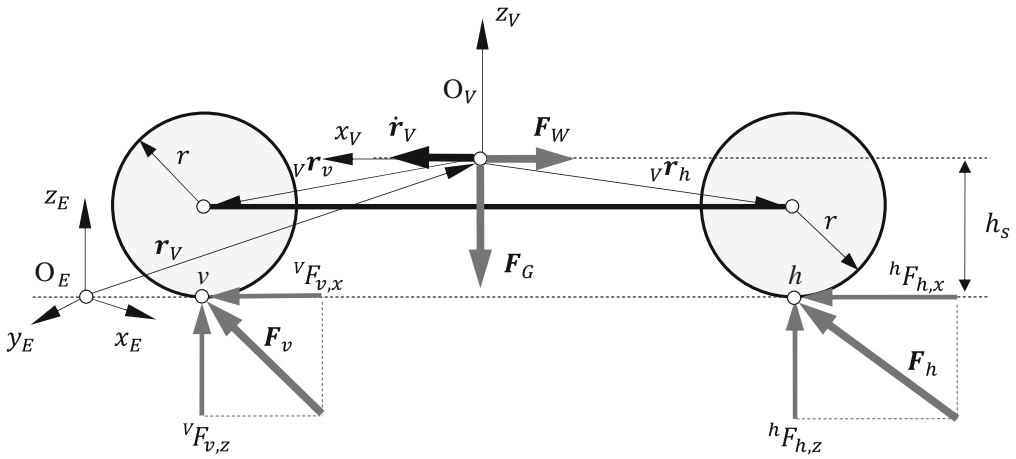


Figura 3.2: Vista lateral do modelo *single-track*. Fonte: [Schramm, Hiller e Bardini 2014]

Devido à consideração de haver rotação somente no eixo z do carro, a matriz de momento de inercia é

$$\boldsymbol{\Theta}_V = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \theta_{zz} \end{bmatrix}, \tag{3.7}$$

os vetores de velocidade e aceleração angular

$$\boldsymbol{\omega}_V = \begin{bmatrix} 0 \\ 0 \\ \psi_V \end{bmatrix}, \qquad \dot{\boldsymbol{\omega}}_V = \begin{bmatrix} 0 \\ 0 \\ \dot{\psi}_V \end{bmatrix}, \tag{3.8}$$

e os vetores para os pontos de aplicação das forças

$${}^V_V\boldsymbol{r}_v = \begin{bmatrix} l_v \\ 0 \\ -h_s \end{bmatrix}, \qquad {}^V_V\boldsymbol{r}_h = \begin{bmatrix} l_h \\ 0 \\ -h_s \end{bmatrix}. \tag{3.9}$$

Pode-se então simplificar a equação vetorial 3.6 para a forma escalar.

$$\theta_{zz}\ddot{\psi}_V = l_v F_{v,y} + l_h F_{h,y} \tag{3.10}$$

Na Seção 2.4.2.2 foi apresentada a Fórmula Mágica que modela as forças que atuam no pneu. Antes de entrar no modelo porém, é necessário reescrever essas forças, que estão descritas no sistema de coordenadas do pneu, no do veículo.

$$\boldsymbol{F}_v = \boldsymbol{T}_V(-\psi_V - \delta)^v\boldsymbol{F}_v \tag{3.11}$$

$$\boldsymbol{F}_h = \boldsymbol{T}_V(-\psi_V)^h\boldsymbol{F}_h \tag{3.12}$$

Em que $\boldsymbol{T}_V(\beta)$ é o tensor de rotação no centro gravidade V do carro, no sentido de z de um dado angulo β e ${}^v\boldsymbol{F}_v$ são as forças descritas pela Equação 2.32. Pode-se, desse modo, calcular

as forças longitudinais e laterais que atuam no pneu descritas no sistema de coordenadas do veículo.

$$\begin{bmatrix} F_{v,x} \\ F_{v,y} \\ F_{v,z} \end{bmatrix} = \begin{bmatrix} \cos(\psi_V + \delta) & \sin(\psi_V + \delta) & 0 \\ -\sin(\psi_V + \delta) & \cos(\psi_V + \delta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} {}^{v}F_{v,x} \\ {}^{v}F_{v,y} \\ {}^{v}F_{v,z} \end{bmatrix} \tag{3.13}$$

$$\begin{bmatrix} F_{h,x} \\ F_{h,y} \\ F_{h,z} \end{bmatrix} = \begin{bmatrix} \cos(\psi_V) & \sin(\psi_V) & 0 \\ -\sin(\psi_V) & \cos(\psi_V) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} {}^{h}F_{h,x} \\ {}^{h}F_{h,y} \\ {}^{h}F_{h,z} \end{bmatrix} \tag{3.14}$$

É importante observar que para o cálculo das forças como descritas pelo modelo de Pacejka é necessário calcular as velocidades no centro das rodas e as cargas normais no pneu.

As velocidades no centro da roda são dadas no sistema de coordenadas inercial pela seguinte relação cinemática:

$$\dot{\boldsymbol{r}}_v = \dot{\boldsymbol{r}}_V + \omega_V \times \boldsymbol{T}_V(\psi_V) {}^{V}_{V}\boldsymbol{r}_v \tag{3.15}$$

$$\dot{\boldsymbol{r}}_h = \dot{\boldsymbol{r}}_V + \omega_V \times \boldsymbol{T}_V(\psi_V) {}^{V}_{V}\boldsymbol{r}_h \tag{3.16}$$

O que resulta em

$$\begin{aligned} \begin{bmatrix} \dot{x}_v \\ \dot{y}_v \\ \dot{z}_v \end{bmatrix} &= \begin{bmatrix} \dot{x}_V \\ \dot{y}_V \\ \dot{z}_V \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \psi_V \end{bmatrix} \times \begin{bmatrix} \cos(\psi_V) & -\sin(\psi_V) & 0 \\ \sin(\psi_V) & \cos(\psi_V) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} l_v \\ 0 \\ -(h_s - r) \end{bmatrix} \\ &= \begin{bmatrix} \dot{x}_V - l_v \dot{\psi}_V \sin\psi_V \\ \dot{y}_V + l_v \dot{\psi}_V \cos\psi_V \\ 0 \end{bmatrix} \end{aligned} \tag{3.17}$$

para as rodas do eixo dianteiro e

$$\begin{aligned} \begin{bmatrix} \dot{x}_h \\ \dot{y}_h \\ \dot{z}_h \end{bmatrix} &= \begin{bmatrix} \dot{x}_V \\ \dot{y}_V \\ \dot{z}_V \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \psi_V \end{bmatrix} \times \begin{bmatrix} \cos(\psi_V) & -\sin(\psi_V) & 0 \\ \sin(\psi_V) & \cos(\psi_V) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} l_h \\ 0 \\ -(h_s - r) \end{bmatrix} \\ &= \begin{bmatrix} \dot{x}_V + l_h \dot{\psi}_V \sin\psi_V \\ \dot{y}_V - l_h \dot{\psi}_V \cos\psi_V \\ 0 \end{bmatrix} \end{aligned} \tag{3.18}$$

para as rodas do eixo traseiro. Para o cálculo do escorregamento, é necessário obter as velocidades com relação ao sistema de coordenadas fixo da roda. Isso é feito considerando a rotação da roda com relação ao sistema de coordenadas inercial,

$${}^{v}\dot{\boldsymbol{r}}_v = \begin{bmatrix} {}^{v}\dot{x}_v \\ {}^{v}\dot{y}_v \\ {}^{v}\dot{z}_v \end{bmatrix} = {}^{v}\boldsymbol{T}_E(-\psi_V - \delta)\dot{\boldsymbol{r}}_v = \begin{bmatrix} \cos(\psi_V + \delta) & \sin(\psi_V + \delta) & 0 \\ -\sin(\psi_V + \delta) & \cos(\psi_V + \delta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{x}_v \\ \dot{y}_v \\ \dot{z}_v \end{bmatrix} \tag{3.19}$$

para as rodas do eixo dianteiro e

$${}^{h}\dot{\boldsymbol{r}}_h = \begin{bmatrix} {}^{h}\dot{x}_h \\ {}^{h}\dot{y}_h \\ {}^{h}\dot{z}_h \end{bmatrix} = {}^{h}\boldsymbol{T}_E(-\psi_V)\dot{\boldsymbol{r}}_h = \begin{bmatrix} \cos(\psi_V) & \sin(\psi_V) & 0 \\ -\sin(\psi_V) & \cos(\psi_V) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{x}_h \\ \dot{y}_h \\ \dot{z}_h \end{bmatrix} \tag{3.20}$$

para as rodas do eixo traseiro.

Já as cargar normais nos pneus podem ser obtidas fazendo o equilíbrio dos momentos de força no centro de gravidade do veículo, na direção z do sistema inercial pode-se obter as cargas normais:

$$F_{v,z} = mg\frac{l_h}{l} - \frac{h_s}{l}\left({}^{V}F_{v,x} + {}^{V}F_{h,x}\right), \qquad F_{h,z} = mg\frac{l_v}{l} + \frac{h_s}{l}\left({}^{V}F_{v,x} + {}^{V}F_{h,x}\right) \tag{3.21}$$

3.1.1 Equações de Estado do Modelo

Observando o desenvolvimento descrito nas seções 3.1 e 2.4.2, pode-se agora montar o modelo na forma de equações de estado como descrito pela Equação 2.6

$$\begin{bmatrix} \dot{x}_V \\ \dot{y}_V \\ \dot{\psi}_V \\ \ddot{x}_V \\ \ddot{y}_V \\ \ddot{\psi}_V \end{bmatrix} = \begin{bmatrix} \dot{x}_V \\ \dot{y}_V \\ \dot{\psi}_V \\ \frac{1}{m}\left(F_{v,x} + F_{h,x} - F_{W,x}\right) \\ \frac{1}{m}\left(F_{v,y} + F_{h,y} - F_{W,y}\right) \\ \frac{1}{\theta_{zz}}\left(l_v\ {}^{V}F_{v,y} - l_h\ {}^{V}F_{h,y}\right) \end{bmatrix} \tag{3.22}$$

com o vetor de estados

$$x = \left[x_V, y_V, \psi_V, \dot{x}_V, \dot{y}_V, \dot{\psi}_V\right]^T \tag{3.23}$$

e a entrada

$$u = \delta \tag{3.24}$$

3.1.2 Linearização do Modelo

Após muita ponderação sobre como abordar a linearização do modelo apresentado na Seção 3.1.1, foi determinado que a utilização de uma ferramenta numérica seria a melhor opção. A ferramenta escolhida foi a `polyfitn()` [John D'Errico 2017], uma extensão para a função polyfit do MATLAB que permite a linearização de modelos com mais de uma variável independente.

Para a utilização dessa ferramenta foi primeiro necessário criar um conjunto de dados representativos da dinâmica nos intervalos de variação estabelecidos para cada estado e entrada do modelo.

$$\begin{array}{l} \{\ \psi_V \in \Re^n \mid \quad -\pi \leq \psi_V < \pi \quad \} \\ \{\ \dot{x}_V \in \Re^m \mid \ -10 \leq \dot{x}_V \leq 10 \ \} \\ \{\ \dot{y}_V \in \Re^p \mid \ -10 \leq \dot{y}_V \leq 10 \ \} \\ \{\ \dot{\psi}_V \in \Re^q \mid -\pi/6 \leq \dot{\psi}_V \leq \pi/6 \ \} \\ \{\ \ \delta \ \in \Re^n \mid -\pi/18 \leq \delta \leq \pi/18 \ \} \end{array}$$

O conjunto de dados necessário para utilizar a função `pylofitn()` é uma combinação vetorial de todos os estados e entradas, como por exemplo, para ψ_V e $\dot{x}_V$:

$$X = \begin{bmatrix} \psi_V[1] & \dot{x}_V[1] \\ \vdots & \vdots \\ \psi_V[n] & \vdots \\ \psi_V[1] & \dot{x}_V[2] \\ \vdots & \vdots \\ \psi_V[1] & \dot{x}_V[m] \\ \vdots & \vdots \\ \psi_V[n] & \vdots \end{bmatrix}$$

Tendo então, o conjunto de dados apresentado acima e o resultado das equações de estado para cada combinação dos estados e entradas, é possível então linearizar o modelo utilizando a ferramenta `polyfitn()` e obter um modelo em espaço de estados. O modelo obtido foi o seguinte:

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & -6.68 \times 10^{-4} & 6.1 \times 10^{-3} & 0 & 0 \\ 0 & 0 & 0 & 0 & 6.6 \times 10^{-3} & 1.4 \times 10^{-3} \\ 0 & 0 & 0 & 0 & 0 & -8.49 \times 10^{-4} \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 14.1166 \\ 0.2398 \end{bmatrix} \tag{3.25}$$

3.2 Controle de Velocidade de Cruzeiro

No Capítulo 4 será introduzido um SMA, onde os veículos deverão controlar suas velocidades. Essa seção apresenta um modelo simples de controle de velocidade de cruzeiro que será utilizado.

Considere um modelo veicular com um grau de liberdade e que a única força de oposição ao movimento seja a resistência do ar, abordado na Seção 2.4.2.1. Como visto, o arrasto é uma função do quadrado da velocidade, o que resultaria em um modelo do tipo:

$$m\dot{v} = -\frac{1}{2}c_W \rho_L A v^2 + u \tag{3.26}$$

porém é desejável ter um modelo linear, como o visto na Figura 3.3. Para tal é necessário linearizar a equação de estado 3.26. Para tal precisamos da função da dinâmica $f(v, u)$

$$\dot{v} = f(v, u) = -\frac{1}{2m}c_W \rho_L A v^2 + \frac{1}{m}u$$

Linearizando a equação de estado em torno de $v^{eq} = 10\text{m}\,\text{s}^{-1}$ e utilizando $m = 1000\text{kg}$, $c_W = 0.33$, $\rho_L = 1.225\text{kg}\,\text{m}^{-3}$ e $A = 2.14\text{m}^2$ tem-se[1]:

$$A = \frac{\delta f(v^{eq}, u^{eq})}{\delta v} = -\frac{0.4325}{1000}v^2 dv = -\frac{0.865}{1000}v^{eq} = -\frac{8.65}{1000} \tag{3.27}$$

[1]Dados disponíveis em [Schramm, Hiller e Bardini 2014]

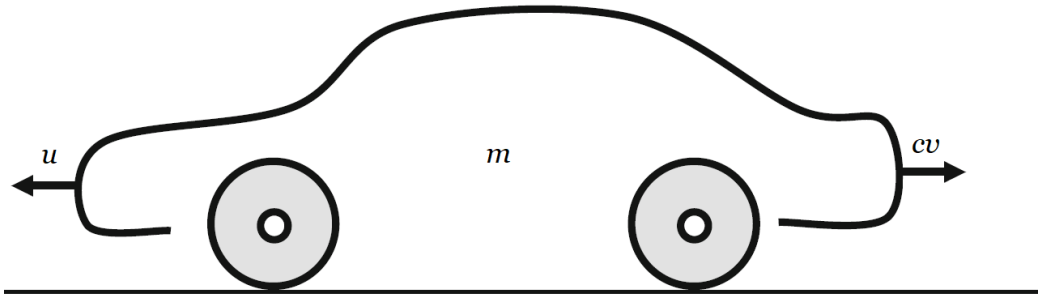


Figura 3.3: Esquemático do modelo de velocidade de cruzeiro. Fonte: Autor.

$$B = \frac{\delta f(v^{eq}, u^{eq})}{\delta u} = \frac{1}{1000} \tag{3.28}$$

Dessa forma o sistema se torna:

$$\dot{v} = -\frac{8.65}{1000}v + \frac{1}{1000}u \tag{3.29}$$

tendo-se então, um simples modelo massa amortecedor como representado na Figura 3.3. Considerando $x = v$, pode-se então colocar esse sistema na forma de equação de estados:

$$\dot{x} = -\frac{8.65}{1000}x + \frac{1}{1000}u \tag{3.30}$$

$$y = x \tag{3.31}$$

e obter um modelo em espaço de estados:

$$\mathbf{A} = \left[-\tfrac{8.65}{1000}\right], \quad \mathbf{B} = \left[\tfrac{1}{1000}\right], \quad \mathbf{C} = \left[1\right] \tag{3.32}$$

É importante observar que, como a matriz $\mathbf{C} = 1$ a saída desse sistema é exatamente o estado, que é uma grandeza física facilmente mensurável. Isso implica que esse sistema não necessita de observador de estado para fazer realimentação de estados.

Além disso como a matriz de dinâmica $\mathbf{A} \in \Re^{1\times1}$, a discretização desse sistema se torna trivial, pois $e^{\mathbf{A}T}$ é uma exponencial simples, e $\mathbf{A}^{-1} = 1/\mathbf{A}$.

Fazendo a discretização do sistema com $T = 50$ms tem-se:

$$\mathbf{A_d} = e^{\mathbf{A}T} = 0.9996 \tag{3.33}$$

$$\mathbf{B_d} = \mathbf{A}^{-1}(e^{\mathbf{A}T} - \mathbf{I})\mathbf{B} = 4.9989 \times 10^{-5} \tag{3.34}$$

Agora é possível projetar uma realimentação de estado com ação integral utilizando o método de alocação de polos. Primeiro é necessário montar o sistema aumentado

$$\mathbf{A_a} = \begin{bmatrix} 0.9996 & 0 \\ -0.9996 & 1 \end{bmatrix}, \quad \mathbf{B_a} = \begin{bmatrix} 4.9989 \times 10^{-5} \\ -4.9989 \times 10^{-5} \end{bmatrix} \tag{3.35}$$

e projetar os polos de malha fechada. É desejável ter uma dinâmica de segunda ordem levemente super-amortecida, com tempo de acomodação próximo de 1.5s

$$T_s = \frac{4}{\zeta \omega_n} \Rightarrow \omega_n = \frac{4}{1 \cdot 1.5} = 2.66 \tag{3.36}$$

portanto os polos contínuos foram alocados próximo de -2.66.

$$p_1 = -2.56$$
$$p_2 = -2.76$$

e os polos discretos

$$p_{1d} = e^{-2.56T} = 0.8799$$
$$p_{2d} = e^{-2.76T} = 0.8711$$

Utilizando a técnica de alocação de polos foi obtido os seguintes ganhos:

$$K = \begin{bmatrix} 4663.6 & -309.8 \end{bmatrix} \tag{3.37}$$

3.3 Simulações e Resultados Obtidos

Nesta seção estão apresentados as simulações e resultados obtidos acerca dos trabalhos discutidos nas seções anteriores deste capítulo.

3.3.1 Modelo *Single-Track*

Foram feitas duas simulações em malha aberta do modelo apresentado anteriormente, utilizando os *softwares* MATLAB e Simulink. Para essas simulações foi considerado que o veículo estava com a velocidade linear do carro e velocidade angular da roda constantes. Além disso, nestas primeiras simulações não foi considerado a resistência aerodinâmica do meio.

Para realizar essas simulações foram utilizadas informações de um catálogo de dados de parâmetros inerciais, colhidos pela NHTSA durante Novembro de 1998 [Heydinger et al. 1999]. Desse catálogo foram obtidos o peso, distância entre eixos, posição do centro de gravidade e momento de inercia do veículo escolhido, um Audi Quattro 4000. Os parâmetros do pneu usados foram baseados em H. Pacejka 2006 e MATLAB 2017.

Foram realizadas duas simulações com os mesmos parâmetros do veículo, porém com diferentes sinais de entrada. Para a primeira simulação foi considerado como sinal de entrada um degrau

$$u(t) = 1^\circ \approx 0.01745\text{rad}, \qquad \forall t > 1\text{s}$$

na angulação da roda. No *software* Simulink foi utilizado o bloco de Função do MATLAB para poder construir com facilidade o modelo não linear.

Para essa primeira simulação espera-se que o veículo faça uma curva suave para o sentido positivo de y_E e, de acordo com a simulação de um modelo linear em H. Pacejka 2006, que a velocidade angular do carro aumente com uma dinâmica próxima a de primeira ordem.

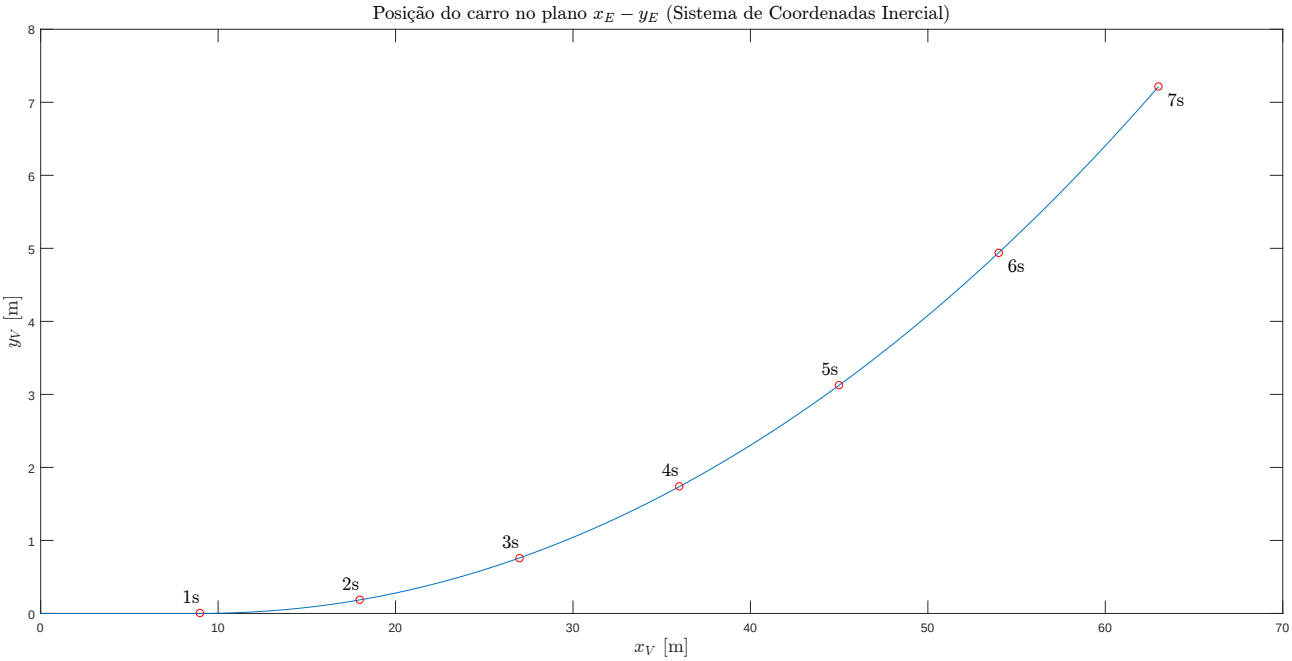


Figura 3.4: Gráfico da posição absoluta obtido na simulação com entrada degrau. Fonte: Autor.

Como pode ser observado na Figura 3.4, o veículo exibiu o comportamento esperado quanto a posição absoluta. Já na Figura 3.5, pode-se observar que o carro se portou de forma próxima à esperada com relação à velocidade angular, o que, levando em conta o caráter não linear do modelo, é um resultado muito positivo.

Já na segunda simulação foi utilizado um sinal composto na entrada, que pode ser descrito pela seguinte equação:

$$u(t) = \begin{cases} \frac{10\pi}{180}t & \text{para} \quad t \leq 1 \\ \frac{10\pi}{180} & \text{para} \quad 0 < t \leq 2 \\ -\frac{10\pi}{180}t + \frac{20\pi}{180} & \text{para} \quad 2 < t \leq 4 \\ -\frac{10\pi}{180} & \text{para} \quad 4 < t \leq 5 \\ \frac{10\pi}{180}t - \frac{50\pi}{180} & \text{para} \quad 5 < t \leq 6 \\ 0 & \text{para} \quad 6 < t \leq 7 \end{cases} \tag{3.38}$$

Essa equação composta define uma rampa inclinação de 10° de 0s a 1s, mantendo esse valor por 1s depois do qual a entrada decai com inclinação de −10° de 2s a 4s, novamente mantendo

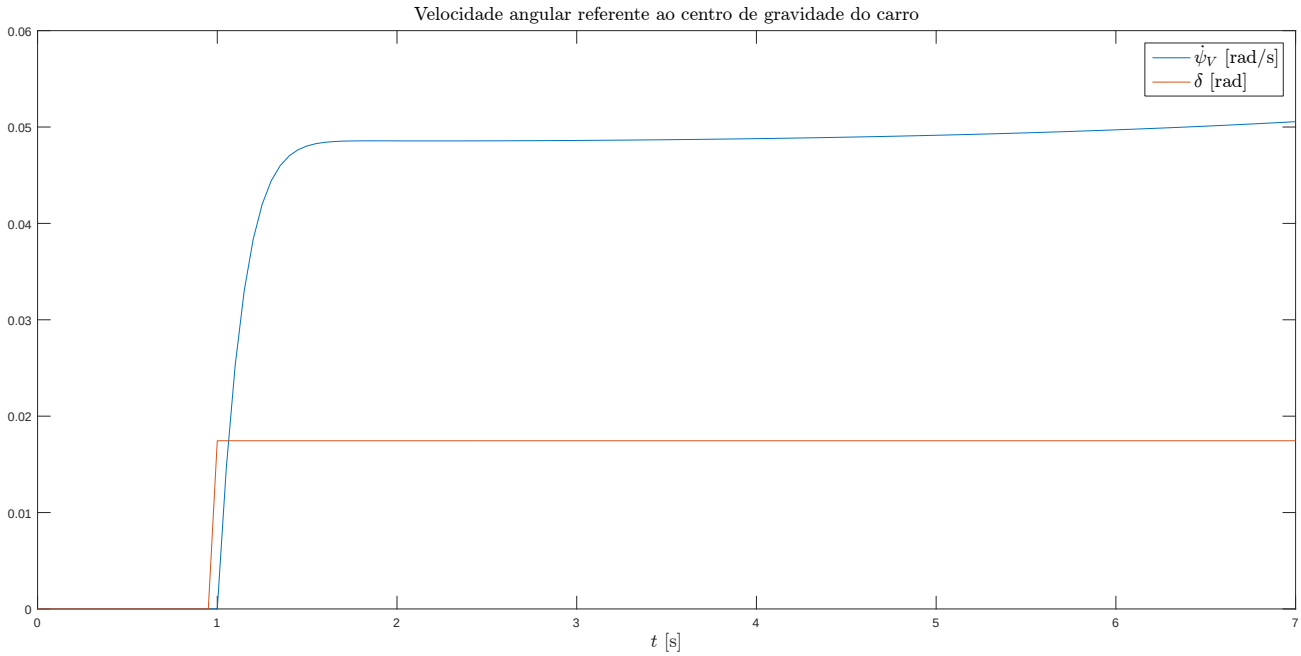


Figura 3.5: Gráfico da velocidade angular do carro obtido na simulação com entrada degrau. Fonte: Autor.

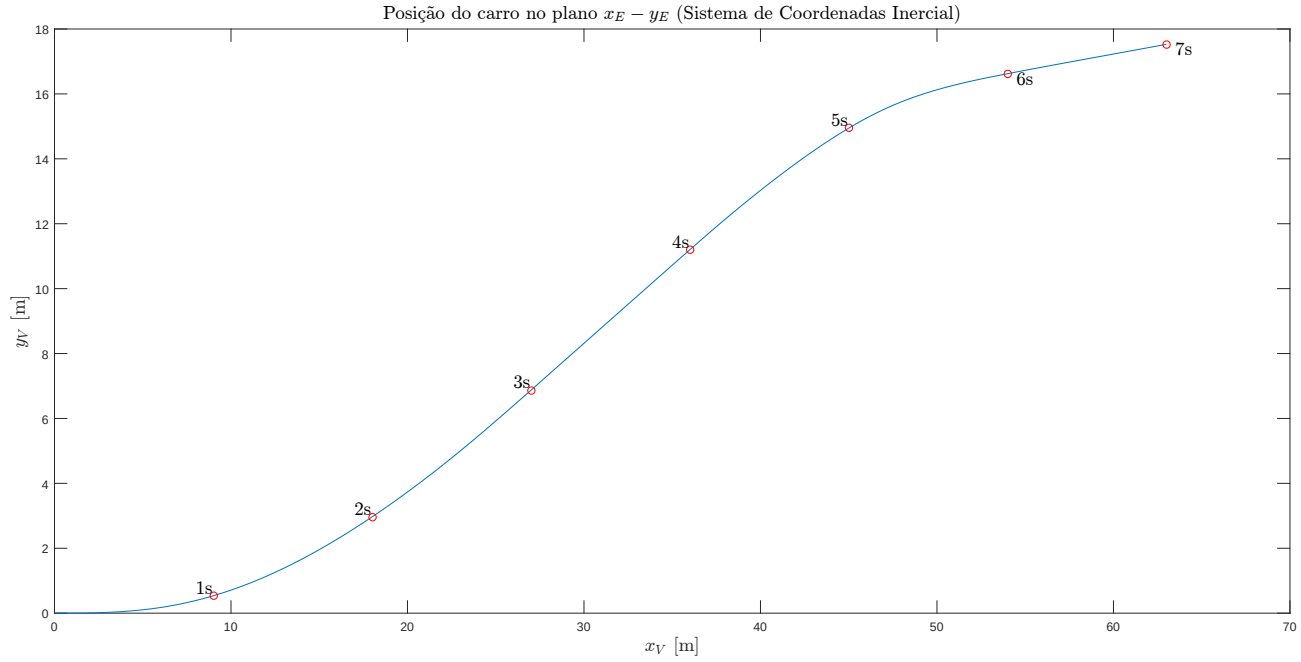


Figura 3.6: Gráfico da posição absoluta obtido na simulação com a entrada composta. Fonte: Autor.

o valor por 1s e então retornando em rampa com a mesma inclinação para 0° e mantendo nesse valor por mais 1s. Essa curva pode ser observada nas figuras 3.7 e 3.8.

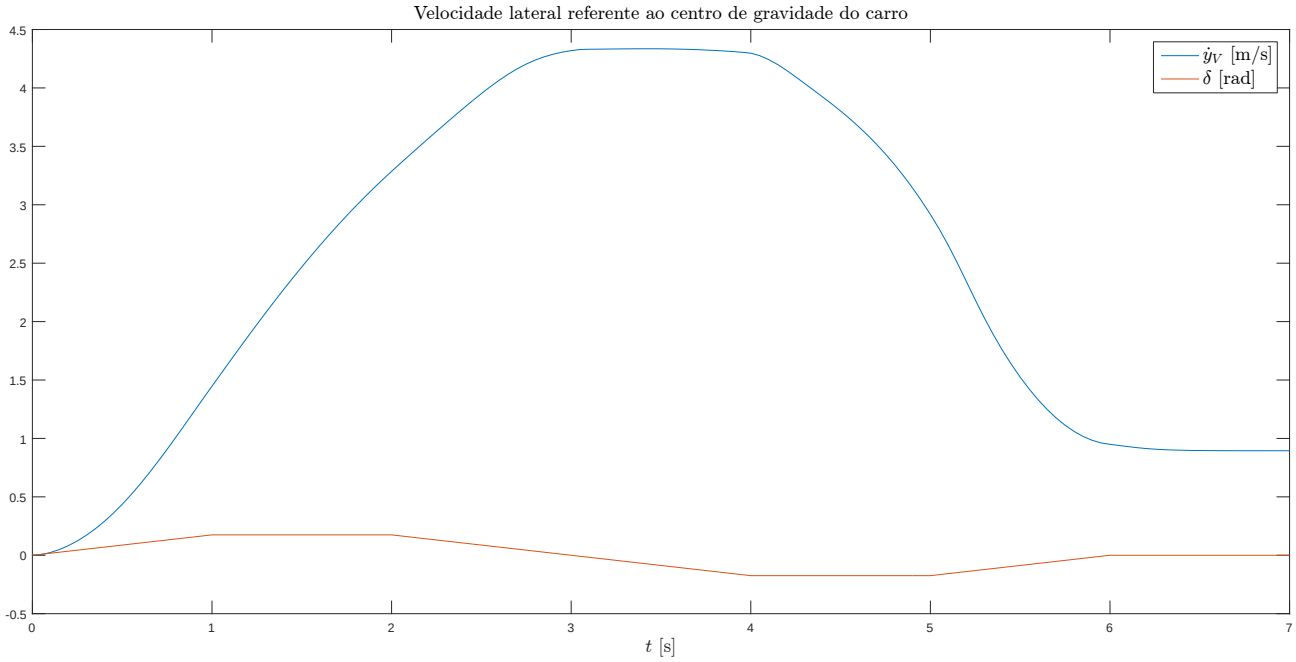


Figura 3.7: Gráfico da velocidade lateral do carro obtido na simulação com a entrada composta. Fonte: Autor.

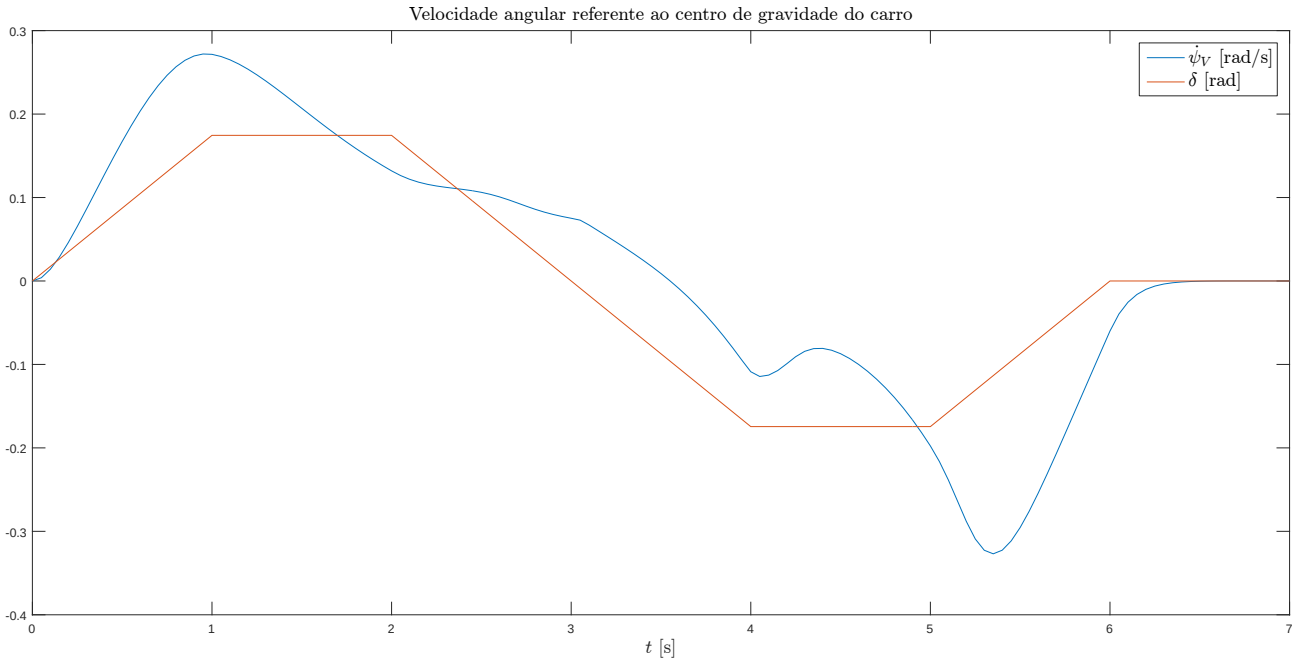


Figura 3.8: Gráfico da velocidade angular do carro obtido na simulação com a entrada composta. Fonte: Autor.

Nessa simulação é esperado que o carro faça uma curva suave à esquerda e logo após comece a desfazer essa curva, alinhando então com a direção original, porém com um deslocamento em y_E. Com relação à velocidade lateral é esperado que esta suba em S por 3s, ponto no qual ela deve descer em S por 3s também, voltando à $0\mathrm{m\,s^{-1}}$. Já a velocidade angular, espera-se que essa siga o formato da entrada com uma dinâmica atrasada.

Como pode ser visto na Figura 3.6, o veículo descreveu uma trajetória próxima àquela esperada, porém sem alinhar totalmente ao final, com o eixo x. Já na Figura 3.7 é mostrado que a velocidade lateral seguiu um comportamento próximo ao esperado, porém não retornou aos $0\mathrm{m\,s^{-1}}$, o que levou o carro à terminar a simulação virado levemente à esquerda. Já a velocidade angular seguiu de uma forma geral, como mostrado na Figura 3.8, a forma da entrada, exibindo, porém, dinâmicas não esperadas.

O fato da simulação apresentar diferenças do resultado previsto era esperado, visto que o modelo apresenta não linearidades bem pronunciadas. De fato, a proximidade entre simulação e previsão representa um resultado muito positivo, visto que as previsões feitas foram baseadas somente em experiências do cotidiano.

3.3.2 Linearização do Modelo *Single-Track*

Ao fazer a análise de controlabilidade do modelo *single-track* linearizado, foi observado que a linearização possuía um estado não controlável. Essa análise foi feita por meio da matriz de controlabilidade

$$C = \begin{bmatrix} 0 & 0 & 0 & -1.6\times10^{-4} & -8.4\times10^{-7} & -5.2\times10^{-9} \\ 0 & 14.1166 & 0.0935 & 6.2\times10^{-4} & 4.1\times10^{-6} & 2.7\times10^{-8} \\ 0 & 0.2398 & -2\times10^{-4} & 1.7\times10^{-7} & -1.5\times10^{-10} & 1.2\times10^{-13} \\ 0 & 0 & -1.6\times10^{-4} & -8.4\times10^{-7} & -5.2\times10^{-9} & -3.2\times10^{-11} \\ 14.1166 & 0.0935 & 6.2\times10^{-4} & 4.1\times10^{-6} & 2.7\times10^{-8} & 1.8\times10^{-10} \\ 0.2398 & -2\times10^{-4} & 1.7\times10^{-7} & -1.5\times10^{-10} & 1.2\times10^{-13} & -1.1\times10^{-16} \end{bmatrix} \tag{3.39}$$

que possui posto 5, portanto o sistema não tem todos os estados controláveis, de fato, apenas um deles não é controlável. Fazendo a decomposição do sistema entre os estados estáveis e instáveis, pode-se fazer uma outra análise. O sistema instável decomposto é o seguinte:

$$A = \begin{bmatrix} 0 & 0 & 0.0625 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 0.0061 & 0.0428 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.0066 \end{bmatrix}, \quad B = \begin{bmatrix} -64150 \\ -6.789 \\ 871.2 \\ -141.2 \\ 3.541 \end{bmatrix} \tag{3.40}$$

O resultado obtido indica que o sistema possui 5 estados instáveis e se estes forem os estados controláveis identificados na primeira análise, então o estado restante de fato não é controlável, porém é alcançável. Analisando a matriz de controlabilidade do sistema instável

$$C = \begin{bmatrix} -64148 & 54.4483 & -0.0462 & -2.81 \times 10^{-4} & -1.71 \times 10^{-6} \\ -6.7891 & 1.7703 & 0.0117 & 7.72 \times 10^{-5} & 5.09 \times 10^{-7} \\ 871.172 & -0.7394 & -0.0045 & -2.73 \times 10^{-5} & -1.66 \times 10^{-7} \\ -141.233 & 0 & 0 & 0 & 0 \\ 3.5407 & 0.0234 & 1.54 \times 10^{-4} & 1.02 \times 10^{-6} & -6.72 \times 10^{-9} \end{bmatrix} \tag{3.41}$$

pode-se observar que ela também não é posto completo, o que configura o sistema linearizado como instável. Esse resultado se faz inesperado, pois é bem conhecido na literatura que veículos terrestres são estáveis sob condições normais. Então é possível que esse resultado tenha ocorrido devido a algum erro na modelagem ou na linearização do sistema não linear.

3.3.3 Controle de Velocidade

A resposta do sistema controlado pode ser observada na Figura 3.9, e o sinal de controle u do sistema pode ser observado na Figura 3.10.

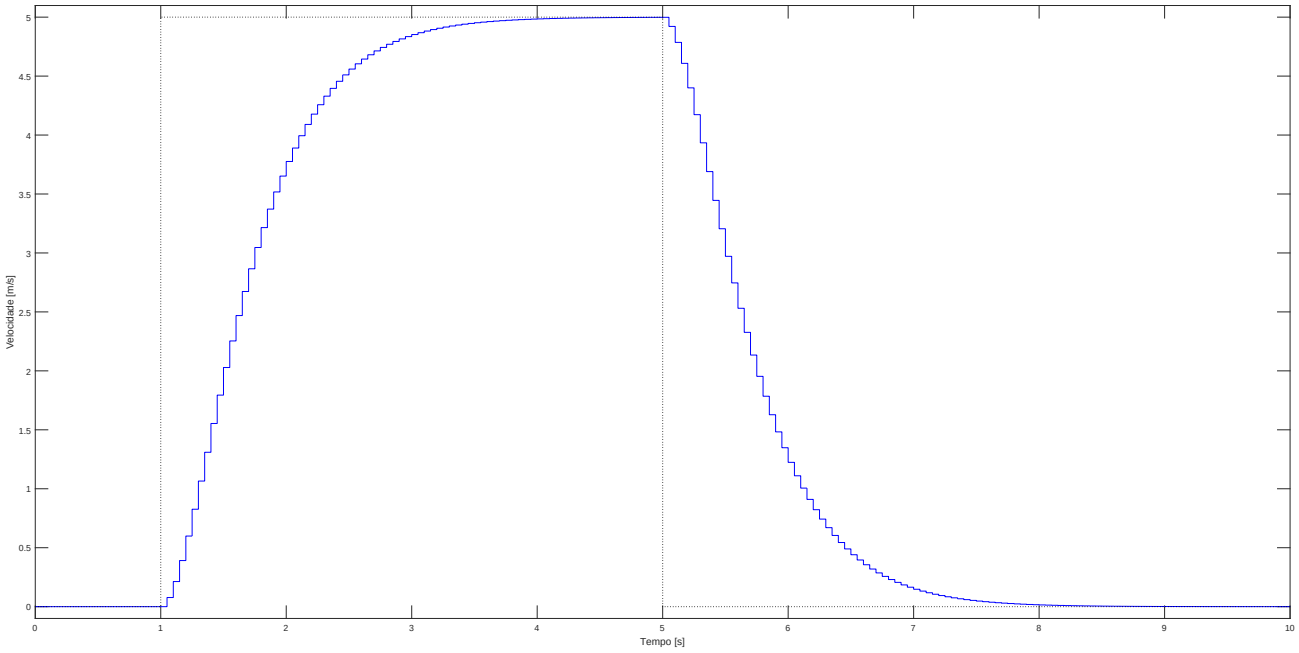


Figura 3.9: Resposta no tempo à aceleração e desaceleração. Fonte: Autor.

O desempenho do controlador, como esperado, seguiu as especificações. O tempo de acomodação obtido foi de 1.75s, um pouco superior ao especificado, visto que o sistema foi projetado levemente super-amortecido, e o tempo de subida foi de 1.25s.

É importante porém fazer uma análise sobre o valor máximo do sinal de controle. Pode-se observar que o pico do sinal de controle se aproxima de 5000N. Essa força é equivalente à exercida por um cabo puxando o veículo, como mostra a Figura 3.3, ou seja, é a força que aparece no contato pneu solo. Apesar de parecer um valor alto, é necessário fazer uma análise em cima das conversões de torque que acontecem no trem de força e a conversão torque-força que o pneu faz, para poder avaliar esse valor.

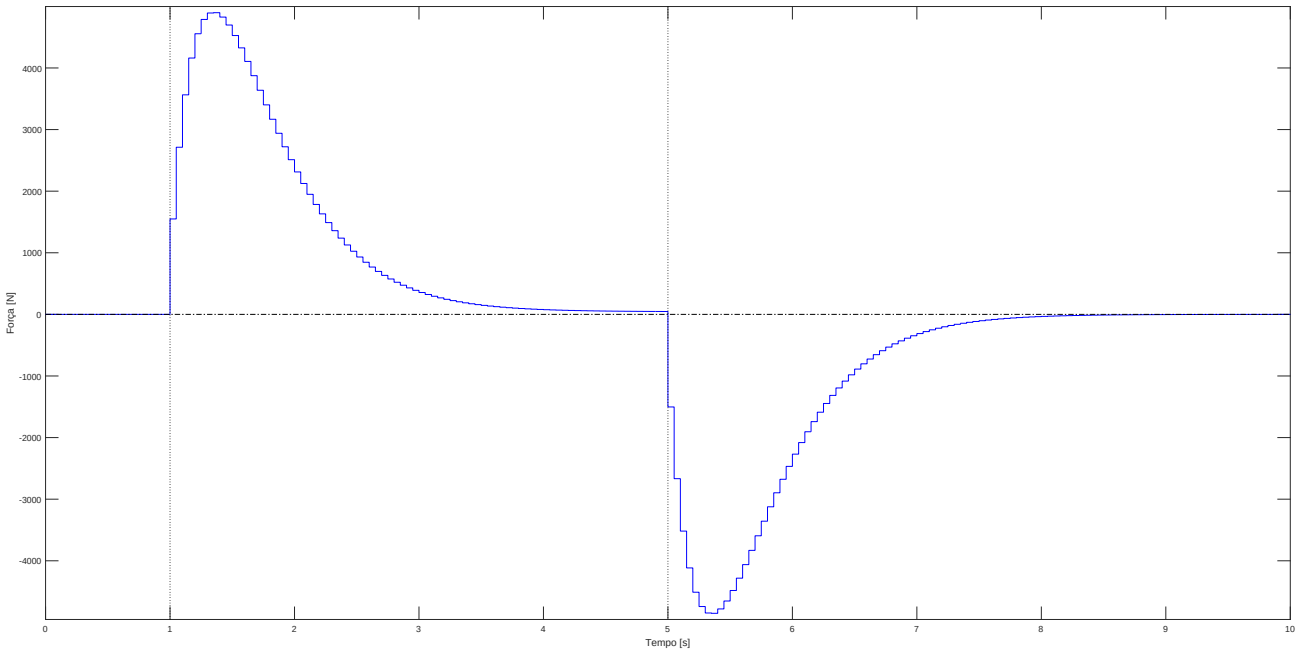


Figura 3.10: Sinal de controle durante a aceleração e desaceleração. Fonte: Autor.

Para chegar à um valor de torque realista é necessário primeiro encontrar o equivalente dessa força como torque no semieixo. Os dados utilizados aqui foram retirados de [Cars Data 2018c] e equivalem ao veículo Palio Weekend 75.

$$T_{se} = Fr_{dyn} = 5000\text{N} \cdot 0.31\text{m} = 1550\text{N m}$$

É necessário, então, computar a influencia do diferencial. Supondo uma relação de engrenagens no diferencial de 3.93 : 1 e eficiência de 95%, então o torque T_{out} no eixo de saída da transmissão seria:

$$T_{se} = T_{out} \cdot 3.93 \cdot 0.95 \quad \Rightarrow \quad T_{out} \approx 415\text{N m}$$

Por último é necessário levar em conta a relação de engrenagens no câmbio. Supondo que o veículo esteja conduzindo em primeira marcha, com uma relação de engrenagem de 3.91 e eficiência de 95%, então o torque T_{in} no eixo de entrada da transmissão vale:

$$T_{out} = T_{in} \cdot 3.91 \cdot 0.95 \quad \Rightarrow \quad T_{in} \approx 111\text{N m}$$

O que representa um resultado válido, visto que esse carro possui torque máximo de 104N m. Esse resultado se repete para dados de outros veículos, como por exemplo, o Corsa 1.2 Comfort 2000-03 [Cars Data 2018b] e o Uno 1.1 Start 1993-94 [Cars Data 2018a]. Dados adicionais sobre valores de relação de engrenagens para diferenciais pode ser encontrados em [Axles & Gear 2018; Jonathan Decker 2018; National Tire & Wheel 2018]

3.4 Conclusão do Capítulo

Neste capítulo foi abordado a continuação da modelagem dinâmica de um veículo iniciada na fundamentação teórica, conectando todas as partes para conseguir o modelo final, e a linearização deste modelo proposto.

Foi também abordado a modelagem em espaço de estados e controle por realimentação de estados com ação integral de um sistema de velocidade de cruzeiro em veículos.

E por fim, foi feito uma analise e discussão das simulações realizadas e dos resultados obtidos.

Pode-se concluir que, com exceção daquele mostrado em 3.3.2, os resultados obtidos foram positivos, mostrando que os modelos descrevem de forma satisfatória o comportamento observado e também, como mostrado em 3.3.3, que o sinal de controle obtido da realimentação de estados para o sistema de velocidade de cruzeiro é completamente compatível com valores reais.

Capítulo 4

Agentes

Existe atualmente uma quantidade considerável de *frameworks* que permitem trabalhar com SMA seguindo as especificações FIPA. Como apresentado na Seção 2.4.9, o JADE é um desses *frameworks* e foi escolhido para a realização desse trabalho pois por ser desenvolvido em JAVA, apresenta o ambiente de programação orientada a objetos, familiar para o autor e por já ter sido utilizado em diversos trabalhos acadêmicos [Kristensen e Smith 2015; Sandita e Popirlan 2015; Spanoudakis e Moraitis 2006].

O objetivo nessa parte é criar agentes capazes de operar em conjunto, além de integrar uma ferramenta de visualização 2D simples para o cruzamento.

4.1 Topologia do Cruzamento

A topologia escolhida para trabalhar com o cruzamento está exposta na Figura 4.1.

Dessa forma o cruzamento possui quatro vias, sendo uma para conduzir em cada sentido. Foram então implementados agentes para poder operar nesse ambiente. Foram idealizados quatro tipos de agentes: *agentcreator*, *car*, *controller* e *tracker*. Suas funções são como descrito a seguir:

- ***agentcreator*:** Esse agente é responsável somente por criar novos agentes do tipo carro. O motivo de implementar esse agente é que dessa forma tem-se um modo eficiente de controlar alguns parâmetros do sistema, como por exemplo a taxa com que novos veículos são adicionados ao cruzamento, a velocidade e localidade que eles nascem;

- ***car*:** Esse agente é responsável por controlar seus parâmetros, evitar acidentes com outros veículos e passar suas informações de posição para o *tracker* e para o *controller*;

- ***controller*:** Esse agente é responsável pelo algoritmo de decisão e informar qual veículo deve entrar no cruzamento;

- ***tracker*:** Esse agente é responsável por rastrear todos os carros nas imediações do cruzamento e fazer a troca de informações entre diferentes instâncias do agente *car* quando necessário. Além disso ele é responsável por instanciar e passar informações para a GUI[1].

[1]Interface Gráfica de Usuário, sigla do inglês.

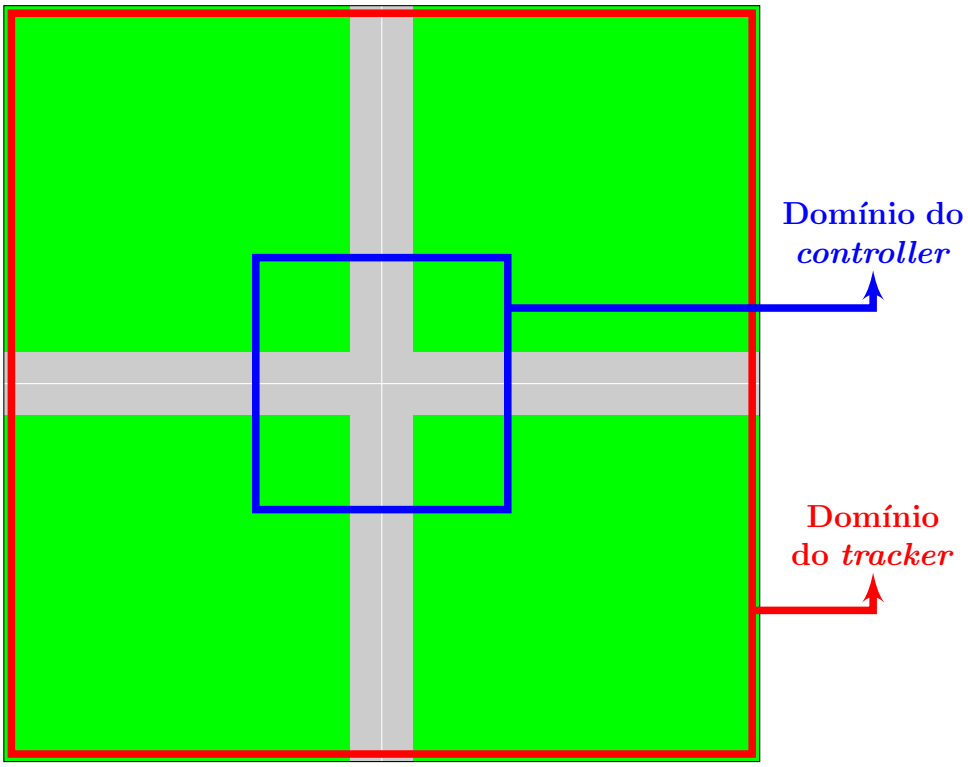


Figura 4.1: Topologia escolhida para o cruzamento. Fonte: Autor

O motivo na escolha desse modelo de topologia é a facilidade que se tem em expandir o sistema, simplesmente concatenando cruzamentos de forma a construir uma rede complexa.

4.1.1 Topologia Macroscópica

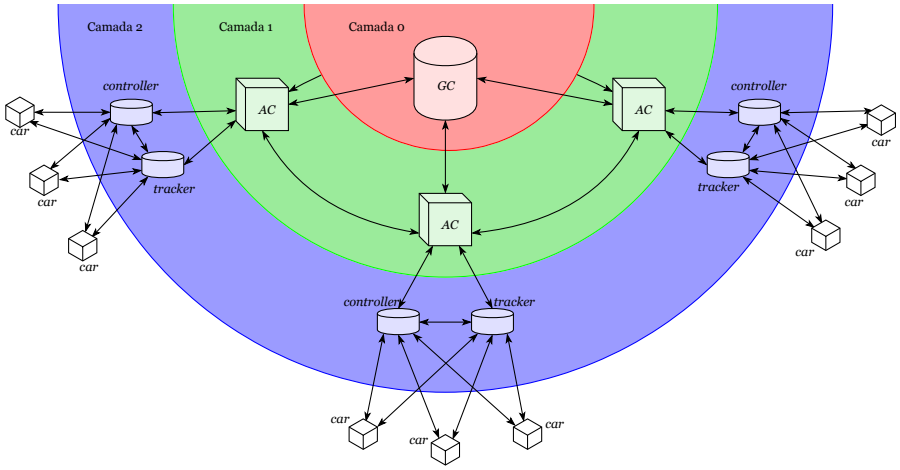


Figura 4.2: Topologia para uma rede de tráfego completa. Fonte: Autor

Pensando em um ambiente macro, seria possível então, utilizando essa topologia, conectar cruzamentos para formar uma rede complexa de transito em que os agentes *car* simplesmente movem-se entre APs. A partir do momento que os veículos não necessitam mais de serem

instanciados pelo *agentcreator*, esse pode assumir um papel maior na cena macroscópica, se tornando um coordenador entre cruzamentos e trabalhando junto com um controlador geral responsável por gerir toda a rede de tráfego e auxiliando veículos que estivessem trocando de AP. Essa topologia está exibida na Figura 4.2.

Nessa topologia cada *agentcreator* (AC) seria responsável por fazer a comunicação entre seu cruzamento e os cruzamentos adjacentes, além de comunicar com o *generalcontroller* (GC). Assim um *tracker* poderia, ao identificar sobrecarga de atividade no cruzamento, pedir ao AC que comunique com o GC para obter rotas alternativas para o transito na região afetada.

A Figura 4.3 demonstra como os cruzamentos poderiam ser concatenados para formar uma rede complexa de trânsito. Não há necessidade de serem todos cruzamentos de quatro pistas em cruz, seria possível concatenar qualquer tipo de cruzamento uns com os outros. Esse benefício é um resultado direto da utilização de IC Distribuída, pois como cada micro problema tem seu conjunto de agentes (*controller* e *tracker*) trabalhando para resolvê-lo, o GC pode trabalhar para otimizar o trânsito sem se preocupar com o que acontece em cada nó.

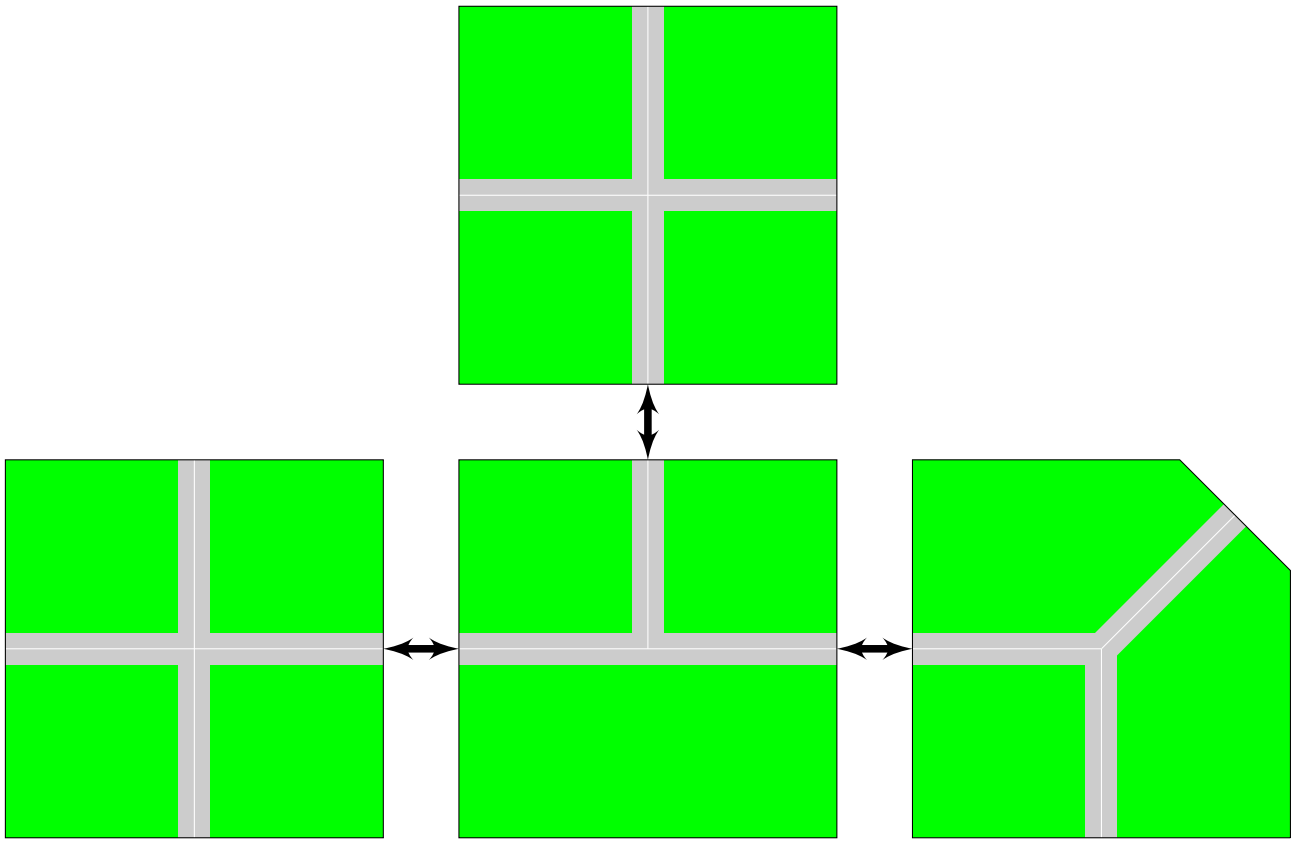


Figura 4.3: Cruzamentos concatenados. Fonte: Autor

4.2 Desenvolvimento dos Agentes

O desenvolvimento dos agentes foi feito pensando na topologia apresentada nas seções anteriores deste capítulo. Os primeiros agentes a serem implementados e testados foram o *agentcreator* e o *car*, por serem fundamentais para o funcionamento de ambos os outros agentes. Note que

todos os agentes implementados possuem as funcionalidades básicas, como por exemplo, se registrar com o DF assim que inicializado e a função `protected void takeDown()`, que realiza uma limpeza antes do agente ser destruído.

A única funcionalidade implementada no *agentcreator* foi a de criar agentes do tipo carro em uma das vias com uma certa velocidade que pode ser variada.

Em suas primeiras versões o agente *car* tinha só uma funcionalidade, calcular sua posição e imprimir seus dados na tela regularmente. Pretendia-se que esse agente pudesse fazer curvas nos cruzamentos e trocar de faixa em pistas com múltiplas faixas. Porém devido aos resultados apresentados na Seção 3.3.2, essa funcionalidade não pode ser adicionada. No entanto o controle de velocidade, como apresentado na Seção 3.2, foi implementado com sucesso nos agentes.

Após a realização dos testes iniciais com o AC e o *car*, foi criado o *tracker*. Suas funcionalidades iniciais eram:

- Se inscrever com o DF para receber atualizações de todos os agentes do tipo *car* que se registrarem com ele;
- Oferecer o serviço de *tracking* à todos os carros que entrarem em seu domínio;
- Receber informações dos veículos e armazena-las; e
- Informar a distância de cada carro com o veículo da frente.

Note que para as funções que requerem a participação de outro agente, o método equivalente foi implementado no agente em questão, como por exemplo as funcionalidades de comunicação do *tracker* com os carros requer que os agentes *car* consigam compreender as mensagem e respondê-las.

A princípio foi idealizado que cada carro iria procurar o agente *tracker* e se inscrever com ele, porem o método implementado reduz o custo computacional do sistema reduzindo drasticamente o número de mensagens enviadas entre agentes.

Como medida de segurança, foi implementado no *tracker* uma condição caso os agentes *car* recusem o serviço de *tracking*, para que ele solicite ao AMS que o agente seja destruído.

Com a implementação do *tracker*, foi possível implementar algumas funcionalidades que antes não eram. No caso do AC, ele passou a poder conferir com o *tracker* a situação das vias antes de criar um agente nela. Já os carros passaram a poder reagir à mudanças de velocidade do veículo que está à frente dele na via.

Por fim foi implementado o *controller*. Esse agente possui uma única função; decidir quais veículos podem passar no cruzamento e quais devem esperar, e para que esse algoritmo possa funcionar, um banco de dados com as informações dos agentes que estão dentro do seu domínio. A princípio o algoritmo de seleção utilizado foi um simples Primeiro a Chegar, Primeiro a Sair. Esse algoritmo simplesmente armazena a ordem de chegada dos veículos ao cruzamento, e após o término da passagem do carro que está atualmente cruzando, envia a bandeira verde ao próximo agente.

Com todos os agentes implementados e funcionais foi possível criar uma GUI para o sistema. Essa GUI consiste de duas classes, `Jade_gui` e `Jade_window`, em que esta é responsável por criar a janela da GUI e fazer a comunicação entre o agente e a GUI, e aquela é responsável por fazer

os desenhos na tela. Essas classes foram desenvolvidas utilizando as ferramentas AWT e Swing do Java para criar os componentes gráficos e desenhar o ambiente na tela.

Para que a GUI funcione, algum agente deve instanciar a classe `Jade_window`. O *tracker* foi escolhido para isto, visto que ele possui as informações de todos os agentes presentes no cruzamento e portanto poderia passa-las facilmente para a GUI.

Deste ponto em diante algumas melhorias foram feitas nos códigos. Foi adicionado uma funcionalidade no *tracker* para ele consultar com o DF o AID do *controller* e passar esse AID para todos os carros no primeiro contato. Antes de implementar essa mudança cada carro buscava no DF o agente *controller* assim que estava para entrar no domínio dele. Essa mudança reduz a 1 o número de buscas no DF por agentes do tipo *controller*.

Foi adicionado no *car*, *tracker* e na GUI métodos para lidar com veículos que chegaram ao fim do ambiente suportado pela AP que eles estão. Os veículos ao chegarem no fim da pista executam a função `doDelete()`, equivalente aos métodos *Destroy/Quit* do ciclo de vida FIPA mostrado na Figura 2.15. O *tracker* reage à mensagem do DF atualizando o registro dos agentes e propaga a informação para a GUI.

Também foi implementada uma nova versão da lista de dados dos veículos no *tracker*, utilizando apenas uma entrada por veículo, ao invés das listas pareadas[2] que era utilizadas até este ponto.

Foi implementado no *controller* uma versão melhorada do algoritmo de decisão. Essa nova versão reconhece agentes conseguintes que estão na mesma via, ou em vias opostas, e deixa eles cruzarem juntos.

4.3 Simulações e Resultados Obtidos

Nessa seção serão apresentados e discutidos os resultados obtidos com o desenvolvimento dos agentes.

Um primeiro resultado, de baixa importância se comparado aos outros, foi o dos testes de eliminação de agentes *car* que não aceitem o serviço de *tracking*. Por sí só esse resultado não tem muita importância, porém ele indicou um fato importante: o DF notifica os agentes inscritos nele de qualquer mudança realizada pelos agentes, incluindo a remoção do registro incluída na função `takeDown()`. Isso demonstrou a importância de implementar um método capaz de reconhecer as diferentes situações notificadas pelo DF.

Um resultado interessante foi o obtido nos primeiros testes com a GUI em que foi possível observar com clareza a formação e propagação das *Stop-and-Go Waves*. Esse é um fenômeno muito estudado no campo da modelagem matemática de Fluxos, e recentemente com diversas propostas de solução, pois ele aparece recorrentemente em diversas situações [Colombo e Groli 2004; Herty e Illner 2007; Stern et al. 2017; J. Wang, Chai e Xue 2016]. Esse fenômeno consiste na formação de uma onda que percorre o tráfego na direção contrária do fluxo, efetivamente reduzindo o fluxo de veículos e aumentando a intensidade de engarrafamentos. Um demonstrativo com 4 *frames* da simulação demonstrando a propagação da *stop-and-go wave* pode ser observado na Figura 4.4.

[2] Cada entrada da lista continha dados de dois veículos

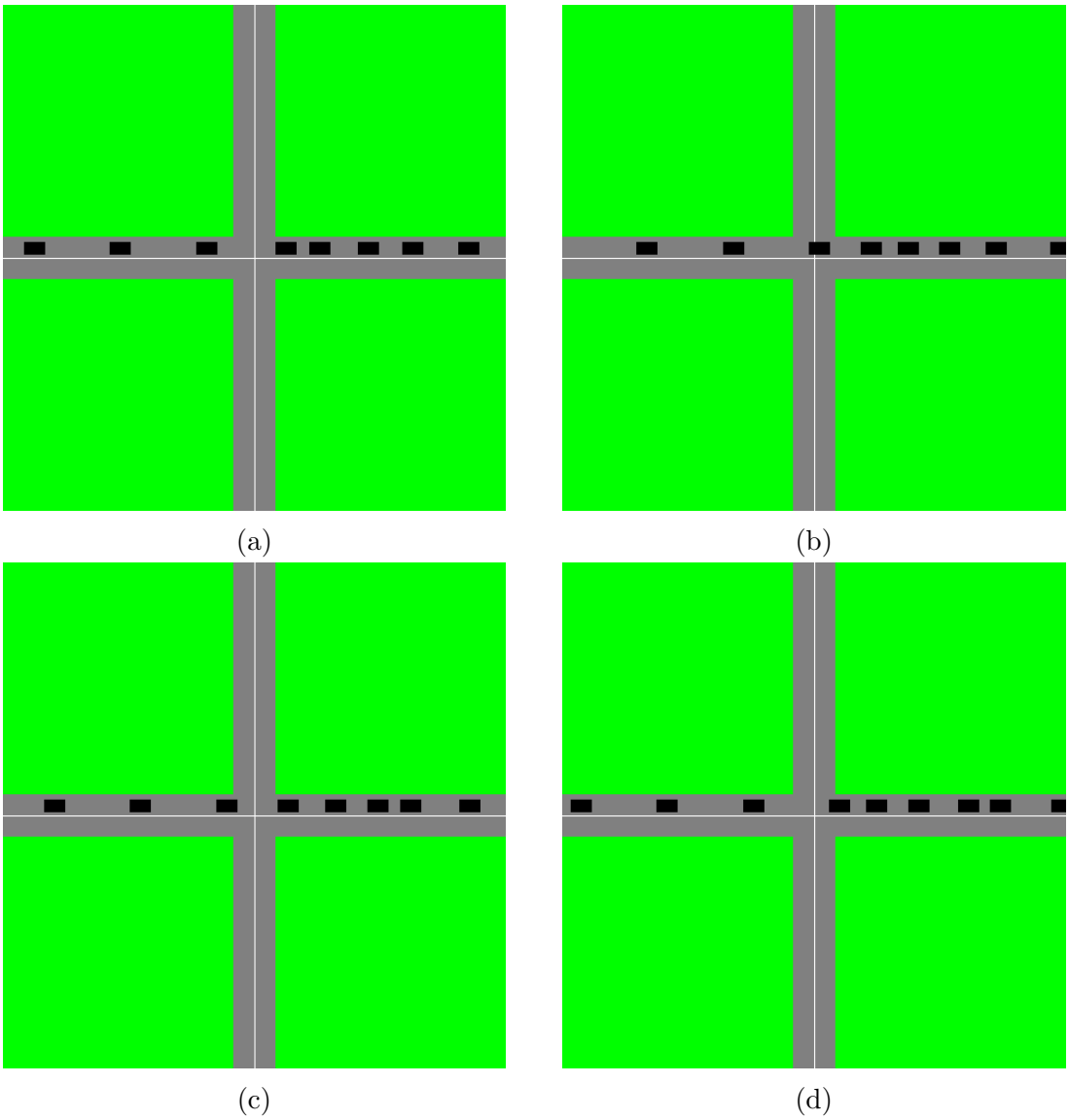


Figura 4.4: Simulação que resultou na observação da *Stop-and-Go Wave*. Fonte: Autor.

Os próximos dois resultados são referentes à atuação do *controlador* com entrada de veículos aleatoriamente pelas vias, antes e depois da implementação do algoritmo de decisão melhorado. Uma sequencia de 6 *frames* capturadas durante a simulação do algoritmo antigo está exibida na Figura 4.5.

Nesta versão do algoritmo de decisão o *controller* sempre esperava o veículo que estava atravessando o cruzamento terminar totalmente de cruzar antes de autorizar a passagem do próximo veículo na fila. Além disso a abordagem utilizada para definir o término da passagem era altamente conservador no sentido que o agente esperava o veículo sair totalmente do cruzamento para considerar que ele havia completado a passagem.

Na versão melhorada do algoritmo o agente além de utilizar uma abordagem menos conservadora na definição do término da passagem, a saída da primeira metade do veículo do cruzamento, também compara a via do próximo veículo da fila, liberando a passagem desde que este esteja na mesma ou na via oposta. é importante notar que esse comportamento não se limita à apenas

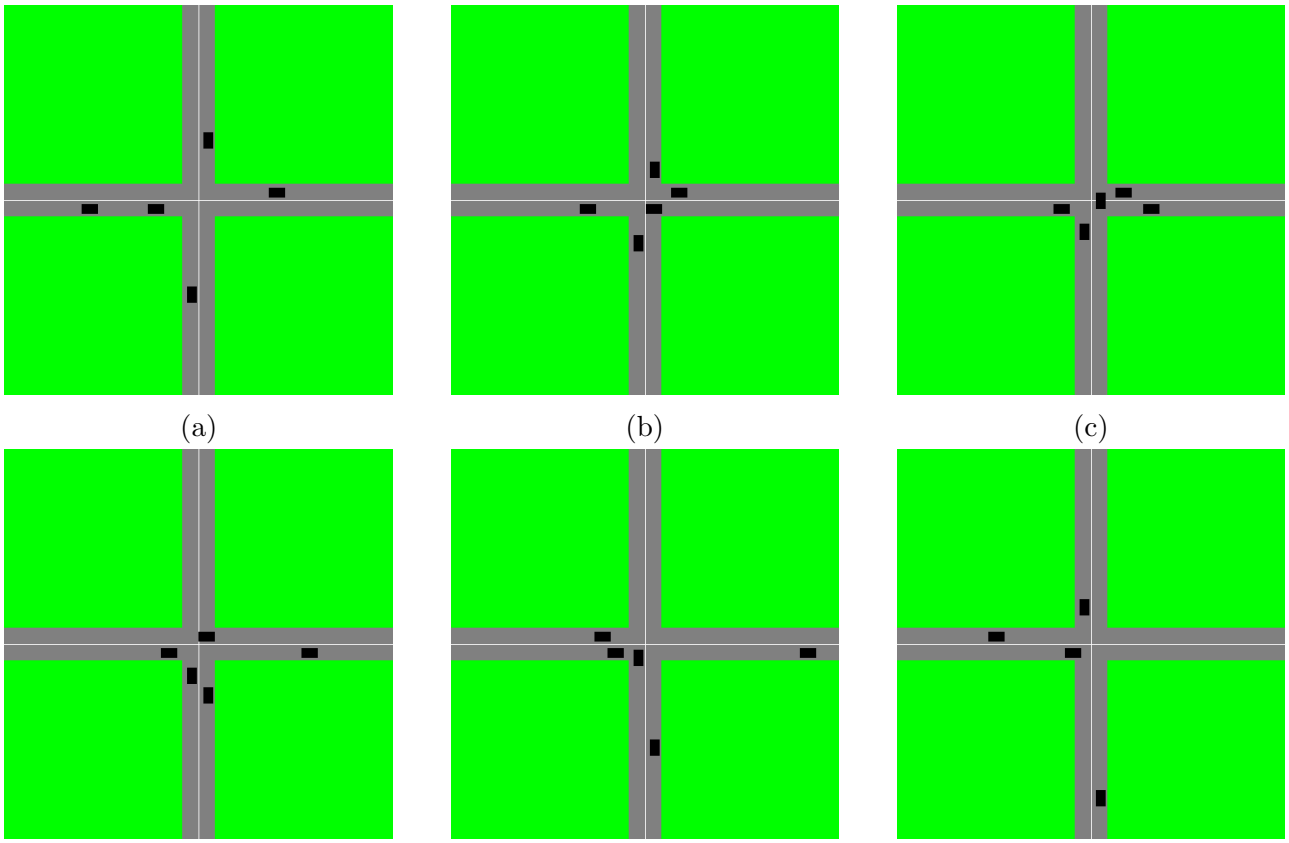


Figura 4.5: Simulação com entrada aleatória de veículos com o algoritmo antigo. Fonte: Autor.

um veículo após o primeiro a começar a passagem pois ele utiliza de referência o ultimo carro a entrar no cruzamento, ou seja, esse método admite encadeamento de autorizações para cruzar.

O resultado obtido dessa simulação está exibido na Figura 4.6 como uma sequencia de 9 *frames*. Pode-se observar nas figuras 4.6d, 4.6e, 4.6g e 4.6h que o *controller* permite a passagem de múltiplos veículos ao mesmo tempo, ocasionando em uma redução do tempo de espera. É importante ressaltar que na simulação demonstrada pela Figura 4.5 o intervalo entre *frames* é de 2s, já na simulação apresentada do algoritmo melhorado esse intervalo é de 1s.

4.4 Conclusão do Capítulo

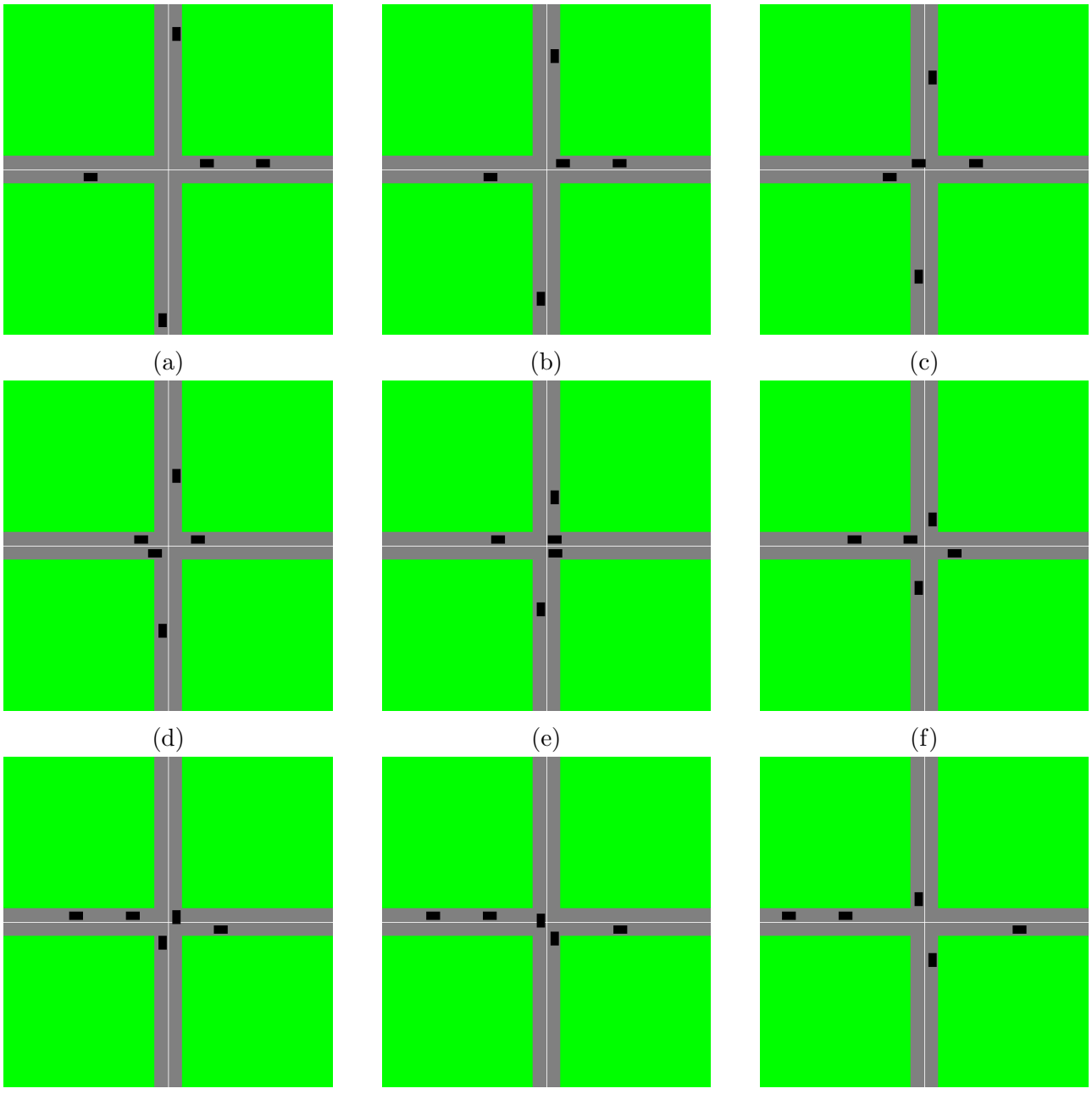
Nesse capítulo foi introduzida a topologia idealizada para o sistema multi-agente que foi implementado. Adicionalmente apresentado a idealização desta topologia em larga escala, utilizando conceitos de sistemas distribuídos para propor um sistema simples capaz de modelar e atuar uma malha viária completa.

Foi também discutido os passos envolvidos no desenvolvimento de cada um dos agentes que compõe o SMA bem como do ambiente de testes proposto.

Por fim foi feito uma discussão sobre as diferentes simulações realizadas, bem como a analise dos resultados.

Conclui-se então que o trabalho desenvolvido nessa etapa produziu resultados satisfatórios.

O Sistema Multi-Agente funcionou como pretendido e o ambiente de simulações e testes proveu uma forma de visualizar com facilidade as ações dos agentes. A modularidade do sistema também proporciona uma grande facilidade de implementar melhorias neste, fato muito importante para trabalhos que envolvem tecnologias recentes.



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

Figura 4.6: Simulação com entrada aleatória de veículos com o algoritmo melhorado. Fonte: Autor.

Capítulo 5

Conclusões e Proposta de Continuidade

Neste capítulo serão apresentadas as conclusões deste texto e as propostas de continuidade para este trabalho, assim como o cronograma proposto para a segunda etapa.

5.1 Conclusão

Neste trabalho realizou-se inicialmente um estudo preliminar sobre modelagem da dinâmica de um veículo objetivando determinar quais assuntos iria ser necessário estudar a fim de obter um modelo final. Durante esta pesquisa foi encontrado uma modelagem na qual o modelo deste trabalho se baseia. Como resultado foi verificado que era necessário fazer um estudo sobre as forças que atuam no contato pneu-pista.

Nesse tópico das forças que atuam no pneu foi descoberto o modelo estacionário de Pacejka, também conhecido por Fórmula Mágica, para essas forças. Para o entendimento desta fórmula foi estudado as influências de características construtivas do pneu na dinâmica deste, as influências das cargas normais e do escorregamento nas forças.

Após concluído estes estudos foi possível fazer as alterações necessárias no modelo encontrado, para chagar no modelo apresentado neste trabalho.

Feito isso iniciou-se a pesquisa por fontes para obter os dados necessários para realizar simulações do modelo proposto. Esses dados incluem os parâmetros da Fórmula Mágica, características estruturais, como distância entre eixo e massa, e parâmetros inerciais do veículo, que são o momento de inercia em torno do eixo z e a localização do centro de gravidade. Para os valores relacionados ao veículo foi encontrado uma base de dados que contêm as informações necessárias para diversos modelos de carros, vans e *pickups*.

A proposta de linearização deste modelo *single-track* obtido não obteve o sucesso esperado, pois o modelo linearizado apresentou um estado instável e não controlável, o que inviabiliza a proposta de controle e utilização do mesmo no sistema multi-agente. O motivo desse resultado inesperado não é conhecido até o momento da escrita deste texto.

O modelo de velocidade que seria aplicado em cascata com o modelo dinâmico para poder-se obter uma dinâmica completa do veículo foi obtido e controlado com sucesso, atingindo os índices de performance estabelecidos e sem comprometer a amplitude do sinal de controle aplicado no sistema.

Inicialmente no quesito do desenvolvimento de agentes para um SMA foi realizado um estudo sobre as normas, ferramentas e metodologias existentes. Após esse estudo inicial foi decidido trabalhar com agentes segundo as normatizações FIPA, e para tal foi decidido utilizar a ferramenta JADE.

A próxima parte nessa etapa foi o estudo mais aprofundado das normas FIPA e instalação da ferramenta escolhida para poder dar início aos trabalhos de programação dos agentes. Foram encontrados alguns problemas na configuração e utilização da ferramenta proposta, o JADE, porém após a solução destes problemas o desenvolvimento fluiu sem demais complicações.

Foram desenvolvidos quatro tipos diferentes de agentes que devem interagir entre si para gerenciar um micro universo, um único cruzamento, com a possibilidade de expansão para aplicar o mesmo conceito em cadeia e montar macro universos complexos, como por exemplo a rede completa de trânsito de uma cidade.

Por ultimo foi desenvolvido um ambiente de testes com visualização gráfica 2D para poder realizar as simulações com os agentes. Esse ambiente de testes foi desenvolvido utilizando as ferramentas AWT e Swing do Java para criar a janela e desenhar o ambiente e os carros no painel.

A implementação dos agentes na topologia proposta ocorreu sem maiores problemas e obtendo resultados altamente satisfatórios com a utilização do ambiente de testes proposto.

Por fim pode-se afirmar que, apesar dos resultados inesperados com relação ao modelo da dinâmica do veículo, o objetivo geral deste trabalho foi atingido.

5.2 Discussão de Melhorias e Propostas de Continuidade

O trabalho em sua forma presente está aberto a melhorias em diversas áreas, algumas das quais serão discutidas aqui.

5.2.1 Modelagem

No quesito do modelo diversas melhorias podem ser feitas, algumas relativamente simples, outras mais complexas e que renderiam por si só trabalhos completos.

É necessário encontrar uma solução para o resultado inesperado na questão do modelo *single-track*. A solução mais simples seria propor um modelo já linear, inspirado em algum dos diversos modelos existentes na literatura.

Alternativamente pode-se estudar a utilização do modelo não linear apresentado ou fuzzy, com um controlador desenvolvido utilizando alguma das técnicas de controle não linear. Esse tipo de implementação tem o potencial de gerar trabalhos completos e publicações sem a necessidade de assuntos adicionais.

Outra proposta também em cima do modelo é estudar a aplicação de alguma técnica de modelagem e controle por redes neurais. Essa área de pesquisa, assim como programação orientada à agentes, é muito nova na comunidade acadêmica e pode gerar resultados muito satisfatórios.

5.2.2 Controle de Velocidade

O modelo proposto para controle de velocidade atualmente é um simples seguidor de referência e, portanto, não faz uso das ferramentas apresentadas pelas habilidades comunicativas dos agentes para obter resultados melhores. É possível nessa área aplicar alguma variação de um modelo de velocidade ótima, ou outros tipos de modelos seguidores de veículos para obter um melhor tempo de reação dos agentes.

Alternativamente, seguindo a linha de raciocínio da seção anterior, pode-se utilizar um modelo não linear completo com controle de velocidade integrado. Nessa situação seria possível implementar um modelo seguidor de veículo em cascata com o controlador não linear, fornecendo a referência de velocidade.

5.2.3 Agentes

Nesse tópico cabem diversas ressaltas sobre melhorias que podem ser implementadas no SMA já existente.

5.2.3.1 Universo Macro

Uma das observações mais importantes no tópico dos agentes seria implementar a topologia macroscópica apresentada anteriormente no trabalho. Esse sistema pode ser desenvolvido tanto em um único computador quanto em uma rede de computadores distribuída.

As principais mudanças que devem ser realizadas para efetivar essa mudança são a implementação de um método de mobilidade de plataformas no *car*, a conversão do *agentcreator* em um agente de cooperação entre cruzamentos e ator do Controlador Geral na topologia macroscópica. Além disso seria necessário implementar mais módulos de comunicação entre os agentes já existentes.

5.2.3.2 Classe *Template*

Em todos os agentes propostos algumas constantes e métodos se repetem, ou são fundamentais para a definição do espaço 2D em que os agentes estão inseridos. Uma grande melhoria no sistema em geral e de extrema importância para a implementação da topologia discutida na Seção acima seria o desenvolvimento dessa classe *template* que os agentes possam implementar para poder garantir uma comunicação facilitada, habilitar a utilização de agentes totalmente heterogêneos, e para transmitir informações de layout aos veículos.

5.2.3.3 Agente *car*

Uma das principais melhorias que pode ser realizada nessa classe é a comunicação consigo mesma. No modelo presente todos os carros implementam a mesma classe, tornando a comunicação entre agentes *car* uma tarefa relativamente simples.

A possibilidade de introdução de agentes heterogêneos do sistema aumenta consideravelmente a complexidade dessa situação. Porém a partir do momento em que uma classe *template* esteja

implementada a introdução de agentes heterogêneos no sistema se torna mais simples devido a utilização de constantes da classe para padronizar a comunicação entre veículos.

Essa possibilidade de comunicação entre carros adiciona um grau de liberdade a mais no sistema, visto que nesse ponto seria possível veículos proativamente agir em conjunto para evitar acidentes, negociar entre sí para resolver problemas de trânsito.

5.2.3.4 Agente *controller*

Nesse agente é possível trabalhar diversas modificações para aprimorar seu funcionamento.

Com relação ao algoritmo de decisão é possível modifica-lo de diversas formas. Sem modificar a base Primeiro a Chegar, Primeiro a Sair é possível fazer adições no sistema de forma a melhorá-lo. Uma adição simples seria a implementação de uma busca rasa na lista por outros veículos que estão esperando para cruzar e estão nas faixas de mesmo sentido do(s) veículo(s) que estão cruzando, reduzindo, em geral, o tempo de espera na fila.

Outras duas possibilidades para esse tópico são a transformação da questão do fluxo no transito em um problema de grafos e aplicar uma ferramenta de otimização para resolvê-lo. A segunda opção seria aplicar um algoritmo de *machine learning* e treinar a inteligência artificial, para otimizar o tempo de espera de veículos na fila.

Outra adição benéfica qualquer outro método de decisão, é um detector de colisão. Com um algoritmo dedicado a detectar rotas de colisão entre os carros o *controller* poderia com mais facilidade avaliar a possibilidade de autorizar o cruzamento de veículos com tempo menor de espera.

Apêndice A

Códigos

A.1 Jade_agentcreator.java

```
import jade.core.AID;
import jade.core.Agent;
import jade.wrapper.AgentContainer;
import jade.core.behaviours.*;
import jade.domain.DFService;
import jade.domain.FIPAAgentManagement.*;
import jade.domain.FIPAException;
import jade.lang.acl.ACLMessage;
import jade.wrapper.AgentController;



/**
 *
 * @author tulio
 */
public class Jade_agentcreator extends Agent{
  protected AID tracker;
  private final static String SERVICE_TYPE = "CAR_CREATOR";
  Behaviour loop;
  private int i = 0;
  private int rand = 0;
  @Override
  protected void setup(){
  System.out.println(getLocalName()+" STARTED");
    DFAgentDescription dfad = new DFAgentDescription();
    dfad.setName(getAID());
    ServiceDescription svcD = new ServiceDescription();
    svcD.setType(SERVICE_TYPE);
    svcD.setName(getLocalName());
    dfad.addServices(svcD);
```

```
try {
  DFAgentDescription[] list = DFService.search(this,dfad);
  if(list.length>0)
    DFService.deregister(this);
  DFService.register(this, dfad);
  System.out.println(getLocalName()
      +" REGISTERED WITH THE DF");
}
catch (FIPAException e) {
  e.printStackTrace();
}

addBehaviour( new CyclicBehaviour(this){
  @Override
  public void action(){

  }
});
// Agent Creation Behaviour
loop = new TickerBehaviour(this, 1300){
  @Override
  protected void onTick(){
    if(tracker==null){
      DFAgentDescription dfd = new DFAgentDescription();
      ServiceDescription sv = new ServiceDescription();
      sv.setType("TRACKER");
      dfd.addServices(sv);
      try{
        DFAgentDescription[] searchResult =
            DFService.search(
                Jade_agentcreator.this, dfd);
        if(searchResult.length>0)
          tracker = searchResult[0].getName();
      }
      catch(FIPAException e){e.printStackTrace();}
    }

    Object [] args = new Object[2];
    rand = (int)(Math.random()*4+1);
    switch(rand){
      case 1:
        args[0]="W1";
        break;
      case 2:
        args[0]="E1";
        break;
      case 3:
```

```
            args[0]="N1";
            break;
          case 4:
            args[0]="S1";
            break;
        }
//        args[0] = "E1"; // direction
        args[1] = "40"; //valocity in pixel/s

        ACLMessage msg = new ACLMessage(ACLMessage.QUERY_REF);
        msg.clearAllReceiver();
        msg.addReceiver(tracker);
        msg.setContent("last-car-pos in:"+args[0]);
        send(msg);

        msg = blockingReceive();
        String s = msg.getContent();
        System.out.println(s);
        int index = s.indexOf("dist:")+5;
        if(Float.parseFloat(s.substring(index))>=40){
          AgentContainer c =
              (AgentContainer)getContainerController();
          try {
            AgentController a = c.createNewAgent(
                "Car-"+i, "Jade_car", args
            );
            a.start();
            System.out.println(getLocalName() +
                " created new agent: Car-"+i);
            i++;
          }
          catch (Exception e){
            System.out.println("Something went wrong! "
                +"Could not create Agent.");
            e.printStackTrace();
          }
        }
      }
    };
    addBehaviour(loop);
  }
  @Override
  protected void takeDown() {
    try { DFService.deregister(this); }
    catch (FIPAException e) { e.printStackTrace(); }
  }
}
```

A.2 Jade_car.java

```
import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.*;
import jade.domain.DFService;
import jade.domain.FIPAAgentManagement.*;
import jade.domain.FIPAException;
import jade.lang.acl.*;

/**
 *
 * @author tulio
 */
public class Jade_car extends Agent{
  //Lane spawn specifications
  private final int W1_xspawn = 0;
  private final int W1_yspawn = 250;
  private final int W1_xconti = 160;
  private final int W1_xconts = 190;
  private final int W1_xcontf = 260;
  private final int E1_xspawn = 480;
  private final int E1_yspawn = 230;
  private final int E1_xconti = 320;
  private final int E1_xconts = 290;
  private final int E1_xcontf = 220;
  private final int N1_xspawn = 230;
  private final int N1_yspawn = 480;
  private final int N1_yconti = 320;
  private final int N1_yconts = 290;
  private final int N1_ycontf = 220;
  private final int S1_xspawn = 250;
  private final int S1_yspawn = 0;
  private final int S1_yconti = 160;
  private final int S1_yconts = 190;
  private final int S1_ycontf = 260;

  // TICK for TickerBehaviour, this is also the time constant in
     [ms]
  private final long TICK = Math.round(1000/20);

  // Control constants
  private final float ts = 1/20;
  private final float A = (float) 0.9996;
  private final float B = (float) (4.9989/100000);
  private final float F = 4663.6f;
  private final float H = (float) -309.8;
```

```
  // Control variables
  private float r = 0;
  private float xak_1 = 0;
  private float u = 0;
  protected float v = 0;
  private float v_1 = 0;

  // Controller constants
  private final static String CROSS = "CROSS";
  private final static String WAIT = "WAIT";
  private final static String ARRIVED = "ARRIVED";
  private final static String SERVICE_TYPE = "CAR";

  // Variables
  private boolean lastMsg = false;
  private String spawn;
  private AID tracker;
  private AID controller;
  private boolean sentMsgToC = false;
  private boolean canGo = false;
  private float distF;
  protected float spawn_v = 0; // velocity at spawn
  protected int o = 0; // orientation
  protected float x = 0; //x_pos
  protected float y = 0; //y_pos
  Behaviour loop;
//  protected AID trackAgent;
  @Override
  protected void setup(){
    // Getting input arguments
    Object[] args = getArguments();
    if (args != null) {
      spawn = (String) args[0];
      spawn_v = Float.parseFloat( (String) args[1] );
      r = spawn_v;
      v = spawn_v;
    }
    // HERE LANE PLACING IS DONE. CHANGE THIS ACORDING TO
    // MODIFICATIONS IN LANE STRUCTURE.
    //
       ------------------------------------------------------------

    switch(spawn){
      case "W1":
        x=W1_xspawn;
        y=W1_yspawn;
        o=0;
```

```
      break;
    case "E1":
      x=E1_xspawn;
      y=E1_yspawn;
      o=180;
      break;
    case "N1":
      x=N1_xspawn;
      y=N1_yspawn;
      o=270;
      break;
    case "S1":
      x=S1_xspawn;
      y=S1_yspawn;
      o=90;
      break;
}
//
   ------------------------------------------------------------


// Registering with the DF
DFAgentDescription dfad = new DFAgentDescription();
dfad.setName(getAID());
ServiceDescription svcD = new ServiceDescription();
svcD.setType(SERVICE_TYPE);
svcD.setName(getLocalName());
dfad.addServices(svcD);
try {
  DFService.register(this, dfad);
  System.out.println(getLocalName()
      +" REGISTERED WITH THE DF: "
      + svcD.getType());
} catch (FIPAException e) {
  e.printStackTrace();
}

// HERE THE MESSAGE HANDLING IS DONE.
addBehaviour( new CyclicBehaviour(this){
  @Override
  public void action(){
    ACLMessage msg = receive();
    if(msg!=null &&
        msg.getPerformative()==ACLMessage.SUBSCRIBE){
      System.out.println(getLocalName()
          +" Message received from: "
          +msg.getSender().getLocalName());
```

```
            tracker = msg.getSender();
            extractController(msg.getContent());
            ACLMessage reply =
                new ACLMessage(ACLMessage.AGREE);
            reply.setContent(spawn);
            reply.clearAllReceiver();
            reply.addReceiver(msg.getSender());
            send(reply);
          }
          else if(msg!=null
              && msg.getPerformative() == ACLMessage.INFORM
              && msg.getSender().equals(controller)){
                    System.out.println(getLocalName()
            + " Message received from: "
            + controller.getLocalName()
            + " -> " + msg.getContent());
                    if(msg.getContent().equalsIgnoreCase(CROSS)){
                        canGo = true;
                    }
                    else if(msg.getContent().equalsIgnoreCase(
                       WAIT)){
                        canGo = false;
                    }
                }
                else if(msg!=null
              && msg.getPerformative() == ACLMessage.INFORM
              && msg.getSender().equals(tracker)){
                    decodeInform(msg.getContent());
                }
                block();
              }
          });

// Setting search parameters
// (UNUSED: CHANGED FROM DF SEARCH METHOD TO DF SUBSCRIPTION
   METHOD
//          STAYING HERE FOR REFERENCE PURPOSES)

/*      DFAgentDescription dfad = new DFAgentDescription();
*   ServiceDescription sd  = new ServiceDescription();
*   sd.setType("TRACKER");
*   dfad.addServices(sd);
*   // Getting search results
*   try{
*     DFAgentDescription[] sr = DFService.search(this,dfad);
*     if(sr.length>0){
*       trackAgent = sr[0].getName();
```

```
*       System.out.println(getLocalName()
*           +": Found a TRACKER Agent: "
*           +trackAgent.getName() );
*     }
*   }
*   catch(FIPAException e){
*     e.printStackTrace();
*   }
*/
    loop = new TickerBehaviour(this, TICK){
      @Override
      protected void onTick(){
        // Velocity computations here
        v_1 = v;
        v = A*v + B*(-F*v-H*(xak_1+r-v));
        xak_1 += r-v_1;

        // Position computations here
        x+=Math.cos(Math.PI*o/180)*v/TICK;
        y+=Math.sin(Math.PI*o/180)*v/TICK;
        ACLMessage msg;
        switch (spawn) {
          case "W1":
            msg = new ACLMessage(ACLMessage.INFORM);
            msg.clearAllReceiver();
            msg.addReceiver(tracker);
            if( x >= W1_xconti
                && x <= W1_xcontf
                && sentMsgToC)
              msg.addReceiver(controller);
            else{
              if(x > W1_xcontf && !lastMsg){
                msg.addReceiver(controller);
                lastMsg=true;
              }
            }
            msg.setContent(spawn+"(x:"+x+")(y:"+y
                +")(v:"+v+")(o:"+o+")");
            send(msg);
            if( x >= W1_xconti && !sentMsgToC){
              ACLMessage cMsg =
                  new ACLMessage(ACLMessage.INFORM);
              cMsg.clearAllReceiver();
              cMsg.addReceiver(controller);
              cMsg.setContent(ARRIVED+"@"+spawn);
              send(cMsg);
              sentMsgToC = true;
```

```
    }
    if( x >= W1_xconts ){
      if(!canGo)
        r = 0;
      else
        r = spawn_v;
    }
    else{
      if( distF != -1 && distF <= 40 )
        r = 0;
      else
        r = spawn_v;
    }
    if( x >= 480 )
      Jade_car.this.doDelete();
    break;
  case "E1":
    msg = new ACLMessage(ACLMessage.INFORM);
    msg.clearAllReceiver();
    msg.addReceiver(tracker);
    if( x <= E1_xconti
        && x >= E1_xcontf
        && sentMsgToC )
      msg.addReceiver(controller);
    else{
      if(x < E1_xcontf && !lastMsg){
        msg.addReceiver(controller);
        lastMsg=true;
      }
    }
    msg.setContent(spawn+"(x:"+x+")(y:"+y
        +")(v:"+v+")(o:"+o+")");
    send(msg);
    if( x <= E1_xconti && !sentMsgToC){
      ACLMessage cMsg =
          new ACLMessage(ACLMessage.INFORM);
      cMsg.clearAllReceiver();
      cMsg.addReceiver(controller);
      cMsg.setContent(ARRIVED+"@"+spawn);
      send(cMsg);
      sentMsgToC = true;
    }
    if( x <= E1_xconts ){
      if(!canGo)
        r = 0;
      else
        r = spawn_v;
```

```
    }
    else{
      if( distF != -1 && distF <= 40 )
        r = 0;
      else
        r = spawn_v;
    }
    if( x <= 0 )
      Jade_car.this.doDelete();
    break;
  case "N1":
    msg = new ACLMessage(ACLMessage.INFORM);
    msg.clearAllReceiver();
    msg.addReceiver(tracker);
    if( y <= N1_yconti
        && y >= N1_ycontf
        && sentMsgToC)
      msg.addReceiver(controller);
    else{
      if(y < N1_ycontf && !lastMsg){
        msg.addReceiver(controller);
        lastMsg=true;
      }
    }
    msg.setContent(spawn+"(x:"+x+")(y:"+y
        +")(v:"+v+")(o:"+o+")");
    send(msg);
    if( y <= N1_yconti && !sentMsgToC){
      ACLMessage cMsg =
          new ACLMessage(ACLMessage.INFORM);
      cMsg.clearAllReceiver();
      cMsg.addReceiver(controller);
      cMsg.setContent(ARRIVED+"@"+spawn);
      send(cMsg);
      sentMsgToC = true;
    }
    if( y <= N1_yconts ){
      if(!canGo)
        r = 0;
      else
        r = spawn_v;
      }
    else{
      if( distF != -1 && distF <= 40 )
        r = 0;
      else
        r = spawn_v;
```

```
    }
    if( y <= 0 )
      Jade_car.this.doDelete();
    break;
  case "S1":
    msg = new ACLMessage(ACLMessage.INFORM);
    msg.clearAllReceiver();
    msg.addReceiver(tracker);
    if( y >= S1_yconti
        && y <= S1_ycontf
        && sentMsgToC)
      msg.addReceiver(controller);
    else{
      if(y > S1_ycontf && !lastMsg){
        msg.addReceiver(controller);
        lastMsg=true;
      }
    }
    msg.setContent(spawn+"(x:"+x+")(y:"+y
        +")(v:"+v+")(o:"+o+")");
    send(msg);
    if( y >= S1_yconti && !sentMsgToC){
      ACLMessage cMsg =
          new ACLMessage(ACLMessage.INFORM);
      cMsg.clearAllReceiver();
      cMsg.addReceiver(controller);
      cMsg.setContent(ARRIVED+"@"+spawn);
      send(cMsg);
      sentMsgToC = true;
    }
    if( y >= S1_yconts ){
      if(!canGo)
        r = 0;
      else
        r = spawn_v;
    }
    else{
      if( distF != -1 && distF <= 40 )
        r = 0;
      else
        r = spawn_v;
    }
    if( y >= 480 )
      Jade_car.this.doDelete();
    break;
  default:
    break;
```

```
        }
      }
    };
    addBehaviour(loop);
  }
  private void decodeInform(String s){
    int si1 = s.indexOf("distF:")+6;
    distF = Float.parseFloat(s.substring(si1));
  }
  protected void extractController(String s){
    if(s!=null){
      int index = s.indexOf(":controller");
      index += (index!=-1)?12:0;
      index = s.indexOf(":name",index);
      index += (index!=-1)?6:0;
      controller = new AID(
          s.substring(index,s.indexOf("/JADE")+5),AID.ISGUID
      );
      index = s.indexOf("http",index);
      if(index!=-1){
        controller.addAddresses(
            s.substring(index,s.indexOf("/acc")+4)
        );
      }
    }
//      System.out.println(getLocalName() + " " + controller.
   toString());
  }
  @Override
  protected void takeDown() {
    try { DFService.deregister(this); }
    catch (FIPAException e) { e.printStackTrace(); }
  }
}
```

A.3 Jade_controller.java

```
import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.*;
import jade.domain.DFService;
import jade.domain.FIPAAgentManagement.*;
import jade.domain.FIPAException;
import jade.lang.acl.*;

import java.util.ArrayList;
import java.util.List;
```

```
/**
 *
 * @author tulio
 */
public class Jade_controller extends Agent{
  private final static String CROSS = "CROSS";
  private final static String WAIT = "WAIT";
  private final static String ARRIVED = "ARRIVED";
  private final static String SERVICE_TYPE = "CONTROLLER";
  int crossingCar = -1;
  Behaviour loop;
  class Car{
    private final int centerX = 60*4;
    private final int centerY = 60*4;
    protected String name;
    protected String lane;
    protected float x;
    protected float y;
    protected float v;
    protected int o;
    protected float dist;

    public Car(String n, String l){ this.name = n; this.lane = l;
        }
    public String getName(){return this.name;}
    public void setName(String n){this.name = n;}
    public String getLane(){return this.lane;}
    public void setLane(String l){this.lane = l;}
    public float getX(){return this.x;}
    public void setX(float x_pos){this.x=x_pos;}
    public float getY(){return this.y;}
    public void setY(float y_pos){this.y=y_pos;}
    public float getV(){return this.v;}
    public void setV(float vel){this.v=vel;}
    public int getO(){return this.o;}
    public void setO(int ori){this.o=ori;}
    public float getDist(){return this.dist;}
    public void calcDist(){
      if(this.lane.equals("W1"))
        this.dist=this.x-this.centerX;
      if(this.lane.equals("E1"))
        this.dist=this.centerX-this.x;
      if(this.lane.equals("N1"))
        this.dist=this.centerY-this.y;
      if(this.lane.equals("S1"))
        this.dist=this.y-this.centerY;
```

```
    }
    public String toString2(){
      return "["+((this.name!=null)?
          this.name.substring(0,this.name.indexOf("@"))
          :this.name)
          +"@"+this.lane+"->(x:"+this.x+",y:"+this.y
          +",v:"+Math.round(this.v*100d)/100d
          +",o:"+this.o+",dist:"+this.dist+")]";
    }
  }
  List<Car> myList = new ArrayList<>();
  @Override
  protected void setup(){
  System.out.println(getLocalName()+" STARTED");
    DFAgentDescription dfad = new DFAgentDescription();
    dfad.setName(getAID());
    ServiceDescription svcD = new ServiceDescription();
    svcD.setType(SERVICE_TYPE);
    svcD.setName(getLocalName());
    dfad.addServices(svcD);
    try {
      DFAgentDescription[] list = DFService.search(this,dfad);
      if(list.length>0)
        DFService.deregister(this);
      DFService.register(this, dfad);
      System.out.println(getLocalName()+" REGISTERED WITH THE DF"
         );
    } catch (FIPAException e) {
      e.printStackTrace();
    }

    // HERE THE MESSAGE HANDLING IS DONE.
    addBehaviour( new CyclicBehaviour(this){
      @Override
      public void action(){
        ACLMessage msg = receive();
        if(msg!=null &&
            msg.getPerformative() == ACLMessage.INFORM){
          String s = msg.getContent();
          if(s.contains(ARRIVED)){
            int index = s.indexOf("@");
            myList.add(
              new Car(msg.getSender().getName(),
                (index!=-1?s.substring(index+1):null)
              )
            );
            if(myList.size()==1){
```

```
            crossingCar=0;
            ACLMessage reply = msg.createReply();
            reply.setPerformative(ACLMessage.INFORM);
            reply.setContent(CROSS);
            send(reply);
          }
          else if(myList.size()==(crossingCar+2)
              && laneMatch(crossingCar+1)){
            crossingCar++;
            ACLMessage reply = msg.createReply();
            reply.setPerformative(ACLMessage.INFORM);
            reply.setContent(CROSS);
            send(reply);

          }
          else{
            ACLMessage reply = msg.createReply();
            reply.setPerformative(ACLMessage.INFORM);
            reply.setContent(WAIT);
            send(reply);
          }
        }
        else if(s.contains("x:")){
          decodeInform(msg);
        }
      }
      block();
    }
  });
  loop = new TickerBehaviour(this, 100){
    @Override
    protected void onTick() {
      if(crossingCar!=-1){
        for(int i=0; i<=crossingCar;i++){
          if(myList.get(i).getDist()>=20){
            myList.remove(i);
            crossingCar--;
            if(myList.size()>crossingCar+1){
              ACLMessage msg =
                  new ACLMessage(ACLMessage.INFORM);
              msg.clearAllReceiver();
              msg.addReceiver(new AID(
                  myList.get(crossingCar+1).getName(),
                  AID.ISGUID));
              msg.setContent(CROSS);
              send(msg);
              crossingCar++;
```

```
            }
          }
        }
        if(myList.size()>crossingCar+1 && crossingCar!=-1){
          if(laneMatch(crossingCar+1)){
            ACLMessage msg =
                new ACLMessage(ACLMessage.INFORM);
            msg.clearAllReceiver();
            msg.addReceiver(new AID(
                myList.get(crossingCar+1).getName(),
                AID.ISGUID));
            msg.setContent(CROSS);
            send(msg);
            crossingCar++;
          }
        }

/*          if(myList.get(crossingCar).getDist()>=20){
            myList.remove(crossingCar);
            if(myList.size()>0){
              ACLMessage msg =
                  new ACLMessage(ACLMessage.INFORM);
              msg.clearAllReceiver();
              msg.addReceiver(new AID(
                  myList.get(0).getName(),AID.ISGUID)
              );
              msg.setContent(CROSS);
              send(msg);
              myList.get(0).isCrossing();
              crossingCar=0;
            }
            else{
              crossingCar=-1;
            }
          }*/
      }

      // CODE FOR DEBUGING USES. IT PRINTS THE WHOLE LIST
//              if(!myList.isEmpty()){
//          System.out.print("[");
//                for(int i=0; i<myList.size();i++)
//                  System.out.print(myList.get(i).toString2()
   +",");
//          System.out.println("]");
//              }
    }
  };
```

```
    addBehaviour(loop);
  }
  private void decodeInform(ACLMessage msg){
    if(myList.size()>0){
      String s = msg.getContent();
      int index = 0;
      for(int i=0; i<myList.size(); i++){
        if(msg.getSender().getName().equals(
            myList.get(i).getName())){
          index = i;
          break;
        }
      }
      int si = s.indexOf("x:")+2;
      float value = Float.parseFloat(
          s.substring(si,s.indexOf(")", si)));
      myList.get(index).setX(value);
      si = s.indexOf("y:")+2;
      value = Float.parseFloat(
          s.substring(si,s.indexOf(")", si)));
      myList.get(index).setY(value);
      si = s.indexOf("v:")+2;
      value = Float.parseFloat(
          s.substring(si,s.indexOf(")", si)));
      myList.get(index).setV(value);
      si = s.indexOf("o:")+2;
      int value1 = Integer.parseInt(
          s.substring(si,s.indexOf(")", si)));
      myList.get(index).setO(value1);
      myList.get(index).calcDist();
    }
  }
  protected boolean laneMatch(int i){
    String s;
    String c;
    if(myList.size()>i+1){
       s = myList.get(i).getLane();
       c = myList.get(i-1).getLane();
    }
    else
      return false;
    if((s.equals("E1") || s.equals("W1"))
        && (c.equals("E1") || c.equals("W1")))
      return true;
    if((s.equals("N1") || s.equals("S1"))
        && (c.equals("N1") || c.equals("S1")))
      return true;
```

```
    return false;
  }
  @Override
  protected void takeDown() {
    try { DFService.deregister(this); }
    catch (FIPAException e) { e.printStackTrace(); }
  }
}
```

A.4 Jade_tracker.java

```
import jade.content.lang.sl.SLCodec;
import jade.content.onto.basic.Action;
import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.*;
import jade.domain.DFService;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.domain.FIPAAgentManagement.SearchConstraints;
import jade.domain.FIPAException;
import jade.domain.FIPANames;
import jade.domain.JADEAgentManagement.JADEManagementOntology;
import jade.domain.JADEAgentManagement.KillAgent;
import jade.lang.acl.*;
//import jade.proto.SubscriptionInitiator;

import java.util.ArrayList;
import java.util.List;

/**
 *
 * @author tulio
 */
public class Jade_tracker extends Agent{
  private final static String SERVICE_TYPE = "TRACKER";
  protected AID controller;
  transient protected Jade_window myGui;
  // Class for "CAR" agent pairings
  class Cars{
    private String name;
    private float x;
    private float y;
    private float v;
    private int o;

    public Cars(String car1, float x, float y){
```

```
      name = car1; this.x = x; this.y = y;
    }
    public Cars(String car1){name = car1;}
    public String getName(){return name;}
    public void setName(String car1){name = car1;}
    public float getX(){return x;}
    public void setX(float x){this.x = x;}
    public float getY(){return y;}
    public void setY(float y){this.y = y;}
    public float getV(){return v;}
    public void setV(float v){this.v = v;}
    public int getO(){return o;}
    public void setO(int o){this.o = o;}
    @Override
    public String toString(){return "["+this.name+"]";}
    public String toString2(){
      return "["+((this.name!=null)?
          this.name.substring(0,this.name.indexOf("@")):
          this.name)
          +"->(x:"+this.x+",y:"+this.y
          +",v:"+Math.round(this.v*100d)/100d
          +",o:"+this.o+")]";
    }
  }
  List<Cars> E1 = new ArrayList<>();
  List<Cars> W1 = new ArrayList<>();
  List<Cars> N1 = new ArrayList<>();
  List<Cars> S1 = new ArrayList<>();
  @Override
  protected void setup(){
    myGui = new Jade_window(this);
    myGui.setVisible(true);

    //Starting routine; Registering with the DF;
    System.out.println(getLocalName()+" STARTED");
    DFAgentDescription dfad = new DFAgentDescription();
    dfad.setName(getAID());
    ServiceDescription svcD = new ServiceDescription();
    svcD.setType(SERVICE_TYPE);
    svcD.setName(getLocalName());
    dfad.addServices(svcD);
    try {
      DFService.register(this, dfad);
      System.out.println(getLocalName()+" REGISTERED WITH THE DF"
        );
    }
    catch (FIPAException e) { e.printStackTrace(); }
```

```
    //Registering FIPA SL language and Ontology
    getContentManager().registerLanguage(
        new SLCodec(), FIPANames.ContentLanguage.FIPA_SL0);
    getContentManager().registerOntology(
        JADEManagementOntology.getInstance());


    //template for the DF Subscription
    DFAgentDescription dfd = new DFAgentDescription();
    ServiceDescription sd = new ServiceDescription();
    sd.setType("CAR");
    dfd.addServices(sd);
    SearchConstraints sc = new SearchConstraints();
    sc.setMaxResults(new Long(1));
/*
 ************************************************************
 * ATENTION: DF SUBSCRIPTION WORKING ONLY WITH BLOCKING METHOD *
 *    THIS IS DUE TO AGENTS BEING UNABLE TO CARRY MULTIPLE     *
 *     CyclicBehaviour NON-BLOCKING METHOD LEFT COMMENTED      *
 *              FOR REFERENCE PURPOSES                         *
 ************************************************************/
// DF Subscription

    send(DFService.createSubscriptionMessage(
        this,getDefaultDF(),dfd,sc));

    // ******** NON-BLOCKING METHOD ********
/*    addBehaviour( new SubscriptionInitiator( this,
*     DFService.createSubscriptionMessage( this,
*         getDefaultDF(), template, null)){
*       protected void handleInform(ACLMessage inform) {
*         try {
*           for (DFAgentDescription elem : newCar) {
*             System.out.println(elem.getName()
*                 + " Entering domain of TRACKER: "
*                 + getLocalName());
*             ACLMessage msg =
*                 new ACLMessage(ACLMessage.SUBSCRIBE);
*             msg.setContent(elem.getName()
*                 + " Entering domain of TRACKER: "
*                 + getLocalName() + ".\n"
*                 + "Subscribe to this TRACKING service");
*             send(msg);
*           }
*         }
*         catch (FIPAException e) {e.printStackTrace(); }
```

```
*        }
*      }
*    );
*/

    // HERE THE MESSAGE HANDLING IS DONE.
    // THIS INCLUDES THE DF SUBSCRIPTION INFORMS HANDLING
    // ***IMPORTANT***
    // all message handling must be done inside this behaviour
    // as there can only be one CyclicBehaviour per agent
    addBehaviour( new CyclicBehaviour(this){
      @Override
      public void action(){
              ACLMessage msg = receive();
              // HANDLER FOR THE DF SERVICE SUBSCRIPTION INFORMS
              //
                 ----------------------------------------------------

              if (msg != null &&
            msg.getSender().equals(getDefaultDF())){
          int isR = -2;
          try{
            isR = isRegistration(msg.getContent());
          }
          catch(Exception e){
            e.printStackTrace();
          }
          switch (isR) {
            case 1:
              if(controller==null){
                DFAgentDescription dfd =
                    new DFAgentDescription();
                ServiceDescription sv =
                    new ServiceDescription();
                sv.setType("CONTROLLER");
                dfd.addServices(sv);
                try{
                  DFAgentDescription[] searchResult =
                      DFService.search(
                      Jade_tracker.this, dfd);
                  if(searchResult.length>0)
                    controller =
                      searchResult[0].getName();
                }
                catch(FIPAException e){
                  e.printStackTrace();
                }
```

```
                }
                try {
                  DFAgentDescription[] dfds =
                      DFService.decodeNotification(
                          msg.getContent());
                  if (dfds.length > 0){
                    ACLMessage subsmsg = new ACLMessage
                        (ACLMessage.SUBSCRIBE);
/*                    System.out.println(
                      getLocalName()+ " "
                      +dfds[0].getName().getLocalName()
                      +" Entering domain of TRACKER: "
                      +getLocalName()
                    );*/
                    subsmsg.setContent(
                      dfds[0].getName().getLocalName()
                      +" Entering domain of TRACKER: "
                      +getLocalName() + ".\n Subscribe"
                      +" to this TRACKING service\n"
                      +":controller("+ controller +")"
                    );
                    subsmsg.clearAllReceiver();
                    subsmsg.addReceiver(
                        dfds[0].getName());
                    send(subsmsg);
                  }
                }
                catch (FIPAException e) {
                  e.printStackTrace();
                }
                break;
              case 0:
                try{
                  DFAgentDescription[] dfds =
                      DFService.decodeNotification(
                          msg.getContent());
                  if(dfds.length>0){
                    String iA = findAgentInList(
                        dfds[0].getName().getName());
                    if(iA==null)
                      System.out.println(getLocalName() + " "
                        +dfds[0].getName().getLocalName()
                        + " terminated for refusing subs"
                        + "cription to TRACKING service."
                      );
                    else{
                      deleteCar(
```

```
                    iA.substring(0,2),
                    Integer.parseInt(
                        iA.substring(2)
                    )
                );
                System.out.println(getLocalName() + " "
                  + dfds[0].getName().getLocalName()
                  + " Leaving domain of TRACKER: "
                  + getLocalName()
                );
              }
            }
          }
          catch (FIPAException e){
            e.printStackTrace();}
          break;
        case -1:
          System.out.println("Could not decode "
              +"DFService inform: the message"
              +" could be corrupted");
          break;
      }
          }
          // HANDLER FOR AGREE REPLIES FROM "CAR" AGENTS
          //
             ------------------------------------------------

      else if(msg!=null
          && msg.getPerformative()==ACLMessage.AGREE
          && !msg.getSender().equals(getDefaultDF())){
        // THIS HERE IS THE LANE PLACING CODE. CHANGE THIS
        // ACCORDING TO MODIFICATIONS TO LANE STRUCTURE
        //-----------------------------------------------
        switch(msg.getContent()){
          case "E1":
            E1.add(new Cars(msg.getSender().getName()));
            break;
          case "W1":
            W1.add(new Cars(msg.getSender().getName()));
            break;
          case "N1":
            N1.add(new Cars(msg.getSender().getName()));
            break;
          case "S1":
            S1.add(new Cars(msg.getSender().getName()));
            break;
        }
```

```
    //-------------------------------------------------
  }
  // HANDLER FOR CANCEL REPLIES FROM "CAR" AGENTS
  //-------------------------------------------------------
  else if(msg!=null
      && msg.getPerformative() == ACLMessage.CANCEL
      && !msg.getSender().equals(getDefaultDF())){
    System.out.println(getLocalName()+" killing agent:"
        +msg.getSender().getLocalName());
    // THIS TRIES TO KILL THE AGENT IF IT DOESN'T
    // AGREE WITH THE SUBSCRIPTION REQUEST
    try{
      KillAgent ka = new KillAgent();
      ka.setAgent(msg.getSender());
      Action a = new Action();
      a.setActor(getAMS());
      a.setAction(ka);
      ACLMessage amsr =
          new ACLMessage(ACLMessage.REQUEST);
      amsr.setSender(getAID());
      amsr.clearAllReceiver();
      amsr.addReceiver(getAMS());

      amsr.setProtocol(FIPANames.InteractionProtocol.
         FIPA_REQUEST);
      amsr.setLanguage(FIPANames.ContentLanguage.FIPA_SL0);
      amsr.setOntology(JADEManagementOntology.NAME);
      getContentManager().fillContent(amsr, a);
      send(amsr);
    }
    catch(Exception e){e.printStackTrace();}
  }
  // HANDLER FOR INFORM MESSAGES FROM "CAR" AGENTS
  //-------------------------------------------------------
  else if(msg!=null
      && msg.getPerformative() == ACLMessage.INFORM
      && !msg.getSender().equals(getDefaultDF())){
    int index = decodeInform(msg);
    ACLMessage reply = msg.createReply();
    reply.setPerformative(ACLMessage.INFORM);
    switch(msg.getContent().substring(0,2)){
      case "W1":
        if(index>0)
          reply.setContent("distF:"
              +(W1.get(index-1).getX()
              -W1.get(index).getX()));
        else
```

```
            reply.setContent("distF:"+-1);
          myGui.passData(W1.get(index).getName(),
              W1.get(index).getX(),
              W1.get(index).getY(),
              W1.get(index).getO()
          );
          break;
        case "E1":
          if(index>0)
            reply.setContent("distF:"
                +(E1.get(index).getX()
                -E1.get(index-1).getX()));
          else
            reply.setContent("distF:"+-1);
          myGui.passData(E1.get(index).getName(),
              E1.get(index).getX(),
              E1.get(index).getY(),
              E1.get(index).getO()
          );
          break;
        case "N1":
          if(index>0)
            reply.setContent("distF:"
                +(N1.get(index).getY()
                -N1.get(index-1).getY()));
          else
            reply.setContent("distF:"+-1);
          myGui.passData(N1.get(index).getName(),
              N1.get(index).getX(),
              N1.get(index).getY(),
              N1.get(index).getO()
          );
          break;
        case "S1":
          if(index>0)
            reply.setContent("distF:"
                +(S1.get(index-1).getY()
                -S1.get(index).getY()));
          else
            reply.setContent("distF:"+-1);
          myGui.passData(S1.get(index).getName(),
              S1.get(index).getX(),
              S1.get(index).getY(),
              S1.get(index).getO()
          );
          break;
      }
```

```
          send(reply);
        }
        // HANDLER FOR QUERY_REF MESSAGES FROM THE AGENT CREATOR
        else if(msg!=null
            && msg.getPerformative() == ACLMessage.QUERY_REF
            && msg.getContent().contains("last-car-pos")){
          String s = msg.getContent();
          int in = s.indexOf("in:")+3;
          ACLMessage replyAC = msg.createReply();
          switch(s.substring(in,in+2)){
            case "W1":
              replyAC.setContent("dist:"+(W1.isEmpty()
                  ?Float.valueOf(999)
                  :(W1.get(W1.size()-1).getX())));
              break;
            case "E1":
              replyAC.setContent("dist:"+(E1.isEmpty()
                  ?Float.valueOf(999)
                  :(480-E1.get(E1.size()-1).getX())));
              break;
            case "N1":
              replyAC.setContent("dist:"+(N1.isEmpty()
                  ?Float.valueOf(999)
                  :(480-N1.get(N1.size()-1).getX())));
              break;
            case "S1":
              replyAC.setContent("dist:"+(S1.isEmpty()
                  ?Float.valueOf(999)
                  :(S1.get(S1.size()-1).getX())));
              break;
          }
          replyAC.clearAllReceiver();
          replyAC.addReceiver(msg.getSender());
          send(replyAC);
        }

        //--------------------------------------------------
        // CODE FOR DEBUGING USES. IT PRINTS THE WHOLE E1 LIST
/*              if(!E1.isEmpty()){
 *                System.out.print("[");
 *                for(int i=0; i<E1.size();i++)
 *                  System.out.print(E1.get(i).toString2()+",");
 *            System.out.println("]");
 *                }
 */
        block();
      }
```

```
    });
    }
  private int decodeInform(ACLMessage msg){
    String s = msg.getContent();
    int index = -1;
    // HERE IS THE INFORM MESSAGE DECODIFICATION.
    // CHANGE THIS ACCORDING TO MODIFICATIONS TO LANE STRUCTURE.
    switch(s.substring(0,2)){
      case "W1":
        for(int i=0; i<W1.size(); i++)
          index += (msg.getSender().getName().compareTo(
              W1.get(i).getName())==0)?i+1:0;
        if(index!=-1){
          int si = s.indexOf("x:")+2;
          float value = Float.parseFloat(
              s.substring(si,s.indexOf(")", si)));
          W1.get(index).setX(value);
          si = s.indexOf("y:")+2;
          value = Float.parseFloat(
              s.substring(si,s.indexOf(")", si)));
          W1.get(index).setY(value);
          si = s.indexOf("v:")+2;
          value = Float.parseFloat(
              s.substring(si,s.indexOf(")", si)));
          W1.get(index).setV(value);
          si = s.indexOf("o:")+2;
          int value1 = Integer.parseInt(
              s.substring(si,s.indexOf(")", si)));
          W1.get(index).setO(value1);
        }
        break;
      case "E1":
        for(int i=0; i<E1.size(); i++)
          index += (msg.getSender().getName().compareTo(
              E1.get(i).getName())==0)?i+1:0;
        if(index!=-1){
          int si = s.indexOf("x:")+2;
          float value = Float.parseFloat(
              s.substring(si,s.indexOf(")", si)));
          E1.get(index).setX(value);
          si = s.indexOf("y:")+2;
          value = Float.parseFloat(
              s.substring(si,s.indexOf(")", si)));
          E1.get(index).setY(value);
          si = s.indexOf("v:")+2;
          value = Float.parseFloat(
              s.substring(si,s.indexOf(")", si)));
```

```
            E1.get(index).setV(value);
            si = s.indexOf("o:")+2;
            int value1 = Integer.parseInt(
                s.substring(si,s.indexOf(")", si)));
            E1.get(index).setO(value1);
          }
          break;
        case "N1":
          for(int i=0; i<N1.size(); i++)
            index += (msg.getSender().getName().compareTo(
                N1.get(i).getName())==0)?i+1:0;
          if(index!=-1){
            int si = s.indexOf("x:")+2;
            float value = Float.parseFloat(
                s.substring(si,s.indexOf(")", si)));
            N1.get(index).setX(value);
            si = s.indexOf("y:")+2;
            value = Float.parseFloat(
                s.substring(si,s.indexOf(")", si)));
            N1.get(index).setY(value);
            si = s.indexOf("v:")+2;
            value = Float.parseFloat(
                s.substring(si,s.indexOf(")", si)));
            N1.get(index).setV(value);
            si = s.indexOf("o:")+2;
            int value1 = Integer.parseInt(
                s.substring(si,s.indexOf(")", si)));
            N1.get(index).setO(value1);
          }
          break;
        case "S1":
          for(int i=0; i<S1.size(); i++)
            index += (msg.getSender().getName().compareTo(
                S1.get(i).getName())==0)?i+1:0;
          if(index!=-1){
            int si = s.indexOf("x:")+2;
            float value = Float.parseFloat(
                s.substring(si,s.indexOf(")", si)));
            S1.get(index).setX(value);
            si = s.indexOf("y:")+2;
            value = Float.parseFloat(
                s.substring(si,s.indexOf(")", si)));
            S1.get(index).setY(value);
            si = s.indexOf("v:")+2;
            value = Float.parseFloat(
                s.substring(si,s.indexOf(")", si)));
            S1.get(index).setV(value);
```

```
            si = s.indexOf("o:")+2;
            int value1 = Integer.parseInt(
                s.substring(si,s.indexOf(")", si)));
            S1.get(index).setO(value1);
          }
          break;
      }
      return index;
    }
    private void deleteCar(String lane, int index){
      // HERE IS THE METHOD TO REMOVE A CAR FROM A LIST.
      // CHANGE THIS ACCORDING TO MODIFICATIONS TO LANE STRUCTURE.
      //
         ------------------------------------------------------------

      switch(lane){
        case "W1":
          myGui.passDelete(W1.get(index).getName());
          W1.remove(index);
          break;
        case "E1":
          myGui.passDelete(E1.get(index).getName());
          E1.remove(index);
          break;
        case "N1":
          myGui.passDelete(N1.get(index).getName());
          N1.remove(index);
          break;
        case "S1":
          myGui.passDelete(S1.get(index).getName());
          S1.remove(index);
          break;
      //
         ------------------------------------------------------------

      }
    }
    private String findAgentInList(String a){
      // HERE IS THE CODE TO FIND AGENTS WITHOUT LANE INFO.
      // CHANGE THIS ACCORDING TO MODIFICATIONS TO LANE STRUCTURE.
      //
         ------------------------------------------------------------

      if(!E1.isEmpty()){
        for(int i = 0; i < E1.size(); i++){
          if(E1.get(i).getName().equals(a))
            return "E1"+i;
```

```
      }
    }
    if(!W1.isEmpty()){
      for(int i = 0; i < W1.size(); i++){
        if(W1.get(i).getName().equals(a))
          return "W1"+i;
      }
    }
    if(!N1.isEmpty()){
      for(int i = 0; i < N1.size(); i++){
        if(N1.get(i).getName().equals(a))
          return "N1"+i;
      }
    }
    if(!S1.isEmpty()){
      for(int i = 0; i < S1.size(); i++){
        if(S1.get(i).getName().equals(a))
          return "S1"+i;
      }
    }
    //
       ------------------------------------------------------------

    // Return null in case no agent was found
    return null;
  }
  private int isRegistration(String s){
    int i = s.indexOf("?x)");
    if(i!=-1)
      s = s.substring(i+4);
    else return i;
    i = s.indexOf(":services");
    if(i!=-1)
      return 1;
    else return 0;
  }
  @Override
  protected void takeDown() {
    if(myGui!=null){
      myGui.setVisible(false);
      myGui.dispose();
    }
    try { DFService.deregister(this); }
    catch (FIPAException e) { e.printStackTrace(); }
  }
}
```

A.5 Jade_gui.java

```
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import javax.imageio.ImageIO;
import javax.swing.JPanel;
import javax.swing.Timer;

/**
 *
 * @author tulio
 */
class Jade_gui extends JPanel implements ActionListener{
  class Car{
    protected String name;
    protected float x;
    protected float y;
    protected int o;

    public Car(String n){ this.name = n; }
    public Car(String n, float x, float y, int o){
      this.name = n; this.x = x; this.y = y; this.o = o; }
    public String getName(){return this.name;}
    public void setName(String n){this.name = n;}
    public float getX(){return this.x;}
    public void setX(float x_pos){this.x=x_pos;}
    public float getY(){return this.y;}
    public void setY(float y_pos){this.y=y_pos;}
    public int getO(){return this.o;}
    public void setO(int ori){this.o=ori;}
    public void update(String n, float x, float y, int o){
      this.name = n; this.x = x; this.y = y; this.o = o; }
  }
  List<Car> myList = new ArrayList<>();
  private final int carL = 5*4;
  private final int carW = 3*4;
  private final int laneS = 40;
  private final int DELAY = 50;
  private String PATH = "C:\\Users\\tulio\\Documents\\Faculdade\\
```

```
    TCC\\Drawing\\";
  private Timer timer;
  private int count = 0;
  private int count2 = 0;

  public Jade_gui(){ timer = new Timer(DELAY,this); timer.start()
     ; }
  public Timer getTimer(){ return timer; }

  private void doDrawing(Graphics g) {
    Graphics2D g2d = (Graphics2D) g;
    int w = getWidth();
    int h = getHeight();

    g2d.setPaint(Color.GREEN);
    g2d.fillRect(0,0,w,h);
    g2d.setPaint(Color.GRAY);
    g2d.fillRect(0,h/2-laneS/2,w,laneS);
    g2d.fillRect(w/2-laneS/2,0,laneS,h);
    g2d.setPaint(Color.WHITE);
    g2d.drawLine(0, h/2, w, h/2);
    g2d.drawLine(w/2, 0, w/2, h);
    g2d.setPaint(Color.BLACK);
    myList.forEach((elem) -> {
      if(elem.getO()==0 || elem.getO()==180)
        g2d.fillRect(Math.round(elem.getX())-carL/2,
            Math.round(elem.getY())-carW/2, carL, carW);
      else if(elem.getO()==90 || elem.getO()==270)
        g2d.fillRect(Math.round(elem.getX())-carW/2,
            Math.round(elem.getY())-carL/2, carW, carL);
    });

    if(count==39){
      BufferedImage buf =
          new BufferedImage(w,h,BufferedImage.TYPE_INT_RGB);
      Graphics2D g2 = buf.createGraphics();
      g2.setPaint(Color.GREEN);
      g2.fillRect(0,0,w,h);
      g2.setPaint(Color.GRAY);
      g2.fillRect(0,h/2-laneS/2,w,laneS);
      g2.fillRect(w/2-laneS/2,0,laneS,h);
      g2.setPaint(Color.WHITE);
      g2.drawLine(0, h/2, w, h/2);
      g2.drawLine(w/2, 0, w/2, h);
      g2.setPaint(Color.BLACK);
      myList.forEach((elem) -> {
        if(elem.getO()==0 || elem.getO()==180)
```

```
            g2.fillRect(Math.round(elem.getX())-carL/2,
              Math.round(elem.getY())-carW/2, carL, carW);
          else if(elem.getO()==90 || elem.getO()==270)
            g2.fillRect(Math.round(elem.getX())-carW/2,
              Math.round(elem.getY())-carL/2, carW, carL);
        });
        g2.dispose();
        try{
          File f = new File(PATH+"ps"+count2+".png");
          ImageIO.write(buf,"png",f);
        }
        catch(IOException e){System.out.println("Can't print!");}
        count2++;
        count=0;
      }
      count++;
    }

    public void getData(String n, float x, float y, int o){
      int i = findAgent(n);
      if(i==-1)
        myList.add(new Car(n,x,y,o));
      else
        myList.get(i).update(n,x,y,o);
//      repaint();
    }

    public void getDelete(String n){
      for(int i=0; i<myList.size(); i++){
        if(myList.get(i).getName().equals(n)){
          myList.remove(i);
          break;
        }
      }
    }

    private int findAgent(String n){
      for(int i=0; i<myList.size(); i++){
        if(myList.get(i).getName().equals(n))
          return i;
      }
      return -1;
    }

    @Override
      public void paintComponent(Graphics g) {
          super.paintComponent(g);
```

```
        doDrawing(g);
    }

  @Override
  public void actionPerformed(ActionEvent e) {
    repaint();
  }
}
```

A.6 Jade_window.java

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.BorderLayout;
import java.awt.Dimension;
import javax.swing.JFrame;
import javax.swing.Timer;

/**
 *
 * @author tulio
 */
class Jade_window extends JFrame implements ActionListener{
  private final Jade_tracker myAgent;
  private final Jade_gui myGui = new Jade_gui();

  public Jade_window(Jade_tracker a){
    myAgent = a;
    initGUI();
  }

  public final void initGUI(){
    myGui.setPreferredSize(new Dimension(480,480));
    add(myGui,BorderLayout.CENTER);

    addWindowListener( new WindowAdapter(){
      @Override
      public void windowClosing(WindowEvent e){
        Timer timer = myGui.getTimer();
        timer.stop();
      }
    });

    pack();
    setSize(482,507);
```

```
    setResizable(false);
    setLocationRelativeTo( null );
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

  }

  public void passData(String n, float x, float y, int o){
    myGui.getData(n,x,y,o);
  }
  public void passDelete(String n){
    myGui.getDelete(n);
  }

  @Override
  public void actionPerformed(ActionEvent e) {
    throw new UnsupportedOperationException("Not supported yet.")
       ;
  }

}
```

Bibliografia

Acikmese, B. e D. S. Bayard. "Probabilistic swarm guidance for collaborative autonomous agents". Em: *2014 American Control Conference*. Jun. de 2014, pp. 477–482.

Ansola, P. García, A. García Higuera, F. J. Otamendi e J. de las Morenas. "Agent-Based Distributed Control for Improving Complex Resource Scheduling: Application to Airport Ground Handling Operations". Em: *IEEE Systems Journal* 8.4 (dez. de 2014), pp. 1145–1157.

Austin, John Langshaw. *How to do things with words*. William James Lectures. Oxford University Press, 1962.

Axles & Gear. *Gear ratio guide for larger tires*. URL: `http://g2axle.com/pages/view/gear_ratio_calculator` (acesso em 05/11/2018).

Bakker, Egbert, Lars Nyborg e Hans B Pacejka. *Tyre modelling for use in vehicle dynamics studies*. Rel. técn. SAE Technical Paper, 1987.

Bellifemine, F.L., G. Caire e D. Greenwood. *Developing Multi-Agent Systems with JADE*. Wiley Series in Agent Technology. John Wiley & Sons, 2007. ISBN: 9780470058404.

Bellman, R. *An Introduction to Artificial Intelligence: Can Computers Think?* Boyd & Fraser, 1978.

Bertozzi, M., L. Bombini, A. Broggi, M. Buzzoni, E. Cardarelli, S. Cattani, P. Cerri, A. Coati, S. Debattisti, A. Falzoni, R. I. Fedriga, M. Felisa, L. Gatti, A. Giacomazzo, P. Grisleri, M. C. Laghi, L. Mazzei, P. Medici, M. Panciroli, P. P. Porta, P. Zani e P. Versari. "VIAC: An out of ordinary experiment". Em: *Intelligent Vehicles Symposium (IV)*. IEEE, 2011, pp. 175–180.

Bimbraw, Keshav. "Autonomous cars: Past, present and future a review of the developments in the last century, the present scenario and the expected future of autonomous vehicle technology". Em: *Informatics in Control, Automation and Robotics (ICINCO), 2015 12th International Conference on*. Vol. 1. IEEE, 2015, pp. 191–198.

Bond, Alan e Leslie Gasser. *Readings in distributed artificial intelligence*. Morgan Kaufmann, 2014.

Borrelli, Francesco, Paolo Falcone, Tamas Keviczky, Jahan Asgari e Davor Hrovat. "MPC-based approach to active steering for autonomous vehicle systems". Em: *International Journal of Vehicle Autonomous Systems* 3.2-4 (2005), pp. 265–291.

Broggi, A., P. Medici, E. Cardarelli, P. Cerri, A. Giacomazzo e N. Finardi. "Development of the control system for the Vislab Intercontinental Autonomous Challenge". Em: *Proceedings of the 13th IEEE Conference on Intelligent Transportation Systems*. IEEE, 2010, pp. 635–640.

Broggi, Alberto, Massimo Bertozzi e Alessandra Fascioli. "Architectural issues on vision-based automatic vehicle guidance: the experience of the ARGO project". Em: *Real-Time Imaging* 6.4 (2000), pp. 313–324.

Cardew, K. H. F. "The Automatic Steering of Vehicles: An Experimental System Fitted to a DS 19 Citroen Car". Em: (1970).

Cars Data. *Ficha técnica FIAT Uno*. URL: https://www.cars-data.com/en/fiat-uno-1.1-i.e.-start-specs/12892 (acesso em 05/11/2018).

— *Ficha técnica Opel Corsa*. URL: https://www.cars-data.com/en/opel-corsa-1.2-16v-comfort-specs/30310 (acesso em 05/11/2018).

— *Ficha técnica Palio Weekend*. URL: https://www.cars-data.com/en/fiat-palio-weekend-75-specs/12147 (acesso em 05/11/2018).

Charniak, E. e D. McDermott. *Introduction to Artificial Intelligence*. Addison-Wesley, 1985.

Chopra, Amit K., Alexander Artikis, Jamal Bentahar, Marco Colombetti, Frank Dignum, Nicoletta Fornara, Andrew J. I. Jones, Munindar P. Singh e Pinar Yolum. "Research Directions in Agent Communication". Em: (2012).

Clark, S.K. *Mechanics of Pneumatic Tires*. U.S. Department of Transportation, National Highway Traffic Safety Administration, 1981.

Colombo, R.M e A Groli. "Minimising stop and go waves to optimise traffic flow". Em: *Applied Mathematics Letters* 17.6 (2004), pp. 697–701. ISSN: 0893-9659.

Crow, Brian P, Indra Widjaja, LG Kim e Prescott T Sakai. "IEEE 802.11 wireless local area networks". Em: *IEEE Communications magazine* 35.9 (1997), pp. 116–126.

Decker, Keith S. *Task environment centered simulation*. 1996.

Dorf, R. C. e R. H. Bishop. *Modern Control Systems*. Pearson Prentice Hall, 2011.

Dresner, K. e P. Stone. "Multiagent traffic management: a reservation-based intersection control mechanism". Em: *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, 2004. AAMAS 2004.* Jul. de 2004, pp. 530–537.

Falcone, P., F. Borrelli, J. Asgari, H. E. Tseng e D. Hrovat. "Predictive Active Steering Control for Autonomous Vehicle Systems". Em: *IEEE Transactions on Control Systems Technology* 15.3 (mai. de 2007), pp. 566–580.

FIPA Abstract Architecture Specification. SC00001L. Foundation for Intelligent Physical Agents. Dez. de 2002.

FIPA Agent Management Specification. SC00023K. Foundation for Intelligent Physical Agents. Mar. de 2004.

FIPA Agent Message Transport Service Specification. SC000067F. Foundation for Intelligent Physical Agents. Dez. de 2002.

FIPA Communicative Act Library Specification. SC00037J. Foundation for Intelligent Physical Agents. Dez. de 2002.

Funke, J., P. Theodosis, R. Hindiyeh, G. Stanek, K. Kritatakirana, C. Gerdes, M. Langer D. and Hernandes, B. Muller-Bessler e B. Huhnke. "Up to the limits: Autonomous Audi TTS". Em: *Intelligent Vehicles Symposium (IV)*. IEEE, 2012, pp. 541–547.

Gaubert, N., M. Beauregard, F. Michaud e J. de Lafontaine. "Emulation of collaborative driving systems using mobile robots". Em: *Systems, Man and Cybernetics, 2003. IEEE International Conference on*. Vol. 1. Out. de 2003, 856–861 vol.1.

Halle, S., J. Laumonier e B. Chaib-Draa. "A decentralized approach to collaborative driving coordination". Em: *Proceedings. The 7th International IEEE Conference on Intelligent Transportation Systems (IEEE Cat. No.04TH8749)*. Out. de 2004, pp. 453–458.

Harrer, M. e P. Pfeffer. *Steering Handbook*. Springer International Publishing, 2016.

Haugeland, J. *Artificial Intelligence: The Very Idea*. Bradford Books. MIT Press, 1985.

Hawkins, Andrew J. "You Can Hail a Self-Driving Uber in San Francisco Starting Today". Em: *The Verge* (14 de dez. de 2016). http://www.theverge.com/2016/12/14/13921514/uber-self-driving-car-san-francisco-launch-volvo-xc90/.

Herty, Michael e Reinhard Illner. "On stop-and-go waves in dense traffic". Em: (2007).

Hespanha, J. P. *Linear Systems Theory*. Princeton University Press, 2009. ISBN: 9781400831890.

Hewitt, Carl. "Offices are open systems". Em: *ACM Transactions on Information Systems (TOIS)* 4.3 (1986), pp. 271–287.

— "The Challenge of Open Systems: Current Logic Programming Methods May Be Insufficient for Developing the Intelligent Systems of the Future". Em: *BYTE* 10.4 (abr. de 1985), pp. 223–242.

Heydinger, G. J., R. A. Bixel, W. R. Garrott, M. Pyne, J. G. Howe e D. A. Guenther. "Measured Vehicle Inertial Parameters: NHTSA's Data Through November 1998". Em: (1999).

John D'Errico. *polyfitn*. URL: https://www.mathworks.com/matlabcentral/fileexchange/34765-polyfitn (acesso em 15/10/2017).

Jonathan Decker. *Differential Gear Ratio Chart*. URL: http://erkal.jonathandedecker.com/differential-gear-ratio-chart/ (acesso em 05/11/2018).

Khalil, Hassan K. *Nonlinear Systems*. Prentice-Hall, New Jersey, 1996.

Khan, M. O. e G. Parker. "Learning live autonomous navigation: A model car with hardware arduino neurons". Em: *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. Out. de 2016, pp. 4118–4123.

Kristensen, T. e K. Smith. "Intelligent traffic simulation by a multi-agent system". Em: *2015 Third World Conference on Complex Systems (WCCS)*. Nov. de 2015, pp. 1–7.

Kurzweil, R. *The age of intelligent machines*. MIT Press, 1990.

Levine, W. S. *The Control Handbook, Second Edition: Control System Applications*. Electrical Engineering Handbook. CRC Press, 2010.

— *The Control Handbook, Second Edition: Control System Fundamentals*. Electrical Engineering Handbook. CRC Press, 2010.

Li, T. H. S., C. Y. Chen e K. C. Lim. "Combination of fuzzy logic control and back propagation neural networks for the autonomous driving control of car-like mobile robot systems". Em: *Proceedings of SICE Annual Conference 2010*. Ago. de 2010, pp. 2071–2076.

Lin, Shou-pon e N. F. Maxemchuk. "An architecture for collaborative driving systems". Em: *2012 20th IEEE International Conference on Network Protocols (ICNP)*. Out. de 2012, pp. 1–2.

Liu, G. P. "Consensus and Stability Analysis of Networked Multiagent Predictive Control Systems". Em: *IEEE Transactions on Cybernetics* 47.4 (abr. de 2017), pp. 1114–1119.

— "Predictive Control of Networked Multiagent Systems via Cloud Computing". Em: *IEEE Transactions on Cybernetics* PP.99 (2017), pp. 1–8.

Lopes, Adriano N. D., Valter J. S. Leite e Luis F. P. Silva. "On the integral action of discrete-time fuzzy TS control under saturated actuator". Em: (jul. de 2018), pp. 1–8. DOI: 10.1109/FUZZ-IEEE.2018.8491459.

Markoff, John. "Google Cars Drive Themselves, in Traffic". Em: *The New York Times* (9 de out. de 2010).

"MATISSE: A Large Scale Multi-Agent System for Simulating Traffic Safety Scenarios". Em: *IEEE 4th Biennial Workshop on DSP for In-Vehicle Systems and Safety*. Jun. de 2009.

MATLAB. *Tire-Road Interaction (Magic Formula)*. URL: https://www.mathworks.com/help/physmod/sdl/ref/tireroadinteractionmagicformula.html (acesso em 07/05/2017).

Moran, M. J. *Introduction to thermal systems engineering: thermodynamics, fluid mechanics, and heat transfer*. Introduction to Thermal Systems Engineering: Thermodynamics, Fluid Mechanics, and Heat Transfer v. 1. Wiley, 2003.

Musoles, C. F. "Towards neuroimaging real-time driving using Convolutional Neural Networks". Em: *2016 8th Computer Science and Electronic Engineering (CEEC)*. Set. de 2016, pp. 130–135.

National Conference of State Legislature. *AUTONOMOUS VEHICLES | SELF-DRIVING VEHICLES ENACTED LEGISLATION*. URL: `http://www.ncsl.org/research/transportation/autonomous-vehicles-self-driving-vehicles-enacted-legislation.aspx` (acesso em 01/06/2017).

National Highway Traffic Safaty Administration. *Federal Automated Vehicles Policy. Accelerating the Next Revolution In Roadway Safety*. Set. de 2016. URL: `https://www.transportation.gov/AV/federal-automated-vehicles-policy-september-2016`.

National Tire & Wheel. *Gearing for Taller Tires*. URL: `https://www.ntwonline.com/gearing-for-taller-tires/` (acesso em 05/11/2018).

Nilsson, N.J. *Artificial Intelligence: A New Synthesis*. Morgan Kaufmann Series in Arti. Morgan Kaufmann Publishers, 1998.

Normung, Deutsches Institut für. *Automotive Engineering; Tyres and Wheels; Concepts and Measuring Conditions*. Rel. técn. DIN 70020-5. 1 de dez. de 1986.

Ogata, K. *Engenharia de controle moderno*. Pearson Prentice Hall, 1982. ISBN: 9788587918239.

Pacejka, H.B. *Tyre and vehicle dynamics*. Butterworth-Heinemann, 2006.

Pomerleau, Dean A. "Knowledge-based training of artificial neural networks for autonomous robot driving". Em: *Robot Learning* (1993), pp. 19–43.

Poole, D.L., A.K. Mackworth e R. Goebel. *Computational Intelligence: A Logical Approach*. Oxford University Press, 1998.

Ranjbar-Sahraei, B., F. Shabaninia, A. Nemati e S. D. Stan. "A Novel Robust Decentralized Adaptive Fuzzy Control for Swarm Formation of Multiagent Systems". Em: *IEEE Transactions on Industrial Electronics* 59.8 (ago. de 2012), pp. 3124–3134.

Rich, E. e K. Knight. *Artificial Intelligence*. McGraw-Hill Higher Education, 1991.

Russell, Stuart e Peter Norvig. *Artificial Intelligence: A modern approach*. 3ª ed. Prentice-Hall, 2010.

Sabetghadam, B., F. Shabaninia, M. Vaziri e S. Vadhava. "Type-2 fuzzy multiagent traffic signal control". Em: *2012 IEEE 13th International Conference on Information Reuse Integration (IRI)*. Ago. de 2012, pp. 509–512.

Sandita, A. V. e C. I. Popirlan. "Developing A Multi-Agent System in JADE for Information Management in Educational Competence Domains". Em: *Second GLOBAL CONFERENCE on BUSINESS, ECONOMICS, MANAGEMENT and TOURISM*. 2015, pp. 478–486.

Schieschke, R. e R. Gnadler. *Modellbildung und Simulation von Reifeneigenschaften*. VDI-Berichte, 1987.

Schramm, Dieter, Manfred Hiller e Roberto Bardini. *Vehicle dynamics: Modeling and Simulation*. Springer, 2014.

Searle, J.R. *Speech Acts: An Essay in the Philosophy of Language*. Cam: Verschiedene Aufl. Cambridge University Press, 1969. ISBN: 9780521096263.

Shoham, Y. "Agent-Oriented Programming". Em: (1990).

Singh, Munindar P. "Social and Psychological Commitments in Multiagent Systems". Em: (1991).

Spanoudakis, N. e P. Moraitis. “The Gaia2JADE Process for Multi-Agent Systems Development”. Em: *Applied Artificial Intelligence Journal* 20.2-4 (2006), pp. 251–273.

Stern, Raphael E., Shumo Cui, Maria Laura Delle Monache, Rahul Bhadani, Matt Bunting, Miles Churchill, Nathaniel Hamilton, R’mani Haulcy, Hannah Pohlmann, Fangyu Wu, Benedetto Piccoli, Benjamin Seibold, Jonathan Sprinkle e Daniel B. Work. “Dissipation of stop-and-go waves via control of autonomous vehicles: Field experiments”. Em: *CoRR* abs/1705.01693 (2017).

Stewart, Jack. “After Probing Tesla’s Deadly Crash, Feds Say Yay to Self-Driving”. Em: *Wired* (20 de jan. de 2017). `https://www.wired.com/2017/01/probing-teslas-deadly-crash-feds-say-yay-self-driving/`.

Stone, Peter e Manuela Veloso. “Multiagent systems: A survey from a machine learning perspective”. Em: *Autonomous Robots* 8.3 (2000), pp. 345–383.

Tanaka, K. e H. O. Wang. *Fuzzy Control Systems Design and Analysis: A Linear Matrix Inequality Approach*. Wiley, 2001. ISBN: 9780471323242.

Taylor, Matthew. “Apple confirms it is working on self-driving cars”. Em: *The Guardian* (4 de dez. de 2016). `https://www.theguardian.com/technology/2016/dec/04/apple-confirms-it-is-working-on-self-driving-cars/`.

The Tesla Team. “All Tesla Cars Being Produced Now Have Full Self-Driving Hardware”. Em: *Tesla* (19 de out. de 2016). `https://www.tesla.com/blog/all-tesla-cars-being-produced-now-have-full-self-driving-hardware/`.

Thorpe, Charles, Martial Herbert, Takeo Kanade e Steven Shafer. “Toward autonomous driving: the CMU Navlab. Part I - perception”. Em: *IEEE expert* 6.4 (1991), pp. 31–42.

Torabi, Behnam, Mohammad Al-Zinati e Rym Z Wenkstern. “MATISSE 3.0: A Large-Scale Multi-agent Simulation System for Intelligent Transportation Systems”. Em: *Proceedings of the 16th International Conference on Practical Applications of Agents and Multi-Agent Systems*. PAAMS 18. Jun. de 2018, pp. 357–360.

Wang, Jianqun, Rui Chai e Xiaoqing Xue. “The Effects of Stop-and-go Wave on the Immediate Follower and Change in Driver Characteristics”. Em: *Procedia Engineering* (2016), pp. 289–298.

Winston, P. H. *Artificial Intelligence*. Addison-Wesley, 1992.

Xavier, P. e Y. J. Pan. “A practical PID-based scheme for the collaborative driving of automated vehicles”. Em: *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*. Dez. de 2009, pp. 966–971.

Yan, S., Y. Teng, J. S. Smith e B. Zhang. “Driver behavior recognition based on deep convolutional neural networks”. Em: *2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*. Ago. de 2016, pp. 636–641.

Al-Zinati, Mohammad e Rym Wenkstern. “MATISSE 2.0: A Large-Scale Multi-Agent Simulation System for Agent-based ITS”. Em: *Proceedings of the 2015 IEEE/WICIACM International Conference on Intelligent Agent Technology*. lAT’ 15. Singapore, Singapore, dez. de 2015, pp. 328–335.