Pontifícia Universidade Católica
do Rio de Janeiro

Ricardo Morrot Lima

Simulação Tridimensional em Tempo Real de Veículos Robóticos em Terrenos Acidentados

Dissertação de Mestrado

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Mecânica da PUC-Rio como requisito parcial para obtenção do título de Mestre em Engenharia Mecânica.

Orientador: Prof. Marco Antonio Meggiolaro

Rio de Janeiro
Setembro de 2010

Ricardo Morrot Lima

Simulação Tridimensional em Tempo Real de Veículos Robóticos em Terrenos Acidentados

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós-graduação em Engenharia Mecânica da PUC-Rio. Aprovada pela Comissão Examinadora abaixo assinada.

Prof. Marco Antonio Meggiolaro
Orientador
Departamento de Engenharia Mecânica – PUC-Rio

Prof. Hans Ingo Weber
Departamento de Engenharia Mecânica – PUC-Rio

Prof. Mauro Speranza Neto
Departamento de Engenharia Mecânica – PUC-Rio

Prof. José Eugenio Leal
Coordenador Setorial do Centro
Técnico Científico – PUC-Rio

Rio de Janeiro, 29 de setembro de 2010

Ricardo Morrot Lima

Graduou-se em Desenho Industrial – Projeto de Produto pela Pontifícia Universidade Católica do Rio de Janeiro em 1997. Pós-Graduação em Análise, Projeto e Gerência de Sistemas pela Pontifícia Universidade Católica do Rio de Janeiro em 2000.

Ficha Catalográfica

Lima, Ricardo Morrot

Simulação tridimensional em tempo real de veículos robóticos em terrenos acidentados / Ricardo Morrot Lima ; orientador: Marco Antonio Meggiolaro. – 2010.

170 f. : il. (color.) ; 30 cm

Dissertação (mestrado) – Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Engenharia Mecânica, 2010.

Inclui bibliografia

1. Engenharia mecânica – Teses. 2. Simulação. 3. Tridimensional. 4. 3D. 5. Tempo real. 6. Veículos robóticos. 7. Terrenos acidentados. 8. Controle de estabilidade. 9. Dinâmica veicular. 10. LuGre. I. Meggiolaro, Marco Antonio. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Engenharia Mecânica. III. Título.

CDD: 621

Não haveria um segundo da minha vida que eu não dedicasse a Deus,
e este trabalho só foi possível porque Ele o permitiu,
à minha querida mãe, Luiza, que me passou toda essa perseverança de vida,
aos meus irmãos, Alexandre e Mônica, de quem tenho bastante orgulho,
e à minha querida sobrinha, Katheryn, que estará eternamente no meu coração.

Agradecimentos

Tenho muito a agradecer.

Agradeço primeiramente ao professor Mauro Speranza Neto pela atenção dada, por ter ouvido a minha intenção e me indicado ao professor Marco Antonio Meggiolaro, meu orientador.

Sinto-me feliz por poder retribuir formalmente ao meu orientador Marco Antonio Meggiolaro pelo desafio a que foi submetido, pela paciência, pelas vastas horas que lhe roubei e, acima de tudo, pela confiança depositada em mim, mantida mesmo nas dificuldades.

Gostaria de agradecer especialmente ao professor Hans Ingo Weber por ter acreditado em mim e por ter me impulsionado para o universo maravilhoso da dinâmica.

Não há como deixar de agradecer a uma figura essencial, sem a qual a jornada seria literalmente um "terreno acidentado": obrigado ao amigo Pedro Blois pela ajuda técnica, científica, pela força e conexão estabelecida.

Aos amigos César Raúl Mamani Choquehuanca, Danny Hernán Zambrano, Cristian Mejia Sanchez e a todos os que colaboraram direta ou indiretamente para a concretização deste sonho.

A toda galera do Tecgraf/MVGEO representada pelo professor Luís Fernando Martha e, em especial, a ele.

Ao CNPq e à CAPES pelo apoio financeiro durante o curso de mestrado.

Meus sinceros agradecimentos.

Resumo

Lima, Ricardo Morrot; Meggiolaro, Marco Antonio. **Simulação Tridimensional em Tempo Real de Veículos Robóticos em Terrenos Acidentados.** Rio de Janeiro, 2010. 170p. Dissertação de Mestrado – Departamento de Engenharia Mecânica, Pontifícia Universidade Católica do Rio de Janeiro.

Esta dissertação aborda conceitos interdisciplinares de Engenharia Mecânica e Engenharia de Software, com foco principal no estudo de sistemas mecânicos. Atualmente, operações de monitoração por meio de veículos autônomos se tornam cada vez mais comuns, enquanto os ambientes a que esses veículos robóticos são submetidos passam a ser cada vez mais hostis, principalmente em relação aos obstáculos e características dos terrenos. O presente trabalho introduz o desenvolvimento de um simulador dinâmico em 3D em tempo real para veículos robóticos em terrenos acidentados. Um algoritmo de interseção é desenvolvido entre um terreno 3D genérico e cada roda de um veículo. Um modelo de força de contato pneu-terreno é implementado, levando em consideração as combinações das derivas longitudinal e lateral. O modelo também inclui os efeitos de corrente contínua de motores, levando-se em consideração a interação entre a parte mecânica e a elétrica, inclusive uma aproximação contínua do modelo de atrito de LuGre, considerando as limitações de potência das baterias do sistema. O simulador também inclui equações para um controle de estabilidade 2D, levando em consideração apenas a estabilização do ângulo de arfagem (*pitch*) do veículo. Este trabalho propõe, além disso, um controle de estabilidade 3D utilizando um indicador de estabilidade que pode ser calculado em tempo real, baseado em uma estimativa de distribuição de forças de contato entre roda e terreno. O simulador é validado mediante comparações com soluções analíticas do comportamento longitudinal do veículo robótico.

Palavras-chave

Simulação; Tridimensional; 3D; Tempo Real; Veículos Robóticos; Terrenos Acidentados; Controle de Estabilidade; Dinâmica Veicular; LuGre.

Abstract

Lima, Ricardo Morrot; Meggiolaro, Marco Antonio. **Three-dimensional Simulation in Real Time of Mobile Robotics on Rough Terrain.** Rio de Janeiro, 2010. 170pp. MSc. Dissertation – Departamento de Engenharia Mecânica, Pontifícia Universidade Católica do Rio de Janeiro.

This dissertation approaches interdisciplinary concepts of Mechanical Engineering and Software Engineering, with a main focus on the study of mechanical systems. Nowadays, the task of monitoring with autonomous vehicles has become more and more common, while the environment to which those robot vehicles are exposed becomes more and more hostile, mainly in relation to the obstacles and characteristics of the terrain. The present work introduces the development of a 3D real-time dynamic simulator of robot vehicles on rough terrain. An intersection algorithm is developed between a 3D generic terrain and each wheel of a vehicle. A tire-soil contact force model is implemented, taking into consideration the combined longitudinal and lateral drifts. The model includes the effects of direct current motors, taking into consideration the interaction between mechanical and electric parts, including a continuous approximation of LuGre's friction model, considering the power limitations of the system batteries. The simulator also includes an equation for a 2D stability control, taking into consideration only the stabilization of the pitch angle of the vehicle. This work also proposes a 3D stability control using an indicator of stability that can be calculated in real time, based on an estimated distribution of wheel-terrain contact forces. The simulator is validated through comparisons with analytic solutions of the longitudinal behavior of the robot vehicle.

Keywords

Simulation; Three-dimensional; 3D; Real Time; Mobile Robots; Rover; Rough Terrain; Traction Control; Vehicle Dynamics; LuGre.

Sumário

1 Introdução 20
1.1. Definição de Rover 24
1.2. Definição de Robô Móvel (*Mobile Robot*) 25
1.3. Objetivo 26
1.4. Estrutura da Dissertação 26

2 Revisão Bibliográfica 28
2.1. Controle de Tração em Terrenos Acidentados 28
2.2. Simuladores Avaliados 29
2.3. Arquitetura dos Simuladores Avaliados 38
2.4. Biblioteca Gráfica dos Simuladores Avaliados 41
2.5. Biblioteca Utilizada na Dinâmica dos Simuladores Avaliados 42

3 Simulador Desenvolvido 45
3.1. Descrição Geral 47
3.2. Modelagem do Terreno 49
3.3. Modelagem do Veículo Robótico 52
3.3.1. Parâmetros Geométricos do Veículo Robótico 52
3.3.2. Forças Externas sobre o Veículo Robótico 54
3.3.3. Forças Internas entre a Roda e o Chassi 55
3.3.4. Algoritmo de Busca do Ponto de Contato entre Roda e Terreno 56
3.4. Modelagem Dinâmica do Veículo Robótico 61
3.5. Modelo da Força de Contato Pneu-Terreno 66
3.5.1. Derivas Longitudinal e Lateral 66
3.5.2. Cálculo das Forças Longitudinal e Lateral 67
3.5.3. Combinação das Derivas 71
3.6. Atuador 71
3.7. Método de Integração 73

4 Controle 74

5 Validação do Software 81
5.1. Validação da Simulação da Arrancada 81
5.2. Validação da Simulação do Salto 85

6 Resultados 88
6.1. Controle Proporcional Simples em Terreno Senoidal 92
6.2. Controle CDTA em Terreno Senoidal 95
6.3. Controle Proporcional Simples em Terreno Plano com uma Rampa 97
6.4. Controle CDTA em Terreno Plano com uma Rampa 100

7 Conclusões 104

8 Referências bibliográficas 106

Anexo A: Organização dos Arquivos do Aplicativo VirtualBotz 3D 113

Anexo B: Diagrama de Classes (UML) do Simulador 115

Anexo C: Comandos do Joystick (Analógico) 121

Anexo D: Teclas de Funcionalidades do Simulador 123
D.1. Terminar execução do simulador 123
D.2. Alternar visão da simulação pelas câmeras interna e externas do veículo robótico 123
D.3. Visualizar legendas 126
D.4. Encerrar (parar) simulação corrente 126
D.5. Iniciar nova simulação 126
D.6. Salvar simulação corrente em vídeo 127
D.7. Visualizar trajetória do centro de massa do veículo robótico após a simulação 129
D.8. Posicionar o veículo robótico pelo centro de massa ao longo da trajetória após a simulação 131

D.9. Alterar escala de rotação e translação do mundo virtual 132

Anexo E: Scripts de Configuração do Simulador 133

Anexo F: Opções de Terrenos 145
Anexo G: Configurando Opção de Terreno no Script da Simulação 151
Anexo H: Renderização do Terreno 153

Anexo I: Configurando o Veículo Robótico no Script da Simulação 155

Anexo J: Configurando os Motores das Rodas
do Veículo Robótico no Script da Simulação 164

Anexo K: Habilitando no Script da Simulação a Geração de Dados
da Simulação Corrente do Veículo Robótico em Arquivos 167

Lista de Figuras

Figura 1 – Protótipo desenvolvido pelo Grupo de Robótica CENPES/Petrobras a ser utilizado na região amazônica 23
Figura 2 – Gasoduto Coari, Manaus (Barral, 2007, p. 20) [4] 24
Figura 3 – CLARAty navegando o Rocky 8 ROAMS Simulator [37] 30
Figura 4 – ROAMS, ou Rover Modeling and Simulation, produzido pelo Jet Propulsion Laboratory da NASA, em uma missão de superfície [25] 31
Figura 5 – Universal Mechanisms simulando um veículo robótico de seis rodas para transporte [12] 32
Figura 6 – Universal Mechanisms simulando a dinâmica de um utilitário sobre uma lombada [12] 32
Figura 7 – Gazebo simulando multirrobôs em ambiente controlado, modificado de [15] 33
Figura 8 – Um exemplo da configuração de rover no RCAST [16] 34
Figura 9 – CRAB Rover durante uma simulação [38] 35
Figura 10 – Aplicação robótica utilizando o simulador RDS [33] 36
Figura 11 – Ambiente visual do RDS para criação de aplicações robóticas [33] 36
Figura 12 – Veículo passando sobre uma rampa unilateral, SIMPACK Automotivo 37
Figura 13 – Arquitetura em camadas do CLARAty, modificado de [10] 39
Figura 14 – Exemplo, modificado de [27], ilustrando a hierarquia de objetos e o conceito de herança de classes 40
Figura 15 – Diagrama da arquitetura do aplicativo Gazebo, modificado de [60] 41
Figura 16 – Veículo para Inspeção Visual Interna de dutos (VIVI) 45
Figura 17 – Modelo de grade regular retangular utilizado pelo VirtualBotz 3D 49
Figura 18 – Extremos do domínio do terreno no VirtualBotz 3D 50
Figura 19 – Ilustração das variáveis de interpolação do terreno 51
Figura 20 – Esquema com as dimensões do chassi 52

Figura 21 – Centro de massa teórico da roda, sem qualquer deformação na suspensão 53
Figura 22 – Dimensões da roda 54
Figura 23 – Esquema de forças externas atuando sobre o veículo robótico 55
Figura 24 – Esquema de forças interna entre a roda e o chassi 55
Figura 25 – Deslocamento imposto sobre a suspensão 56
Figura 26 – Ilustração da discretização da circunferência inferior da roda 57
Figura 27 – Ilustração do esquema do cálculo de $\boldsymbol{X_{cp}}$' 58
Figura 28 – Ilustração da convergência do ponto de contato da roda com o terreno 59
Figura 29 – Esquema do ponto de contato da roda com o terreno 61
Figura 30 – Esforços agindo entre a roda e o terreno 63
Figura 31 – Curva produzida pela versão original da **"Fórmula Mágica"** em [53] 68
Figura 32 – Região admissível para as forças de tração nas rodas 1 e 2 [56] 75
Figura 33 – Combinações da região D e Γ [56] 76
Figura 34 – Pontos de interseção entre Γ e o menor diamante das linhas de mesma potência para os quatro casos básicos 77
Figura 35 – Sistema de cálculo dos eixos de estabilidade 79
Figura 36 – Gráfico 2D do momento de estabilidade do veículo robótico na tela de visualização do aplicativo VirtualBotz 3D 80
Figura 37 – Definição das grandezas utilizadas no cálculo da solução analítica para o teste de arrancada do veículo robótico 82
Figura 38 – Definição das grandezas utilizadas no cálculo da solução analítica para o teste de salto do veículo robótico 85
Figura 39 – Perfil do terreno senoidal desenhado no MATLAB 89
Figura 40 – Início da simulação: cenário da primeira simulação, com o veículo robótico VIVI, no terreno senoidal 89
Figura 41 – Veículo robótico VIVI no terreno senoidal superando o primeiro aclive 90
Figura 42 – Perfil da rampa do terreno desenhado no MATLAB 91
Figura 43 – Cenário da segunda simulação com o veículo robótico VIVI, no terreno plano com uma rampa ao final 91

Figura 44 – Veículo robótico VIVI no início da rampa
na segunda simulação, no terreno plano com uma rampa 92
Figura 45 – Velocidade do centro de massa do veículo
com o controlador proporcional simples com compensação de gravidade 93
Figura 46 – Força normal com o controlador proporcional simples
com compensação de gravidade 93
Figura 47 – Gráfico da deriva longitudinal da roda traseira
com o controlador proporcional simples com compensação de gravidade 94
Figura 48 – Potência dos motores com o controlador proporcional
simples com compensação de gravidade 95
Figura 49 – Gráfico da velocidade do centro de massa do veículo
com o controle CDTA 95
Figura 50 – Força normal com o controle CDTA 96
Figura 51 – Gráfico da deriva longitudinal da roda traseira
com o controle CDTA 96
Figura 52 – Potência dos motores com o controle CDTA 97
Figura 53 – Gráfico da velocidade do centro de massa do veículo
com o controle proporcional simples em terreno plano com uma rampa 98
Figura 54 – Força normal com o controlador proporcional simples
em terreno plano com uma rampa 98
Figura 55 – Gráfico da deriva longitudinal da roda traseira
com o controle proporcional simples em terreno plano com uma rampa 99
Figura 56 – Potência dos motores com o controlador proporcional
simples em terreno plano com uma rampa 99
Figura 57 – Velocidade do centro de massa do veículo
com o controle CDTA em terreno plano com uma rampa 101
Figura 58 – Força normal com o controle CDTA em terreno plano
com uma rampa 101
Figura 59 – Deriva longitudinal da roda traseira com o controle CDTA
em terreno plano com uma rampa 102
Figura 60 – Potência dos motores com o controle CDTA
em terreno plano com uma rampa 103
Figura 61 – Diagrama de classes (UML) do terreno 118

Figura 62 – Diagrama de classes (UML) do veículo robótico 119
Figura 63 – Diagrama de classes (UML) para matrizes 3x3 e vetores 3x1/1x3 120
Figura 64 – Comandos dos manetes do joystick 121
Figura 65 – Comandos dos botões do joystick 122
Figura 66 – Teclas de funcionalidades do simulador (teclado padrão internacional) 123
Figura 67 – Veículo robótico observado pela câmera virtual 1, permitindo a movimentação do mundo virtual de forma independente do veículo robótico 124
Figura 68 – Visão pela câmera virtual interna do veículo robótico durante várias simulações com diferentes tipos de terrenos 125
Figura 69 – Imagem obtida pela câmera virtual externa que acompanha o veículo robótico 126
Figura 70 – Janela de seleção do CoDec para geração do vídeo AVI da simulação 127
Figura 71 – Imagem de um vídeo AVI gerado com a visão do observador, externa ao veículo 127
Figura 72 – Imagem de um vídeo AVI gerado com a câmera virtual interna do veículo robótico 128
Figura 73 – Imagem de um vídeo AVI gerado com a visão da terceira câmera virtual do aplicativo VirtualBotz 3D, em uma simulação com o tipo de piso similar ao dos pilotis da PUC-Rio 128
Figura 74 – Veículo robótico com os vetores ***Z*** do centro de massa após o término da simulação 129
Figura 75 – Veículo robótico com os vetores ***X***, ***Y*** e ***Z*** do centro de massa após o término da simulação 130
Figura 76 – Veículo robótico com o centro de massa representado por uma linha contínua, após o término da simulação 130
Figura 77 – Veículo robótico retornando aos centros de massa anteriores ao do término da simulação 131
Figura 78 – Veículo robótico avançando às posições dos centros de massa até o término da simulação 132
Figura 79 – Exemplo de opção de terreno senos 145

Figura 80 – Exemplo de opção de terreno *flat* 146
Figura 81 – Ilustração do terreno de grade regular retangular 146
Figura 82 – Exemplo da ordenação da malha do terreno 148
Figura 83 – Simulação utilizando a leitura de terreno do tipo *mesh* (arquivo de malha "terrain_mesh2.msh") 148
Figura 84 – Exemplo de opção de terreno *image* (mapa de altura) com o arquivo "heightmap3.bmp" definido para altura máxima "1.0" 149
Figura 85 – Exemplo de opção de terreno *image* (mapa de altura) com o arquivo "heightmap3.bmp" definido para altura máxima "5.0", com o veículo robótico superando o aclive do terreno em diferentes instantes 149
Figura 86 – Exemplo de opção de terreno *image* (mapa de altura) com o arquivo "heightmap52.bmp" definido para altura máxima "1.0" 150
Figura 87 – As três simulações utilizam o mesmo terreno criado a partir de um mapa de altura (arquivo "heightmap63.bmp"), sendo a ilustração (a) sem nenhuma textura, com (b) textura "grass3.bmp" e (c) com "tile31.bmp" 154
Figura 88 – Modelo de roda desenhado no software Blender com apresentação em vértices e arestas 157
Figura 89 – Modelos de roda com definição dos materiais 158
Figura 90 – Exemplos de chassi bem simples, ambos com poucos detalhes e malha triangular 159
Figura 91 – Imagens da composição rodas mais chassi feita pelo simulador VirtualBotz 3D 159
Figura 92 – Visualização em Microsoft Excel do gráfico gerado pelo torque de uma das rodas do veículo robótico em uma simulação 168
Figura 93 – Visualização em Microsoft Excel do gráfico do sinal de controle PID de uma das rodas do veículo robótico durante uma simulação 169
Figura 94 – Visualização em Microsoft Excel do gráfico do vetor ***z*** do centro de massa do veículo robótico em uma simulação 169
Figura 95 – Visualização em Microsoft Excel do gráfico do vetor ***z*** do centro de massa de uma das rodas dianteiras do veículo robótico, em vermelho, e uma das rodas traseiras, em azul, durante uma simulação 170

Lista de Tabelas

Tabela 1 – Comparativo entre os simuladores avaliados 38
Tabela 2 – Biblioteca gráfica dos sistemas avaliados 42
Tabela 3 – Simuladores avaliados e suas bibliotecas de dinâmica 43
Tabela 4 – Coeficientes dos pneus para a **"Fórmula Mágica"** em [54], valores em *kN* 68
Tabela 5 – Coeficientes a_1 até a_8 para o pneu do veículo 69
Tabela 6 – Parâmetros do motor Magmotor [48] utilizado no robô VIVI 72
Tabela 7 – Parâmetros do veículo para solução analítica 81
Tabela 8 – Especificações do veículo robótico VIVI 88
Tabela 9 – Pastas do aplicativo 113
Tabela 10 – Localização e descrição de todas as bibliotecas do aplicativo 115
Tabela 11 – Exemplos de alguns dos dados dos fabricantes de motores [48] 165

Abreviaturas

API	Application Program Interface
AS2TM	AESCO Soft Soil Tire Model
ASCII	American Standard Code for Information Interchange
ASL	Autonomous Systems Lab
AVI	Audio Video Interleaved
CAD	Computer Aided Design
CDTA	Controle Dinâmico para Terrenos Acidentados
CUDA	Compute Unified Device Architecture
CENPES	Centro de Pesquisas e Desenvolvimento Leopoldo Américo Miguez de Melo
CLARAty	Coupled-Layer Architecture for Robotic Autonomy
CODEC	Contração do termo “COder-DECoder”
CPU	Central Processing Unit
DEM	Digital Elevation Model
FEM	Finite Element Modeller
GLUT	The OpenGL Utility Toolkit
GNU GPL	GNU General Public License
GPGPU	General-Purpose computing on Graphics Processing Units
GPU	Graphics Processing Unit
GPS	Global Positioning System
IMU	Inertial Measurement Unit
MER	Mars Exploration Rover
NASA	National Aeronautics and Space Administration
ODE	Open Dynamics Engine
RCAST	Rover Chassis, Analysis and Simulation Tools
RDS	Microsoft Robotics Developer Studio
RGS	Rover Graphical Simulator
ROAMS	Rover Analisys, Modeling and Simulation
RUR	Rossum's Universal Robots
TIN	Triangular Irregular network

UCP	Unidade Central de Processamento
UM	Universal Mechanisms
UML	Unified Modelling Language
Win32	Plataforma de programação em janelas (Windows) para 32 bits
WM2D	Working Model 2D
.MSH	Arquivo contendo coordenadas de uma coleção de polígonos
.MTL	Arquivo de biblioteca de materiais referenciados pelos arquivos wavefront (.OBJ)
.OBJ	Arquivo contendo coordenadas 3D de objeto em três dimensões

“O valor das coisas não está no tempo que elas duram, mas na intensidade com que acontecem. Por isso existem momentos inesquecíveis, coisas inexplicáveis e pessoas incomparáveis.”

Fernando Sabino

“A vida não dá e nem empresta, não se comove e nem se apieda. Tudo quanto ela faz é retribuir e transferir aquilo que nós lhe oferecemos.”

Albert Einstein

“Se as pessoas soubessem o quão duramente eu trabalhei para obter a minha habilidade, ela não pareceria tão maravilhosa depois de tudo.”

Michelangelo

1
Introdução

A Simulação Computacional é um recurso muito utilizado por ter um custo muito menor do que os obtidos utilizando experimentos. Quase todos os campos da ciência aplicada, como Matemática, Física, Química, Biologia e Informática, fazem esse tipo de simulação, o que torna possível o avanço científico cada vez mais rápido e resultados previsíveis a partir da modelagem computacional. O uso da simulação computacional possibilita alguns fatores importantes, como a otimização de custos e a diminuição substancial do tempo gasto em novas descobertas.

Durante muitos anos, as simulações eram feitas exclusivamente em supercomputadores de alto desempenho. Assim, os grupos de pesquisas que não dispunham de grandes recursos econômicos não conseguiam bons resultados, pois para se fazer uma modelagem o mais realista possível do sistema estudado, era necessária uma quantidade muito grande de processamento e isso aumentava o custo em muitas ordens de grandeza. Com a implementação, em 1995, dos primeiros clusters de computadores, conjuntos de computadores pessoais que trabalham em paralelo para resolver uma determinada tarefa, reduziu-se significativamente o custo do processamento em relação ao processamento em supercomputadores. Dessa forma, os grupos de pesquisas com menos recursos econômicos conseguiam obter melhores resultados.

O conceito de processamento paralelo foi amplamente difundido, e atualmente as indústrias que fabricam processadores passaram a utilizá-lo; já é possível encontrar no mercado microcomputadores com vários processadores, na ordem de algumas dezenas. Sistemas com multiprocessadores estão por toda parte, em especial na área de jogos computacionais, que necessitam de um grande poder de cálculos matriciais na formação de imagens tridimensionais. De olho no mercado de entretenimento, as indústrias de jogos assumiram uma posição pioneira na produção de processadores para placas de vídeos com mais de 100

núcleos (processadores) em um único chip, dentre elas a NVidia (http://www.nvidia.com.br/page/home.html) com a placa GeForce GTX 580 com 512 núcleos e 16 engines PolyMorph, que buscam reduzir ao mínimo o efeito de pixelização de imagens, vídeos e jogos.

O surgimento de novas placas de vídeo com alta capacidade de processamento de imagem veio abrir novas possibilidades na área de pesquisa, sendo absorvidas rapidamente por toda a comunidade científica para fazer parte de suas ferramentas de Simulações Computacionais. Atualmente, no Brasil, alguns grupos têm se mostrado interessados em adaptar seus programas de Simulação para usarem CUDA e GPGPU, recursos computacionais existentes nas placas de vídeo para disponibilizar seus processadores paralelos na programação de efeitos visuais e até mesmo como suporte à álgebra computacional. O software MATLAB MathWorks também incluiu suporte ao CUDA a partir da versão de 2007 [32].

A Robótica também tem se beneficiado, nos últimos anos, de todo esse avanço computacional. Milhares de microprocessadores, microcontroladores e circuitos integrados dos mais diversos tipos têm gerado uma grande explosão tecnológica na criação de robôs com dispositivos e ferramentas acessíveis a qualquer usuário. Entretanto, a criação de robôs móveis de ponta, em especial os rovers, ainda tem um alto custo de desenvolvimento e produção. Com a Simulação Computacional abre-se uma ampla e nova vertente: a simulação virtual desses robôs, auxiliando em projetos e no desenvolvimento de algoritmos de controle.

Rovers são veículos robóticos de exploração, que podem ser utilizados neste ou em outros planetas. Os robôs para exploração de outros planetas têm obtido notoriedade em recentes estudos de simulações virtuais na área de Robótica [1 e 2]. Mas igualmente importantes são os rovers projetados para a exploração do nosso planeta. Há regiões que apresentam terrenos tão intrigantes e desafiadores quanto os encontrados em outros planetas. Ao observar com o cuidado necessário o nosso planeta, e mais precisamente o Brasil, percebe-se um dos mais ricos biomas terrestres com cerrados, florestas, caatingas, campos, pantanal, dentre outros. Tudo isso é um grande desafio para empresas que exploram os recursos existentes nessas regiões e que precisam de monitoração constante.

O acesso a essas áreas geralmente envolve obstáculos que aumentam muito o custo operacional. Por isso, a simulação virtual de veículos robóticos, em vários tipos de configurações de terrenos, deixa de ser uma simples alternativa para ser uma necessidade, diminuindo os custos de desenvolvimento e permitindo planejar melhor os experimentos.

O estudo do comportamento de um rover em um determinado tipo de terreno não traz apenas economia em termos financeiros e, consequentemente, em tempo de execução, mas também vem facilitar o planejamento e o desenvolvimento da tecnologia robótica, aumentando o grau de eficiência e segurança dos futuros veículos.

Uma das regiões mais estudadas é a Amazônica [4], por apresentar uma estrutura rica de solos, florestas e milhares de veios de rios. Vencer os obstáculos impostos pelos diversos tipos de terrenos é um desafio muito grande.

O estudo de incursões de um rover em solos amazônicos ou em qualquer tipo de terreno irregular é sem dúvida um rico laboratório de pesquisa para criação de uma grande gama de veículos robóticos monitores a serem aplicados em parques industriais, campos abertos e até mesmo em residências.

As plataformas computacionais existentes ainda não são capazes de fazer a simulação de veículos robóticos em terrenos acidentados (*rough terrain*), e que permitam prever o comportamento dinâmico com a possibilidade de implementação de técnicas avançadas para controle de estabilidade e tração. Elas são, sim, capazes de modelar terrenos com ondulações, cujos gradientes/inclinações variam pouco ao longo da projeção do veículo sobre o terreno, mas terrenos realmente acidentados podem apresentar gradientes/inclinações completamente diferentes em cada uma das rodas do veículo. Tais estudos de comportamentos dinâmicos em terrenos acidentados têm a vantagem de possibilitar reduções de custos operacionais bem significativos, já que toda a operação será virtual.

Com o projeto Cognitus [4], a Petrobras tem desenvolvido uma série de robôs voltados para monitoração e chamados de AmazonBots. Esses robôs são projetados com o objetivo principal de monitorar dutos em uma região de difícil acesso, onde a prevenção de riscos ambientais é encarada com alta prioridade.

Vencer os mais de 1.500 km de monitoração de dutos, por onde são escoados todo o petróleo e gás explorados na bacia Amazônica, tem sido um

grande obstáculo para a Petrobras [4], já que toda essa extensão passa por uma mescla de vários tipos terrenos. A detecção de anomalias ambientais não é a única forma de minimizar um problema, já que a partir de uma anomalia a ação deve ser paliativa, isto é, conter o problema. Já a prevenção, aliada a robôs incursores, pode trazer um alto grau de controle, onde cada incursor será capaz de fazer coletas de dados do meio ambiente, com diagnósticos em tempo real e disponibilizados via satélite. Para isso, há de se trabalhar sinergicamente na construção de rovers capazes de gerar resultados satisfatórios.

Um exemplo de aplicabilidade de uma plataforma de simulação virtual é o robô ambiental híbrido (Figura 1) desenvolvido pelo CENPES/Petrobras, dentro do projeto Cognitus. Ele será usado para monitorar uma região da Amazônia por onde passa o gasoduto Coari-Manaus, com 420 km de extensão (Figura 2). Como seu uso será para monitoramento ambiental, exige um avançado sistema de controle de tração e estabilidade. Foi todo animado virtualmente, mas sem nenhuma modelagem dinâmica.

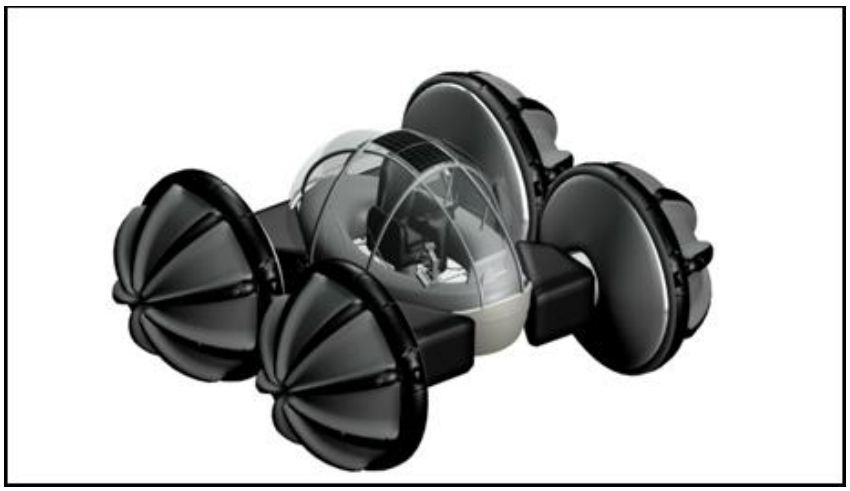


Figura 1 – Protótipo desenvolvido pelo Grupo de Robótica CENPES/Petrobras a ser utilizado na região amazônica

Figura 2 – Gasoduto Coari, Manaus (Barral, 2007, p. 20) [4]

1.1.

Definição de Rover

O conceito de rover muitas vezes traz a ideia de atividades em outros planetas. No entanto, eles não estão restritos a essas atividades. Esta é a definição sugerida por alguns centros de pesquisa na área:

> *"Um manipulador reprogramável e multifuncional projetado para transportar materiais, partes, ferramentas, ou outros dispositivos especializados através de vários movimentos programados a fim de realizar uma grande variedade de tarefas."*
>
> **Robot Institute of America (1979)**

Algumas outras definições de rover, porém, sugerem seu uso apenas em outros planetas:

> *"Veículo para exploração da superfície de um corpo extraterrestre (como a Lua ou Marte)."*
>
> **Phoenix Mars Mission**

> *"Um rover é um veículo planetário de exploração espacial projetado para se mover sobre uma superfície de um planeta ou outro corpo astronômico."*
>
> **Wikipédia (http://en.wikipedia.org/wiki/Rover_(space_exploration))**

Entre as várias definições de rover, fica bem claro que ele é um tipo de robô bem específico e muito vinculado ao conceito de exploração de terreno. E é exatamente esse tipo de tarefa que um rover executa ao chegar ao seu destino neste ou em outro planeta: explorar o ambiente.

1.2. Definição de Robô Móvel (*Mobile Robot*)

Segundo o Robotics Research Group, da Universidade do Texas, em Austin [23], a definição mais condizente com o pensamento de ideias que envolvem a definição de robô seria a do Webster:

> *"Um dispositivo automático que executa tarefas normalmente atribuídas a humanos, ou uma máquina na forma de um humano."*

Semelhante à definição de um robô industrial, da Organização de Padrões Internacional, um robô móvel pode ser definido assim:

> *"Um robô móvel é um sistema autônomo capaz de atravessar um terreno com obstáculos naturais ou artificiais. Seu chassi é equipado com rodas ou pernas e possivelmente um manipulador fixado ao seu chassi para segurar objetos, ferramentas ou dispositivos especiais. Várias operações de pré-planejamento são executadas baseadas numa estratégia de navegação pré-programada que leva em conta o estado atual do ambiente" [24].*

Com esta definição, qualquer máquina inteligente e autônoma que se move em um ambiente é chamado de *mobile robot*, ou robô móvel. Logo, pode-se chamar também de veículo robótico ou veículo autônomo.

Os robôs móveis podem ter várias aplicações, como operação em áreas de perigo, exploração planetária (rovers), inspeção de dutos e vários tipos de serviços domésticos de limpeza e segurança. É perceptível que a atuação desses tipos de robôs vai das indústrias às residências, levando a crer que farão parte da nossa vida diária em um futuro bem próximo.

1.3. Objetivo

O objetivo principal desta dissertação é desenvolver um simulador dinâmico em 3D para veículos robóticos que permita o deslocamento por vários tipos de ambientes com terrenos acidentados e o uso de diversas técnicas de controle.

1.4. Estrutura da Dissertação

O capítulo 2 apresenta referências a trabalhos de pesquisas aplicados ao controle de tração de rovers em terrenos acidentados, ou irregulares, como Barral [4], Auderi [5] e Iagnemma [1 e 2]. Mostra uma avaliação de 11 simuladores de veículos robóticos disponíveis no mercado, focando características relacionadas à renderização 3D, licença de uso, facilidades de uso de diversos tipos de terrenos, possibilidade de usar diferentes tipos de veículos robóticos e de fazer uma simulação em tempo real, portabilidade, arquitetura de software, biblioteca gráfica e biblioteca de dinâmica.

O capítulo 3 apresenta, no início, a relevância de um simulador desse porte no estudo de comportamentos de robôs reais. Também se descrevem os recursos implementados e toda a sua organização, em detalhes. Neste capítulo, é possível entender a modelagem utilizada para o terreno, modelagem do veículo robótico separado em parâmetros geométricos, forças externas sobre o veículo, forças internas entre a roda e o chassi e o algoritmo de busca do ponto de contato entre roda e terreno. A modelagem dinâmica do veículo robótico é apresentada, bem como o modelo da força de contato pneu-terreno, as equações usadas para calcular as derivas longitudinal e lateral e o cálculo das forças longitudinal e lateral. Por fim, descreve-se a implementação do atuador para cada roda do veículo robótico e método de integração.

O capítulo 4 apresenta o controle implementado com ênfase no controle dinâmico para terrenos acidentados (CDTA) de estabilidade 2D proposto por Silva [56] para terrenos acidentados. Além disso, explica-se a importância do uso de um joystick no simulador.

O capítulo 5 apresenta a validação do simulador, comparando-o com modelos analíticos de casos simples conhecidos.

O capítulo 6 apresenta os resultados de simulações comparando o controle proposto por Silva [56] com o controle PID com compensador de gravidade.

O capítulo 7 apresenta as conclusões.

2
Revisão Bibliográfica

2.1.
Controle de Tração em Terrenos Acidentados

Trabalhos de pesquisas aplicados a controle de tração de robôs móveis em terrenos acidentados, ou irregulares, foram desenvolvidos por Barral [4] e Auderi [5]. Em [4], o autor apresenta um método de controle de tração para um robô móvel que permite, ao mesmo rover, o deslocamento por vários tipos de terrenos acidentados, de forma a minimizar o consumo de potência em terrenos planos e regulares, e maximizar a estabilidade em terrenos irregulares. Em [5], são desenvolvidas técnicas de controle de capotagem e deslizamento de um robô móvel, visando garantir a locomoção do robô em terrenos irregulares e inclinados. No estudo, diante das condições encontradas, a estabilidade do veículo passou a ser um fator fundamental para o desenvolvimento de um controle eficaz que garantirá segurança nas operações, evitando capotagens e ajudando nas tomadas de decisões, e até recusando a trajetória comandada, se esta lhe oferecer um obstáculo não superável.

Iagnemma [2] apresenta um método que propõe minimizar a razão entre a força de tração e a força normal, medidas com sensores colocados em cada roda, para controlar melhor o deslizamento. Este método tem a vantagem de não precisar conhecer as características do terreno e a velocidade do robô. Simulações realizadas demonstraram bons resultados, entretanto não foram apresentados experimentos.

O método de controle de tração em terrenos acidentados, apresentado por Iagnemma em [1], utiliza como dados de entrada medições das propriedades do terreno e de sua geometria, com o intuito de otimizar o torque nas rodas e obter máxima tração ou mínimo consumo de potência, variando de acordo com o grau

de dificuldade de cada terreno. Os resultados apresentados tanto na simulação como nas situações reais demonstraram a efetividade do método proposto.

2.2.
Simuladores Avaliados

Entre os simuladores de veículos robóticos disponíveis no mercado, a grande parte está direcionada para exploração de terrenos em outros planetas.

- **CLARAty**

CLARAty [10] é um dos softwares de simulação de rovers mais conhecidos na comunidade acadêmica. Na verdade ele deve ser considerado não simplesmente um simulador, mas uma estrutura capaz de reunir várias bibliotecas de veículos e visualizadores 3D para rovers. Chamado oficialmente de framework e não de simulador, é um padrão para Coupled-Layer Architecture for Robotic Autonomy [27, 28, 29 e 30], sendo um esforço colaborativo entre várias instituições, inclusive a NASA Jet Propulsion Laboratory (JPL), NASA Ames Research Center, Carnegie Mellon e a Universidade de Minnesota. O software completo inclui um grande número de módulos para a programação de robôs, mas até o momento a NASA apenas liberou um conjunto de funcionalidades ao público. O CLARAty não é um software de código aberto, pois não está enquadrado às regras da GNU General Public License [13] – ou GNU GPL, ou simplesmente GPL. Essa designação de licença foi idealizada por Richard Stallman em 1984, com o objetivo de criar um sistema operacional totalmente livre. A parte pública das bibliotecas do CLARAty pode ser baixada em [39], inclusive sua licença de uso em [40] (JPL Open Source License). Contudo, a maior parte das bibliotecas ainda continua de uso restrito. Segundo informações no site da NASA JPL, qualquer tipo de uso do software CLARAty, fora das atividades acadêmicas ou amadoras, deve ser encaminhado ao JPL a fim de que possa ser analisado.

Na página do CLARAty em [39] há informações relatando que a parte pública não foi testada completamente por causa de constantes mudanças em alguns de seus módulos; entretanto, a parte do repositório que contém o material

privado vem sendo testada há anos. Isso significa que a versão liberada é classificada como *lite*, ou seja, uma versão gratuita e com várias restrições de uso, defasada em relação à sua versão completa.

São 44 os módulos liberados, equivalentes a 10% de todos os módulos compreendidos no software CLARAty, que abrangem desde recursos de matemática, rotação de matrizes com ângulos de Euler, quatérnios e transformações de coordenadas, inclusive transformações de quatérnios, até a infraestrutura responsável pela interface entre transformações e mecanismos com as partes móveis do rover. Englobam também modelos de rodas, pernas, veículos híbridos, câmeras, motores, suporte para I/O (input/output) analógico e digital, dentre vários outros recursos.

A Figura 3 mostra um dos vários exemplos recuperados do site da CLARAty, que ilustra a simulação do rover Rocky 8, criado com o simulador ROAMS [25] e utilizando o framework CLARAty com o módulo Morphing Navigator para a simulação de navegação. Essa capacidade de utilização de várias bibliotecas para criação de rovers e os mais diversos tipos de controles faz do CLARAty um software bastante modular, ideal para reutilização.

Figura 3 – CLARAty navegando o Rocky 8 ROAMS Simulator [37]

- **Rover Graphical Simulator**

O Rover Graphical Simulator (RGS) [11 e 14] é outro software comercial produzido pela NASA e opera com rovers em terrenos planos com obstáculos. Seu objetivo é desviar deles, em vez de superá-los. Entretanto, esse software é uma simulação gráfica em 2D. Inclui bibliotecas de lógica fuzzy para o desvio de obstáculos.

- **ROAMS**

O ROAMS [25 e 36], ou Rover Analisys, Modeling and Simulation, é mais um software produzido pelo Jet Propulsion Laboratory da NASA. Ele tem o objetivo de modelar e simular futuras missões de pouso e exploração em Marte (Figura 4). Para isso, conta com uma ferramenta de simulação para análise, projeto, desenvolvimento e teste de operação de rovers em superfícies planetárias. Não é um software comercial e está sendo usado atualmente pelo NASA's Mars Program como um teste virtual para vários tipos rovers, terrenos, ambientes e componentes, tais como sensores, câmeras, dentre outros. Um dos recursos mais fortes desse programa é o mapeamento de terrenos.

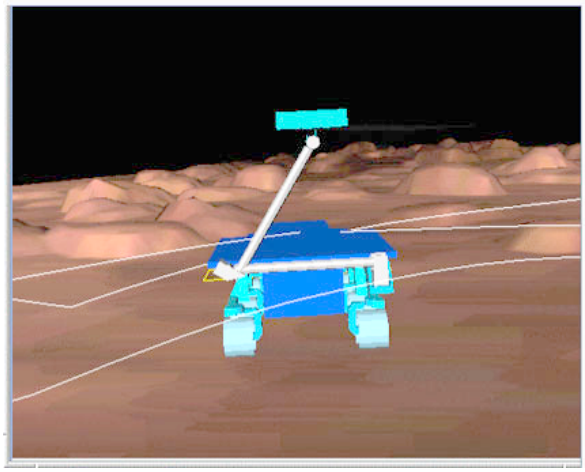


Figura 4 – ROAMS, ou Rover Modeling and Simulation, produzido pelo Jet Propulsion Laboratory da NASA, em uma missão de superfície [25]

- **Universal Mechanisms**

O Universal Mechanisms (UM) [12] é um software comercial de simulação de veículos desenvolvido pelo Laboratório de Mecânica Computacional da Bryansk State Technical University, na Rússia, capaz de lidar com terrenos acidentados. No entanto, ele gera esses terrenos por meio de equações, o que o torna restrito e lento. Embora faça a simulação de veículos robóticos (Figura 5), o foco deste software está na simulação de trens e caminhões de cargas em rodovias, com terrenos planos e pequenos obstáculos como quebra-molas e lombadas (Figura 6).

Figura 5 – Universal Mechanisms simulando um veículo robótico de seis rodas para transporte [12]

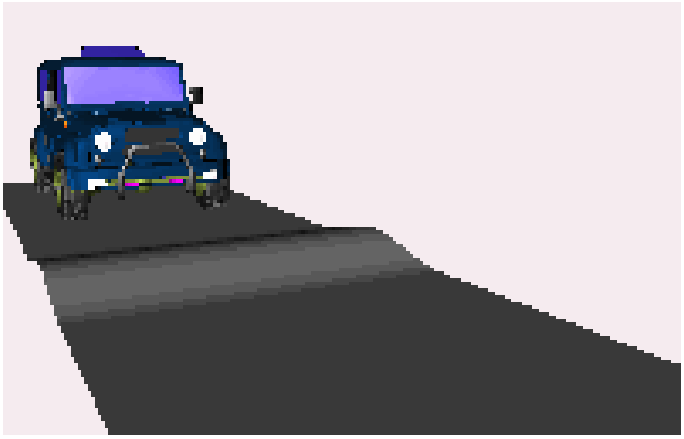


Figura 6 – Universal Mechanisms simulando a dinâmica de um utilitário sobre uma lombada [12]

- **Gazebo**

Gazebo [15 e 31] é um software de simulação em três dimensões, e o seu forte é a simulação de populações de robôs. Para isso, ele conta com recursos como sensores de obstáculos que dão um retorno realístico da interação entre os objetos e inclui uma simulação precisa da física de corpos rígidos (Figura 7).

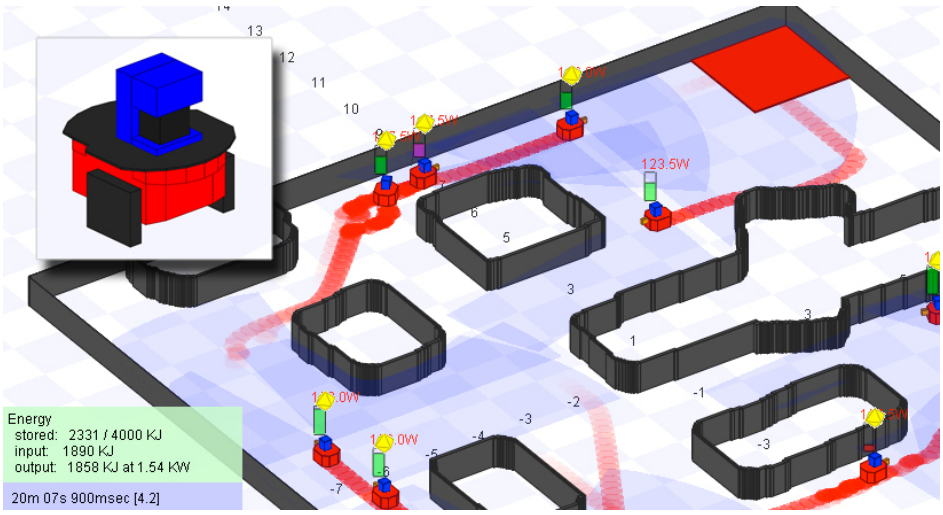


Figura 7 – Gazebo simulando multirrobôs em ambiente controlado, modificado de [15]

Entre os simuladores avaliados, o Gazebo tem um ferramental bem superior aos demais, incluindo sensores virtuais de varrimento de imagem a laser, sonar, GPS, IMU (Inertial Measurement Unit), tipos diferentes de câmeras, adição de objetos com interação entre si e com os robôs, e ainda é um programa freeware. Sua licença está dentro das regras GNU General Public License.

- **RCAST**

O RCAST [16], ou Rover Chassis, Analysis and Simulation Tools, estuda o comportamento e otimização da suspensão de um rover planetário específico. Simula (Figura 8) a dinâmica de multicorpos e a correspondente interação de roda-terreno (Bauer et al., 2005a). Desenvolvido no ambiente Simulink do MATLAB MathWorks, ele utiliza o software comercial AESCO Soft Soil Tire

Model (AS2TM) na modelagem computacional da interação roda-terreno para predizer a sua locomoção, comparado com uma análise quase-estática.

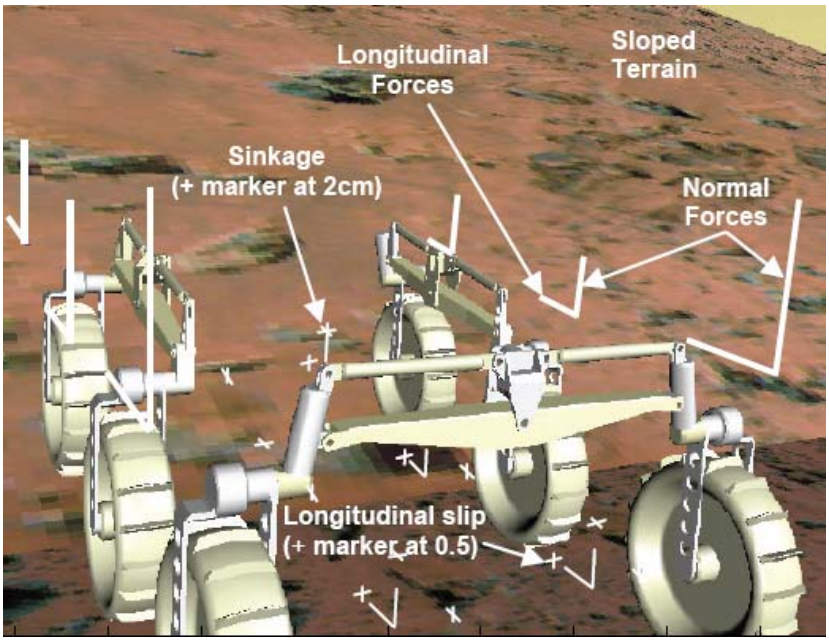


Figura 8 – Um exemplo da configuração de rover no RCAST [16]

- **Working Model 2D**

O Working Model 2D [17], embora seja um simulador dinâmico 2D, é capaz de detectar a colisão de corpos, além de ter uma ferramenta satisfatória [18] na criação e simulação de robôs móveis em terrenos acidentados. O WM2D é um simulador comercial.

- **CRAB Rover**

O CRAB Rover [19] é um robô suíço articulado de seis rodas do Autonomous Systems Lab (ASL). Projetado para exploração em terrenos acidentados, sua tarefa é identificar obstáculos e, se possível, evitá-los (Figura 9). Ele compete diretamente com o MER, Mars Exploration Rover, da NASA [20]. Simulações computacionais foram feitas para validar o modelo CRAB e desenvolver algoritmos de controle eficientes. Toda a simulação foi desenvolvida

em C e C++, e permitiu a criação dentro do simulador dos algoritmos de controle reais, utilizados no rover, sem maiores modificações. A biblioteca ODE [21] também foi utilizada no projeto CRAB Rover para criar o rover e os modelos de terrenos baseados na dinâmica de corpos rígidos.

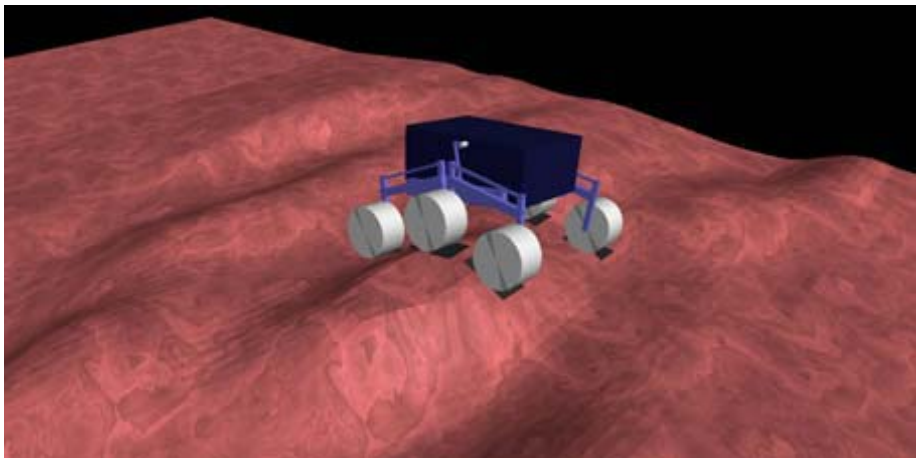


Figura 9 – CRAB Rover durante uma simulação [38]

- **Adams**

O Adams Multbody Dynamics, da MSC Software [22], é um programa bem reconhecido e considerado padrão em muitos setores da indústria, utilizado basicamente como ferramenta de simulação em sistemas de múltiplos corpos. Ele é baseado na fórmula de Newton-Euler e usa Euler-Lagrange para equações do movimento. O software resolve um sistema de equações diferenciais e usa os mais recentes métodos numéricos. Entretanto, para gerar resultados, ele consome tempo e capacidade de processamento demais. Tem a facilidade de modelar rovers de qualquer topologia, mas exige grandes recursos para desenhar um novo modelo e preparar a simulação.

- **Microsoft Robotics Developer Studio 2008**

Microsoft Robotics Developer Studio 2008 (RDS) [33] é um software para o ambiente Microsoft Windows desenvolvido para criação de aplicações robóticas

(Figura 10). Na simulação de aplicações robóticas em ambientes 3D virtuais, se faz necessário o uso da biblioteca PhysX AGEIA Technologies Inc. [34]. O RDS tem também recursos de interação com o robô, monitoração em tempo real dos sensores virtuais robóticos e respostas aos motores e atuadores. É uma plataforma bastante amigável, que permite aos *end-users* (ou usuários finais, não programadores ou especialistas) criar aplicações robóticas por meio de um ambiente de programação visual (Figura 11).

Por ser produto da Microsoft, é o único simulador, dentre os avaliados, a utilizar a biblioteca gráfica DirectX [35].

Figura 10 – Aplicação robótica utilizando o simulador RDS [33]

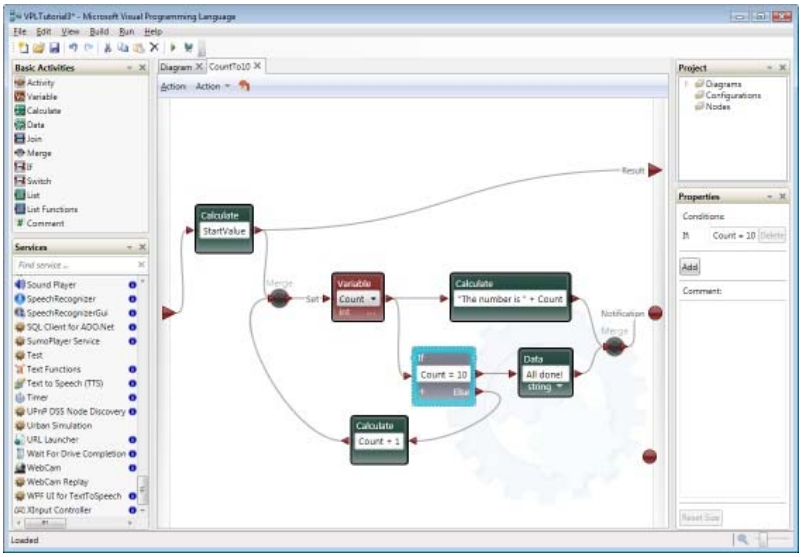


Figura 11 – Ambiente visual do RDS para criação de aplicações robóticas [33]

- **SIMPACK – Multi-Body Simulation Software**

O SIMPACK [47] é uma ferramenta de simulação não-linear de propósito geral para múltiplos corpos em 3D. Ideal para simular sistemas mecânicos, análise de vibrações, cálculo de forças e aceleração, descreve e prediz o movimento de sistemas complexos de múltiplos corpos. Os modelos utilizados podem ser construídos pela ferramenta que o acompanha, ou importados de modelos feitos em CAD e FEM. Utiliza o OpenGL como interface gráfica e foi desenvolvido em Fortran90 (última versão). Não é um software freeware e tem bibliotecas para licenciamento, além de permitir trabalhar com o MATLAB/Simulink.

O SIMPACK está dividido em quatro pacotes: Automotivo, Motores, Veículos sobre Trilhos e Sistemas de Turbinas Eólicas. O pacote mais relacionado a esta dissertação é o Automotivo. Na Figura 12 tem-se um exemplo retirado do vídeo "jump_start_spider_03.avi" do website da SIMPACK, que ilustra o comportamento dinâmico de um veículo passando por uma rampa unilateral.

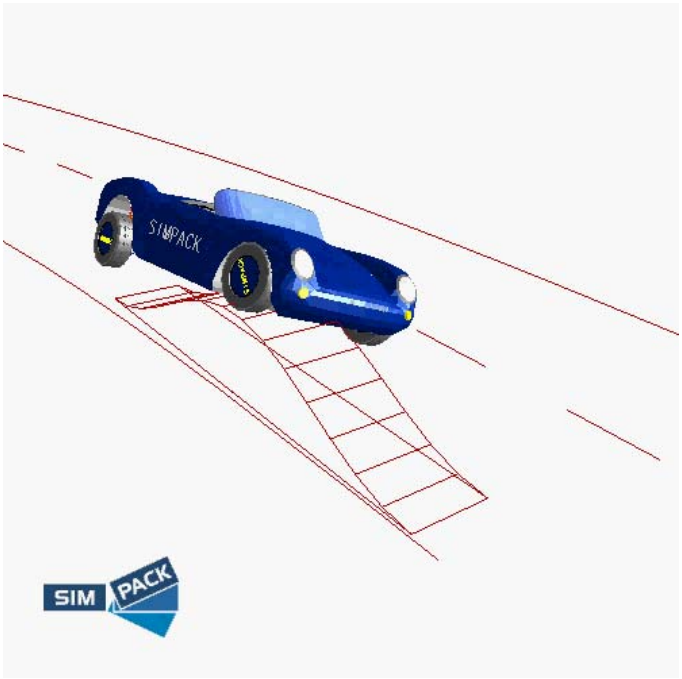


Figura 12 – Veículo passando sobre uma rampa unilateral, SIMPACK Automotivo

A Tabela 1 resume os recursos de cada simulador avaliado, para efeitos comparativos.

Tabela 1 – Comparativo entre os simuladores avaliados

	3D	GNU GPL (freeware)	Terreno Acidentado	Diferentes Rovers	Real Time	Sistema Operacional
ADAMS	✓	✗	✓	✓	não informado	Windows e Linux
CLARAty	✓	✗	✓	✓	✓	Linux
CRAB Rover	✓	✗	✓	CRAB I, II e III	não informado	não informado
Gazebo	✓	✓	✓	✓	✓	Linux
RCAST	✓	✗	✓	dedicado (um único)	não informado	não informado
ROAMS	✓	✗	✓	✓	✓	Linux
Rover Graphical Simulator	✗	✗	✓	não informado	✓	Linux
Universal Mechanisms	✓	✗	✓	✓	não informado	Windows
Working Model 2D	✗	✗	✗	✓	✓	Windows
Microsoft Robotics Developer Studio	✓	✗	✗	Lightweight	✓	Windows
SIMPACK	✓	✗	✓	Veículos	✓	Windows, HP UX, Silicon Graphics

2.3.

Arquitetura dos Simuladores Avaliados

Neste item, tem-se uma breve visão de como os softwares avaliados são implementados, e quais os recursos gráficos utilizados. Infelizmente, poucos são os softwares que expõem a sua arquitetura, até mesmo porque aqueles que são comercializados utilizam a sua discrição como uma vantagem competitiva. Entretanto, a arquitetura mais discutida é a do CLARAty, associada a vários artigos já publicados.

- **CLARAty**

O termo CLARAty é um padrão para Coupled-Layer Architecture for Robotic Autonomy [27-30]. Ele propõe uma arquitetura de duas camadas (Figura 13): uma de decisão e uma funcional. Na camada de decisão estão envolvidas as

tarefas de planejamento, persistência de dados e execução. Na camada funcional estão todas as interfaces do sistema/hardware e suas capacidades. Eis algumas razões para a arquitetura ter sido desenvolvida com orientação ao objeto:

- por ser modular, pode-se implementar facilmente as características de hardware de um sistema robótico;
- em todos os níveis de modularização, funcionalidade e persistência de dados do sistema de informação, os dados podem ser codificados e compartimentalizados em um único local lógico;
- a própria estruturação do software permite o uso das propriedades de herança para administrar a complexidade do desenvolvimento do software; e
- essa estrutura pode ser graficamente desenhada e documentada utilizando o padrão de linguagem gráfica UML (*Unified Modeling Language*).

Figura 13 – Arquitetura em camadas do CLARAty, modificado de [10]

O diagrama da Figura 14 mostra uma breve ilustração da hierarquia de classes encontrada na camada funcional do CLARAty. É uma abstração para demonstrar a estrutura de classes na camada Funcional. Como subclasse tem o objeto rover, que agrega o braço (manipulador) robótico e a estrutura de locomoção. Enquanto esses objetos têm um compromisso específico com a classe "Meu Rover", cada uma dessas classes é derivada de uma classe pai, que tende a ser muito mais genérica, subindo até as superclasses.

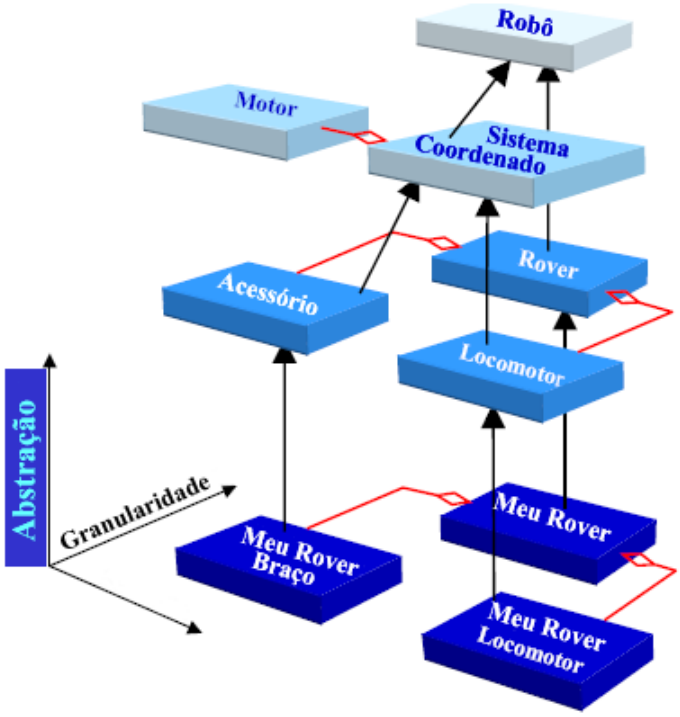


Figura 14 – Exemplo, modificado de [27], ilustrando a hierarquia de objetos e o conceito de herança de classes

- **Gazebo**

A arquitetura do aplicativo Gazebo (Figura 15) é bem clara e modularizada. Essencialmente, existem três blocos principais:

  - modelo – envolve o mundo, forma dos objetos em cena, forma dos robôs, sensores e dispositivos que representam os objetos no mundo real

- interface – praticamente todos os recursos gráficos da simulação utilizando a biblioteca GLUT [59] do OpenGL
- dinâmica – cálculos dinâmicos utilizando a biblioteca ODE [21]

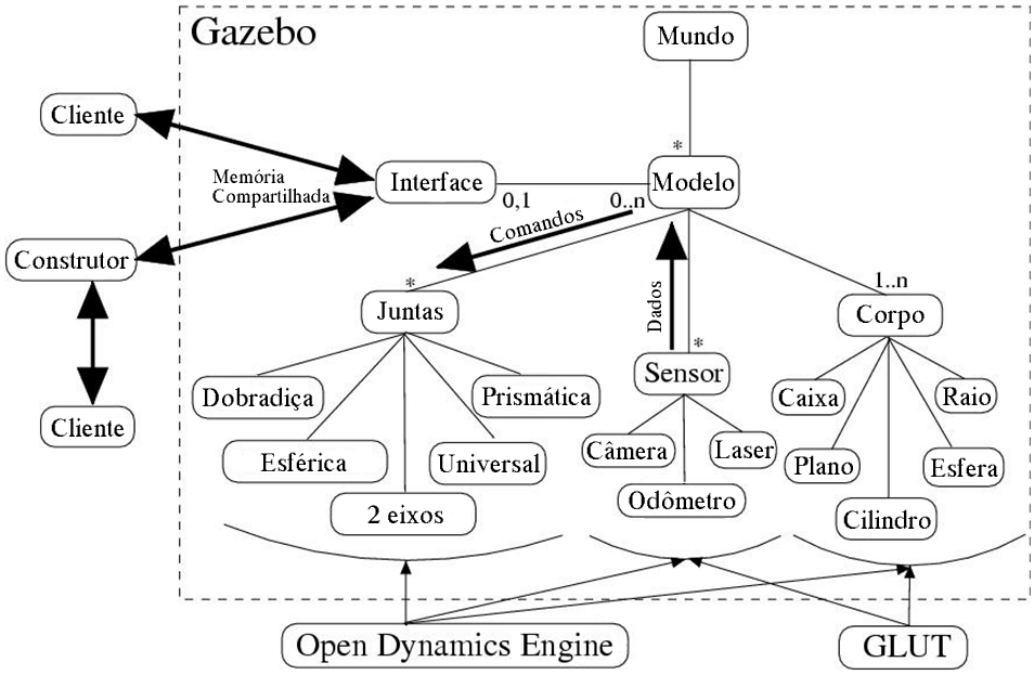


Figura 15 – Diagrama da arquitetura do aplicativo Gazebo, modificado de [60]

2.4.

Biblioteca Gráfica dos Simuladores Avaliados

As principais bibliotecas gráficas em evidência para a construção de objetos em 3D são a OpenGL e Direct3D.

A Direct3D é parte do pacote DirectX desenvolvido pela Microsoft® Corporation. É uma API que provê funcionalidades para criar objetos gráficos em três dimensões, podendo usar os aceleradores de hardware das placas gráficas, se houver. Seu uso é restrito à plataforma Microsoft Windows.

A OpenGL é outra API que provê funcionalidades para criar objetos gráficos em três e duas dimensões, também há a possibilidade de usar os aceleradores da placa gráfica. Está disponível em quase todos os sistemas operacionais atuais, não se limitando apenas ao Microsoft Windows, Mac OS X e Linux.

Abaixo, uma tabela informa a biblioteca gráfica de cada simulador avaliado.

Tabela 2 – Biblioteca gráfica dos sistemas avaliados

	Biblioteca Gráfica	
	OpenGL	**DirectX**
ADAMS	✓	✗
CLARAty	✓	✗
CRAB Rover	✓	✗
Gazebo	✓	✗
RCAST	✓	✗
ROAMS	✓	✗
Rover Graphical Simulator	✓	✗
Universal Mechanisms	✓	✗
Working Model 2D	não informado	não informado
Microsoft Robotics Developer Studio	✗	✓
SIMPACK	✓	✗

2.5.

Biblioteca Utilizada na Dinâmica dos Simuladores Avaliados

O desenvolvimento de simuladores dinâmicos requer a utilização de bibliotecas para solucionar as equações da dinâmica de corpos rígidos, que podem ser públicas, comerciais ou proprietárias (Tabela 3). Entretanto, nem todas as bibliotecas existentes no mercado são de fácil compreensão. Além disso, elas têm mais recursos que o necessário para muitos dos projetos. Para contornar essas situações, algumas das bibliotecas tendem a ser específicas e resolver apenas parte do problema.

Para evitar que partes do código da biblioteca sejam adicionadas sem necessidade, boa parte delas tende a ser modularizada, permitindo a sua link-edição (ou ligação, que consiste em unir ao programa principal, no caso o simulador, apenas as referências resolvidas, ou seja, as chamadas de funções que são referenciadas) com a parte necessária ao simulador ou jogo.

A indústria de jogos tem se beneficiado muito dessas bibliotecas, já que elas permitem a otimização, resultando em menor tempo de execução e mais estabilidade face à precisão, fazendo com que elas sejam vocacionadas para a simulação em tempo real (requerida pelos jogos).

Tabela 3 – Simuladores avaliados e suas bibliotecas de dinâmica

	Biblioteca de Dinâmica
ADAMS	solução proprietária
CLARAty	solução proprietária
CRAB Rover	ODE
Gazebo	ODE
RCAST	AS2TM
ROAMS	Darts/Dshell (solução proprietária)
Rover Graphical Simulator	solução proprietária
Universal Mechanisms	solução proprietária
Working Model 2D	solução proprietária
Microsoft Robotics Developer Studio	PhysX AGEIA
SIMPACK	solução proprietária/ tem também opção para importar equações (apenas para versão licenciada)

Uma atenção especial é dada para o fato de a biblioteca PhysX estar inclusa no hardware de placas de vídeo como ATI e NVidia. Logo, ao utilizar esses recursos, obtém-se um ganho computacional superior ao de qualquer outra biblioteca que não esteja implementada em hardware. Atualmente, a PhysX AGEIA [34] é parte da empresa NVidia. A PhysX é muito eficiente para gerar simulações realísticas, mas não necessariamente realistas. Ou seja, o usuário (em geral de jogos envolvendo efeitos dinâmicos) tem a sensação de realismo ao visualizar as saídas do PhysX, mas os cálculos são aproximados e normalmente não obedecem às condições de contorno em articulações. Isso acontece porque o PhysX modela qualquer sistema como um sistema multicorpos de parâmetros concentrados, conectados por molas e amortecedores. Assim, para definir, por exemplo, uma junta prismática (telescópica) é preciso unir as partes móveis por molas muito rígidas nas direções perpendiculares ao movimento da junta.

A ODE [21] é uma biblioteca voltada para estruturas articuladas, ideal para veículos com pernas. Em sua documentação, [41] enfatiza-se que ela é direcionada para simulações em tempo real, com prioridade para velocidade e estabilidade face à precisão. Dividida em dois níveis de simulação, dinâmica e colisões, tem o objetivo de permitir a flexibilidade e maior utilidade da biblioteca, maximizando com isso seu uso e composição. A parte da dinâmica tem por objetivo cuidar das propriedades dinâmicas dos corpos, tais como massas, velocidades e momentos de inércia, sem levar em consideração as formas dos corpos, enquanto a parte da colisão cuida da forma dos corpos.

A ODE é uma das bibliotecas públicas de junções (articulações) mais ricas, se não for a mais rica. Permite oito tipos diferentes de junções de até 3 graus de liberdade cada. Além disso, permite também a composição de corpos e junções em série, tudo bem documentado e com ilustrações [41].

A biblioteca dinâmica DARTS é uma solução proprietária do laboratório de mesmo nome, da NASA, The DARTS Simulation Laboratory [57]. DARTS também é o acrônimo de *Dynamics And Real-Time Simulation*. Essa biblioteca é vencedora do *NASA Software Award* de 1997, tida como uma biblioteca computacional do mais alto desempenho para a dinâmica de multicorpos flexíveis.

Dshell [57] é um framework (abstração estrutural capaz de reunir diversas ferramentas e funcionalidades para geração de aplicativos) responsável pela criação de simulação em tempo real.

A biblioteca AS2TM [58], AESCO Soft Soil Tire Model, está disponível apenas sob licença de uso comercial e tem um modelo computacional de interação roda-terreno.

No próximo capítulo, o simulador desenvolvido é descrito.

3
Simulador Desenvolvido

O simulador de veículos robóticos desenvolvido para esta dissertação tem mais de 9.690 linhas de códigos e foi todo escrito em C++ (Win32/API), com a utilização da interface gráfica OpenGL. Denominado VirtualBotz 3D, é uma generalização 3D de trabalhos feitos em 2D por Barral [4] e Santos [5]. Houve um cuidado particular no realismo do sistema, em especial na modelagem da dinâmica, para permitir características não-lineares em terrenos acidentados que revelem ter alto grau de importância para o comportamento de robôs reais, como o Veículo para Inspeção Visual Interna de dutos (VIVI, Figura 16) desenvolvido pelo Laboratório de Robótica da PUC-Rio para inspeção de tubulações.

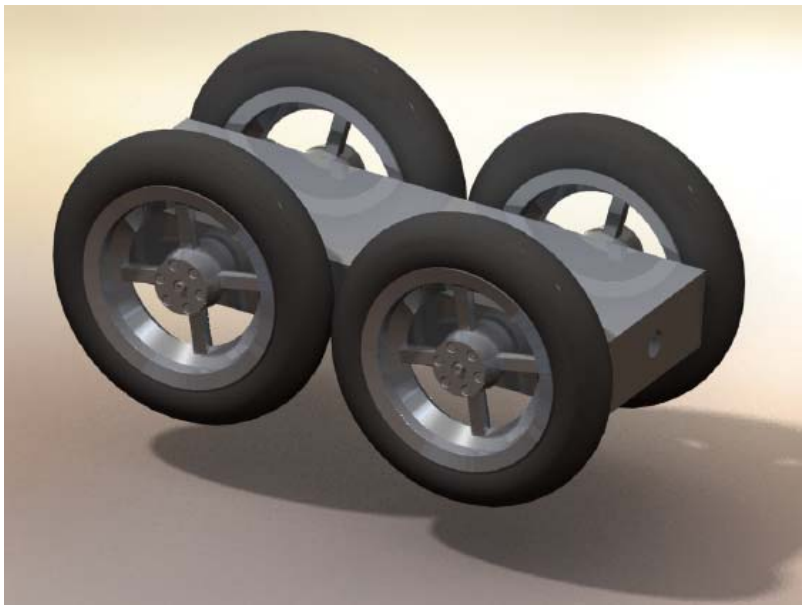


Figura 16 – Veículo para Inspeção Visual Interna de dutos (VIVI)

O simulador desenvolvido permite a visualização 3D em tempo real da interação entre robôs móveis e os mais diversos tipos de terrenos, além de

- importar perfis de terrenos em forma de matriz de elevação e mapa de altura, inclusive com a possibilidade de construir perfis de terrenos a partir de funções;
- importar texturas para o terreno e alterar escalas a partir do script do simulador;
- importar diferentes partes do robô, como chassi e roda, por meio de arquivos no formato wavefront (.OBJ), com características visuais de materiais;
- implementar diferentes tipos de algoritmos de controle;
- interagir com o veículo robótico no ambiente virtual em tempo real com a utilização de um joystick analógico;
- alterar o ponto de vista utilizando o teclado ou o joystick;
- utilizar diferentes câmeras, permitindo a visualização de diferentes ângulos, com ajustes também pelo teclado ou joystick, dependendo do tipo de câmera;
- gravar dados do sistema como velocidades, acelerações, torques, forças, etc., em arquivos texto, para posterior análise do algoritmo de controle ou comportamento dinâmico do sistema;
- gravar a simulação em arquivo de vídeo em diferentes tipos e formatos;
- poder implementar diferentes tipos de motores de corrente contínua usados na locomoção do sistema, mediante a alteração dos parâmetros em um arquivo de script em formato texto;
- visualizar a trajetória do veículo robótico na simulação por meio de vetores ou linha contínua, com a possibilidade de comparar trajetórias ao final de uma simulação;
- poder, após a simulação, posicionar o veículo robótico em um ponto específico da sua trajetória;
- se o computador tiver mais de um processador, poder reservar um processador exclusivo para executar os cálculos dinâmicos;
- poder utilizar um passo temporal de integração da ordem de 10^{-7}s;
- poder ajustar a frequência para o controle.

Os arquivos de códigos gerados em linguagem C++ estão divididos em pastas, para uma percepção clara e imediata de sua organização (**Anexo A**), e separados em classe de negócio e de interface. O simulador pode ser transportado para qualquer outra plataforma sem nenhuma dificuldade. As pastas estão divididas em

- *_library* – contendo os arquivos com definição de tipos, variáveis globais e pastas de classes de negócio e interface, dentre outras
- *_library/Botz* – classes de negócio
- *_library/Graph* – classes gráficas para leitura de arquivos de imagens e outros arquivos do gênero
- *_library/Math* – classes para matrizes e vetores
- *_library/OpenGL* – classes de interface
- *_library/Tools* – ferramentas para manipulação de matrizes (*arrays*) de ponteiros e geração de valores aleatórios

No **Anexo B** foram gerados os diagramas em UML (*Unified Modeling Language*), para que se tenha uma visão clara da organização das classes envolvidas no sistema, assim como seus relacionamentos. É bom salientar que o padrão de linguagem gráfica UML [46] é uma ferramenta que permite a visualização em gráficos padronizados, em forma de diagrama, e que não tem a capacidade de informar quais passos devem ser tomados e muito menos como projetar o sistema. Por trás de todas as suas notações gráficas, o UML acaba gerando a semântica do sistema, ferramenta de extrema importância para os desenvolvedores.

3.1.

Descrição Geral

Na modelagem do simulador, foi utilizado o conceito de modularização da Programação Orientada a Objetos (**Anexo B**). Assim que o programa é executado, a janela principal exibe uma imagem do terreno e do veículo robótico, que será chamada de “mundo virtual”, ou simplesmente “mundo”. O veículo robótico se

movimentará comandado apenas por um joystick analógico conectado ao computador. Assim como o veículo robótico, o mundo virtual também pode ser movimentado, ou melhor, reposicionado, utilizando o próprio joystick, caso tenha dois manetes distintos, ou então, o mouse. Ambos os manetes de reposicionamento do mundo permitem a sua rotação, aproximação ou afastamento, e o reposicionamento do observador (**Anexo C**).

Para a simulação, utilizam-se dois arquivos-texto contendo toda configuração da simulação. No primeiro arquivo, o principal (**Anexos E**), apenas se faz referência ao segundo arquivo de simulação/configuração, propriamente dito, onde se encontram as definições do terreno e do veículo robótico (**Anexo G** e **I**, respectivamente). Dessa forma, é possível guardar várias configurações a serem executadas assim que necessário, apenas comentando ou não a linha contendo o arquivo corrente de simulação/configuração. Assim como os arquivos contendo os códigos do simulador, o código executável e seus acessórios também estão organizados em pastas bastante intuitivas:

- *Build* – pasta contendo os arquivos executáveis do simulador, tanto para a versão de teste, chamada de beta, quanto para a versão final, chamada de release. Nesta pasta também estão os arquivos scripts para configuração da simulação.
- *Build/Heightmaps* – pasta com imagens para mapas de alturas do terreno.
- *Build/Obj_files* – pasta com todos os arquivos de modelo para terreno do tipo malha regular, *mesh* (.msh), para rodas e chassi do tipo wavefront (.OBJ), incluindo os arquivos de materiais (.MTL) para os modelos wavefront.
- *Build/Report* – pasta com todas as saídas de dados geradas em arquivos específicos durante uma simulação.
- *Build/Scripts* – pasta com todos os scripts de simulações.
- *Build/Tests* – pasta com arquivos com coordenadas *X*, *Y* e *Z* para comparar trajetórias do veículo robótico.

- *Build/Textures* – pasta com todas as texturas a serem utilizadas pelo simulador.
- *Build/Videos* – pasta com todos os arquivos de vídeo gerados para cada simulação, se necessário.

3.2.
Modelagem do Terreno

Os modelos mais utilizados na representação de relevos (terrenos acidentados) são curvas de nível, grade regular retangular conhecida também por *digital elevation model* (DEM), rede triangular irregular (*triangular irregular network* – TIN), conhecida por triangulação, e o diagrama de Voronoi.

Para o simulador VirtualBotz 3D foi utilizado o modelo grade regular retangular (Figura 17). Os modelos de terrenos (**Anexo F**) podem ser criados a partir do conjunto de amostras do terreno ou por qualquer modelador geométrico 3D que possibilite a armazenagem dos dados em forma de matriz de elevação.

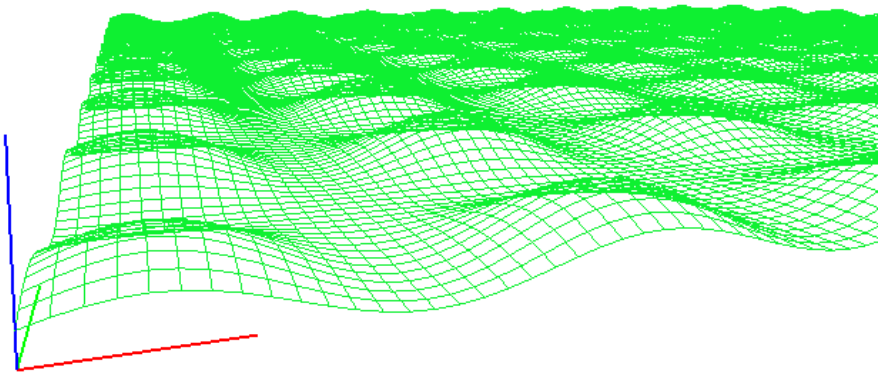


Figura 17 – Modelo de grade regular retangular utilizado pelo VirtualBotz 3D

O terreno é especificado por meio de uma matriz de duas dimensões denominada $\underline{M}$, onde cada elemento em $\underline{M}$ corresponde à coordenada z, ou altura do terreno, em um sistema de coordenadas (x, y) em um plano horizontal. Os

extremos do domínio do terreno no VirtualBotz 3D (Figura 18) têm as coordenadas (x_{min}, x_{max}) e (y_{min}, y_{max}).

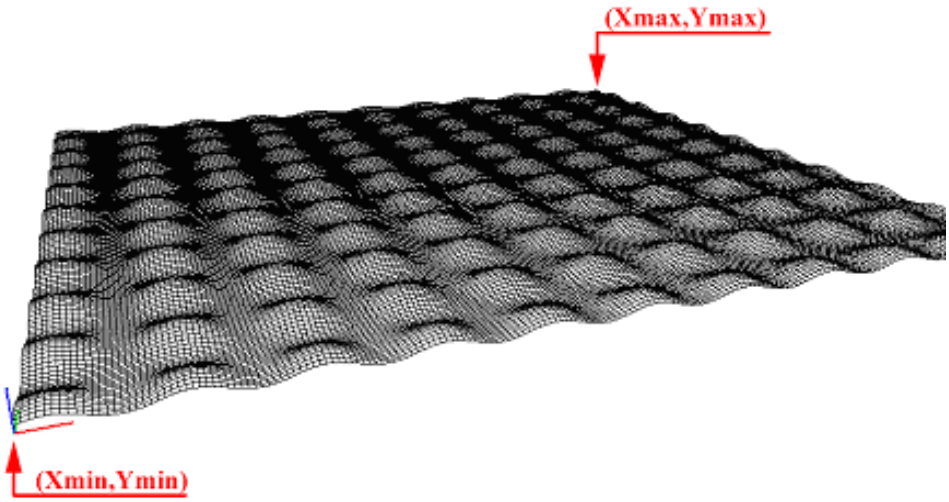


Figura 18 – Extremos do domínio do terreno no VirtualBotz 3D

$$x = x_{min} + i\frac{(x_{max} - x_{min})}{(n_x - 1)} \tag{3.1}$$

$$y = y_{min} + j\frac{(y_{max} - y_{min})}{(n_y - 1)} \tag{3.2}$$

Para os elementos i e j da matriz $\underline{M}$, n_x e n_y são respectivamente o número de linhas e colunas de $\underline{M}$.

Para estabelecer a altura do terreno entre os elementos da matriz $\underline{M}$, foi implementado um algoritmo de interpolação linear que busca o elemento da matriz imediatamente anterior, tanto em x quanto em y, no ponto desejado, e o pondera, utilizando os outros três elementos vizinhos da seguinte forma:

$$z_1 = \underline{M}_{ix+1,iy}\left(aux_x - i_x\right) + \underline{M}_{ix,iy}\left(1 - \left(aux_x - i_x\right)\right) \tag{3.3}$$

$$z_2 = \underline{M}_{ix+1,iy+1}\left(aux_x - i_x\right) + \underline{M}_{ix,iy+1}\left(1 - \left(aux_x - i_x\right)\right) \tag{3.4}$$

$$z_f = z_2\left(aux_y - i_y\right) + z_1\left(1 - \left(aux_y - i_y\right)\right) \tag{3.5}$$

onde (i_x, i_y) são as coordenadas do elemento de $\underline{M}$ imediatamente anterior ao elemento desejado, aux_x e aux_y são as coordenadas do ponto desejado convertidas para o sistema de coordenadas da matriz $\underline{M}$, $\underline{M}_{i,j}$ é o elemento (i, j) de $\underline{M}$, e z_f é o resultado da interpolação dos elementos do terreno na posição desejada. A Figura 19 ilustra as variáveis descritas.

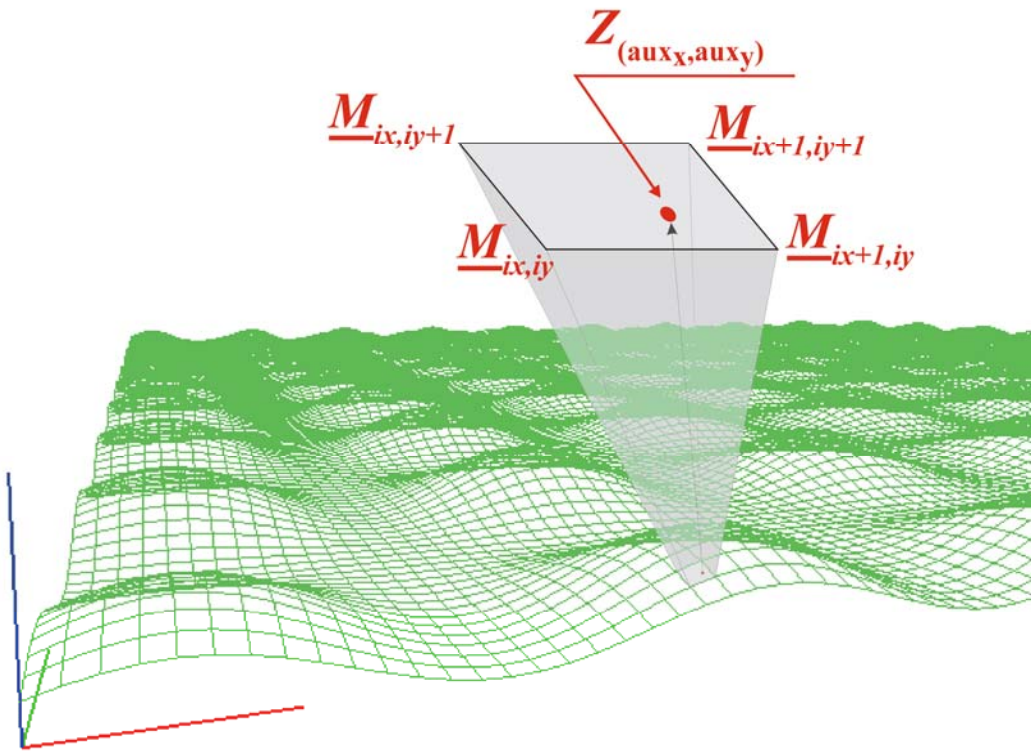


Figura 19 – Ilustração das variáveis de interpolação do terreno

3.3.
Modelagem do Veículo Robótico

3.3.1.
Parâmetros Geométricos do Veículo Robótico

O veículo é modelado como um chassi rígido, contendo n rodas com suspensão independente do tipo mola-amortecedor não linear. Para visualizar a configuração do veículo robótico no simulador, vide o **Anexo I**.

Na Figura 20 encontra-se o veículo robótico e a localização de seu centro de massa, que recebe o nome de $\boldsymbol{X_c}$. A sua orientação é obtida a partir de três vetores unitários $\boldsymbol{n}$, $\boldsymbol{t}$ e $\boldsymbol{b}$, respectivamente nas direções $\boldsymbol{x}$, $\boldsymbol{y}$ e $\boldsymbol{z}$. As dimensões do chassi do veículo são definidas nessas três direções, havendo a possibilidade de posicionar o centro de massa de forma assimétrica com o uso de duas constantes em cada direção. Desta forma, pode-se dizer que a largura do chassi é dada pela soma de w_1 com w_2, o comprimento pela soma de l_1 e l_2, e a altura pela soma de h_{topo} e h_{base}, onde o primeiro é a distância entre o teto e o centro de massa do veículo e o segundo é a distância até o fundo do chassi. A posição do seu centro de massa é representada pelo vetor $\boldsymbol{X_c}$ representado em um sistema inercial global.

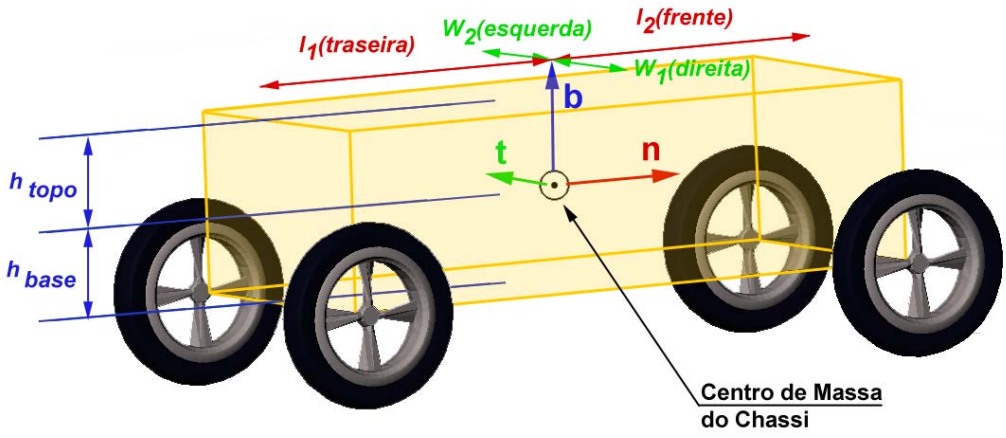


Figura 20 – Esquema com as dimensões do chassi

Cada roda do veículo tem o seu centro de massa (Figura 21) representado por três possíveis vetores: $\boldsymbol{X_{teo}}$ (posição teórica, sem nenhuma deformação da suspensão segundo o referencial global), $\boldsymbol{X_{ntb}}$ (posição teórica, sem nenhuma deformação da suspensão segundo o referencial **ntb** local) e $\boldsymbol{X_{cm}}$ (posição real, considerando os efeitos da suspensão segundo o referencial global).

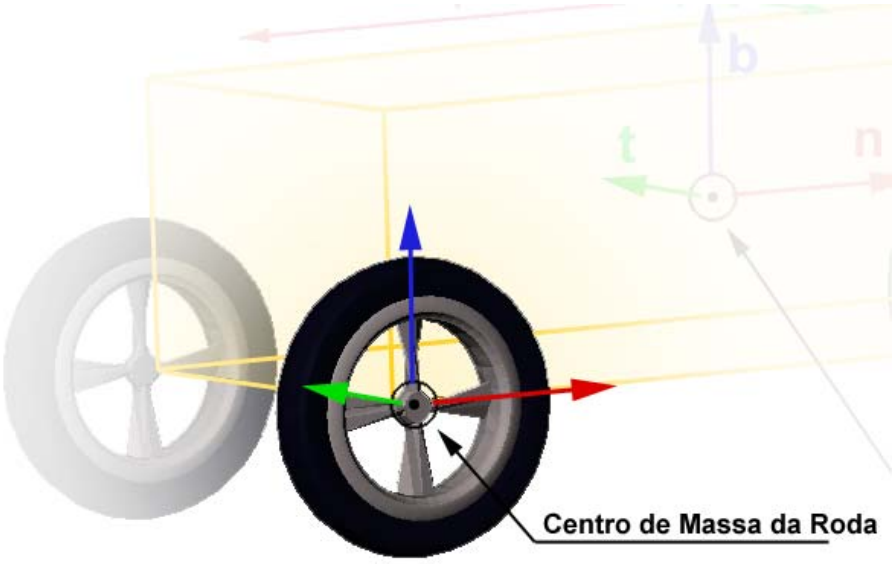


Figura 21 – Centro de massa teórico da roda, sem qualquer deformação na suspensão

A Figura 22 ilustra as dimensões da roda, apresentando o centro de massa teórico da roda sem qualquer deformação na suspensão, a uma distância (x, y, z) do centro de massa do veículo robótico, com raio r e largura representada por w. Na ilustração, a roda em destaque tem dois aros, um na parte mais externa e outro na parte mais interna da roda com o chassi. Logo, a coordenada y local de cada roda deverá ser o deslocamento em y do centro de massa do veículo $(\pm w/2)$.

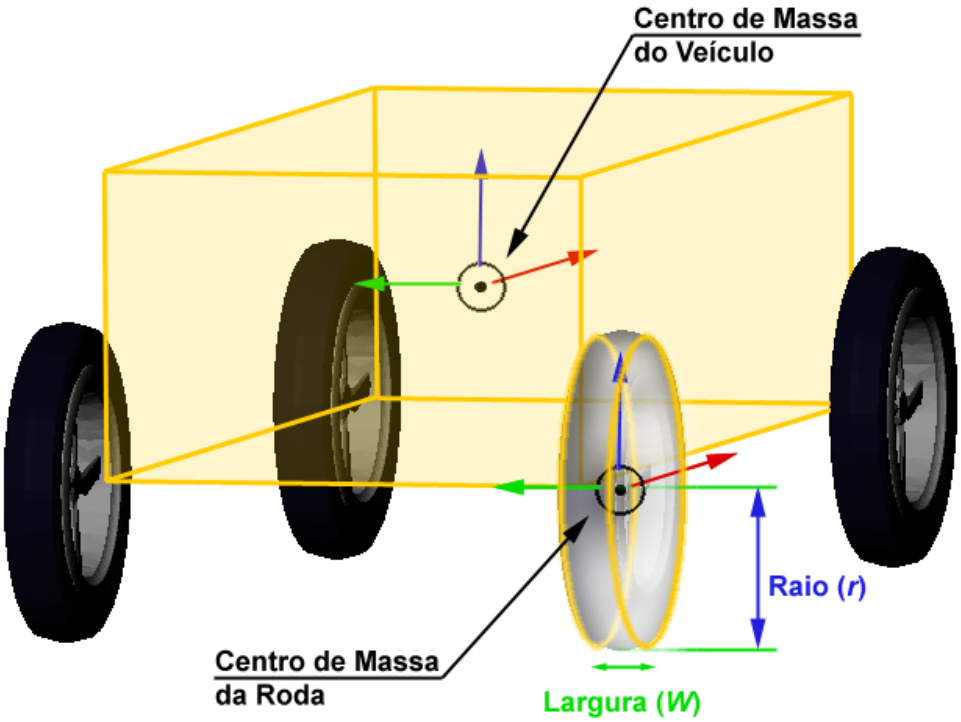


Figura 22 – Dimensões da roda

As definições das coordenadas locais do centro de massa de cada roda devem ser feitas no atributo “coordenadas local do centro de massa (*x, y, z*)” de cada roda do veículo robótico, localizado no arquivo script de configuração do simulador. Para maiores detalhes, vide os **Anexos E**, **G**, **I**, **J** e **K**.

3.3.2.

Forças Externas sobre o Veículo Robótico

As forças externas (Figura 23) sobre o veículo robótico podem ser descritas pelas forças de interação com o terreno e a força gravitacional ***P***. A força de interação com o terreno pode ser decomposta em três vetores ortogonais: $\boldsymbol{F_x}$, que representa a força de tração; ***N***, a força normal; e $\boldsymbol{F_y}$, a componente transversal.

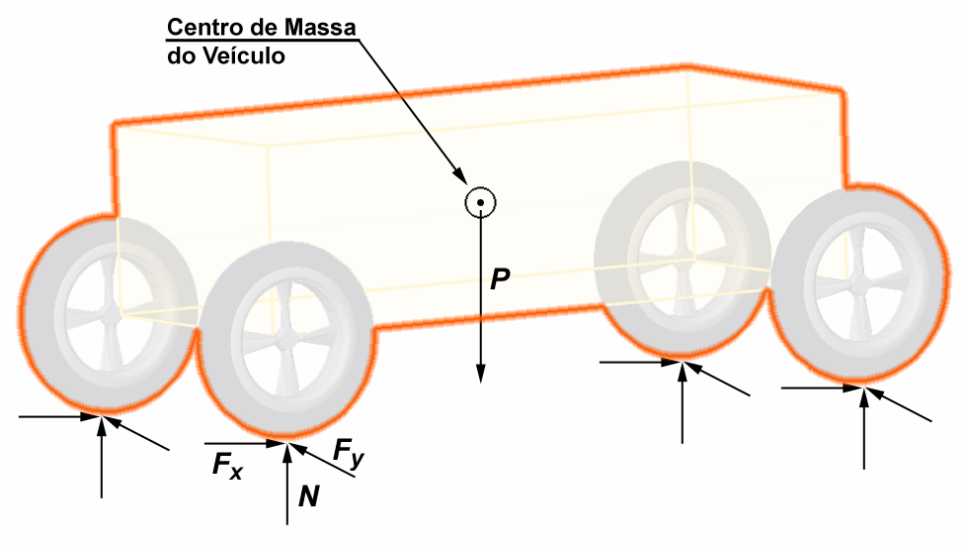


Figura 23 – Esquema de forças externas atuando sobre o veículo robótico

3.3.3.

Forças Internas entre a Roda e o Chassi

A Figura 24 mostra as forças de atuação nos eixos de cada roda. Para cada centro de massa de cada roda, tem-se a posição $\boldsymbol{X_{cm}}$, que corresponde à posição real da roda com deformação da suspensão, segundo o referencial global. Há, também, no centro de massa de cada roda, vetores representando as componentes da força: $\boldsymbol{F_{susp}}$, componente que representa a força da roda na direção $\boldsymbol{b}$, atuando diretamente sobre a suspensão; e $\boldsymbol{F_n}$, componente da força que impulsiona o chassi para a frente, na direção $\boldsymbol{n}$.

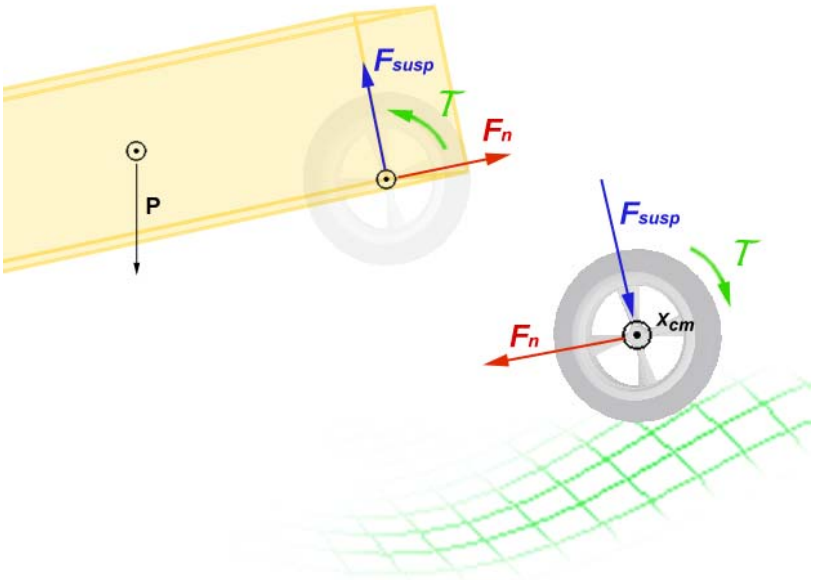


Figura 24 – Esquema de forças interna entre a roda e o chassi

3.3.4.

Algoritmo de Busca do Ponto de Contato entre Roda e Terreno

Nesta seção, todo o cálculo numérico das deformações das suspensões se acomodando ao terreno é apresentado, para determinar os pontos de contato entre cada roda e o terreno.

Para encontrar o ponto de contato da roda com o terreno, primeiro é preciso verificar o deslocamento da suspensão imposto pelo terreno (Figura 25). Na implementação do simulador, para evitar vibrações de alta frequência das suspensões, que exigiriam um passo de integração muito reduzido, modelaram-se as rodas como se não tivessem massa. Assim, a suspensão estará sempre em repouso, se não houver interação com o terreno, ou comprimida, se houver interação com o terreno, sendo impossível que a suspensão seja tracionada. Note, porém, que o momento de inércia de cada roda não é desprezado, o que influencia na aceleração do sistema.

Para o deslocamento de cada suspensão utiliza-se uma variável h. Admite-se que, caso esse deslocamento esteja dentro da faixa de valores de 0 (zero) até um valor de saturação h_{sat}, as equações para a força exercida pela suspensão são lineares; e, caso o deslocamento seja maior que o limite h_{sat}, utiliza-se uma relação não linear para a força.

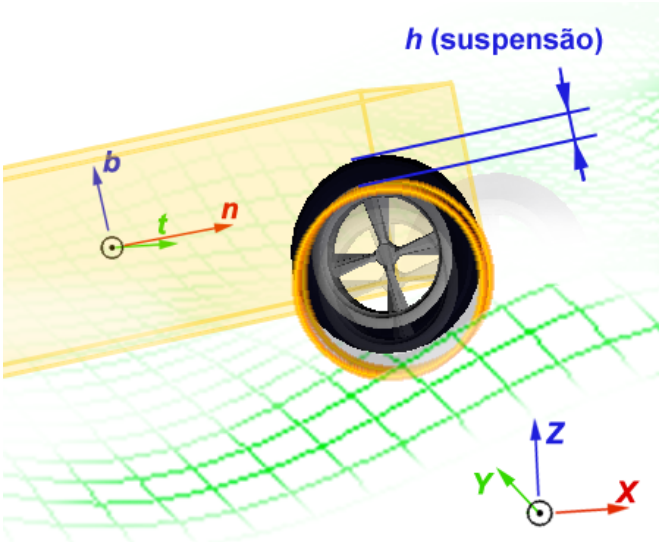


Figura 25 – Deslocamento imposto sobre a suspensão

Para determinar o ponto de contato da roda com o terreno, é preciso levar em conta que cada roda tem o seu centro geométrico representado por três possíveis vetores (como visto na Seção 3.3.1.):

- $\boldsymbol{X}_{ntb}$ – posição teórica do centro geométrico da roda sem nenhuma deformação da suspensão, segundo o referencial **ntb** local
- $\boldsymbol{X}_{teo}$ – posição teórica do centro da roda caso não houvesse nenhuma deformação da suspensão, segundo o referencial global
- $\boldsymbol{X}_{cm}$ – posição real do centro da roda considerando os efeitos da suspensão, segundo o referencial global

O algoritmo de determinação do ponto de contato entre a roda e o terreno calcula primeiro o $\boldsymbol{X}_{teo}$:

$$\boldsymbol{X}_{teo} = \boldsymbol{X}_{cm} + \boldsymbol{X}_{ntb}(x)\cdot\boldsymbol{n} + \boldsymbol{X}_{ntb}(y)\cdot\boldsymbol{t} + \boldsymbol{X}_{ntb}(z)\cdot\boldsymbol{b} \tag{3.6}$$

Logo, uma vez determinado o centro geométrico teórico de cada roda, é preciso calcular o seu centro de massa, levando em consideração os efeitos da suspensão. Com isso, tem-se também o ponto de contato da roda com o terreno. Para determinar o deslocamento h da suspensão e o ponto de contato da roda com o terreno $\boldsymbol{X}_{cp}$, discretiza-se a parte inferior da circunferência da roda em n_{fatias}, ou pontos (Figura 26).

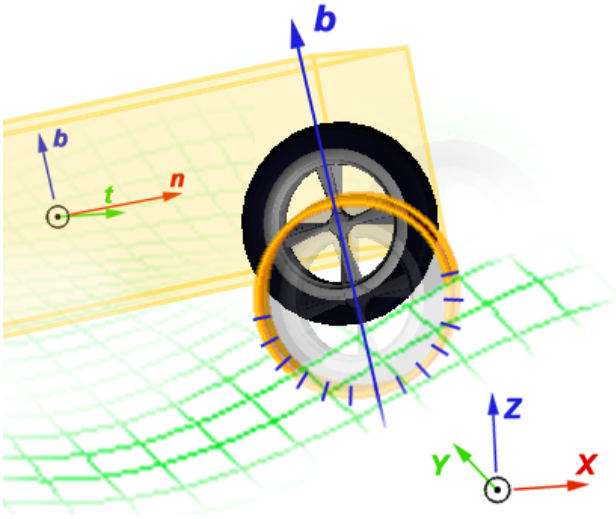


Figura 26 – Ilustração da discretização da circunferência inferior da roda

No script de simulação, descrito no **Anexo E** e **I**, há um atributo que permite escolher o valor da discretização dessa circunferência:

```
...
quantidade de divisoes da roda = 18
...
```

Um número muito elevado permite maior precisão no cálculo do ponto de contato roda-terreno, mas pode reduzir a velocidade de cálculo do simulador, dependendo do computador usado.

Para o algoritmo, cada ponto de discretização corresponderá a um ângulo γ entre a linha de trabalho da suspensão e a força normal (Figura 27). Desse modo, é possível estimar os efeitos da suspensão mediante um deslocamento *h*. O ponto de contato entre a roda e o terreno é representado pelo vetor $\boldsymbol{X_{cp}}$, de acordo com o sistema de coordenadas globais. Entretanto, conhecer o $\boldsymbol{X_{cp}}$**'** é fundamental para encontrar o deslocamento da suspensão.

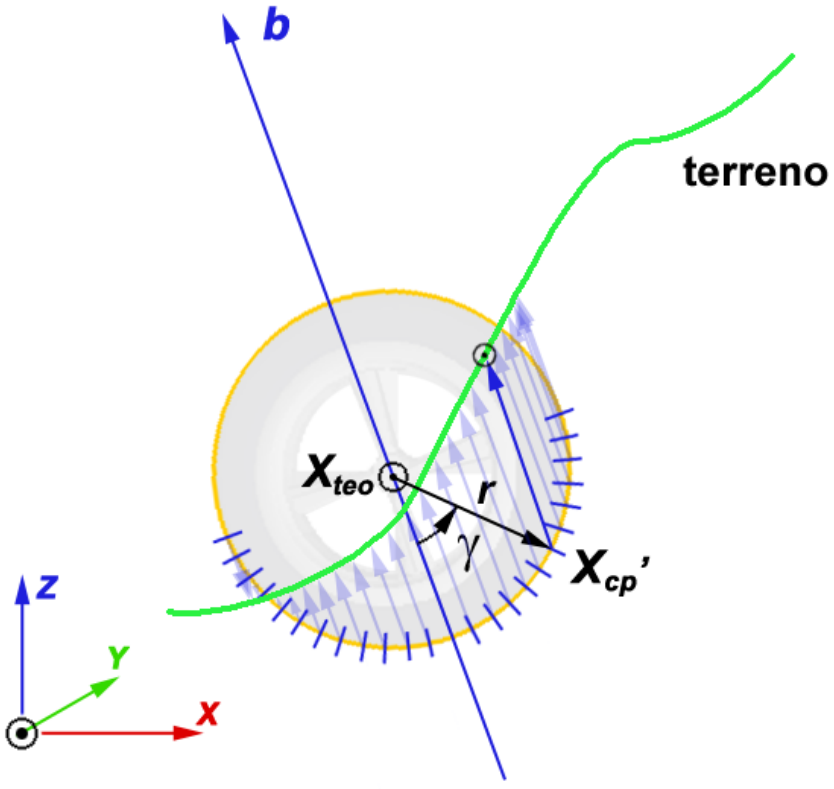


Figura 27 – Ilustração do esquema do cálculo de $\boldsymbol{X_{cp}}$**'**

Para encontrar $\boldsymbol{X_{cp}}$ é preciso, em primeiro lugar, procurar dentre todos os pontos discretizados $\boldsymbol{X_{cp}}$**'** aquele que gera o maior deslocamento de *h* (Figura 28).

$$\boldsymbol{X_{cp}}' = \boldsymbol{X_{teo}} + r \cdot sin(\gamma) \cdot \boldsymbol{n} - r \cdot cos(\gamma) \cdot \boldsymbol{b} \pm {}^{w}/_{2} \cdot \boldsymbol{t} \qquad (3.7)$$

Nesse ponto, há o interesse em descobrir o ponto de interseção entre a roda e o terreno a partir do vetor $\boldsymbol{X_{cp}}$**'** (Figura 28).

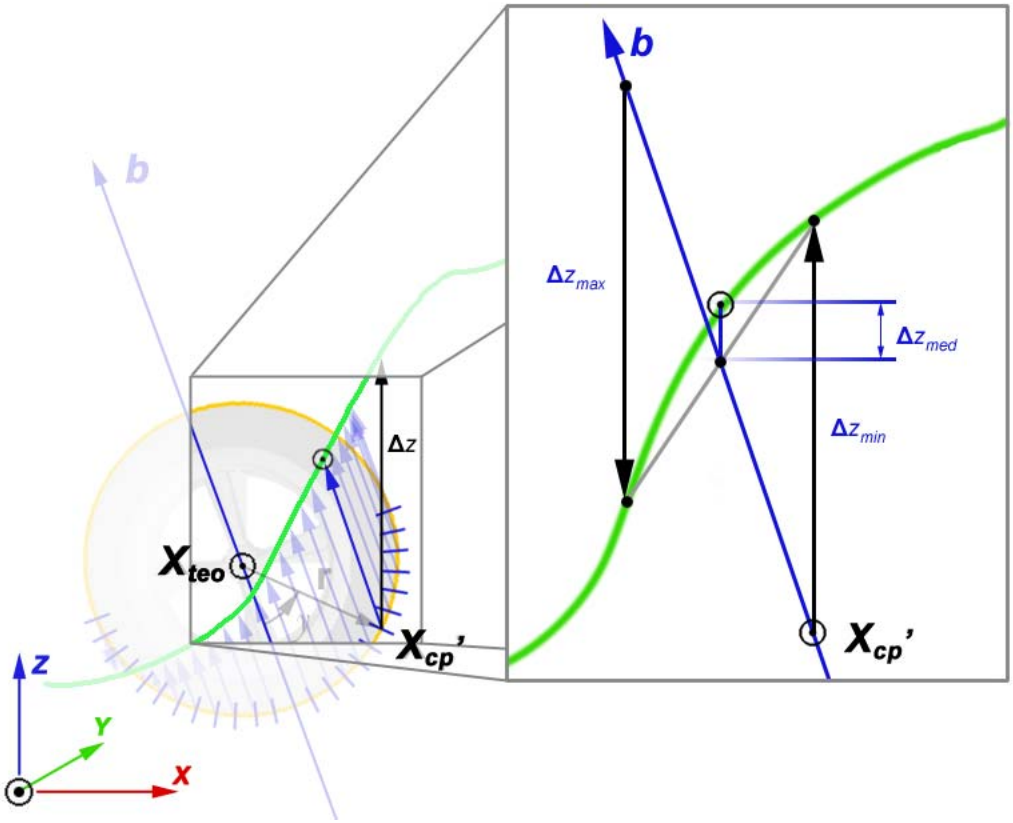


Figura 28 – Ilustração da convergência do ponto de contato da roda com o terreno

Com $\boldsymbol{X_{cp}}$**'** calculado, procura-se então o valor de Δz_{med}, definido na figura, de modo a encontrar um deslocamento *h* maior ou igual a zero e menor ou igual ao deslocamento h_{sat}. Para a estimativa do deslocamento *h*, utiliza-se um método iterativo em que se procura um valor de *h* que satisfaça a equação $Z_{cp} - f\left(X_{cp}, Y_{cp}\right) = 0$. Para isso, inicialmente calcula-se, para um dado γ, à distância Δz entre a coordenada Z_{cp} do ponto discretizado da roda, para os valores

extremos admissíveis para h. Com a ponderação dos valores de Δz encontrados, estima-se h_{med}:

$$\begin{aligned} Z_{cp}\left(h_{min}\right) &= \boldsymbol{X_{cp}}' + h_{min}\cdot\boldsymbol{b}\left(z\right) \\ \Delta z_{min} &= Z_{cp}\left(h_{min}\right) - f\left(\boldsymbol{X_{cp}} + h_{min}\cdot\boldsymbol{b}\left(x\right), Y_{cp} + h_{min}\cdot\boldsymbol{b}\left(y\right)\right) \end{aligned} \tag{3.8}$$

$$\begin{aligned} Z_{cp}\left(h_{max}\right) &= \boldsymbol{X_{cp}}' + h_{max}\cdot\boldsymbol{b}\left(z\right) \\ \Delta z_{max} &= Z_{cp}\left(h_{max}\right) - f\left(\boldsymbol{X_{cp}} + h_{max}\cdot\boldsymbol{b}\left(x\right), Y_{cp} + h_{max}\cdot\boldsymbol{b}\left(y\right)\right) \end{aligned} \tag{3.9}$$

$$h_{med} = h_{min} + \frac{\left(h_{max} - h_{min}\right)\left(-\Delta z_{min}\right)}{\left(\Delta z_{max} - \Delta z_{min}\right)} \tag{3.10}$$

$$\begin{aligned} Z_{cp}\left(h_{med}\right) &= \boldsymbol{X_{cp}}' + h_{med}\cdot\boldsymbol{b}\left(z\right) \\ \Delta z_{med} &= Z_{cp}\left(h_{med}\right) - f\left(\boldsymbol{X_{cp}} + h_{med}\cdot\boldsymbol{b}\left(x\right), Y_{cp} + h_{med}\cdot\boldsymbol{b}\left(y\right)\right) \end{aligned} \tag{3.11}$$

Determinadas as distâncias $\Delta z_{\min}$, Δz_{med} e $\Delta z_{\max}$, verificam-se os seus sinais. Caso $\Delta z_{\max}$ seja positivo e Δz_{med} negativo, ou vice-versa, então o valor desejado encontra-se entre os valores de h_{max} e h_{med}: logo, h_{med} passará a ser o h_{min} da iteração seguinte. Para o caso em que Δz_{med} é negativo e $\Delta z_{\min}$ positivo, ou vice-versa, o h_{med} encontrado passará a ser o novo h_{max} da próxima interação. Se ambos $\Delta z_{\max}$ e $\Delta z_{\min}$ forem positivos, então a roda não faz contato com o terreno e, portanto, $h = 0$. Finalmente, se ambos forem negativos, então a suspensão está trabalhando na região não linear e, com isso, o h_{med} passa a ser o h_{sat}. O método iterativo termina quando a variação absoluta entre valores sucessivos de h_{med} é menor que um valor percentual de h_{sat} predeterminado ou quando h_{med} atinge um dos valores extremos 0 (zero) ou h_{sat}.

Tendo encontrado a melhor estimativa do deslocamento h de cada suspensão, calcula-se o centro de massa da roda e, em seguida, o seu ponto de contato com o terreno:

$$\boldsymbol{X}_{cm} = \boldsymbol{X}_{teo} + h \cdot \boldsymbol{b} \tag{3.12}$$

$$\boldsymbol{X}_{cp} = \boldsymbol{X}_{cm} + r \cdot sin(\gamma) \cdot \boldsymbol{n} - r \cdot cos(\gamma) \cdot \boldsymbol{b} \pm {}^{w}\!/_{2} \cdot \boldsymbol{t} \tag{3.13}$$

A Figura 29 mostra a posição real do centro de massa da roda, considerando o deslocamento da suspensão imposto pelo terreno, para todas as rodas do veículo robótico, além do ponto de contato da roda com o terreno, ambos segundo o referencial global.

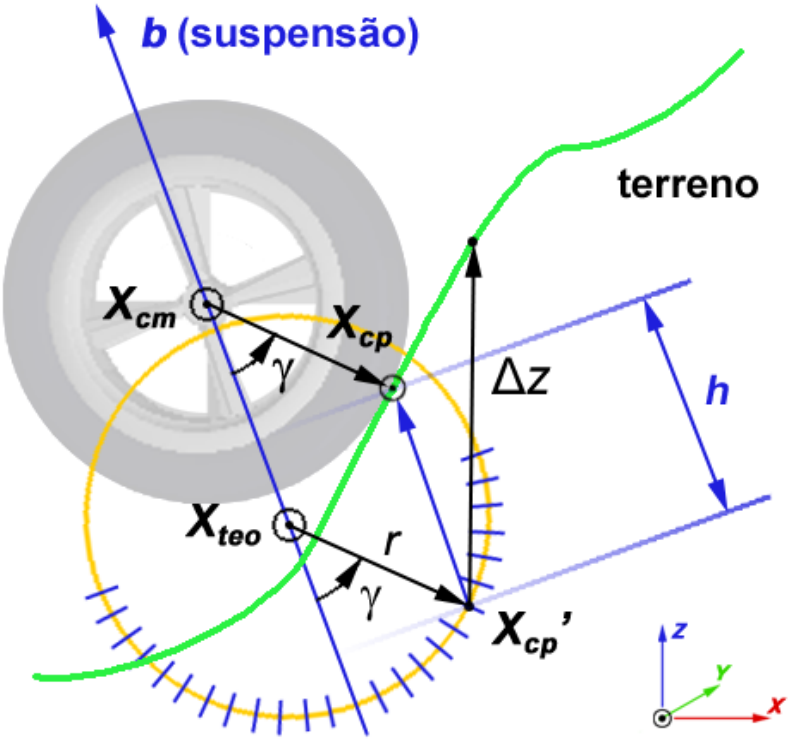


Figura 29 – Esquema do ponto de contato da roda com o terreno

3.4.

Modelagem Dinâmica do Veículo Robótico

O modelo proposto para o cálculo dos esforços que agem sobre o veículo robótico não leva em consideração o esterçamento das rodas, que normalmente

não existe na maioria dos veículos robóticos, pois eles têm o *tank steering* como diferencial de locomoção. Assim, todas as rodas sempre estarão paralelas entre si.

A partir do cálculo do ponto de contato $\boldsymbol{X_{cp}}$, mostrado na Seção 3.3.4, determina-se o deslocamento imposto à suspensão e sua velocidade e, com isso, pode-se calcular a amplitude da força da suspensão F_{susp} – incluindo os efeitos de rigidez e amortecimento.

Pelo equilíbrio de forças, a amplitude da força de impulsão do veículo F_n e a amplitude da força normal N são obtidas em função de F_{susp} e F_x por meio de

$$F_n = \frac{F_x - F_{susp} sin(\gamma)}{cos(\gamma)} \tag{3.14}$$

$$N = \frac{F_{susp} - F_x sin(\gamma)}{cos(\gamma)} \tag{3.15}$$

lembrando que o γ é o ângulo entre $\boldsymbol{N}$ e $\boldsymbol{b}$ (linha de trabalho da suspensão).

Assumindo que inicialmente a roda não esteja escorregando, a amplitude da força longitudinal F_x entre a roda e o terreno será dada pela relação entre o torque aplicado à roda e o seu raio. Logo, a princípio, tem-se que $F_x = \tau/r$, onde τ é o torque aplicado à roda e r, o raio. No entanto, é preciso verificar se é possível aplicar F_x, pois se a roda estiver acima do limite de aderência do pneu, esse valor deve ser ajustado utilizando o coeficiente de atrito dinâmico.

A Figura 30 mostra os esforços agindo entre a roda e o terreno. Pode-se ver que se $cos(\gamma) \leq 0$ o ponto de contato entre a roda e o terreno encontra-se no hemisfério superior da roda em relação ao sistema do chassi. Nesse caso específico, por se tratar de uma colisão ou travamento do veículo, assume-se que $F_n = 0$, levando a $F_x = F_{susp}$ e $N = 0$. Para $cos(\gamma) \geq 0$, é necessário cogitar dois casos:

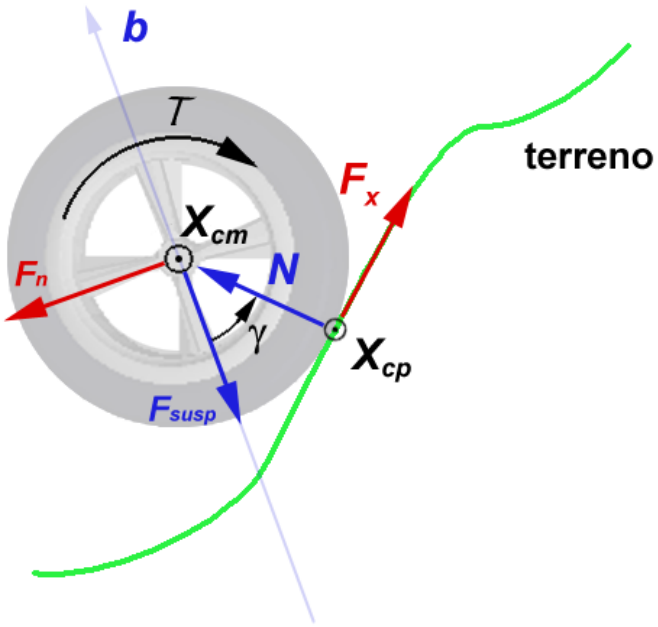


Figura 30 – Esforços agindo entre a roda e o terreno

1. Se $F_{susp} \leq \left(F_x sin(\gamma)\right)$, tem-se que, pela Equação (3.15), $N \leq 0$. Como não é possível ter uma força normal negativa, assume-se que $N = 0$. Se a força normal é nula, a roda não faz contato com o terreno, logo a força de tração F_x também é nula e a suspensão encontra-se relaxada (pois desprezou-se a inércia de translação das rodas) e portanto $F_{susp} = 0$;

2. Também pela Equação (3.15), se $F_{susp} > \left(F_x sin(\gamma)\right)$ então $N > 0$. Sabendo que $-\mu N \leq F_x \leq \mu N$, onde μ é o coeficiente de atrito estático, a força de tração pode ser analisada de duas maneiras:

   a. Se $F_x \geq 0$, então $F_x \leq \mu N$. Da Equação (3.15) tem-se que

$$F_x \leq \left(\mu \frac{F_{susp}}{cos(\gamma)} - \mu F_x \frac{sin(\gamma)}{cos(\gamma)} \right) \tag{3.16}$$

logo,

$$F_x\left(cos\left(\gamma\right)+\mu sin\left(\gamma\right)\right)\leq \mu F_{susp} \tag{3.17}$$

Se $\left(cos\gamma+sin\left(\gamma\right)\right)>0$, então

$$F_x \leq \frac{\mu F_{susp}}{cos\left(\gamma\right)+\mu sin\left(\gamma\right)} \tag{3.18}$$

caso a inequação (3.18) não seja satisfeita, o valor de F_x deverá ser ajustado de acordo com

$$F_x = \frac{\mu^* F_{susp}}{cos\left(\gamma\right)+\mu^* sin\left(\gamma\right)} \tag{3.19}$$

onde μ^* é o coeficiente de atrito dinâmico, pois a roda estaria escorregando. No entanto, se $\left(cos\left(\gamma\right)+\mu sin\left(\gamma\right)\right)\leq 0$, então o valor encontrado para F_x não precisa ser corrigido, pois o primeiro termo encontrado da inequação (3.17) será sempre negativo e menor que o segundo termo porque, nesse caso, F_{susp} é sempre positivo;

b. Se $F_x<0$, então $-F_x \leq \mu N$. Da mesma maneira descrita acima, da Equação (3.15) chega-se a

$$F_x\left(cos\left(\gamma\right)-\mu sin\left(\gamma\right)\right)\geq -\mu F_{susp} \tag{3.20}$$

Se $\left(cos\left(\gamma\right)-sin\left(\gamma\right)\right)>0$, então

$$F_x \geq \frac{-\mu F_{susp}}{cos\left(\gamma\right)-\mu sin\left(\gamma\right)} \tag{3.21}$$

Caso a inequação (3.21) não seja satisfeita, o valor de F_x deverá ser ajustado de acordo com

$$F_x = \frac{-\mu^* F_{susp}}{cos(\gamma) - \mu^* sin(\gamma)} \tag{3.22}$$

No entanto, se $(cos(\gamma) - \mu sin(\gamma)) \leq 0$, então o valor encontrado para F_x não precisa ser corrigido, pois $F_{susp} > 0$.

Uma vez determinada a força de tração F_x admissível e a força da suspensão F_{susp}, a partir da deformação h e da taxa de deformação dh/dt da suspensão, calcula-se a força de tração F_n das rodas

$$F_n = \left(\frac{F_x - F_{susp} sin(\gamma)}{cos(\gamma)} \right) n \tag{3.23}$$

onde ***n*** é o vetor unitário que define a direção "para a frente" do referencial local do veículo. A força F_b (força que a suspensão exerce sobre o chassi) será o produto do módulo da força da suspensão F_{susp} por ***b*** (vetor unitário que define a direção "para cima" do referencial local do veículo), logo

$$F_b = F_{susp} \mathbf{b} \tag{3.24}$$

Para a força lateral/transversal (na direção de ***t***), foi implementado o modelo de atrito simples proporcional à força normal de maneira que $F_y = (\mu N)t$.

No entanto, esse modelo de força transversal deixa o sistema próximo a uma instabilidade numérica, gerando oscilações devido ao fato de F_y nesse modelo poder assumir apenas três valores, $-\mu N$, 0 ou μN, sem incluir os valores intermediários. Com isso, foi necessário usar um modelo mais sofisticado, descrito a seguir.

3.5.
Modelo da Força de Contato Pneu-Terreno

Como mencionado no item anterior, um modelo simplificado da força de contato pneu-terreno não é uma boa opção, e por isso foi implementado o modelo de interação intitulado "**Fórmula Mágica**" [53], que será detalhado a seguir.

3.5.1.
Derivas Longitudinal e Lateral

Deriva é um fenômeno frequentemente usado como deslizamento em tração ou frenagem. Esse deslizamento é calculado a partir da velocidade angular da roda e a da velocidade linear do seu centro de massa, como pode ser visto nas equações 3.25 e 3.26. No modelo implementado foi necessário encontrar as derivas longitudinal e lateral para o cálculo das forças de interação.

Quando um torque propulsor é aplicado a uma roda de forma que gire a uma determinada velocidade sem patinar, diz-se que a velocidade angular e a translação do centro de massa da roda são proporcionais. Entretanto, quando essas grandezas são desproporcionais, diz-se que está ocorrendo um deslizamento ou deriva, que pode ser tanto de tração quanto de frenagem.

Para a deriva longitudinal, foram implementadas duas equações, uma para tração e outra para frenagem, medidas percentualmente.

Deriva longitudinal para tração:

$$\left(\frac{\omega r - V}{\omega r}\right) \cdot 100 \tag{3.25}$$

onde ω é a velocidade angular, r é o raio e V é a velocidade linear do centro de massa da roda.

Deriva longitudinal para frenagem:

$$\left(\frac{V-\omega r}{V}\right)\cdot 100 \tag{3.26}$$

Para o cálculo da deriva lateral, usa-se a equação a seguir, em radianos:

$$arctan\left(\frac{V_x}{V_y}\right) \tag{3.27}$$

onde V_x é a componente longitudinal e V_y a componente lateral, ambas pertencentes à velocidade linear do centro de massa da roda.

3.5.2.

Cálculo das Forças Longitudinal e Lateral

O modelo utilizado para o cálculo das forças da interação pneu-terreno é um método semiempírico chamado de "**Fórmula Mágica**" ("***Magic Formula***", [53]). Pacejka [53] coloca que essa abordagem, por se tratar de uma curva ajustada de acordo com dados experimentais, é mais exato que outros métodos existentes. As equações envolvidas são descritas a seguir:

$$y = Dsin\left[Carctan\left\{Bx - E\left(Bx - arctanBx\right)\right\}\right] \tag{3.28}$$

$$Y(X) = y(x) + S_V \tag{3.29}$$

$$x = X + S_H \tag{3.30}$$

onde

Y – variável de saída F_x e F_y

X – variável de entrada $tan(\alpha)$ ou κ

e

B é o fator de rigidez;

C é o fator de forma;

D é o valor de pico;

E é o fator de curvatura;

S_H é o offset horizontal;

S_V é o offset vertical;

A fórmula mágica $y(x)$ produz uma curva que passa através da origem $x = y = 0$, obtendo um valor de pico x_m, onde y_a é o valor final para onde a assíntota converge. Na imagem à direita na Figura 31, vê-se como o argumento do $sin(\beta)$ varia de acordo com x e a distorção que isso causa no seno.

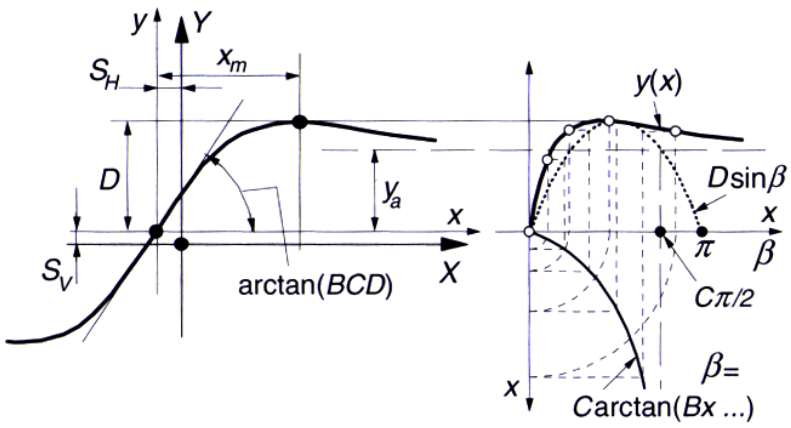


Figura 31 – Curva produzida pela versão original da **"Fórmula Mágica"** em [53]

Tabela 4 – Coeficientes dos pneus para a **"Fórmula Mágica"** em [54], valores em *kN*

Valores dos coeficientes dos pneus de um veículo para a "Fórmula Mágica"

	Carga, F_z, kN	B	C	D	E	S_h	S_v	BCD
F_y, N	2	0.244	1.50	1936	−0.132	−0.280	−118	780.6
	4	0.239	1.19	3650	−0.678	−0.049	−156	1038
	6	0.164	1.27	5237	−1.61	−0.126	−181	1091
	8	0.112	1.36	6677	−2.16	0.125	−240	1017
M_z, N · m	2	0.247	2.56	−15.53	−3.92	−0.464	−12.5	−9.820
	4	0.234	2.68	−48.56	−0.46	−0.082	−11.7	−30.45
	6	0.164	2.46	−112.5	−2.04	−0.125	−6.00	−45.39
	8	0.127	2.41	−191.3	−3.21	−0.009	−4.22	−58.55
F_x, N	2	0.178	1.55	2193	0.432	0.000	25.0	605.0
	4	0.171	1.69	4236	0.619	0.000	70.6	1224
	6	0.210	1.67	6090	0.686	0.000	80.1	2136
	8	0.214	1.78	7711	0.783	0.000	104	2937

Os valores da Tabela 4 foram obtidos a partir de um ajuste (*curve fitting*) de coeficientes com dados experimentais em função da força normal (***N***) aplicada sobre o pneu. Eles já incorporam o coeficiente de atrito entre pneu e um tipo de terreno não especificado pelo autor [54].

Os valores de B, C, D, E, S_V e S_H dependem da força normal F_z e do ângulo de cambagem γ_c e devem ser estimados a partir de curvas experimentais do pneu. Porém, na presente dissertação, γ será sempre zero, pois as rodas são perpendiculares ao chassi. A dependência dos coeficientes com relação à força normal foi modelada (mediante ajuste de curvas) de acordo com as equações mostradas a seguir.

Tabela 5 – Coeficientes a_1 até a_8 para o pneu do veículo

Valores dos coeficientes a_1 até a_8 para o pneu do veículo (Fz em kN)

	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8
F_y, N	−22.1	1011	1078	1.82	0.208	0.000	−0.354	0.707
M_z, N · m	−2.72	−2.28	−1.86	−2.73	0.110	−0.070	0.643	−4.04
F_x, N	−21.3	1144	49.6	226	0.069	−0.006	0.056	0.486

Para o cálculo do valor de pico D, usa-se

$$D = a_1 F_z^2 + a_2 F_z \tag{3.31}$$

onde F_z está em kN e a_1 e a_2 são coeficientes empíricos apresentados na Tabela 5. Lembrando que o termo D é usado tanto para a força transversal (F_x) como longitudinal (F_y).

Para BCD, que representa as rigidezes, são apresentadas duas equações, uma para a força transversal e outra para a longitudinal.

Para F_x tem-se

$$BCD = \frac{a_3 F_z^2 + a_4 F_z}{e^{a_5 F_z}} \tag{3.32}$$

e, para F_y,

$$BCD = a_3 sin\left[a_4 arctan\left(a_5 F_z \right) \right] \tag{3.33}$$

onde a_3, a_4 e a_5 são coeficientes empíricos que podem ser vistos na Tabela 5.

Para o termo E, o fator de curvatura, tem-se uma única equação tanto para a força longitudinal quanto para a transversal:

$$E = a_6 F_z^2 + a_7 F_z + a_8 \tag{3.34}$$

onde a_6, a_7 e a_8 são coeficientes empíricos (ver Tabela 5).

Para o termo C, que representa o fator de forma, praticamente independente de F_z, usam-se as constantes a seguir para F_y e F_x:

para F_y,

$$C = 1.19$$

e para F_x,

$$C = 1.69$$

O termo B, o fator de rigidez, tanto para a força longitudinal quanto lateral, é obtido por

$$B = \frac{BDC}{C \cdot D} \tag{3.35}$$

Neste trabalho, para os termos S_H e S_V, que representam o offset horizontal e vertical, respectivamente, foi fixado um valor constante zero tanto para F_x quanto para F_y:

$$S_H = S_V = 0 \tag{3.36}$$

3.5.3.

Combinação das Derivas

Em [53], Pacejka descreve uma combinação eficiente das derivas longitudinal e lateral quando ambas atuam simultaneamente. Esse método foi desenvolvido por Michelin e publicado por Bayle, Forissier e Lafon (1993).

A próxima equação atenua a força longitudinal:

$$G_{x\alpha} = \frac{cos\left(C_{x\alpha} arctan\left(B_{x\alpha} \cdot a_{S\alpha} - E_{x\alpha}\left(B_{x\alpha} \cdot a_{S\alpha} - arctan\left(B_{x\alpha} \cdot a_{S\alpha}\right)\right)\right)\right)}{cos\left(C_{x\alpha} arctan\left(B_{x\alpha} \cdot S_{H\alpha} - E_{x\alpha}\left(B_{x\alpha} \cdot S_{H\alpha} - arctan\left(B_{x\alpha} \cdot S_{H\alpha}\right)\right)\right)\right)} \tag{3.37}$$

enquanto a equação a seguir é responsável pela atenuação da força lateral:

$$G_{y\kappa} = \frac{cos\left(C_{y\kappa} arctan\left(B_{y\kappa} \cdot \kappa_S\right)\right)}{cos\left(C_{y\kappa} arctan\left(B_{y\kappa} \cdot S_{H\kappa}\right)\right)} \tag{3.38}$$

3.6.

Atuador

Os atuadores são dispositivos com a meta de converter energia elétrica, hidráulica ou pneumática, por exemplo, em energia mecânica, possibilitando, por meio dos sistemas de transmissão, o posicionamento e movimentação das partes móveis do robô. Os atuadores mais comuns em sistemas robóticos são os motores elétricos, que podem ser motores de corrente contínua (DC) com escova (*brushed*) ou sem escova (*brushless*), ou motores de corrente alternada (AC), normalmente usados em indústrias.

A presente dissertação fará uso apenas do motor **Magmotor S28-150**, um atuador motor de corrente contínua com escovas na modelagem do simulador VirtualBotz 3D. Para maiores detalhes de configurações de outras características de motores DC, vide o **Anexo J**.

Tabela 6 – Parâmetros do motor Magmotor [48] utilizado no robô VIVI

Magmotor S28-150	
Parâmetro	**Valor**
Tensão elétrica nominal (V)	24
Potência máxima de saída (W)	2,183
Massa (kg)	1,7
Potência / Peso (W/kg)	1,284
K_t (N·m/A)	0,03757
K_v (RPM/V)	254
R_{motor} (Ω)	0,064
I_{sem_carga} (A)	3,4

Para simular uma força de atrito nos eixos dos motores, foi implementada uma versão modificada do modelo de fricção de LuGre, originalmente proposto por Canudas de Wit *et al.* [61]. A esta versão é dado o nome de Aproximação Contínua do Modelo de Fricção de LuGre [55], na qual um termo descontínuo é aproximado por meio de uma função contínua. A força de atrito que age entre o movimento dos corpos é escrita da seguinte forma:

$$F_a = \sigma_0 z + \sigma_1 \dot{z} + \sigma_2 \dot{y} \tag{3.39}$$

Nela, têm-se os coeficientes de rigidez, amortecimento e viscosidade.

Em [55] é proposta a seguinte equação para $\dot{z}$:

$$\dot{z} = S_1(\dot{y})\dot{y} - \frac{S_2(\dot{y})}{f_s(\dot{y})} z \tag{3.40}$$

A próxima função descreve as características de viscosidade do atrito para velocidades constantes:

$$f_s(\dot{y}) = \frac{1}{\sigma_0}\left[F_c + (F_s - F_c) e^{-(y/\dot{y}s)^2} \right] \tag{3.41}$$

onde F_c é a força de atrito de Coulomb e F_s é a força de atrito estático. Definem-se também

$$S_0(\dot{y}) = \frac{2}{\pi} arctan(k_v \dot{y}) \tag{3.42}$$

$$S_1(\dot{y}) = (S_o(\dot{y}))^2 \tag{3.43}$$

$$S_2(\dot{y}) = m(\dot{y}) = \dot{y} S_0(\dot{y}) \tag{3.44}$$

onde k_v é uma constante positiva.

3.7. Método de Integração

A principal vantagem de implementar o simulador em C++ não diz respeito apenas à modularização do sistema e consequente manutenção, aspecto que se refere à facilidade de modificar um sistema a fim de corrigir defeitos, adequá-lo a novos requisitos ou a um ambiente novo, ou aumentar a sua suportabilidade, mas também permite a otimização do código a fim tirar proveito de recursos computacionais, como a programação de processos, permitindo assim a execução em tempo real com um passo temporal de integração da ordem de 100ns.

Em vista da complexidade e arbitrariedade das inclinações do terreno, não foi possível usar métodos implícitos de integração como o Newmark-Beta, devido ao fato de o simulador não ajustar uma função ao perfil do terreno e, com isso, não ser possível calcular gradientes de terreno que seriam necessários em métodos implícitos. O simulador usa método explícito e daí a necessidade de um passo de integração pequeno.

Devido ao esforço aplicado na otimização do código para diminuir o passo de integração, o simulador hoje contempla a possibilidade de o veículo robótico ter quantas rodas o usuário definir, em qualquer lugar do chassi.

No próximo capítulo, os algoritmos de controle para o sistema aqui modelado são apresentados.

4
Controle

Para implementar o controle, é necessário um joystick analógico. Ele permite enviar comandos de alto nível, informando a direção e velocidades desejadas a partir da movimentação do manete (*pad*) (**Anexo C**). O controle recebe os dados e calcula o torque de cada roda.

Este trabalho implementou o algoritmo de controle dinâmico para terrenos acidentados (CDTA) de estabilidade 2D proposto por Silva [56] para terrenos acidentados, que visa manter a tração nas rodas sempre inferior ao limite de atrito dado por μN, as forças normais sob cada roda sempre positivas (tentando mantê-las sempre em contato com o terreno), não saturar os motores e minimizar a potência dissipada neles.

Para tal, a força de tração da roda *i* (F_{xi}) deverá atender aos seguintes critérios:

1. $\left|F_{x_i}\right| < F_{sat_i}$
2. $\left|F_{x_i}\right| < F_{n_i}$
3. $\left|F_{x_i}\right| < F_{t\max_i}$

onde $F_{sat_i} = T_{sat} \cdot r_i$, onde T_{sat} é o limite de saturação do motor e r_i é o raio da roda; F_{ni} é o limite sobre o qual a força normal continuará positiva (como visualizado na próxima equação) e $F_{t\max_i}$ é a força máxima de tração proposta pelo autor como μN.

$$F_i < \frac{F_{susp_i}}{sin\left(\gamma_i - \alpha\right)} \tag{4.1}$$

No entanto, houve a necessidade de usar o coeficiente *D*, da Equação (3.31), que corresponde o valor máximo da força de tração utilizada na "**Fórmula Mágica**", pois no simulador desenvolvido nesta dissertação o modelo de tração implementado é mais realista. Com isso, $F_{t\max_i} = D$ da Equação (3.31), onde os coeficientes a_1 e a_2 são respectivamente -21,3 e 1144, como mostra a Tabela 5. Essas restrições e condições formam uma região *D* admissível de forças de tração para as rodas, como mostra a Figura 32.

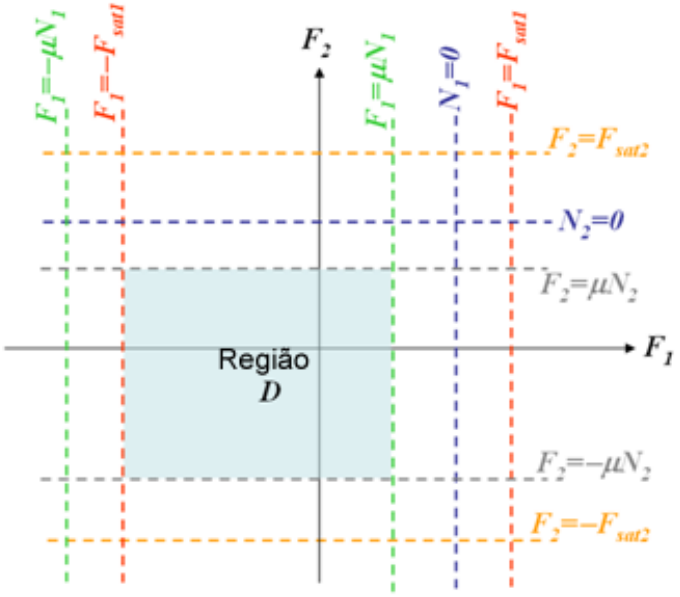


Figura 32 – Região admissível para as forças de tração nas rodas 1 e 2 [56]

A dinâmica longitudinal do veículo foi modelada conforme [56]:

$$a_1 = n_1 \cdot F_1 + n_2 \cdot F_2 + a_0 \tag{4.2}$$

onde a_1 é aceleração do veículo, F_i é a força escalar de tração da roda *i*, n_i é um fator que depende da inclinação do veículo (α), do ângulo de contato da roda *i* com o terreno (γ_i) e da massa do veículo (m), e a_0 é a aceleração devida à inclinação do terreno e depende de α, de γ_i, da força F_{susp_i} da suspensão e de m.

Como a aceleração do veículo é proporcional às forças de tração das rodas, o controle de velocidade pode ser definido como um controlador proporcional. Ou seja,

$$U := Kp\left(Vd - Vl\right) \tag{4.3}$$

onde Kp é o ganho proporcional, Vd é a velocidade desejada, Vl é a velocidade lida e U é o sinal de controle.

Com isso, combinando as equações (4.2) e (4.3),

$$n_1 \cdot F_1 + n_2 \cdot F_2 + a_0 = Kp(Vd - Vl) \tag{4.4}$$

A equação fornece uma relação linear entre as forças, gerando mais uma restrição para as combinações das forças. Esse conjunto de combinações denominado Γ pode ser visto na Figura 33.

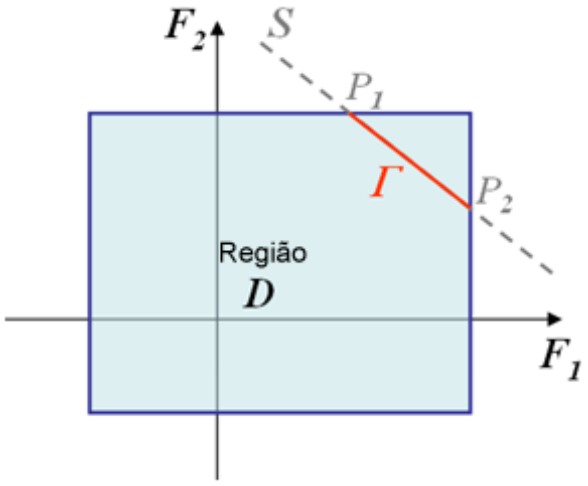


Figura 33 – Combinações da região *D* e Γ [56]

Geralmente, a interseção entre os conjuntos D e Γ provê infinitas combinações para as forças nas rodas traseiras e dianteiras. Nesses casos, a escolha das forças é feita mediante a minimização da potência dissipada nos motores.

A potência total, por sua vez, foi modelada em [56] como:

$$Pt = \nabla_1 \cdot \left|F_1\right| + \nabla_2 \cdot \left|F_2\right| \tag{4.5}$$

onde ∇_i depende da velocidade desejada (Vd), da aceleração angular do veículo ($\dot{\alpha}$), da distância vertical entre o centro de massa da roda e o centro de massa do veículo, e de α e γ_i.

Essa restrição de potência gera regiões de combinações de forças onde a potência é a mesma, em forma de um diamante (Figura 34). A minimização da potência ocorre ao se chegar à menor região que intersecta Γ, como mostra a Figura 34.

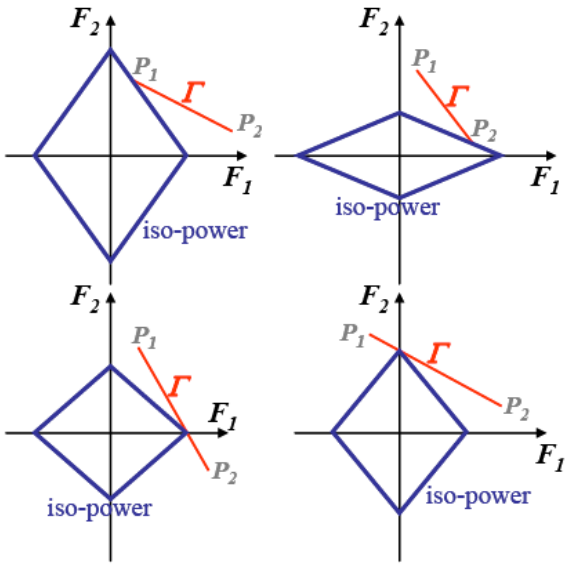


Figura 34 – Pontos de interseção entre Γ e o menor diamante das linhas de mesma potência para os quatro casos básicos

O algoritmo de controle consiste então de calcular as potências nos pontos P_1 e P_2 e as interseções do conjunto Γ com os eixos, e de escolher a combinação com a menor potência.

A abordagem utilizada para o cálculo de estabilidade é o método desenvolvido em [52] para medir a estabilidade veicular, chamado de momento de estabilidade *SM* (*stability moment*) [50 e 51]. Esse sistema é um cálculo da influência das componentes das forças de contato entre roda e terreno agindo sobre um veículo [52].

Em princípio, pode-se estimar a força de contato entre roda e terreno, necessária para calcular o *SM*, por meio de sensores de força embarcados na roda. Entretanto, esses sensores têm um alto custo e normalmente não estão presentes em veículos de passageiros. Para evitar o uso deste tipo de sensor, Peters e Iagnemma [52] propuseram uma alternativa chamada Cálculo Indireto do

Momento de Estabilidade. Nesse cálculo, a expressão é montada com os termos inerciais que podem ser medidos na prática com sensores de baixo custo.

$$\left[\sum_{i=1}^{l} p_b^i \times F_b^i\right] \cdot r_b^j = \left[\sum_{i=1}^{l+1} R_b^i\left(\omega_i \times I_i \varpi_i + I_i \dot{\varpi}_i\right) + \sum_{i=1}^{l+1} c_b^i \times m_i a_b^i - \sum_{i=1}^{k} q_b^i \times B_b^i\right] \cdot r_b^j \quad (4.6)$$

A expressão na direita da igualdade na Equação (4.6) é o método indireto com o somatório de momentos usado no aplicativo VirtualBotz 3D, e a expressão da esquerda é o método direto, dependente de sensores de alto custo.

O método *SM* indireto requer o conhecimento ou medidas dos seguintes valores:

(1) Velocidade angular, aceleração angular e aceleração linear do chassi do veículo e rodas, expressas em um eixo de coordenada de um corpo fixo.
(2) Conhecimento do centro de gravidade do chassi e rodas relativos ao ponto de contato entre roda e terreno.
(3) Conhecimento da magnitude e direção de qualquer tipo de força não desprezível em *B*. Essas forças podem incluir arrasto aerodinâmico ou colisões com outros corpos.
(4) O termo *r* diz respeito aos eixos de estabilidade.

A Figura 35 mostra os eixos de estabilidade utilizados no sistema, levando em conta que eles devem formar um polígono convexo. Os eixos representados por r^j são linhas adjacentes conectadas aos pontos de contato entre rodas e terreno.

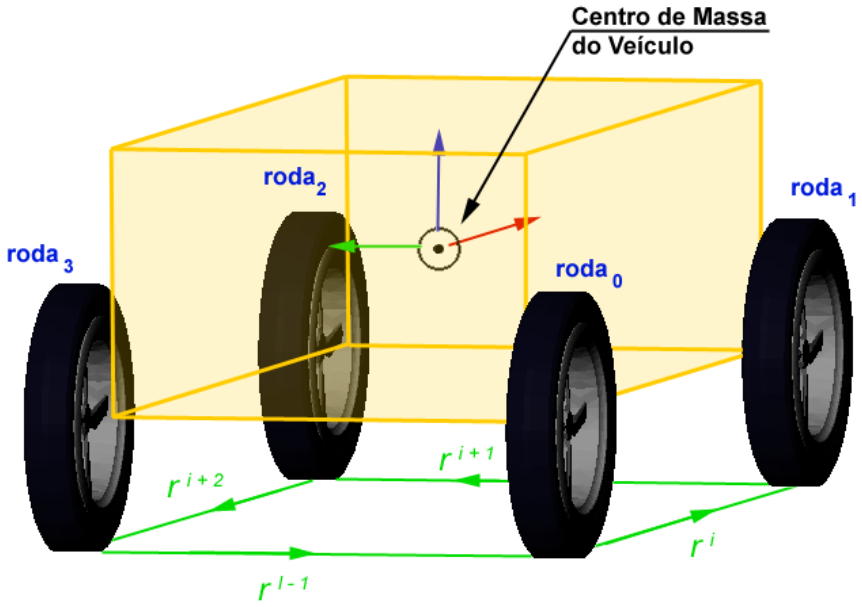


Figura 35 – Sistema de cálculo dos eixos de estabilidade

Cálculo de r^i :

$$r^i = \frac{p^{i+1} - p^i}{\left\| p^{i+1} - p^i \right\|}, \qquad \text{onde} \quad i \in \{1 \ldots, l-1\} \tag{4.7}$$

$$r^l = \frac{p^1 - p^l}{\left\| p^1 - p^l \right\|}, \qquad i = l \tag{4.8}$$

A força de contato entre roda e terreno causa um momento sobre todos os eixos que não deve ultrapassar a linha de força da ação. A magnitude desse momento é empregada como base proposta ao cálculo de estabilidade do veículo, o termo *SM*.

Para melhor visualização das forças agindo sobre o veículo robótico, foi implementado no aplicativo VirtualBotz 3D um gráfico simples em 2D apenas para mostrar o *SM* durante uma simulação.

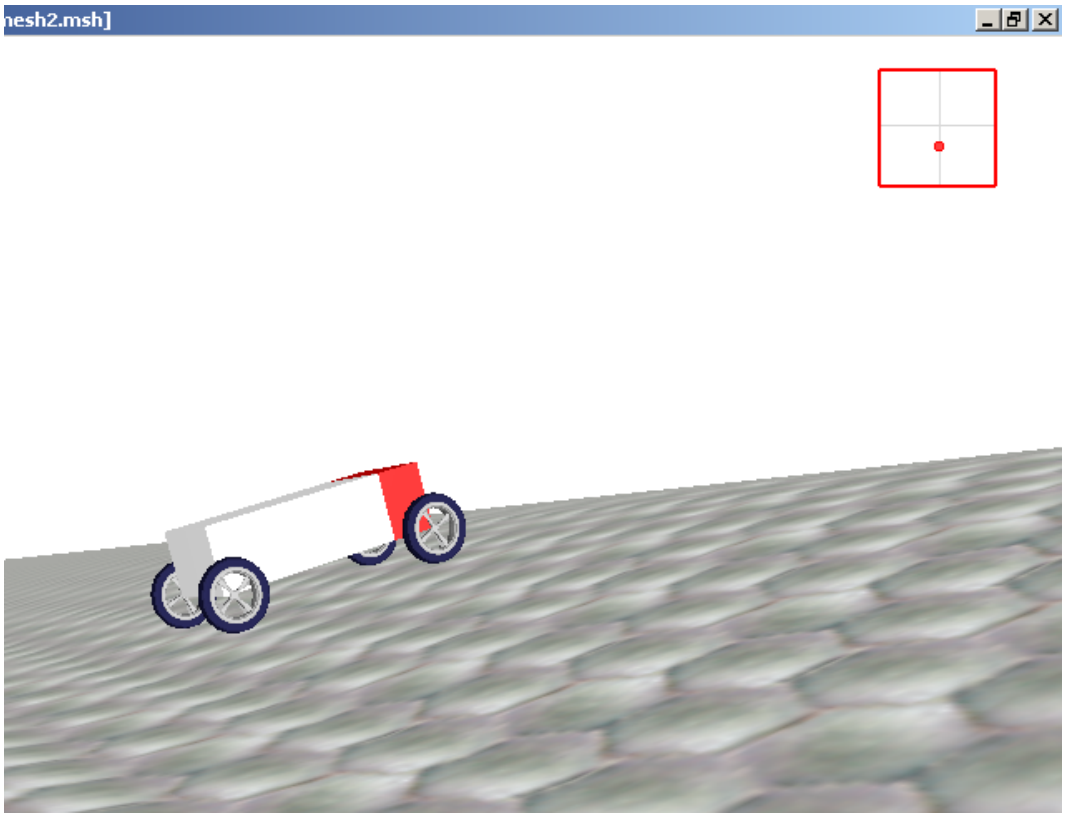


Figura 36 – Gráfico 2D do momento de estabilidade do veículo robótico na tela de visualização do aplicativo VirtualBotz 3D

Para que o veículo se encontre estável, com as quatro rodas em contato com o terreno, é preciso que a medida de estabilidade *SM* seja maior que -1 e menor do que 1, como mostrado abaixo.

$$-1 < SM_{long} < 1 \quad \text{e} \quad -1 < SM_{trans} < 1 \tag{4.9}$$

onde, para *SM*, tem-se a direção longitudinal e transversal, ou seja,

$$SM_{long} = \frac{SM_F - SM_B}{SM_F + SM_B} \quad \text{e} \quad SM_{trans} = \frac{SM_L - SM_R}{SM_L + SM_R} \tag{4.10}$$

No próximo capítulo, a validação do software, desenvolvido no presente estudo, é apresentada.

5
Validação do Software

Para garantir que os resultados deste trabalho sejam confiáveis, é preciso validar o simulador quanto às leis da física. Para tal, este capítulo apresenta dois casos onde há soluções analíticas.

No primeiro caso têm-se os resultados da simulação de uma **arrancada**. No segundo, a simulação de um **salto**. Para ambas as simulações são apresentadas soluções analíticas de modo a comparar os resultados. Foi utilizado um incremento no tempo de simulação (Δt) de 0,65 ms.

Os dados do veículo robótico usado em ambos os casos são apresentados na tabela a seguir:

Tabela 7 – Parâmetros do veículo para solução analítica

Parâmetro	**Valor**
Massa (kg)	50
Comprimento (cm)	64
Largura (cm)	22
Altura (cm)	10
Raio da roda (cm)	17
Rigidez (N/m)	10^4
Amortecimento (Ns/m)	500
Distância longitudinal entre eixos (cm)	36
Distância entre as rodas do mesmo eixo (cm)	32

5.1.
Validação da Simulação da Arrancada

O primeiro teste proposto para comparar os resultados do simulador com resultados analíticos é o da arrancada do veículo (Figura 37). Ele consiste em

acelerar o veículo mediante torques constantes nas rodas (no caso, só nas rodas traseiras) e monitorar as forças normais sobre as rodas e a inclinação do chassi em relação ao eixo ***t***.

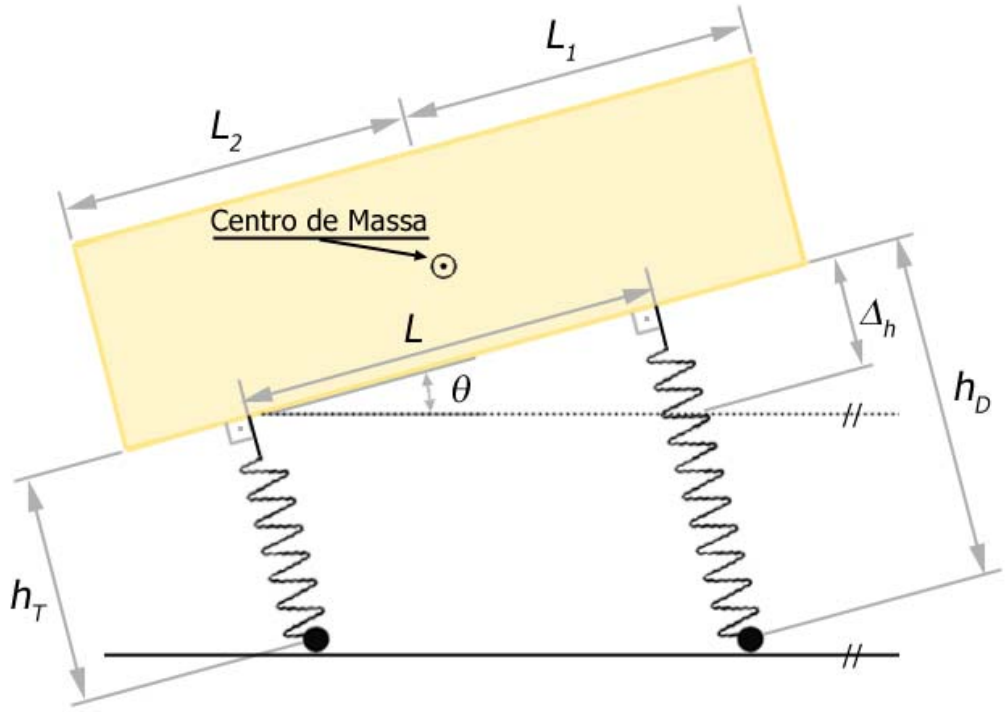


Figura 37 – Definição das grandezas utilizadas no cálculo da solução analítica para o teste de arrancada do veículo robótico

$$\Delta h = h_D - h_T \tag{5.1}$$

mas

$$h_D = F_{N_D} / k \tag{5.2}$$

$$h_T = F_{N_T} / k \tag{5.3}$$

$$\theta = tg^{-1}\left(\frac{\Delta h}{L}\right) \tag{5.4}$$

onde h_D é o deslocamento da suspensão dianteira, h_T é o deslocamento da suspensão traseira, F_{N_D} é força normal sobre a roda dianteira, F_{N_T} é a força normal sobre a roda traseira, k é a rigidez da suspensão e θ é o ângulo de guinada.

Segundo [62], as forças normais agindo sobre as rodas de um veículo de quatro rodas simétrico em relação ao plano *XZ* podem ser obtidas da seguinte forma:

$$F_{N_D} = \frac{PL_1}{2L} - \frac{Ph\ddot{X}}{2Lg} \tag{5.5}$$

$$F_{N_T} = \frac{PL_1}{2L} + \frac{Ph\ddot{X}}{2Lg} \tag{5.6}$$

onde F_{N_D} são as forças normais nas rodas dianteiras (numeradas como 0 e 3), F_{N_T} são as forças normais nas rodas traseiras (numeradas como 1 e 2), ***P*** é a força peso do veículo robótico, *L* é a distância entre os eixos dianteiro e traseiro, L_1 e L_2 são, respectivamente, as distâncias do centro de massa do veículo a sua extremidade da dianteira e traseira, *h* é a altura do centro de massa do veículo até ao terreno, *g* é a constante da gravidade e $\ddot{X}$, a aceleração do centro de massa do veículo.

O teste foi feito sob quatro condições diferentes nas simulações. Primeiro, foi imposta uma aceleração de 0,5739 m/s^2 e a circunferência das rodas foi discretizada, para calcular o ponto de contato entre roda e terreno, em 20 divisões. O resultado analítico encontrado nessas condições é de 116,3382 N para a força normal sob a suspensão dianteira e 128,9118 N para a traseira. Na simulação, esses valores foram de 115,9463 N para a dianteira e a traseira foi de 129,2491 N, o que representa um erro percentual em relação ao teórico de 0,34% para a suspensão dianteira e 0,26% para a traseira.

Para o ângulo de guinada do veículo (θ), a Equação (5.4) resulta em um valor de 0,2117º, utilizando os valores das forças normais extraído da simulação. Esse valor encontrado é o mesmo da simulação.

O erro nas forças normais encontradas se deve à discretização da roda para a procura do ponto de contato com o terreno, pois uma pequena variação desse ângulo de contato (γ) resulta em um erro no somatório de forças, tendo em vista que a força normal terá uma inclinação diferente da real. Quanto maior for essa diferença, maior será o erro encontrado.

Em um segundo teste, explorou-se a possibilidade de aumentar a discretização das rodas para 2.000 divisões. Essa alteração permitiu uma melhora para 0,03% no erro da suspensão dianteira e 0,02% para a traseira. Para o ângulo de guinada o erro continuou nulo.

No entanto, para essa melhoria o tempo médio de cálculo da iteração passou de 0,64 ms para 3,63 ms, representando um aumento de 467% do tempo de processamento da iteração no simulador. Tendo em vista que essa melhoria é muito cara computacionalmente, para as outras simulações deste trabalho serão utilizadas apenas 20 divisões nas rodas.

Outros dois testes de arrancadas foram feitos com acelerações maiores. No primeiro, foi imposta uma aceleração de 1,15 m/s^2, que gerou um erro na força normal sobre a suspensão dianteira de 0,70% e um erro de 0,41% na traseira. No segundo teste, com uma aceleração de 1,91 m/s^2, foi encontrado um erro para a força normal sobre a suspensão dianteira de 1,23% e de 0,46% na traseira.

Nas duas últimas simulações, os ângulos de guinada (θ) encontrados foram de respectivamente 0,4212º e 0,697º e coincidem com os ângulos encontrados no modo analítico.

Para terrenos planos e horizontais, a variação do ângulo teta é igual à variação do ângulo de contato gama, pois (para esse veículo) as suspensões estão fixas e sempre perpendiculares ao chassi. Com isso, esses ângulos aumentam para acelerações maiores. Com uma discretização de 20 divisões para o semicírculo representativo da roda (para o cálculo do ponto de contato roda-terreno), tem-se que os possíveis pontos de contato estão a cada 9º, onde o ângulo 0º foi convencionado ser a posição de repouso do veículo em terreno plano horizontal. Portanto, conforme o ângulo teta se distancia de 0º, mas permanece menor que 4,5º (metade da resolução), o erro das forças normais aumenta, pois o ângulo do ponto de contato real gama (que deveria ser igual a teta) está se distanciando do calculado (0º), como foi dito anteriormente.

Um último teste de arrancada foi feito para ilustrar o problema descrito acima. Aumentou-se a discretização para 258 divisões, de maneira que a resolução para a procura do ponto de contato entre a roda e o terreno fosse de 0,6977º, muito próximo ao ângulo de guinada do veículo na última simulação. Ao manter a mesma aceleração imposta ao veículo, o erro encontrado para a força sobre a suspensão dianteira caiu para 0,09% e para 0,07% para a suspensão traseira. Esses

valores são próximos aos encontrados para a simulação com 2.000 divisões, apesar de serem utilizadas apenas 258 divisões.

Com isso, mostra-se que os erros encontrados para os valores das forças normais dianteiras e traseiras não dependem das condições dinâmicas ou cinemáticas do sistema, e sim do nível de detalhamento adotado para a representação geométrica do veículo robótico simulado.

5.2.
Validação da Simulação do Salto

O teste de simulação de **salto** (Figura 38) a fim de validar os resultados do simulador, consiste em monitorar a distância percorrida do veículo imediatamente após um salto a partir de uma rampa. Utilizando o recurso de configuração das condições iniciais do sistema por intermédio de scripts (**Anexo E** e **I**), foram configuradas, para as simulações, condições inicias que retratassem o estado do sistema logo após o salto por uma rampa ao invés de simular o veículo passando, efetivamente, por ela.

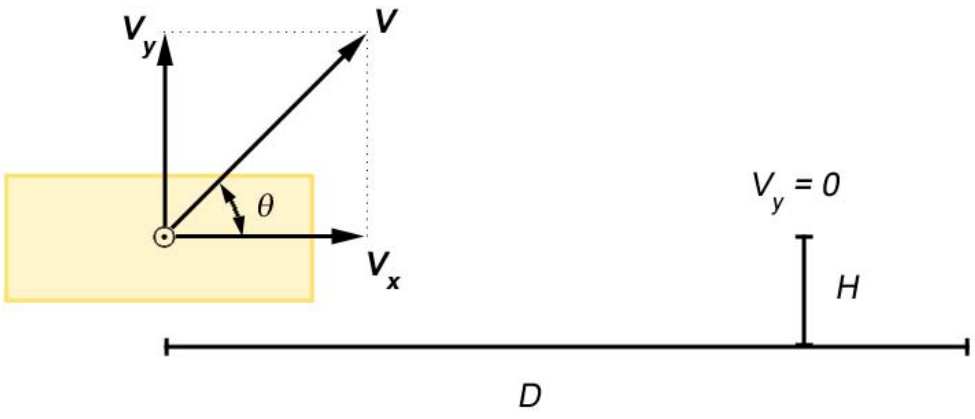


Figura 38 – Definição das grandezas utilizadas no cálculo da solução analítica para o teste de salto do veículo robótico

Como não foram consideradas forças aerodinâmicas no sistema durante o salto do veículo, a única componente da velocidade que é alterada é a vertical, devido à influência da gravidade. Portanto, para determinar a distância horizontal total percorrida pelo veículo durante o salto, é preciso calcular o tempo necessário para que o veículo pare de subir (a componente vertical da velocidade seja nula) e depois calcular o tempo até que ele volte para o terreno.

Sabendo-se que

$$S = S_0 + V_0 t + \frac{1}{2} a t^2 \tag{5.7}$$

e que

$$V = V_0 + at \tag{5.8}$$

tem-se que

$$O = V_y - g t_1 \therefore t_1 = \frac{V_y}{g} \tag{5.9}$$

onde V_y é a componente vertical da velocidade de saída da rampa do veículo robótico e t_1 é o tempo necessário para que o veículo chegue ao ponto mais alto da trajetória. Com isso, calcula-se a atura deste ponto, como mostra a Equação (5.10).

$$M = h + V_y t_1 - \frac{1}{2} g t_1^2 \tag{5.10}$$

onde M é altura máxima atingida pelo veículo e h é a sua altura inicial (assumida 0, para os cálculos). Em seguida, pode-se calcular o tempo de queda do veículo até o terreno como mostra a Equação (5.11).

$$M = \frac{1}{2} g t_2^2 \setminus t_2 = \sqrt{\frac{2M}{g}} \tag{5.11}$$

As Equações (5.12) e (5.13) mostram como determinar a distância horizontal total percorrida pelo veículo durante o salto, uma vez que, como as rodas não estão em contato com o terreno, não há aceleração nessa direção. Com isso, a distância total será

$$T = t_1 + t_2 \tag{5.12}$$

$$D = V_x T \tag{5.13}$$

onde D é a distância horizontal percorrida, T é o tempo total durante a trajetória e V_x é a componente horizontal da velocidade do veículo.

O veículo robótico inicia o salto em x = 0,0 e com velocidade em módulo de V = 15m/s. Foram feitos três testes, com variações apenas do ângulo inicial da velocidade.

No primeiro teste, o ângulo θ é de 22,5º e ele toca o terreno, resultando em uma distância horizontal total de salto de 16,214 m. No resultado analítico obtido nessas mesmas condições, a distância é de 16,218 m o que representa um erro de 0,02%.

No segundo teste, a simulação foi inicializada com um ângulo θ de 35º. Ele toca o terreno a uma distância de 21,548 m. O resultado analítico obtido nessas mesmas condições é de 21,553 m o que representa um erro de 0,02%.

No terceiro teste, a simulação foi inicializada com um ângulo θ de 45º. Ele toca o terreno a uma distância de 22,930 m. O resultado analítico obtido nessas mesmas condições é de 22,936 m, o que representa um erro de 0,02%.

Esses resultados mostram a boa concordância entre os valores encontrados no simulador e os resultados analíticos correspondentes.

No próximo capítulo, os resultados de simulações deste estudo são apresentados.

6
Resultados

Foram feitas simulações comparando o controle proposto por [56] com o controle proporcional com compensador de gravidade. Para tal, o veículo robótico, com as especificações da VIVI (Tabela 8), foi simulado em um **terreno senoidal** e em um **terreno plano com uma rampa**.

Tabela 8 – Especificações do veículo robótico VIVI

VIVI	
Parâmetro	**Valor**
Massa (kg)	50
Comprimento (cm)	64
Largura (cm)	22
Altura (cm)	10
Raio da roda (cm)	17

Segue a equação das ondulações do terreno senoidal utilizado na primeira simulação.

$$0.2 \cdot (1.0 + cos(y \cdot pi / 22.0) \cdot 1.0) \tag{6.1}$$

A Figura 39, desenhada no MATLAB, mostra o perfil da ondulação do terreno senoidal.

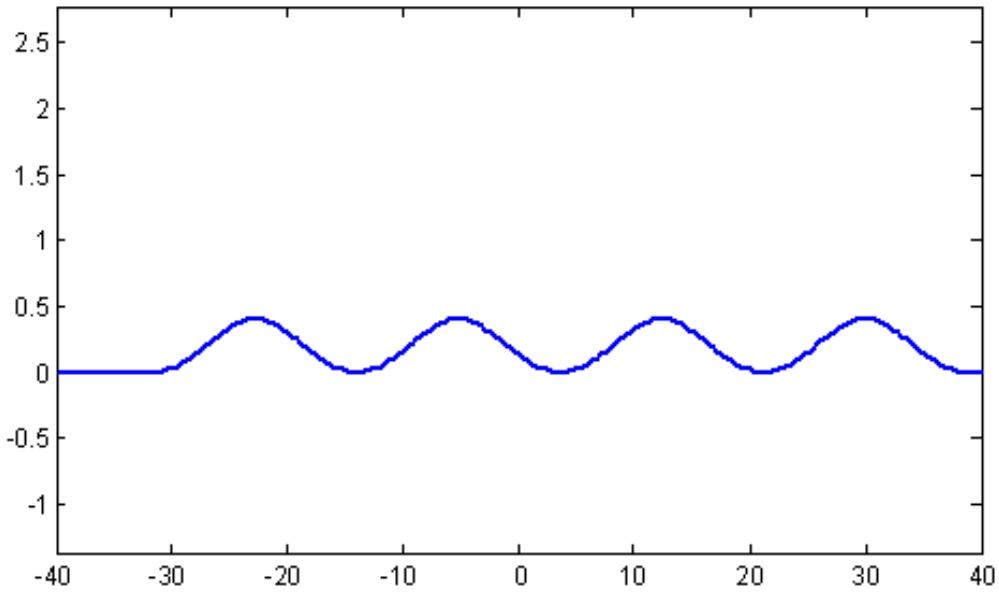


Figura 39 – Perfil do terreno senoidal desenhado no MATLAB

As Figuras 40 e 41 mostram o cenário da simulação do terreno senoidal renderizado pelo simulador VirtualBotz 3D. A Figura 40 apresenta o início da simulação, enquanto a Figura 41 mostra o veículo robótico perto do aclive a ser superado.

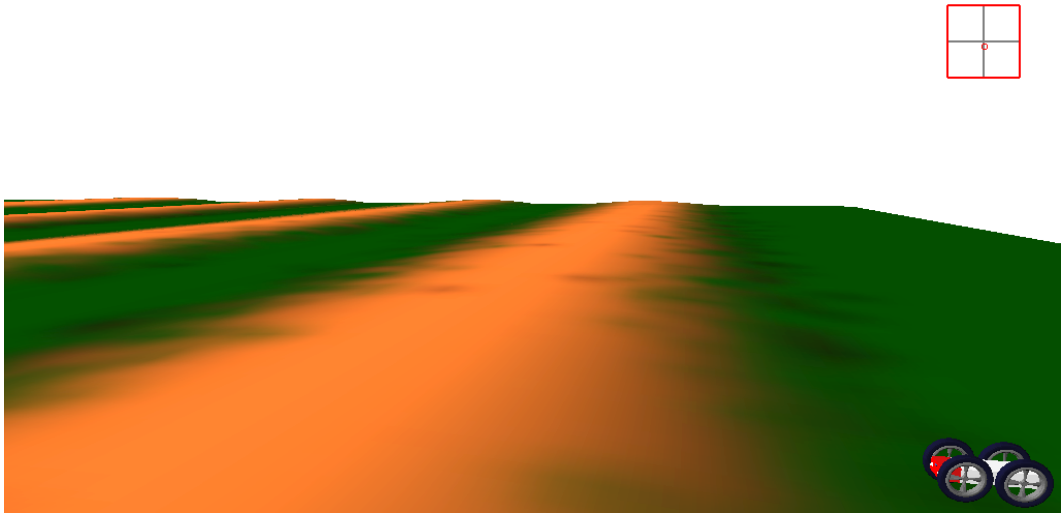


Figura 40 – Início da simulação: cenário da primeira simulação, com o veículo robótico VIVI, no terreno senoidal

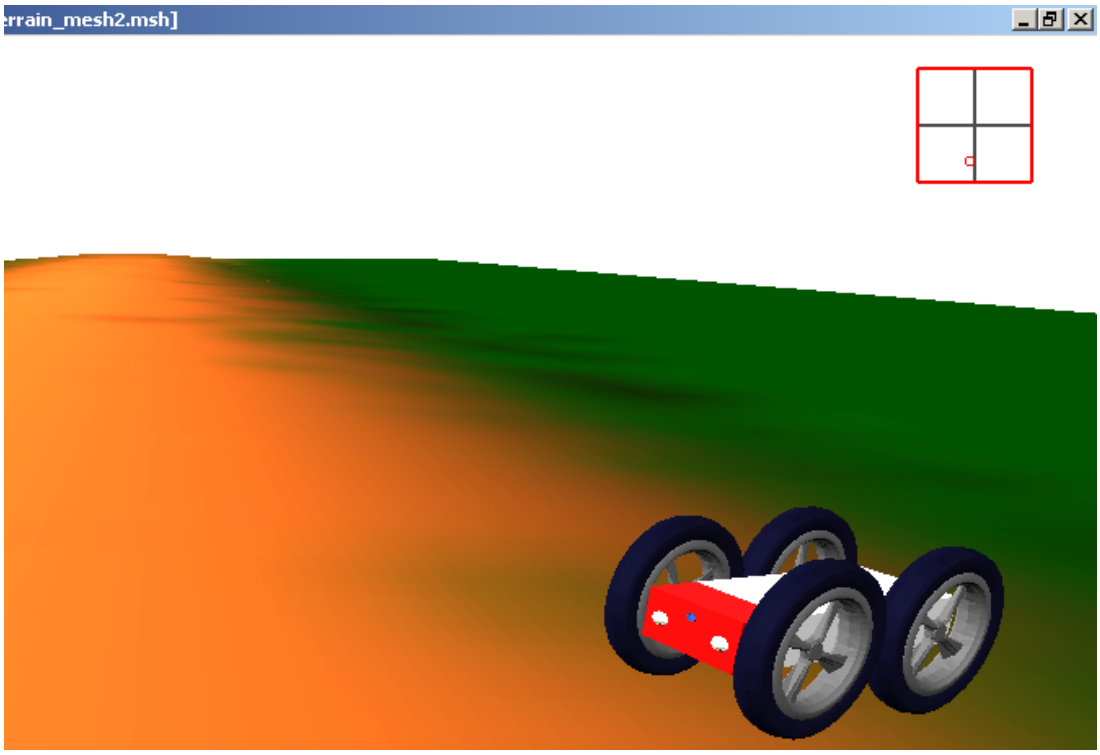


Figura 41 – Veículo robótico VIVI no terreno senoidal superando o primeiro aclive

Na segunda simulação, com terreno plano com uma rampa, tem-se a seguinte equação da rampa:

$$0.4 \cdot (1.0 + sin(y \cdot pi / 8.0)) \quad (6.2)$$

onde y pertence ao intervalo de 16m a 22m.

O perfil da rampa foi desenhado também no MATLAB e é apresentado na Figura 42.

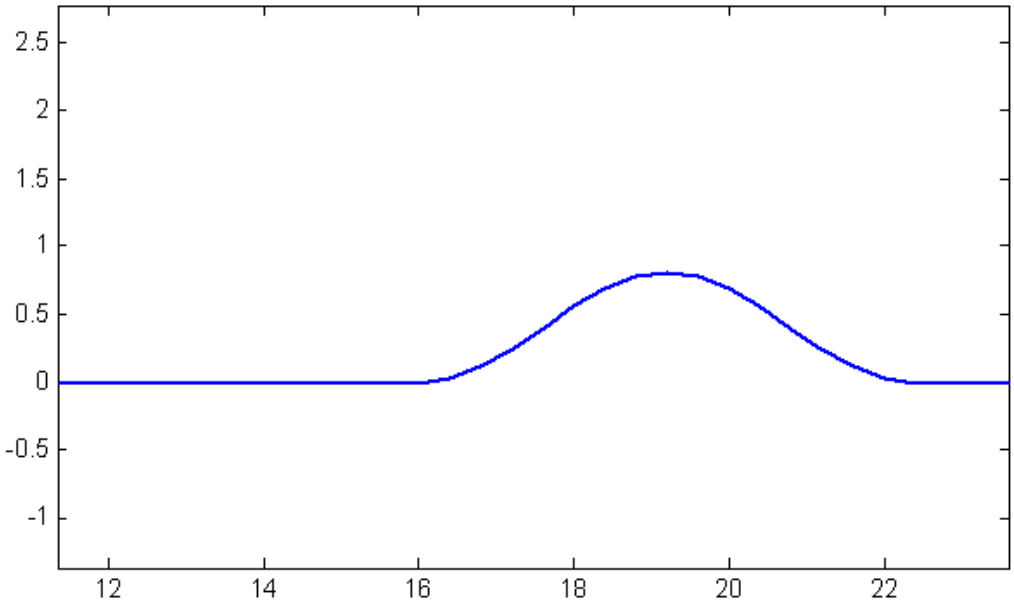


Figura 42 – Perfil da rampa do terreno desenhado no MATLAB

As Figuras 43 e 44 ilustram o cenário da simulação do terreno plano com uma rampa renderizado pelo simulador VirtualBotz 3D, desenvolvido ao longo deste estudo. Na Figura 43 tem-se o início da simulação, enquanto a Figura 44 mostra o veículo robótico perto da rampa a ser superada.

Figura 43 – Cenário da segunda simulação com o veículo robótico VIVI, no terreno plano com uma rampa ao final

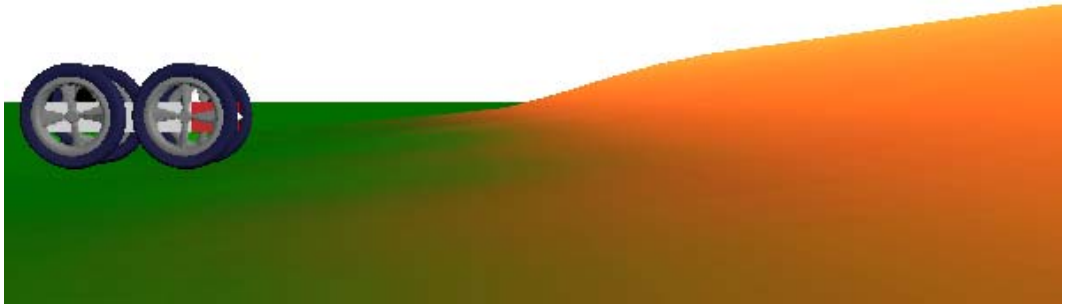


Figura 44 – Veículo robótico VIVI no início da rampa na segunda simulação, no terreno plano com uma rampa

A velocidade desejada foi de 10m/s para ambos os controles implementados, e o ganho proporcional utilizado foi de 100. A simulação foi feita com um passo de integração dT = 0.000065s, e o controlador foi executado a uma frequência de 10kHz. Foi implementada virtualmente uma bateria elétrica capaz de fornecer de 36V a 80A. Para simular a dinâmica apenas em 2D, um mesmo torque foi enviado para ambas as rodas traseiras do robô, e outro torque, para ambas as rodas dianteiras, evitando assim que o robô executasse curvas. O mesmo ganho proporcional foi utilizado em ambos os controladores.

6.1.

Controle Proporcional Simples em Terreno Senoidal

As figuras a seguir demonstram os resultados para a simulação feita com o controlador proporcional simples e compensador de gravidade no terreno senoidal.

A Figura 45 mostra que o controlador proporcional não foi capaz de compensar o atrito, tendo se afastado do *setpoint* ao final da simulação.

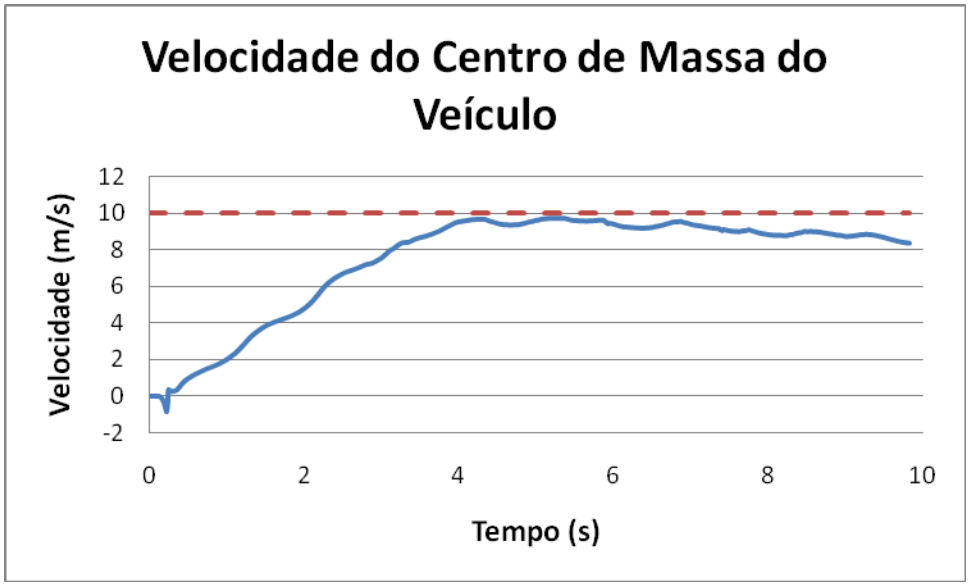


Figura 45 – Velocidade do centro de massa do veículo com o controlador proporcional simples com compensação de gravidade

A Figura 46 apresenta a variação das normais sobre as rodas dianteira e traseira. Pela simplicidade deste controle, as forças normais não são monitoradas e, com isso, em vários momentos da simulação, a roda dianteira descolou do terreno (força normal nula). Esse comportamento não é desejável, pois em uma situação crítica, a estabilidade do veículo pode ser comprometida e, além disso, o veículo passa a ter menor força de tração máxima possível pela redução das forças normais.

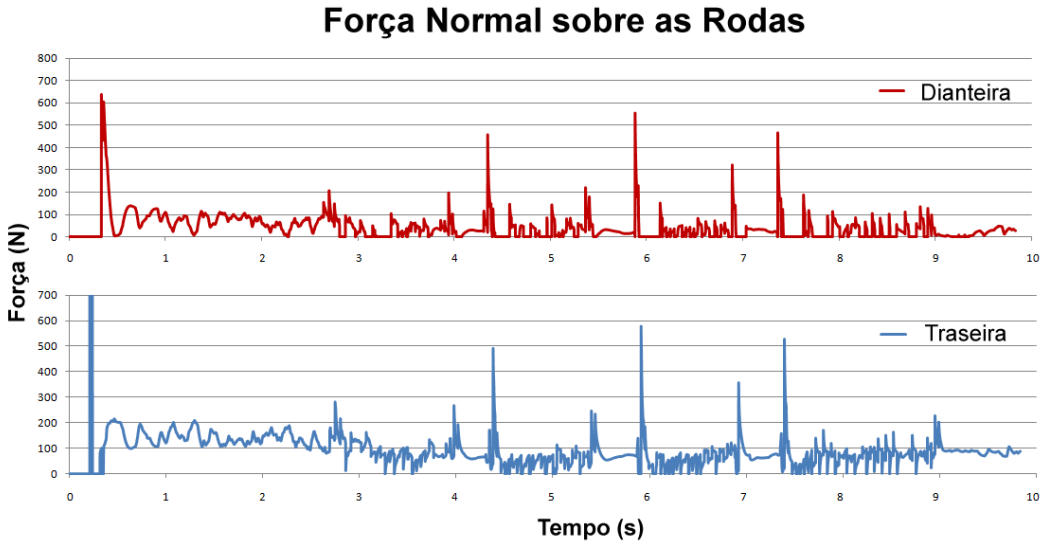


Figura 46 – Força normal com o controlador proporcional simples com compensação de gravidade

A Figura 47 mostra como a deriva longitudinal foi alta durante a simulação, evidenciando que as rodas deslizaram mais que o devido durante o percurso. Essa derrapagem excessiva fez com que as rodas trabalhassem com uma força de tração menor que a possível. Também indica como a deriva longitudinal oscilou, justificando o comportamento oscilatório do centro de massa do veículo.

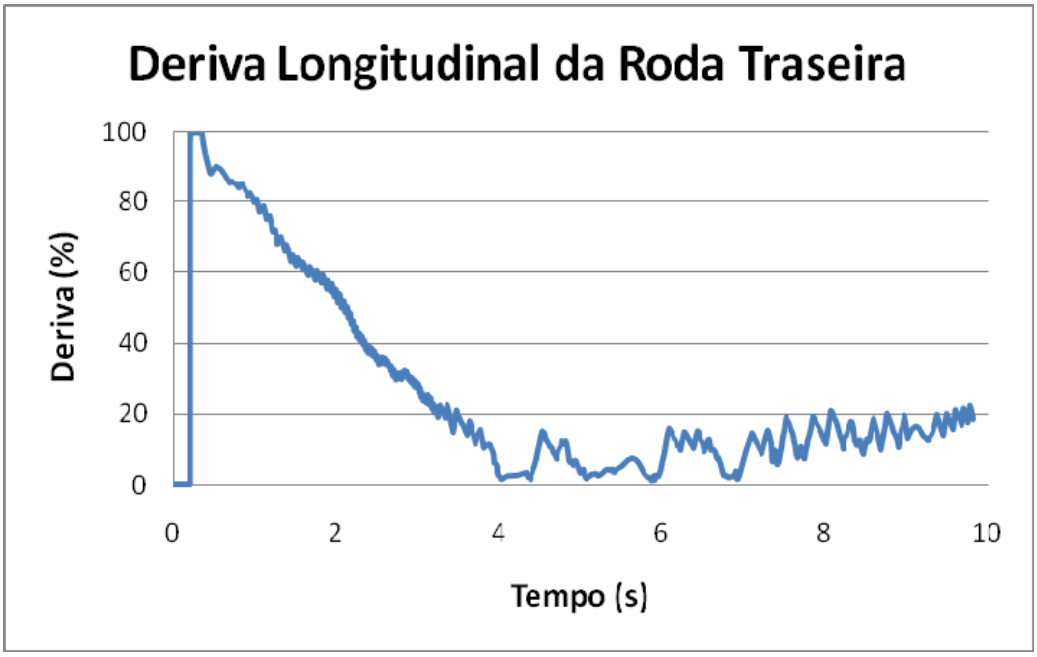


Figura 47 – Gráfico da deriva longitudinal da roda traseira com o controlador proporcional simples com compensação de gravidade

Outro problema identificável pelo desempenho desse controlador é que a potência dissipada nos motores é muito alta, como pode ser visto na Figura 48. Esse fator acaba comprometendo a autonomia do veículo robótico, pois a energia consumida pelas baterias é maior do que a necessária.

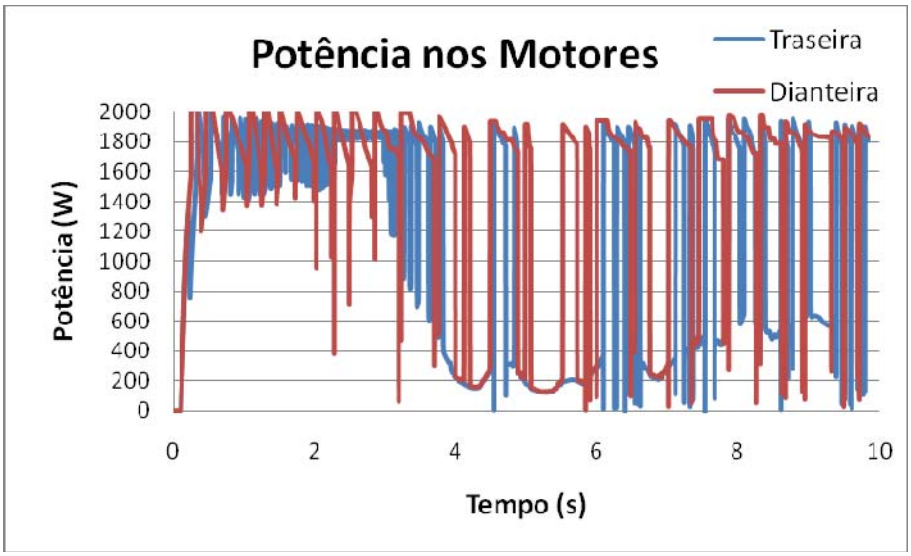


Figura 48 – Potência dos motores com o controlador proporcional simples com compensação de gravidade

6.2.

Controle CDTA em Terreno Senoidal

Essa simulação foi realizada sob as mesmas condições da anterior e com o mesmo tipo de terreno senoidal. O controle permitiu um valor maior de tração nas rodas, possibilitando uma convergência mais rápida da velocidade. Além disso, o controle de tração e a preocupação de manter a força normal positiva o máximo possível fizeram com que o veículo mantivesse a velocidade desejada até o final da simulação, como visto na Figura 49.

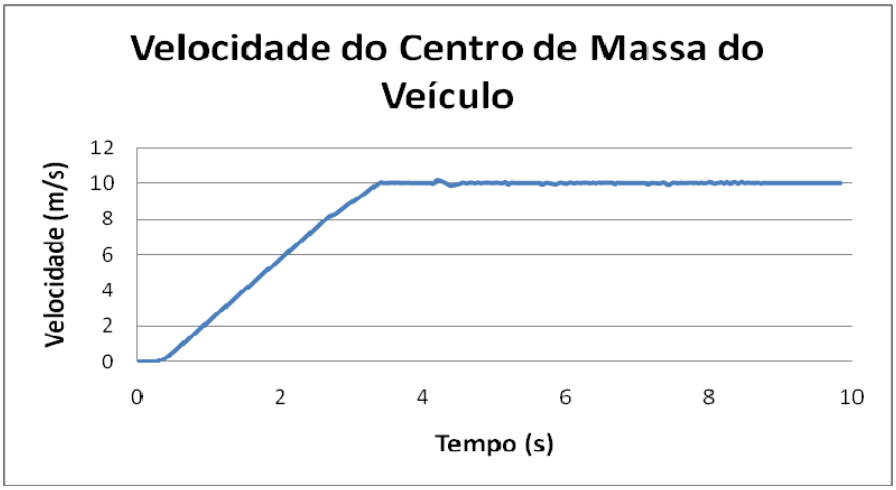


Figura 49 – Gráfico da velocidade do centro de massa do veículo com o controle CDTA

A Figura 50 mostra que o controle foi capaz de manter as normais positivas durante mais tempo que o proporcional simples, maximizando assim a força de contato com o chão.

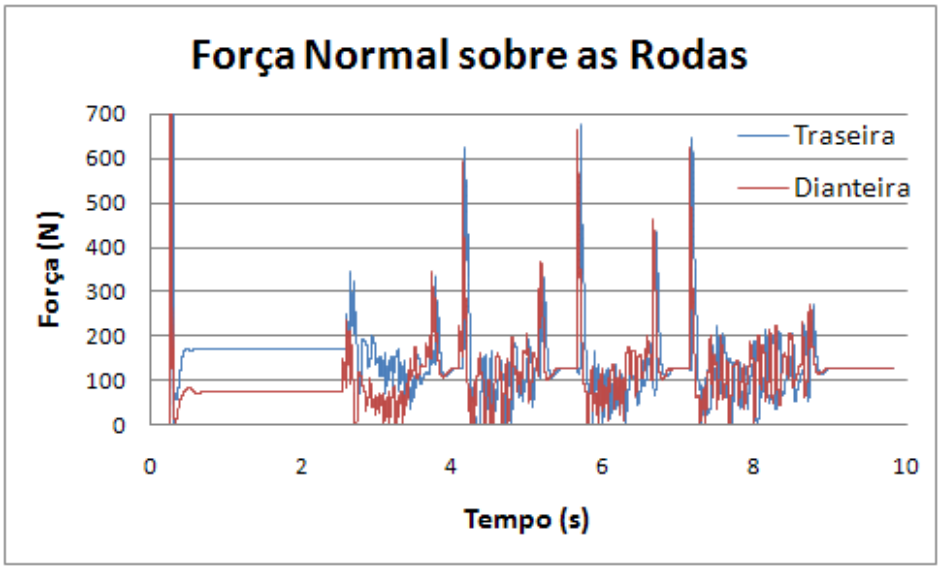


Figura 50 – Força normal com o controle CDTA

A Figura 51 demonstra que a deriva longitudinal com esse controle atingiu valores menores em módulo do que com o controle proporcional simples. Isso evidencia que as rodas derraparam menos, maximizando assim as forças de tração de cada pneu.

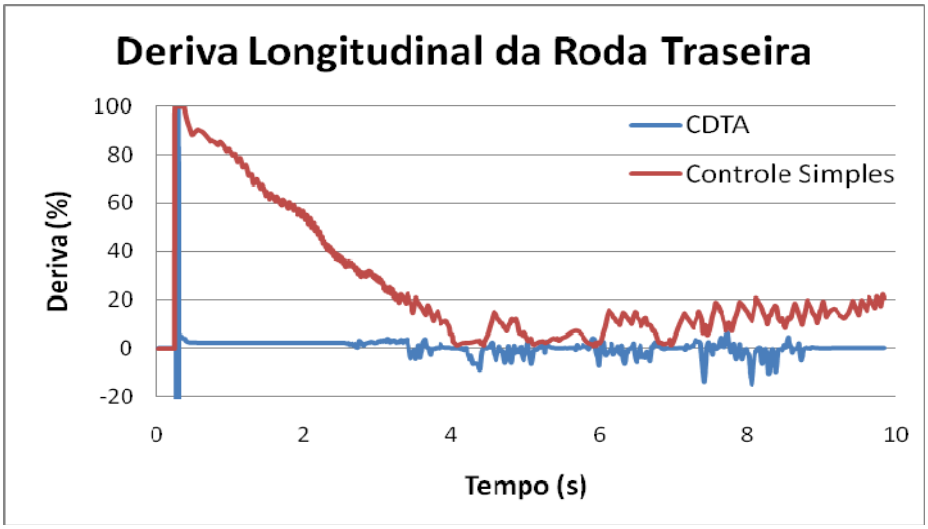


Figura 51 – Gráfico da deriva longitudinal da roda traseira com o controle CDTA

O controle CDTA conseguiu fazer com que a potência dissipada nos motores permanecesse abaixo dos 1000W durante grande parte da simulação. Em contraste, com o controle proporcional simples, a potência dissipada oscilou constantemente em torno de 2000W (Figura 52).

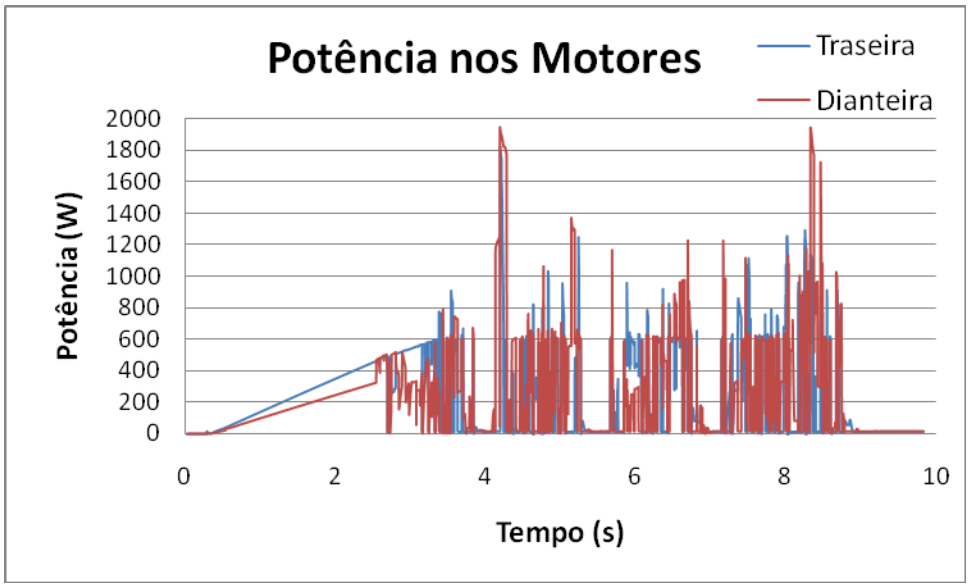


Figura 52 – Potência dos motores com o controle CDTA

6.3.

Controle Proporcional Simples em Terreno Plano com uma Rampa

Esta simulação foi feita com o controlador proporcional simples com compensação de gravidade em um terreno plano com uma rampa. Nesta simulação, não houve estabilização do veículo após passar pela rampa.

Como pode-se observar na Figura 53, o controlador não foi capaz de compensar o atrito, e se afastou do *setpoint* ao final da simulação: mesmo comportamento do controle proporcional simples no terreno senoidal.

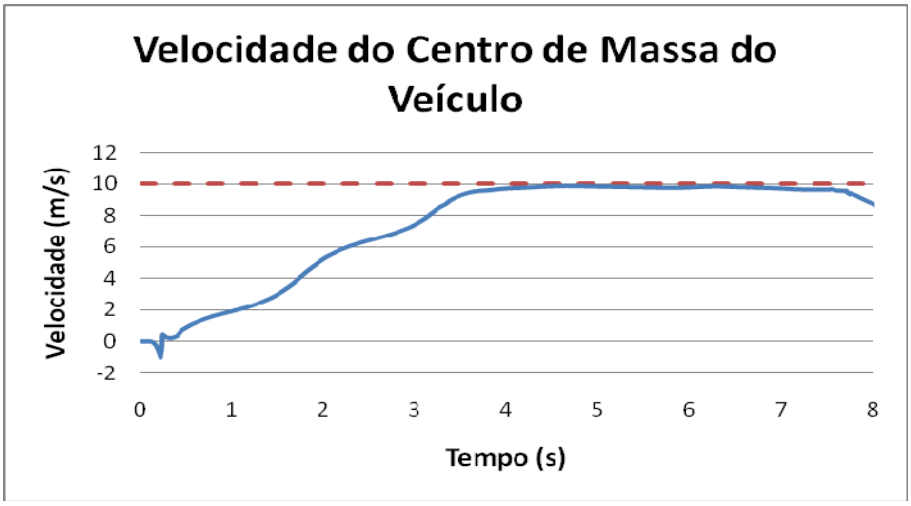


Figura 53 – Gráfico da velocidade do centro de massa do veículo com o controle proporcional simples em terreno plano com uma rampa

A Figura 54 mostra a variação das normais sobre as rodas dianteira e traseira. Pela simplicidade do controle, as forças normais não são monitoradas, o que permite o descolamento entre as rodas e o terreno (força normal nula). Logo, esse tipo comportamento não é desejável, pois em uma situação crítica compromete a estabilidade e a tração do veículo.

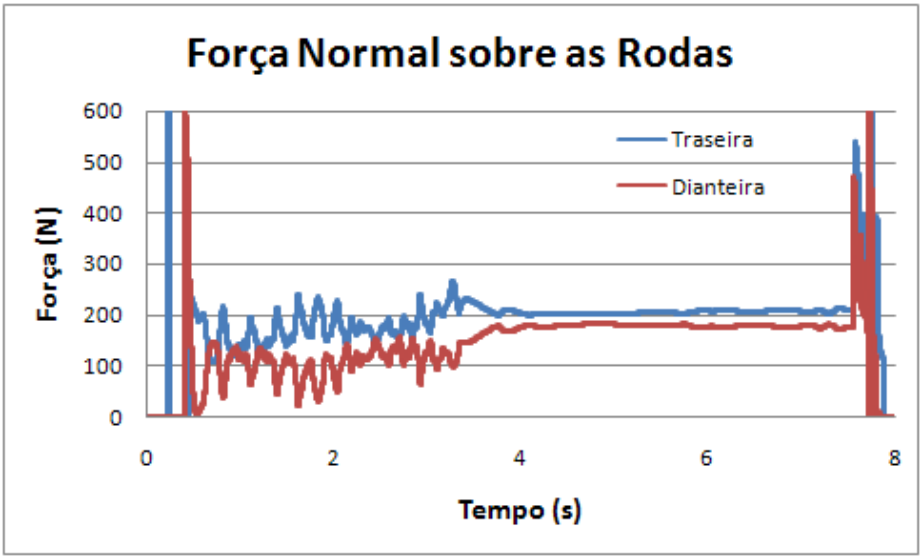


Figura 54 – Força normal com o controlador proporcional simples em terreno plano com uma rampa

A Figura 55 indica como a deriva longitudinal foi alta durante a simulação, evidenciando que durante o percurso as rodas deslizaram mais que o devido. Essa derrapagem excessiva fez com que as rodas trabalhassem com uma força de tração

bem menor que a desejada. No início da simulação, pode-se observar como a deriva longitudinal oscilou, justificando o comportamento oscilatório do centro de massa do veículo (Figura 53).

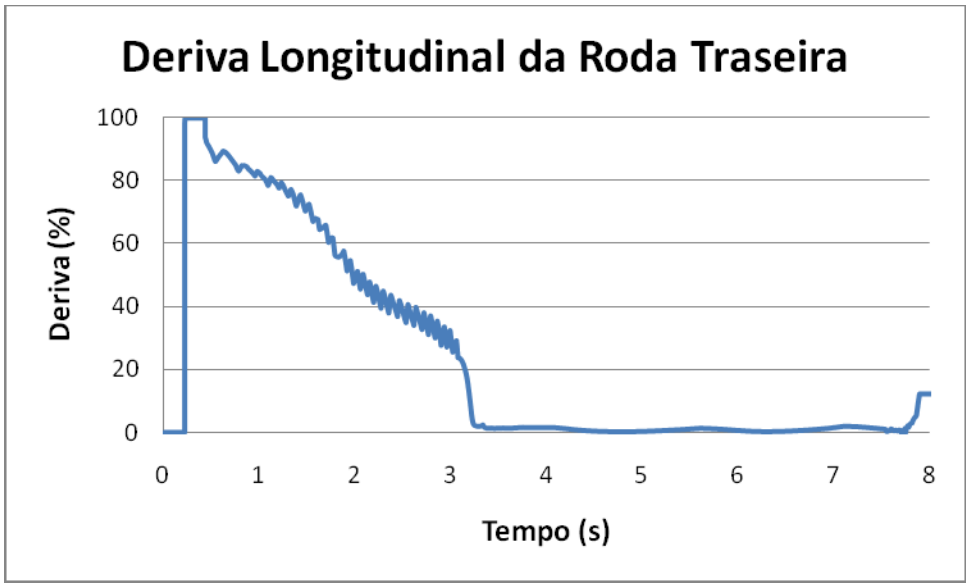


Figura 55 – Gráfico da deriva longitudinal da roda traseira com o controle proporcional simples em terreno plano com uma rampa

Outro problema identificável pelo desempenho desse controlador é que a potência dissipada nos motores é muito alta, como pode ser visto na Figura 56. Esse fator acaba comprometendo a autonomia do veículo robótico, pois a energia consumida pelas baterias é maior do que a necessária.

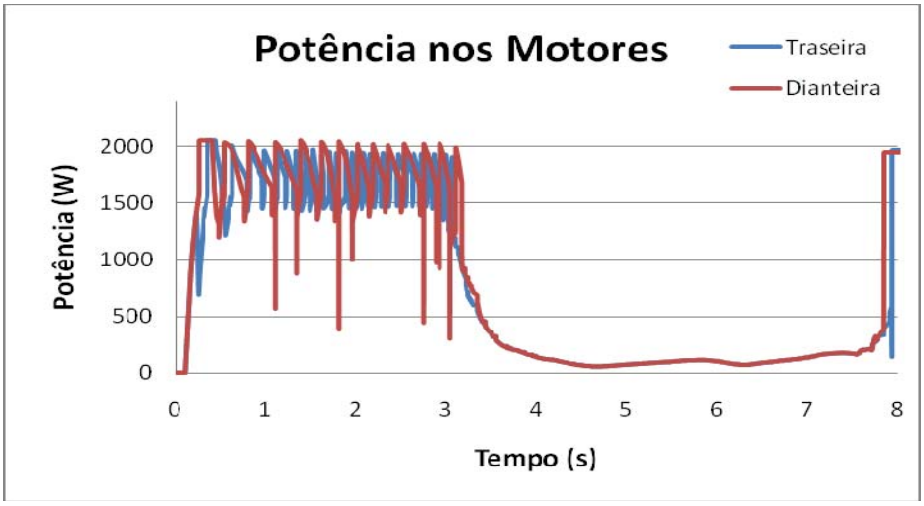


Figura 56 – Potência dos motores com o controlador proporcional simples em terreno plano com uma rampa

Como o veículo não conseguiu superar a rampa, essa simulação foi interrompida após o oitavo segundo, com o veículo "capotado". O simulador considera que o veículo capotou quando o vetor ***b*** estiver apontando para baixo, ou seja, quando a sua componente *z* for negativa. Com isso, o simulador interrompe a simulação.

6.4.
Controle CDTA em Terreno Plano com uma Rampa

Nesta simulação, sob condições idênticas às da seção anterior e com mesmo tipo de terreno plano com uma rampa, o controle CDTA permitiu um valor maior de tração nas rodas, possibilitando uma convergência mais rápida da velocidade. O controle de tração e a preocupação de manter a força normal positiva o máximo possível fizeram com que o veículo mantivesse a velocidade desejada até o final da simulação.

A Figura 57 mostra que o veículo robótico, ao passar pela rampa, por volta de 7,5s, a uma velocidade desejada, perde o contato com o terreno. As normais vão à zero (Figura 58), mas, imediatamente o controle começa a corrigir a velocidade, que converge em seguida. Quando o veículo perde o contato com o terreno, o controle não é capaz de manter a velocidade desejada.

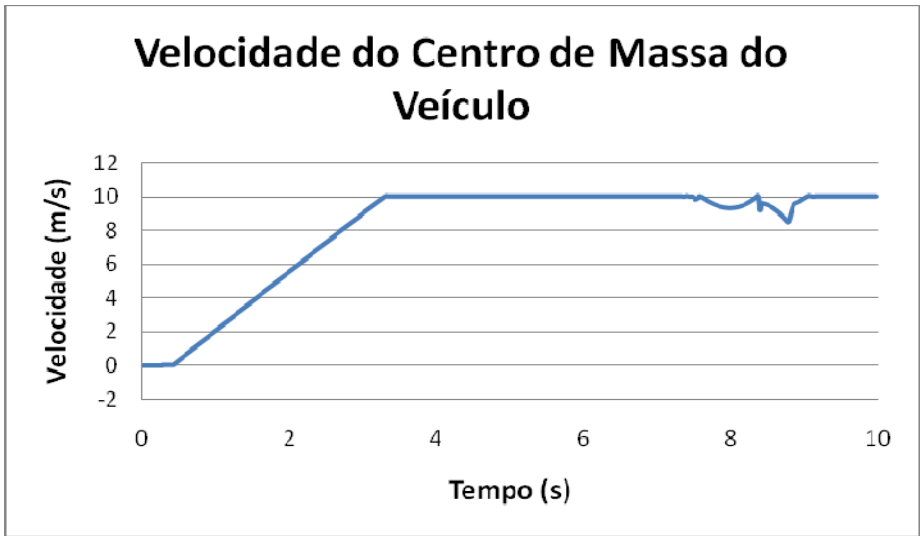


Figura 57 – Velocidade do centro de massa do veículo com o controle CDTA em terreno plano com uma rampa

Na Figura 58 pode-se visualizar que o controle foi capaz de manter as normais positivas mais tempo que o proporcional simples, maximizando assim a força de contato com o chão.

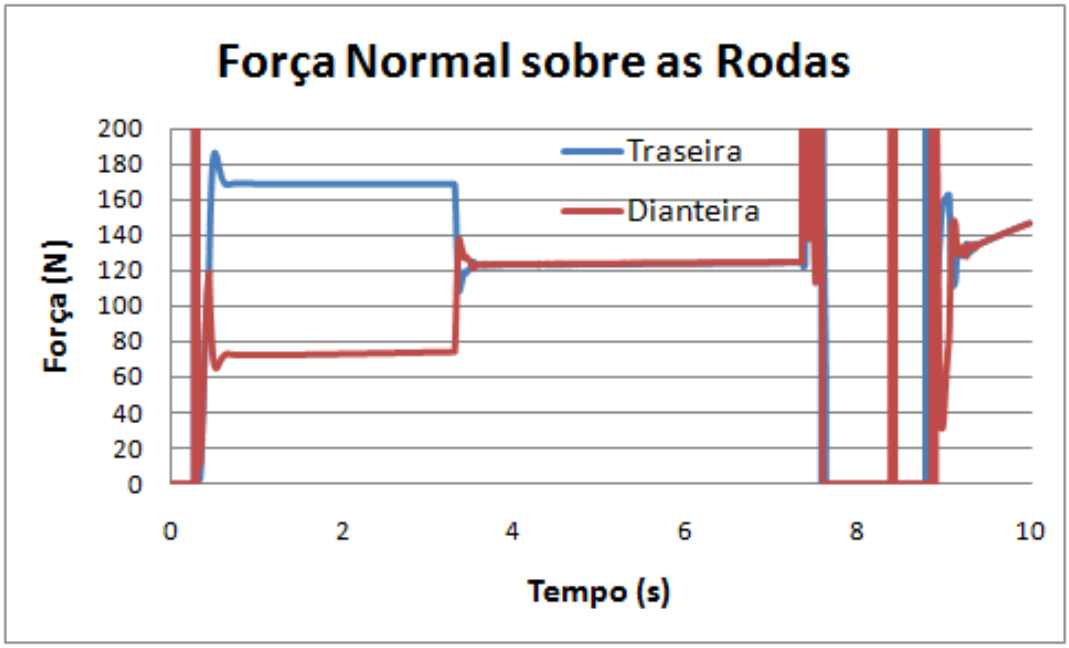


Figura 58 – Força normal com o controle CDTA em terreno plano com uma rampa

A Figura 59 indica que a deriva longitudinal com esse controle atingiu valores menores, em módulo, que os do controle proporcional simples,

evidenciando que as rodas derraparam menos e maximizando assim as forças de tração.

Pode-se observar na Figura 59 que a simulação do controle simples, em vermelho, é interrompida ao passar pela rampa por volta de 8s de simulação. Esse fato se deve pelo controle simples não conseguir manter o veículo controlado.

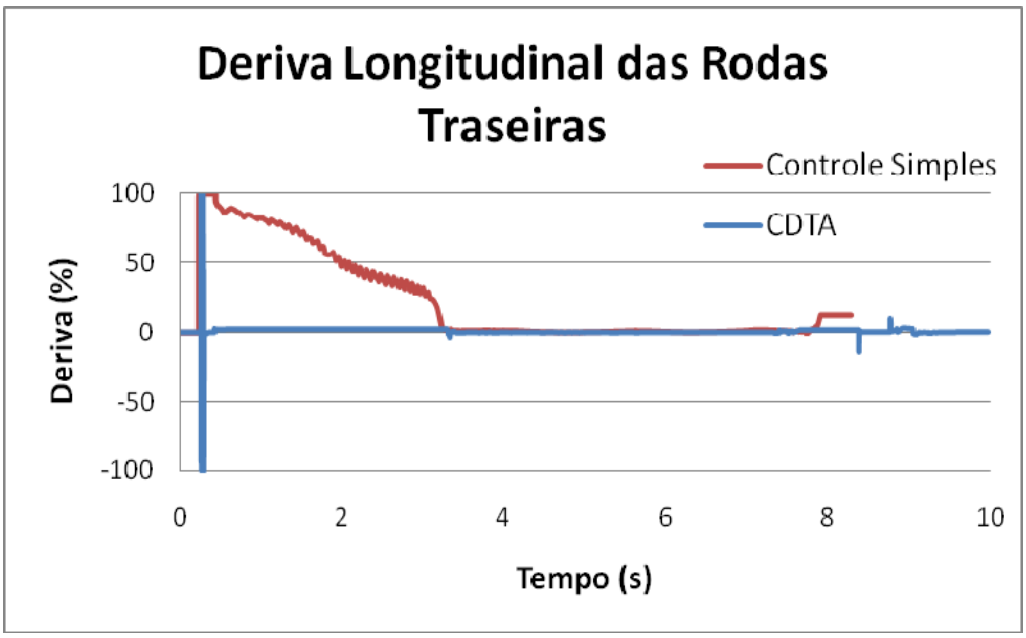


Figura 59 – Deriva longitudinal da roda traseira com o controle CDTA em terreno plano com uma rampa

O controle conseguiu fazer com que a potência dissipada nos motores permanecesse abaixo dos 1000W durante grande parte da simulação, enquanto o controle proporcional simples oscilou em torno de 2000W (Figura 60).

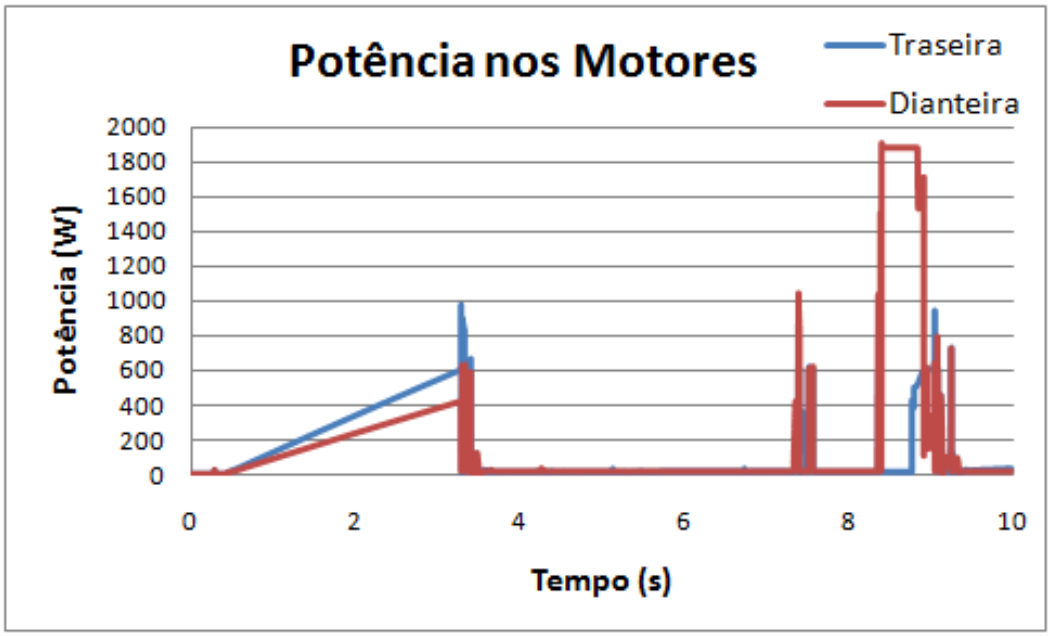


Figura 60 – Potência dos motores com o controle CDTA em terreno plano com uma rampa

No próximo capítulo, as conclusões deste estudo são apresentadas.

7
Conclusões

Neste estudo, foi desenvolvido um simulador 3D em tempo real para veículos robóticos em terrenos acidentados, batizado de VirtualBotz 3D. A plataforma de simulação foi toda desenvolvida em linguagem C++, utilizando o conceito de modularização através da programação orientada a objetos. Como ferramenta de interface gráfica, foi utilizada a biblioteca OpenGL.

Houve um cuidado particular no realismo do sistema, em especial na dinâmica e nos modelos. A intenção foi incluir características não-lineares em terrenos acidentados que se revelam muito importantes para o comportamento de robôs reais, como os robôs experimentais desenvolvidos pelo Laboratório de Robótica da PUC-Rio.

Esse simulador permite a importação e definição de diversos tipos de terrenos e veículos robóticos, assim como a definição das características dos motores de corrente contínua (DC). Ao levar em consideração as características físicas de um motor DC, o presente estudo teve um cuidado especial na modelagem da interação da parte mecânica com a elétrica. A consideração das limitações físicas das baterias elétricas, e o uso de uma aproximação contínua do modelo de atrito de LuGre, trouxeram ao simulador uma característica bem realista.

Foi desenvolvido também um algoritmo de interseção entre cada roda e o terreno, o que permitiu construir o simulador sem a necessidade de utilizar qualquer biblioteca comercial de cálculo de colisões. Para modelar a força de contato entre cada pneu e o terreno, utilizou-se a "**Fórmula Mágica**", que leva em consideração as derivas lateral e longitudinal combinadas e a força normal sobre as rodas.

Para garantir que os resultados deste trabalho fossem confiáveis, foram feitas validações do simulador quanto às leis da física. Apresentaram-se dois casos, um de **arrancada** e um de **salto**, onde há soluções analíticas. Na validação

da **arrancada** foram feitos quatro testes, onde em dois dos quais foi explorada a representação geométrica das rodas no simulador. Constatou-se que um nível maior de detalhamento ocasionou uma melhora do erro de 0,34% para 0,03% para as forças na suspensão dianteira. No entanto, essa melhora custou um aumento no tempo de processamento de 467% no simulador, sugerindo que esta não é significativa se comparada ao custo que representa. Nos outros testes verificou-se que, sem a melhora mencionada, o erro na força sobre a suspensão dianteira não passou de 1,23%. Para os testes de guinada não foram encontrados erros significativos entre as simulações e os resultados analíticos. O erro obtido em todos os testes de validação de **salto** foi de 0,02%.

O modelo de controle dinâmico para terrenos acidentados de estabilidade 2D proposto por Silva, e implementado no presente estudo através do simulador VirtualBotz 3D, permitiu uma validação de modelos analíticos do comportamento longitudinal do veículo robótico. Nas duas simulações feitas com características de terrenos distintas, pode-se observar que, ao utilizar o controle CDTA ao controle simples, o veículo robótico apresentou maior tração nas rodas e uma convergência muito mais rápida da velocidade. Isso aconteceu porque a preocupação de manter a força normal positiva o máximo possível permitiu que o veículo mantivesse a velocidade desejada até o final da simulação. Ao manter as forças normais sob controle, o sistema permite a maximização da força de contato com o chão. É evidenciado também que o CDTA, ao maximizar a força de tração em cada pneu, evitou que a deriva longitudinal atingisse valores indesejáveis. Outro ganho demonstrado com o CDTA foi quanto à minimização da potência exigida aos motores: pode-se observar que, enquanto no controle proporcional simples, a potência oscilou em torno de 2000W, no CDTA, ela permaneceu abaixo dos 1000W em grande parte da simulação. Um ponto muito importante é que o CDTA foi capaz de estabilizar o veículo depois do salto pela rampa na segunda simulação.

8
Referências bibliográficas

[1] IAGNEMMA, K., DUBOWSKY, S. **Mobile Robots in Rough Terrain: Estimation, Motion Planning, and Control with Application to Planetary Rovers**. New York: Springer, 2004.

[2] IAGNEMMA, K., DUBOWSKY, S. **Traction Control of Wheeled Robotic Vehicles in Rough Terrain with Application to Planetary Rovers**. Massachusetts Institute of Technology, Department of Mechanical Engineering, Cambridge, USA, 2004.

[3] LAMON, P., SIEGWART, R. **Wheel Torque Control in Rough Terrain - Modeling and Simulation**. Proceedings of the IEEE International Conference on Robotics and Automation, Barcelona, 2005.

[4] SILVA, Alexandre F. Barral. **Modelagem de Sistemas Robóticos Móveis para Controle de Tração em Terrenos Acidentados**. Pontifícia Universidade Católica do Rio de Janeiro – PUC-RIO, Departamento de Engenharia Mecânica, abril de 2007.

[5] SANTOS, Auderi V. **Controle de Capotagem e Deslizamento de Sistemas Robóticos Móveis em Terrenos Acidentados**. Pontifícia Universidade Católica do Rio de Janeiro – PUC-RIO, Departamento de Engenharia Mecânica, maio de 2007.

[6] CAMPION, G., CHUNG, W. **Wheeled Robots**. Springer Handbook of Robotics Siciliano, Khatib (Ed.). N^{o} 17, Págs. 391-410, Springer, 2008.

[7] BURGARD, W., HEBERT, M. **World Modeling**. Springer Handbook of Robotics Siciliano, Khatib (Eds.). N^{o} 36, Págs. 391-410, Springer, 2008.

[8] CHAKRABORTY, N., GHOSAL, A. **Kinematics of Wheeled Mobile Robots on Uneven Terrain**. Department of Mechanical Engineering, Indian Institute of Science, maio de 2004.

[9] SWEET, A. (University of California, Berkeley), BLACKMON, T. (NASA Ames Research Center), GUPTA, V. (NASA Ames Research Center). **Simulation of a Rover and Display in a Virtual Environment**. University of California, Berkeley, 1999.

[10] CLARAty, NASA.
Disponível em <http://claraty.jpl.nasa.gov/man/overview/index.php>.
Acesso em 05 de agosto de 2010.

[11] Rover Graphical Simulator (RGS), NASA.
Disponível em <http://www.techbriefs.com/content/view/1782/34/>.
Acesso em 05 de agosto de 2010.

[12] Universal Mechanisms, Laboratory of Computational Mechanics Bryansk State Technical University, Rússia.
Disponível em <http://www.umlab.ru/>. Acesso em 05 de agosto de 2010.

[13] GNU General Public License.
Disponível em <http://www.gnu.org/licenses/licenses.html#GPL>.
Acesso em 05 de agosto de 2010.

[14] NASA Tech Briefs. **Rover Graphical Simulator - Technical Support Package**. NPO-35223. NASA's Jet Propulsion Laboratory, Pasadena, California.

[15] GAZEBO - 3D Multiple Robot Simulator with Dynamics.
Disponível em <http://playerstage.sourceforge.net/index.php?src=gazebo>.
Acesso em 05 de agosto de 2010.

[16] BAUER, R., BARFOOT, T., LEUNG, W., RAVINDRAN, G. **Dynamic Simulation Tool Development for Planetary Rovers**. Jet Propulsion Laboratory, California Institute of Technology, Pasadena, Califórnia 91109.

International Journal of Advanced Robotic Systems, Vol. 5, Nº 3, ISSN 1729-8806, Págs. 311-314, 2008.

[17] Design-Simulation. Working Model 2D.
Disponível em <http://www.design-simulation.com/wm2d/index.php>.
Acesso em 05 de agosto de 2010.

[18] THUEER, T., KREBS, A., SIEGWART, R., LAMON, P. **Performance Comparison of Rough-Terrain Robots – Simulation and Hardware**. Journal of Field Robotics - Special Issue on Space Robotics, Vol. 24, Ed. 3, Págs. 251 – 271, março de 2007.

[19] Autonomous System Labs. CRAB Rover.
Disponível em <http://www.asl.ethz.ch/robots/>. Acesso em 20 de agosto de 2010.

[20] LINDEMANN, R.A., BICKLER, D.B., HARRINGTON, B.D., ORITZ, G.M., VOORHEES, C.J. **Mars Exploration Rover Mobility Development - Mechanical Mobility Design**, Development, and Testing. Robotics & Automation Magazine, IEEE, Vol. 13, Págs. 19-26, 2006.

[21] Russell Smith , Open Dynamics Engine.
Disponível em <http://ode.org/>. Acesso em 05 de agosto de 2010.

[22] Adams Multibody Dynamics.
Disponível em <http://www.mscsoftware.com/Contents/Products/CAE-Tools/Adams.aspx>. Acesso em 20 de agosto de 2010.

[23] Robotics Research Group, The University of Texas, Austin. Disponível em <http://www.robotics.utexas.edu/rrg/>. Acesso em 20 de agosto de 2010.

[24] DUDEK, G., JENKIN, M. **Computational Principles of Mobile Robotics**. United Kingdom, Cambridge, 2000.

[25] ROAMS (Rover Analysis, Modeling and Simulation). Disponível em <http://dshell.jpl.nasa.gov/ROAMS/index.php>. Acesso em 05 de agosto de 2010.

[26] ESTLIN, T., YEN, J., PETRAS, R., MUTZ, D., CASTAÑO, R., RABIDEAU, G., STEELE, R., JAIN, A., CHIEN, S., MJOLSNESS, E., GRAY, A., MANN, T., HAYATI, S., DAS, H. **An Integrated Architecture for Cooperating Rovers**. Artificial Intelligence, Robotics and Automation in Space, Proceedings of the Fifth International Symposium, ISAIRAS '99, held 1-3 June, 1999 in ESTEC, Noordwijk, the Netherlands. Edited by M. Perry. ESA SP-440. Paris: European Space Agency, Págs. 255-262, 1999

[27] VOLPE, R., NESNAS, I., ESTLIN, T., MUTZ, D., PETRAS, R., DAS, H. **The CLARAty Architecture for Robotic Autonomy**. Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California 91109.

[28] NESNAS, R.I.A.D., SIMMONS, R., GAINES, D., KUNZ, C., DIAZ-CALDERON, A., ESTLIN, T., MADISON, R., GUINEAU, J., MCHENRY, M., SHUL, I., APFELBAUM, D. **CLARAty: Challenges and Steps Toward Reusable Robotic Software**. Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California 91109. International Journal of Advanced Robotic Systems, Vol. 3, No. 1, ISSN 1729-8806, Págs. 23-30, 2006

[29] NESNAS, I.A.D., WRIGH, A., BAJRACHARYA, M., SIMMONS, R., ESTLIN, T., KIM, W.S. **CLARAty: An Architecture for Reusable Robotic Software**. Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California 91109.

[30] NESNAS, I.A.D. **CLARAty: A Collaborative Software for Advancing Robotic Technologies**.

[31] USC Robotics Research Lab. **Player, Stage and Gazebo**. Disponível em <http://www-robotics.usc.edu/?l=projects:playerstagegazebo>. Acesso em 05 de agosto de 2010.

[32] NVidia White Paper. **Accelerating MATLAB with CUDA™ Using MEX Files**. WP-03495-001_v01, setembro de 2007.

[33] RDS - Microsoft Robotics Developer Studio 2008. Disponível em <http://msdn.microsoft.com/en-us/library/bb483024.aspx>. Acesso em 05 de agosto de 2010.

[34] PhysX AGEIA Technologies, Inc. . Disponível em Disponível em: <http://www.nvidia.com/object/physx_new.html>. Acesso em 05 de agosto de 2010.

[35] Microsoft DirectX. Disponível em <http://www.microsoft.com/games/en-US/aboutGFW/pages/directx.aspx>. Acesso em 05 de agosto de 2010.

[36] JAIN, A., BALARAM, J., CAMERON, J., GUINEAU, J., LIM, C., POMERANTZ, M., SOHL, G. **Recent Developments in the ROAMS Planetary Rover Simulation Environment**. Jet Propulsion Laboratory, California Institute of Technology.

[37] CLARAty Movies, NASA. Disponível em <http://claraty.jpl.nasa.gov/man/overview/movies/index.php>. Acesso em 05 de agosto de 2010.

[38] KREBS, A., THUEER, T., CARRASCO, E., SIEGWART, R. **Towards Torque Control of the CRAB Rover**. Autonomous Systems Lab, ETH Zurich.

[39] CLARAty Download, NASA. Disponível em <http://claraty.jpl.nasa.gov/man/software/download/index.php>. Acesso em 05 de agosto de 2010.

[40] CLARAty Public License, NASA. Disponível em <http://claraty.jpl.nasa.gov/man/software/license/public_src/index.php>. Acesso em 05 de agosto de 2010.

[41] SMITH, R. Open Dynamics Engine v0.5 User Guide, fevereiro de 2006. Disponível em <http://ode.org/ode-latest-userguide.html>. Acesso em 05 de agosto de 2010.

[42] SIGGRAPH. **Performance OpenGL: Platform Independent Techniques**. Course # 37, 2002.

[43] Khronos Group, OpenGL: Overview. São Francisco. Disponível em <http://www.opengl.org/about/overview/>. Acesso em 05 de agosto de 2010.

[44] Autodesk, Alias|Wavefront Maya. Disponível em <http://usa.autodesk.com/adsk/servlet/pc/index?siteID=123112&id=13577897>. Acesso em 05 de agosto de 2010.

[45] Amsterdam Blender Institute, Blender, Amsterdam. Disponível em <http://www.blender.org/>. Acesso em 05 de agosto de 2010.

[46] OMG, UML - Unified Modeling Language Specification, Versão 1.5, formal/03-03-01, março de 2003.

[47] SIMPACK AG. Disponível em <http://www.simpack.com/home-simpack.html>. Acesso em 05 de agosto de 2010.

[48] RioBotz COMBAT TUTORIAL, Versão 2.0, março 2009.

[49] MADDEN, J.D. **Mobile Robots: Motor Challenges and Materials Solutions**. Science, Vol. 318, 16 de novembro de 2007.

[50] PAPADOPOULOS, E., REY, D. **The Force-Angle Measure of Tipover Stability Margin for Mobile Manipulators**. Vehicle System Dynamics, Vol. 33, 1ª Edição, Págs. 29-48, janeiro de 2000.

[51] PETERS, S.C., IAGNEMMA, K. **An Analysis of Rollover Stability Measurement for High-Speed Mobile Robots**. Proceedings of 2006 IEEE International Conference on Robotics and Automation, Orlando, Flórida, Págs. 3711–3716, maio de 2006.

[52] PETERS, S.C., IAGNEMMA, K. **Stability Measurement of High-Speed Vehicles**. Vehicle System Dynamics, Vol. 47, 6ª Edição, Págs. 701-720, junho de 2009.

[53] PACEJKA, H.B. **Tire and Vehicle Dynamics**. SAE – Society of Automotive Engineers, Inc., 2ª Edição, Págs. 156-215, 2006.

[54] WONG, J.Y. **Theory of Ground Vehicles**. John Wiley & Sons, Inc., 4ª Edição, 2008.

[55] SOBCZYK, S., MARIO, R., PERONDI, E.A., CUNHA, M.A.B. **A Continuous Approximation of the LuGre Friction Model**. 20th International Congress Of Mechanical Engineering – COBEM 2009, Gramado, RS, Brazil, 15 e 20 de november de 2009.

[56] SILVA, A.F.B., SANTOS, A.V., MEGGIOLARO, M.A., Neto, M.S. **A Rough Terrain Traction Control Technique for All-Wheel-Drive Mobile Robots**. ABCM, 2010.

[57] The DARTS Simulation Laboratory, NASA. Disponível em <http://www-robotics.jpl.nasa.gov/facilities/facility.cfm?Facility=7>. Acesso em 05 de agosto de 2010.

[58] AESCO GbR Automotive Engineering, Software & Consulting. Disponível em < http://www.english.aesco.de/>. Acesso em 05 de agosto de 2010.

[59] GLUT. **The OpenGL Utility Toolkit**. OpenGL. Disponível em < http://www.opengl.org/resources/libraries/glut/ >. Acesso em 05 de agosto de 2010.

[60] KOENIG, N. **Stage and Gazebo - The Instant Expert's Guide**. USC Robotics Research Labs, 15 de setembro de 2004.

[61] CANUDAS DE WIT, C., OLSSON, H., ASTROM, K.J., LISCHINSKY, P. **A New Model for Control Systems with Friction**. IEEE Transactions on Automatic Control, Vol. 40, Nº 3, Págs. 419-425, 1995

[62] JAZAR, NAKHAIE, G. **Vehicle Dynamics: Theory and Application**. Springer, 2008.

Anexo A:
Organização dos Arquivos do Aplicativo VirtualBotz 3D

O aplicativo está organizado em pastas (ou diretórios) para facilitar a compreensão de suas partes. Este **Anexo** mostra apenas uma visão abrangente de sua organização. Para maiores detalhes, consulte os outros anexos.

Tabela 9 – Pastas do aplicativo

Pasta	Descrição
VirtualBotz_Version_2.000.5/	
_library/	pasta com as bibliotecas em c++
Botz/	pasta com as classes de negócio
Graph/	pasta com as classes para recursos gráficos (classes de negócio)
Math/	pasta com as classes para operações vetoriais e matriciais (classes de negócio)
OpenGL/	pasta com as classes de visualização em OpenGL
Tools/	pasta com as bibliotecas de uso geral
Build/	
Heightmaps/	pasta com os arquivos de Mapas de Alturas para construção de terrenos
Obj_files/	pasta com os arquivos de objetos 3D e de materiais para rodas e veículos
Report/	pasta com os arquivos de dados gerados pelo aplicativo durante a simulação
Scripts/	pasta com os arquivos de configuração das simulações do aplicativo
Tests/	pasta com os arquivos de testes para comparações de resultados
Textures/	pasta com os arquivos de texturas do terreno

Videos/	pasta com os arquivos de vídeo gerados pelo aplicativo durante as simulações
Projects/	
VS6/	pasta com o projeto para a versão em Microsoft Visual Studio *6.0*
VS9/	pasta com o projeto para versão em Microsoft Visual Studio *9.0*

Anexo B:
Diagrama de Classes (UML) do Simulador

As classes do simulador estão organizadas nas seguintes pastas, nos respectivos arquivos de código mostrados na tabela a seguir:

Tabela 10 – Localização e descrição de todas as bibliotecas do aplicativo

Localização em pastas	Descrição
_library/	
typedefs.h	biblioteca para declarações / definições de tipos globais usados no sistema
globalvars.h *globalvars.cpp*	variáveis globais
main.cpp	programa principal
Graph/	
CVRImage.h *CVRImage.cpp*	biblioteca gráfica para leitura de bitmap padrão e OpenGL
Math/	
CMtx33.h *CMtx33.cpp*	biblioteca de matemática para manipulação de matriz 3x3
CVec3.h *CVec3.cpp*	biblioteca de matemática para manipulação de vetor de três posições
Botz/	
CVRAvi.h *CVRAvi.cpp*	biblioteca para Audio Video Interleave
CVRControl.h *CVRControl.cpp*	biblioteca responsável pelo controle do veículo robótico
CVRCorner.h *CVRCorner.cpp*	biblioteca base para criação do veículo robótico

CVRMobileRobot.h *CVRMobileRobot.cpp*	biblioteca responsável pela criação do veículo robótico (chassi + rodas)
CVRMotor.h *CVRMotor.cpp*	biblioteca responsável pela criação dos motores
CVRMotorDataSheet.h *CVRMotorDataSheet.cpp*	biblioteca para criação da folha de dados dos motores
CNumericalMethod.h *CNumericalMethod.cpp*	biblioteca para o método de integração do sistema
CVRState.h *CVRState.cpp*	biblioteca de histórico dos estados do sistema
CVRTerrain_.h *CVRTerrain_.cpp*	biblioteca para criação de terrenos
CVRThread.h *CVRThread.cpp*	biblioteca para manipulação de processos
CVRToolsFiles.h *CVRToolsFiles.cpp*	biblioteca responsável pela leitura dos scripts do aplicativo
CVRWheel.h *CVRWheel.cpp*	biblioteca para criação das rodas do veículo robótico
OpenGL*/*	
COGLComponent.h *COGLComponent.cpp*	biblioteca para superclasse “componente” OpenGL
COGLTerrain.h *COGLTerrain.cpp*	biblioteca OpenGL para o terreno
COGLObject.h *COGLObject.cpp*	biblioteca OpenGL para objetos 3D
COGLTexture.h *COGLTexture.cpp*	biblioteca OpenGL para texturas
COGLWheel.h *COGLWheel.cpp*	biblioteca OpenGL para rodas
COGLMobileRobot.h *COGLMobileRobot.cpp*	biblioteca OpenGL para veículos robóticos

OGLSettings.h *OGLSettings.cpp*	biblioteca de configuração para visualização em OpenGL
***Tools**/*	
ToolsArrays.h	biblioteca para manipulação de vetores
ToolsRandom.h *TooslRandom.cpp*	biblioteca para obtenção de números aleatórios

- Classes relacionadas com o terreno

A Figura 61 mostra o diagrama de classes em UML em duas camadas da modelagem do terreno.

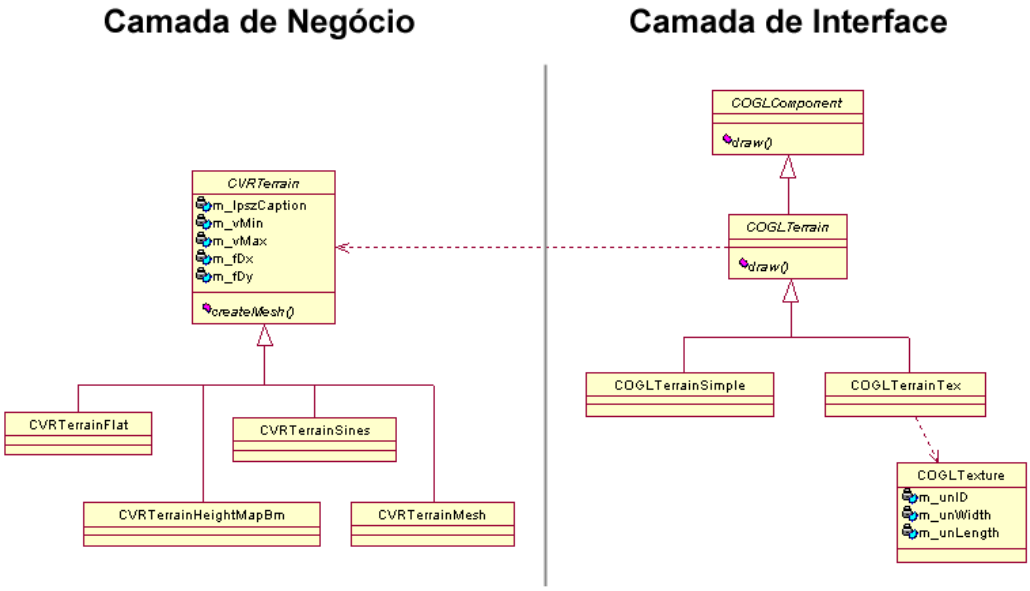


Figura 61 – Diagrama de classes (UML) do terreno

- Classes relacionadas com o veículo robótico

A Figura 62 mostra o diagrama de classes em UML em duas camadas da modelagem do veículo robótico.

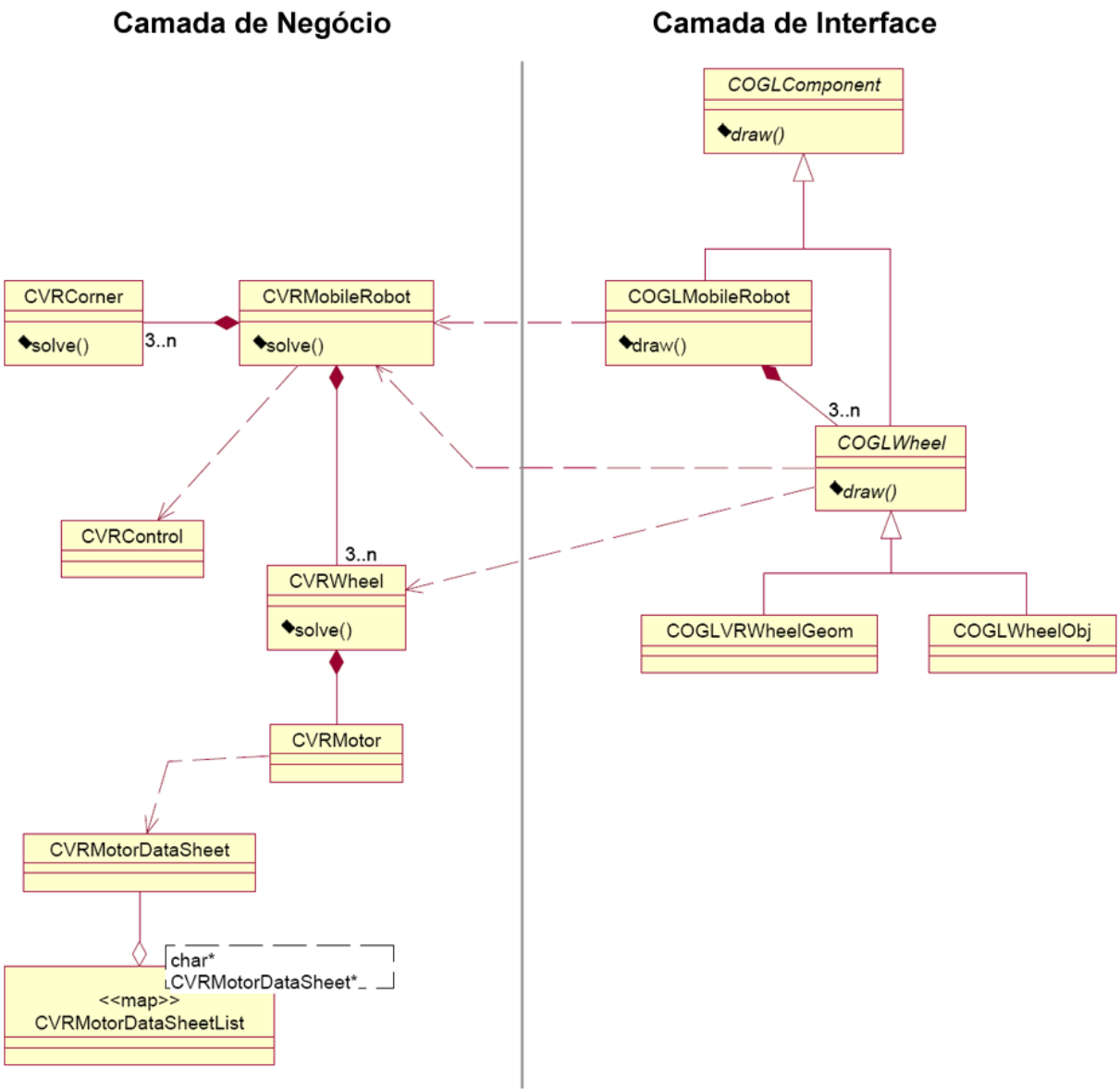


Figura 62 – Diagrama de classes (UML) do veículo robótico

- Classes extras

A Figura 63 mostra o diagrama de classes em UML da modelagem para as classes de operações matriciais.

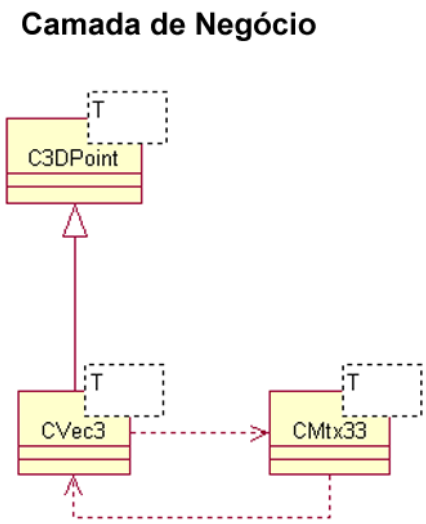


Figura 63 – Diagrama de classes (UML) para matrizes 3x3 e vetores 3x1/1x3

Anexo C:
Comandos do Joystick (Analógico)

O simulador deve funcionar com qualquer tipo de joystick analógico. Para isso, devem ser instalados os drivers que acompanham o próprio joystick. Os joysticks com dois *pads* (ou manetes) de comandos (Figura 64) seguem a seguinte configuração: os comandos do *pad* da esquerda são para o veículo robótico, e os comandos do *pad* da direita, para o mundo virtual, com as opções rotação, translação e *zoom*, selecionadas uma por vez ao pressionar o botão de comando.

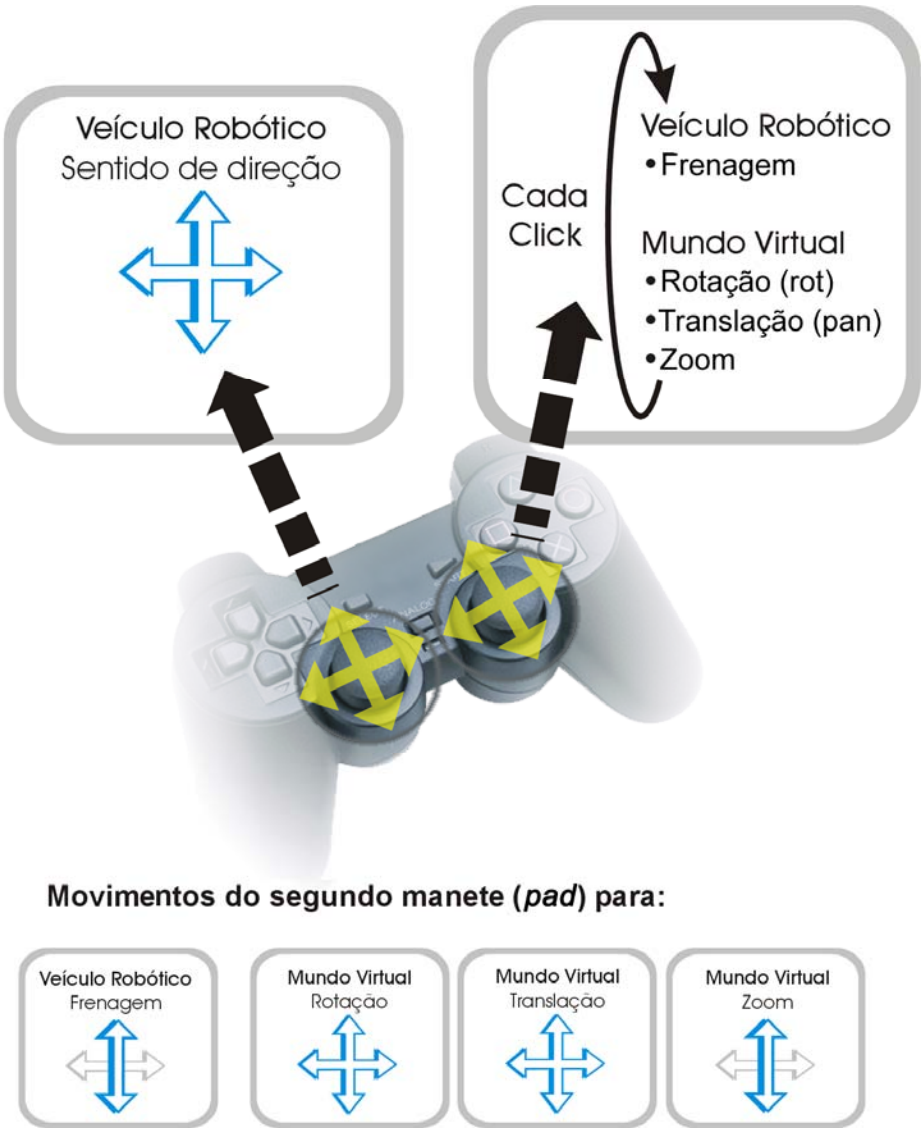


Figura 64 – Comandos dos manetes do joystick

O simulador também utiliza os quatro botões da direita do joystick analógico (Figura 65) após a parada de uma simulação (tecla "**P**"). Eles servem para avançar ou recuar o veículo robótico para as posições do centro de massa guardado no histórico da simulação. Essa operação pode também ser realizada através do teclado (ver item "**D.8. Posicionar o veículo robótico pelo centro de massa ao longo da trajetória após a simulação**" do **Anexo D**).

Ao pressionar uma vez o botão 1 (com o triângulo verde), a movimentação de recuo ou avanço do veículo robótico é feita automaticamente. Para o recuo do veículo robótico, deve-se pressionar o botão 4 (com o quadrado rosa) e, para o avanço, o botão 2 (com o círculo vermelho).

Figura 65 – Comandos dos botões do joystick

Anexo D:
Teclas de Funcionalidades do Simulador

O simulador tem algumas teclas de funcionalidades que estão em destaque na Figura 66.

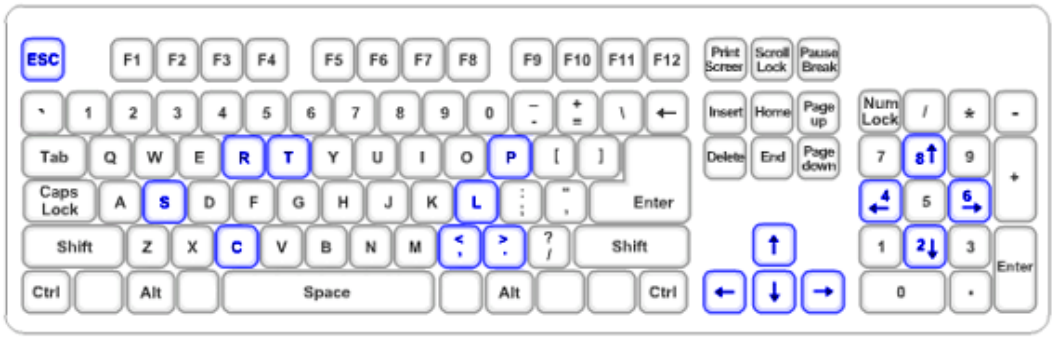


Figura 66 – Teclas de funcionalidades do simulador (teclado padrão internacional)

D.1. Terminar execução do simulador

A tecla "**Esc**" termina a execução do aplicativo, após salvar todos os arquivos, liberar todos os recursos de memória e inclusive restaurar o VSync (sincronização vertical da placa de vídeo), se estiver sendo utilizado.

D.2. Alternar visão da simulação pelas câmeras interna e externas do veículo robótico

A tecla "**C**" (ou "**c**") alterna entre os três tipos de câmeras existentes no aplicativo, que são as seguintes: 1) observador fixo em um ponto no espaço, que, porém, permite rotações e translações do mundo virtual com o mouse a fim de permitir uma visão panorâmica da área de teste (Figura 67); 2) câmera interna posicionada no interior do veículo robótico (Figura 68), cujo observador se movimenta conforme a dinâmica do veículo robótico, inclusive nas capotagens; 3) câmera posicionada externamente ao veículo a uma distância fixa (Figura 69).

Para a segunda e a terceira câmeras, não é permitida a utilização do mouse para manipulação do mundo virtual.

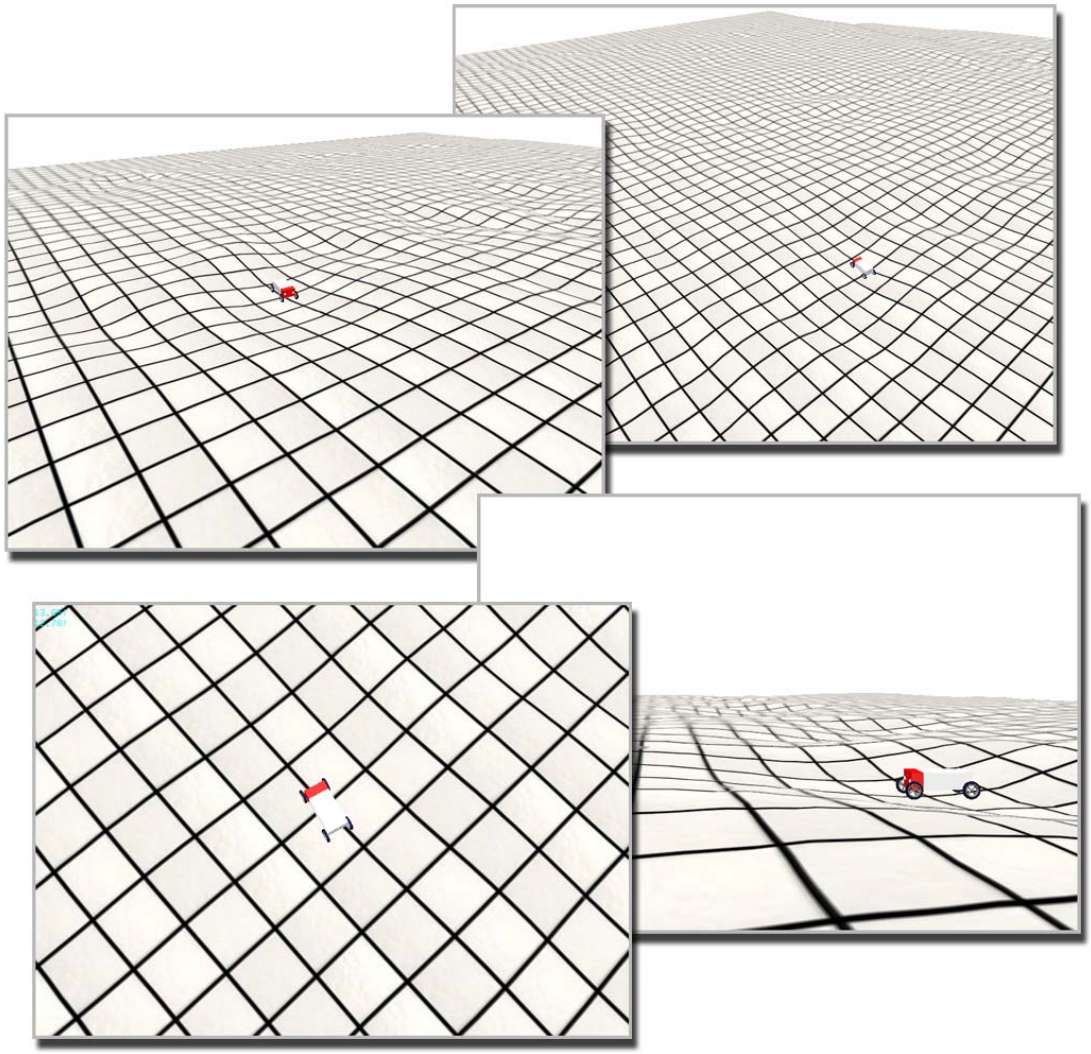


Figura 67 – Veículo robótico observado pela câmera virtual 1, permitindo a movimentação do mundo virtual de forma independente do veículo robótico

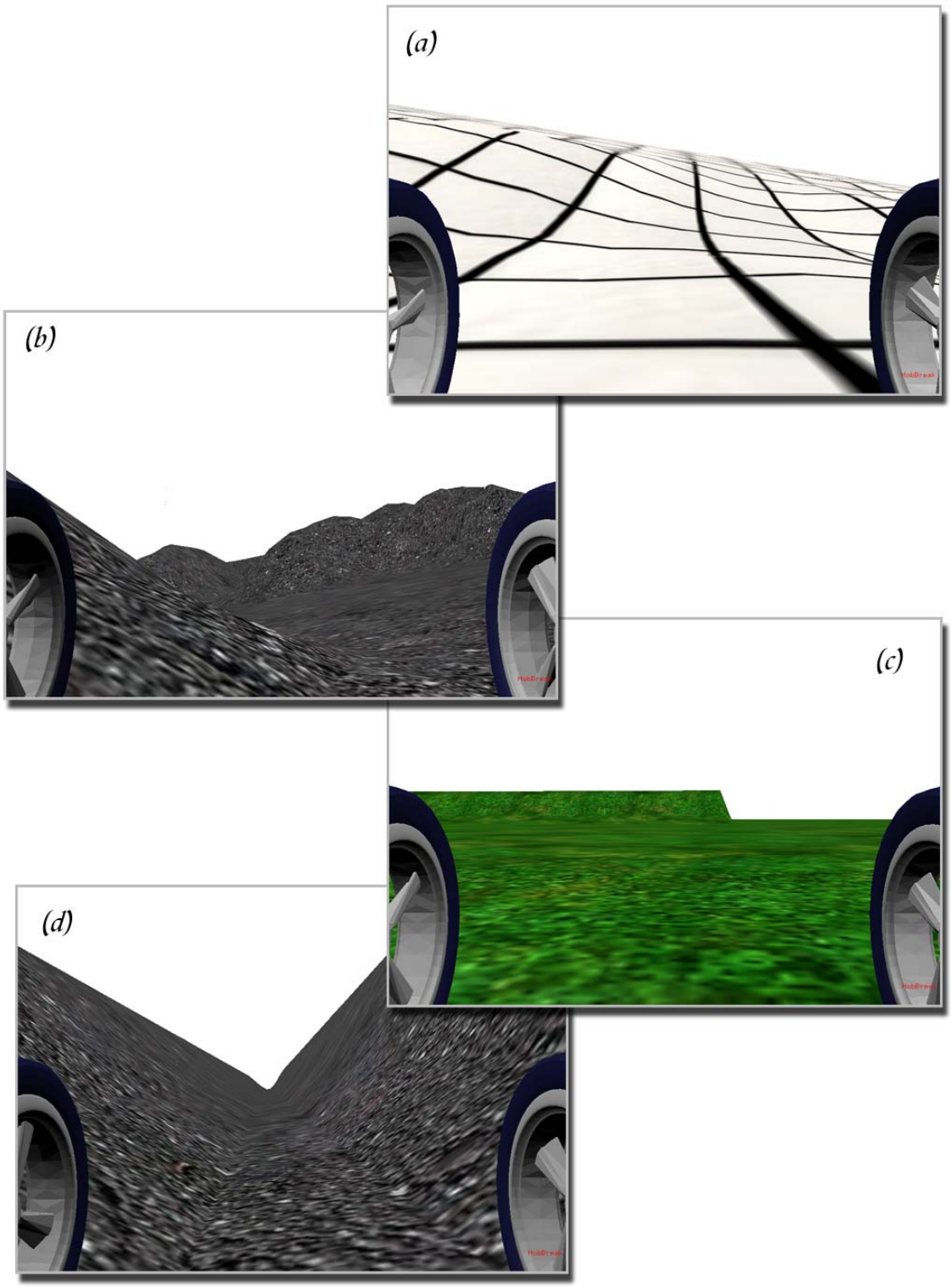


Figura 68 – Visão pela câmera virtual interna do veículo robótico durante várias simulações com diferentes tipos de terrenos

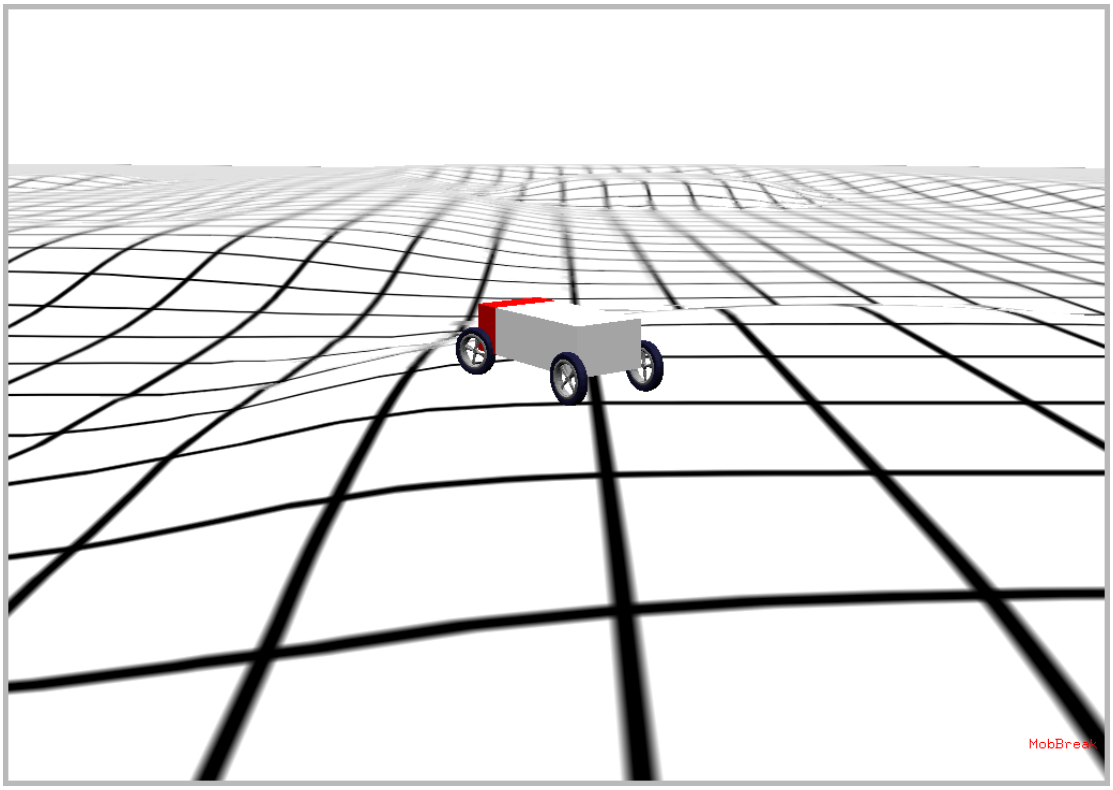


Figura 69 – Imagem obtida pela câmera virtual externa que acompanha o veículo robótico

D.3. Visualizar legendas

A tecla "**L**" (ou "**l**") torna as legendas visíveis ou invisíveis.

D.4. Encerrar (parar) simulação corrente

A tecla "**P**" (ou "**p**") interrompe a execução da simulação corrente e cessa toda a dinâmica e controle do veículo. Para iniciar uma nova simulação, pressione a tecla "**R**" (ou "**r**").

D.5. Iniciar nova simulação

A tecla "**R**" (ou "**r**") inicia uma nova simulação levando em conta os dados iniciais definidos no script de simulação. Essa tecla pode ser pressionada a qualquer momento durante a execução do aplicativo.

D.6. Salvar simulação corrente em vídeo

A tecla “**S**” (ou “**s**”) salva o histórico de frames da simulação em um arquivo no formato AVI. Há a opção de escolher entre vários tipos de formatos de acordo com os codificadores e decodificadores instalados no computador (Figura 70). O simulador pergunta qual é o CoDec do vídeo a ser utilizado antes de gerar o arquivo AVI (Figuras 71 a 73).

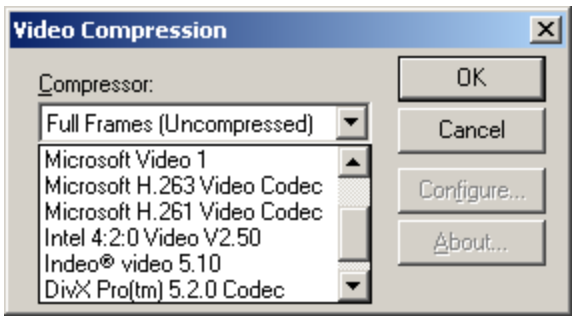


Figura 70 – Janela de seleção do CoDec para geração do vídeo AVI da simulação

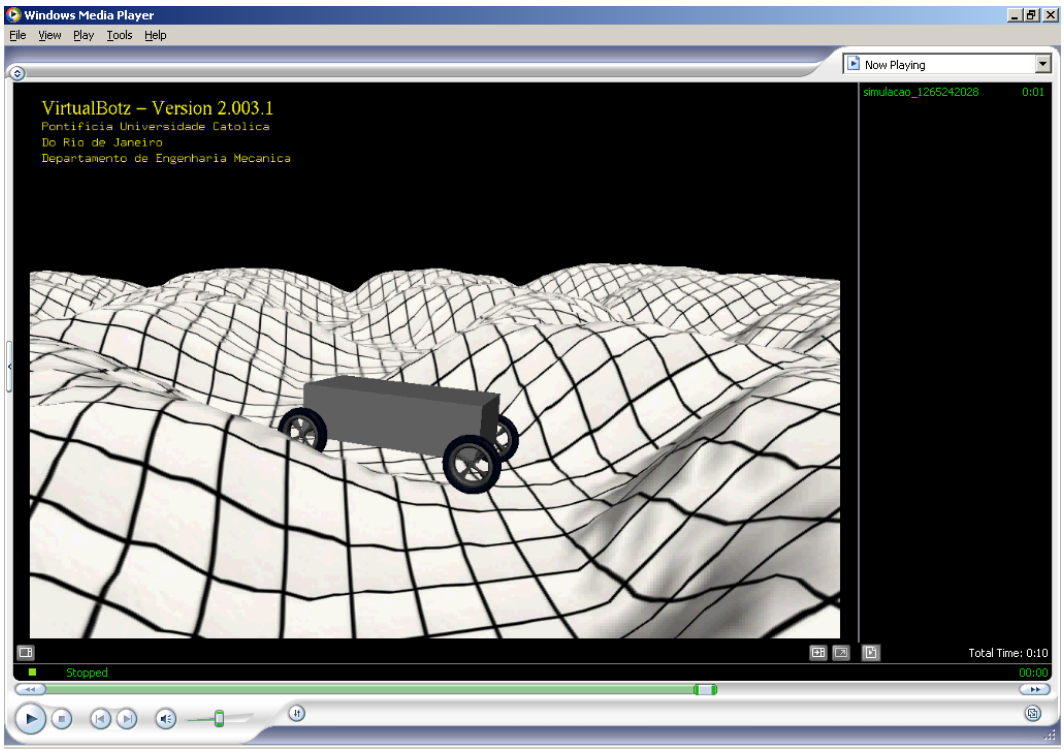


Figura 71 – Imagem de um vídeo AVI gerado com a visão do observador, externa ao veículo

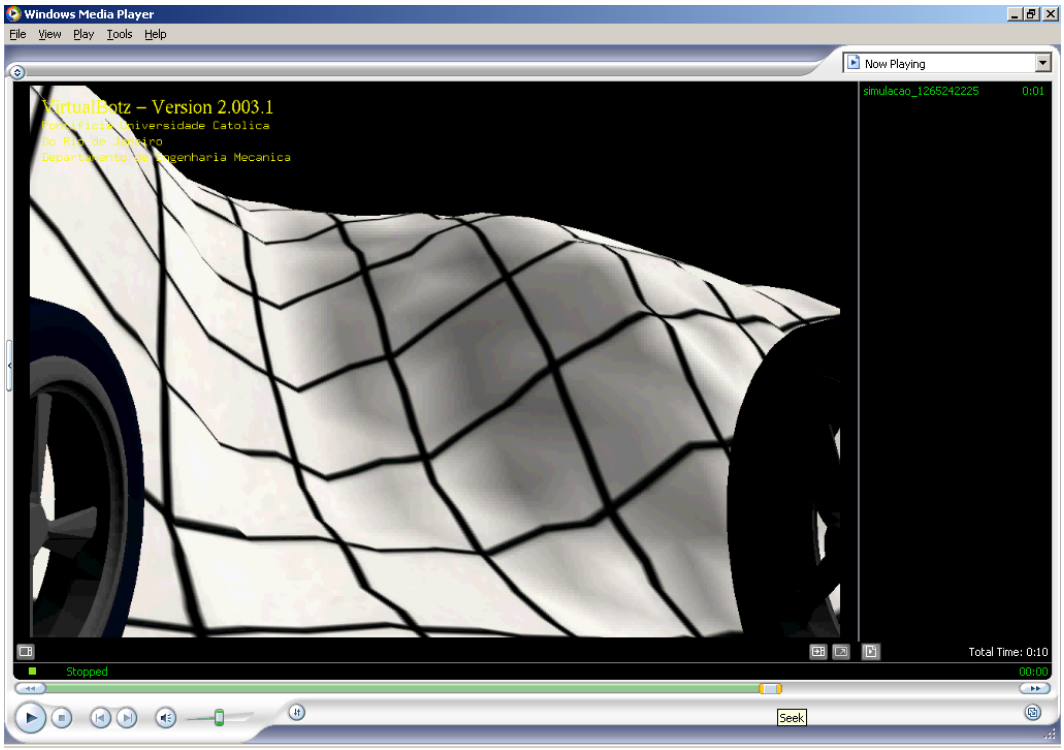


Figura 72 – Imagem de um vídeo AVI gerado com a câmera virtual interna do veículo robótico

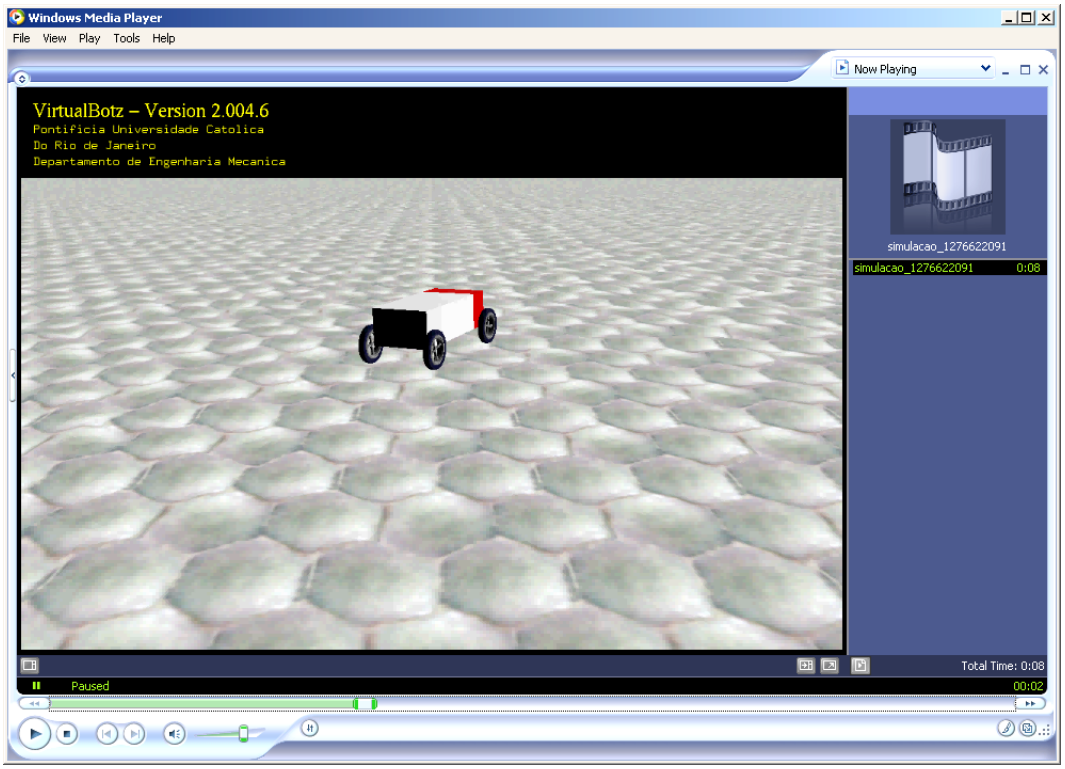


Figura 73 – Imagem de um vídeo AVI gerado com a visão da terceira câmera virtual do aplicativo VirtualBotz 3D, em uma simulação com o tipo de piso similar ao dos pilotis da PUC-Rio

D.7. Visualizar trajetória do centro de massa do veículo robótico após a simulação

A tecla "**T**" (ou "**t**") visualiza as últimas posições do centro de massa do veículo robótico após a interrupção da simulação. Pressionar sucessivamente essa tecla alterna o estilo de apresentação para uma das três opções: vetor ***Z***, vetores ***X***, ***Y*** e ***Z*** ou linha. A quantidade de posições é decorrente da definição da quantidade máxima de *frames* no histórico, predefinido com 2.000 *frames*. Para guardar mais *frames*, basta alterar o atributo "histórico máximo de frames" do script de simulação (**Anexo E**). A seguir estão alguns exemplos dos estilos de apresentação obtidos ao pressionar a tecla "**T**" (Figuras 74 a 76):

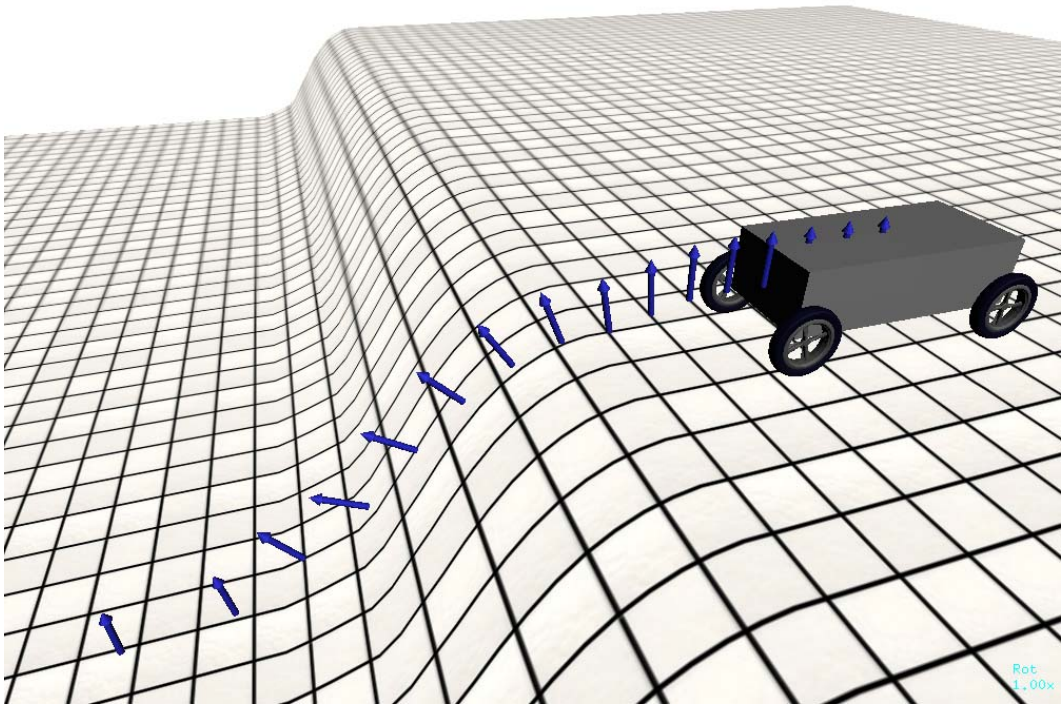


Figura 74 – Veículo robótico com os vetores ***Z*** do centro de massa após o término da simulação

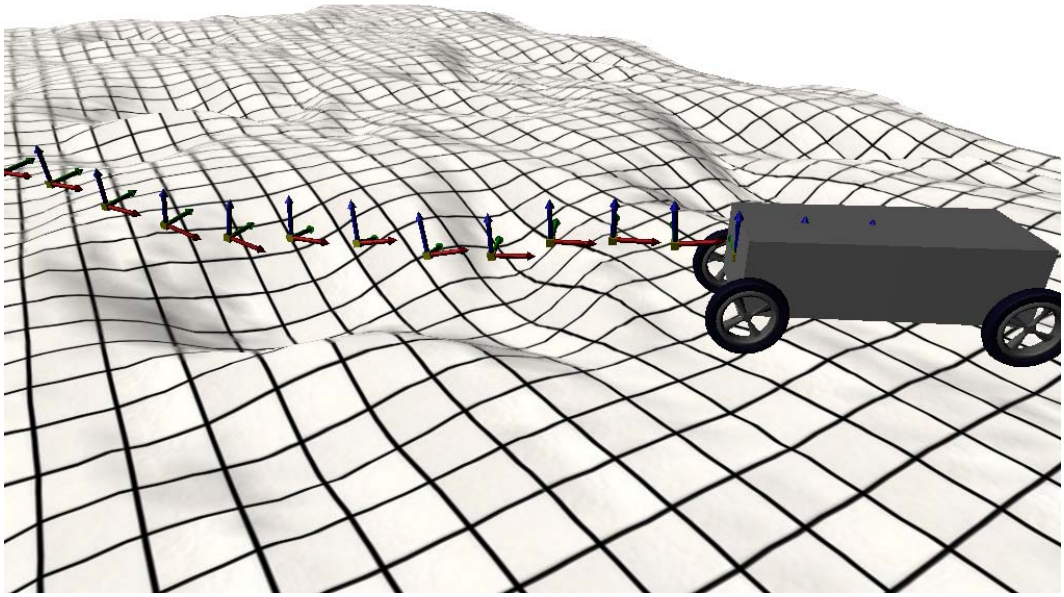


Figura 75 – Veículo robótico com os vetores ***X***, ***Y*** e ***Z*** do centro de massa após o término da simulação

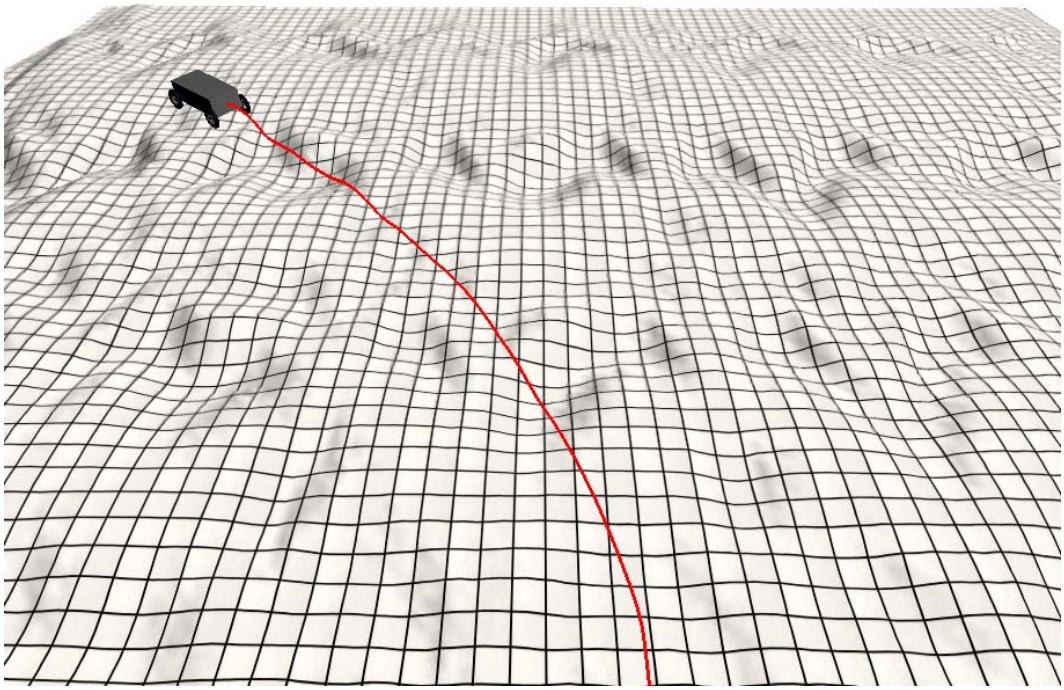
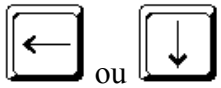


Figura 76 – Veículo robótico com o centro de massa representado por uma linha contínua, após o término da simulação

D.8. Posicionar o veículo robótico pelo centro de massa ao longo da trajetória após a simulação

 ou

A tecla seta "**para esquerda**" ou "**para baixo**" permite o retorno do veículo robótico às posições do centro de massa guardado no histórico ao término da simulação (Figura 77).

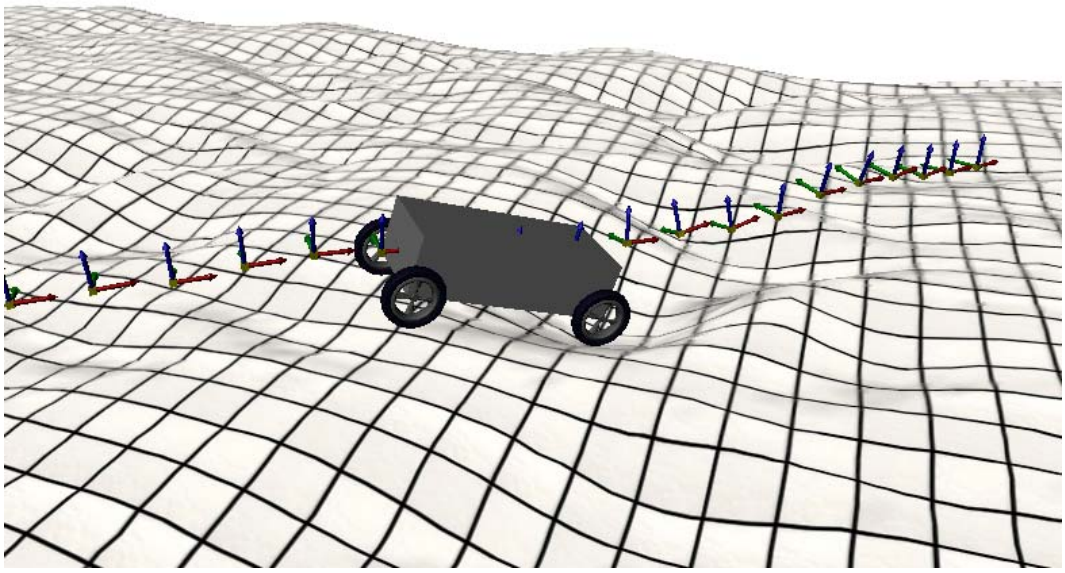


Figura 77 – Veículo robótico retornando aos centros de massa anteriores ao do término da simulação

 ou 

A tecla seta "**para cima**" ou "**para a direita**" permite avançar o veículo robótico às posições do centro de massa guardadas no histórico ao término da simulação (Figura 78).

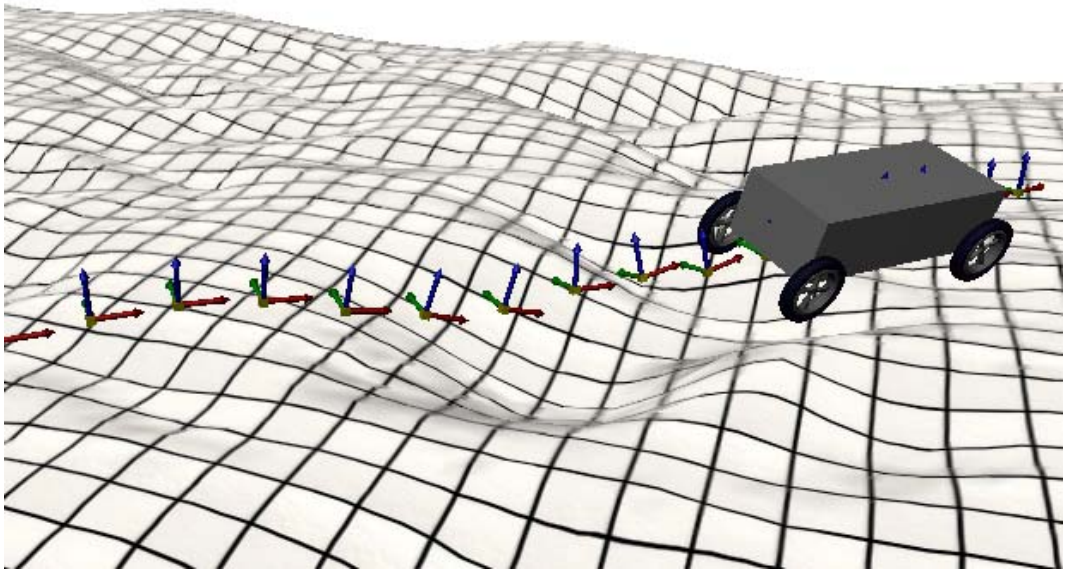


Figura 78 – Veículo robótico avançando às posições dos centros de massa até o término da simulação

D.9. Alterar escala de rotação e translação do mundo virtual

A tecla "**sinal de menor**" permite diminuir a escala de rotação e translação do observador, de 0,05 a 4,00.

A tecla "**sinal de maior**" permite aumentar a escala de rotação e translação do observador, de 0,05 a 4,00.

Anexo E:
Scripts de Configuração do Simulador

Como mencionado na parte “Descrição Geral” do simulador, o software é composto de dois arquivos scripts em formato textual simples, localizados na pasta “Scripts”. É importante salientar que os nomes de atributos não usam nenhum caractere especial, tampouco recursos gráficos linguísticos, como acentuação gráfica ou outros quaisquer. Portanto, o nome de atributo deve conter apenas caracteres textuais simples.

No primeiro arquivo, o principal, de nome “main_script.txt”, indica-se qual arquivo de configuração será utilizado para a simulação. A seguir são mostrados o arquivo principal e a chamada do arquivo de simulação (ou configuração) na linha em negrito. Observe que é possível conservar várias linhas de arquivos de simulação, desde que estejam comentadas com o símbolo “#” (denominado “tralha” ou “jogo da velha”) na primeira coluna de cada linha.

O simulador permite a alteração de qualquer dado dos arquivos scripts e posterior execução sem que seja necessário terminar a execução do programa. Para isso basta fazer a alteração dos dados, salvar o arquivo e teclar “**R**” (ou “**r**”) no simulador.

```
#
## main_script.txt
##
##   DESENVOLVIDO POR : Ricardo Morrot Lima
##              e-MAIL : morrot@gmail.com
##
##               CURSO : Mestrado em Engenharia Mecânica
##             MATÉRIA : Dissertação de Mestrado
##          ORIENTADOR : Prof. Marco Antonio Meggiolaro
##
##                 ANO : 2008-2009
##
## Este arquivo de script é o primeiro a ser lido a cada execução
```

```
## do aplicativo ou a cada restart (tecla "R") durante a
## execução do aplicativo.
##
## Aqui dentro apenas está definido o nome do próximo arquivo
## script que contém todas as definições do terreno, veículo
## robótico, controle e método numérico. Dessa forma, poderão ser
## executadas ao mesmo tempo várias instâncias do aplicativo com
## definições diferentes de terrenos, etc.
##
## Evite deixar qualquer tipo de caractere, inclusive o espaço em
## branco, no final das linhas.
##
## O nome do arquivo script de simulação não deve conter
## caracteres em branco. Devem-se evitar também caracteres ditos
## especiais como cedilhas, acentos, etc.
##
## A última linha deste arquivo não deve conter nada,
## obrigatoriamente.
#
######################################################################
## Configuração da Placa de vídeo ###################################
# Sincronismo Vertical
#     desligado - quando a desempenho é mais importante do que a
#                 qualidade da imagem
#     ligado    - prioridade na qualidade da imagem
sincronismo vertical = desligado
#
######################################################################
#########################
script de simulacao = mobilerobot_test.txt
#script de simulacao = mobilerobot_test_-_rough_terrain.txt
#
## Scripts de comparação do MATLAB com o C++
#
#script de simulacao = mr_test_-_flat_terrain_MATLABVersusC++.txt
#script de simulacao = mr_test_-_rough_terrain_MATLABVersusC++.txt
#script de simulacao = mr_test_-_flat_terrain.txt
#script de simulacao = mr_test_-_heightmap_rough_terrain.txt
#script de simulacao = mr_test_-_heightmap_1_rough_terrain.txt
```

No arquivo de simulação/configuração, não se deve alterar nada além do necessário. É preferível salvar o arquivo com um novo nome e indicá-lo no arquivo principal a alterar e salvar e/ou inserir vários comentários, pois polui o arquivo.

Em nenhum momento é permitida a troca na ordem dos atributos do arquivo de simulação. Isso pode gerar inconsistência de dados e consequente erro de leitura de dados ou erro na execução da simulação.

A seguir é apresentado um exemplo de arquivo (script) de simulação.

```
## mobile_robot_test.txt : Arquivo de configuração para simulação
##
##   DESENVOLVIDO POR : Ricardo Morrot Lima
##              e-MAIL : morrot@gmail.com
##
##               CURSO : Mestrado em Engenharia Mecânica
##             MATÉRIA : Dissertação de Mestrado
##          ORIENTADOR : Prof. Marco Antonio Meggiolaro
##
##                 ANO : 2008-2009
##
## Este arquivo de script deve permanecer na ordem em que se
## encontra.
## Ao definir novos valores para os atributos, preste atenção
## para não gerar inconsistência de dados com esses valores.
##
## Evite deixar qualquer tipo de caractere, inclusive espaço em
## branco, no final das linhas.
##
## Importante!
##
##   - Os nomes de quaisquer arquivos, como por exemplo os
##     de malha, modelos OBJ, texturas, não devem conter espaço em
##     branco ou qualquer outro caractere gráfico linguístico,
##     como acentuação, entre outros.
```

```
#
#####################################################################
## Título do script (entre aspas)
titulo = "Teste de subida na rampa terrain_mesh2.msh"
#
## Dados do Terreno ################################################
#
# Tipos possíveis para o terreno
# 1 - Rubber Asphalt Dry ( borracha em asfalto seco )
# 2 - Rubber Asphalt Wet ( borracha em asfalto molhado )
# 3 - Rubber Concrete Dry ( borracha em concreto seco )
# 4 - Rubber Concrete Wet ( borracha em concreto molhado )
# 5 - Dry Sand ( areia seca )
# 6 - Sandy Loam ( terreno arenoso, mais pra areia)
# 7 - Clayey Soil ( terreno argiloso, de barro)
# 8 - Snow ( neve )
# 9 - Washed Sand ( "areia lavada", sem materiais orgânicos e sal )
# 10 - Dried Bentonite Clay ( um determinado terreno argiloso seco )
# 11 - Compacted Topsoil ( terra batida, compactada )
#
tipo do terreno = 3
#
# Coordenadas extremas do terreno
vmin = { -40.0, -40.0, 0.0 }
vmax = { 40.0, 40.0, 0.0 }
#
# Discretização do terreno
pontos = 200.0
#
# Textura do terreno
# Importante!
# - O arquivo de textura do terreno deve estar na pasta
#   (diretório) "Textures\".
# - Caso não utilize nenhuma textura digite "none", sem aspas,
#   para o atributo "texture file".
# - Anti-aliasing do terreno deve ser "yes" ou "no", sem aspas.
anti-aliasing = yes
#texture file = none
texture file = solo2.bmp
#
# Tipos possíveis de terreno: senos, flat, mesh, image
```

```
tipo do terreno = image
#
# Apenas para mesh ----------------------------------
# Se o terreno não for lido de um arquivo de malha
# comente as linhas a seguir.
# Importante!
# - O arquivo do mapa de altura deve estar na pasta
#   (diretório) "Obj_files\".
#mesh file = terrain1.msh
#
# Apenas para image (mapa de altura) -----------------
# Se o terreno não for lido de um arquivo de imagem
# comente as linhas a seguir.
# Importante!
# - O arquivo do mapa de altura deve estar na pasta
#   (diretório) "Heightmaps\".
image file = heightmap52.bmp
altura da escala de cinza = 1.0
##
######################################################################
## Dados dos Motores #################################################
##
## Cada roda pode ter um atuador motor com características
## específicas ou todas podem ter o mesmo tipo de atuador motor
## com as mesmas características. Entretanto, o número de
## atuadores motor não pode ultrapassar a quantidade de rodas
## estipuladas logo acima.
##
## - Para definir atuadores motor de mesma marca com diferentes
##   especificações, basta colocar nomes diferentes, por exemplo:
##       motor = 0
##       nome, modelo e versao = Magmotor S28-150 ver1.0
##       ...
##       motor = n
##       nome, modelo e versao = Magmotor S28-150 ver1.2
##
## - Os atributos abaixo serão os mesmos para todos os atuadores
##   motor definidos nesta seção.
##
## Regras:
##
```

```
##   1) O primeiro atuador motor deve começar em zero.
##   2) Não pode ser utilizado o caractere espaço em branco
##      no nome do modelo.
#
# ------------------------------------------------
motor = 0
nome do modelo e versao = Magmotor_S28-150
torque constante Kt[Nm/A] = 0.03757
velocidade constante Kv[(rad/s)/V] = 26,61698
resistencia total do motor Rmotor[Ohm] = 0.148
corrente Inoload[A] = 3.4
# Para limitar a potencia da bateria PN3600,
# fazer Imax = 80 e Vmax = 24
corrente maxima[A] = 80
tensao maxima[V] = 36
caixa de reducao (adimensional) = 7.14
#
#####################################################################
## Dados do Veículo Robótico ########################################
robot opengl file name = chassi_2.obj
## Orientação inicial do veículo robótico
vetor n = { 1.0, 0.0, 0.0 }
vetor t = { 0.0, 1.0, 0.0 }
vetor b = { 0.0, 0.0, 1.0 }
## Posição inicial do veículo robótico
centro de massa = { -46.0, 0.0, 2.3 }
## Velocidade linear inicial
velocidade do centro de massa = { 0.0, 0.0, 0.0 }
## Velocidade angular
vetor w = { 0.0, 0.0, 0.0 }
## Centro de massa do veículo robótico
## Vetor centro de massa = { right, left, front, back, top, bottom }
right = 1.0
left = 1.0
front = 2.0
back = 2.0
top = 0.7
bottom = 0.5;
## Massa em quilos
massa = 120.0
quantidade de rodas = 4
```

```
quantidade de divisoes da roda = 18
rigidez do chassi = 1e4
amortecimento = 5e2
## Definição dos corners (quinas)
quantidade de corners = 8
## Posição dos corners na coordenada n-t-b local ----------------
corner = 0
ntb = { -back, -right, -bottom }
corner = 1
ntb = { front, -right, -bottom }
corner = 2
ntb = { front, left, -bottom }
corner = 3
ntb = { -back, left, -bottom }
corner = 4
ntb = { -back, -right, top }
corner = 5
ntb = { front, -right, top }
corner = 6
ntb = { front, left, top }
corner = 7
ntb = { -back, left, top }
#
## Rodas ---------------------------------------------------------
## Cada roda tem a sua configuração independente. Isso quer dizer
## que se no atributo "quantidade de rodas" em "Dados do Veículo
## Robótico" forem definidas seis rodas, as seis rodas deverão
## ser definidas uma a uma nesta seção.
##
## Regras:
##
##   1) A primeira roda deve começar em zero.
##   2) As coordenadas de cada roda devem seguir uma sequência
##      anti-horária. Essa sequência é fundamental para o controle
##      de estabilidade do veículo.
#
# ------------------------------------------------
roda = 0
nome do motor = Magmotor_S28-150
modelo da roda nome do arquivo = supertire_43.obj
rigidez = 1e4
```

```
amortecimento = 5e2
largura = 0.2
raio = 0.6
deslocamento h = 0.0
saturacao de h = 0.2
velocidade relativa da suspensao = 0.0
torque de saturacao = 30.0
# Torque constante: no ou yes (valendo o que está em torque,
# logo abaixo)
torque constante = no
torque = 0.0
# Deslizamento da roda: no ou yes
desliza = no
# Pneu
rigidez lateral do pneu = 3e4
rigidez longitudinal do pneu = 3e4
coordenadas local do centro de massa (x,y,z) = { -2.0, -1.1, -0.5 }
#
# ------------------------------------------------
roda = 1
nome do motor = Magmotor_S28-150
modelo da roda nome do arquivo = supertire_43.obj
rigidez = 1e4
amortecimento = 5e2
largura = 0.2
raio = 0.6
deslocamento h = 0.0
saturacao de h = 0.2
velocidade relativa da suspensao = 0.0
torque de saturacao = 30.0
# Torque constante: no ou yes (valendo o que está em torque,
# logo abaixo)
torque constante = no
torque = 0.0
# Deslizamento da roda: no ou yes
desliza = no
# Pneu
rigidez lateral do pneu = 3e4
rigidez longitudinal do pneu = 3e4
coordenadas local do centro de massa (x,y,z) = { 2.0, -1.1, -0.5 }
#
```

```
# ------------------------------------------------
roda = 2
nome do motor = Magmotor_S28-150
modelo da roda nome do arquivo = supertire_43.obj
rigidez = 1e4
amortecimento = 5e2
largura = 0.2
raio = 0.6
deslocamento h = 0.0
saturacao de h = 0.2
velocidade relativa da suspensao = 0.0
torque de saturacao = 30.0
# Torque constante: no ou yes (valendo o que está em torque,
# logo abaixo)
torque constante = no
torque = 0.0
# Deslizamento da roda: no ou yes
desliza = no
# Pneu
rigidez lateral do pneu = 3e4
rigidez longitudinal do pneu = 3e4
coordenadas local do centro de massa (x,y,z) = { 2.0, 1.1, -0.5 }
#
# ------------------------------------------------
roda = 3
nome do motor = Magmotor_S28-150
modelo da roda nome do arquivo = supertire_43.obj
rigidez = 1e4
amortecimento = 5e2
largura = 0.2
raio = 0.6
deslocamento h = 0.0
saturacao de h = 0.2
velocidade relativa da suspensao = 0.0
torque de saturacao = 30.0
# Torque constante: no ou yes (valendo o que está em torque,
# logo abaixo)
torque constante = no
torque = 0.0
# Deslizamento da roda: no ou yes
desliza = no
```

```
# Pneu
rigidez lateral do pneu = 3e4
rigidez longitudinal do pneu = 3e4
coordenadas local do centro de massa (x,y,z) = { -2.0, 1.1, -0.5 }
#
#####################################################################
## Controle ########################################################
# Timer sempre em milissegundos
timer = 50
torque Tal = 30.0      # torque Tal constante de torque das rodas
#
#####################################################################
# Métodos numéricos (Euler) ########################################
delta T = 0.001
#
# A próxima opção de configuração guarda uma grande quantidade
# de dados durante a simulação. Não utilize valores muito altos
# para o máximo de frames. Isso pode gerar trocas (swap) de
# memória com o disco rígido, tornando a simulação lenta e com
# efeito quebradiço, ou seja, com pausas.
# Quando o máximo de frames é igual a zero, o aplicativo é
# informado de que a simulação otimiza a memória e não guarda
# um histórico com muitos registros de dados, apenas o
# necessário para a simulação em tempo real.
#
#historico maximo de frames = 0
#historico maximo de frames = 630
#historico maximo de frames = 830
#historico maximo de frames = 1000
historico maximo de frames = 2000
#historico maximo de frames = 4000
#historico maximo de frames = 6150
#historico maximo de frames = 10000
## o limite abaixo é o máximo permitido(evite!)
#historico maximo de frames = 2147483647
#
#####################################################################
# Geração de vídeo, do tempo no histórico da simulação, em
# formato AVI ###################
#
# - O nome do arquivo NÃO DEVE conter ponto, extensão, espaço em
```

```
#   branco ou qualquer outro caractere gráfico linguístico,
#   como acentuação, entre outros.
# - O arquivo gerado será gravado na pasta "Videos".
#
nome do arquivo AVI = simulacao
#
# - Caso o atributo "timestamp", logo abaixo, esteja "yes", todos
#   os arquivos gerados conterão em seu nome a data e a hora
#   (timestamp), facilitando o armazenamento de várias visões para
#   a mesma simulação. Se o atributo estiver "no", então o arquivo
#   será reescrito.
timestamp = yes
#
# - O frame rate ideal para arquivos AVI é na faixa de 15 a 25
#   frames por segundo, inclusive.
frame rate = 25
#
######################################################################
# Graph ##############################################################
# Você pode especificar um arquivo com coordenadas XYZ em
# sequência. Esse arquivo será mostrado como uma linha gráfica.
# Do contrário, basta colocar "none" para que nenhum gráfico seja
# mostrado.
# É imprescindível que esse arquivo esteja na pasta (diretório)
# "testes\".
XYZ file stream = "none"      ## nenhum arquivo de gráfico será mostrado
#XYZ file stream = "MATLAB_xCM_mobile_robot.txt"
#XYZ file stream = "MATLAB_xCM_mobile_robot_flat.txt"
cor da linha = { 1.0, 0.0, 0.0 }          ## cor da linha em RGB
##
##
######################################################################
# Geração de arquivos de dados #######################################
#
# - O nome do arquivo NÃO DEVE conter ponto, extensão, espaço em
#   branco ou qualquer outro caractere gráfico linguístico, como
#   acentuação, entre outros.
# - Os arquivos gerados serão gravados na pasta "Report".
# - Para desligar a saída dos dados, basta digitar a palavra "off"
#   no início da linha referente ao dados a ser desligado. Para
#   ligar, basta digitar "on".
```

```
#
on,centro de massa do veiculo = mobile_xCM.txt
on,centro de massa das rodas do veiculo = wheels_xCM.txt
off,torque das rodas = wheels_torques.txt
off,sinal de controle das rodas = wheels_ControlSign.txt
##
## FIM
```

Anexo F:
Opções de Terrenos

Há quatro formas de configurar um terreno para simulação:

- Senos

Senos (Figura 79) é um formato predefinido a partir de equações senoidais para gerar um terreno ondulado. Para mudar a característica desse terreno, deve-se acessar o arquivo “CVBTerrain_.cpp” e mudar a equação, como visto a seguir:

```
void CVRTerrainSines::createMesh(void)
{
   unsigned int ix, jy;

   for (jy = 0; jy < getDimY(); jy++)
      for (ix = 0; ix < getDimX(); ix++)
      {
         setHeight(jy, ix, (0.3 * (1.0+sin(jy*PI/10.0)*1.0)*(1.0+sin(ix*PI/10.0)*1.0)) );
//       setHeight(jy, ix, (0.3 * (1.0+sin(jy*PI/20.0)*1.0)) );
//       setHeight(jy, ix, (0.3 * (1.0+sin(jy*PI/20.0)*1.0)*(1.0+sin(ix*PI/20.0)*1.0)) );
      }
}
```

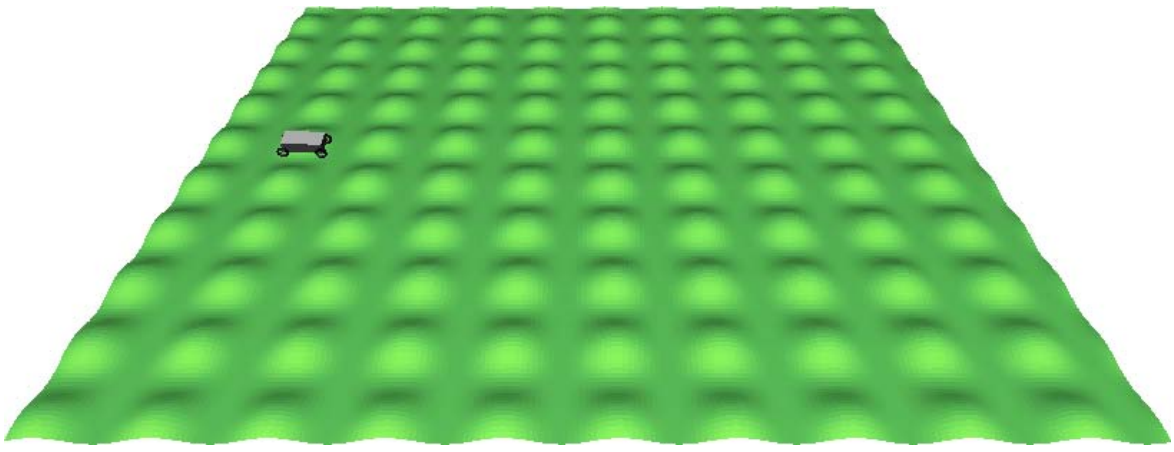


Figura 79 – Exemplo de opção de terreno senos

- *Flat*

Flat é um formato predefinido que permite a seleção de um terreno plano horizontal, sem nenhuma ondulação (Figura 80).

Figura 80 – Exemplo de opção de terreno *flat*

- *Mesh*

Mesh é um formato de malha que, especificamente para o terreno, deve ser aberta. O importante dessa malha para o sistema é que as coordenadas de cada vértice estejam numa grade regular retangular, ou seja, formem polígonos regulares retangulares (Figura 81). Dessa forma, o sistema recupera o arquivo de malhas com todos os vértices. As dimensões *dX* e *dY* entre os vértices devem ser iguais para esse tipo de malha de terreno (Figura 81).

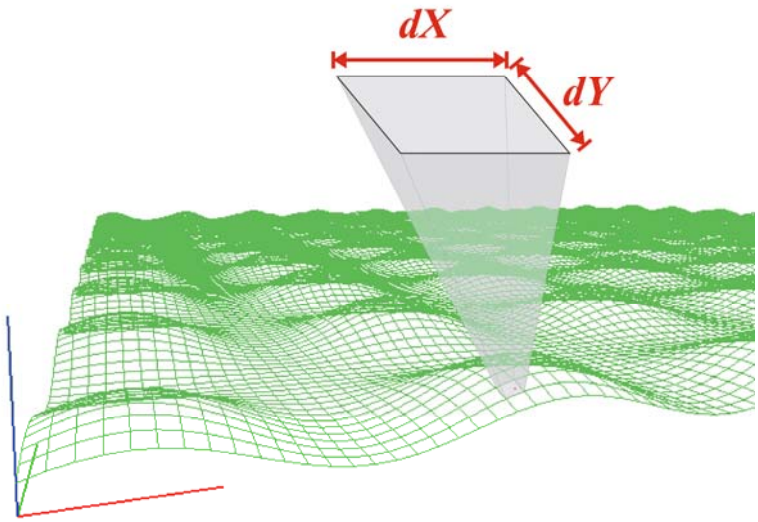


Figura 81 – Ilustração do terreno de grade regular retangular

O arquivo contendo a malha deve ser do tipo texto, contendo na primeira linha o número total de vértices e, a partir da segunda linha, começando em zero, o número do vértice e as coordenadas *x*, *y* e *z*, todos separados por espaço em branco. Veja o exemplo abaixo:

Arquivo “terrain_mesh.msh”.

```
40401
0 -40.000000 -40.000000 0.000000
1 -40.000000 -39.600000 0.000000
2 -40.000000 -39.200000 0.000000
3 -40.000000 -38.800000 0.000000
4 -40.000000 -38.400000 0.000000
5 -40.000000 -38.000000 0.000000
6 -40.000000 -37.600000 0.000000
7 -40.000000 -37.200000 0.000000
8 -40.000000 -36.800000 0.000000
9 -40.000000 -36.400000 0.000000
10 -40.000000 -36.000000 0.000000
11 -40.000000 -35.600000 0.000000
12 -40.000000 -35.200000 0.000000
13 -40.000000 -34.800000 0.000000
14 -40.000000 -34.400000 0.000000
15 -40.000000 -34.000000 0.000000
16 -40.000000 -33.600000 0.000000
17 -40.000000 -33.200000 0.000000
18 -40.000000 -32.800000 0.000000
19 -40.000000 -32.400000 0.000000
...
40390 40.000000 36.000000 0.000000
40391 40.000000 36.400000 0.000000
40392 40.000000 36.800000 0.000000
40393 40.000000 37.200000 0.000000
40394 40.000000 37.600000 0.000000
40395 40.000000 38.000000 0.000000
40396 40.000000 38.400000 0.000000
40397 40.000000 38.800000 0.000000
40398 40.000000 39.200000 0.000000
40399 40.000000 39.600000 0.000000
40400 40.000000 40.000000 0.000000
```

A malha dentro do arquivo deve seguir uma ordenação de linha por colunas (Figuras 82 e 83) de modo a evitar uma sobrecarga de processamento desnecessário para reordenamento toda vez que for aberto um arquivo para leitura e renderização. O reordenamento do arquivo, caso ocorra, será sempre pelas coordenadas X_{min}/Y_{min} e X_{max}/Y_{max}.

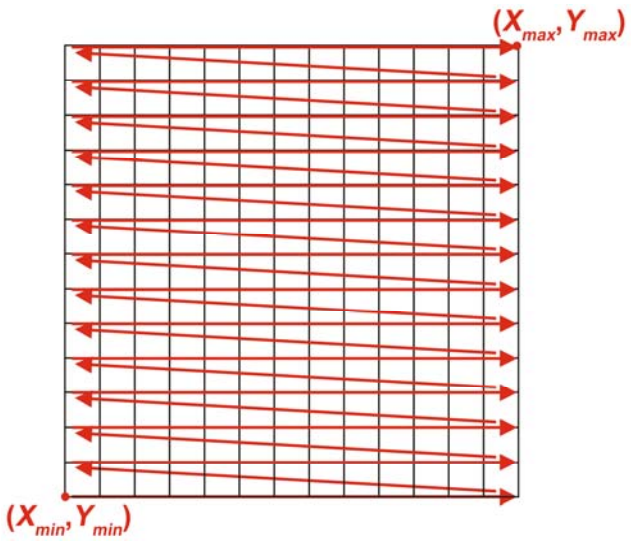


Figura 82 – Exemplo da ordenação da malha do terreno

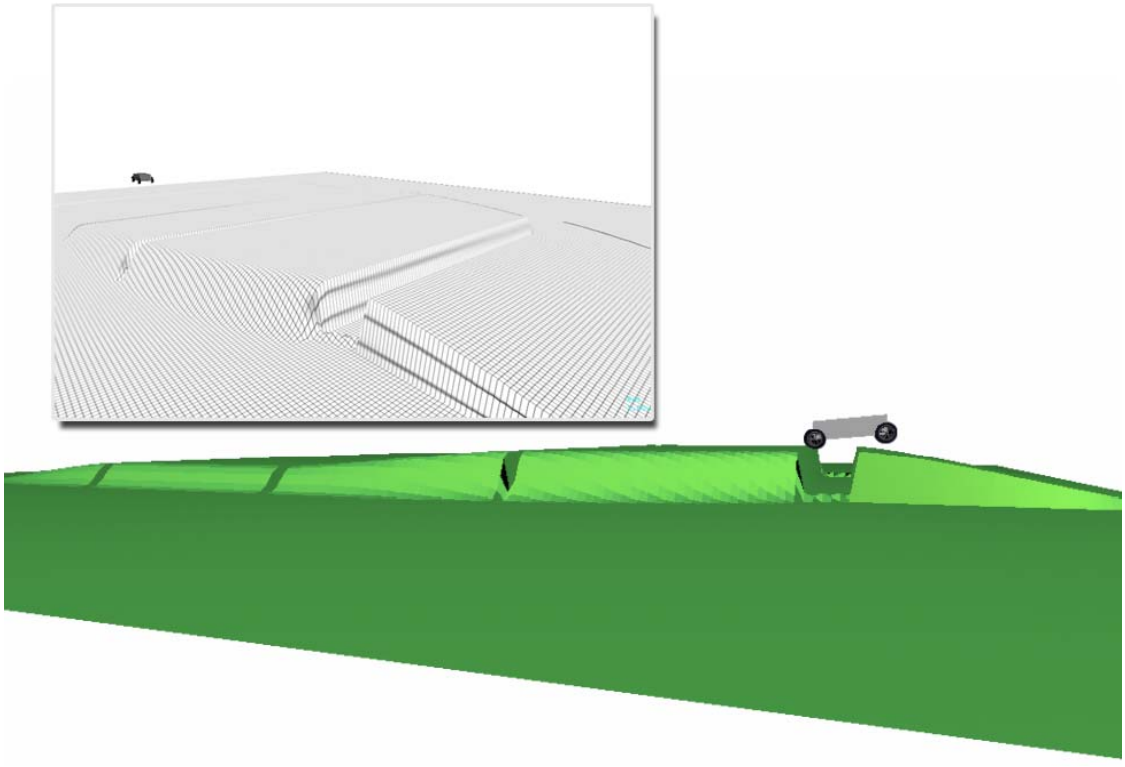


Figura 83 – Simulação utilizando a leitura de terreno do tipo *mesh* (arquivo de malha "terrain_mesh2.msh")

- *Image*

Image é uma opção de leitura de terreno a partir de uma imagem gráfica em escala de cinza, ou seja, um mapa de altura. Nessa opção há a possibilidade de definir a altura máxima da escala. Exemplos são mostrados nas Figuras 84, 85 e 86.

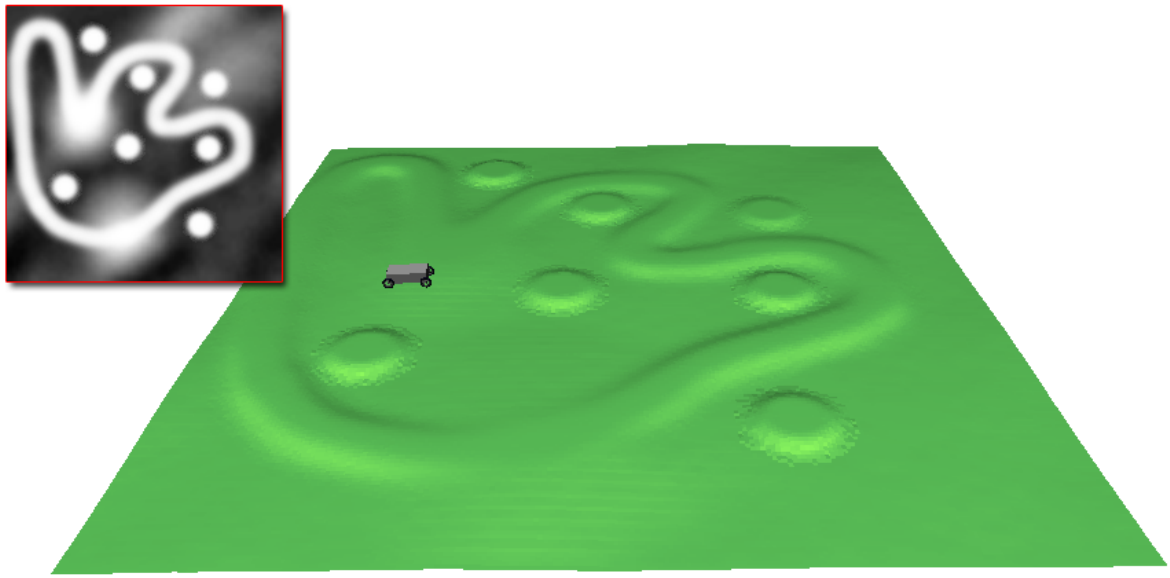


Figura 84 – Exemplo de opção de terreno *image* (mapa de altura) com o arquivo "heightmap3.bmp" definido para altura máxima "1.0"

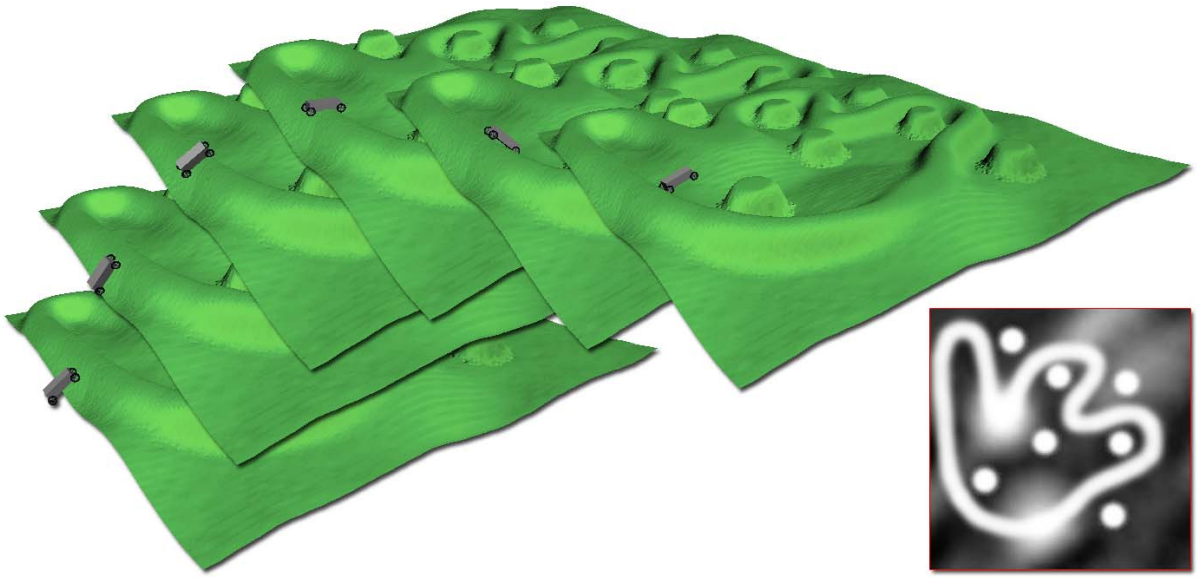


Figura 85 – Exemplo de opção de terreno *image* (mapa de altura) com o arquivo "heightmap3.bmp" definido para altura máxima "5.0", com o veículo robótico superando o aclive do terreno em diferentes instantes

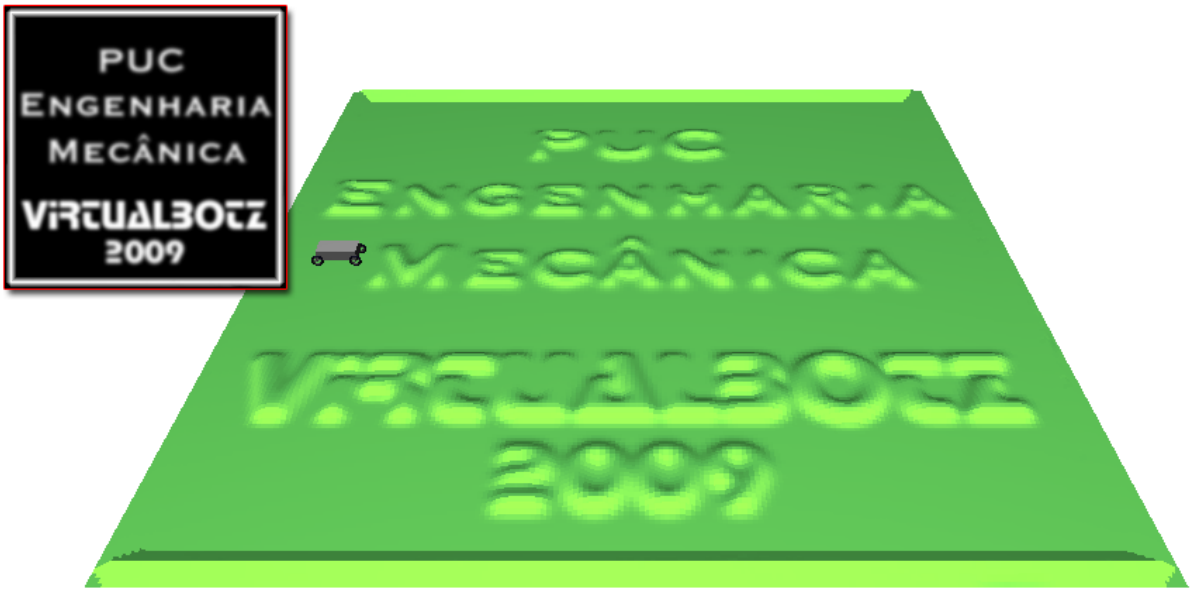


Figura 86 – Exemplo de opção de terreno *image* (mapa de altura) com o arquivo “heightmap52.bmp” definido para altura máxima “1.0”

Anexo G:
Configurando Opção de Terreno no Script da Simulação

No script corrente, logo a seguir, o terreno está selecionado como *image* (mapa de altura) e, na área de opções do mapa de altura, as linhas com o nome do arquivo e a escala para altura máxima não estão comentadas (sem o símbolo "#"). O mesmo procedimento serve para a opção *mesh*. Já no caso de uma das opções de terreno senos ou *flat*, devem-se comentar as linhas correspondentes às opções de *mesh* e *image*, com o símbolo "#" posicionado na primeira coluna de cada linha.

```
...
#
#####################################################################
## Dados do Terreno #################################################
#
# Coordenadas extremas do terreno
vmin = { -40.0, -40.0, 0.0 }
vmax = { 40.0, 40.0, 0.0 }
#
# Discretização do terreno
pontos = 200.0
#
# Textura do terreno
# Importante!
# - O arquivo de textura do terreno deve estar na pasta
#   (diretório) "Textures\".
# - Caso não utilize textura alguma, digite "none", sem aspas,
#   para o atributo "texture file".
# - Anti-aliasing do terreno deve ser "yes" ou "no", sem aspas.
anti-aliasing = yes
#texture file = none
texture file = solo2.bmp
#
# Tipos possíveis de terreno : senos, flat, mesh, image
```

```
tipo do terreno = image
#
# Apenas para mesh ----------------------------------
# Se o terreno não for lido de um arquivo de malha,
# comente as linhas a seguir.
# Importante!
# - O arquivo do mapa de altura deve estar na pasta
#   (diretório) "Obj_files\".
#mesh file = terrain_mesh.msh
#
# Apenas para image (mapa de altura) -----------------
# Se o terreno não for lido de um arquivo de imagem,
# comente as linhas a seguir.
# Importante!
# - O arquivo do mapa de altura deve estar na pasta
#   (diretório) "Heightmaps\".
image file = heightmap52.bmp
altura da escala de cinza = 1.0
#
########################################################################
...
```

Anexo H:
Renderização do Terreno

A renderização de terrenos é um assunto muito discutido na comunidade acadêmica. O objetivo desta dissertação não é definir um algoritmo novo de renderização de terrenos, muito menos entrar em detalhes sobre as diferentes técnicas de implementação existentes, e sim, em vista da quantidade de dados a ser administrada durante a execução do simulador, usar a estrutura que melhor se adéque [42].

O simulador leva em conta a discretização do terreno em uma série de pequenos intervalos para *x* e *y* de forma que a parte do algoritmo do simulador responsável pelo cálculo do ponto de contato das rodas do veículo robótico com o terreno calcule com exatidão a posição de contato. Quanto menor o intervalo, maior será a quantidade de pontos, ou vértices, e maior será o custo operacional do cálculo. Os exemplos expostos nos **Anexos F** e **G** apresentam terrenos de dimensão 80x80 com uma discretização de 200 pontos tanto para *x* quanto para *y*.

```
...
# Coordenadas extremas do terreno
vmin = { -40.0, -40.0, 0.0 }
vmax = { 40.0, 40.0, 0.0 }
#
# Discretização do terreno
pontos = 200.0...
...
```

Logo, tem-se uma matriz de 200x200 com 40.000 vértices com intervalos regulares de 0.40m. Para manter a fidelidade do terreno, evitando que haja perda na qualidade visual (Figura 87) e principalmente não venha a prejudicar a velocidade de cálculo do programa, foi utilizada a sequência de QUADS [42].

Se um vetor de índice para os vértices (*vertexes array*) fosse usado para primitivas GL_TRIANGLES, seriam gerados 237.606 elementos do tipo Glfloat.

Com o GL_QUADS, passa-se a ter 158.404 elementos. Esse cálculo começa a tomar proporções bem grandes quando levam-se em conta os vetores das normais, as coordenadas de textura e os vértices, com 40.000 elementos para cada vetor. Isso sem contar os vetores de vértices e normais com (*x*, *y*, e *z*) e o vetor de coordenadas de textura com (*s*, *t* e *r*).

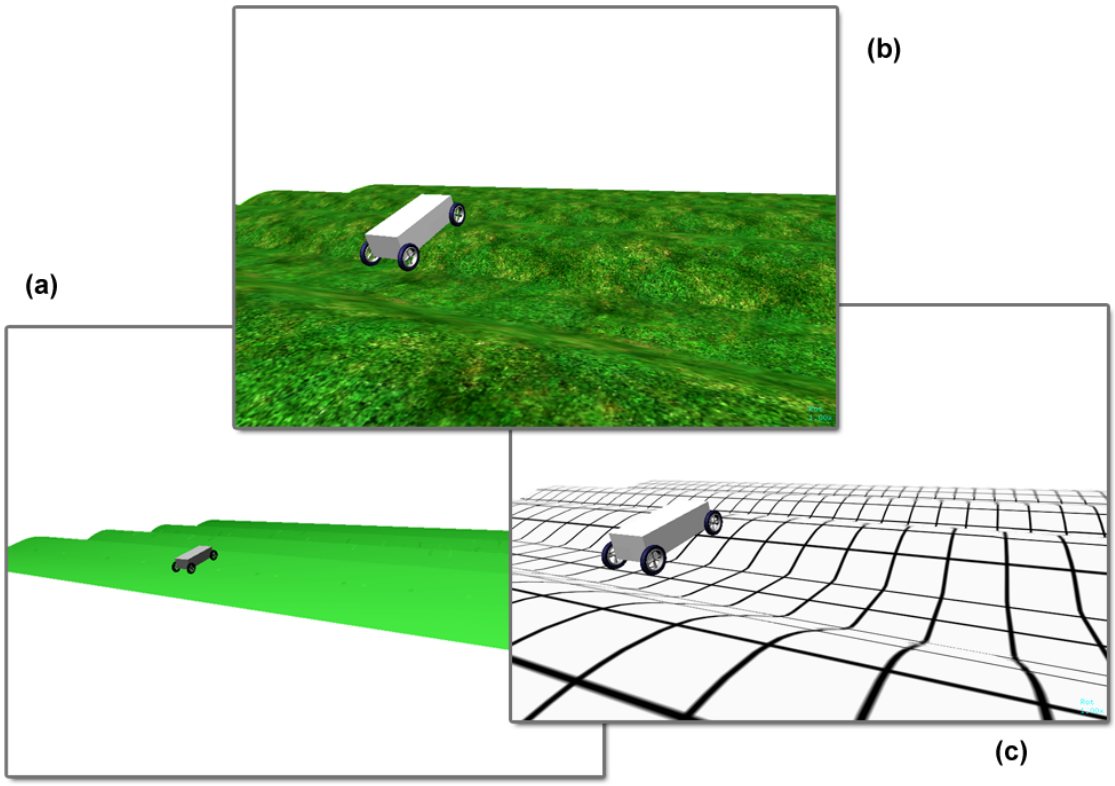


Figura 87 – As três simulações utilizam o mesmo terreno criado a partir de um mapa de altura (arquivo “heightmap63.bmp”), sendo a ilustração (a) sem nenhuma textura, com (b) textura “grass3.bmp” e (c) com “tile31.bmp”

Anexo I:
Configurando o Veículo Robótico no Script da Simulação

A configuração do veículo robótico é feita logo após a configuração do terreno (**Anexos F** e **G**). Devem-se levar em consideração vários fatores antes de começar a definir os valores dos atributos do veículo:

- orientação inicial
- posição inicial
- velocidade linear inicial
- velocidade angular
- centro de massa
- massa (kg)
- número de rodas
- número de divisões de cada roda (parte inferior)
- rigidez do chassi
- amortecimento
- número de quinas do chassi do veículo
- posição das quinas na coordenada ntb local

Imediatamente após as definições acima, é preciso definir as rodas, uma após a outra, iniciando por zero, até totalizar o valor definido para o atributo "quantidade de rodas" menos um. Seguem as definições para cada roda:

- número da roda (iniciando em zero)
- modelo da roda (nome do arquivo)
- rigidez
- amortecimento
- largura
- raio
- deslocamento da roda
- saturação do deslocamento

- velocidade relativa da suspensão
- torque de saturação
- torque constante (se a roda terá ou não um torque constante)
- torque (para o caso de a roda ter torque constante)
- deslizamento da roda
- coordenada local do centro de massa

Na apresentação visual do simulador VirtualBotz 3D, cada roda pode ter um modelo diferente ou apenas um único modelo para todas as rodas (desde que os seus atributos sejam iguais). Ao definir um modelo para uma roda, o programa verifica se ele já existe, evitando com isso a redundância de modelos. Todos os modelos devem permanecer na pasta "Obj_files" e devem ser do tipo wavefront (.OBJ) com malha triangular. Deve-se evitar o uso de modelos muito densos para não sobrecarregar o computador e, com isso, ocasionar perda de desempenho. Alguns fatores importantes que influenciam e melhoram consideravelmente o desempenho são grande quantidade de memória principal, boa placa gráfica com *chipset* NVidia ou ATI e uma quantidade razoável de memória da placa gráfica.

O modelo wavefront é muito usado em softwares de modelagem 3D, animação e efeitos visuais. É inclusive utilizado pelos softwares de modelagem 3D Alias|Wavefront Maya [44] e Blender [45], sendo este freeware. Os arquivos ".OBJ" são usados normalmente para guardar coordenadas 3D de objetos em três dimensões, mapas de texturas e outras informações úteis para padrão de imagens 3D. Dessa forma, os arquivos podem ser importados por vários programas de edição de imagem em três dimensões. Na Figura 88 tem-se um exemplo de roda desenhado no software Blender e salvo no formato wavefront, sempre com faces triangulares.

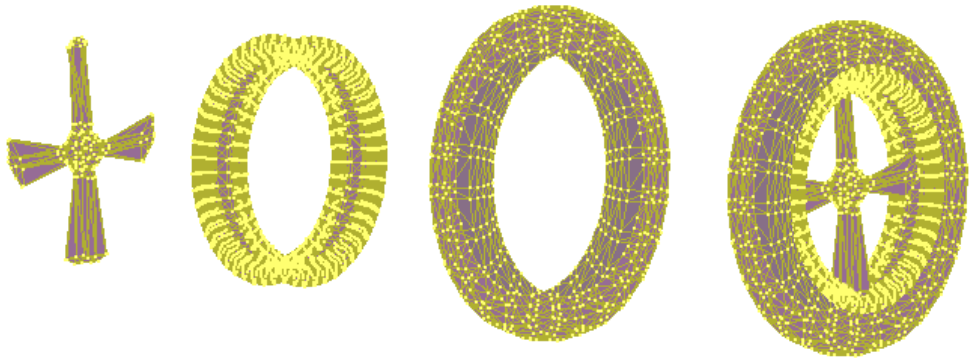


Figura 88 – Modelo de roda desenhado no software Blender com apresentação em vértices e arestas

Outro recurso implementado no simulador é a utilização de materiais coloridos para os modelos (Figura 89). Utilizando o mesmo formato de modelo (o wavefront), podem-se selecionar materiais de diferentes cores para cada grupamento de vértices/arestas e os salvar junto com os arquivos do wavefront. Os dados com materiais associados aos grupamentos de vértices/arestas ficarão no arquivo de extensão ".MTL".

Os arquivos ".MTL" são chamados biblioteca de materiais e contêm definições de um ou vários materiais, como por exemplo cor e textura. Esses arquivos são referenciados pelos arquivos wavefront e normalmente são salvos no formato ASCII. No caso de utilização de materiais texturizados para as rodas, como ilustrado no terreno, os arquivos de textura devem ficar na pasta "/*Textures*".

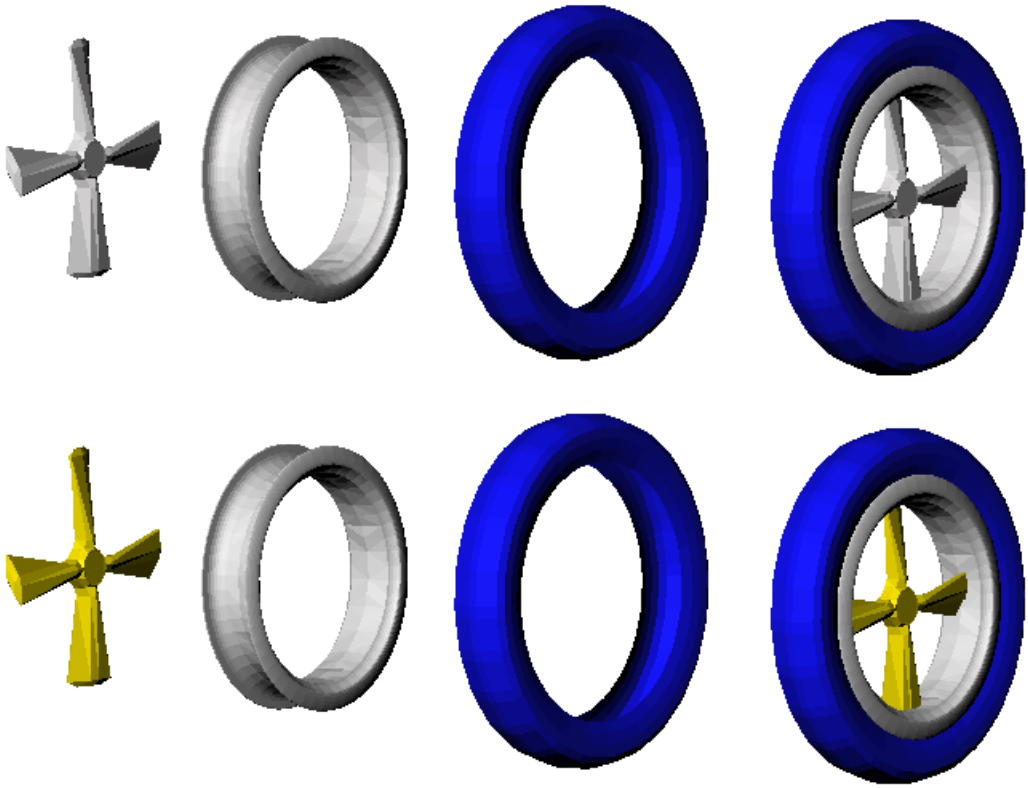


Figura 89 – Modelos de roda com definição dos materiais

O mesmo procedimento de modelagem visual da roda, descrito nos parágrafos anteriores, deve ser utilizado para o chassi do veículo robótico (Figuras 90 e 91). É importante salientar que o veículo utilizado neste estudo tem um formato de paralelepípedo, mas poderia ser tão ou mais elaborado que a roda mostrada acima. Na figura a seguir, há dois exemplos simples de chassi feitos no modelador geométrico Blender, ambos com faces triangulares.

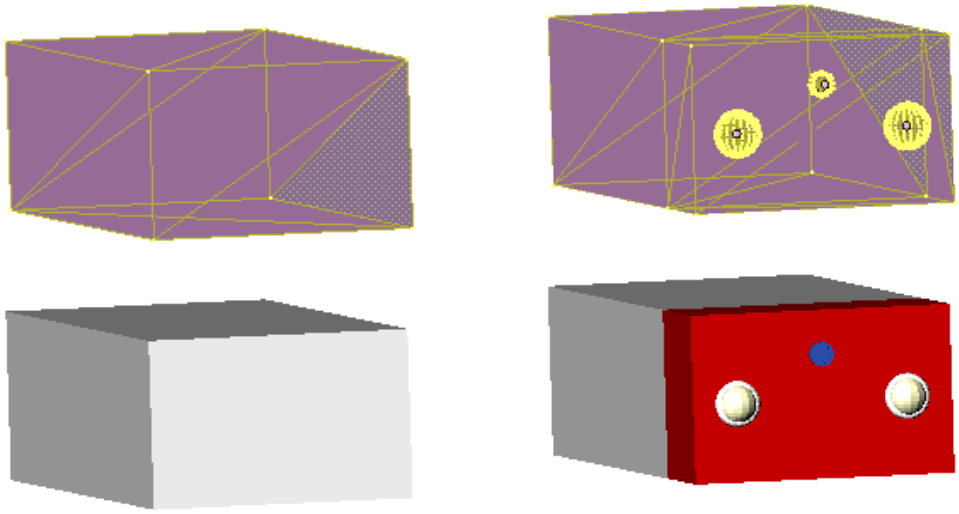


Figura 90 – Exemplos de chassi bem simples, ambos com poucos detalhes e malha triangular

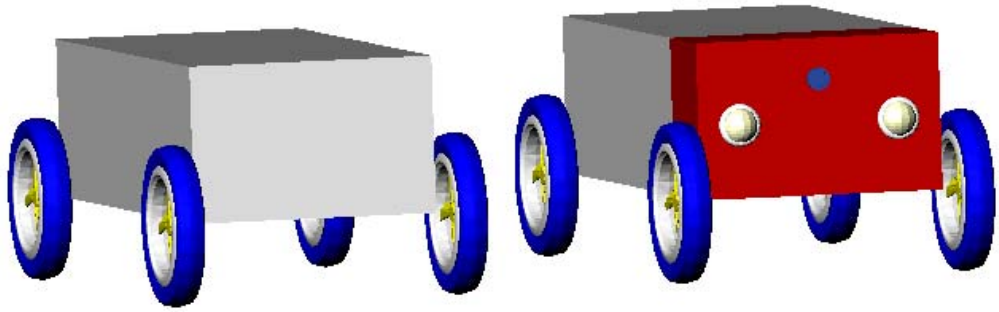


Figura 91 – Imagens da composição rodas mais chassi feita pelo simulador VirtualBotz 3D

Segue um trecho do script de simulação para configuração do veículo robótico e das rodas. O script integral encontra-se no **Anexo E**. Observe que, mesmo definindo rodas com materiais coloridos (linhas em negrito), basta informar o nome do arquivo modelo de formato wavefront (“.OBJ”), e o próprio programa se encarrega de procurar o arquivo de materiais (“.MTL”).

```
#
#####################################################################
## Dados do Veículo Robótico ########################################
robot opengl file name = chassi_2.obj
## Orientação inicial do veículo robótico
vetor n = { 1.0, 0.0, 0.0 }
vetor t = { 0.0, 1.0, 0.0 }
vetor b = { 0.0, 0.0, 1.0 }
## Posição inicial do veículo robótico
centro de massa = { -46.0, 0.0, 2.3 }
## Velocidade linear inicial
velocidade do centro de massa = { 0.0, 0.0, 0.0 }
## Velocidade angular
vetor w = { 0.0, 0.0, 0.0 }
## Centro de massa do veículo robótico
## Vetor centro de massa = { right, left, front, back, top, bottom }
right = 1.0
left = 1.0
front = 2.0
back = 2.0
top = 0.7
bottom = 0.5;
## Massa em quilos
massa = 120.0
quantidade de rodas = 4
quantidade de divisoes da roda = 18
rigidez do chassi = 1e4
amortecimento = 5e2
## Definição dos corners (quinas)
quantidade de corners = 8
## Posição dos corners na coordenada n-t-b local ----------------
corner = 0
ntb = { -back, -right, -bottom }
corner = 1
ntb = { front, -right, -bottom }
corner = 2
ntb = { front, left, -bottom }
corner = 3
ntb = { -back, left, -bottom }
corner = 4
```

```
ntb = { -back, -right, top }
corner = 5
ntb = { front, -right, top }
corner = 6
ntb = { front, left, top }
corner = 7
ntb = { -back, left, top }
#
## Rodas ----------------------------------------------------------
## Cada roda tem a sua configuração independente. Isso quer dizer
## que se no atributo "quantidade de rodas" em "Dados do Veículo
## Robótico" forem definidas seis rodas, as seis rodas deverão
## ser definidas uma a uma nesta seção.
##
## Regras:
##
##   1) A primeira roda deve começar em zero.
##   2) As coordenadas de cada roda devem seguir uma sequência
##      anti-horária. Essa sequência é fundamental para o controle
##      de estabilidade do veículo.
#
# -------------------------------------------------
roda = 0
nome do motor = Magmotor_S28-150
modelo da roda nome do arquivo = supertire_43.obj
rigidez = 1e4
amortecimento = 5e2
largura = 0.2
raio = 0.6
deslocamento h = 0.0
saturacao de h = 0.2
velocidade relativa da suspensao = 0.0
torque de saturacao = 30.0
# Torque constante: no ou yes (valendo o que está em torque,
# logo abaixo)
torque constante = no
torque = 0.0
# Deslizamento da roda: no ou yes
desliza = no
# Pneu
rigidez lateral do pneu = 3e4
```

```
rigidez longitudinal do pneu = 3e4
coordenadas local do centro de massa (x,y,z) = { -2.0, -1.1, -0.5 }
#
# ------------------------------------------------
roda = 1
nome do motor = Magmotor_S28-150
modelo da roda nome do arquivo = supertire_43.obj
rigidez = 1e4
amortecimento = 5e2
largura = 0.2
raio = 0.6
deslocamento h = 0.0
saturacao de h = 0.2
velocidade relativa da suspensao = 0.0
torque de saturacao = 30.0
# Torque constante: no ou yes (valendo o que está em torque,
# logo abaixo)
torque constante = no
torque = 0.0
# Deslizamento da roda: no ou yes
desliza = no
# Pneu
rigidez lateral do pneu = 3e4
rigidez longitudinal do pneu = 3e4
coordenadas local do centro de massa (x,y,z) = { 2.0, -1.1, -0.5 }
#
# ------------------------------------------------
roda = 2
nome do motor = Magmotor_S28-150
modelo da roda nome do arquivo = supertire_43.obj
rigidez = 1e4
amortecimento = 5e2
largura = 0.2
raio = 0.6
deslocamento h = 0.0
saturacao de h = 0.2
velocidade relativa da suspensao = 0.0
torque de saturacao = 30.0
# Torque constante: no ou yes (valendo o que está em torque,
# logo abaixo)
torque constante = no
torque = 0.0
```

```
# Deslizamento da roda: no ou yes
desliza = no
# Pneu
rigidez lateral do pneu = 3e4
rigidez longitudinal do pneu = 3e4
coordenadas local do centro de massa (x,y,z) = { 2.0, 1.1, -0.5 }
#
# ------------------------------------------------
roda = 3
nome do motor = Magmotor_S28-150
modelo da roda nome do arquivo = supertire_43.obj
rigidez = 1e4
amortecimento = 5e2
largura = 0.2
raio = 0.6
deslocamento h = 0.0
saturacao de h = 0.2
velocidade relativa da suspensao = 0.0
torque de saturacao = 30.0
# Torque constante: no ou yes (valendo o que está em torque,
# logo abaixo)
torque constante = no
torque = 0.0
# Deslizamento da roda: no ou yes
desliza = no
# Pneu
rigidez lateral do pneu = 3e4
rigidez longitudinal do pneu = 3e4
coordenadas local do centro de massa (x,y,z) = { -2.0, 1.1, -0.5 }
#
...
```

Anexo J:
Configurando os Motores das Rodas do Veículo Robótico no Script da Simulação

O aplicativo permite a configuração de *n* motores de corrente contínua (DC) no script da simulação, funcionando como uma folha de dados (*datasheet*) do fabricante do motor. Os dados reais inseridos nessa parte do aplicativo contribuem para a escolha do modelo ideal para o veículo robótico durante uma simulação. No entanto, é importante lembrar que, mesmo incluindo *n* tipos de motores, o simulador só permite a utilização de um motor por roda. Os atributos a serem configurados para cada motor são os seguintes:

- K_t – constante de torque do motor, em **Nm/A**
- K_v – constante de velocidade do motor, em **(rad/s)/V** (igual a $1/K_t$)
- R_{motor} – resistência elétrica do motor, em **Ohm**
- I_{noload} – corrente elétrica do motor, em ampéres (**A**)
- I_{max} – corrente máxima drenada pelo motor
- V_{max} – tensão elétrica máxima do motor
- caixa de redução (adimensional)
- potência máxima da bateria

Na simulação para este trabalho foi utilizado o modelo de motor **Magmotor S28-150**, mas o aplicativo permite a configuração de qualquer um dos motores mostrados na Tabela 11.

Tabela 11 – Exemplos de alguns dos dados dos fabricantes de motores [48]

				
Nome	**Etek**	**Magmotor S28-150**	**Magmotor S28-400**	**NPC T64 (w/gearbox)**
Tensão (V)	48	24	24	24
Potência (W)	11,185	2,183	3,367	834
Massa (kg)	9.4	1.7	3.1	5.9
Peso/Potência	1,190	1,284	1,086	141
I_{stall} / I_{no_load}	526	110	127	27
K_t (N·m/A)	0.13	0.03757	0.0464	0.86
K_v (RPM/V)	72	254	206	10
R_{motor} (Ω)	0.016	0.064	0.042	0.16
I_{no_load} (A)	5.7	3.4	4.5	5.5

Segue a parte do script de simulação que permite a configuração dos motores do veículo robótico. O script integral está no **Anexo E**.

```
...
#######################################################################
## Dados dos Motores ##################################################
##
## Cada roda pode ter um atuador motor com características
## específicas ou todas podem ter o mesmo tipo de atuador motor
## com as mesmas características. Entretanto, o número de atuadores
## motor não pode ultrapassar a quantidade de rodas estipuladas
## logo acima.
##
## - Para definir atuadores motor de mesma marca com diferentes
##   especificações, basta colocar nomes diferentes, por exemplo:
##       motor = 0
##       nome, modelo e versao = Magmotor S28-150 ver1.0
##       ...
##       motor = n
##       nome, modelo e versao = Magmotor S28-150 ver1.2
##
## - Os atributos abaixo serão os mesmos para todos os atuadores
##   motor definidos nesta seção.
##
## Regras:
##
```

```
##   1) O primeiro atuador motor deve começar em zero.
##   2) Não pode ser utilizado o caractere espaço em branco
##      no nome do modelo.
# ------------------------------------------------
motor = 0
nome do modelo e versao = Magmotor_S28-150
torque constante Kt[Nm/A] = 0.03757
velocidade constante Kv[(rad/s)/V] = 26,61698
resistencia total do motor Rmotor[Ohm] = 0.064
corrente Inoload[A] = 3.4
# Para limitar a potência da bateria PN3600,
# fazer Imax = 80 e Vmax = 24
corrente maxima[A] = 80
tensao maxima[V] = 36
caixa de reducao (adimensional) = 7.14
#
motor = 1
nome do modelo e versao = Magmotor_S28-400
torque constante Kt[Nm/A] = 0.0464
velocidade constante Kv[(rad/s)/V] = 21,5517
resistencia total do motor Rmotor[Ohm] = 0.042
corrente Inoload[A] = 4.5
# Para limitar a potência da bateria PN3600,
# fazer Imax = 80 e Vmax = 24
corrente maxima[A] = 80
tensao maxima[V] = 36
caixa de reducao (adimensional) = 7.14
...
#
## Rodas ----------------------------------------------------
...
roda = 0
nome do motor = Magmotor_S28-400
...
roda = 1
nome do motor = Magmotor_S28-150
...
roda = 2
nome do motor = Magmotor_S28-150
...
roda = 3
nome do motor = Magmotor_S28-400
```

Anexo K:
Habilitando no Script da Simulação a Geração de Dados da Simulação Corrente do Veículo Robótico em Arquivos

O aplicativo gera algumas saídas de dados em arquivos do tipo texto. Para a escolha dos tipos de dados, basta localizar no script a área “geração de arquivos de dados” e selecionar o necessário. Os nomes dos arquivos podem ser alterados, porém a gravação será sempre na pasta “Report”.

Tipos de saída de dados gerados pelo aplicativo VirtualBotz 3D:

- Posição do centro de massa do veículo robótico
- Posição do centro de massa das rodas do veículo robótico
- Torque das rodas
- Sinal de controle das rodas

```
...
######################################################################
# Geração de arquivos de dados ######################################
#
# - O nome do arquivo NÃO DEVE conter ponto, extensão, espaço em
#   branco ou qualquer outro caractere gráfico linguístico, como
#   acentuação, entre outros.
# - Os arquivos gerados serão gravados na pasta "Report".
# - Para desligar a saída dos dados, basta digitar a palavra "off"
#   no início da linha referente ao dado a ser desligado. Para
#   ligar, basta digitar "on".
#
off,centro de massa do veiculo = mobile_xCM.txt
off,centro de massa das rodas do veiculo = wheels_xCM.txt
on,torque das rodas = wheels_torques.txt
off,sinal de controle das rodas = wheels_ControlSign.txt
##
...
```

Os arquivos seguem um padrão para serem facilmente aplicados a programas gráficos como Grapher e similares ou planilhas gráficas como Microsoft Excel e similares. As Figuras 92, 93, 94 e 95 foram geradas no Microsoft Excel a partir de dados salvos durante uma simulação de um veículo robótico.

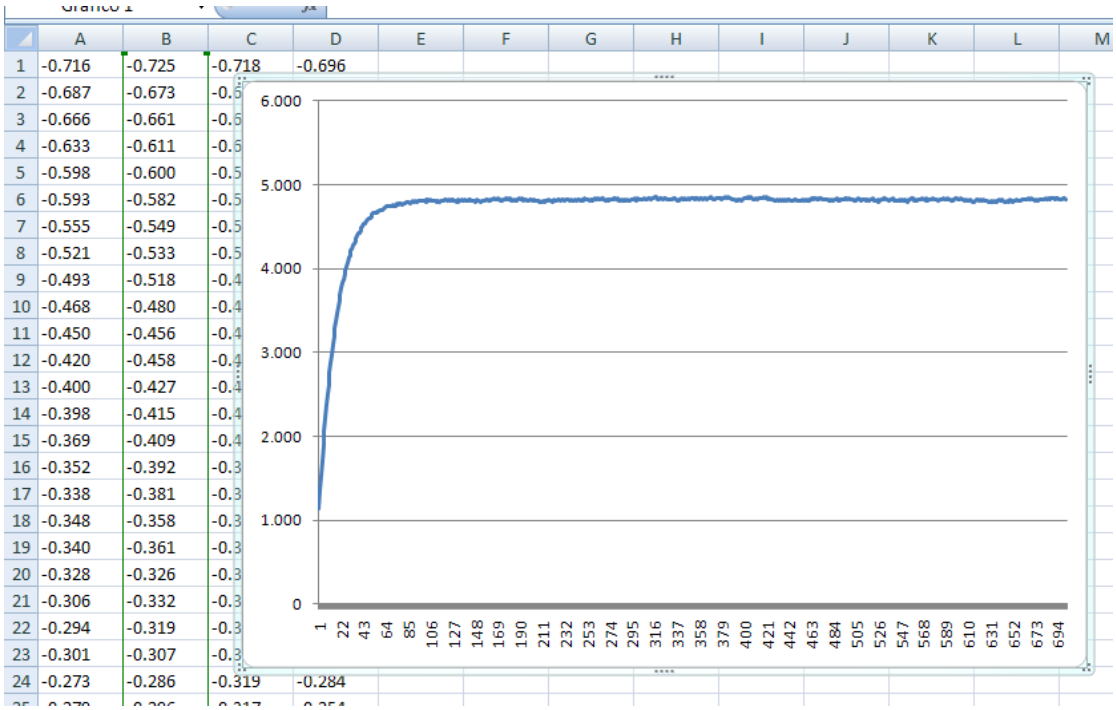


Figura 92 – Visualização em Microsoft Excel do gráfico gerado pelo torque de uma das rodas do veículo robótico em uma simulação

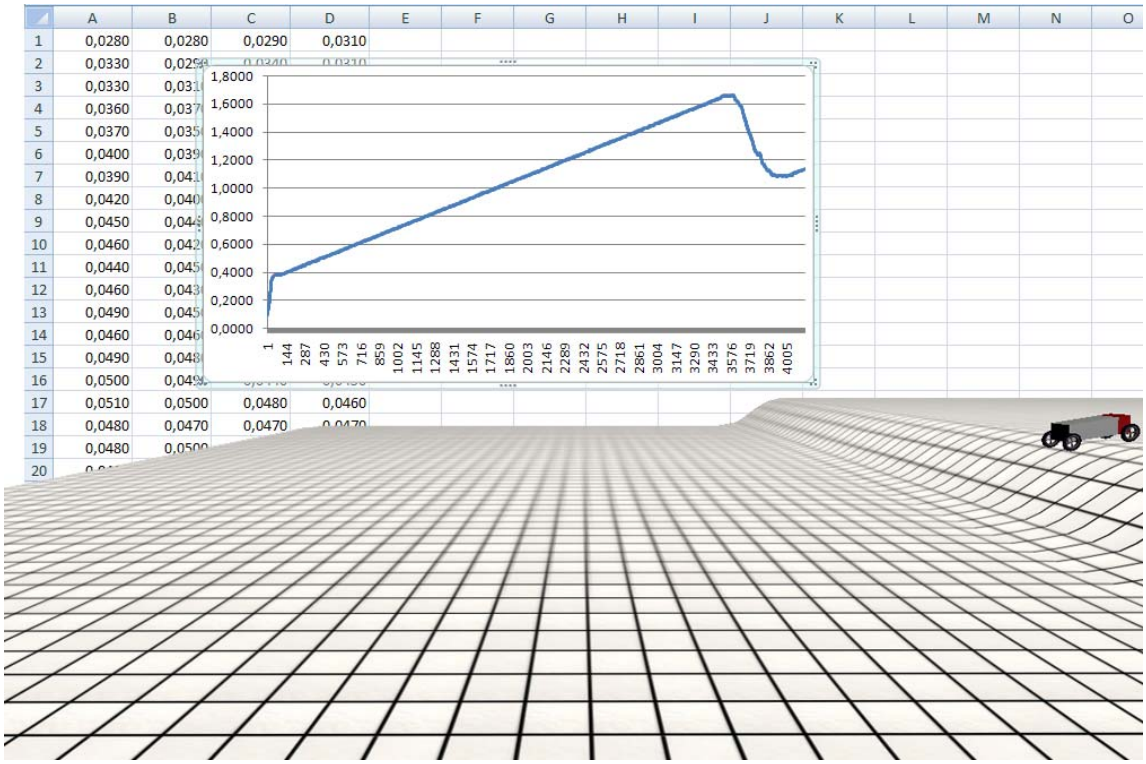


Figura 93 – Visualização em Microsoft Excel do gráfico do sinal de controle PID de uma das rodas do veículo robótico durante uma simulação

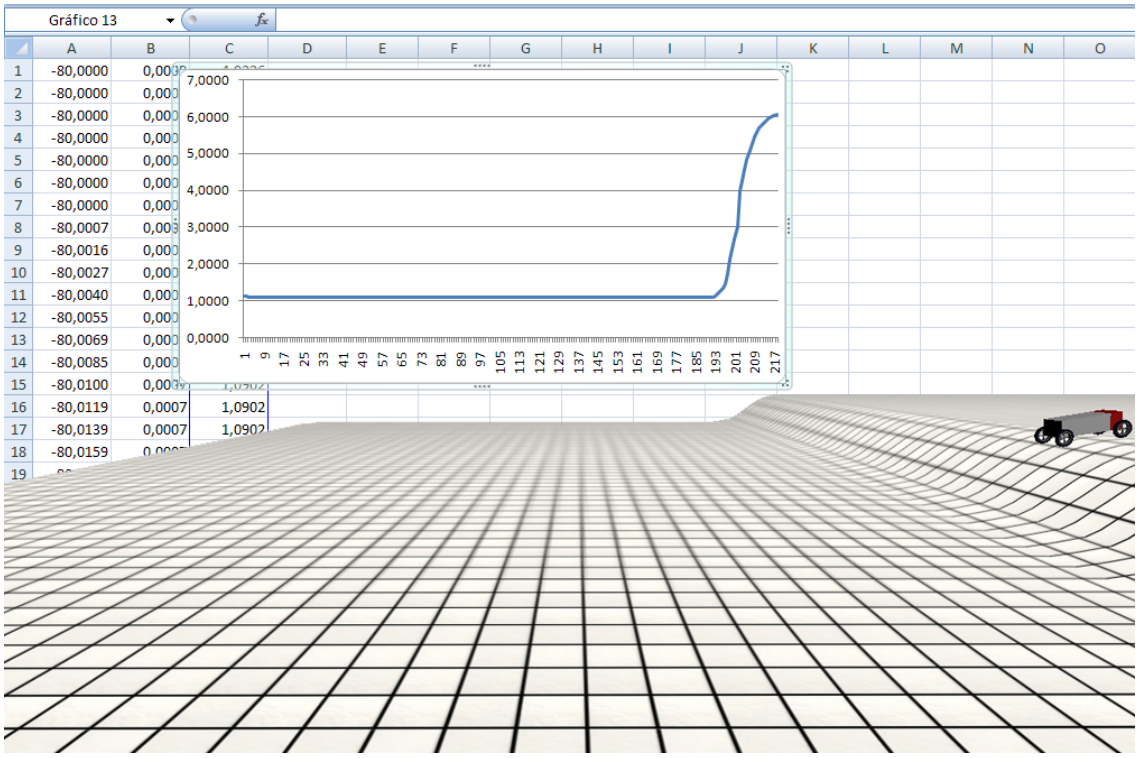


Figura 94 – Visualização em Microsoft Excel do gráfico do vetor ***z*** do centro de massa do veículo robótico em uma simulação

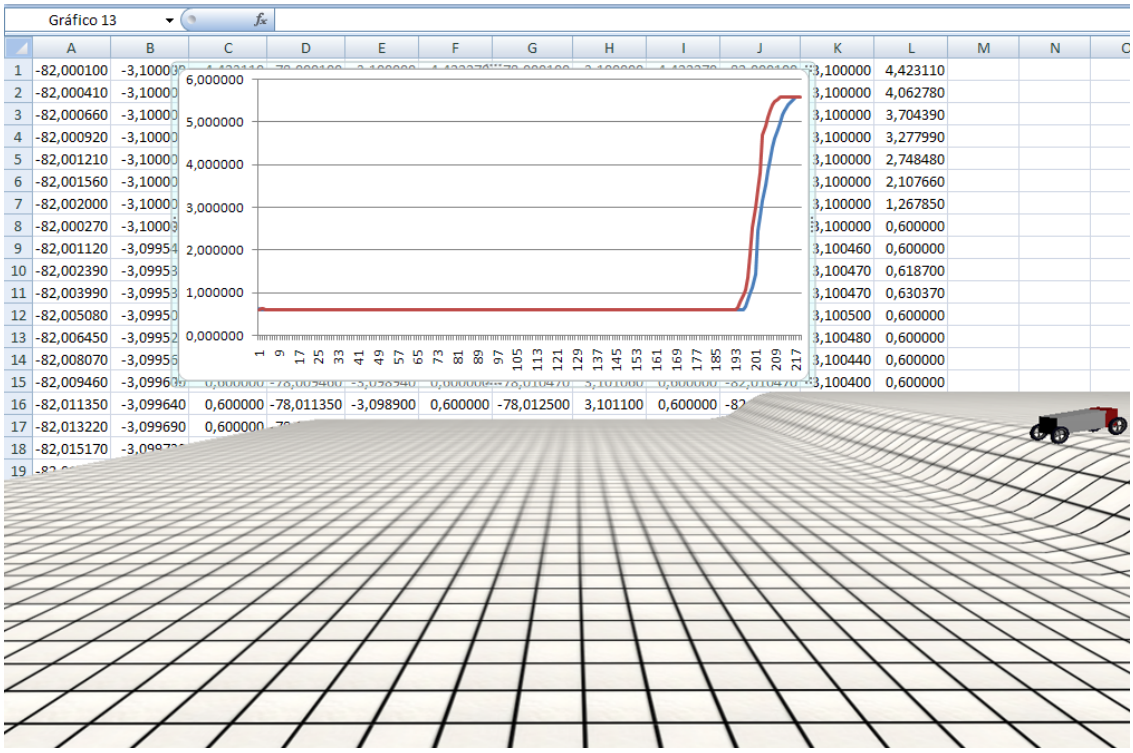


Figura 95 – Visualização em Microsoft Excel do gráfico do vetor ***z*** do centro de massa de uma das rodas dianteiras do veículo robótico, em vermelho, e uma das rodas traseiras, em azul, durante uma simulação