UNIVERSIDADE ESTADUAL DE CAMPINAS

Faculdade de Engenharia Mecânica

VINICIUS BENITES BASTOS

Desenvolvimento e integração de ambiente 3D de simulação para algoritmos de navegação por imagem

CAMPINAS

2015

VINICIUS BENITES BASTOS

Desenvolvimento e integração de ambiente 3D de simulação para algoritmos de navegação por imagem

Dissertação de Mestrado apresentada à Faculdade de Engenharia Mecânica da Universidade Estadual de Campinas como parte dos requisitos exigidos para obtenção do título de Mestre em Engenharia Mecânica, na Área de Mecânica dos Sólidos e Projeto Mecânico.

Orientador: Prof. Dr. Paulo Roberto Gardel Kurka

ESTE EXEMPLAR CORRESPONDE À VERSÃO FINAL DA DISSERTAÇÃO DEFENDIDA PELO ALUNO VINICIUS BENITES BASTOS, E ORIENTADA PELO PROF. DR. PAULO ROBERTO GARDEL KURKA.

.....

ASSINATURA DO ORIENTADOR

PROF. DR. PAULO ROBERTO GARDEL KURKA
Matrícula 228842
FEM/UNICAMP

CAMPINAS

2015

Agência(s) de fomento e nº(s) de processo(s): FAPEAM, 002/2014

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca da Área de Engenharia e Arquitetura
Elizangela Aparecida dos Santos Souza - CRB 8/8098

B297d

Bastos, Vinicius Benites, 1991-
Desenvolvimento e integração de ambiente 3D de simulação para algoritmos de navegação por imagem / Vinicius Benites Bastos. – Campinas, SP : [s.n.], 2015.

Orientador: Paulo Roberto Gardel Kurka.
Dissertação (mestrado) – Universidade Estadual de Campinas, Faculdade de Engenharia Mecânica.

1. Robótica. 2. Processamento de imagem. 3. Blender (Programa de computador). 4. Simulação. I. Kurka, Paulo Roberto Gardel,1958-. II. Universidade Estadual de Campinas. Faculdade de Engenharia Mecânica. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: Development and integration of 3D simulated environment for image based algorithms
Palavras-chave em inglês:
Robotics
Image processing
Blender (Computer program)
Simulation
Área de concentração: Mecânica dos Sólidos e Projeto Mecânico
Titulação: Mestre em Engenharia Mecânica
Banca examinadora:
Paulo Roberto Gardel Kurka [Orientador]
Niederauer Mastelari
Éric Rohmer
Data de defesa: 27-11-2015
Programa de Pós-Graduação: Engenharia Mecânica

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA MECÂNICA
COMISSÃO DE PÓS-GRADUAÇÃO EM ENGENHARIA MECÂNICA
DEPARTAMENTO DE SISTEMAS INTEGRADOS

DISSERTAÇÃO DE MESTRADO ACADEMICO

Desenvolvimento e integração de ambiente 3D de simulação para algoritmos de navegação por imagem

Autor: Vinicius Benites Bastos

Orientador: Prof. Dr. Paulo Roberto Gardel Kurka

A Banca Examinadora composta pelos membros abaixo aprovou esta Dissertação:

Prof. Dr. Paulo Roberto Gardel Kurka , Presidente
UNICAMP - FEM

Prof. Dr. Éric Rohmer
UNICAMP - FEEC

Prof. Dr. Niederauer Mastelari
UNICAMP - FEM

Campinas, 27 de novembro de 2015.

DEDICATÓRIA

Dedico este trabalho a minha família pelo exemplo na vida de nunca desistir, pelo incentivo, que me impulsionou durante todos os momentos do mestrado, pelo amor e carinho incondicionais, e por me ensinarem que o conhecimento é um ‘bem’ que nunca me será tirado.

AGRADECIMENTOS

Primeiramente, agradeço aos meus pais e minha avó Maria, que me deram apoio e me incentivaram durante todo o período do mestrado.

Menciono, também, meus sinceros agradecimentos às pessoas que ajudaram na realização deste trabalho:

Ao Prof. Dr. Paulo Roberto Gardel Kurka, orientador deste trabalho, pelos seus conhecimentos a mim transmitidos, sua paciência e dedicação, sua atenção e boa vontade.

Ao meu amigo Marcus, pelo incentivo na realização do mestrado, e por toda a parceria no desenvolvimento do mesmo.

Aos professores da Faculdade de Engenharia Mecânica pelos ensinamentos transmitidos ao longo do curso.

A todos os colegas que contribuíram de forma direta ou indireta para a melhora deste trabalho, em especial ao Jaime pelos conselhos valiosos.

A minha namorada pelo apoio e incentivo.

A FAPEAM pelo suporte financeiro.

“Simple' does not mean 'easy'. I have learned that the things that seem the simplest are often the most powerful of all.”

(Christie Golden, Thrall: Twilight of the Aspects)

RESUMO

A presente proposta aplica-se à criação de ambientes de simulação 3D voltados ao estudo e desenvolvimento de técnicas e algoritmos de navegação de robôs móveis baseados em imagens (ANBI). ANBI's necessitam operar de forma robusta e eficiente. Testes de eficiência dos ANBI podem ser realizados de maneira física, a partir da realização de experimentos em um protótipo, ou através de simulação numérica. Poucas são as plataformas de simulação numérica existentes para testes de ANBI, limitadas a esquemas gráficos e modelos simplificados da cinética de robôs. Nesse sentido, torna-se necessário a criação de uma plataforma de testes numéricos de ANBI's que integre aplicações sofisticadas de simulação de ambientes reais para a navegação e processamento de imagens e dados. Este trabalho propõe assim, a elaboração de uma plataforma de integração de programas de alto nível para a construção de modelos de ambientes 3D e testes de ANBI de robôs móveis. Serão utilizadas técnicas para aplicação de textura e iluminação, assim podendo representar fielmente a geração de imagens renderizadas com relação à sua versão no ambiente real. A pesquisa integrará scripts de processamento de imagem, na linguagem de programação C++, e controle da trajetória, desenvolvido no Matlab, com a plataforma Blender, que é programada em Python.

Palavras Chave: ambiente 3D, robótica móvel, processamento de imagem, Blender e simulação.

ABSTRACT

This proposal applies to the creation of 3D simulated environments for the study and development of mobile robot image based navigation algorithms and techniques (IBNA). IBNA's need to operate robustly and efficiently. The efficiency test of IBNA can be performed in a physical way, from conducting experiments on a prototype, or by numerical simulation. A few numerical simulation platforms exists for IBNA tests, limited to graphic schemes and simplified models of robots kinetics. Thus, it is necessary to create a numerical test platform of IBNA's that integrates sophisticated simulated applications of real environments for navigation with data and image processing. This work proposes the development of high-level programs integration platform for building 3D model’s environments and the test of IBNA in mobile robots. Techniques will be used for applying texture and lighting, in order to accurately represent the generation of rendered images regarding to its version in the real environment. The research will integrate image processing scripts, in the programming language C++, and trajectory control, developed in Matlab, with Blender platform, which is programmed in Python.

Key Words: 3D environment, mobile robotic, image processing, Blender and simulation.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Este trabalho descreve todo o planejamento, execução e conclusão de um simulador para robôs móveis, integrado com ambientes de programação de alto nível. Este simulador é desenvolvido em ambiente computacional e consiste na movimentação e controle de um robô móvel diferencial por meio de *softwares* externos, com captura e armazenamento de imagens ao longo de sua trajetória.

1.1 Apresentação

"Simuladores são ambientes computacionais que emulam o acontecimento de algum fenômeno real que os usuários conseguem manipular, explorar e experimentar" (JONASSEN, 1996, p. 78). Nesta pesquisa propõe-se a criação de ambientes de simulação 3D voltados ao estudo e desenvolvimento de técnicas e algoritmos de navegação de robôs móveis baseados em imagens (ANBI).

No modelo geralmente adotado para o desenvolvimento de atividades com robôs, um grupo composto por construtores e programadores, planeja e realiza um conjunto de testes pertinentes, observando e analisando o resultado de cada teste. Idealmente estes testes devem ser realizados em um robô real, mas essa tarefa pode se tornar ineficiente, principalmente quando existe o intuito de uma série de testes diferentes com o robô.

> "Para diminuir o custo e o tempo de pesquisa no desenvolvimento de robôs, várias pesquisas desenvolvem e usam ferramentas próprias para prover uma maneira fácil e simples de testar ideias, teorias e programas com robôs sem depender fisicamente da máquina. A maneira mais simples de fazer isso é a partir de simuladores computacionais" (BECKER, 2010, p. 9).

Este projeto foi desenvolvido no Departamento de Sistemas Integrados da FEM, e consististe na criação de uma série de aplicações utilizando as linguagens Matlab e Python. Contendo, a construção de plataformas robóticas virtuais, e sua integração com algoritmos de processamento de imagens e controle de trajetórias.

Para a estimação de trajetórias de robôs móveis serão utilizadas técnicas de odomêtria e fluxo óptico. A plataforma Blender 2.65, foi a base de construção, tanto dos modelos tridimensionais, como dos algoritmos de animação e reprodução de movimentos.

O desenvolvimento dos modelos foi feito utilizando a linguagem de programação Python, assim como, técnicas para a construção manual das estruturas do robô e do ambiente. Também foram utilizados conhecimentos sobre o comportamento da iluminação em ambientes específicos, e da construção de materiais com propriedades impactantes. Tornando, este pacote, valioso, tanto para o estudo e desenvolvimento de novas técnicas de controle e processamento de imagem, como para a aplicação de situações reais e prevenção de condições de perigo em implementações mais práticas.

Os tópicos discutidos a seguir serão, Motivação, onde serão expostos os principais pontos para o desenvolvimento da pesquisa, em seguida, serão citados os Objetivos geral e específicos do trabalho, e por último será comentada a organização do trabalho, resumindo o que estará presente em cada tópico subsequente.

1.2 Motivação

Para a realização do trabalho, foram analisados vários pontos acerta de simuladores computacionais, e suas aplicações na robótica móvel, sua utilização traz uma série de vantagens para qualquer aplicação nesta área, sendo de grande importância para qualquer projeto. Dentre estas vantagens, cabe-se destacar:

- **Economia de recursos financeiros:** o custo de um software de simulação é menor do que o custo de robôs comerciais. Além disso, quando se está utilizando simuladores não é necessária a compra de equipamentos para a utilização no ambiente no qual o robô ficará, já que todos estes equipamentos serão virtuais;
- **Facilidade de criação do ambiente que será utilizado pelo robô**: a criação do ambiente real exige a utilização de equipamentos, o que demanda tempo para a organização de tais equipamentos na sala. A criação do ambiente virtual consome um tempo menor, já que não é necessário mover equipamentos, apenas adicioná-los no ambiente virtual;

- **Economia de tempo:** em ambientes simulados há uma economia de tempo na realização de atividades. Dentre os principais fatores, podemos citar a facilidade de criação do ambiente que será utilizado pelo robô e a facilidade de reconfiguração do ambiente para novos testes (reposicionar robôs e equipamentos). Isso permite que, em um mesmo intervalo de tempo, o ambiente virtual permita a realização de um maior número de experimentos com o robô.
- **Utilização de robôs de maior qualidade:** os robôs sofrem desgastes com o tempo, o que faz com que o seu funcionamento varie com o tempo. Por exemplo, motores antigos não se comportam da mesma forma que motores novos. Robôs simulados não possuem desgastes em suas peças, e por isso não há erros inesperados nos testes, oriundos do desgaste das peças do robô;
- **Facilita o teste de novos algoritmos e modelos de robôs:** devido à economia de tempo na reconfiguração do ambiente de testes, os testes são bem mais simples no ambiente virtual; e
- **Facilita o teste com vários robôs:** este tipo de teste seria mais difícil no ambiente real, já que o grupo de testes deveria possuir vários robôs.

Além dos fatores citados acima, cabe-se destacar que atualmente, não existe um simulador computacional voltado exclusivamente para a aplicação de técnicas e algoritmos de navegação de robôs móveis baseados em imagens (ANBI), tornando-se necessário, neste caso, desenvolver seu próprio modelo do zero, utilizar banco de imagens reais, ou construir um robô móvel compatível com a aplicação.

A utilização de simuladores nas aplicações de robótica móvel também facilita a aquisição de dados, e não necessita de um sistema de captação de movimento para obter a posição atual do robô, desta forma, simplificando e barateando o desenvolvimento de pesquisas.

1.3 Objetivos

1.3.1 Geral

- Desenvolvimento de simulador para robôs móveis, integrado com ambientes de programação de alto nível.

1.3.2 Específicos

- Escolha e projeto do robô a ser simulado;
- Desenvolvimento de modelo 3D seguindo parâmetros necessários para a movimentação do robô no ambiente virtual;
- Criação de ambientes de simulação 3D (somente sólidos);
- Definição e comparação de tecnicas de textura e iluminação a serem utilizadas para representação fiel do ambiente na geração de imagens;
- Projeto dos algoritmos para navegação; e
- Validação do ambiente.

1.4 Organização do Trabalho

O trabalho está dividido em sete capítulos. Este primeiro capítulo apresentou uma introdução ao projeto a ser descrito nesta dissertação. O Capítulo 2 corresponde à computação gráfica, nele serão abordados conceitos e equações ligadas a aplicações desta área, o Capitulo 3 corresponde à simulação de robôs móveis, onde serão destacados principais desenvolvimentos na robótica móvel, também serão comparados simuladores de robôs móveis e implementações de modelagens, utilizando o Matlab/Simulink.

O Capítulo 4 corresponde à descrição da proposta, aonde são apresentados o projeto e a metodologia utilizada para a sua execução. No Capítulo 5, são discutidos os detalhes de implementação, descrevendo todas as fases no desenvolvimento do trabalho.

O Capítulo 6 corresponde à validação de tudo o que foi desenvolvido uma vez que é feita a apresentação dos testes e análise de resultados. Para finalizar este trabalho tem-se o capítulo de Conclusão. Nele são apresentadas conclusões obtidas a partir da análise dos testes e propostas para trabalhos futuros.

2 COMPUTAÇÃO GRÁFICA

"A Computação Gráfica (CG) é uma área da Ciência da Computação que se dedica ao estudo e desenvolvimento de técnicas e algoritmos para a geração (síntese) de imagens através do computador" (MANSSOUR, 2006, p. 1).

Hoje em dia, a computação gráfica está presente em quase todas as áreas do conhecimento humano, da engenharia que utiliza as tradicionais ferramentas CAD (*Computer-Aided Design*), até a medicina que trabalha com modernas técnicas de visualização para auxiliar o diagnóstico por imagens. Nesta área, também têm sido desenvolvidos sistemas de simulação para auxiliar no treinamento de cirurgias endoscópicas. Outros tipos de simuladores são usados para treinamento de pilotos e para auxiliar na tomada de decisões na área do direito (por exemplo, para reconstituir a cena de um crime).

> "Atualmente, com as facilidades disponíveis nas bibliotecas gráficas existentes, a programação das aplicações está mais simples. Por exemplo, OpenGL (*Open Graphics Library*), também definida como uma "interface para hardware gráfico", é uma biblioteca de rotinas gráficas e de modelagem, bidimensional (2D) e tridimensional (3D), portável e rápida" (COHEN, 2006).

Duas áreas têm uma relação bastante próxima com a computação gráfica: Processamento de Imagens (PI) e Visão Computacional. A área de processamento de imagens abrange o estudo e a pesquisa de técnicas para realizar a manipulação de imagens, tais como ajustes de cor, brilho, contraste ou aplicação de filtros, entre outras. Sistemas de PI são encontrados atualmente, por exemplo, em consultórios de cirurgiões plásticos e salões de beleza. Neste caso, uma pessoa pode ver como será o resultado de uma plástica ou corte de cabelo através de simulações no computador.

A Visão Computacional trabalha com a análise de imagens, buscando obter a especificação dos seus componentes para identificação dos modelos geométricos que a compõem. Uma aplicação de técnicas de Visão Computacional é o reconhecimento automático de impressões digitais.

2.1 Representação de Objetos

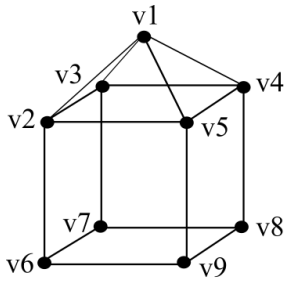
Em computação gráfica, modelos são usados para representar entidades e fenômenos do mundo físico real no computador. Existem várias categorias ou métodos de construção de modelos tridimensionais. Cada um tem suas vantagens e desvantagens, adaptando-se melhor para uma ou outra aplicação.

A Modelagem consiste em todo o processo de descrever um modelo, objeto ou cena, de forma que se possa desenhá-lo. Na verdade, a modelagem engloba dois tópicos de estudo: formas de representação dos objetos, que se preocupa com a forma (ou estruturas de dados) como os modelos são armazenados; e técnicas de modelagem dos objetos, que trata das técnicas interativas (e também das interfaces) que podem ser usadas para criar um modelo de objeto.

Segundo Watt (2000), é possível enumerar quatro formas de representação, de acordo com a importância e frequência de utilização: malha de polígonos; superfícies paramétricas; Geometria Sólida Construtiva (CSG); e enumeração de ocupação espacial.

A forma mais comum de representar modelos 3D é através de uma malha de polígonos. Ou seja, define-se um conjunto de vértices no espaço (geometria) e como esses vértices devem ser ligados para formarem polígonos fechados, chamados de face (topologia), que podem ser triângulos ou quadrados. O armazenamento desse tipo de estrutura é usualmente realizado através de vetores de estruturas, matrizes ou listas. Por exemplo, a Figura 2.1 apresenta a lista de vértices e faces necessárias para desenhar uma casa simplificada.

Figura 2. 1: Exemplo de objeto representado por uma malha de polígonos.



Vértices (geometria)

1	x1 y1 z1
2	x2 y2 z2
3	x3 y3 z3
4	x4 y4 z4
5	x5 y5 z5
6	x6 y6 z6
7	x7 y7 z7
8	x8 y8 z8
9	x9 y9 z9

Faces (topologia)

1	v1 v4 v5
2	v1 v5 v2
3	v1 v2 v3
4	v1 v3 v4
5	v4 v3 v7 v8
6	v5 v4 v8 v9
7	v2 v5 v9 v6
8	v3 v2 v6 v7
9	v6 v9 v8 v7

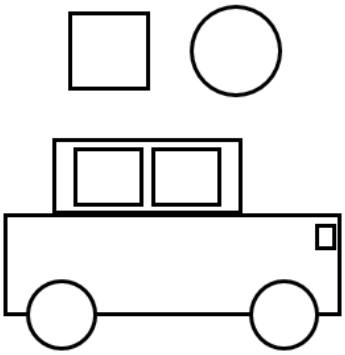
Fonte: Autor.

Sendo armazenados os modelos, será feito o instanciamento de primitivas, ou seja, o sistema de modelagem definirá um conjunto de objetos primitivos 3D que são relevantes para a área de aplicação. Tais objetos podem ser representados, por exemplo, por malhas de polígonos ou superfícies paramétricas.

2.2 Transformações Geométricas

Após serem armazenados os modelos dos objetos, estes podem ser parametrizados, tanto em termos de transformações geométricas, como em outras propriedades, e agrupados. Para isto, as transformações mais utilizadas são rotação, translação e escala. A Figura 2.2, ilustra a construção de um objeto complexo através de primitivas simples.

Figura 2. 2: Objeto construído a partir de primitivas simples.



Fonte: Autor.

Como os objetos são armazenados em matrizes, as transformações são operações algébricas que alteram os valores de cada coordenada, assim podendo alterar drasticamente a representação inicial do sólido. Uma transformação de coordenadas da forma representada pelas equações (2.1) e (2.2), é denominada uma transformação “afim”.

$$\overrightarrow{v'} = A\vec{v} + b \tag{2.1}$$

Ou

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \quad (2.2)$$

Neste caso, as coordenadas (x', y', z') do vetor $\overrightarrow{v'}$, que definem um ponto no espaço, são uma função linear de (x, y, z) e a_{ij} e b_i são constantes determinadas pelo tipo de transformação. As transformações afim têm a função de modificar a posição dos pontos no espaço, ou dos objetos no espaço.

Essas transformações tem a característica geral de transformar linhas paralelas em linhas paralelas e mapear pontos finitos em pontos finitos. O grupo de transformações afins do espaço define a geometria afim, que estuda as razões e proporções entre objetos geométricos.

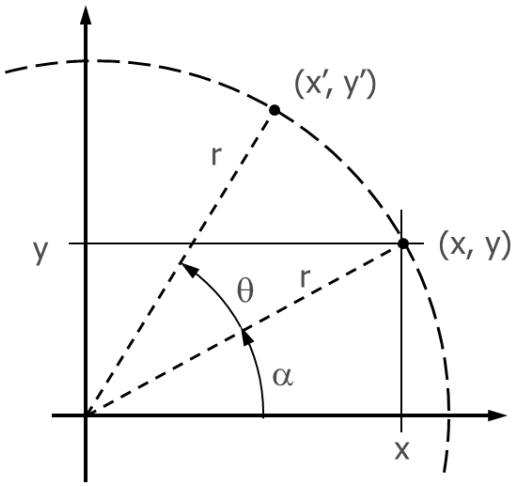
Note-se que em geometria afim, paralelismo é um conceito importante, sendo relações entre linhas paralelas uma parte substancial da geometria e os teoremas da geometria afim são idênticos aos da geometria euclidiana. Rotação, translação, escalamento, espelhamento e cisalhamento são exemplos de transformações afins detalhados a seguir.

2.2.1 Rotação

A rotação é o giro de um determinado ângulo de um ponto em torno de um eixo de referência, sem alteração da distância entre eles. Esta operação é aplicada normalmente sobre todos os pontos de uma figura, o que possibilita que ela seja rotacionada. Vários programas gráficos dispõem desta operação, sendo que alguns restringem o ângulo de rotação a valores fixos, tais como, 90° e 180°.

Para o cálculo da matriz de rotação, será considerado inicialmente apenas duas coordenadas, por exemplo, x e y. Assim, na Figura 2.3, o ponto P, de coordenadas (x,y), será rotacionado de um ângulo θ em torno do eixo z, no sentido anti-horário, até a posição do ponto P' (x',y'). A linha que une o ponto P à origem do sistema de coordenadas está rotacionada de um ângulo α em relação ao eixo x.

Figura 2. 3: Exemplo de rotação no espaço 2D.



Fonte: Autor.

De acordo com a Figura 2.3 pode-se gerar de imediato as relações representadas pelas equações (2.3), (2.4) e (2.5).

$$r = \sqrt{x^2 + y^2} \tag{2.3}$$

$$cos\alpha = \frac{x}{r}\,, \qquad sen\alpha = \frac{y}{r} \tag{2.4}$$

$$\cos(\theta + \alpha) = \frac{x'}{r}, \qquad sen(\theta + \alpha) = \frac{y}{r} \tag{2.5}$$

Como:

$$\cos(\,\theta + \alpha) = cos\theta cos\alpha - sen\theta sen\alpha \tag{2.6}$$

$$\mathrm{sen}(\,\theta + \alpha) = sen\theta cos\alpha - cos\theta sen\alpha \tag{2.7}$$

Substituindo (2.3) em (2.4), e em seguida (2.4) e (2.5) em (2.6) e (2.7), poderão ser obtidas as relações representadas nas equações (2.8) e (2.9).

$$x' = xcos\theta - ysen\theta \tag{2.8}$$

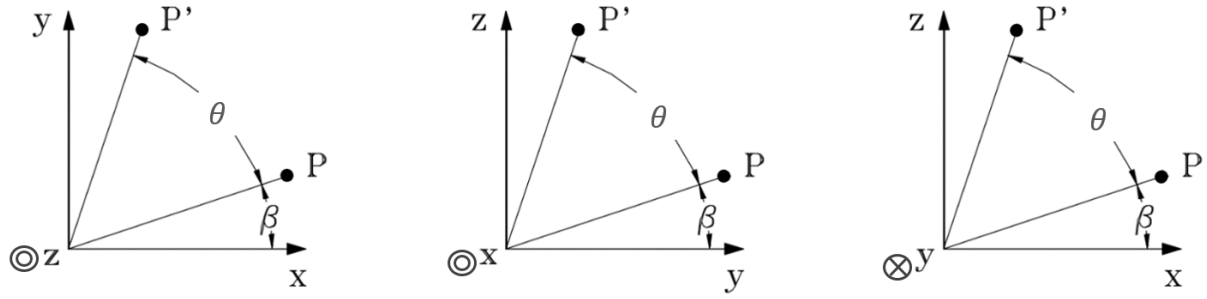
$$y' = xsen\theta + ycos\theta \tag{2.9}$$

Sendo representada por sua forma matricial na equação (2.10).

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos\theta & -sen\theta \\ sen\theta & cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{2.10}$$

Considerando-se as figuras 2.4a, 2.4b e 2.4c e a equação (2.10), pode-se facilmente deduzir as fórmulas para o cálculo no espaço tridimensional da rotação de um ponto P para o ponto P', sendo o plano de rotação perpendicular ao eixo z, y e x.

Figura 2. 4: Rotações realizadas com diferentes eixos de referência.



Fonte: Autor.

Desta forma, utilizando-se do conceito de coordenadas homogêneas, pode-se chegar às equações (2.11), (2.12) e (2.13).

$$Rotação\ em\ torno\ do\ eixo\ z\ ... \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} cos\theta & -sen\theta & 0 \\ sen\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad (2.11)$$

$$Rotação\ em\ torno\ do\ eixo\ y\ ... \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} cos\theta & 0 & sen\theta \\ 0 & 1 & 0 \\ -sen\theta & 0 & cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad (2.12)$$

$$Rotação\ em\ torno\ do\ eixo\ x\ ... \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\theta & -sen\theta \\ 0 & sen\theta & cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad (2.13)$$

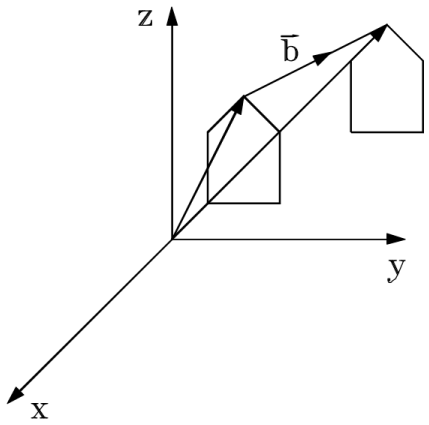
De uma forma mais geral, pode-se definir um eixo qualquer normalizado (x_n, y_n, z_n) e aplicar uma rotação de ângulo θ, para isto pode-se chegar na relação apresentada na equação (2.14), alinhando-se um dos eixos x, y, z, com o eixo de rotação definido, em seguida aplica-se a rotação neste eixo, e retornam-se os eixos referência para sua posição original. Para redução do tamanho da equação, considera-se $c = cos\theta$ e $s = sen\theta$.

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} x_n^2 + (1 - x_n^2)c & x_n y_n (1 - c) - z_n s & x_n z_n (1 - c) + y_n s \\ x_n y_n (1 - c) + z_n s & y_n^2 + (1 - y_n^2)c & y_n z_n (1 - c) - x_n s \\ x_n z_n (1 - c) - y_n s & y_n z_n (1 - c) + x_n s & y_n^2 + (1 - y_n^2)c \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad (2.14)$$

2.2.2 Translação

A translação é alteração da posição de um ponto através da soma de constantes de deslocamento as suas coordenadas, é normalmente aplicada sobre todos os pontos de uma figura, de maneira a possibilitar a sua movimentação no espaço (Figura 2.5). O exemplo clássico em computação gráfica de aplicação desta transformação é a função *pan*, disponível em vários sistemas gráficos.

Figura 2. 5: Exemplo de translação em um objeto.



Fonte: Autor.

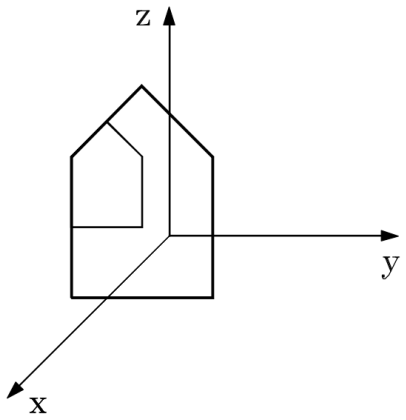
Em termos de transformação afim, a translação corresponde à soma de um vetor de deslocamento ao vetor que define o ponto que se deseja deslocar. Assim, na equação (2.15), o vetor com as componentes b_i corresponde ao vetor de deslocamento.

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \quad (2.15)$$

2.2.3 Escalamento

A Mudança de Escala corresponde à multiplicação das coordenadas de um ponto por valores iguais ou diferentes. É normalmente aplicada sobre todos os pontos de uma figura com o objetivo de ampliar ou reduzir a sua dimensão ou então distorcer a sua forma geométrica (Figura 2.6). O uso clássico desta operação em computação gráfica é a função *zoom in* (ampliação) ou *zoom out* (redução).

Figura 2. 6: Exemplo de escalamento aplicado em um objeto.



Fonte: Autor.

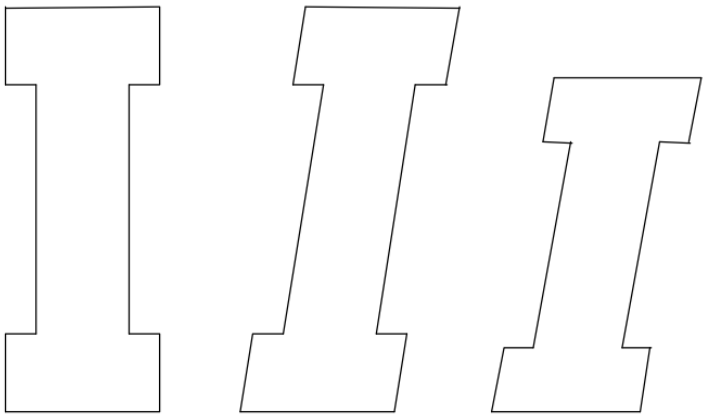
Quando somente a operação de escalamento é realizada, na posição da matriz *A* na equação (2.2) fica a matriz E, onde e_x, e_y, e_z são os fatores de escala das coordenadas x , y e z , respectivamente. Sendo assim, na equação (2.16), pode-se notar a representação desta transformação.

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} e_x & 0 & 0 \\ 0 & e_y & 0 \\ 0 & 0 & e_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{2.16}$$

2.2.4 Cisalhamento

Outra transformação afim importante de ser estudada é o cisalhamento (*shear*), cujo exemplo clássico para o sistema de coordenadas bidimensional que explica a sua função é o da italização de um caráter (Figura. 2.7).

Figura 2. 7: Exemplo de cisalhamento aplicado em um objeto 2D.



a) b) c)

Fonte: Autor.

Neste caso, há uma variação no valor da coordenada x em função do valor da y (fig. 2-7a e 2-7b), sendo a equação (2.17) a correspondente à transformação, tendo $0 < m_x < 1$ e $m_y = 0$.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & m_x \\ m_y & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{2.17}$$

Pode-se associar uma outra transformação a de cisalhamento, como, por exemplo, o escalamento da coordenada y (fig. 2-7c), conforme exemplificado na equação (2.18), onde $e_x = 1$ e $0 < e_y < 1$; m_x *e* m_y com os valores iguais à equação (2.17).

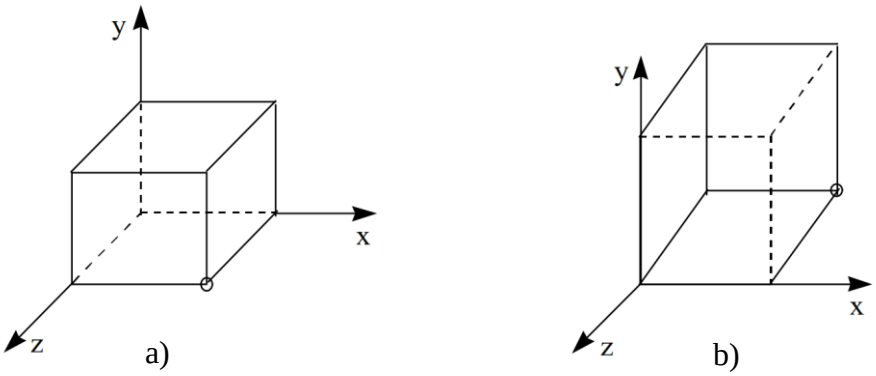
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} e_x & m_x \\ m_y & e_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{2.18}$$

A equação (2.19) ilustra o uso desta transformação para o caso tridimensional. Sendo para m_{ij}, a deformação de i em relação à j.

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 1 & m_{xy} & m_{xz} \\ m_{yx} & 1 & m_{yz} \\ m_{zx} & m_{zy} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{2.19}$$

O efeito do cisalhamento em um ambiente 3D pode ser visto com a deformação de o paralelepípedo representado na Figura 2.8a em todas as coordenadas. Desta forma, gerando o seu correspondente na Figura 2.8b.

Figura 2. 8: Exemplo de cisalhamento aplicado em um objeto 3D.

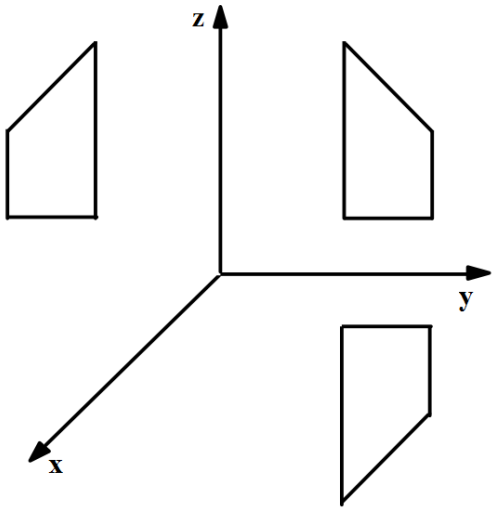


Fonte: Autor.

2.2.5 Espelhamento

Uma operação bastante conhecida em computação gráfica é o espelhamento, a qual consiste em rotacionar um objeto em torno de um eixo de tal maneira que os pontos do objeto na posição original e na rotacionada mantenham a mesma distância em relação a uma linha de referência, caso bidimensional, ou a um plano de referência, caso tridimensional. Na Figura 2.9, há o espelhamento de um objeto em torno do eixo y em relação ao plano xy e em torno do eixo z em relação ao plano xz. Em ambos os casos, o ângulo de rotação é 180°.

Figura 2. 9: Exemplo de espelhamento aplicado em um objeto.



Fonte: Autor.

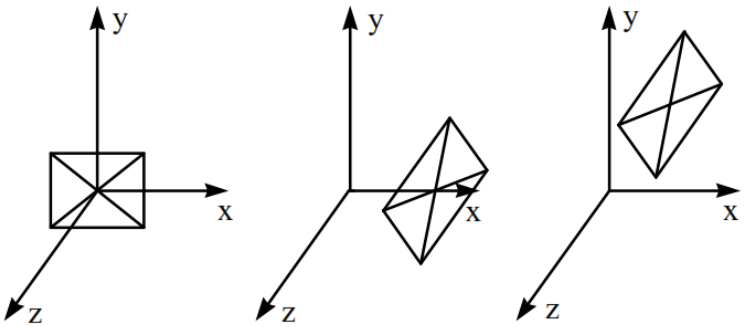
Com a utilização destas transformações podem-se criar objetos bem mais complexos do que as primitivas utilizadas. A Figura 2.10 apresenta uma mesa formada pelo instanciamento de três cilindros, com diferentes alturas e raios.

Figura 2. 10: Mesa modelada por instanciamento de primitivas.

Fonte: Autor.

As transformações geométricas também podem ser utilizadas para “animar” o desenho/gráfico, sendo aplicadas em instantes diferentes, criando a ilusão de movimentação na cena. A Figura 2.11 ilustra um exemplo de animação em três instantes diferentes do mesmo objeto.

Figura 2. 11: Exemplo de efeito de animação.



Fonte: Autor.

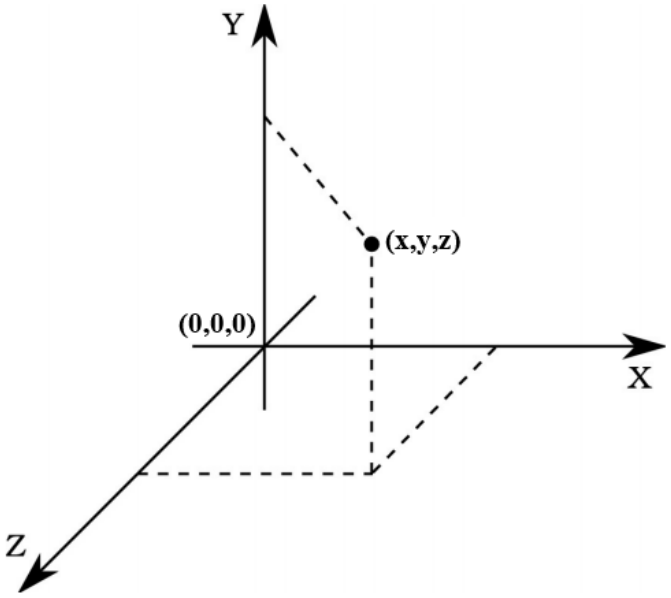
2.3 Visualização 3D

Existe um conjunto de técnicas em computação gráfica que permite transformar as informações a respeito de um modelo contidas em uma estrutura de dados, em uma imagem que pode ser exibida em um monitor. Portanto, considera-se que uma imagem consiste em uma matriz de pontos e um modelo é uma representação computacional de um objeto.

Para entender como funciona o processo de visualização é importante conhecer o conceito de Universo, que pode ser definido como a região do espaço utilizada em uma aplicação. Como a descrição geométrica de um modelo normalmente envolve coordenadas geométricas, é preciso adotar um sistema de referência que irá definir uma origem em relação à qual todos os posicionamentos do universo são descritos. Em geral, este Sistema de Referência do Universo (SRU) consiste em três eixos ortogonais entre si (x, y, z) e uma origem (0, 0, 0). Uma coordenada, então, é formada pelos valores de x, y e z, que correspondem às posições ao longo dos respectivos eixos (denominados cartesianos) e todos os procedimentos são definidos em relação a este sistema de referência. Uma representação deste sistema pode ser vista pela Figura 2.12.

A primeira etapa do processo de visualização 3D é a definição da cena 3D. Nesta etapa cada um dos objetos que farão parte do mundo 3D é incluído e posicionado no SRU. Este posicionamento é feito através de operações de escala, rotação e translação.

Figura 2. 12: Representação do Sistema de Referência do Universo (SRU).



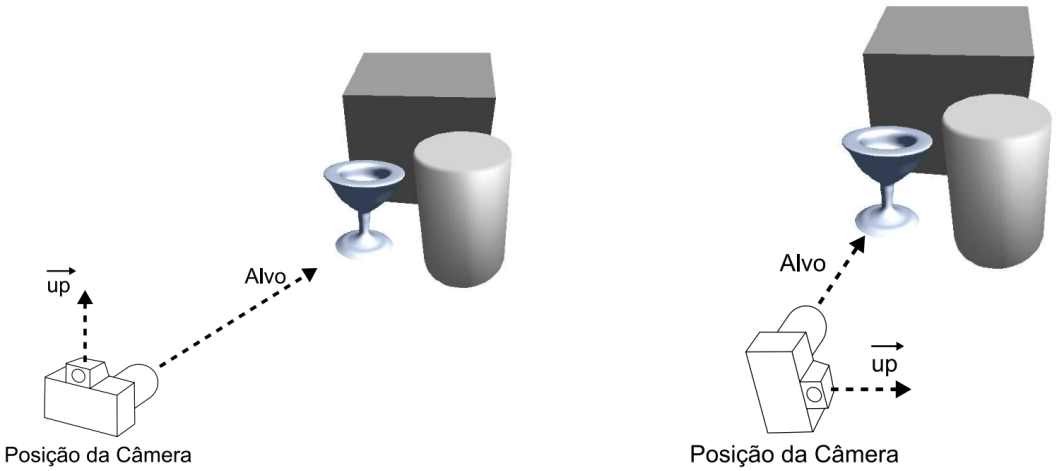
Fonte: Autor.

O próximo passo consiste na especificação do observador virtual, que define de que local se deseja que a cena 3D seja exibida, por exemplo, de cima, do lado direito ou do lado esquerdo. Portanto, a especificação do observador inclui a sua posição e orientação, ou seja, onde ele está e para onde está olhando dentro do universo. A necessidade da existência desse observador deve-se ao fato de que um mesmo conjunto de objetos no universo 3D, visto de diferentes lugares tem, para quem os observa, diferentes coordenadas para cada posição.

Como a imagem gerada a partir da posição e orientação do observador é estática, se faz analogia com uma foto. Pode-se dizer que se obtém uma fotografia quando a câmera está numa determinada posição direcionada para o objeto.

A posição da câmera é dada por um ponto (x, y, z) em relação ao mesmo universo no qual os objetos estão posicionados (SRU) e sua orientação é dada por um ponto alvo (x, y, z) e um vetor, aqui chamado de *up*. A Figura 2.13 ilustra estes conceitos: a câmera é posicionada de duas maneiras diferentes, ocasionando a geração de duas imagens distintas: na Figura 2.13a, os objetos aparecerão da mesma maneira que estão posicionados; na Figura 2.13b os objetos aparecerão inclinados 90º para a direita.

Figura 2. 13: Modelo de câmera sintética.



a) Câmera: orientação normal b) Câmera: orientação inclinada

Fonte: Autor.

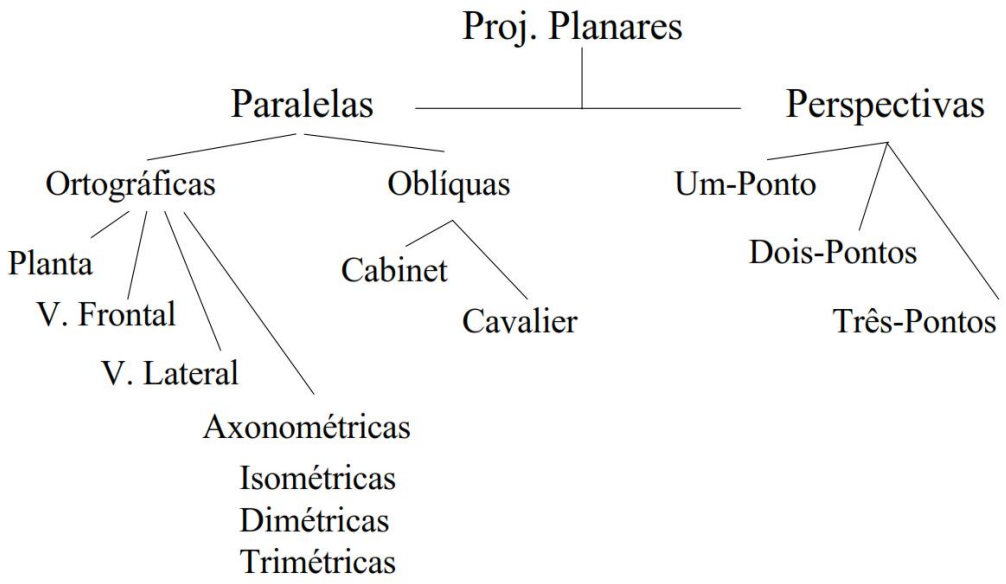
A partir da posição e orientação da câmera é criado um novo sistema de referência, o chamado Sistema de Referência da Câmera (SRC). O SRC é criado para que seja possível definir a posição de cada objeto em relação ao observador, ou seja, em relação à origem do SRC, pois só assim se pode saber qual é o objeto que está mais próximo ou mais afastado do observador, bem como se um objeto está encobrindo outro.

2.4 Projeções

Quando se está trabalhando com a representação de objetos no espaço tridimensional existe uma etapa obrigatória: mapear suas representações 3D para imagens 2D que serão exibidas em um dispositivo como um monitor. Esta operação de obter representações bidimensionais de objetos tridimensionais é chamada de projeção.

Como a maioria dos objetos é representada por uma coleção de vértices, sua projeção é definida por raios de projeção (segmentos de retas) chamados de projetantes, que passam através de cada vértice do objeto e interseccionam um plano de projeção. Esta classe de projeções é chamada de projeções geométricas planares, e podem ser classificadas de acordo com a Figura 2.14.

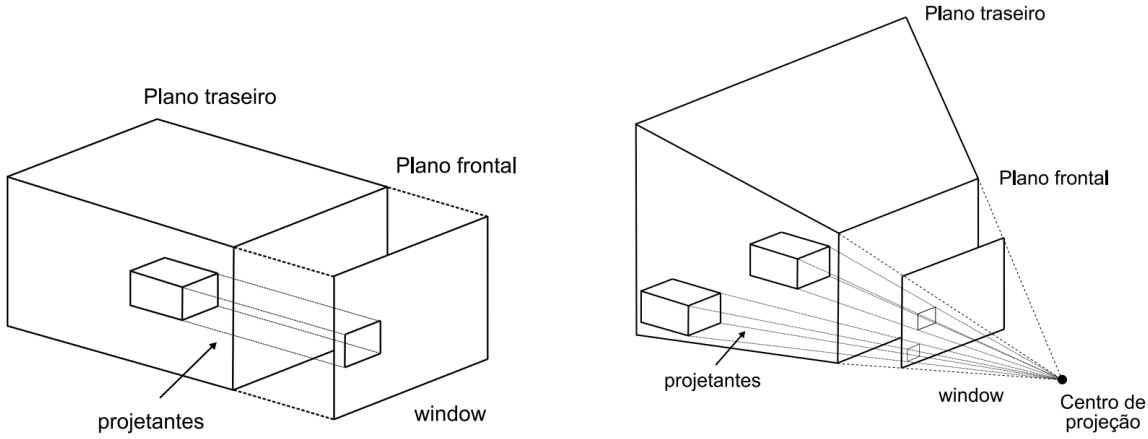
Figura 2. 14: Arvore de classificação das projeções planares.



Fonte: Autor.

As projeções usualmente são divididas em dois tipos principais: Projeção Paralela Ortográfica, na qual as projetantes são paralelas entre si, passam pelos vértices dos objetos e interseccionam o plano com um ângulo de 90º (Figura 2.15a); Projeção Perspectiva (Um-Ponto), quando as projetantes emanam de um único ponto que está a uma distância finita do plano de projeção e passam pelos vértices (Figura 2.15b).

Figura 2. 15: Representação de projeções para a geração de imagens.

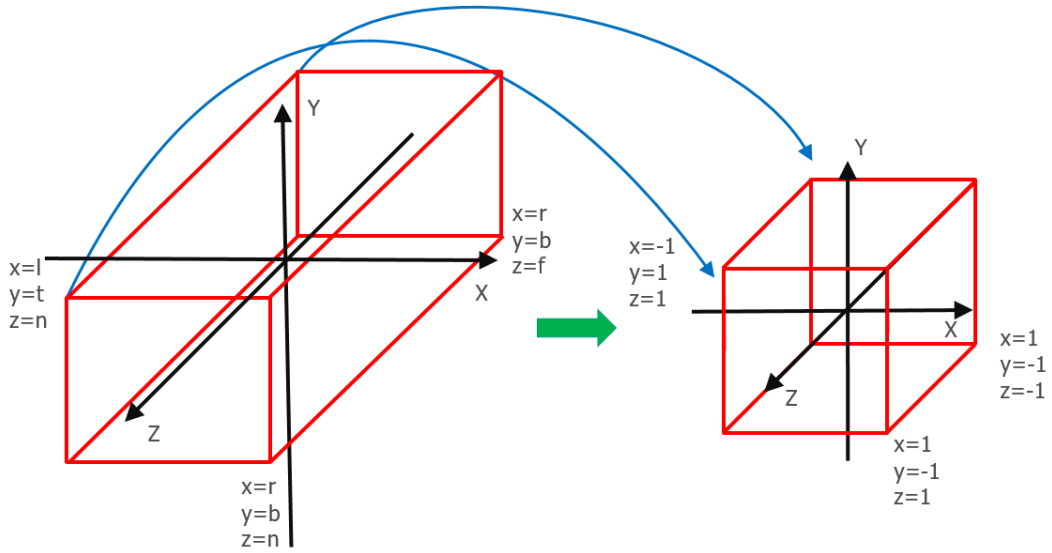


a) Projeção Paralela Ortográfica b) Projeção Perspectiva (Um-Ponto)

Fonte: Autor.

Na projeção paralela ortográfica (Figura 2.16) não há alteração nas medidas do objeto. Sua construção é bastante simples, pois, basicamente, consiste em omitir uma das componentes de cada vértice.

Figura 2. 16: Esquema detalhado da conversão de espaços para a projeção ortográfica.



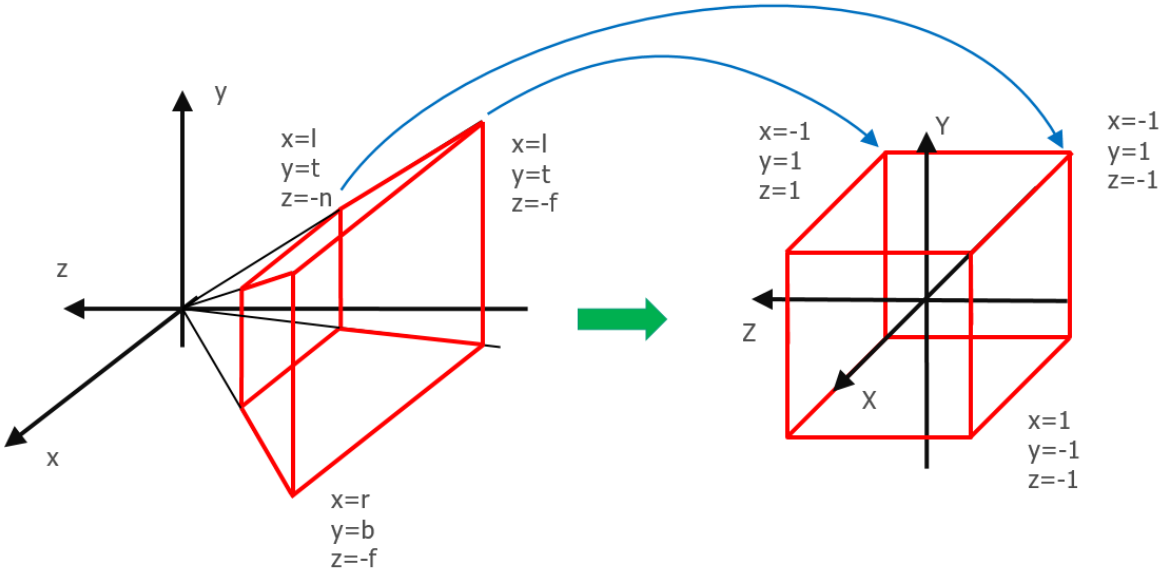
Fonte: Autor.

A conversão de uma coordenada do espaço inicial para a sua correspondente no espaço transformado pela projeção ortográfica é representada pela equação (2.20), utilizando o conceito de coordenadas homogêneas.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & \frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (2.20)$$

Entretanto, a projeção perspectiva é mais utilizada, uma vez que representa melhor o que acontece na realidade. Por exemplo, se dois objetos possuem o mesmo tamanho e estão posicionados a diferentes distâncias do plano de projeção, o objeto que está mais longe vai parecer menor do que o objeto que está próximo, conforme ilustra a Figura 2.17.

Figura 2. 17: Esquema detalhado da conversão de espaços para a projeção perspectiva.



Fonte: Autor.

Novamente, pode-se converter uma coordenada do espaço inicial para o transformado, a relação desta conversão está representada na equação (2.21).

$$\begin{bmatrix} x' \\ y' \\ z' \\ z \end{bmatrix} = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{2.21}$$

Sendo possível se calcular os valores homogêneos (x_h, y_h, z_h) pela relação apresentada na equação (2.22).

$$\begin{bmatrix} x_h \\ y_h \\ z_h \\ 1 \end{bmatrix} = -\frac{1}{z} \begin{bmatrix} x' \\ y' \\ z' \\ z \end{bmatrix} \tag{2.22}$$

Na visualização 3D também é preciso definir um volume de visualização, ou seja, a região exata do universo 3D que se deseja visualizar. O tamanho e a forma do volume de visualização dependem do tipo de projeção.

Para a projeção paralela ortográfica, os quatro lados do volume de visualização e os planos frontal e traseiro na direção do eixo z, formam um paralelepípedo. Para a projeção perspectiva o volume de visualização é um tronco de pirâmide, limitado pelos planos frontal e traseiro, cujo topo é o centro de projeção, como demonstra a Figura 2.15b.

Os planos frontal e traseiro do volume de visualização são paralelos ao plano de projeção, formando um volume de visualização limitado por seis planos e permitindo excluir partes da cena de acordo com a profundidade.

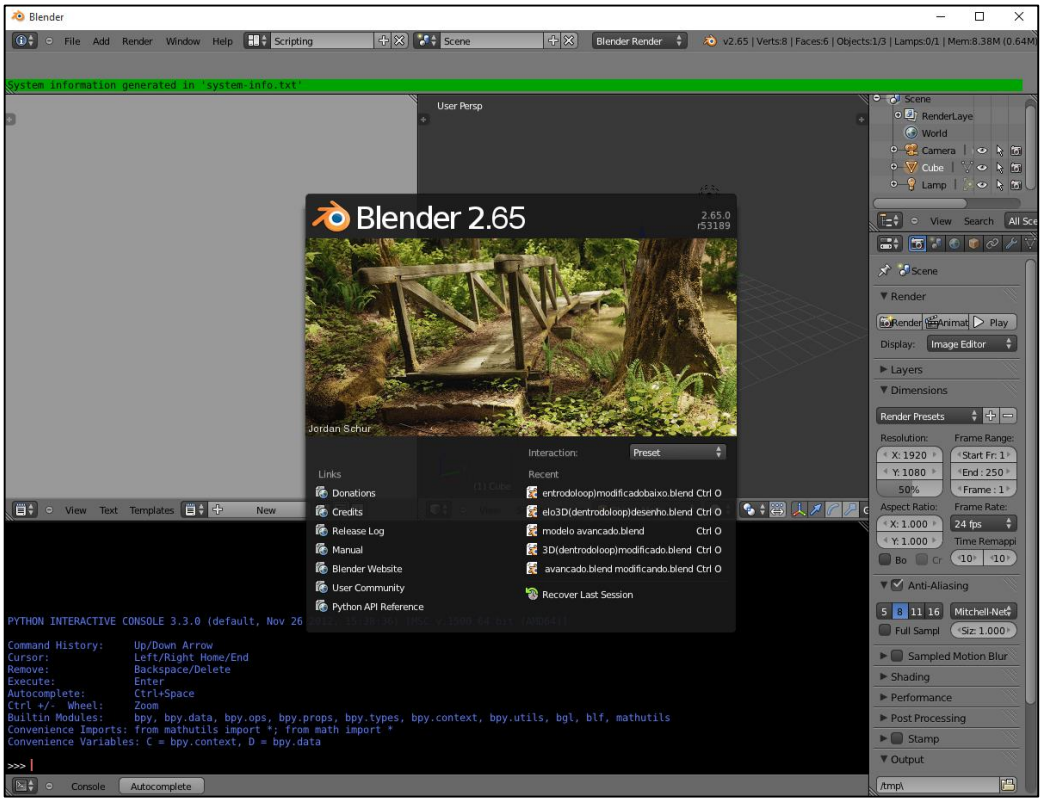
2.5 Blender – Plataforma de Simulação

Após uma comparação entre softwares livres existentes no mercado para o desenvolvimento da realidade virtual, optou-se por utilizar o Blender 2.65, da Blender Foundation. O Blender possui um excelente custo benefício, já que é um software gratuito e conta com inúmeros recursos presentes nos softwares mais completos e caros do mercado. Outra vantagem do Blender é que ele está disponível para um grande número de sistemas operacionais, entre eles a família Windows e as versões do gratuito Linux. A Figura 2.18, apresenta o ambiente de *scripting*, onde foi desenvolvida maior parte do projeto.

O Blender é baseado na biblioteca OpenGL, que lida tanto com desenhos na forma vetorial, como no formato pixel-a-pixel. Uma das características que possibilitou a utilização do Blender no projeto é que além da modelagem 3-D o Blender possui uma Game Engine (motor de jogos). A *Game Engine* trabalha como um compilador em tempo real, interpretando o ambiente a cada instante.

Outro ponto positivo deste software é a capacidade de trabalhar com scripts para serem utilizados durante a simulação. Estes scripts são escritos e compilados no Python, uma linguagem de programação gratuita e utilizada em computação gráfica. O Blender também suporta formatos de importação de arquivos com extensão (*.dxf), que são provenientes de programas tipo CAD, tornando a modelagem arquitetônica (paredes e objetos simétricos) mais fácil, rápida e precisa.

Figura 2. 18: Representação de ambiente do Blender 2.65.



Fonte: Autor.

Alguns recursos do Blender são:

- Dinâmica de corpos rígidos e macios;
- Dinâmica de fluídos e partículas;
- Modelagem por subdivisão e por curvas;
- Módulos para animação, *inverse kinematics*, *bones* e esqueletos e animação não-linear;
- *Game-engine*;
- Módulo de edição de vídeo e áudio;
- *Shaders*, NURBS, escultura digital (como o Z-Brush), Radiosidade, *Ray-Tracing*;
- Compatível com renderizadores externos (YafRay, Aqsis, Indigo, PovRay, etc);
- Programação de interatividade (jogos e passeios virtuais 3D);
- Importa e exporta modelos compatíveis com outros programas (3DS, VRML, X, OBJ, LWO, etc).

O Blender já foi utilizado até mesmo pela NASA para a construção de modelos 3D. Muitos modelos na "*NASA 3D resources*" (NASA, 2015), estão no formato nativo '.blend'. Além disso, é utilizado também na indústria cinematográfica, sendo o seu primeiro grande projeto internacional, *Spider-Man 2*, onde seu papel principal foi na criação de animações e pré-visualizações.

> "*As an animatic artist working in the storyboard department of Spider-Man 2, I used Blender's 3D modeling and character animation tools to enhance the storyboards, re-creating sets and props, and putting into motion action and camera moves in 3D space to help make Sam Raimi's vision as clear to other departments as possible*" (ZIERHUT, 2004).

Desta forma, a plataforma se prova mais do que capaz de suprir todas as necessidades do trabalho, assim podendo ser utilizada nas aplicações envolvendo robôs móveis relatadas no trabalho.

3 SIMULAÇÃO DE ROBÔS MÓVEIS

Neste capítulo serão abordadas as principais aplicações da robótica móvel na atualidade, descrevendo suas funcionalidades, além disso, apresentará a classificação de robôs móveis quanto aos seus componentes. Em seguida poderão ser vistos os aplicativos mais utilizados para a simulação de robôs, sendo destacadas suas vantagens de implementação, e por último, serão mostrados trabalhos envolvendo a ferramenta Matlab/Simulink, contendo a modelagem de sistemas, com breves comentários acerca de seus resultados.

3.1 Robôs móveis

Robôs convencionais são máquinas ancoradas em uma estrutura física, que podem ser programadas para executarem determinadas tarefas com eficiência e precisão. São geralmente utilizados em grandes instalações industriais, como por exemplo, na indústria automobilística. No entanto esses tipos de robôs possuem muitas limitações, sendo a falta de mobilidade e autonomia um de seus pontos fracos.

Para contornar esse problema, começou-se o desenvolvimento de robôs dotados da capacidade de locomoção e com certo grau de autonomia. Surgiam, então, os chamados robôs móveis. “O robô móvel pode ser definido como uma máquina com liberdade de locomoção e que possui a capacidade de interagir de forma autônoma com o ambiente onde está inserida” (RIBEIRO, 2001).

> “A evolução dos robôs, e em especial dos robôs móveis, tem recebido nos últimos anos um amplo destaque junto à mídia e à sociedade de um modo geral. Onde no passado se falava muito em robôs industriais e braços mecânicos robóticos, atualmente as atenções se voltaram para robôs móveis, capazes de se deslocar no ambiente em que se encontram, e principalmente nos Robôs Móveis Autônomos – RMAs e Veículos Autônomos Inteligentes” (JUNG, 2005).

3.1.1 Aplicações de robôs móveis

O estudo da robótica móvel é um tema bastante relevante e atual, onde esta área de estudos, pesquisas e desenvolvimento apresentou um grande salto em seu desenvolvimento nas últimas duas décadas. A aplicação prática de robôs móveis junto a diferentes atividades em nossa sociedade vem demonstrando o quão promissor é o futuro desta área. Exemplos destas aplicações são:

- **Domésticas** - aspiradores de pó e cortadores de grama robóticos;
- **Industriais -** transporte automatizado e veículos de carga autônomos;
- **Urbanas** - transporte público e cadeiras de rodas robotizadas;
- **Militares -** sistemas de monitoramento aéreo remoto - VANTs, transporte de suprimentos e de armamento em zonas de guerra, sistemas táticos e de combate;
- **Segurança e defesa civil e militar** - controle e patrulhamento de ambientes, resgate e exploração em ambientes hostis.

Pode-se citar aqui alguns exemplos famosos de Robôs Móveis, resultantes da pesquisa e desenvolvimento que vem ocorrendo nesta área: os robôs de exploração espacial como o *Mars Pathfinder's Sojourner, Spirit* e *Opportunity Rovers* (Bajracharya, 2008); robôs domésticos usados para limpar a casa como o *Roomba e Scooba* (iRobot, 2009) (Fig. 3.1a), e para cortar grama como o AutoMower (Huskvarna, 2009; Sahin, 2007) (Fig. 3.1b); os robôs com pernas capazes de caminhar como o cachorro Aibo (Sony, 2009) (Fig 3.1c), o humanoide Asimo (Honda, 2009) (Fig. 3.1d) e o *BigDog* (Boston Dymanics, 2009); veículos terrestres não tripulados como o Stanley de Stanford (Thrun, 2006; Gibbs, 2006; Jung, 2005), que competiu e venceu o Darpa *Challenge* em 2005; e veículos aéreos não tripulados (UAVs) como os VANTs brasileiros do Projeto Arara (Neris, 2001) e AGplane (AGX 2009). Estes exemplos demonstram claramente os progressos e resultados da pesquisa e desenvolvimento em robótica móvel desta última década.

Figura 3. 1: Exemplos de Robôs Móveis.

Fonte: WOLF (2009).

3.1.2 Classificações de robôs móveis

O foco principal dos experimentos realizados no trabalho será o robô diferencial, porém aplicações voltadas a outros robôs podem utilizar a mesma metodologia, pois esta será feita de modo generalizado, apenas recebendo coordenadas do modelo dinâmico implementado em um *software* externo. Desta forma, para outros tipos de robô, com diferentes atuadores e sensores, seria necessário somente adicionar as informações desejadas ao seu modelo e desta forma utilizar o ambiente.

Os Robôs Móveis Autônomos, como se pode constatar pelos exemplos da Figura 3.1, possuem diferentes configurações de dispositivos de hardware embarcados, de acordo com a função e as tarefas para as quais são projetados. Os principais dispositivos de Hardware de um robô são seus sensores e atuadores. A Tabela 3.1 apresenta uma lista de robôs de acordo com seus atuadores (DUDEK, 2000).

Tabela 3.1: Exemplos de robôs móveis para diferentes atuadores.

Atuador	Principal Tipo/Função	Exemplos
Base Fixa	Braço robótico com base fixa	Robôs industriais PUMA
Base Móvel: Rodas	2 Rodas independentes (diferencial)	Robôs Khepera e Pioneer P3-DX
	3 Rodas (triciclo, omni-directionais)	Robô BrainStem PPRK
	4 Rodas (veículos robóticos - ackermann)	Stanley - Stanford (Darpa Challenge)
Base Móvel: Esteira	Esteira (Slip/Skid locomotion - tracks)	Tanques e veículos militares
Base Móvel: Juntas e	Bípedes	Robôs Humanóides
Articulações	4 Patas (quadpods)	Robôs Sony Aibo, BigDog
	6 Patas (hexapods)	Robôs Inseto (Lynxmotion Hexapods)
Base Móvel: Propulsão	Veículos aéreos com hélices	Aviões, Helicópteros e Dirigíveis
Hélices ou Turbinas	Veículos aquáticos com hélices	Barcos autônomos
	Veículos sub-aquáticos	Submarinos autônomos
Outros tipos	Braços manipuladores com base móvel	Garras (Grippers) embarcadas
	Garras com ou sem feed-back sensorial	Mão robótica
	Mecanismos de disparo	Disparo do chute (futebol de robôs)

Fonte: DUDEK (2000).

Hardware de um robô são seus sensores e atuadores. A Tabela 3.2 apresenta uma lista dos sensores mais usados em robótica (DUDEK, 2000). As Tabelas 3.1 e 3.2 demonstram o quão complexo pode ser o projeto de um sistema robótico, que envolve a especificação e seleção de diferentes componentes sensores e atuadores, cada um com suas especificidades, e a combinação destes em um sistema autônomo. Este sistema deve ser projetado de modo a ser dotado de dispositivos capazes de prover os dados necessários obtidos através dos seus sensores, para que o sistema de controle robótico inteligente possa planejar e realizar o acionamento dos seus dispositivos de modo a executar a ação desejada.

Tabela 3.2: Sensores para robôs móveis.

Sensor	Principal Função	Exemplos
De Posição e Orientação	Determinar a posição absoluta	GPS (Sistema de Posicionamento Global)
	ou direção de orientação do robô	Bússola [Compass]
		Inclinômetro
		Triangulação usando marcas (Beacons)
De Obstáculos	Determinar a distância até um objeto	Sensor Infra-Vermelho (IR - Infrared)
	ou obstáculo	Ultrassom (Sonar)
		Radar
		Sensor Laser (Laser rangefinder)
		Sistemas de Visão Estéreo (Stereo Vision)
De Contato	Determinar o contato com um objeto	Sensores de Contato (Bumbers, Switches)
	ou posição de contato com marcação	Antenas e "bigodes" (Animal whiskers)
		Marcações (barreiras óticas e magnéticas)
De Deslocamento	Medir o deslocamento do robô	Inercial (Giroscópio, Acelerômetros)
e Velocidade	Medidas relativas da posição e	Odômetro (Encoders: Optical, Brush)
	orientação do robô	Potenciômetros (Angular)
		Sensores baseados em Visão
Para Comunicação	Envio e recepção de dados e sinais	Sistemas de Visão e Sensores Óticos
	externos (troca de informação)	Sistemas de Comunicação (RF)
Outros tipos	Sensores magnéticos, indutivos, capacitivos, reflexivos	
	Sensores de temperatura, carga (bateria), pressão e força, etc.	
	Detectores: detector de movimento, de marcações, de gás/odores	

Fonte: DUDEK (2000).

Um dos grandes desafios da robótica é justamente como integrar as informações vindas de todos estes sensores, de modo a gerar comandos e controlar os diferentes dispositivos de atuação do robô, garantindo que a tarefa seja executada de modo correto e sem colocar em risco tanto o robô quanto aqueles que o cercam.

O robô deve preservar a sua integridade bem como dos elementos presentes nos ambientes (seres humanos, objetos como utensílios e mobiliário, outros robôs). Um robô só deve agir de modo ativo sobre um determinado objeto-alvo, se realmente for programada a sua ação sobre este objeto.

Esta questão sempre foi alvo de muitas discussões em propostas como as 3 Leis da Robótica de Asimov (1968), pois um robô móvel pode causar danos tanto a pessoas como a objetos que o cercam, de modo deliberado ou não, constituindo-se assim de um Sistema Embarcado Crítico (executa missões críticas), onde seu desenvolvimento deve imperativamente envolver um cuidadoso projeto tanto de hardware como de Software.

Em função desta execução de tarefas críticas, que podem causar tanto danos em humanos como materiais, o projeto do sistema de controle de um robô móvel deve garantir um controle robusto, tolerante a falhas e a situações imprevistas, constituindo assim um sistema de controle inteligente. Além disso, o próprio projeto do sistema de controle do robô pode apresentar o perigo de danos ao robô e ao seu redor, assim tornando o trabalho muito mais difícil.

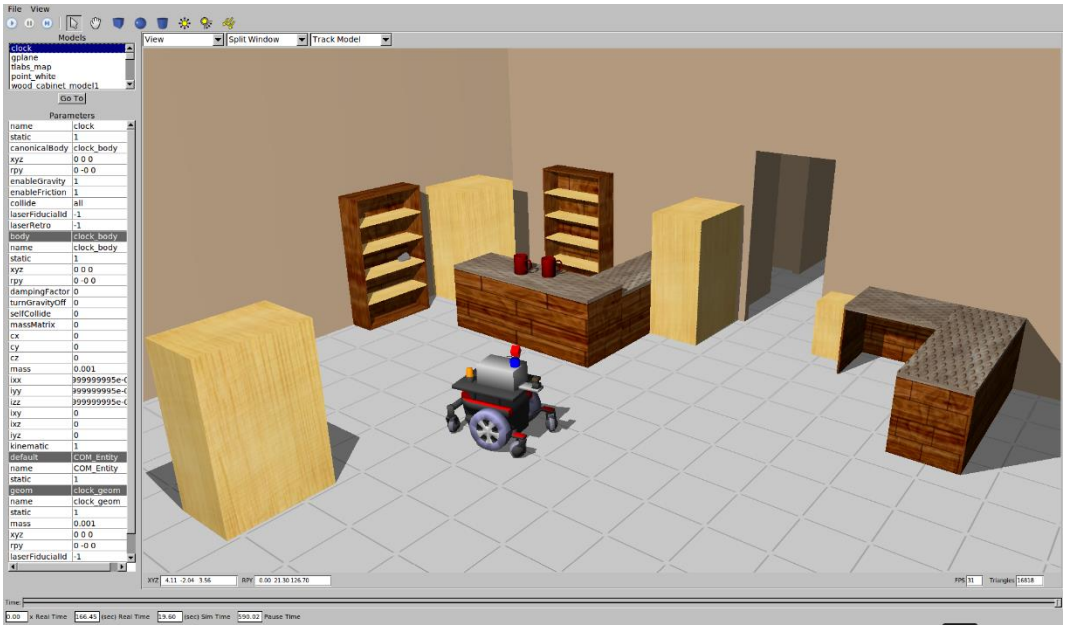
3.2 Simuladores de robôs móveis

Este tópico visa expor alguns dos principais ambientes de simulação utilizados atualmente, dando uma breve descrição. A escolha vai ser feita de acordo com o número de publicações encontradas fazendo uso dos ambientes, porém, visto que robótica é uma área em grande desenvolvimento em vários países, serão focadas as plataformas de simulação que tiverem maior relação com os aspectos do trabalho.

3.2.1 Gazebo

O Gazebo (Gazebosim, 2015) é um simulador robótico aberto, que permite manipular de forma eficiente e precisa populações de robôs em ambientes complexos internos e externos. Utiliza a *engine* OGRE para gerar ambientes com luzes, sombras e texturas. Gera dados de sensores, opcionalmente com ruído, para lasers, câmeras 2D/3D, sensores estilo Kinect, de contato e mais. Na Figura 3.2 pode ser visto um exemplo de interface para este simulador.

Figura 3. 2: Exemplo de ambiente Gazebo.



Fonte: *Traclabs* (2015).

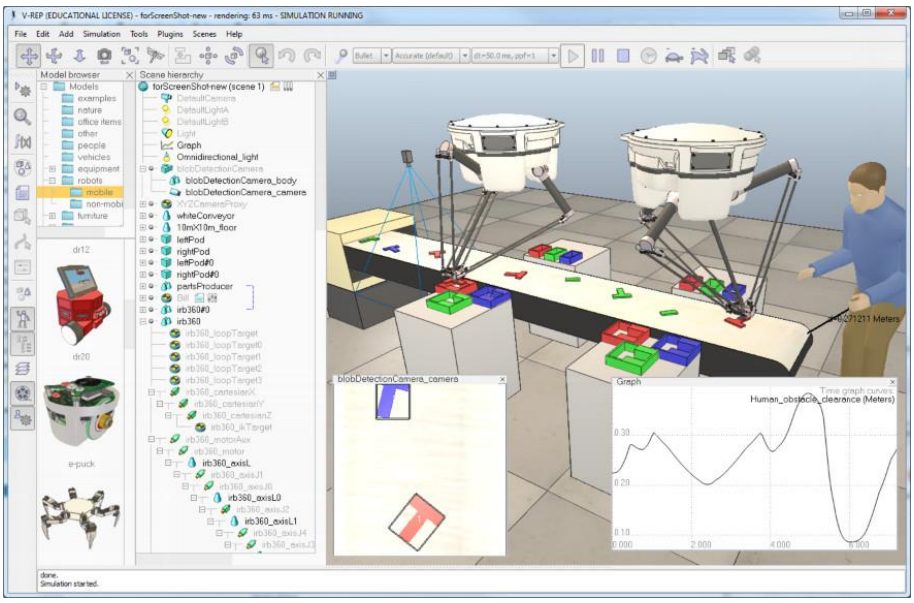
Podem ser desenvolvidos *plug-ins* com acesso direto à sua API, contém vários modelos de robôs prontos na sua biblioteca, pode utilizar o protocolo TCP/IP para simulação remota, e possui a opção de executar comandos por um terminal, facilitando o controle.

3.2.2 V-REP

O V-REP é um simulador robótico que possui a versão aberta e pró, porém, a versão paga é disponibilizada gratuitamente para estudantes e pesquisadores. O programa é baseado

na arquitetura de controle distribuído, ou seja, cada objeto pode ser controlado individualmente, por *script* embarcado, um *plugin*, um *add-on/Remote API*, ou uma solução personalizada usando o protocolo TCP/IP, UDP, serial, entre outros. Na Figura 3.3, pode ser visto um exemplo da interface no ambiente de desenvolvimento.

Figura 3. 3: Exemplo de ambiente V-REP.



Fonte: Coppelia (2015).

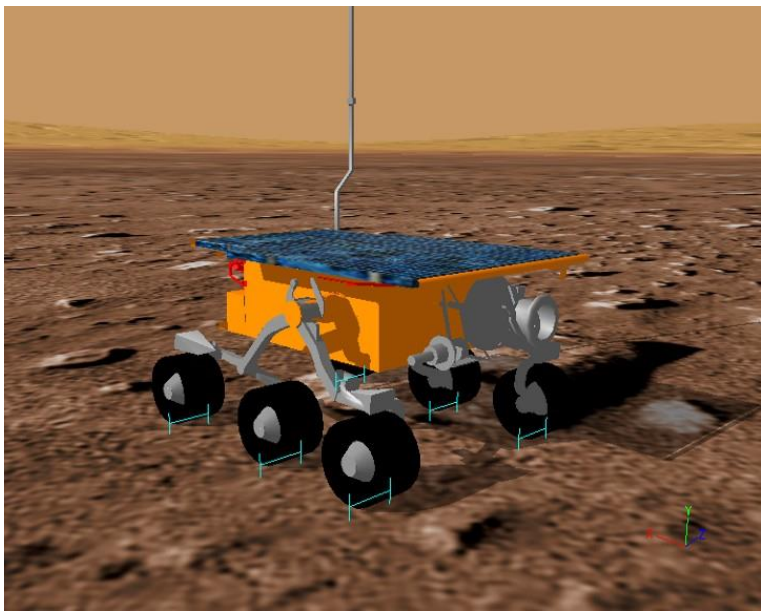
O modo de operação do aplicativo, faz com que seja ideal para aplicações utilizando múltiplos robôs. Seus controladores podem ser escritos em C, C++, Python, Java, Lua, Matlab, Octave ou Urbi, e também realizam a conexão com OpenCV, ou outras bibliotecas externas. O V-REP é muito utilizado para um desenvolvimento rápido de algoritmos, simulações de automação fabril e educação relacionada à robótica.

3.2.3 Webots

O Webots é um *software* pago, utilizado para diversos tipos de simulações na área da robótica. Possui a possibilidade de especificar limiares de colisão independente da sua representação gráfica, seus arquivos são legíveis por humanos e podem ser modificados em

simples arquivos de texto, além disso, contém diversos modelos de robôs em sua biblioteca. A Figura 3.4 contém um exemplo de ambiente gerado neste simulador.

Figura 3. 4: Exemplo de ambiente Webots.



Fonte: Cyberbotics (2015).

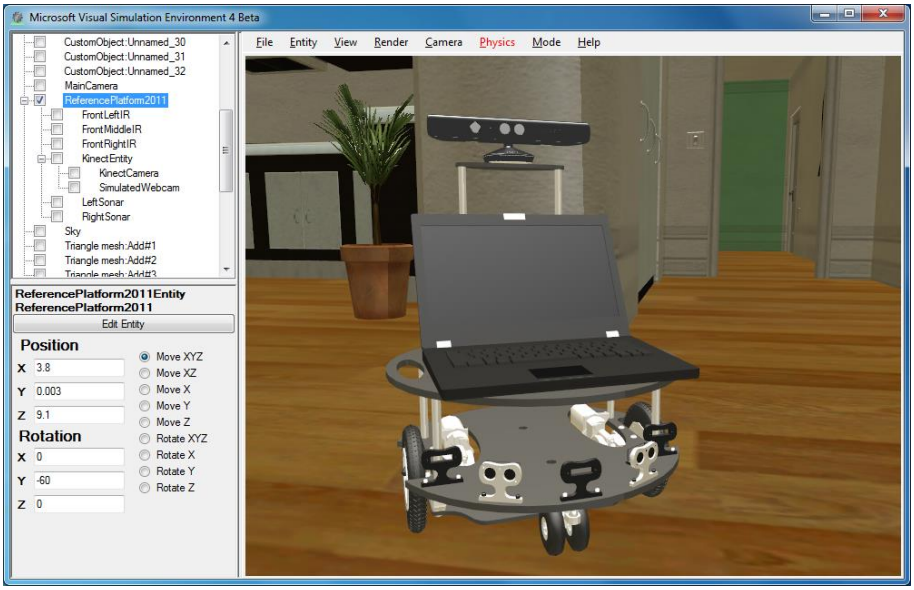
Os *scripts* podem ser escritos em C, C++, Python, Matlab e URBI, possuindo conexão com bibliotecas externas, como OpenCV. O processo de simulação pode ser controlado de forma programada, onde o programa supervisor pode fazer capturas da tela, produzir vídeos, gravar trajetórias, mover objetos, e alterar as propriedades dos objetos enquanto a simulação está rodando.

3.2.4 *Microsoft Robotics Developer Studio*

Microsoft® *Robotics Developer Studio* 4 habilita amadores e desenvolvedores profissionais, ou não, para criar aplicações robóticas visando um grande número de cenários. Por ser um aplicativo da Microsoft, possui uma serie de aplicações envolvendo o seu sensor Kinect, o programa pode suportar vários tipos de plataformas robóticas, podendo rodar diretamente da plataforma, ou se tiver um *PC (Personal Computer)* embarcado com Windows,

ou controlando através de um canal de comunicação, como Wifi ou Bluetooth®. A Figura 3.5 apresenta um exemplo da tela de desenvolvimento deste simulador.

Figura 3. 5: Exemplos de Robôs Móveis.



Fonte: I-MSDN (2015).

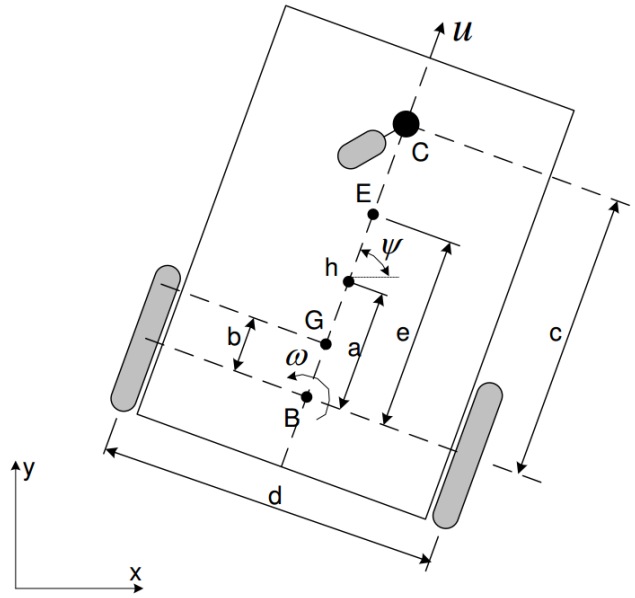
Além de dar suporte para o *Microsoft Visual Studio* 2010, o aplicativo apresenta uma linguagem de programação visual, que permite a desenvolvedores criar aplicações simplesmente arrastando e soltando componentes em espações e ligando-os. Este programa também consegue criar um ambiente de simulação virtual utilizando a *engine* NVIDIA™ PhysX™ para simulações de jogos 3D com iterações físicas do mundo-real.

3.3 Modelos em Matlab/Simulink

Aplicações de navegação robótica contam com a habilidade do robô de estimar o seu caminho atual, e com a decisão do controlador para realizar as correções do movimento para permanecer na trajetória. Comportamentos tais como se movimentar para uma posição especifica ou desviar de um obstáculo precisam de uma estimativa confiável das coordenadas reais do robô para funcionar corretamente. Esta seção apresenta um modelo dinâmico de um robô para geração de trajetória.

A Figura 3.6 mostra a representação do robô móvel do tipo uniciclo, seus parâmetros e variáveis de interesse. Na figura, u e ω são, respectivamente, as velocidades linear e angular desenvolvidas pelo robô, G é seu centro de massa, C é a posição de sua roda castor, E é a posição de uma ferramenta a bordo do robô (não será considerada), h é o ponto de interesse (de coordenadas x e y no plano XY), ψ é a orientação do robô, e *a* é a distância entre o ponto de interesse e o ponto central do eixo virtual que conecta as rodas de tração (ponto B).

Figura 3. 6: Representação de robô diferencial.



Fonte: MARTINS (2009).

Os modelos dinâmicos desenvolvidos por De La CRUZ (2006) foram baseados no modelo proposto por Zhang et al. (1998), que apresentam como sinais de entrada os valores de torque a serem aplicados às rodas esquerda e direita. No entanto, robôs comerciais geralmente aceitam comandos de velocidades linear e angular, e não de torque para seus motores.

Neste contexto, em (De La CRUZ, 2006) foram propostos dois modelos para robôs móveis tipo uniciclo que levam em conta sua cinemática e sua dinâmica. Num deles os sinais de entrada são as tensões aplicadas aos motores das rodas esquerda e direita (considerando um robô com tração diferencial), e no outro são as referências de velocidade linear e angular do robô. O modelo completo cujos sinais de entrada são as tensões nos motores pode ser escrito pela equação (3.1).

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\psi}\\ \dot{u}\\ \dot{\omega}\end{bmatrix}=\begin{bmatrix}\cos\psi & -a\sin\psi\\ \sin\psi & a\cos\psi\\ 0 & 1\\ \frac{\theta_3^0}{\theta_1^0}\frac{r^2\omega^2}{u} & -\frac{2u}{\omega}\frac{\theta_4^0}{\theta_1^0}\\ -2r^2\omega\frac{\theta_3^0}{\theta_2^0} & -\frac{\theta_4^0}{\theta_2^0}d^2\omega\end{bmatrix}\begin{bmatrix}u\\ \omega\end{bmatrix}+\begin{bmatrix}0 & 0\\ 0 & 0\\ 0 & 0\\ \frac{2r}{\theta_1^0} & 0\\ 0 & \frac{2rd}{\theta_2^0}\end{bmatrix}\begin{bmatrix}\frac{v_r+v_l}{2}\\ \frac{v_r-v_l}{2}\end{bmatrix}+\begin{bmatrix}\delta_x\\ \delta_y\\ 0\\ \bar{\delta}_\mu\\ \bar{\delta}_\omega\end{bmatrix} \quad (3.1)$$

Onde, $\delta=[\delta_x,\delta_y,0,\bar{\delta}_\mu,\bar{\delta}_\omega]$, é o vetor de incertezas associado ao robô móvel, em que δ_x e δ_y são funções das velocidades de deslizamento e da orientação do robô, $\bar{\delta}_\mu$ e $\bar{\delta}_\omega$ são funções de parâmetros físicos como massa, inércia, diâmetros das rodas e pneus, parâmetros dos motores e servos, forças nas rodas, entre outros, v_r e v_l, são as tensões aplicadas nas rodas da esquerda e direita respectivamente. Os parâmetros do modelo representados por $\theta_1^0,\theta_2^0,\theta_3^0,\theta_4^0$, podem ser vistos nas equações (3.2), (3.3), (3.4) e (3.5).

$$\theta_1^0=\frac{R_a}{k_a}(mr^2+2I_e)[V.s^2] \quad (3.2)$$

$$\theta_2^0=\frac{R_a}{k_a}\left(I_ed^2+2r^2(I_z+mb^2)\right)[V.m^2.s^2] \quad (3.3)$$

$$\theta_3^0=\frac{R_a}{k_a}mb[V.s^2/m] \quad (3.4)$$

$$\theta_4^0=\frac{R_a}{k_a}\left(\frac{k_ak_b}{R_a}+B_e\right)[V.s/rad] \quad (3.5)$$

Onde Ra $[V/A]$ é a resistência elétrica dos motores, kb $[V.s/rad]$ é sua constante eletromotriz multiplicada pela relação das engrenagens, ka $[N.m/A]$ é sua constante de torque multiplicada pela relação das engrenagens, Be $[N.s/rad]$ é o coeficiente de atrito viscoso rotacional, m $[kg]$é a massa do robô, Iz $[kg.m^2]$ é seu momento de inércia no ponto G, Ie $[kg.m^2]$ é o momento de inércia de cada grupo rotor-redução-roda, r$[m]$ é o raio de suas rodas e b$[m]$ e d$[m]$ são as distâncias mostradas na Figura 3-6.

A equação anterior é útil nos casos em que se pode controlar diretamente as tensões aplicadas em cada motor do robô. No entanto, robôs comerciais geralmente possuem controladores internos que recebem referências de velocidades para cada motor, e não permitem que suas tensões sejam controladas diretamente. Neste contexto, em (De La CRUZ, 2006) foi

considerado que os controladores internos são do tipo PD (Proporcional Derivativo), com ganhos proporcionais $k_{PT} > 0$ e $k_{PR} > 0$, e derivativos $k_{DT} \geq 0$ e $k_{DR} \geq 0$. Assim obteve-se a relação da equação (3.6).

$$\begin{bmatrix} v_u \\ v_\omega \end{bmatrix} = \begin{bmatrix} k_{PT}(u_{ref} - u) - k_{DT}\dot{u} \\ k_{PR}(\omega_{ref} - \omega) - k_{DR}\dot{\omega} \end{bmatrix} \tag{3.6}$$

Os valores de u e ω são definidos pelas equações (3.7) e (3.8), respectivamente.

$$u = \frac{1}{2}[r(\omega_r + \omega_l)] \tag{3.7}$$

$$\omega = \frac{1}{d}[r(\omega_r - \omega_l)] \tag{3.8}$$

Sendo ω_r e ω_l, as velocidades angulares das rodas esquerda e direita, respectivamente. A partir dessas equações, o modelo foi reescrito na equação (3.9).

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \\ \dot{u} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} \cos\psi & -a\sin\psi \\ \sin\psi & a\cos\psi \\ 0 & 1 \\ -\frac{\theta_4}{\theta_1} & \frac{\theta_3}{\theta_1}\omega \\ -\frac{\theta_5}{\theta_2}\omega & -\frac{\theta_6}{\theta_2} \end{bmatrix} \begin{bmatrix} u \\ \omega \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ \frac{1}{\theta_1} & \frac{1}{\theta_2} \end{bmatrix} \begin{bmatrix} u_{ref} \\ \omega_{ref} \end{bmatrix} + \begin{bmatrix} \delta_x \\ \delta_y \\ 0 \\ \delta_u \\ \delta_\omega \end{bmatrix} \tag{3.9}$$

Onde u_{ref} e ω_{ref} são os sinais de referência de velocidades linear e angular, respectivamente, $\theta = [\theta_1 \;\; \theta_2 \;\; \theta_3 \;\; \theta_4 \;\; \theta_5 \;\; \theta_6]^T$ é o vetor de parâmetros do modelo e $\delta = [\delta_x \;\; \delta_y \;\; 0 \;\; \delta_u \;\; \delta_\omega]^T$ é o vetor de incertezas associado ao modelo do robô móvel. Como no modelo anteriormente apresentado, δ_x e δ_y são funções das velocidades de deslizamento e da orientação do robô, enquanto δ_u e δ_ω são funções de parâmetros físicos.

Os valores de $\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6$, são representados pelas equações (3.10), (3.11, (3.12, (3.13), (3.14), (3.15), respectivamente.

$$\theta_1 = \left[\frac{R_a}{k_a}(mr^2 + 2I_e) + 2rk_{DT}\right]\frac{1}{2rk_{PT}}[s] \tag{3.10}$$

$$\theta_2 = \frac{\left[\frac{R_a}{k_a}\left(I_e d^2 + 2r^2(I_z + mb^2)\right) + 2rdk_{DR}\right]}{2rdk_{PR}}[s] \tag{3.11}$$

$$\theta_3 = \frac{R_a}{k_a}\frac{mbr}{2k_{PT}}[s.m/rad^2] \tag{3.12}$$

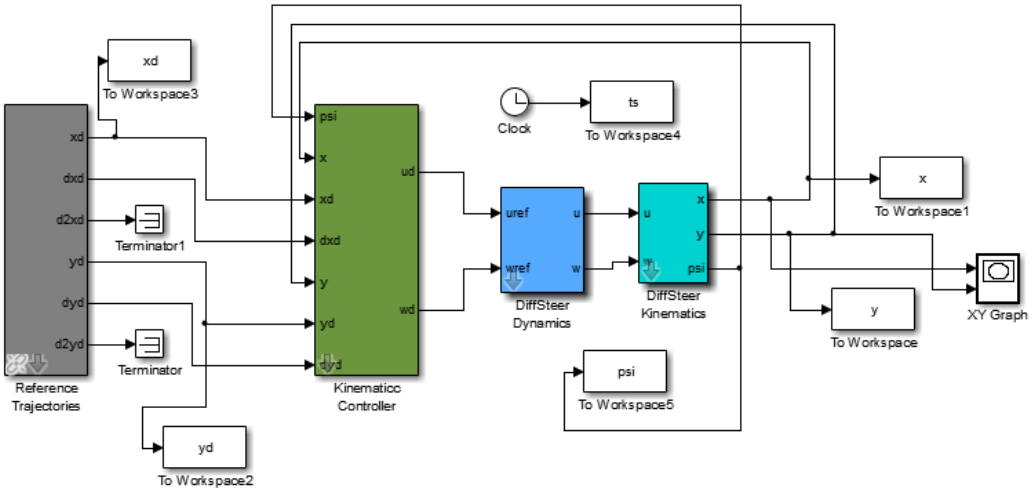
$$\theta_4 = \frac{R_a}{k_a}\left(\frac{k_a k_b}{R_a} + B_e\right)\frac{1}{rk_{PT}} + 1 \tag{3.13}$$

$$\theta_5 = \frac{R_a}{k_a}\frac{mbr}{dk_{PR}}\ [s/m] \tag{3.14}$$

$$\theta_6 = \frac{R_a}{k_a}\left(\frac{k_a k_b}{R_a} + B_e\right)\frac{d}{2rk_{PR}} + 1 \tag{3.15}$$

De acordo com o modelo descrito na equação (3.9), foi disponibilizado por Martins (2009), o diagrama de blocos para o simulink, desta forma facilitando a sua implementação no Matlab. Na Figura 3.7, pode ser visto o esquema geral do diagrama, onde pode ser notado, que além da geração da trajetória, também pode ser feito seu controle, porém este é um recurso que não irá ser explorado no trabalho.

Figura 3. 7: Diagrama de blocos para robô diferencial (Matlab/Simulink).



Fonte: MARTINS (2009).

No diagrama de blocos da Figura 3.7, deve-se determinar a trajetória de referência $(x_{ref}, y_{ref}, \psi_{ref})$, para que então a partir destes valores sejam gerados u_{ref} e ω_{ref}, e seja feita a compensação de acordo com a dinâmica do robô, gerando suas velocidades linear e angular (u, ω), e pelo bloco cinemático, sua coordenada já compensada com a orientação (x, y, psi).

Esta representação é possível pois, a equação (3.9) pode ser dividida no modelo dinâmico, equação (3.16), e no modelo cinemático, equação (3.17).

$$\begin{bmatrix} \dot{u} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} -\frac{\theta_4}{\theta_1} & \frac{\theta_3}{\theta_1}\omega \\ -\frac{\theta_5}{\theta_2}\omega & -\frac{\theta_6}{\theta_2} \end{bmatrix} \begin{bmatrix} u \\ \omega \end{bmatrix} + \begin{bmatrix} \frac{1}{\theta_1} & 0 \\ 0 & \frac{1}{\theta_2} \end{bmatrix} \begin{bmatrix} u_{ref} \\ \omega_{ref} \end{bmatrix} + \begin{bmatrix} \delta_u \\ \delta_\omega \end{bmatrix} \tag{3.16}$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} \cos\psi & -a\sin\psi \\ \sin\psi & a\cos\psi \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ \omega \end{bmatrix} + \begin{bmatrix} \delta_x \\ \delta_y \\ 0 \end{bmatrix} \tag{3.17}$$

4 PROPOSTA DE AMBIENTE INTEGRADO DE SIMULAÇÃO

A proposta consiste no desenvolvimento de um ambiente de simulação 3D, que esteja integrado com o Matlab, com o objetivo de reproduzir a trajetória de robôs segundo seu modelo dinâmico. Neste caso o robô escolhido foi o do tipo diferencial, para isto foi implementado o diagrama de blocos no Simulink disponibilizado por Martins (2009). As etapas para o desenvolvimento do projeto serão a construção do ambiente e integração com o modelo dinâmico.

4.1 Construção do ambiente de simulação

A construção do ambiente de simulação 3D precisa ser muito bem planejada, pois a sua utilização será não só para ilustrar a movimentação do robô, como também, renderizar imagens a partir de reproduções virtuais de duas câmeras em cada posição da trajetória, para uma posterior utilização. Dessa forma deve-se decidir os tipos de lugares que se deseja reproduzir para então iniciar o trabalho de construção em si.

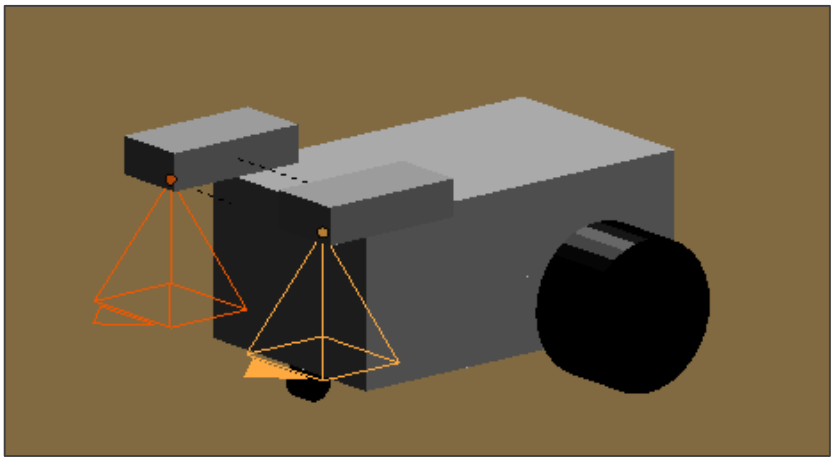
Para a criação do simulador, optou-se por reproduzir somente lugares fechados, como quartos, escritórios, galpões e outros. O motivo para isto é que atualmente as aplicações de ANBI's (Algoritmos de Navegação Baseados em Imagem) em ambientes fechados e estruturados, apresentam melhores resultados. Além deste fato, membros do time pesquisa estão trabalhando justamente neste tipo de ambiente, facilitando e ampliando a sua utilização.

Outro motivo, é que por estarem privados de fatores externos, e estando em piso liso não inclinado, se torna mais simples e confiável a reprodução da movimentação do robô, considerando somente a sua dinâmica. Sendo que, na geração de imagens também poderá ser desconsiderada a influência de luzes externas e movimentação de outros corpos no ambiente.

4.1.1 Robô Virtual

Como o robô não aparecerá nas imagens renderizadas, foi dispensada a utilização de texturas na sua reprodução, sendo assim, sua formação foi gerada pelo instanciamento de primitivas simples. Tendo como forma final a representada na Figura 4.1.

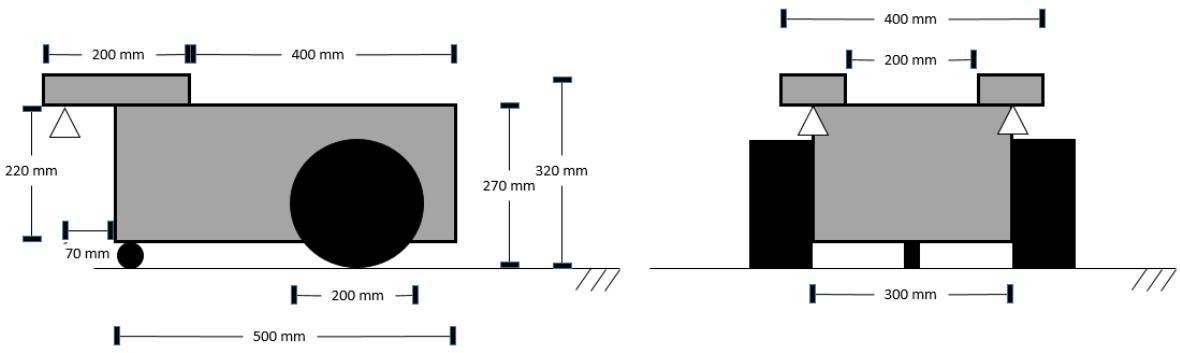
Figura 4. 1: Modelo para robô.



Fonte: Autor.

Para reprodução da movimentação, é necessário que se defina uma escala para o sistema métrico dentro ambiente, outro fator é que as medidas das dimensões do robô também afetarão as o resultado da simulação. Todas as medidas do robô podem ser vistas na Figura 4.2.

Figura 4. 2: Medidas do robô utilizado.



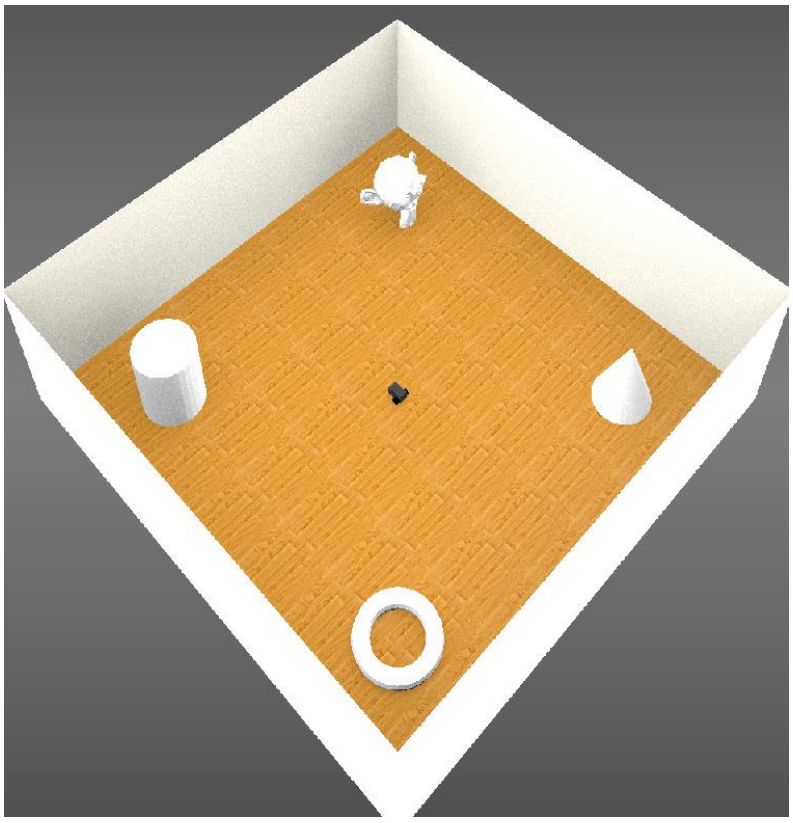
Fonte: Autor.

Pode-se perceber pelas Figuras 4.1 e 4.2, que a câmera fixada no robô aponta para baixo. Só foi realizado desta forma devido à necessidade de gerar imagens do chão durante o percurso. Isto se deve ao fato de outros trabalhos do time de pesquisa estarem utilizando-as, porém seria muito simples alterar a orientação da câmera, capturando imagens à frente do robô.

4.1.2 Ambiente 3D

Foram construídos três ambientes para a utilização integrada com o Matlab. O primeiro e o segundo ambiente foram feitos na integra, com o instanciamento de primitivas, estudo da iluminação, definição e mapeamento de texturas. Para facilitar referências futuras aos ambientes, estes serão nomeados, o ambiente da Figura 4.3 pode ser chamado ‘Quarto simples’, neste caso o objetivo foi a definição de metodologias, logo, para simplificar a análise de resultados, teve sua construção de forma limitada.

Figura 4. 3: Ambiente de simulação 'Quarto simples'.

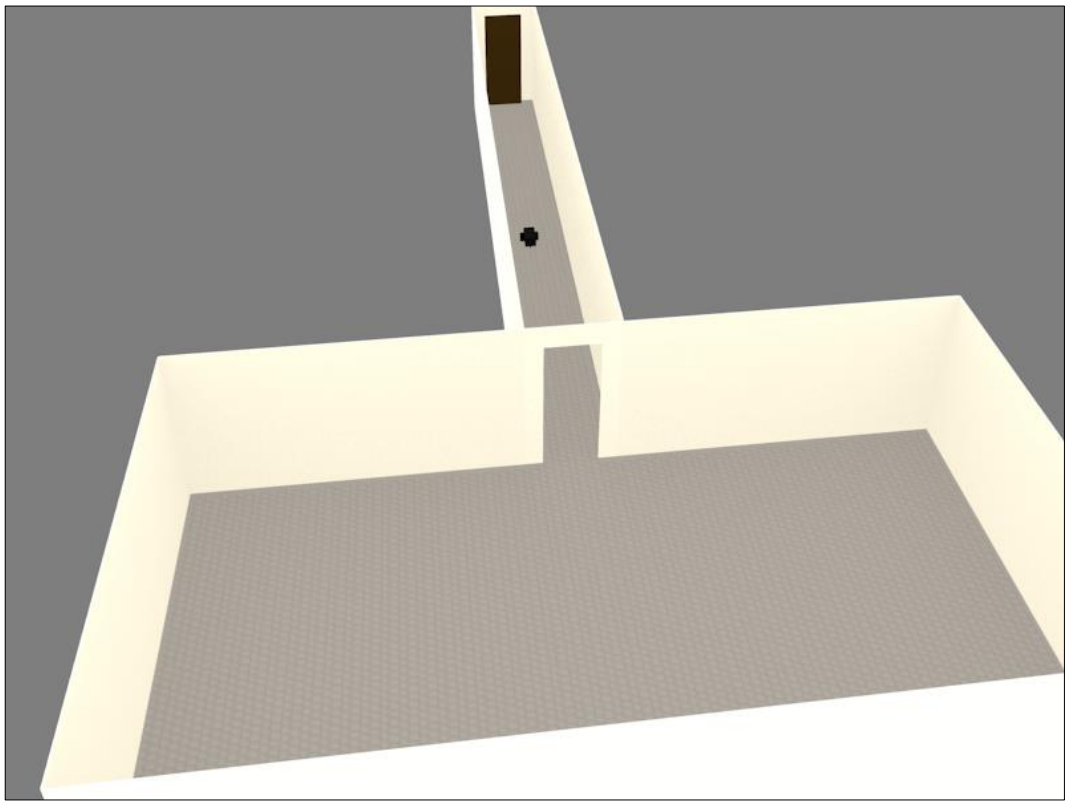


Fonte: Autor.

As dimensões do ambiente na Figura 4.3 são 10x10 metros, e as paredes possuem altura de 3 metros. Apesar de ser simples, o ambiente já apresenta texturas, assim gerando imagens de boa qualidade e por não ter muitos elementos, torna o processamento muito mais leve, assim agilizando a simulação.

O segundo ambiente pode ser chamado ‘Corredor’, neste caso as texturas foram criadas a partir de um ambiente real, logo, o objetivo foi deixar as paredes e chão o mais próximo possível, sendo assim, foi dispensada a utilização de objetos. O ambiente pode ser visto na Figura 4.4.

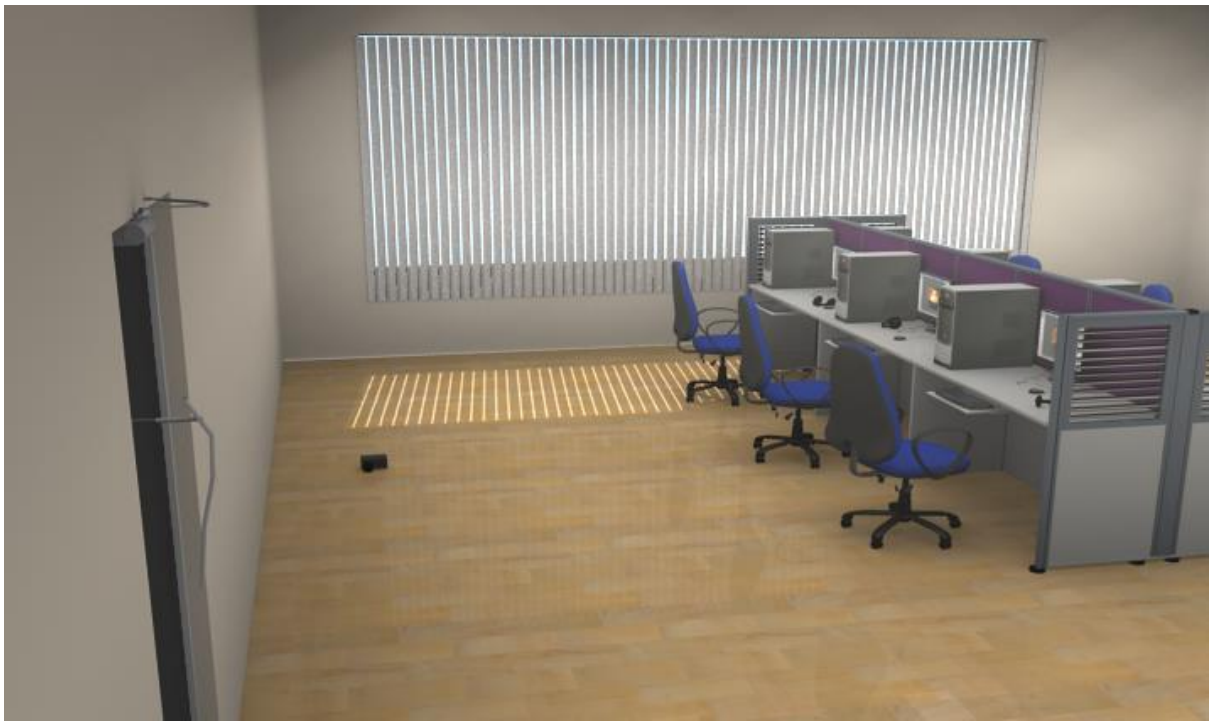
Figura 4. 4: Ambiente de simulação 'Corredor'.



Fonte: Autor.

Para o terceiro ambiente o nome é ‘Escritório’, nele foi utilizada uma base parcialmente desenvolvida, e então foram modificados aspectos de escala, métrica e nomenclatura de objetos, também foi feita inserção do robô e criação, posicionamento e ajuste de parâmetros das câmeras. O terceiro ambiente, que representa um escritório é mostrado na Figura 4.5.

Figura 4. 5: Ambiente de simulação 'Escritório'.



Fonte: Autor.

As dimensões do ambiente de simulação apresentado na Figura 4.5 são: 12 metros de comprimento, 9 metros de largura e as paredes com 3 metros de altura. Percebe-se claramente a melhor qualidade do ambiente, porém isso se reflete no alto tempo de processamento, e por este motivo, ao possuir ambos prontos, o leque de aplicações se torna mais amplo.

4.2 Integração com o modelo dinâmico

Possuindo os ambientes totalmente finalizados e configurados, o próximo passo é comunica-los com o modelo feito no Matlab, ou seja, configurar os parâmetros do modelo para que seja reproduzido o movimento do robô desejado e em seguida, enviá-los para o Blender.

4.2.1 Modelo dinâmico

O modelo utilizará as equações (3.16) e (3.17) para calcular as coordenadas compensadas de trajetória, logo, é necessário especificar os parâmetros desejados, assim como

citado no tópico 3.3. Por apresentar construção semelhante ao *Pioneer 3DX*, apresentado na Figura 4.6, serão utilizados seus valores para a reprodução dos movimentos de um robô diferencial.

Figura 4. 6: *Pioneer* P3-DX.

Fonte: Bielefeld University (2011).

Os valores para o modelo *Pioneer* 3DX, foram obtidos experimentalmente e divulgados por Martins (2009), estes são θ =[0.2604 0.2509 -0.000499 0.9965 0.00263 1.0768], seus valores são inseridos no bloco dinâmico do robô. Além disso, foi selecionada a velocidade linear máxima $u = 0.75\ m/s$, velocidade angular máxima $\omega = 1.75\ rad/s$, aceleração linear máxima $\dot{u} = 0.3\ m/s^2$ e aceleração angular máxima $\dot{\omega} = 0.3\ m/s^2$.

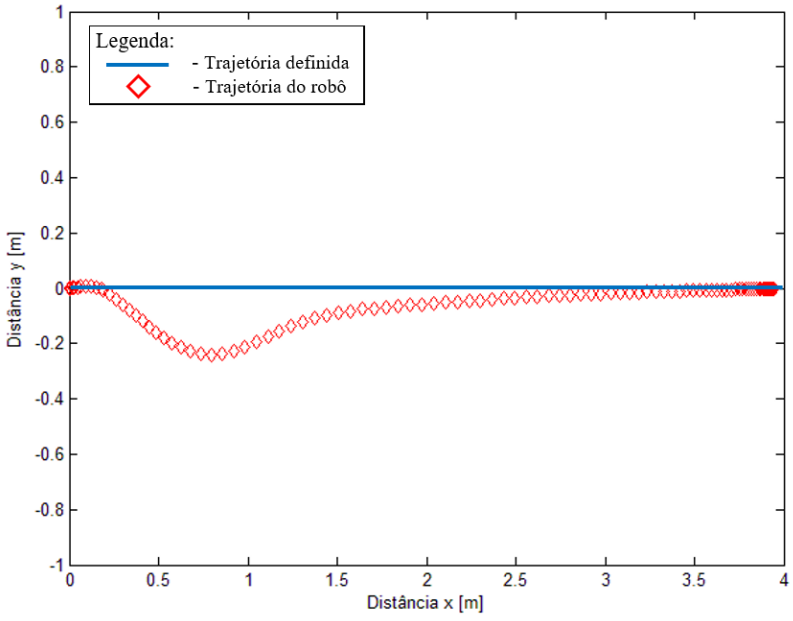
Além destes parâmetros, também deve-se especificar a distância entre as rodas do robô é $d = 300mm$, e a distância do centro do eixo até o ponto de interesse é de $a = 200mm$. Terminada esta parte deve-se determinar o tipo de trajetória desejada, as opções são em círculo, oito, linha ou ponto fixo, a opção selecionada é a última, esta escolha é totalmente arbitrária pois somente modificara o método de retorno dos valores das coordenadas.

4.2.2 Integração

Para movimentar o robô, primeiramente é necessário que se entre com a trajetória desejada no Matlab, na forma matricial [x1, y1; x2, y2; ...; xn, yn]. Os valores de xn e yn, são os pontos onde se deseja que o robô se posicione, ou seja, se a trajetória tiver mais de um ponto de destino, ele fará um de cada vez.

Para chegar na coordenada determinada, será utilizado o modelo do simulink apresentado na Figura 3-7, dessa forma, a trajetória será dividida em pequenos passos, e cada um terá uma posição (x, y, ψ). Um exemplo de trajetória é apresentado na Figura 4.7, para o robô na orientação inicial y+. Neste caso a trajetória seria somente para o ponto [4,0], ou seja, o robô sairia de (0,0,π/2) e iria para (4,0,0).

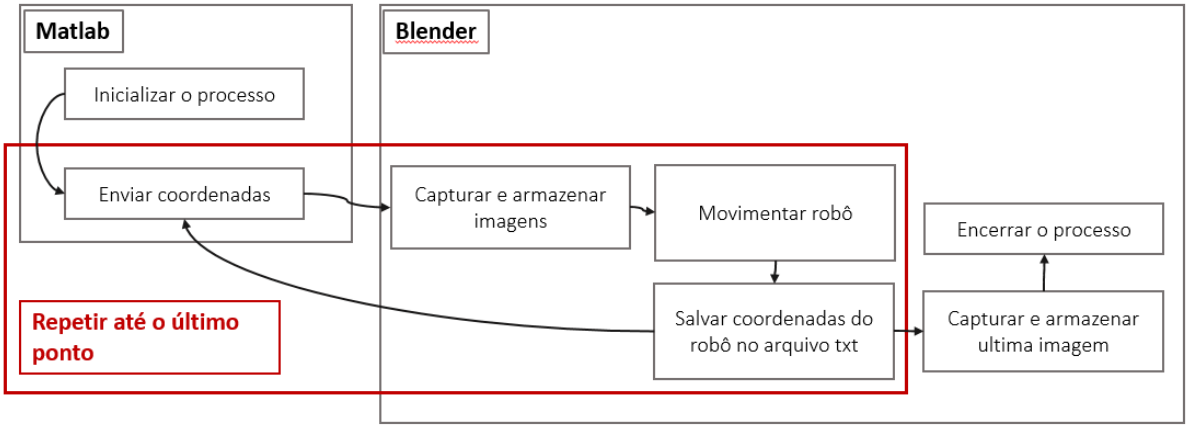
Figura 4. 7: Exemplo de trajetória.



Fonte: Autor.

Desse modo, será criada uma matriz, com todos os valores de (x, y, ψ) necessários para completar o caminho definido. Para facilitar o entendimento, é apresentado um diagrama de blocos explicando o funcionamento das etapas seguintes na Figura 4.8.

Figura 4. 8: Diagrama de funcionamento do projeto.



Fonte: Autor.

A etapa inicializar o processo, apresentada na Figura 4.8, consiste do que foi descrito anteriormente, ou seja, definição da trajetória e criação da matriz com as coordenadas (x,y) e orientação psi. Note, que apesar de ser um ambiente 3D, não é definida uma coordenada z, isso se deve ao fato de não haver elevações ou qualquer tipo de desnível no piso.

O próximo passo é enviar as coordenadas para um arquivo de texto, somente uma por vez, e alterar o valor que informa ao Blender, a disponibilidade de leitura do arquivo, assim que receber o aviso, o Blender irá capturar e salvar as imagens tiradas por ambas as câmeras, e em seguida se movimentará para a coordenada especificada, assumindo também a respectiva orientação.

Ao chegar na coordenada, o Blender, enviará sua posição para o arquivo de texto, e informará o Matlab que o arquivo está disponível. Assim que receber o aviso, o Matlab enviará o passo seguinte, para o arquivo de texto, e passará a vez para o Blender. Este processo descrito se repetirá até que a matriz gerada inicialmente pelo Matlab se esgote.

As imagens geradas são arquivadas em um diretório definido e apresentam resolução 720x580 e formato (*.JPEG), sendo possível acessa-las posteriormente. A Figura 4.9 apresenta um exemplo de imagem renderizada a partir do ambiente.

Figura 4. 9: Imagem renderizada do ambiente de simulação 'Quarto simples'.

Fonte: Autor.

Os parâmetros da câmera para a geração da Figura 4-9 são: distância focal 32 mm, e tamanho do sensor 35 x 28. mm. Para validar as imagens obtidas no ambiente de simulação, pretende-se aplicar técnicas de navegação por imagem desenvolvidas pelos companheiros de laboratório.

5 DETALHES DE IMPLEMENTAÇÃO

Para o melhor entendimento do trabalho, neste tópico serão abordados todos os detalhes desde o desenvolvimento do ambiente de simulação, até a metodologia para a sua validação. Os tópicos a seguir são respectivamente; criação de primitivas, onde serrão mencionados os sólidos base utilizados e quais edições foram necessárias; em seguida será abordado o instanciamento destas primitivas de forma a formar sólidos mais complexos; e a partir desta base para os ambientes, serão explicados os aspectos referentes à renderização, como a iluminação; e a textura.

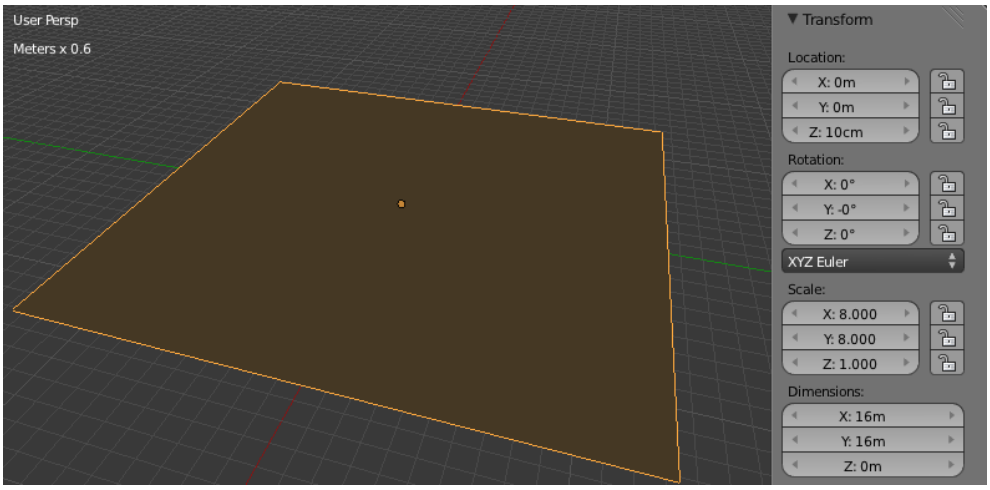
Tendo em vista que somente a primeira parte do trabalho se remeta à construção dos ambientes ainda resta o funcionamento dos scripts, portanto, os tópicos relacionados serão; composição de ambientes, onde algumas funções implementadas serão discutidas e exemplificadas; comunicação com softwares externos, onde será explicado como está sendo feita a interface com outros aplicativos utilizados; também será demonstrado como o modelo dinâmico gera os valores de coordenada e orientação; e por último serão relacionadas as metodologias para a validação de aspectos importantes deste simulador.

Cada um dos sub-tópicos irá discutir elementos presentes em cada um dos três ambientes de simulação.

5.1 Criação de primitivas

Para o ambiente simulação ‘Quarto simples’, o objetivo foi a construção da metodologia e estudo da plataforma Blender, as primitivas foram algumas das já disponibilizadas pelo programa. Para a definição do chão foi utilizado ‘*plane*’, que como pode ser visto na Figura 5.1, pela dimensão ‘z’ igual a ‘0’, seria um objeto de somente 2 dimensões, e foi escolhido pois neste ambiente não se teve o intuito de apresentar qualquer tipo de profundidade no solo.

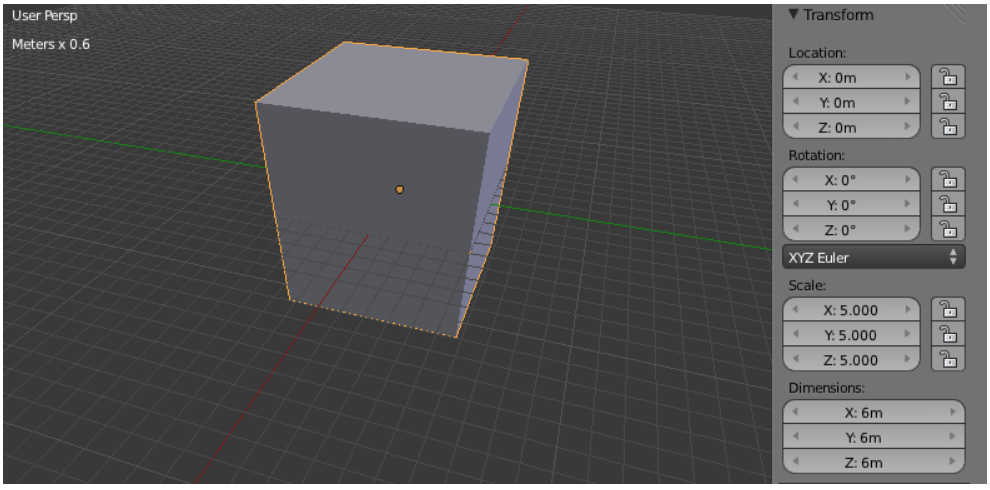
Figura 5. 1: Exemplo de primitiva '*plane*'.



Fonte: Autor.

Na Figura 5.2, pode ser cista a primitiva '*cube*', como o próprio nome já sugere, se trata de um cubo cujas dimensões podem ser modificadas. Foi utilizado, tanto no robô, quanto na construção das paredes. Estra primitiva é muito útil para o projeto pelo fato dele se basear em ambientes internos, onde estruturas com estas características regulares são mais comuns.

Figura 5. 2: Exemplo de primitiva '*cube*'.

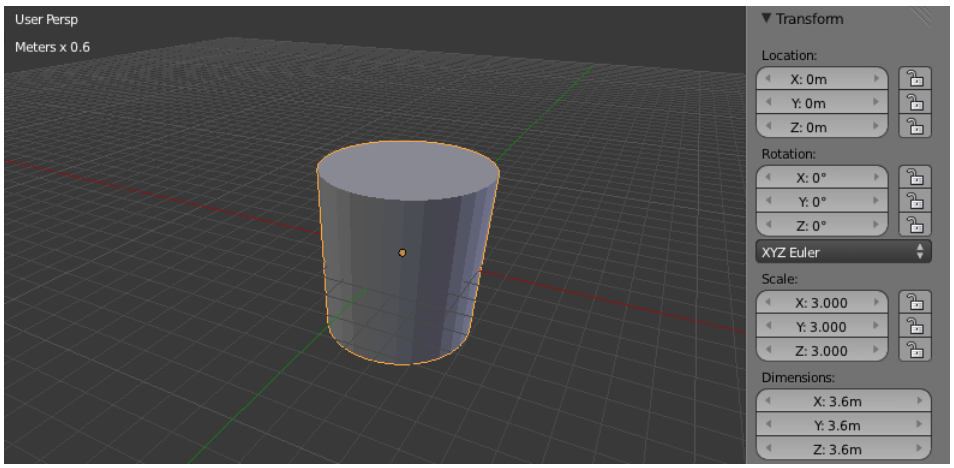


Fonte: Autor.

Outro objeto utilizado foi o '*cylinder*', tanto no ambiente somente para dar ideia de localização, como no robô para a representação das rodas. Está representado na Figura 5.3,

assim como a primitiva da Figura 5.2, apresenta 3 dimensões, neste caso também deve ser especificado o número de faces laterais do cilindro.

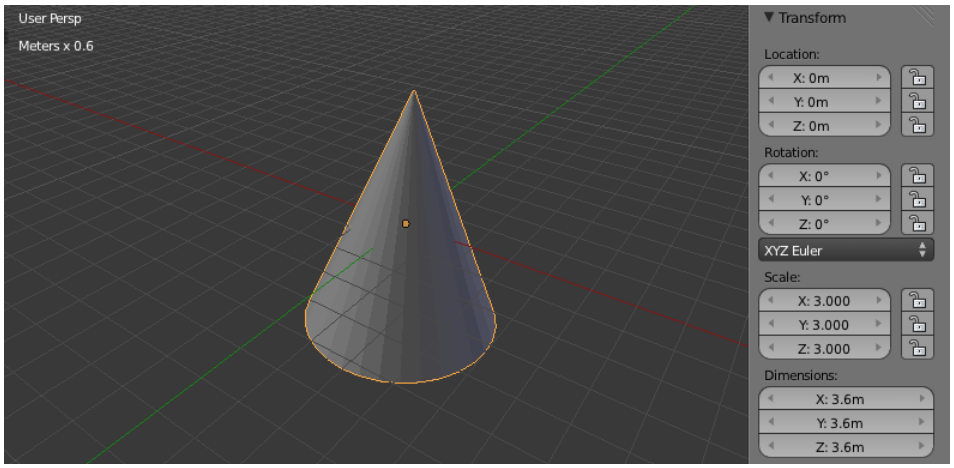
Figura 5. 3: Exemplo de primitiva '*cylinder*'.



Fonte: Autor.

Além do cilindro, outros objetos foram utilizados para dar ideia de orientação ao se mover pelo ambiente. Um deles é a primitiva '*cone*', Figura 5.4, que apresenta 3 dimensões e assim como o cilindro deve indicar o número de faces laterais.

Figura 5. 4: Exemplo de primitiva '*cone*'.

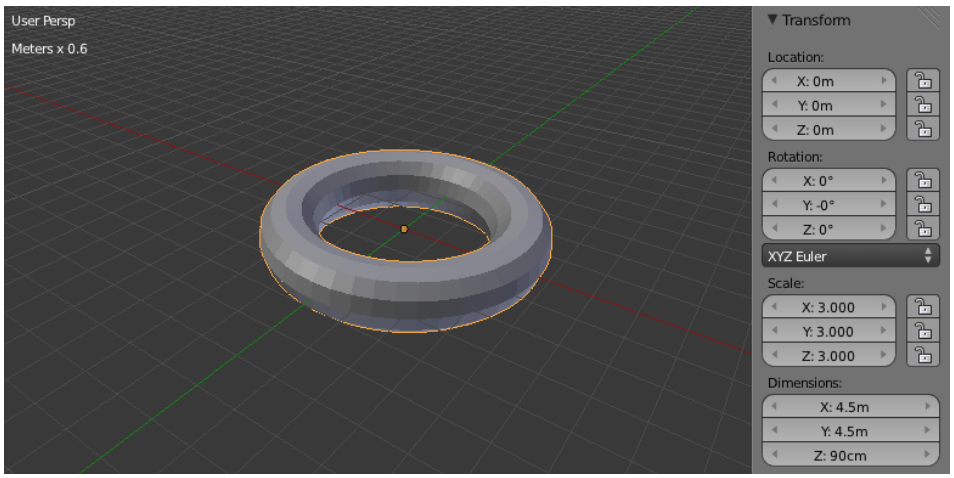


Fonte: Autor.

Outra primitiva é a '*torus*', esta primitiva aparenta o formato de rosca, desta forma é mais complexa do que as anteriores, para ela é necessário especificar o menor e o maior raio,

além do número de lados em cada anel e o número de anéis. A Figura 5.5 apresenta um exemplo deste objeto.

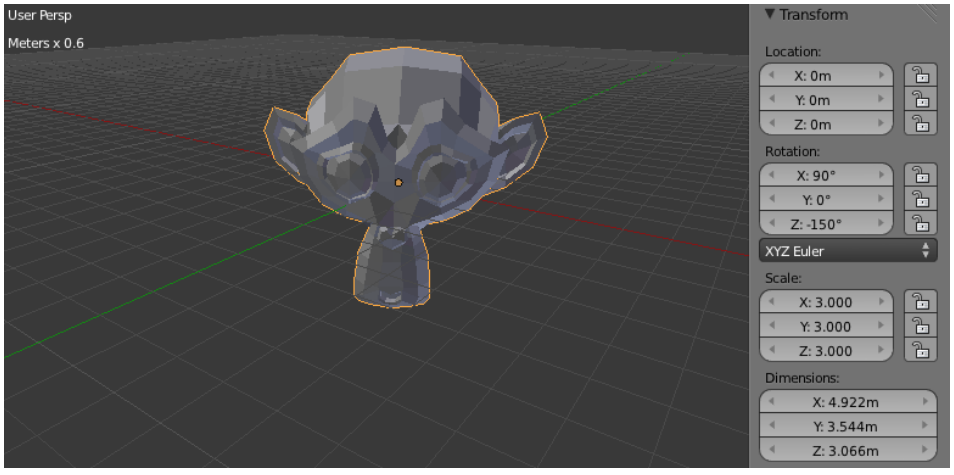
Figura 5. 5: Exemplo de primitiva '*torus*'.



Fonte: Autor.

A próxima primitiva, também utilizada somente para efeitos de localização, é a '*monkey*', neste caso o objeto é bem mais complexo e lembra a cabeça de um macaco. Não é necessário especificar nenhum parâmetro na sua geração, um exemplo pode ser visto na Figura 5.6.

Figura 5. 6: Exemplo de primitiva '*monkey*'.

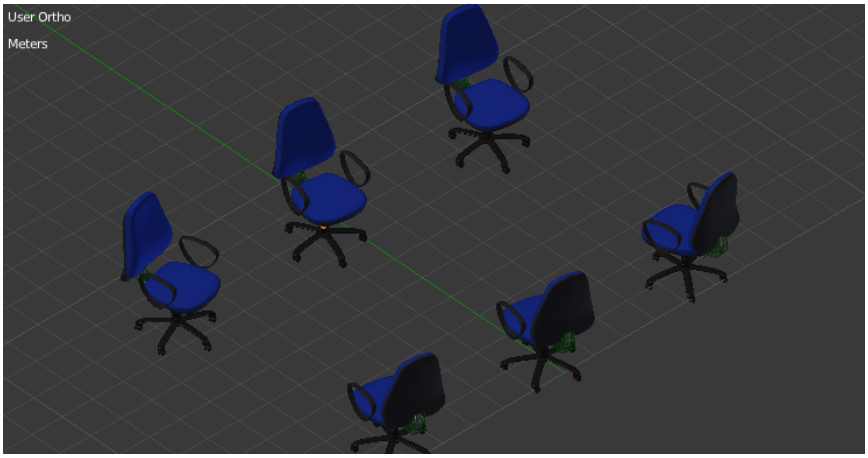


Fonte: Autor.

Para o primeiro ambiente, sendo o mais simples, as primitivas utilizadas já foram exemplificadas, porém, como já foi dito, o objetivo neste caso era estabelecer e testar uma metodologia de forma rápida, ou seja, com poucos detalhes e tempo de renderização. Além de primitivas simples, outros aspectos foram retirados para agilizar este processo, porém, isto será melhor discutido no decorrer do trabalho.

O segundo ambiente, ‘Escritório’, teve todas as primitivas e objeto importados diretamente de um modelo externo, e como o Blender é um programa aberto e muito utilizado, existem muitos sólidos, materiais e outros recursos disponíveis para a implementação. Neste caso os objetos são bem mais complexos, como os da Figura 5.7, que representariam cadeiras. Além deste, vários outros foram importados.

Figura 5. 7: Exemplo de objeto elaborado: 'cadeira'.



Fonte: Autor.

Estes objetos são um conjunto de primitivas já instanciadas, porém, por já estarem prontos, podem ser tratados como primitivas simples, e somente serem duplicados e posicionados na cena.

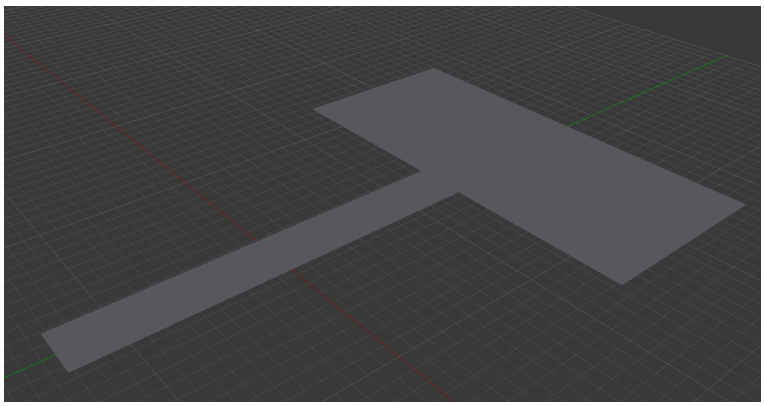
Para o terceiro ambiente, ‘Corredor’, só foram utilizadas as primitivas, ‘*cube*’, ‘*plane*’ e ‘*cylinder*’, pois como já foi explicado o foco deste ambiente foi criar a partir de um local real, sua representação virtual, com a finalidade de poder comparar resultados em ambos.

5.2 Instanciamento

Tendo criado as primitivas, o próximo passo é instancia-las de forma a criar as salas e os objetos. Primeiramente serão abordadas as etapas para criação de salões, e para isto podem ser adotadas diferentes metodologias, pois, algumas características uteis para algumas aplicações não são uteis, ou não são levadas em consideração para outras.

Basicamente deve-se assumir que o objetivo é trabalhar com ambientes *indoor*, ou seja, localizados no interior, além disso, sem inclinação, deformações e profundidades no solo. Sendo assim, pôde ser utilizada a primitiva '*plane*' para a representação do chão, a mesma só precisou ser escalada de forma a representar a área desejada. Este piso, foi colocado 10 cm acima do *grid* para evitar problemas de visualização. A Figura 5.8, demonstra a utilização primitiva para a construção de um corredor que leva a um salão.

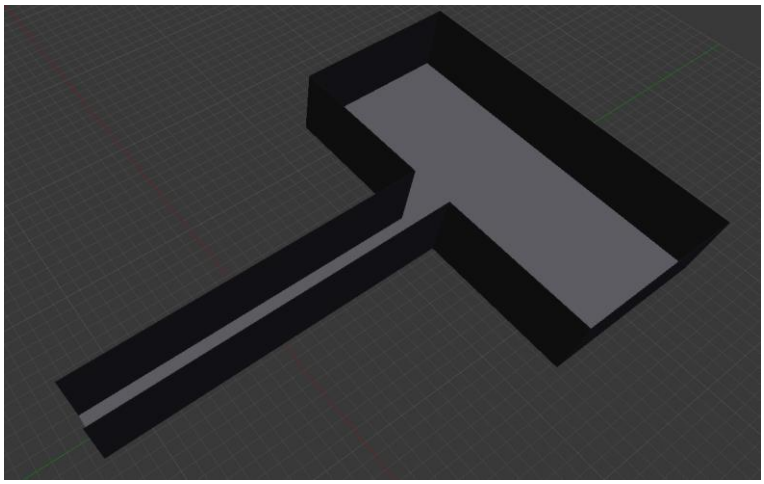
Figura 5. 8: Instanciamento de planos para construção de chão.



Fonte: Autor.

A partir do solo podem ser criadas as paredes, para isto, foi utilizada a primitiva '*cube*', pois desejava-se volume neste caso. Simples operações de escala, rotação e translação, são o suficiente para cria-las. Deve-se ressaltar que, novamente, a escolha da metodologia para a criação dos elementos de cenário depende do objetivo, e varia para cada aplicação. Na Figura 5.9, pode ser visto o ambiente da Figura 5.8 já modificado.

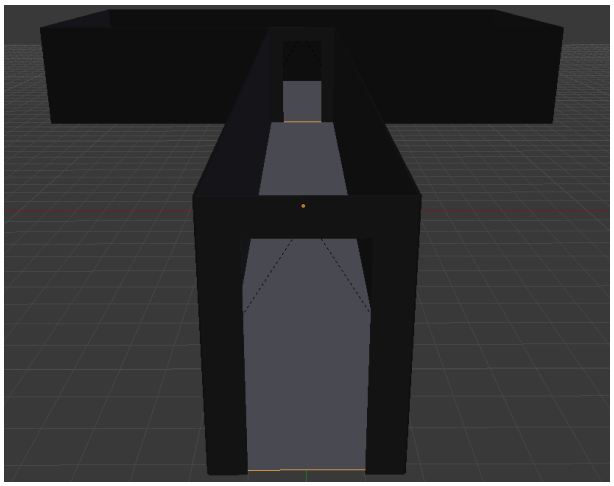
Figura 5. 9: Instanciamento de cubos para construção de paredes.



Fonte: Autor.

Além de estruturas simples, podem ser feitas mais complexas, como por exemplo as passagens para portas. Neste caso utilizou-se 3 cubos para a criação e em seguida eles foram unidos em um só objeto mais complexo. A Figura 5.10, demonstra os elementos de passagem já montados e posicionados.

Figura 5. 10: Instanciamento de cubos para construção de passagens.

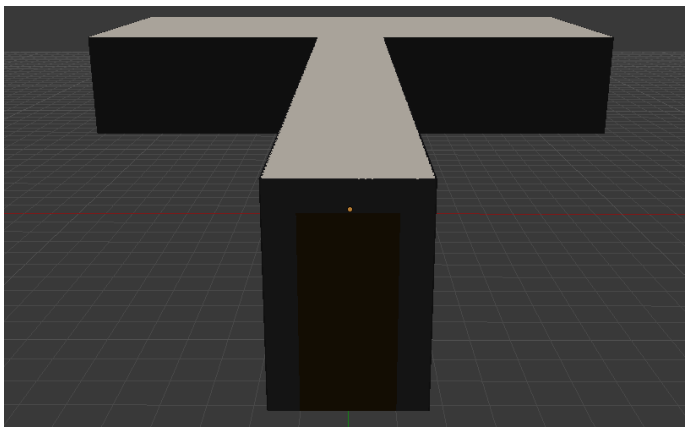


Fonte: Autor.

Como se desejava um ambiente totalmente fechado, foi necessário adicionar ainda portas e teto. Para o teto, do mesmo modo que para o chão, foi utilizada a primitiva ‘*plane*’, e

para a porta a primitiva ‘*cube*’ de modo a preencher os espaços deixado. A Figura 5.11, mostra o ambiente base já pronto.

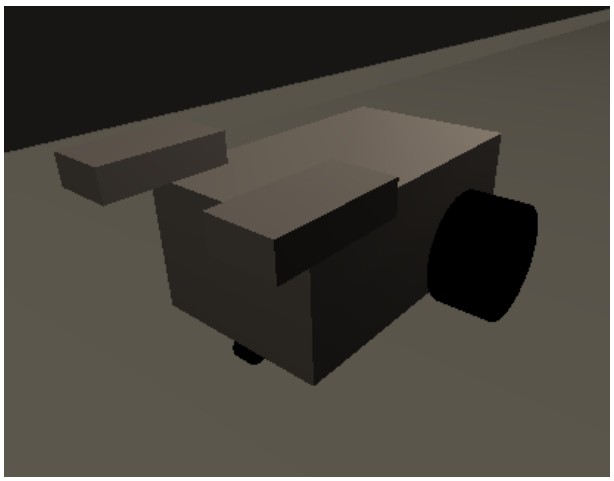
Figura 5. 11: Ambiente 'Corredor' fechado e visto de fora.



Fonte: Autor**.**

Além dos ambientes, também foi necessário criar o robô. Como já foi dito no trabalho, o robô não faz parte da renderização, sendo assim, a sua criação serve apenas para representação visual da movimentação, com o objetivo de observação e análise. Por este motivo, somente foi necessário indexar primitivas para formar um objeto de aparência similar ao robô *Pioneer* P3-DX, que é utilizado como base para os experimentos. Na Figura 5.12, pode ser visto o robô já formado por cubos e cilindros instanciados.

Figura 5. 12: Instanciamento de primitivas para construção do robô.



Fonte: Autor.

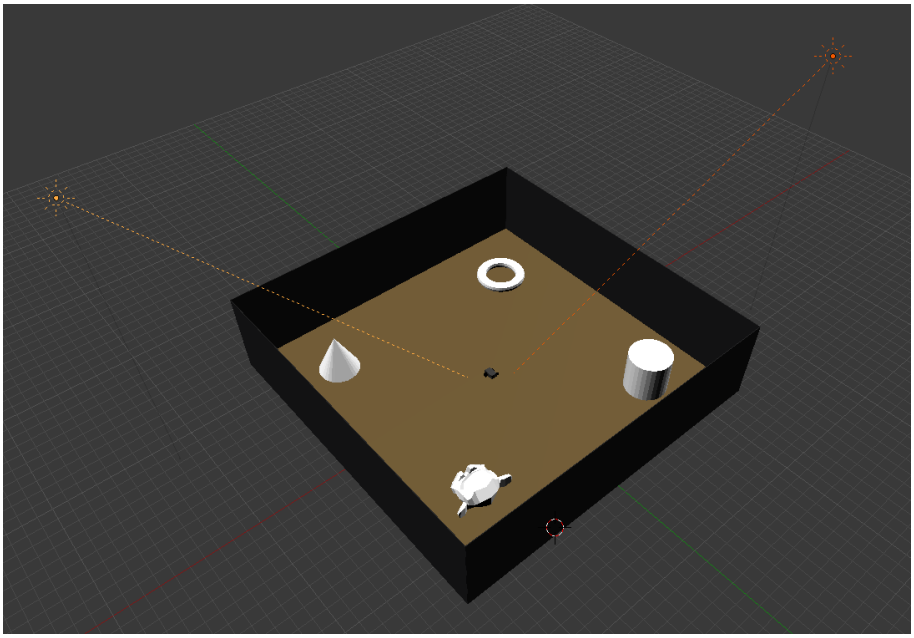
Os dois blocos superiores server para indicar a fixação de câmeras, como já foi mostrado no tópico 4. As características destas câmeras e orientação são variáveis e serão discutidas nos tópicos a seguir.

5.3 Iluminação

Em cada ambiente de simulação foi utilizado um tipo diferente de iluminação, o objetivo foi isolá-los para estudar suas características separadamente. No ambiente ‘Quarto simples’, não há a presença de teto, e apesar de ser um ambiente fechado, desejou-se utilizar a fonte de luz ‘*sun*’, que seria equivalente a uma fonte de luz pontual amplificada.

Na Figura 5.13 pode ser visto que a ausência de teto possibilitou o posicionamento desejado, e que se apresentasse um teto sem transparência, o ambiente ficaria totalmente escurecido. As fontes estão selecionadas em laranja e amarelo, foram utilizadas duas para poder regular melhor a iluminação desejada.

Figura 5. 13: Posicionamento de fontes de luz 'Quarto Simples'.

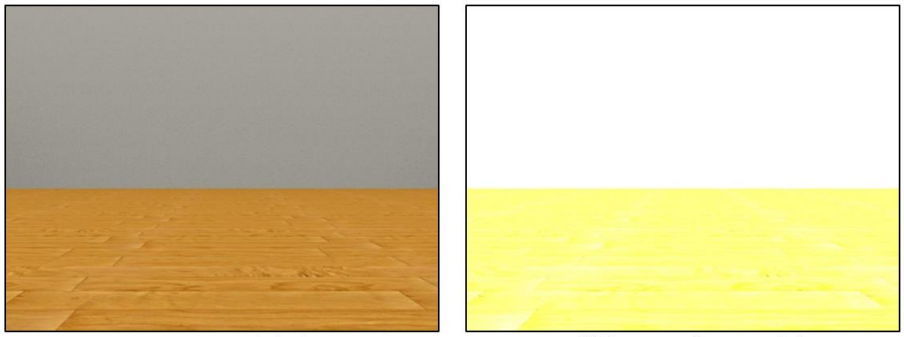


Fonte: Autor.

Ao colocar as fontes dentro do quarto seria causado o efeito indesejado de ‘estourar’ a foto, ou seja, ela ficaria extremamente iluminada, assim perdendo as características mais

importantes da imagem, e inviabilizando a extração de dados. Na Figura 5.14a pode ser vista a imagem com iluminação apropriada e na Figura 5.14b a imagem ‘estourada’.

Figura 5. 14: Comparação entre iluminações 'Quarto Simples'.

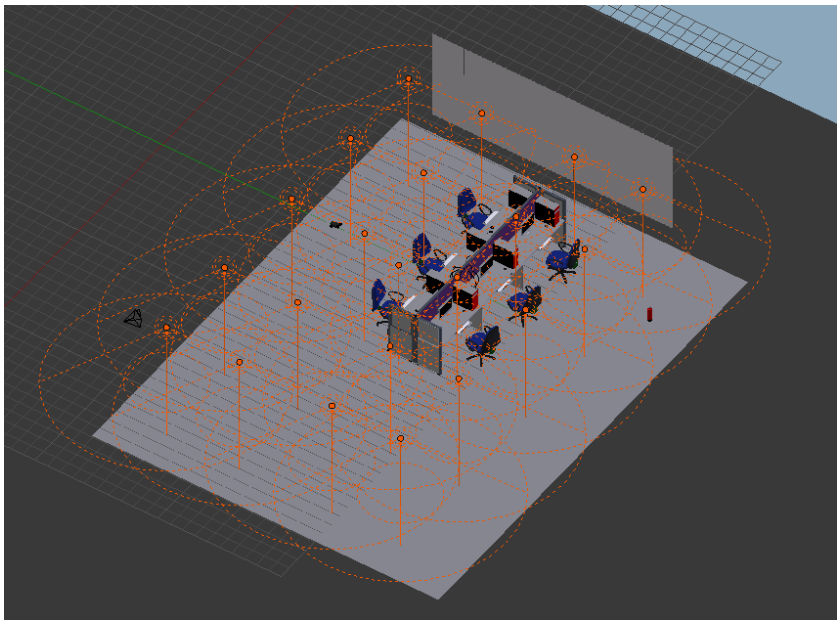


a) Imagem original b) Imagem ‘estourada’

Fonte: Autor.

Os outros dois outros ambientes de simulação utilizaram fontes de luz mais amenas podendo ser colocadas dentro do quarto, e por este motivo puderam apresentar uma cobertura. A fonte utilizada no ambiente ‘Escritório’, foi a ‘*Spot*’ com um ângulo de 108º. Na Figura 5.15 pode ser visto o posicionamento destas fontes de luz.

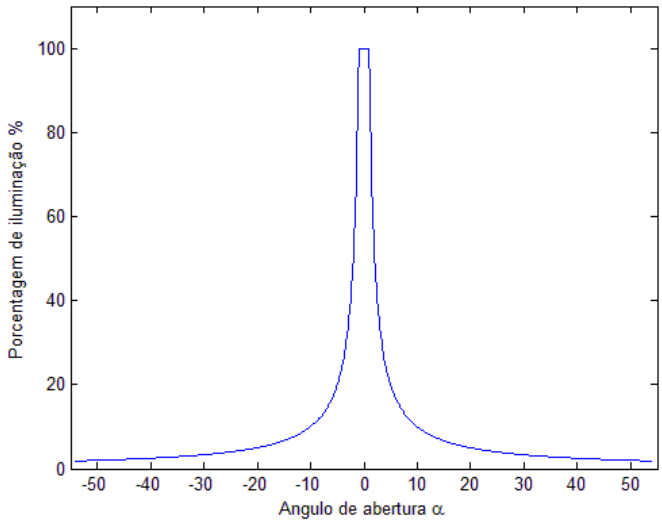
Figura 5. 15: Posicionamento de fontes de luz 'Escritório'.



Fonte: Autor.

Na Figura 5.15 podem ser vistos círculos em torno de cada fonte, estes se devem à abertura de 108º já mencionada, o comportamento desta fonte é descrito na Figura 5-17. Este tipo de fonte luminosa apresenta a maior luminosidade no seu centro e vai sendo reduzida até o ângulo limite. A curva apresentada na Figura 5.16, se refere ao decaimento linear inverso, ou seja, a intensidade vai diminuindo de acordo com o inverso da angulação de abertura. Percebe-se então um comportamento bem diferente da pontual, que seria uma fonte emitindo luz em todas as direções com valor de intensidade constante.

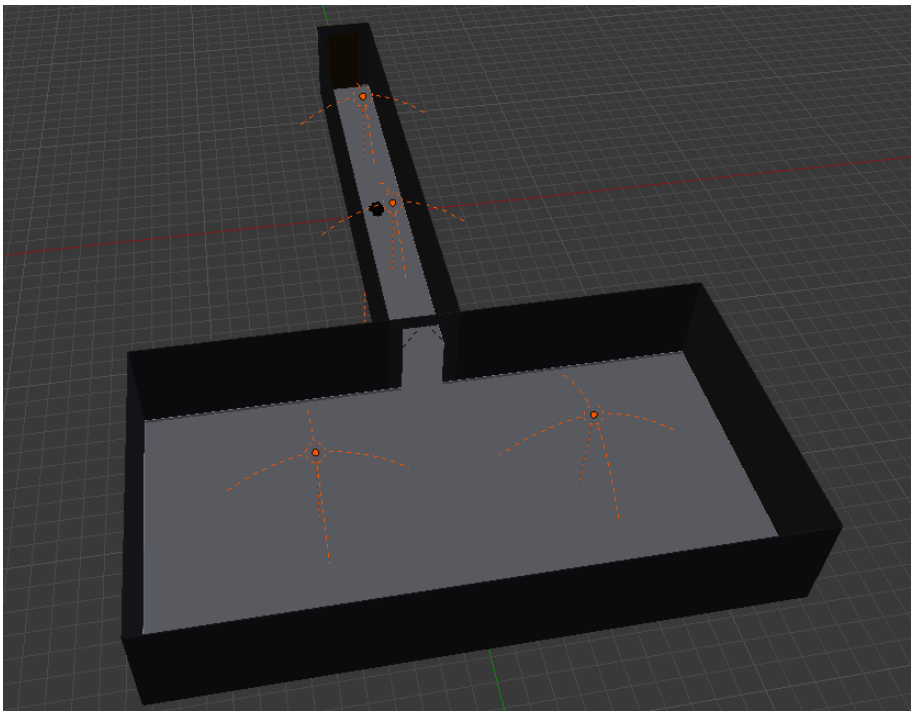
Figura 5. 16: Curva de decaimento de iluminação para fonte '*spot*'.



Fonte: Autor.

A última fonte utilizada foi a ‘*hemi*’ esta fonte não tem um decaimento de iluminação pela angulação, e não emite luz em todas as direções. Ela ilumina um volume de uma meia esfera de acordo com a sua orientação e a distância do ponto iluminado. A iluminação é constante em todas as direções, assim apresentando decaimento somente com a distância. Na Figura 5.17, pode ser vista a disposição das fontes no ambiente ‘Corredor’.

Figura 5. 17: Posicionamento de fontes de luz 'Corredor'.



Fonte: Autor.

Na Figura 5.17, as fontes novamente estão destacadas em vermelho, nelas é possível perceber os traços de propagação da luz em forma de uma meia esfera. Outro aspecto das fontes é o seu valor de instensidade definido, sendo estes valores escolhidos de forma empírica de modo a renderizar as imagens com uma qualidade suficientemente boa para os algoritmos de processamento.

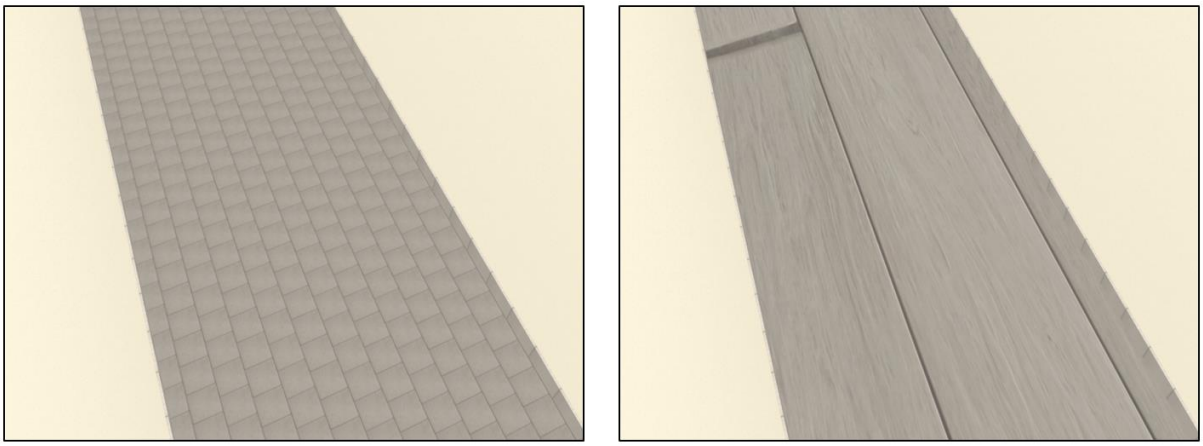
5.4 Texturas

Como só foram utilizadas texturas para o chão, paredes e teto, não foram necessárias muitas imagens e nem técnicas complexas de mapeamento. Para as primitivas '*plane*' foi utilizado o mapeamento '*flat*', o qual ajusta as coordenadas (x, y) da imagem de origem à duas coordenadas especificadas no objeto. Este tipo de mapeamento foi escolhido, pois como já foi dito, estas primitivas não apresentam uma terceira dimensão, assim tendo a imagem distribuída em suas duas dimensões restantes.

Além de ajustar as coordenadas da imagem original com o plano, foi necessário fazer a distribuição para que o resultado não ficasse fora da proporção desejada. Na Figura 5-18, pode

ser vista uma comparação para um dos pisos do ambiente ‘Corredor’ antes de depois deste ajuste.

Figura 5. 18: Comparação entre mapeamentos de textura.



a) Mapeamento ajustado b) Mapeamento direto

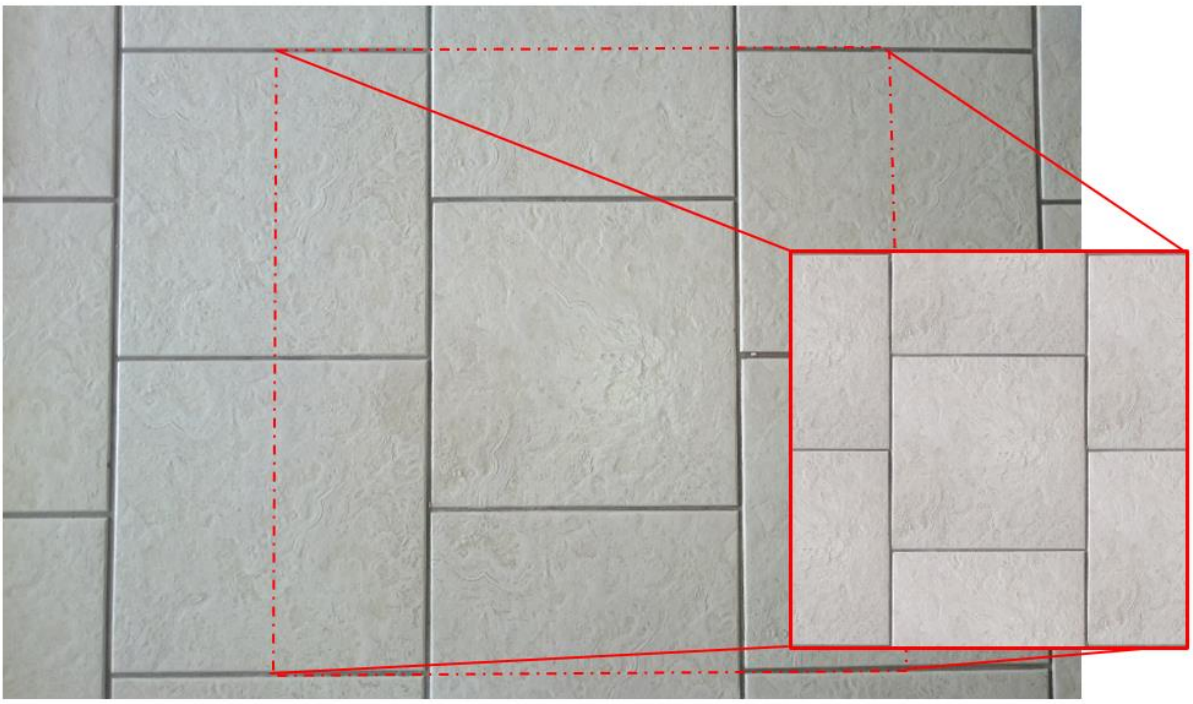
Fonte: Autor.

Pode ser percebido claramente na Figura 5-18a que o caso é de um piso com lajotas, porém, se o mapeamento for feito de forma direta, ou seja, simplesmente mapeando as coordenadas de borda, o resultado poderá não sair de acordo com o esperado, assim como a Figura 5-18b, onde a imagem de textura foi esticada e tirada de proporção.

Para as primitivas ‘*cube*’, o mapeamento utilizado apresenta o mesmo nome, e se ajusta à todas as faces do cubo de uma vez só. Um problema que poderia ocorrer neste caso é que a proporção da primitiva não se manteve igual para todos os lados, assim a textura provavelmente sairia de um modo indesejado em alguns deles. Este caso não foi um problema no trabalho pois nas paredes temos 2 faces do cubo que realmente importam e 4 dispensáveis, além disso, as faces não importantes são bem menores impossibilitando de qualquer forma diferenciar a textura partindo de uma distância razoável.

As imagens utilizadas foram obtidas a partir de bancos de textura disponíveis na internet, e para o caso do ambiente ‘Corredor’, como desejava-se replicar um lugar existente, as texturas foram feitas a partir de fotografias tradas e adaptadas para tal. A Figura 5.19 ilustra o processo de extração da textura na imagem.

Figura 5. 19: Extração de textura para piso.



Fonte: Autor.

O processo para a criação da textura envolve mais do que somente obter a imagem desejada, assim como pode ser visto na Figura 5.19, foi necessário identificar um padrão que se repetiria caso posicionado lado-a-lado e que representasse a mesma distribuição do piso. Outro problema foi corrigir casos de inclinação em relação aos três eixos espaciais, produzindo diferenças entre distâncias equivalentes, e também foi essencial que se igualasse a iluminação por toda a textura para evitar artefatos nas imagens renderizadas no ambiente de simulação.

5.5 Composição de ambientes

A partir dos passos adotados nos tópicos anteriores foi possível construir os ambientes desejados para as aplicações do trabalho, porém, esta foi somente a primeira parte. Para poder salvar corretamente cada um dos sólidos desejados foi utilizado o formato *collada* (*.dae), este formato mantém os objetos como individuais, as suas respectivas coordenadas já transformadas e seus nomes personalizados.

Além de mantê-los separados, é possível carregar todo o ambiente de uma vez só, o que facilita no desenvolvimento do script. Como já foi mencionado, cada ambiente teve sua função principal, o ‘Quarto simples’ foi utilizado para desenvolver aspectos principais da metodologia para movimentação e aquisição de imagens, o ‘Corredor’ serve para a comparação entre um ambiente real e o gerado a partir de suas texturas, e por último, o ‘Escritório’ utiliza funções mais caras em termos de processamento, para gerar renderizações melhores, além de conter outros objetos de cena mais complexos do que nos dois primeiros.

As chamadas dos ambientes e também todas as outras funcionalidades do simulador foram programadas utilizando a linguagem ‘*python*’, e as principais funções utilizadas foram as de chamada de ambientes, atribuição de material, movimentação do robô, renderização e captura de imagens, e a leitura e escrita em arquivos (*.txt).

Devido ao formato escolhido, a chamada dos sólidos foi feita de maneira bem simples, pois foi necessário somente especificar o diretório e carrega-los, porém, infelizmente este formato não guarda propriedades do material do objeto, e para isto é necessário atribuir todos eles a cada chamada. Alguns dos parâmetros principais para criação de materiais estão relacionados abaixo:

- **Cor difusa:** Será a cor principal do objeto, e apresenta diferenças de acordo com o posicionamento da normal em relação à luz do ambiente, deve-se selecionar o algoritmo desejado para o cálculo e os coeficientes da cor;
- **Cor especular:** Cor relacionada à reflexão da luz, dá a ideia de polimento, ou até mesmo de liquido, deve-se selecionar algoritmo desejado para o cálculo e os coeficientes da cor;
- **Transparência:** Deve-se selecionar o algoritmo desejado para a simulação de transparência e também o quão transparente;
- **Reflexão:** Novamente deve-se escolher o algoritmo desejado para a reflexão e também a intensidade e profundidade;
- **Sombra:** Definir se o objeto produz e recebe sombra;
- **Textura:** A textura fica conectada ao material, e é opcional como no caso dos outros parâmetros.

Para o caso do ambiente ‘Quarto simples’, os materiais apresentavam somente cor difusa e sombra, para o cálculo da cor difusa foi utilizado o algoritmo de Lambert disponível como

opção na plataforma. O segundo ambiente, ‘Corredor’, além das propriedades anteriores, contém também cor especular pelo algoritmo de Phong.

Já o ambiente ‘Escritório’ apresenta todas as propriedades descritas acima, sendo transparência e reflexão calculadas por *ray-tracing*, e todos os objetos emitindo e recebendo sombra. Além destas propriedades, cada material teve a sua textura definida e aplicada como explicado no sub-tópico anterior.

A movimentação do robô foi feita a partir de translações e rotações de acordo com o passo e coordenadas definidas pelo modelo dinâmico utilizado, cada um destes movimentos é passado para o simulador através de um arquivo (*.txt), que é lido e escrito simultaneamente por ambos os programas, Blender e Matlab, o processo em detalhes para a sua comunicação será melhor explicado no sub-tópico seguinte.

Para a captura das imagens foi necessário definir os parâmetros intrínsecos da câmera desejada, algumas propriedades de renderização. Para as câmeras, os parâmetros foram os pré-definidos ‘*default*’, que são CCD 35 x 28.19 mm e distância focal 32 mm, também foram utilizados parâmetros para a câmera do celular xperia z3, que são CCD 6.17 x 4.55 mm (1/2.3”) e distância focal 25 mm.

Para a renderização, o objetivo é chegar em uma relação qualidade *vs* tempo de processamento, que seja suficientemente agradável para o proposito desejado, sendo assim, algumas propriedades podem ser utilizadas para melhorar a qualidade das imagens renderizadas:

- ***Anti-aliasing*:** Evita efeitos de serrilhamento nas imagens geradas, aumenta bastante a qualidade, porém, também o tempo de renderização;
- ***Ray-tracing*:** Algoritmo utilizado para uma representação fiel do comportamento da luz, acompanhando todas as iterações de cada raio com cada ponto em cada objeto, sendo assim, é extremamente ‘pesado’ para a geração de imagens;
- **Resolução:** Tamanho de saída em pixels da imagem renderizada, largura x altura, quanto maior, melhor a qualidade da imagem e também seu tamanho e tempo de geração e armazenamento;
- **Formato:** Definição do formato para armazenamento das imagens, alguns formatos apresentam processos de compressão, assim diminuindo o espaço de armazenamento, porém aumentando o tempo de geração e o processamento.

Para cada ambiente e teste executado estes parâmetros foram alterados, desta forma pôde-se avaliar a qualidade das imagens geradas.

5.6 Comunicação com *software* externo

Como parte da ideia do trabalho é a movimentação do robô a partir do seu modelo dinâmico, foi necessário comunicar o ambiente de simulação com o aplicativo que geraria a trajetória do robô, neste caso o Matlab. Para esta comunicação foi escolhida a criação de arquivos de texto, pois além de comunicar, seria possível ter um registro para consulta posterior ao experimento.

Foram feitos dois arquivos, ‘Matlab.txt’ e ‘Blender.txt’, o primeiro arquivo apresenta as informações enviadas do Matlab para o Blender, e o segundo o contrário. Além de enviar as informações, o arquivo Matlab.txt também controla o processo de leitura e escrita e é responsável por encerrar o processo. No primeiro arquivo, as informações se dispõem de acordo com a organização da Figura 5.20:

Figura 5. 20: Organização de dados no arquivo 'Matlab.txt'.

$$C\ X\ I_x X_1 . X_2 X_3 X_4 X_5\ Y\ I_Y Y_1 . Y_2 Y_3 Y_4 Y_5\ R\ I_R R_1 . R_2 R_3 R_4 R_5\ F$$

Fonte: Autor.

Sendo o valor ‘C’ em verde é o que determina quem aguarda a resposta e quem escreve no arquivo, pois, os dois devem alternar até a finalização de todos os movimentos. Se for ‘0’ o Blender escreve, e se for ‘1’, o Matlab. Os valores em preto, são somente para facilitar uma assimilação visual da informação, ou seja, eles são ignorados juntamente com os espaços pelos *scripts* de leitura.

Os valores em roxo são os indicadores, eles determinam se o valor é positivo ou negativo, ou seja, para ‘I’ igual a ‘1’, o valor é positivo, e ‘0’, é negativo. Os termos em azul, são referentes a variações na coordenada (δx, δy, $\delta\psi$), sendo o primeiro termo a parte inteira, e os outros quatro a parte decimal. Claramente pode ser visto que o valor máximo para uma

movimentação é de ‘9.9999’, porém, isso pode ser controlado com a velocidade linear do robô, e a rotação tem um valor máximo de 2π, que é menor do que o limite.

O último valor em vermelho, ‘F’, é o que determina quando o processo foi encerrado, sendo assim, enquanto seu valor for igual a ‘0’, os dois ambientes continuam a movimentação. Este termo recebe ‘1’, quando a última coordenada gerada pelo modelo dinâmico for passada para o simulador.

Como já foi dito, o valor ‘C’ determina quem monitora e quem escreve, porém, o Blender passa a vez para o Matlab, quando já enviou ao seu arquivo ‘Blender.txt’, a sua posição atual, que se não receber nenhum ruído, deve ser a mesma enviada pelo modelo dinâmico. O formato de apresentação destas informações é semelhante ao outro arquivo de texto, na Figura 5.21 segue o seu esquema de organização:

Figura 5. 21: Organização de dados no arquivo 'Blender.txt'.

$X\ I_x X_0 X_1 . X_2 X_3 X_4 X_5 X_6 X_7 X_8\ Y\ I_Y Y_0 Y_1 . Y_2 Y_3 Y_4 Y_5 Y_6 Y_7 Y_8\ R\ I_R R_0 R_1 . R_2 R_3 R_4 R_5 R_6 R_7 R_8$

Fonte: Autor.

Neste caso, os valores dos indicadores são apresentados com o sinal, sendo assim a ausência de ‘-’, significa um valor positivo. Além disso, os valores enviados são os de posição e rotação, sou seja (x, y, ψ), por este motivo devem ser maiores suportando até ‘99.99999999’ metros.

Para cada passo enviado, as imagens de ambas as câmeras é renderizada e armazenada de acordo com o número do passo. Por exemplo no passo 52, o a imagem armazenada pela primeira câmera tem a identificação ‘Camera01 – 52.jpg’, e a segunda câmera, ‘Camera02 – 52.jpg’. Esta forma de armazenamento é útil, pois mais adiante deverá ser gerado um vídeo com as imagens para um algoritmo externo de estimação de trajetória, utilizando a biblioteca OpenCV, na linguagem C++.

5.7 Integração com modelo dinâmico

Para enviar as coordenadas em cada passo, seguindo a movimentação de um robô diferencial, é necessário que estes sejam determinados pelo modelo dinâmico do robô. Desta forma, neste sub-tópico estarão os detalhes para a aquisição destes passos e organização para envio ao simulador. Visto que o modelo foi desenvolvido em outro trabalho, detalhes para este desenvolvimento estarão presentes nas referências da dissertação.

Primeiramente, é necessário definir o ponto inicial e final da trajetória, isto pode ser feito de dois modos, se somente for necessário sair da origem e chegar a um ponto especifico, será determinado este ponto, e então é só rodar o modelo em ‘simulink’, guardando os valores em cada passo da orientação e posição. Porém, se o robô tiver uma trajetória mais complexa, será necessário rodar várias vezes o modelo até completar toda a trajetória.

Ao final da primeira etapa, já teremos todos os passos, contento a posição e orientação do robô. Deste modo, o passo seguinte é calcular a variação destes valores, e isto pode ser feito simplesmente subtraindo-se o valor de uma posição pela anterior. O resultado desta operação será armazenado no vetor para ser enviado ao Blender em cada passo pelo arquivo ‘Matlab.txt’, assim como descrito no sub-tópico anterior.

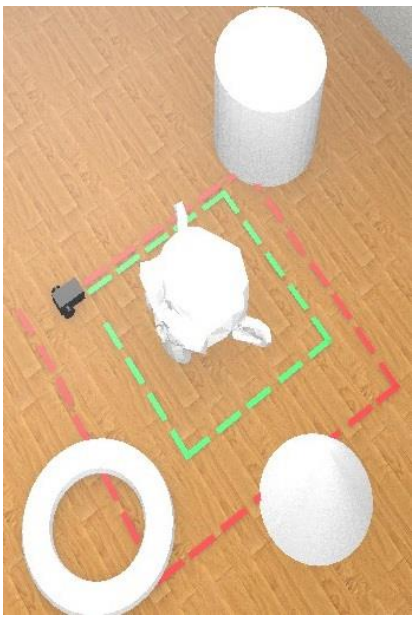
Após enviar uma coordenada, o Matlab aguarda até que o valor da posição e orientação do robô no Blender seja enviado pelo arquivo ‘Blender.txt’, e ele receba o comando de partir para o próximo passo. Ao receber estes valores, eles são armazenados em uma matriz (n x 3), sendo ‘n’, o número de passo total do caminho. Estes valores servirão para comparações futuras, e até mesmo se for desejado adicionar algum ruído de qualquer tipo no Blender para ser controlado no Matlab.

Ao final teremos duas matrizes, uma com a posição gerada melo modelo dinâmico no simulink/Matlab, e a outra com os valores retornados pelo Blender através do arquivo ‘Blender.txt’. Além disso, como já foi dito, todas as imagens renderizadas ao longo do caminho estarão salvas no diretório especificado.

5.8 Validação

A validação do ambiente será feita em três etapas. Primeiramente será verificado qual o erro de posição, comparando os valores retornados pelo arquivo ‘Blender.txt’ com os gerados pelo modelo dinâmico do robô, tanto para uma trajetória simples, quanto para uma mais complexa. Ao analisar este valor, poderemos verificar qual o erro máximo, para evitar casos como o da Figura 5.22, onde um grande erro de trajetória poderia implicar no robô entrando em algum objeto. Claro que neste caso o erro deveria ser muito grande, porém a imagem serve para ilustrar o possível problema.

Figura 5. 22: Exemplificação de problema em trajetórias com erro elevado.



Fonte: Autor.

O valor desejado de erro é variável, pois depende da aplicação. Por exemplo, se for campo aberto, mesmo um grande erro, não causará muitos problemas na geração de imagens, já se for um labirinto, ou corredor estreito, estas imagens podem se tornar inviáveis pelo fato do robô entrar em alguma das paredes.

O erro da trajetória (*D)* será calculado pela diferença das distâncias ente o ponto atual e o (0,0), ou seja, em cada passo será o valor será como o descrito pela Equação (5.1), onde (x_{Matlab}, y_{Matlab}) são os valores gerados pelo modelo dinâmico, e $(x_{Blender}, y_{Blender})$ são os

valores retornados através do arquivo ‘Blender.txt’. Já para a rotação, poderá ser utilizada a diferença direta entre os dois vetores.

$$D = \sqrt{(x_{Matlab}^2 + y_{Matlab}^2)} - \sqrt{(x_{Blender}^2 + y_{Blender}^2)} \quad (5.1)$$

A segunda análise será sobre a capacidade de utilização das imagens renderizadas, este experimento será feito utilizando o trabalho de um colega de laboratório, Marcus Vinicius, que fará o processo de estimação de trajetória utilizando odomêtria visual e processamento de imagens para a aquisição dos pontos. Está etapa verificará se esta estimação pôde ser feita utilizando o simulador e os seus resultados.

A última etapa será uma análise com relação à qualidade das imagens renderizadas *vs* tempo de renderização, isto será feito comparando imagens com diferentes níveis de *aliasing*, e *ray-tracing*. O método para comparação será o erro médio absoluto (*MAE*) entre duas imagens conforme é variada a qualidade, a formula para este erro está representada na Equação (5.2).

$$MAE = \frac{1}{m.n}.\sum_{i=1}^{m}\sum_{j=1}^{n}\left|\hat{I}_{(i,j)} - \hat{I}_{(i,j)}\right| \quad (5.2)$$

Os valores de ‘m’ e ‘n’ são os tamanhos das imagens, $\hat{I}$ e I são as duas imagens comparadas. Fica claro que este método só consegue comparar duas imagens de tamanhos iguais. Por último, será feita uma análise qualitativa entre o ambiente ‘Corredor’ e sua versão real, comparando imagens dos dois vídeos.

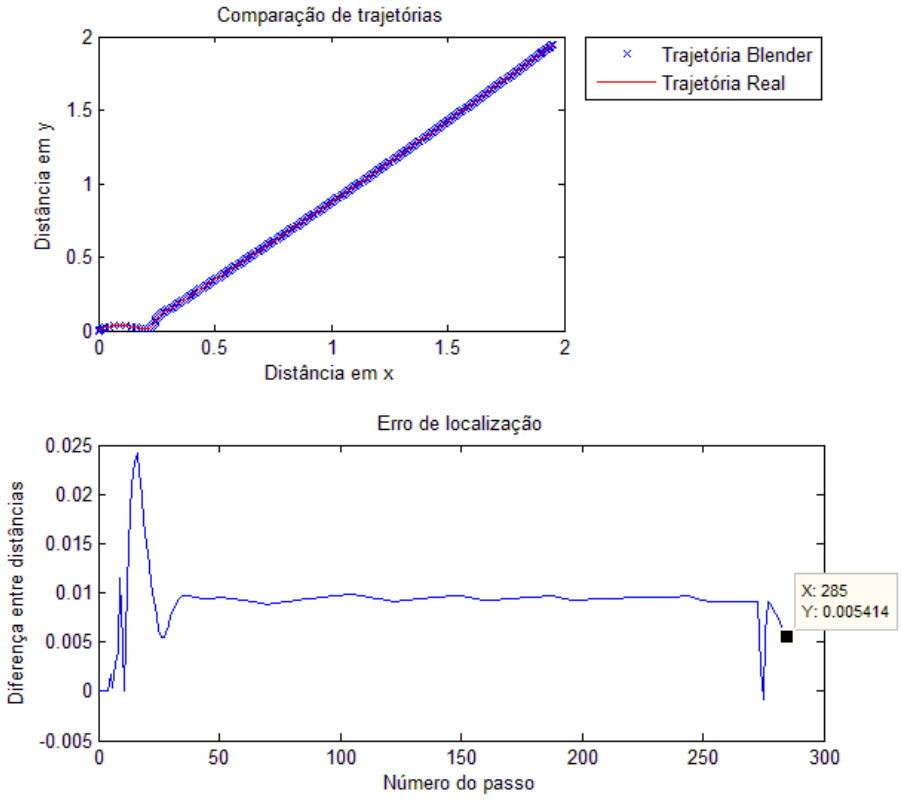
6 RESULTADOS E DISCUSSÕES

Neste tópico serão apresentados os resultados e análises verificadas ao se realizar a validação do simulador. Será dividido em três partes; definição de caminhos desejados; armazenamento e utilização das imagens obtidas e análise de qualidade das imagens. Cada etapa será de acordo com o que foi descrito no sub-tópico (5.8) Validação.

6.1 Definição de caminhos desejados

Primeiro será analisado o resultado para uma trajetória simples, sendo do ponto (0,0) até o ponto (2,2). Na Figura 6-1 é apresentada a trajetória gerada pelo modelo dinâmico e a retornada pelo simulador, além de um gráfico relacionando o erro em cada passo.

Figura 6. 1: Comparação de trajetórias (um ponto).



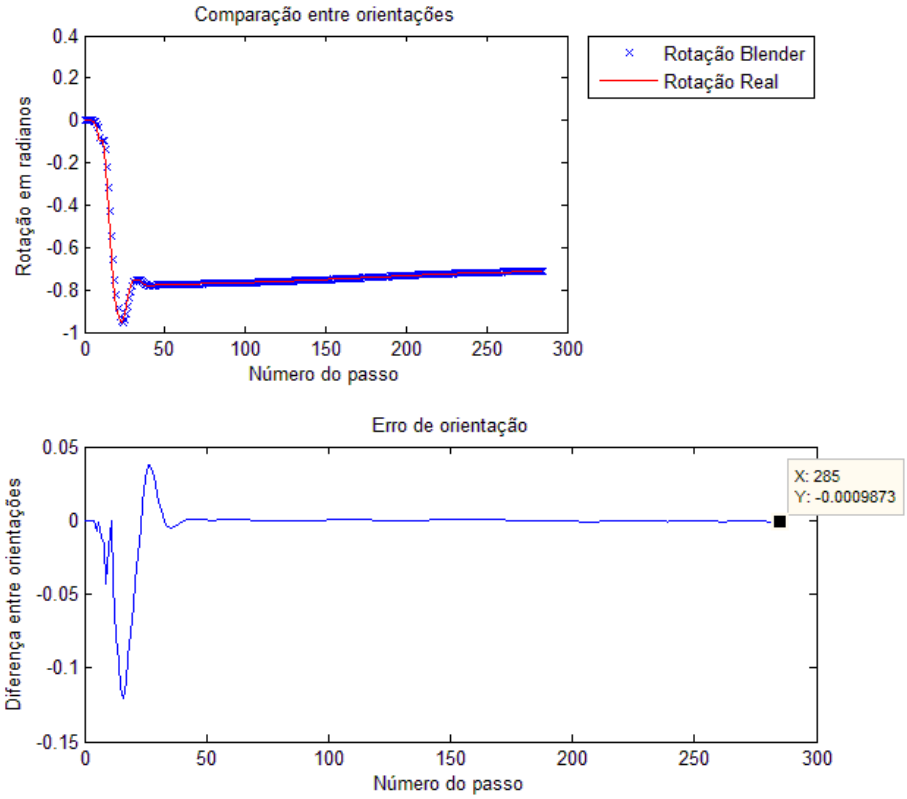
Fonte: Autor.

Neste trajeto foram necessários somente 285 passos para terminar, aparentemente pelo gráfico de comparação a trajetória enviada ao simulador e a retornada parecem idênticas, porém, ao verificar o gráfico de erro, temos um valor máximo próximo a ‘0.025’metros.

Nos pontos de início e final da trajetória é onde a velocidade linear tem seus menores valores, e pode ser percebido pela Figura 6.1, que apresentam, também, maiores valores de erro. Outro ponto importante, é que apesar de ter um valor máximo de ‘0.025’ metros, o valor do erro para a posição final foi de somente ‘0.005414’metros.

Além da localização, também deve ser analisada a orientação do robô (Figura 6.2), e neste caso, novamente, aparenta estar de acordo com os valores definidos. Porém, ao aumentar a velocidade angular, ou seja, o espaçamento entre cada valor aumenta, temos também um pico no erro, sendo seu maior valor absoluto, aproximadamente, ‘0.12’ radianos.

Figura 6. 2: Comparação de orientações (um ponto).

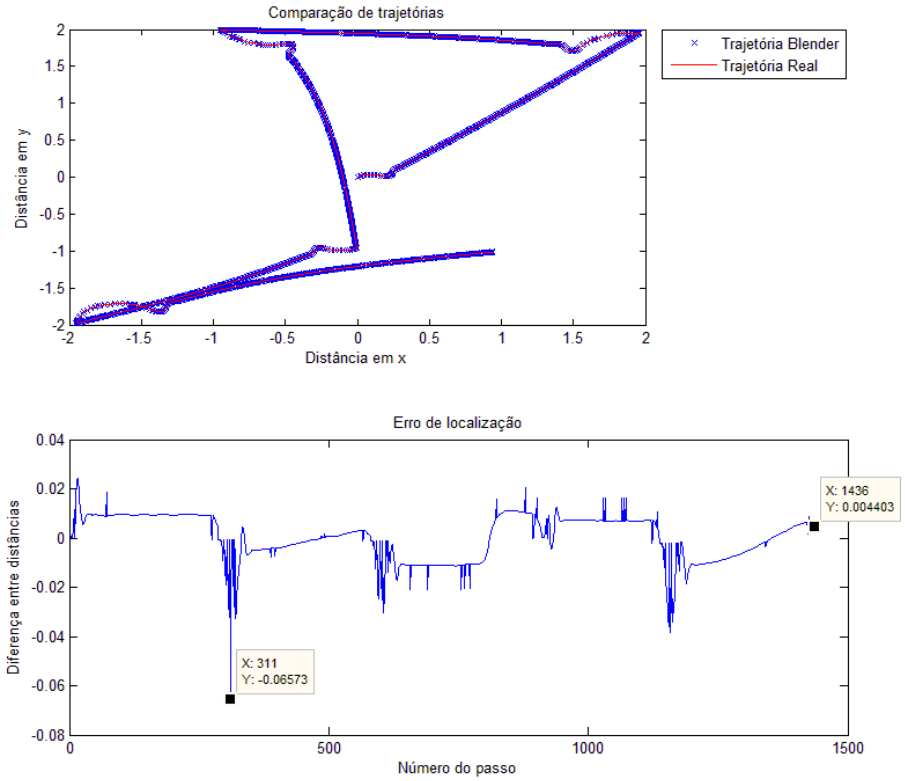


Fonte: Autor.

Outro ponto que pode ser analisado pela Figura 6.2, é que o valor final do erro é bem inferior ao máximo, assim como no caso da localização, sendo seu valor absoluto igual a ‘0.0009873’ radianos.

Com relação a trajetos definidos, o anterior foi bem simples, porém, a mesma análise será feita para um caminho mais complexo. Este caminho parte da posição (0,0), e então vai para os pontos, (2,2), (-1,2), (0,-1), (-2,-2) e (1,-1), nesta ordem. A Figura 6.3, apresenta os resultados para comparação dos valores de trajetória do modelo dinâmico e retornadas pelo simulador.

Figura 6. 3: Comparação de trajetórias (cinco pontos).



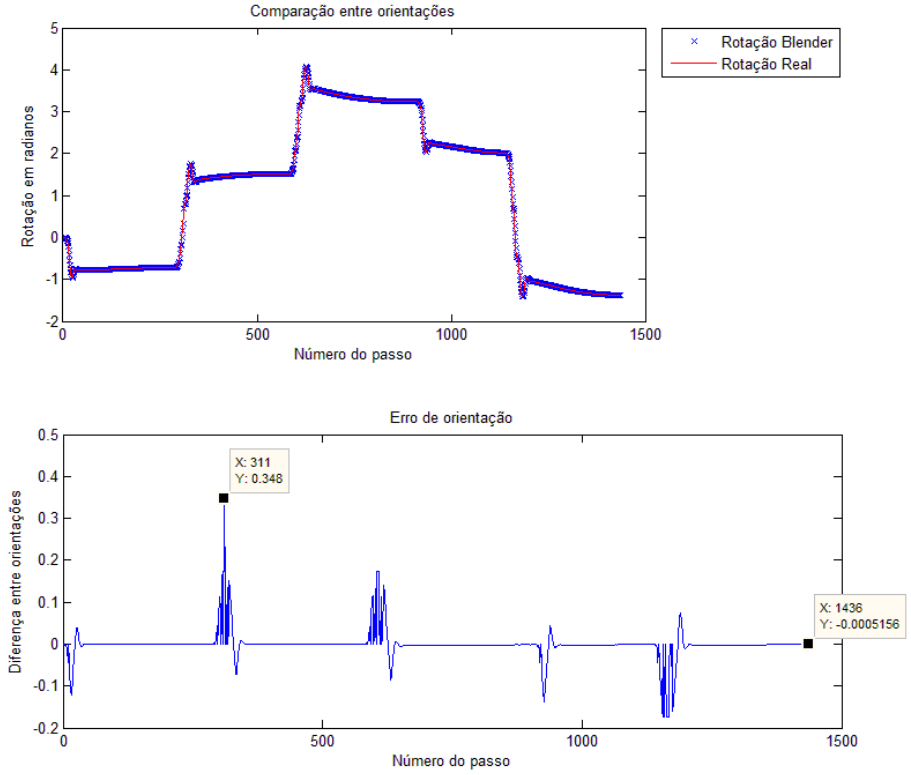
Fonte: Autor.

Pela Figura 6.3, novamente, percebe-se que a trajetória ficou bem similar ao desejado, porém, em uma análise mais profunda, verifica-se que o valor máximo de erro foi maior, do que o caso do caminho mais simples, tendo seu valor absoluto igual a ‘0.06573’ metros, ou seja, 6.573 cm, no passo igual a 311. Por ser mais complexo, este trajeto também apresentou mais passos, sendo necessários 1436 para completá-lo.

Um aspecto positivo, é que o valor do erro final se manteve bem menor ao máximo, sendo igual a ‘0.00403’ metros, o equivalente a 4.03 mm, e neste caso, ainda inferior ao erro final do caso mais simples, por este motivo, pode-se afirmar que o erro não escala de acordo com o número de pontos da trajetória.

Visando completar a análise deve-se olhar a Figura 6.4, onde está apresentada a comparação de orientações do robô para o mesmo caminho. Por esta figura pode-se perceber que a orientação enviada e recebida do simulador foi semelhante em todos os passos.

Figura 6. 4: Comparação de orientações (cinco pontos).



Fonte: Autor.

Na Figura 6.4, ainda pode ser visto que o passo onde o erro máximo de orientação ocorreu foi o mesmo do de trajetória, o 311, e o valor máximo foi ‘0.348’ radianos, além disso, novamente os picos de erro acontecem nos momentos de velocidade angular elevada, porém, apesar de um valor relativamente alto de erro, no ponto final foi bem inferior, sendo somente ‘0.0005156’ radianos.

6.2 Armazenamento e utilização das imagens obtidas

Este sub-tópico exibirá resultados relacionados a estimação de trajetória utilizando as imagens renderizadas pelo simulador. Como já foi explicado anteriormente, a orientação da câmera as propriedades de geração de imagens e ambiente, podem variar de acordo com a aplicação. A trabalho permite a criação de imagens com características complexas como reflexão e transparência, além da aplicação de filtros e utilização de técnicas para ampliar a qualidade da imagem, na Figura 6.5 pode ser visto um exemplo de imagem renderizada utilizando os recursos já mencionados.

Figura 6. 5: Exemplo de imagem de boa qualidade.

Fonte: Autor.

Apesar desta capacidade, o objetivo principal desta análise será a possibilidade de realizar a estimação de trajetória com as imagens do simulador. Neste caso a câmera está orientada para baixo, ou seja, as fotografias serão do piso, além disso os parâmetros intrínsecos determinados foram; distancia focal igual a 32 mm, CCD 35 x 28.18 mm, além disso as imagens tiveram resolução 720 x 580 e o formato foi (*.jpeg).

O ambiente utilizado foi o ‘Quarto simples’, contendo todas as características já apresentadas. Na Figura 6.6, pode ser visto um exemplo das imagens obtidas por ambas as câmeras neste ambiente.

Figura 6. 6: Imagem renderizada com pontos de referência.

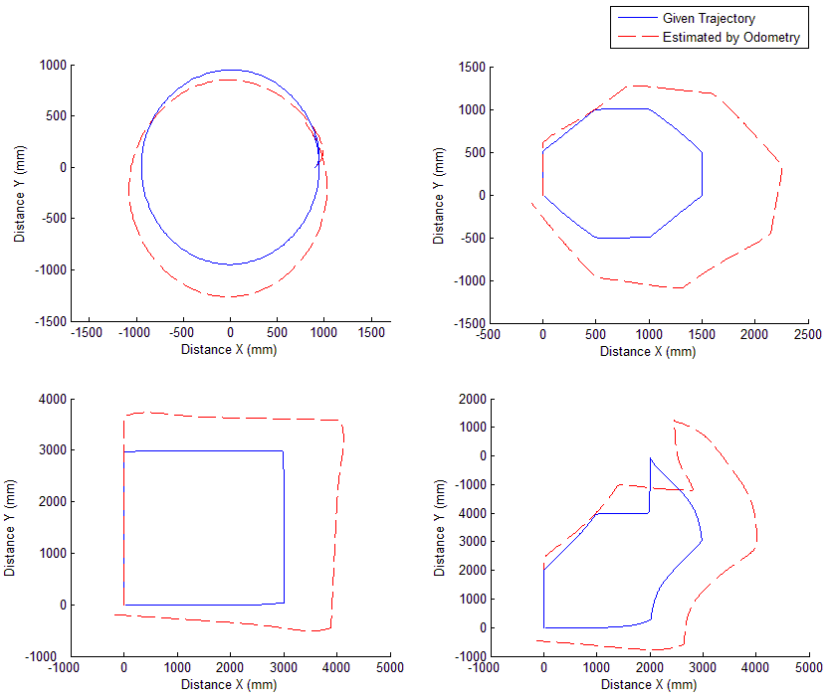
Fonte: Autor.

Na Figura 6-6 também podem ser vistos pontos vermelhos, estes pontos servem de base para a execução dos algoritmos de odomêtria, e graças à presença desta textura, pode ser criado um número bem maior de pontos comparando com uma representação contendo somente sólidos, ou com texturas simples, assim aumentando a precisão dos *scripts* de estimação.

Esta estimação foi feita no trabalho de (LIMA, 2015), de duas formas diferentes, a primeira utilizando somente odomêtria visual diretamente com os pontos correlacionados, já na segunda etapa, foi aplicada uma regularização deste resultado utilizando redes-neurais. Na Figura 6.7, os resultados com aplicação somente da odométrica podem ser visualizados, em azul temos a trajetória especificada, e em vermelho tracejado, o caminho estimado pelo algoritmo.

Pela análise desta imagem pode-se perceber uma semelhança entre os formatos dos caminhos estimado e fornecido, sendo assim é um resultado que ainda pode ser trabalhado de diversas formas, e mesmo neste estado já fornece informações importantes sobre a rota traçada. Outro aspecto é que o comportamento permaneceu semelhante para as três formas geométricas, circulo, hexágono e quadrado, como também, para um formato mais complexo.

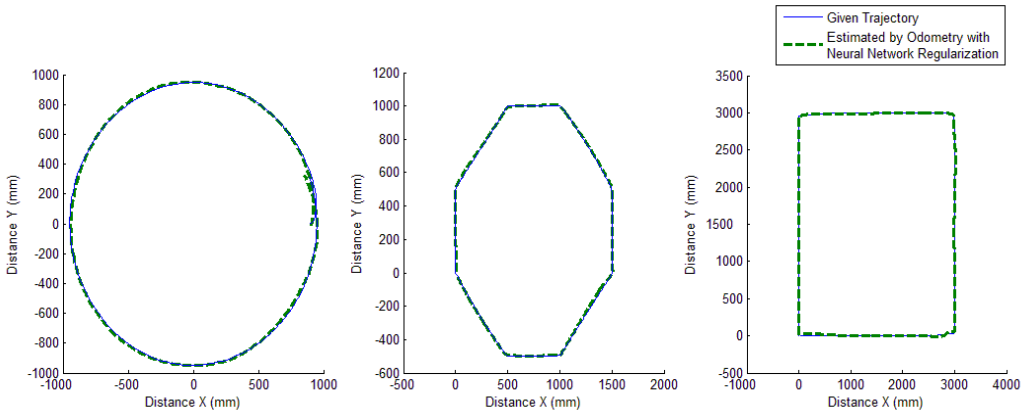
Figura 6. 7: Estimação de trajetória utilizando odomêtria visual.



Fonte: LIMA, (2015).

Os resultados da Figura 6.7 puderam ser melhorados aplicando uma regularização utilizando redes neurais. Sendo assim, na Figura 6.8 são mostrados os novos resultados, agora com um formato bem mais similar ao da trajetória, assim validando a utilização das imagens do simulador para a sua estimação.

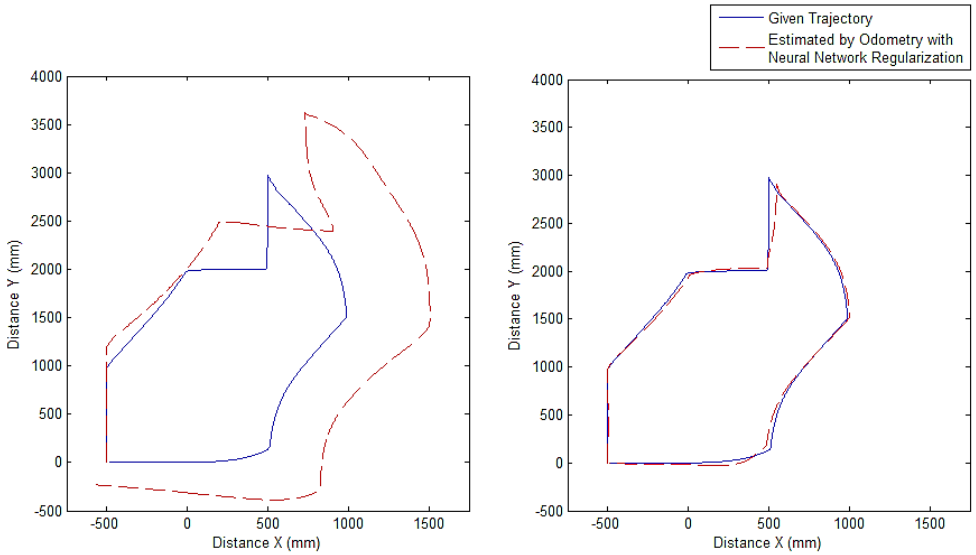
Figura 6. 8: Estimação de trajetória utilizando odomêtria visual com rede-neural.



Fonte: LIMA, (2015).

A rede pode ser treinada com as formas geométricas simples, e já apresentou um bom resultado para o caminho mais complexo. Na Figura 6.9, pode ser visto a comparação entre a odomêtria pura e a com rede-neural já treinada.

Figura 6. 9: Comparação de resultados para rede-neural treinada.



Fonte: LIMA, (2015).

Fica clara a melhora do resultado ao se observar a Figura 6.9, e a estimação pode ser executada novamente. Mais detalhes sobre o algoritmo e as técnicas utilizadas na estimação destas trajetórias podem ser vistas no trabalho aqui referenciado.

6.3 Análise de qualidade das imagens

Para analisar a qualidade das imagens, primeiramente será verificada a diferença entre os tempos de renderização para diferentes níveis de filtro anti-aliasing, o algoritmo do filtro é o Mitchell-Netravali, já disponibilizado na plataforma. Outro aspecto importante é que para a aquisição do tempo de processamento, deve ser levada em conta a configuração da máquina utilizada.

Neste experimento foi utilizado um *Notebook* da marca ASUS, modelo S46CM-WX119H, cujo processador é um Intel Core I7 – 3517U – 1.9 Ghz, com 8 Gb de memória RAM, e 2 GB de memória dedicada para vídeo pela sua GPU GEFORCE GT-635M.

Na Tabela 6.1 podem ser vistos os tempos de renderização para o ambiente ‘Escritório’ sem utilizar o *ray-tracing*, nela estão sendo comparadas imagens com filtro *anti-aliasing* de diversos níveis, e o número da comparação que servirá para a análise da Figura 6.10. Os tempos estão no formato (ss:ff), sendo ‘s’ o equivalente a segundos e ‘f’ para centésimo de segundo.

Tabela 6.1: Tempos de renderização sem *ray-tracing.*

Número da Comparação	Elementos comparados	Tempos de renderização
1	*aliasing* 0 com 5	(01:09) e (02:58)
2	*aliasing* 5 com 8	(02:58) e (03:78)
3	*aliasing* 8 com 11	(03:78) e (04:67)
4	*aliasing* 11 com 16	(04:67) e (06:59)

Fonte: Autor.

Os tempos de renderização neste caso são muito pequenos, tendo seu menor valor igual a um segundo e nove centésimos, porém, o maior valor, para *aliasing* 16x, chega a mais de 6.5 segundos, o que pode impactar na geração de múltiplas imagens seguidamente. Em seguida, a Tabela 6.2, informa os tempos de renderização para ambientes utilizando *ray-tracing*. Os tempos estão no formato (mm:ss:ff), sendo ‘m’ o equivalente a minutos, ‘s’ a segundos, e ‘f’ para centésimos de segundo.

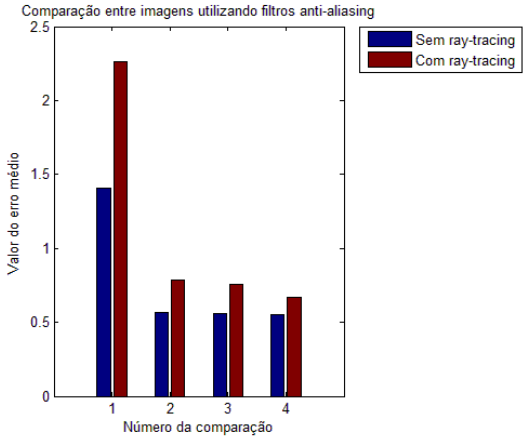
Tabela 6.2: Tempos de renderização com *ray-tracing.*

Número da Comparação	Elementos comparados	Tempos de renderização
1	*aliasing* 0 com 5	(00:24:34) e (01:48:24)
2	*aliasing* 5 com 8	(01:48:24) e (02:44:06)
3	*aliasing* 8 com 11	(02:44:06) e (03:36:36)
4	*aliasing* 11 com 16	(03:36:36) e (05:14:64)

Fonte: Autor.

Ao acrescentar o *ray-tracing* na renderização, temos um aumento drástico nos tempos, pois todas as cores serão calculadas de forma diferente, e muito mais pesada em termos de processamento, desta forma deve ser bem pensada a necessidade da reprodução de efeitos mais complexos na geração de imagens, e na qualidade destes efeitos. A Figura 6-10 informa o erro médio absoluto para cada uma das comparações especificadas nas Tabelas 6.1 e 6.2, sendo destacadas as sem *ray-tracing* em azul, e as contendo-o, em vinho.

Figura 6. 10: Estudo de efeito de filtro *anti-aliasing* em imgem.

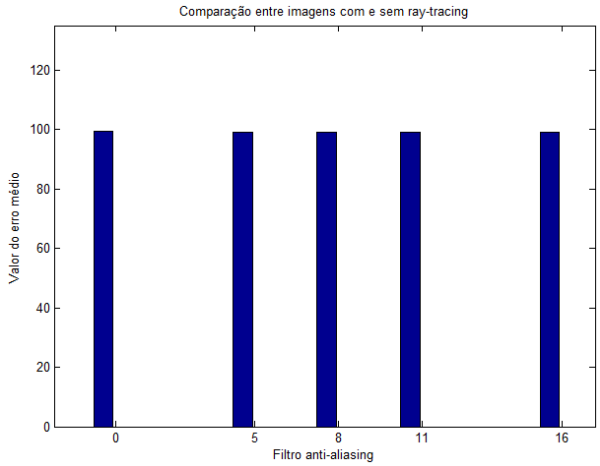


Fonte: Autor.

Sabendo que o valor máximo de cada pixel é 255, temos que estes erros médios são relativamente pequenos, o que deve acontecer mesmo, pois os filtros apenas amenizariam serrilhados presentes na imagem, assim melhorando a sua qualidade. Um ponto a ser notado é que ao se utilizar o filtro, para ambos os casos, temos uma modificação maior na imagem, comparando ao aumenta-lo, e como o aumento dos tempos de renderização é de forma quase constante, este seria o caso em que mais valeria a pena, ou seja, filtro *anti-aliasing* 5x.

Além de comparar os efeitos dos filtros nas imagens, serão comparadas as imagens com o mesmo filtro, porém, com e sem os efeitos de *ray-tracing*. A Figura 6.11 representa esta comparação.

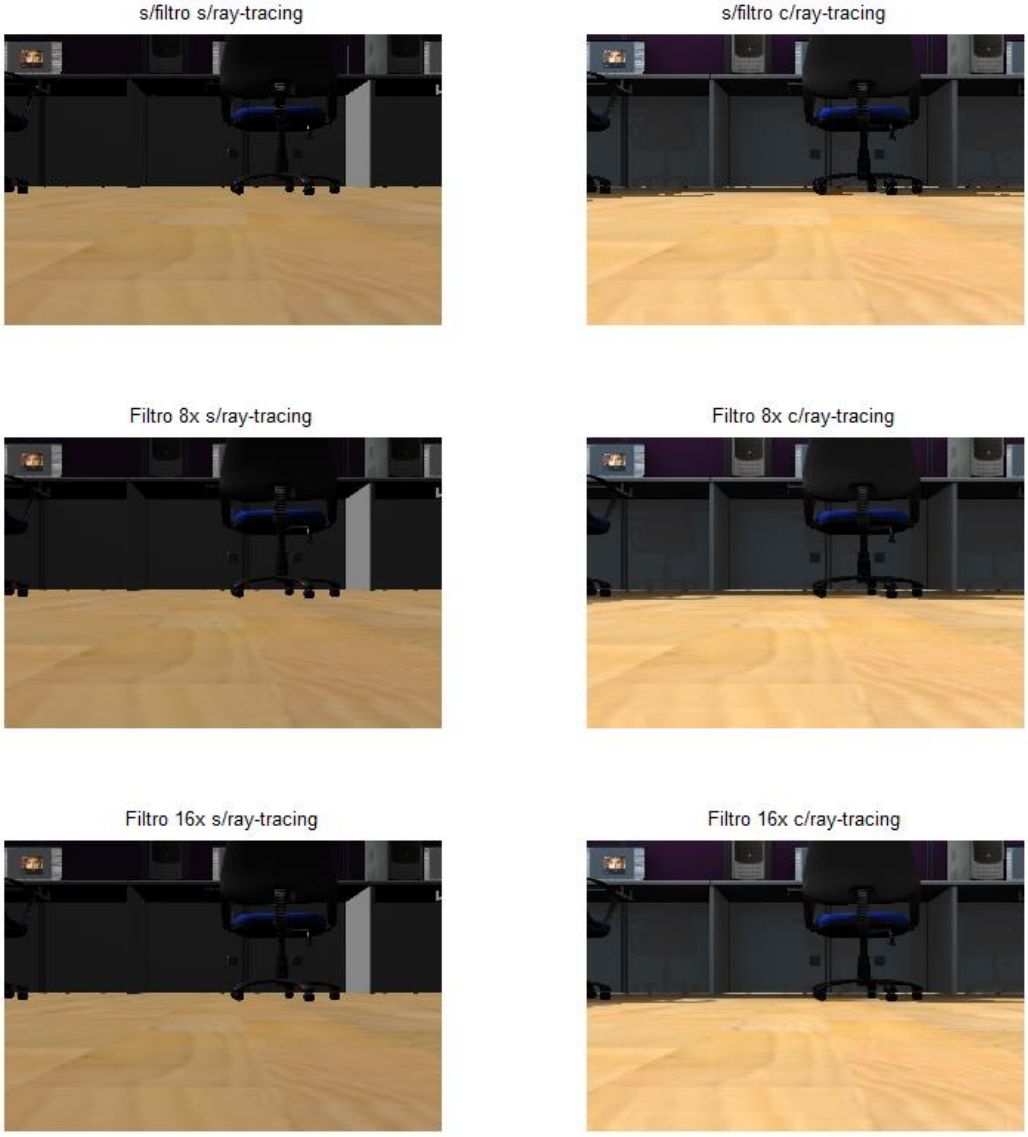
Figura 6. 11: Efeito da utilização de *ray-tracing* em imagem.



Fonte: Autor.

O que pode ser notado pela Figura 6.11, é que a diferença entre as imagens aumentou bastante, e que se comporta de forma constante, ou seja, a variação é de mesma intensidade independente do filtro, o que já era esperado, pois algumas outras características estarão presentes, como reflexão e transparência, assim, adicionando estes efeitos, é gerada uma grande modificação no valor dos pixels, e estes efeitos serão os mesmos para todas as imagens. A Figura 6.12 apresenta um comparativo entre imagens sem e com os níveis de filtro *anti-aliasing* iguais a 8x e 16x, com o objetivo de realizar uma análise qualitativa dos efeitos.

Figura 6. 12: Comparação entre imagens para filtros *anti-aliasing*.



Fonte: Autor.

Ainda pode ser feita uma análise qualitativa com relação ao ambiente de simulação baseado em um ambiente real. Na Figura 6.13 estão apresentadas imagens para ambos os casos, real e virtual.

Figura 6. 13: Comparação entre ambiente real e virtual.

a) Ambiente de Simulação b) Ambiente Real

Fonte: Autor.

Comparando a Figura 6.13a com a Figura 6.13b, pode-se perceber uma claridade maior, e uma invariância de iluminação no primeiro caso, outro aspecto é a maior nitidez na imagem a), porém, a semelhança entre ambas é evidente, aumentando ainda mais a gama de aplicações do trabalho.

7 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho teve como seu objetivo principal o planejamento, execução e conclusão de um simulador para robôs móveis, integrado com ambientes de programação de alto nível. Desta forma foi feito em duas partes, primeiro criando o simulador, e em seguida integrando-o. No desenvolvimento do simulador foi possível notar que as ferramentas presentes em uma plataforma como o Blender, voltada para a computação gráfica em geral, são muito extensas se comparado a um simulador de robótica especifico, isto porque este simulador também teria várias outras funcionalidades não presentes neste ambiente, como outros sensores e atuadores já programados e disponíveis em uma biblioteca, porém, como o objetivo principal é a geração de imagens, a implementação se fez necessária para uma maior liberdade na manipulação nos elementos de renderização.

É claro que esta liberdade teve um preço, pois todos os aspectos específicos à robótica deveriam ser implementados ao ambiente, e nesta parte que entrariam os modelos desenvolvidos nos ambientes de programação de alto nível, como o Matlab, pois juntando estes elementos é possível criar um conjunto onde se tenham ferramentas para a renderização e para aplicações especificas, além de uma grande liberdade para alterar ou até mesmo desenvolver estas ferramentas.

Além desta vantagem, por ser utilizado no desenvolvimento de jogos e animação, e também ser aberto, o Blender possui uma grande comunidade produzindo conteúdo, assim facilitando a geração de ambientes específicos, ou propiciando estes ambientes já prontos, um exemplo que foi dado no trabalho é que até mesmo a NASA tem uma página com muito conteúdo disponível.

Outro fator, é que ao se rodar duas aplicações, utilizando processamento simultâneo, surge o problema do tempo, mas como o caso é de uma simulação, não existe a necessidade da execução em tempo real. O que deve ser observado é que para ambientes de alta complexidade, os tempos de renderização podem ser de horas, ou até mesmo dias, sendo assim, deve ser bem analisada a real demanda destes efeitos para estudo, e o que se espera conseguir com isto.

Um exemplo de utilização para o trabalho foi apresentado nos resultados, onde até mesmo com a câmera voltada para o piso já foi possível implementar um algoritmo de estimação de trajetórias. Outras aplicações seriam, por exemplo, identificação de possíveis

colisões no caminho do robô, identificação de objetos na cena, iteração com estes objetos, e aplicações em geral utilizando processamento de imagem e visão computacional.

Além disso, o Blender incorpora um motor de física que pode ser usado para gerar a dinâmica dos objetos na cena e no caso de colisão entre um objeto e o robô, avisar o Matlab da força e torque aplicado no robô. Assim este conjunto de força e torque serão aplicados ao modelo no Matlab como ruído nas equações dinâmicas.

Podem ser acrescentados ao simulador outros aspectos da física dos objetos, podendo gerar condições similares até mesmo de outros planetas, e também, criações de objetos ou seres móveis no ambiente, pois isto seria um obstáculo a ser superado por algoritmos de visão, caso o movimento não pudesse ser previsto, além é claro de implementar no próprio ambiente novas técnicas de computação gráfica, já que este é de código aberto e baseado na biblioteca OpenGL.

As utilizações e possibilidades para as áreas de computação gráfica e robótica móvel são inúmeras e estão crescendo bastante. A metodologia deste trabalho pode ser aplicada facilmente para a movimentação de qualquer outro robô móvel possuindo o seu modelo dinâmico, desta forma podendo ser generalizada e ampliando a sua aplicação até mesmo para mais graus de liberdade e estudos para modelos ainda a serem desenvolvidos.

REFERÊNCIAS

AGX. ***AGX - AGplane***. Disponível em: < http://www.agx.com.br/ >. Acesso em: 25 out. 2015.

ASIMOV, I. ***I, Robot.*** *Voyager – HarperCollinsPublishers*. 1968. 249p.

BAJRACHARYA, M.; MAIMONE, M.W.; HELMICK, D. ***Autonomy for Mars Rovers: Past, Present, and Future.*** *IEEE Computer*, Vol. 41, *Issue* 12, *Dec*. 2008 p.44-50.

BECKER, Guilherme L. **Desenvolvimento de um simulador para um veículo autônomo.** Trabalho de Conclusão de Curso, UFMG, Belo Horizonte/MG, 2010. 80 p.

Bielefeld University. ***Pioneer 3DX.*** *Computer Engineering Group*. 2011. Disponível em: < https://www.ti.uni-bielefeld.de/html/research/equipment.html >. Acesos em: 12 set. 2015.

Boston Dynamics. ***Robotic Products (BigDog and RHex)***. Disponível em: < http://www.bostondynamics.com/content/sec.php?section=robotics >. Acesso em: 25 set. 2015.

COHEN, M.; MANSSOUR, I.H. **OpenGL - Uma Abordagem Prática e Objetiva**. São Paulo: Novatec, 2006. 486 p.

Coppelia. ***V-rep***. Disponível em: < http://www.coppeliarobotics.com/ >. Acesso em: 15 Out. 2015.

Cyberbotics. ***Webots***. Disponível em: < http://www.cyberbotics.com/ >. Acesso em: 15 Out. 2015.

De La CRUZ, C.; CARELLI, R. ***Dynamic modeling and centralized formation control of mobile robots.*** In: 32nd IEEE *Conference on Industrial Electronics*. [S.l.: s.n.], 2006. p. 3880–3885.

DUDEK, G.; Jenkin, M. (2000). ***Computational Principles of Mobile Robotics.*** Cambridge, London, UK: The MIT Press, 280 p.

Gazebosim. **Gazebo**. Disponível em: < http://www.gazebosim.org/ >. Acesso em: 15 Out. 2015.

GIBBS, W. ***Innovations from a robot rally***. *Scientific American*. *vol*.294, *January* 2006, p.64-71.

iRobot. ***Cleaning Robots (Roomba, Scooba, Dirt Dog, Verro).*** Disponível em: < http://www.irobot.com/ >. Acesso em: 17 jun. 2015.

Honda. ***Humanoid robot ASIMO***. Disponível em: < http://world.honda.com/ASIMO/ >. Acesso em: 23 jun. 2015.

Huskvarna. ***Automower (Robotic lawn mower)***. Disponível em: < http://www.automower.com/ >. Acesso em: 25 jun. 2015.

I-MSDN. ***Reference Platform***. Disponível em: < https://i-msdn.sec.s-msft.com/dynimg/ >. Acesso em: 15 Out. 2015.

JONASSEN, David. **O uso de novas tecnologias na educação a distância e a aprendizagem construtivista.** Em Aberto, Brasília, ano 16, n.70, abr/jun, 1996. 88 p.

JUNG, C. R.; OSÓRIO, F. S.; KELBER, C.; HEINEN, F. **Computação embarcada: Projeto e implementação de veículos autônomos inteligentes**, In: Anais do CSBC'05 XXIV Jornada de Atualização em Informática (JAI). 2005. São Leopoldo, RS: SBC, v. 1, p.1358–1406.

MANSSOUR, Isabel H. **Introdução à Computação Gráfica.** RITA, Volume XIII, Número 2, 2006. 25 p.

LIMA, Marcus V. P. **Reconstrução e Regularização de Trajetórias via Odometria Visual e Redes Neurais.** 2015. 62 f. Dissertação (Mestrado) - Curso de Engenharia Mecânica, Unicamp, Campinas, 2015.

MARTINS, F. N.; CARELLI, R.; SARCINELLI-FILHO, M.; BASTOS-FILHO, T. F. ***Dynamic Modeling and Adaptive Dynamic Compensation for Unicycle-Like Mobile Robots.*** 14th International Conference on Advanced Robotics - ICAR 2009, Germany, June, 22-26, 2009.

Microsoft. ***Microsoft Robotics Developer Studio***. Disponível em: < http://www.microsoft.com/robotics/ >. Acesso em: 15 Out. 2015.

NASA 3D resources. Disponível em: < http://nasa3d.arc.nasa.gov/detail/jpl-vtad-cassini >. Acesso em: 25 set. 2015.

NERIS, L. de Oliveira . **Um piloto automático para as aeronaves do projeto ARARA.** USP – ICMC. Dissertação de Mestrado do PGCCMC. Dez. 2001.

RIBEIRO, C.H.C.; COSTA, A.H.R.; e ROMERO, R.A.F. **Robôs móveis inteligentes: princípios e técnicas.** In: A.T. Martins, e D.L. Borges (Ed.), I Jornada de Atualização em

Inteligência Artificial - JAIA. Anais do XXI Congresso da Sociedade Brasileira de Computação, Fortaleza, CE, Vol. 3, 2001.

SAHIN, H.; GUVENC, L. ***Household robotics: autonomous devices for vacuuming and lawn mowing.*** IEEE Control Systems Magazine, Vol.27, N.2, Apr 2007 p.20-96

Sony. ***AIBO Entertainment Robots.*** Disponível em: < http://support.sony-europe.com/aibo/ >. Acesso em: 25 set. 2015.

THRUN, S. et al. Stanley: ***The Robot that Won the DARPA Grand Challenge***. Journal of Field Robotics, Vol. 23, No. 9, June 2006, p.661-692.

Traclabs. **Gazebo**. Disponível em: < http://traclabs.com/wp/wp-content/ >. Acesso em: 15 Out. 2015.

WATT, A. ***3D Computer Graphics.*** *3rd Edition*. Harlow: Addison-Wesley, 2000. 570 p.

WOLF, D. F.; SIMÕES, E. do Valle; OSÓRIO, F. S.; JUNIOR, O. T. **Robótica Móvel Inteligente: Da Simulação às Aplicações no Mundo Real.** 2009. 51p.

ZHANG, Y. et al. ***Dynamic model based robust tracking control of a differentially steered wheeled mobile robot.*** *American Control Conference*, v. 2, 1998.

ZIERHUT, Anthony. ***Animatics for Motion Pictures.*** Los Angeles, 2004.

APÊNDICE A – TRABALHOS PUBLICADOS

2015 International Conference on Indoor Positioning and Indoor Navigation (IPIN), 13-16 October 2015, Banff, Alberta, Canada

vSlam Experiments in a Custom Simulated Environment

Marcus V P Lima, Vinicius B Bastos, Paulo R G Kurka, Darla C Araujo
University of Campinas
Integrated Systems Laboratory
Campinas, Brazil
mvpleng@fem.unicamp.br

***Abstract*— In order to perform vSLAM experiments, a camera capturing information along a certain path is necessary. What if the resources to execute this are not available or a proof of concept is needed before a real application? This paper describes a simple method of combining MATLAB and Blender to perform vSLAM experiments in a custom created environment. A virtual robot and camera are created to navigate through the simulated space to capture textured images and objects that are used for the image processing. The images or video created by Blender can be easily processed by MATLAB as shown in this work. In addition, a simple application is demonstrated in the experiments section.**

Keywords— vSLAM; Simulated Environment; MATLAB; Blender;

I. INTRODUCTION

Simultaneous localization and mapping (SLAM) is a primary task for the navigation of autonomous robots [1]. The use of images to do such localization changes the abbreviation to vSLAM, where the "v" stands for visual. In the past years, a grown interest in Visual Odometry (VO) is observed, as shown in [2] and [3]. The use of cameras as sensors for mobile robot navigation seems to have many advantages in comparison to other alternatives. In first place, cameras are generally cheaper than most of other sensors such as laser, sonar, GPS, etc… in addition, they are low energy consumption devices and offer a bigger flexibility considering the high amount of information that can be extracted from its images. The process of visual odometry can be divided in three steps: detection of feature points in an image (corners and blobs), feature matching (identifying similar features in different images) and estimation of trajectory (using stereoscopic images and epipolar geometric relations). A variety of algorithms can perform reliable detection of features, such as the Shi-Tomasi[4] and Harris[5] corner detectors and the SIFT[6], SURF[7] detectors of bubbles and the Optical Flow technique [8] .

A 3D creation suite can be used to test such techniques in rendered images using high quality textures to give more realistic characteristics – as if the experiments were executed in a real environment. Blender [9] is an open-source 3D suite with those described features plus a Python scripting area to perform custom applications.

This work presents a method of using computational-only tools to perform experiments of vSLAM. The process is divided in three steps: robot dynamics, virtual environment and integration. The first step describes the implementation of the desired robot's dynamics in MATLAB and the use of existing projects for direct application. In the virtual environment section, the creation of the virtual robot and cameras, the texturing method and the necessary scripting to execute the navigation and acquire image data are described. The final step contains explanation of the integration of MATLAB and Blender for usage in vSLAM experiments.

The application of the proposed method is shown in section V. A simple differential robot is programmed to follow certain paths (in according to its dynamics) in a simulated indoor environment and through the optical flow technique, the path is estimated using the digital images captured in Blender

II. ROBOT DYNAMICS

This section describes the implementation of the robot's dynamics to be used in the vSLAM experiment. This is not a required step, but highly recommended to create a more realistic movement of the robot, in according to its behavior in a real application.

The work in [10] presents the creation of a robotics toolbox for MATLAB. This package allows the readily creation and manipulation of datatypes fundamental to robotics such as homogeneous transformations, quaternions and trajectories. This project is required for the application shown in this work.

The creation of a robotic system in Simulink can be exemplified in work [11], showing the parameter identification of real robots for simulation purposes. The result is a Simulink project that uses the robot's real parameters to perform navigation tests. Fig. 1 shows a simple system used for the robot dynamics simulation created with the tools described above.

The robot type chosen in this work is a unicycle-like. De la Cruz and Carelli[12] proposed a dynamic model (shown in Eq. 1) for this kind of robot represented in Fig. 2. The parameters and variables of interest in the model are the linear and angular

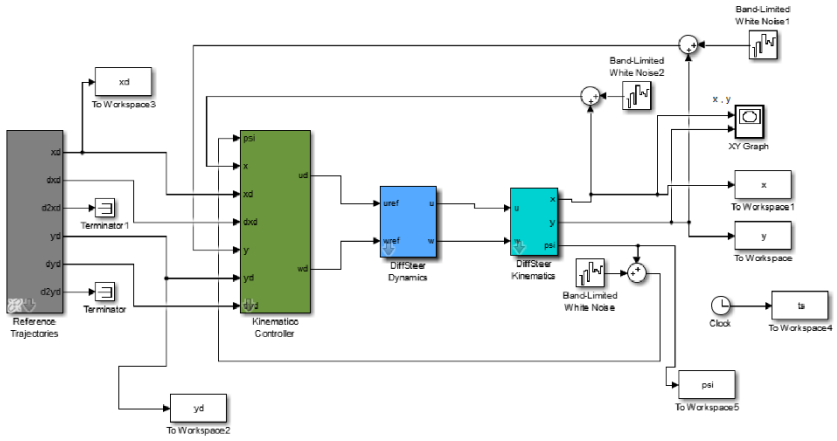


Fig. 1 – Differential robot system in Simulink

velocities of the robot, u and ω, respectively, the robot orientation angle ψ and the distance between wheels, a.

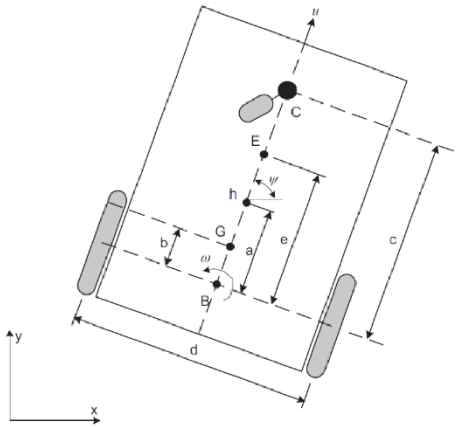


Fig. 2. The unicycle-like mobile robot [11]

The kinematics equation of motion of such a robot is given as:

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\psi}\\ \dot{u}\\ \dot{\omega}\end{bmatrix} = \begin{bmatrix}\cos\psi & -a\sin\psi\\ \cos\psi & a\sin\psi\\ 0 & 1\\ -\frac{\theta_4}{\theta_1} & \frac{\theta_3}{\theta_1}\omega\\ -\frac{\theta_5}{\theta_2}\omega & -\frac{\theta_6}{\theta_2}\end{bmatrix}\begin{bmatrix}u\\ \omega\end{bmatrix} = \begin{bmatrix}u\cos\psi - a\omega\sin\psi\\ u\cos\psi + a\omega\sin\psi\\ \omega\\ \frac{\theta_3}{\theta_1}\omega^2 - \frac{\theta_4}{\theta_1}u\\ -\frac{\theta_5}{\theta_2}u\omega - \frac{\theta_6}{\theta_2}\omega\end{bmatrix}, \quad (1)$$

where:

$$\theta_1 = \left[\frac{R_a}{k_a}(mr^2 + 2I_e) + 2rk_{DT}\right]\frac{1}{2rk_{PT}}[s]$$

$$\theta_2 = \frac{\left[\frac{R_a}{k_a}\left(I_e d^2 + 2r^2(I_z + mb^2)\right) + 2rdk_{DR}\right]}{2rdk_{FR}}[s],$$

$$\theta_3 = \frac{R_a}{k_a}\frac{mbr}{2k_{PT}}[sm/rad^2],$$

$$\theta_4 = \frac{R_a}{k_a}\left(\frac{k_a k_b}{R_a} + B_e\right)\frac{1}{rk_{PT}} + 1\ [1], \qquad \theta_5 = \frac{R_a}{k_a}\frac{mbr}{dk_{PR}}[s/m],$$

$$\theta_6 = \frac{R_a}{k_a}\left(\frac{k_a k_b}{R_a} + B_e\right)\frac{d}{2rk_{PR}} + 1\ [1].$$

In a practical implementation of the robot's model, the kinematics equation must include the desired values of the linear and angular velocities, u_{ref} and ω_{ref}, respectively, a vector of identified parameters θ and uncertainties of the position and velocities δ, yielding:

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\psi}\\ \dot{u}\\ \dot{\omega}\end{bmatrix} = \begin{bmatrix}\cos\psi & -a\sin\psi\\ \cos\psi & a\sin\psi\\ 0 & 1\\ -\frac{\theta_4}{\theta_1} & \frac{\theta_3}{\theta_1}\omega\\ -\frac{\theta_5}{\theta_2}\omega & -\frac{\theta_6}{\theta_2}\end{bmatrix}\begin{bmatrix}u\\ \omega\end{bmatrix} + \begin{bmatrix}0 & 0\\ 0 & 0\\ 0 & 0\\ \frac{1}{\theta_1} & 0\\ 0 & \frac{1}{\theta_2}\end{bmatrix}\begin{bmatrix}u_{ref}\\ \omega_{ref}\end{bmatrix} + \begin{bmatrix}\delta_x\\ \delta_y\\ 0\\ \delta_u\\ \delta_\omega\end{bmatrix} \quad (2)$$

$$\boldsymbol{\theta} = [\theta_1\ \theta_2\ \theta_3\ \theta_4\ \theta_5\ \theta_6]^T \qquad \boldsymbol{\delta} = [\delta_x\ \delta_y\ 0\ \delta_u\ \delta_\omega]^T$$

The input of the system is the desired trajectory and the output is the vector of positions ("x,y") and orientation ("psi") at each iteration defined in the Simulink project. The array of positions is used by Blender to perform the input trajectory. This process is detailed in section IV.

2015 International Conference on Indoor Positioning and Indoor Navigation (IPIN), 13-16 October 2015, Banff, Alberta, Canada

III. VIRTUAL ENVIRONMENT

In this section, the virtual environment created with Blender for the vSLAM applications is detailed. This open-source software is great for photorealistic 3D visualizations, therefore makes a perfect tool to be used with image processing due to the high quality of image details. The literature [13] describes the process of creating such realistic environments.

A sketch of a robot can be easily created using the forms available in the software: plane, cube, circle, sphere, cylinder, cone etc. An example of a robot used in the application described in section V is shown if Fig. 3 The robot shape will

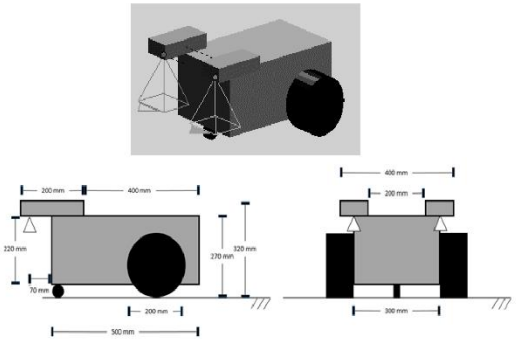


Fig. 3 – Sketch of the differential robot created in Blender

not have any effect, since the dynamics is only presented in the MATLAB model. However, it is possible to add effects such as wind and gravity by a custom Python script in the Blender software.

Scripting in Blender is supported by a large number of built-in and custom-created functions available in Blender's Org. Forum. To move and translate the robot, only 3 lines of code are used:

```
bpy.data.objects['Robot_001'].select = True
bpy.ops.transform.translate(value=(x,y,0))
bpy.ops.transform.rotate(value= r, axis=(0,0,1))
```

The first line is to select the object, then the command "bpy.ops.transform.translate" sends the object to a determined "x,y,z" position. Since in the application in this work is 2D, we set the 'z' to be zero. Also, the rotation of the robot corresponding to is orientation is on the vertical axis ('z' in Blender) and the 'x' and 'y' rotation are zero.

The camera can be easily added as it is already built in the software. Characteristics such as resolution, focus, sensor size, frame rate and aspect ratio are editable. Other rendering options are available: anti-aliasing, shading and post processing.

Textures are essential to the vSLAM, more details in the image provide the image processing algorithms more information for feature detections. The simplest way to add a texture is to fill an object or a plane with an image. The work in [14] describes a more advanced method to create more realistic objects. Fig. 4 illustrates the texture used in the application shown in section V.

It is important to acknowledge that the bigger the quality of the textures, the higher the processing needed to render the images. A way to improve processing speed is to replicate a piece of the texture in a lower resolution along the object or plane

Fig. 4 – Texture applied in the floor of the virtual environment

IV. INTEGRATION & VISUAL ODOMETRY

This section describes the communication between MATLAB and Blender to exchange the data necessary for the vSLAM experiments.

The integration is done by text file manipulation. The robot's movement array can be written in a ".txt" file by MATLAB code and read by a Python script in Blender and vice versa. The file contains the information separated by a simple space in a specific order which is known by the writing and reading software.

In Blender for example, to read the data the code below was used:

```
rfile = open('C:\Transfer\RobotPosition.txt','r')
data = rfile.read()
x = data[3:9]
y = data[9:15]
psi= data[16:22]
rfile.close
```

With this simple code, it is possible to capture the information written on the ".txt" file in a determined period of time to generate the animation. Some additional code might be needed depending on the precision of the numbers used and if the coordinates considered are only positive and/or negative.

After making the robot follow a desired path generated by matlab, it is necessary to capture images from each or a determined number of movement steps. In the main code, a picture can be automatically generated using an existing camera on the Blender's virtual environment by adding the code:

```
bpy.context.scene.objects.active = bpy.data.objects["Camera"]
bpy.data.objects['Camera'].select = True
bpy.context.scene.camera = bpy.data.objects["Camera"]
bpy.data.scenes['Scene'].render.filepath = "C:\Transfer\Images\Camera01 - %1.0f.jpg" % (NumName)
bpy.ops.render.render( write_still = True )
bpy.data.objects['Camera'].select = False
```

The process above, consists in selecting the camera and rendering the image in its line of view and save in a determined desired path. These images are used later in an image processing algorithm to estimate the robot's translation and rotation. This methodology allows the usage of any interface able to read images and apply image processing, using the most common languages such as C++, C#, Java, Python, Matlab.

Images of video produced by Blender are sent to a specific path configured by the user. In this work, we used MATLAB to execute the image processing. The images are accessed in the path defined in the code above. Fig 5 shows a summary of the communication process.

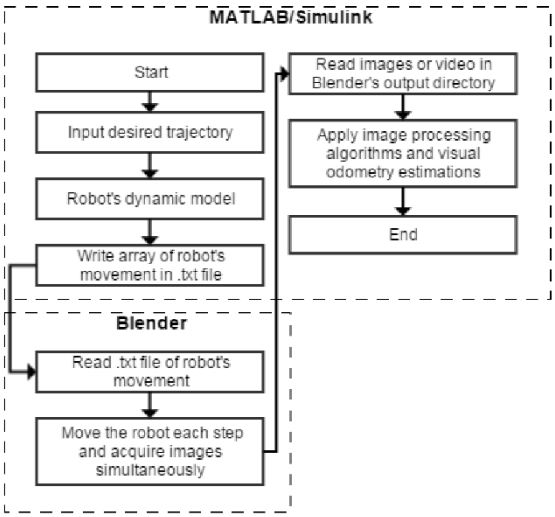


Fig. 5 – Summary of the communication process between MATLAB and Blender

Another software could be used for image processing. In example, a C# or Java application could read the Blender's output files instead of MATLAB.

V. VSLAM APLLICATION

This section describes a simple vSLAM application done in a simulated environment through the method described above.

The experiment consists in using stereoscopic images and use the optical flow technique to estimate the robot's trajectory.

The robot is programmed to follow the trajectories shown in Fig. 6. During its path, a couple of images are captured in each step of the movement and processed in MATLAB. Fig. 7 shows an example of interest points identified by the Optical Flow algorithm.

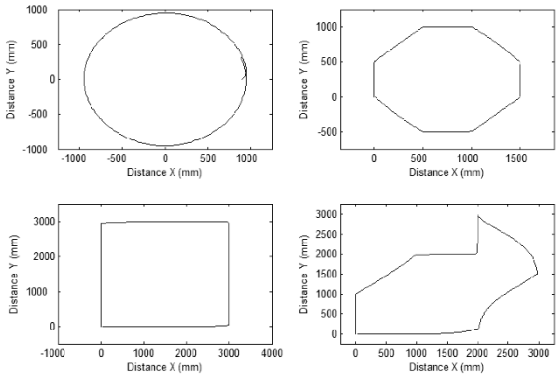


Fig. 6 – Paths used in the vSLAM experiment

Fig. 7 – Finding of interest points in left and right images

In many VO application, a type of data regularization is used. Generally, the data provided by the most used VO algorithms has a considerable imprecision and brings the need of an additional filtering step. In this work, we decided to use a Neural Network Filter. A simple network can be designed using MATLAB's Neural Fitting tool "nftool". Figure 8 illustrates the process and the characteristics of the Neural Network. The specifications used were 70% training, 15% Validation, 15% Testing, 10 hidden neurons and Bayesian Regularization. The inputs to train the network were the Circle, Octagon and Rectangle trajectories' 'x','y' and 'psi' in Figure 6 estimated by VO and the targets (output) were the known real values of such paths obtained with the robot dynamics simulation in Simulink . The concept idea was to train the network with different paths (straight, diagonal and

circular) to improve the results provided with a non-trained path.

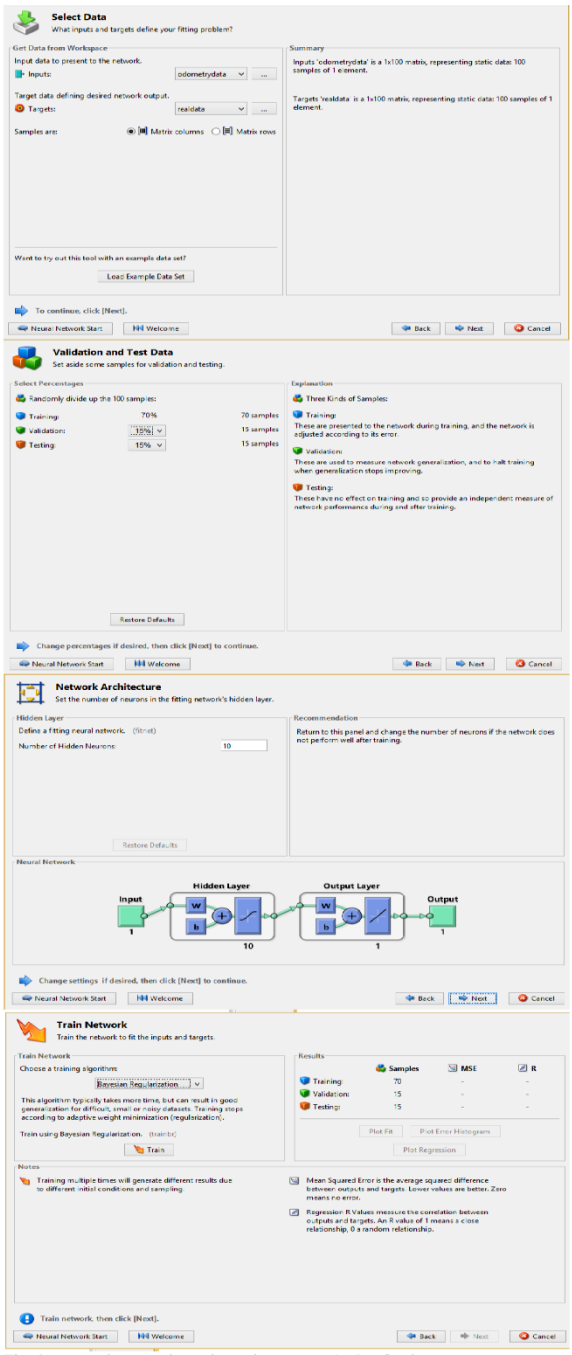


Fig. 8 – Neural Network Design using MATLAB's nftool

The results of the training are shown above in Fig. 9

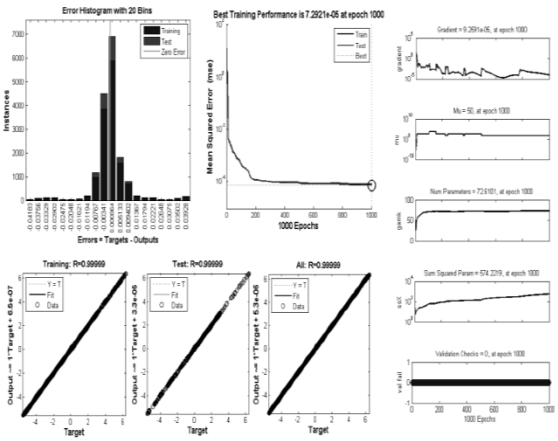


Fig. 9 – Neural Network training results

Using the information provided by the algorithm, the odometry estimation is executed and further filtering is applied to regularize the result. Fig. 10 illustrates the results obtained.

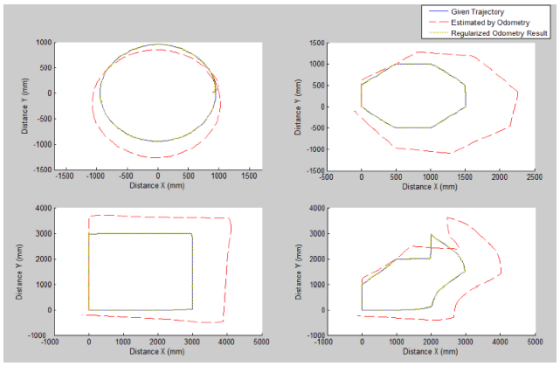


Fig. 10 – Results of the vSLAM experiment in an simulated environment

VI. CONCLUSION

The method described in this work can the successfully be applied in vSLAM experiments for proof of concept and even in a simulated environment with similar characteristics to the real ambient for simulations before the real experiment.

The communication between MATLAB and Blender requires some programming experience. Poor coding can lead to undesired delay in the whole process.

High quality textures provide rich information to the image processing algorithms, however processing speed is heavily affected. A lower resolution image can be used for better performance

Although the experiment present was in an indoor environment, the same methodology can be applied to outdoor ambient. Processing speed might be negatively affected

according to the number of objects and details in the environment.

Any researcher that desired fast results and easy deployment can use this method. This method provide a great way to test results before acquiring expensive equipment for vSLAM applications.

We further desire to apply the results of this work to control the trajectory of a robot in any indoor environment with minimal light conditions needed using training data to feed the Neural Network and improve the odometry results.

ACKNOWLEDGMENT

The authors wish to thank FAPEAM, the research-founding agency from the state of Amazonas, Brazil, for financing this work and the University of Campinas for providing the necessary infrastructure.

REFERENCES

[1] Weiss, Stephan, Davide Scaramuzza, and Roland Siegwart. "Monocular‐SLAM‐based navigation for autonomous micro helicopters in GPS‐denied environments." Journal of Field Robotics 28.6 (2011): 854-874.

[2] Huang, Albert S., et al. "Visual odometry and mapping for autonomous flight using an RGB-D camera." International Symposium on Robotics Research (ISRR). 2011.

[3] Konolige, Kurt, Motilal Agrawal, and Joan Sola. "Large-scale visual odometry for rough terrain." Robotics Research. Springer Berlin Heidelberg, 2011. 201-212.

[4] Sinha, U. "The Shi-Tomasi Corner Detector.", 2010.

[5] Ryu, J-B., H-H. Park, and J. Park. "Corner classification using Harris algorithm." Electronics letters 47.9 (2011): 536-538.

[6] Liu, Ce, Jenny Yuen, and Antonio Torralba. "Sift flow: Dense correspondence across scenes and its applications." Pattern Analysis and Machine Intelligence, IEEE Transactions on 33.5 (2011): 978-994.

[7] Knopp, Jan, et al. "Hough transform and 3D SURF for robust three dimensional classification." Computer Vision–ECCV 2010. Springer Berlin Heidelberg, 2010. 589-602.

[8] Sun, Deqing, Stefan Roth, and Michael J. Black. "Secrets of optical flow estimation and their principles." Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on. IEEE, 2010.

[9] Brito, Allan. "Blender 3D: Jogos e animações interativas." São Paulo: Novatec (2011).

[10] Corke, Peter. "A robotics toolbox for MATLAB." Robotics & Automation Magazine, IEEE 3.1 (1996): 24-32.

[11] Martins, Felipe N., et al. "An adaptive dynamic controller for autonomous mobile robot trajectory tracking." Control Engineering Practice 16.11 (2008): 1354-1363.

[12] De La Cruz, Celso, and Ricardo Carelli. "Dynamic modeling and centralized formation control of mobile robots." *IEEE Industrial Electronics, IECON 2006-32nd Annual Conference on*. IEEE, 2006.

[13] Brito, Allan. Blender 3D: Architecture, Buildings, and Scenery: Create photorealistic 3D architectural visualizations of buildings, interiors, and environmental scenery. Packt Publishing Ltd, 2008.

[14] DANA, Kristin. Bidirectional Texture Function and 3D Texture. Computer Vision: A Reference Guide, p. 46-50, 2014.