Faculdade de Engenharia da Universidade do Porto

FEUP

Development of an Intelligent Wheelchair 3D Simulator/Visualizer

Maria João Tavares Barbosa

Projecto de Dissertação realizado no âmbito do Mestrado Integrado
em Engenharia Informática e Computação
Áreas de Foco <Simulação Robótica/Motores de Jogos
/Cadeiras de Rodas Inteligentes/Interacção e Multimédia>

Orientador: Luís Paulo Gonçalves dos Reis (Professor Doutor)
Co-orientador: Rui Pedro Amaral Rodrigues (Professor Doutor)

Junho de 2011

Development of an Intelligent Wheelchair 3D Simulator/Visualizer

Maria João Tavares Barbosa

Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo Júri:

Presidente: Pedro Manuel Henriques da Cunha Abreu (Professor Doutor)

Vogal Externo: Artur José Carneiro Pereira (Professor Doutor)

Orientador: Luís Paulo Gonçalves dos Reis (Professor Doutor)

---

19 de Julho de 2011

Resumo

Com o aumento da esperança média de vida da população dos países mais desenvolvidos, manter e aumentar a qualidade de vida das mesmas é agora uma tarefa essencial. Tal como, o aumento da qualidade de vida das pessoas com algum tipo de deficiência física ou mental que as incapacite a nível motor, pois estas pessoas têm o mesmo direito que as restantes à própria independência e autonomia.

Antes de mais é necessário apresentar o projecto que se pretende desenvolver como dissertação de mestrado, cujo tema é "Development of an Intelligent Wheelchair Simulator/Visualizer".

Esta proposta de tese é uma investigação científica em torno de cadeiras de rodas inteligentes, ou seja, pretende focar principalmente as questões de independência e autonomia relacionadas com a capacidade motora.

Cadeiras de rodas inteligentes são a solução considerada para permitir menor dependência de utilizadores de cadeiras de rodas. A sua usabilidade é um factor crítico no desenvolvimento das mesmas e são, cada vez mais, lançados projectos de cadeiras de rodas inteligentes, cada um com o seu foco distinto. Os focos destes projectos apontam ora para metodologias de interacção com o utilizador, ora para a navegação autónoma, ora para o próprio hardware da cadeira de rodas inteligente. No entanto, ainda existem tópicos pouco ou nada explorados nesta área, mesmo havendo estudos de tecnologias e/ou metodologias relacionados já explorados.

O presente documento introduz os conceitos base para a elaboração de um trabalho de investigação científica na área de engenharia informática, focado nas subáreas de inteligência artificial, computação gráfica, robótica e, interacção e multimédia. Este trabalho científico pretende investigar e desenvolver um simulador e visualizador do comportamento de cadeiras de rodas inteligentes num mundo virtual semelhante ao mundo real. As abordagens gerais ao tema consideradas são: motores gráficos, físicos, de jogos e simuladores robóticos.

Este projecto foca essencialmente características de simulação com grande fidelidade física, características de visualização com grande fidelidade gráfica e automatização de configuração dos vários parâmetros necessários para uma simulação realística.

Outros objectivos, igualmente importantes, são a automatização de configuração do mundo virtual, cadeira de rodas e dispositivos de leitura de dados, bem como, a importação de objectos 3D modelados externamente e a possibilidade de comunicação de uma cadeira de rodas real com o simulador.

Este documento descreve uma aplicação desenvolvida com o objectivo de substituir dois módulos de um projecto mais geral, concretamente o módulo de simulação e visualização 3D. Portanto, pode ser considerado independentemente ou como uma parte de um todo do projecto IntellWheels iniciado em 2006 pelo Laboratório de Inteligência Artificial da Faculdade de Engenharia da Universidade do Porto. Como tal, no documento está presente uma apresentação geral do mesmo.

Inicialmente estudou-se as vantagens e desvantagens da utilização de motores gráficos, motores físicos, motores de jogos e/ou simuladores robóticos como plataforma base de desenvolvimento do projecto em questão. A decisão recaiu sobre a utilização do simulador robótico USARSim, simulador este que se encontra adaptado a robôs de rodas, construído por cima do motor de jogos Unreal Engine 3. O motor de jogos em questão tem alta fidelidade gráfica e física, e tem permanecido em constante actualização. O uso deste simulador robótico, em particular, como base do desenvolvimento desta aplicação diminuiu a dificuldade de implementação da mesma e permitiu uma programação de mais alto nível.

Ao longo do documento são especificados os requisitos da aplicação desenvolvida, o estudo das hipóteses e tecnologias consideradas aquando e para a implementação do simulador/visualizador. A implementação deste projecto é caracterizada, bem como, os resultados obtidos, e será apresentada como dissertação do mestrado integrado em engenharia informática e computação na Faculdade de Engenharia da Universidade do Porto.

Abstract

With the increase of the population's average life expectancy in most developed countries, maintaining and improving quality of life is now an essential task. The same applies to increasing the quality of life of people with some kind of physical or mental impairment that makes them physically handicapped, because these people have the same right as others to their own independence and autonomy.

This project, "Development of an Intelligent Wheelchair Simulator/Visualizer", is presented as a master's thesis. The thesis proposal is to do a scientific research on intelligent wheelchairs, which means that it intends to focus mainly on issues of independence and autonomy related to motor and locomotion skills.

Wheelchairs are the intelligent solution considered to allow wheelchair's users to be less dependent in charitable and compassionate people. Its usability is a critical factor in intelligent wheelchair's development and there is an increasing number of projects launched in this research area, each with a distinct focus. The foci of these projects point toward methodologies of user's interface, autonomous navigation, or the intelligent wheelchair hardware itself. However, there are still some topics with little or no work done, although the related technology and methodology needed for, has already been explored and investigated.

This document introduces the basic concepts for developing a scientific research work in the area of computer engineering, focused on subfields of artificial intelligence, computer graphics and robotics. This scientific work aims to investigate and develop a simulator and visualizer of the intelligent wheelchair behavior in a virtual world similar to the real world. The general approaches to the subject under consideration are: graphical engines, physical engines, game engines and robotic simulators.

This project concentrates mainly on features of high fidelity physics simulation, visualization features with high fidelity graphics and automation of various configuration parameters required for a realistic simulation.

Other objectives, equally important are the configuration's automation of the virtual world, the wheelchair 3D model and the input devices. Also the capability of importing externally modeled 3D objects and the ability to connect a real wheelchair to the simulator.

This document describes an application developed with the aim of replacing two modules of a broader project, namely the module of simulation and 3D visualization. Therefore, it can be considered independently or as a part of the main project named "IntellWheels" started in 2006 by the Artificial Intelligence Laboratory of FEUP (Faculty of Engineering of the University of Porto). For that reason, the present document has an overview of the IntellWheels platform.

Initially the advantages and disadvantages of using graphical engines, physical engines, game engines and robotic simulators as base platforms for development of the project in question were studied. The decision rested on the use of the robotic simulator named USARSim, which is suitable for wheeled robots and was built over the game engine Unreal Engine 3. This game engine has high graphical and physical fidelity and has been constantly

updated. This choice decreased the implementation's difficulty of the project proposal and allowed a programming at a higher level of abstraction.

Throughout the document the application's requirements, the technologies' study and the assumptions made are specified. The project's implementation is characterized, as well as the obtained results and it will be submitted as an integrated master's thesis in computer engineering and computer science at the Faculty of Engineering of the University of Porto.

Agradecimentos

Durante este longo processo de aprendizagem conheci e lidei com muitas pessoas, quero agradecer o contributo de todos os que me apoiaram e me ajudaram a amadurecer tanto a nível de conhecimento como a nível pessoal e profissional.

Primeiro um grande agradecimento a toda a minha família, da mais antiga à mais recente, sem vocês nada teria sido possível. Com especial atenção aos meus pais Renata Barbosa e Alexandre Valente Sousa, não esquecendo segundas mães, avós, bisavós, tios e tias, primos e primas. A todos vós obrigada pelo apoio, carinho, amizade, amor e pulso.

Em segundo, gostava de agradecer a todos os meus inicialmente colegas e posteriormente amigos, bem como aos meus amigos e amigas externos ao curso. Vocês que ouviram desabafos intermináveis sobre contratempos inesperados, e sempre me tentaram aconselhar da melhor forma possível, mantendo os meus pés em terra firme. A todos vós obrigada pelo apoio, carinho, amizade e teimosia que foram demonstrando.

Por último, mas não menos importante, quero agradecer a todos os professores que me acompanharam ao longo do meu percurso escolar, antes e durante este curso, e que de alguma forma me ajudaram a crescer e a aprender. Sempre me incentivaram a melhorar, transmitindo conhecimento, e por isso vos agradeço. Com especial atenção aos que além de desempenharem o seu papel de professor também desempenharam o papel de amigo e por quem comecei a sentir mais que respeito, admiração e carinho.

Obrigada, Maria João Tavares Barbosa.

Índice

Lista de Figuras

Lista de Tabelas

Glossário

Autocad – Programa de software que permite desenhar e modelar imagens a duas e três dimensões.
BGE – Motor de Jogos Blender Game Engine.
COG – Centro Geométrico (Center of Geometry)
CS – Motor Gráfico Crystal Space.
EEG – Electroencefalogramas (Electroencephalography).
FOV – Campo de Visão (Field Of View).
FTP – Protocolo de transferência de ficheiros (File Transfer Protocol)
GPS – Sistema de posicionamento global (Global Positioning System); envolve satélites e computadores que conseguem determinar a latitude e longitude do receptor.
HPL - Motor de Jogos HPL Engine entitulado HPL segundo o nome do autor (Howard Phillips Lovecraft)
jME – Motor Gráfico jMonkey Engine.
Maya – Programa de software que permite desenhar e modelar imagens a três dimensões
NGD – Biblioteca gráfica, Newton Game Dynamics.
ODE – Motor Físico Open Dynamics Engine.
OGRE – Motor Gráfico Object-Oriented Graphics Rendering Engine.
OpenGL – Biblioteca gráfica aberta (Open Graphics Library); consiste numa plataforma que permite programar aplicações de software que produz imagens a duas e três dimensões.
OSG – Motor Gráfico Open Scene Graph.
RFID – Identificação por rádio frequência (Radio Frequency Identification).
TCP – Protocolo de comunicação (Transmission Control Protocol).
UDK – Unreal Development Kit.
UDP – Protocolo de comunicação (User Datagram Protocol).
UE3 – Unreal Engine 3.
USARSim – Simulador robótico de pesquisa e salvamento urbano (Unified System for Automation and Robot Simulation)
USB – Consiste numa especificação para estabelecer comunicação entre dispositivos e um controlador principal (Universal Serial Bus).
UT3 – Jogo denominado Unreal Tournament 3.
UU – Unidades do motor de jogos Unreal (Unreal Units).
VFH – Histograma de campo vectorial (Vector Field Histogram).
XSI – o mesmo que Autodesk Softimage XSI, programa de software para modelação gráfica a três dimensões.
3DStudio Max – Programa de software que permite desenhar e modelar imagens a três dimensões (o mesmo que 3dsMax).

Capítulo 1

Introdução

Devido aos avanços tecnológicos em informática na área da computação gráfica e física e na simulação robótica e inteligência artificial é possível, hoje em dia, desenvolver um simulador e visualizador de cadeira de rodas inteligente com grande fidelidade gráfica e física, a custo reduzido.

O projecto pretende construir uma aplicação de software que permita testar o comportamento de vários modelos de cadeira de rodas com diferentes tipos de sensores e que proporcione grande facilidade de navegação e alta usabilidade por parte do utilizador final.

As características essenciais que a cadeira de rodas inteligente deverá implementar são a navegação independente com segurança, flexibilidade e capacidade de evitar obstáculos, comunicação com outros dispositivos e adaptabilidade ao utilizador.

Em termos de reconhecimento de entrada de dados deverá ser possível navegar a cadeira de rodas, com maior ou menor precisão, através de comandos de voz, expressões faciais, movimentos de cabeça, entradas do teclado e movimentos do joystick.

O objectivo final do projecto é possibilitar a coordenação da cadeira de rodas virtual com a cadeira de rodas real através de comunicação em rede, procedendo à implementação do conceito realidade aumentada.

Numa frase, o projecto consiste em criar um simulador e visualizador de uma cadeira de rodas inteligente, de preferência, com os dois módulos separados (módulo de simulação e módulo de visualização) que permita simular e visualizar num ambiente virtual o comportamento real de uma cadeira de rodas.

1.1 Motivação e Enquadramento no Mundo Real

O presente trabalho científico pretende proporcionar o aumento da autonomia e independência das pessoas mais velhas e/ou com algum tipo de deficiência motora. Este projecto pode ter aplicações variadas com grande contributo para o bem-estar humano.

A principal meta deste projecto é ajudar pessoas com deficiências físicas de locomoção, directa ou indirectamente. Estes indivíduos não têm uma qualidade de vida igual à da restante população embora já haja tecnologia que permita desenvolver aplicações de software suficientemente inteligentes para tal.

O projecto em questão tem várias aplicabilidades no mundo real, tais como:

- Teste de usabilidade e segurança de um edifício ou outra construção por parte de pessoas que dependem de uma cadeira de rodas para se movimentarem;

- Teste de fiabilidade de um dado modelo de cadeiras de rodas em circunstâncias de risco que podem gerar situações perigosas para o utilizador da mesma;
- Treino de pessoas que ficaram recentemente dependentes de uma cadeira de rodas para se movimentarem, sem correrem riscos de segurança, garantindo o seu bem-estar físico;
- Sensibilização para comportamentos de risco que podem comprometer a integridade física do próprio ou outros afectados por tal comportamento, como, por exemplo, a condução de veículos sob o efeito de álcool ou substâncias químicas que produzem alterações nos sentidos, aumentando o tempo de resposta do condutor.

1.2 Objectivos e Requisitos Gerais

O principal objectivo deste projecto é desenvolver um simulador e visualizador de cadeira de rodas inteligente.

Os pontos-chave do projecto a desenvolver são, primariamente:

- Alta fidelidade gráfica – O visualizador deve conseguir desenhar a cena (mundo virtual) e a cadeira de rodas virtual com a maior semelhança possível com o mundo real. Ou seja, as texturas devem ter uma qualidade aceitável, a luminosidade deve parecer real e a sensação de profundidade deve ser alcançada.
- Alta fidelidade física – As forças físicas devem ser representadas de tal forma que toda a parte física do mundo virtual seja coerente com o mundo real. Como, por exemplo, o movimento da cadeira de rodas que não é um movimento singular, mas o conjunto do movimento das suas rodas e elevação do assento.
- Permitir a importação de objectos modelados em três dimensões – Estes objectos podem transformar-se em objectos da cena ou corresponder à forma da cadeira de rodas virtual. O objectivo é permitir a importação de objectos modelados em Autocad, 3DStudio Max e Maya.
- Automatizar a descrição do cenário de visualização e simulação – O mundo virtual deverá ser facilmente e semi-automaticamente criado, tendo por base uma descrição geral do mesmo e não uma descrição pormenorizada de pontos e segmentos de recta.

Além dos pontos-chave referidos acima foram consideradas as seguintes metas, igualmente relevantes na contribuição cientifica deste ambiente de simulação:

- Automatizar a configuração de diferentes modelos de cadeira de rodas;
- Automatizar a configuração de outros dispositivos de leitura de dados, diferentes tipos de sensores;
- Sistema multi-plataforma com arquitectura modular, sendo o módulo de simulação distinto do módulo de visualização;
- Integração da aplicação desenvolvida com o controlo inteligente do projecto “IntellWheels”.

O projecto em questão deve implementar serviços com alta fidelidade, disponibilidade e segurança.

Em relação a requisitos de usabilidade, a interface do utilizador deve ser consistente e a documentação do utilizador final deve especificar em pormenor todos os detalhes de implementação do mesmo.

O requisito de interface mais importante é que o sistema deve proceder à análise dos dados de entrada de dispositivos de leitura como os sensores da cadeira de rodas real.

Na implementação, a aplicação deve ser independente do sistema operativo, ou seja, ser multi-plataforma.

Como restrição de design, considera-se que o sistema deve, se possível, mediante as tecnologias escolhidas, ter uma arquitectura modular, com o módulo de simulação distinto do módulo de visualização.

Em relação aos requisitos de suporte é importante considerar a extensibilidade, a manutenibilidade, a compatibilidade, a adaptabilidade e possibilidade de instalação e configuração do sistema.

Alguns serviços como a paragem da cadeira de rodas, em caso de colisão iminente com outros obstáculos, devem ter, como requisito de desempenho, exactidão de acção e alta disponibilidade.

O sistema deve também ter alta confiabilidade, ou seja, baixa frequência e gravidade de falha e possibilidade de recuperação e/ou previsão de falha.

1.3 Estrutura do Documento

O documento está dividido em seis capítulos, correspondendo este capítulo à introdução ao tema do documento.

No decorrer deste documento pretende-se especificar as opções de desenvolvimento de software possíveis e justificar a metodologia de desenvolvimento escolhida. Por conseguinte, são apresentados todos os conceitos necessários à compreensão do projecto e das tomadas de decisão. Alguns projectos desenvolvidos na área de cadeiras de rodas inteligentes são igualmente referidos.

O capítulo dois pretende explicar sucintamente em que consiste o trabalho científico desenvolvido e proceder a uma apresentação geral do projecto IntellWheels, projecto onde se enquadra o trabalho científico especificado ao longo deste documento.

O terceiro capítulo apresenta o estado da arte e pode ser subdividido em dois subcapítulos: cadeira de rodas inteligente e abordagens de simulação de ambientes virtuais. O primeiro refere interfaces de utilizador inteligentes e adaptativas, navegação de cadeira de rodas inteligente e alguns projectos de cadeiras de rodas inteligentes. O segundo introduz e caracteriza motores gráficos, motores físicos, motores de jogos e simuladores robóticos, bem como conceitos importantes relativos aos mesmos.

As opções de implementação seleccionadas são devidamente apresentadas e todas as decisões tomadas são justificadas de forma clara. No quarto capítulo o objectivo é detalhar a implementação do projecto em si, por outras palavras, este capítulo descreve o conceito e implementação do simulador e visualizador. O capítulo quatro pode ser subdividido em oito subcapítulos: Metodologia geral seleccionada; Outras decisões tecnológicas; Arquitectura modular; Interface gráfica do utilizador; Particularidades do USARSim e Unreal Engine; Modelação do meio de simulação; Modelação do comportamento robótico; e Interligação com o projecto 'IntellWheels'.

O capítulo cinco corresponde à experimentação e análise de resultados e pode ser decomposto nas seguintes secções: Simulação do Comportamento Robótico; Visualização do Ambiente de Simulação; Módulo de Controlo Robótico IntellWheels.

O último capítulo, sexto capítulo, apresenta as conclusões deduzidas aquando a experimentação e possível trabalho futuro.

Capítulo 2

Contexto do projecto de dissertação

Este capítulo pretende enquadrar o projecto de dissertação, correspondente ao desenvolvimento de um simulador/visualizador a três dimensões de cadeira de rodas inteligente, contextualizando-o no projecto IntellWheels.

Inicialmente, são enunciadas as características gerais do projecto IntellWheels para que o leitor tenha percepção do contexto do projecto de dissertação apresentado no presente documento. Em seguida, pretende-se especificar os objectivos do trabalho científico, ou seja, as metas alcançadas em conjunção com o projecto principal (projecto IntellWheels).

A meta principal deste trabalho científico é demonstrar a possibilidade de integração dos módulos de simulação e de visualização virtual na plataforma de simulação de cadeiras de rodas inteligentes, resultante do projecto IntellWheels. Não pretendendo este trabalho ser o produto final de substituição dos dois módulos, mas um protótipo de substituição possível de ser estendido para acompanhar o crescimento do projecto "mãe". Como por exemplo, a implementação de mais configurações, semi ou totalmente automáticas, desejáveis numa aplicação deste género e a extensão de novas funcionalidades. Este trabalho pretende, igualmente, demonstrar os prós da utilização de simuladores robóticos, entre outras metodologias, no desenvolvimento dos módulos acima enunciados, bem como, explicitar a complexidade de integração necessária para tal.

2.1 Plataforma IntellWheels

O projecto IntelWheels consiste numa plataforma de simulação e visualização de cadeira de rodas inteligente desenvolvido no laboratório de inteligência artificial na Faculdade de Engenharia da Universidade do Porto. [7] [67] [66]

Em seguida, serão referidas as características principais desta plataforma modular que se baseia no paradigma de sistemas multi-agente. Em relação à arquitectura do simulador, este pode ser dividido em vários módulos.

Os módulos principais são:

- Control Plataform – Plataforma de controlo responsável por listar as acções possíveis e executar a acção sugerida;
- Multimodal Interface – Interface multi-módulo responsável por identificar a acção que o utilizador pretende executar através da informação que os restantes módulos lhe enviam.

Outros módulos, igualmente importantes, correspondem a módulos de reconhecimento e análise de dados de entrada. Estes módulos são responsáveis por interpretar os seguintes dados de entrada:

- Comandos de voz;
- Expressões faciais;
- Movimentos de cabeça;
- Entradas do teclado;
- Movimentos do joystick.

A figura 2.1.1 mostra a estrutura do módulo de interface multimodal do projecto IntellWheels, sua interacção com os módulos principais e sua subdivisão.

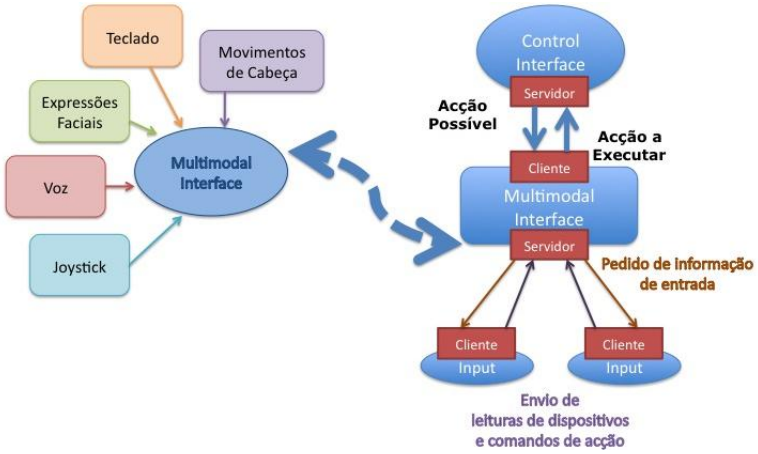


Figura 2.1.1 – Organização da Interface Multimodal de IntellWheels

Este projecto foi testado através de um protótipo de cadeira de rodas inteligente, presente na figura 2.1.2.

Figura 2.1.2 – Imagem do protótipo de cadeira de rodas do projecto IntellWheels

A característica mais distintiva e inovadora deste projecto é a implementação do conceito de realidade aumentada. A realidade aumentada é a junção em tempo real da realidade do mundo real com a realidade virtual, ou seja, o aproveitamento do conhecimento virtual aplicado ao mundo real [42]. Por exemplo, quando se vê um jogo de futebol na televisão e, na repetição de um remate, aparece uma linha vermelha para se compreender melhor se o passe anterior ao remate foi ou não fora-de-jogo. Neste exemplo, a linha vermelha é a realidade virtual, a filmagem do jogo é o mundo real e a sobreposição das duas corresponde à realidade aumentada.

Neste projecto a realidade aumentada é aplicada na troca de informação entre a simulação virtual do movimento da cadeira de rodas inteligente e a cadeira de rodas que se move no mundo real. Na próxima figura está representada essa troca de informação.

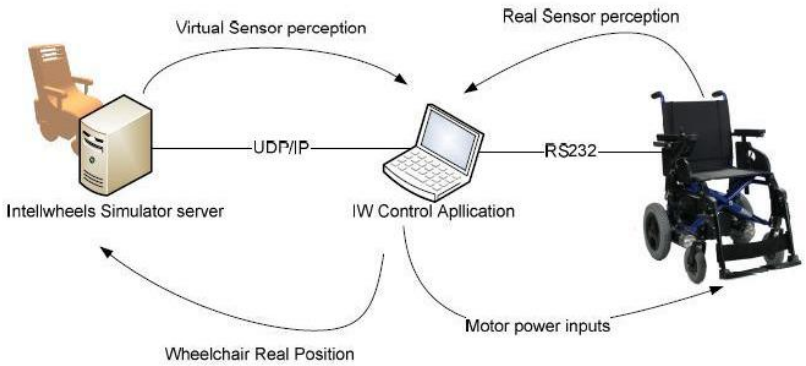


Figura 2.1.3 – Representação da troca de informação do simulador IntellWheels com a cadeira de rodas real.

Como é possível ver na figura 2.1.3, a cadeira de rodas real envia as percepções dos sensores à aplicação de controlo de cadeira de rodas inteligente que, por sua vez, calcula a posição real da cadeira de rodas e envia essa posição ao simulador IntellWheels. O simulador IntellWheels envia então a sua percepção sensorial virtual à aplicação de controlo que, por sua vez, envia os comandos de controlo do motor da cadeira de rodas real à própria e esta altera o seu movimento conforme comandada.

Na figura abaixo são mostradas as posições de movimento da cadeira de rodas em duas fases tanto no mundo real como na sua representação virtual.

Figura 2.1.4 – Representação real e virtual das posições inicial e final do movimento da cadeira de rodas inteligente

Na figura 2.1.4, no canto superior esquerdo é possível ver a cadeira de rodas real na posição inicial do seu movimento e no canto inferior esquerdo a sua representação virtual. No canto superior direito da mesma figura é possível ver a cadeira de rodas real na posição final do seu movimento e no canto inferior direito a sua representação virtual.

A figura 2.1.5 consiste na representação virtual criada pelo visualizador a duas dimensões de um exemplo de movimento da cadeira de rodas num corredor.

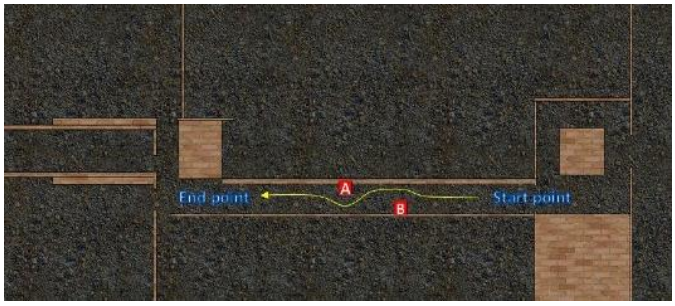


Figura 2.1.5 – Exemplo de visualização no visualizador a duas dimensões

Em seguida é apresentado um exemplo de visualização do mundo virtual através do visualizador a três dimensões, na figura 2.1.6.

Figura 2.1.6 – Exemplo de visualização no visualizador a três dimensões

A plataforma IntellWheels segue uma arquitectura modular, pode ser decomposta nos seis módulos básicos apresentados na figura abaixo (figura 2.1.7). Módulo de planeamento, interface, simulação, comunicação, navegação e hardware. [67] [66]

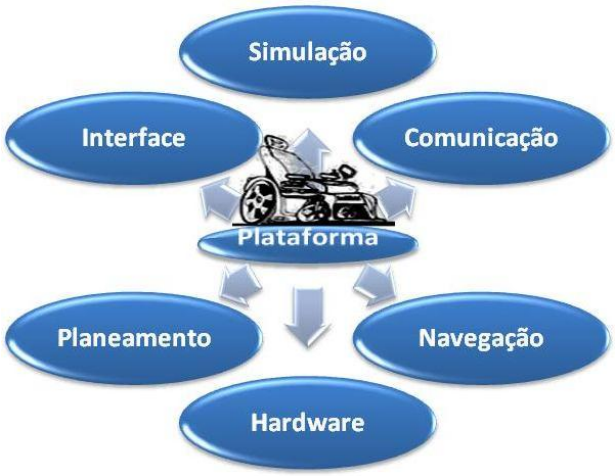


Figura 2.1.7 – Módulos básicos da plataforma IntellWheels

Em suma, num ponto de vista crítico, o projecto IntellWheels tem actualmente as seguintes características:

- Arquitectura modular;
- Realidade Aumentada;
- Simulador de objectos/robôs e tratamento de colisões a duas dimensões;
- Visualizador a três dimensões tem pouca qualidade a nível de texturas, sensação de profundidade e iluminação da cena;
- Difícil configuração de novos modelos de cadeiras de rodas e de novos dispositivos de leitura de dados;
- Apenas compatível com o sistema operativo Windows.

2.2 Conjunção dos dois projectos

A plataforma IntellWheels tem seis módulos principais enunciados na secção anterior, mas para a conjunção dos dois projectos só interessa considerar três módulos em particular, já enunciados: o simulador a duas dimensões, o visualizador a duas dimensões e o visualizador a

três dimensões. Estes módulos foram desenvolvidos recorrendo à linguagem de programação C/C++. O visualizador a três dimensões foi desenvolvido recorrendo à biblioteca gráfica OpenGL.

Na primeira fase de preparação para dissertação no âmbito da investigação científica proposta ao longo deste documento foi considerada a abordagem ao problema como uma adaptação modular do projecto IntellWheels. Se a escolha de abordagem se mantivesse, os objectivos da proposta de investigação científica seriam a adaptação do simulador a duas dimensões para simulador a três dimensões, acompanhado de reestruturação de funcionalidades como a detecção de colisão de objectos, entre outras. Outro objectivo seria a adaptação do visualizador a duas dimensões à nova versão do simulador. Por fim, posteriormente seria desenvolvido um novo visualizador a três dimensões.

Durante esta primeira fase foi decidido que seria mais proveitoso o desenvolvimento de um simulador e visualizador, a três dimensões, recorrendo a uma de quatro metodologias, utilização de motores gráficos, físicos, de jogos e/ou simuladores robóticos. Mantendo o objectivo de integração na plataforma IntellWheels, considerando a agregação da aplicação desenvolvida ao módulo de simulação da mesma, especificamente ao módulo de controlo inteligente do comportamento da cadeira de rodas virtual.

Uma decisão não menos importante foi qual o motor/simulador a utilizar como base de desenvolvimento da aplicação pretendida. Várias características foram consideradas aquando a escolha como, por exemplo, a fidelidade gráfica e física proporcionada, linguagem de programação, licença de proprietário e plataformas suportadas. A escolha recaiu sobre a utilização do simulador robótico USARSim, desenvolvido no topo do motor de jogos Unreal Engine 3. Esta decisão apoiou-se na elevada fidelidade gráfica e física, no facto da linguagem de programação ser C++, na facilidade de desenvolvimento, capacidade de comunicação em rede, bem como outros factores apontados no capítulo de conceito e implementação (capítulo quatro). Vários motores e simuladores robóticos foram considerados e analisados, as suas características gerais estão descritas no próximo capítulo (estado da arte).

Capítulo 3

Estado da Arte

Este capítulo pretende resumir os conceitos essenciais à compreensão do presente trabalho cientifico, apresentando o estado da arte em questão.

Numa primeira etapa pretende-se explicar a nomenclatura cadeira de rodas inteligente, focando os tópicos: interface de utilizador inteligente/adaptativa e navegação de cadeiras de rodas inteligentes; e referindo projectos de cadeiras de rodas inteligentes.

Numa segunda etapa são indicadas possíveis abordagens ao problema, ou seja, abordagens de simulação de ambientes virtuais. As metodologias gerais seleccionadas para análise foram motores gráficos, motores físicos, motores de jogos e simuladores robóticos. Nesta secção são introduzidos os conceitos base de motores/simuladores e, em seguida, procede-se à caracterização e comparação dos mesmos.

3.1 Cadeira de Rodas Inteligente

Uma cadeira de rodas pode ser vista como um transporte munido de rodas, cuja navegação é conseguida manualmente ou recorrendo a motores e foi concebida com o intuito de permitir mobilidade a indivíduos fisicamente incapacitados. O conceito de cadeira de rodas inteligente é o passo natural de investigação científica que permitiu o melhoramento das características de navegação tradicionais de uma cadeira de rodas e sua adaptabilidade ao utilizador.

Em essência, uma cadeira de rodas inteligente define-se como sendo um mecanismo de locomoção que assiste um utilizador que tem algum tipo de incapacidade física, quando um sistema de controlo artificial aumenta ou substitui o controlo do utilizador.

Os dois projectos principais de cadeiras de rodas inteligentes considerados neste documento são a plataforma 'IntellWheels', discutida no capítulo anterior, e o projecto 'Wheelesley'. O primeiro consiste num simulador e visualizador de cadeira de rodas inteligente baseado em realidade aumentada que implementa um conjunto de comandos de navegação de diferentes tipos de leitura de dados, de acordo com o paradigma de sistemas multi-agente. O segundo projecto consiste numa cadeira de rodas inteligente que aceita comandos de voz, aprende o ambiente físico fechado que a rodeia e navega num ambiente físico aberto através de localização GPS.

3.1.1 Interfaces de Utilizador Inteligentes/Adaptativas

A interface de utilizador é definida como a interface existente entre um ser humano e um computador, esta é uma componente muito importante num sistema computorizado que interaja com o ser humano.

A área de investigação científica de interfaces de utilizador inteligentes e adaptativas tem como objectivo facilitar a comunicação homem-máquina e recorre a modelos de utilizadores o que permite a automatização da adaptabilidade ao utilizador em questão. Muitas interfaces de utilizador adaptativas usam a aprendizagem máquina com o objectivo de melhorar esta interacção, para que o utilizador atinja o seu objectivo de uma forma mais fácil, intuitiva, rápida e com elevada satisfação. [52]

Os domínios de conhecimento essenciais numa interface adaptativa são:

- Conhecimento do utilizador;
- Conhecimento da interacção (modalidades de interacção e gestão de dialogo);
- Conhecimento da tarefa/domínio;
- Conhecimento das características do sistema.

As interfaces de utilizador inteligentes podem ser dividas em quatro classes principais:

- Manipulação directa de interface adaptativa;
- Interfaces informativas;
- Interfaces geradoras;
- Programação por demonstração.

A manipulação directa de interface adaptativa é baseada num modelo de utilizador que pretende prever as preferências e objectivos do mesmo ou reconhecer padrões comportamentais do utilizador. Estas previsões foram adaptadas em quatro formas principais:

- Execução especulativa – As acções previstas são tomadas especulativamente, de tal forma que, quando o utilizador fornece um comando, os passos de execução necessários já estão a ser executados ou já foram completos;
- Completação de padrão – Se os padrões comportamentais são evidentes, então vários comandos são combinados num único meta comando;
- Emissão rápida – Os comandos que executam uma acção ou sequência de acções previstas podem estar facilmente acessíveis ao utilizador para a sua rápida emissão;
- Assistência – O sistema pode oferecer ajuda e assistência baseada no seu conhecimento dos objectivos do utilizador.

Devido à vasta quantidade de informação disponível e acessível actualmente a partir de computadores é obvia a necessidade de filtragem da mesma. Esta necessidade é satisfeita por interfaces informativas que adoptam dois métodos principais:

- Baseado em conteúdo – Cada objecto é representado por um conjunto de descritores, como palavras num documento, e o sistema observa quais os objectos que são aceites e rejeitados pelo utilizador, aprendendo quais os descritores que indicam um provável positivo ou um provável negativo;
- Filtragem colaborativa (ou social) – Usado quando os descritores não são facilmente identificados ou o interesse do utilizador é subjectivo; Primeiro são identificados os conjuntos de objectos aceites e rejeitados pelo utilizador e, de

seguida, são intersectados com os conjuntos de objectos de outros perfis de utilizadores, podendo então ser sugeridos outros objectos de interesse potencial.

As interfaces geradoras criam informação nova a partir de valores de informação observados previamente. Ao inferir informação, o sistema reduz o tempo de entrada de informação por parte do operador humano, podendo até melhorar a qualidade dessa mesma informação. Esta geração de informação é melhorada a partir da aprendizagem por parte do sistema computorizado, por outras palavras, quanto mais informação é gerada maior qualidade tem.

A programação por demonstração é um método de comunicação de programas interactivos simples ao computador, sem ter de escrever o programa. O utilizador demonstra as acções requeridas e o sistema de programação por demonstração infere o conjunto de operações necessárias.

Quando se constrói um sistema de programação por demonstração deve-se ter em conta as considerações listadas abaixo:

- Representação das acções do utilizador;
- Representação de previsões;
- Detecção de ciclo;
- Generalização;
- Domínio de conhecimento;
- Programas de validação;
- Condições de terminação;
- Nível de intrusão.

Os principais requisitos na implementação de interfaces inteligentes e adaptativas são:

- Usabilidade – requisito funcional que avalia a facilidade que o utilizador tem em interagir com a interface;
- Acessibilidade – requisito funcional que avalia o grau de acessibilidade do sistema, normalmente focado em indivíduos com deficiências e no seu direito em aceder a tudo e todos;
- Segurança – requisito que expressa a confiança no sistema, relativamente à não ocorrência de avarias catastróficas, ou seja, avarias que colocam em causa a vida humana e/ou o ambiente.

3.1.2 Navegação de cadeira de rodas inteligente

A navegação de cadeiras de rodas inteligentes depende dos módulos de análise e reconhecimento de dispositivos de leitura de dados implementados.

Alguns dos dispositivos de leitura de dados usualmente implementados no âmbito da navegação de cadeiras de rodas são:

- Joystick (tradicional e/ou USB);
- Teclado;
- Câmara;
- Microfone;
- Sensor de toque;
- Sensor sonar;
- Sensor infra-vermelho;
- Dispositivo de localização GPS.

Todos os dispositivos de leitura de dados enunciados acima têm de ter um módulo de análise de dados a eles associado.

Alguns dos módulos de análise e tratamento de dados de maior dificuldade de implementação são os responsáveis por tratar:

- Comandos de voz (o ruído externo pode não permitir a identificação do comando pretendido);
- Movimentos de cabeça;
- Movimentos de dedo indicativo (ou outro);
- Detecção de expressões faciais.

A navegação com segurança em cadeiras de rodas inteligentes é proporcionada, em parte, pela detecção de colisões iminentes, proporcionada pela análise de dispositivos de leitura de dados, como os sensores sonar e os sensores de infra-vermelhos. A posição destes sensores tem de ser estudada cuidadosamente para que a colisão entre a cadeira de rodas inteligente e outros objectos não ocorra em caso algum.

3.1.3 Principais Projectos de Cadeiras de Rodas Inteligentes

Alguns protótipos/projectos de cadeiras de rodas inteligentes são apresentados sucintamente nas próximas secções e estão ordenados cronologicamente.

3.1.3.1 Cadeira de Rodas Autónoma

Este projecto foi iniciado a Fevereiro de 1985 por Nadaras com o objectivo de implementar um sistema computorizado que controla uma cadeira de rodas operada de forma totalmente autónoma. [33]

Esta cadeira de rodas é um veículo motorizado comum, equipado com um micro computador, uma câmara digital e um sensor de distância ultra-sónico.

Sempre que necessário o veículo processa todos as operações de planeamento, navegação, detecção de colisão eminente e desvio, localização de objectos e controlo de movimento.

O computador controla o movimento da cadeira a partir da interacção com o joystick da mesma. No entanto, a cadeira é um dispositivo impreciso, o que significa que, na maior parte das vezes, não se move como comandada.

O planeamento e estratégia de navegação é possível a partir da geração de uma sequência de operações primitivas de navegação entre a localização actual e a localização objectivo, após a inclusão de um modelo do mundo real.

Os dois mecanismos principais de sensores utilizados correspondem à visão e ao varrimento ultra-sónico. A visão permite a descrição detalhada do aspecto espacial do ambiente real tanto ao humano como à máquina. O varrimento ultra-sónico é usado primariamente para orientar a cadeira de rodas em relação às paredes e às entradas.

O controlo do movimento usa a visão e quando perde a sua localização actual e se desorienta, realinha-se relativamente a uma parede até voltar a ter percepção da sua localização actual no mapa.

A figura 3.1.3.1 corresponde ao protótipo de cadeira de rodas utilizado no projecto de cadeira de rodas autónoma apresentado neste projecto.

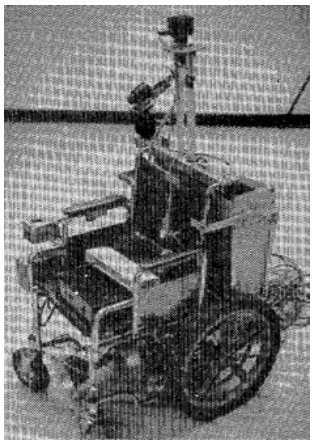


Figura 3.1.3.1 – Protótipo de cadeira de rodas autónoma projectado por Madarasz

3.1.3.2 Cadeira de rodas inteligente omnidireccional

Em 1993 foi apresentada uma arquitectura de controlo modular para uma cadeira de rodas omnidireccional por H. Hoyer e R. Hölper. A figura abaixo mostra o protótipo de cadeira de rodas inteligente deste projecto. [22]

Figura 3.1.3.2 – Protótipo de cadeira de rodas inteligente omnidireccional

3.1.3.3 Cadeira de rodas inteligente com duas pernas

Em 1994 foi a vez de Wellman apresentar um projecto inovador de cadeira de rodas inteligente, a inovação reflectiu-se principalmente na estrutura física da cadeira de rodas. O protótipo desta cadeira de rodas corresponde à figura 3.1.3.3. [58]

Figura 3.1.3.3 – Protótipo de cadeira de rodas inteligente de duas pernas

Este projecto consistiu no desenho e implementação de um protótipo versátil de cadeira de rodas, com duas pernas extra, que consegue executar grande parte das tarefas facultadas num veículo caminhante.

Esta cadeira de rodas pode utilizar as pernas como braços, ou seja, visto que o suporte da cadeira nem sempre é necessário, as pernas podem assim pegar em objectos e interagir com o ambiente externo. As pernas são utilizadas igualmente como suporte da cadeira, quando o objectivo é subir escadas ou o terreno é irregular.

3.1.3.4 NavChair

O projecto NavChair foi iniciado em 1994 e desenvolvido para satisfazer as necessidades das pessoas com algum tipo de deficiência que, outrora, não conseguiam operar os sistemas de cadeiras de rodas existentes. [3] [59]

Este projecto foi desenvolvido como uma aplicação de robôs móveis preparados para desvio de obstáculos e adaptado a cadeiras de rodas motorizadas.

O NavChair é um sistema homem-máquina, no qual a máquina partilha o controlo da cadeira de rodas com o utilizador da mesma. Por exemplo, no caso de ser necessário evitar um obstáculo, o comando da máquina sobrepõe o comando dado pelo utilizador.

O módulo de detecção e tratamento de colisões eminentes foi implementado adaptando o método histograma de campo vectorial (VFH) por permitir eficiente desvio de obstáculos baseado em sensores sonar de robôs móveis.

A figura abaixo corresponde ao protótipo de cadeira de rodas inteligente do projecto NavChair.

Figura 3.1.3.4 – Protótipo de cadeira de rodas inteligente NavChair

3.1.3.5 Tin Man

O projecto Tin Man I foi desenvolvido por Miller e Slak em 1995 e consiste num protótipo de cadeira de rodas inteligente com foco em cadeiras de rodas robóticos, ao invés do foco em robôs moveis presente nos projectos anteriores. [41] [59]

O sistema foi construído numa cadeira de rodas comercial da Vector Wheelchair Corporation e tem cinco tipos de sensores: codificadores DRIVE MOTOR, sensores de contacto, sensores de proximidade infra-vermelhos, sensores sonar de distância e FLUXGATE COMPASS (bússola de controlo de fluxo).

Os modos semi-automáticos possíveis são: guiado pelo humano com desvio de obstáculos, movimento em frente ao longo de um caminho, movimento do ponto/localização inicial para o ponto/localização final.

Em 1998 foi desenvolvido o protótipo Tin Man II com o objectivo de diminuir a dependência de sensores de contacto, modificar a interface do utilizador, aumentar a velocidade operacional e criar um sistema que facilite o processo de teste e verificação.

Os quatro modos básicos de operação passaram a ser: guiado pelo humano com desvio de obstáculos, virar evitando obstáculos, movimento em frente evitando obstáculos e modo manual.

Figura 3.1.3.5 – Protótipos de cadeira de rodas do Tin Man I e II

3.1.3.6 MAid

O projecto MAid ("Mobility Aid for Elderly and Disabled People") foi desenvolvido em 1998 com o objectivo de ajudar as pessoas idosas e com algum tipo de deficiência, mas não deseja reinventar ou re-implementar o conjunto de operações comuns como seguir parede e passagem por portas. [49] [50]

Visto o público alvo ter maioritariamente uma boa capacidade de condução e navegação da sua cadeira de rodas foi considerado como objectivo a condução de cadeira de rodas em ambientes sobrelotados, onde o número de objectos em mutante movimento é enorme e a possibilidade de colisão é muito recorrente.

Esta cadeira tem dois tipos de modos de operação, modo semi-autónomo NAN (narrow area navigation) para manobras em espaços pequenos e o modo totalmente autónomo WAN (wide area navigation) para manobras em espaços apinhados.

O MAid, em modo WAN, sobreviveu a dezoito horas de teste na estação central de Ulm e a trinta e seis horas na feira de Hannover, duas a três horas diárias durante sete dias em horário com maior número de visitas.

Este projecto teve, portanto, resultados experimentais muito bons, pois foi possível concluir que a cadeira de rodas em modo automático prevenia colisões em ambientes de rápida mutação com sucesso durante um espaço de tempo superior a trinta e seis horas.

A figura 3.1.3.6 mostra o protótipo de cadeira de rodas robótica solitário do lado esquerdo e do lado direito da imagem está o protótipo em pleno teste na estação central de Ulm.

Figura 3.1.3.6 – Cadeira de rodas robótica MAid solitária e em acção

3.1.3.7 Projecto FRIEND

O projecto FRIEND, desenvolvido em 1999, é constituído por um braço robótico funcional com uma interface de utilizador amigável para pessoas com incapacidades motoras. [6]

O desenvolvimento da interface multimédia de utilizador foi focado essencialmente em satisfazer os requisitos das pessoas com agilidade insuficiente a nível de dedos e mãos. Pessoas cujo o uso de joystick e teclado, para condução da cadeira de rodas, não é possível.

Neste projecto foram considerados dois sistemas de assistência que representam os dois principais conceitos de reabilitação robótica: movimentos pré-programados num ambiente estruturado e movimentos controlados pelo utilizador num ambiente desestruturado. Um dos sistemas é o HANDY, por ser pouco flexível e a reprogramação das operações só puder ser feita por especialistas, este sistema foi descartado. O outro sistema considerado é o MANUS, um braço robótico com comandos elementares, e foi adaptado ao projecto em questão.

As principais características do projecto são: o controlo de movimento pelo utilizador recorrendo a comandos de voz, o controlo do braço robótico, fixado na cadeira, pelo utilizador recorrendo a comandos de voz e a possibilidade de programar movimentos do braço robótico recorrendo a programação por demonstração.

Figura 3.1.3.7 – Braço robótico MANUS aplicado a uma cadeira de rodas

3.1.3.8 Cadeira de Rodas Inteligente baseada em agentes ACCoMo

O protótipo ACCoMo foi desenvolvido em 2004 por Tomoki Hamagami e Hironori Hirata para ajudar pessoas fisicamente incapacitadas a nível de mobilidade em ambientes fechados (entre portas).

Esta implementação baseia-se em robôs móveis e aplica os conceitos de aprendizagem máquina e paradigma de agente.

Este projecto consiste em desenvolver uma cadeira de rodas motorizada e inteligente com um comportamento autónomo, cooperativo e colaborativo. Esse comportamento é formulado a partir da aprendizagem e evolução de agentes inteligentes ACCoMo, através da sua experiência em ambientes reais e/ou virtuais.

O comportamento autónomo tem a capacidade de evitar obstáculos, paredes e colisões. Para que a resposta operacional seja rápida recorreram à aplicação de uma rede neuronal de duas camadas ao módulo de evitação de obstáculos do agente.

O comportamento cooperativo é visível aquando a cooperação dos agentes para adquirirem comportamento eficiente como colectivo, este comportamento foi conseguido através de programação genética.

O comportamento colaborativo consiste em garantir a comunicação com o utilizador e/ou outros equipamentos. Dois dos dispositivos cujo comportamento é colaborativo são: um ecrã

de toque inteligente e uma antena de identificação por rádio frequência (RFID) usada na navegação. [19]

O protótipo de cadeira de rodas inteligente deste projecto corresponde à imagem da figura 3.1.3.8, onde é descrita a configuração física da mesma.

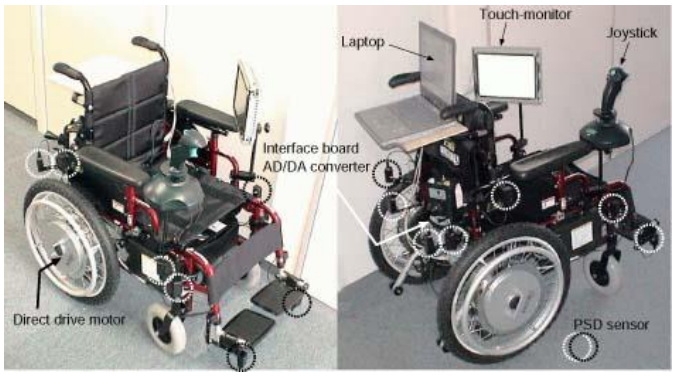


Figura 3.1.3.8 – Protótipo da cadeira de rodas inteligente baseada em agentes ACCoMo

3.1.3.9 Cadeira de rodas inteligente conduzida por ondas cerebrais

Este projecto foi iniciado em 2005 por H. Lakany da Universidade de Essex no Reino Unido e é o projecto mais inovador lançado recentemente.

O objectivo foi classificar os sinais electroencefalogramas (EEG) gravados a partir de um indivíduo, aquando o controlo de um joystick, movendo-o em diferentes sentidos.

A partir desta gravação de sinais desenvolveu-se um método baseado em extrair as propriedades espacio-temporais salientes dos sinais EEG, recorrendo à transformada contínua de oscilação do tipo onda. De seguida, foi realizada a classificação do sinal baseada em rede neuronal, estes resultados permitiram diferenciar as diferentes direcções de movimento. [29]

Nos últimos anos, a investigação relacionada com o controlo de braços robóticos aplicados a cadeiras de rodas e com o controlo de cadeiras de rodas inteligentes por pensamento evolui exponencialmente.

Na figura 3.1.3.9a é possível ver o protótipo de braço robótico aplicado a uma cadeira de rodas e controlado por pensamento [4].

Figura 3.1.3.9a – Braço robótico aplicado a uma cadeira de rodas controlado por ondas cerebrais [4].

Na figura 3.1.3.9b é possível ver o protótipo de cadeira de rodas inteligente conduzida por pensamento. Este protótipo foi desenvolvido por investigadores do BSI (Brain Science Institute) e financiado pelo Toyta Collaboration Center no Japão.

O sistema computorizado implementado permite conduzir uma cadeira de rodas através da análise das ondas cerebrais, analisadas cada cento e vinte cinco milissegundos. No caso de

erro, ou seja, caso a interpretação de comando pensado esteja errada, o utilizador pode accionar o travão de emergência, bastando encher a bochecha onde se encontra a ventosa de ar. Pode visualizar este protótipo em acção no vídeo em [40].

Figura 3.1.3.9b – Protótipo de cadeira de rodas inteligente conduzida por ondas cerebrais [40]

3.1.3.10 Projecto de cadeira de rodas autónoma do MIT

Um projecto de cadeira de rodas inteligente está a ser desenvolvido no MIT (Massachusetts Institute of Technology) com o objectivo de melhorar a independência de utilizadores de cadeiras de rodas. [55] [11] [10]

O objectivo principal do projecto é aumentar a autonomia da cadeira de rodas inteligente recorrendo às seguintes metodologias:

- Sensores para analisar o ambiente real em torno da cadeira de rodas;
- Interface responsável por interpretar comandos de voz;
- Dispositivo sem fios para determinar a actual localização em termos de espaço fechado de edifício, ou seja, determinar em que quarto de um determinado andar se encontra;
- Software de controlo de motor para comandar o movimento da cadeira de rodas.

O aspecto mais inovador deste projecto de cadeira de rodas inteligente autónoma é a capacidade de aprender a sua localização absoluta e relativa, tanto em espaços abertos como fechados, e ser capaz de conduzir o seu ocupante para a localização pretendida. Esta aprendizagem pode ser adquirida através de uma visita guiada, pois a cadeira de rodas inteligente tem a capacidade de seguir uma pessoa ao longo do edifício e criar um mapa virtual ao longo deste percurso através de comandos de voz.

Actualmente o sistema de navegação em espaços interiores baseia-se em redes sem fios, logo é necessário criar e configurar uma rede sem fios no edifício em questão. No entanto, outras tecnologias podem vir a ser implementadas para retirar a dependência de instalar redes sem fios no local de utilização de cadeira de rodas.

A navegação em espaços exteriores abertos é conseguida através do conhecimento da posição actual, localização GPS.

Este projecto inclui igualmente a capacidade de evitar colisões e contornar obstáculos como paredes, mobília, outros objectos e pessoas em andamento.

Os principais investigadores envolvidos na etapa do projecto responsável pela abordagem de comandos de voz são Nicholas Roy e Seth Teller (figura 3.1.3.10), mas há outros

investigadores que colaboraram com o projecto em questão. Este projecto foi subsidiado pela Nokia e Microsoft.

Figura 3.1.3.10 – Fotografia do protótipo de cadeira de rodas, Nicholas Roy e Seth Teller

Em resumo, o projecto de cadeira de rodas inteligente desenvolvido por investigadores do MIT é caracterizado por:

- Cadeira de rodas inteligente e autónoma;
- Aceita comandos de voz para se movimentar;
- Aprende o mundo real em espaços fechados memorizando um mapa virtual obtido a partir de "visita guiada";
- Movimento no mundo real em espaços exteriores através de localização GPS.

3.2 Abordagens de Simulação de Ambientes Virtuais

Esta secção introduz, caracteriza, classifica e compara vários motores gráficos, motores físicos, motores de jogos e simuladores robóticos.

3.2.1 Motores Gráficos

Como enunciado no início deste capítulo, uma das abordagens possíveis na implementação de um simulador e visualizador de uma cadeira de rodas inteligente é a utilização de motores gráficos como base para esta implementação. Nesta subsecção são enunciadas as vantagens de utilização de motores gráficos e uma curta descrição dos mais conhecidos e adaptáveis ao tema, bem como a comparação dos mesmos.

3.2.1.1 Introdução

Um motor gráfico é uma aplicação de software que facilita o desenho de objectos a duas ou três dimensões no espaço virtual.

Na medida em que o programador, por exemplo no desenho de um rectângulo, indica os quatro vértices do rectângulo pretendido segundo a ordem contra-relógio (se este estiver virado para a frente) ou a ordem dos ponteiros do relógio (se este estiver virado para trás) e o motor gráfico encarrega-se de desenhar os vértices, as arestas, preencher a face e torná-la visível do lado da frente ou do lado de trás, dependendo da ordenação dos vértices dados.

Outras funcionalidades, maioritariamente presentes num qualquer motor gráfico, são o desenho de figuras mais complexas, como a esfera, o disco e o cone, e também o tratamento da percepção de luz e profundidade.

O motor gráfico é usualmente parte integrante de um motor de jogo mas, cada motor de jogos tem por base um determinado motor gráfico.

Nas próximas subsecções serão analisados e discutidos os motores gráficos mais conhecidos.

3.2.1.2 Caracterização de alguns motores

Nesta subsecção são analisados e descritos alguns motores gráficos que, como declarado na introdução, correspondem a uma das abordagens possíveis à investigação científica considerada.

3.2.1.2.1 CrystalSpace

O CrystalSpace (CS) é principalmente um motor gráfico, cujo desenvolvimento se iniciou em 1999, capaz de desenhar ambientes virtuais 3D em tempo real. [14] [37] [56]

Este motor é um programa de software com licença de proprietário aberta criado para o desenvolvimento de videojogos modernos e ambientes virtuais. No entanto, foi classificado como motor gráfico por ser essa a sua vertente melhor implementada em vez de ser classificado como um motor de jogos.

A licença de proprietário é a licença GNU, licença pública geral menor (LGPL - “Lesser General Public License”).

A linguagem de programação do CS é C++, mas permite a integração de código escrito em Python (melhor suportada), Perl e Java.

O motor aceita como plataformas de desenvolvimento as maiores plataformas actuais, como os sistemas operativos Windows, Linux e MacOS/X e os processadores x86, AMD64 e PowerPC.

O desenho gráfico é baseado em OpenGL, o tratamento de luz inclui luzes estáticas, semi-estáticas e dinâmicas, permite a aplicação de texturas a objectos e representa sombras simples e sombras variáveis ponto a ponto.

3.2.1.2.2 jMonkey Engine

O motor jMonkey Engine (jME) é um motor gráfico concentrado em permitir um alto desempenho gráfico, foi criado por Mark Powell em Junho de 2003, sendo depois desenvolvido por outros. [21]

A licença de proprietário aplicada ao motor gráfico foi a BSD por ser o tipo de licença de código aberto menos restritiva sem o tornar de domínio público.

A linguagem de programação é Java, pode ser usado para criar sistemas gráficos 2D e 3D e baseia-se em OpenGL.

O facto de ser cem por cento em Java torna o grau da sua portabilidade enorme.

3.2.1.2.3 OGRE (Object-Oriented Graphics Rendering Engine)

O motor gráfico OGRE foi desenvolvido de forma a providenciar uma programação orientada a objectos, a sua linguagem de programação é C++ e a sua licença de proprietário é a licença GNU, licença pública geral menor (LGPL - "Lesser General Public License"). [34]

As plataformas suportadas são Windows, Linux e MacOS/X, mas não suporta tão bem a última enunciada. O OGRE é um motor independente da plataforma e com alta portabilidade.

Este motor é caracterizado pela sua flexibilidade, visto permitir a comunicação com ferramentas externas como bibliotecas físicas e de inteligência artificial. Assim como permite configurar propriedades de objectos em ficheiros de texto que são lidos aquando a inicialização da aplicação a ser criada.

3.2.1.2.4 OpenSceneGraph

Os principais contribuidores para o desenvolvimento do motor gráfico OpenSceneGraph (OSG) foram Robert Osfield, Paul Martz e Bob Kuehne. [44] [51]

A sua linguagem de programação é C++ e a sua licença de proprietário é a licença GNU, licença pública geral menor (LGPL - "Lesser General Public License").

Este motor suporta os seguintes sistemas operativos: Windows, Linux, MacOS, Solaris, SunOS, FreeBSD, Irix, Playstation.

3.2.1.3 Comparação dos motores

Alguns dos motores descritos na secção anterior têm uma pequena vertente responsável por implementar o comportamento físico, mas são considerados motores gráficos por ser essa a sua vertente principal.

Na tabela abaixo estão descritas as propriedades principais de cada motor gráfico descrito na secção anterior.

Motor Gráfico	Linguagem de Programação	Licença de Proprietário	Plataformas Suportados
CrystalSpace	C++ (Python, Perl e Java)	GNU LGPL	Windows, Linux MacOS/X, x86, AMD64 e PowerPC
jMonkey Engine	Java	BSD	Windows, Linux, MasOS/X, Solaris e SunOS
OGRE	C++	GNU LGPL	Windows, Linux e MacOS/X
OSG	C++	GNU LGPL	Windows, Linux, MacOS, Solaris, SunOS, FreeBSD, Irix, Playstation

Tabela 3.2.1 – Tabela de comparação de motores gráficos

3.2.2 Motores Físicos

Como enunciado no início deste capítulo, uma das abordagens possíveis na implementação de um simulador e visualizador de uma cadeira de rodas inteligente é a utilização de motores físicos como base para esta implementação. Nesta secção são enunciadas as vantagens de utilização de motores físicos e uma curta descrição dos mais conhecidos e adaptáveis ao tema, bem como a comparação dos mesmos.

3.2.2.1 Introdução

Um motor de física é uma aplicação de software, normalmente parte integrante de um motor de jogos, responsável por representar as leis de Newton num espaço três dimensões virtual.

O objectivo deste tipo de motor é permitir ao programador, que não tem obrigatoriamente de conhecer leis físicas do mundo real, uma abstracção dessas mesmas leis e apresentar uma interface de mais alto nível que permita a interacção entre objectos representados num mundo virtual com características físicas presentes no mundo real de forma a tomar mais realista a aplicação desenvolvida.

A importância de uma funcionalidade versus outra, obtidas por se utilizar um motor de física é relativa, pois depende directamente dos objectivos e contexto da aplicação a desenvolver com a ajuda do motor de física escolhido.

Embora o motor de física deva ser escolhido tendo em conta as funcionalidades do mesmo, relevantes para a concretização do projecto do programador, algumas funcionalidades sobressaem.

Do ponto de vista de que é essencial um motor de física para facilitar a implementação da aplicação do programador, sobressaem as seguintes funcionalidades do motor:

- Análise e tratamento da força gravitacional a que estão sujeitos todos os objectos da cena;
- Análise e tratamento do movimento dos vários objectos da cena, tendo em conta a força de atrito entre o objecto e o plano onde este está assente;
- Detecção e tratamento de colisões entre objectos, considerando a sua massa, forma, ponto de equilíbrio, posições e velocidades ao longo do tempo e as forças a que estão sujeitos.

Nas próximas subsecções serão analisadas em mais profundidade os motores de física mais conhecidos.

3.2.2.2 Caracterização de alguns motores

Nesta subsecção são descritos sumariamente quatro motores físicos: Bullet, Havok, ODE e PhysX.

3.2.2.2.1 Bullet

O motor de física Bullet consiste numa biblioteca com funcionalidades de detecção de colisão e representação de dinâmica de corpo rígido para jogos e animação. Esta biblioteca é de código aberto segundo os termos da licença ZLib. [9] [8] [30]

O sistema principal de compilação do Bullet é o cmake que consegue auto-gerar ficheiros de projecto para as plataformas de desenvolvimento Microsoft Visual Studio, Apple Xcode, KDevelop e makefiles para Unix.

O Bullet está melhor optimizado para x86 SIMD SSE, Cell SPE e CUDA.

As principais funcionalidades do motor físico em questão são:

- Simulação de corpo rígido e corpo flexível/macio com detecção discreta e contínua de colisões;

- As formas de colisão incluem: esfera, caixa, cilindro, cone e banda elástica convexa (convex hull), ou seja, invólucro convexo de um conjunto de vértices que formam um polígono; Este último é obtido usando GJK e malha triangular não convexa;
- O suporte de corpos flexíveis inclui objectos deformáveis, roupa e corda; Tem um conjunto rico de limitações de corpo rígido e flexível com restrições de limite e motoras;
- As plataformas suportadas são: PlayStation 3, XBox 360, Wii, Mac, iPhone, Linux, Windows, entre outras.

Em relação à licença de proprietário é possível ver na tabela abaixo as diferentes licenças aplicadas a ficheiros de código.

Licença de proprietário	Número de ficheiros a que foi aplicada
Simplified BSD License	826
GNU Lesser General Public License v3	51
Zlib/libpng license	19
GNU General Public License 2.0	2

Tabela 3.2.2.1 – Tabela com a correspondência da licença de proprietário ao número de ficheiros de código a que foi aplicada (motor físico Bullet)

Em relação a linguagens de programação já foram usadas: C++, C, CMake, Objective-C, Automake, shell script, Python, Perl e DOS batch script. As mais utilizadas até ao momento foram as duas primeiras enunciadas.

3.2.2.2.2 Havok Physics

O Havok Physics é um motor de física desenvolvido pela empresa irlandesa Havok recorrendo às linguagens de programação C e C++. Foi desenhado primariamente para jogos de consola, computador e videojogos, e permite o tratamento de colisão em tempo real e dinâmica de corpo rígido a três dimensões. [36] [20]

Este motor proporciona múltiplos tipos de restrições dinâmicas entre corpos rígidos e tem uma biblioteca de detecção de colisão muito optimizada. Como implementa simulação dinâmica, assegura um grande realismo nos mundos virtuais criados a partir do mesmo.

Neste momento, o presente motor de física é compatível com Microsoft Windows, Xbox, Xbox 360, GameCube da Nintendo, Wii, PlayStation (2, 3 e Portable) da Sony, Linux e MacOS/X.

Parte do código fonte do motor é aberto, o que fornece liberdade em personalizar algumas funcionalidades do motor ou mudar a portabilidade do motor, mas há outras bibliotecas que são fornecidas em formato binário.

Este motor tem muitas funcionalidades, fornecendo suporte às seguintes áreas principais:

- Detecção de colisão;
- Implementação de restrições e dinâmica;
- Controlo de personagem;
- Ferramentas de criação de conteúdo para 3DStudio Max, Maya e XSI;
- Implementação de integração de veículo;
- Reflexão e serialização de classes.

3.2.2.2.3 Open Dynamics Engine (ODE)

O motor de física Open Dynamics Engine começou a ser desenvolvido em 2001 e tem dois componentes principais, motor de simulação da dinâmica de corpo rígido e motor de detecção de colisão. Este software é gratuito e os direitos da utilização de código estão sob ambas as seguintes licenças, BSD e LGPL ("Lesser General Public License"). [35] [43]

Este motor é independente de plataforma e tem disponível uma API de C/C++. As suas funcionalidades principais são:

- Simulação da dinâmica de interacções entre corpos no espaço;
- Simulação de corpos articulados, com aplicação de vários tipos de objectos de união;
- Não estar preso a nenhum pacote gráfico em particular;
- Suportar várias formas geométricas como caixa, esfera, cápsula, malhas de triângulos em 3D, cilindros e malhas de peso;
- Implementar a detecção de colisão com fricção;
- Ser usado para simulação de veículos, objectos em ambientes de realidade virtual e criaturas virtuais.

As áreas de foco da utilidade do ODE são o desenvolvimento de jogos de computador, ferramentas de desenho 3D e ferramentas de simulação.

3.2.2.2.4 PhysX

O motor PhysX é um motor físico proprietário desenvolvido pela empresa Ageia, adquirido pela Nvidia e foi desenvolvido recorrendo à linguagem de programação C++. [47] [46]

Este motor é compatível com as seguintes plataformas: MacOS/X, Windows, Linux, Nintendo Wii, Sony PlayStation 3 e Microsoft Xbox 360. Não é possível ver nem modificar o código que trata a complexidade de interacção física nem usá-lo gratuitamente, à excepção de utilizadores e desenvolvedores de Windows e Linux. Este motor é gratuito para Sony PlayStation 3.

As funcionalidades principais disponibilizadas pelo PhysX são:

- Sistema físico para objectos complexos de corpo rígido;
- Controlo avançado de personagem;
- Dinâmica de veículos articulados;
- Criação e simulação de fluidos volumétricos;
- Desenho e reprodução de panos e roupas;
- Corpos flexíveis;
- Simulação de campo de força volumétrica.

3.2.2.3 Comparação dos motores

Todos os motores físicos descritos acima têm funcionalidades com grande fidelidade física e, à excepção de PhysX, são todos de código aberto. As linguagens de programação mais comuns são o C e C++ e são todos programas de desenvolvimento de software que suportam grande parte das plataformas existentes.

As funcionalidades que têm em comum são o tratamento de detecção de colisões e a simulação da dinâmica de corpo rígido e flexível, variam maioritariamente nas formas geométricas que disponibilizam entre outras funcionalidades mais singulares. [5]

Na tabela abaixo estão evidenciadas as características principais dos motores de física descritos na secção anterior.

Motor Físico	Linguagem de Programação	Licença de Proprietário	Plataformas Suportados
Bullet	C++, C, CMake, Objective-C, Automake, shell script, Python, Perl e DOS batch script	BSD / GNU LGPL v3/ Zlib / GNU GPL 2.0	PlayStation 3, XBox 360, Wii, Mac, iPhone, Linux, Windows, entre outras
Havok Physics	C e C++	Proprietário/ shareware	Microsoft Windows, Xbox, Xbox 360, GameCube da Nintendo, Wii, PlayStation (2, 3 e Portable) da Sony, Linux e MacOS/X
Open Dynamics Engine (ODE)	C e C++	BSD e GNU LGPL	Independente de Plataforma
PhysX	C++	Proprietário, Gratuito e Comercial	MacOS/X, Windows, Linux, Nintendo Wii, Sony PlayStation 3 e Microsoft Xbox 360

Tabela 3.2.2.2 – Tabela de comparação de alguns motores físicos

3.2.3 Motores de Jogos

Como enunciado no início deste capítulo, uma das abordagens possíveis na implementação de um simulador e visualizador de uma cadeira de rodas inteligente é a utilização de motores de jogos como base para esta implementação. Nesta secção são enunciadas as vantagens de utilização de motores de jogos e uma curta descrição dos mais conhecidos e adaptáveis ao desenvolvimento do projecto em questão, bem como a comparação dos mesmos.

3.2.3.1 Introdução

Um motor de jogos é uma aplicação de software com processamento gráfico em tempo-real que facilita a criação de jogos por abstracção de conceitos físicos e gráficos. Por outras palavras, o utilizador do motor de jogo não precisa essencialmente de compreender a arquitectura de mais baixo nível, referente a fórmulas matemáticas e físicas responsáveis pela visualização e detecção de colisão de objectos no espaço, entre outras funcionalidades.

Os motores de jogos podem ser divididos em vertentes de foco como, por exemplo, o Stratagus que é direccionado para a implementação de jogos de estratégia. Outros motores de jogos, como o Aleph One, o Cube, o Cube 2 e o ioquake3 já são direccionados para o desenvolvimento de jogos de atirador na primeira pessoa.

3.2.3.2 Caracterização de alguns motores

Nas próximas subsecções são caracterizados motores de jogos mais direccionados ou adaptáveis ao projecto em questão.

3.2.3.2.1 Unreal Engine 3

O motor de jogos Unreal Engine 3 é um motor de jogos com alta fidelidade gráfica e física desenvolvido pela empresa "Epic Games, Inc". A sua primeira utilização ficou demonstrada pelo jogo "Unreal" lançado em 1998. Desde essa altura, o motor de jogos já foi usado como base para diferentes jogos e, igualmente, como base para a construção de simuladores e ferramentas de design, abrangendo muitas áreas de investigação. [64]

O código fonte deste motor de jogos foi escrito recorrendo à linguagem de programação C++, embora seja apenas possível visualizá-lo e alterá-lo se, se comprar o jogo "Unreal Tournament 3", que disponibiliza o motor de jogos Unreal 3 e o editor referente à época de lançamento do jogo. [63]

Por outro lado, não é necessário aceder ao código do motor de jogos para se utilizar a gama completa de funcionalidades que disponibiliza. Apenas é necessário descarregar um pacote de desenvolvimento denominado UDK (Unreal Development Kit). Este pacote incluí o motor de jogos Unreal Engine 3, o editor que é constantemente actualizado, entre outras funcionalidades recentes, não disponíveis aquando a compra do jogo. [62] [65]

Em suma, o pacote disponível aquando compra do jogo tem licenças restritivas, ao revés do pacote denominado UDK que é uma versão gratuita se para desenvolvimento com fins não comerciais, disponível para o público geral, actualizado numa base mensal.

Neste momento, o editor de Unreal do UDK tem já funcionalidades de importação que facilitam esta tarefa, bem como outras inúmeras funcionalidades ou melhoradas ou novas em comparação com o editor disponibilizado pela compra do jogo.

Actualmente o motor de jogos Unreal Engine 3 é compatível com as seguintes plataformas: Windows, Linux, PlayStation 3, Mac OS X, iOS, PlayStation Vita, Wii U, Xbox, Xbox 360.

Para uma boa fidelidade física de simulação, este motor implementa o motor de física PhysX desenvolvido por Ageia e adquirido pela Nvidia. O motor PhysX é gratuito tanto para desenvolvimento com fins comerciais ou não comerciais e foi devidamente estudado na secção anterior de caracterização de alguns motores físicos. [46]

O motor de jogos utiliza uma linguagem de script de uso exclusivo no desenvolvimento de jogos ou simulações do próprio, denominada unreal script.

O que distingue este motor de jogos dos restantes é a disponibilização de um editor que inclui vários sub-editores, ora para construção de mundos virtuais, propriedades físicas, edição de materiais (e texturas), definição de animação, luminosidade, ligação de ficheiros de unreal script, etc.

Para saber mais informação sobre as funcionalidades deste motor de jogos basta consultar as páginas Web disponíveis em [61] [62] [65].

3.2.3.2.2 Blender Game Engine

O Blender é um programa gratuito com código fonte aberto para criação de conteúdo 3D, disponível para os maiores sistemas operativos segundo a licença geral pública GNU (General

Public License). O motor de jogos Blender Game Engine é um dos componentes do Blender e foi programado em C++. Este motor foi desenhado e implementado de forma a ser o mais independente possível do Blender e inclui suporte de funcionalidades como som em OpenAL 3D e configuração de propriedades em Python. [18] [39] [17]

As funcionalidades principais do motor de jogos são:

- Uso de um sistema de blocos gráficos, sendo cada um uma combinação de sensores, controladores e actuadores; Proporcionando o controlo de movimento e visualização de objectos;
- Suporte de áudio;
- Uso de um sistema para integração de sombras e das características físicas de corpo flexível;
- Uso de OpenGL para comunicar com o hardware gráfico;
- Compatível com as seguintes plataformas: Windows 2000, XP, Vista; Mac OS X (PPC and Intel); Linux (i386); Linux (PPC); FreeBSD 5.4 (i386); SGI Irix 6.5; Sun Solaris 2.8 (sparc).

Para saber mais sobre as funcionalidades deste motor de jogos basta consultar a página Web em [17].

3.2.3.2.3 ClanLib

O motor de jogos ClanLib, cujo desenvolvimento se iniciou em 1999, é um motor de jogos multiplataforma em C++ que suporta Microsoft Windows, Linux e Mac OS X. A utilização do ClanLib está restringida à licença BSD adaptada ao mesmo. [38] [12]

Este motor baseia-se em OpenGL na sua componente gráfica, tem duas bibliotecas disponíveis para tratamento de som (Vorbis e MikMod) e tem classes para tratamento de detecção de colisão, GUIs, XML, comportamento em rede, entre outras.

O motor tem como objectivo providenciar as características necessárias à criação de jogos e simulação de objectos e seu comportamento. A um nível baixo está a configuração de propriedades de visualização, som, dados de entrada, rede e ficheiros. Estas configurações estão organizadas de tal forma que permitem uma programação orientada a objectos. A um nível superior está a camada de construção e desenvolvimento do jogo pretendido.

As principais funcionalidades do motor de jogos em questão são:

- Suporte de Som (wav, formatos tracker e ogg-vorbis);
- Suporte de dispositivos de entrada de dados como o teclado, o rato e o joystick;
- Suporte de rede;
- Suporte de imagens nos formatos jpeg, png, pcx e tga;
- Suporte de fontes de texto;
- Detecção de colisão;
- Suporte de animações;
- Suporte de OpenGL 3, OpenGL 1 e SSE;
- Base de dados que suporta SQLite;
- Plataforma GUI passível de ser personalizada usando CSS.

Para mais informação acerca das funcionalidades e vantagens da utilização do ClanLib no desenvolvimento e construção de jogos, consultar o site oficial do motor de jogos em [38].

3.2.3.2.4 Exult

O motor de jogos Exult consiste num programa para desenvolvimento de jogos distribuído segundo a licença publica geral GNU (GPL General Public License) e consiste numa re-implementação do motor de jogos Ultima VII com alguns melhoramentos. [16] [15]

Este motor foi desenvolvido recorrendo à linguagem de programação C++, usa a biblioteca gráfica SDL e é multiplataforma. As plataformas suportadas são: Mac OS/X, Microsoft Windows, Sharp Zaurus, Microsoft Windows Mobile e BeOs. Não oficialmente existe a possibilidade de ligação a GP2X, Xbox, Symbian OS e PSP.

Para mais informação acerca deste motor de jogos, consultar a documentação do Exult existente no seu site oficial em [16].

3.2.3.2.5 HPL Engine

O HPL Engine é um motor de jogos a três dimensões criado pela empresa Frictional Games, desenvolvido recorrendo à linguagem de programação C++ e compatível com as bibliotecas OpenGL, OpenAL e Newton Game Dynamics. A última garante interacção de objectos com grande fidelidade física. O motor em questão é suportado nas plataformas Windows, Linux e Mac OS/X. [24] [23]

Este motor tem duas reencarnações: HPL Engine 1 e HPL Engine 2. O primeiro lançamento deste motor permitia o desenho de jogos a duas dimensões e continha uma camada para o desenho de jogos a três dimensões. O segundo lançamento consistiu numa melhoria de processos de desenho de objectos, de desenho de sombras e de outros efeitos visuais.

A licença actualmente adoptada pelo motor de jogos é de código proprietário, licença publica geral GNU (General Public License) para o HPL Engine 1 e licença proprietária para o HPL Engine 2.

3.2.3.2.6 Irrlicht Engine

O Irrlicht Engine é um motor de jogos de código aberto com grande performance em tempo real a três dimensões desenvolvido recorrendo à linguagem de programação C++, embora também seja possível recorrer a linguagens como .NET, Java, Perl, Ruby, FreeBasic, Python, Lua, Delphi, C++ Builder e Game Maker. [26] [27]

O desenvolvimento deste motor de jogos foi inicializado em 2003 por Nikolaus Gebhardt e em 2006 a equipa de desenvolvimento já contava com dez membros.

As plataformas suportadas por este motor são Windows, Mac OS/X, Linux, Windows CE e ligações a Xbox, PlayStation Portable, SymbianOS e iPhone.

O motor em questão tira partido das funcionalidades de D3D, OpenGL, DirectX 8 e 9 e OpenGL ES, bem como do seu software para proporcionar funcionalidades de desenho e construção de jogos.

Este motor de jogos suporta inúmeros formatos de ficheiros: ficheiros de 3DStudio Max, modelos de Quake 2 (MD2), objectos de Maya (.obj), mapas de Quake 3 (.bsp), objectos de Milkshape3D e ficheiros de DirectX (.x). Outros formatos de objectos 3D são suportados a partir de plugins externos.

Em suma, o motor de jogos Irrlicht tem muita documentação disponível e permite a criação de aplicações completas a duas e três dimensões, sejam estas jogos ou visualizações científicas, através de uma interface de fácil uso.

As suas funcionalidades incluem representações visuais como sombras dinâmicas, sistemas de partículas, animação de personagens, tecnologias de espaços interiores e exteriores e detecção de colisão.

A utilização do motor de jogos e sua distribuição estão autorizados segundo a licença zlib licence. Todas as suas funcionalidades estão extensivamente descritas no site oficial em [26].

3.2.3.3 Comparação dos motores

Nesta secção é apresentada uma tabela com as principais características de todos os motores de jogos sucintamente descritos na secção acima.

Motor de Jogos	Desenvolvedores	Linguagem de Programação	Plataformas compatíveis	Licença de código
Unreal Engine 3	Epic Games, Inc	C++ (unreal script para programação de jogos e seus modelos)	Windows, Linux, PlayStation 3, Mac OS X, iOS, PlayStation Vita, Wii U, Xbox, Xbox 360	Aberto se fins não comerciais / Proprietário se fins comerciais
Blender Game Engine	The Blender Foundation	C, C++ e Python	Windows 2000, XP, Vista; Mac OS X (PPC and Intel); Linux (i386); Linux (PPC); FreeBSD 5.4 (i386); SGI Irix 6.5; Sun Solaris 2.8 (sparc)	GNU GPL (versão 2 ou posterior)
ClanLib Engine	ClanLib Team	C++	Microsoft Windows, Linux e Mac OS X	BSD
Exult Engine	Exult Team		Mac OS/X, Microsoft Windows, Sharp Zaurus, Microsoft Windows Mobile e BeOs	GNU GPL
HPL Engine 1/2	Frictinal Games	C++	Windows, Mac OS/X e Linux	GNU GPL / Proprietário
Irrlicht Engine	Nikolaus Gebhardt inicialmente, dez membros posteriormente	C++ (.NET, Java, Perl, Ruby, FreeBasic, Python, Lua, Delphi, C++ Builder e Game Maker)	Windows, Mac OS/X, Linux, Windows CE e ligações a Xbox, PlayStation Portable, SymbianOS e iPhone	zlib

Tabela 3.2.3 – Tabela de comparação de alguns motores de jogos

3.2.4 Simuladores Robóticos

Como enunciado no início deste capítulo, uma das abordagens possíveis na implementação de um simulador e visualizador de uma cadeira de rodas inteligente é a utilização de simuladores robóticos como base para esta implementação. Nestas subsecções são explicados os conceitos base necessários à compreensão de simuladores robóticos, enunciadas as suas vantagens de utilização e uma curta descrição dos mais conhecidos e adaptáveis ao tema, bem como a comparação dos mesmos.

3.2.4.1 Introdução

Um simulador robótico é uma plataforma de desenvolvimento de software para simulação do comportamento robótico num ambiente virtual, permitindo modelação de robôs.

A modelação de um robô depende do objectivo para que este foi desenvolvido e da meta que se pretende alcançar.

Esta modelação só pode ser realizada após uma análise pormenorizada do comportamento esperado do robô num dado conjunto de situações.

Nas próximas subsecções deste capítulo serão abordados os tipos de robô, a formulação do comportamento robótico e ambiente de simulação robótica.

3.2.4.2 Aplicação de Robôs no Mundo Real

Para melhor compreensão deste documento é importante apresentar a definição de robô. Um robô é uma máquina que tem capacidade de interagir com objectos físicos e de ser electronicamente programável para executar uma determinada tarefa. Tendo em conta o objectivo da aplicação a desenvolver, o robô é uma cadeira de rodas, mas numa primeira aproximação seria considerado um robô virtual, ou seja, a representação virtual da cadeira de rodas e do seu ambiente de interacção virtual.

Para se proceder à classificação de robôs é usual a sua divisão em duas grandes classes: robôs industriais e robôs de serviços.

Um robô industrial, segundo a definição ISO 8373, é uma máquina reprogramável, passível de ser controlável de forma automática. Esta é multi-propósito e programável em três ou mais eixos. Esta máquina pode estar fixa num local ou ser móvel e é usada em aplicações industriais automatizadas.

Um robô de serviços, segundo a definição proposta pela IFR, é um robô que opera semi ou autonomamente, para prestar serviços úteis para o bem-estar dos humanos ou equipamento, excluindo operações de manufacturação.

No esquema abaixo estão representados alguns tipos e exemplos de aplicações robóticas de robôs industriais (figura 3.2.4.1).

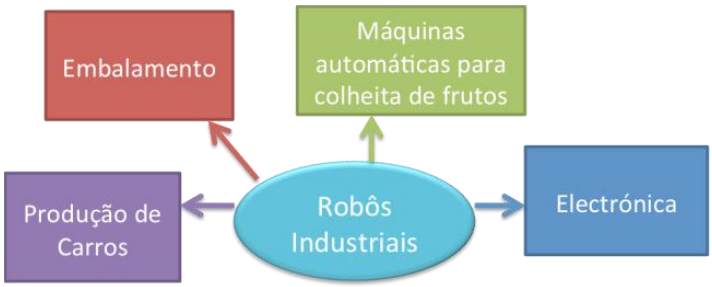


Figura 3.2.4.1 – Esquema de aplicações robóticas que se enquadram em robôs industriais

No esquema abaixo estão representados alguns tipos e exemplos de aplicações robóticas de robôs de serviços (figura 3.2.4.2).

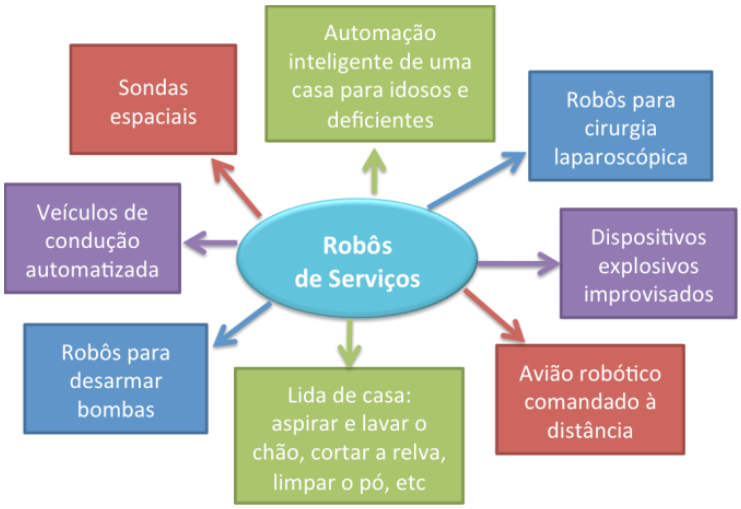


Figura 3.2.4.2 - Esquema de aplicações robóticas que se enquadram em robôs de serviços

3.2.4.3 Formulação do Comportamento Robótico

O comportamento robótico pode ser parte integrante de onze classes comportamentais:

- Exploração/Direccional;
- Orientado a Objectivos/Metas;
- Adversidade/Protecção;
- Percorrendo um Caminho;
- Postural;
- Social/Cooperativo;
- Tele-operacional;
- Perceptual;
- Locomoção;
- Manipulativo;
- Aprendizagem.

Este comportamento pode ser representado por diagramas de estímulo-resposta, notação funcional e/ou diagramas de aceitação de estados finitos.

Os diagramas de estímulo-resposta são diagramas que representam as acções de resposta de um robô ao receber um ou mais estímulos do ambiente em que se insere. A figura 3.2.4.3 é um exemplo de diagrama estímulo-resposta. Estes estímulos podem ser qualquer tipo de leitura de dispositivos integrados no robô.

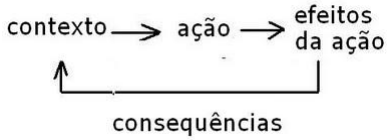


Figura 3.2.4.3 - Exemplo de diagrama estímulo-resposta

A notação funcional é uma outra representação do comportamento robótico, deverá ser escrita numa linguagem natural ou pseudo-código desde que se mantenha intuitiva.

Se o objectivo for a passagem de conhecimento comportamental apenas a programadores, pode ser escrita, por exemplo, em linguagem lógica ou outra linguagem perceptível e compreensível pelos mesmos.

Ao representar o comportamento robótico em diagramas de aceitação de estados finitos, a arquitectura comportamental é lida por qualquer um mais facilmente e sugere uma arquitectura de implementação.

Estes diagramas têm sempre um estado inicial, a partir do qual, dependendo do estímulo recebido, é gerada uma determinada acção, mudando o sistema para um outro estado e assim sucessivamente. Na figura 3.2.4.4 está um exemplo de diagrama de estados.

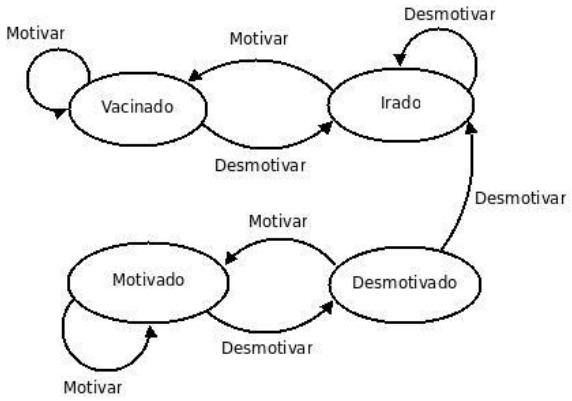


Figura 3.2.4.4 – Exemplo de diagrama de estados sobre os estados de um professor

A formulação do comportamento robótico é importante na simulação do robô cadeira de rodas virtual.

3.2.4.4 Ambientes de Simulação

Os simuladores robóticos são usados para criar aplicações embebidas para robôs sem dependerem fisicamente da máquina física (robô do mundo real).

Em seguida serão classificados por comparação alguns ambientes de simulação.

Dos vários ambientes de simulação robótica vão ser resumidamente caracterizados os mais conhecidos.

3.2.4.4.1 Classificação de alguns ambientes de simulação

Para proceder à classificação dos simuladores robóticos seleccionados vão ser considerados três graus subjectivos: alto, médio e baixo. Os critérios considerados são a fidelidade física, fidelidade funcional, facilidade de desenvolvimento e custo.

Neste contexto, fidelidade física pode ser descrita como o grau de extensão de propriedades que o ambiente virtual emula do mundo real.

Para este critério são analisadas a fidelidade visual e auditiva e a capacidade de resolução de imagem a nível de texturas, sombras, luminosidade e reflexão. Por exemplo, quando um veículo se afasta ou aproxima, a intensidade do som produzido pelo mesmo deve variar num simulador com alta fidelidade física.

O critério de fidelidade funcional é o grau de semelhança entre o comportamento físico dos objectos e/ou robôs no ambiente virtual e no mundo real. Este é avaliado tendo em conta a capacidade física de simulação, como por exemplo, a capacidade de modelar a aceleração de cada roda de um veículo de forma independente.

Segundo o artigo [13], a facilidade de desenvolvimento é definido obtendo a resposta a várias questões, que se enumeram:

- Quão facilmente pode um ambiente ser criado para conduzir um exercício de treino embutido no simulador?
- Quão facilmente pode o simulador ser modificado para simular um novo equipamento?
- Há documentação acessível com apoio do autor?
- Que linguagens de programação podem ser usadas para modificar o simulador?

Um exemplo de um simulador com grau alto atribuído a este critério seria um simulador que fornece documentação de desenvolvimento, apoia a importação de objectos modelados em três dimensões fora do ambiente e que pode ser programável em várias linguagens de programação.

As principais características a ter em conta para a atribuição de grau ao critério custo são o tempo de instalação e execução bem como o custo monetário das ferramentas de software e o custo final para o utilizador.

Graus de classificação do critério custo:

- Grau baixo – simuladores gratuitos com instalador incluído;
- Grau médio – simuladores gratuitos e difíceis de instalar ou simuladores com custo modesto e fáceis de instalar;
- Grau alto – simuladores de custo elevado, neste caso não se tem em conta a facilidade/dificuldade de instalação.

Os critérios declarados acima são usados para classificar e comparar o conjunto de simuladores robóticos identificados na tabela abaixo.

Simulador	Fidelidade Física	Fidelidade Funcional	Facilidade de Desenvolvimento	Custo	Tipo de licença
USARSim	Alto	Médio	Médio	Médio	Aberto
Webots	Médio	Médio	Médio	Médio	Proprietário
Simbad	Médio	Baixo	Médio	Baixo	Aberto
Gazebo	Médio	Baixo	Alto	Médio	Aberto
MS Robotics Studio	Alto	Alto	Médio	Baixo	Proprietário
SimRobot	Médio	Baixo	Médio	Baixo	Aberto
SubSim	Médio	Baixo	Médio	Médio	Aberto

Tabela 3.2.4 – Tabela de comparação de ambientes de simulação robótica

3.2.4.4.2 Caracterização de alguns ambientes de simulação robótica comerciais

Os ambientes de simulação e modelação robótica comerciais mais conhecidos são:

- AnyKode Marilou – Especificado e preparado para robôs móveis, humanóides e braços articulados;
- Webots – Tem um propósito educacional e está preparado para detecção de colisões entre objectos e para a simulação da dinâmica rígida de corpo;

- Microsoft Robotics Developer Studio (HRDS) – Tem como alvo a população académica, amadores e desenvolvedores comerciais e é um ambiente baseado em Windows que permite fácil acesso a sensores robóticos e actuadores;
- Actin – Permite cooperação entre múltiplos manipuladores de robôs e pode ser usado recorrendo a comunicação em rede (TCP/IP e/ou UDP/IP);
- Workspace 5 – Ambiente baseado em Windows que permite a criação, manipulação e modificação de imagens em 3DCad e definição e utilização de múltiplos dispositivos de comunicação.

Estes simuladores robóticos não vão ser usados no desenvolvimento da aplicação porque a ideia é não haver custos associados à compra/manutenção de software.

3.2.4.4.3 Caracterização de alguns ambientes de simulação robótica não comerciais

Os ambientes de simulação robótica abertos são caracterizados a seguir por não terem qualquer custo associado à sua compra/manutenção, sendo por essa razão considerada a sua selecção.

Dos vários existentes só foram caracterizados os mais conhecidos e mais adaptáveis ao projecto em questão.

3.2.4.4.3.1 SubSim

O simulador robótico SubSim é de entre os seleccionados o menos adaptável ao projecto, no entanto as suas características fundamentais são:

- Sistema de simulação de veículos submarinos autónomos proposto em 2007 para a competição de simulação ICAUV;
- Interface de baixo nível da aplicação é programável em C e C++, a de alto nível é RoBIOS;
- Disponível gratuitamente na Universidade "University of Western Austrália" no Laboratório de Robôs Móveis ("Mobile Robot Lab");
- Desenvolvido em 2004 por Boeing, Koestler e Petitt; em 2005 por Ruehl e Bielohlawek; em 2006 por Haas e Boeing; administrado por Braunl. [54]

3.2.4.4.3.2 SimRobot

SimRobot é um simulador robótico capaz de simular o comportamento de robôs arbitrários definidos pelo utilizador no espaço tridimensional. Este foi desenvolvido por Uwe Siems, Christoph Herwig, Thomas Röfer, Jan Kuhlmann e pode ser executado em vários sistemas operativos, entre os quais, Unix, Windows 3.x/95/98/ME/NT4/2000/XP, OS/2. Este simulador tem grande flexibilidade de implementação, o que permite a construção de diferentes modelos de robôs de uma forma fácil. Esta flexibilidade está presente no simulador devido à preocupação do mesmo em ter uma grande diversidade de corpos genéricos, sensores e actuadores já implementados, disponíveis para facilitar a modelação dos robôs. Como o principal objectivo deste simulador é garantir alta usabilidade, a arquitectura deste incluí

vários mecanismos de visualização, manipulação directa de actuadores e interacção com o mundo virtual. [1] [32] [31]

Algumas características importantes na implementação de um robô no SimRobot são:

- A especificação da cena (mundo virtual e caracterização do robô) é descrita usando a linguagem de anotação XML (RoSiML – Robot Simulation Markup Language) que é carregada em tempo-real;
- O simulador foi desenvolvido usando a linguagem de programação C++;
- Para simular a dinâmica de corpo rígido usa o motor de física ODE;
- A visualização da cena é baseada em OpenGL;
- Implementa quatro classes principais de sensores: câmara, sensor de distância, sensor de toque e estado do actuador;
- Como prova da sua capacidade de modelar sistemas com rodas existe a simulação da cadeira de rodas autónoma "Rolland" baseada num modelo 3D providenciado pelo seu fabricante Meyra. [31]

3.2.4.4.3.3 Simbad

O Simbad é um ambiente de simulação robótica autónoma a três dimensões, desenvolvido em java e direccionado para a educação e investigação científica. [53] [25]

O seu intuito é proporcionar uma base simples para o estudo e teste de inteligência artificial situada, aprendizagem máquina e algoritmos de inteligência artificial no contexto de robôs e agentes autónomos.

Este simulador tem embebido dois pacotes que podem ser utilizados independentemente cuja área de inteligência artificial alvo é a robótica evolucionária. O primeiro é uma biblioteca de redes neuronais (PicoNode) e o segundo é uma biblioteca de algoritmos evolucionários (PicoEvo) que se foca em algoritmos genéticos, estratégias evolucionárias e programação genética.

As principais características do simulador Simbad são:

- Desenvolvido na linguagem de programação java, os requisitos para a sua instalação e utilização são o Java 1.4.2 (ou versão mais recente) e Sun Java 3d 1.3.1 (ou versão mais recente);
- Multiplataforma (Windows, Mac Os X, Linux, IRIX, AIX);
- Licença GPL ("GNU General Public Licence");
- Os administradores do projecto, também programadores, são Louis Hugues e Nicolas Bredeche.

As principais funcionalidades do simulador são:

- Simulação simples ou multi-robô;
- Sensores visuais: câmaras de cor mono-escópica;
- Sensores de distância: sensores sonar e infra-vermelhos;
- Sensores de contacto;
- Simulação local ou em rede;
- Interface do utilizador extensível;
- Motor físico simples (construído no interior do simulador);
- Uso da linguagem Python com Jython.

3.2.4.4.3.4 Gazebo

O Gazebo é um simulador multi-robô a três dimensões para ambientes espaço aberto, ou seja, direccionado para mundos virtuais fora de portas. Este simulador tem a capacidade de simular populações de robôs, sensores e objectos, implementa sensores realísticos e permite uma interacção física entre objectos de forma plausível e precisa. [48] [28]

As principais características do simulador Gazebo são:

- Simulação de sensores robóticos como os sensores sonar, laser de varrimento para descoberta de distância, GPS e IMU, e câmaras tanto monocular como estéreo;
- Modelos comuns de robôs como Pioneer2DX, Pioneer2AT e SegwayRMP;
- Simulação física realística da dinâmica de corpo rígido tirando partido do motor físico ODE, ou seja, os robôs podem empurrar e pegar em objectos do mundo virtual, sendo essa interacção fisicamente plausível;
- Os modelos geométricos simples podem ser graficamente melhorados através da inclusão de imagens modeladas em programas 3D externos, no entanto a visualização de robôs/objectos/mundo virtual é baseada em OpenGL;
- Os programas podem interagir directamente com o simulador, sem interagirem com o visualizador (Player);
- Licença GNU GPL ("GNU General Public Licence");
- Multiplataforma (Windows, Linux e Mac Os X).

3.2.4.4.3.5 USARSim

O USARSim (simulador de pesquisa e salvamento urbano) é um simulador robótico flexível a três dimensões e começou a ser desenvolvido no Outono de 2002. [60] [2] [57] [45]

Este simulador é capaz de reproduzir a aparência e dinâmica de robôs genéricos e objectos num dado ambiente virtual, facilita igualmente a sua modelação e animação.

Este simulador é baseado no motor de jogos Unreal Engine 3, o que facilita a boa qualidade gráfica, de simulação física e ligação em rede, tal como proporciona grande versatilidade da linguagem de programação e providencia um poderoso editor visual. Em relação à sua arquitectura implementa o motor físico Karma.

Esta plataforma de simulação robótica tem limitações como o número de articulações de um robô, o suporte limitado de cenários multi-robô e tratamento de colisões aproximado.

Por outro lado, este foi o simulador escolhido para a competição de simulação RoboCup Rescue por ser um simulador direccionado para robôs com rodas e concebido para o tópico pesquisa e salvamento.

As principais vantagens de simulação do USARSim são:

- O código desenvolvido para controlar robôs dentro do simulador pode ser transportado "tal e qual" para o software de robôs reais;
- O simulador é uma ferramenta para teste inicial e integração tardia, ou seja, pode ser usado para verificar o impacto das decisões de desenho antes de ser implementado fisicamente num sistema real;
- Incluí modelos de sensores, robôs e actuadores usados correntemente por investigadores da área de robótica.

Em resumo, as características que proporcionam o êxito deste simulador são:

- O modelo de distribuição permitiu que a comunidade de investigadores que usam o presente simulador contribuísse com enumeras extensões;

- Investigadores da área de robótica podem concentrar-se em aspectos directamente relacionados com a robótica, esquecendo a preocupação com a fidelidade gráfica entre outras funcionalidades proporcionadas pelo simulador;
- A disponibilidade de interfaces compatíveis com o conjunto de controladores mais usados permite a migração de código do simulador para o robô real e vice-versa sem custos;
- O facto do simulador se unir à competição Robocup forçando um desenvolvimento constante mapeado pelas necessidades robóticas reais;
- O esforço de validação convenceu as pessoas que o USARSim é uma ferramenta vantajosa para desenvolver código que pode ser eventualmente executado em sistemas robóticos reais;
- O contínuo desenvolvimento de novas funcionalidades e/ou melhoramento do próprio editor do motor de jogos a que o USARSim se adapta.

Capítulo 4

Conceito e Implementação do Simulador e Visualizador

O objectivo principal deste projecto foi implementar um simulador e visualizador de cadeira de rodas inteligente. Ao longo do processo de implementação da aplicação pretendida, seleccionou-se como plataforma base de desenvolvimento o simulador robótico USARSim, esta decisão está devidamente justificada na secção seguinte (Metodologia geral seleccionada).

A aplicação foi construída responsabilizando-se pela configuração do mapa (mundo virtual), configuração da cadeira de rodas e seus sensores, compilação dos ficheiros de configuração do USARSim e execução dos comandos de condução da cadeira de rodas virtual.

As primeiras duas secções deste capítulo pretendem explicitar a metodologia seguida e as principais decisões tecnológicas.

Na terceira e quarta secção são apresentadas, respectivamente, a arquitectura modular da aplicação desenvolvida e a interface gráfica proporcionada ao utilizador da mesma.

Numa quinta fase são identificadas as particularidades gerais do USARSim em conjunto com o Unreal Engine 3, especificando em pormenor a configuração de robôs, a configuração de sensores e os comandos USARSim utilizados.

Na secção seis pretende-se descrever a modelação do ambiente de simulação que foi dividido em três partes: modelação do mapa do mundo virtual, criado para teste da aplicação; modelação da cadeira de rodas virtual, criada para teste do comportamento robótico; e plataformas de desenvolvimento, onde se refere as principais plataformas usadas aquando das tarefas de modelação.

Na secção seguinte do capítulo é descrito o agente inteligente de controlo do robô e as funcionalidades até ao momento implementadas para o mesmo. A secção final refere a possível interligação da aplicação ao projecto “IntellWheels”, objectivo que não foi totalmente implementado. Nesta secção é apresentada uma sugestão de forma de interligação.

No final do actual capítulo, o leitor deverá ter uma boa percepção de como foi implementada a aplicação desenvolvida e porquê, tal como ter noção das decisões efectuadas, tecnológicas e ao nível da implementação.

4.1 Metodologia geral seleccionada

Ao longo do documento foram apresentadas quatro abordagens possíveis para a realização dos objectivos essenciais do projecto, a desenvolver na época de dissertação de mestrado

integrado. Entre os motores gráficos, físicos, de jogos e simuladores robóticos optei por simuladores robóticos abertos por alguns já terem capacidades equivalentes a motores de jogos (que incluem normalmente motores físicos e gráficos) mas permitirem, em adição, modelação do comportamento robótico de uma forma fácil e semi-automática.

Após optar por um simulador robótico com tipo de licença aberta, estudou-se fundamentalmente dois simuladores robóticos: USARSim e Gazebo.

Os simuladores SimRobot, SubSim e Simbad foram descartados do conjunto de possíveis soluções por serem, em comparação, os menos aptos para o projecto em questão. Estes três são simuladores robóticos mais direccionados para a fidelidade física onde obtêm uma classificação média, o que os distingue dos outros nesta característica. No entanto, a sua fidelidade gráfica obtém uma classificação baixa por ser conseguida através da biblioteca gráfica OpenGL em vez de implementarem um motor gráfico com maior fidelidade.

O simulador robótico Gazebo é um ambiente de simulação em rede a três dimensões, tem um sensor do tipo câmara e permite objectos complexos dentro do ambiente virtual. Estas características são bastante vantajosas para o projecto em questão, no entanto, o simulador garante baixa fidelidade gráfica pelos mesmos motivos dos simuladores referidos acima. Como tal, a preferência na utilização de um dado simulador robótico, tendo em conta este projecto, recai sobre o USARSim.

As razões que justificam a escolha do simulador USARSim, em detrimento de outros simuladores considerados acima, são:

- Ter um suporte avançado em robôs com rodas, permitindo a configuração das caixas de rotação das rodas, cujo movimento é caracterizado de forma independente;
- Permitir a importação de robôs/objectos modelados externamente, o que facilita o objectivo de importação de imagens três dimensões e o objectivo de automatizar a modelação de diferentes modelos de cadeira de rodas;
- Ser possível programar os robôs ou controlá-los em rede, o que é indispensável na implementação de realidade aumentada [42].

O simulador robótico USARSim tem actualmente várias versões disponíveis, as versões utilizadas aquando implementação e teste correspondem as [74] [75] [77].

4.2 Outras decisões tecnológicas

A linguagem de programação escolhida foi o C++ devido à preferência e familiarização com a mesma. Para compilar e executar o programa foi utilizada a plataforma Eclipse em conjunto com o *plugin* de integração de Qt no Eclipse, concordante com qualquer versão de Qt a partir da versão 4.1.0, descarregada de [69].

O presente projecto pretende ser executado em rede, ou seja, deveria ser possível comunicar com o simulador robótico escolhido em rede, como tal, para facilitar a implementação e permitir que a aplicação fosse multi-plataforma, foi considerada inicialmente a utilização da biblioteca Asio do Boost. Esta é uma biblioteca multi-plataforma de C++ para rede e programação de baixo nível de entrada/saída de dados que fornece aos programadores um modelo consistente assíncrono que usa um C++ moderno.

Durante a implementação, após a correcta configuração do eclipse e programação do módulo de comunicação, foram avaliadas as opções de desenvolvimento do módulo de interface gráfica das configurações gerais de simulação.

Para facilitar a implementação foi considerado o uso das duas bibliotecas para desenvolvimento de interfaces gráficas em C++ mais conhecidas: wxWidgets e Qt.

A biblioteca wxWidgets é uma biblioteca para criação de aplicações para as plataformas Windows, OS X, Linux e Unix e para algumas plataformas moveis como Windows Mobile, iPhone SDK e GTK+ incorporado. Esta biblioteca proporciona um aspecto equivalente à plataforma para a qual se desenvolve a aplicação.

Qt foi a biblioteca seleccionada por ter todas as características essenciais requeridas para o desenvolvimento de janelas e menus, já conter um pacote de instalação preparado especialmente para a integração da biblioteca no Eclipse e disponibilizar documentação extensa e bem organizada.

Após a escolha da utilização de Qt, verificou-se que a mesma continha igualmente um módulo de rede de programação de alto nível, portanto a utilização da biblioteca Asio do Boost foi descartada. A nível de programação de funcionalidades em rede, o módulo de rede da biblioteca Qt permite a transferência de ficheiros por FTP (*File Transfer Protocol*), comunicação síncrona por TCP/IP entre outras funcionalidades não pertinentes para a aplicação em questão.

Para proceder à organização da documentação da aplicação desenvolvida foi utilizado o sistema de documentação Doxygen preparado para a extracção da estrutura e documentação do código fonte da aplicação. Na wiki do projecto está presente a documentação em HTML.

4.3 Arquitectura Modular

Como referido em capítulos anteriores, a necessidade desta aplicação surge no seio do projecto da plataforma 'IntellWheels'. Projecto este que já inclui ferramentas de interface com a cadeira de rodas física, construída para o efeito recorrendo ao menor custo possível de recursos base. Além desta fase inicial, foi apresentado em Fevereiro deste ano uma interface de comunicação do utilizador da cadeira de rodas com a mesma, que aceita várias categorias de comandos (comandos de voz, comandos accionados por movimentos da cabeça, etc.).

Para o referido projecto geral já foi desenvolvido um simulador a duas dimensões para tratamento de colisões e um visualizador a duas e três dimensões (este último recorrendo à tecnologia OpenGL). Esta proposta de projecto de dissertação surge na necessidade de existência de um simulador e visualizador a três dimensões com características físicas e gráficas fiéis à realidade.

Um dos objectivos chave desta proposta de tese é construir uma aplicação com arquitectura modular e/ou que seja facilmente adaptada aos restantes módulos do projecto global. Como tal, todo o simulador e visualizador 3D foi construído, considerando a existência prévia dos módulos responsáveis pelo comportamento da cadeira de rodas física (baixo nível – controlo dos motores), comunicação da leitura dos seus sensores e gestão de acções comportamentais (alto nível – prioridade de cada categoria de comandos; inteligência artificial).

A presente aplicação foi dividida nos sete módulos identificados no esquema 4.3.1, cada um especifico para um dado tipo de funcionalidades. O esquema referido enquadra as classes desenvolvidas nos módulos respectivos.

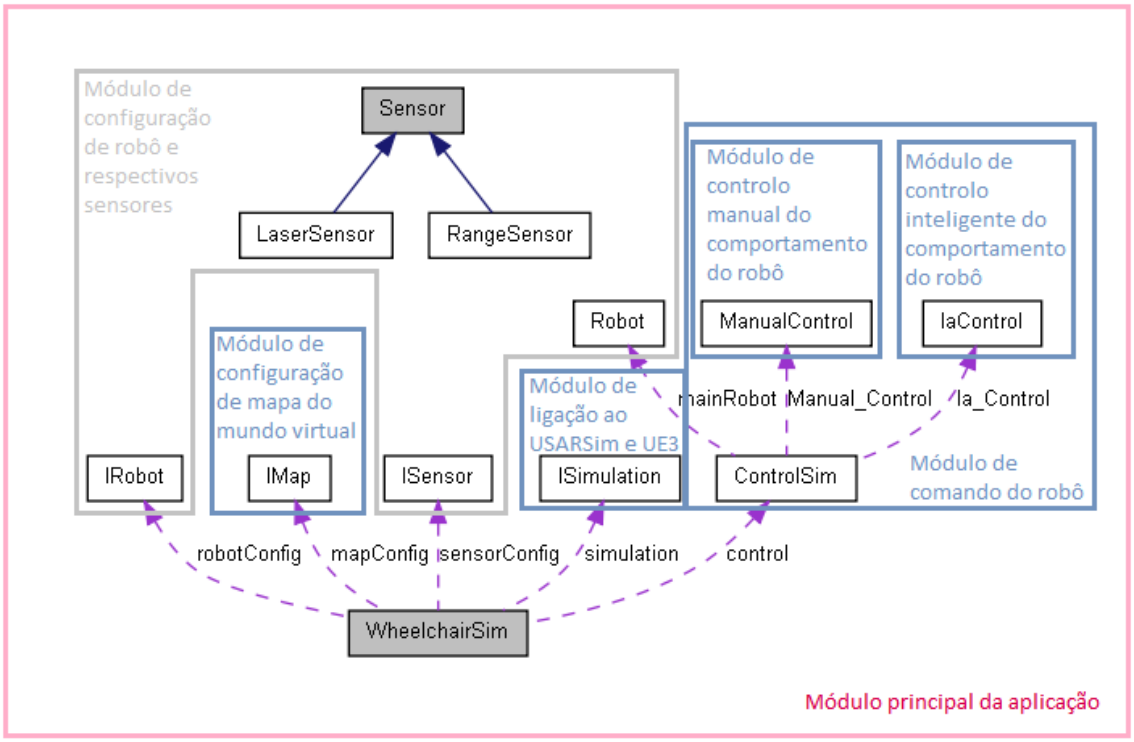


Figura 4.3.1 – Diagrama dos módulos da aplicação desenvolvida

No decorrer da presente secção é apresentada uma descrição sucinta de cada um dos sete módulos identificados no diagrama (Figura 4.3.1).

Módulo de configuração de mapa do mundo virtual – Corresponde à implementação da classe IMap que, além de promover uma interface amigável para a configuração do mapa a ser usado na simulação, permitindo o *browse* do ficheiro do mapa usando a API da plataforma Windows, está também responsável por escrever os ficheiros de configuração e execução do mesmo (caso ordenado pelo utilizador); O ficheiro do mapa pode estar no formato '.ut3', se perante a utilização do motor de jogos Unreal Engine 3 lançado aquando o lançamento do jogo, ou no formato '.udk' se perante a utilização do motor de jogos do UDK.

Módulo de configuração de robô e respectivos sensores – Corresponde à implementação das classes IRobot e ISensor responsáveis, respectivamente, pela interface gráfica de configuração do robô e sensores a ele associados. Este módulo inclui igualmente a inicialização do robô e suas características principais, bem como a inicialização dos sensores, guardando as configurações iniciais nas classes Robot e Sensor (LaserSensor e RangeSensor).

Módulo de ligação ao USARSim e UE3 – Correspondente à classe ISimulation, este módulo está responsável pela compilação do USARSim, inicialização da simulação (emitindo um sinal que é captado pelo IMap que por sua vez procede à execução do mapa), estabelecimento de ligação por TCP *socket* ao motor de jogos Unreal, supervisão desta ligação e envio e recepção de mensagens entre ambos.

Módulo de comando do robô – Este módulo inclui a visualização das características principais do robô, o pedido de posições iniciais possíveis para o robô, a inicialização do mesmo numa dada posição do mapa, o pedido de informação acerca da configuração e características geográficas do robô e sensores e a visualização dos valores lidos pelos sensores anteriormente colocados no robô. Este módulo corresponde à implementação da classe ControlSim que serve de intermediário entre as interfaces gráficas dos módulos de controlo manual e inteligente do comportamento do robô e os comandos de controlo enviados para o

módulo de ligação ao USARSim e UE3, procedendo igualmente ao *parse* e compreensão das respostas destes comandos de controlo.

Módulo de controlo manual do comportamento do robô - Este módulo permite controlar o robô previamente colocado no mapa através dos seguintes comandos pré-configurados: acender o foco de luz do robô, andar em frente, andar para trás, ir mais depressa, ir mais devagar, parar, recolocar o robô na orientação correcta (*flip*), virar à esquerda ou à direita segundo um ângulo especificado pelo utilizador em graus e condução a partir do equivalente às setas cima, baixo, esquerda e direita presentes na interface.

Módulo de controlo inteligente do comportamento do robô - Este módulo permite controlar o comportamento do robô de uma forma automatizada, e permite a ligação ao controlo inteligente da plataforma 'IntellWheels'.

Módulo principal da aplicação - Este módulo liga todos os restantes dando congruência à aplicação final e está responsável por permitir a simulação de múltiplos robôs e sua configuração, pela ligação e gestão das funcionalidades dos diferentes módulos e pela verificação dos requisitos base da aplicação, como a correcta instalação do motor de jogos do Unreal e do simulador robótico USARSim, entre outras.

Na Figura 4.3.2 estão representadas as classes omitidas do diagrama da arquitectura modular apresentado acima, ou seja as incluídas pela classe Robot.

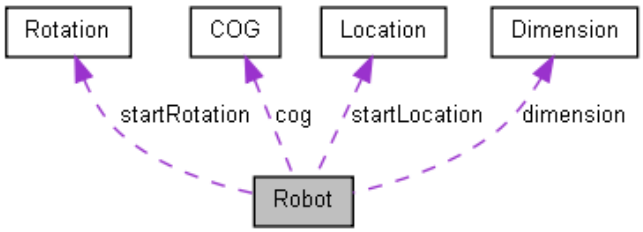


Figura 4.3.2 – Digrama de colaboração da classe Robot

Visto que a tecnologia principal utilizada para a implementação desta aplicação é Qt, cuja funcionalidade de maior potencial é a interligação de classes através de sinais, toda a aplicação foi desenvolvida recorrendo a esta vantagem programática. Como seria exaustiva a visualização de um único diagrama com todos os sinais emitidos e recebidos e funções directamente chamadas aquando emissão dos respectivos sinais, apenas é apresentada uma breve descrição da gestão de comunicação entre os diferentes módulos.

Inicialmente, a simulação é iniciada ao executar o mapa de teste identificado pelo utilizador, em seguida, o módulo de ligação ao USARSim e Unreal Engine estabelece a ligação, e o módulo de comando de robô fica activo. Este módulo permite executar uma série de funcionalidades como pedir posições iniciais, iniciar robô numa dada posição e controlá-lo.

Na figura 4.3.3 é possível visualizar as opções implementadas, as funções correspondentes pertencem à classe "ControlSim" (representada pela caixa do lado direito). A caixa mais à esquerda do esquema corresponde ao sinal enviado sempre que se efectua um pedido ao servidor de Unreal. Este sinal ('sendCommand(QString)') é por sua vez captado pela função de envio de pedido ('sendRequest(QString)') da classe ISimulation que, por sua vez, gere o envio de comandos e respectivas respostas. Não sendo da sua competência avaliar as respostas, mas sim, garantir o estabelecimento de ligação e a correcta troca de mensagens.

Na interface gráfica da classe ManualControl é possível conduzir o robô manualmente, após clique num qualquer botão é emitido um dos três sinais – *drive*, *lightAction* ou *flipAction* - que por sua vez são captados pela classe ControlSim.

Na interface gráfica da classe IaControl é possível conduzir o robô de forma automática, na semelhança com a classe anterior, cada modo automático emite um sinal identificativo da acção captado pela classe ControlSim que providencia essa acção.

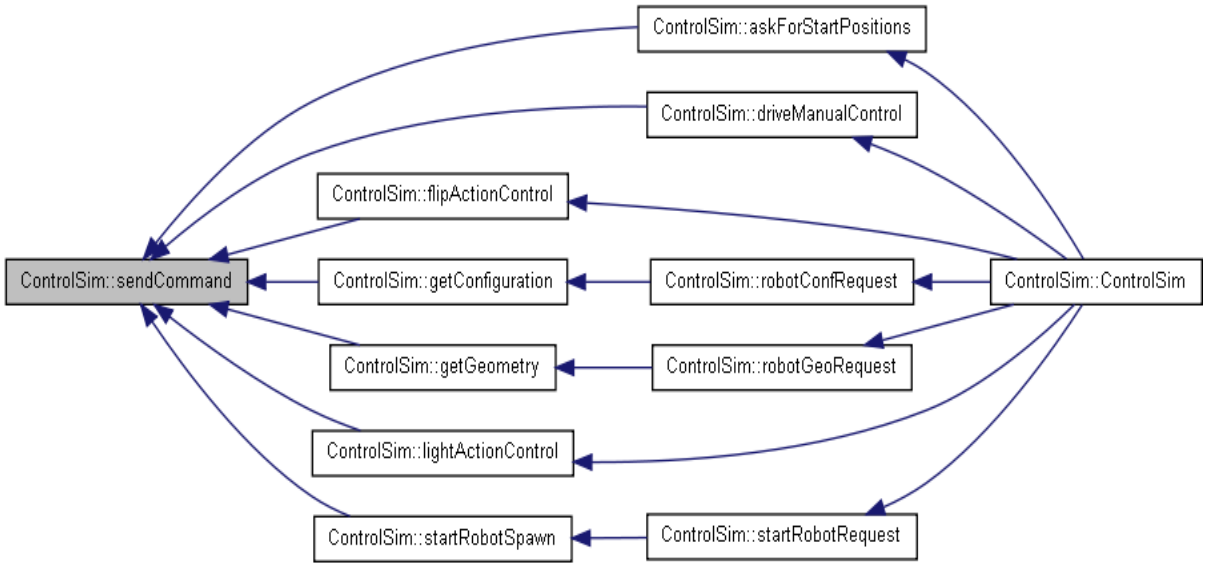


Figura 4.3.3 – Digrama do módulo de controlo do comportamento robótico

4.4 Interface Gráfica do Utilizador

Nesta secção são apresentadas as principais janelas de diálogo criadas usando a tecnologia Qt. Estas interfaces gráficas estão dispersas pelos módulos apresentados na secção anterior, Arquitectura Modular, e vão ser identificadas em conjunto com as funcionalidades proporcionadas pelos botões, entre outros objectos de interacção.

4.4.1 Janela 'Map'

Na figura 4.4.1.1 pode-se visualizar a interface gráfica a que o utilizador tem acesso para proceder à identificação do mapa a utilizar durante a simulação robótica, bem como adicionar, editar e remover robôs a usar durante a simulação. Esta interface permite igualmente iniciar a simulação.

Como visível na figura da página seguinte, a parte superior desta interface é composta por quatro botões, uma linha de edição de texto e uma *checkbox*. O botão 'Browse' abre uma janela do Windows correspondente à árvore de directórios e ficheiros do computador, onde se poderá seleccionar o ficheiro mapa. A localização inicial do gestor de ficheiros corresponde ao directório em que usualmente se guardam os mapas de extensão '.udk' quando se instala a recente adaptação ao USARSim ("C:\UDK\USARGame\Content\Maps"), caso se especifique um ficheiro de extensão '.ut3', o directório de defeito passa a ser o que corresponde à localização usual por parte dos utilizadores de USARSim em conjunto com UT3 (C:\Program Files\Unreal Tournament 3\usarsim\Unpublished\USARSimMaps). Para identificar a localização do ficheiro correspondente ao mapa pode escrever-se directamente na linha de edição de texto sem recorrer ao gestor de ficheiros, o mapa tem de estar no formato '.ut3' ou '.udk', dependendo da versão de motor de jogos a ser utilizada.

Na *checkbox* com a legendagem "Also make '.bat' files", permite ao utilizador escolher se a aplicação deve escrever ou não o ficheiro de execução (*batch*) do mapa ('nomeDoMapa.bat'), no caso do mapa ser de extensão 'ut3' e esta opção estiver assinalada, a aplicação escreve igualmente o ficheiro de configuração inicial do mapa ('nomeDoMapa.ini') necessário para a correcta simulação do mesmo. Ao clicar no botão 'Load', a aplicação verifica a existência do mapa seleccionado e, posteriormente, escreve ou não os ficheiros dependendo da opção do utilizador (no caso dos ficheiros não existirem cria-os ignorando a opção do utilizador). Se não houver erros o botão 'Save' é activado. O botão 'Save' grava a configuração do mapa actual e o botão 'Default' grava a configuração do mapa utilizado por defeito. A *checkbox* de título "Run remote map" deverá ser assinalada se, se desejar correr um mapa num outro computador, mas o utilizador terá de tomar certas medidas antes de colocar esse mapa a correr para que a simulação robótica decorra sem incidentes. No entanto é lançada uma mensagem de aviso quando esta opção é assinalada (figura 4.4.1.2).

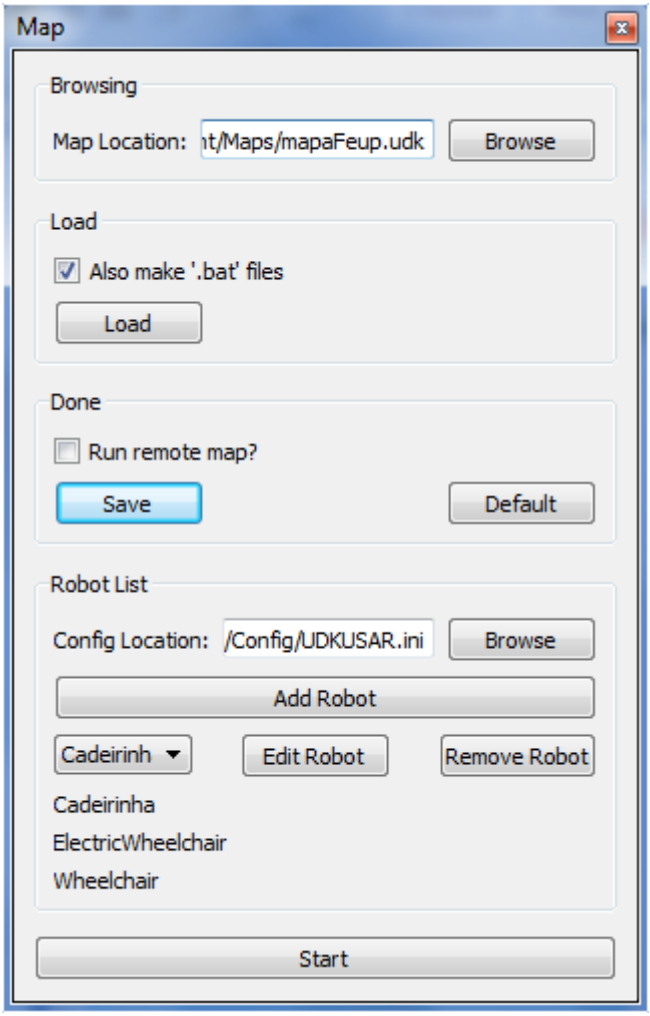


Figura 4.4.1.1 – Interface Gráfica de Configuração do mapa ('Map')

Na parte inferior desta interface encontra-se a lista de robôs a simular posteriormente, inicialmente a lista está vazia tal como a *combobox* de selecção. Primeiro deve-se seleccionar a localização do ficheiro de configurações iniciais, escrevendo directamente a localização do mesmo ou clicando no botão 'Browse'. Em seguida pode-se adicionar, editar ou remover robôs recorrendo, respectivamente, aos botões 'Add Robot', 'Edit Robot' e 'Remove Robot'. Quando se tiver finalizado a configuração de todos os robôs que se pretende simular, basta que o utilizador clique no botão 'Start' para proceder ao início da simulação.

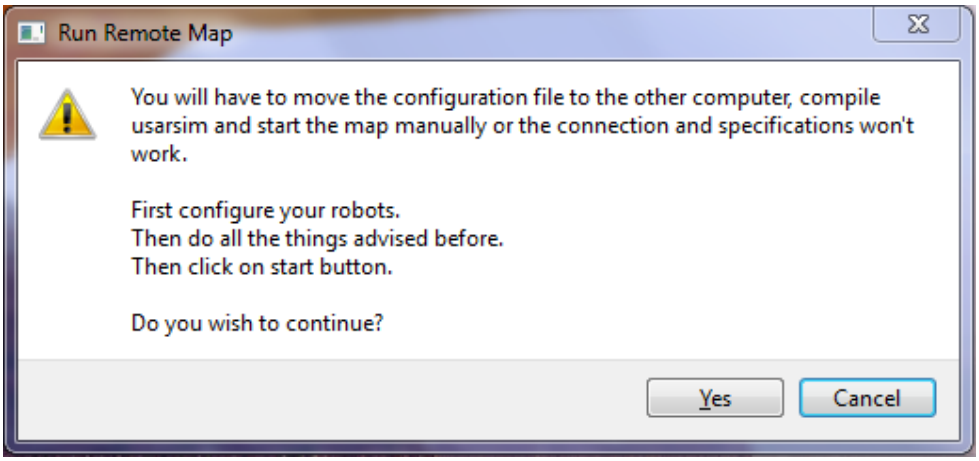


Figura 4.4.1.2 – Interface Gráfica – Mensagem de Aviso de "Run remote map" ('Map')

Tal como evidenciado na mensagem de aviso (figura 4.4.1.2), embora a aplicação permita a ligação ao ambiente de simulação em modo remoto, podem surgir problemas de sincronização de configurações iniciais se o utilizador não seguir os passos indicados. Uma meta futura seria criar uma aplicação mínima cujo único objectivo fosse comunicar com a aplicação desenvolvida e executar as acções já implementadas em ambiente remoto.

4.4.2 Janela 'Robot Configuration'

Na figura 4.4.2.1 pode-se visualizar a interface gráfica a que o utilizador tem acesso para proceder à configuração do robô e dos seus sensores, a utilizar durante a simulação robótica.

Esta interface gráfica permite configurar o robô e adquire a informação necessária do ficheiro de configuração anteriormente seleccionado na janela denominada 'Map'. A primeira linha de edição de texto com o título "Robot Name" é onde o utilizador deve identificar o nome do robô, em seguida, pode-se seleccionar o tipo de robô (os tipos de robô foram previamente descobertos a partir da leitura do ficheiro de configuração). Posteriormente, pode-se clicar no botão 'Load' que por sua vez inclui na lista de sensores os sensores anteriormente adicionados à configuração daquele tipo de robô.

A segunda linha de edição de texto de título "Sensor Name" é onde o utilizador deve identificar o nome do sensor que pretende adicionar ao robô. Em seguida é possível seleccionar o tipo de sensor e identificar a sua localização e rotação relativa ao centro do robô. Os possíveis tipos de sensores são previamente identificados a partir da leitura do mesmo ficheiro de configurações iniciais.

Ao clicar no botão de adicionar ('Add Sensor') o sensor é adicionado à tabela e internamente à classe 'Robot'. O botão 'Remove Sensor' permite remover qualquer sensor, previamente adicionado, da lista de sensores e do robô actual. Ao clicar no botão 'Advanced' deveria ser visualizada uma outra janela onde apareceriam as características configuradas no ficheiro anteriormente identificado com a possibilidade de as alterar automaticamente. Também deveria haver um outro botão denominado 'Advanced' inserido no grupo denominado 'Load Robot' para modificar de forma automatizada os parâmetros de configuração da classe do robô. No entanto, neste momento, as funcionalidades correspondentes às configurações avançadas de sensores e de classe de robô não estão implementadas.

O botão 'Save' grava as configurações do robô e seus sensores internamente e o botão 'Default' ignora qualquer configuração descrita anteriormente através da interface e assume internamente as configurações por defeito do robô e respectivos sensores. Ambos os botões após clique fecham a presente janela.

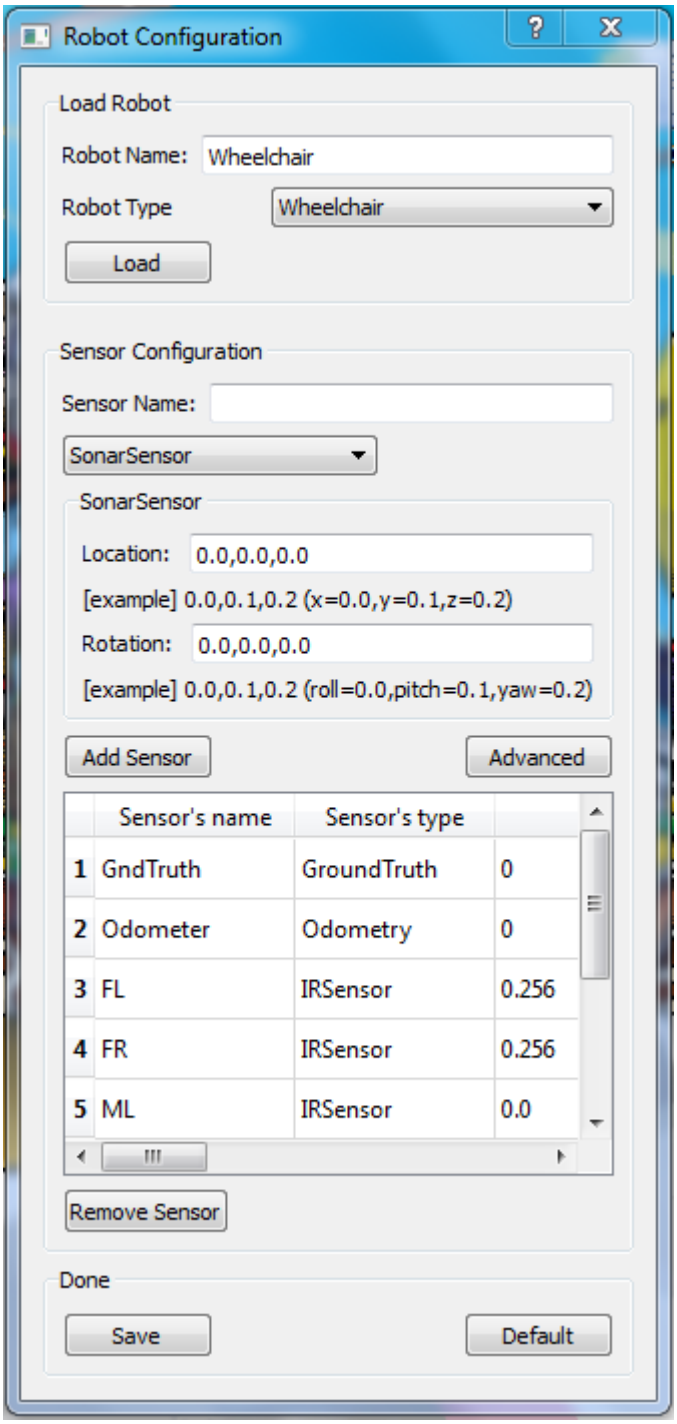


Figura 4.4.2.1 – Interface Gráfica de Configuração do robô e seus sensores ('Robot Configuration')

4.4.3 Janela 'Connect'

Nas figuras 4.4.3.1-3 pode-se visualizar a interface gráfica a que o utilizador tem acesso para proceder ao início da simulação, à ligação por TCP socket ao IP e porta do motor de jogos do Unreal Engine 3 e à interrupção da mesma ligação.

Esta interface gráfica tem dois botões: o botão 'Connect' inicia a ligação ao motor de jogos Unreal Engine 3; e o botão 'Stop' termina a ligação previamente estabelecida. Esta janela tem igualmente duas linhas de edição de texto com os títulos "Host" e "Port" que é

onde o utilizador tem de colocar respectivamente o endereço IP e a porta a que se quer ligar. Como visível na janela de diálogo, esta disponibiliza uma tabela de mensagens referindo o estado actual da simulação e da ligação ao motor de jogos. Nas figuras abaixo é possível visualizar a mesma janela noutros estados da simulação.

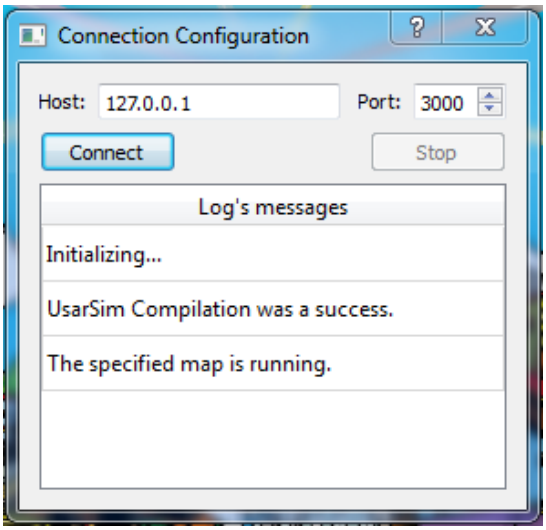


Figura 4.4.3.1 – Interface Gráfica de simulação ('Connect')

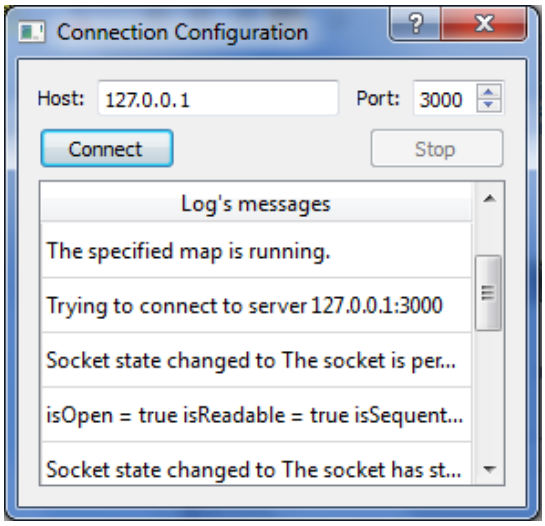


Figura 4.4.3.2 – Interface Gráfica de simulação ('Connect')

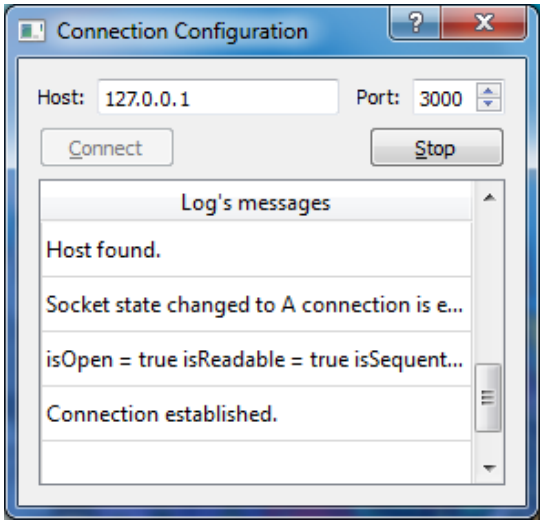


Figura 4.4.3.3 – Interface Gráfica de simulação ('Connect')

4.4.4 Janela ‘Control’

Na figura 4.4.4.1 pode visualizar-se a interface gráfica a que o utilizador tem acesso para pedir as posições iniciais do mapa para o robô, cujo nome e classe se encontram identificados, e colocá-lo numa dada posição do mapa. O utilizador pode igualmente pedir informações geográficas e de configuração do mesmo. Outra particularidade desta interface é a visualização dinâmica dos valores lidos pelos sensores do robô.

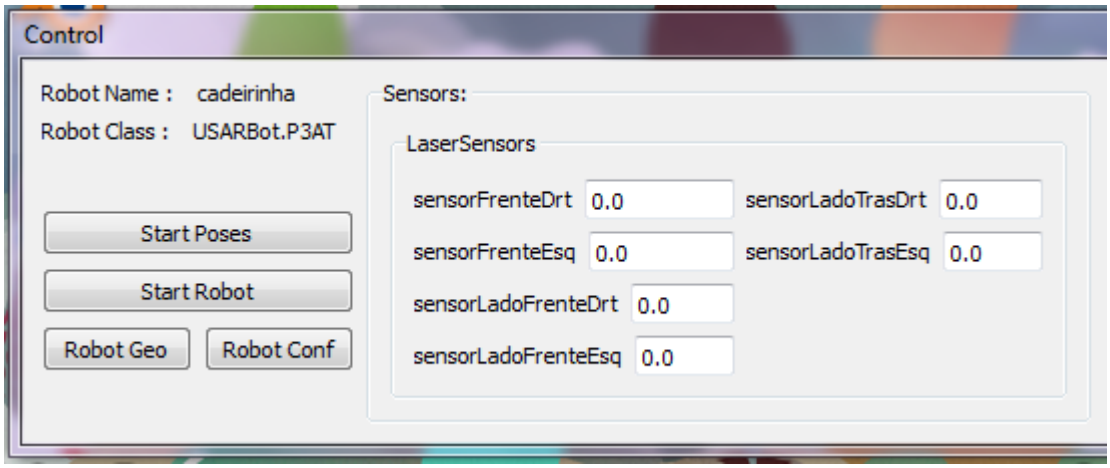


Figura 4.4.4.1 – Interface Gráfica de controlo do robô (‘Control’)

No topo da janela visualiza-se o nome e classe do robô actual com a respectiva denominação, ‘Robot Name:’ e ‘Robot Class’. O botão ‘Start Poses’ ao ser clicado pede as posições iniciais configuradas para o mapa actualmente em execução e o botão ‘Start Robot’ inicia o robô fazendo o *spawn* do mesmo no motor de jogos Unreal Engine 3 que se torna visível no ecrã de simulação.

Por sua vez, os botões ‘Robot Geo’ e ‘Robot Conf’ ao serem clicados pedem ao motor de jogos, respectivamente, a informação geográfica e as configurações iniciais do robô.

Nas figuras 4.4.4.2 e 4.4.4.3 pode-se visualizar a totalidade desta janela que no lado direito é composta por duas janelas organizadas em separadores. A primeira mostra o separador “IA Control” seleccionado e a segunda mostra o separador “Manual Control” seleccionado.

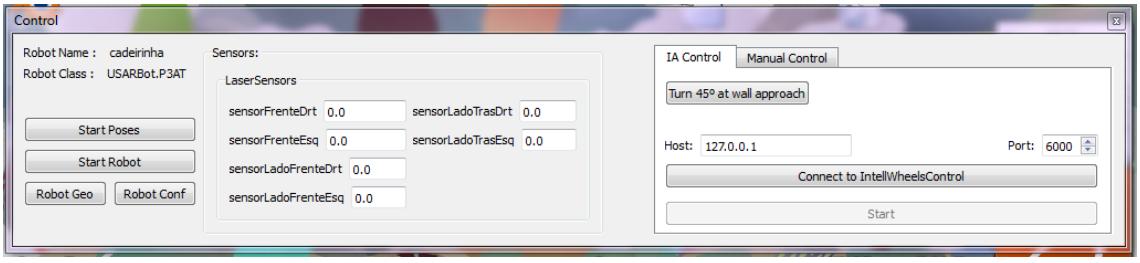


Figura 4.4.4.2 – Interface Gráfica de controlo do robô (‘Control’ e ‘IA Control’)

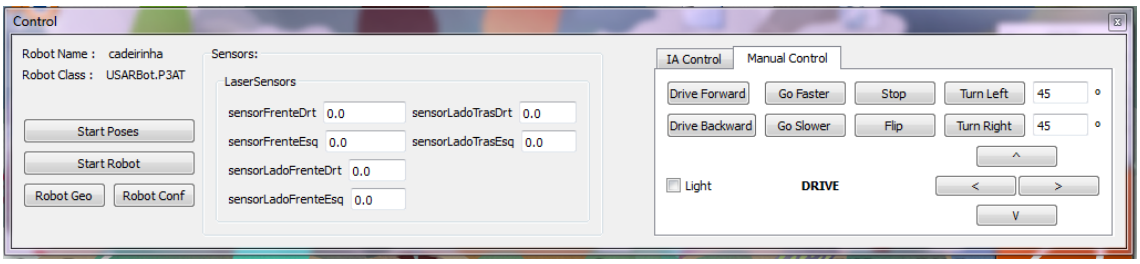


Figura 4.4.4.3 – Interface Gráfica de controlo do robô (‘Control’ e ‘Manual Control’)

4.4.5 Janela ‘IA Control’

Na figura 4.4.5.1 vê-se a janela ‘Control’ com a secção de “IA Control” seleccionada, esta permite interagir com o robô, através de três tipos de controlo comportamental inteligente implementados até ao momento.

O botão ‘Turn 45º at wall approach’ não tem até ao momento qualquer tipo de funcionalidade ou acção a ele atribuída. A ideia era implementar um modo de controlo inteligente com o objectivo de mover o robô em frente e, aquando aproximação de uma parede, rodá-lo quarenta e cinco graus de forma a evitar a parede.

O botão ‘Connect to IntellWheelsControl’ permite ligar a aplicação desenvolvida ao controlador de robô da plataforma ‘IntellWheels’ a um dado IP (‘Host’) e porta (‘Port’). Ao se clicar neste botão é estabelecida comunicação por UDP e a aplicação espera receber do controlador ‘IntellWheels’ o registo de robô cadeira de rodas. Em seguida é possibilitado o clique no botão ‘Start’ que emite a ordem de início de simulação, no sentido em que a partir desse momento a aplicação inicia o envio sistemático de todas as medidas dos sensores e, espera receber, também de uma forma sistemática, os valores de força de acção a aplicar aos motores esquerdo e direito da cadeira de rodas inteligente.

Nesta fase de simulação a aplicação desenvolvida não vai alterar as configurações do presente robô já aplicadas ao mesmo aquando *spawn* no motor de jogos, por não ser possível alterar essas configurações em tempo real.

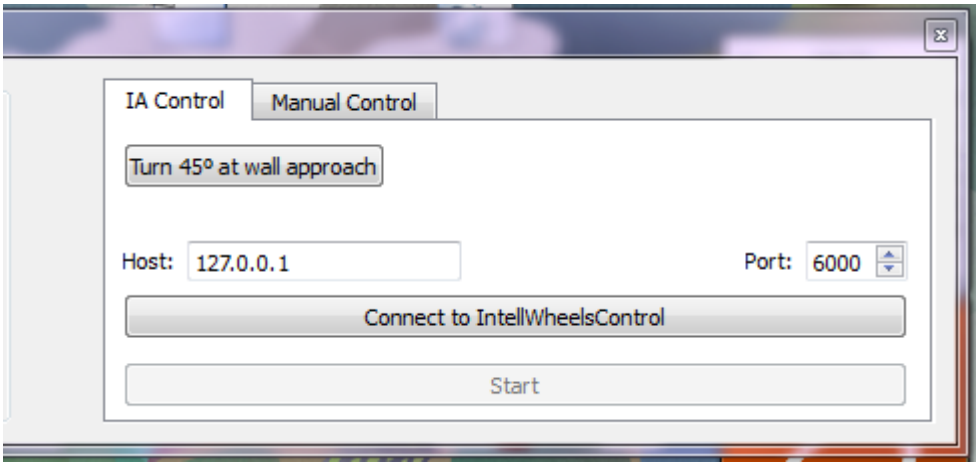


Figura 4.4.5.1 – Interface Gráfica de controlo inteligente do robô (‘IA Control’)

4.4.6 Janela ‘Manual Control’

Na figura 4.4.6.1 vê-se a janela ‘Control’ com a secção de “Manual Control” seleccionada, esta permite ao utilizador conduzir o robô de forma manual.

Os botões ‘Drive Forward’ e ‘Drive Backward’ servem para mandar o robô seguir, respectivamente, em frente e para trás. Ambos os comandos emitem a ordem de seguir a uma velocidade constante, de sentido simétrico. Os botões ‘Go Faster’ e ‘Go Slower’ servem, respectivamente, para acelerar ou desacelerar o robô, correspondendo a aumentar ou diminuir a velocidade absoluta do robô. Ou seja, esteja o robô a andar para a frente ou para trás, ao clicar no botão de seguir mais rápido, o valor absoluto da velocidade aumenta e o robô anda efectivamente mais rápido. Caso o robô esteja parado, ao clicar em qualquer um dos dois botões não há mudanças no comportamento do robô.

O botão ‘Stop’ pára o robô instantaneamente e o botão ‘Flip’ endireita as rodas do robô ou o próprio robô se este tiver as rodas desalinhadas relativamente ao seu eixo de rotação ou este estiver virado ao contrário.

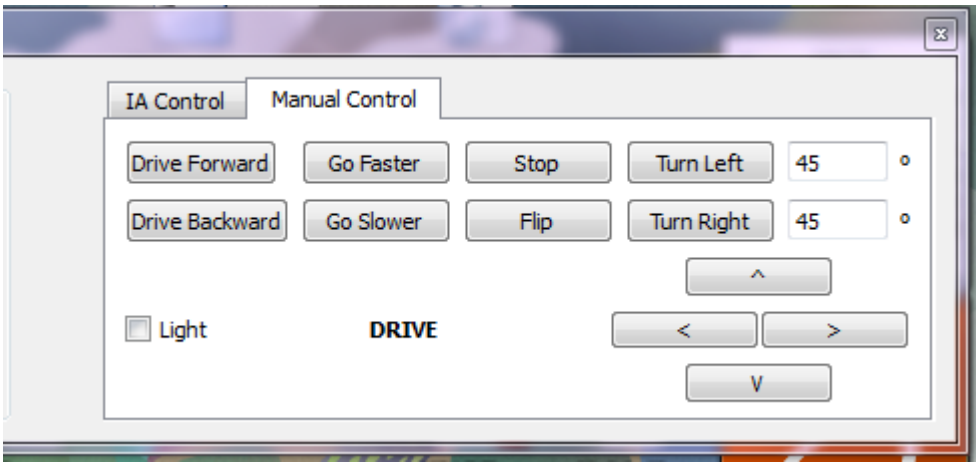


Figura 4.4.6.1 – Interface Gráfica de controlo manual do robô (‘Manual Control’)

Os botões ‘Turn Left’ e ‘Turn Right’ permitem rodar o robô segundo o ângulo medido em graus inserido pelo utilizador nas linhas de edição de texto correspondentes, posicionadas à frente dos respectivos botões. Estas duas funcionalidades estão a executar o objectivo pretendido, mas não foi possível testar em concreto a precisão do movimento.

A *checkbox* ‘Light’ permite acender ou desligar um foco de luz do robô, algo que pode ser útil quando se está a conduzir um robô num troço do mapa que não esteja bem iluminado. No entanto, aquando a construção do modelo gráfico e mecânico, é necessário incluir na sua estrutura o foco de luz à semelhança do robô P3AT do USARSim. Esta é a única funcionalidade da presente interface gráfica que não está directamente relacionada com a condução do veículo.

Os quatro botões de título ‘DRIVE’, representados por setas emitem comandos que conduzem o robô ora para a frente, ora para os lados, ora para trás. Os movimentos de rotação proporcionados pelas setas dos lados rodam o robô aproximadamente um terço de quarenta e cinco graus, a cada clique.

4.4.7 Janela ‘WheelchairSim’

Na Figura 4.4.7.1 (página seguinte) pode-se visualizar o protótipo inicial da janela principal da interface gráfica desenvolvida para o simulador e visualizador de cadeira de rodas inteligente a três dimensões, à excepção da janela de configuração do robô e respectivos sensores, todas as janelas estão visíveis e encaixam na janela principal. As janelas secundárias – Map, Connect e Control – pertenciam à estrutura da janela e podiam ser arrastadas para fora da janela principal ou recolocadas pelo utilizador aquando a sua interacção com a aplicação. No ecrã central, esta figura mostra a inclusão da visualização da imagem de simulação criada pelo motor de jogos do Unreal Engine 3 (versão de compra do jogo Unreal Tournament). Nesta fase de desenvolvimento apenas se considerava a existência de um único robô para simulação.

Na versão actual da aplicação já se possibilita a gestão de simulação de vários robôs, até ao máximo configurado no Unreal e USARSim, por defeito o máximo corresponde a dezasseis robôs, e igual número de ligações por TCP socket permitidas.

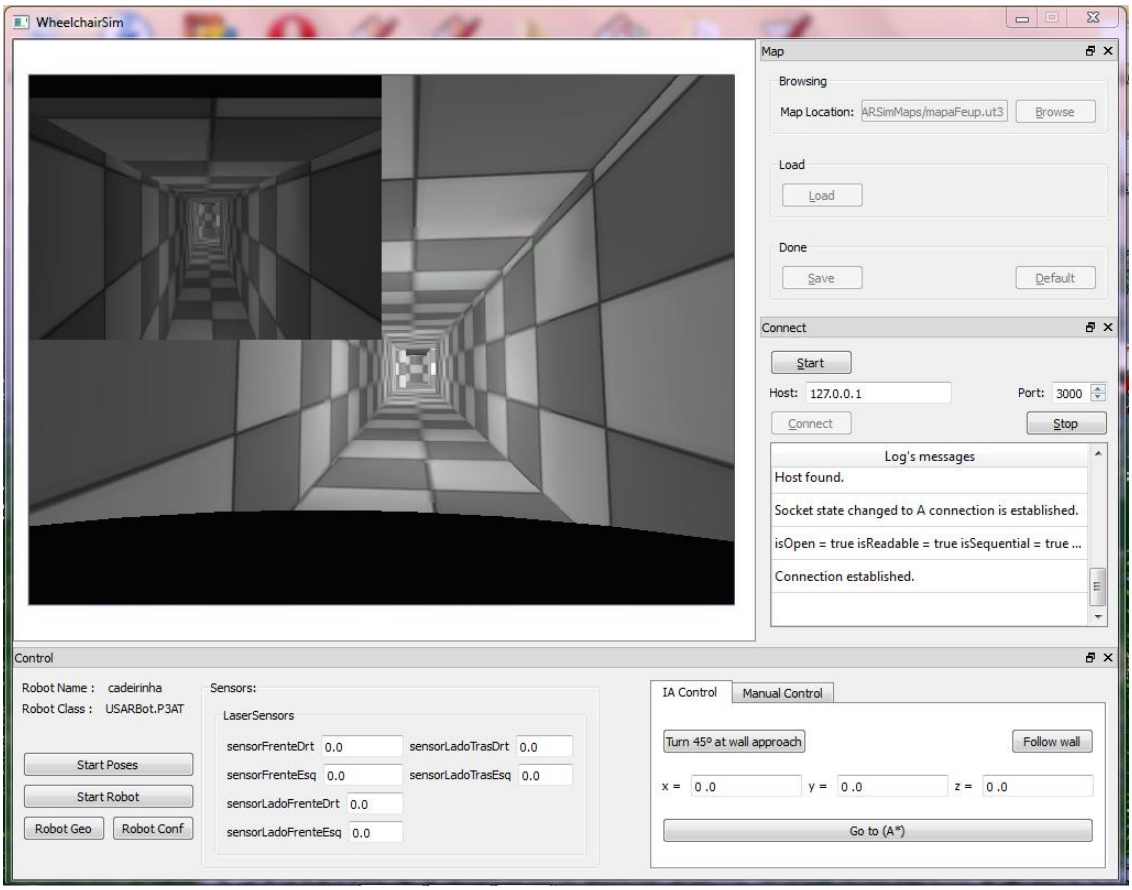


Figura 4.4.7.1 – Protótipo inicial da Janela 'WheelchairSim' recorrendo ao UE3 (versão do jogo)

Na Figura 4.4.7.2 pode-se visualizar a janela que aparece inicialmente demonstrativa do ambiente de simulação criado a partir do editor do motor de jogos Unreal Engine 3 (versão do jogo) e visualizado recorrendo ao motor de jogos em conjunto com o simulador robótico USARSim, cujo mapa de simulação é o de nome "mapaFeup.ut3" criado em semelhança ao corredor e aos dois laboratórios de inteligência artificial existentes na Faculdade de Engenharia da Universidade do Porto.

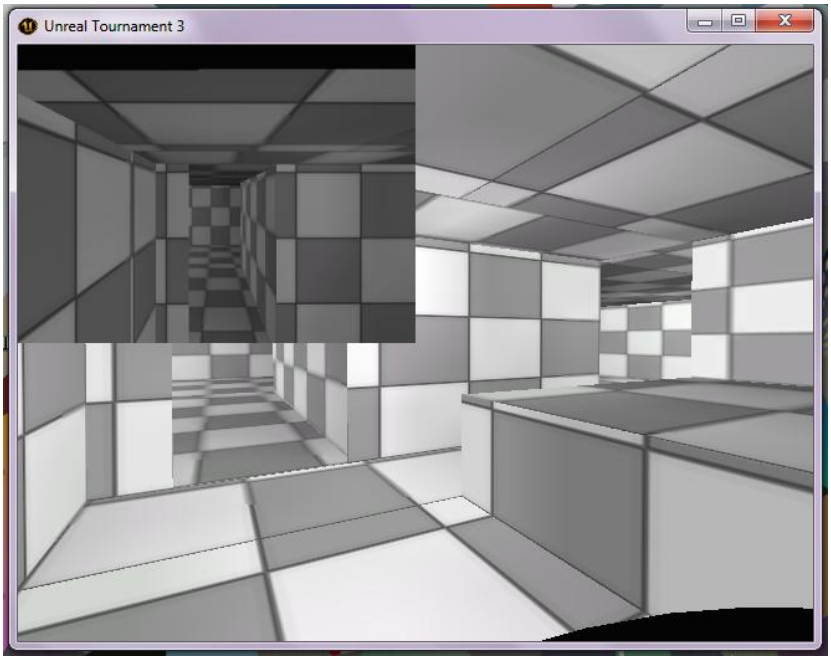


Figura 4.4.7.2 – Ambiente de Simulação criado através do editor versão do jogo ("mapaFeup.ut3")

Posteriormente foi modelado graficamente, ao nível dos materiais e texturas e, de seguida, importado para o editor de UDK (versão de Fevereiro de 2011) adoptando o nome "mapaFeup.udk".

O robô utilizado nesta simulação foi o "P3AT.upk" disponibilizado em conjunto com o simulador robótico USARSim. No canto esquerdo superior aparece um quadrado que representa a imagem do sensor câmara aplicado ao robô numa posição superior ao mesmo. A mancha preta pertence ao robô o que indica que a nossa posição actual é imediatamente acima do robô.

Uma particularidade interessante é o facto de podermos mover a nossa posição ao longo do ambiente de simulação não alterando a posição do robô, o que permite inúmeros pontos de vista da simulação, mantendo, neste caso, o ponto de visto de primeira pessoa do robô através da observação da imagem da câmara.

Figura 4.4.7.3 Ambiente de Simulação ("USARSim Deathmatch")

Na Figura 4.4.7.3 está presente o ecrã que aparece quando se clica na tecla ESC estando com o rato por cima do ecrã de simulação (motor UE3, versão do jogo). Aparece a amarelo o jogador de nome "Player" que representa o nosso ponto de vista alterável através das setas do teclado e a cinzento aparece o nome do nosso robô, neste caso denominado "cadeirinha".

Na Figura 4.4.7.4 apresentam-se os menus proporcionados pela janela principal, actualmente as acções dos menus ainda não estão activas.

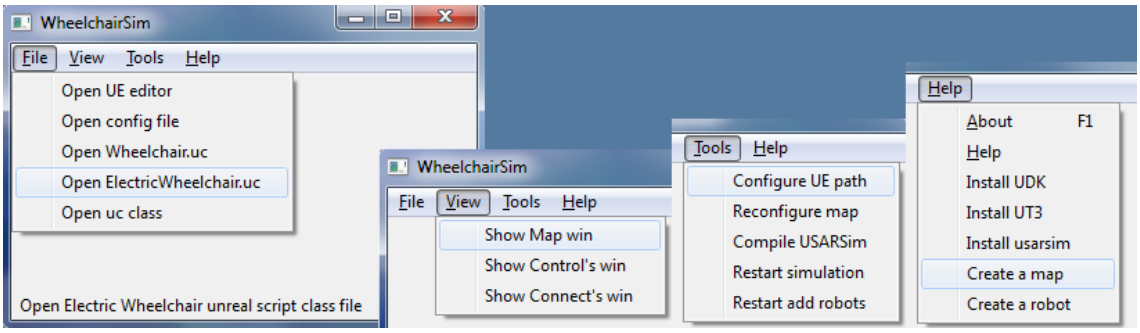


Figura 4.4.7.4 Menus da Janela Principal ('WheelchairSim')

Visto pretender-se criar um simulador robótico para cadeiras de rodas inteligentes considerou-se o objectivo essencial de possibilidade de simulação de várias cadeiras de rodas inteligentes ao mesmo tempo. Como tal, para cada configuração de robô, previamente adicionado à simulação, são iniciadas duas janelas de diálogo, uma do tipo 'Connect' que permite a ligação ao motor de jogos e uma do tipo 'ControlSim' que permite o controlo do comportamento robótico (leitura dos valores dos seus sensores e movimentação geográfica da mesma – controlo manual e controlo automático).

4.5 Particularidades do USARSim e Unreal Engine 3

Nesta secção do capítulo de implementação o objectivo é referir as características principais do simulador robótico USARSim relevantes para o projecto em questão. Como tal, inicialmente, é descrita a arquitectura do sistema de simulação e, em seguida, são abordados os pontos-chave relacionados com os tipos de robôs e sensores disponíveis actualmente.

Na figura 4.5.1 é apresentada a arquitectura do sistema que se encontra dividido em três componentes principais: o 'Unreal Engine' que faz o papel de servidor; o 'Gamebots' responsável pela comunicação entre o servidor e o cliente; e o cliente 'Controller' responsável por controlar os robôs no simulador.

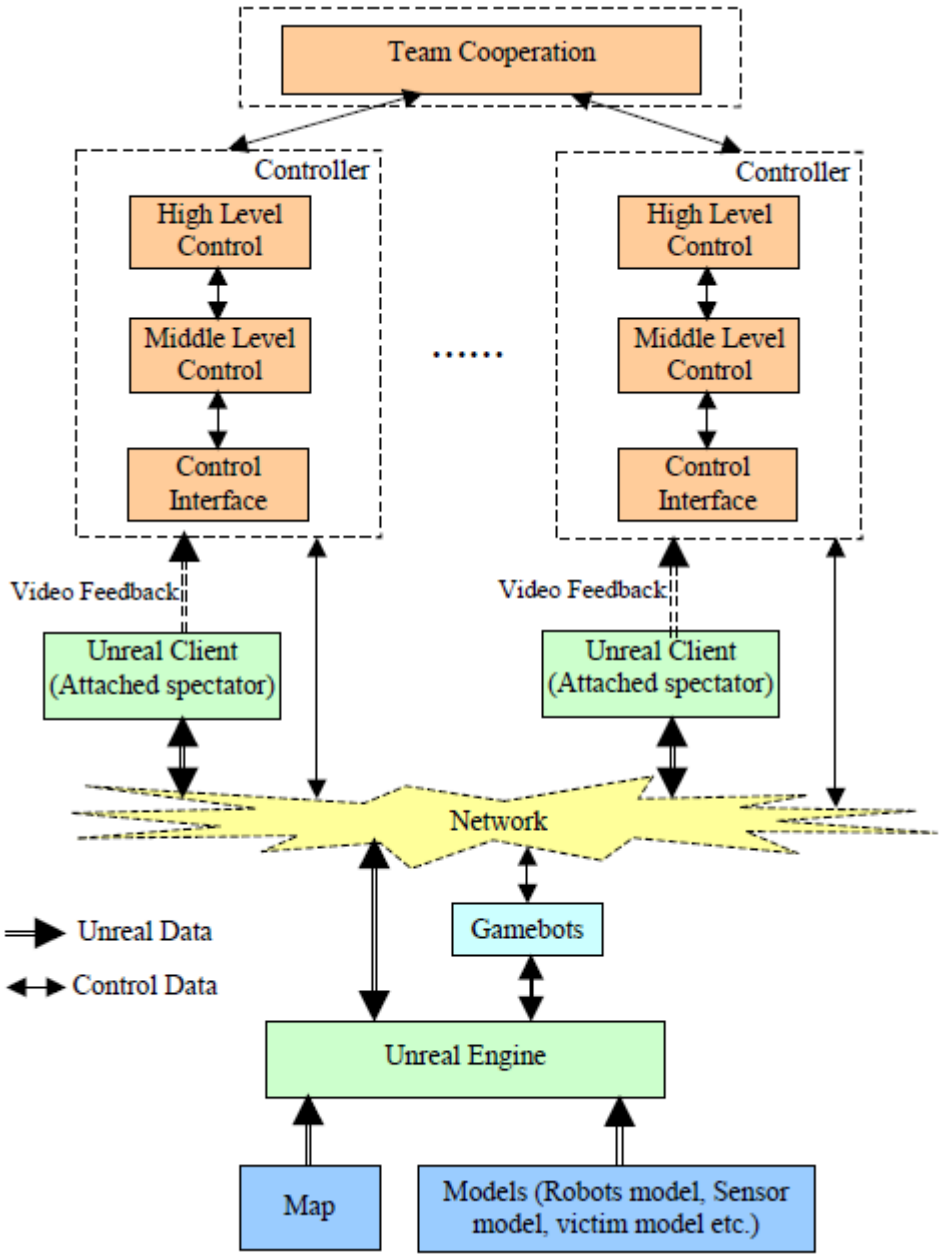


Figura 4.5.1 Ambiente de Simulação ("USARSim Deathmatch") [68]

Normalmente, o 'Controller' (lado cliente da aplicação) funciona da seguinte forma: liga-se ao servidor de Unreal e, de seguida, envia um comando ao USARSim para fazer o *spawn* do

robô. Depois da criação do robô no simulador, o 'Controller' escuta a informação dos sensores e envia comandos para controlar o robô.

O 'Gamebots' (a ponte entre o 'Unreal Engine' e o 'Controller') abre uma socket TCP/IP para comunicar: o endereço de IP corresponde ao IP da máquina onde está a correr o servidor de Unreal, o número da porta é 3000 por defeito e o número máximo de ligações é dezasseis. Estas configurações podem ser alteradas no ficheiro "BotAPI.ini" do sistema de directórios do Unreal, na secção [BotAPI.BotServer]. A aplicação desenvolvida permite seleccionar-se o endereço IP e porta a que se quer ligar.

A arquitectura de cliente/servidor do Unreal torna possível a adição de múltiplos robôs ao simulador, mas como cada robô usa uma socket para comunicar é necessário que o controlador crie uma ligação para cada robô.

Em relação a unidades, o motor de jogos usa a sua própria unidade para medir comprimentos e ângulos, denominada "Unreal Unit" (UU). Há duas excepções: usa graus para medir o *Field Of View* (FOV) e usa radianos em funções trigonométricas. Para converter de uma unidade para outra pode-se ter em conta que 250 UU corresponde a um metro; ou que 32768 UU corresponde a PI radianos que corresponde a 180 graus que representa meio circulo.

A base do simulador robótico USARSim traduz-se em três componentes principais, a simulação do ambiente interactivo, dos robôs e dos sensores e actuadores. Uma descrição mais detalhada vai ser providenciada relativamente aos robôs e seus sensores, os comandos usados vão ser igualmente referenciados.

Na realidade existem duas versões principais do motor de jogos do *Unreal Engine 3*, uma versão mais antiga está disponível aquando a compra do jogo *Unreal Tournament 3* e uma versão mais recente e em constante adaptação está disponível quando se descarrega do site oficial o *Unreal Development Kit* (UDK). Cada um destes motores disponibiliza um editor similar, no entanto, sendo o editor do UDK mais recente este tem funcionalidades de grande potencial. Uma das novas funcionalidades disponibilizadas por este editor corresponde à importação de objectos no formato '.FBX' que aumenta a facilidade de importação de objectos modelados em *3dsMax*, sejam estes objectos estáticos ou objectos com esqueleto.

Neste momento, já foi lançada uma nova versão do USARSim compatível com o motor de jogos do UDK (versão de Fevereiro de 2011), logo foram efectuadas todas as transformações de formatos e mudança de hierarquias de objectos necessárias à correcta utilização deste simulador robótico. Tendo sido o mapa do mundo virtual criado primariamente para o *Unreal Engine 3* lançado aquando o lançamento do jogo, este teve de ser adaptado ao motor de jogos do UDK. Em relação à cadeira de rodas virtual, esta foi implementada para UDK, logo não funciona na versão anterior do motor de jogos.

As diferenças mais relevantes entre os dois motores de jogos são o facto da versão disponibilizada com o jogo ter maior número de texturas, materiais, veículos, armas e personagens e permitir o acesso ao código fonte C++ do motor de jogos. No entanto, a versão do UDK permite a execução de tudo o que é criado pelo próprio, sem o utilizador ter de instalar o UDK, e facilita a importação de modelos em *3dsMax*, *Maya* ou *XSI*.

4.5.1 Configuração de Robôs

O USARSim implementa cinco classes de robôs distinguidos pelo seu tipo de locomoção: "Skid Steered Robot"; "Ackerman Steered Robot"; "Underwater Robot"; "Aerial Vehicle" e "OmniDrive Robot".

Como o objectivo foi implementar um robô semelhante a uma cadeira de rodas só três tipos de robôs foram considerados: “Skid Steered Robot” que corresponde a um veículo conduzido através da manobra das rodas laterais e derrapagem, exibindo um movimento semelhante a um carrinho de choque (ambas as rodas de cada lado rodam com o mesmo ângulo; “Ackerman Steered Robot” que corresponde a um veículo conduzido através da manobra das rodas da frente e rodas de trás, exibindo um movimento semelhante a um carro de corrida; e “OmniDrive Robot” cujo comportamento de condução pode ser concebido através da configuração de diferentes rotações e velocidades para cada roda.

Para o segundo robô enunciado acima, é necessário considerar o princípio de Ackerman que descreve a relação entre um par de rodas (por exemplo as da frente) quando o veículo roda. Simplificadamente diz que quando um veículo pretende fazer uma curva, as rodas do lado interior rodam sob um diâmetro inferior ao diâmetro das rodas do lado exterior do veículo. Como acontece em carros de corrida que dispõem de braços angulares nos eixos de cada par de rodas, para que a sua prestação em curvas seja melhor e de maior aderência ao chão.

Nesta aplicação foi implementado um modo de controlo dos três tipos de robôs enunciados e respectivo comando de condução.

Ao ser identificado pela aplicação que se está perante um robô do tipo “Skid Steered Robot”, o comando usado, tanto no controlo manual como no controlo inteligente é o seguinte: **DRIVE {Left** *float***} {Right** *float***} {Normalized** *bool***} {Light** *bool***} {Flip** *bool***}**. Os dois primeiros parâmetros correspondem às velocidades de rotação, respectivamente das rodas do lado esquerdo e do lado direito. A unidade é em radianos por segundo caso se esteja a usar valores absolutos, ou varia entre menos cem e cem caso se esteja a usar valores normalizados. O terceiro parâmetro especifica a unidade, se verdadeiro está-se perante valores normalizados, se falso está-se perante valores absolutos. O quarto parâmetro identifica se o foco de luz dianteiro do robô está ligado ou não e o quinto parâmetro ordena o robô a voltar à sua posição correcta (por exemplo, se o robô capota ou as suas rodas se soltam, emitindo esta ordem, o robô volta à sua posição correcta de rodas para baixo). Ambos os parâmetros aceitam apenas verdadeiro ou falso.

Outra classe de robô explorada foi o “Ackerman Steered Robot”, o comando testado, no controlo manual é o seguinte: **DRIVE {Speed** *float***} {FrontSteer** *float***} {RearSteer** *float***} {Normalized** *bool***} {Light** *bool***} {Flip** *bool***}**. O primeiro parâmetro identifica a velocidade de rotação do robô e os dois parâmetros seguintes correspondem aos ângulos de orientação das rodas, respectivamente da frente e de trás. A unidade é em radianos por segundo caso se esteja a usar valores absolutos, ou varia entre menos cem e cem caso se esteja a usar valores normalizados. O quarto, quinto e sexto parâmetros especificam respectivamente o mesmo que o terceiro, quarto e quinto parâmetros do comando enunciado anteriormente.

No caso de se especificar um robô que não seja do primeiro tipo identificado, outros comandos “DRIVE” do USARSim terão de ser chamados, algo que a aplicação desenvolvida não suporta visto não ser o objectivo primário pretendido da mesma.

Outras formas de controlo consideradas foram o modo de controlo de um robô do tipo “OmniDrive Robot” que consiste em controlar cada roda separadamente fornecendo o seu número de identificação e a sua velocidade de rotação e ângulo de orientação. Outro modo de controlo interessante seria o controlo de cada *joint* de um qualquer robô separadamente fornecendo o seu nome de identificação e o seu ângulo de orientação, ora para modificar o seu ângulo de rotação, ou a sua velocidade angular, ou o seu torque.

Diferentes abordagens poderiam ser estudadas para esta aplicação, seria interessante investigar em maior profundidade as formas de controlo enunciadas no parágrafo anterior.

O USARSim dispõe de vários exemplos de robôs reais modelados internamente, um ou mais para cada tipo de locomoção com as suas respectivas propriedades de configuração. Alguns exemplos são: P2AT; StereoP2AT; P2DX; ATRVJr; HMMWV (Hummer); SnowStorm; Sedan; Cooper; Submarine; Tarantula; Zerg; Talon; QRIO; ERS; Soryu; Kurt2D; Kurt3D; Lisa; TeleMax; AirRobot; Passarola; RugBot; Kenaf. O robô P2DX e P3AT são os robôs mais semelhantes à cadeira de rodas configurada para o projecto 'IntellWheels', ao nível da construção mecânica da mesma. Para mais informação acerca destes modelos pode consultar o manual do USARSim. [68]

4.5.2 Configuração de Sensores

Nesta secção são apresentados todos os tipos de sensores que podem ser colocados no robô, previamente configurados no simulador robótico.

Na figura 4.5.2.1 é possível visualizar-se os tipos de sensores disponibilizados pelo USARSim e sua árvore hierárquica.

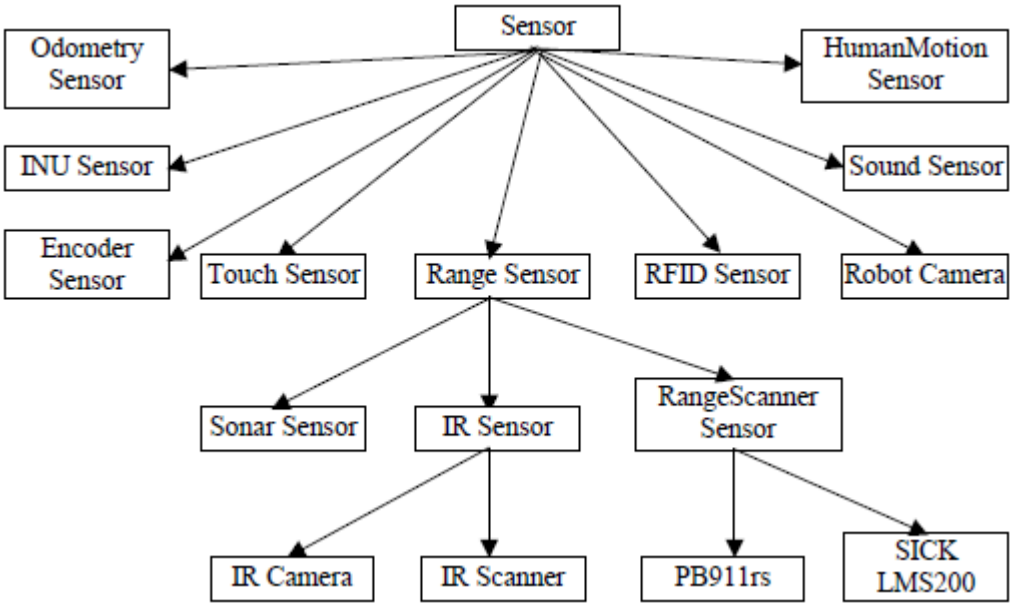


Figura 4.5.2.1 Diagrama da Hierarquia da Classe Sensor do USARSim [68]

Para que um qualquer sensor seja colocado num robô é necessário adicionar uma linha ao ficheiro de configuração do robô, especificando-se o nome, o tipo, a posição e a direcção do mesmo. Em seguida, basta enviar um comando de pedido de leitura do sensor e esperar a resposta correspondente.

Uma outra organização das classes de sensores é apresentada abaixo devido à semelhança entre mensagens de sensores, mas a diferença maior é o facto de se subdividir "Range Sensor" em "Sonar Sensor" e "IR Sensor" e o facto de se apelidar "Laser Sensor" ao conjunto "RangeScanner Sensor" e "IRScanner". Estes grupos de sensores foram completamente implementados na aplicação desenvolvida por serem essenciais ao cumprimento dos objectivos.

Descrição sucinta dos sensores apresentados anteriormente adaptada do manual do USARSim [68]:

- **Sensor de estado** ("*State Sensor*") – Não é preciso configurá-lo e é responsável por reportar o estado do robô no motor de jogos UE3;
- **Sensor de distância** ("*Range Sensor*") - Pode ser de dois tipos (sonar e IR); Se for IR emite uma linha do sensor até ao primeiro obstáculo e devolve a distância

percorrida ou a distância máxima de alcance do sensor caso esta seja menor que a primeira; Se for sonar, em vez de uma linha, emite várias ao longo de um cone de alcance previamente configurado e a distancia devolvida é a menor das distâncias;

- **Sensor de laser** ("*Laser Sensor*" ou "*Range Scanner Sensor*") – Este sensor é semelhante ao anterior no sentido em que foi implementado como uma série de sensores de distância, cuja leitura é obtida rodando o sensor de distância; No modo automático este *scan* é realizado em predeterminados intervalos; Pode ser de dois tipos ("*RangeScanner*" ou "*IRScanner*"), o primeiro usa um sensor de distância que só emite uma linha de detecção de obstáculos e o segundo usa um sensor de distância do tipo IR cuja linha de detecção pode atravessar materiais transparentes;
- **Sensor de odometria** ("*Odometry Sensor*") – Sensor responsável por estimar a posição corrente do robô relativamente à sua posição inicial, necessita que seja especificado qual a roda direita e esquerda do robô;
- **Sensor GPS** ("*GPS Sensor*") – Este sensor simula a localização obtida por satélite, primeiro descobre a localização do robô (em metros) e converte-a para latitude e longitude;
- **Sensor INS** ("*INS Sensor*") – Este sensor estima a posição e orientação actual do robô, após adicionar ruído às velocidades angulares e distâncias percorridas pelo mesmo a cada passo temporal;
- **Sensor de codificação** ("*Encoder Sensor*") – Este tipo de sensor mede o ângulo de rotação de uma determinada parte em torno do eixo do sensor;
- **Sensor de toque** ("*Touch Sensor*") – Este sensor segue o mesmo método do sensor de distância, várias linhas são emitidas a partir da frente do sensor para detectar objectos; O sensor é tratado como um botão que emite um sinal de toque assim que um objecto se encontra muito perto, a menos de 4.7 milímetros;
- **Sensor RFID** ("*RFID Sensor*") – Este sensor simula um leitor de RFID e é usado para detectar etiquetas RFID que se encontrem a seu alcance;
- **Sensor de Vitima** ("*Victim Sensor*") – Este sensor simula um sensor de localização de vítimas, o seu modo de operação é semelhante ao "*Range Scanner*" e retorna as partes atingidas da vítima (pode ser falso alarme);
- **Sensor de detecção de movimento humano** ("*Human Motion Detection*") – Este sensor simula um sensor eléctrico que mede a energia emitida pelos objectos, de forma a classificar o objecto como sendo ou não humano;
- **Sensor de som** ("*Sound Sensor*") – Este sensor detecta todas as fontes de som e calcula a que se encontra a menor distância do robô;
- **Câmara do robô** ("*Robot Camera*") – Funciona como uma câmara real, as cenas vistas a partir do sensor são capturadas anexando o ponto de vista no motor de jogos UT à própria;
- **Câmara omnidireccional** ("Omnidirectional Camera") – Funciona de forma equivalente ao sensor anterior à excepção de ser omnidireccional.

Relativamente aos sensores enunciados, a aplicação desenvolvida considera por defeito um sensor câmara colocado no topo do robô e permite a adição e remoção de sensores do tipo sensor de distância e sensor laser ao mesmo. O sensor de estado não necessita de configuração porque é sempre criado aquando a criação de um robô. Como tal, na próxima secção, serão apresentados os comandos e mensagens relativos aos sensores de estado, de distância, de laser e de câmara.

Visto o estudo destes dois tipos de sensores ter sido mais aprofundado, são apresentadas abaixo todas as configurações possíveis dos mesmos.

Em relação a sensores de distância, estes podem ser sonar ou IR e sua configuração é distinta. No caso dos sensores laser que podem ser do tipo "RangeScanner" ou "IRScanner", os seus atributos de configuração são idênticos diferenciando-se apenas na etiqueta de identificação ([USARBot.RangeScanner] ou [USARBot.IRScanner]).

Exemplo de configuração de sensor de distância do tipo sonar:

```
[USARBot.SonarSensor]
HiddenSensor=true
bWithTimeStamp=true
Weight=0.4
MaxRange=5.0
BeamAngle=0.3491
Noise=0.05
OutputCurve=(Points=((InVal=0.000000,OutVal=0.000000),(InVal=5.000000,
OutVal=5.000000)))
```

Exemplo de configuração de sensor de distância do tipo IR:

```
[USARBot.IRSensor]
HiddenSensor=true
bWithTimeStamp=true
MaxRange=5.0
Noise=0.05
OutputCurve=(Points=((InVal=0.000000,OutVal=0.000000),(InVal=5.000000,
OutVal=5.000000)))
```

Os parâmetros enunciados nos exemplos têm o seguinte significado: 'HiddenSensor' diz se o sensor está ou não visível; 'bWithTimeStamp' diz se os dados lidos pelo sensor devem ou não vir acompanhados do tempo em Unreal aquando leitura; 'Weight' identifica o peso do sensor em quilogramas; 'MaxRange' identifica o alcance máximo do sensor em metros, 'BeamAngle' identifica o ângulo do cone de detecção do sensor em radianos; 'Noise' configura a amplitude de ruído aleatório do robô; 'OutputCurve' corresponde à curva de distorção do sinal.

Em relação aos parâmetros do próximo exemplo, estes têm o seguinte significado: 'ScanInterval' corresponde ao intervalo de tempo entre *scan*, utilizado quando o sensor está em modo automático; 'Resolution' corresponde à resolução de *scan* e a unidade é inteiro (65535 equivalente a 360 graus); 'ScanFov' corresponde ao *Field Of View* de *scan*, tem igual unidade e correspondência de unidades que o parâmetro anterior; 'bPitch' indica o plano de *scan*, se verdadeiro significa que o plano de *scan* é o XOZ; 'bYaw' é equivalente ao parâmetro anterior, mas se verdadeiro significa que o plano de *scan* é o XOY.

Exemplo de configuração de sensor laser do tipo "RangeScanner":

```
[USARBot.RangeScanner]
HiddenSensor=False
bWithTimeStamp=False
Weight=0.4
```

```
    MaxRange=1000.000000
    ScanInterval=0.5
    Resolution=800
    ScanFov=32768
    bPitch=false
    bYaw=true
    Noise=0.0
    OutputCurve=(Points=((InVal=0.000000,OutVal=0.000000),(InVal=1000.000
000,OutVal=1000.000000)))
```

4.5.3 Comandos USARSim utilizados

Nesta secção são descritos todos os comandos e mensagens relevantes para a aplicação desenvolvida à excepção dos comandos de condução do robô que já foram descritos na secção de configuração de robôs por estarem directamente relacionados com o tipo de locomoção do mesmo.

Para se obter as posições iniciais do mapa basta enviar o comando '**GETSTARTPOSES**', esta informação é importante porque ajuda na escolha de uma posição inicial para o robô. Aquando a concepção do mapa através do Unreal Editor, um a dezasseis "UTPlayerStart" podem ser adicionados ao mesmo e serão estas as localizações a serem retornadas quando pedidas as posições iniciais.

Antes de se criar o robô já é possível obter a informação geométrica e de configuração tanto do robô como dos seus sensores através dos comandos '**GETGEO**' e '**GETCONF**'. Apenas é necessário considerar mensagens relativas à geometria e configuração de robôs do tipo "GroundVehicle" e relativas a sensores.

Relativamente ao robô, depois de se enviar o comando de pedido de informação geométrica, obtém-se a seguinte resposta: **GEO {Type** GroundVehicle**} {Name** *string***} {Dimensions** *x,y,z***} {COG** *x,y,z***} {WheelRadius** *float***} {WheelSeparation** *float***} {WheelBase** *float***}**. O primeiro parâmetro identifica o tipo de robô, o segundo o seu nome, o terceiro as suas dimensões em metros e o quarto a posição do seu centro de gravidade em metros. O quinto parâmetro corresponde ao raio das rodas do veículo, o sexto corresponde à distância de separação dos pares de rodas e o sétimo corresponde à distância entre as rodas da frente e as rodas de trás, a unidade dos parâmetros enunciados é o metro.

Relativamente aos sensores, obtém-se a seguinte resposta de informação geométrica: **GEO {Type** *string***} {Name** *string* **Location** *x,y,z* **Orientation** *x,y,z* **Mount** *string***}**, o primeiro parâmetro identifica o tipo de sensor. Em seguida, aparece o nome do sensor, depois a sua localização e orientação relativa à base onde foi montado e, por último o nome da base onde o sensor foi montado. Se houver mais do que um sensor de um dado tipo a mensagem alonga-se, pois apareceram tantos segmentos (nome, localização, orientação e base de montagem) quantos sensores deste tipo.

Relativamente ao robô, depois de se enviar o comando de pedido de informação de configuração, obtém-se a seguinte resposta: **CONF {Type** GroundVehicle**} {Name** *string***} {SteeringType** *string***} {Mass** *float***} {MaxSpeed** *float***} {MaxTorque** *float***} {MaxFrontSteer** *float***} {MaxRearSteer** *float***}**, onde o tipo será sempre "GroundVehicle". O parâmetro nome corresponde ao nome do robô e o seguinte ao tipo de condução do robô, pode ser "AckermanSteered", "SkidSteered" ou "OmniDrive". Em seguida, são apresentados os valores de massa em quilogramas, velocidade angular máxima em radianos por segundo, o torque

máximo em unidades de Unreal, o ângulo máximo de viragem das rodas da frente em radianos e o ângulo máximo de viragem das rodas de trás. Caso se trate de um robô de condução de tipo "SkidSteered", os dois últimos parâmetros serão zero.

Relativamente aos sensores obtém-se a seguinte mensagem de informação de configuração: **CONF {Type** *string***} {*Name Value*} {*Name Value*} (…)**, o primeiro parâmetro corresponde ao tipo de sensor e os pares nome valor descrevem as características do sensor em questão, sendo o nome o tipo de atributo do sensor e o valor o valor de configuração desse atributo.

Para se iniciar o robô no mapa usa-se o seguinte comando: **INIT {ClassName** *robot_class***} {Name** *robot_name***} {Location** *x,y,z***} {Rotation** *r,p,y***}**, em que '*robot_classe'* corresponde ao nome da classe do robô (por exemplo USARBot.P3AT), '*robot_name*' corresponde ao nome que se pretende dar ao robô e os dois últimos parâmetros são, respectivamente, a localização e rotação iniciais do robô.

A mensagem de estado do robô varia conforme o tipo de robô que pode ser "GroundVehicle", "LeggedRobot", "NauticVehicle" ou "AerialVehicle". Neste caso, só interessa referir a mensagem de estado correspondente a robôs do tipo "GroundVehicle", ou seja, **STA {Type** GroundVehicle**} {Time** *float***} {FrontSteer** *float***} {RearSteer** *float***} {LightToggle** *bool***} {LightIntensity** *int***} {Battery** *int***}**. Na qual, o instante de tempo está em segundos e representa o tempo desde que o servidor de Unreal começou a sua execução. Os dois parâmetros seguintes correspondem ao ângulo actual de curvatura, respectivamente, das rodas da frente e das rodas de trás. O parâmetro seguinte identifica se o foco de luz do robô está ligado ou não e, em seguida qual a intensidade desse mesmo foco. Por último, identifica-se o tempo de vida da bateria do robô em segundos.

Em relação às mensagens de sensores só vão ser enunciadas as relativas ao sensor de distância e ao sensor laser. A mensagem de sensores de distância corresponde a **SEN {Type** *string***} {Name** *string* **Range** *float***} {Name** *string* **Range** *float***}**, onde tipo pode ser "Sonar" ou "IR" e cada par seguinte identifica um único sensor, o par desmembra-se em nome do sensor e alcance do mesmo em metros. As mensagens de sensor laser são independentes, ou seja, cada sensor laser emite uma mensagem que corresponde a **SEN {Type** *string***} {Name** *string***} {Resolution** *float***} {FOV** *float***} {Range** *r1,r2,r3…***}**, onde o tipo pode ser "RangeScanner" ou "IRScanner" e o nome é o nome do sensor em questão. Em seguida aparece a resolução do sensor em radianos, o campo de visão (*field of view)* do sensor em radianos e uma série de valores de alcance em metros. No caso dos sensores laser, para se proceder ao *scan* envia-se um comando do tipo: **SET {Type** *string***}{Name** *string***}{Opcode** *string***} {Params** *value***}**, onde tipo corresponde ao tipo do sensor, nome ao nome do sensor, código de operação é "SCAN" e não tem parâmetros de operação. O estado da operação pode ser "OK", o que significa que a operação foi um sucesso, ou pode ser "Failed", o que significa que a operação falhou.

4.6 Modelação do Meio de Simulação

Nesta secção são abordados os principais conceitos de modelação do meio de simulação que incluem a modelação gráfica do mundo virtual, a modelação do robô (cadeira de rodas virtual) e as plataformas de desenvolvimento relevantes para a modelação gráfica.

Na subsecção das plataformas de desenvolvimento são também referidos os repositórios de texturas e objectos a três dimensões de onde se descarregou elementos gráficos para a população do mapa modelado para teste da aplicação desenvolvida.

4.6.1 Modelação do Mapa do Mundo Virtual

Nesta secção são abordadas as principais etapas para o desenvolvimento de um mapa do mundo virtual com o objectivo de testar o comportamento de um robô no mundo real. Para exemplificar esta modelação foi considerado o mapa de um dos corredores da Faculdade de Engenharia da Universidade do Porto e duas salas adjacentes que correspondem aos dois laboratórios de inteligência artificial existentes actualmente na mesma. Nas figuras 4.6.1.1, 4.6.1.2 e 4.6.1.3 é possível visualizar-se três pontos de vista do mapa criado.

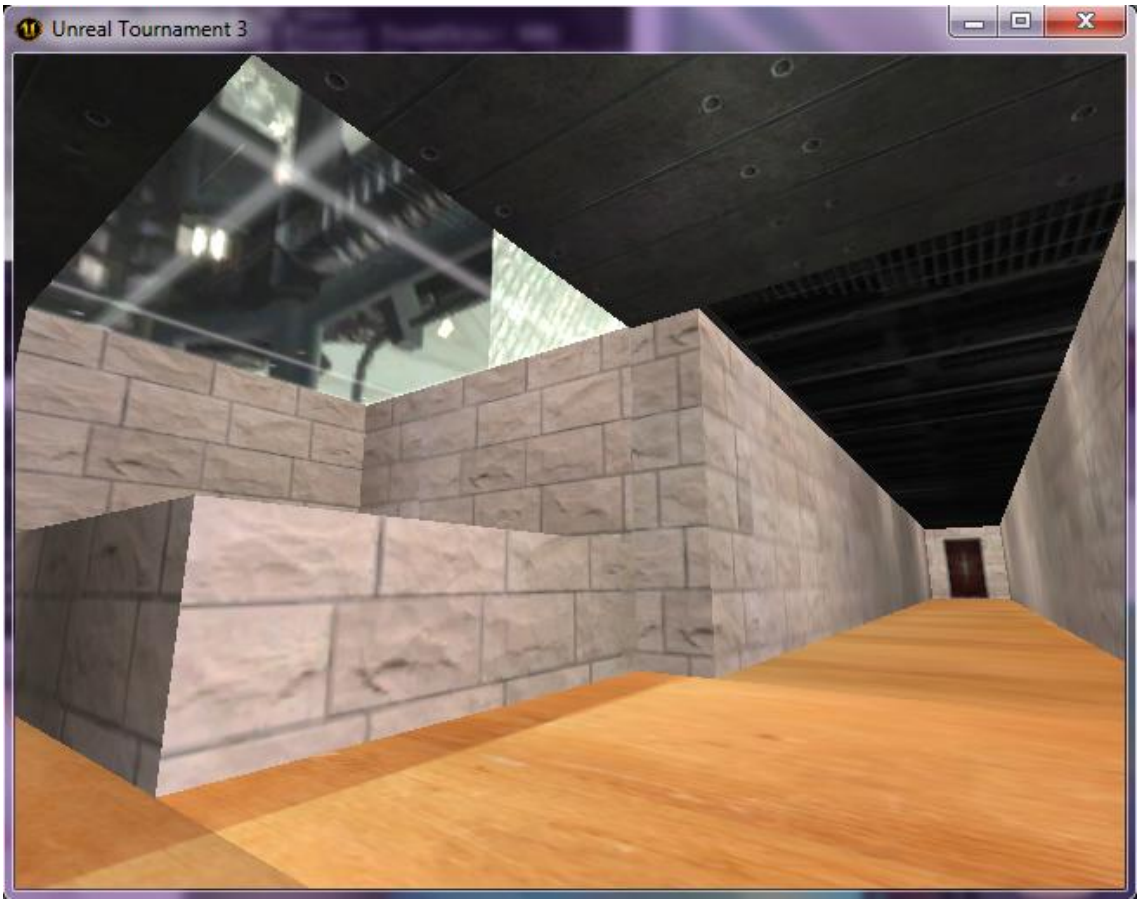


Figura 4.6.1.1 Ponto de vista central do mapa da FEUP

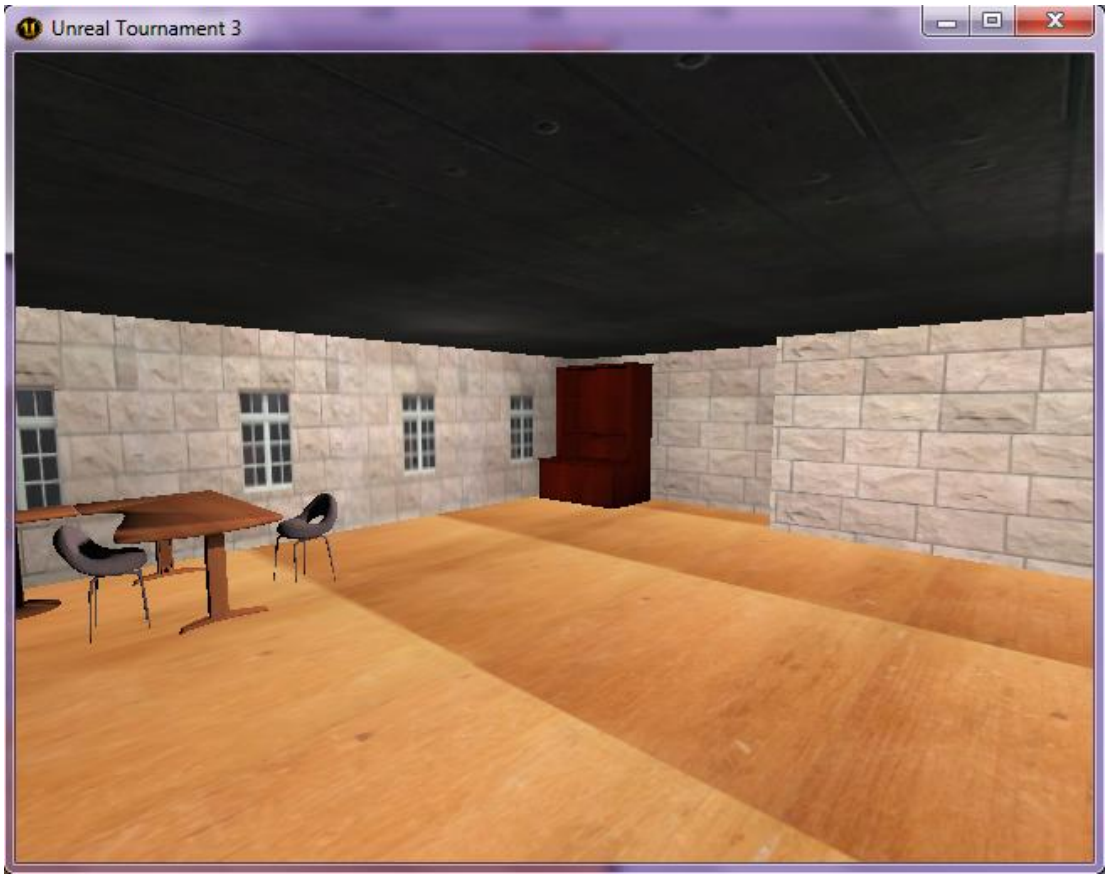


Figura 4.6.1.2 Ponto de vista da sala esquerda do mapa da FEUP

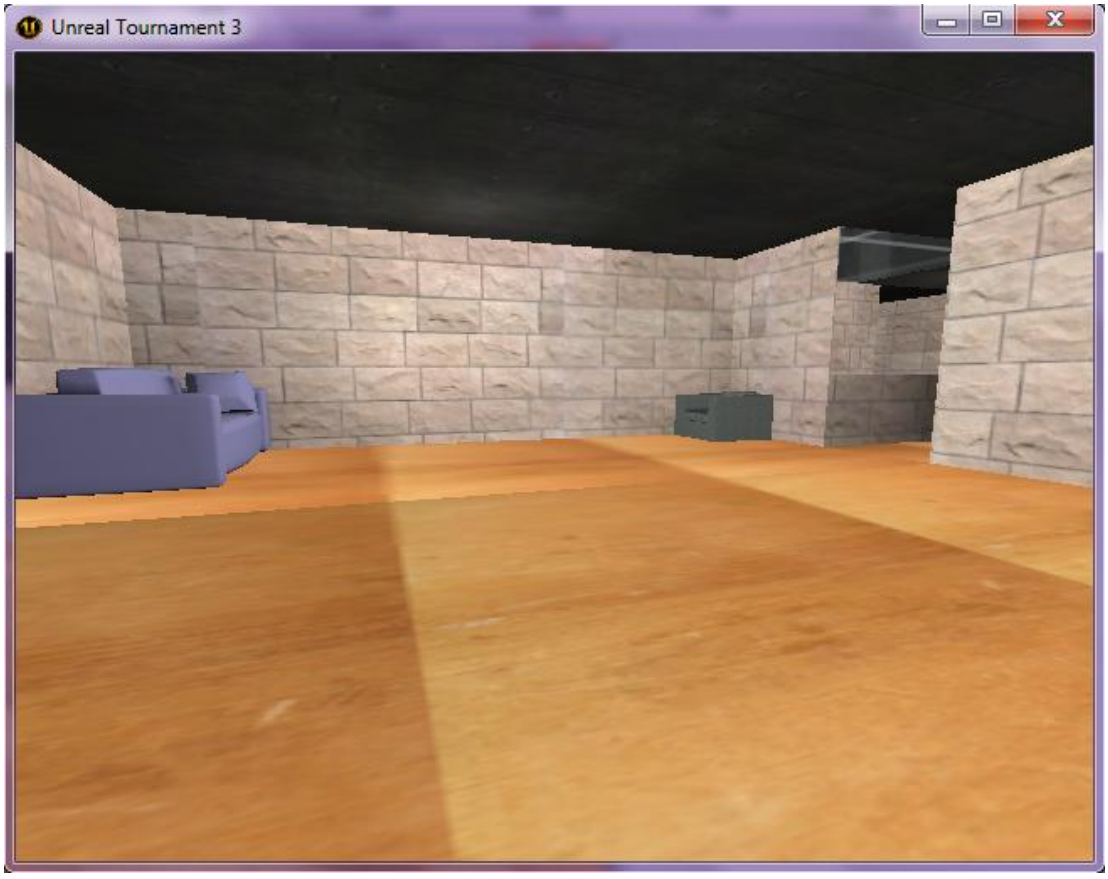


Figura 4.6.1.3 Ponto de vista da sala direita do mapa da FEUP

A modelação gráfica do mundo virtual pode ser dividida nas seguintes etapas:

- Modelação do espaço principal – Esta modelação corresponde à construção dos blocos principais do mapa recorrendo a formas cúbicas e funções de subtracção, adição, intersecção, união de planos, entre outras.
- Iluminação do mapa do mundo virtual – Vários tipos de luz podem ser colocados no mapa, mas é necessário ter em atenção que deve haver luz que esteja configurada para afectar objectos dinâmicos.
- Colocar localizações iniciais para o robô – No total podem ser colocadas dezasseis localizações iniciais, só uma é essencial para correr o mapa na aplicação desenvolvida. No entanto, ao se fazer a compilação do mapa com menos do que dezasseis localizações iniciais de jogador, a sua compilação dá erro e esse erro deve ser ignorado.
- Criação/Importação de texturas – A criação de texturas pode ser feita em qualquer programa de desenho, mas para que a sua importação tenha sucesso há duas condições: as dimensões da imagem têm de ser potencias de dois e o seu formato pode ser '.bmp', mas é mais usual o formato '.tga' (correspondente a um mapeamento de luminosidade e textura).
- Criação/Importação de materiais – A criação de materiais pode ser feita ao se seleccionar uma textura a partir da qual se deseja criar um material (só um material pode ser aplicado a uma qualquer superfície), depois de criado o material é necessário abrir o editor de material e configurar o mesmo. Para importar um material é necessário que este esteja no formato de importação de objectos '.T3D', mas é igualmente preciso ter as texturas previamente importadas.
- Aplicação de texturas/materiais aos blocos principais do mapa – Como referido anteriormente só um material pode ser atribuído a uma qualquer superfície, para o atribuir a um bloco principal do mapa (por exemplo uma parede) basta seleccionar a superfície e aplicá-lo, em seguida é conveniente abrir o menu de propriedades da superfície para adaptar e configurar a aplicação do material à mesma.
- Modelação gráfica de objectos da cena – Todos os objectos de cena são modelados graficamente recorrendo a plataformas de modelação a três dimensões que não o Unreal Editor, as suas texturas têm de estar em ficheiros de imagem separados para importação separada das mesmas.
- Importação de objectos de cena – Os objectos colocados em cena têm de estar num formato suportado pelo editor de Unreal. Uma plataforma de modelação especialmente útil na conversão de formatos, alteração de orientação e alteração de escala dos objectos para o Unreal Editor é o Blender, pois permite importar grande parte dos formatos de modelação gráfica a três dimensões e, em conjunto com um script em Python é possível exportar esses mesmos objectos para o formato '.ase' facilmente importado pelo Unreal Editor. Outra forma de importação de objectos de cena consiste em recorrer ao Unreal Editor UDK para importação de ficheiros no formato '.fbx'.
- Aplicação de materiais aos objectos da cena – A aplicação de materiais aos objectos de cena é realizada através do editor de objecto do Unreal Editor, após criação e configuração do material em questão.
- Aplicação de limites de colisão aos objectos da cena – A aplicação de propriedades de colisão do objecto pode ser aplicada no Blender ou no Unreal

Editor, caso se use o último, os limites de colisão são aplicados recorrendo à janela de edição de objecto. Para aplicar os limites de colisão estão disponíveis cinco funções automatizadas, a última contêm três parâmetros configuráveis. Através da visualização dos limites de colisão é possível compreender se estes limites estão bem calculados.

- Aplicação de características físicas aos objectos da cena – Podem ser criados modelos físicos dos objectos de cena, como por exemplo atribuir um dado factor de quebra a um qualquer osso do objecto que indica que o osso se vai partir aquando colisão de um objecto se a força exercida for superior à configurada.
- Garantir a correcta escala e iluminação do robô a ser utilizado pela aplicação desenvolvida – Para garantir a correcta escala e iluminação do robô deve-se importar o robô disponível no formato '.upk' e colocá-lo no mapa, em seguida adapta-se a escala e iluminação do mapa ao mesmo.

À excepção destas etapas é necessário referir que todas as texturas, materiais e objectos importados e incluídos no mundo virtual devem ser colocados no pacote de nome sugerido pelo Unreal Editor, correspondente ao nome do mapa em questão. Outra particularidade a ter em conta é que apenas os objectos/materiais/texturas importados, e incluídos no mapa aquando gravação do mesmo, ficam gravados nesse pacote permanentemente. Por outras palavras, se durante a modelação do mapa se incluírem texturas e criarem materiais e estes não forem aplicados a objectos presentes do mapa, fechando o Unreal Editor e reabrindo-o, estes terão desaparecido.

A figura 4.6.1.4 corresponde a um ponto de vista do mapa criado onde se pode visualizar o pormenor de uma cadeira do tipo espreguiçadeira, desde a complexidade da sua forma ao pormenor da sua textura. Este modelo a três dimensões e textura importada foram descarregados de dois repositórios distintos referenciados na secção 4.6.3.

Figura 4.6.1.4 Ponto de vista do mapa da FEUP incluindo pormenor de cadeira

Na figura acima (4.6.1.4) também é possível visualizar-se o pormenor de uma mesa, sua textura e texturas de paredes, tecto e chão do mapa.

4.6.2 Modelação da Cadeira de Rodas Virtual

Nesta aplicação foi utilizado o robô P3AT disponibilizado pelo simulador robótico USARSim e um modelo simples de cadeira de rodas para teste do comportamento robótico de uma cadeira de rodas virtual.

Os robôs modelados para o USARSim, como é o caso do robô P3AT (utilizado no teste da aplicação desenvolvida aquando a sua implementação), têm como limite de colisão caixas em forma de paralelepípedo, cada caixa corresponde a um osso do esqueleto do robô. O modelo de colisão inicial do P3AT está visível na figura 4.6.2.1 e corresponde ao modelo de colisão construído por defeito através do editor do motor de jogos Unreal Engine 3 aquando construção do modelo físico. A janela de edição do modelo físico permite mover, escalar e rodar estas caixas de colisão. Por outro lado, a janela de edição da malha representativa do robô permite alterar o modelo de colisão previamente adoptado, para um modelo de colisão multi-polígono. O modelo de colisão multi-polígono do robô P3AT está representado na figura 4.6.2.2 e corresponde a uma malha adjacente a todas as faces do mesmo.

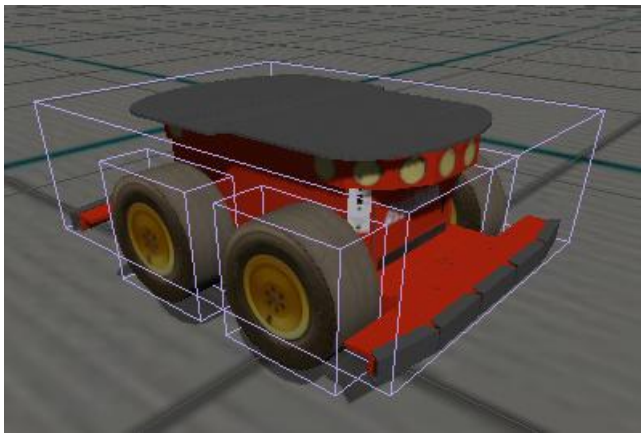


Figura 4.6.2.1 Modelo de colisão inicial do Robô P3AT [68]

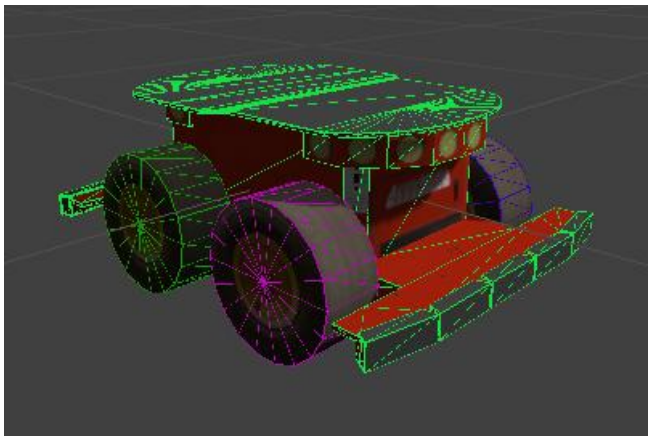


Figura 4.6.2.2 Modelação de colisão *"per-poly collision"* do Robô P3AT [68]

A modelação de robôs compatíveis com o simulador robótico USARSim e motor de jogos Unreal pode ser dividida nas seguintes tarefas:

1. Construção do modelo gráfico a três dimensões (3dsMax, Maya ou XSI) que corresponde à construção dos vários segmentos geométricos constituintes do robô;
2. Construção do esqueleto da malha que corresponde à construção dos ossos constituintes do esqueleto do robô e sua atribuição à "pele" de cada uma das malhas constituintes do mesmo;

3. Aplicação de texturas/materiais (aplicação de *UV mapping*), esta aplicação pode ser realizada ainda perante os programas de modelação gráfica ou só posteriormente no editor do motor Unreal (também é possível criar animações nesta fase de modelação e importá-las posteriormente para o editor de Unreal);
4. Exportação para FBX ou PSK e PSA, se se estiver a usar o editor correspondente ao jogo Unreal Tournament 3 é necessário usar o *plugin* ActorX para 3dsMax, Maya ou XSI que permite exportar a malha com esqueleto para o formato '.psk' e as animações construídas para o formato '.psa'; se se estiver a usar o editor do UDK, exporta-se a malha e as animações para o formato '.fbx' sem ser necessário nenhum *plugin* adicional;
5. Importação para Unreal Editor que consiste em importar a malha com esqueleto previamente exportada nos formatos '.fbx' ou '.psk', conforme o editor em questão, através da janela de visualização de conteúdos;
6. Aplicação de Material que consiste em importar as texturas necessárias, criar ou importar os materiais necessários e aplicá-los ao robô;
7. Construção da árvore de animação do robô;
8. Construção do modelo físico do robô, após a criação das caixas de colisão não deve ser necessário ajustá-las, caso seja o esqueleto não é o mais adequado;
9. Construção do modelo de colisão do robô que consiste em adicionar o nome dos ossos do esqueleto do robô ao campo denominado "*per-poly collision*" na janela de edição da malha;
10. Modelação mecânica do robô (construção da classe do robô em unreal script).

No decorrer do desenvolvimento da aplicação foi modelada uma cadeira de rodas virtual seguindo os passos anteriormente enumerados, a modelação gráfica da mesma foi feita recorrendo ao 3dsMax 2012. Nas figuras 4.6.2.3a e 4.6.2.3b pode-se visualizar o modelo gráfico concebido, bem como, respectivamente, o modelo físico e de colisão da cadeira. Na figura 4.6.2.4 apresenta-se a árvore de animação desenhada para o modelo de cadeira de rodas virtual correspondente.

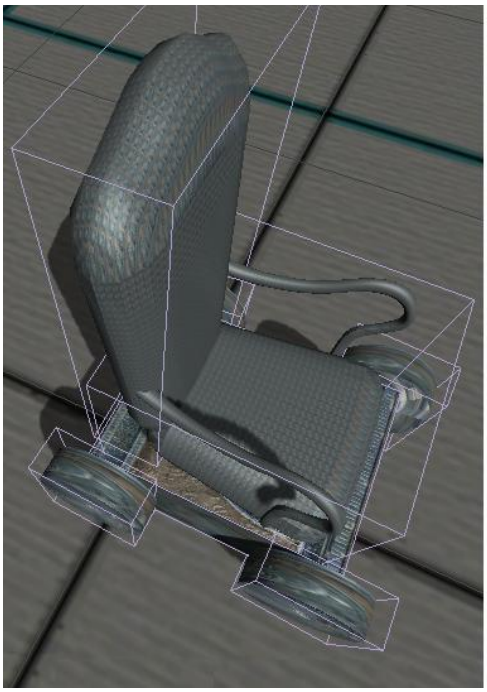


Figura 4.6.2.3a Modelo Físico da Cadeira de Rodas Virtual ("Wheelchair")

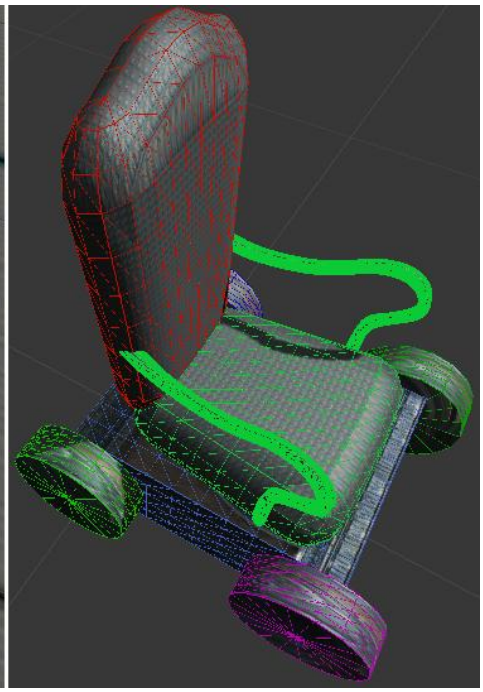


Figura 4.6.2.3b Modelo de Colisão da Cadeira de Rodas Virtual ("Wheelchair")

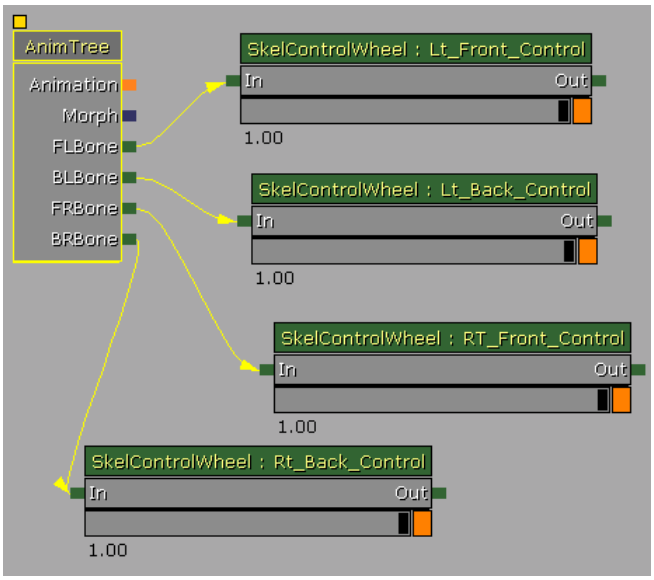


Figura 4.6.2.4 Árvore de Animação da Cadeira de Rodas Virtual ("Wheelchair")

Na figura acima (figura 4.6.2.4) está presente a árvore de animação criada para a classe de robô implementada, denominada "Wheelchair". Os ossos foram colocados exactamente no centro das rodas com o sentido da traseira da roda para a frente, em seguida, foram ligados ao osso principal da cadeira, a pele da respectiva malha foi criada e, por fim, os ossos foram adicionados à pele de cada malha editável (sendo cada uma representativa de cada roda).

A ordem das tarefas enunciadas acima é muito importante para a correcta modelação do esqueleto de qualquer robô. Por exemplo, no caso de desconectar um qualquer osso do osso principal, se eliminar o osso da pele da respectiva malha, se mover o osso e voltar a ligar tudo; a imagem até pode parecer estar correcta, mas não estará. O que se confirmará aquando importação para o editor do motor de jogos, apenas pelo simples facto de não se ter eliminado a pele e se ter reaproveitado a mesma para readição do mesmo osso.

Outra norma que se deve seguir na modelação do esqueleto é construir-se sempre de forma unitária e directamente no plano que permite colocá-lo já com a orientação correcta. Por outras palavras, o melhor é construir-se um osso de cada vez. Não vários encadeados e, posteriormente, desligá-los uns dos outros (algo que acontece por defeito na construção de ossos em 3dsMax). Por exemplo, no caso de se construir os ossos todos a partir do centro, inicialmente ligados, por mais que se desligue os ossos do osso principal, os mova e oriente de acordo com o pretendido, o referencial do osso de cada roda não irá corresponder à posição do próprio, mas sim à posição central da ligação de cada osso roda ao osso principal.

Em relação à orientação do osso (objecto com referencial interno), não esquecer que o ambiente de desenvolvimento é a três dimensões e o osso por defeito é simétrico ao eixo onde foi alongado. Logo, o osso pode parecer ter a orientação correcta a olho nu, mas na realidade estar rodado cento e oitenta graus sob um plano e cento e oitenta graus sob outro, aparentando visualmente uma correcta construção.

A orientação dos ossos deverá ser tal que o eixo X aponte ao longo e para o fim do osso, o eixo Z aponte para cima e o eixo Y aponte para a direita, do ponto de vista de quem está sentado na cadeira de rodas. Esta orientação tem também de se reflectir no osso principal e no corpo da cadeira.

Na construção do esqueleto de qualquer veiculo, todos os ossos colocados têm de descender de um único osso principal para que o veículo seja compatível com o motor de jogos Unreal Engine 3.

Nas figuras 4.6.2.5 e 4.6.2.6 são apresentadas as correctas orientações dos ossos responsáveis pela animação das rodas, através da demonstração do referencial de uma roda da frente, contextualizada no corpo de uma cadeira de rodas.

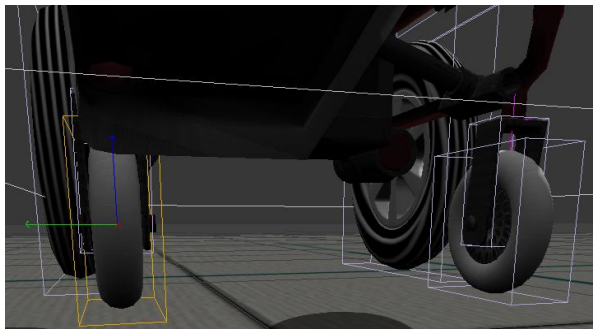


Figura 4.6.2.5 Referencial do osso responsável pela Animação da Roda (1-2)

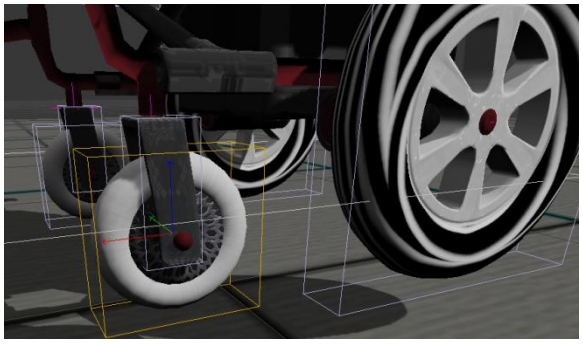


Figura 4.6.2.6 Referencial do osso responsável pela Animação da Roda (2-2)

Após a detecção da causa das contrariedades que surgiram, no decorrer da modelação da cadeira de rodas simples "Wheelchair", considerou-se desenvolver um modelo mais complexo de cadeira de rodas que fosse mais similar à cadeira de rodas eléctrica do projecto 'IntellWheels'. Esta modelação não foi realizada do zero, mas sim a partir de um modelo em 3dsMax que correspondia ao modelo exacto da cadeira de rodas real do projecto principal. O resultado final é apresentado na figura 4.6.2.7.

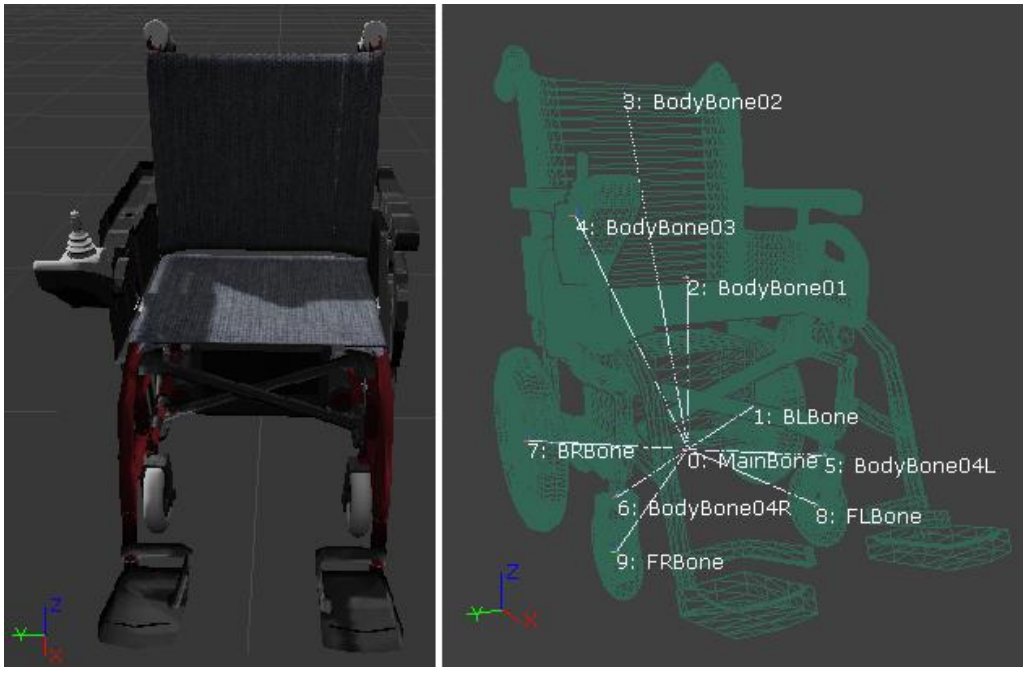


Figura 4.6.2.7 Cadeira de Rodas Eléctrica Virtual ("ElectricWheelchair")

Este modelo de cadeira de rodas difere da anterior no sentido em que as rodas já não chocam com o corpo principal do robô, corpo este que é considerado aquando a detecção de colisão do objecto. Como tal seria de esperar que a animação das rodas aquando viragem decorresse sem qualquer problema.

Como neste modelo mais realista as rodas da frente têm os seus respectivos apoios, a árvore de animação difere do modelo anterior, no ponto de vista em que, efectivamente, as rodas continuam a girar para a frente e para trás sob o plano XOZ sem incluírem os apoios nessa animação, mas o seu movimento de rotação sob o eixo Z tem de ser acompanhado pelos respectivos apoios. Para declarar o acompanhamento dos apoios quando as rodas rodam no eixo Z, movimento este visualizado aquando viragem da cadeira, é necessário criar um módulo de controlo do tipo "SkelControl_LookAt" para cada apoio (ou seja, para cada osso anteriormente colocado no centro de rotação de cada apoio).

Na figura 4.6.2.8 pode-se visualizar a árvore de animação construída para o modelo de robô denominado ("ElectricWheelchair").

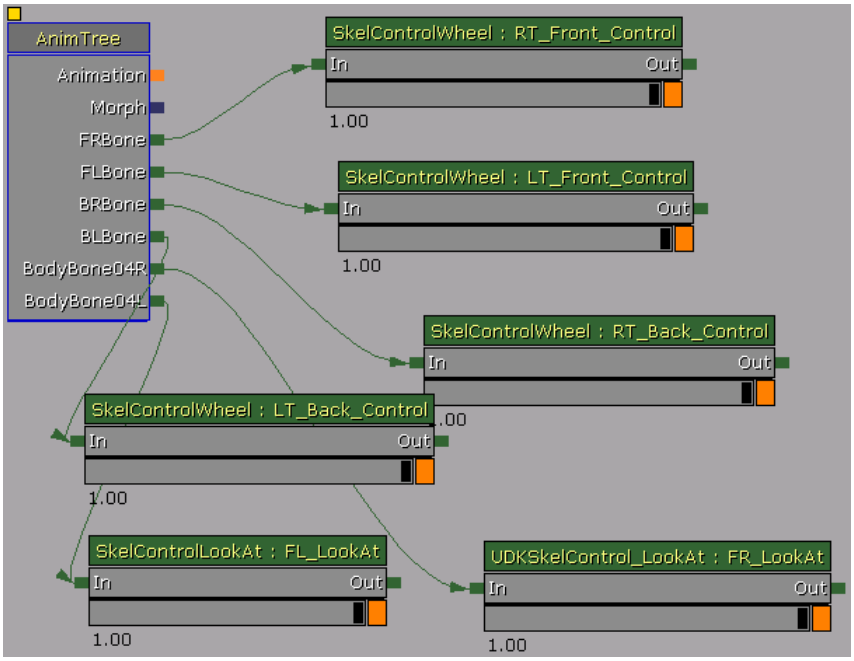


Figura 4.6.2.8 Árvore de Animação da Cadeira de Rodas Virtual ("ElectricWheelchair")

Este modelo de cadeira de rodas virtual foi construído na tentativa de aumentar o realismo de simulação da aplicação desenvolvida.

4.6.3 Plataformas de Desenvolvimento

Nesta secção são referidas as plataformas de desenvolvimento utilizadas, bem como os formatos de objectos modelados passíveis de serem importados para as mesmas. Para a modelação do mapa criado para teste da aplicação desenvolvida foram importados objectos a três dimensões não desenvolvidos pelo próprio adquiridos no formato "*.3ds*", correspondente a objectos modelados recorrendo ao *3DStudio Max*. Estes e outros elementos gráficos descarregados de diferentes repositórios e utilizados na população do presente mapa de teste não têm como propósito fins comerciais, nem devem ser usados com esse propósito.

O Unreal Editor, plataforma de desenvolvimento disponibilizado pelo motor de jogos Unreal, foi a plataforma de modelação do mundo virtual mais utilizada. Esta plataforma tem uma curva de aprendizagem bastante acentuada por ser um editor gráfico com sub-editores

no seu interior. Por outras palavras, cada tipo de objecto tem um editor a ele associado, desde o editor de textura, ao editor de material, ao editor de objecto, ao editor de animação, entre outros, sendo difícil a percepção da existência dos mesmos. Alguns dos formatos de ficheiros que esta plataforma suporta são os seguintes:

- ASC – Formato de ficheiro gráfico 3D criado pelo 3D Studio Max;
- ASE – Diminutivo de ASCII Scene Exporter;
- DXF – Ficheiro gráfico de imagem 3D originalmente criado por AutoDesk;
- LWO – Formato do programa LightWave;
- T3D – Ficheiro de texto que armazena uma lista de objectos mapeados em Unreal;
- PSK, PSA e FBX (este apenas no editor UDK) – Ficheiros de malhas com esqueleto.

A plataforma de desenvolvimento Blender é uma plataforma de criação de objectos a três dimensões muito poderosa que permite a execução de *scripts* em Python. Esta plataforma permite a importação de imagens a duas e três dimensões nos seguintes formatos:

- TGA, JPG, OpenEXR, DPX, Cineon, Radiance HDR, Iris, SGI Movie, IFF, AVI, Quicktime, GIF, TIFF, PSD, MOV (Windows e MacOS X);
- 3D Studio Max, AC3D, COLLADA, FBX Export, DXF, Wavefront OBJ, DEC Object File Format, DirectX, Lightwave, MD2, Motion Capture, Nendo, OpenFlight, PLY, Pro Engineer, Radiosity, Raw Triangle, Softimage, STL, TrueSpace, VideoScape, VRML, VRML97, X3D Extensible 3D, xfig export.

Em conjunto com o *script* em Python é possível exportar-se qualquer objecto a três dimensões para o formato '.ase' facilmente importado para o editor do Unreal e é nesse contexto que esta plataforma tem interesse acrescido para a aplicação desenvolvida.

Outras ferramentas de modelação gráfica a três dimensões podem ser utilizadas para criar objectos de cena desde que o formato final ou exportado esteja enunciado na lista acima. A textura do objecto deve estar contida num ficheiro separado para que a sua aplicação ao objecto seja possível no editor do Unreal. Alguns dos repositórios de imagens a duas e três dimensões existentes e utilizados na construção do mapa criado para teste da aplicação foram o TurboSquid [70], o Mayang's Free Textures [71], o CG Textures [72] e o Bishop Digital Image [73].

Concluindo, o editor de Unreal serve para criar, colorir e montar o mundo virtual e o Blender é bastante útil como intermediário na importação de objectos ou ferramenta de criação dos mesmos. Na modelação gráfica e estrutural dos dois modelos de cadeiras de rodas virtuais criados/adaptados tirou-se partido do ambiente de modelação *3DStudio Max*.

4.7 Modelação do comportamento robótico

Esta secção pretende explicitar as características gerais relativas à construção do comportamento robótico da cadeira de rodas virtual, o estudo de possibilidades de implementação da modelação robótica e a análise das possibilidades consideradas.

Para se compreender o comportamento robótico é necessário avaliar o comportamento mecânico de uma cadeira de rodas eléctrica real e as possibilidades de implementação promovidas pelo simulador robótico USARSim.

Na realidade uma cadeira de rodas eléctrica como a utilizada para teste da plataforma 'IntellWheels' tem dois tipos de liberdade de movimento de viragem: virar sob uma das rodas, mantendo uma das rodas sensivelmente no mesmo ponto de contacto do chão (dependendo

do atrito); e virar girando as duas rodas traseiras em sentidos opostos, o que provoca um movimento de viragem de raio maior.

Em relação à possibilidades de implementação promovidas pelo USARSim optou-se por usar os comandos de condução disponíveis correspondentes ao controlo de robôs do tipo "SkidSteeredRobot" que permitem explicitar a força a aplicar a cada lado do robô.

O modelo de cadeira de rodas virtual foi construído considerando que a mesma deveria ter dois motores, aplicando cada qual a cada roda traseira da mesma, e que as rodas da frente não deveriam ter motor. Também se considerou que as rodas frontais deveriam ser livres em qualquer tipo de movimento e que as rodas traseiras deveriam ser fixas no contexto de movimento da cadeira aquando viragem.

Para facilitar o desenvolvimento das classes em unreal script dos dois modelos de cadeiras de rodas implementados, estas foram inicialmente derivadas das classes de dois robôs já implementados no USARSim. Estas classes são responsáveis pela atribuição de características mecânicas imprescindíveis à simulação de qualquer tipo de veículo do Unreal Engine 3.

Os dois robôs adoptados como exemplo foram o P3AT e o P2DX, este último já atribui às rodas da frente todo o grau de liberdade necessário à simulação realista de movimento das mesmas. A principal diferença entre as duas classes de robôs consideradas como exemplo é o facto do primeiro modelo de robô ser uma subclasse de "SkidSteeredRobot" e não identificar explicitamente as *joints* do seu corpo e o segundo já o explicitar. Actualmente o robô P2DX ainda não foi portado para o USARSim versão UDK, como tal, para apresentação no presente documento foi escolhida a abordagem por semelhança ao robô P3AT.

Na figura 4.7.1 e 4.7.2 pode-se visualizar, respectivamente, as características iniciais e finais, de configuração do modelo mais simples de cadeira de rodas virtual.

As características iniciais de configuração correspondem à atribuição da malha esqueleto do robô, da árvore de animação, do corpo físico, entre outras. Algumas propriedades do robô são declaradas externamente a estas configurações, no ficheiro de configurações iniciais do USARSim: 'maxFrontSteerAngle'; 'maxRearSteerAngle'; 'MaxTorque'; 'MaxDSpeed'; 'PGain'; 'IGain'; e 'DGain'. Esta deslocação pode gerar confusão porque se tem de ter em conta as propriedades do nosso robô e de quem ele descender.

```
//////////////////////Mesh, Animation, Physics & Collision  parameters///////

Begin Object Name=SVehicleMesh
  SkeletalMesh=SkeletalMesh'Wheelchair.SkeletalMesh.Wheelchair'
  AnimTreeTemplate=AnimTree'Wheelchair.SkeletalMesh.Wheelchair_AnimTree'
  PhysicsAsset=PhysicsAsset'Wheelchair.SkeletalMesh.Wheelchair_Physics'
  BlockRigidBody=true
  RBChannel=RBCC_GameplayPhysics
  RBCollideWithChannels=(Default=TRUE,GameplayPhysics=TRUE,EffectPhysics=TRUE)
  End Object

  Begin Object
  Class=DynamicLightEnvironmentComponent
  Name=MyLightEnv01
  ObjName=DynamicLightEnvComponent_01
  End Object

  Components(0)=MyLightEnv01
  Components(1)=SVehicleMesh

  Mass=100;  // From Pawn Class
  COMOffset = (x = 0.0, y = 0.0, z = 0.0)
  InertiaTensorMultiplier=(x=1,y=1,z=1);
  //MaxSpeed = 20;
  bCanFlip=True
```

Figura 4.7.1 Início da configuração do modelo "Wheelchair" (extracto de unreal script)

As configurações finais correspondem à criação do objecto "SimObject" da classe "SVehicleSimBase" e respectivos atributos, para compreender melhor estes atributos pode-se consultar a página *online* de guia técnico de veículos para Unreal Engine 3 em [76]. Em

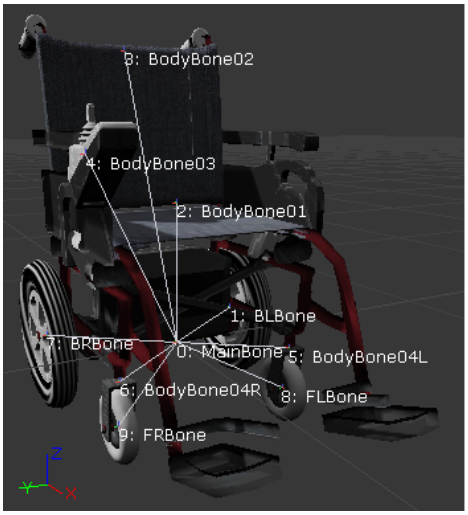
relação à definição destas últimas configurações, seria interessante explorar com maior profundidade a modificação do comportamento da cadeira aquando alteração dos valores dos diferentes parâmetros.

```
Begin Object Class=SVehicleSimBase Name=SimObject ObjName=SimObject
        Archetype=SVehicleSimBase'Engine.Default__SVehicleSimBase'
    bTurnInPlaceOnSteer=True
    //bClampedFrictionModel = True
    //bWheelSpeedOverride=True
    WheelLongExtremumSlip=0.001
    WheelLongExtremumValue=0.40
    WheelLongAsymptoteSlip=0.002
    WheelLongAsymptoteValue=0.20
    WheelLatExtremumSlip=0.005
    WheelLatExtremumValue=0.07
    WheelLatAsymptoteSlip=0.007
    WheelLatAsymptoteValue=0.05
    WheelInertia=1.00
End Object

SimObj=SimObject
```

Figura 4.7.2 Fim da configuração do modelo "Wheelchair" (extracto de unreal script)

Em relação à programação das rodas e seus atributos, pode visualizar-se na figura 4.7.3, a cadeira de rodas virtual (modelo "ElectricWheelchair"), seu esqueleto e valores atribuídos às diferentes propriedades de configuração das rodas do lado direito (frente e traseira). Tal como visível, é necessário identificar para cada roda (todas descendem da classe "SVehicleWheel") o nome do osso correspondente e o nome do controlo de esqueleto previamente construído na árvore de animação.




```
Begin Object Class=SVehicleWheel Name=RFWheel ObjName=RFWheel
    //Steer=0.00;
    MotorTorque=0.00
    BrakeTorque=0.00
    ChassisTorque=0.00
    bPoweredWheel=False
    SteerFactor=1.0
    LongSlipFactor=4000.000000
    LatSlipFactor= 20000.00
    SkelControlName="RT_Front_C
    BoneName="FRBone"
    WheelRadius=15
    SuspensionTravel=1.000000
//  SuspensionSpeed=40.00
    Side=SIDE_Right
    Name="RFWheel"
End Object
Wheels[1]=RFWheel
```

```
Begin Object Class=SVehicleWheel Name=RR
    //Steer=0.00;
    MotorTorque=120.00
    BrakeTorque=0.00
    ChassisTorque=10.00
    bPoweredWheel=True
    SteerFactor=0.0
    LongSlipFactor=4000.000000
    LatSlipFactor= 20000.00
    SkelControlName="RT_Back_Control"
    BoneName="BRBone"
    WheelRadius=40
    SuspensionTravel=1.000000
//  SuspensionSpeed=40.00
    Side=SIDE_Right
    Name="RRWheel"
End Object
Wheels[3]=RRWheel
```

Figura 4.7.3 Configuração das rodas da "ElectricWheelchair" em unreal script

O parâmetro 'Side' que identifica o lado da roda relativo ao corpo do robô apenas é importante por se estar a configurar um robô que descende da classe do USARSim "SkidSteeredRobot" que necessita saber o lado para efeitos de condução da mesma.

Nas rodas da frente 'bPoweredWheel' foi colocado a zero porque não deve ser aplicada energia às mesmas e 'SteerFactor' foi colocado a um, o que significa que as rodas rodarão de acordo com o movimento de viragem da cadeira no sentido do mesmo. Às rodas traseiras é aplicada a energia e estas não devem rodar ('SteerFactor'=0.0). O parâmetro 'WheelRadius' é

onde se configura o raio das rodas, na realidade não é necessário ter grande precisão, apenas um valor próximo da realidade.

Para se modelar o comportamento robótico de movimentação convém ter-se noção dos parâmetros de configuração da hierarquia do robô. Embora esta tenha sido implementada de forma a facilitar a modelação do veículo, pode, igualmente, acarretar dificuldades. Principalmente aquando tentativa de resolução de problemas de movimentação causadas por atributos cujo valor por defeito impede a correcta simulação de movimento. Para dar ao leitor uma leve noção da hierarquia de construção de veículos foi desenhado o esquema presente na figura 4.7.4 que tenta demonstrar de forma simplificada a hierarquia de veículos proporcionados pelo simulador robótico em conjunto com o motor de jogos.

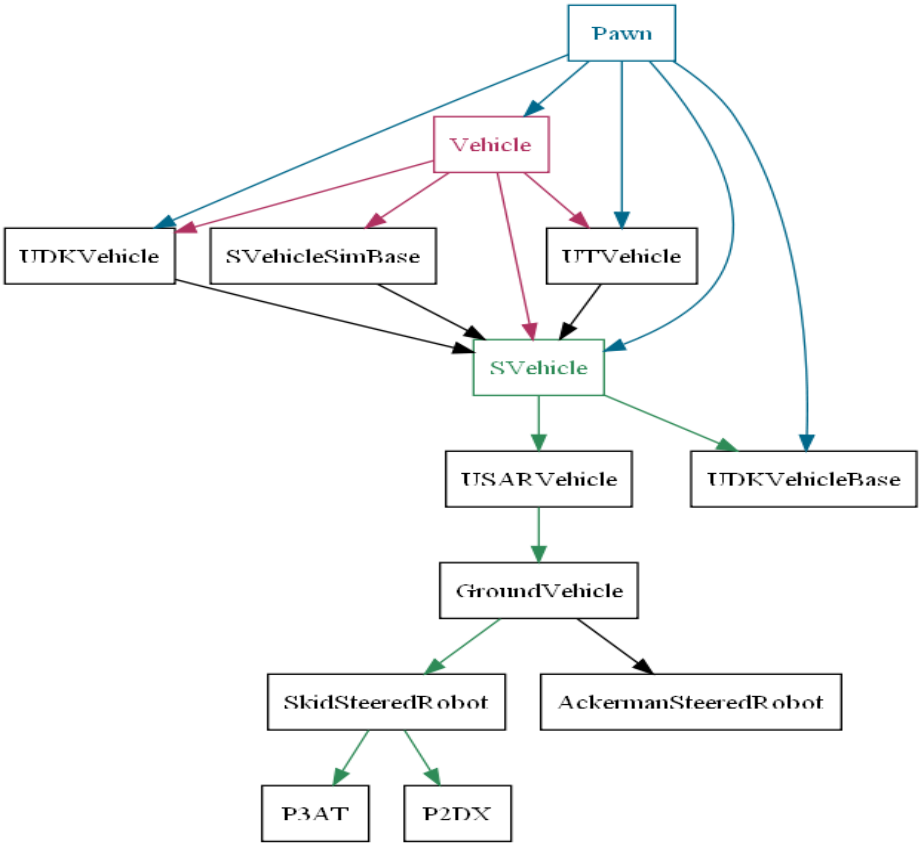


Figura 4.7.4 Esquema da Hierarquia dos veículos de USARSim e Unreal Engine 3

4.8 Interligação com o Projecto 'IntellWheels'

Esta secção englobada no capítulo de implementação pretende descrever todo o processo de ligação da aplicação ao módulo de controlo de cadeira de rodas virtual do projecto 'IntellWheels'. Efectivamente o trabalho desenvolvido comunica com o módulo de controlo, consegue enviar os valores dos sensores e receber tanto o registo inicial da cadeira de rodas, como os valores de energia a aplicar aos motores da mesma.

O protocolo de comunicação do módulo de controlo e o anterior simulador baseia-se em duas normas rígidas: sendo o simulador um simulador a duas dimensões, o módulo de controlo apenas indica valores de posição no eixo X e Y; e esse mesmo módulo regista a sua cadeira de rodas no simulador, não permitindo o oposto.

Na próxima figura (figura 4.8.1) é possível visualizar-se um esquema representativo da comunicação módulo de controlo e módulo de simulação, a troca de mensagens entre os módulos assenta em *sockets* UDP, o registo da cadeira de rodas só ocorre uma vez, a partir desse momento, a troca de mensagens (MedidasSensores e Actions) funciona ciclicamente. O protocolo de comunicação referente ao módulo de visualização não é referido porque, para a aplicação desenvolvida, o módulo de visualização corresponde à visualização gerada pelo motor de jogos.

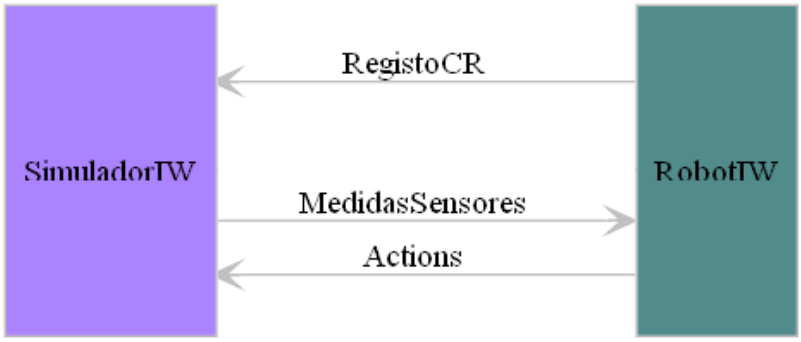


Figura 4.8.1 Protocolo de Comunicação de IntellWheels (módulo de controlo e módulo de simulação)

O protocolo de comunicação foi implementado à semelhança, na aplicação desenvolvida, mas as características da cadeira de rodas que o controlador 'IntellWheels' tenta registar são na grande maioria ignoradas porque o robô já foi configurado e o USARSim já foi compilado à priora. Implementou-se este protocolo de comunicação na aplicação desenvolvida para se provar conceptualmente que era possível ligar as duas plataformas e transmitir os valores de sensores.

Este protocolo tem de ser adaptado às plataformas existentes actualmente, deveria haver de alguma forma colaboração na configuração do robô ou configuração do robô a nível da aplicação desenvolvida e registo da cadeira de rodas virtual no controlador.

Outra medida interessante seria criar uma ligação por TCP socket para cada um dos robôs, e enviar os valores de sensores e comandos de locomoção apenas quando esses valores mudassem. Não deveria ser uma socket UDP para toda a informação.

Por outro lado, seria interessante criar um *schema* para as mensagens XML transmitidas entre as duas plataformas de forma a validar automaticamente a sua forma e conteúdo, o que melhoraria a compreensão das mensagens trocadas.

```
<Robot Name="WheelChair" Type="Simulated" Height="0.66" Width="1.04"
     COMass="0.81" MaxSpeed="0.5" AccelerationCurve="0.83">
        <IRSensor Id="0" X="0.72" Y="0.24" Angle="0.0" Cone="60"/>
        <IRSensor Id="1" X="0.72" Y="-0.24" Angle="0.0" Cone="60"/>
        <IRSensor Id="2" X="0.54" Y="0.24" Angle="90" Cone="60"/>
        <IRSensor Id="3" X="0.54" Y="-0.24" Angle="-90" Cone="60"/>
        <IRSensor Id="4" X="0.05" Y="0.28" Angle="90" Cone="60"/>
        <IRSensor Id="5" X="0.05" Y="-0.28" Angle="-90" Cone="60"/>
        <IRSensor Id="6" X="-0.03" Y="0.28" Angle="180" Cone="60"/>
        <IRSensor Id="7" X="-0.03" Y="-0.28" Angle="180" Cone="60"/>
</Robot>
```

Figura 4.8.2 Exemplo de Mensagem - Registo de cadeira virtual (protocolo de comunicação)

```
<Actions LeftMotor="0" RightMotor="0"/>
```

Figura 4.8.4 Exemplo de Mensagem - Comando de condução (protocolo de comunicação)

Nas figuras 4.8.2-4 são apresentados exemplos de mensagens trocadas entre os dois módulos: a primeira imagem corresponde à mensagem de registo da cadeira de rodas virtual; a segunda imagem corresponde à mensagem que deve ser enviada pelo simulador, com os

valores dos sensores e respectivo *timestamp*; e a terceira imagem demonstra o comando de condução enviado pelo controlador.

```
<Measures Time="0">
 <Width="1.04" Heigth="0.66" CenterOfMass="0.81" Type="Simulated">
   <Sensors Collision="No" Compass="0" Ground="-1">
     <IRSensor Id="0" Value="0.223214", X="6.72" Y="11.24" Dir="0", Cone="60"/>
     <IRSensor Id="1" Value="0.268646", X="6.72" Y="10.76" Dir="0", Cone="60"/>
     <IRSensor Id="2" Value="0.362319", X="6.54" Y="11.24" Dir="90", Cone="60"/>
     <IRSensor Id="3" Value="0.324675", X="6.54" Y="10.76" Dir="-90", Cone="60"/>
     <IRSensor Id="4" Value="0.367647", X="6.05" Y="11.28" Dir="90", Cone="60"/>
     <IRSensor Id="5" Value="0.47619", X="6.05" Y="10.72" Dir="-90", Cone="60"/>
     <IRSensor Id="6" Value="0.183824", X="5.97" Y="11.28" Dir="180", Cone="60"/>
     <IRSensor Id="7" Value="0.694444", X="5.97" Y="10.72" Dir="180", Cone="60"/>
     <Landmark Id="1" deltaX="0" deltaY="0" Angle="0"/>
     <BeaconSensor Id="0" Value="NotVisible"/>
     <GPS X="6" Y="11" Dir="0"/>
   </Sensors>
   <Leds EndLed="Off" ReturningLed="Off" VisitingLed="Off"/>
   <Buttons Start="Off" Stop="On"/>
</Measures>
```

Figura 4.8.3 Exemplo de Mensagem - Medições dos sensores (protocolo de comunicação)

Capítulo 5

Experimentação e Resultados

Neste capítulo são apresentadas as principais dificuldades de desenvolvimento em conjunto com os resultados obtidos, permitindo uma análise estruturada do contributo científico deste projecto. Este capítulo foi dividido em três áreas de foco: Simulação do comportamento robótico; Visualização do ambiente de simulação; e Módulo de controlo robótico 'IntellWheels'.

5.1 Simulação do Comportamento Robótico

Esta secção apresenta a definição e análise do estado actual referente à simulação do comportamento robótico que pode ser dividida em três tópicos: comunicação da aplicação com o simulador robótico USARSim e motor de jogos Unreal Engine; tratamento de colisões da cadeira de rodas virtual com o mundo simulado; e condução da cadeira de rodas inteligente.

5.1.1 Comunicação com o Unreal Engine 3 e USARSim

Relativamente ao protocolo de comunicação definido entre a aplicação desenvolvida e o simulador robótico em conjunto com o motor de jogos, foi possível verificar que todas as mensagens eram enviadas e seus comandos executados.

Inicialmente, a aplicação permite configurar um ou mais robôs com determinadas características e adicionar e remover sensores, especificando a sua localização e orientação (relativa ao centro do robô). Para proceder à configuração específica de cada sensor ainda é necessário que o utilizador edite o ficheiro de configurações iniciais do USARSim, esta operação de edição tem de ser executada antes do pedido de início de simulação proporcionado pelo botão 'Start' da janela de configuração 'Map'.

Por defeito, os sensores de tipo 'IRSensor' foram configurados com uma distância máxima de detecção equivalente a três metros, uma distância mínima de vinte centímetros e um ângulo de cone equivalente a vinte graus. Os sensores de tipo 'SonarSensor' foram configurados com uma distância máxima de detecção de cinco metros, uma distância mínima de dez centímetros e ângulo de cone equivalente a vinte graus.

Na figura 5.1.1 pode visualizar-se os extractos do ficheiro de configuração definido, correspondentes às configurações dos sensores do tipo 'IRSensor' e 'SonarSensor'. Para automatizar esta configuração de sensores bastaria implementar a funcionalidade do botão 'Advanced' de sensores da janela 'Robot Configuration'.

```
[USARSensor.SonarSensor]
bUseGroup=True
HiddenSensor=false
ScanInterval=0.2 ;seconds
MaxRange=5.0 ;meters
MinRange=0.1 ;meters
beamAngle=0.698 ;radians
maxAngleOfIncidence=0.1745329252 ;radians
numberOfCones=10
tracesPerCone=16
bSendRange=true
Noise=0.05
```

```
[USARSensor.IRSensor]
MaxRange=3.0 ;meters
MinRange=0.2 ;meters
beamAngle=0.349 ;radians
bSendRange=true
Noise=0.05
```

Figura 5.1.1 Configuração de IRSensor e SonarSensor

Depois de ser executada a acção de compilação do USARSim, os comandos responsáveis pelo pedido de posições iniciais do mapa, pelo *spawn* do robô no ambiente de simulação, pela condução do robô e pela recepção dos valores dos sensores ocorrem sem problemas.

5.1.2 Tratamento das Colisões

Na realidade, relativamente ao tratamento de colisões não foi necessário implementar nenhum algoritmo de raiz, pois o motor de jogos Unreal Engine 3 já trata as colisões por defeito. As únicas opções que podem modificar o corpo de colisão são a identificação do corpo físico do robô e a opção de usar *per-poly-collision* para cada osso do robô modelado.

Um problema identificado nos manuais do simulador robótico USARSim é o tratamento de colisão das rodas, por defeito o motor de jogos não liga à colisão das mesmas com qualquer objecto do mundo virtual, visto que este só tem em consideração a caixa de identificação do corpo principal do robô associado ao osso principal.

Os veículos modelados para o motor de jogos têm uma ou mais caixas de colisão que combinadas envolvem a totalidade do veículo, associadas como conjunto ao osso principal do mesmo (usualmente uma única caixa existe). Contraditoriamente, os robôs modelados para o simulador robótico têm um corpo físico definido como decomposição de caixas, cada caixa associada a um osso e respectiva malha.

Sendo uma questão de optar e ao se ter receio da caixa de colisão ser pouco realista, optou-se por manter a definição do corpo físico tal como o simulador robótico o promove e analisar a aplicação de colisão por polígono. No seguimento da configuração de colisão, aplicou-se per-poly-collision a todos os ossos das duas cadeiras de rodas modeladas e verificou-se visualmente a malha de colisão do conjunto.

De qualquer forma a correcta colisão entre a cadeira de rodas virtual e objectos do mundo, no presente motor de jogos, só é possível se os objectos criados e incluídos no mundo tiverem uma malha de colisão a eles associada, algo que foi possível provar ainda aquando modelação do mapa. Este teste foi realizado recorrendo à execução do mapa e tentativa de movimento através das setas do teclado, ao tentar passar pelos objectos incluídos no mundo virtual (bancos, secretárias, sofás, etc.), a continuação do movimento só era possível se não colidisse com os mesmos.

Em conclusão, o tratamento da colisão entre a cadeira de rodas virtual e os objectos de cena bem como o espaço do mundo virtual foi devidamente configurada e testada.

5.1.3 Condução da Cadeira de Rodas Inteligente

Os modelos de cadeira de rodas criados têm a desvantagem de apenas serem compatíveis com a versão de UDK de Fevereiro de 2011 e versões posteriores. Como tal, o robô P3AT que é compatível com o motor de jogos aquando lançamento do jogo foi utilizado para teste de condução de robô.

De acordo com os testes realizados nas duas versões de motor de jogos enunciadas em conjunto com as respectivas versões do simulador robótico USARSim, recorrendo a estes três veículos, concluiu-se que os problemas de movimentação de robô aquando viragem eram provocados pela ainda recente adaptação de USARSim lançada para UDK.

No motor de jogos versão de lançamento do jogo, o robô P3AT comporta-se da forma esperada, enquanto na versão UDK de Fevereiro tanto o robô P3AT e as cadeiras de rodas virtuais não rodam de acordo com o esperado nem em conformidade com o movimento realista.

No entanto, os comandos de condução são iguais nas duas versões de USARSim, embora os ficheiros de configuração do simulador robótico tenham sofrido grandes alterações. No tempo de desenvolvimento do projecto foram analisados os comentários de desenvolvimento dos responsáveis pela adaptação do simulador ao UDK e modificações que os próprios acharam mais importantes, contudo não foi encontrada solução para o problema.

Como ainda se trata de uma adaptação recente, espera-se que a adaptação do simulador que está, neste momento, em fase de desenvolvimento permita uma correcta movimentação de qualquer robô (aquando viragem) em breve.

Todas as configurações essenciais à correcta movimentação dos modelos de cadeira de rodas estão de acordo com as especificações tanto do simulador como do motor de jogos e, após várias tentativas de reconfiguração e até de compreensão e alteração dos ficheiros base do simulador, a tarefa de correcção do movimento de viragem da cadeira não foi concluída com sucesso.

Devido às grandes modificações dos ficheiros tanto de configuração base do simulador como do robô P3AT, as tentativas de reconfiguração acarretaram grande dificuldade.

5.2 Visualização do Ambiente de Simulação

Nesta secção é realizada a análise do ambiente de visualização da simulação robótica que foi estruturada da seguinte forma: visualização do mundo virtual e visualização do corpo e animação da cadeira de rodas virtual.

5.2.1 Mundo Virtual

Relativamente ao mundo virtual criado na semelhança com o corredor dos laboratórios de inteligência artificial, este resultou num ambiente graficamente rico. As tarefas essenciais para a sua modelação foram referidas no capítulo de implementação e considera-se que a reprodução desses passos facilitará a construção de mapas pelo utilizador da aplicação.

Como qualquer editor de mundos virtuais, os editores do motor de jogos implicam uma curva de aprendizagem muito acentuada, mas, depois da habituação e maturação do conhecimento prático, a construção de mapas torna-se fácil.

Depois do período inicial de aprendizagem foi possível verificar que o editor disponível pelo motor de jogos é uma ferramenta de modelação de mapas muito poderosa.

Uma opção de desenvolvimento considerada no âmbito deste projecto foi a automatização de construção de mapas, de preferência, a transformação dos mapas existentes para a plataforma 'IntellWheels' disponíveis em XML em mapas de *unreal*.

Aquando identificação dos ficheiros de importação e exportação de mapas e dos ficheiros de programação de mapas, concluiu-se que tal tarefa não poderia ser automatizada nas condições existentes actualmente.

Por um lado, os ficheiros de XML usados na declaração de mapas da plataforma 'IntellWheels' não estão acompanhados de um *schema*, por outro, partindo do principio que era possível gerar a programação de construção de mapa (formato '.t3d'), esta implicaria um esforço enorme de reestruturação do conteúdo em XML para uma estrutura completamente diferente. Isto, excluindo ainda o tempo de aprendizagem de uma outra linguagem de programação, habituação à mesma e, ainda, estudo da solução de compilação de uma estrutura para outra, com as devidas preocupações de ordenação e teste de parâmetros.

Ou seja, se futuramente se desejasse realizar tal tarefa, o melhor seria construir novos mapas em XML, acompanhados de *schema* da linguagem XML implementada, com uma estrutura mais semelhante à estrutura de programação de mapas do motor de jogos. Algo que acarretaria a mudança do *parser* interno do módulo de controlo de cadeiras de rodas inteligentes da plataforma 'IntellWheels'.

Outra opção de automatização de construções de mapas considerada foi a importação automática de mapas já em formatos de modelação a três dimensões, mas conclui-se que tendo um editor de jogos que permite de forma fácil importar um mapa acedendo a uma opção do menu principal, que pode de seguida compilá-lo e apresentar todo o tipo de erros do mapa e permitir ao utilizador corrigi-los, não teria qualquer interesse refazer o que o editor já permite, até porque tal como o motor de jogos, o editor também é gratuito.

5.2.2 Corpo e Animação da Cadeira de Rodas

Relativamente ao corpo e animação da cadeira de rodas virtual à excepção da animação de viragem das rodas frontais toda a animação ocorre de acordo com o esperado.

Na primeira cadeira de rodas denominada "Wheelchair" considerou-se que a causa provável das rodas da cadeira não rodarem é por estas colidirem com o corpo principal do veículo. Como o motor de jogos usa o corpo do chassis para as colisões poderia ser esta a causa, no entanto é estranho que este seja o motivo porque os atributos 'bCollideAll' dos ossos das rodas e do osso principal foram colocados a falso.

Para confirmar, após a construção do modelo mais realista de cadeira de rodas denominado 'ElectricWheelchair' foi testada a animação de viragem das rodas da cadeira. Embora, neste modelo, as rodas já não colidirem com o corpo principal do mesmo, visto que além das rodas já serem livres, o corpo físico principal já está representado por uma caixa que inclui todo o corpo da cadeira, a animação não corresponde ao esperado.

Outro modelo analisado foi um modelo simples adaptado do robô P2XT desenvolvido pelo laboratório de inteligência artificial compatível com a versão de simulador adaptado ao motor de jogos da versão de lançamento do jogo que efectua o movimento de viragem correcto, mas não se visualiza a animação de viragem das rodas.

Durante a simulação não foi possível visualizar a animação de viragem de rodas, no entanto, durante a configuração dos parâmetros de animação, é possível verificar visualmente essa mesma animação

5.3 Módulo de Controlo Robótico IntellWheels

Nesta secção pretende-se sintetizar as principais dificuldades de integração e incompatibilidades da aplicação desenvolvida e o controlador de cadeira de rodas inteligente da plataforma 'IntellWheels'.

Tal como referido na secção corresponde à integração do módulo de controlo 'IntellWheels' do capítulo de implementação, embora se tenha integrado as duas plataformas, essa integração não corresponde a uma solução final mas, apenas, a uma prova de conceito que pretende demonstrar a possibilidade de integração das duas plataformas. Nessa secção foi referido o protocolo de comunicação em vigor actualmente e as incompatibilidades das mensagens trocadas entre as duas plataformas.

A comunicação não deveria ser por *sockets* UDP nem tratada de forma sincronizada, ou seja, neste momento as mensagens recebidas e enviadas são mantidas na porta respectiva segundo uma dada ordem e as aplicações esperam pela próxima ordem para executarem as suas acções. Este tipo de configuração da comunicação sincronizada implica que as mensagens sejam lidas numa dada ordem e que o envio da próxima mensagem só seja efectuado aquando recepção de uma dada mensagem.

Um sistema como o implementado que trata do controlo de uma cadeira de rodas, corresponde a um sistema crítico, logo tem de ser tolerante a falhas. Como tal, é essencial estudar um protocolo de comunicação que se adapte às duas aplicações.

Neste momento, o registo da cadeira não é realizado antes de correr o simulador USARSim em conjunto com Unreal, logo as configurações de robô previamente definidas não podem ser alteradas e a mensagem é ignorada. Esta funcionalidade pode ser facilmente implementada na aplicação desenvolvida para se permitir a configuração da cadeira de rodas em colaboração com o módulo de controlo, ou ser a aplicação desenvolvida a enviar o registo das características da cadeira de rodas virtual.

As mensagens XML têm de ser adaptadas no módulo de controlo da plataforma 'IntellWheels' para que a aplicação desenvolvida possa receber as posições e orientações dos sensores a três dimensões em vez de as receber a duas dimensões.

O módulo de controlo de cadeira de rodas da plataforma do projecto 'IntellWheels' tem actualmente três modos de funcionamento: simulação, real e realidade aumentada. O modo de funcionamento escolhido para o teste do protocolo de comunicação foi o modo de simulação. O controlador interagia com o antigo simulador de forma idêntica quer em modo de realidade aumentada, quer em modo de simulação. Portanto, pode-se considerar que a integração das duas plataformas não depende do modo de operação do módulo de controlo.

Capítulo 6

Conclusão e Trabalho Futuro

Este capítulo tem como objectivo apresentar uma síntese do trabalho realizado e contextualizá-lo segundo uma perspectiva de análise geral, enquadrada nos objectivos considerados inicialmente.

No final deste capítulo são apresentadas perspectivas de desenvolvimento futuro com o intuito de promover a continuação do presente trabalho e apresentar uma visão mais ampla do projecto principal.

6.1 Síntese e Análise do Projecto

A meta principal do presente trabalho foi criar um simulador e visualizador de cadeira de rodas inteligente: estudar as soluções existentes; projectar os módulos de visualização gráfica e simulação virtual; e implementar um simulador robótico com inúmeras hipóteses de configuração e possibilidade de registo de vários robôs que se enquadrasse na plataforma 'IntellWheels'.

No decorrer deste trabalho foi desenvolvida uma aplicação que permite a configuração de um mapa para simulação, a configuração de um ou mais robôs com um ou mais sensores e a condução dos robôs previamente configurados. Também é responsável por evocar a visualização da simulação e gere todo o ambiente de simulação virtual. A aplicação desenvolvida permite, igualmente, a integração com o módulo de controlo de cadeira de rodas inteligente da plataforma 'IntellWheels'.

Para o desenvolvimento do projecto foi necessário:

- Aprendizagem de diferentes ferramentas (ferramentas de modelação 3D, diferentes sub-editores do editor de jogos UE3, configuração de parâmetros dos sensores e robôs do USARSim, linguagem de programação unreal script, bibliotecas de Qt, entre outras);
- Construção de dois modelos de cadeira de rodas (modelação gráfica, adaptação ao USARSim e configuração, modelação e programação para UDK), incluindo os seus modelos físicos, mecânicos e de animação;
- Construção do mapa virtual (modelação gráfica no editor de UE3 versão do jogo);
- Análise de Solução para geração automática de mapas;
- Importação de objectos de cena e aplicação de tratamento de colisões (Blender e editor de UE3);
- Comunicação entre a aplicação e o USARSim e UE3;
- Comunicação entre a aplicação e o módulo de controloIW (há recepção do registo da cadeira, mas a informação é descartada);
- Análise de Solução para a comunicação com o módulo de controloIW;

- Configuração automática de robôs e seus sensores;
- Leitura automática dos valores dos seus sensores (com especial atenção aos sensores do tipo infra-vermelhos e sonar) e transferência dessa informação;
- Arquitectura modular correspondendo o módulo de visualização à janela de visualização do motor de jogos e o módulo de simulação à aplicação desenvolvida em conjunto com o USARSim.

Na generalidade, foram implementadas as funcionalidades pretendidas ou foram sugeridas soluções para os problemas encontrados. A funcionalidade de maior importância que não foi solucionada e testada atempadamente, apenas realizada uma prova de conceito, foi a arquitectura da comunicação com o módulo de controloIW.

A comunicação entre o controlador e a aplicação desenvolvida deveria ser diferente o que implicava uma mudança de protocolo, de preferência tirando vantagem da comunicação por TCP socket, uma para cada robô visto que o controlo cooperativo entre agentes é realizado pelo controlador robótico de 'IntellWheels' e para a presente aplicação não é necessária a junção de informação de todas as cadeiras em simulação.

No âmbito geral considera-se que o projecto desenvolvido conseguiu de facto construir um simulador e visualizador robótico para cadeiras de rodas inteligentes, no entanto, novas funcionalidade deveriam ser implementas e alguns aspectos da arquitectura desenvolvida deveriam ser reanalisados.

Conclui-se que parte do comportamento de movimento da cadeira não é o correcto devido à adaptação do simulador USARSim ao UDK ainda ser recente.

6.2 Propostas de Desenvolvimento Futuro

Nesta secção pretende-se apresentar propostas de continuação do trabalho desenvolvido, englobando igualmente o módulo de controlo da plataforma ‘IntellWheels’. As abordagens ao trabalho desenvolvido, sugeridas como desenvolvimento futuro correspondem a:

- Análise, reestruturação e implementação de um protocolo de comunicação adaptado aos dois módulos (ControloIW e aplicação desenvolvida) que permita a validação automática da estrutura e conteúdo das mensagens XML a serem trocadas e respectiva documentação;
- Aumentar o grau de profundidade das configurações de simulação, abrangendo qualquer configuração pertinente implementada pelo simulador robótico USARSim;
- Automatizar a construção das classes em unreal script para modelos de cadeiras de rodas inteligentes, aumentando o grau de profundidade de configuração dos mesmos, abrangendo configurações do motor de jogos;
- Implementar uma solução de configuração de cadeira de rodas inteligente que envolva colaboração no registo da mesma, entre a aplicação desenvolvida e o controladorIW;
- Gerar automaticamente a documentação da plataforma ‘IntellWheels’ que corresponde a um conjunto de módulos independentes, com o intuito de diminuir a complexidade de familiarização com o projecto;
- Criar manuais de utilização e manuais de desenvolvimento para cada módulo do projecto ‘IntellWheels’.

Referências

[1] *Autonomous Mobile Robotics Toolbox - User's manual*. (s.d.). Obtido em 01 de 08 de 2010, de IUSIANI - Universidad Las Palmas de Gran Canaria: http://serdis.dis.ulpgc.es/~ii-srm/MatDocen/notas_practicas/Simrobot/simguide_en.html

[2] Balaguer, B., Balakirsky, S., Carpin, S., Lewis, M., & Scrapper, C. *USARSim: a validated simulator for research in robotics and automation*.

[3] Bell, D. A., Borenstein, J., Levine, S. P., Koren, Y., & Jaros, L. (1994). *An Assistive Navigation System for Wheelchairs Based upon Mobile Robot Obstacle Avoidance*. University of Michigan, Department of Mechanical Engineering1 and Rehabilitation Engineering Program. IEEE.

[4] Bland, E. (27 de 02 de 2009). *Wheelchair Arm Controlled by Thought Alone*. Obtido em 01 de 08 de 2010, de Discovery News: http://dsc.discovery.com/news/2009/02/27/wheelchair-thought.html

[5] Boeing, A., & Bräunl, T. *Evaluation of real-time physics simulation systems*.

[6] Borgerding, B., Ivlev, O., Martens, C., Ruchel, N., & Gräser, A. *FRIEND Functional Robot Arm with User Friendly Interface for Disabled People*. Institute of the Automation Technology (IAT), University of Bremen.

[7] Braga, R. A., Petry, M., Moreira, A. P., & Reis, L. P. (2008). INTELLWHEELS A Development Platform for Intelligent Wheelchairs for Disabled People. *International Conference on Informatics in Control, Automation and Robotics* (pp. 115-121). IEEE.

[8] *Bullet_(software)*. (n.d.). Retrieved 08 01, 2010 from Wikipedia: http://en.wikipedia.org/wiki/Bullet_(software)

[9] *BulletFull*. (n.d.). Retrieved 08 01, 2010 from BulletPhysics: http://bulletphysics.com/Bullet/BulletFull/

[10] Chandler, D. (2008). Finding its own way. *MIT - TechTalk , 53* (3), 8.

[11] Chandler, D. (19 de 09 de 2008). *Robot wheelchair finds its own way - MIT invention responds to user's spoken commands*. Obtido em 01 de 08 de 2010, de MIT news: http://web.mit.edu/newsoffice/2008/wheelchair-0919.html

[12] *ClanLib*. (n.d.). Retrieved 08 01, 2010 from Wikipedia: http://en.wikipedia.org/wiki/ClanLib

[13] Craighead, J., Murphy, R., Burke, J., & Goldiez, B. (2007). A Survey of Commercial & Open Source Unmanned Vehicle Simulators. *IEEE International Conference on Robotics and Automation* (pp. 852-857). Roma: IEEE.

[14] *Documentation*. (n.d.). Retrieved 08 01, 2010 from CrystalSpace3D: http://www.crystalspace3d.org/main/Documentation

[15] *Exult*. (n.d.). Retrieved 08 01, 2010 from Wikipedia: http://en.wikipedia.org/wiki/Exult

[16] *Exult*. (n.d.). Retrieved 08 01, 2010 from Sourceforge: http://exult.sourceforge.net/docs

[17] *Features*. (n.d.). Retrieved 08 01, 2010 from Blender: http://www.blender.org/features-gallery/features/

[18] *Game_Blender*. (n.d.). Retrieved 08 01, 2010 from Wikipedia: http://en.wikipedia.org/wiki/Game_Blender

[19] Hamagami, T., & Hirata, H. (2004). Development of Intelligent Wheelchair Acquiring Autonomous, Cooperative, and Collaborative Behavior. *International Conference on Systems, Man and Cybernetics* (pp. 3525-3530). IEEE.

[20] *Havok_(software)*. (n.d.). Retrieved 08 01, 2010 from Wikipedia: http://en.wikipedia.org/wiki/Havok_(software)

[21] *Home*. (n.d.). Retrieved 08 01, 2010 from jMonkeyEngine: http://www.jmonkeyengine.com/home/

[22] Hoyer, H., & Hölper, R. (1993). Open Control Architecture for an Intelligent Omnidirectional Wheelchair. *Proc. of the 1st TIDE Congress*. Brussels: IOS Press.

[23] *hpl*. (n.d.). Retrieved 08 01, 2010 from moddb: http://www.moddb.com/engines/hpl

[24] *HPL_Engine*. (n.d.). Retrieved 08 01, 2010 from Wikipedia: http://en.wikipedia.org/wiki/HPL_Engine

[25] Hugues, L., & Bredeche, N. *Simbad : an Autonomous Robot Simulation Package for Education and Research*.

[26] *irrlicht features*. (n.d.). Retrieved 08 01, 2010 from Sourceforge: http://irrlicht.sourceforge.net/features.html

[27] *Irrlicht_Engine*. (n.d.). Retrieved 08 01, 2010 from Wikipedia: http://en.wikipedia.org/wiki/Irrlicht_Engine

[28] Koenig, N., & Howard, A. (2004). Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator. *International Conference on Intelligent Robots and Systems* (pp. 2149-2154). Sendai: IEEE.

[29] Lakany, H. (2005). Steering a wheelchair by thought. *The IEE International Workshop* (pp. 199-202). IEEE.

[30] *latest analyses of Bullet Physics*. (n.d.). Retrieved 08 01, 2010 from Ohloh: http://www.ohloh.net/p/bulletphysics/analyses/latest

[31] Laue, T., & Röfer, T. *SimRobot – Development and Applications*.

[32] Laue, T., Spiess, K., & Röfer, T. *SimRobot – A General Physical Robot Simulator and its Application in RoboCup*.

[33] Madarasz, R. L., Heiny, L. C., Cromp, R. F., & Mazur, N. M. (1986). The Design of an Autonomous Vehicle for the Disabled. *Journal of Robotics and Automation , RA-2* (3), 117-126.

[34] *Main Page*. (n.d.). Retrieved 08 01, 2010 from Ogre3D: http://www.ogre3d.org/

[35] *Main Page*. (n.d.). Retrieved 08 01, 2010 from ODE: http://www.ode.org/

[36] *Main Page*. (n.d.). Retrieved 08 01, 2010 from Havok: http://www.havok.com/

[37] *Main_Page*. (n.d.). Retrieved 08 01, 2010 from CrystalSpace3D: http://www.crystalspace3d.org/main/Main_Page

[38] *Main_Page*. (n.d.). Retrieved 08 01, 2010 from Clanlib: http://clanlib.org/wiki/Main_Page

[39] *Main_Page*. (n.d.). Retrieved 08 01, 2010 from wiki gameblender: http://www.wiki.gameblender.org/index.php?title=Main_Page

[40] Miguel, R. S. (29 de 06 de 2009). *Toyota Wheelchair Guided by Thought Alone*. Obtido em 1 de 08 de 2010, de Tech News World: http://www.technewsworld.com/rsstory/67457.html

[41] Miller, D. R., & Slack, M. G. (1995). Design and Testing of a Low-Cost Robotic Wheelchair Prototype. *Autonomous Robots* (2), 77-88.

[42] Namee, B. M., Beaney, D., & Dong, Q. (2010). *Motion in Augmented Reality Games: An engine for creating plausible physical interactions in augmented reality games.* Dublin.

[43] *Open_Dynamics_Engine*. (n.d.). Retrieved 08 01, 2010 from Wikipedia: http://en.wikipedia.org/wiki/Open_Dynamics_Engine

[44] *OSG*. (n.d.). Retrieved 08 01, 2010 from OpenSceneGraph: http://www.openscenegraph.org/projects/osg

[45] Pepper, C., Balakirsky, S., & Scrapper, C. *Robot Simulation Physics Validation.*

[46] *PhysX*. (n.d.). Retrieved 08 01, 2010 from Wikipedia: http://en.wikipedia.org/wiki/PhysX

[47] *physx_new*. (n.d.). Retrieved 08 01, 2010 from Nvidia: http://www.nvidia.com/object/physx_new.html

[48] *PlayerStage Gazebo*. (s.d.). Obtido em 01 de 08 de 2010, de Sourceforge: http://playerstage.sourceforge.net/index.php?src=gazebo

[49] Prassler, E., Scholz, J., & Fiorini, P. *A Robotic Wheelchair Roaming in a Railway Station.*

[50] Prassler, E., Scholz, J., & Fiorini, P. *Navigating a Robotic Wheelchair in a Railway Station During Rush-Hour*.

[51] *resources*. (n.d.). Retrieved 08 01, 2010 from BricoWorks: http://www.bricoworks.com/resources.html

[52] Ross, E. (2000). *Intelligent User Interfaces: Survey and Research Directions.* Technical Report: CSTR-00-004, University of Bristol, Department of Computer Science.

[53] *Simbad Project Home*. (s.d.). Obtido em 01 de 08 de 2010, de Sourceforge: http://simbad.sourceforge.net/

[54] *SubSim - An Autonomous Submarine Simulation System*. (s.d.). Obtido em 01 de 08 de 2010, de Robotics & Automation Lab - The University of Western Australia: http://robotics.ee.uwa.edu.au/auv/subsim.html

[55] Teller, S., & Roy, N. (s.d.). *Autonomous Wheelchair*. Obtido em 01 de 08 de 2010, de CSAIL: http://www.csail.mit.edu/videoarchive/research/robo/autonomous-wheelchair

[56] Tyberghein, J., Sunshine, E., Richter, F., Gist, M., & Brussel, C. V. (2010, Junho 16). *Developing video games and virtual environments with the Crystal Space engine*. Retrieved Julho 31, 2010 from CrystalSpace3D: http://www.crystalspace3d.org/mediawiki/images/3/3c/CrystalSpace-DEnG-VE.pdf

[57] *usarsim - Main_Page*. (s.d.). Obtido em 01 de 08 de 2010, de Sourceforge: http://usarsim.sourceforge.net/wiki/index.php/Main_Page

[58] Wellman, P., Krovi, V., & Kumar, V. *An Adaptive Mobility System for the Disabled*. PA 19104-6315, University of Pennsylvania, Department of Mechanical Engineering and Applied Mechanics, GRASP Laboratory, Philadelphia.

[59] Yanco, H. A. (1998). Integrating Robotic Research: A Survey of Robotic Wheelchair Development. *AAAI Spring Symposium on Integrating Robotic Research, Stanford University*.

[60] Zaratti, M., Fratarcangeli, M., & Iocchi, L. *A 3D Simulator of Multiple Legged Robots based on USARSim*. Rome.

[61] Games, E. (n.d.). *Features of Unreal Engine 3*. Retrieved 06 2011, from Unreal Engine: http://www.unrealengine.com/features

[62] Games, E. (n.d.). *home*. Retrieved 06 2011, from Unreal Development Kit: http://www.udk.com/

[63] *Unreal Engine 3*. (n.d.). Retrieved 06 2011, from wikipedia: http://en.wikipedia.org/wiki/Unreal_Engine_3

[64] *Unreal Engine*. (n.d.). Retrieved 06 2011, from wikipedia: http://en.wikipedia.org/wiki/Unreal_Engine

[65] *desenvolvimento de jogos - UDK*. (n.d.). Retrieved 06 2011, from wikidot: http://desenvolvimentodejogos.wikidot.com/udk

[66] Braga, R. A. (09 de 2010). *Plataforma de Desenvolvimento de Cadeiras de Rodas Inteligentes*. Porto: FEUP.

[67] Malheiro, P. M. (07 de 2008). *Intelligent Wheelchair Simulation*. Porto: FEUP.

[68] USARSim. (n.d.). *USARSim-manualUT2004_v3.1.3*. Retrieved 06 2011, from SourceForge: http://sourceforge.net/projects/usarsim/files/Documentation/3.1.3/USARSim-manual_3.1.3.pdf/download

[69] Nokia. (n.d.). *eclipse integration*. Retrieved 06 2011, from Qt: http://qt.nokia.com/developer/eclipse-integration/

[70] *home*. (n.d.). Retrieved 06 2011, from turbosquid: http://www.turbosquid.com/

[71] *Textures*. (n.d.). Retrieved 06 2011, from Mayang: http://mayang.com/textures/index.htm

[72] (n.d.). Retrieved 06 2011, from CGTextures: http://cgtextures.com/

[73] *Textures*. (n.d.). Retrieved 06 2011, from Bishop Digital Image: http://www.bishopdigitalimage.com/texture.htm

[74] sandern. (n.d.). *UDKUSARSim souce code*. Retrieved 06 2011, from svn: http://svn.sandern.com/nao/trunk/UDKUSARSim/

[75] *USARSim (source code)*. (n.d.). Retrieved 06 2011, from USARsimCD - Simulation: http://www.cs.cmu.edu/~softagents/USAR/CD/USARsimCD/Simulation/USARSim/

[76] Games, E. (n.d.). *Vehicles Technical Guide*. Retrieved 06 2011, from Unreal Developer Network: http://udn.epicgames.com/Three/VehiclesTechnicalGuide.html

[77] usarsim. (n.d.). *Source Code Usarsim*. Retrieved 06 2011, from sourceforge: http://usarsim.git.sourceforge.net/git/gitweb.cgi?p=usarsim/usarsim;a=summary