Faculdade de Engenharia da Universidade do Porto

Intelligent Wheelchair Simulation

Pedro Miguel Candeias de Castro Malheiro

Dissertation executed under the
Integrated Master in Electrotechnical and Computer Engineering
Major in Automation

Supervisor: Prof. Dr. Luis Paulo Reis

July 2008

Dedicated to

My parents, for their support, trust and for having always believed in me

Abstract

Today's society is more and more concerned with the integration of disabled people in the community. Wheelchair patients are an example of a group that still faces segregation, mainly due to their dependence on other people for a large part of their daily routines. With the increase of wheelchair users and the disparity of afflictions that impose such a condition, scientific research in the area has become more pertinent and significant. Some handicaps that affect arm movements, motor coordination or sight, make it impossible to drive a common powered wheelchair. Patients who have such physical disabilities will benefit from intelligent wheelchair projects, which can give some independence back to them.

Numerous intelligent wheelchair prototypes are in development worldwide and several hardware platforms have been developed. As such, the challenge now is to develop the control algorithms, in particular complex high level control strategies. Direct implementation of new software modifications in real wheelchairs is not viable due to financial costs and, more importantly, to the involvement of human risk. The solution is to use simulation for the control model validation thus bringing great savings both in time and money to algorithm development, testing and validation.

This dissertation describes the development of an intelligent wheelchair simulator. It starts by going through the description of the intelligent wheelchair concept and the reasons of its increasing importance. It justifies the need for simulation during the development process, all the way from initial testing until the final implementation. The document continues with the description of the intelligent wheelchair project - Intelwheels – under development at the Artificial Intelligence and Computer Science Lab of the University of Porto that directly led to this simulation development.

This document also presents the state of the art in robotic simulation. Special attention is given to Ciber-Rato, a simulator developed at University of Aveiro for a robotics competition, which was the base for the simulator developed during this project, called Intelwheels Simulator. It then goes through the architecture and the implemented algorithms of the Intellwheels simulator itself. The developed robotic controlling agents are presented, with

special focus of the modifications made to the Intellwheels Control Software, previously developed. Finally it goes through the graphic application for realistic simulation viewing and discusses the results of real, virtual and mixed reality tests that prove the capabilities and advantages of the simulator developed and its importance in IW development.

Resumo

Cada vez é maior a preocupação da sociedade com a integração na comunidade de pessoas portadoras de deficiência. Os utilizadores de cadeiras de rodas são exemplo de um grupo social que ainda se defronta com segregação, devida, em grande parte, à sua dependência relativamente a inúmeras tarefas quotidianas. A investigação científica nesta área ganhou importancia e significado com o aumento do número de pessoas com necessidade de utilizar cadeira de rodas. Deficiências tais como as que afectam os movimentos dos braços, a coordenção motora ou até a visão tornam impossível a condução de uma cadeira de rodas electrica com um joystick convencional. As pessoas que sofrem de tais incapacidades irão certamente beneficiar com projectos de desenvolvimento de cadeiras de rodas "inteligentes", capazes de lhes restituir alguma independência.

Numerosos protótipos de cadeiras de rodas inteligentes são alvo de desenvolvimento em todo os mundo, tal como o têm sido várias plataformas de hardware. Assim, o desafio centra-se agora no desenvolvimento de algoritmos de controlo, particularmente estratégias de controlo de alto nível. A aplicação directa, em cadeiras de rodas reais, de novos algoritmos não é viável não só devido a razões financeiras, mas, sobretudo, pelo risco de segurança pessoal associado. A solução surge com o recurso à simulação para a validação dos modelos, possibilitando significativas poupanças tanto em tempo como em recursos económicos no desenvolvimento, teste e validação do software.

Esta dissertação procura descrever o desenvolvimento de um simulador de cadeiras de rodas inteligentes, começando pela descrição do conceito de cadeira de rodas inteligente assim como pelas razões que justificam a sua importância crescente. É justificada a necessidade de simulação durante todo o processo de desenvolvimento, desde o primeiro teste de baixo-nivel até à sua implementação final. O documento descreve ainda o projecto de cadeira de rodas inteligente que conduziu directamente ao desenvolvimento desta simulação – Intelwheels – em desenvolvimento no Laboratório de Inteligência Artificial e Ciência de Computadores (LIACC), na Universidade do Porto.

O documento descreve ainda o estado actual da simulação robótica, analisando a arquitectura e os algoritmos implementados. Foi dada especial atenção ao Ciber-Rato, um simulador desenvolvido na Universidade de Aveiro destinado à competição robótica que esteve na base do designado Simulador Intelwheels desenvolvido neste projecto. São ainda apresentados os agentes de controlo robótico desenvolvidos, focando em particular as modificações introduzidas no sofware de Controlo Intelwheels, anteriormente desenvolvido no LIACC. Finalmente aborda a aplicação grafica desenvolvida destinada à visualização da simulação de uma forma realista. São apresentados os resultados de testes em ambientes

reais, virtuais e mistos que procuram mostrar as capacidades e vantagens do simulador desenvolvido assim como a sua importância no contexto do desenvolvimento de cadeiras de rodas inteligentes.

Acknowledgments

I would like to thank my supervisor, Prof. Dr. Luis Paulo Reis, for initially suggesting this dissertation and for the precious reviews of my work. I also want to thank Rodrigo Braga for the fantastic work environment he provided during this time. Not only did I gain experience and knowledge but I also enjoyed it.

I want to thank my parents, my brother and my sister for the support and for all the help they gave me.

Joana… thank you for reviewing my writing and especially for bearing by my side the load of the working days and long nights. Finally…

Last but not least, thank you to my friends. Bits and pieces of your words have surely echoed, in some form, along my work as well.

List of Contents

List of Figures

List of Tables

Abbreviations

List of abbreviations

AR – Augmented Reality
COM – Center of Movement
DCOSS – Distributed Computing in Sensor Systems
FEUP – *Faculdade de Engenharia da Universidade do Porto*
IP – Internet Protocol
IR – Infra-Red
IW – Intelligent Wheelchair
LIACC – *Laboratório de Inteligencia Artificial e de Computadores*
MR – Mixed Reality
OS – Operating System
UDP – User Datagram Protocol
UML – Unified Modeling Language
VR – Virtual Reality
XML – Extensible Markup Language

Chapter 1

Introduction

1.1 - Motivation

Nowadays one can witness the increase of world population carrying some form of physical incapability, affecting locomotion. Based on World Health Organization (WHO) data, it is estimated that around 2% of world population (130 million people) live with physical handicaps[1][2]. This number is due to various reasons and it has been, in fact, growing. The aging of population due to life expectancy increase, environmental degradation and sub nutrition lead to the appearance of chronic diseases which, together with factors like traffic and work accidents, wars and congenital deficiencies, contribute to the increase of people with mobility difficulties[2][3].

With the objective of responding to numerous mobility problems, various intelligent wheelchair related projects have been created in the last years. They try not only to give mobility to handicapped people but, more importantly, they are aiming at doing it in an autonomous way, independent of third party help. Aside with other projects, the Artificial Intelligence and Computer Science Lab of the University of Porto (LIACC) is developing an Intelligent Wheelchair (IW) prototype[2]. Through hardware such as sensors, communication boards and simple human-chair interface devices, the IW is capable of understanding high level orders, such as going from a room to a toilet[4][5]. A complex system of navigation with trajectory planning algorithms, communication and interaction (with other IWs and other intelligent systems), make this IW a solution that will give patients high degrees of independence[6]. They will be able to so, even if their levels of disability are such that, in a common electrical wheelchair, would not be possible for them to correctly drive it with the joystick.

The development of this IW project is in an advanced stage of development. Therefore, the challenge is now to test and validate the high level control solutions: tests to intelligence

and to reliability. In fact, testing new control algorithms in real systems is not viable due to low IWs' availability, time and space needed. Moreover, placing real IW prototypes in a real hospital situation would create risks for humans, which is not acceptable.

Solving this problem involves the development of a simulation environment that will allow testing every aspect of a modification or an addition to the existing system, without submitting the real system resources to disturbances. The advantages of developing a simulation system for IWs and their surrounding environment are numerous when comparing to an immediate real implementation. Disposing of a large number of IWs, for collaborative algorithm testing, would incur in large associated costs while, on the other hand, simulating multiple IWs in a computer will not have any addition cost [7]. Through simulation it is possible to compress time, in the sense that viewing all the consequences of a modification in a fraction of the otherwise necessary real time. Error tracking is simplified: finding the "why" of a certain occurrence is possible through an isolation and detailed analysis of a specific event or period of time. Additionally, requirement specification is possible and more focused. For example, simulation can identify what will be the necessary resolution for a given sensor, so that the system will behave correctly.

The IW project currently being developed in LIACC has the objective of being as modular as possible so that it can be adapted to any electric wheelchair, any type of sensors and motors. In order to continue this philosophy, the IW simulator should be no different. Apart from being a high level control algorithm test board, the simulator must be able to adapt to any type of sensors and hardware characteristics used in each chair.

1.2 - Objectives

This project will be based on the IW prototype developed at LIACC and on the "Ciber-Rato" software, developed by University of Aveiro[8]. "Ciber-Rato" is a robotic simulator for a competition where the goal is to develop an agent which will control the simulated robot and guide it through a maze[9]. The agent must, through signals from the simulator (that represent sensor values), send power values to left and right motors of the simulated differential robot.

Based on "Ciber-Rato", the new intelligent wheelchair simulator must contain every functionality needed to test the LIACC's IW system. Therefore, the main objectives of this project are:

- Development of a simulation server, based on "Ciber-Rato", which is able to simulate the wheelchair's body, sensoring information and world map. The simulator should accept purely virtual wheelchair agents as well as real wheelchairs (allowing augmented reality mode).

- Development of a simulation viewer, adapted to IW requirements, such as the correct view of robot's body (allowing different body types and sizes) and "first person" viewing capabilities.
- Adaptation of the wheelchair's control software to allow it to work as a purely virtual robotic agent (connecting only to the simulator), as a purely real wheelchair controller (connection only to the real wheelchair) or in an augmented reality mode (connecting to both the real wheelchair and the simulator).

1.3 - Dissertation Structure

This dissertation is organized in 8 chapters, the first of which is this introduction to the intelligent wheelchair simulation project.

On the second chapter it is given an overview the LIACC's "Intellwheels" project. The motivation, vision, architecture and present stage of development are discussed.

The third chapter will offer a vision on the main projects in robotic simulation, with emphasis on intelligent wheelchair simulation projects. It analyses the state-of-the-art on this topic and special attention is given to University of Aveiro's "Ciber-Rato", thoroughly describing the simulator.

On chapter four, it is presented the developed simulator, designated "IntellWheels Simulator", in accordance to the main project's name. It starts by defining the software's architecture and continues with the explanation of the modifications to "Ciber-Rato" and the new implemented algorithms.

The fifth chapter contains the work done on developing robot controlling agents that connect with the simulator. A robot represents any type of object, depending on the control it is applied. This chapter analyses the developed generic robotic agent and the door agent. It finishes by describing the applied modifications to the IntellWheels Controller software for added simulation functionalities, including communication requirements.

Chapter six goes through the developed visualization agent for the simulation. The motivations for its development as well as the requirements of an intelligent wheelchair-specific viewer are discussed. It also introduces the 2D and 3D drawing software and algorithms.

The seventh chapter explains the methodology for the experimental tests. It expresses the results of both real and simulator tests for better comparison. It also demonstrates the simulation projects' functionalities with an augmented reality test.

On the eighth and final chapter of this dissertation, conclusions on the entire simulation project are drawn with more detail on the final test results. To finalize, the chapter recommends possible paths for additional development.

Chapter 2

Intellwheels Project

The need for an intelligent wheelchair simulator has its origin in the intelligent wheelchair project, which is being developed in LIACC, at the Faculty of Engineering of University of Porto. Although there are more than 40 distinct IW related publications registered with IEEE since year 2000 [10], the Intellwheels Project aims to contribute to the advance in the field. Its main objective is to create a complete intelligent system with hardware and software that can be incorporated into any commercial electric wheelchair. In addition, the system is to be installed in such method that causes little, or none, visual or design impact, which otherwise would further discriminate their handicapped users. Moreover, it must contemplate every type of wheelchair user, from those with small locomotion disabilities to those with mental handicaps that prevent normal arm and hand movements. This is achieved through an advanced software control system that goes from simple shared control, where it "merely" guarantees that the user's manual control does not take him to dangerous situations (such as going through gaps on the ground, steps and collisions), to complex high lever orders made through voice recognition, path planning and autonomous driving and strategy definition for multiple high level goal achievements.

2.1 – Architecture

Intellwheels project's architecture is presented in Figure 2.1, where the different modules of the project are evident. The red square highlights the simulation module which was then main focus and the point of start for this dissertation.

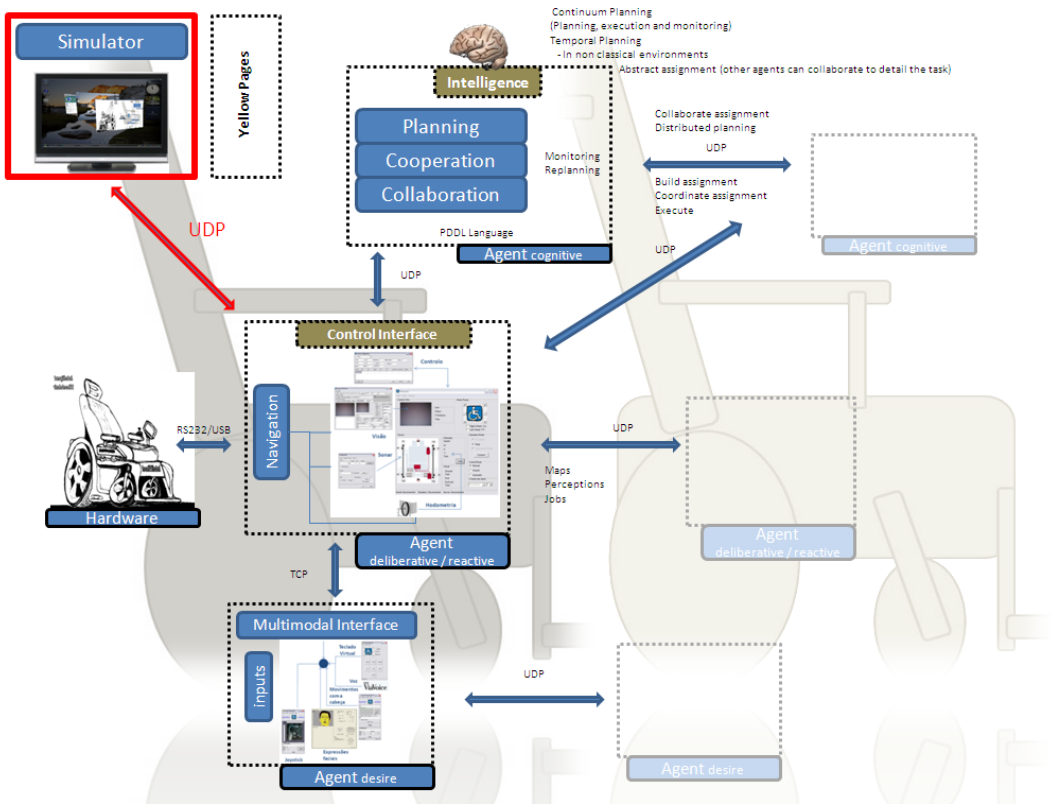


Figure 2.1 - Intellwheels Project Architecture (from[2])

2.1.1 – Hardware

The main hardware of any electric wheelchair are the motors and batteries, but the core of an IW are its sensors. It is through them that it can perceive the world and make intelligent decisions on the orders to give to the motors. Intellwheels' wheelchairs contain sonar and infra-red sensors for object distance detection and encoders on their motors for position calculation. Electronic acquisition plates are also installed for that is what permits remote actuation on the motors and sensor information gathering and sending for the control software. These plates connect to the computer hosting the control software through RS232.

2.1.2 – Main Interface

The main interface is where all the information is gathered. It is through it that all the other modules reach the actual wheelchair and, as such, it is responsible for handling all UDP/IP connections. Every detail of the configurations is dealt with in this main application. It displays relevant information in real time: sensor readings, speed, position, orientation, motor power and operational mode (real, augmented reality or simulated). Figure 2.2 displays the visual form of the module.

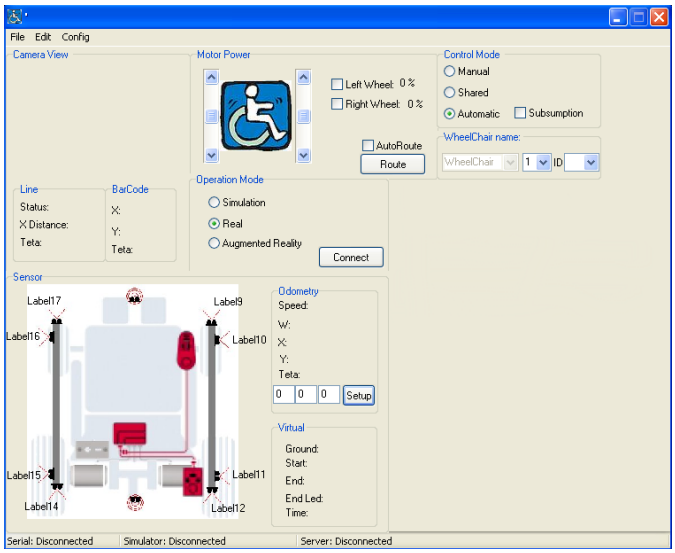


Figure 2.2 - Main Application (adapted from [1])

2.1.3 – Intelligence

Intellwheels project has a multilevel control architecture [2], as illustrated in Figure 2.3. The lower levels - basic control - are handled by the main application itself, separating the higher levels – tactical 4th level and the strategy level - from the Intelligence module.

Figure 2.3 - Intellwheels Control Architecture from [2].

The cognitive agent is responsible for high level decisions, such as continuous planning, runtime monitoring and cooperation with other intelligent agents. An example of the 4th level is the generation of path to achieve a specific location, though the application of A*[1] algorithm. Over this stage a strategic defining level sets the sequence for wheelchair objective (e.g. “pick up” patient from room 10 and take him to doctor office number 2).

[1] A* is a path planning algorithm [6] [3], widely used in intelligent robotics.

The result of this module will be a series of low level instructions to be given to the wheelchair's motors through the main application, independently of details of how those orders are given.

2.1.4 – Multimodal Interface

Making the wheelchair's control correctly and fully understand the user's orders is essential, otherwise the control algorithms and intelligence could work against the patient instead of working for him. Simultaneously to the development of this dissertation, Marcio Sousa is developing a project designated "Multimodal Interface for an Intelligent Wheelchair". This multimodal interface is where all the possible user inputs are handled and it has two main objectives:

- Recognizing sequences of commands which represent specific high medium level orders. This functionality works in a very similar form to "combos" in computer games, where depending on the order of the buttons pressed, the arrow keys could originate different moves.
- Creating the possibility of receiving those orders through as many different inputs as needed: voice recognition, face recognition, joystick or keyboard.

2.1.5 – Simulator

The simulation module was the main focus of this dissertation and a vital one on the Intellwheels project.

This module creates a virtual world and its main objective is to test the control algorithms. In fact, using the real environment every time the control application is modified is not viable. On the other hand, it is not possible to validate a change without a form of testing it. The control application may connect to the simulator, instead of the real wheelchair, and all the consequences of a modification can be verified in a matter of seconds.

However, the simulator's involvement in the IW project is even greater, as the notion of augmented reality is introduced. An interaction of real wheelchair with virtual ones sets the tone for a complete new range of possible testing. Large scale cooperative tests (intelligence module) are possible, no matter how little real IW prototypes are available (Figure 2.4).

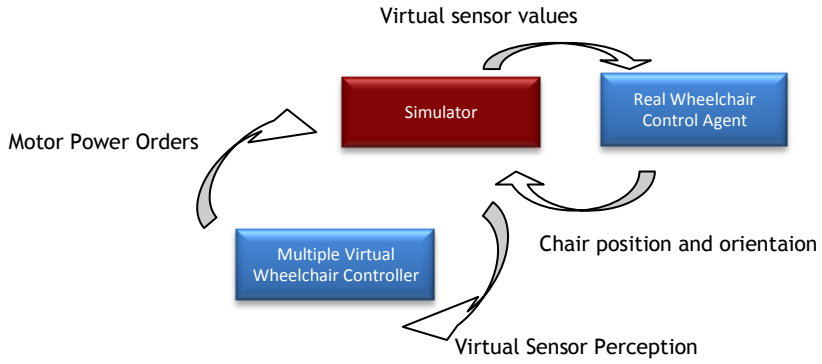


Figure 2.4 - Augmented Reality Concept.

The possibilities for a system where any object (even virtual, intelligent or static) can be placed for real testing are vast:

- Obstacle avoiding.
- Complex path planning.
- Cooperation scenarios.

A complete description of the simulator, its capabilities, algorithms and modes of operation are included further ahead, in Chapter 4 of this dissertation.

2.2 – Development Status

At the present stage, the Intellwheels project is in a midpoint state of development, which adds importance to the simulation project. The present stage requires intensive high level algorithm testing, to which the simulator brings faster, simpler and more focused verification methods.

In terms of hardware implementation, two commercial electric wheelchairs have been modified and transformed into intelligent wheelchair prototypes. Both of them are equipped sonar and IR sensors for proximity calculation, cameras for ground marking orientation, encoders for speed calculation and position determination and communication plates for computer and software control connection.

Low level controlling is fully developed [1], as basic functions like straight driving, turning and going to a point in the Cartesian system were already implemented with success. Medium and high level algorithms, such as automatic object avoidance and path planning, are well advanced as successful tests have been registered. Strategy defining procedures are in design status. Despite this, the basis for implementation is completely developed and a fully functioning intelligent prototype is in the verge of achievement.

The multimodal interface was being developed at the same time as this simulation project itself. Notwithstanding, apart from common powered wheelchair joystick, the IW is already

able to accept direct commands (forward, backward and turning) through the WII and Playstation remote controllers and a virtual keyboard. The camera for face recognition was developed apart from the Intellwheels project even though it is fully operative, its integration with the current control application is not yet complete.

The Simulation module was completely achieved and it is now possible to fully test control algorithms in any type of scenario, with as many virtual or real IW as intended. Moreover, additional possibilities for its usage, such as full intelligent environment (e.g. intelligent doors) were proposed which increase value of the module.

2.3 – Summary

This chapter presented the intelligent wheelchair project being developed in LIACC – Intellwheels Project. It introduced the project's architecture, presenting all of its modules: hardware, main application, intelligence, multimodal interface and simulation. It sums up by presenting the current stage of development of each one of the modules.

This dissertation's main applicability will be on the Intellwheels and its simulator module. As such, its architecture and the concepts presented on this chapter will be referred to in every other section ahead. Moreover, the initial step for this thesis was the review of the state-of-the-art in robotic simulation, which is done in the following chapter.

Chapter 3

Robotic Simulation

This Simulation project was started with the decision to use the Ciber-Rato simulator usage already taken (which is detailed in depth ahead, in section 3.2). The reasons that lead to this are connected to the proven performance and flexibility of this software, as it has been used in different applications and adaptations. It has been successfully used for various competitions: Micro-Rato[11][8][12][13] (2001-2008), CiberMouse@RTSS[14] (2007-2008) and CiberMouse@DCOSS[15] (2008). It was also previously used at LIACC in several research projects such as a computational study on emotions and temperament in Multi-Agent Systems[16][17] and development of cooperative rescue operations[18]. Moreover, previous work had already been done in the Intellwheels project on Ciber-Rato usage for intelligent wheelchair simulation, proving the software's value and suggesting further development on the topic[3].

Despite the choice of the base application already dealt with, it was of relevance to study similar simulation projects in order to incorporate additional concepts into this dissertation.

3.1 - Generic Robotic Simulators

A reference in robotic scientific development is RoboCup. RoboCup is, more than a competition, a complete initiative with the vision of creating a robotic soccer team to win a match against a human one [19]. During the meetings organized each year, robotic soccer and search and rescue competitions are held. They have both simulated and real environment contests, which have been used as inspiration for other robotic conferences around the world, such as Micro-Rato[11] itself.

The RoboCup Simulation 2D League has various points in which it touches this IW simulation project. This competition creates a virtual soccer field and models the two teams, each with 11 players. In an attempt to closely mimic the real robot competition, the virtual bodies of the players are circular, with differential virtual motors for movement control and

possess sensors to aid decision making. Finally each team has a coach, which is a special agent that holds unique information concerning the whole simulation. The coach differs from the player agents in the sense that the data they receive are limited (e.g. by distance) are have noise variation. Because of the differences between each agent inside a team, the focus on this competition is on high level cooperation and control algorithms[20]. For visual simulation following, 2D and 3D viewers are available, although the 3D characteristics are not modeled by the simulator and only appear for better appeal.

Ciber-Rato follows the concepts of the RoboCup 2D League and, consequently, so does the Intellwheels Simulator, developed during this dissertation.

A 3D League was later developed, which implemented profound modifications to the 2D version, especially in terms of world modelling and communications handling, as the SPADES[2] platform was introduced. Once again, in an effort to add more realism to the simulation itself, apart from the full 3D environment, a visual sensor was the main form of perceiving the world. Furthermore, the robot model evolved from the initial sphere to a humanoid, in 2007.

Other robotic simulators were studied during the initial part of this dissertation, which include:

- Gazebo / Player Project[21][22], a simulator for specific robot models, and is capable of generating a wide number of robots with sensors in a three dimensional world.
- UsarSim[23], a simulator that takes advantage of the realistic graphic and physics power of the Unreal Engine, developed by Epic Games enterprise. Through TCP protocol connections, this simulator allows the connection a control application and provide it with a wide variety of sensors: encoders, touch, proximity, RFID, camera, sound and motion sensors.
- Microsoft released, in 2007, a generic robotic simulation environment called Microsoft Robotics Studio[24]. Its target destination was not only the academic developers but commercial ones as well, as the product could simulate a wide range of robotic hardware. Moreover, visual programming language (VPL) was used, allowing easy controlling. Rich environments could be set, as it uses the AGEIA PhysX[25] for world physics calculations. The software rapidly proved its value, as illustrates the 2007 RoboCup[20].

3.2 – Intelligent Wheelchair Simulators

A literature review for other projects focused on intelligent wheelchair simulation was performed and two projects stood out.

[2] System for Parallel Agent and Discrete Event Simulation (SPADES) is an agent focused framework for communication handling, agent-world interaction and physics modeling [46].

3.2.1 - Bremen Autonomous Wheelchair

The Bremen Autonomous Wheelchair (BAW)[26], initiated a simulation related project, motivated by reasons that are shared with this Intellwheels intelligent wheelchair project. As a consequence of developing new hardware platform, there is no commercial simulator that can be directly used. Moreover, the final decision on sensory and communication equipment is not yet defined thus creating the need for a software that is flexible enough to adapt. The Bremen team started with a basic software, developed in their own university, called SimRobot[27], and then set a methodology where they would systematically expand the application each time they would find it necessary. They would submit the real chair to a test and then apply the same test to the simulator. Afterwards, the results from both of them were compared and, in case differences were found, they would upgrade the SimRobot accordingly.

The entire simulation project is, therefore, an enduring evolution towards the equilibrium between the real environment and the simulated one.

Differences, from the objectives of the Intellwheels project, arise when conceptual architectures and simulation objectives are compared. While Bremen simulation is based on a single chair, Intelwheels is multi-agent based, in the sense that a dynamic, more complex environment with multiple intelligent and collaborative objects is intended. Furthermore, BAW simulation segregates completely real and virtual worlds, leaving no room for augmented reality model.

3.2.2 - Vehicule Autonome pour Handicapés Moteurs

An intelligent wheelchair developed in 1998 at the University of Metz, in France - Vehicule Autonome pour Handicapés Moteurs (VAHM)[28] – was, in 2000, extended with a simulation project[29]. The objective of this project was to solve a difficulty encountered during the development: real disabled patient testing. Costly, time wasting and often physically harmful, these tests became a burden that led to need of a simulator.

The simulator's concept was to enable the wheelchair with computer connection abilities and create a software that would simulate the world perception to the chair. The chair was to be the same has the real environment only with a breaking system disabling actual movement, although allowing encoder perception. The patient is then equipped with a virtual reality helmet with which he would see the virtual world. This blend of real variables with the virtual world created mixed reality environment is very similar to what Intellwheels project implements (which will be thoroughly detailed in chapter 4). The chair provides the simulator with real encoder values and the software, in return, sends measures of virtual ultrasonic sensors.

Overall architecture, on the other hand, is then distinguished from Intelwheels'. VAHM's simulation project range ends with the single chair test, while Intellwheels interest goes further into higher level multiple IW collaborative algorithms.

3.3 – University of Aveiro's Ciber-Rato

Ciber-Rato is a robotic simulation software developed for a competition, held at University of Aveiro. The reason for its development was to provide a form of integrating participants whose hardware skills weren't sufficient enough for real robot building, for the older "Micro-Rato" competition. Through "Ciber-Rato" they could concentrate solely on the control algorithms and software issues, as their robots are purely virtual[8].

The game consists of 3 "mice" (robotic agents that control their robot by sending motor power inputs to the simulator) finding their way to a beacon. The agents are applications developed by the contestants, thus separate of the simulator itself. They communicate with it via UDP protocol and XML messaging. The virtual robots have a circular body and have differential drive: a simulated motor for each of its two wheels (illustrated in Figure 3.4). At start, the world is unknown to the agents and they rely on 3 IR sensors, 1 ground sensor, a bumper sensor and a beacon compass to achieve the objective (in a testing mode, a GPS sensor can be used for debugging but its usage is not allowed in the actual competition). The agent that reaches the beacon with the best score (less time and less collisions) wins the match.

The "Ciber-Rato" Simulator developers also created a modified version for a different competition, in the 2008 edition of the International Conference on Distributed Computing in Sensor Systems (DCOSS '08). The goal now is to make a team of 5 agents go to a single beacon and the main difference, in terms of simulation, is that the robots can send messages to each other. Their communication is limited by distance and message size, approaching a more realistic scenario and encouraging development of information exchange algorithms.

3.3.1 – Architecture

Being "Ciber-Rato" itself based on the RoboCup simulation league, it follows the same basic concepts: a distributed architecture where the simulation engine works as a server for all other agents (clients) to connect to[8]. Figure 3.1 gives an overview on the system's conceptual design.

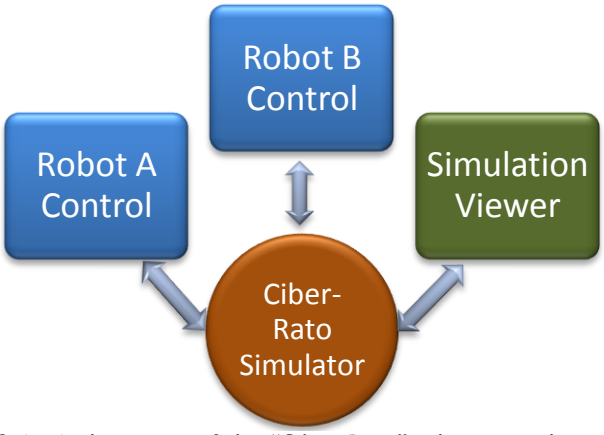


Figure 3.1 - Architecture of the "Ciber-Rato" robotic simulator system

Since every agent is an external application, they can all be developed in a different programming language, having only the concern of using the same communication protocol as the simulator. This possibility is particularly valuable since, in the actual competition, each contestant can use their own computer and operating system.

3.3.2 – Communications

Information exchange between the agents and the simulator and even between the simulator and the viewer are made through UDP and IP protocols. This allows not only to run simulation with different applications for each agent but to run them from different computers, as long as they are connected through an IP network.

The messages themselves are in XML language. Its usage allows not only an easy processing by the programs but, on the other hand, it is concise, formal and human-legible [30]. Figure 3.2 is an example of the XML message that needs to be sent to the simulator server, to initiate the robot simulation.

```
<Robot
        Name="IntellWheels"
        Id="1">
</Robot>
```

Figure 3.2 - XML message for robotic agent registration.

It identifies the main tag has a Robot and defines its name has "IntellWheels" and its Id as "1".

In order to initiate a correct simulation, with robotic and visualization agents, a series of XML messages should be sent to the simulator. A possible sequence is represented in the UML sequence diagram on Figure 3.3.

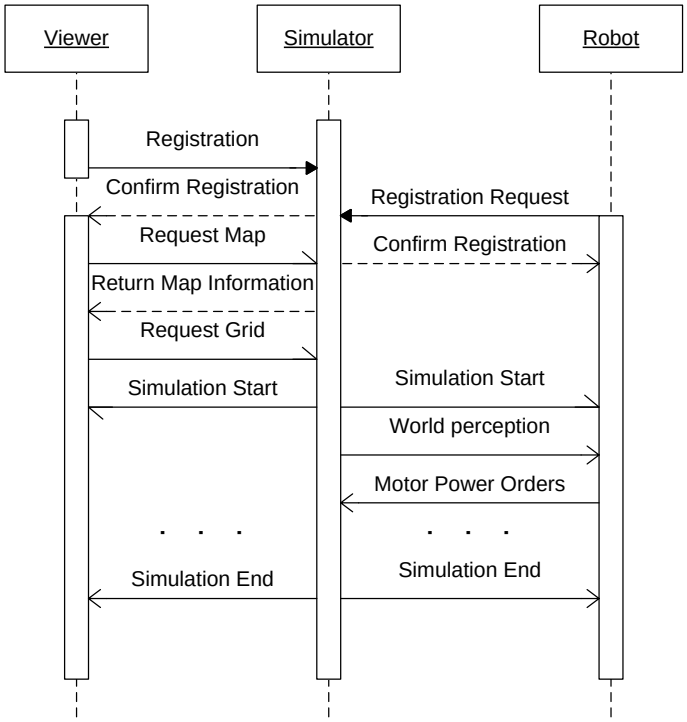


Figure 3.3 - UML sequence diagram of robotic and viewing agents' messaging with the simulator.

3.3.3 – Virtual Robot

The body of all simulated robots is circular, with a radius of 0.5 units. They are differential robots, i.e., they are modeled with two wheels, as shown in Figure 3.4.

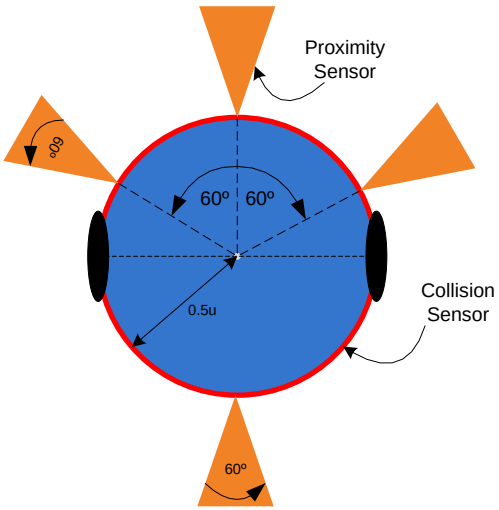


Figure 3.4 - Ciber-Rato's robots' body and sensors.

The center of the circular body is also the center of movement of the robot as the wheels are located symmetrically. Each wheel has its motor and can be controlled by the agent through an XML message, containing the power to be applied in each motor.

```
<Actions
        LeftMotor="0.1"
        RightMotor="-0.1"
/>
```

Figure 3.5 - XML message for robot counter-clockwise rotation

As an example, the command represented in Figure 3.5 will set the robot to rotate in its own center.

By default every robot has their four proximity sensors placed in the perimeter, radially oriented. The arc cone of the sensor's sight is fixed at 60º. On the DCOSS '08 version of "Ciber-Rato", the location of the sensors can be set by each robot. An angle, relative to the robot's frontal direction, will define where, in the circle perimeter, will the sensors be placed [31]. Although not completely configurable, this possibility gives the "Ciber-Rato" simulator a more general and adaptable software for other uses, such has this intelligent wheelchair simulation project.

A beacon sensor is also available. Through this sensor, the simulator will send the robot controlling application a value in degrees (from -180º to 180º), indicating the direction of the beacon, relatively to the robot's current direction. Approaching realistic conditions, if the beacon if too far or if the obstacle between the robot and the beacon is too high, the simulator will not send the beacon sensor reading.

To inform on the robot's direction, a compass sensor is offered, located in the center of the robot. The sensor points in the direction of the rising X axis, as shown in Figure 3.6.

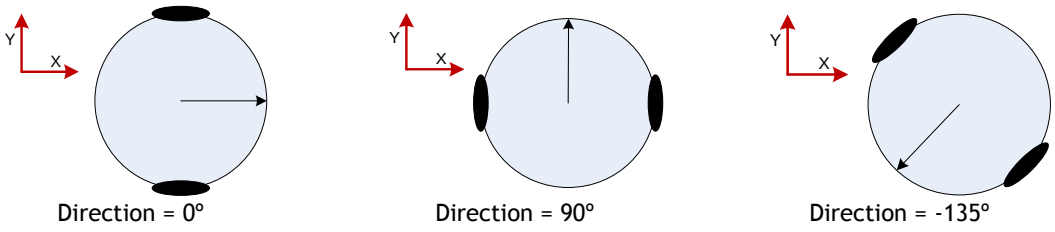


Figure 3.6 - Robot direction information, sent by Ciber-Rato's compass sensor.

The sensor's values range from -180º to +180º. Figure 3.6 a. represents a robot oritented purely on the X axis (direction=0º), on Figure 3.6 b. is show a robot directed only on the Y axis (direction=90º) and Figure 3.6 c. exemplifies a robot facing towards negative X and Y coordinates, relatively to its center (Direction=-135º).

A GPS sensor is also available, returning information on X and Y positions and, also, on robot orientation. These values have addictive noise (as exposed on Table 3.1), with different maximum variations, which confers this sensor a distinct function to the regular compass.

Table 3.1 - Ciber-Rato's sensor characteristics

Sensor	Range	Resolution	Noise Type	Deviation
Proximity	[0.0;100.0]	0.1	adivtive	0.1
Beacon	[-180º;+180º]	1	Adivtive	2.0
Compass	[-180º;+180º]	1	Adivtive	2.0
GPS (position)	N/A	1	Adivtive	0.5
GPS (orientation)	[-180º;+180º]	1	adivtive	5.0
Collision	N/A			
Ground	N/A			

The last two sensors defined in Table 3.1 – Collision sensor and Ground sensor – have binary responses. The collision sensor will return "Yes" in case the robot touches any other robot or a wall. The Ground sensor will return "Yes" if the robot's center is over a target area, defined in the map XML document. In the competition, the target area is a circle with a radius of 2.0 units.

3.3.4 – Robot-Robot Communication

The DCOSS version of Ciber-Rato was meant to encourage information exchange. To achieve this, a robot-robot communication feature was implemented, only on this version, which allows messaging between robots. They are not directly peer to peer. Instead, the messages are sent via the simulation server (through UDP protocol, similarly to the motor power orders) and broadcasted to every robot. This feature allows every robot to receive all the information and decide itself whether to use it or not. To approach real situations, only robots within 8 units of distance of the emissary robot will receive the message. It is possible to send a message of a maximum of 100 bytes but, on the other hand, it is possible to receive a total of 400 bytes at once.

3.3.5 – Map and Wall Modeling

It is possible to load, from an external flat file, the map, in XML language. The map outer limits are a rectangle which is defined by height and width information. Inside the limits there can be walls which are defined by the coordinates of their corners and its height. For competition purposes, a wall can have different heights, which affect the compass sensor of the robot: if the wall is high the beacon will not be in the robot's line of sight, thus disabling compass sensor readings. An ordered sequence of consecutive corner coordinates (minimum of three corners) defines a wall and one map can contain any amount of walls (Annex A).

Figure 3.7 exemplifies a wall construction in XML and its representation, made by the Ciber-Rato Viewer.

```
[…]
<Wall Height="1.00">
  <Corner X="16.50" Y="10.00" />
  <Corner X="16.50" Y="5.50" />
  <Corner X="21.00" Y="5.50" />
  <Corner X="21.00" Y="6.50" />
  <Corner X="17.50" Y="6.50" />
  <Corner X="17.50" Y="10.00" />
</Wall>
[…]
```

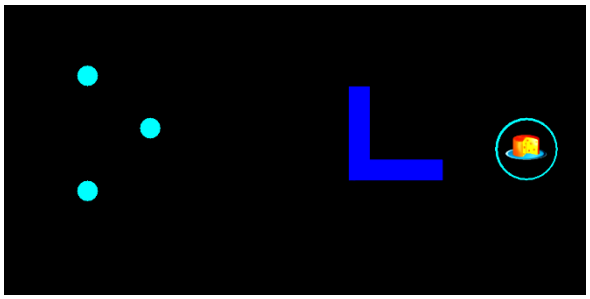


Figure 3.7 - Ciber-Rato Viewer's design of a XML modeled wall.

The wall is defined on the content of a XML file of a Ciber-Rato Map (on the left). Through those six corners, the viewer can then graphically draw the wall, as shown on the print screen of the Ciber-Rato Viewer, on the right. The wall is the blue, L-shaped piece. To adjust to the competitions rules, a second XML file is needed (associated with the map) which contains information characterizing the beacon's position and the target areas (which activates the robot's ground sensor).

3.3.6 – Simulation Viewer

To visually follow the games, a simulator viewer was also developed, again in C++ programming language[32], using QT libraries[33].

Similarly to the robotic controlling agents, the viewer was also an external application that connected to the simulation server with UDP protocol, under IP. To correctly initialize the viewing the software must send the sequence of XML messages shown in Table 3.2.

Table 3.2 - Sequence of XML messages sent from viewer to simulation server, at initialization.

	XML message	Message purpose
1	<View/>	Register itself with the simulation server, as a viewing agent
2	<LabReq/>	Request information regard map limits and wall locations
3	<GridReq/>	Request information concerning the starting points of the robots

Using the assumption that the agents are robots with the characteristics previously explained, the visual appearance of the robots was made through the loading of a bitmap image file. Also, assuming that there would only be three agents connected, a different image was loaded, depending whether the robot's Id was 1, 2 or 3.

Competition related information is also displayed: number of collisions for each robot, elapsed time, robot score and the robot state (through allusive images), as illustrated in Figure 3.8.

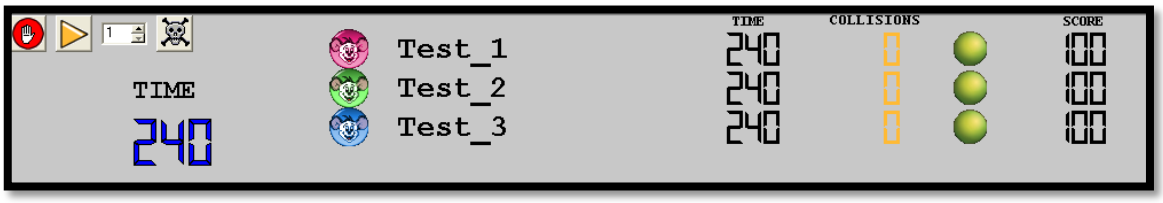


Figure 3.8 - Original "Ciber-Rato" Viewer information display

Additionally trough the information display, the viewer is also capable of some simulation orders. It can start and stop the simulation and remove any specific robot from the simulation. Removing a robot, aside from stop drawing in the viewer screen, the simulation engine will then skip collision verification and sensor information calculation.

3.4 – Summary

The chapter gave an overview on status of the robotic simulation state of the art by present some important projects, such as Robocup, Gazebo/Player Project, UsarSim and the Microsoft Robotics Studio. Since the key topic is intelligent wheelchairs, more focused literature review revealed the Bremen Autonomous Wheelchair and the *Vehicule Autonome pour Handicapés Moteurs* (VAHM) projects.

The main section of this chapter was the Ciber-Rato simulator presentation. Both versions of Ciber-Rato (Micro-Rato and DCOSS) have already implemented some features as noise in sensor readings, which is very important in IW realistic simulation. However, the main setback with the adoption of "Ciber-Rato" as the base simulation environment is the lack of flexibility in parameterization of the robots' characteristics. In fact, the main requirement of an IW simulator is to be able to simulate and handle wheelchairs of different dimensions as well as different sensors and their location. The simplest example is the wheelchair's body: as "Ciber-Rato" simulates only circular robots it is unfit for wheelchair simulation for the error of rounding a rectangle (with its height and width parameters) to a circle if too great. Moreover, originally, there is no form of setting different radius for different robots, which means that all wheelchairs would have to be the same size.

This scenario justifies the need for an IW-specific simulator that is generic enough to model a wide variety of IWs and their sensors. In addition, the study revealed that Ciber-Rato is a good base for the functionalities needed.

Chapter 4

Intellwheels Simulator

The main objective of this project was to develop simulation software to function as a test board for control algorithms for intelligent wheelchairs. Converging with the larger project it is inserted into (described in Chapter 2), the software was designated as "Intellwheels Simulator".

As core functions, this application creates a virtual world, complete with map definition, where robotic agents can connect to. The simulator regulates the connection attempts, handles the communications and returns to the agents the perception of the world, similarly to what a real robot would get from the real environment around it, through its sensors.

The robot control software should treat the awareness information not discriminating it from real or simulated, therefore producing the result independently: it produces orders every connected actuators, being real or virtual. This scenario leads to the subject of reality definitions. In fact, the usage of the same software for real situations as for virtual tests, suggests a leap forward into the augmented reality concept, in which virtual world objects interact with the real world.

This chapter will go through the simulator's conceptual architecture, including how the support for mixed reality was implemented. It goes through the modifications made to Ciber-Rato simulation environment and the new algorithms implemented to correctly simulate IWs.

4.1 – Architecture

Being essentially based in the Ciber-Rato source code, the Intellwheels Simulator has its main basic architecture. Conceptually it is illustrated in Figure 4.1.

In a higher abstraction level, it consists of a central simulation server to which every agent, independently of its type, will connect to. Furthermore, to have a structure as modular as possible, the agents are external applications, developed in any kind of language

and running in any type of operating system, must connect via IP and UDP protocols. Through this obligation, the spectrum of possibilities for agent development is greatly broadened.

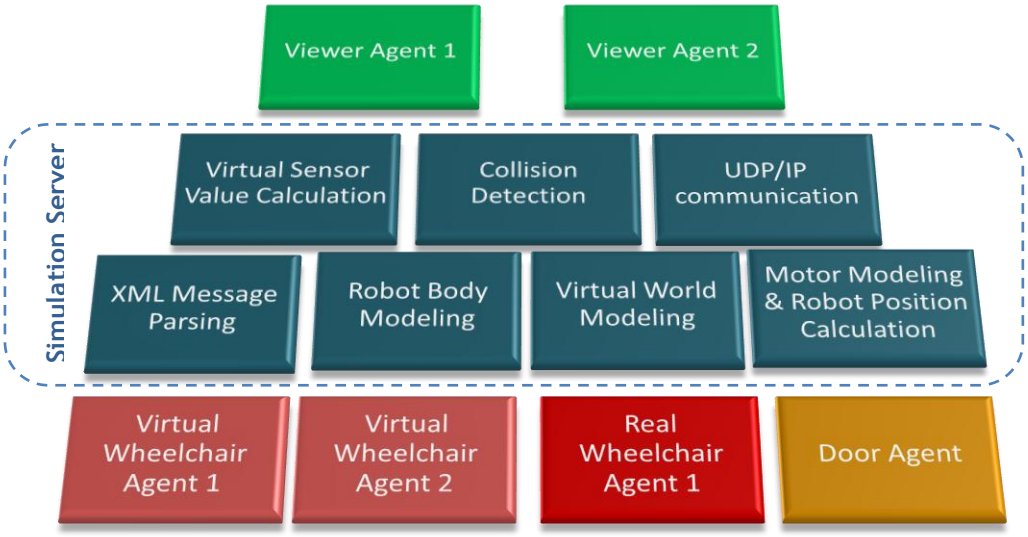


Figure 4.1 - Intellwheels Simulator's architecture

The simulator server is responsible for all calculations concerning simulation (collision detection, position calculation, wheel motor emulation and world perception sensors' values). It is also the assurance of communications between every intelligent agent (independently of their type). Viewer agents are able to graphically draw the modeled world, as the simulator sends them map definition, and robotic agents' positions. These agents, on the other hand, have a more intense interaction with the server. They not only receive information concerning their virtual sensors' perception but also need to send power input orders to their virtual motors.

The physical implementation of this architecture resulted in the usage of laptop computers. They house the simulation and agent applications, which connect through a Wi-Fi wireless network under protocol 802.11g and cabled Ethernet connections, as illustrated in Figure 4.2.

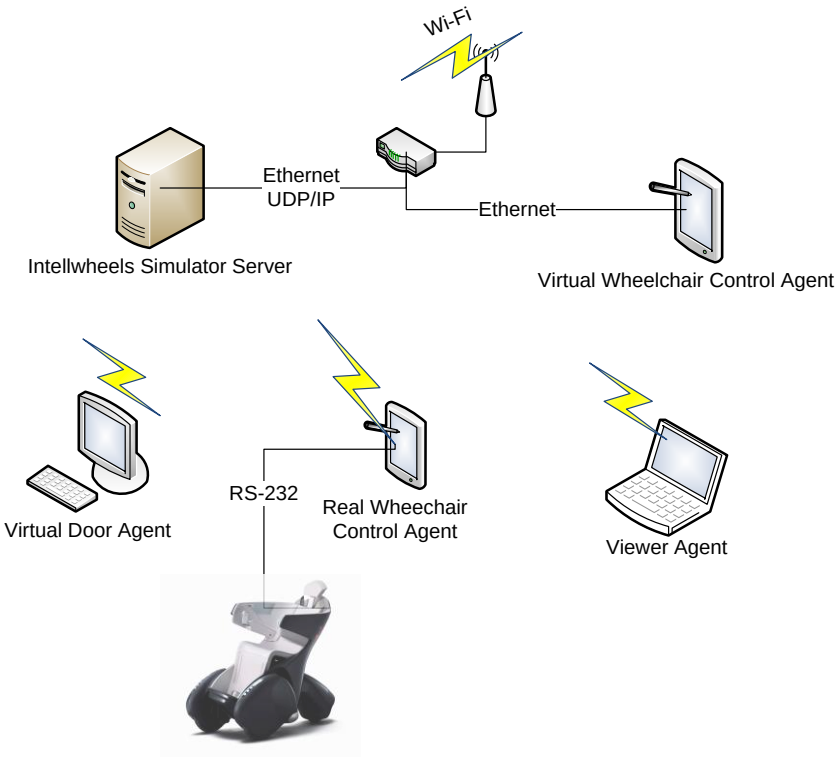


Figure 4.2 - Technology implemented for Intellwheels simulation.

The core of the system is a central computer that runs the simulator server, to which every agent application connects to. The information exchange is made through XML messaging which ensure human and machine-readable content.

The system is composed by a simulator server, which can be a Linux OS or a Windows OS (although, during this project, it was only compiled a Windows version of Intellwheels Simulator). It sets a UDP listen port, to which it will await agents' registration requests. Through specifically ports, attributed individually to each agent, it sends information of their concern: sensor perception (in case of robotic agents) and map, collisions and positioning information (in case of viewer agents). The simulator is also capable of accepting incoming messages to these ports to update the simulation: robot action orders and simulation commands from the viewers.

4.1.1 – Augmented Reality Theory

By definition, augmented reality (AR) is system that allows interaction between real and virtual objects, in a real world[34]. An AR system will synchronize both realities with each other thus ensuring consistency in information merging. Moreover, the concealing must be in real time in order to allow the direct association between virtual and real data.

The conceptual gap between virtual reality (VR) and augmented reality is filled with sublevels of mixed reality definitions[35], as portrayed in Figure 4.3.

Figure 4.3 - Real to Virtual continuum

In real environment there is no interaction what so ever between the physical objects and computer generated information. The System is composed solely by its real objects (walls, tables, chairs, etc.) and perceptions (sonar sensor readings, VGA cameras, etc.).

A similar consideration can be made to the purely virtual environment. A system is mathematically modeled and the entire perception of the world is limited to what the virtual data contains. Every influencing parameter is calculated within a computer and the results are based entirely on the initially programmed information. In the middle of the virtual and the real worlds there can exist blended reality constructions.

In an augmented reality situation, the world is expanded with virtual data. An intelligent agent (e.g. a human person) is able to alter their decisions based on this additional information. A maze solving attempt is an example on where the virtual information would affect the real world. If a person had entered a known building floor and was to go to room 1, shown in Figure 4.4, the choice on which side to go to would be through the left, shorter path.

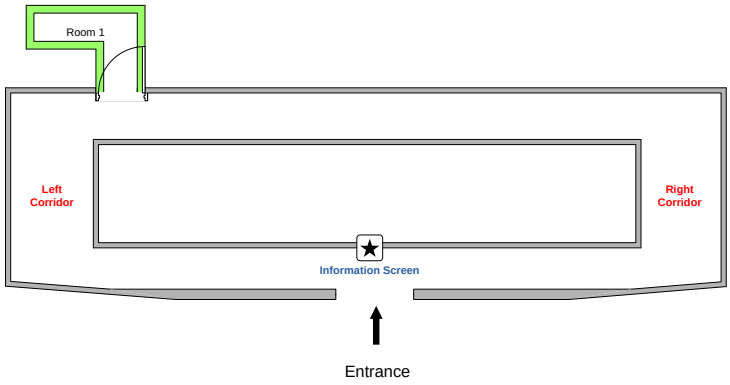


Figure 4.4 - Maze example.

In reality either side will take the subject to its destination, although one path is preferred to the other. In spite of this, if the person was to be informed, by a visual screen at the entrance, that the corridor to the left was blocked due to repainting work in progress, he would chose to take the path to the right. The information received is taken as true, although it may not, thus turning virtual information relevant in the real world.

Conceptually, the augmented virtuality is very similar to the augmented reality, with the differentiation that the influence, in this case, is made by the real world, affecting the virtual workflow. The undergoing virtual process, generated by the computer is disturbed by real world perceptions, therefore adjusting itself to the data received.

The aggregation of the augmented reality and augmented virtuality sets the definition of mixed reality. The two-way interaction of the concepts creates a richer environment where both worlds gain additional information and, as consequence, can produce better judgments in decision making stages.

4.1.2 – Mixed Reality Support

An important part of the simulator is its capability of admitting the connection of different robotic agents. Specifically, it is possible to distinguish an agent that controls a virtual IW from an agent the controls a real IW ergo, there simulator can register two types of robotic agents: “Real” and “Simulated”.

If a type “Simulated” robotic agent connects to the server, it will treat it as a controller for a purely virtual robot. The simulation will then provide it with the world perception, through the modeled virtual sensors. It will also accept incoming XML messages containing actions that set the desired input power to be given to the motors which, consequently, will be a parameter that the simulation engine itself will use to calculate the robot’s following position. It is a completely virtual environment.

In a case where the robot’s type is “Real”, the simulator will regard this agent as an application controlling a real IW, in a mixed reality mode. It is expected that the agent

provides the simulation with the IW's X and Y coordinates (in meters) as well as the angle (in degrees). This allows the virtual world modeled in the simulator to expand with information concerning the real wheelchair. On the other hand, knowing the real wheelchair's position, direction and physical characteristics, the simulator can virtually insert sensors on to it and calculate their values. As an example, the simulator could detect the proximity of the real wheelchair to any other object in the simulation, being virtual or real, like another IW. In sending this new data to the real wheelchair, the simulator is augmenting its reality perception, now acknowledging more information than it could by itself.

This kind of scenario confers the simulator a mixed reality support characteristic that greatly increases the testing capabilities of the software. The IW prototype numbers and costs are no longer obstacles in cooperative and complex experimentations.

4.2 – Programming Technologies

Being based in Ciber-Rato, the Intellwheels simulator is in C++ language. It also uses a set of libraries, with special classes and functions: QT libraries from Trolltech[36]. These libraries are cross-platform (in the sense that they can be used in various operating systems, including Windows and Linux) and provide various class libraries that aid in the low level functions, allowing a higher level of programming. Visual graphic drawing is an example of the contribution that QT made, with cutting time spent on, for example, window and button creation.

Since all the other software applications developed for the Intellwheels Project have been and are being developed under Windows OS, this simulator project should follow the same pattern. This ensures better computability between interacting software and reduces the combined diversity of programming software requirements.

To code and compile under windows it was used the Microsoft Visual Studio C++ integrated software. Not only does this software provide a simple to use programming environment, it also allows direct QT integration, valued ability in these circumstances.

4.3 – Modifications to Ciber-Rato

Although already including a wide range of robotic simulation requirements', the Ciber-Rato by itself is not IW simulating ready, in the sense that it does not fully meet its needs, hence the need for modifications.

Main alterations were done in every function that related to the robot's body definition, starting from the body itself. Changing from circular body to a rectangular one imposes algorithm modification to various modules, mainly collision detection, angular speed and sensor value calculation.

4.3.1 – Robot Body

Intellwheels simulator assumes that all robots have a rectangular form, with a configurable height and width. The original "Ciber-Rato" simulators' robots were circular and with a fixed radius. The error created by approximating a wheelchair's shape to a circle is too great, therefore not even modifying the software to allow adaptable radius would produce satisfactory results, as shown on Figure 4.5.

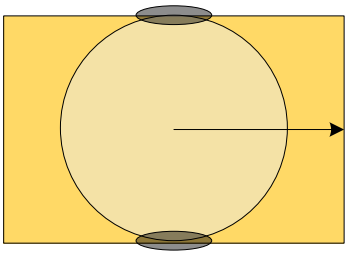


Figure 4.5 - Overlaps of a rectangle shaped robot with a circular shaped one.

Moreover, a wheelchair does not have its center of movement where the center of the physical form. Instead, it is near the rear of the robot, where the axis of the wheels is. On both wheelchairs, as in most of wheelchairs available in the market, the center is on half of its height and on between 70%~90% of the robot's width, as exemplified in Figure 4.6.

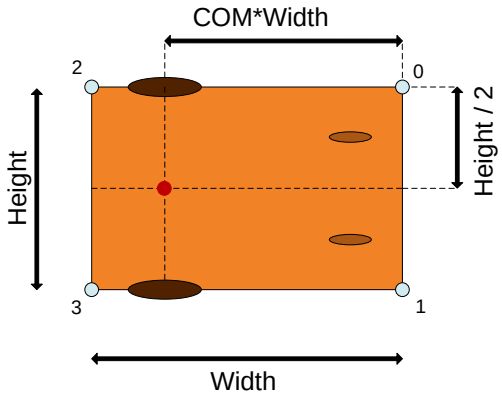


Figure 4.6 - Virtual Body for modeled by Intellwheels simulator.

The robot's Center of Movement (COM) ranges from 0.00 to 1.00 and defines the position of the robot's wheels' axis, relatively to its width. A COM of 0.8 would set the axis closer to the rear of the robot whereas a COM of 0.5 would set the axis would set it in the center of the robot. This point, where the axis of the wheels meet the half of the robot's height will be referred to as the robot's center.

To fully acknowledge the complete body location of the robot, its corners' coordinates and its orientation must be calculated. This is done based solely on the robot's center

position (X, Y and angle) and physical characteristics (width, height and COM). Through the physical characteristics, the relative position of each corner is determined (see Table 4.1).

Table 4.1 - Robot's corners relative position coordinates

Corner Id	X relative position (m)	Y relative position (m)
0	COM*Width	Height/2
1	COM*Width	- Height/2
2	- (1-COM)*Width	Height/2
3	- (1-COM)*Width	- Height/2

The current absolute position can be computed by applying a transform matrix, using the center position information, as illustrated in Figure 4.7.

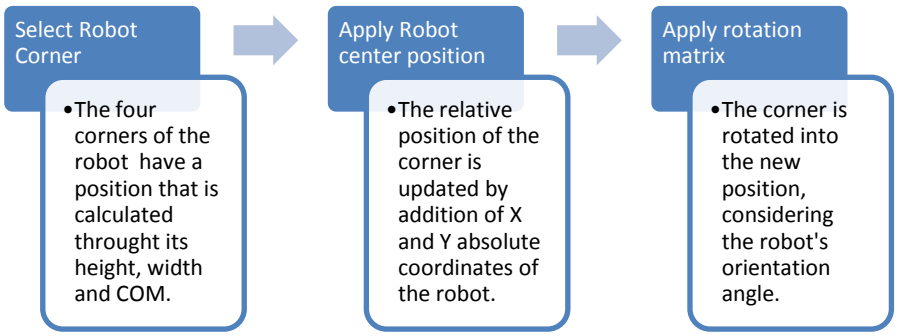


Figure 4.7 - Algorithm for the determination of the robot's corner absolute position.

The rotation matrix, referred of in the matrix equation 4.1, is a generic equation that will rotate any given point, using the Z as the axis around which the spin will be done. Assuming that the robot's center is in the Z axis, the matrix can be applied to determine the robot's corners absolute coordinates.

$$\begin{bmatrix} X_{abs} \\ Y_{abs} \\ 0 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} X_{rel} \\ Y_{rel} \\ 0 \end{bmatrix}, \quad (4.1)$$

where X_{rel} and Y_{rel} are the corner's position, relative to the robot center, X_{abs} and Y_{abs} are the absolute position and θ is the robot's angle. The application of the (5.1) matrix equation resulted in the a new one for each coordinate of each corner, all displayed in Table 4.2.

Table 4.2 - Equations implemented to determine corner positions, relatively to the robot's center coordinates

Corner Id	Equation for X absolute calculation	Equation for Y absolute calculation
0	$Rx + BigDiag * \cos\left(\text{asin}\left(\frac{height}{2 * BigDiag} + R_\theta\right)\right)$	$Ry + BigDiag * \sin\left(\text{asin}\left(\frac{height}{2 * BigDiag} + R_\theta\right)\right)$
1	$Rx + BigDiag * \cos\left(-\text{asin}\left(\frac{height}{2 * BigDiag} + R_\theta\right)\right)$	$Ry + BigDiag * \sin\left(-\text{asin}\left(\frac{height}{2 * BigDiag} + R_\theta\right)\right)$
2	$Rx + SmallDiag * \cos\left(\pi - \text{asin}\left(\frac{height}{2 * BigDiag} + R_\theta\right)\right)$	$Ry + SmallDiag * \sin\left(\pi - \text{asin}\left(\frac{height}{2 * SmallDiag} + R_\theta\right)\right)$
3	$Rx + SmallDiag * \cos\left(-\pi + \text{asin}\left(\frac{height}{2 * BigDiag} + R_\theta\right)\right)$	$Ry + SmallDiag * \sin\left(-\pi + \text{asin}\left(\frac{height}{2 * SmallDiag} + R_\theta\right)\right)$

For all equations show in the table above, Rx and Ry stand for the robot's absolute center coordinates, BigDiag is the distance from the center to corners 0 and 1 and SmallDig is the distance to 1 and 2 corners, as equations 4.2 and 4.3 detail.

$$BigDiag = \sqrt{\left(\frac{height}{2}\right)^2 + (COM * width)^2} \tag{4.2}$$

$$SmallDiag = \sqrt{\left(\frac{height}{2}\right)^2 + ((1 - COM) * width)^2} \tag{4.3}$$

These calculations define the robot's body and are, therefore, a key for input for all simulation functionalities.

4.3.2 - Top Speed and Motor Acceleration Curve

One of the basic wheelchair's parameters that needs to be configurable is the top speed. Each IW will have different maximum speed and the simulation engine was adapted in that direction. In function for next position calculation, the current velocity is initially determined. This equation now in function of the maximum speed parameter of that unique robot (it is a class variable), therefore allowing differentiation between robots. A robotic agent, at registration, should indicate its maximum speed. If it fails to do so, the simulator will set the default speed of 1 meter per second.

An additional modification was made on the dynamic characteristics of the motors' acceleration curves. Since the original software was designed to ensure all robots were equal, every robot connected had to had the same dynamic characteristic. In this IW simulation environment it is expected that the robots connecting may have different dynamic characteristics. Similarly to size and center of movement characteristics, a new robot registration parameter was implemented to allow each robot to define their curve. For

acceleration control, an equation (4.4) was used when calculating the motor's output power, given an input power value.

$$output_n = (1 - curve) * input_n + curve * output_{n-1}\,, \tag{4.4}$$

where $output_n$ is the new motor power output, *curve* is a value between 0.00 and 1.00, defining the slope of the acceleration curve, and $output_{n-1}$ is the power value from the previous period. Figure 4.8 plots the output of equation (4.4). Chart on the left has a curve parameter at 0.2. Chart on the right has curve parameter at 0.8. Both charts use the input constant at 100%. This equation was applied for the robot's acceleration curve characteristic.

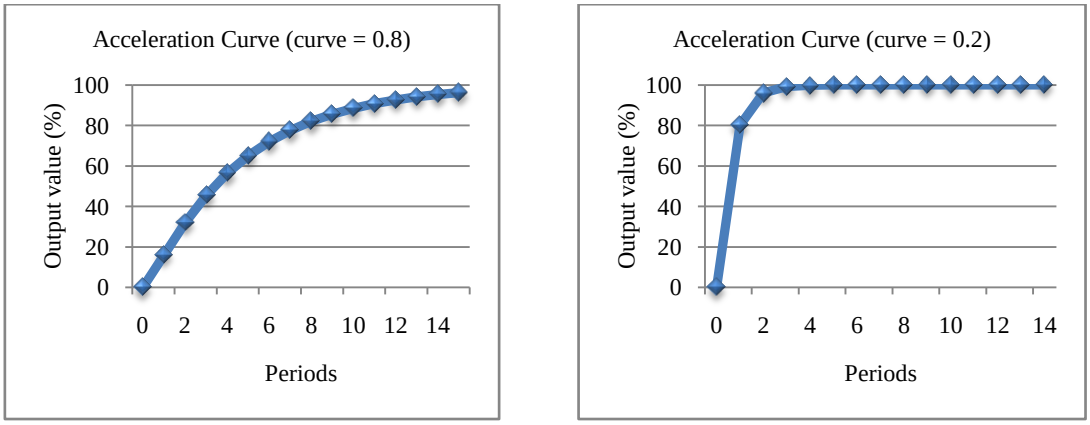


Figure 4.8 - Output charts of a situation where curve=0.2 (on the right) and curve=0.8 (on the left) and both with constant input at 100%.

As an example, if period time was set to 100ms, an IW with curve = 0.8 would take approximately 1.5 seconds to reach the maximum speed, whereas an IW with curve = 0.2 would take only 0.4 seconds, as demonstrated on Figure 4.8. Considering the error 1%, an IW with curve = 0.8 would take approximately 2.1 seconds to reach the maximum speed whereas an IW with curve = 0.2 would take only 0.3 seconds.

In ensuring that this parameter is configurable, the simulator is able to improve the convergence to the real wheelchairs' characteristics, optimizing the environment for control algorithm tests.

4.3.3 – Next Position Calculation

Having modeled the wheelchair's motor response, through it it is possible to calculate the position of the robot on the following period. Using the robot's top speed and the output, given by the motor acceleration equation, the simulator calculates the robot's linear (eq. 4.5) and angular (eq. 4.6) speeds.

$$LinearSpeed = MaxSpeed * \frac{(RightMotorOutput + LeftMotorOutput)}{2} \tag{4.5}$$

$$AngularSpeed = MaxSpeed * \frac{(RightMotorOutput - LeftMotorOutput)}{height} \quad (4.6)$$

The linear speed will affect the robot's next X and Y coordinates, whereas the angular speed will affect the robot's next orientation. Equations 4.7, 4.8 and 4.9 allow these calculations.

$$X_{n+1} = X_n + LinearSpeed * \cos(\theta_n) * TimeStep \quad (4.7)$$

$$Y_{n+1} = X_n + LinearSpeed * \sin(\theta_n) * TimeStep \quad (4.8)$$

$$\theta_{n+1} = \theta_n + AngularSpeed * TimeStep \quad (4.9)$$

where X_n and Y_n are the current robot's center coordinates, $X_{n+1}$and Y_{n+1} are the calculated values for the robot's next coordinates, θ_n and θ_{n+1} are current and next values for the orientation and TimeStep is the time period of time, in seconds, between calculations. This later parameter is used so that the robot's speed is the indicated by the agent at registration, may be in meters/second. Finally it is relevant to mention that the orientation angle is normalized to the ranges from -180º do + 180º and this restriction must be taken care of at the time of the next angle calculation. In a case where θ_{n+1} calculation results in a value over +180, it is transformed by subtracting 360º. Similarly, if it decreases under -180 it will be added 360º.

Every period, the next position of the robot is calculated and then, the values are used to move the robot. Although being its main usage, this calculation serves other purposes, such as aid in the collision algorithm. The collision detection must be tested with the next and not the current robot's position. If it was not as such, the limit that is made to the robot's movements, while in collision would permanently block its movement.

4.3.4 – Collision Detection

The starting point for the adaption of the DCOSS version of "Ciber-Rato" Simulator was the conversion from circular to rectangular body robots. The main usage of the robot's body is in the collision detection verification. A robot's shape has to be defined in mathematical equations that will enable the detection of intersection with other objects. In this simulator there will only be modeled 2 types of objects: walls and robots. Therefore collision checking will only have to be performed with these two types.

Originally, for Robot-Robot collision checking, the "Ciber-Rato" simulator checked whether the distance that separates the robots' centers (through X and Y coordinates) was smaller than two times a robot's radius (all robots were circular with the same radius). This simple algorithm is not applicable for different radius robots neither for rectangle shaped robots. The wheelchairs' size and position on the map were now modeled by four parameters:

Center of movement point, the wheelchair's orientation angle, the width and height. Through this information all the robots' corners coordinates can be calculated. Using this information the new collision detection algorithm is as follows:

- Using pairs of corners as line segment defining points, it is calculated an equation for one line segment for each robot.
- The intersection point of the two lines is calculated. If lines are parallel no point is calculated for there is no intersection.
- Both X and Y coordinates are checked to find whether they are located within each robot line segment. If so, then there is a collision between the two robots.
- This process is repeated until the 4 lines of each robot are checked with the lines of every other robot.

Figure 4.9 gives an example on one of the tests performed during this algorithm.

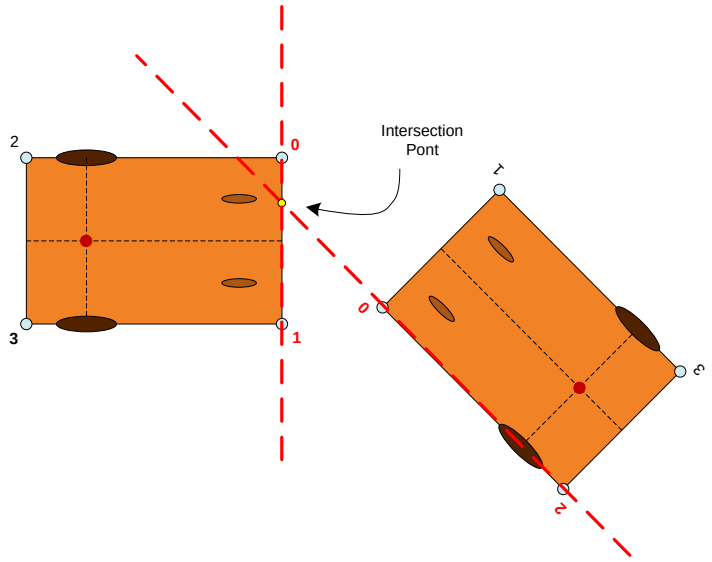


Figure 4.9 - Line Intersection Verification

The line defined by corners 0 and 1 of wheelchair A is being checked against the line defined by corners 0 and 2 of robot B. The intersection point is within the line 0->2 segment of robot A, but it is outside the line segment of robot B. Therefore, no collision is detected. Every robots' lines must be checked with the other robots', a cycle illustrated in Figure 4.10. During this cycle the "Determine Intersection" function is called (taking two corners of each robot as parameters) and returns true or false, depending whether a collision was identified. The "Determine Intersection" function is represented in UML diagram of Figure 4.11.

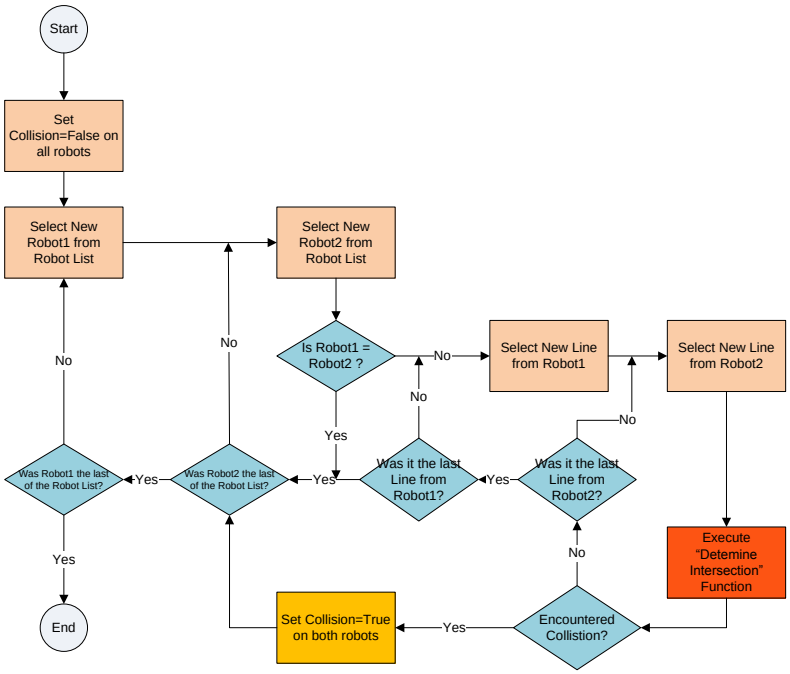


Figure 4.10 - Cycle for checking collision on all lines from every robot

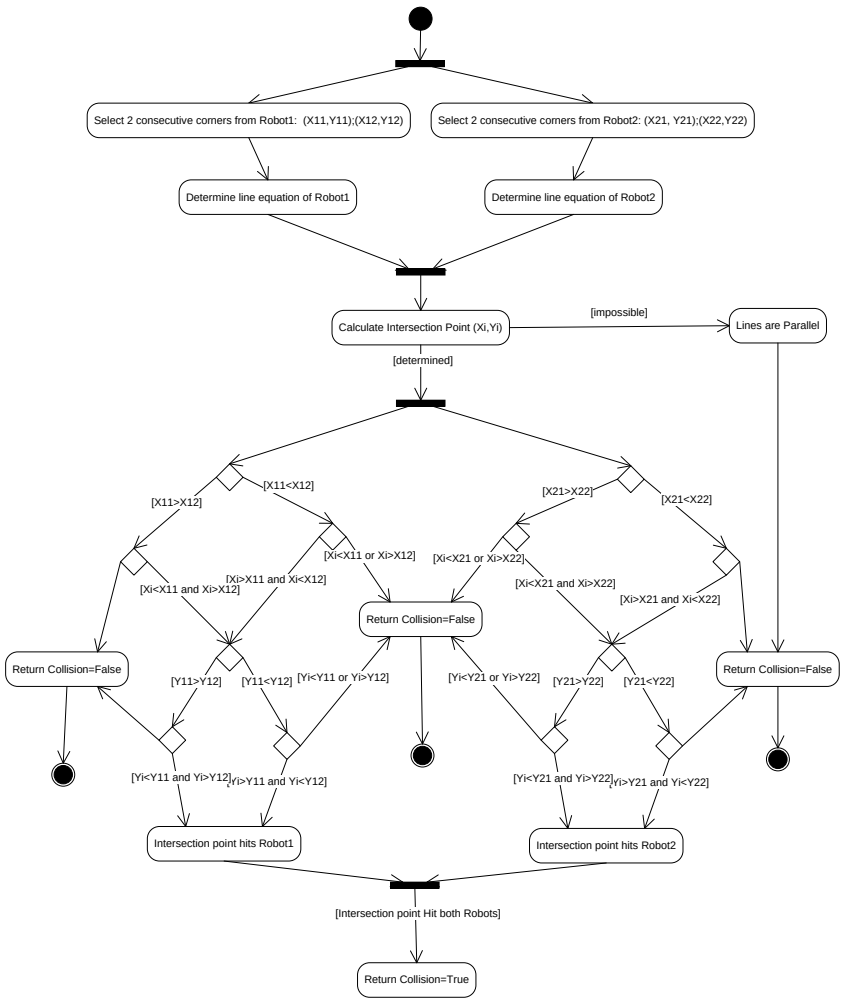


Figure 4.11 - UML diagram of the "Determine Intersection" function.

Robot-Wall collision checking also endured structural changes but, after concluding the robot-robot check it was a simple implementation. In the simulator, walls are stored in an array of walls and each wall is an array of corners (a corner is defined by its X and Y coordinates). Hence, the concept of the wall collision check algorithm is similar to the Robot-Robot. Every two consecutive corners define a line segment that will be checked with the robots line segment. If the intersection point is within the two segments, then a collision exists.

4.3.5 – Proximity Sensors

The proximity sensor positioning was the next functionality to be adapted. Originally, the Ciber-Rato simulated an infra-red sensor that could only be positioned in the perimeter of a circle, with a fixed cone of sight and a fixed direction, radial to the robot. To be true to the rectangular form, the sensor definition was modified, as illustrated in Figure 4.12.

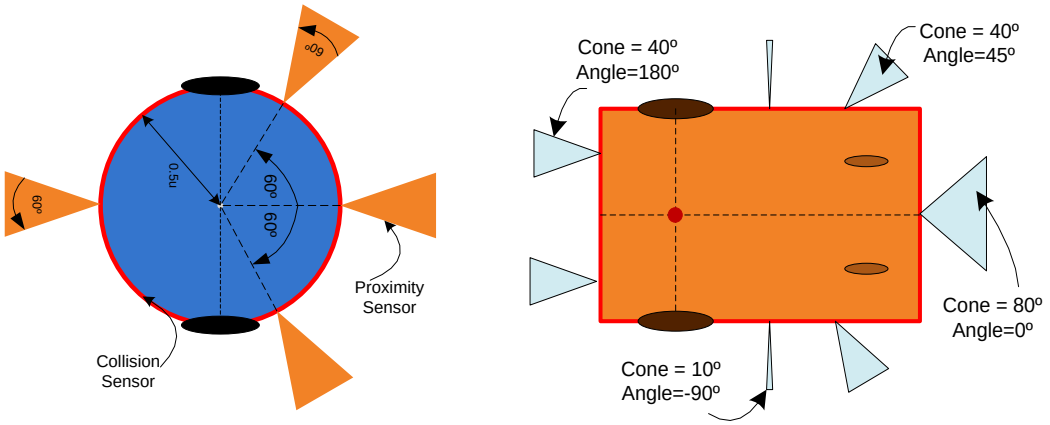


Figure 4.12 - Comparison between the modeled body's and sensors of the simulated robots in Ciber-Rato (a) and Intellwheels (b).

It can now be configurable by X and Y coordinates, relatively to the robot's movement center, and both the cone of sight and the direction can be redefined. All these parameters are now configurable by the agent, at the time of registration with the simulator. The sheer modification of enabling configurable cone allows the agent to register different proximity sensors. A wider cone would resemble a sonar proximity sensor whereas as thinner cone would be more similar to an infrared proximity sensor.

To better approach to the real IW used in LIACC, the simulator raised the robots' proximity sensor number to 8.

4.4 – XML Communications

The Extensible Markup Language (XML) is now a widely used standard, mainly due to its characteristic of facilitating communications across different systems[30]. Specifically in the Intellwheels Simulator's environment, it is expected that different applications, developed in different platforms exchange data in an easy human understandable way. Ciber-Rato originally was set for its usage and, with proven success, the concept endured in Intellwheels Simulator.

XML tags were defined for every kind information exchange between the simulator and the agents.

4.4.1 – Registering Physical Characteristics

The first action that a robotic agent should take is to register itself with the simulation server, through the UDP protocol. In order to so, a XML message must be sent to port 6000 of the IP of the computer that is running Intellwheels simulator. This registration can be as simple as a single XML tag containing the IW's name or as complex as a full clarification of all its characteristics. Table 4.3 details all the tags that can be inputted at registration.

Table 4.3 - XML tags for robot registration

Tag	Definition	Type	Range	Default Value
Name	robot's name	string	Up to 20 characters	N/A
Id	Robot's Id number	Integer	[1;Map Grid]	Simulator
Height	Robot's Height	float	>0 (meters)	1.0
Width	Robot's Width	float	>0 (meters)	1.0
COM	Robot's Center of Movement	float	]0;1[	0.5
X	Robot's starting X coordinate	float	>0 (meters)	Map Grid
Y	Robot's starting Y coordinate	float	>0 (meters)	Map Grid
DIR	Robot's staring angle in degrees	float	[-180;+180] (degrees)	Map Grid
Type	Type of agent connecting	N/A	{Simulated, Real, Door}	Simulated
MaxSpeed	Robot's top speed	float	>0 (meters per second)	0.5
AccerelationCurve	Robot's acceleration curve	float	]0;1]	0.5

Apart from the Name, all the other XML tags will assume a default value if they are omitted. Figure 4.13 exemplifies a registration message that the simulator would acknowledge and accept.

```
<Robot
Name="IntellWheels" Id="1" Width="1.0" Heigth="1.0" DIR="0.0" COMass="0.5" Type="Real"
X="13.0" Y="7.0" DIR="0.0" MaxSpeed="0.5" AccelerationCurve="0.7">
</Robot>
```

Figure 4.13 - Intellwheels robotic agent registration XML message

4.4.2 – Registering Sensors

Intellwheels allows registering up to 8 robot sensors wherever they are needed. Usually an IW will have their proximity sensors located near the perimeter, but they can also be placed more to its inside. The sensor's positioning is now made through a definition of their X and Y coordinates relative to the robot's center and their cone of sight and orientation can also be configured. The registering of sensors must be done at the initial robot registration with the server and with all the tags shown in Table 4.4.

Table 4.4 - Sensor Registration XML tags

Tag	Definition	Type	Range
Id	Unique identification	integer	[1;8]
X	X coordinate of the sensor position, relatively to the center of the robot	Float	>= 0 (meters)
Y	Y coordinate of the sensor position, relatively to the center of the robot	Float	>= 0 (meters)
Angle	Angle, in degrees, of the sensor direction.	Float	[-180;180] (degrees)
Cone	Arc of sensor vision, in degrees.	float	]0;180[(degrees)

In order to simplify the future development of controlling agents, virtual sensor will be created and placed even if its registration is not done. The simulator will place the sensors in accordance with the physical characteristics of the wheelchair. Four sensors will be placed on the perimeter with default relative positions, cones and orientations. Table 4.5 details this positioning, which is illustrated in Figure 4.14. It is important to notice that sensor registering and positioning is independent of the robot type. They shall be created and placed in their positions, independently if the agent is a simulated robot controller or a real robot one controller.

Table 4.5 - Default sensor characteristics

ID	X (m)	Y (m)	Angle (°)	Cone (°)
0	width * COM	0.0	0.0	60
1	0.0	0.5 * height	90	60
2	0.0	-0.5 * height	-90	60
3	- width * (1-COM)	0.0	180	60

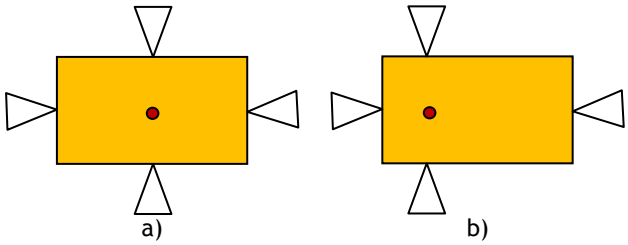


Figure 4.14 - Default sensor location with COM=0.5, a), and COM=0.8, b).

Even though these default sensor will be placed, if sensor individual positioning is intended, that information must be sent together with the robot's physical characteristics information. Figure 4.15 is an example of a XML message containing sensor information that is accepted by the simulator.

```
<Robot
Name="Teste" Id="1" Width="2.0" Heigth="1.0" DIR="0.0" COM="0.5" X="11.0" Y="11.0">
<IRSensor Id="0" X="1.0" Y="0.0" Angle="0.0" Cone="60"/>
<IRSensor Id="1" X="0.0" Y="0.5" Angle="90" Cone="50"/>
<IRSensor Id="2" X="0.0" Y="-0.5" Angle="-90" Cone="30"/>
<IRSensor Id="3" X="-1.0" Y="0.0" Angle="180" Cone="60"/>
</Robot>
```

Figure 4.15 - XML message for robot registration sensor definition

4.4.3 – Moving Robot

After robot registration (independently of its type), every communication to must be sent trough the new UDP port specified by the simulator. Moving the robot is also done through XML messaging, however, it now depends on its type.

For a robot type "Simulated", the simulator will be responsible for new position calculation. It will also handle the modeling of the motors and thus the translation of the input power to robot's speed. As such, to move the robot, its controlling agent must send a

XML message, sending the power values to attribute to each motor, as exemplified in Figure 4.16.

```
<Actions LeftMotor="0.1" RightMotor="-0.1" />
```

Figure 4.16 - XML message example for motor order

The power values range from -0.15 to 0.15. During the development of this project it was considered to change this range to -100 to + 100, as it would set the power setting more obvious values. On the other hand, this modification would invalidate the usage of agents developed for the original "Ciber-Rato" to control the motors. Because of this, the original range was kept.

In case the connecting agent is a real IW controller (in augmented reality mode) the simulator relinquishes the task of position determination to the agent itself. Conceptually, the simulator in working in augmented virtuality mode and so, the agent must inform it, at all times, it's X and Y coordinates (in meters) and orientation (in degrees). Through this action it is possible to allow interaction between the real and virtual world, particularly updates on virtual sensor value calculation and collision detection. The XML message that the agent must send to the simulation server is as illustrated in Figure 4.17.

```
<Actions X="10" Y="5" DIR="45"/>
```

Figure 4.17 - XML message of type real robot position information

4.4.4 – Viewer Agent

One of the key features of Intellwheels is the possibility of connecting a viewer agent, to visually represent the simulation. The application is external to the simulator itself, alike the robotic agents, requiring only the usage of UDP/IP protocols to exchange XML messages. Regarding these messages, they should be sent in a particular order, so that the visualization is proper. Once again, it was intended that the Ciber-Rato viewers could be applied to this simulator and, therefore, the sequence of XML messages is identical to the Ciber-Rato's, illustrated in Table 3.2. To illustrate, Figure 4.18 gives an example of the initial registration message sent from the viewer agent and the simulator's response.

Agent sends:	Simulator Responds:
<View/>	<Reply Status="Ok"> <Parameters SimTime="1800" CycleTime="1000" CompassNoise="2" BeaconNoise="2" ObstacleNoise="0.1" MotorsNoise="1.5" RunningTimeout="1350" GPS="Off" ScoreSensor="Off" ShowActions="False" NBeacons="1" RequestsPerCycle="2" ObstacleRequestable="On" BeaconRequestable="On" GroundRequestable="On" CompassRequestable="On" CollisionRequestable="Off" ObstacleLatency="1" BeaconLatency="5" GroundLatency="1" CompassLatency="5" CollisionLatency="1" BeaconAperture="3.141593" /> </Reply>

Figure 4.18: XML messages of viewer registration

It is appropriate to mention that, although Ciber-Rato remains connectable to Intelwheels simulator, not all information sent to it is used (such as robot's physical characteristics). In spite of this, the visualization may not be accurate but can still give some rough visual information on the simulation.

4.4 – Summary

The Intellwheels Simulator, expanded the Ciber-Rato project it is was based on, acquiring important features which are critical for intelligent wheelchair simulation.

This chapter stated by giving an overview on the intellwheels conceptual architecture and the technology it used for the development and implementation. It presented the concepts of the multi-agent system and the support for external application connection. Intellwheels provides a new mode of IW simulation where it is possible to connect not only agents for virtual robots but, at the same time, real IW controllers which can, themselves work on augmented reality mode. The simulator, on the other hand, will be under an augmented virtuallity environment, receiving information of real wheelchairs and calculating their interaction result with the virtual objects modeled.

Every robot is modeled with a rectangle shaped body, with configurable center of movement, height and width. Additional physical characteristics, different in every electrical wheelchair, are also adaptable, such as the acceleration curve and the maximum speed it can achieve. A new algorithm was developed for the key function of collision detection, taking the rectangle shape on consideration. The proximity sensors can be defined by their opening cone of sight and their orientation, in degrees, and can be placed through X and Y coordinates relatively to the robot's center. Finally, the simulators XML messages were detailed, exemplifying how robotic agents and viewer agents register and communicate with the simulator.

Chapter 5

Simulation Agents

During the development of the Intellwheels Simulator, a need for agents that would test the implemented algorithms was evident. Although the first objective was to modify and adapt the Intellwheels Control Agent (previously developed by LIACC), so it would be able to connect with the simulator and control the virtual robot, other simpler and more generic controlling agents where created.

The applications were built with Borland[37] Delphi 7. Delphi 7 is an integrated software development environment that allows visual, event-oriented programming through Pascal programming language[38]. The main reason for its decision was related with assuring homogenous software usage throughout most of the Intellwheels models (detailed in Chapter 2).

This chapter will be initiated by explaining the basic Intellwheels agents developed for simulator testing purposes and shall finalize by presenting the project's main control agent, with its features, possibilities, adoptions and operational modes.

5.1 – Simple UDP Agent

To overcome the initial difficulties in the Ciber-Rato study, an agent was developed, in Delphi 7, to test UDP communications with Ciber-Rato as well as XML messaging. This application was also to work as a base code for communication handling in all robotic agents to be developed afterwards.

The main purpose of this application was to have the ability to connect to the original Ciber-Rato as a robotic agent and as a simulation viewer agent. As additional functionalities, it is be able to create a text file with the XML messages sent and received and four movement buttons (up, down, left, right) to control the robotic agent, shown in Figure 5.1.

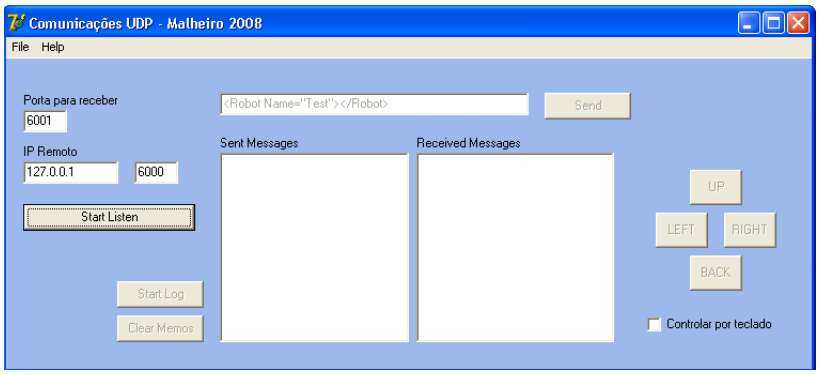


Figure 5.1 - Simple UDP Agent

Concerning, the UDP connection, the main difficulties encountered were related to the definition of the IP ports. Delphi 7 contains a UDP component (from the TidUDPServer class) that can be "drag and dropped" onto the form, instantiating it, remaining the task of its configuration. The initial step, to register, is taken by the agent, as it defines a listen port and a destination port and sends the registration XML message, ensuring that the local listen port is the "Source Port" in the UDP datagram, as shown in Figure 5.2[39], for that will be the port that the simulator will respond to.

The simulator, by default, is listening to every communication sent to port 6000. Once it receives a message, it is analyzed and checked for a Robot or Viewer agent XML registering message. If the message does not match with any of these, the message will be ignored. In a case it is a robot connecting and if the Id is specified, it must not already be in use in the simulation, otherwise the registry will be denied.

In a successful registration, the simulator will send an XML message confirming it. In this first message, the UDP datagram sent will specify a new port, to which every robot action (or viewer command, depending on which agent type connected) must be sent.

+	Bits 0-35	16-31
0	Source Port	Destination Port
32	Length	Checksum
64	Data	
...	...	

Figure 5.2: UDP packet header structure

The simulator binds the robotic agent's sending IP and port to this new port. No communication can be done to any other port, from this point forward. The main reason for this behavior is to ensure that port 6000 remains free for new robot registration.

Up, Down, Left and Right buttons on this agent simple send an action message to the simulator, as shows Table 5.1.

Table 5.1 - XML messages for agent's buttons.

Button	XML message	Purpose
UP	*<Actions LeftMotor="0.1" RightMotor="0.1"/>*	Move the robot forward
Down	*<Actions LeftMotor="-0.1" RightMotor="-0.1"/>*	Move the robot backward
Left	*<Actions LeftMotor="-0.1" RightMotor="0.1"/>*	Rotate the robot anti-clockwise
Right	*<Actions LeftMotor="0.1" RightMotor="-0.1"/>*	Rotate the robot clockwise

Apart from the messages sent through the buttons, the application allows custom message sending, through an edit box seen at the top of the application (Figure 5.1). This permits that any message can be sent to the simulator, testing its response. All ingoing and outgoing communications are shown in the debug text boxes and can be stored in a "log.txt" file, through the "Start Log" button.

This application was developed to serve as the simplest UDP communications test tool and simulator response validation. Additionally, it proved to be a fine core source not only for more advanced, generic wheelchair robotic simulators but for the Intellwheels Control Agent itself and even the Intellwheers Viewer agent, detailed in Chapter 5. All these applications, in some form, use functions of this "Simple UDP Agent".

5.2 – Wheelchair Robotic Agent

As the simulator evolved into an IW simulator, the requirements to test the algorithms being implemented became more complex, thus requesting the development of an agent that would validate those modifications. Direct implementation on the existing Intellwheels Control Software would implicate increased difficulty due to the current complexity of the application. For that reason the solution was to expand the previous "Simple UDP Agent" into a Wheelchair Robotic Agent. Its objectives are:

- Allow multiple robotic connections in only one application;
- Permit the manual definition of the physical characteristics of the wheelchair;
- Customize the sensor's positioning, cone and orientation;
- Enable the option for definition of the robot's initial position on the map;
- Give Visual information of the virtual sensors readings;
- Admit simple, low level, controlling of the wheelchair through buttons and the arrow keys on the keyboard.

To implement the multiple robot connection, the answer was to dynamically create instances of the Delphi UDP component. A solution that would solve not only this requirement but also allow storing of different IWs' individual information is to create a class. The TRobot class was developed and it would store all the information concerning the robot's physical characteristics and communication related information. Figure 5.3 is the UML diagram

representation of the implemented TRobot class, indicating the major attributes and operations involved. Programming robustness was taken into consideration, therefore the class's attributes were created as protected attributes, and operations were created to allow secure access.

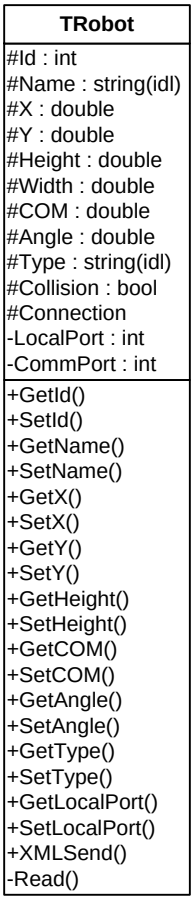


Figure 5.3 - TRobot Class

In a situation where this application has multiple robots connected to the simulator, each instance of the TRobot class will have a unique listen port (enforced by the simulator) and the information will be stored only into the robot that matched that specific port. Figure 5.4 is a print screen of the application, in a moment where three robots were connected to the simulator.

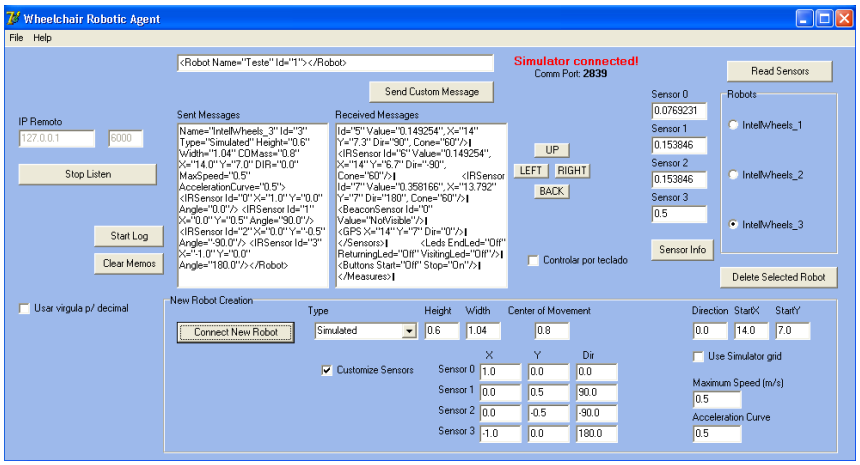


Figure 5.4 - Wheelchair Robotic Agent

Individual selection, information view and control of each robot is possible through a combo box that keeps a list of the TRobot instances. Once a movement button is pressed, or a custom message is sent, the application uses the XMLSend operation of the selected TRobot instance to do it. Similarly, once an instance receives a message to its port, a check of the combo box selection is made and, in case the selected one matches the received information, it will update the values on the main form of the application.

Although it possesses no intelligent behavior (being simply reactive or pre-planned), this application proved to be valuable in simulator testing for XML messaging handling as well as for the core algorithm implementations detailed in Chapter 4.

5.3 – Door Agent

To provide a wider variety of testing fields for the IW control algorithms, another type of agent was allowed to connect to the simulator. The type “Door Agent” modeled doors, for they are key objects in this type of simulation environment. Since every parameter of the robot’s rectangle form is now adjustable by each agent, it is possible to turn a robot with a very small height, proportional width and a center of movement at the bottom of the robot (e.g.: COM=0.99) into a conventional door. The similarities are such that this application is itself an adaptation of the Wheelchair Agent presented before.

Although it would be possible to create additional restrictions to a door type agent’s movement inside the simulator software, this solution was declined, allowing the agent itself to limit the movement (by the virtual left and right motor power orders). This way, it is possible to create any kind for door needed. To test and use this functionality, a Door agent was created with the capability of connection of multiple predefined and custom doors.

The predefined types created were:

- Normal Door
    - Height=1.0m; width=0.1m; COM=0.99;
    - Left Motor power = - Right motor power
- Sliding Door
    - Height=1.0m; width=0.1m; COM=0.99;
    - Left Motor power = Right Motor power
- Rotating Door
    - Height=1.0m; width=0.1m; COM=0.5;
    - Left Motor power = - Right motor power

A low level button was created that would open and close the door, depending on its type. Figure 5.5 illustrates the implemented algorithm for this control button.

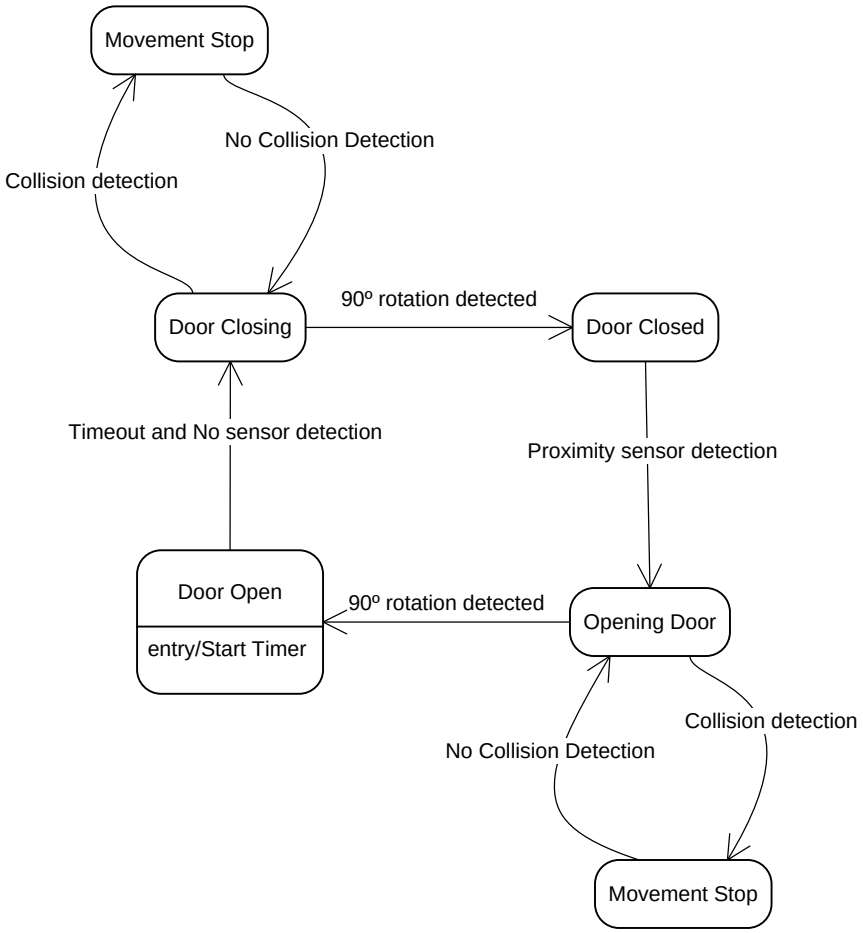


Figure 5.5 – UML stat diagram of the door control algorithm

To demonstrate the extra capabilities of the door as an intelligent agent, sensor treatment was implemented into the application for automatic door open usage. If the door's

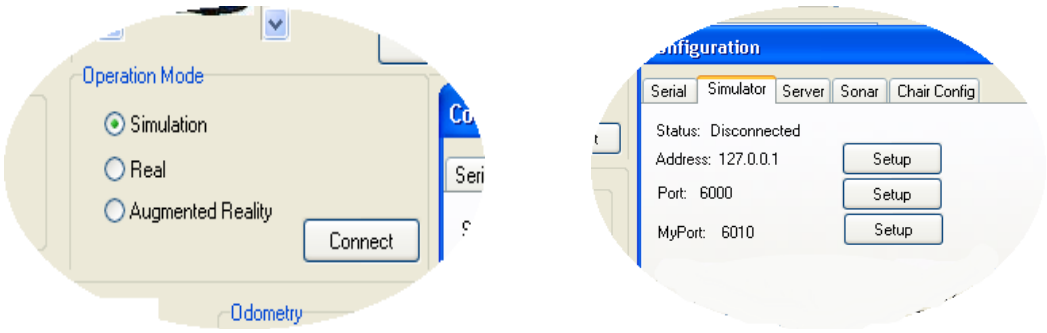
proximity sensors indicate an object close enough, it would automatically open the door. This behavior is similar to the behavior of a common automatic door. So, after a fixed period of time without sensor detection, the doors will close. Moreover, through the collision detection sensor, the door stops its movement in case of contact, waits a determined period of time and then tries to proceed the previous action.

Possibilities for an intelligent robotic door agent are vast and in the last chapter of this dissertation, future work on this matter is proposed. A sample of the capabilities is the communications module. A particularly interesting feature is to open or close from received communications rather than by the proximity sensors. In fact, with integration with the IW's controlling agent, the door could open as a part of a trajectory plan of the wheelchair. This would set the environment more intelligent, efficient and more secure.

5.4 - Intellwheels Control Agent

One of the main objectives of this dissertation is to allow the same controlling software to be used for real wheelchair control, virtual wheelchair or both (simultaneously). Furthermore, this should be done in such a fashion that the same medium level algorithms could be used transparently to the hardware (or virtual hardware) to which the software is connected.

The software itself was originally developed already with the intent of, later, adding the simulation features. This was done by the creation of the simulator configuration tab (Figure 5.6-b.) and the operation mode selection (Figure 5.6-a.).



a. b.

Figure 5.6 - Simulation-Ready structures on Intellwheels main application (adapted from [1])

These design spaces were left with no code within and, apart from the connection handling, the modifications to this software went deeper inside, onto sensor information treatment, motor power order decisions and simulation communication requirements.

5.4.1 - Virtual Reality Mode

The first modification implemented on this agent was the adaptation to pure virtual environment control, which corresponds to Simulation selection in the "Operation Mode" (Figure 5.6-a). The control applications connect to the simulation server through UDP and IP. The simulator will provide this application world perception through GPS sensor (for positioning), compass (for orientation), virtual sonars and IR sensors (for obstacle detection), as Figure 5.7 represents.

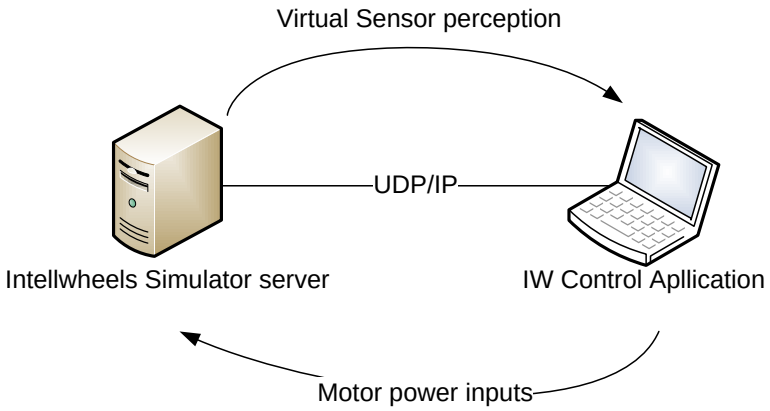


Figure 5.7 - Simulated Operation Mode

The advantages of this operation mode are the testing possibilities it allows. It is possible to test the functionality of the control algorithms in a sensor error free environment and, on the other hand, test error treatment excluding other wheelchair movement tribulations.

5.4.2 - Real Mode

Real wheelchair connection was already implemented at the beginning of this project. It consists on a RS232 connection with the acquisition boards on the wheelchair for sonar and IR sensor readings and motors' power inputs send, illustrated in Figure 5.8.

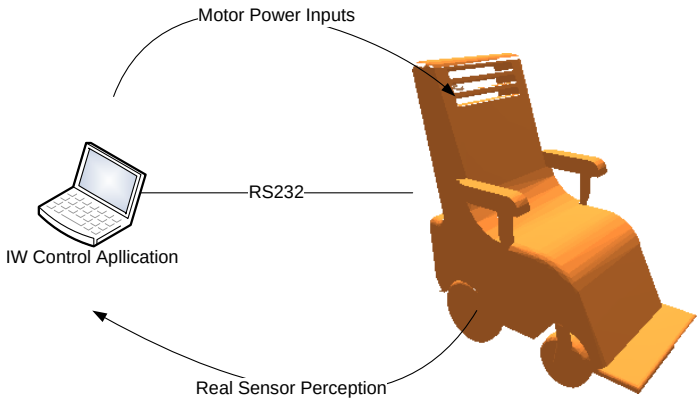


Figure 5.8 - Purely Real Operation Mode

The control of the chair can either be manual or in an automatic mode where the power inputs are defined in function of the planned task. The greatest inconvenience of this mode is the lack of reliability that real sensors can offer. The sonar and IR sensors have random reading errors and, on some objects and surfaces, they might even fail to detect anything at all (e.g. tables, black or rough surfaces).

The chair's localization is made through odometery. The impulses read on each encoder will allow left and right wheels speed calculation which, no its turn, permits angular movement as well as X and Y variations. Once more, this is a real sensor and, in case of wheel sliding, impulse readings induce into false speed calculations, thus making the wheelchair "lose" its position.

5.4.3 - Augmented Reality Mode and Sensor Merging

Final operation mode (Figure 5.6) will make the application connect simultaneously to the real wheelchair, through RS232, and to the virtual world of the simulator, through IP and UDP. This mode requires the localization calculations, made through real wheelchair sensors, to be very precise as the virtual world generated by the simulator relies solely on that information for real wheelchair incorporation.

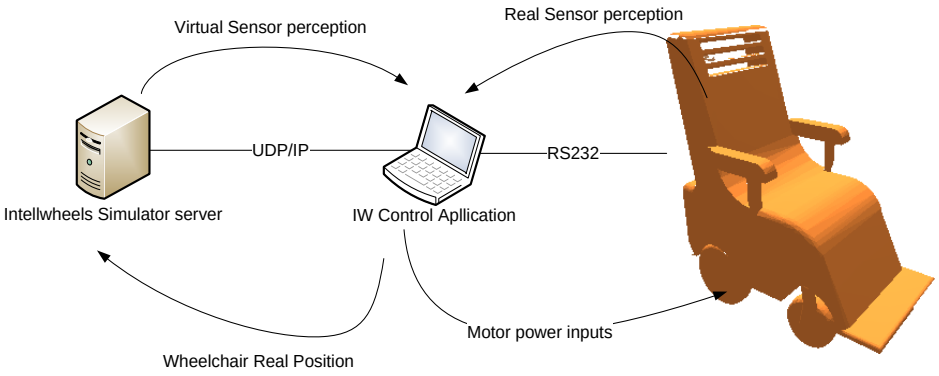


Figure 5.9 - Augmented Reality Operation Mode

As the simulator creates a virtual projection of the real chair on its world, it is able to place virtual sensors onto it and provide the control software with additional perception information. In addition to the real sensors, control algorithms may count with error and variation free information that the simulator can provide.

Summing up, the IW control application will receive real and virtual sensor information and send the wheelchair's position to the simulator and motor power inputs to the real wheelchair.

During the augmented reality operation mode this application will treat information from virtual and real sensors. The challenge is how to merge all the information to achieve the best possible control decision. Since this is a project that deals with humans, the information merging algorithm implemented followed a safe, conservative ideology. If the virtual and real

sensors offered different perceptions on the world around, the control should decide using the values that better protect the chair (and the patient himself). As example, if the virtual proximity sensors read an object 30cm away in front, and the real sensors read the same object at 60 cm, the closer value is the one to be used in the decision for motor power input. Figure 5.10 graphically represents this algorithm.

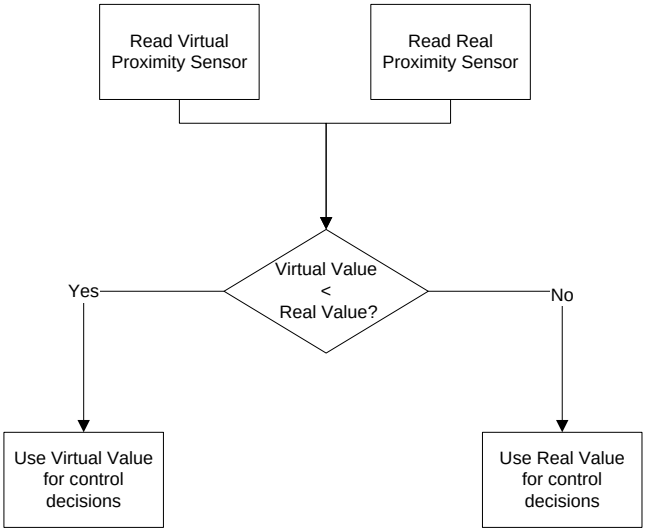


Figure 5.10 - Sensor Merging Algorithm

5.4.4 - Simulator Connection

The development of the simulator allowed parameter configuration that was not foreseen in the initial stages of the application development. As detailed in chapter 4, the registration of the wheelchair with the simulator should include all of its physical characteristics. Therefore, a modification to the simulator configuration form was made, to allow the parameter configuration (Figure 5.11).

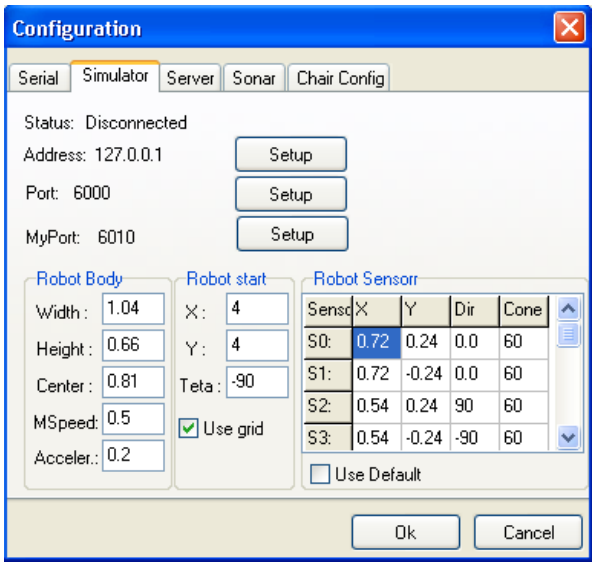


Figure 5.11- Simulator Configuration Form

Since it was intended to model as closely as possible the real wheelchairs, the prototypes in LIACC were measured resulting in the definition of their physical characteristics. Comparison between virtual and real sensors was also looked-for. As a result the location of every sensor was measured (relatively to the wheelchair's center of movement). This information must also be sent in the initial connection with the simulator. Figure 5.12 shows the XML message to be sent for full IW agent registration.

```
<Robot
        Name="WheelChair1"
        Type="Simulated"
        Height="0.66" Width="1.04" COM="0.81"
        MaxSpeed="0.5" AccelerationCurve="0.2">
        <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>
```

Figure 5.12 - XML message for IW registration, for prototype wheelchair.

5.5 – Summary

This chapter discussed the developed agents for connection with the simulator. The agents were vital in various stages of the simulator's development, even in the initial Ciber-Rato study, to which the Simple UDP Tool intended.

The Wheelchair Agent was later developed, as the simulator was able to accept full electric wheelchair characteristics (from size, center of movement, acceleration to top speed and sensor definition). This agent is also able to perform basic commands for movement (forward, back, turn left and right) which allows full test of the Intellwheels capabilities for wheelchair simulation.

One of the main objectives of this dissertation was to be able to have one unique application for real and virtual wheelchair controlling. This was successfully completed and moreover, and augmented reality feature was able to be implemented, where the application is able to receive and reason with virtual and world information, simultaneously.

Chapter 6

Intellwheels Viewer

Visualization is of great significance in simulation in the sense that it is a mean to easily understand a large quantity of information, which would otherwise be too great or complex for most people to fully grasp. Graphical representation is now taken for granted and it would be unconceivable to develop a simulator without some sort of visual illustration. Humans construct and comprehend the world in a graphical way for we have an innate ability to process graphic information in a preconscious, involuntary fashion, similar to breathing[40]. Visualization is, therefore, the foundation for our understanding. In spite of its importance it is critical to ensure quality in a few elements, when developing simulation graphics:

- There must exist good interactivity during the simulation, in order to display the information the user intends to see;
- Skepticism to computer generated images is still very high and to provide credibility, it is necessary to ensure realism;
- Associated with the realism factor is the animation's performance. If jumps or glitches are seen, it is difficult to extract conclusions;
- The animation must be flexible enough to enable and disable parts of it, avoiding heavy computational and visual efforts.

Taking these concepts into consideration, is becomes clear that the original viewer for Ciber-Rato would not fit the needs of the Intellwheels Simulator. The decision to develop a new viewer from scratch was made based on various factors:

- Ciber-Rato Viewer was not flexible for it was focused on a competition environment and, even more critical, restricted to robots with the same circular shape and radius. Visualizing rectangular shapes, with variation height and width, was not possible;
- The drawing of the robot's themselves was done through the loading of bitmap images which invalidates the possibility of dynamic modifications, through the simulator;
- Taking realism into consideration, a 3D display of the simulation became important. It would be possible to view the entire simulation as if one was sitting on the real wheelchair. Ciber-Rato did not contain a 3D display option and modification of the

original source code would be more difficult and more time consuming, without better final results.

6.1 – Architecture

Conceptually, the viewer developed contains 5 main software modules (Figure 6.1).

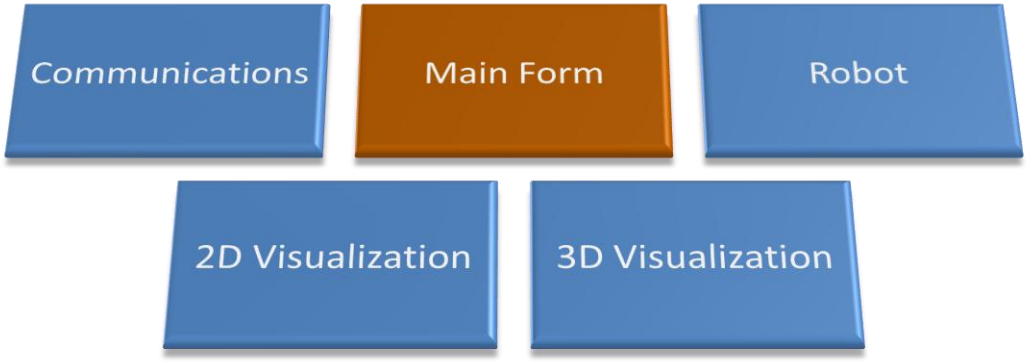


Figure 6.1 - Intellwheels Viewer Architecture

The main module (main form) is where everything comes together. It permits the communication configuration activation, robot selection (for selected information display) and visualization selections. This main module is also responsible for storing the information concerning the map's characteristics and wall definition. The map and wall information is sent from the simulator to the viewer at the beginning and that is the information that will be used for the drawing modules. It is appropriate to mention that since the simulator sends the wall definition by their corner definition that concept will be kept.

The communications module contains the IP/UDP configurations and handling and XML message parsing. This module ensures that the messages sent by the simulator are correctly received and transformed into system variables for direct usage from the other modules.

The Robot module is where all the robots' information is stored. Their physical characteristics, status, position and orientation are kept and secured here. It is through this module that the rest of the application will access updated and ordered information on the robots, either for show purposes or for calculations.

The 2D and 3D modules have similar functioning modes. They access the map and robot's information and reproduce them graphically. The only special characteristic of the 3D module is that is calls and uses external OpenGL libraries[41][42]. Since the simulator only provides 2D information, the 3D viewer will generate the Z axis coordinate in such a wall that will allow realism and good simulation visualization at the same time.

In terms of sequence of events, after the main form is created, the only operation allowed is the configuration of the UDP parameters. Once the task is done (or default values are accepted) the application must send the simulator a registration XML message, from which it will receive a confirmation. The next steps are to request map information after which robot information will continuously be sent, with the frequency defined for simulation step. When requested, the 2D and 3D display modules will graphically draw the information already stored in the robot and map modules.

Alike the agents, this viewer was also developed with Delphi 7[37][38], ensuring, once again, consistency on the projects requirements of programming software.

6.1 - Main Form

The main module (main form) is where every configuration parameter can be adjusted, starting from the local UDP listening port and the IP and port for the remote simulation sever.

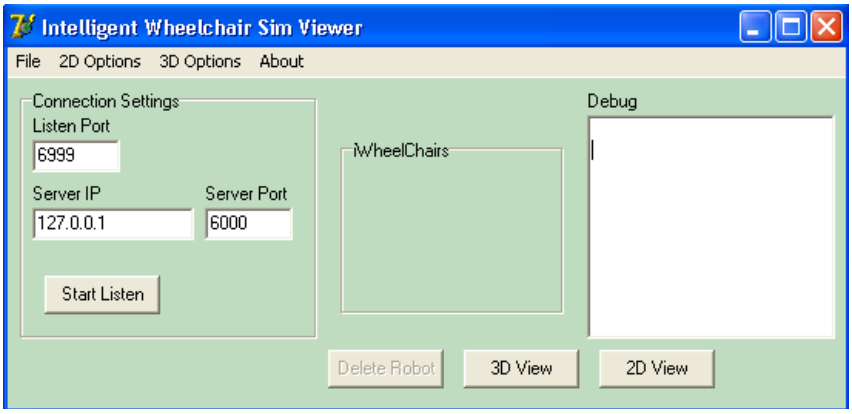


Figure 6.2 - Intellwheels Viewer Main Form

When the user presses the “Start Listen” button, the UDP component will be activated, and immediately send a registration XML message to the server (as detailed in chapter 4, section 4.4). This form allows modifying parameters specific for 2D and 3D viewing (which will be explained later on this chapter) and contains a combo box where every Wheelchair is listed. The selected wheelchair on that list will have its information (physical characteristics, status, position and orientation) updated on the memo box at the side.

6.2 – Communication Handling

Communication between the viewer and de simulator is made through the UDP protocol and XML messages. Similarly to the robotic agent connection, the viewer must connect itself by sending a registration message to port 6000 of the server. The response of the server will be made through a port that will, from then on, be bind to a unique application. As a consequence, every communication to the simulator from this specific viewer must be done

through that new port. A good example is the map information request. If the simulator responded from port 4000 to the registration, the map request XML message must be sent to the port 4000 of the simulator.

After this initial adaptation, the simulator will continuously send information on the robots within the simulation. A XML message of robot information update (exemplified in Figure 6.3) is sent every simulation step, which ensures the real time characteristic of the visualization.

```
<Robot

 Name="Teste" Id="1" Height="1" Width="1" CenterOfMovement="0.5" Type="Simulated"
Time="0" Collisions="0" Collision="False" State="Stopped">
<Position X="6" Y="11" Dir="0"/>

</Robot>
```

Figure 6.3 - Viewer XML received message of robot information

The simulator sends a message for each robot every time step. For example, if there are 5 IW agents connected to the simulator, it will send the viewer 5 XML messages with every information needed for robot drawing: position, orientation, dimensions, COM, type and collision details.

The same XML parser used in the robotic agents (Chapter 4) was used in this application (proving the advantages of maintaining the same programming software through the various applications of the project.

6.3 - 2D Viewer

The simplest form of viewing the entire simulation is in a 2D viewer. It displays information of the complete simulation including every robotic agent.

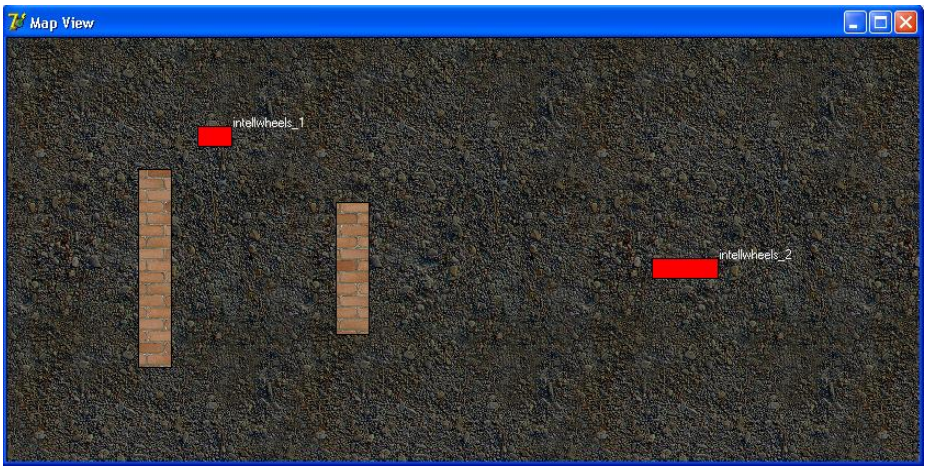


Figure 6.4 - 2D View

6.3.1 – Robot and Wall Drawing

The map's outer limits are draw through the definition of a polygon whose vertexes are calculated with the map's height and width. Figure 6.5 is a section of the source code, in Pascal language, where that functionality is implemented.

```
outerlimits : array of TPoint;
[...]
form2.Canvas.Brush.Bitmap:=Bitmap;
 SetLength(outerlimits,4);
 canvas.MoveTo(0,0);
 outerlimits[0]:=Point(0,0);
 outerlimits[1]:=Point(0,Round(Form1.LabHeight));
 outerlimits[2]:=point(Round(Form1.LabWidth),Round(Form1.LabHeight));
 outerlimits[3]:=point(Round(Form1.Labwidth),0);
 canvas.Polygon(outerlimits);
```

Figure 6.5 - Map outer limit drawing code

Form2 is the base canvas for the 2D drawings. For map outer limit drawing, a rectangle with vertexes of (0,0), (0,Height), (Width,Height), (Width,0) were drawn.

With the communication module, the application received the information regarding wall vertexes and stored. Since each wall is an array of vertexes and a vertex is an array with 2 values type double – X and Y coordinates – map is constituted map an array of array of array of double, as illustrated in the code below.

```
Wall : array of array of array of double;
CurrentWall : array of TPoint;
 [...]
 Form2.Canvas.Brush.Bitmap := Bitmap_wall;
 For i:=0 to Length(Form1.Wall)-1 do begin
   k:=Length(Form1.Wall[i]);
   SetLength(CurrentWall,Length(Form1.Wall[i]));
   For j:=0 to (Length(Form1.Wall[i])-1) do begin
     canvas.MoveTo(Round(Form1.Wall[i,j,0]),Round(Form1.LabHeight- Form1.Wall[i,j,1]));
     CurrentWall[j]:=point(Round(Form1.Wall[i,j,0]),Round(Form1.LabHeight)-Round(Form1.Wall[i,j,1]));
   end;
   canvas.Polygon(CurrentWall);
 end;
```

Figure 6.6 - Implemented code for wall drawing

A wall is drawn by ordering a polygon draw of every vertex when the vertex list ends. The Polygon draw function automatically connects every vertex and prints it onto the canvas.

Although Robot drawing uses the same Delphi "Polygon" function, the calculations required to determine the corner points of the robot are more complex. Since the simulator

only sends information on the position and orientation, the absolute coordinates of the corners must be calculated. The implementation was, fortunately, relatively easy as the algorithm is the same already implemented in the simulator (see 4.3.1). An adaptation from C++ language to Pascal solved the corner determination problem.

6.3.2 – 2D Options

A few options are available in this application, to increase the value of the 2D visualization. Through the "2D options" tab at the top of the application window it is possible to change the zoom of the 2D view, add name labels to the robots and display a unitary grid on the floor.

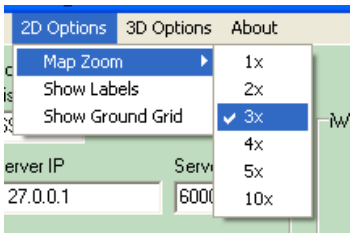


Figure 6.7 - Intellwheels Viewer's 2D Options

In terms of algorithm the zoom change will affect a parameter that multiplies the maps height and width as well as the coordinates of the walls' vertexes, robots' centers and their height and width.

6.4 - 3D Viewer

Although the most critical simulation information is correctly represented in the 2D viewer (correct shape, positioning and orientation of the robots and walls), an illustration closer to reality was considered necessary. Taking Matthew Rohrer's conclusions [40] on the preconscious image processing capabilities of human beings, the visualization for this intelligent wheelchair simulation will as better as its proximity to real visualization (Figure 6.8). Therefore the objective was not only to create a 3D environment but to allow world visualization as we would perceive it while sitting on the real chair.

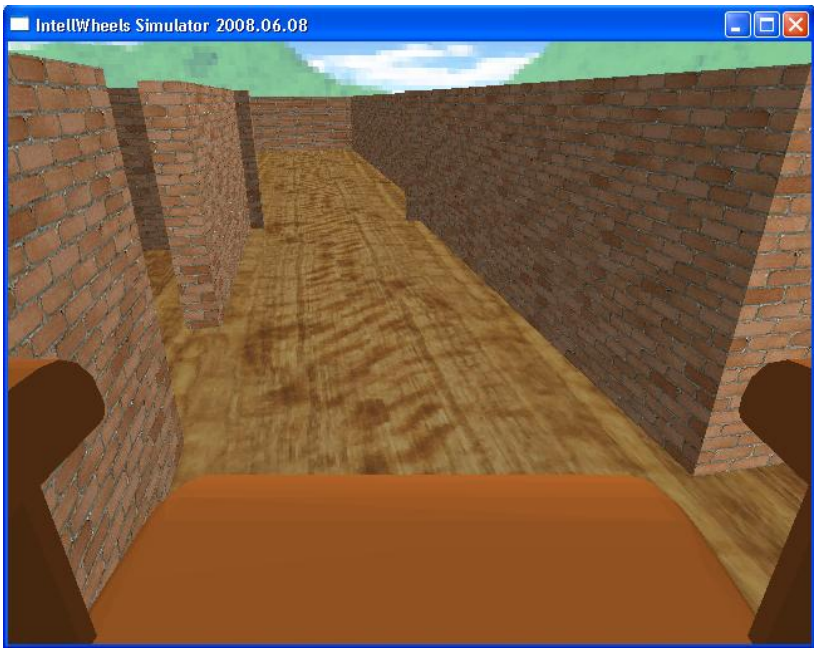


Figure 6.8 - Intellwheels Viewer 3D, 1st person view.

6.4.1 – OpenGL in Delphi 7

Although it is important to maintain the Delphi 7 development environment as base, its native graphical drawing capabilities are somewhat limited. In fact, there are no high level functions for 3D drawing, which would demand unacceptable effort for low-level drawing on a canvas. The solution adopted for was to use OpenGL libraries, within then Delphi 7 environment.

Through OpenGL libraries, it became possible to draw the objects in a very similar way to the 2D mode, on a Delphi form canvas. One can draw the shapes by their polygons and their coordinates, relatively to a given center. The difference is in the extra Z coordinate that needs to be referred however, since the simulation itself is conceptually in 2D, the extra coordinate it purely for visual purposes. As an example, drawing a cube is done by indicating the corner coordinates of each of the six faces. When the camera viewing point is set, the OpenGL motor itself automatically handles the complete redrawing of the shown image. It continues to do so automatically, once the camera view position changes.

In what concerns drawing calculations, the only major divergence from the 2D visualization if the complex models for the objects. While the simulator models the wheelchairs and doors by rectangles, as long as they occupy the same space in X and Y coordinates on the 3D viewer, there is no restriction on how the object itself is drawn. In fact, if one could actually see the chair, instead of a mere cube, it would make the visualization (and consequently the simulation itself) more credible. In Figure 6.9 is evident the difference, even though both objects have the same height width and COM (which is all that is used for simulation calculations and for 2D drawing).

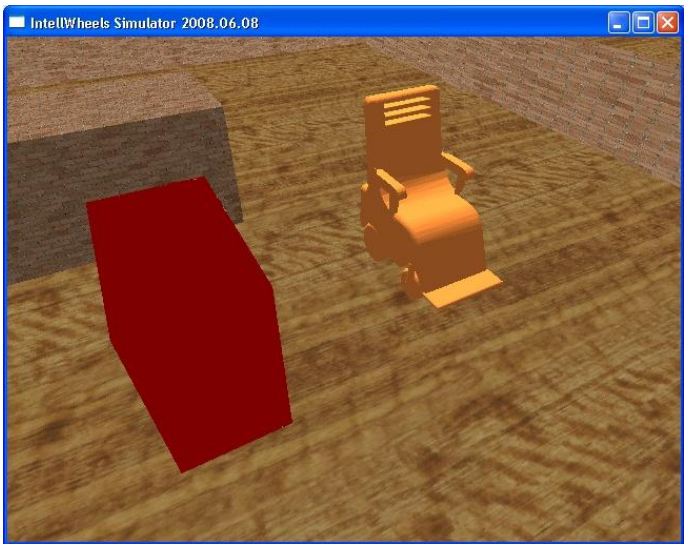


Figure 6.9 - 3D viewing: Shape vs Shapless Chair.

This kind of modeling is too complex to be made “by hand” through low level programming. Instead, a 3D drawing software was used to create the 3D model of the chair, and saved in a stereolithography file type (STL)[43][44]. STL is a flat file type which stores information of the 3D object’s vertexes (X, Y and Z coordinates) and faces’ normal orientation. To load this object onto the visualization, a STL parser was developed which read the STL file and stored the face and vertex information onto an OpenGL display list. The implemented algorithm, in Pascal language, is included in Annex B.

With the 3D objects loaded, the drawing of the simulation is done by resizing the objects to the information on stored on each robot (position, orientation, COM and height and width).

6.4.2 – 3D Options

Being the 3D version more complex than the 2D, there are a few additional parameters that can be configured, as shown in Figure 6.10.

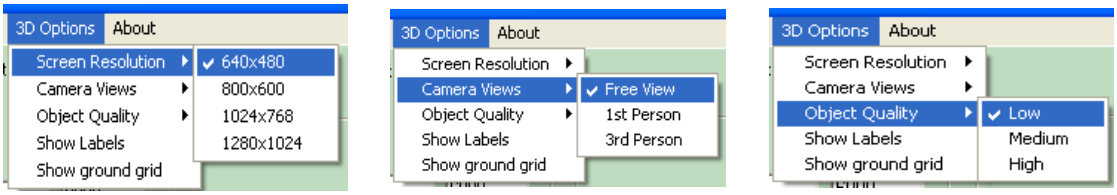


Figure 6.10 - 3D Options

These options include:

- Resolution variation (default is 640x480 but more powerful computers may allow better resolution);

- Camera view variation:
  - Free View – user can move the camera freely through the 3D world (Figure 6.9);
  - 1st Person – Camera is fixed on the chair, as if one was sitting on it (Figure 6.8);
  - 3rd Person – Camera is placed with fixed coordinates relatively to the chair and behind it.
- Object quality – The objects on STL files have 3 versions where with different number of vertexes. More vertexes correspond to increase in object quality but requires significantly more loading time and processor capacity during simulation run;
- Name labels on robotic agents (including IWs and intelligent doors);
- Ground grid drawing for easy position identification for the viewer.

Figure 6.11 shows a simulation with ground grid, high object quality (chairs), 640x480 resolution and free camera viewing.

Figure 6.11 - 3D viewing with free camera mode

6.5 – Summary

Due to the available Ciber-Rato original viewer's limitations, mostly its lack of flexibility, it was not liable to attempt its adaptation for intelligent wheelchair simulation. This showed

the need to develop a new viewer that was constructed using Delphi and OpenGL (for 3D building).

The developed viewer successfully creates credible graphical representation of the simulation and contributes to the entire simulation project with an evolved scenario. The Intellwheels Viewer implements drawing algorithms and 3D model loading functions that produce a full fluid, realistic and focused visual representation of intelligent wheelchair simulation. Moreover, the application itself is flexible enough in the sense that it can be easily expanded for increased performance or modified for different purposes, other than IW simulation. The application developed successfully creates credible graphical representation of the simulation and contributes to the entire simulation project with an evolved scenario.

Chapter 7

Simulator Tests

To validate the performance of the simulator and confirm its importance for intelligent wheelchair development, a series of tests was performed. These tests were based on driving analysis from real, virtual and augmented reality runs, which were compared against each other.

7.1 – Experiment Definitions

These tests were performed indoors, on the floor where LIACC is set, in FEUP. They consisted of various wheelchair driving runs. To achieve realistic simulation, LIACC's floor was modeled onto an XML file (Annex D includes the created XML file), so it could be read by the simulator, and later displayed in the Intellwheels Viewer (Figure 7.2). For realism comparison between the real plant and the modeled map, Figure 7.1 is the CAD drawing of the floor LIACC is in: 1st floor, block I.

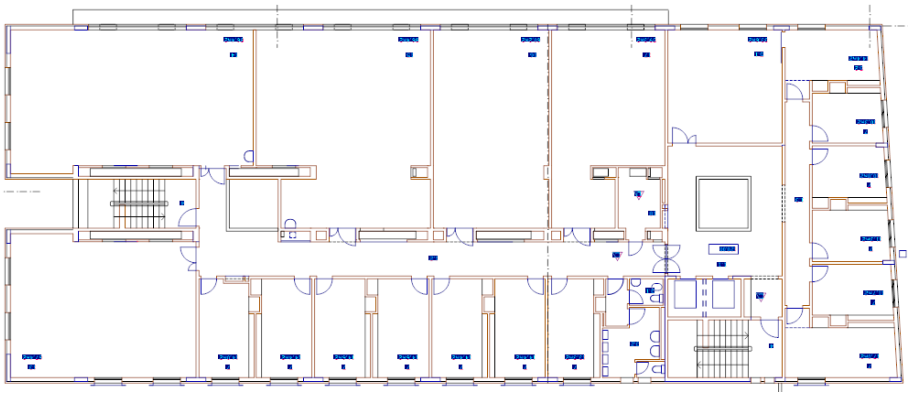


Figure 7.1 – CAD drawing of the plant for the LIACC floor.

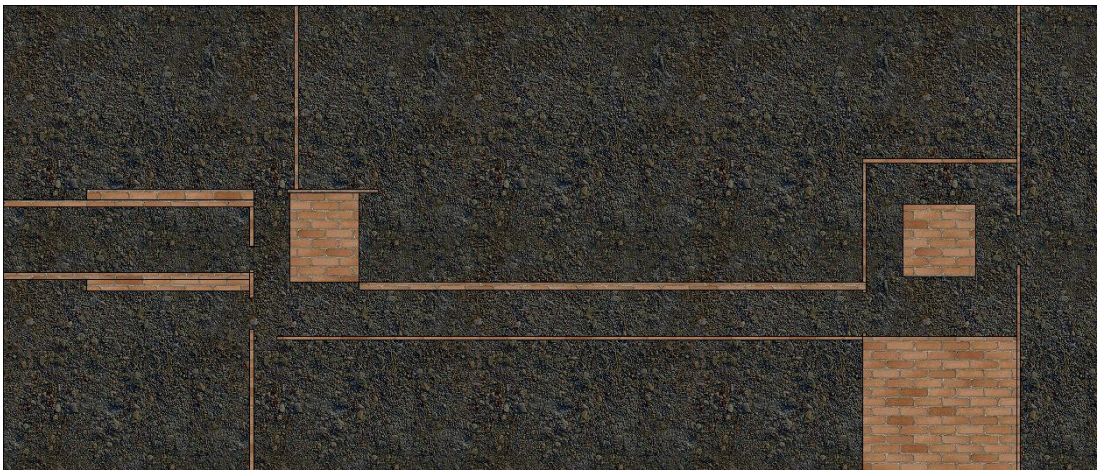


Figure 7.2 – Intellwheels modeled map of LIACC floor

The additional room divisions (shown in Figure 7.1) were not necessary for any of the required tests and, therefore, where not converted into XML map file. Nevertheless, the dimensions are faithful to the real floor as are the positions of the walls, on the modeled map.

7.2. – Dynamic Characteristic Tests

In order to correctly simulate the acceleration curve and the top speed, the first stage of the tests was to determine these characteristics of the real wheelchair. This initial test consisted on driving the wheelchair forward, inputting maximum values to the motors while starting from full stop (speed=0m/s). This test was performed with the Intellwheels Control Software and, through its log of odometry readings, the acceleration curve displayed in Figure 7.3 was obtained.

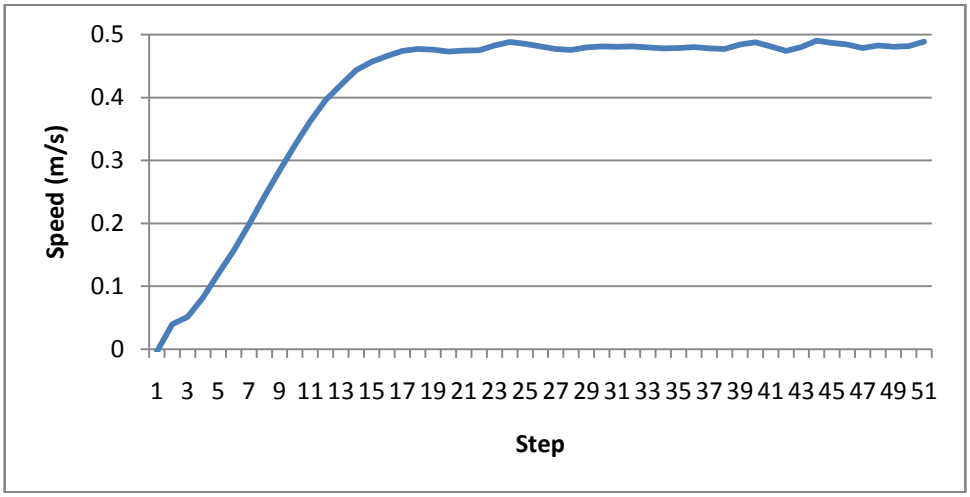


Figure 7.3 - Acceleration Curve of Real Wheelchair.

The evident variations are due to odometry errors, another problem of real tests and consequently an advantage of simulated environment.

A new line of the log file was registered every time step (100ms). The data analysis revealed a maximum speed of approximately 0.49 m/s and a rise time (from 10% to 90% of top speed) of 1.4 seconds.

Using this information and the new functionalities of the Intellwheels simulation, in terms of robot configuration parameters, it was possible to model a very similar characteristic for the virtual wheelchair. Using the same maximum speed detected for the real wheelchair (0.26m/s) and an "AccelerationCurve"=0.83 parameter (AccelerationCurve defined in Chapter 4), the Intellwheels Control Software connected only the simulator. The log of the position allowed the determination of the speed, which provided the results shown in Figure 7.4.

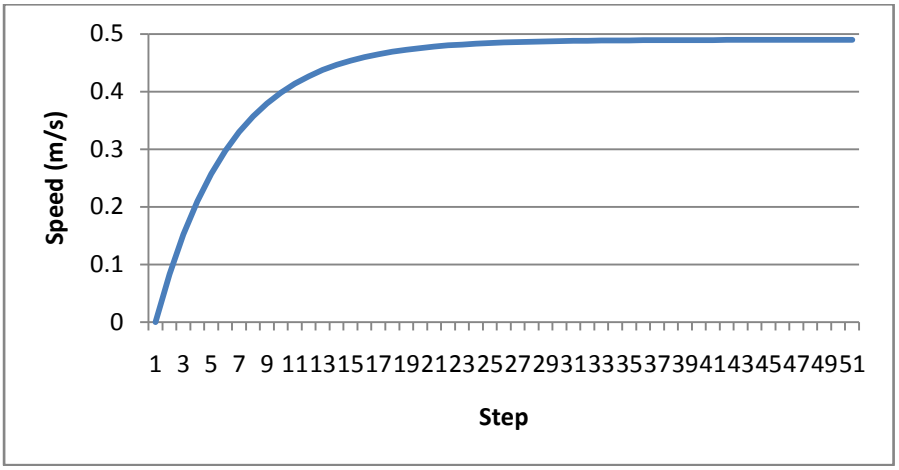


Figure 7.4 - Acceleration Curve of Virtual Wheelchair

By performing an analysis on the log file, similar to the previous, it was concluded that the maximum speed was the expected 0.49m/s and the rise time was of 1.6 seconds. Despite not being a perfect match to the real wheelchair's results, this difference of 0.2 seconds was low enough. Another noticeable difference is the curve itself. The simulated acceleration has a logarithmic shape which although slightly different than the real one, once again does not represent significant consequences in the outcome of the tests nor on the conclusions drawn from them.

This simulator was now able to model an electric wheelchair very similar to the real one and these simulation parameters were used throughout the remaining tests.

7.3 – Obstacle Avoidance Test

The aim of this test was to compare real, virtual and augmented reality IW performances. It consisted on driving through a course that imposed a kind of slalom. The check points are marked on the map below (Figure 7.5) and their coordinates in Table 3.1. The chair must go

from checkpoint #1 to checkpoint #2 and deviate from the obstacles put along the way. It is important to refer that the obstacle position is not known by the chair, only the proximity sensors (either real or virtual) will send that perception to the control software (in autonomous driving).

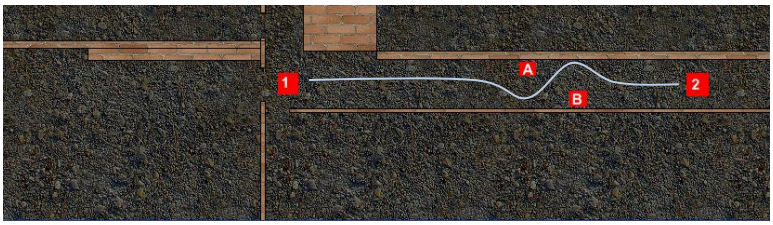


Figure 7.5 – Section of 2D LIACC map, with checkpoints and obstacles

Marks 1 and 2 are simply coordinates for the starting and finishing points of the tests, whereas marks A and B are obstacles and were placed in the same position and same dimension in both real and virtual environments.

Table 7.1 - Checkpoint Coordinates

Point	X (m)	Y (m)
1	10.0	6.1
2	20.0	6.1

Table 7.2 - Obstacle Coordinates

Obstacle	X_{min} (m)	Y_{min} (m)	X_{max} (m)	Y_{max} (m)
A	17	6.1	17.6	6.4
B	19.4	5.5	20	5.9

Table 7.2 defines the obstacles placed for the collision avoidance tests, detailed further in this chapter. Both objects are shaped as a rectangle (in the real and in the virtual environments). The minimum point and the maximum point coordinates define the position and size of the rectangle.

7.3.1 – Real Manual Driving

While sitting on the actual wheelchair, the user would command it using the PSX joystick to drive the wheelchair through the defined path. Trajectory deviation and final position and orientation are registered for comparison procedures. For this experiment, there is no connection with the simulator, only real environment is being tested.

7.3.2 – Virtual Manual Driving

Through the first person view option on the Intellwheels Viewer, a virtual wheelchair must be driven through the defined path. It is important to refer that the controller used for input orders is the same. There is no connection to the real wheelchair, and the user must drive only by seeing the display screen (Figure 7.7 displays the view of the operator).

7.3.3 – Real Automatic Driving

For this test, the control algorithms for automatic driving are tested as well as noise treatment for the sonar proximity sensors. The Intellwheels Control Software must be set up (through its implemented control algorithms) in such manner that it should go past the checkpoints defined and automatically deviate from real obstacles. No simulation connection is present during this experiment.

7.3.4 – Virtual Automatic Driving

Using the same automatic plan used in the previous test, the IW control application was disconnected for the real wheelchair and connected only to the simulation server. Through its virtual sensors, the wheelchair must navigate, once again, through the checkpoints and avoid virtual objects.

7.3.5 – Augmented Reality Automatic Driving

On this test, the wheelchair uses perception from the real world (sonars and IR sensors) as well as from the virtual world (simulator generated sensors). The controlling agent will be submitted to the same autonomous driving as before, although now it disposes of virtual sensors for additional collision avoidance.

7.3.6 – Obstacle Avoidance Results

During the test, a log of the control software's awareness of the chair's position (X and Y coordinates) was made. Annex C includes a log file example and the manual log sheet. The results, for each of the operation modes, are graphically represented in Figure 7.6.

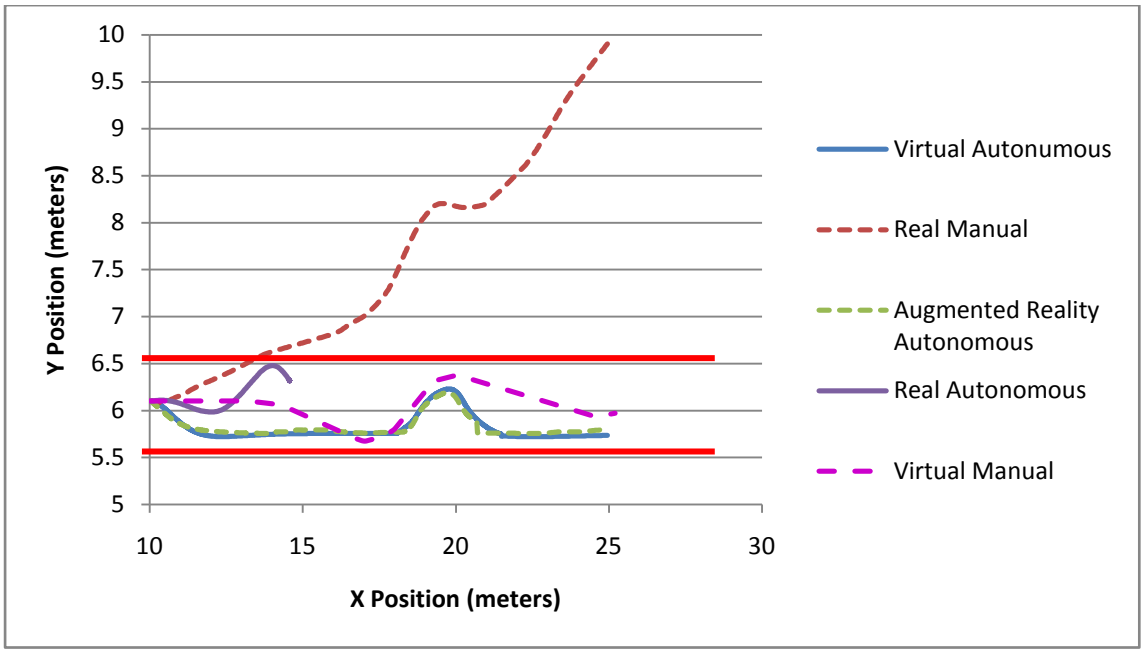


Figure 7.6 – Trajectory results from obstacle avoidance test

The red horizontal lines evident in Figure 7.6 represent the corridor walls delimitating the free space in which the chairs can move. Shown below, Table 3.1 indicates whether the run was successful and how many times the wheelchair touched a wall or obstacle.

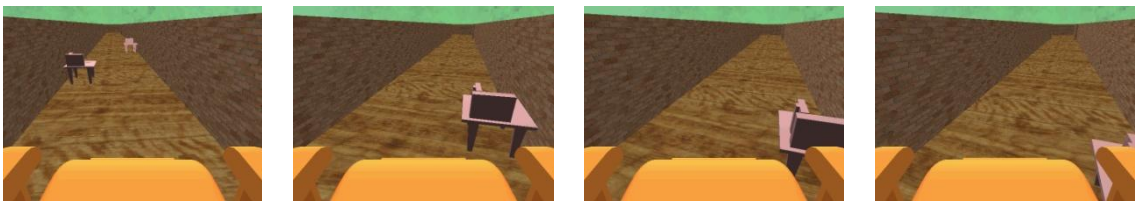
Table 7.3 - Obstacle Test Success Information

Operation mode	Reached Checkpoint #2 (Yes/No)	Collision Count
Real/Manual	Yes	0
Virtual/Manual	Yes	0
Real/Autonomous	No	1
Virtual/Automatic	Yes	0
Augmented/ Autonomous	Yes	0

The first test taken was the Real mode with manual driving. In terms of Control Agent testing, only the low level algorithms are tested (basic movements). As these have already been correctly implemented during the initial stage of prototype development [2][1], this run would provide valuable information of the odometer sensor. It is evident on the graphic that the readings were not reliable, for the X and Y coordinate calculations (determined from the odometer readings) are incorrect. In fact, the powered wheelchair was manually controlled through the corridor, without ever crashing. Had the chair be driven automatically relying solely on odometry it would undoubtedly collide with the wall.

Purely virtual test ran successfully without difficulties. The Intellwheels Viewer (Chapter 5) was set to 1st person viewing mode for added realism – illustrated in Figure 7.7 - and controlling the chair manually (with the PS2 controller) was easy. The log of the control agent

registered X and Y coordinates without error and, as it is evident on Figure 7.6, there were no collisions.



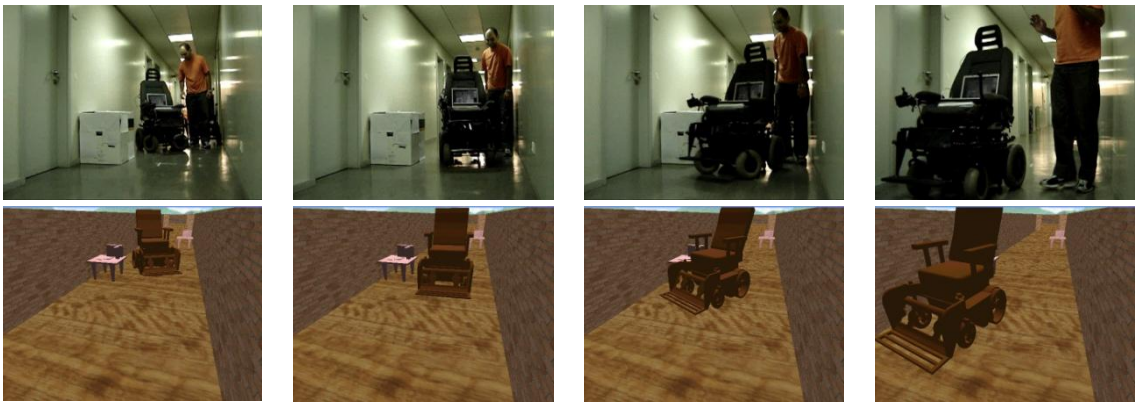
a b c d

Figure 7.7 - Virtual Reality, Manual Control Test

On the test with the real wheelchair (no simulation connection) on autonomous driving, some problems were encountered since, as it can be seen on Table 7.3, the chair collided with the wall. Despite the fact that the odometry errors were little (the chair's position calculation still placed it inside the corridor) this test was still unsuccessful. This outcome could have been cause either by imperfect sonar readings, or by algorithmic errors. This type uncertainty is one of the problems that simulation can solve by eliminating noise from sensors. A conclusion on the subject is done on this chapter's summary and on chapter 8 – Conclusions.

The virtual environment (real wheelchair not involved) autonomous driving test performed with a successful outcome. The chair successfully drove itself from point #1 to point #2 and deviated from the obstacles without any collisions. It is possible to conclude that, in a noise-free environment, the control algorithm uses correctly the sensor readings and avoids impacts.

The test with the real wheelchair in augmented reality operation mode on autonomous driving was a great success. The real movement of the chair was mimicked perfectly in the virtual world as shown in Figure 7.8.



a b c d

Figure 7.8 - Augmented reality test

Once again, similarly to the Real/Autonomous test, the odometry errors were very small. This led to the correct positioning of the wheelchair on the simulator which, on its turn, resulted in correct and virtual sensor readings. Consequently the wheelchair avoided real object, through its virtual representation. The wheelchair had, in fact, a position log very close to the virtual/autonomous test, as shown in Figure 7.6. Through this test, sensor merging algorithm implementation was validated.

7.4 – Automatic Door

In order to verify the correct implementation of the developed door agent (detailed in Chapter 5), a simple test was performed. The door agent was defined with height=0.1m, width=1.0m and COM=0.99. Two proximity sensors were defined at each side of the door, to detect approaching objects, as illustrated by Figure 7.9.

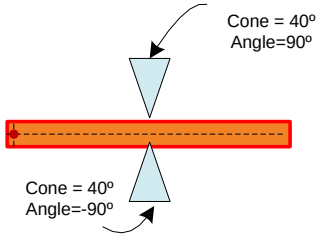
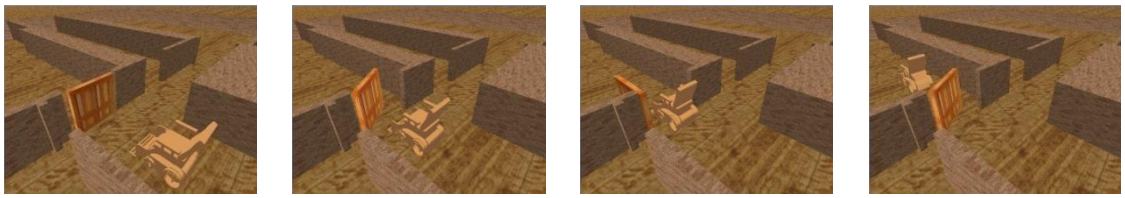


Figure 7.9 - Representation of the modeled door

Using the Wheelchair Agent, a virtual wheelchair was connected to the simulator and the test of door automatic opening was executed. Figure 7.10 is a series of print screens of the Intellwheels 3D viewer. The IW agent ordered the chair to move forward, regardless of what its own proximity sensors detect. On the other hand, the door agent was programmed to open if an object was detected and close only when the sensors stop detecting (as detailed in Chapter 5).



a b c d

Figure 7.10 - Automatic opening of the door

7.5 – Summary

This chapter presented the tests performed to compare simulated and real wheelchair movement. Tests were repeated through 5 operational and control methods:

- Real mode with manual control;
- Virtual mode with manual control;
- Real mode with automatic control;
- Virtual mode with automatic control;
- Augmented reality mode with automatic control.

The results showed that simulation-aided algorithm testing performed far better than purely real tests. The main reason for this was the errors in odometer-based positioning (which accumulates error) and sonar noise which sets the chair's controlling software into erratic decisions. It is also noticeable that the behavior of the wheelchair is almost equal in virtual and augmented reality modes.

Through these tests it was possible to determine that the simulation is an important mean to allow control algorithm test. It is possible to completely eliminate sensor reading errors, thus concentrating purely on the control section. Despite problems with real sensors, this simulation's value was proved with the success of the virtual environment tests. The control algorithm is correctly implemented, which is a conclusion that, without simulated testing, could not have been reached.

Chapter 8

Conclusions

This chapter will argue on the achievement of the initially proposed objectives for the project. It will discuss the results that were achieved from the tests performed, making a comparison between the system capabilities before the implementation of a simulation module and its present capabilities. It will also discuss the additional practical applications that the developed systems can have and finalizes with a view on what else can be expect on future work on this subject.

8.1 - Objective Achievement

The main goal for this research project was to build a simulation environment capable of testing the challenges that arise during the development of an intelligent wheelchair system. As a result, this work yielded not only a single application, but a complete testing system, with:

- World simulation, including map loading capabilities;
- Virtual robotic bodies, giving the objects movement capabilities limited by the rules of physics;
- Virtual sensors, for movement control and noise treatment algorithms testing;
- Augmented reality, for real and virtual environment interaction;
- Generic robotic agents, allowing various object load onto the simulation;
- Realistic computer generated graphics, conferring additional simulation credibility.

As a final implementation, the Intellwheels Control Software, previously developed in LIACC, was adapted now allowing operability with the simulator. Through this implementation, the developed simulator was proven a success. Not less important, the algorithms already implemented on it could be tested with focus on their purposes. It is now possible to ensure the functionality of a movement control algorithm (such as the obstacle

avoidance subsumption algorithm), eliminating sensor noise errors, since the virtual sensors can provide error-free readings.

The simulator is now in use at the present stage of the Intellwheels Project development, at LIACC, in different modules: “Main application” and “Intelligence” (see Chapter 2) for control and stratergy algorithm tests and even in “Multimodal interface” for integration and order validation.

8.2 – Main Results

This simulator was submitted to tests with wheelchairs under manual and autonomous driving. These tests were able to prove the concept that the simulator is a capable application when generating all the virtual information that an intelligent agent requires for autonomous navigation in an unknown environment.

The tests were also able to establish that, in a virtual environment, the IW autonomous driving, performs better than in the real environment, navigating with encoders for speed and position calculation and real sonar sensors for autonomous driving. From these results one is able to infer that additional development is required on noise treatment for these sensors, rather than holding the control algorithms responsible for failure in autonomous driving.

8.3 – Simulator’s Capabilities

The original intent of this simulator was to aid in the development of IWs and it is indeed being used in such manner.

The simulator’s capabilities stretch beyond IW wheelchair simulation, due to its origins (Ciber-Rato) and to the generalization that was applied to it. In fact, this simulator can now accept connections from any type of robotic agents, limited only to how the differential robot modeled is able to move. Car and pedestrian simulation can be performed, not only through their physical interaction (collisions) but emotional relations as well, since data communication between agents is available. Having a distributed architecture, Intellwheels Simulator expects the agents to be external applications that connect through UDP. Because of this attribute, it is able to involve in a unique simulation a vast number of intelligent agents, adding the possibility of testing algorithms results in a dynamic, complex environment.

A limit may be imposed by the visualization software (Intellwheels Viewer), as it does not yet have the full capability of dynamically loading any type of 3D object. Currently it possesses a small range of 3D models and they are allocated to the type XML tag of the object: only doors, tables and wheelchairs have dedicated 3D models, at the current version. Nevertheless, the current application’s source code is very flexible and basic shapes are loaded in case a different type is identified, thus providing the sense of size and volume in

the virtual world's space. In fact, since it constructs the modules depending on the physical characteristics provided by the simulator, the images offered will reflect the occupied space of the object, independently of its nature.

Due to the origins of Intellwheels Simulator, it accepts robotic agents from the original Ciber-Rato Simulato, with minor ajustments. The competition orientated mentality of the agent needs to be modified, as well as sensor registration, but these are not core changes to a controller software. Furthermore, the Ciber-Rato competition itself may evolve through the new implemented parameters. The Intellwheels' concepts of using intelligent robotic agents as dynamic scenario and the mixed reality feature (real and virtual environment interaction) could add value and interest to the University of Aveiro's competition.

8.4 – Future Work

Time constraints were severe during the development of this research project. Thus, a great number of features, that would add value to the simulation project, could not be implemented in time for the imposed writing and presentation deadlines.

For the simulator itself, the most relevant issues are related with the inclusion of additional sensors. Although not being presently in use by the Intellwheels Control Software, a digital camera is physically mounted on the chair. Ground marking localization algorithms, if implemented, using this camera, will add increased accuracy to the IW's awareness of its position on the world. This action will help in the solution of one of the problems detected during the tests performed on this dissertation.

Linked to this aspect are the encoder sensors. The real wheelchair uses encoders for movement calculation but, on the other hand, the simulated robot receives this information directly, through a virtual GPS. The possibility of encoder based navigation will increase realism and allow testing navigation algorithms themselves.

A final note on what could be done concerning the sensor simulation concerns the characteristic curve definition. Although the proximity sensors have configurable parameters, the output equation itself is not re-definable. Allowing this would approximate the virtual behavior even more to the real one.

Simulation agent development can still improve immensely, especially intelligent control for objects (other than wheelchairs). An unexplored, although available, feature is the communication module. Increased messaging capabilities between agents will allow:

- Doors that open by communication orders instead of proximity perception.
- Distributed planning: wheelchairs and other devices could jointly create plans to fulfill some given tasks in a cooperative manner.
- Strategy and tactics: wheelchairs could choose, between themselves, which one would fulfill an order given by an outside entity. A gain in service quality will be achieved with such an implementation.

- Complex calculations, such as path planning, could benefit from distributing computing between connected agents. Threads for the calculations could be spread among the other agents, allowing a uniform capacity usage.

Finally, in respect of the Intellwheels Viewer application, developed for simulation visualization, the potential is vast for future implementations. The user could use information, specific for each robotic agent, sent from the simulator to load (or even dynamically construct) a 3D model of the object. A possible implementation is to analyze the name of the agent and, through it, guess an object type. With a list of 3D object models, it would select the most appropriate and load it, through OpenGL, adjusting to the individual size definitions.

Associated with this topic is the notion of scenario-related object agent integration. A simulation, in order to be as interactive as possible, could allow direct insertion of objects, during its run. An intuitive mean to do so is by visualizing the world and space onto which it will be inserted. If the viewer had the capability of creating and controlling a table, cabinet or door agents it would greatly simplify the inserting task and converging with the notion of interaction importance on computer generated graphics [36].

A last functionality, that would increase the flexibility and the applicability of the viewer application, is to fully integrate the viewer with the Ciber-Rato competition. Having been designed for XML messaging, at the image of the original competition's software, the additional coding would bring interesting results. A selectable operation mode, choosing from IW Simulation or Ciber-Rato Simulation, would define how the robots are drawn: wheelchairs or circular robots. This work would certainly bring added value to the competition and probably interest more participants.

8.5 – Final Remarks

Having worked with the Ciber-Rato since the first day of this research project, the following final comments arise. As a matter a fact, the software proved to be very flexible as the adaptation modifications were implemented with success. Integration of the new algorithms and functions within the original code itself was good, due to its well structured characteristic.

This dissertation was made on a short period of time, however the developed applications have proved to be of value to LIACC's Intellwheels Project as conclusions concerning the previously developed control algorithms have already been taken. These new applications developed are a very solid base for further work and the possibilities for expansion of agents, viewer and the simulator itself are vast.

Bibliography

[1] M. R. Petry, "Desenvolvimento de um Protótipo e de Metodologias de Controlo de uma Cadeira de Rodas Inteligente," Dissertation for Masters Degree, Departamento de Engenharia Electrotecnica e de Computadores, Faculdade de Engenharia da Universidade do Porto, Porto, 2008.

[2] R. A. M. Braga, M. Petry, A. P. Moreira, and L. P. Reis, "INTELLWHEELS - A Development Platform for Intelligent Wheelchairs for Disabled People," in *5th International Conference on Informatics in Control, Automation and Robotics*, vol. I, Funchal, Madeira, Portugal, 2008, pp. 115-121.

[3] R. A. M. Braga, M. Petry, E. Oliveira, and L. P. Reis, "Multi-Level Control Of An Intelligent Wheelchair In a Hospital Environment Using A Cyber-Mouse Simulation System," in *5th International Conference on Informatics in Control, Automation and Robotics*, Funchal, Madeira, Portugal, 2008, pp. 179-182.

[4] P. M. Faria, R. A. M. Braga, E. Valgôde, and L. P. Reis, "Platform to Drive an Intelligent Wheelchair using Facial Expressions," in *Proceedings 9th International Conference on Enterprise Information Systems - Human-Computer Interaction (ICEIS 2007)*, Funchal, Madeira, 2007, pp. 164-169.

[5] P. M. Faria, R. A. M. Braga, E. Valgôde, and L. P. Reis, "Interface framework to drive an intelligent wheelchair using facial expressions," in *IEEE International Symposium on Industrial Electronics (ISIE 2007)*, 2007, pp. 1791-1796.

[6] R. A. M. Braga, M. R. Petry, A. P. Moreira, and L. P. Reis, "Platform for intelligent wheelchairs using multi-level control and probabilistic motion model," in *8th Portuguese Conference on Automatic Control, Controlo 2008*, Vila Real, Portugal, 2008.

[7] J. Banks, "Introduction to Simulation," in *Proceedings of the 2000 Winter Simulation Conference*, vol. I, Phoenix, Arizona, United States, 2000, pp. 7-13.

[8] N. Lau, A. Pereira, A. Melo, A. Neves, and J. Figueiredo, "Ciber-Rato: Um Ambiente de Simulação de Robots Móveis e Autónomos," *Revista do DETUA*, 2002.

[9] N. Lau, A. Pereira, A. Melo, J. Neves, and J. Figueiredo, "Ciber-Rato: Uma Competição Robótica num Ambiente Virtual," *Revista do DETUA*, vol. 3, no. 7, pp. 647-650, Sep. 2002.

[10] Institute of Electrical and Electronics Engineers. (2008, Apr.) IEEE - The world's leading professional association. [Online]. www.ieee.org

[11] Universidade de Aveiro. (2008, Jun.) Concurso Micro-Rato. [Online]. http://microrato.ua.pt

[12] L. P. Reis, "Ciber-Feup - Ensino de Robótica e Inteligência Artificial através da Participação em Competições Robóticas," *Electrónica e Comunicações*, vol. 7, no. 3, Sep. 2002.

[13] L. Almeida, P. Fonseca, L. J. Azevedo, and B. Cunha, "The Micro-Rato Contest: Mobile Robotics for All," in *CONTROLO 2000, The Portguese Control Conference*, Guimarães, Portugal, 2000.

[14] RTSS - Real-Time Systems Symposium. (2008, Jun.) RTSS - Real-Time Systems Symposium. [Online]. http://www.rtss.org/

[15] (2008, Jun.) CiberMouse@DCOSS08. [Online]. http://www.ieeta.pt/lse/ciberdcoss08/

[16] D. Barteneva, N. Lau, and L. P. Reis, "Implementation of Emotional Behaviors in Multi-Agent System using Fuzzy Logic and Temperamental Decision Mechanism," in *Proceedings of EUMAS 2006*, Lisbon, Portugal, 2006, pp. 5-15.

[17] D. Barteneva, N. Lau, and L. P. Reis, "Bylayer Agent-Based Model of Social Behavior: How Temperament Influences on Team Performance," in *21st European Conference on Modelling and Simulation – ECMS 2007*, Prague, Czech Republic, 2007, pp. 181-187.

[18] L. Lemos, F. Cruz, and L. P. Reis, "Sistema de Resgate e Salvamento Coordenado Utilizando o Simulador Ciber-Rato," in *CISTI 2007 - 2ª Conferência Ibérica de Sistemas e Tecnologias de Informação, Novas Perspectivas em Sistemas e Tecnologias de Informação*, Porto, Portugal, 2007.

[19] M. Chen, et al. (2002, Aug.) RoboCup Official Site. [Online]. http://www.robocup.org/

[20] N. Lau and L. P. Reis, "FC Portugal - High-level Coordination Methodologies in Soccer Robotics," in *Robotic Soccer*, Dec. 2007, p. 598.

[21] Gazebo / Player Project. (2008, Mar.) Gazebo / Player Project. [Online]. http://playerstage.sourceforge.net/index.php?src=gazebo

[22] B. Gerkey, R. Vaughan, and A. Howard, "The Player/Stage Project:Tools for Multi-Robot and Distributed Sensor Systems," in *International Conference on Advanced Robotics*, Coimbra, Portugal, 2003, pp. 317-323.

[23] S. Carpin, M. Lewis, J. Wang, S. Balakirsky, and C. Scrapper, "USARSim: a robot simulator for research and education," in *IEEE International Conference onRobotics and Automation*, 2007, pp. 1400-1405.

[24] Microsoft. (2008, Mar.) Microsoft Robotics. [Online]. http://msdn.microsoft.com/en-us/robotics/default.aspx

[25] AGEIA. (2008, Mar.) AGEIA PhysX. [Online]. http://www.ageia.com/physx/

[26] T. Röfer, "Strategies for Using a Simulation in the Development of the Bremen Autonomous Wheelchair," in *Proceedings of the 12th European Simulation Multiconference on Simulation - Past, Present and Future*, 1998, pp. 460-464.

[27] Universität Bremen. (2008, Apr.) SimRobot - 3D-Robotiksimulator. [Online]. http://www.informatik.uni-bremen.de/simrobot/

[28] G. Bourhis and Y. Agostini, "The Vahm Robotized Wheelchair: System Architecture and Human-Machine Interaction," *Journal of Intelligent and Robotic Systems*, vol. 22, no. 1, pp. 39-50, May 1998.

[29] H. Niniss and A. Nadif, "Simulation of the behaviour of a powered wheelchair using virtual reality," in *Proc. 3rd Intl Conf. Disability, Virtual Reality & Assoc. Tech*, Alghero, Italy, 2000.

[30] J. C. Lopes and C. Ribeiro. (2008, Feb.) João Correia Lopes | Homepage. [Online]. http://paginas.fe.up.pt/~jlopes/teach/2007-08/LAPD/lectures/01-XML-intro.pdf

[31] Departamento de Electrónica, Telecomunicações e Informática. (2008, Mar.) CiberMouse at DCOSS08. [Online]. http://www.ieeta.pt/lse/ciberdcoss08/docs/ciberDCOSS08_Rules.pdf

[32] B. Stroustrup, *The C++ programming language*, 2nd ed.. USA: Addison-Wesley Longman Publishing Co., Inc, 1991.

[33] M. K. Dalheimer, *Programming with Qt*, 2nd ed.. O'Reilly, 2002.

[34] R. Azuna, et al., "Recent Advances in Augmented Reality," *IEEE Computer Graphics and Applications*, vol. 21, no. 6, pp. 34-37, Nov. 2001.

[35] P. Milgram and F. Kishino, "A Taxonomy of Mixed Reality Visual Displays," in *IEICE Transactions on Information Systems*, vol. E77-D, 1994, pp. 1321-1329.

[36] Trolltech. Code Less. Create More. Deploy Everywhere. - Trolltech.

[37] Borland Software Company. Borland Software Company. [Online]. http://www.borland.com

[38] M. Cantù, *Mastering Delphi 7*, 1st ed.. Sybex, 2003.

[39] D. P. Reed. (1980, Aug.) Internet Engeneering Task Force. [Online]. http://tools.ietf.org/html/rfc768

[40] M. R. Rohrer, "Seeing is Believing: The Importance Of Visualization in Manufacturing Simulation," in *Winter Simulation Conference*, 2000, pp. 1211-1216.

[41] OpenGL. (2008, May) The Industry's Foundation for High Performance Graphics. [Online]. http://www.opengl.org/

[42] M. Woo, J. Neider, and T. Davis, *OpenGL Programming Guide, Third Edition: The Official Guide to Learning OpenGL, Version 1.2*. Addison-Wesley, 1999.

[43] E. Béchet, J. C. Cuilliere, and F. Trochu, "Generation of a finite element MESH from stereolithography (STL) files," *Computer-Aided Design*, vol. 34, no. 1, pp. 1-17, Jan. 2002.

[44] M. Burns, "The StL Format," in *Automated Fabrication*. Prentice Hall, 1989, ch. Section 6.5.

[45] A. Neves, J. Figueiredo, N. Lau, A. Pereira, and A. Melo, "O Visualizador do Ambiente de Simulação Ciber-Rato," *Revista do DETUA*, vol. 3, no. 7, pp. 651-654, Sep. 2002.

[46] P. Riley and G. Riley, "SPADES - a distributed agent simulation environment with software-in-the-loop execution," in *Winter Simulation Conference*, vol. 1, 2003, pp. 817-825.

[47] T. Bräunl, "The EyeSim Mobile Robot Simulator," Computer Science Department of The University of Auckland, 2000.

[48] B. Martins, E. Valgôde, P. Faria, and L. P. Reis, "Multimedia Interface with an Intelligent Wheelchair," in *Proc. of CompImage 2006 -Computational Modelling of Objects Represented in Images: Fundamentals Methods and Applications*, Coimbra, Portugal, 2006, pp. 267-274.

[49] M. Burns, "The STL File Format," in *Automated Fabrication - Improving Productivity in Manufacturing*. Prentice Hall, 1993, ch. Section 6.5.

[50] Faculdade de Engenharia da Universidade do Porto. (2008, Jun.) Faculdade de Engenharia da Universidade do Porto. [Online]. http://www.fe.up.pt/

ANNEXES

A – Ciber-Rato map XML file

File name: ThesisExampleMap.XML
File content:

```
<Lab Name="ThesisExampleMap" Width="28.000000" Height="14.000000">
	<Beacon X="25.000000" Y="7.000000" Height="30.000000" />
	<Target X="25.000000" Y="7.000000" Radius="1.500000" />
	<Wall Height="2.500000">
		<Corner X="11.500000" Y="11.500000" />
		<Corner X="11.500000" Y="2.500000" />
		<Corner X="13.500000" Y="2.500000" />
		<Corner X="13.500000" Y="11.500000" />
	</Wall>
	<Wall Height="1.000000">
		<Corner X="16.500000" Y="10.000000" />
		<Corner X="16.500000" Y="5.500000" />
		<Corner X="21.000000" Y="5.500000" />
		<Corner X="21.000000" Y="6.500000" />
		<Corner X="17.500000" Y="6.500000" />
		<Corner X="17.500000" Y="10.000000" />
	</Wall>
</Lab>
```

B – STL Parser for 3D Model Loading

During the OpenGL programming for the 3D viewer, a STL file type parser was developed, in order to load 3D models. The algorithm for the parser is detailed below, in the STL loading and vertex parsing code sections. Since the 3D viewer was built with Delphi 7, the code is in Pascal Language.

```
AssignFile(STLFile,filename);
Reset(STLFile);

  while not Eof(STLFile) do
  begin
    ReadLn(STLFile, straux);
    IF pos('facet normal',straux)>0 then begin
      FacetCount:=FacetCount+1;
      SetLength(FacetNormalMatrix,FacetCount);
      FacetNormalMatrix[FacetCount-1,0]:=ParseSTLFacet(straux,1,1);
      FacetNormalMatrix[FacetCount-1,1]:=ParseSTLFacet(straux,1,2);
      FacetNormalMatrix[FacetCount-1,2]:=ParseSTLFacet(straux,1,3);
    end;
IF pos('vertex',straux)>0 then begin
      STLmessage:=STLmessage+straux;
      vertexcount:=vertexcount+1;
      SetLength(vertexmatrix,vertexcount);
      vertexMatrix[vertexcount-1,0]:=ParseSTLVertex(straux,1,1);
      vertexMatrix[vertexcount-1,1]:=ParseSTLVertex(straux,1,2);
      vertexMatrix[vertexcount-1,2]:=ParseSTLVertex(straux,1,3);
end;
```

```
Function ParseSTLVertex(STLMsg : string; VertexNo: integer; ValueNo: integer): double;
var
  PValue : double;
  submsg:string;
  aux : integer;
  i, j: integer;
  VertexCT, ValueCount:integer;

begin
  VertexCT:=1;
  ValueCount:=1;
  submsg:=STLMsg;

  For VertexCT:=1 to VertexNo do begin
    IF Pos('vertex',submsg)<0 then exit;
    submsg:=RightStr(submsg,Length(submsg)-Pos('vertex',submsg)-6);

    IF VertexNo=VertexCT then
    begin
      IF POS('vertex',submsg)>0 then
        submsg:=LeftStr(submsg,Pos('vertex',submsg)-2);
      for ValueCount:=1 to ValueNo do begin
        IF ValueCount=ValueNo then
        begin
          IF ValueCount<3 then submsg:=LeftStr(submsg,Pos(' ',submsg)-1);
          PValue:=StrToFloat(submsg);
          ParseSTLVertex:=Pvalue;
          exit;
        end;
        submsg:=RightStr(submsg,Length(submsg)-Pos(' ',submsg));
      end;
    end;
  end;
end;
```

C - Test Registrations (Log File Example)

This file is an example of the logs produced by the Intellwheels Control Software. The character comma separates the collums. The sequence of columns represents the (orderly) the following details: Time Step; Ticks Left encoder; Ticks Right encoder; Sonar Front; Sonar Right; Sonar Back; Sonar Left; Robot Orientation; X coordinate (meters); Y coordinate (meters).

File name: "log.txt"

File content:

V=[

0.0600,0.00,0.00,174.00/145.92,72.96/78.07,210.50/49.75,94.54/106.20,0.00,10.00,6.1

0.0600,0.00,0.00,174.00/145.92,72.96/78.07,210.50/49.75,94.54/106.20,0.00,10.00,6.1

0.0600,0.00,0.00,174.00/145.92,72.96/78.07,210.50/49.75,94.54/106.20,0.00,10.00,6.1

0.0600,0.00,0.00,174.00/145.92,72.96/78.07,210.50/49.75,94.54/106.20,0.00,10.00,6.1

0.0600,0.00,0.00,174.00/145.92,72.96/78.07,210.50/49.75,94.54/106.20,0.00,10.00,6.1

0.0600,0.00,0.00,174.00/145.92,72.96/78.07,210.50/49.75,94.54/106.20,0.00,10.00,6.1

0.0600,0.00,0.00,174.00/145.92,72.96/78.07,210.50/49.75,94.54/106.20,0.00,10.00,6.1

]

D – LIACC XML MAP

To increase reality and also to allow more augmented reality rests, a XML map of the LIACC's floor on FEUP was created. This XML map is compatible with Intellwheels and Ciber-Rato simulators.

```
<Lab Name="IW_LIACC" Width="44.000000" Height="17.75">
    <Wall Height="2.500000">
        <Corner X="9.370000" Y="0.000000" />
        <Corner X="9.370000" Y="5.400000" />
        <Corner X="9.519000" Y="5.400000" />
        <Corner X="9.519000" Y="0.000000" />
    </Wall>
    <Wall Height="2.500000">
        <Corner X="9.519000" Y="5.130400" />
        <Corner X="9.618400" Y="5.130400" />
        <Corner X="9.618400" Y="4.992100" />
        <Corner X="9.519000" Y="4.992100" />
    </Wall>
    <Wall Height="2.500000">
        <Corner X="10.4159" Y="5.130400" />
        <Corner X="32.84" Y="5.130400" />
        <Corner X="32.84" Y="4.9903" />
        <Corner X="10.4159" Y="4.9903" />
    </Wall>
    <Wall Height="2.500000">
        <Corner X="9.370000" Y="6.59" />
        <Corner X="9.519000" Y="6.59" />
        <Corner X="9.519000" Y="7.66" />
        <Corner X="9.370000" Y="7.66" />
    </Wall>
    <Wall Height="2.500000">
        <Corner X="9.519000" Y="7.66" />
        <Corner X="9.370000" Y="7.66" />
        <Corner X="9.370000" Y="7.56" />
        <Corner X="9.519000" Y="7.56" />
    </Wall>
```

<Wall Height="2.500000">
<Corner X="0.0" Y="7.56" />
<Corner X="0.0" Y="7.29" />
<Corner X="9.37" Y="7.29" />
<Corner X="9.37" Y="7.56" />
</Wall>
<Wall Height="2.500000">
<Corner X="3.18" Y="7.29" />
<Corner X="3.18" Y="6.8725" />
<Corner X="9.37" Y="6.8725" />
<Corner X="9.37" Y="7.29" />
</Wall>
<Wall Height="2.500000">
<Corner X="9.37" Y="8.56" />
<Corner X="9.519" Y="8.56" />
<Corner X="9.519" Y="10.75" />
<Corner X="9.37" Y="10.75" />
</Wall>
<Wall Height="2.500000">
<Corner X="9.519000" Y="10.6" />
<Corner X="9.619000" Y="10.6" />
<Corner X="9.619000" Y="10.75" />
<Corner X="9.519000" Y="10.75" />
</Wall>
<Wall Height="2.5">
<Corner X="0.0" Y="10.33" />
<Corner X="0.0" Y="10.05" />
<Corner X="9.519" Y="10.05" />
<Corner X="9.519" Y="10.33" />
</Wall>
<Wall Height="2.5">
<Corner X="3.18" Y="10.33" />
<Corner X="9.519" Y="10.33" />
<Corner X="9.519" Y="10.75" />
<Corner X="3.18" Y="10.75" />
</Wall>
<Wall Height="2.500000">
<Corner X="11.08" Y="10.75" />

```
            <Corner X="11.08" Y="17.75" />
            <Corner X="11.23" Y="17.75" />
            <Corner X="11.23" Y="10.75" />
</Wall>
<Wall Height="2.5">
            <Corner X="10.81" Y="10.75" />
            <Corner X="14.26" Y="10.75" />
            <Corner X="14.26" Y="10.6" />
            <Corner X="10.81" Y="10.6" />
</Wall>
<Wall Height="2.5">
            <Corner X="10.91" Y="10.6" />
            <Corner X="13.52" Y="10.6" />
            <Corner X="13.52" Y="7.21" />
            <Corner X="10.91" Y="7.21" />
</Wall>
<Wall Height="2.5">
            <Corner X="13.52" Y="6.91" />
            <Corner X="32.84" Y="6.91" />
            <Corner X="32.84" Y="7.21" />
            <Corner X="13.52" Y="7.21" />
</Wall>
<Wall Height="2.5">
            <Corner X="32.84" Y="6.91" />
            <Corner X="32.84" Y="6.79" />
            <Corner X="32.69" Y="6.79" />
            <Corner X="32.69" Y="6.91" />
</Wall>
<Wall Height="2.500000">
            <Corner X="38.71" Y="0.0" />
            <Corner X="38.71" Y="7.85" />
            <Corner X="38.56" Y="7.85" />
            <Corner X="38.56" Y="0.0" />
</Wall>
<Wall Height="2.500000">
            <Corner X="32.69" Y="5.13" />
            <Corner X="32.69" Y="0.0" />
            <Corner X="38.56" Y="0.0" />
```

```
                <Corner X="38.56" Y="5.13" />
        </Wall>
        <Wall Height="2.500000">
                <Corner X="32.69" Y="5.13" />
                <Corner X="32.69" Y="5.25" />
                <Corner X="32.84" Y="5.25" />
                <Corner X="32.84" Y="5.13" />
        </Wall>
        <Wall Height="2.500000">
                <Corner X="37.0" Y="7.40" />
                <Corner X="37.0" Y="10.17" />
                <Corner X="34.23" Y="10.17" />
                <Corner X="34.23" Y="7.40" />
        </Wall>
        <Wall Height="2.5">
                <Corner X="38.56" Y="9.75" />
                <Corner X="38.71" Y="9.75" />
                <Corner X="38.71" Y="17.75" />
                <Corner X="38.56" Y="17.75" />
        </Wall>
        <Wall Height="2.5">
                <Corner X="38.56" Y="11.75" />
                <Corner X="32.69" Y="11.75" />
                <Corner X="32.69" Y="11.90" />
                <Corner X="38.56" Y="11.90" />
        </Wall>
                <Wall Height="2.5">
                <Corner X="32.84" Y="11.75" />
                <Corner X="32.84" Y="7.21" />
                <Corner X="32.69" Y="7.21" />
                <Corner X="32.69" Y="11.75" />
        </Wall>
</Lab>
```