Intelligent Wheelchair Simulation: Requirements and Architectural Issues

Marcelo Petry, Antonio Paulo Moreira
Robotics and Intelligent Systems - INESCPorto
Faculty of Engineering, University of Porto – FEUP/UP
Porto, Portugal
marcelo.petry@fe.up.pt, amoreira@fe.up.pt

Luis Paulo Reis, Rosaldo Rossetti
Artificial Intelligence and Computer Science Laboratory
Faculty of Engineering, University of Porto – FEUP/UP
Porto, Portugal
lpreis@fe.up.pt, rossetti@fe.up.pt

***Abstract*—Using robotic simulators, researchers are able to improve the algorithms of their robotics platforms before testing them in real environments. In fact, the safe environment provided by simulation is important, especially for robots that are constantly in contact with human beings, such as assistive robots and intelligent wheelchairs. Here, we propose to take advantage of an available general robotics simulator to model the IntellWheel's intelligent wheelchair prototype and its environment, enabling patient's drills and creating a test bed to refine and to experiment new methodologies of autonomous navigation, obstacle avoidance and human-machine interaction. As a result of the evaluation of four general simulators, the USARSim simulator has demonstrated to be more suitable to serve as basis for the IntellWheels simulation prototype. The development of a rough model of the intelligent wheelchair and of an appropriate test environment proved that with some modifications USARSim is able to provide a very realistic simulation platform for Intelligent Wheelchairs.**

Keywords- Intelligent Wheelchair, Simulation, Mixed Reality

I. INTRODUCTION

In the attempt of assisting people with mobility problems, many research projects of intelligent wheelchairs (IWs) have been created over the last years [1]. According to the general concept, an intelligent wheelchair can be defined as a robotic device built from an electric powered wheelchair, provided with a sensorial system, actuators and processing capabilities. At the same time, it is assumed that IW may present at least some skills such as autonomous navigation, autonomous planning, extended human-machine interaction, semi-autonomous behavior with obstacle avoidance, cooperative and collaborative behaviors. Thus, IW may be a good solution to assist severely handicapped people who are unable to operate classic electric wheelchairs by themselves in their daily activities. The aim of this paper is to take advantage of an available robotics simulator to model an intelligent wheelchair and its behavior, in order to create a test bed for new methodologies of autonomous navigation, obstacle avoidance and human-machine interaction, for instance.

Up to a recent past the use of simulations for simulating IWs (as any robot in general) was quite restricted due to the lack of general simulators. Usually, the existing simulators were developed to deal with some quite specific situations and environments. The development of a new tool for the simulation of IWs is time and resource consuming, and frequently is out of the project's scope. However, this reality started changing due to the release of general simulators and to the advantages of using robotics simulators.

Simulations have a great potential for low cost analysis, since it is able to give researchers access to cost-prohibitive sensors and robotic platforms. In addition, simulators provide the ability to compress time, and so, to evaluate the results of time-consuming experiments much faster. They are pedagogically proven technique for training [2], so they can be used to drill people in safe environments. They allow testing under repeatable and controllable conditions, simplifying debugging (e.g. the same scenario can be precisely generated to trigger a known error). Unlike real testing environments, which may not be accessible, or may only be accessible at certain times, simulated environments have unlimited availability [3]. For example, experiments that require special natural illumination (i.e. sun light) may be accessible for just some hours a day, and experiments requiring special weather conditions (like fog, rain, etc.) may be accessible just a few times a year. Simulations also provide researchers virtual access to different testing environments, making these virtual testing very cost effective. Actually, with the right modelling, the behavior of the robot can be tested in any environment (from a reconstruction of a laboratory up to urban environments, desserts, catastrophes, lakes, oceans, others planets, etc). Finally, the extensive use of simulators allows researchers to safely refine their algorithms before testing the robot behaviour in real environments. Although researchers are no more required to develop a simulator from scratch, the simulation results just reflect the reality when the simulation requirements are considered and when the appropriate models are introduced.

A. Requirements for the simulation of robotic systems

The requirements for simulating mobile robots may differ according to the purpose of the simulation. For testing motion control, a higher level of detail in multi-body may be important. On the other hand, for testing sensor data processing, a higher fidelity in the sensors measures is desirible. If the simulation aims to evaluate higher level of abstractions, like global localization, ground truth data should be provided. If machine vision is used by the robot, a good rendering is required.

Essentially, simulation requirements can be classified into physical fidelity and functional fidelity. The former concerns with how the simulation looks, sounds and feels. In other words, it is the ability of the simulator to render high resolution textures, shaders, lighting and reflection. The second concerns with the simulation of most of the forces acting on robots and on its actuators, including not only but gravity, dragging, accelerations and collisions [4].

B. *Characteristics of the intelligent wheelchair*

The IntellWheels Project is focused on creating a platform to develop intelligent wheelchairs, to help people with severe disabilities to live a more normal life. It is mainly concerned with the research and design of a multi-agent system to enable an easy integration of distinct sensors, actuators, user input devices, navigation methodologies, intelligent planning techniques and cooperation methodologies. This platform aims to facilitate the development and testing of new methodologies and techniques, and to integrate it with minor modifications into most of the commercially available electric wheelchairs. We believe that this platform can bring real capabilities to the wheelchairs, such as intelligent planning, autonomous and semi-autonomous navigation, thus making it possible to execute the desired displacement through a high-level command language.

As depicted in the Fig. 1, the hardware architecture of the IntellWheels prototype is composed by a set of encoders, ultrasounds and infrared sensors, input controls and by an ordinary powered wheelchair. The software platform, as illustrated in Fig. 2, relies on a multi-agent system (MAS) architecture composed basically by four agents, currently under development in Object Pascal [5], [6].

In order to achieve that, in Section 2, we will analyse the existing robotics simulators to model the IntellWheels intelligent wheelchair prototype. Then, throughout Section 3, we will discuss about the architecture and the main features of selected simulator. In Section 4 we describe preliminary results of the simulation, and finally, in Section 5, we will present the conclusions and suggested future improvements for this work

II. RELATED WORK

Currently, an extensive number of simulators are available for robotics research. In [4], Craighead *et al.* identify the weakness and strengths of 14 commercial and open source simulators.

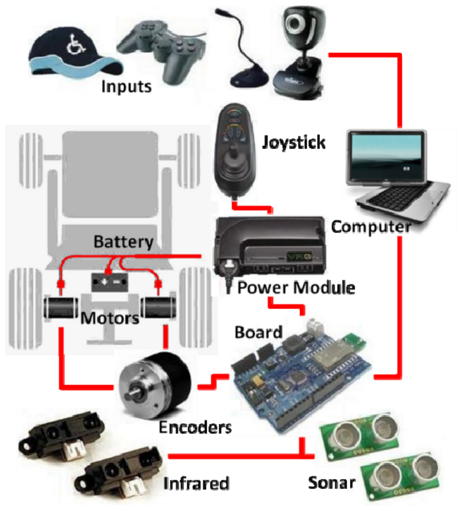


Figure 1. IntellWheels hardware architecture

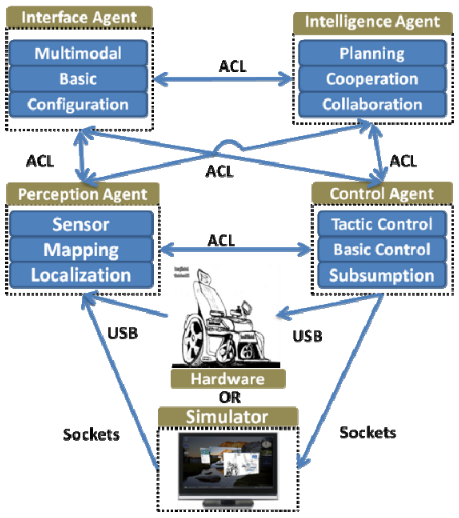


Figure 2. IntellWheels software architecture

However, in our case, specificities of the IntellWheels may be taken into account when choosing a tool to simulate intelligent wheelchairs. We also agree with Carpin et al. [7], when they claim that the simulation of robotic platforms should not consider a robot as an isolated entity, but as an entity which interact and is affected by the environment where it is situated. Therefore, we have restricted our analysis on four simulators that are able to offer these possibilities: Unified System for Automation and Robot Simulation (USARSim) [8], Microsoft Robotics Developer Studio (RDS) [9], Webots [10] and Gazebo [11].

A. *Criteria for evaluating robotics simulators*

In order select the available robotic simulator that better model the IntellWheels prototype, we propose the evaluation of the simulator using a set of seven criteria:

1) *Import 3D models* – we define this criteria as the ability of a simulator to import three-dimensional models of objects from typical Computer Aided Design (CAD) programs (such as Solidworks, Autocad, Pro-engineer, etc). We believe that this ability can facilitate the development of a more realistic model, thus improving the simulation. The evaluation of this criterion receives "yes" when the simulator supports importing objects and "no" when it does not.

2) *Programming language* – in this criterion we identify which programming languages are supported by the simulator to create the program that controls the robot. A wide support in the programming langue criteria is desired. In addition, we look specifically for a simulator that supports object pascal, once the IntellWheels platform is currently under development in that language. The evaluation of this criterion receives the list of the supported languages.

3) *External agent support* – concerns the ability to run the agent(s) that control the robot from outside of the simulator. This characteristic is desired because we want to be able to distribute the agents that control an IntellWheels prototype and the agents that provide additional services in more than one computer. This way, it is possible to increase the robustness of the system, since an agent can assume the tasks of other agents that for any reason are not answering. The evaluation of this criterion receives "yes" when the simulator supports external agents and "no" whenever it does not.

4) Multi-thread support – is the ability of the simulator to run more than one simulation task simultaneously. This ability is important to improve the simulation efficiency. The evaluation of this criterion receives "yes" when the simulator supports multi-thread and "no" when it does not support.

5) Physics Engine - concerns the identification of the libraries used for computing physics simulation. The main task of all physics engines is to solve the motion of the system given the forces acting on it. Therefore, they play a very important role in the simulation of dynamic systems because they are directly responsible for its functional fidelity. On the other hand, physics engines have a indirect responsibility also in the physical fidelity of the simulation. Particularly, the way that a simulation looks is closely dependent on the type of features the physic engine is able to simulate. For example, simulations with deformable objects demonstrate a greater realism over those which consider objects as rigid bodies, the simulation of fluids, like fog, may be important for machine vision and for video feedback, and so on. In subsection 2.2 the weaknesses and strengths of each library will be discussed in more detail. The evaluation of this criterion receives the name of the library used in each simulator.

6) License – corresponds to the monetary cost for the developer and for the end user. The evaluation of this criterion can receive the value "Open Source" for those simulators that are released with their source code, "free" for simulators that are available without any monetary compensation and without their source code, and "commercial" for those simulators that require monetary compensation.

7) Sensors – in this last criterion we identify which sensors are released with the simulators and if the simulator allows developers to create new sensors.

B. Evaluation results

Each simulator was evaluated through its User Manual or equivalent documentation, and results can be summarized in Table 1. With the exception of Gazebo, all simulators evaluated can import 3D models from typical CAD tools. However, when comparing the programming language, only USARSim and Gazebo can cope with a wider support. It is possible because these simulators rely on a client/server architecture with communication through UDP protocol, which also provides the support to external agents. Regarding multi-thread support, only USARSim and Microsoft Robotics Studio are able to benefit from the simultaneous task processing. Despite several libraries for physics computation available (PhysX, Bullet, JigLib, Newton, ODE, Tokamak, True Axis) [4], only PhysX and ODE were used by the four robotics simulators chosen for comparison.

ODE (Open Dynamics Engine) is an open-source library that is designed for simulations of rigid bodies and articulated bodies dynamics. For this reason, this library is not able to support the simulation of deformable objects, particles and fluids. ODE is platform independent with an easy to use C/C++ API. The kind of applications ODE was developed for also explains some of its characteristics, since ODE was developed to support speed, the physics accuracy tends to be compromised. On the other hand, PhysX is a proprietary solution widely used in Epic games. It provides support to the main platforms for games and graphics (such as PS3, XBOX, PC, etc.). Its main advantage consists in supporting not only rigid and articulated bodies, but also fluids (such as water, blood, smoke, gas, etc.) and particles (such as sparks, scattered glass fragments, dust, etc.). PhysX has a faster physics integration algorithm, and provides a more stable simulation when dealing with the collision of several objects [12]. In addition to the physics library, nVidia has also developed a special hardware device: the Physics Processing Unit (PPU).

With respect to the license, Gazebo and USARSim are open source simulators. At this point, it may be noticed that despite USARSim is open source, it relies on a proprietary engine and so has a small monetary cost corresponding to the Unreal Tournament 3. In its latest version Microsoft has combined the previous Express, Standard and Academic licenses into one license (RDS 2008 R3) free of charge, while the Webots has a commercial license with versions that costs from €250,00 Eur. (EDU version) up to €2600,00 Eur. (PRO version). Finally, the analyses of the sensors criteria revealed that all four simulators present the basic sensors used in the IntellWheels. The only severe limitation was observed in the RDS, which does not allow researchers to develop new sensors. For these reasons, USARSim was selected to simulate the IntellWheels prototypes. We have considered the lack of support for Object Pascal of the RDS and Webots, the limitation in the development of new sensors of the RDS, the cost of Webots, and the lack to support multi-task processing and to import 3D models as the main problems of the other simulators. USARSim, on the other hand, presents a superior physics engine, has the validation of several sensors and actuators and is probably the most used robotics simulator within the scientific community.

III. USARSim SIMULATOR

As elegantly described by Carpin *et al.* [7], "USARsim is a general-purpose multi-robot simulator that can be extended to model arbitrary application scenarios". It was designed to create physically accurate simulations of robots for research in fields like the human-robot interaction and multi-robot coordination. The simulator is built upon a commercial game engine thanks to the architecture of the Unreal Tournament 3, which separates the game logic and rules from simulation dynamics and environmental data. This way the game core code was reused and applied to a more comprehensive simulation, providing USARsim with high realistic visual rendering and high performance physics simulation. A further advantage relies in the fact that every improvement driven by the gaming industry translates directly into simulation advantages, which is particularly true for hyper realistic rendering and physical simulation [13].

The simulator is open source under the GPL licensing, and platform independent, running under operating systems like Windows, Linux an MacOS. USARSim is highly configurable and extensible, allowing users to develop new sensors, to model new robots and to create and re-create virtually any desired environment. As a consequence of its advantages, USARSim has become quite widespread within the scientific community, which has released a number of improvements. Simultaneously, researchers have published several papers with quantitative evaluations that demonstrate a very close similarity between the real world and USARSim system and sensors [8].

TABLE I. MAIN CHARACTERISTICS OF THE MOST USED 3D SIMULATORS IN THE ROBOTICS

Features		USARSim	RDS	Webots	Gazebo
Import 3D models		yes	yes	yes	no
Programming language		Any (UDP)	C#, VB, JScript, IronPython	C,C++,Java, Python, MATLAB	Any (TCP/UDP)
External agent support		yes	no	no	yes
Multi-thread support		yes	yes	no	no
Physics Engine		UT3 with PhysX	PhysX	ODE	ODE
License		Open Source*	Free	Commercial	Open Source
Sensors	**Camera**	yes	yes	yes	yes
	Touch sensors	no	yes	yes	yes
	Sonar Sensors	yes	yes	yes	yes
	Infra-red	yes	yes	yes	yes
	Sound sensor	yes	no	no	no
	GPS	yes	yes	yes	yes
	RFID	yes	no	no	yes
	Laser Range Finders	yes	yes	yes	yes
	Create new sensor	yes	no	yes	yes

The Unreal Engine is responsible for the sound, physics engine (collision detection and collision response), scripting, animation, threading, streaming, memory management and for rendering 3D graphics. On the initialization of the simulator, the Unreal Engine loads the set of geometrical models that describes all the objects in the simulation environment. For each object it is possible to specify its shape, colour and texture (among other properties). In addition, the Unreal Engine also loads a set of classes of compiled scripts that govern the behaviour of loaded models [14].

However, once the Unreal Engine is proprietary, it is not possible to establish a straight communication between the clients and the server. Instead, all the information exchange (in both directions with the engine) may occur through the network by means of a middleware application called Gamebots. The client side includes the Unreal client and the controller or the user side applications. Unreal clients are responsible for providing video feedback, rendering the simulated environment. The whole system architecture is depicted in Fig. 3.

A. Communication and control

All communication between the controllers and the Unreal Server is made through Gamebots. This middleware opens a TCP/IP socket for communication, allowing up to 16 connections (by default). The communication follows the Gamebots protocol, which is divided into Messages and Commands, following the structure:

data_type {segment1} {segment2}…

Where: 'data_type' specifies the type of the data and 'segment' specify the list of name/value pairs.

In Gamebots, Messages are a specific type of communication that contains information about the robot state (data_type = STA) or about the sensor data collected (data_type = SEN). On the other hand, commands contain instructions to control the world (data_type = INI), the robot (data_type = DRIVE), the camera (data_type = CAMERA) or robot's sensors (data_type = SENSOR). The main commands used for controlling a simulated intelligent wheelchair are briefly described above:

1) *Init* - This command adds a robot to the simulation, and is instantiated as:

INIT {ClassName robot_name} {Location x,y,z}

Where: {ClassName robot_name} 'robot_name' is the class name of the robot and {Location x,y,z} 'x,y,z' is the stat position of the robot in Unreal Unit.

2) *Drive* - This command is used to control the left and right side wheels, on a percentage of maxValue:

DRIVE {Left float} {Right float} {Light bool} {Flip bool}

Where: {Left float} 'float' is the spin speed for the left side wheel and {Right float} 'float' is spin speed for the right side wheel. Their range is –1.0 ~1.0 (move backward and move forward respectively). {Light bool} 'bool' is the Boolean value for turning on or turning off the headlight. {Flip bool} 'bool' is the Boolean value for flipping the robot.

3) *Camera* – This command controls the robot camera orientation and focus.

CAMERA {Rotation pitch,yaw,roll} {Zoom int}

Where: {Rotation pitch,yaw,roll} 'pitch,yaw,roll' is the relative value or absolute rotation angle of the camera. {Zoom int} 'int' is the zoom value. Positive values means zoom in, while negative values means zoom out.

B. Sensors

In USARSim, each virtual sensor is treated as an instance of a sensor class. Despite all the objects of a sensor class have the same sensing capability, it is possible to configure each sensor individually with different parameters (e.g. noise, distortion), allowing them to be simulated as close as possible of their counterparts in real systems. In addition, it is also possible to create a new type of sensors derived from a pre-existing sensor class.

Currently, just a few classes of sensors were already ported from the previous version of USARSim to the new UT3 version. Among those, we can find three classes of sensors that may be used in the simulation of the IntellWheels prototype: encoder, ground through and Range Finder sensors. Fig. 4 depicts the main classes of sensor present in USARSim.

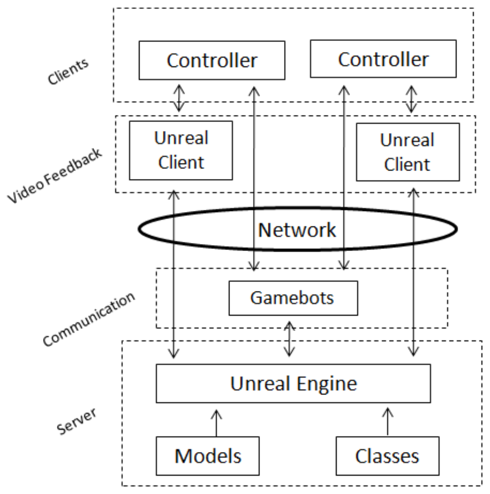


Figure 3. Architecture of the USARSim simulator (adapted from [8])

Classes of sensors drawn in blue were already ported from the previous version of USARSim and are distributed with the beta release [8]. Classes in green and purple were not ported from UT2004. However, in order to simulate the full set of sensors present in IntellWheels prototype, two subclasses of sensor (drawn in purple) were specially implemented in this project. A brief description of each class and its main characteristics will be presented bellow:

1) *Encoder* - This sensor measures a part's spin angle around the sensor's axis. The returned value is a tick count which is the real angle divided by the sensor's resolution. There are three parameters that can be set up in this sensor:

- Resolution: minimum spin angle the sensor can recognize [radians]
  Noise: it is the maximum amplitude of the noise [% of the truth measure]. The returned value containing the number of ticks (NTicks) with noise is then computed through the following equation:

$$NTicks = (1 + rand(-noise, noise)) * NTicks. \tag{1}$$

- Wheel: attach the encoder to its respective wheel. To perform such set up, one may use 'W' followed by the wheel number as the name of the sensor (e.g. EncoderW1).

The output of this sensor is a Message containing the type and name of the sensor and the number of ticks:

SEN {Type Encoder} {Name EncoderW1} {Tick NumTicks}

2) *Ground truth sensor* - This sensor returns accurate measures of the robots global position and orientation. Since it does not introduce any noise in its measures, the output data may be used to verify the performance of the robots localization algorithms, as well as for debugging and for testing high level algorithms (e.g. decision, planning, collaboration). The output of the Ground truth sensor is a Message with the type and the name of the sensor, the position and orientation of the robot at a given time stamp (Timestamp):

SEN {Time Timestamp} {Type GroundTruth} {Name GndTruth} {Location x,y,z} {Orientation roll,pitch,yaw }

3) *Infra-red sensor* - The IR sensor class implements the simulation of Infra-red sensors. This kind of sensor is used to detect the distance to the closest point (object) that lies on the line from the sensors position with the direction of the sensor. A full set up of this sensor includes the following parameters:

- HiddenSensor: Boolean variable used to indicate whether the sensor will be visually.
- MaxRange: maximum distance in which the sensor can detect objects [m].
- ScanInterval: time difference between two consecutive readings.
- Noise: it is the maximum amplitude of the noise [% of the truth measure]. The returned value containing the range distance (d) with noise is then computed through the following equation:

$$d = \big(1 + rand(-noise, noise)\big)d. \tag{2}$$

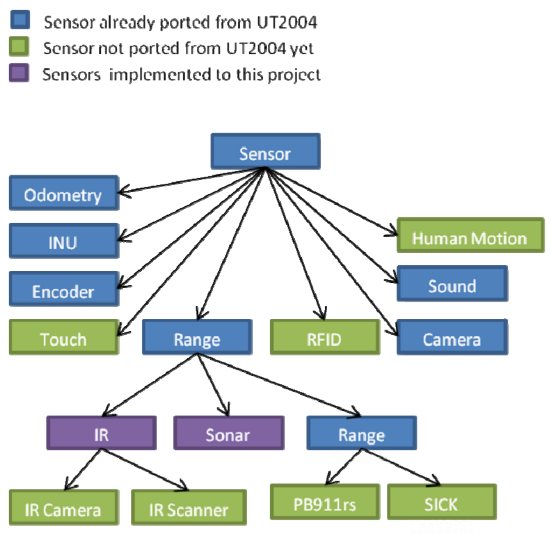


Figure 4. Classes of sensors in USARSim

- bWithTimeStamp: Boolean variable used to indicate whether the time stamp in the output message is included.

A typical IR Sensor output Message includes information about the time stamp (Timestamp), the type of the sensor (Range), its name (IR1) and the measured distance (DistanceValue):

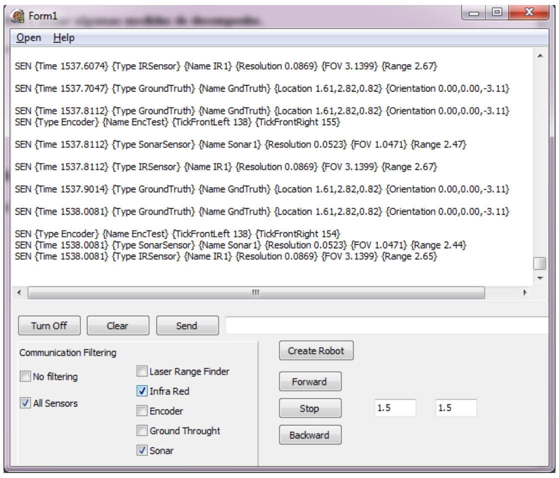
SEN {Time Timestamp} {Type Range} {Name IR1 Range DistanceValue}

4) *Sonar* - Sonar sensor class was designed to implement the simulation of ultrasound sensors, following the same concept used to implement it in previous versions of USARSim (as a series of IR sensors). Thus, data is obtained by rotating the sensor step by step (resolution) from the start to the end direction (field of view). Finally, the lowest from the data gathered by the sensor is then returned in the output Message. This presents the folowing parameters:

- HiddenSensor: Boolean variable used to indicate whether the sensor will be visible.
- MaxRange: maximum distance in which the sensor can detect objects [m].
- ScanInterval: time difference between two consecutive readings [s].
- Resolution: number of radians of each step [rad].
- Noise: it is the maximum amplitude of the noise [% of the truth value measured]. The returned value containing the range distance (d) with noise is then computed through (2).
- ScanFov: sensor's field of view [rad].
- bWithTimeStamp: Boolean variable used to whether include or not the time stamp in the output message.

The output message of sonar sensors contains information about the time stamp (Timestamp), the type of the sensor (Range), its name (Sonar1) and the measured distance (DistanceValue):

SEN {Time Timestamp} {Type Range} {Name Sonar1 Range DistanceValue}



```
SEN {Time 1537.6074} {Type IRSensor} {Name IR1} {Resolution 0.0869} {FOV 3.1399} {Range 2.67}

SEN {Time 1537.7047} {Type GroundTruth} {Name GndTruth} {Location 1.61,2.82,0.82} {Orientation 0.00,0.00,-3.11}

SEN {Time 1537.8112} {Type GroundTruth} {Name GndTruth} {Location 1.61,2.82,0.82} {Orientation 0.00,0.00,-3.11}
SEN {Type Encoder} {Name EncTest} {TickFrontLeft 138} {TickFrontRight 155}

SEN {Time 1537.8112} {Type SonarSensor} {Name Sonar1} {Resolution 0.0523} {FOV 1.0471} {Range 2.47}

SEN {Time 1537.8112} {Type IRSensor} {Name IR1} {Resolution 0.0869} {FOV 3.1399} {Range 2.67}

SEN {Time 1537.9014} {Type GroundTruth} {Name GndTruth} {Location 1.61,2.82,0.82} {Orientation 0.00,0.00,-3.11}

SEN {Time 1538.0081} {Type GroundTruth} {Name GndTruth} {Location 1.61,2.82,0.82} {Orientation 0.00,0.00,-3.11}

SEN {Type Encoder} {Name EncTest} {TickFrontLeft 138} {TickFrontRight 154}
SEN {Time 1538.0081} {Type SonarSensor} {Name Sonar1} {Resolution 0.0523} {FOV 1.0471} {Range 2.44}
SEN {Time 1538.0081} {Type IRSensor} {Name IR1} {Resolution 0.0869} {FOV 3.1399} {Range 2.65}
```

Figure 5. GUI of the robot controller

IV. PRELIMINAR EXPERIMENTS

In order to run initial experiments of communication between the control agent and the simulator we have used a robotic platform released with the USARSim simulator. P3AT is a skid-steer robot with 50cm x 49cm x 26cm of body size and four 21.5cm diameter wheels from ActivMedia Robotic. With proper configuration, the robot was provided with a camera and a set of eight sonars and eight infrared sensors assembled in a semi-circular ring around its body. In order receive the feedback from the simulator, a simple Graphic User Interface (GUI) was developed (Fig. 5). Through that interface, it is possible to create the robot in USARSim, parse the message containing the measures of each sensor and steer the vehicle in the simulated world.

V. CONCLUSIONS

In this paper, we have discussed the benefits of simulation in robotics and presented the requirements and characteristics for the simulation of intelligent wheelchairs – more specifically to the prototype developed in IntellWheels project. A set of seven criteria were proposed to assist in the evaluation of robotics simulators.

The results of the evaluation have demonstrated that both RDS and Webots lack on supporting Object Pascal. Also, RDS has a severe limitation in the development of new sensors, and Webots has high monetary cost associated. USARSim, on the other hand, has demonstrated a superior physics engine and a validation of several sensors and actuators. Thus, USARSim was selected for simulating the IntellWheels prototype. Once some important classes of sensors had not been ported from the previous version of the USARSim simulator, we have implemented one class for simulating the ultra-sound sensors and one class for simulating the infra-red sensors. Preliminary results were achieved using a P3AT platform configured with eight sonars and infrared sensors. A robot controller was developed in order to create and steer the robot in the simulated world and to parse the messages received from simulator.

As future work, we intend to design a realistic model of the wheelchair and of a cluttered environment. In addition, with the integration of the agents that control the IntellWheels prototype, we intend to create mixed reality environments. Drills of patients with real wheelchairs in virtual scenarios could be performed with increased realism, eliminating the risk of injuries and the stress of steering the wheelchair in the real environment. Furthermore, mixed reality experiments would make it possible to test the real IW in several scenarios (e.g. narrow corridors, crowded places, moving objects) in a safe (free of collisions with real objects, reducing the risk damaging the equipment) and inexpensive way (reduced time demanded to create scenarios, and minimum infra-structure cost).

ACKNOLEDGMENT

The first author would like to acknowledge FCT (grant SFRH/BD/60727/2009) for his PhD scholarship funding. This work was supported by the Fundação para a Ciência e Tecnologia through the project "INTELLWHEELS - Intelligent Wheelchair with Flexible Multimodal Interface" (grant ref. FCT/RIPD/ADA/109636/2009).

REFERENCES

[1] R. C. Simpson, "Smart wheelchairs: A literature review," *Journal of Rehabilitation Research and Development,* vol. 42, pp. 423-435, 2005. ISSN:0748-7711 .

[2] M. Friedmann, K. Petersen and O. von Stryk, "Simulation of multi-robot teams with flexible level of detail," *Simulation, modeling, and programming for autonomous robots*. vol. 5325: Springer, 2008, pp. 29-40. ISSN:0302-9743 .

[3] C. Pepper, S. Balakirsky and C. Scrapper, "Robot simulation physics validation," Workshop on Performance Metrics for Intelligent Systems, Washington DC,USA, pp.97-104, 2007.

[4] J. Craighead, R. Murphy, J. Burke, B. Goldiez, "A survey of commercial & open source unmanned vehicle simulators," in *IEEE Int. Conf. Robotics and Automation*, Rome, Italy, pp. 852-857, 10-14 April 2007. ISBN: 1-4244-0601-3.

[5] R. A. M. Braga, M. Petry, A. P. Moreira and L. P. Reis, "Concept and design of the intellwheels development platform for intelligent wheelchairs," in *Informatics in control, automation and robotics*. vol. 37, Springer-Verlag, 2009, pp. 191-203. ISSN:978-3-642-00270-0.

[6] R. A. M. Braga, M. Petry, A. P. Moreira and L. P. Reis, "Intellwheels - a development platform for intelligent wheelchairs for disabled people," in *Int. Conf. Informatics in Control, Automation and Robotics*, Funchal, PORTUGAL, pp. 115-121, May 11-15 2008. ISBN: 978-989-8111-31-9.

[7] S. Carpin, M. Lewis, J. Wang, S. Balakirsky, C. Scrapper, "Usarsim: A robot simulator for research and education," *IEEE Int. Conf. Robotics and Automation - ICRA*, Rome, Italy, pp. 1400-1405, April 2007. ISBN: 1-4244-0601-3

[8] B. Balaguer, S. Balakirsky, S. Carpin, M. Lewis and C. Scrapper, "Usarsim: A validated simulator for research in robotics and automation," Workshop on "Robot simulators: available software, scientific applications and future trends", at IEEE/RSJ IROS 2008

[9] Microsoft. *Microsoft robotics developer studio 2008,* http://www.Microsoft.Com/robotics/.(Consulted on January 2011)

[10] O. Michel, "Webotstm: Professional mobile robot simulation," *International Journal of Advanced Robotics Systems,* vol. 1, pp. 39-42, 2004

[11] Gazebo. *Gazebo user manual,*available at http://playerstage.Sourceforge.Net/index.Php?Src=doc. (Consulted on January 2011)

[12] M. Ma, M. McNeill, S. McDonough, J. Crosbie and L. Oliver, "Physics fidelity of virtual reality in motor rehabilitation," the Physics Fidelity of Virtual Reality in Motor Rehabilitation, Laval, France, pp. 35-41, April 2006

[13] M. Lewis, J. Wang and S. Hughes, "Usarsim: Simulation for the study of human-robot interaction," *Journal of Cognitive Engineering and Decision Making,* vol. 1, pp. 98-120, 2007

[14] J. Wang. (2005, october). *Usarsim v2.0.2 - a game based simulation of the nist reference arenas,* http://sourceforge.Net/projects/usarsim.(Consulted on February 2011)