A 3D Model of a Humanoid for USARSim Simulator

Nicola Greggio†, Giovanni Silvestri*, Stefano Antonello†, Emanuele Menegatti*†, Enrico Pagello*†
* Intelligent Autonomous Systems Laboratory
Department of Information Engineering
University of Padua, Italy
† IT+Robotics S.r.l. Vicenza, Italy.
A spin–off of the University of Padua

Abstract*— This paper focuses on the creation of a realistic simulation of a humanoid robot in a virtual environment using USARSim (Urban Search and Rescue Simulator). The robot we chose to model is a Robovie-M, by Vstone. This model will help researchers to implement and study different behaviors of the robot without its physical presence. The dynamics and the appearance of the robot and the other objects within the simulator are faithfully reproduced in the virtual environment. Moreover, the virtual robot in USARSim can be controlled with the same program controlling the real robot, a program tool we developed in our laboratory. We present a 3D model of the humanoid robot Robovie-M for the USARSim simulator and the computer program used as tool to control the 3D model of the robot within USARSim, and the real robot. We present the results of our experiments about the qualitative simulation of the robot's behaviors and the analysis of fps in case of single and multiple robot simulation.

I. Introduction

Simulating robotic behaviours has different important applications. Firstly, it allows researchers to develop and test programs in a virtual environment, without the physical presence of the robot. This may be extremely useful, for instance, in situation in which there are many people who are working on different tasks related to the same robot. In this case, each person can work independently without interfering with the others. Moreover, simulators are very useful in education: Students can develop different algorithms to study the robots reaction, without been physically in the laboratory or without the risk to damage the robot. Generally, a Humanoid Robot is a complex unit that is built with expensive electronic and mechanical parts.

Simulation is rapidily developing since a few years ago. 2D simulators are widely used in evaluating different kind of robots behaviours. They are easy to use and customize. They can be useful in many applications to test different kinds of robots, environments, and situations. On the other hand, they are limited to a 2D representation, which if can be enough for wheeled robot, it may not be sufficient for more complex robots. For example, robots with many degrees of freedom cannot properly simulated in a 2D world. Legged robots, and humanoid robots in particular, need a more complex simulator creating a faithfully 3D simulation.

3D simulators solve these problem. Again, there are advantages and disadvantages in this choice. This type of simulators can solve problems of depicting a humanoid robots behaviors in environment in which 2D simulators cannot, but they require to set many more parameters to give an effective result. In 3D simulation there are many problems: Modeling, animation, and virtual environments rendering are few examples. There are numerous different 3D simulators, and each of them presents advantages and disadvantages, so the choice must take into consideration all the aspects, in terms of fluidity of the simulation and reusability [1]. The ASURA RoboCup Software by the Asura RoboCup team [8] has been developed for the simulation of four legged robots. It allows to use strategies and sensors acquisition, but it lack in representing dynamic simulation, so giving a poor representation of the virtual environment. SimRobot simulator by Laue *et al.* [7], supports different models with a great flexibility of control, because there have been implemented different body models, sensors, and actuators. Dynamics is simulated with the open-source physical engine Open Dynamic Engine (ODE) [10]. UCHILSIM by Zagal and Ruiz-del-Solar [9] uses the ODE engine for the physical simulation. This project has been developed to became a standard framework for the AIBO robots' simulation. However, it seems to be no longer developed. Webots [11] is a commercial software for robotic simulation. It uses the ODE engine. It has an extensive library of actuators, sensors and robots. In addition, the mechanical features of the robots are well defined. However, it lacks in the quality of the 3D graphical representation of the virtual environment. USARSim (Urban Search And Rescue Simulator) [4] is a simulation environment useful for many kinds of applications that require a great fidelity in visual simulation and in physics properties simulation of an environment. In addition, we choose USARSim because there is a great community that is developing it, firstly for rescue tasks, and after that for RoboCup competitions [14], [15], [16], [17], [18]. USARSim has just used to simulate AIBO robots [1] and other robots. We used it to simulate a humanoid robot for the first time.

In section 2, we firstly present the USARSim simulator. Then, in section 3 we present the software architecture of our robot tool controlling. In section 4 we describe the 3D model of the Robovie-M humanoid robot. In section 5, we describe the experiments performed with the real robot and with the simulated robot to analyze the fidelity of our virtual model.

Proceedings of the Workshop on Humanoid Soccer Robots
of the 2006 IEEE-RAS International Conference on Humanoid Robots
Genoa, Italy, December 4, 2006
pp. 17-24
ISBN: 88-900426-2-1

II. The USARSim Simulator

USARSim is a high fidelity simulation of Urban Search and Rescue (USAR) robots and environments intended as a research tool for the study of human-robot interaction (HRI) and multirobot coordination.

It is a robot simulator based on the industrial game engine Unreal Engine 2 [5]. The current version of USARSim is based on the Unreal Engine game engine released by Epic Games with Unreal Tournament 2004. The engine to run the simulation can be inexpensively obtained by buying the game. It is a complete game development framework for todays PC. With its Unreal Editor, it is possible to use its tools to rapidly develop objects and environment. In addition, with its Unreal Script, which is an ad-hoc script language, it is possible to define the behavior of the objects. In fact, Unreal Engine 2 has been developed for the development of networked multi-player 3D games. In particular, USARSim simulates the robots intended for the Urban Search and Rescue (USAR) [4] tasks and the reference test arenas developed by the (National Institute of Standard And Technology (NIST).

Using USARSim problems like modeling, animation, and virtual environments rendering are automatically solved. High fidelity at low cost is made possible by building the simulation on top of a game engine. A commercial platform can provide superior visual rendering and physical modeling with respect to simulators developed by researchers. Then, it is useful to offload to it the most difficult aspects of simulation. This has the advantage that a full effort can be devoted to the robotics-specific tasks of modeling platforms, control systems, sensors, interface tools and environments. These tasks are in turn, accelerated by the advanced editing and development tools integrated with the game engine leading to a virtuous spiral in which a widening range of platforms can be modeled with greater fidelity in less time. The current release of the simulation consists of: various environmental models (levels), models of commercial and experimental robots, and sensor models.

Unreal spectators can observe the scenes by using egocentric (attached to the robot) or exocentric (third person) views of the simulation. Robot control programs can be written using one of the following tools: the GameBot interface, the MOAST System [6], the Player interface, or the Pyro middleware [4].

Our robot has been modeled using the program 3DStudio: The visualizing part, which appears on the screen, has been imported in USARSim as a static meshes (*.usx) file, while the physics property of the model has been described in Unreal Script language, using a file for each robot's part. In the following we call the simulated robot virtual robot (VR), and the real Robovie-M real robot (RR).

III. Robostage: The Robot Controlling Tool

We developed a computer program, called Robostage, to control the Robovie-M robot. It provides a graphical interface for the user.

We developed Robostage with two aims.

1. *To control different versions of control boards.* Currently, there are available three control boards for the robot:
   - One has a CPU Renesas H8/3684, which is functioning at 16MHz with three Microchip PIC16F877 for the servomotors' control, an accelerometer on two axis, and a serial port RS232C, by Vstone
   - Another is the CPU Renesas SH2/7054, which is functioning at 40MHz, a RAM of 262144 word of 16bit, an extern EEPROM of 64 Kbyte, an accelerometer on two axis, and a serial port RS232C, by Vstone
   - The last is the Korebot Board, which has a Intel XSCALE PXA-255 400MHz processor, a ram of 64MB, a USB Client port, and two serial ports RS232, by K-Team [12].

   Our program allows a programmer to use all the control boards. There is a code section for each control board, and it is possible to select which one to use using the graphical interface. The program serves to control the robot in real time and to program different complex movements starting from simpler ones. The graphical interface of the program is inspired by the original controlling software provided by Vstone with the Robovie-M robot.
2. *To control the simulated robot and the real robot using the same program and graphical interface.* As a result, it allows to verify the accuracy of the simulation. The virtual robot is controlled by our program tool as a device. A device is intended as an external tool that is directed by the program. In addition, the simulated robot and the real robot can be controlled at same time, one via TCP/IP and the other via serial port, respectively.

A. Functioning

The principal characteristics of Robostage are:

- Interfacing to the real robot by serial port
- Interfacing to the USARSim server and creation, in a preferred position, of the virtual robot within the running map
- Visualizing the information given by the RR and VR
- Sending commands to the joints, at the same time, too, to the RR and VR.

Robostage is composed by a single thread, for reading information and sending commands to the robot's control board, during each thread cycle. It communicates with the real robot using a serial port. Robostage sends the commands to the virtual robot communicating with the USARSim simulator by TCP/IP protocol. It sends information to the simulator.

The program provides three different modules to control the real and virtual robot:

a. USARSim module
b. Serial module
c. Rendering panel

The USARSim device controls the robot's virtual model. It sends commands to the joints of the model in USARSim via TCP/IP, permitting the model's movement. Dependly on

the available computer power, the simulator generates more or less frames, resulting in a more or less fluidity of the images on the screen.

The serial module controls the real robot, sending commands to its control board. These commands are the servomotors' position of the rotors. These positions are interpreted by the control board in order to produce the PWM signals needed to activate the servos subsequently. These positions are the state (the value of the 22 joints) at which the VR or the RR has to be ported. The format of data, of course, is different dependently on if the RR or the VR will receive them. The string that the serial module has to be sent to a robot has to be of the following type:

```
@11>7f>8e>1b>80>7a>80>1e>a4>10>e0>80>71>e4>7f>85>7f>de>5c>ef
>0a>8e>77<80<80<80<80<80<80<80<80<80<80
```

While at each cycle, the robot answers sending the following type of string:

```
>7f>8e>1b>80>7a>80>1e>a4>10>e0>80>71>e4>7f>85>7f>de>5c>ef>0a
>8e>77<80<80<80<80<80<80<80<80<80<80
```

These strings need to be read with sets of three characters: For example "$>7f$" means to activate a particular servomotor (a "$<$" means to not activate a servomotor) and to port it to the position "$7f$" (in exadecimal). The particular servomotor to which the set of three characters refers depends on the particular position of that set within the whole string. The last 10 servomotors have been inserted in the string type for likely future expansions. The Robovie-M's control board has other 10 controller for other 10 servomotors. These last motors are considered not activated ("$<$") and placed in a default position ("80"). The sending string differs from the receiving string only for the first three characters: "@xx" that indicates the speed (xx, in exadecimal) with which the robot has to assume the static position described by the sending string.

The rendering panel will be explained in Section *B*. This merely manages a integer vector, with dimension 22. It contains the values of the single joints placed in the same order discussed for the serial module.

Robostage has been developed with wxWidgets libraries (developed under GNU GPL licence [3]), to consent the program's porting under different OS without excessive changes on the code. So far, we ported the program on Windows and Mac OS X.

B. GUI

The main window is divided in four as shown in Fig. 1.

The first section is the control panel. It is situated at the top right of the main window. The control panel is composed by 22 bars. Each bar controls one servomotor. The set of all the servomotors' position values corresponds to a robot's static position, which is visualized in the rendering panel. We call each static position *stance*.

The second section is the frame panel. It is situated at the bottom left. This is a collection of frames. A complex movement, as a walking movement, is obtained by a sequence of frames. In the frame panel each frame is represented as a secondary window, like the ten windows in Fig. 1. Each

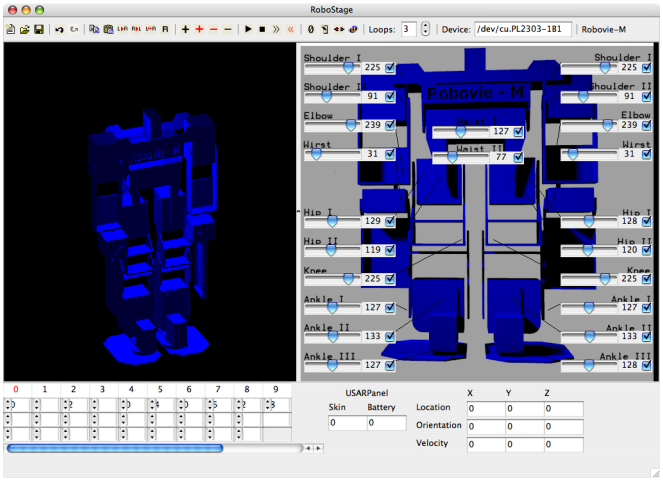


Fig. 1. Robostage graphical interface

frame window has its own frame number (the top section of each frame window), like the red one "0". There are other three sections of the frame window. The first one represents the speed with which the robot will arrive in that static position. The second one is a pointer to the frame that will be processed after the current one. The third one is the number of times that the frame will be repeated. There is a model, called Sim module, which is a tool of our program, created to interpolate the robot's frames in order to create a fluid movement. Since these movements cannot be fluidly executed by the robot if they are too much different, they need to be interpolated in order to be performed without jerks. In order to obtain this, the user should create more frames as possible to make a complex movement. Then, the program does another interpolation obtaining a even more fluid final robot's behavior. The Sim module provides this to the virtual robot's input and to the real robot's input.

The third section is the USAR Panel. It is situated at the bottom right of the main window. This displays a feedback information from USARSim. In fact, it provides information about the speed and position of the center of gravity (COG) of the robot in the virtual environment.

The last section is the rendering panel, situated at the top left of the screen. It is only a visualization of each static position (stance) of the robot. It works only if the real robot and the virtual robot are not connected (i.e. in use). It helps users to get a fast visualization of a stance qualitatively, without the need of USARSim or the real robot. When a complex behavior is loaded, the rendering panel shows the robot executing this movement. Using a black background the robot is visualized as a blue picture. This is controlled by Robostage, just as it were the real robot, i.e. managing the command panel, which is situated at the right of the rendering panel. The rendering panel has been implemented by using the openGL libraries (developed under GNU GPL licence [3]).

IV. The Robovie Model

Robovie-M is a humanoid robot produced by the Japanese Vstone [13]. We used the version two of this robot. It has 22 DOF (degrees of freedom), and consequently 22 servomotors, so distributed:

- 6 for each inferior art (legs)
- 4 for each superior art (arms)
- 2 for the trunk

The humanoid robot Robovie-M uses 22 Sanwa servomotors. The characteristics of the servomotors are in Table I.

Motore	Coppia	Velocitá	Dimensioni
Hyper ERG-VB	13 Kg x cm (6V)	60°/0.1s (6V)	39x20x37.4 mm
SPEC-APZ	4 Kg x cm (4.8V)	60°/0.2s (4.8V)	39x20x35.5 mm

TABLE I

Robovie-M: Technical characteristics of the servomotors.

Its dimensions are 290x240x65mm, with a complexive weight of 1.9 Kg. The robot is distributed without the camera, and its control board does not support a camera device. We used the control board with the CPU Renesas SH2/7054, previously described. This not only is more powerful than the previous one, but also allows a camera's supporting for image acquisitions. In addition, we modified the robot in our laboratory, in order to give it the possibility of mounting a camera. We developed a metallic head within which inserting the camera in a fixed position respect to the shoulders of the robot. In Fig. 2(b) it is possible to observe the head we build for our robot. The control board's power supply is given by a a set of batteries (five batteries of 1.2 V and 2300 mA) that gives 6 V as output, and that is able to provide a current of 6 A, needed to run the 22 servomotors of the robot.

The virtual model of the Robovie-M robot has been developed using the program 3DStudio. We took the size and weight measures of each single part of the Robovie-M, including the servomotors. Then, we drawn the virtual model of the robot with 3DStudio. The model needed to be exported as *.ASE files (one file for each part). This is because an *.ASE file is a generic mesh file, which can be recognized and imported by the Unreal Editor, as static meshes. This is the 3D model. There is the need of a script for each virtual robot's part. These scripts define the physical parameters of the virtual model of the robot, such as masses, frictions, and inertial tensors.

V. Experiments, platforms, and methodology used

Unreal Engine 2 and Unreal Script are configurable and finely tunable in order to create a realistic simulation (for instance one can set the parameters for friction and gravity). Without a precise simulation, an error in the initial phases of the robot's movements will propagate and will increase during the simulation, affecting the final results negatively. For this reason, we visually compared the real robot's behavior with the simulated one in order to define the best values of the *PenetrationScale* and the *ContactSoftness* parameters of the simulator. Marco Zaratti explained them in [14]. These are:

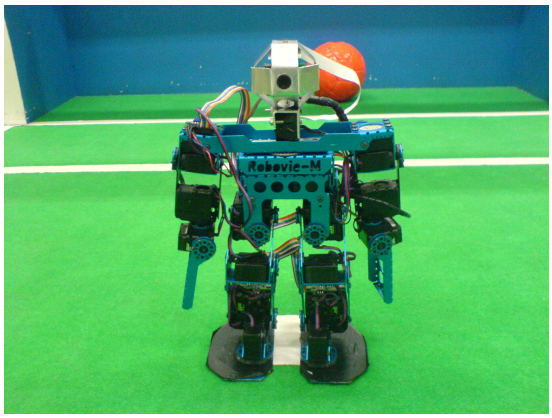
- *PenetrationScale* [default: 1.0]. Since interpenetration is caused by collision, this causes objects to appear to have impacted each other much harder (or softer, if the value is lower) than they really did. Penetration is multiplied by this amount. We set this value as: 5.
- ContactSoftness [default: 0.01] (karma units). This parameter is important in the cases of two, or more, objects collide. In these cases there are some initial interpenetration. The amount of penetration depends on how fast the two objects were going before they collided. Since the objects are considered as rigid object, after collision, Kea's projection feature will push the objects apart to reduce the penetration to zero. However, sometimes Kea will push the objects too far, and the contact will be broken. This can be a problem, for example, when objects are resting on the ground. When the contact is broken, the object will fall a short distance into the ground, the contact will be re-made and the object will be pushed out again. This process can result in resting objects that jitter or twitch from time to time. One solution is to set the softness option on the contact. This will cause two objects that are being forced together to naturally inter-penetrate slightly, preventing contact breaking. We set this value as: 0.001.

In addition to the visual confrontation, USARSim can return time and spatial coordinates of the simulated objects. Our first experiments are aimed at understanding the USARSim simulator's precision in simulating the robot behavior. Obtaining a precise simulation of a single step of a robot is the first thing to create a truthful simulation of more complex behaviors, such as kicking a ball. The Fig. 2(a) and the Fig. 2(b) show the virtual model of our Robovie-M in the simulator environment and the real one in our laboratory. We choose to analyze the real and virtual robots' walked distance and we rescaled the time spent by the virtual robot. We also choose to analyze and to compare the coordinates of the center of gravity (COG) of the robot in the frames of reference of the world.

Firstly, we made the RR performing a single left step followed by a right step. This is the basic behavior of a more complex walking movement. Secondly, we analyzed a more complex movement, which may be seen as the repetition of the basic one, i.e. a basic walk. We generated a motion sequence to make 8 steps along a straight line. This motion has been written on the RR and on the VR and we repeated this motion 10 times (both for the RR and the VR). Again, we compared the time and the coordinates of the robot in the real and in the simulated world. We placed the RR on the floor, in our laboratory. Then we started the RR from a fixed starting point and we made a video with a digital camera, which allows a time reference. The ground-truth position of the RR was given by a flexible ruler placed along its trajectory. We let the robot walking straight.

We took the time necessary to complete the 8-step walking task by the real robot. During the experiment, we evaluated the position of the robot every 25 seconds, in order to plot the behavior of the robot, in terms of speed, trajectory, and

(a) The VR



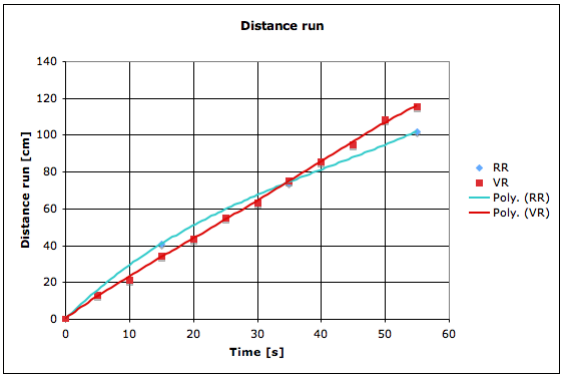
(b) The RR

Fig. 2. The VR drawn by the USARSim simulator and the RR Robovie-M

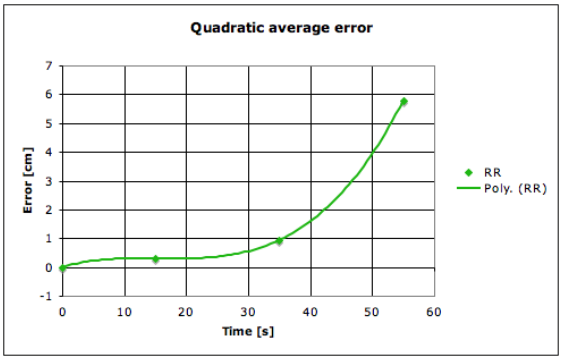
distance run as functions of the time. This was achieved thank to the flexible ruler placed along the robot walking direction. Then, we loaded the VR in the Unreal environment, and we did the same thing. USARSim gives the values of the speed, time, and position of the COG of the robot every thread cycle. USARSim uses a ”heavy” graphical engine. This latter consumes several processor’s and memory’s resources, causing slow down during the simulation. This surely affects the time spent by the VR in performing its 8-step walking task. We simulated the tests without running any other application, so giving the most resources to the computer as possible. We used an Apple PowerBook G4 with 768 MB of RAM memory, nVidia GeForce FX Go5200 64 MB of VR, and a G4 1.5 GHz PowerPc processor, with the OS X 10.4 operative system.

Then we compared the data of the RR with these of the VR. The plot in Fig. 3(a) shows the VR’s and the RR’s walked distance as function of time. Here, it is possible seeing how the RR and the VR maintain the same speed during the first 40 s. Then, the VR is faster. The plot in Fig. 3(b) shows the walk in terms of the quadratic error between the virtual and real walk. These graphs have been obtained by averaging the results of our 10 testing walks of the RR and our 10 testing walks of the VR.

In the plots in Fig. 4(a), 4(b), and 4(c) there are represented three test cases of the trajectories of the two robots in three cases, as function of the run distance. We made 10 tests, and we took three of these, because of their similarity to each other. We made the VR and the RR walking along a straight direction, which is represented in these graphs as the x-coordinate. The plot in Fig. 5 represents the error in this three cases, as function of the run distance, intended as the difference of the y-values of the trajectory coordinates. The y-values represent the values of the deviations of the robots’ trajectories from the given (imposed, represented by the x-axes) direction, along its orthogonal axes. In each case the error has been calculated as the difference between the VR values and the RR values, respectively. Subsequently, the plots in Fig. 6(a) and in Fig. 6(b) represent the average of the ten cases and the quadratic error, respectively.



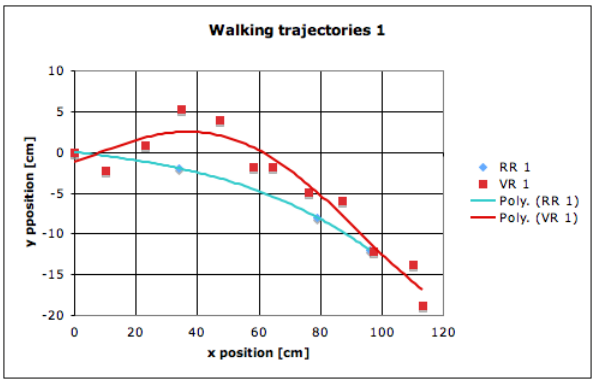
(a) Averaged



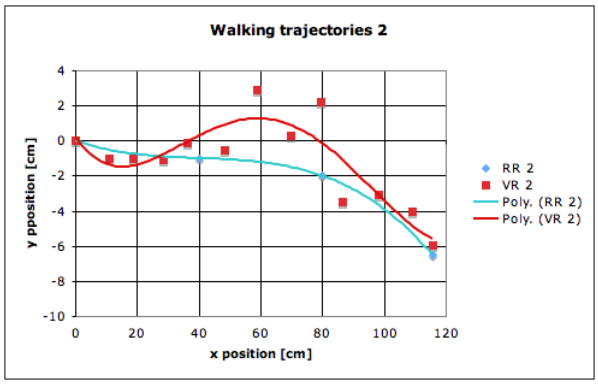
(b) Quadratic error

Fig. 3. VR’s vs RR’s walking progression

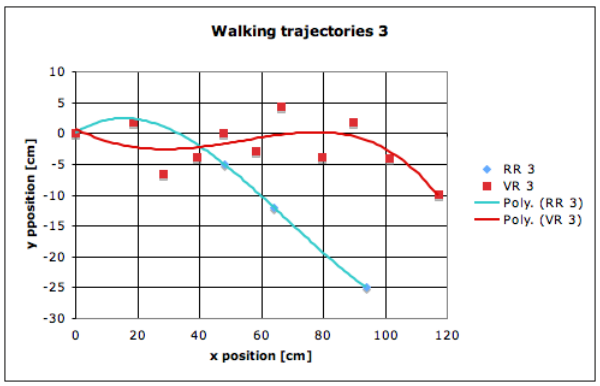
As an additional experiment, we tested the performance of the USARSim in terms of frames per second. Since using more than one robot is necessary specific in order to use the simulator to test virtual soccer games in which there are presented more than one robot. Using more than one robot, the computational complexity of the simulation increases, than the performances decreases, in terms of scenes fluidity and speed. We tested the performance of USARSim using frame speed (fps) frames per second. USARSim directly return this information to the user. In addition, it allows also to store it



(a) Test 1



(b) Test 2



(c) Test 3

Fig. 4. VR's vs RR's walking progression test

in a file log, with the other information (robot's speed, time, etc.). Two tests have been performed: One with one VR and the other with two VRs. In each test the robots were still fixed during the first 7 s, and they moved after that.

The plot in Fig. 7(a) shows the rendering of USARSim during, firstly a one VR simulation (blue line), and then a two-robot simulation (red line). The physics performance has been obtained redirecting the visualization to a map's corner. With this expedient, it has been possible not to have to visualize polygons and textures, which need to be rendered. The plot in Fig. 7(b) shows the physics performance of the simulator. As

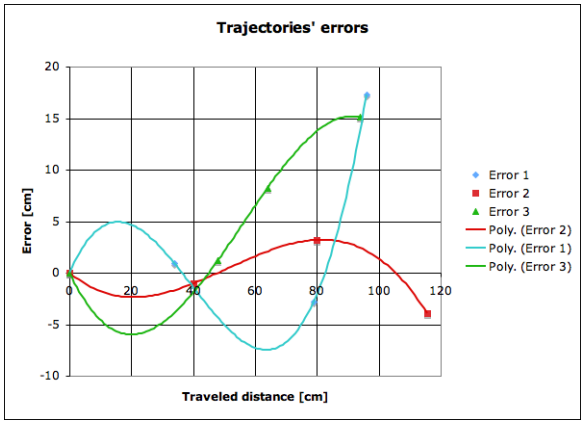
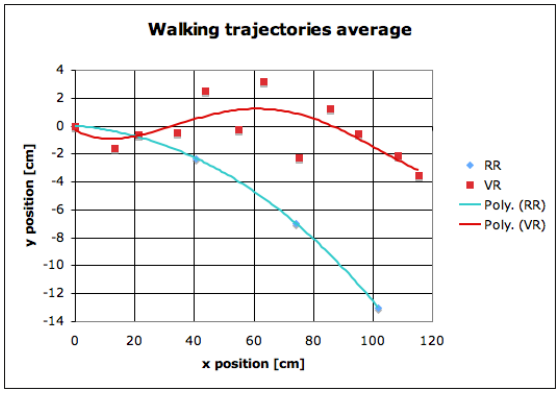
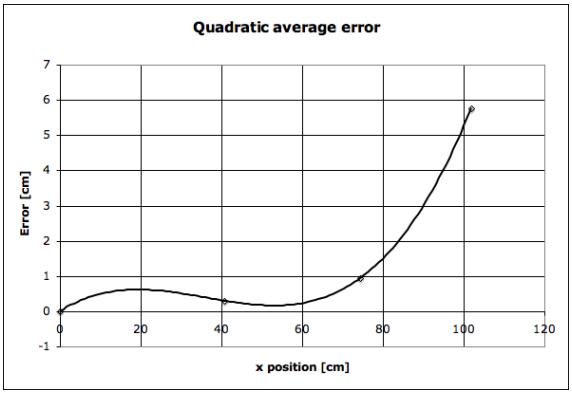


Fig. 5. Error of the trajectories in each of the three cases as function of the run distance



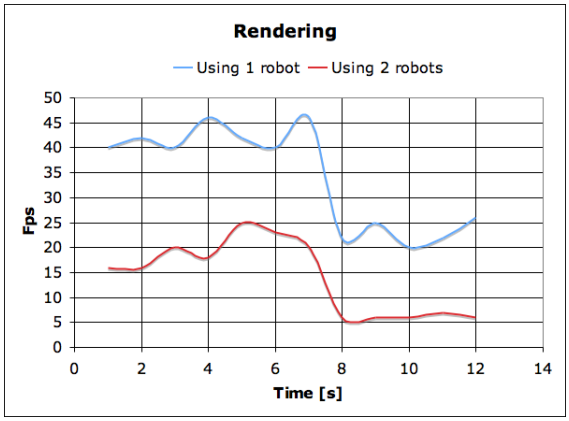
(a) Averaged



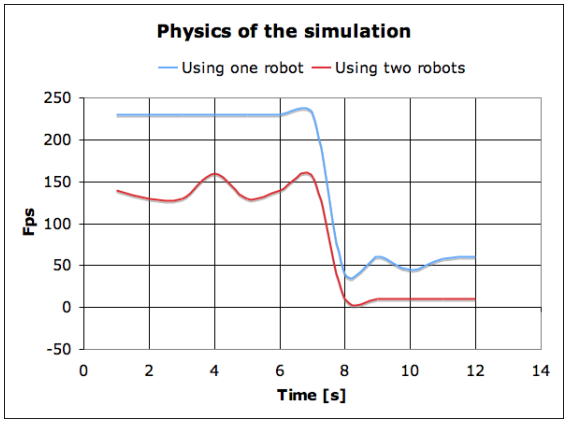
(b) Quadratic error

Fig. 6. VR's vs RR's walking progression

in the previous plot we simulated a one-VR and a two-VRs condition, with the robots in a fixed position (again before the 7 s) and during a moving condition (again after the 7 s).



(a) Rendering using two VRs



(b) Physics performance using two VRs

Fig. 7. USARSim performance

Finally, we simulated a penalty kick: The robot has to perform 8 steps and after that to kick a ball placed on the penalty kick spot in a regular RoboCup field. We placed the real robot in the laboratory and tested three different kick penalty situations. As in the previous experiment, we placed a flexible ruler along the RR walking direction, in order to obtain the exact values of position, in terms of x and y coordinates. Then, we tested the time required to perform this task. We made different movies with a digital camera, to document these tests. At the link *"http://www.dei.unipd.it/~emg/downloads/penaltyComparison.wmv"* there is the movie, representing the RR and the VR, during the performing of the same kick penalty.

VI. Discussion

We successfully tested the simulator by implementing the Robovie-M robot and controlling it at a full frame rate (30 fps). The simulator is currently used for the development of our RoboCup team at IAS-Lab of the University of Padua. The simulated robot seems to be effectively adherent to the real robot. The experimental results show that their behavior are quite similar, intended as precision of movements in the space. Sometimes differences may occur, which are caused not by simulation errors, but by a casual error during the walking action. Since walking results not only on the programmed movement but also by the starting conditions of the robot, even small differences in the robot's starting positions may affect the final results. We think that these affect the results most.

Another fact is that a robot may slip on the floor while it is walking, therefore changing its trajectory. One very simple explanation may be the following one: The laboratory's floor and the virtual floor may do not have the same friction. Under this consideration, the explanation for this discrepancy is that the RR slips more on the floor, changing its direction more than the VR from the right direction, and then loosing more time during its walking. To solve this problem it is possible to modify the friction coefficient in USARSim. The problem is that to do this and simulate a real condition, one has to know the exact real value of the current floor's friction coefficient. Without this one has to find it setting different values in USARSim empirically, in order to find the one that best approximate the real situation. However, it is not so useful, considering that doing so the simulator needs to be calibrated every time one uses the robot with different floors. In addition, this problem does not affect the time spent by the VR to cover the simulated distance considerably. We are interested in a qualitative comparison of the RR's and VR's behaviors.

Another explanation considers the source of the information given by USARSim. This is obtained by the program considering the COG of the robot model. For the RR it was not possible to evaluate the position and speed of the its COG, then we evaluated the position of the RR's feet. This is a more precise information considering the robot's trajectory. In fact, the VR's trajectory plots show great oscillations in their data. This is caused by the fact that the barycenter oscillates during a walking ever, while the feet are stable for about half the step's time. With these bias, it is clear how much the VR's positions are affected by these systematic errors.

Finally, the last consideration is about the fact that each trajectory of the VR is different from the previous one. This is because the physical engine of UNreal Tournament Karma [2], has an internal engine that solves a set of linear equations for each simulation's step. Karma is a library of the USARSim's MathEngine. Karma uses the Lagrange's multiplying method to model the rigid bodies. Within this model, the effect of the joints is modeled by forces that act to maintain the joint. To calculate these forces it is solved a set of linear equations using the linear prediction method (LPC). Kea is the solver of these equations. It calculates, at the end of each time-step, as the forces are applied to satisfy the joints. Kea, solves the differential equations with a certain degree of approximation that may be different every time. So the result may be different every time.

VII. Conclusion

In this paper we discussed and analyzed the usefulness and validity of a particular simulator, called USARSim. We described a program to generate movements for the humanoid robot Robovie-M. We proposed a virtual model in order to use it within the simulator. We analyzed the behavior and the performance of a Robovie-M robot's simulated activity based on a simple basic task like walking along a straight line. Finally, we analyzed the USARSim's performance in terms of graphical fluidity, measured in fps, during the experimental case of simulating first one robot and then two robots at the same time.

The next step of this research will be visualizing a virtual scene by using a property of USARSim, simulating the camera on the head of our robots. Visualizing a scene of the virtual environment in a egocentric view, it will be possible to process it as it is currently done with the images obtained with the camera of the RR. So, it should be possible to program the VR to act in the virtual environment as the RR does in the real environment. The further step, therefore, will be to write a computer program that accepts the data previously analyzed from the visual scene given by USARSim, and will elaborate them in order to send commands to the VR by Robostage. This computer program will reproduce the functioning of the control board of the RR via software. Doing so, the VR will be completely independent, as a simulation software should require.

References

[1] Marco Zaratti, Marco Fratarcangeli and Luca Iocchi A 3D Simulator of Multiple Legged Robots based on USARSim, 10th RoboCup International Symposium (CD-ROM Proceedings), Bremen, Germany June 2006

[2] MathEngine. MathEngine Karma User Guide, March 2002. Accompanying Karma Version 1.2.

[3] GNU Org. Free Software foundation. URL: http://www.gnu.org

[4] USARSim Urban Search and Rescue Simulation. URL: http://usarsim.sourceforge.net/

[5] Unreal Engine 2. URL: http://www.unrealtechnology.com

[6] MOAST System URL: http://moast.sourceforge.net/

[7] T. Laue, K. Spiess, and T. Röefer, SimRobot - A General Physical Robot Simulator and Its Application in RoboCup, in RoboCup 2005: Robot Soccer World Cup IX, ser. Lecture Notes in Artificial Intelligence. Springer.

[8] Asura: The United Team of Kyushu. URL: http://www.asura.ac

[9] Karma physiscal engine. http://wiki.beyondunreal.com/wiki/Karma.

[10] Open Dynamics Engine. URL: http://www.ode.org

[11] Cyberbotics. URL: http://www.cyberbotics.com/products/webots/

[12] K-Team. URL: http://www.k-team.com

[13] Vstone Corporation. URL: http://Vstone.co.jp/

[14] URL: http://digilander.libero.it/windflow/index.htm

[15] Jijun Wang. *"USARSim. A Game-based Simulation of the NIST Reference Arenas"*. University of Pittsburg, May 2005. URL: http://sourceforge.net/projects/usarsim/

[16] S. Carpin, M. Lewis, J. Wang, S. Balakirsky, C. Scrapper. "Bridging the gap between simulation and reality in urban search and rescue". Robocup 2006: Robot Soccer World Cup X, Springer, 10th RoboCup International Symposium (CD-ROM Proceedings), Bremen, Germany June 2006

[17] S. Balakirsky, C. Scrapper, S. Carpin, M. Lewis. "USARSim: providing a framework for multi-robot performance evaluation". Proceedings of PerMIS 2006

[18] S. Carpin, T. Stoyanov, Y. Nevatia, M. Lewis, J. Wang. "Quantitative Assessments of USARSim Accuracy". Proceedings of PerMIS 2006