Extending Virtual Robots towards RoboCup Soccer Simulation and @Home

Sander van Noort and Arnoud Visser

Universiteit van Amsterdam, Science Park 904, Amsterdam, The Netherlands

Abstract. The RoboCup is an initiative to promote the development of robotics in a social relevant way. The competition consists of several leagues and it would be beneficial if developments in one league could be reused in other leagues. This paper describes the development of a simulation model for a humanoid robot inside USARSim, which could be the basis of synergy between the Rescue Simulation, Soccer Simulation and @Home League. USARSim is an existing 3D simulator based on the Unreal Engine, which provides facilities for good quality rendering, physics simulation, networking, a highly versatile scripting language and a powerful visual editor. This simulator is now extended with the dynamics of a walking robot and validated for the humanoid robot Nao. On this basis many other robotic applications as benchmarked in the RoboCup initiative become possible.

Keywords: simulation, multiple kinematic chains, dynamics

1 Introduction

Robotic simulation is essential in developing control and perception algorithms for robotics applications. Simulation creates the environment with known circumstances, which allows rapid prototyping of applications, behaviors, scenarios, and many other high-level tasks. Robot simulators have been always used in developing complex applications, and the choice of a simulator depends on the specific tasks we are interested in simulating. Yet, the level of realism of a simulator is also important in this choice.

A 3D simulator for mobile robots must also correctly simulate the dynamics of the robots and of the objects in the environment, thus allowing for a correct evaluation of robot behaviors in the environment. Moreover, real-time simulation is important in order to correctly model interactions among the robots and between the robots and the environment. Since simulation accuracy is computationally demanding, it is often necessarily an approximation to obtain real-time performance [1].

In this paper the focus is on the humanoid Nao robot, which is selected by the RoboCup organization as the standard platform for the Soccer competition. In addition, this robot is also used in the @Home competition [2, 3] (see Fig. 1). This robot is widely used in many research institutes around the globe. The Nao

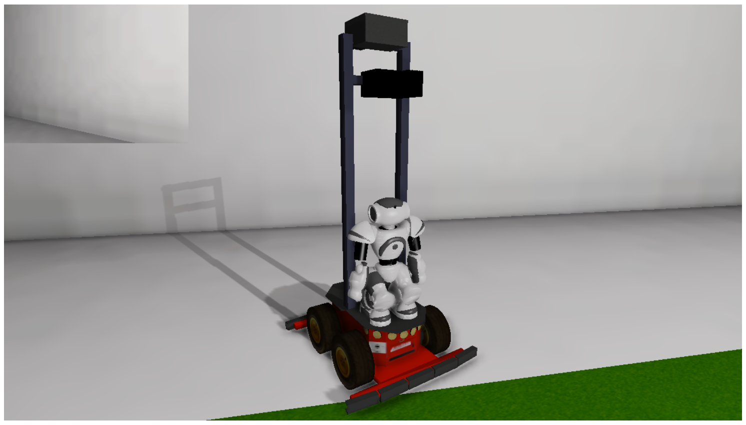


Fig. 1: Configuration of a humanoid robot on a wheeled platform in USARSim, as used in the RoboCup @Home [2].

contains several kinematic chains (legs, arms, head), which means that its model can be the basis of other robots with multiple kinematic chains.

A model is described to replicate the dynamics of the Nao robot in USARSim [4]; an existing 3D simulator based on the Unreal Engine. Inside USARSim robots are simulated on the sensor and actuator level, making a transparent migration of code between real robots and their simulated counterparts possible. USARSim is an open source project, available on sourceforge[1]. It includes a powerful editor to create worlds and allows experiments, benchmarks and competition scenarios to be set up easily.

2 Related Work

There are many robotic simulator platforms available. The first legged robot developed inside USARSim was the Aibo [5]. The first humanoid robot developed inside USARSim was the Robovie-M [6], developed by the Artisti Humanoid team. Both models were developed on basis of the Unreal Engine 2 / Karma Physics engine. With this engine four Aibo's or two Robovie-M could be simulated before the framerate dropped below an acceptable level. Currently USARSim is based on Unreal Engine 3 / NVidia PhysX. The latter physics engine is more focused on parallelization to make optimal usage of modern cpu's.

Inside the RoboCup @Home League simulation are sparsely used [2, 3, 7–10]. Teams typically use older simulation environments, such as Gazebo [7, 8] or Carmen [9]. Another possibility is to use a commercial package like Webbots [10]. Essential for this League is to be able to use innovative robot and sensor combinations, a rich environment with a wide variety of shapes and textures,

[1] http://usarsim.sourceforge.net

natural lighting, support of the Kinect and preferably a ROS interface. USARSim fulfills all those prerequisites [11].

SimSpark[2] is the official 3D RoboCup simulator and is primarily made for this goal. The simulator is open source and freely available. It uses a client-server architecture, where agents (i.e. robot controllers) are the clients that communicate with the simulation server. A limited number of robots (mainly the Nao) are supported, although it is made easy to add new robots with `rsg` files that describe the physical representation of a robot.

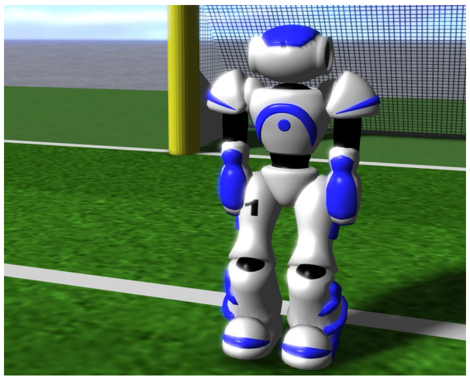


Fig. 2: Screenshot of SimSpark, the simulator used in the Soccer Simulation League

SimSpark always starts a football simulation, including a soccer field, game states and referee. The robots are controlled using a custom protocol, not the native interface of the Nao.

3 Simulation Model

The RoboCup version of the Nao (H21 model) has 21 joints, resulting in 21 degrees of freedom (DOF). There is also an academic version with 25 degrees of freedom, which has 2 additional DOF in each hand.

The movement of each joint can be described by a rigid body equation[12]. The first step is to definition of unconstrained motion as described in equation (1). This equation contains four vectors, it takes both the spatial information $x(t)$, $R(t)$ and the linear and angular momentum $P(t), L(t)$ into account. $F(t)$ and $\tau(t)$ are external forces and the input to solve this equation. The linear and angular speed $v(t)$, $\omega(t)$ can be derived from the linear and angular momentum when the total mass M and the inertia tensor $I(t)$ of a rigid body is known.

[2] http://simspark.sourceforge.net

$$\frac{d}{dt}Y(t) = \frac{d}{dt}\begin{bmatrix} x(t) \\ R(t) \\ P(t) \\ L(t) \end{bmatrix} = \begin{bmatrix} v(t) \\ \omega(t)^* R(t) \\ F(t) \\ \tau(t) \end{bmatrix} \tag{1}$$

The inertia tensor $I(t)$ is time dependent, but can be calculated from the inertia tensor I_{body} in body space, which is a fixed property, by taking the orientation of the body into account $I(t) = R(t) I_{body} R(t)^T$.

$$\left[v(t) = \tfrac{P(t)}{M}, \omega(t) = I(t)L(t) \right]^T \tag{2}$$

The next step is to take contacts into account. When the rigid body encounters a contact it imposes a constraint on the movement.

Two different types of contacts can be distinguished. The first is a contact caused by bumping into another rigid body or into the world. The other type of contact is caused by having a joint defined between two rigid bodies which are part of the robot.

3.1 Joint definition and convention

As said in the previous section, a joint connects two rigid bodies and limits the movement in some way. The type of movement limitation results in different types of joints, like a rotational joint, translational joint (also called prismatic joint), spherical joint, screw joint, etc.

A rotational joint, also called *revolute joint*, is as the name suggests capable of rotating around an axis. This type of joint allows one degree of freedom (DOF) between the two rigid bodies, namely the range of motion around the specified axis. In case of this type of joint the motion is usually also limited to a specified range around the axis.

It is important how the relative position and orientation of the frames is characterized. A commonly used convention to describe this is the *Denavit Hartenberg* (DH) notation. This convention uses homogeneous transformation matrices to describe the relative positions of the frames (coordinate systems). This convention is used in USARSim. A full description can found in the book Robotics, chapter 2.2.10, by K.S Fu *et al.*[13].

The Denavit Hartenberg representation is visualized in figure 3. Red lines show the z axes (motion axis) of the joints, while the yellow and green lines show respectively the y and x axes of the joints. The middle blue line shows the start z axis. The other blue lines represent the end points of the five joint chains. Each transformation is represented by a translation / rotation matrix.

3.2 Shape Definition

The shape of the robot is needed to detect collisions between parts of the robot. To define the shape of each part use can be made of the representations of the

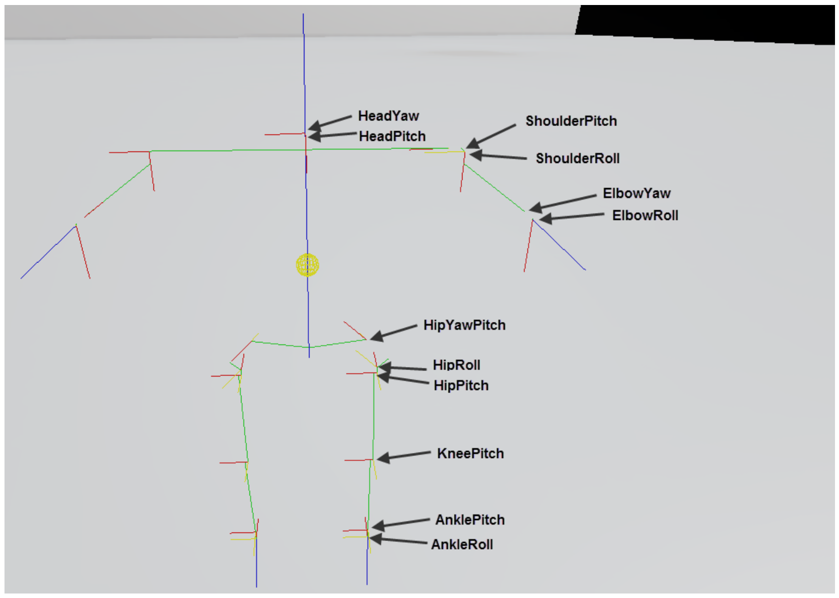


Fig. 3: Visualization of joints according to the Denavit Hartenberg convention. Red lines show the z axes, yellow the y axes and green the x axes of the joints.

Unreal Engine. Two collision representations are relevant for simulations of robot in USARSim. The first collision representation is intended for *static meshes* in Unreal Engine. Static meshes are a type of meshes that are not dynamic. This name does not imply they cannot move or interact with the world. The advanced option for static meshes is to check collisions per polygon against the static mesh 3D model itself and is potentially expensive to use. There is also a (simplified) collision hull option, but this option is not used for robots inside USARSim. Additionally there is a collision representation which is intended for *skeletal meshes* in the Unreal Engine. Skeletal meshes are used for game characters, not for USARSim robots.

The second collision representation is intended for PhysX and is created in the same way as the advanced static mesh version. The PhysX collision model is used in the physics simulation. However sensors will usually involve collision detection with the first representation. For example a simulated sonar sensor uses Unreal Engine tracing to detect objects in the world, which uses the Unreal Engine collision model. Both representations are needed for a realistic simulation of a robot. Care has been taken (as can be seen in Fig. 4) to keep both representations equivalent for the Nao robot. Both the link and shape representation are described in more detail in [1].

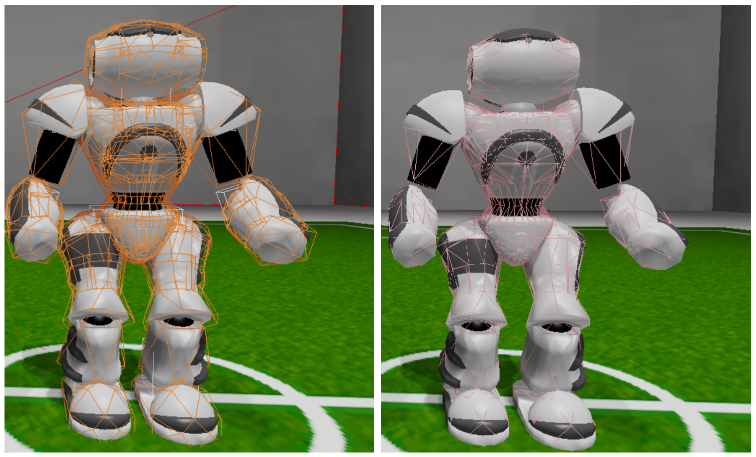


Fig. 4: The left picture shows the PhysX collision model, the right picture the Unreal Engine collision model.

4 Experiments

The experiments are divided into two categories; experiments which check general properties for constrained rigid body motion and experiments that are directly related to the proposed Nao model. The basic experiments for constrained rigid body motion are described in an earlier paper [1]. Here we concentrate on the possible applications.

4.1 Advanced Experiments

In this section experiments are done with the simulated and real Nao. The results of these experiments are compared to see how close they resemble each other. The experiments all consist of the combined movement of multiple joints. A more simple version of this experiment would be the movement of a single joint (for instance turning the head). Such simple experiments are performed and show close correspondence. The more advanced experiments are more interesting in the sense that they show sometimes unexpected results due to the interaction of the constraints in between joints. Alternative advanced experiments would the Tai Chi balance (demonstrated in [1]) and collisions between two robots (demonstrated in [5]).

Walking Realistic walking comparable to the walking behavior of the real Nao is crucial in a humanoid simulation. During a RoboCup match a robot will have to walk a large part of the time.

For this experiment several walking and turning tests were done for the simulated and real Nao using the included walk engine of the Nao provided by

Aldebaran. This walk engine uses a simple dynamic model inspired by work of Kajita *et al.*[14] and is solved using Quadratic programming [15]. When walking at full speed it can reach a velocity of $9.52 cm/s$ and $42 deg/s$ when turning.

In this test the Nao was set to do a single full step with the left leg. The joint angles of the real and simulated Nao were recorded and compared.

Fig. 5 shows the average joint angles of the LKneePitch joint (i.e. the left knee) with standard deviation over ten recordings of the real and simulated Nao. The standard deviation for the real Nao is lower than the simulated Nao. The same behavior is also seen for the standard deviations of the other joints.

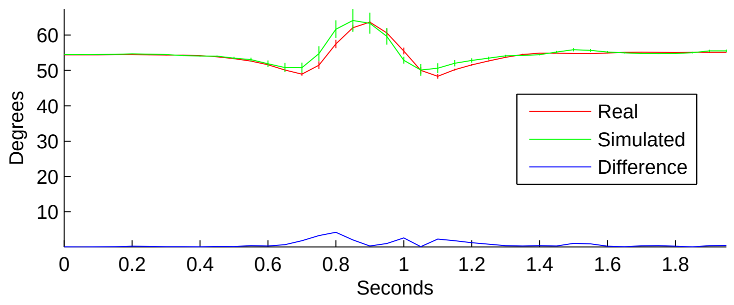


Fig. 5: Average joint angles with standard deviation of the LKneePitch joint while executing a single step. Joint angles were averaged over ten runs for the real (red) and simulated (green) Nao. The blue line shows the difference between the joint angles trajectories.

More walking experiments (including walking multiple steps straight and in circles) are described in [1].

Kicking Another motion is a kick of a ball with the right leg. The motion was performed ten times for both the real as simulated Nao robot. The recorded joint angles were averaged and the standard deviation was computed.

Figure 6 shows the average joint angles of the RAnkleRoll joint with the standard deviation and the difference between the average joint angles. This joint is interesting because the joint angles trajectory is not the same. During the kick motion the RAnkleRoll joint is told to move to 10 degrees in half a second and stay at 10 degrees for the remaining part of the motion.

The angles trajectory shows not much difference in moving towards 10 degrees, although the standard deviation of the real Nao angles is higher than the simulated Nao angles. However when staying at 10 degrees the real Nao joint is not able to maintain this angle around 1.5 second. In this case the Nao fails to reproduce the behavior of the real joints because we did not include the restric-

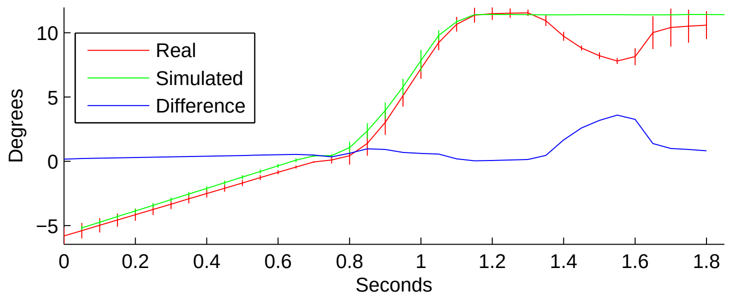


Fig. 6: Joint angles and standard deviation of the RAnkleRoll joint while executing a kick motion. Results were averaged over ten runs. The red line shows the angles trajectory of the real Nao, while the green line shows the same for the simulated Nao. The blue line shows the difference between the two angle trajectories.

tions of the collision hull of this particular joint in our model[3]. The joint angle range of this joint is limited by the movements of the AnklePitch joint. Around 1.5 second the RAnklePitch joint moves from around -30 degrees to -60 degrees. At this joint angle the RAnkleRoll becomes limited to a range of between -6 and 3 degrees.

5 Full Application Experiment

To test how well the performance is for real applications, the source code of the Dutch Nao Team[16] has been tested with USARSim.

This application not only involves walking around, but also perception of the ball and dedicated behaviors like kicks and standing up.

To test real applications an intermediate program has been created, UsarNaoQi, which works as a proxy server, converting NaoQi messages in USARSim messages and vice versa. NaoQi is the framework provided by Aldebaran and allows the user to control the Nao in various programming languages (C++, Python, C# or Urbi).

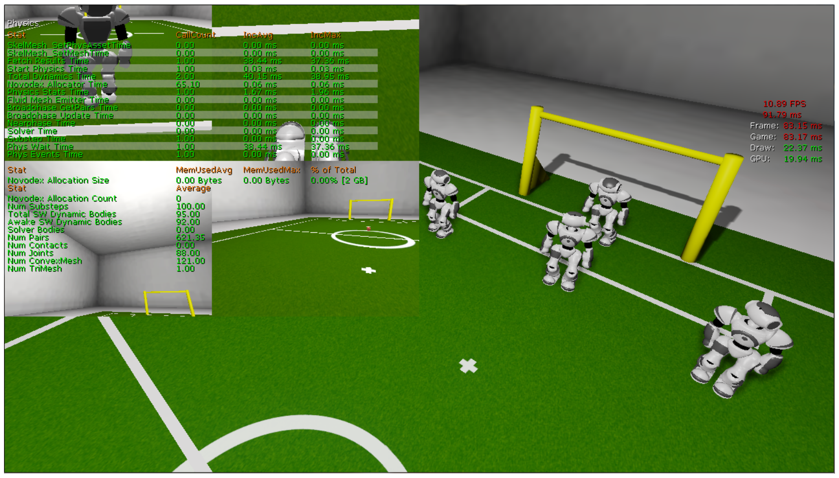
The source code of the Dutch Nao Team is written in Python, and could be directly applied. The code was fully functional, the robots could standup, position themselves on the field, locate the ball and kick the ball. The only observed difference is in the approach of the ball; the Dutch Nao Team code makes a number of small steps to get in a good position behind the ball. In simulation those steps are too small; the Nao needs too much time to position itself.

[3] http://www.aldebaran-robotics.com/documentation/nao/hardware/kinematics/nao-joints-33.html.

The experiment was performed by putting a number of Nao robots in the simulated RoboCup environment. The average frames per second (FPS) was recorded for two different scenarios. In the first scenario the Nao is simply standing and doing nothing. In the second scenario we executed the Nao with robot controller from the Dutch Nao Team. The controller was set in *play* mode. In this mode the Nao will walk around scanning for the ball.

The experiment was performed on a computer with an Intel iCore 7 920 processor and an AMD Radeon HD 6850 graphics card. USARSim was used in combination with the UDK December build 2011. UsarNaoQi was set to use a time step of 10ms; the Naos in USARSim sent status updates at a rate of 100 times per seconds (joint angle updates).

Without any Naos the scene is rendered at a FPS of 320. With one and two Naos the FPS drops to around 110 and 55 respectively, which is enough for running a decent simulation. With three Naos the FPS drops to 30, which is still acceptable. With four Naos the simulation frame rate drops to 10 FPS, resulting in incorrect movements.



```
Physics
Stat                          CallCount   IncAvg      IncMax
SkelMesh_SetPhysAssetTime     0.00        0.00 ms     0.00 ms
SkelMesh_SetMeshTime          0.00        0.00 ms     0.00 ms
Fetch Results Time            1.00        38.44 ms    37.36 ms
Start Physics Time            1.00        0.03 ms     0.03 ms
Total Dynamics Time           2.00        40.15 ms    38.95 ms
Novodex Allocator Time        65.10       0.06 ms     0.06 ms
Physics Stats Time            1.00        1.67 ms     1.96 ms
Fluid Mesh Emitter Time       0.00        0.00 ms     0.00 ms
Broadphase GetPairs Time      0.00        0.00 ms     0.00 ms
Broadphase Update Time        0.00        0.00 ms     0.00 ms
Nearphase Time                0.00        0.00 ms     0.00 ms
Solver Time                   0.00        0.00 ms     0.00 ms
Substep Time                  1.00        0.00 ms     0.00 ms
Phys Wait Time                1.00        38.44 ms    37.36 ms
Phys Events Time              1.00        0.00 ms     0.00 ms

Stat                          MemUsedAvg  MemUsedMax  % of Total
Novodex Allocation Size       0.00 Bytes  0.00 Bytes  0.00% [2 GB]
Stat                          Average
Novodex Allocation Count      0
Num Substeps                  100.00
Total SW Dynamic Bodies       95.00
Awake SW Dynamic Bodies       92.00
Solver Bodies                 0.00
Num Pairs                     621.35
Num Contacts                  0.00
Num Joints                    88.00
Num ConvexMesh                121.00
Num TriMesh                   1.00
```

```
   10.89 FPS
   91.79 ms
Frame:  83.15 ms
Game:   83.17 ms
Draw:   22.37 ms
GPU:    19.94 ms
```

Fig. 7: Four Naos in action with the physics statistics displayed

To find the performance bottlenecks in the simulation various profiler tools provided by UDK are used (PhysX statistics and UnrealScript code profiler). Using these tools reveals that when simulating four Naos half of the frame time is spent in the physics. The remaining part of the time goes to the sonar sensor (tracing), receiving and processing messages in the bot connection with the controller, sending the current status to the controller (joint angles) and updating the current joint angles.

6 Discussion

Sensors The experiments are limited to the motion of the simulated Nao caused by the movement of the joints. However the Nao is equipped with a wide range of sensors (as discussed in the introduction). The different sensors like the cameras, bumpers, sonars and inertial unit also contribute to the behavior of the Nao. More research is needed specifically aimed at these sensors. For example the Nao is equipped with two cameras. Although the camera sensors obviously function, it is not possible to say much about the correct working of these cameras without validation. Figure 8 shows an example of the problems you encounter when simulating a camera. The sensitivity of the camera of the different Nao versions results in a different camera image. Such differences would need to be modeled to simulate a camera properly. Although Unreal Engine already offers excellent rendering options, the current implementation in USARSim limits the simulation of the camera to simply capturing the image and sending it without modification.

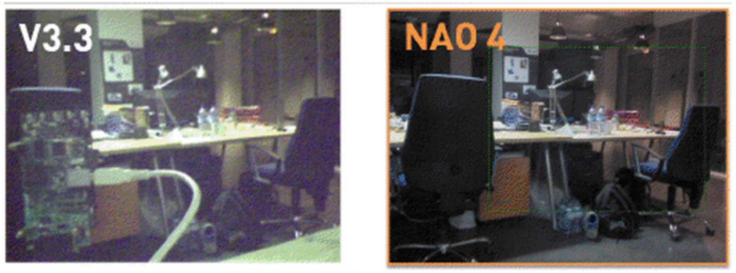


Fig. 8: Camera image of Nao 3.3 vs 4.0 (Courtesy Aldebaran Robotics)

Servo motor Another interesting research option is to extend the simulator with a more realistic servo motor and gears simulation, as used in the MoToFlex simulator[17]. In a physics engine a common way to control the joints of a robot is to set the desired joint angles and leave it to the physics engine to satisfy the constraints between the links (as described in this paper). This approach is not the most realistic way to drive a joint and the method seen in the MoToFlex simulator[17] could improve the behavior of the joints.

Scaling issues Due several design choices it is not possible to scale up to high number of simulated Nao robots. Scaling up the number of Naos is important for simulating a RoboCup scenario. The goal of the RoboCup competition is to play a real football match with 22 Naos. Part of the reason why the simulation cannot scale up to this number of Naos is due the choice of the collision model, the physics timing step and possible overhead of message parsing and other sonar sensor. Scaling of the number of Naos could be improved by simplifying the collision model by using more simple shapes (spheres, boxes) and lowering the Physics timestep settings. The same could be applied to the message parsing (moving the code from Unreal Script to C for example) and the sonar sensor (using less traces to determine the sonar distance).

Extending to other legged robots One of the goals was to make a generic model that could be applied to other limb typed robots, like the ABB Frida or a spider like robot. The collision tools of Unreal Engine allows to quickly create varies collision shapes, varying from simple models (boxes, spheres) to complex convex models, possible based on the visual shape of the robot.

This model can easily be applied to other robots with multiple kinematic chains as shown in figure 9.

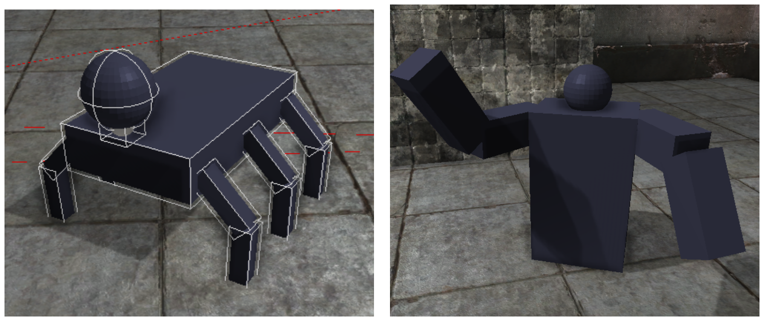

(a) A spiderlike robot (b) A two arm manupulator robot

Fig. 9: Example of robots with multiple kinematic chains

7 Conclusion

In this paper we demonstrated that the simulation of the Nao in USARSim resembles reality quite closely. Our current model is usable in practice on the condition that one keeps in mind the limits of the method; like the walking behavior and the scaling issues with the number of Naos. The combination of Unreal/USARSim provides several advantages over other robot simulators. The simulation is at such a level that transparent migration of code between real robots and their simulated counterparts is possible. In this paper this is demonstrated with an intermediate program, UsarNaoQi, which enables access to the simulated robot with its native interface. Using this interface several experiments have been performed with both the real and simulated robot. The model developed for this humanoid robot demonstrates that robots with complex dynamics could be realistically modeled inside USARSim, which could be the basis of the introduction of other models of complex robots into USARSim like two-arm manipulators and/or service robots.

References

1. van Noort, S., Visser, A.: Validation of the dynamics of an humanoid robot in usarsim. In: Proceedings of the Performance Metrics for Intelligent Systems Workshop (PerMIS'12). (March 2012)
2. van Elteren, T., Neculoiu, P., Oost, C., Shantia, A., Snijders, R., van der Wal, E., van der Zant, T.: Borg - the robocup@home team of the university of groningen - team description paper. In: Proc. CD of the 15th RoboCup International Symposium. (July 2011)
3. Dessimoz, J.D., Gauthey, P.F.: Rh6-y toward a cooperating robot for home applications. In: Proc. CD of the 15th RoboCup International Symposium. (July 2011)
4. Carpin, S., Lewis, M., Wang, J., Balakirsky, S., Scrapper, C.: Usarsim: a robot simulator for research and education. In: Proceedings of the International Conference on Robotics and Automation (ICRA'07). (2007) 1400–1405
5. Zaratti, M., Fratarcangeli, M., Iocchi, L.: A 3D simulator of multiple legged robots based on USARSim. In: Robocup 2006: Robot Soccer World Cup X. Volume 4434 of Lecture Notes in Artificial Intelligence., Springer (2007) 13–24
6. Greggio, N., Menegatti, E., Silvestri, G., Pagello, E.: Simulation of Small Humanoid Robots for Soccer Domain. Journal of the Franklin Institute **346**(5) (June 2009) 500–519
7. Lunenburg, J., Clephas, T., Dirkx, N., Willems, B., Elfring, J., Sandee, J., van de Molengraft, M.: Tech united eindhoven team description 2011. In: Proc. CD of the 15th RoboCup International Symposium. (July 2011)
8. Alenyà, G., Tellez, R.: The reem@iri 2012 robocup@home team description. In: Proc. CD of the 16th RoboCup International Symposium. (June 2012)
9. del Solar, J.R., Correa, M., Lee-Ferng, J., Hevia-Koch, P., Parra, I., Mascar, M.: Uchile homebreakers 2010 team description paper. In: Proc. CD of the 14th RoboCup International Symposium. (June 2010)
10. Lallee, S., lise Jouen, A., Petit, M., Madden, C., Boucher, J.D., Weitzenfeld, A., Dominey, P.F.: Cooperative human robot interaction with the nao humanoid: Technical description paper for the radical dudes. In: Proc. CD of the 15th RoboCup International Symposium. (July 2011)
11. Balakirsky, S., Kootbally, Z.: USARSim/ROS: a combined framework for robot control and simulation. In: Proceedings of the ASME 2012 International Symposium On Flexible Automation (ISFA 2012. (June 2012)
12. Baraff, D.: An introduction to physically based modeling: rigid body simulation I - unconstrained rigid body dynamics. SIGGRAPH Course Notes (1997)
13. Fu, K., Gonzalez, R., Lee, C.: Robotics: control, sensing, vision, and intelligence. McGraw-Hill (1987)
14. Kajita, S., Tani, K.: Experimental study of biped dynamic walking. Control Systems Magazine, IEEE **16**(1) (1996) 13–19
15. Wieber, P.: Trajectory free linear model predictive control for stable walking in the presence of strong perturbations. In: Proceedings of the International Conference on Humanoid Robots. (2006) 137–142
16. Verschoor, C., et al.: Dutch nao team - code release 2011 and technical report 2011. Published online, Universiteit van Amsterdam (October 2011)
17. Urbann, O., Kerner, S., Tasse, S.: Rigid and soft body simulation featuring realistic walk behaviour. In: RoboCup 2011: Robot Soccer World Cup XV. Lecture Notes in Computer Science, Springer Berlin / Heidelberg (2012)