MODELING OF ROBOT LEARNING IN MATLAB/SIMULINK ENVIRONMENT

Bojan Nemec, Leon Žlajpah

Jožef Stefan Institute
Jamova 39, 1000 Ljubljana, Slovenia

bojan.nemec@ijs.si(Bojan Nemec)

Abstract

The paper describes our environment for off-line programming and control design of robot systems developed in Matlab/Simulink environment. A special emphasis has been given on robot learning. Nowadays, it is commonly accepted that preprogrammed robots are applicable only in highly structured environments. In order to bring robotic technology in every-days life as well to be used for small batches of unstructured production, it is required that robots poses certain level of self-adaptation. One of the important aspect of the self adaptation is unsupervised learning, which imitates a learning processes of animals and human beings. Learning is a long process, based on many successful or unsuccessful repetitions of a given task. For that a judgment on how successful was the previous attempt is crucial. Usually, learning assures a convergence to the globally optimal solution. Since successful learning depends on many parameters, like given rewards, learning speed, noise rejection, etc. the simulation becomes an essential tool for designing and tuning the learning algorithms. In the paper, we describe our simulation models for learning the ball-in-a-cup game, which is often used as a test bed for various learning algorithms.

Keywords: robot simulation, robot learning, robot programming.

Presenting Author's Biography

Nemec Bojan is senior research associate at Dept. of Automatics, Biocybernetics and Robotics, Jozef Stefan Institute. He received BS, MSc and PhD degree from the Univerity of Ljubljana in 1979, 1982 and 1988 respectively. In 1993 he spent his sabbatical leave at the Institute for Real-Time Computer Systems and Robotics, University of Karlsruhe. His research interests include robot control, robot simulation, sensor guided control, service robots and biomechanical measurements in sport. Between 2002 and 2005 he was a task leader in the largest NAS European project EUROShoE. He has published over 100 conference and journal papers and is author of 1 patent, and co-author of a book.

1 Introduction

Nowadays, the system designer can relay on a variety of different software tools which can significantly increase his efficiency. Among them, the simulation has been recognized as an important tool in designing new products, investigating their performances and also in developing applications of these products. For complex systems such as robots the simulation tools can certainly enhance the design, development, and even the operation of the robotic systems. Augmenting the simulation with visualization tools and interfaces, one can simulate the operation of the robotic systems in a very realistic way.

The simulation tools for robotic systems can be divided into two major groups: tools based on general simulation systems and special tools for robot systems. Tools based on general simulation systems are usually special modules, libraries or user interfaces which simplify the building of robot systems and environments within these general simulation systems. Special simulation tools for robots cover one or more tasks in robotics like off-line programming, design of robot work cells, kinematic and dynamic analysis, mechanical design. They can be specialized for special types of robots like mobile robots, underwater robots, parallel mechanisms, or they are assigned to predefined robot family. Depending on the particular application different structural attributes and functional parameters have to be modeled. The majority of the robot simulation tools focus on the motion of the robotic manipulator in different environments and among them an important group are the tools for the design of robot control systems.

Currently, many different simulation tools are available, which could be used in research and teaching laboratories. However, these tools are not always fulfilling all the requirements of the research and teaching activities in robotic laboratories. Reconfigurability and openness are features already recognized by many as essential in the development of advanced robot control algorithms. Not only is it important to have easy access to the system at all levels (e.g. from high-level supervisory control all the way down to fast servo loops at the lowest level), but it is a necessity to have open control architectures where software modules can be modified and exteroceptive sensors like force/torque sensors and vision systems can be easily integrated. Reconfigurability should also be reflected when more fundamental changes to the controller architecture are required, in the necessity of quickly being able to make modifications in the original design and verify the effect of these modifications on the system. In other words, the user should be able to quickly modify the structure of the control without having to alter the simulation system itself.

One of the widely used general simulation platforms for the modeling and simulation of various kind of systems MATLAB. Developed by MathWorks, MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, symbolic computation and interfacing with programs written in other languages. A number of additional toolboxes efficiently solves problems regarding technical computing, embedded systems, control systems, digital signal processing, communication systems, image and video processing, mechatronics, computational biology, computational finance and others. Therefore, it is not surprising that it has been used intensively for the simulation of robot systems. One of such tools, The Robotics Toolbox [1], provides many functions that are required in robotics and addresses areas such as kinematics, dynamics, and trajectory generation. The Toolbox is useful for simulation as well as for analyzing the results from experiments with real robots, and can be a powerful tool for education. However, it is not integrated into Simulink environment and not appropriate for and for the hardware-in-the-loop simulation. However, it is not very good for the simulation in Simulink and for the hardware-in-the-loop simulation "SimMechanics Toolbox" [2] extends Simulink with the tools for modeling and simulating mechanical systems. With SimMechanics, one can model and simulate mechanical systems with a suite of tools to specify bodies and their mass properties, their possible motions, kinematic constraints, and coordinate systems and to initiate and measure body motions.

In the paper, we present our MATLAB/Simulink based simulation environment, dedicated to effective design of high an low level control algorithms, sensory interaction and testing of the hardware in the loop. One of the major benefits of the proposed approach is, that allows to include the real hardware at any point of the simulation loop. Therefore, is is possible to switch from the simulation to the real hardware experiment without any modification of the program code. This is extremely important when testing new control algorithms and strategies for solving complex tasks, usually related to service and humanoid robotics. These tasks usually require that robots poses learning capabilities and certain level of self-adaptation. Nowadays robot learning has been intensively investigated as a promising way to execute complex tasks arising from the needs of humanoids and service robotics. In the past, a number of algorithms for the robot learning were proposed, starting from the supervised learning based on neural networks, support vector machines, to the unsupervised learning based on reinforcement learning, genetic algorithms, etc.. Common to all these approaches is that they require relatively large number of repetitions in order to learn the required skill. Especially when developing new algorithms or just when tuning the learning parameters, this becomes very time consuming and tedious procedure especially if executed on the real system. Therefore, the simulation is used for learning as a rule. However, if we want to use skills learned on simulation also for the target system we need precise models. Our approach is presented through an example of learning a robot to play the ball-in-a-cup game, also known as the "kendama" game. The aim of this game is to swing the ball hanging from a cup down on a string and to catch it with the cup.

2 Simulation environment

In the last years, the scope of our research has been oriented more in the development of control systems for humanoid and service robots. These robots have in general a complex mechanical structure with many degrees-of-freedom. Consequently, complex kinematic and dynamic models are necessary to simulate them adequately. Furthermore, the control methods and algorithms we are developing are usually a part of the higher robot control levels and it is assumed that the low level close-loop control algorithms are a solved issue. These high level control algorithms can become very complex and may even require parallel computation distributed over more computers.

Therefore, we had to reconsider the concept of the control design environment we will use in future. We have augmented our simulation environment with components that

- simulate the kinematics and dynamics of an arbitrary chosen kinematic chain describing different manipulators
- enable integration of different sensor systems like vision and force sensors
- enable simulation of scenarios for complex robot tasks
- model the robots' environments
- visualize the robots and their environment
- enable integration of real robots in the simulation loop

Based on our good experience with MATLAB/Simulink we have decided that MATLAB/Simulink will be the kernel of our simulation tools. However, some of the above requirements can be easier fulfilled by using other tools. For example, the visualization of the robot and the environment can be easily done by dedicated graphics tools. Furthermore, advanced robot control strategies rely intensively on feedback sensor information. The very complex sensor system is the vision system, which can have several configurations and can be implemented on a single computer or on a computer cluster composed of many computers running different operating systems. To integrate such a diversity of hardware components in an unique framework we have decided to use the ethernet communication and the UDP protocol. In this way, we have a maximal possible "degree-of-openness" of the system. In our environment, each block can represent a real system or a model of that system. Note that because we are using ethernet communication between the blocks, different software tools on different platforms can be used to simulate specific parts of the system. Consequently, the simulation environment can consist of several interacting applications, each representing a part of the system.

In Simulink, a system is modeled by combining input-output blocks. To gain the transparency we try to represent a system by the block structure with several hierarchical levels, i.e. by combining different basic blocks subsystems are built which become a single block at the higher level. Figure 1 shows the Robot systems block library. The goal of the library is to provide blocks which are needed to simulate robotic systems and can not be modeled with standard blocks. First of all, this are the blocks for robot kinematic and dynamic models, the blocks for sensors systems, the typical transformations present in robot systems and the special interface blocks for robots, sensors and all other communications. Additionally, the library includes some blocks with standard subsystems like task space controllers, trajectory generation modules, etc.

3 Modeling of the ball-in-a-cup game

When learning different tasks different models are needed. Unlike when learning a tennis swing, where a human needs just to learn the goal function for the very moment when the racket hits the ball, the Ball-in-a-Cup task where the ball and the cup constantly interact relies on the complete dynamics of the cup and the ball. In order to build a simulation model, we first need an accurate dynamic model of both parts. The part, i.e. the subsystems of the ball-in-a-cup game, were modeled using the Open Dynamics Engine (ODE) package [3]. With ODE, we can successfully simulate the interaction (collisions) between the ball and a cup, both modeled as rigid bodies. Unfortunately, ODE has no capability of simulation the ball on a rope. Therefore, we had to build a special model for this purpose. Let us briefly present the dynamic model of the ball-on-a rope system.

Ball on a rope acts as a pendulum as long as the rope is not loose and as a free flying object otherwise. Both models are well known, but the problem is the switching between both models. It turns out, that is more appropriate and accurate to model the ball as a free flying object with external forces acting on it. External forces are air drag forces and the forces resulting from the rope tension. The overall dynamical model can be described by the set of equations

$$
\begin{aligned}
& m\ddot{\mathbf{z}} = \mathbf{f}_r + \mathbf{f}_a \qquad (1)\\
& \mathbf{f}_r = \begin{cases} (K_s d + K_d \dot{d})\mathbf{h}, & d > 0 \\ 0, & d \leq 0 \end{cases}\\
& \mathbf{f}_a = K_a \dot{\mathbf{z}}\\
& d = (l - l_0),
\end{aligned}
$$

where $\mathbf{z}$ is the vector of the ball position, l is actual rope length, l_0 is the length of the untended rope, $\mathbf{h}$ is the unit vector describing the tent rope direction, K_a is the air drag constant and K_s and K_d are the rope stiffens and rope damping respectively.

Since the ball-in-a-rope game is very sensitive to the exact trajectory guidance over the entire game phase, we need an accurate dynamic model of the robot. The robot could also be modeled using ODE, but it turns out that the simulation model is far more accurate, numerically stable and efficient if modeled explicitly. The dynamics of the n degrees of freedom articulated robot arm can

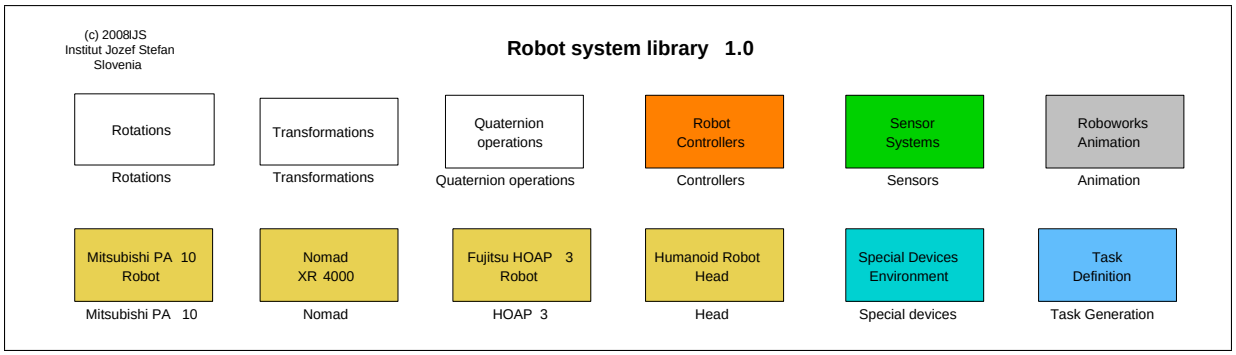

Fig. 1 Simulink Robot systems library

be presented with the well know equation

$$\ddot{\boldsymbol{q}} = \boldsymbol{H}^{-1}(\tau - \boldsymbol{C}\boldsymbol{q} - \boldsymbol{\xi} - \boldsymbol{g} - \boldsymbol{J}^T\boldsymbol{F}), \qquad (2)$$

where $\boldsymbol{q}$ is n dimensional vector of joint angles, τ is n dimensional vector of joint torques, $\boldsymbol{H}$ is an $n \times n$ symmetric, positive definite inertia matrix , $\boldsymbol{C}$ is $n \times n$ matrix of nonlinear terms due to the centrifugal and Coriolis forces, $\boldsymbol{\xi}$ is n dimensional vector of friction forces, $\boldsymbol{g}$ is n dimensional vector of gravitational forces, $\boldsymbol{J}$ is $n \times m$ dimensional Jacobian matrix, m is the number of the task coordinates and $\boldsymbol{F}$ is an m dimensional vector of environment contact force acting on the end-effector. Nowadays the calculation of matrices $\boldsymbol{H}$, $\boldsymbol{C}$ and vector $\boldsymbol{g}$ is becoming easy thanks to a number of dedicated program packages for efficient dynamics calculation such as SD-Fast [4], Modelica [5], Newton [6], etc. Many of them return not only the efficient mathematical model, but also the program code. On contrary, a correct friction model remains still a problem. Namely, the friction differs from the robot to robot of the same type and also changes with the time. Therefore, the friction should be estimated and compensated with an appropriate algorithm. In our lab we have developed a friction estimation and compensation algorithm based on two layer neural network [7].

An important feature of the simulation is also the visualization. It is very important to visualize the simulation results. Especially in robotics it is necessary to "see" the motion of the robot and objects in the working environment. In our system we relay on external software for the visualization and animation of robots. In general, joint angles of robotic manipulators as well as the position and orientation of the other simulated objects in the scene are passed to the visualization tools using TCP/IP or UDP protocol. Currently, we have integrated into our simulation environment two visualization software packages - RoboWorks [8] and Blender[9]. Roboworks incorporates simple, but efficient modeler. Because of its simplicity RoboWorks package is the favorable tool for the visualization of simpler systems, i.e. one or two robots in non-complex environment. Figure 2 shows Roboworks model of the robot arm Mitsubishi Pa10 mounted on mobile platform Nomad XR4000 during the simulation of the ball-in-a-cup game. For more complex scenes we use Blender, an open source multi-platform 3D computer animation program, which has a lot of features that are potentially interesting for engineering purposes, such as the simulation and programming of robots, machine tools, humans and animals, etc. and the visualization and post-processing of all sorts of data that come out of such biological or artificial devices. Blender supports also scripts (via Python interfaces to the core C/C++ code) and has the capability of placing moving cameras at any link of the kinematic chain, it supports the real time photo realistic rendering for the virtual reality simulation and has also a physics engine for the simulation of the interactions between entities.

Fig. 2 Roboworks model of the ball-in-a-cup game

One of the key issues in the ball-in-a-cup game is also the prediction of the catching pose. The prediction is based on the estimation of the ball trajectory. Based on the dynamic model of the free-flying object, we can compute the ball trajectory using past observations and applying appropriate regression algorithm. A straightforward way would be to simply use the data from the ball-on-a-rope simulation block, but this violates our concept of transparency and interchangeability of the simulation blocks and the real-hardware blocks. Therefore, we have developed a special module for the cam-

era simulation. It captures images directly form the computer screen and is thus independent from the visual representation block. It acts as a virtual camera. In this block, we can simulate also different lighting conditions. Appropriate image processing algorithms from our library are then used in order to retrieve the required information from the real or virtual camera.

In order to define the initial trajectory for the swing-up phase of the ball-in-a-cup game, we need the simulation of the kinesthetic guidance. The simulation of the kinesthetic guidance is provided by using the haptic device, which controls the position and the velocity of the virtual hand holding the ball-on-a-rope system and delivers back the resulting forces. In this way it mimics the real kinesthetic feeling of playing a real ball-on-a-rope game. The figure 3 displays the simulation setup of the ball-in-a-rope game.

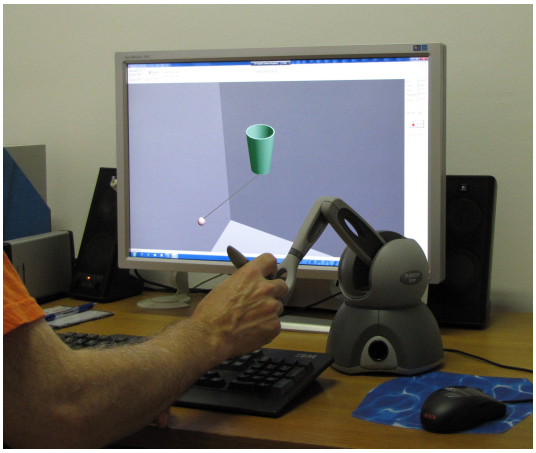

Fig. 3 Simulation of the kinesthetic guidance of the ball-in-a-cup game

4 Simulation of the supervised learning

Generally, supervised learning is a machine learning technique for deducing trajectories from a set of training data. In robotics, this kind of learning is often referred as learning by imitation. The imitation involves the interaction of perception, memory, and motor control. A straightforward approach is to mimic the human motion by recording a sequence of movements and to reproduce the same motion with the robot. In general, this approach fails due to different dynamics capabilities of the robot and humans. Therefore, the trajectory to be executed by the robot has to be modified appropriately. A central issue in the trajectory generation and modification is the choice of the representation (or encoding) of the trajectory. One of the most suitable representations, which facilitates adaptation, are the Dynamic Motion Primitives (DMP) [10]. In the standard DMP formulation, the motion in each task coordinate is represented as a damped mass-spring system perturbed by an external force. Such a system can be modeled with a set of differential equations

$$\begin{aligned} \dot{v} &= \frac{1}{\tau}\left(K(g-y) - Dv + f(x)\right), \\ \dot{y} &= \frac{v}{\tau} \end{aligned} \tag{3}$$

where v and y are the velocity and position of the system, x is the phase variable, which defines the time evolution of the trajectory, τ is the temporal scaling factor, K is the spring constant, and D is the damping. The phase variable x is defined by

$$\dot{x} = -\frac{\alpha x}{\tau}. \tag{4}$$

For the trajectory generation it is necessary that the dynamic system is critically damped and thus reaches the goal position without overshoots. A suitable choice is $D = 2\sqrt{K}$, where K is chosen to meet the desired velocity response of the system. Function $f(x)$ is a nonlinear function which is used to adapt the response of the dynamic system to an arbitrary complex movement. A suitable choice for $f(x)$ was proposed by Ijspeert et al. [11] in the form of a linear combination of M radial basis functions

$$f(x) = \frac{\sum_{j=1}^{M} w_j \psi_j(x)}{\sum_{j=1}^{M} \psi_j(x)} x, \tag{5}$$

where ψ_j are Gaussian functions defined as $\psi_j(x) = \exp(-\frac{1}{2\sigma_j^2}(x - c_j)^2)$. Parameters c_j and σ_j define the center and the width of the j-th basis function, while w_j are the adjustable weights used to obtain the desired shape of the trajectory. A suitable method for determining weights w_i from the demonstration trajectory is locally weighted least square regression [11].

First, in order to obtain an example for the imitation we have recorded the motions of a human player by the kinesthetic teach-in. In the simulation environment, we have used the haptic device "Phantom Omni", which controls the model of the ball-on-rope. In another setup, a haptic device might be substituted by a real time motion capture system 'OptoTrack'. In our integrated environment this can be done by simply substituting the model block. This makes the system transparent regarding the use of different hardware. As previously stated, the captured trajectory has to be adjusted according to the robot dynamics. For that, we have first encoded a set of training trajectories into DMP formulation. Each trajectory has been encoded using 22 parameter, goal g, duration τ and 20 weights w_i of radial basis functions ($M = 20$). The resulting parameters were averaged over entire set of the trajectories. It turned out that we can adapt the captured trajectory to the simulated robot dynamics by tuning just few of parameters, among which the temporal scaling τ was dominant.

5 Simulation of the reinforcement learning

In this section we present methods and tools for learning of swing-up motion for the ball-in-a-cup without

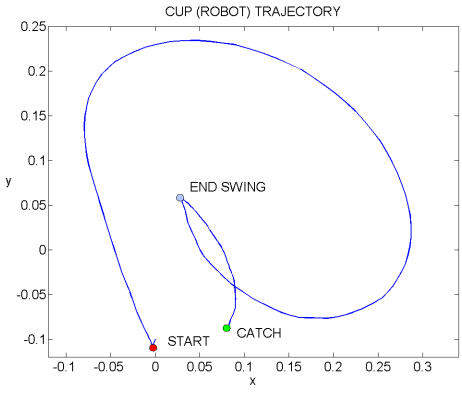

Fig. 4 Cup trajectory during the ball-in-a-cup game obtained with imitation learning

any previous knowledge of the system or the environment. We decided to evaluate SARSA-learning algorithm for this problem [12]. All previous studies used the function approximation reinforcement learning algorithms [13, 14], where the goal of the learning process was to tune the parameters of the swing-up trajectory. In most cases the learning process starts with a pre-learned trajectory, e.g. trajectory obtained from the kinesthetic guidance. On contrary, SARSA algorithm can be applied without any presumption of the environment model and has to implicitly learn the system dynamics trough exploration. During the exploration, the system collect rewards and tunes the control policy based solely on the accumulated rewards. The modified algorithm for the SARSA learning with eligibility traces [12] is described by the following Eq.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma Q(s_{t+1}, \quad a_{t+1}) - Q(s_t, a_t)]e(s) \tag{6}$$

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & \text{if } s \neq s_t \\ 1 & \text{if } s = s_t \end{cases}, \tag{7}$$

where Q is the is the action-value function and s_t and a_t are the state and the action of the agent at time t, respectively. α and γ denote the learning rate and discount rate; these two parameters determine the learning rate and the convergence, respectively. r_t denotes the reward associated with the agent state, obtained in the current time t. The Eq. 7 describes the exponential decay of the eligibility traces. Each time the agent is in a state s_t, the trace for that state is reset to 1. Eligibility traces help the agent to enhance the learning speed on good actions. One of the key properties of the learning system is how we assign states s and actions a. Best results were obtained by selecting states composed of the cup position x, cup velocity $\dot{x}$, angle between the cup and the ball φ and angular velocity $\dot{\varphi}$. For SARSA learning, states have to be discretized. The cup position has been described with two values, cup velocity with three values, ball angle with 18 values spanning over the entire circle and the ball angular velocity with 3 values, forming all together a set of 324 values. The action value has been chosen to be the cup acceleration $\ddot{x}$ described with 5 values spanning from the maximal negative to the maximal positive acceleration. Commonly, ϵ-greedy method is used for the learning. In our case we have obtained a much faster learning with a random set of harmonic trajectories used as input trajectories in the entire rollout. The learning goal was to swing the ball to the desired angle $\varphi(t_0)$ with the desired angular velocity $\dot{\varphi}(t_0)$. The behavior of the learning system is determined mostly by the reward assigned to current state. The final state was rewarded with a positive reward.

One of the major problem of swing-up learning is the limited displacement of the cup in x direction due to the limited workspace of the robot. To prevent the robot from reaching its limits, the excessive displacement x has been penalized by a negative reward. The learning requires many trails in order to learn the appropriate policy, which makes the simulation almost indispensable tool for robot learning. In average, 220 to 300 rollouts have been required to learn the swing-up trajectory. During the learning, the system implicitly builds models of the system kinematics, dynamics and limitations. In real environment, learning has to be continued in order to match the dynamics of the real robot, using the final value of the action value matrix Q, learned in the simulation, as an initial action value matrix in the real robot. Figure 5 shows one example of the obtained

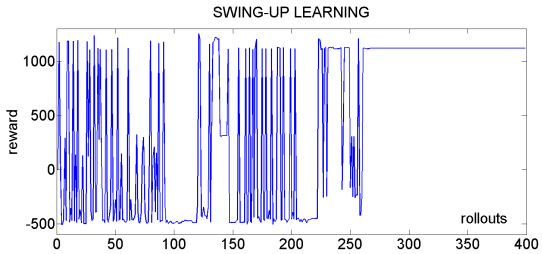

Fig. 5 Reward during swing-up learning

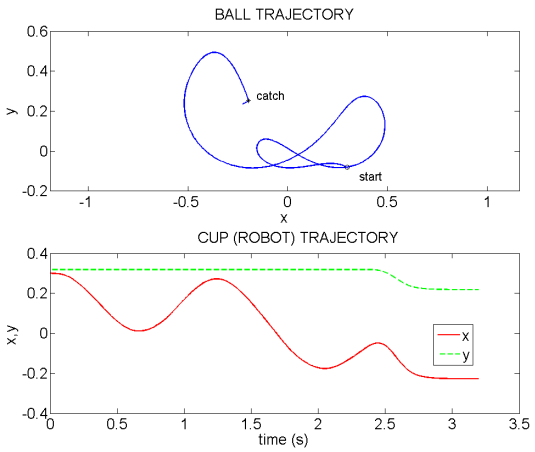

Fig. 6 Learned ball and swing-up trajectory

rewards during the learning. As we can see from the Fig. 5, the learning ends after approx. 260 rollouts, the robot repeats the learned trajectory and gets a constant reward. Figures 6 and 2 show the ball trajectory, the learned robot trajectory and the simulated robot pose after the successful learning, respectively. Note that the robot does not necessarily learns equal swing-up trajectory if we repeat leaning procedure again, because an infinite number of trajectories can swing-up the ball to the desired height. Note also that the learned trajectory different from the human demonstrated trajectory (Fig 4). Human demonstrator has used only one swing to swing-up the ball and the swing-up trajectory is in the $x - y$ plane. In contrast to that, the robot has been instructed to learn the swing up trajectory by moving x coordinate only. Due to the restricted accelerations, the learned trajectory has required two swings in order to swing-up the ball to the desired height.

6 Conclusions

The concept of the presented simulation environment is a result of our experience using the robots in research and education. It has proved that our environment is a very useful and effective tool for fast and safe development and testing of advanced control schemes and task planning algorithms, including force control and visual feedback, as well as learning algorithms, which are more and more required in contemporary service and humanoid robotics. The main part is implemented in MATLAB/Simulink and we have developed models for the robots and sensors used in our laboratory. To integrate the variety of components in an unique framework we have decided to allow the use of different tools for their simulation. So, the simulation environment can be composed of more than one application and the ethernet is used for the communication between them. In this way, our environment is very open and can be very easily extended and adapted to different requirements and applied to any types of robotic manipulators. One of the most important features of our simulation environment is that the testing on real robots is made very easy; the real systems is simply replaced in the simulation loop by proper interface blocks. For that purpose, we have developed interfaces for the robots and sensors. The presented control design environment has proved to be a very useful and effective tool for fast and safe development and testing of advanced control schemes and task planning algorithms, including force control and visual feedback. The software can be very easily extended and adapted to different requirements and applied to any types of robotic manipulators. Last but not least, it is an efficient tool for educational purposes.

7 References

[1] P. I. Corke. A Robotics Toolbox for MATLAB. *IEEE Robotics & Automation Magazine*, 3(1):24 – 32, 1996.

[2] The Mathworks. *SimMechanics, User's Guide*, 2005.

[3] Open Dynamics Engine, http://ode.org/ode.html.

[4] Symblic Dynamics, Inc. *SD/FAST User's Manual*, 1994.

[5] Modelica, http://www.modelica.org.

[6] Newton Game Dynamics, http://physicsengine.com/.

[7] L. Žlajpah T. Petrič. Application of neural networks for dynamic modeling of robotic mechanisms. In *Proceedings of the 7th EUROSIM Congress on Modeling and Simulation*, Prague, Czech Republic, 2010.

[8] RoboWorks™: http://www.newtonium.com/public_html/Products/RoboWorks/RoboWorks.htm.

[9] Blender: http://www.blender.org/.

[10] Stefan Schaal, Peyman Mohajerian, and Auke Ijspeert. Dynamics systems vs. optimal control – a unifying view. *Progress in Brain Research*, 165(6):425–445, 2007.

[11] A.J. Ijspeert, J. Nakanishi, and S. Shaal. Learning Rhythmic Movements by Demonstration using Nonlinear Oscilators. In *Proc. of the 2002 IEEE/RSJ Int. Conf. On Intelligent Robots and Systems*, pages 958 – 963, Lausanne, Suisse, 2002.

[12] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.

[13] K. Doya. Reinforcement learning in continuous time and space. *Neural Computation*, 12(1):219 – 245, 2000.

[14] J. Peters J. Kobler. Policy search for motor primitives in robotics. *Neural Information Processing Systems (NIPS)*, 2008.