# Reinforcement Learning Algorithm Application and Multi-body System Design by Using MapleSim and Modelica

Onder Tutsoy, Martin Brown, Hong Wang and Joe Riel

***Abstract*—** Advanced intelligent systems such as robots must be capable to interact with dynamic environment and adapt their behavior to it efficiently. Currently, modeling humanoid robots with sophisticated learning and cognitive capabilities is one of the most challenging issues in the field of intelligent robotics. Robots must be equipped with the ability to modify and add to its knowledge base information gained from its past failings. This might provide stable robust walking on unseen terrains as well. Moreover, a further critical stage in designing and evaluating such a sophisticated complex system is modeling and simulation. This paper describes preliminary work on designing a simple multi-body system by using MapleSim, which is a tool for multi-body modeling/simulation and reinforcement learning algorithm is applied to this multi-body system in terms of using Modelica models.

## INTRODUCTION

Humanoid robot (HR) design and development with advanced control and learning techniques have gained significant attention over the last decades. In order to enhance the capability of HR for large movements, variable speeds, various constraints and respond to uncertainty in the real world, intelligent control algorithms, such as reinforcement learning (RL) is considered [1]. RL offers an efficient way to move from traditional robotics to true autonomy and versatility by generating adaptable walking gaits based on the changing dynamics in the environment.

Manuscript received May 1, 2012.

Onder Tutsoy is with the Electrical and Electronic Engineering Department, Control Systems Centre, University of Manchester, Manchester, CO M13 9PL UK, phone: (0)161 306 4772; (e-mail: Onder.Tutsoy@ postgrad.manchester.ac.uk).

Martin Brown is with the Electrical and Electronic Engineering Department, Control Systems Centre, University of Manchester, Manchester, CO M13 9PL UK (e-mail: martin.brown@manchester.ac.uk).

Hong Wang is with the Electrical and Electronic Engineering Department, Control Systems Centre, University of Manchester, Manchester, CO M13 9PL UK (e-mail: hong.wang@manchester.ac.uk).

Joe Riel is with the MapleSoft company, Waterloo, Canada, N2V 1K8.

RL is basically an on-line learning technique to map from an input (situation) to output (action) by performing a trial-error process in order to maximize the performance of learning by gaining higher numerical positive reinforcement signal (reward) from the system.

In 1983, Barto and Sutton developed a temporal difference (TD) method and demonstrated its efficiency on pole balancing problem [1]. In 1988, Sutton achieved to formulate discrete TD($\lambda$) learning algorithm and specified its properties [2]. An important work in continuous time and space RL was introduced by Kenji in 2000. He explicitly derived the critic/value function equations for TD(0), TD($\lambda$) and the value gradient based policy learning. He tested these algorithms on a nonlinear pendulum and cart-pole swing-up tasks [3]. In terms of RL humanoid robot applications, an efficient actor-critic based RL algorithm was implemented on a physical robot by Hamid [4]. It controls the 6 joint angles by using the peripheral controllers' evaluation and CMAC neural network. Tedrake (2004) introduced a stochastic RL algorithm and implemented it on a 3D, 6 Degrees of Freedom (DOF) robot [5]. Gen et al. (2008) implemented a continuous RL algorithm which is based on CPG and a policy gradient method to a full body 3D humanoid [6].

Designing a bipedal robot, which is the most sophisticated type of legged robot, is a formidable challenge for the researchers because of the complexity of the mechanism and difficulties in construction and modeling. As deriving dynamic/kinematic equations of the rigid multi-body systems by hand result in an inefficient representation of the systems, software packages for modeling multi-body systems have been developed. It is important to choose appropriate multi-body modeling software package which derives and simplifies the equations of the system. Moreover, they should be flexible enough to allow researchers for their own learning/control algorithm application to the designed multi-body system.

This paper presents a preliminary work on multi-body modeling/simulation and RL application by using the MapleSim software together with Modelica models. For the knowledge of the authors this is the first application of the RL algorithm to a multi-body system by basically combining two softwares' properties, namely MapleSim and Modelica. In section 1, actor-critic based RL algorithm is reviewed. In section 2, various multi-body modeling simulators are briefly introduced. Moreover, modeling a pendulum with MapleSim

and writing RL codes with Modelica are introduced as well. In section 3, a simulation experiment is performed and simulation results are presented.

## I. Literature Review and Background

In value function (critic) learning, TD learning algorithms and TD error play a central role. This section initially presents the TD learning algorithms and then reviews residual gradient learning equations that will be used in the rest of the paper. Finally, it reviews controller (actor) equations of the RL.

### A. Value Function

Value function is the long term prediction of the discounted future rewards that could be obtained in the future. Basically, TD methods are investigated to estimate the unknown value function [3, 7]. Discounted infinite horizon value function:

$$V(x_k)=\sum_{t=0}^{\infty}\gamma^t r(x_{k+t}) \quad (1)$$

where $r(x_k)=r(x_k,u_k)$ is the immediate reward obtained at each discrete time sample $k$, $x_k$ is the state, $V(x_k)$ is the value function, $\gamma\,(0<\gamma<1)$ is the discount factor, $u_k$ is the torque. The discount factor should be chosen such that (1) converges [3, 8]. If the reward is state based, then $r(x_k)=|x_k|$. Basically, a state based reward provides an instant feedback about learning. Characteristic of the value function changes as the selected rewards carry different properties of the learning. However, exact value function representation generally can only be employed to small scale problems. Thus, exact representation is not feasible anymore for a system with large number of states [9]. Furthermore, value function and controller should be represented approximately. A parametric value function approximation scheme which is linear in its parameters is:

$$\hat{V}(x_k,w_k)=\phi(x_k)^T w_k \quad (2)$$

where $\phi(x_k)$ is the basis function vector and $w_k$ is the parameter vector. The next section introduces residual gradient learning which is a type of TD learning algorithm.

### B. Residual Gradient Learning

It is noticed that TD learning algorithms can diverge for more general function approximation schemes. The residual gradient algorithm was developed to eliminate the divergence issue associated with TD(0) learning and this section reviews the residual gradient learning algorithm.

#### 1) Temporal Difference Error

With respect to the equation (1), it can be understood that the value function is basically the sum of the discounted future rewards [2, 7]. Therefore, it represents the long term performance of the learning process.

$$V(x_k)=r(x_k)+\gamma r(x_{k+1})+....+\gamma^{\infty} r(x_{\infty}) \quad (3)$$

Equation (3) can be written in terms of current reward and future value function, which motivates the "differenced error" used in TD learning.

$$V(x_k)=r(x_k)+\gamma V(x_{k+1}) \quad (4)$$

This defines an optimality condition which is satisfied as long as the approximated value function is accurate. Otherwise, a temporary error occurs between the instant reward and the differenced value functions. This is known as TD error occurring due to difference between the instantaneous reward, $r(x_k)$, and estimated reward, $\hat{r}(x_k,w_k)=\hat{V}(x_k,w_k)-\gamma\hat{V}(x_{k+1},w_k)$.

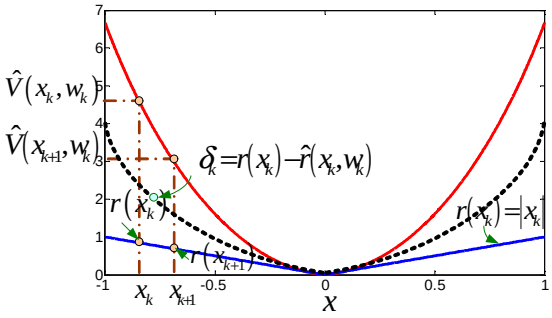


Figure 1: Configuration of the TD error, instant reward and value function. The reward is state based where $x\in(-1,1)$ and the discount factor is 0.85 for the value function.

The temporary error between the instant and the expected reward is given by:

$$\delta_k=r(x_k)-\hat{r}(x_k,w_k) \quad (5)$$

where $\delta_k$ is called as TD error [3]. Substituting the estimated reward in (5) gives:

$$\delta_k=r(x_k)-\left(\phi(x_k)-\gamma\phi(x_{k+1})\right)^T w_k \quad (6)$$

The TD error is equivalent to the error used in many instantaneous learning algorithms in the literature, such as least mean square (LMS) algorithm.

#### 2) Residual Gradient Parameter Update

Baird investigated residual gradient algorithm to overcome the divergence issue associated with the TD(0) learning algorithm. With this learning algorithm, gradient descent is performed on the squared TD error and guarantees the convergence to a local optimum.

$$E_k=\frac{1}{2}|\delta_k|^2=\frac{1}{2}\left|r(x_k)-\left(\hat{V}(x_k,w_k)-\gamma\hat{V}(x_{k+1},w_k)\right)\right|^2 \quad (7)$$

The residual gradient algorithm directly minimizes the TD error by updating its parameters based on gradient descent.

$$w_{k+1}=w_k-\eta\frac{dE_k}{dw_k} \quad (8)$$

The residual gradient parameter update rule in terms of the residuals is given by:

$$w_{k+1}=w_k+\eta\delta_k\left[\phi(x_k)-\gamma\phi(x_{k+1})\right] \quad (9)$$

The parameter update direction is the weighted difference between successive basis function vectors. Residual learning

is stable for the learning rates lying in the range of $0 < \eta < 2 / \left\| \phi(x_k) - \gamma \phi(x_{k+1}) \right\|_2^2$. Appropriate learning rate selection plays an important role in the parameter convergence rate analysis.

### C. Controller and Controller Parameter Update

In order to update the parameters of the controller to reach optimal control policy, TD error is used as a reinforcement signal and same noise signal is added to keep the parameter update sign in the correct direction.

$$w_{k+1}^a = w_k^a + \eta^A \delta_k n_k \frac{dA(x_k, w_k^a)}{dw_k^a} \tag{10}$$

And the controller is:

$$u_k = u^{\max} \left( \tanh A(x_k, w_k^a) \right) + \sigma_k n_k \tag{11}$$

where $A(x_k, w_k^a)$ is the function approximation with the actor parameter $w_k^a$ and noise standard deviation is $\sigma_k = \sigma_0 \min\left(1, \max\left(0.08, \left(V_{\max} - \hat{V}(x_k, w_k)\right)/V_{\max}\right)\right)$.

## II. MULTIBODY MODELLING & RL APPLICATION

Generating realistic mathematical models is a laborious but essential process for multi-body system design and analysis. Developed models are also crucial part of control and learning algorithms design. In this part of the paper, initially existing multi-body modeling softwares are reviewed, then modeling with MapleSim and writing Modelica custom component model are presented.

### A. Existing Simulators for Multi-body Modeling

Over the last two decades a number of software packages are launched for multi-body modeling and simulation. Each of them has advanced or weak properties in terms of modeling, simulation, controller design and user's own algorithm application to multi-body systems.

OpenHRP3 (Open Architecture Human-centered Robotics Platform version 3) is a software specifically developed for robot simulation. It contains a number of software components and calculation libraries that are essential for robotic applications. A library consisting of position, torque, vision and inclination sensors plays an important role on designing controllers for robots. Moreover, this software includes useful tools for walking and balancing such as collision detection. 3D visualization interface allows users to see the model and behavior of the model during the simulation. Although this software is free and open software, it can be directly only used for HRP robots. It is not feasible to apply it to other robots as it does not allow the user to modify or customize major features of OpenHRP. In addition available documentation is limited and technical support is not provided [10].

Honda, Sony and Fajitsu have been designed more simulators for the humanoid robots Asimo, Qrio and Hopa1. Commercial software packages, for instance Webots and RoboWorks, are efficient tools for multi-body modeling and control. However, they do not have advanced libraries for actuators, sensors, ground reaction forces and impacts. To eliminate these disadvantages Reichenbach has initiated a new dynamic simulator [10, 11]. It has advanced sensor options like collision detection sensor, virtual zero moment point (ZMP) sensor, pressure sensor, laser sensor, audio sensor. Generated robot model is now tested and compared with the real robot. Hence, this software is still under development.

A further software called Robotran which is a symbolic multi-body modeling and simulation package designed at the University Catholique de Louvain in Belgium. It has three major features: graphical user interface, symbolic equation generator and interface between Robotran and Matlab. It generates models in symbolic form which leads to reliable numerical accuracy and faster simulation. However, documentation on modeling and software usage is insufficient and online help is limited. It is unable to generate C-code as well. MapleSim, which is newly released symbolic modeling tool for creating and simulating complex physical system, is able to generate stand alone C-code. This leads the controller to be accurately exported to the real system. In addition, it is capable to integrate the multi-body modeling together with actuator models.

### B. MapleSim: Symbolic Multi-body Modeling

MapleSim enables users to create models including components from various engineering fields. Sophisticated symbolic and numeric properties of MapleSim yield accurate mathematical models that depict the behavior of systems. It is also able to simplify the equations of the physical system. MapleSim component library contains more than 400 components which are used to design electrical, hydraulic, 1D and multi-body mechanical models. Majority of the blocks are from Modelica standard library 3.1. Together with the existing MapleSim component library, users can write their own mathematical models by using the custom modeling component property. Hence the users are able to define their own equations and properties of the component, for example parameters and port variables. In terms of using the components from multi-body library, multi-body systems can be built easily.

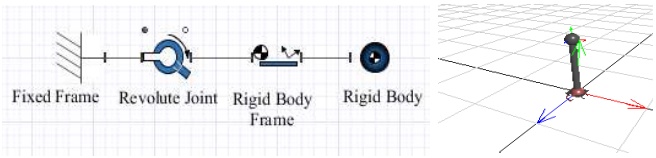


Figure 2: a) Constructed pendulum in model workspace, b) 3D visualization of the pendulum. Sphere represents the

revolute joint and rigid body, the cylinder represents the planar link.

MapleSim also provides 3D visualization of the model in 3D environment. An important useful property of this software is that the users can visualize the simulation results by playing animation which depicts the model movement. MapleSim templates enable users to analyze the created model. The model can be linearized, discretized and controllers can be designed and the response of the systems might be analyzed.

### C. *Model Generation with Modelica Custom Component*

One of the most significant features of MapleSim is that it allows the users to create their own custom component to design various learning/control algorithms. The previously reviewed RL algorithm is applied to a pendulum by using Modelica custom component. For example, the Modelica model for the TD error is:

```
model TDError
  extends Maplesoft.Icons.CustomComponent;
  parameter Real gama = 0.8;// "Discount factor";
  parameter Real failureState = 0.3;// "Failure state";
  Real r[1,1];// "Reward";
  Modelica.Blocks.Interfaces.RealInput V[1,1] annotation(
   Placement(transformation(
    extent = {
     {-110,82},{-90,62}},
    rotation = 0)));
 …
  Modelica.Blocks.Interfaces.RealOutput TDError[1,1]
annotation(
   Placement(transformation(
    extent = {
     {90,-13},{110,7}},
    rotation = 0)));
equation
r[1,1]=(cos(phi)-cos(failureState))/(1-cos(failureState));
TDError = r + gama * V - preV;// "TD error";
end TDError;
```

As can be seen from the table, initially the name of the model is specified and the model parameters are introduced. Then, the inputs and outputs of the TDError model are defined. In the equation section, differential equations, algebraic equations and equations including for/while loops, if /when conditions, other Modelica functions can be also used. For instance, the Modelica code for value function parameter update includes a for loop, when condition together with sample and pre functions.

```
model valueFunctionParamUpdate
  extends Maplesoft.Icons.CustomComponent;
  parameter Real gama = 0.8;// "Discount factor";
  parameter Real nc = 0.6;// "Critic learning rate";
  parameter Real sn = 10;//"Number of basis";
  Boolean clock;
  Modelica.Blocks.Interfaces.RealInput preWc[sn*sn,1]
annotation
  (Placement(transformation(
   extent = {
    {-110,102},{-90,82}},
   rotation = 0)));
 …
 Modelica.Blocks.Interfaces.RealOutput Wc[sn*sn,1] annotation
  (Placement(transformation(
  extent = {
   {90,-43},{110,-63}},
  rotation=0)));
initial equation
Wc=fill(0,sn*sn,1);
preWc=fill(0,sn*sn,1);
equation
  Wc= preWc + nc * (prebPen - gama * bPen)*scalar(TDError);
  clock = sample(0, 0.01);
  when clock then
     for i in 1:sn*sn loop
      preWcUpdate[i,1] = pre(Wc[i,1]);
     end for;
  end when;
end valueFunctionParamUpdate;
```

Here also an initial equation is defined to assign initial values for the current and previous values of the parameter vectors. To use the previous parameter vector at the current time, sample and pre functions are used. Sample function holds the previous value for 0.01s for this example. The figure below shows the RL algorithm applied to a pendulum in MapleSim in terms of using the Modelica custom component and Maple blocks.

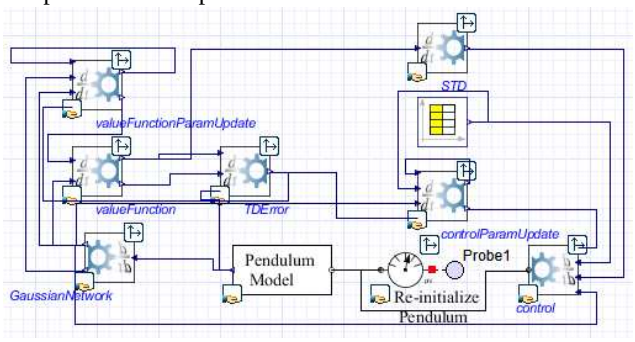


Figure 3: RL applied to a pendulum in MapleSim, where STD represents the standard deviation of the noise and re-initialize pendulum is basically an angle sensor written in Modelica to re-initialise the pendulum in the case of failure.

The block between the STD and controlParamUpdate is a time lookup table where data caries the properties of random Gaussian noise. In order to show the learning and convergence results of the RL algorithm applied to designed pendulum model in MapleSim, a simulation experiment is performed.

## III. Simulation Experiment

This section initially introduces the simulation parameters and then analyzes the simulation results.

### A. *Simulation Experiment*

Equations of the pendulum can be obtained by using MapleSim equations template which is located in the attachment palette. The differential equation representing the dynamics of the pendulum is $ml^2\ddot{x} = mgl\sin(x) + u$ where

$m$ is the mass, $l$ is the length of the pendulum having both unity value for the simulation. The simulation length is 120 seconds, the learning rate for value function parameter update is $\eta = 0.6$, the learning rate for the controller parameter update is $\eta^A = 0.9$, the discount factor is $\gamma = 0.8$, maximum torque is $u^{\max} = 5$ and noise standard deviation constant is $\sigma_0 = 0.53$. The noise is randomly distributed zero mean Gaussian. It is accepted that the pendulum fails if its position is $|x| \geq 0.3$ and the reward is given by $r_k = (\cos(x) - \cos(0.3))/(1 - \cos(0.3))$. The pendulum is re-initialized with $[x \quad \dot{x}] = [0.011 \quad 0.001]$ in the case of failure. Basis function is $b_k = a_k(\mathrm{x}) / \sum_{l=1}^{K} a_l(\mathrm{x})$ where $a_k(\mathrm{x}) = e^{-\left\| s_k^T (\mathrm{x} - c_k) \right\|^2}$. Here, $s_k$ and $c_k$ are the size and centre of the $k^{th}$ basis function. The size of the basis function is 10*10 for this simulation experiment. All the initial parameter vectors for the value function and the controller are zero for this simulation experiment.

### B. States of the Pendulum

The RL algorithm learns to generate appropriate control action from failures. Therefore, it is expected that the pendulum will fail for a number of times until it achieves to generate correct control action to stabilize the pendulum. Then, it should converge to the states which yield the maximum instant reward and also maximum value function.

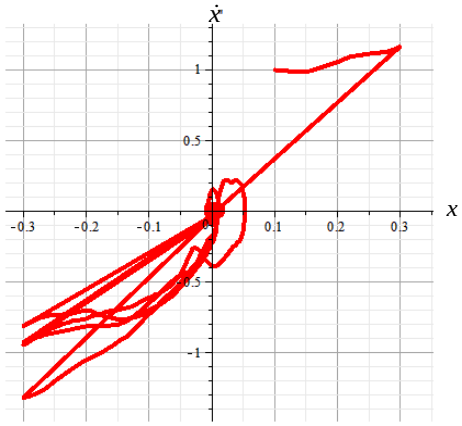


Figure 4: States of the pendulum during the whole simulation.

As can be seen from the Figure 4, the angle and velocity states of the inverted pendulum converge to states around zero. Initially, the learning algorithm is unable to keep the pendulum upright since the entire assigned initial values for the value function and controller are zero. Basically, the number of the failures depends on the initial states, initial parameter vectors, amount of the exploration noise and selected learning parameters. For example, if the selected discount factor $\gamma$ is around 0.9, then the number of the failures is significantly increasing. More importantly, the simulation results are becoming inconsistent between each simulation run. To analyze the possible reason of this situation, a test problem, where the results will be published soon, is performed. This work shows that the basis function space is almost singular for the pendulum around 0.9 discount factor. On the other hand, with appropriate learning parameters, as new feedbacks through the instant reward are obtained about the outcome of the learning, the RL algorithm adjusts its weights/parameters. The amount of the update depends on the TD error and learning rates. After a number of trials, the RL algorithm learns the optimal control action which keeps the pendulum upright.

### C. Value Function

Value function is the prediction part of the RL algorithm. The target is to maximize the value function in order to reach optimal learning/control outcomes.

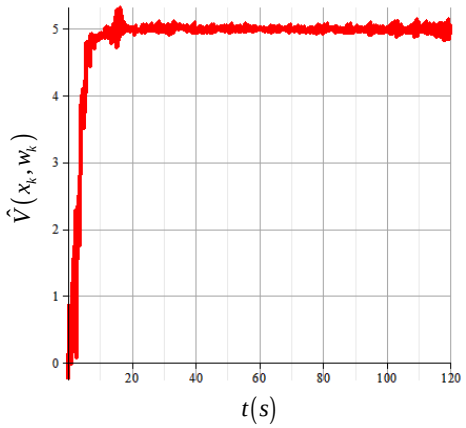


Figure 5: Estimated value function.

The Figure 5 depicts the learning process of the value function. The maximized optimal value function must have value of 5 since the discount factor is 0.8. Initially, its value is around zero but as the learning occurs it reaches its maximum. If there is a bias in the value function learning, then modelling error will be unavoidable and the generated control action will be biased as well.

### D. Control Signal

The main target of the RL algorithm is to learn to generate optimal control signal that stabilizes the systems.

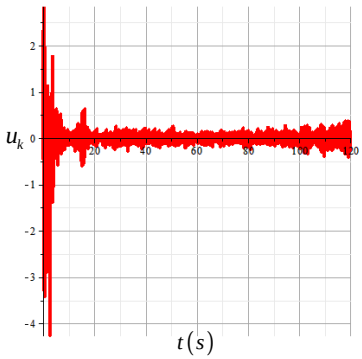


Figure 6: Learned control action.

As shown by the Figure 6, initially the RL algorithm generates large control signals which cause failures. However, from this failure experience, it is able to learn to generate suitable amount of control signal to keep the pendulum upright.

### *E. TD Error*

Parameter of the value function and the controller are updated based on the amount of the TD error.

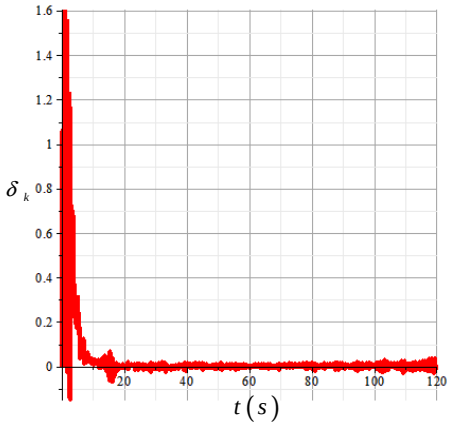


Figure 7: TD error.

As can be seen from the Figure 7, initially TD error is large; hence amount of the parameter update is large as well. As the optimal learning occurs, the value function is maximized and the maximum reward is obtained. Therefore, TD error is minimum.

### *F. Noise Standard Deviation*

The noise is basically used for the exploration purpose. However, after optimal learning little amount of noise is still added to investigate the robustness of the RL algorithm.

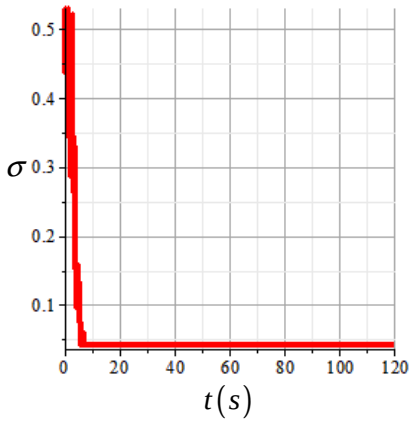


Figure 8: Noise standard deviation.

At the beginning of the learning, maximum amount of noise is added to attain the optimal learning outcomes. After learning, the noise plays disturbance role to examine the efficiency of the learning.

## IV. Conclusion

Results show that the RL algorithm is able to learn optimal value function and control action. It also emphasizes that the RL algorithm is capable to manage the external disturbances after learning. The next target of the authors is to apply this algorithm to a simplified HR in terms of using MapleSim and Modelica features.

## References


1. A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *Artificial neural networks*, vol. 13, pp. 81-89, 1990.
2. R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol.3, pp. 9-44, August 1988.
3. K. Doya, "Reinforcement Learning in Continuous Time and Space," *Neural Computation*, vol. 12, pp. 219-245, 2000.
4. H. Benbrahim, "Biped dynamic walking using reinforcement learning," *Robotics and Autonomous Systems*, vol. 22, pp. 283-302, 1996.
5. T. Russ, "*Stochastic policy gradient reinforcement learning on a simple 3D biped*," presented at the 2004 Intelligent Robots and Systems, 2004, Japan.
6. G. Endo, J. Morimoto, T. Matsubara, J. Nakanishi and G. Cheng, "Learning CPG-based biped locomotion with a policy gradient method: application to a humanoid robot," *The International Journal of Robotics*, vol. 27, pp.213-228, 2008.
7. R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, Cambridge, 1998, pp.20- 322.
8. M. Geist, and O. Pietquin, "Parametric value function approximation: A unified view," presented at the 2011 Adaptive Dynamic Programming and Reinforcement Learning (ADPRL), 2011 IEEE Symposium, France.
9. L. Busoniu, R, Babuska, B. D. Schutter and D. Ernst, "Approximate reinforcement learning: An overview," presented at the 2011 Adaptive Dynamic Programming and Reinforcement Learning (ADPRL),France.
10. G. A. Medrano-Cerda, H. Dallali, M. Brown, N. G. TSagarakis and D.G. Caldwell "Modelling and simulation of the locomotion of humanoid robots," presented at the 2010 UK Automatic Control Conference (UKACC), Conventry.
11. T. Reichenbach, "A dynamic simulator for humanoid robots," *Artificial life and robotics*, vol. 13, pp. 561-565, 2009.