Neural Networks PhD Course Activity Report

Mario Bambagini, Francesco Prosperi
{name.surname@sssup.it}
Scuola Superiore Sant'Anna, Pisa

July 18, 2011

1 Introduction

The goal of this project is to develop a Ball and Plate balancing control system.

The ball and plate system consists of a ball on a plate, capable of being tilted along each of two horizontal axis of the plate itself. The plate, initially horizontal, is tilted aiming to control the position of the ball, so the ball is indirectly controlled.

We designed the balancing control system using a neural network based on the reinforcement-learning model, that is the problem faced by an agent that learns behavior through trial-and-error interactions with a dynamic environment. Such an approach, not making use of a formalized system description as digital control techniques do, programs agents by reward and punishment without needing to specify how such a *task* is to be achieved.

The project consists in two programs, written in C, one for the learning phase, (training the neural networks) and the other for the graphical representation (using the neural networks) using the *OpenGL* library for a 3D simulation.

The rest of the report is organized as follows: Section 2 introduces a linearized ball and plate model, core of the implemented simulator. Section 3 presents the system architecture together with the neural network tuning. Section 4 proves the effectiveness of our approach, showing the results of the neural network training. Section 5 contains a description of the developed programs and how to use them. Section 6 ends the report with the concluding remarks.

2 Ball and plate physical model

In order to present the mathematical model of the ball and plate system, Figure 1 shows the xz view of the coordinate system, introducing the needed parameters to define.
The yz view is identical to the xz view, considering $\hat{y}$, y, $\dot{y}$, $\ddot{y}$, θ_y, ω_y, $\dot{\omega}_y$ in place of $\hat{x}$, x, $\dot{x}$, $\ddot{x}$, θ_x, ω_x and $\dot{\omega}_x$, respectively. Hence, the yz view is omitted.

The fixed joint O, situated in the middle of the plate, allows the surface to rotate around it within a specific angle range. Moreover, the actuator and the plate

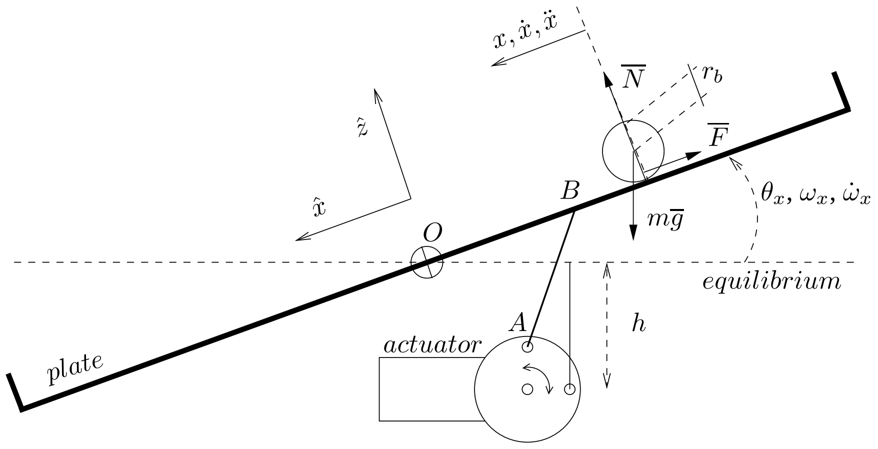


Figure 1: xz view of the ball and plate physical model.

are not joint directly as an arm is inserted between them. The junction is length h, connecting the actuation point A and the actuated point B.

In order to simplify the computation, both the sliding and rolling frictions are considered negligible.

The complete set of equations is continuos, non-linear and couples the two modes of motion. After a discretization and linearization process about the operating point, which is the equilibrium configuration ($x = y = \theta_x = \theta_y = 0$), the equations of motion for the ball, along the x and z axis, are the following:

$$\begin{aligned}
\ddot{x}[k] &= \frac{5}{7}\, g\, sin(\theta_x[k]) - \left(r_b + \frac{5}{7}\, h\right) \dot{\omega}_x[k] \\
m\ddot{z}[k] &= N - mg\, cos(\theta_x[k]) = 0 \\
\dot{x}[k] &= \ddot{x}[k]\Delta t + \dot{x}[k-1] \\
x[k] &= \dot{x}[k]\Delta t + x[k-1] \\
\omega[k] &= \frac{\theta[k] - \theta[k-1]}{\Delta t} \\
\dot{\omega}[k] &= \frac{\omega[k] - \omega[k-1]}{\Delta t}
\end{aligned}$$

Where Δt is the control period (the time elapsed since the previous control event until the actual). The first two equations are computed according to the Newton's laws and the last four by the classical kinematic formulas, assuming a motion uniformly accelerated during each Δt.

Note that linearizing decouples the two modes of motion. Thus, the system can be treated as two different systems operating simultaneously. Hence, similar but independent controllers can be used for controlling each coordinate of the ball motion. This lets us to design only one controller as the other one is exactly the same. As the desired movements of the ball are only on the xy plane, the vertical acceleration (along z) is set equal to 0.

3 Reinforcement-learning model

Depending on what information is provided to the learning agent, learning methods can be grouped into three categories: supervised learning, unsupervised learning and reinforcement learning. Reinforcement learning is a compromise between supervised learning and unsupervised learning. It learns to map situations to actions via online and trial-and-error exploration. Instead of providing desired input/output pairs for training, or no information other than the data, the environment provides a feedback to the learning agent on how good the agent's outputs are after the learning agent interacts with the environment.

3.1 Architecture

Our neural network architecture is based both on the box model proposed by [1] and the ASE-ACE model proposed by [2], as shown in Figure 2.

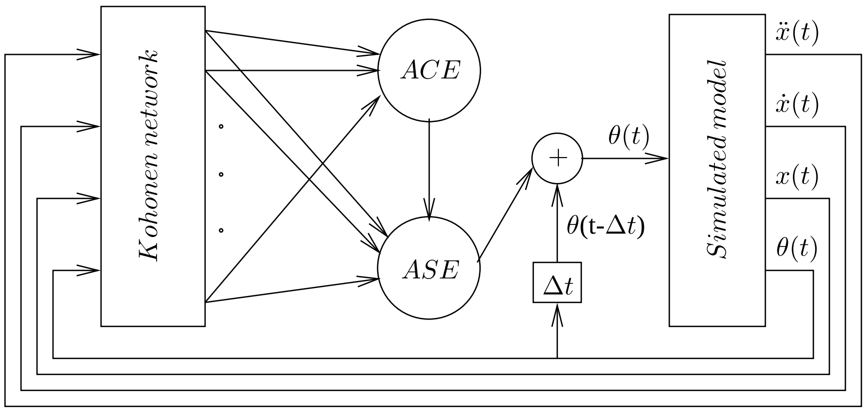


Figure 2: Neural network architecture.

The *Simulated model* block reproduces the physics of the ball and plate implementing the equations reported in Section 2. According to its input, that is the updated plate angle, such a module computes all the other system parameters which compose the system state S as

$$S = [\quad x \quad \dot{x} \quad \ddot{x} \quad \theta_x \quad]^T.$$

The state S is quantized through a *Self Organizing Map* (SOM), also known as *Kohonen network*. The training parameters for such a network are reported in Table 1. Note that a situation of R_{min} equal to 1 implies that only the winning neuron is updated.

n	number of nodes	700
n_r	number of rows	35
n_c	number of columns	20
α_{max}	initial learning rate	0.4
α_{min}	final learning rate	0.0
R_{max}	initial radius	10
R_{min}	final radius	1
T_{max}	max training duration (iterations)	10^8

Table 1: Training parameters for the Kohonen network.

n	700
α	10.0
λ	0.75

Table 2: ASE parameters.

The *Associative Search Element* (ASE) associates a correct action in response to a quantized input. Moreover, the *Adaptive Critic Element* (ACE) produces a secondary reinforcement in order to let the ASE learn also when failures occur less frequently.

The ASE's output consists of the plate angle variation, which is added to $\theta(t - \Delta t)$, to obtain the new plate angle.

The parameters of the *Associative Search Element* (ASE) and *Adaptive Critic Element* (ACE) are reported respectively in Table 2 and Table 3. In order to understand the meaning of such parameters, please refer to the handout slides available at [3].

Initially, the failure condition was detected whenever the ball position hit the plate borders. Although this solution is simple and conceptually correct, unfortunately it presents an unpleasant drawback, that is a slow and ineffective learning. This effect is due to the fact that the simulations, in which the ball rolls from a border to another and back, are considered under control. To avoid this kind of situation, the chosen approach consists in reducing progressively the valid area during the simulation: at the beginning such an area covers the whole plate; after a

n	700
λ	0.5
β	0.4
γ	1.0

Table 3: ACE parameters.

while, it shrinks proportionally down to the lower bound. Each simulation not able to carry the ball near the plate center in a limited time generates a failure.

4 Results

The possible plate angles are bounded to be within the range [-10°, +10°] with valid variations of $\pm 1°$: such constraints are due to the fact that a wider range has a negative impact on the ball acceleration, making the control more difficult. Moreover, varying the plate angle beyond $\pm 1°$ implies an unrefined tuning and a higher $\dot{\omega}$.

The control period for the simulation, Δt, is set to 20ms, being a good trade-off between computational load and the system dynamics. Longer intervals imply a harder learning because the ball shift widens causing angle saturations and hence, a greater ball speed.

The plate is square and its side length is 0.40 meters.

The learning phase, reported in Figure 3, took more than 30 hours executing about 27000 simulations. Each simulation starts placing the ball in a random position on the plate, initially horizontal, and executes at most 10^6 learning steps.

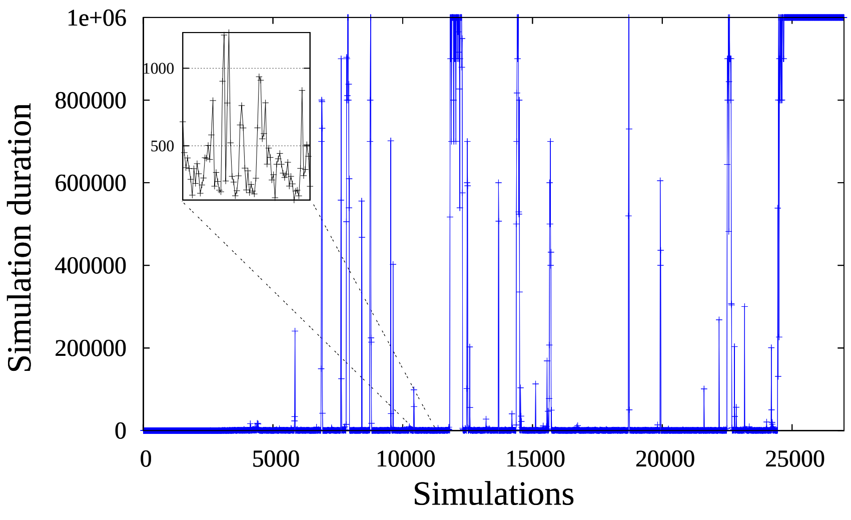


Figure 3: Learning progress.

Up to about 24000 simulations, failures occured few steps after the beginning of the training procedure, except for some sporadic cases in which the ball was well controlled. From about 24000 simulations till the training session end (about 27000), the networks were able to control the ball, keeping always it within the red boundaries, until the simulation steps limit (10^6).

The networks always worked well during the next tests, being able to avoid failures also when some moderate perturbations were applied through the graphical program.

5 Implementation

The project is divided in two different programs written in C: one for the neural network learning and the other for the graphical usage. This approach helps to speed up the learning phase as the viewer executes a learning update frequency equal to the Frames Per Second (FPS) (more or less 30). On the other hand, the dedicated program runs at the effective CPU speed, executing more than thousands of steps per second.

The learning program is command-line and its synopsis is:

```
trainer [-k file] [-c file] [-s file] [-o file] [-r]
```

where:

- -*k* specifies the file describing the Kohonen network that is deserialized and serialized at the beginning and at the end of the learning process, respectively. If the file does not exist or the option is not specified then the network is randomly initialized. Moreover, the neural network is not serialized if the option is not expressed;
- -*c* is similar to -*k* but concerns the ACE network;
- -*s* is similar to -*k* but concerns the ASE network;
- -*o* writes in the specified file the learning advances in a Comma Separated Values (CSV) format;
- -*r* starts the learning. Without typing it, the program prints a brief description in order to explain the option meanings and exits.

The program execution is ended by sending it a SIGKILL signal (CTRL+C), causing the neural networks serializations, if possible.

The visualization program uses the neural networks (only Kohonen and ASE) to show the control usage. Its synopsis is:

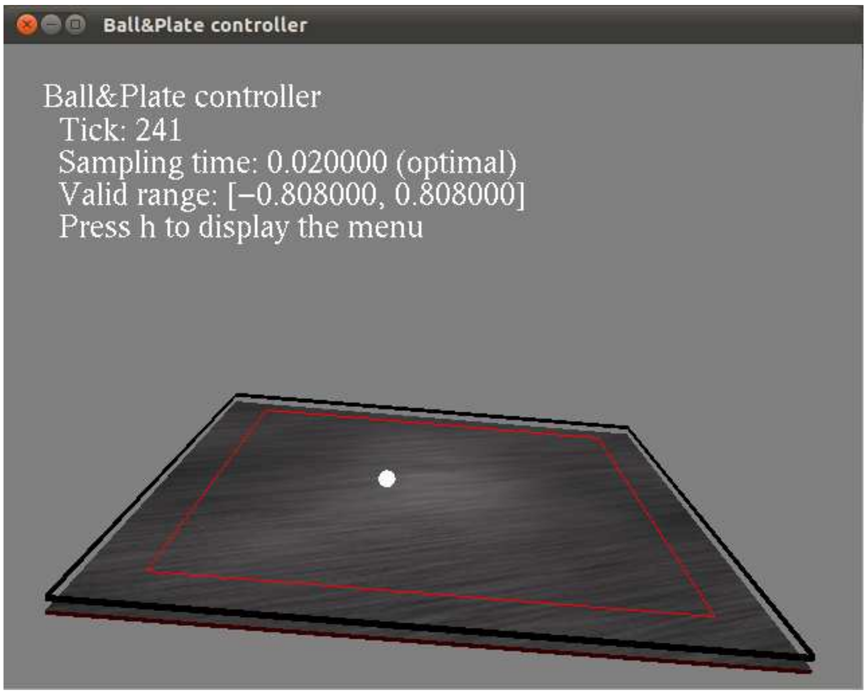
```
ball_and_plate [-k file] [-s file] [-r]
```

where the options have the same meanings.

The graphical program, as shown in Figure 4, presents a minimalistic layout reporting several basic information and a representation of the ball and plate system, drawn using the OpenGL library. The surface within the red boundaries represents the area in which the controller has to keep the ball to avoid a failure.

At the top, the program shows the tick number (how many control steps have been executed), the sampling time (the time elapsed since the last control event) and the side length of the red boudaries (as percentage of the whole plate and according to the plate center).

Pressing either *h* or *H* key, the program displays a menu to explain the available commands. The keys *w*, *s*, *a* and *d* allow to rotate the point of view along the X



Ball&Plate controller
Tick: 241
Sampling time: 0.020000 (optimal)
Valid range: [−0.808000, 0.808000]
Press h to display the menu

Figure 4: Program screenshot.

and Z axis. To perturbate the system, three features are available. The *b* key blocks the plate until another press of the same key. The *directional arrows* let the user to apply an external force to the ball (not to the plate). The last feature, enabled by the *c* key, switches between the optimal and real sampling: with the optimal approach the controller considers a deterministic sampling every Δt (the same used for the learning) while, with real sampling, the controller works with the actual time elapsed since the last screen refresh. The last configuration depends on the graphical capability of the host machine and the real sampling time usually lasts longer than the optimal one causing a performance degradation so significant to make the controller ineffective. Finally, the *p* and *r* keys pause and reset the simulation, respectively.

In order to simplify the programs invocations, two scripts are provided: *trainer.sh* and *ball_and_plate.sh*. Their task is to invoke the relative programs passing them the pre-trained neural networks files.

6 Conclusions

This report presented a solution for the well-known Ball and Plate system based on a neural controller.

Focusing on the controller, a Kohonen network encoded the system state to the ASE, which computed the new plate angle. Moreover, the ACE produced a second reinforcement for the ASE.

The results showed the convergence of the learning phase and the effectiveness of the performed control.

A How to use our tool

This section briefly describes the procedure in order to run the trainer and ball_and_plate programs both under UNIX and CYGWIN platforms. In the following, for each platform, a list of the needed packages and console commands is presented.

A.1 GNU/Linux

Packages

- freeglut3
- freeglut3-dev

Console commands

```
$ cd Ball_and_Plate_controller
$ make
$ ./trainer -k kohonen.nn -s ase.nn -c ace.nn -r
or
$ ./trainer.sh
$ ./ball_and_plate -k kohonen.nn -s ase.nn -r
or
$ ./ball_and_plate.sh
```

A.2 CYGWIN

Packages

- gcc
- make
- opengl

Console commands

```
$ cd Ball_and_Plate_controller
$ make cygwin
$ ./trainer -k kohonen.nn -s ase.nn -c ace.nn -r
or
$ ./trainer.sh
$ ./ball_and_plate -k kohonen.nn -s ase.nn -r
or
$ ./ball_and_plate.sh
```

References

[1] D. Michie and R. A. Chambers, “Boxes: an experiment in adaptive control,” in *Machine Intelligence*, 1968.

[2] A. G. Barto, R. S. Sutton, and C. W. Anderson, *Neuronlike adaptive elements that can solve difficult learning control problems*. Piscataway, NJ, USA: IEEE Press, 1990, pp. 81–93. [Online]. Available: http://portal.acm.org/citation.cfm?id=104134.104143

[3] G. Buttazzo. (2008) Introduction to neural networks. [Online]. Available: http://retis.sssup.it/ giorgio/slides/neural-nets/nn.html