# CSE 6369 - *Reinforcement Learning*

Homework 1- Spring 2020

Code Documentation

## Simulation Code

The simulation code provided is C code that uses X11 to display a graphical simulation of the cart-pole system. To run it you need to have an X server running on your system. The code is provided for two systems: i) OSX (tested on HIgh Sierra), and ii) 64 bit Linux (tested on omega.uta.edu). To run the code you need to have an X-server running (on OSX this means starting it from an X11 terminal, on Linux this is a given, and if you want to remotely run it remotely on Omega from your Windows computer you need to connect to it from within an X-server (such as Cygwin) using *ssh -X "username"@omega.uta.edu* (to establish an X connection) and should then be able to compile and run the code from that terminal window.
To build the code, the system uses the *imake* system.
To start you have to download the appropriate code archive and uncompress it. This directory contains the following files:

**Imakefile** This file is used to create a machine specific Makefile by typing *xmkmf*.

**Pole_sim.h** This file contains the definitions for the simulation, including the nominal force multipliers (F_X, F_Y, F_Z), the number of dimensions used in the simulation (P_DOF), and the data structure containing the position, velocity, and accelerations of the cart-pole system.

**Pole_sim.c** This file contains the dynamic equations of the cart-pole system.

**Xpole.h** This file contains the function definitions for the graphics code for the simulation.

**Xpole.c** This file contains the graphics code for the simulation.

**Global.c** This file contains miscellaneous global variables.

**misc.h** This file contains miscellaneous macros.

**Matrix_fncs.h** This file contains the function definitions for the basic matrix operations.

**Matrix_fncs.c** This file contains basic matrix operations.

**lib** This directory contains a graphics library you need.

**include** This directory contains the header files needed to build the graphics.

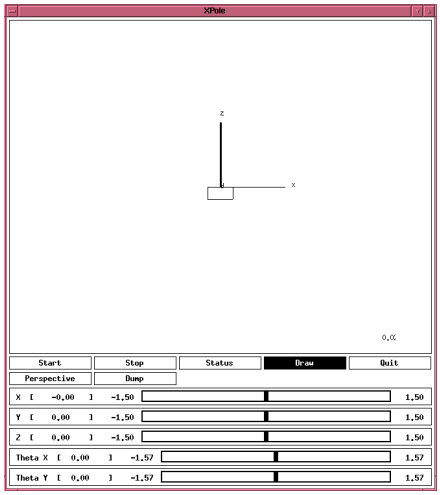
**Reinf_learn.h** This file contains the function definitions for the learning routines

**Reinf_learn.c** This is the file you have to edit in order to implement the learning algorithms.

Of these files, Reinf_learn.c is the only one you will need to add code to. To move the simulation from 1D to 2D you will need to change the P_DOF definition in the Pole_sim.h file from 1 to 2.

## The Simulation Environment

To generate the simulation program you first have to create a Makefile by typing *xmkmf*. Once the Makefile has been created, you have to type *make*. This will create the graphical simulator and learning interface *PoleBalance*. If you change the dimensionality you should type *make clean* before re-building the code to make sure everything is changed to 2D.
This interface should look as follows:




The graphical interface displays the cart-pole system in the top window and a set of sliders at the bottom which show the configuration of the system. The *Start* button will start the simulation and call your routine 50 times per second to determine which of the two actions to take. During the simulation the system will put the cart in the center location with the pole in the straight upward direction and then runs a trial until either the pole falls or the cart reaches the end of its track (1.5m in each direction - near the sides of the display window). Once this happens, the simulator will output the balancing time and the percentage of random actions that were taken, and re-start the cart in the center position for the next trial. The *Stop* button will stop the simulation.
Pressing the *Draw* button during simulation button will cycle through 3 different display modes. In the first mode the simulator will re-display the cart-pole system every few control steps. The second mode will re-display once every second, and the third mode will display it once every 20 (simulated) seconds. The later modes therefore use less CPU time and can run the simulation much faster. (While the simulation is running, the program outputs the achieved balancing time and the percentage of random actions taken in the terminal window.)

Pressing the *Dump* button will write the file *learning.dat* which contains a series of tuples indicating the trial number and corresponding balancing time for all trials run so far. This file can be used to draw a learning curve which shows how the balancing performance improves over time.

## The Learning Code Interface:

The Reinf_learn.c file contains the main function for your learning code in the *pole_learn()* function. All of your learning code should go into this function which receives the information about the cart-pole system (and whether the trial has just been reset) and from which you return the force vector to apply to the cart as well as information regarding how many exploration actions have been taken so far and whether a new trial should be started. The function has the following structure:

```
void pole_learn(pole, reset, force, fail, explore)
  Polesys *pole;
  int reset;
  double force[3];
   int *fail;
   int *explore;
 {
  ...
 }
```

The data structure definition for the cart-pole system is in *Pole_sim.h* and contains pole angles, angular velocities, and angular accelerations, the cart location, velocities, and accelerations, as well as the trial simulation time.
The force vector stores the forces in X, Y, and Z direction that should be applied to the cart in the next time step.

## Example Learning Executable

The tar archive also contains a compiled sample executable *XpoleTD_1D* which is a 1D Actor-Critic learner. You can use this as a reference to see if your learning behaves similarly.