EE5112 Human Robot Interaction: Mini Lab

WANG Yexiang, e1373807

Task1

Dynamic Model of a Robotic Arm

The general dynamic model for an n-degree-of-freedom (DOF) robotic arm is given by the Euler-Lagrange equation:

$$\tau = D(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) + F(\dot{q})$$

Inertia Matrix D(q)

The inertia matrix D(q) describes how the mass of the robotic arm is distributed across its joints. It depends on the configuration (joint angles) q, and it is symmetric and positive definite. Each element Dij(q) represents how much the motion of jointi is affected by the acceleration of joint j. Larger values mean more inertia (mass or rotational inertia) is associated with that movement.
Its size: n×n, where n is the number of DOF. It reflects the mass and inertia properties of the links.

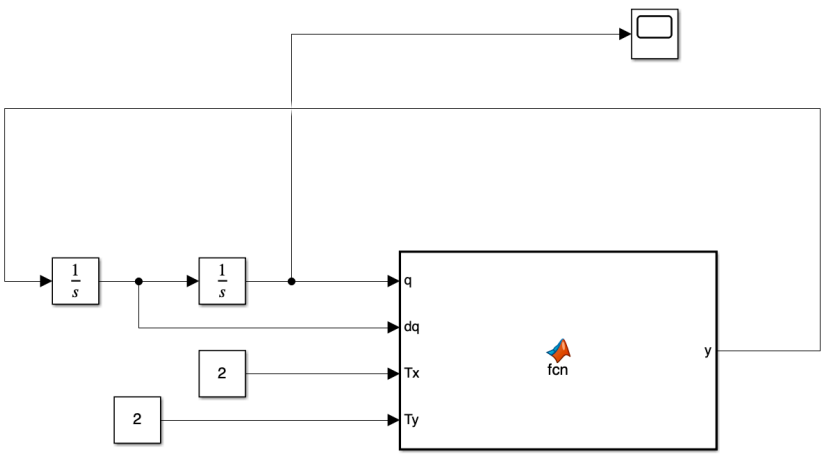
Coriolis and Centrifugal Matrix C(q, q')

The Coriolis and centrifugal matrix C(q, q') accounts for the effects that arise due to the motion of the robotic arm. The centrifugal force depends on the square of the velocity, while the Coriolis force arises due to the interaction of velocities of different joints.

Gravitational Force Vector G(q)

The gravitational force vector G(q) describes the torque needed at each joint to counteract the gravitational pull on the robotic arm, depending on its configuration q.

Simulink Block: Based on the data provided this time, I have built the following Simulink block diagram:

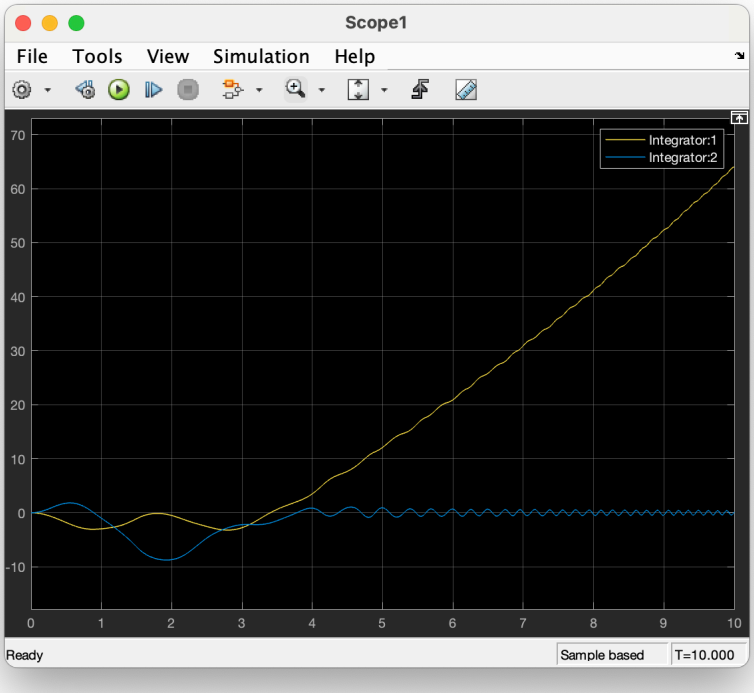


The function block is for the dynamic modeling of the robotic arm, and its code is as follows:

as2 ▸ MATLAB Function

```
function y=fcn(q,dq,Tx,Ty)
    m1=2;m2=4;l1=0.6;l2=0.4;
    lc1=l1/2;lc2=l2/2;
    I1=m1*l1^2/3;
    I2=m2*l2^2/3;
    g=9.8;

    p1=m1*lc1^2+m2*l1^2+I1;
    p2=m2*lc2^2+I2;
    p3=m2*l1*lc2;
    p4=m1*lc1+m2*l1;
    p5=m2*lc2;

    D=[[p1+p2+2*p3*cos(q(2)),p2+p3*cos(q(2))];
        [p2+p3*cos(q(2)),p2]];
    C=[[-p3*dq(2)*sin(q(2)),-p3*(dq(1)+dq(2))*sin(q(2))];
        [p3*dq(1)*sin(q(2)),0]];
    G=[p4*g*cos(q(1))+p5*g*cos(q(1)+q(2));p5*g*cos(q(1)+q(2))];

    u=D^(-1)*([Tx;Ty]-C*q-G);

y = u;
```

Based on the data provided this time, the input (torque) is set to 2, meaning a constant force of 2 is applied to both joints. After running the simulation for some time, the waveform of q is observed using the scope block, as shown in the following figure:



From the figure, it can be seen that the angle of joint 1 continues to increase, while the angle of joint 2 (relative to joint 1) oscillates and becomes faster over time. This behavior is intuitive, as applying a constant force causes the angle to keep increasing.

Task2

A PID controller is a feedback loop control system that continuously calculates the error between a desired setpoint and a measured process variable. The controller attempts to minimize this error by adjusting control inputs (e.g., motor speed, valve position) using three components:

Proportional (P): This component generates an output that is directly proportional to the error. A larger error leads to a larger correction. However, using only P control can lead to steady-state errors.

Integral (I): This component sums up past errors to eliminate steady-state error, allowing the system to reach the setpoint more accurately. It corrects small persistent errors that P control alone cannot fix.

Derivative (D): This component predicts future error by looking at the rate of change of the error. It helps stabilize the system and reduce overshoot by slowing down the response when the error changes rapidly.

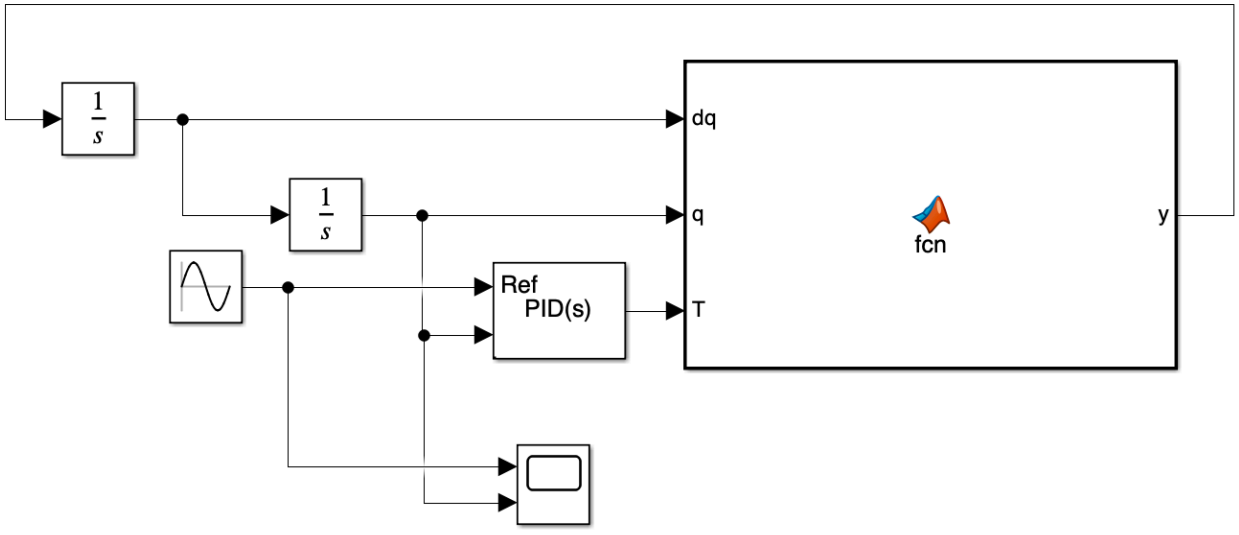
The combination of these three terms is written as:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}$$

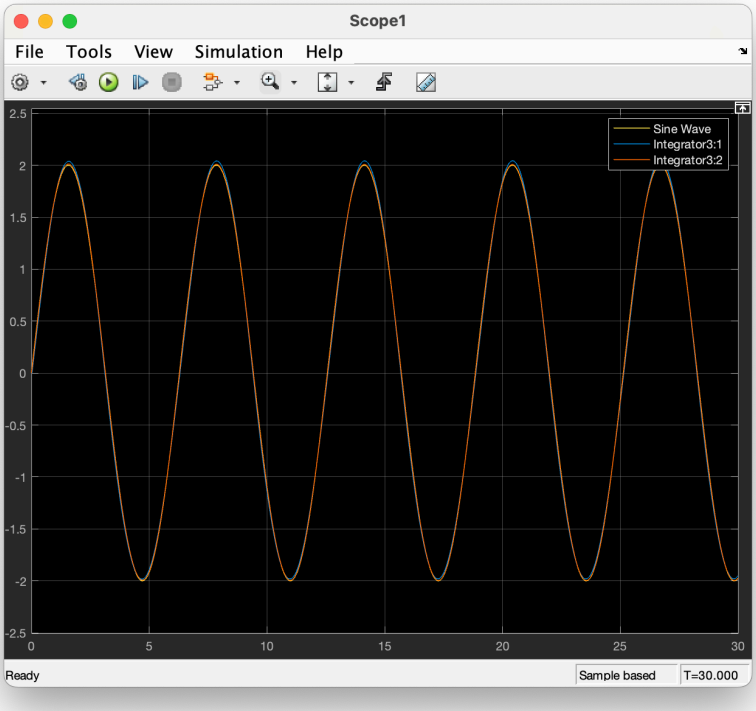
where:

1. u(t) is the control output.
2. e(t) is the error (difference between the setpoint and process variable).
3. Kp, Ki, and Kd are the proportional, integral, and derivative gains, respectively.

The PID control function is implemented in Simulink using the PID block, and its block diagram is shown as follows:



The parameters are adjusted to Kp=500, Ki=80, Kd=200, and the final result is as follows:



It can be seen that the final output nearly overlaps with the target function, indicating that the control performance is very good.

Overview of PID Function:

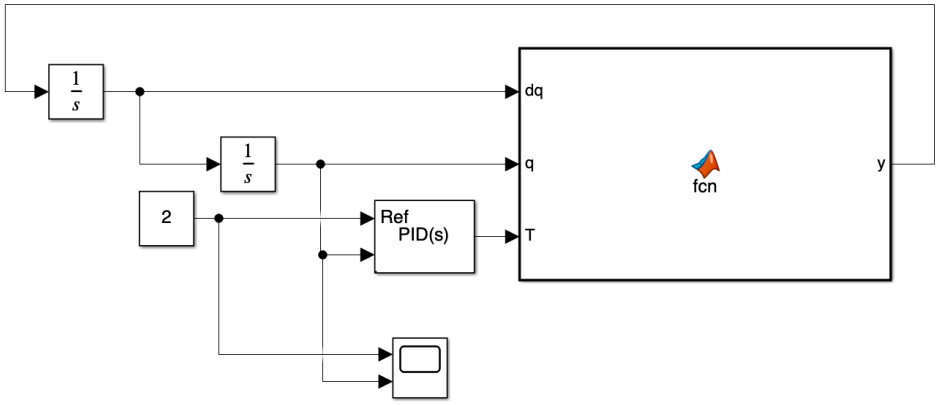
1. P (Proportional) generates response speed and force. If too small, the response is slow; if too large, it can cause oscillations. It forms the basis for I and D.

2. I (Integral) eliminates steady-state errors and improves accuracy in the presence of system deviations and external forces. It also increases response speed but can cause overshoot if too large, leading to oscillations.

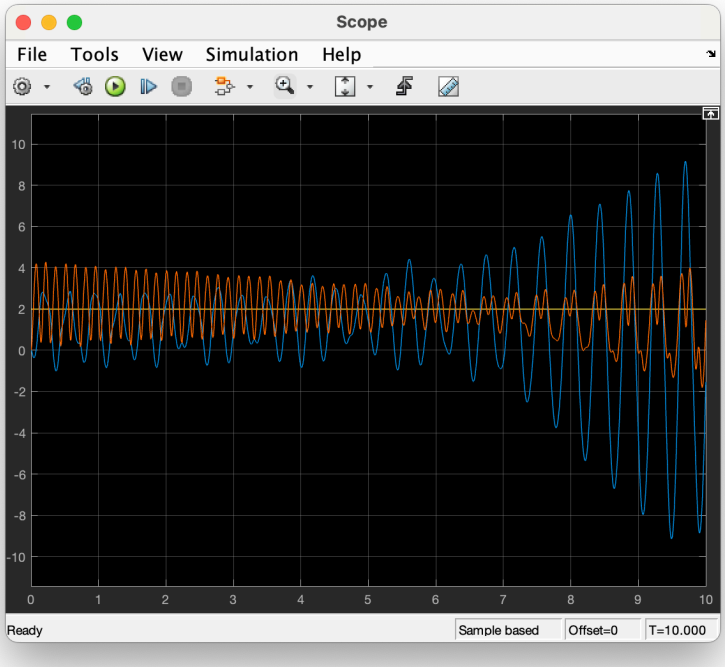
3. D (Derivative) suppresses overshoot and oscillations. If too small, the system may overshoot; if too large, it may slow down the response. Another role of D is to resist sudden external disturbances and prevent system abrupt changes.

The adjustment sequence is: P > I > D

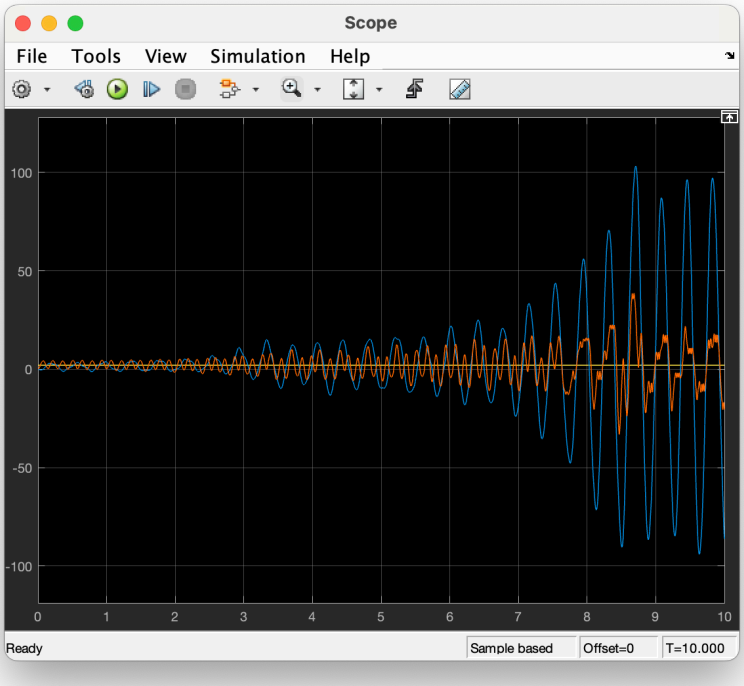
Since the target function is a sine wave, the response after adjusting the parameters is not very pronounced. Therefore, a constant target function is used instead, and the block diagram is shown as follows:



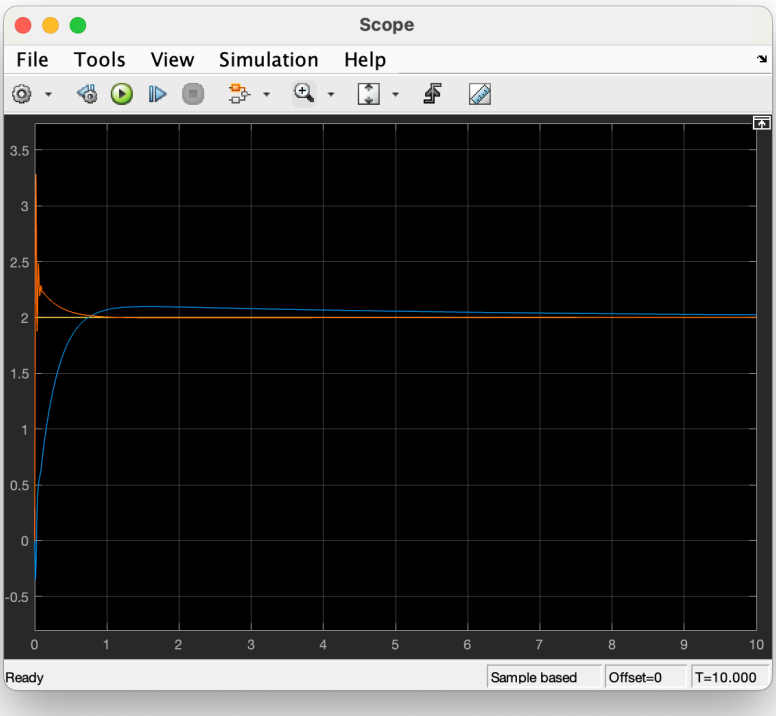
Next, following the P-I-D adjustment sequence, adjust the parameters. First, set Kp=500, and the response is as follows:



It can be observed that the waveform begins to oscillate. Next, set Kp=500 and Ki=80, and the waveform is as follows:



It can be observed that when the integral term is added, the steady-state error decreases. However, due to the oscillations, the steady-state error gradually increases. After adding the derivative term to reduce the oscillations, with Kp=500, Ki=80, Kd=120, the waveform is as follows:



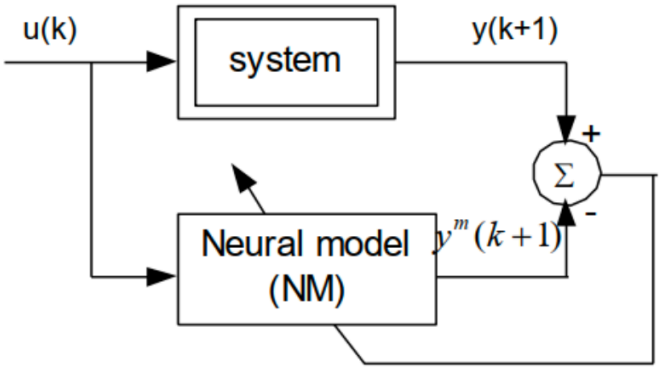
It can be seen that both control quantities gradually converge to the target value, achieving the desired control effect.

Task3

This part uses the method of [2].

The structure of the newly proposed control algorithm, which integrates nonlinear PID-based neural networks, is shown in the figure below. The PID controller is cascaded with the plant neural network model. The error signal, which is the difference between the plant model output and the desired plant output, is used solely to adjust the PID parameters. In this adaptive control scheme, the plant model weights are adjusted offline, and the fixed weights are then used to adjust the PID controller gains online. This control algorithm features a simple structure and reduces computation time.

To develop the neural network model of the plant, a feedforward neural network is used to learn the system, and a back-propagation algorithm is employed to train the weights. The block diagram of the identification system is shown below:



Assume the unknown nonlinear system is represented by:

$$y(k+1) = f\left[y(k), y(k-1), \ldots, y(k-n+1), u(k), u(k-1), \ldots, u(k-m+1)\right]$$

where y(k) is the scalar output, u(k) is the scalar input, n and m are the output and input orders, respectively, and f[…] s the known nonlinear function to be estimated by a neural network.

The neural model for the unknown system can be expressed as:

$$y^m(k+1) = \hat{f}\left[y(k), y(k-1), \ldots, y(k-n+1), u(k), u(k-1), \ldots, u(k-m+1)\right]$$

where ym is the output of the neural model and f_hat is the estimate of f.

The neural model weights are adjusted to minimize the following cost function:

$$E = \frac{1}{2}\left(y(k+1) - y^m(k+1)\right)^2$$

In each control cycle, the controller gains are adjusted. The cost function and the neural model with fixed weights are used to make these adjustments. To speed up the learning process while keeping the system stable, stability during weight updates has been studied. This method includes using momentum to consider past adjustments. The formula for updating the gains with the descent method is:

$$\Delta k_{p,I,D}(t) = -\alpha \frac{\partial E(t)}{\partial k_{p,I,D}} + \beta(t)\Delta k_{p,I,D}(t-1)$$

Alpha is the learning rate and beta is the momentum rate tuned according the flowing equation:

$$\beta(t) = \beta_0 \exp(-b_{P,C})$$

where beta0 is its initial value.

The adaptive momentum tuning method is meant to speed up the adjustment of controller parameters when the controller isn't very robust, and to slow down changes when the controller is performing well.

The derivates is computed as below:

$$\frac{\partial E}{\partial k_p} = \frac{\partial E(t)}{\partial y^m}\frac{\partial y^m}{\partial u(t)}\frac{\partial u(t)}{\partial K_p} = -(y - y^m)\frac{\partial y^m}{\partial u(t)}e_p(t)$$

$$\frac{\partial E}{\partial k_I} = \frac{\partial E}{\partial y^m}\frac{\partial y^m}{\partial u(t)}\frac{\partial u(t)}{\partial K_I} = -(y - y^m)\frac{\partial y^m}{\partial u(t)}e_I(t)$$

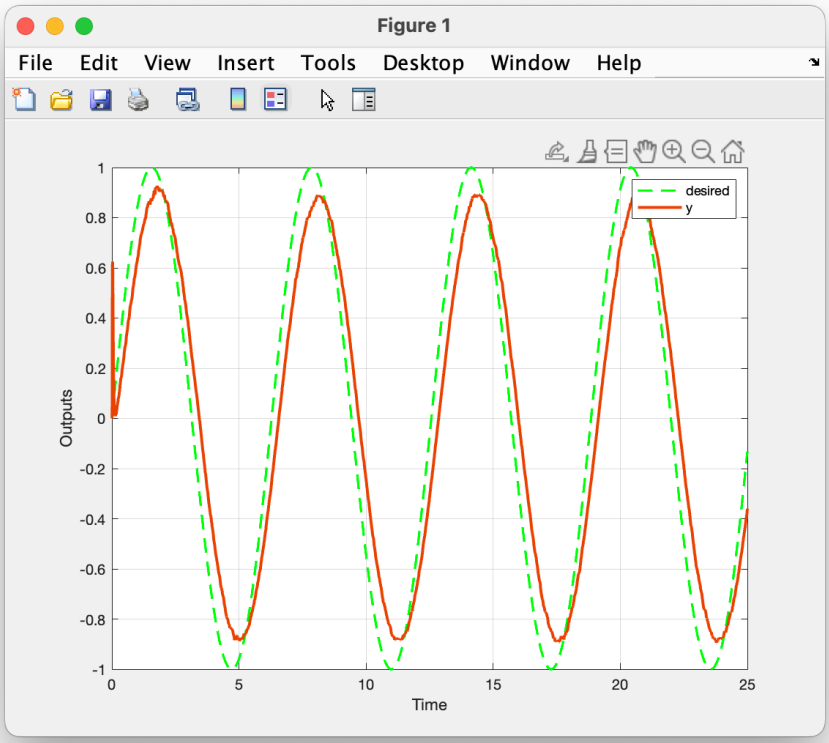
$$\frac{\partial E}{\partial k_D} = \frac{\partial E(t)}{\partial y^m}\frac{\partial y^m}{\partial u(t)}\frac{\partial u(t)}{\partial K_D} = -(y - y^m)\frac{\partial y^m}{\partial u(t)}e_D(t)$$

As the hidden and output neuron functions were defined by the logistic sigmoid function.

$$\frac{\partial y^m}{\partial u} = y^m(1 - y^m)\sum_j W_j^m O_j^m (1 - O_j^m) W_{1j}^m$$

Here, Oj is the output of the j-th neuron in the hidden layer of the neural model, W1 and W2 arethe weights connecting the input neurons to the intermediate layer and the intermediate layer to the output, respectively.

Using MATLAB code to set q2 as the control variable, the effect is as follows:



It can be seen that the output is nearly consistent with the target function. Although there is still some error, the control performance is acceptable.

reference

[1] https://blog.csdn.net/viafcccy/article/details/107988093

[2] https://github.com/iman-sharifi-ghb/Function-Approximation-and-Adaptive-PID-Gain-Tuning-using-Neural-Networks-and-Reinforcement-Learning