AI Programming (IT-3105) Spring 2025
Introductory Project:
A General Purpose JAX-based Controller

Due Date: Thursday, February, 6, 2025 (Last possible demonstration session)

Purposes:

1. Gain hands-on familiarity with Python's JAX package for automatic differentiation.
2. Understand and implement two basic approaches to PID control: one classic and one based on AI.

1 Introduction

In this project, you will build a general-purpose PID controller that can both a) simulate a wide variety of controllable systems, and b) apply either of two types of PID controllers to a control task: the traditional 3-parameter PID model and a neural-network-based version.

For each combination of controller and controlled-entity, your system will employ Python's JAX module to compute gradients and then use those gradients to *intelligently* update the controller parameters according to a basic gradient descent (or ascent) algorithm.

In general, all of the documents listed below (found in the *Materials* section of the course web page) contain useful information for this project:

1. *General Information on Coding Expectations for this Course* (**code-expectations.pdf**) - Ignore this at your own risk!!
2. *Representations for Subsymbolic AI (neural nets in particular)* (**subsymbolic-reps.pdf**) - Only the introductory section is relevant for this project; the rest may be helpful for other projects.

2 Control Systems

In standard terminology of control theory, a control system consists of a **plant** and a **controller**. The plant is any system whose behavior the controller will try to regulate. In simple systems, like those in this

document, that behavior is merely the output value of the plant, which can be viewed as a function that converts inputs to one or more outputs. This output scalar or vector is often symbolized by Y. The output of the controller, known as the **control value** and often symbolized by U, is one of the inputs for the plant function, either directly or after some modification. The input to the controller is an error term (E), which represents the difference between the goal / target (T) behavior of the plant and its actual behavior (Y).

Figures 1 and 2 provide a high-level view of the system that you will design and implement for this project. Modularity is an essential aspect of this implementation: each plant must be its own object (in the object-oriented language of your choice) and must seamlessly integrate with each of the two controllers. See the section entitled *The Critical Divide* in code-expectations.pdf for further discussion of the separation between AI system and SimWorld, which, in this project, is a separation between the controller (whether traditional or AI-based) and the simulated plant. **Failure to display this modularity can result in significant point loss during project evaluation.**

3 Two PID Controllers

This project explores two basic implementations of PID controllers. In the classic case (Figure 1) a PID controller contains three parameters (k_p, k_i, k_d) that combine to map the current error (E), its integral over time, and its derivative to a control output, U. In the more AI-driven approach (Figure 2), these same 3 error values are mapped to U via a neural network whose size can vary. In the neural network controller, the tuneable parameters are the weights and biases of the neurons, while in the classic PID model, only the 3 k parameters admit tuning.

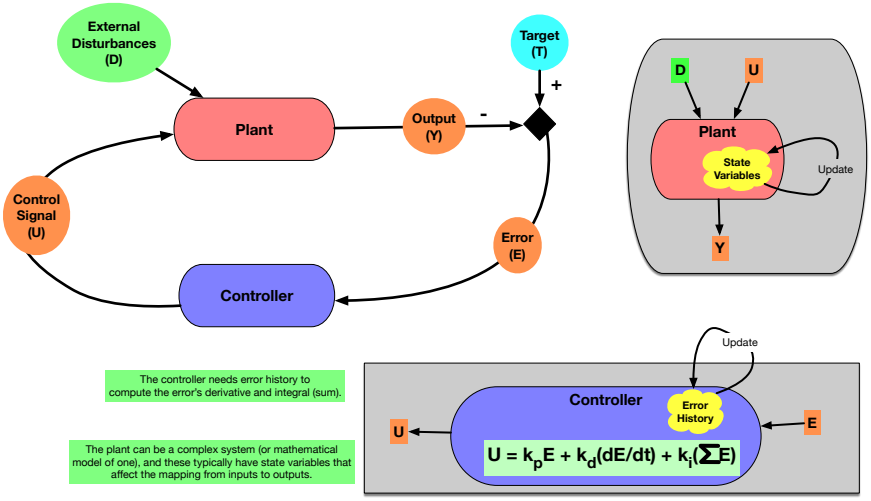

Figure 1: Generic Model of a standard 3-parameter (k_p, k_i, k_d) PID Controller

4 System Overview

Your system will consist of a controller object, a plant object and a system object (CONSYS) that includes both controller and plant. The controller should be subclassed into both a traditional PID controller and a neural-net-based controller. Each plant should be completely modular such that new plants can be added and tested without requiring any changes to the controllers nor to CONSYS.

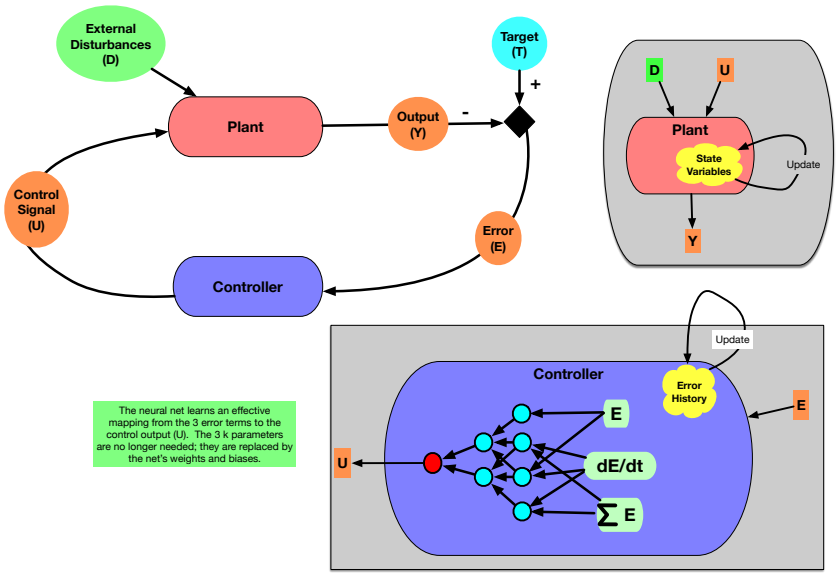

Figure 2: General model of a controller with the PID component realized by a neural network: it still inputs the proportional, integral and derivative errors, but now the tuneable parameters are all the weights and biases of the neural network, which can be of any size.

A typical run of the system should proceed as follows:

1. Initialize the controller's parameters (Ω): the three k values for a standard PID controller and the weights and biases for a neural-net-based controller.
2. For each epoch:
   (a) Initialize any other controller variables, such as the error history, and reset the plant to its initial state.
   (b) Generate a vector of random noise / disturbance (D), with one value per timestep.
   (c) For each timestep:
      - Update the plant
      - Update the controller
      - Save the error (E) for this timestep in an error history.
   (d) Compute the mean squared error (MSE) over the error history.
   (e) Compute the gradients: $\frac{\partial(MSE)}{\partial\Omega}$
   (f) Update Ω based on the gradients.

Figure 3 summarizes these activities, and Figure 4 captures the main interactions between JAX and CONSYS, as discussed in the lecture and available in the lecture notes.

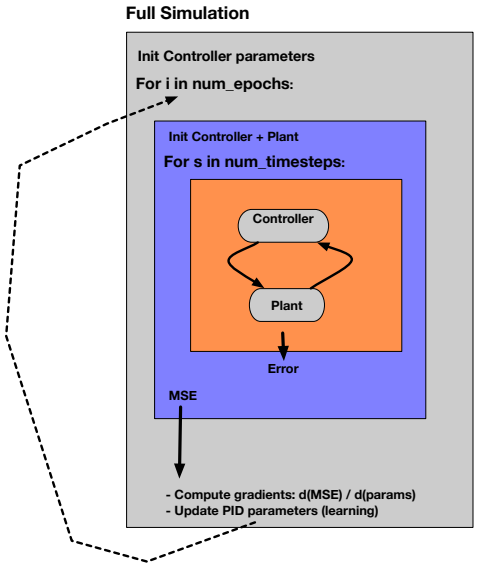

Figure 3: Overview of the running system.

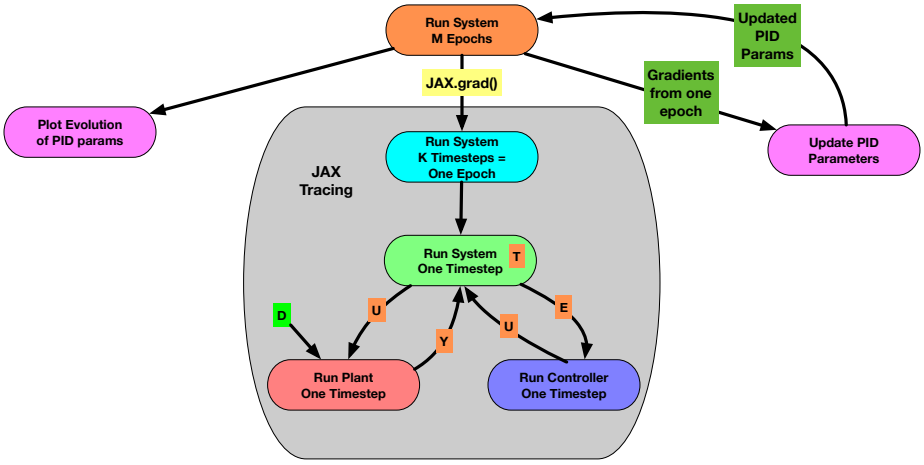

Figure 4: Overview of the interaction between JAX and the entire CONSYS, where D = noise / disturbance, U = control output, Y = plant output, E = T - Y = error.

5 Two Control Tasks

The following are two simple control tasks that your system must tackle with both a standard PID controller and a neural-net based controller, and both using JAX to compute gradients, which are then used to update the controller's parameters.

5.1 The Bathtub Model

This is a control-theory classic, and one of the simplest plants having any mildly interesting behavior. The bathtub is assumed to have a constant cross-sectional area (A) from top to bottom. It has a drain of cross-sectional area (C), which is typically a small fraction of A (e.g. C = A/100). The height of water in the bathtub is H, and the velocity (V) of water exiting through the drain is:

$$V = \sqrt{2gH} \tag{1}$$

where g is the gravitational constant ($9.8 m/sec^2$).

Thus, the flow rate (Q m^3/sec) of exiting water is:

$$Q = VC \tag{2}$$

The bathtub receives two inputs on every timestep (which you can assume is 1 sec in duration):

- U, the output of the controller
- D, a random *noise / disturbance* amount of water that varies with each timestep.

The bathtub volume (B) thus changes as follows:

$$\frac{\partial B}{\partial t} = U + D - Q \tag{3}$$

The change to water height is then:

$$\frac{\partial H}{\partial t} = \frac{\frac{\partial B}{\partial t}}{A} \tag{4}$$

Given a particular starting height of the bathtub: H_0, the **control task** is to keep it at that level, despite the constant disturbance D and flow loss Q. You are free to assume that the bathtub itself has infinite height.

5.2 Cournot Competition

The second plant (controlled system) comes from economics and involves two rival producers of a product (X). Both produce a particular amount: q_1 and q_2 of X, but both will receive the same price, p(q) for their product, where:

$$q = q_1 + q_2 \tag{5}$$

The price is inversely proportional to total production and is modelled (where p_{max} is the highest possible price) as:

$$p(q) = p_{max} - q \tag{6}$$

To simplify the calculations, assume that $0 \leq q_1 \leq 1$, $0 \leq q_2 \leq 1$, and enforce these constraints in your code.

The simulation runs from the perspective of producer 1, who uses the controller output (U) to compute $\frac{\partial q_1}{\partial t}$ at each timestep. Thus,:

$$U = \frac{\partial q_1}{\partial t} \tag{7}$$

and

$$q_1(t+1) = U + q_1(t) \tag{8}$$

Conversely, changes to q_2 constitute *noise / disturbance* (D) in this model, such that:

$$q_2(t+1) = D + q_2(t) \tag{9}$$

At each timestep:

1. q_1 updates based on U.
2. q_2 updates based on D.
3. $q = q_1 + q_2$
4. $p(q) = p_{max} - q$

Then, on each timestep, producer 1's profit (P_1) is computed as follows:

$$P_1 = q_1 * [p(q) - c_m] \tag{10}$$

where c_m is the marginal cost: the cost to produce each item, independent of the total number produced.

In this model, the target value (T) denotes the goal profit for each timestep. Thus:

$$E = T - P_1 \tag{11}$$

and the error (E) serves as input to the controller on each timestep.

In this model, c_m can be a small fraction, such as 0.1, but you are free to experiment with different values of it along with values of p_{max} and T.

5.3 Plant Number 3

You must choose a 3rd plant to run in your system. To find such an example, it is probably easier to search for mathematical models within a particular area, such as population biology, or chemistry, or economics or whatever your favorite area is. That seems easier than searching for *control theory examples*, which doesn't seem to yield too much. You should be looking for a relatively simple mathematical model and considering:

1. Does the model involve differential or difference equations? It should.
2. What variable might one want to control (i.e., keep near a target value)? For example, in population biology, it might be the number of rabbits, in chemistry, the concentration of chlorine in a swimming pool.
3. What control output to use? For the rabbit example, one could just add more rabbits or remove some of their predators. For the swimming pool example, the mass or volume of chlorine added per timestep is a reasonable control output.
4. How to introduce noise? This should be pretty easy. You can usually apply noise to just about any variable in a plant.

If you see obvious answers to these questions when looking at the model, then it should be easy to code it up as a new plant module in your CONSYS.

5.4 Noise

In all plants / models used (both the two above and the additional model that you choose), it is important to have a time series for the noise / disturbance parameter (D): one value for each timestep. A new series must be generated for each run of the system so that each timestep of each epoch receives a D value that should not be predictable from previous values. Any of python or numpy's random-number generators works fine. In the two models above, the controller should be able to work with D values in the range $[-0.01, +0.01]$, but that range may change for your third model.

6 Pivotal Parameters

As discussed in the file **code-expectations.pdf**, each project in this class has pivotal parameters that should be easily set and modified (via a GUI, config file, or simple call to your main function). Here, the

word *parameter* refers to those of your entire system, not to the controllers themselves, which also have parameters. So we can call these higher-level, meta-parameters the *configuration parameters*.

For this assignment, the pivotal configuration parameters are the following:

1. The plant to simulate: bathtub, Cournot competition, your additional model, etc.
2. The controller to use: classic or AI-based
3. Number of layers and number of neurons in each layer of the neural network. Your system should handle anywhere between 0 and 5 hidden layers.
4. Activation function used for each layer of the neural network. Your system must include at least Sigmoid, Tanh and RELU.
5. Range of acceptable **initial** values for each weight and bias in the neural network.
6. Number of training epochs
7. Number of simulation timesteps of the CONSYS per epoch
8. Learning rate for tuning the controller parameters, whether classic PID or neural-net-based.
9. Range of acceptable values for noise / disturbance (D).
10. Cross-sectional area (A) of the bathtub
11. Cross-sectional area (C) of the bathtub's drain.
12. Initial height (H_0) of the bathtub water.
13. The maximum price (p_{max}) for Cournot competition.
14. The marginal cost (c_m) for Cournot competition.
15. At least two parameters for your third plant.

7 Visualization

To assess the performance of your system, the following two visualization capabilities must be included:

1. The Progression of Learning. This is a simple plot (Figure 5, left) with the episode number on the x axis and the mean-squared error (MSE) on the y axis. Note that this is the MSE for only the error values of the given epoch.
2. (Standard PID controller only). A plot of the changes to the three k parameters across the epochs (Figure 5, right). Nothing similar is required for the neural-network-based controller.

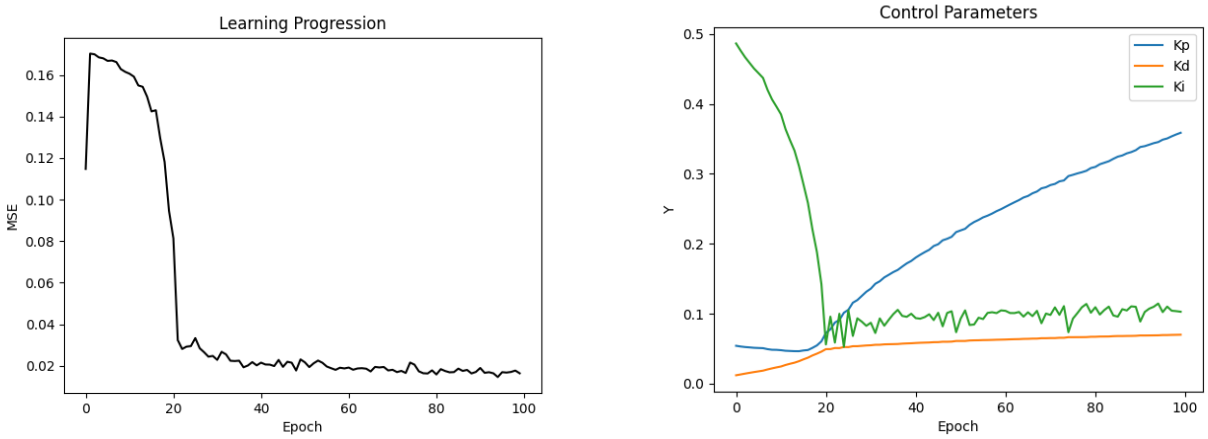

Figure 5: Two essential visualizations for standard PID control simulations. (Left) Plot of the mean-squared error (MSE) over the training epochs. (Right) Plot of the three PID-control parameters as a function of the training epoch.

8 Demonstrating Your Code

You will give an in-person demonstration of your code to a member of the IT-3105 staff. Under special circumstances, this can be done online, but if you are living in Trondheim and have no mobility issues, then you will be expected to meet with a staff member for a brief (10-20 minute) demo. If, however, a staff member has online demo sessions, everyone is free to sign up for those. There are no guarantees that staff members will provide these as a general courtesy. They may only provide them for students with special needs.

Even if you work in a group, each member will give a separate demonstration, with no other group members present.

Any online *tricks* with group members hiding off-screen and giving tips will result in failure for the whole group on the project. Refusal to turn on a camera during an online demo is also grounds for immediate failure. Do not sign up for an online demo if your camera is not working.

For this project, bring a fully-completed lab report to the demo session. That report must have a simple summary of 6 different runs: the three different plants regulated by the two different controllers. A run summary consists of the following:

1. The configuration parameters for the run, presented in a table.
2. The one or two visualizations of progress, as described above.
3. A paragraph or two of text that summarizes the run.

In addition, your report must include a complete mathematical description of the third plant model that you choose.

One lab report per group is sufficient. Each member will be expected to have the report at their demo

session.

At the demo, we will go through the lab report and your code. To test the flexibility of your code, we will ask you to make changes to your configuration parameters and do a few runs. These changes will not be major, but, for example, when this document states that your neural network should include anywhere from 0 to 5 hidden layers, or one of several activation functions, then we may ask you to try a network with no hidden layers or with RELU instead of Sigmoid activations. We will also be checking for the modularity of your code. It should be clear from a brief look at the code that a 4th or 5th plant could be added to your system with no changes needed to the rest of the code.

We will be asking you questions about your code, so it is extremely important that you can answer them. These questions are not designed to *trip you up*, but only to verify that you have done the work, or, in the case of group work, that you understand both your own and your partner's code. If there is any doubt about this, you will be called in to a follow-up demonstration for the course instructor.

9 Deliverables

1. Well-structured code – be sure to read **code-expectations.pdf** carefully – that you can discuss and explain (in general and in detail) with a reviewer on demonstration day.
2. The lab report (explained above)

A zip file containing your commented code must be uploaded to BLACKBOARD immediately before or after your demonstration session. You will not get explicit credit for the code, but it is crucial that we have the code online in the event that you decide to register a formal complaint about your grade (for the entire course).

10 Warning

When coding up AI systems, whether from scratch (as in this class) or using advanced libraries, the difference between error-free functioning of your system and *intelligent* functioning is often very significant. Getting the system up and running (without crashes) may take a week or two, but another few days (or weeks) may be needed to tune the system to successfully solve problems. Hopefully, you will find JAX to be a useful tool for learning controller parameters, but determining meta-parameters such as learning rates, initial neural-network weights, etc. can still take awhile. Budget your time accordingly.