IT3105 Artificial Intelligence Programming - Jax PID Controller tuner

Andreas Bentzen Winje

Contents

List of Tables

List of Figures

1 Plant 3 description: RobotArm

Plant three implements a two-dimensional robot arm of length l and mass m and an angular position θ. The objective of the system is to minimize the angle between its current position and target position. The robot arm is subject to gravity, and its torque is subject to both coulomb and viscous friction. In addition, the arm has an interval of legal position values where crossing the boundary of this interval results in a collision and thus a reflection of the current angular velocity.

The full set of parameters defined and their descriptions for the RobotArm-plant can be seen in table 1. Following is a mathematical description of the plant's step function.

Parameter	**Description**
target	target angle
angle0	initial angle
delta_time	size of a timestep
mass	mass of the arm
length	length of the arm
gravity	gravitational acceleration
multiplier	multiplier of control signal
coulomb_fric	coulomb friction coefficient
viscous_fric	viscous friction coefficient
interval	legal interval for the angle

Table 1: Configuration parameters of the robot arm-plant and their descriptions.

For each step of the plant, both the current angular position and velocity are updated and returned, and the update of these values is approximated using the implicit Euler's method as such:

$$\omega_t = \hat{\omega}_{t-1} + a_t * (\Delta t)$$
$$\theta_t = \hat{\theta}_{t-1} + \omega_t * (\Delta t)$$

where ω and θ are the angular velocity and position, respectively. The velocity is calculated from the moment of inertia (I) and torque (τ) of the arm as such:

$$I = \frac{1}{3} * m * l^2$$
$$a_t = \frac{\tau_t}{I}$$

The torque is dependent on the control signal to the plant, and is calculated like this:

$$\tau_t = U - G + D - \tau_{fric}$$

Where U is the control input, D is the disturbance, G is the torque of gravity and τ_{fric} is the torque

of friction. The torque from gravity, assuming an equal mass distribution along the robot arm, and the friction is given by this:

$$G = \frac{l}{2} * m * g * \cos\theta_t$$
$$\tau_{fric} = K_C * sign(\omega_t) + K_V * \omega_t$$

To limit the position to the legal interval, and to reflect the velocity when the interval is breached, we calculate the final position and velocity like this:

$$\hat{\omega}_t = -sign(\omega_t - \omega_{max}) * sign(\omega_t - \omega_{min}) * \omega_{t+1}$$
$$\hat{\theta}_t = \min(\theta_{max}, \max(\theta_{min}, \theta_t))$$

Finally, the error of the system is given by:

$$e_t = T - \hat{\theta}_t - \hat{\omega}_t$$

Here, we add the velocity to the equation to prevent it from becoming too high, which indeed proved to be a problem in preliminary tests.

2 Setup and configuration

All runs used the parameters for the system, controllers, and plants. The parameters used can be seen in table 2, where each parameter type is separated by a line and named in bold text. The neural controller implements 6 different activation functions, named "relu", "sigmoid", "tanh", "leaky_relu", "softmax" and "linear" in the configuration file. For the tests with the neural controller, one hidden layer of 3 neurons was used, and both activations used the "linear" function, as can be seen in table 2.

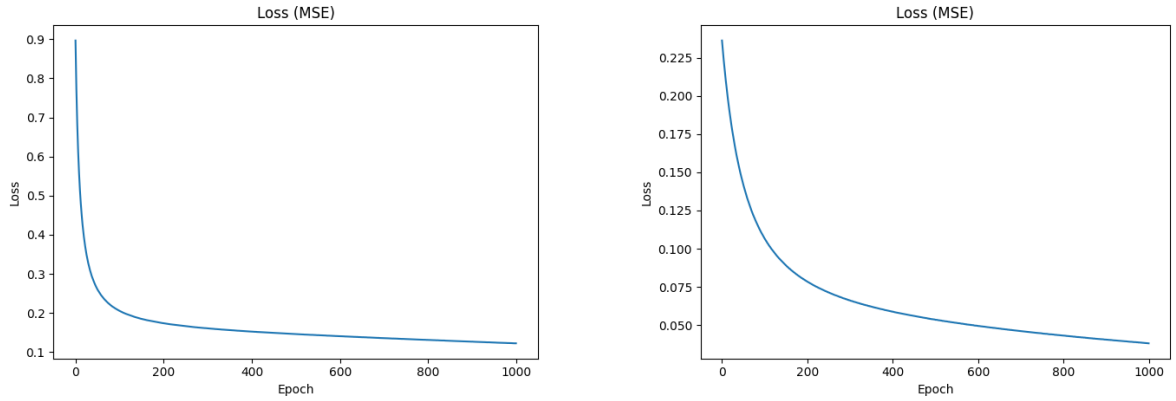
System	
epochs	1000
steps_per_epoch	40
learning-rate	0.01
noise_range	[-0.01, 0.01]
Default controller	
params_range	[0,1]
Neural controller	
inputs	3
outputs	1
hidden_layers	[3]
activations	["linear","linear"]
init_weight_range	[0,1]
init_bias_range	[0,1]
Bathtub plant	
target	3.14
area	100
cross_section	1
gravity	9.81
Cournot plant	
target	0.3
max_price	2.1
margin_cost	0.1
RobotArm plant	
target	0.25*3.14
angle0	0
delta_time	0.1
mass	1.
length	2.0
gravity	9.81
multiplier	1
coulomb_fric	0.4
viscous_fric	0.6
interval	[-3.14, 3.14]

Table 2: Parameters for the system, both controllers and all three plants.

3 Results

3.1 Plant 1: Bathtub

The loss graphs for the bathtub runs can be seen in fig. 1. Here we can see that the loss drops steadily for both controllers and that the neural controller starts at a lower loss than the default controller. This could be due to randomness the initialization of the control parameters.



(a) **(b)**

Figure 1: Loss graph for bathtub plant-run for both the default (a) and the neural (b) controller.

In fig. 2, the control parameters for the default controller for each epoch can be seen. Here we can see that the propositional part increases slightly over the whole run, the integral part increases a lot over the whole run, but not so much towards the end, and the derivative part stays unchanged for the whole run.

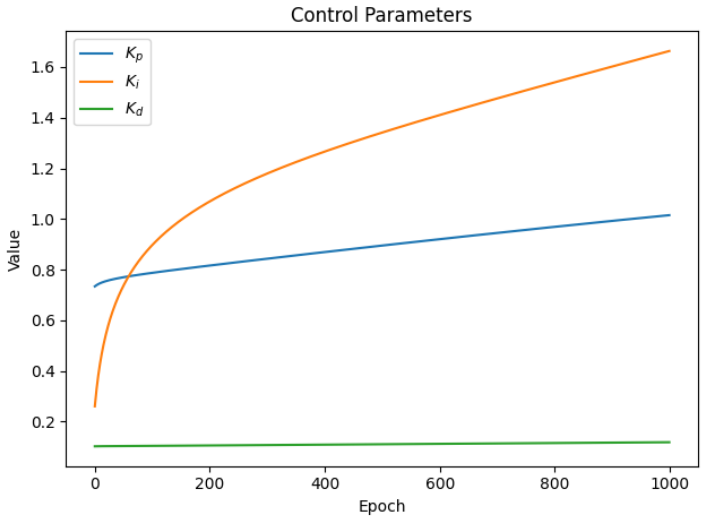
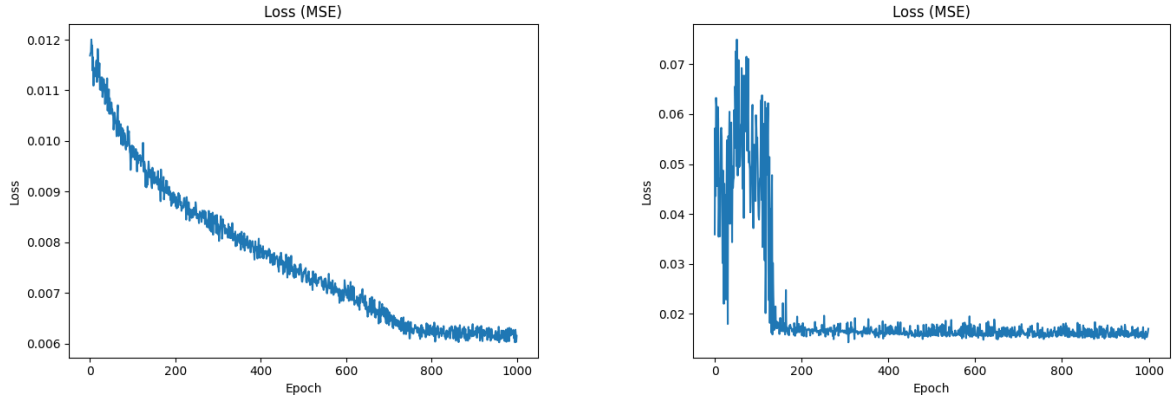


Figure 2: Default controller's control parameters over time for bathtub plant-run.

3.2 Plant 2: Cournot

Figure 3 shows the loss graphs for both controllers over both runs for the Cournot plant. Here we see that the graphs are much more jagged than for the bathtub run in fig. 1, but we also see that it is much lower and never exceeds 0.1.



(a) **(b)**

Figure 3: Loss graph for Cournot plant-run for both the default (a) and the neural (b) controller.

If we look at the control parameters in fig. 4 we see that the changes are very minimal, with the propositional and integral parts only changing by about 0.1 to 0.15.

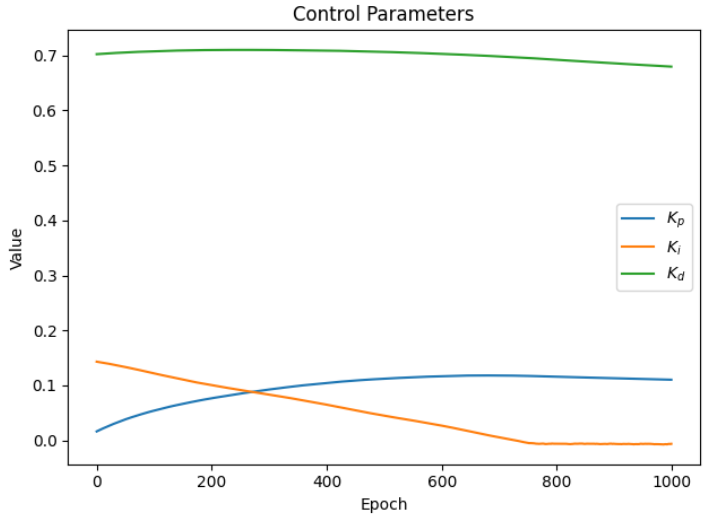
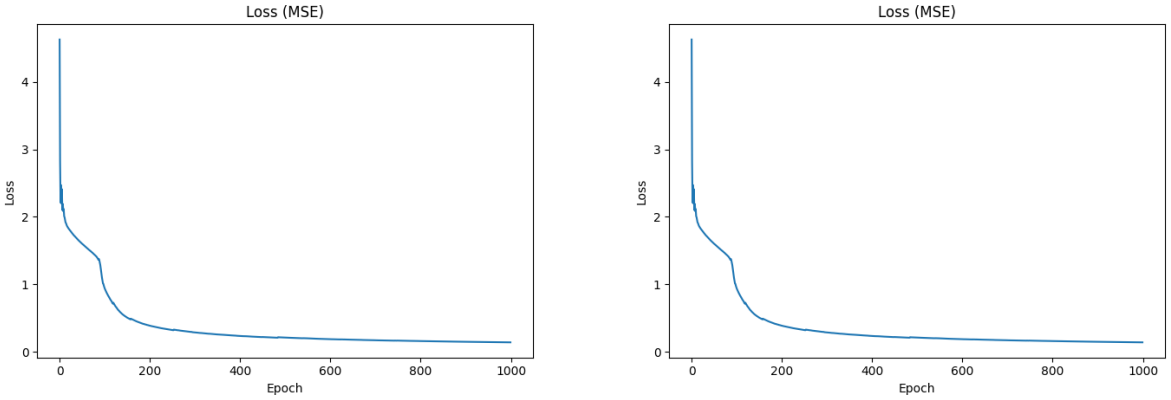


Figure 4: Default controller's control parameters over time for Cournot plant-run.

3.3 Plant 3: RobotArm

The loss graphs for the RobotArm plant can be found in fig. 5. Here we see a shard drop in loss for both controllers, with minimal to no drop towards the end, suggesting that an optimum was reached quickly.

The control parameters for the RobotArm plant can be found in fig. 6. Here we again see a lot of change in the propositional and integral parts, and minimal or no change in the derivative part.



(a) **(b)**

Figure 5: Loss graph for RobotArm plant-run for both the default (a) and the neural (b) controller.

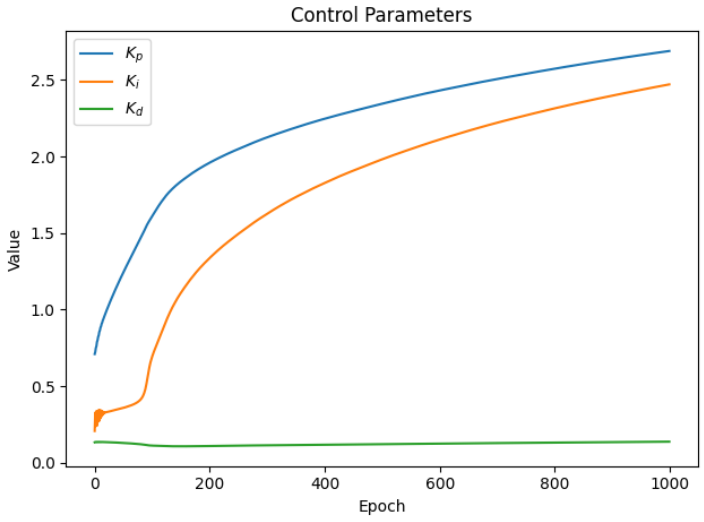
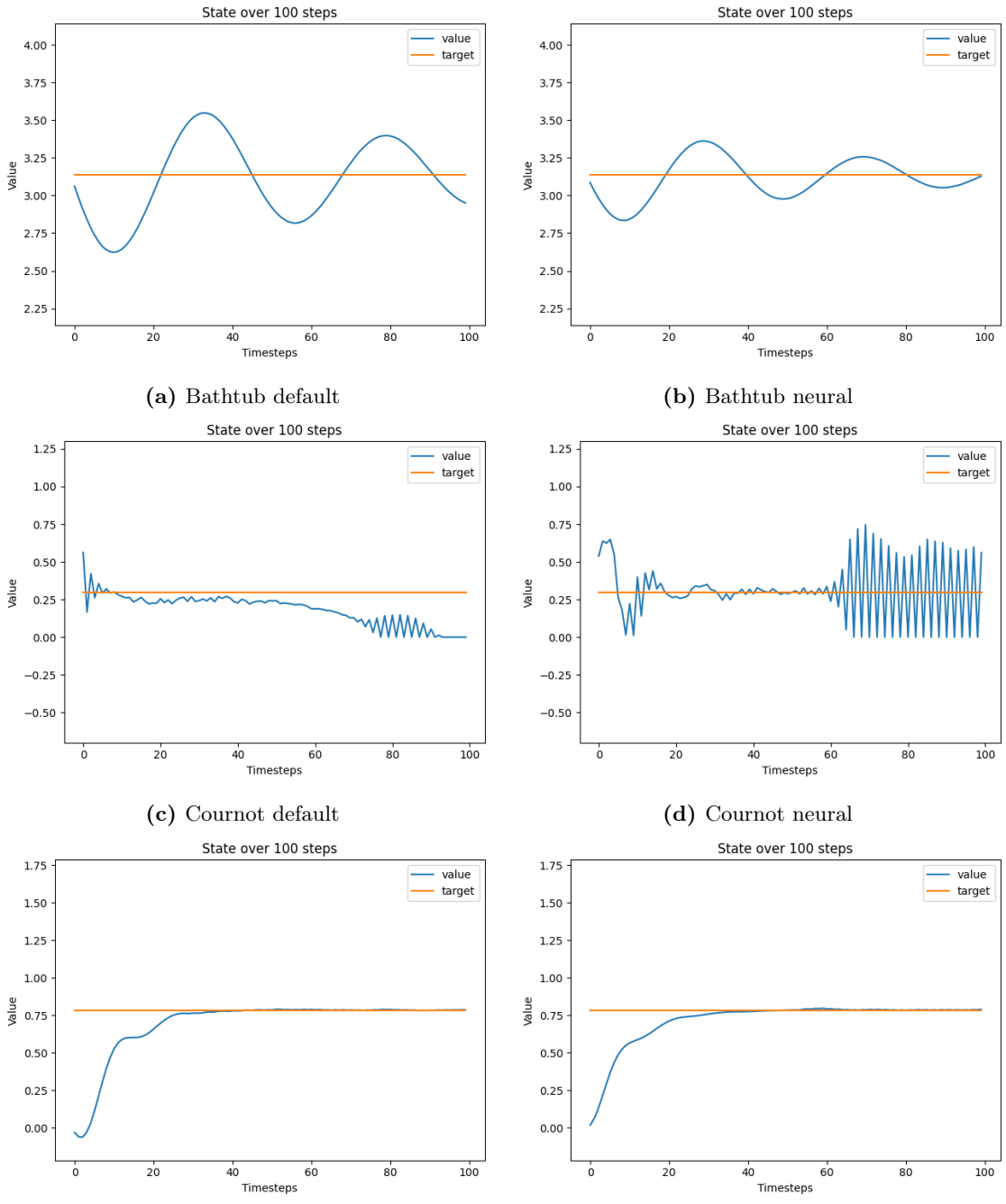


Figure 6: Default controller's control parameters over time for RobotArm plant-run.

3.4 Simulated examples

As an additional visualization, fig. 7 shows the state of the plants against their targets when simulated for 100 steps using tuned controllers. As we can see, both the bathtub and the RobotArm plant do well, as they both approach their target, although slowly for the bathtub plant. The Cournot plant, however, seems to become unstable after around 50-60 steps.



(a) Bathtub default

(b) Bathtub neural

(c) Cournot default

(d) Cournot neural

(e) RobotArm default

(f) RobotArm neural

Figure 7: Plant value and target for runs with tuned controllers over 100 steps. Here, the first two simulated the bathtub plant for default (a) and neural (b) controllers, the next two simulated the Cournot plant for default (c) and neural (d) controllers, and the last two simulated the RobotArm plant for default (a) and neural (b) controllers.