Data Drive Linear Control using Almost Linear Recurrent Neural Networks

Andrei Prioteasa
As part of the HUMML internship in the SS25
https://github.com/PrioteasaAndrei/al-rnn-control

July 9, 2025

1 Foreword

This report summarizes the work I have done as part of the HUMML laboratory, during the SS25. My internship officialy started in the WS24/25, but the idea pursued during that time did not lead to any relevant results. Nevertheless, since it has been a lot of time and effort, I have included these results in a separate report. See attached.

2 Overview

In this project, I extend the functionality of the existing AL-RNN framework, by including a linear control mechanism (Linear Quadratic Regulator). A desired goal state in the observation space is provided to the model and the model will learn to control the system to reach that goal state using optimal linear commands for each linear subregion of the latent space.

The extended model is able to learn the dynamics of the system and to control it to the goal state, while being trained only on random actions applied to the system. The approach is tested on the Inverted Pendulum and Cartpole environments from OpenAI Gymnasium. We show that, in the case of the inverted pendulum, a linear control strategy is not sufficient in order to stabilize the system in the upright position and an energy based control mechanism should be desired. We show that the model is able to stabilize the CartPole environment if allowed continous commands.

3 Motivation

The main motivation of this project can be formulated as follows:

Given a Piecewise Linear Approximation of the dynamical system (as generated by an AL-RNN), can we use optimal linear control per PWL region to stabilize the system to a desired goal state?

That is to say, are the optimal commands computed for each linear subregion a good approximation of the optimal control strategy for the system?

4 Data Generation

In order to assess our objective, we will use the two common benchmarks for control problems:

- The CartPole environment, where the control is allowed to be continous
- The Inverted Pendulum environment (with the same consideration as above)

Note: *The Inverted Pendulum environment is a continuous control problem, where the control is allowed to be continous, so our assumption does not limit generalization. In the case of the CartPole environment, the control command space is discrete (0 or 1) and our predicted commands are continuous. One could*

perhaps learn a projection from the continuous command space generated by the AL-RNN using LQR and then use the discrete control commands to stabilize the system, but we did not test this approach.

We design a data generation pipeline for both environments, where actions are sampled uniformly from the action space and applied to the system for a fixed number of steps. The data is then used to train the AL-RNN model.

In the case of the Inverted Pendulum environment, we have empirically shown that generating a single long trajectory using random actions is equivalent (in terms of exploration of the state space) to generating multiple short trajectories using random actions and combining the resulting dataset. We generated a trajectory of **9500** time steps, where at each time step, a random action is applied to the system and use it to train our ALL-RNN control model.

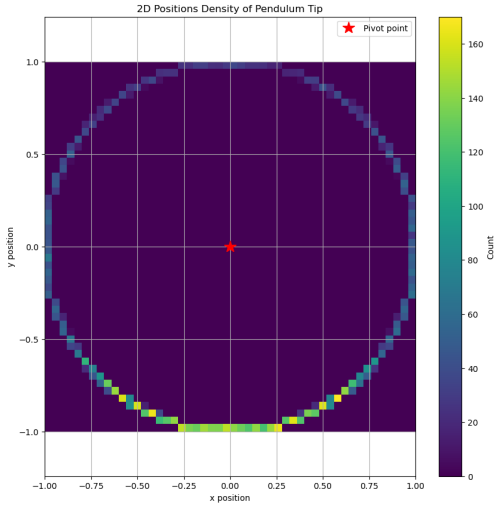


Figure 1: Density of states in the training dataset for the Inverted Pendulum environment.

In the case of the CartPole environment, trajectories generated by applying random actions to the system end up being too short to train the model (average length of 40 time steps before the episode ends). We generate a total of 1000 episodes, each with a minimum length of 40 time steps and a maximum length of 100 time steps (retrying until the episode is at least 40 time steps long). We pad the generated trajectories with NaNs to achieve a constant length of 100 time steps. We modify the existing Time Series Dataset class to only sample batches that do not contain NaNs, in this way the boundary between episodes is properly handled during training.

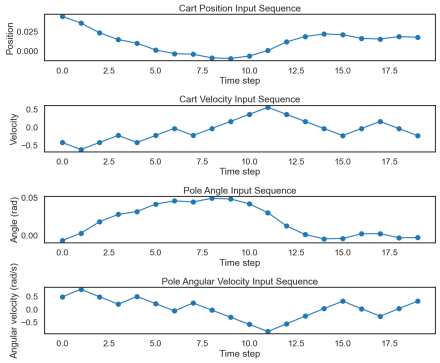


Figure 2: Sample training data from the CartPole environment.

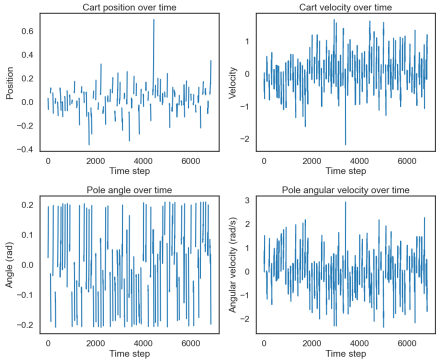


Figure 3: Training episodes padded with NaNs.

5 AL-RNN Control Model

The AL-RNN control model extends the base model by incorporating external inputs and applying optimal linear control through an LQR controller:

$$z_{t+1} = A \cdot z_t + W^T \Phi^*(z_t) + C^T u_t + h \tag{1}$$

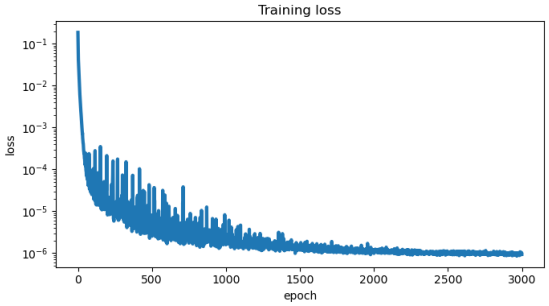
$$\Phi^*(z_t) = [z_{t,1},\ldots,z_{t,M-P},\text{ReLU}(z_{t,M-P+1}),\ldots,\text{ReLU}(z_{t,M})]^T \tag{2}$$

$$z_0 = B \cdot x_0 \tag{3}$$

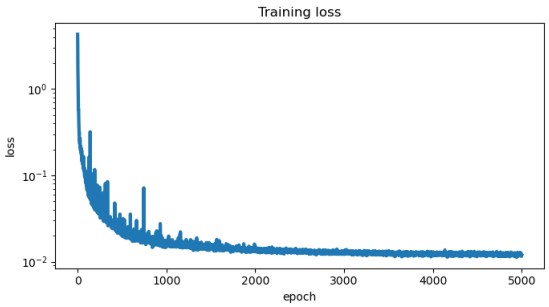
- $u_t \in \mathbb{R}^{control\_dim}$ is the control command at time t
- $C \in \mathbb{R}^{M \times control\_dim}$ is the control matrix that maps the control command to the latent space, initialized as $C \sim \mathcal{N}(0, 0.1^2)$
- $B \in \mathbb{R}^{M \times M}$ is the projection matrix that projects the observation space to the latent space

B^T is used to project the latent space to the observation space. Respectively, I use C to project the command from the latent space to the command space, C^T is used to project the command from the command space to the latent space.

The same training procedure with teacher forcing is used as in the base model, with the addition of the control command u_t and the control matrix C. We find an optimal 5000 training epochs for the Inverted Pendulum environment and 3000 training epochs for the CartPole environment. The loss curves are shown in Figure 4 and Figure 4 respectively.



(a) CartPole environment.



(b) Inverted Pendulum environment.

Figure 4: Training loss curves for the AL-RNN control model.

6 Linear Subregion Equations

For each activation pattern $\alpha_i \in \{0,1\}^P$, the model defines a linear subregion with equations:

$$W_{subregion} = \text{diag}(A) + W \cdot D_{subregion} \tag{4}$$

$$h_{subregion} = h \cdot D_{subregion} \tag{5}$$

where $D_{subregion}$ is a diagonal matrix constructed as:

$$D_{subregion} = \text{diag}([1,\ldots,1,\alpha_1,\ldots,\alpha_P]) \tag{6}$$

with the first $M - P$ entries being 1 and the last P entries being the binary activation pattern $\alpha \in \{0,1\}^P$. A dictionary of linear subregion equations for each unique activation pattern in the form of $\alpha_i \rightarrow (W_{subregion}, h_{subregion})$ is collected. Some of these subregions are initialized during training, others have not been seen during training and have to be initialized during inference, thus we devise two initialization strategies:

- **Full initialization:** All posible activation that have not been seen during training are initialized before testing. For a large P, this may become computationally expensive.
- **On-the-fly initialization:** We initialzie each unseen activation pattern on-the-fly, by computing the $(W_{subregion}, h_{subregion})$ during inference. This is computationally more feasible for large

P, but increases inference time dramatically.

The **Full initialization** strategy is preferd, because the dictionary can be computed once and then stored for future use.

7 Linear Optimal Control

A Linear Quadratic Regulator (LQR) control is implemented that generates the optimal control command u_t for the current state z_t and the goal state z_{goal}, corresponding to each linear subregion α_i.

The goal state z_{goal} is the state that the model is trying to reach (in latent space), which is directly projected from the observation space to the latent space through the learned projection matrix B:

$$z_{goal} = x_{goal} B \tag{7}$$

The following steps are taken to compute the optimal control command u_t:

1. For each time step t, retrieve (or initialize) the corresponding linear subregion equations $(W_{subregion}, h_{subregion})_i$ for the current activation pattern α_t:

2. Solve the Discrete Algebraic Riccati Equation (DARE):

$$P = \text{DARE}(W_{subregion}, B_{control}, Q_{lqr}, R_{lqr}) \tag{8}$$

3. Compute the LQR gain matrix:

$$K_{lqr} = (R_{lqr} + B_{control}^T P B_{control})^{-1} (B_{control}^T P W_{subregion}) \tag{9}$$

   where:

   - $B_{control} = 0_{M \times M}$
   - $Q_{lqr} = I_{M \times M}$ is the state cost matrix, initialized as the identity matrix
   - $R_{lqr} = k \cdot I_{action\_dim \times action\_dim}$ is the control cost (control energy) matrix, where k is a hyperparameter

     We set $B_{control}[i, 0] = 1$ for those dimensions i that correspond to the control input. This is the weight of the control input (in case control is multi-dimensional) and has to be distributed between the dimensions of the control input.

4. Compute the optimal control input in the current subregion in the observation space

$$u_t = 2.0 \cdot \tanh(-K_{lqr}(z_t - z_{goal})) \tag{10}$$

5. Project control to latent space and update state:

$$u_t^{latent} = C^T u_t^{obs} \tag{11}$$

$$z_{t+1} = W_{subregion}^T z_t + u_t^{latent} + h_{subregion} \tag{12}$$

7.1 Effect of control

To assess the effect of the control command in the observation space, we compare the state evolution with and without control by computing the difference between the two trajectories:

$$\begin{aligned} z_{t+1}^{control} &= W_{subregion}^T z_t + u_t^{latent} + h_{subregion} \\ z_{t+1} &= W_{subregion}^T z_t + h_{subregion} \\ x_{t+1}^{control} &= z_{t+1}^{control} B^T \\ x_{t+1} &= z_{t+1} B^T \\ \Delta x_{t+1} &= x_{t+1}^{control} - x_{t+1} \end{aligned} \tag{13}$$

where Δx_{t+1} represents the isolated effect of the control command in observation space at time $t + 1$.

7.2 Efficiency of the control mechanism

Since linear control is by definition, the most efficient (in terms of computational cost) control mechanism, we note that there is no significant increase in inference time using the LQR control mechanism when the equations of the linear subregions are precomputed (*unfortunately I have not saved the results of this experiment*).

LQR adds a total of **11 matrix multiplications** (6 for computing the Ricatti Equation and 5 for computing the LQR gain matrix) and **2 matrix inversions** (for computing the LQR gain matrix and the Ricatti Equation). Since both the LQR gain matrix and the Ricatti Equation are computed only once for each subregion, the computational cost of the LQR control mechanism is **constant** with respect to the length of the trajectory and depends only on the number of subregions (what we mean here by constant is that the computational cost is independent of the length of the trajectory **if the subregions are precomputed**. In the case of on-the-fly initialization, the computational cost is linear with respect to the length of the trajectory).

8 Experimental results

In this section we present the results of the experiments for the Inverted Pendulum and CartPole environments. We conclude that control is achieved in the CartPole environment (which shows almost linear dynamics in the vecinity of the goal state), but not **deterministically** in the Inverted Pendulum environment. In most trials (slighlty different initial states), the control mechanisms saturates the control command to the upper limit of the torque command and is not able to stabilize the system in the upright position. On some occasion, the model successfully stabilizes the system in the upright position, but this is not deterministic and further investigation is required. Based on the current observation, we conclude that this may be because of applying *tanh* smoothing to the control command or because of the trade-off between the position and angular velocity objectives.

Literature suggests an energy based control mechanism (physically informed about the dynamics of the system) or a 2-phase control mechanism (first phase to lift the pendulum up and second phase to stabilize it in the upright position) would be required.

Model	Dstsp	DH
IP (P=5)	6.17	0.175
CP (P=5)	8.86	5.14e-5

Table 1: State space (Dstsp) and Hellinger (DH) distances for trained models. IP: Inverted Pendulum, CP: CartPole.

8.1 Inverted Pendulum

The Inverted Pendulum environment imposes a $[-2, 2]$ restriction on the torque command, which effectively limits the control command from directly lifting the pendulum up. Instead, in order to achieve the upright position, momentum has to be created first, and then a linear control strategy could be perhaps used in the vecinity of the equilibrium point.

We tested our approach by first fitting an AL-RNN model to learn the dynamics of the system and then using the LQR control mechanism to stabilize the system in the upright position. Since LQR assumes no restrictions on the command space and we are dealing with a restricted command space, we observe that our strategy saturates the control command to the upper limit of the torque command (Figure 10) and only in some ocassions, the model is able to stabilize the system in the upright position (using no command smoothing and a higher control weight).

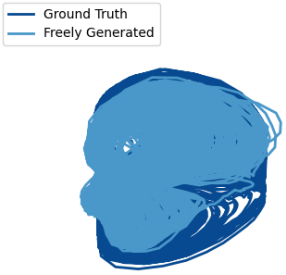


Figure 5: Freely generated trajectory of the Inverted Pendulum environment using the trained AL-RNN model (P=5) with external input.

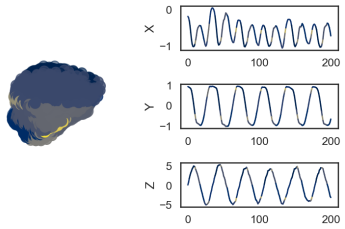


Figure 6: Segments of the freely generated trajectory of the Inverted Pendulum environment using the trained AL-RNN model with external input and P=5.

Since the LQR control mechanism allows for weighted control objectives (out of the dimensions of the observation space), a trade-off must be made in between the accuracy of the position objective vs the angular velocity objective.

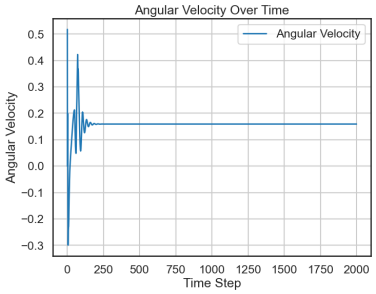


Figure 7: Evolution of angular velocity of the Inverted Pendulum environment under linear control.

8.1.1 Hyperparameter analysis: control gain and smoothing

The matrix R_{lqr} is a hyperparameter of the LQR control mechanism that controls the weight of the control command. We experiment with different values of R_{lqr} and observe the effect on the control command and the trajectory of the system. This is equivalent with using more energy for the control command (keep in mind that LQR assumes unrestricted control command, while the control command is restricted to the torque command [-2,2] in the Inverted Pendulum environment). We initialize our gain matrix as the identity matrix scaled by a factor k:

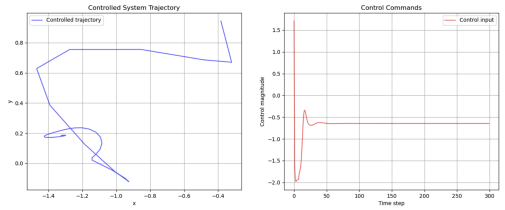


Figure 8: Saturated equilibirum point of the Inverted Pendulum environment.

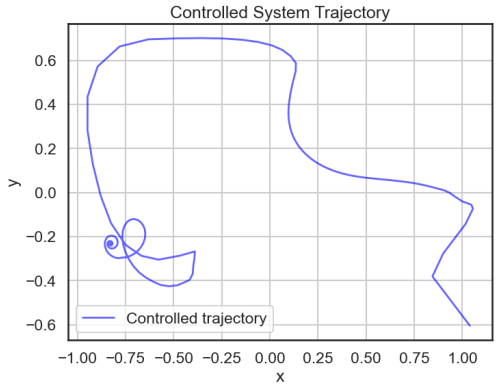


Figure 9: Trajectory of the Inverted Pendulum environment using the trained AL-RNN model with external input and LQR control.

$$R_{lqr} = k \cdot I_{action\_dim \times action\_dim}$$

Figure 11 shows the effect of the control gain on the control command and the trajectory of the system using 3 distinct configurations. We were only able to achieve the desired goal state (0,0) in the unsmoothed trajectory (**and not deterministically**). **In all three cases, the angular velocity objective is achieved.**

8.1.2 Study on the number of unique activation patterns

For the test trajectory of the Inverted Pendulum environment (length 500), we report an average of 19.5 (+/- 12) unique activation patterns, making the number of linear equations that need to be computed and

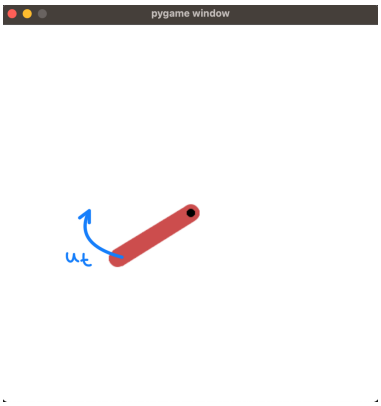


Figure 10: Saturated equilibirum point of the Inverted Pendulum environment.

saved for each activation pattern limited and computationally feasible.

8.1.3 Future directions

Further investigation would be required to find an appropiate control mechanism that can use the linear subregions created by the AL-RNN model to stabilize the system in the upright position while dealing with a restricted command space.

8.2 CartPole

The CartPole environment is an easier benchmark for control problems, that is designed to simulate simple actuators. The control command is a binary value, which is either 0 or 1 and the goal is to keep the pole upright. Since the envirnoment is designed to simulate a simple actuator, the state space is bounded to a much smaller region (see Figure 13) than the Inverted Pendulum environment and allows for a linear approximation of the control strategy.

Since the version of the AL-RNN I have been working on does not have an ordinal decoder, we decide to relax the command restrictions to allow for continous control commands. We expect that our LQR control mechanism will be able to stabilize the system to the desired goal state since the problem is highly linear.

We train two models for 3000 epochs with P=3 and P=5. While the P=3 model is not able to achieve the desired goal state (further investigation is required to understand why), the P=5 model is able to stabilize the system to the desired goal state (Figure 14 and 15).

9 Conclusion

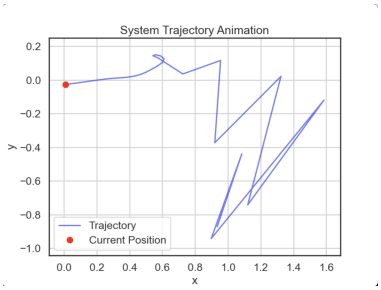
In this project we explored the use of linear control in the context of AL-RNN models. We tested the hypothesis that optimal linear control in each PWL region is a good approximation of the optimal control strategy for the system.

The results for the Inverted Pendulum environment showcase that LQR may not be an appropriate control mechanism for the system; the result of our control mechanisms produces different non-deterministic results; in many of the trials the control command saturates at the upper limit of the torque command and instead an energy based control mechanism would be required.

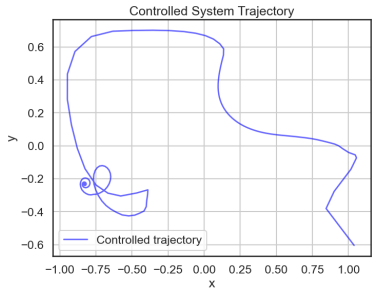
We found that the hypothesis is true for the Cart-Pole environment, where the control command is able to stabilize the system to the desired goal state. This is in part due to the almost-linear dynamics of the environment in the vecinity of the equilibrium point.

We experimented with different control hyperparameters and influence the weight and smoothness of the control command and implemented an extended AL-RNN linear control framework that we released to the public.

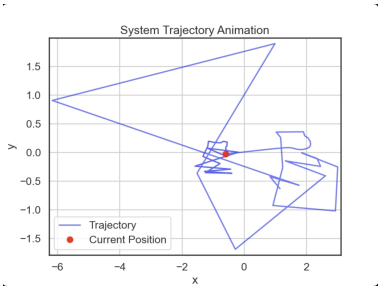
In the future, we would like to further investigate the use of linear control in the context of AL-RNN models through the use of an ordinal decoder and test different contorl mechanisms that work on restricted command spaces.



(a) Controlled trajectory without tanh smoothing and k=1.0 (P=5, Inverted Pendulum)



(b) Controlled trajectory with tanh smoothing and k=1e-2 (P=5, Inverted Pendulum)



(c) Controlled trajectory with tanh smoothing and k=0.1 (P=5, Inverted Pendulum)

Figure 11: Effect of different control gain configurations on the Inverted Pendulum system. Note that only in the un-smoothed trajectory, the position objective (0,0) is actually achieved. In all three cases, the angular velocity objective is achieved.

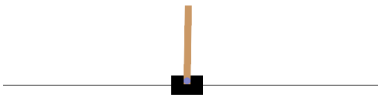


Figure 12: CartPole environment.

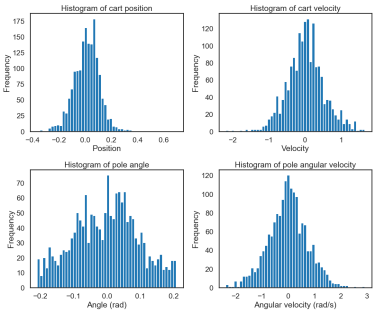


Figure 13: Histogram of the training data for the CartPole environment.

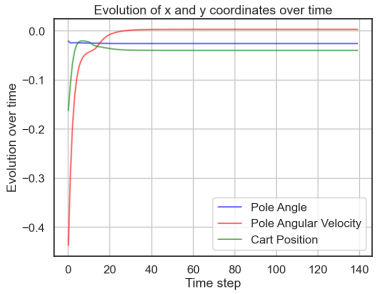


Figure 14: Example 1 of a working control trajectory for P=5.

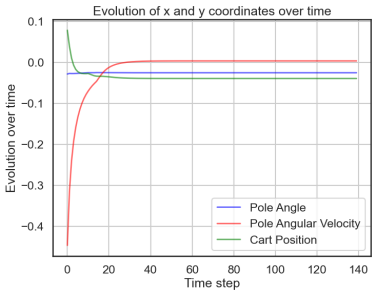


Figure 15: Example 2 of a working control trajectory for P=5.

10 Appendix

In adition to this report, I have included 2 animations of the Inverted Pendulum environment using the trained AL-RNN model with external input and LQR control that show the influence of hyperparameters on the control strategy. This control trajectories are more sharp than the ones shown in the report, caused by larger control weights.

10.1 Control framework

We designed a unified control framework that can be used to control any gymnasium environment using different control strategies, by expanding on the `DynamicsController` class. The framework is implemented in the `src/model.py` file and is used to control the Inverted Pendulum and CartPole environments. The code is shown below:

Listing 1: Control framework

```
class DynamicsController:
    def __init__(self, config, env, x_goal, control_mechanism: str = 'LQR'):
        """
        Initialize controller with a trained dynamics model

        Args:
            model: Trained AL_RNN_control model
            env: gym environment
            x_goal: goal state in observation space
            control_mechanism: control mechanism to use, currently only LQR is
                supported
        """
        ...

    def get_env_dimensions(self):
        """
        Get the dimensions of observation and action spaces for a gym environment

        Args:
            env: A gymnasium environment

        Returns:
            obs_dim: Dimension of observation space
            action_dim: Dimension of action space
        """
        ...

    def init_LQR_controller(self):
        """
        Initialize the LQR controller with control matrix B_control and weight
            matrices Q_lqr and R_lqr
        """
        ...

```

```
    def set_goal(self, goal_state):
        """
        Set the control goal state in observation space
        """
        ...

    def control_step(self, state):
        """
        Compute single control step using specified control mechanism (currently
            only LQR)

        Returns:
            control_orbit: Predicted next state
            control_commands: Control command to reach that state
        """
        ...

    def run_episode(self, max_steps, plot: bool = False):
        """
        Run an episode of the environment with the dynamics controller

        Args:
            max_steps: Maximum number of steps to simulate
            plot: Whether to plot the trajectory and control commands
        """
        ...
```

The AL-RNN control model extends the base model by incorporating external inputs and applying optimal linear control through an LQR controller. The code is shown below:

Listing 2: AL-RNN control model

```
class AL_RNN_control(AL_RNN):
    def __init__(self, M, P, N, input_dim):
        super(AL_RNN_control, self).__init__(M, P, N)
        self.input_dim = input_dim
        self.C = nn.Parameter(torch.randn(self.M, self.input_dim) * 0.1)

    def forward(self, z, external_input):
        z_unactivated = torch.clone(z)
        z[:, -self.P :] = F.relu(z[:, -self.P :])
        return self.A * z_unactivated + z @ self.W.t() + external_input @ self.C.t
            () + self.h
```

A new prediction function is implemented that uses the LQR control mechanism to predict the next state. The code is shown below:

Listing 3: Control prediction function using LQR

```
@torch.no_grad()
```

```
def predict_free_sequence_LQR(model:AL_RNN_control,x,T,B_control,Q_lqr,R_lqr,env:str='pendulum'):
    b,N=x.size()
    Z=torch.empty(size=(T,b,model.M),device=x.device)

    z=x@model.B
    z[:,0:N]=x
    Z_control_commands=torch.empty(size=(T,b,model.input_dim),device=x.device)
    Z_control_effect=torch.empty(size=(T,b,N),device=x.device)

    assert model.z_goal is not None,"z_goal is not set"
    assert model.linear_subregion_eq_dict is not None,"linear_regions_dict is not set"

    for t in range(T):
        z_prev=z.clone()
        activation_pattern=model.get_activation_pattern(z_prev)

        if tuple(activation_pattern) not in model.linear_subregion_eq_dict:
            W_subregion,h_subregion=model.get_linear_subregion_equations(activation_pattern)
            model.linear_subregion_eq_dict[tuple(activation_pattern)]=(W_subregion,h_subregion)
            print("Added new linear subregion equations for region:",tuple(activation_pattern))

        W_subregion,h_subregion=model.linear_subregion_eq_dict[tuple(activation_pattern)]

        P=(torch.from_numpy(scipy.linalg.solve_discrete_are(W_subregion.cpu().numpy(),B_control.cpu().numpy(),Q_lqr.cpu().numpy(),R_lqr.cpu().numpy())).float().to(z.device))
        assert torch.isfinite(P).any(),"P is not finite or is nan"

        K_lqr=torch.inverse(R_lqr+B_control.T@P@B_control)@(B_control.T@P@W_subregion)
        assert torch.isfinite(K_lqr).any(),"K_lqr is not finite or is nan"

        error=z_prev-model.z_goal

        u_t=-(K_lqr@error.T).T
        u_t_obs=10.0*torch.tanh(u_t)
        u_t_latent=u_t_obs@model.C.t()

        z=(z_prev@W_subregion.T)+u_t_latent+h_subregion
        Z[t]=z
        Z_control_commands[t]=u_t_obs

        z_wo_control=(z_prev@W_subregion.T)+h_subregion
        x_with_u=z@model.B.T
```

```
        x_wo_u=z_wo_control@model.B.T

        delta_x=x_with_u-x_wo_u
        Z_control_effect[t]=delta_x
    return Z.permute(1,0,2),Z_control_commands.permute(1,0,2)
```

Training and hyperparameter tuning is made available through the `train.py` script. The code is shown below:

Listing 4: Training script with command line arguments

```
def train(config: dict, env: str = "pendulum"):
    # Training configuration arguments:
    # data_path_train/test: Paths to training and test datasets
    # model_save_path: Where to save the trained model checkpoint
    # plot_save_path: Where to save training plots and visualizations
    # train_cutout: Length to truncate training data
    # P: Number of piecewise linear units in the model
    # M: Total number of units in the model
    # batch_size: Number of sequences per training batch
    # sequence_length: Length of training sequences
    # alpha: Teacher forcing ratio (0-1)
    # n_interleave: Interval between teacher forcing
    # num_epochs: Number of training epochs
    # start_lr/end_lr: Initial and final learning rates
    # batch_per_epoch: Number of batches per epoch
    # ssi: State space investigation parameter
    # T_gen: Length of generated sequences
    # T_r: Transient period cutoff
    # plotting_range: Number of timesteps in plots
    # train_sh_metric_len: Length of training metrics
    # env: Environment name (pendulum/cartpole)
    ...
```

Language	Files	Blank	Comment	Code
Jupyter Notebook	5	0	25,739	1,417
CSV	2	0	0	1,202
Julia	26	347	289	1,198
Python	12	339	252	881
TeX	4	146	0	469
JSON	7	35	0	210
Markdown	5	53	0	141
TOML	1	4	0	38
Total	62	924	26,280	5,556

Table 2: Some code statistics for the project. Quite a lot more effort than expected was put into the project.

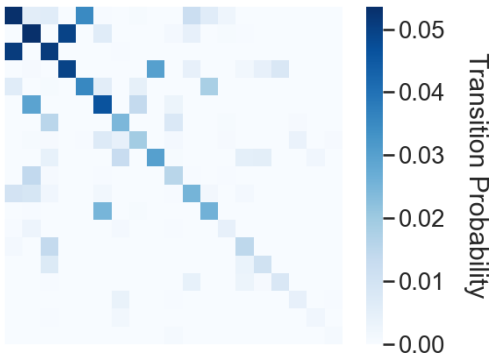


Figure 16: Transition probability matrix of the Inverted Pendulum environment using the trained AL-RNN model with external input and LQR control.