

# State Estimation and Control of Dynamic Systems from High-Dimensional Image Data

Ashik E Rasul, Hyung-Jin Yoon

Department of Mechanical Engineering,  
Tennessee Technological University, Cookeville, TN, USA  
arasul42@tntech.edu, hyoon@tntech.edu

## Abstract

Accurate state estimation is critical for optimal policy design in dynamic systems. However, obtaining true system states is often impractical or infeasible, complicating the policy learning process. This paper introduces a novel neural architecture that integrates spatial feature extraction using convolutional neural networks (CNNs) and temporal modeling through gated recurrent units (GRUs), enabling effective state representation from sequences of images and corresponding actions. These learned state representations are used to train a reinforcement learning agent with a Deep Q-Network (DQN). Experimental results demonstrate that our proposed approach enables real-time, accurate estimation and control without direct access to ground-truth states. Additionally, we provide a quantitative evaluation methodology for assessing the accuracy of the learned states, highlighting their impact on policy performance and control stability.

## Introduction

Autonomous systems operating in real-world environments, such as unmanned aerial vehicles (UAVs) and autonomous vehicles, frequently rely on actionable state information extracted from high-dimensional sensory data like images or videos. Humans possess a natural perceptual capability to interpret sequential visual cues and infer temporal properties such as velocity. However, replicating this ability in artificial systems involves two key challenges: extracting important spatiotemporal features from high-dimensional data and estimating the true state of the system from noisy observations.

Deep Neural Networks (DNNs) have consistently demonstrated superior performance over traditional computer vision methods (Grigorescu et al. 2020) in perception-based tasks for autonomous systems. Convolutional neural networks (CNNs) are widely used for extracting hierarchical spatial features from images (LeCun, Bengio, and Hinton 2015). To capture sequential dependencies in visual data, recurrent neural networks (RNNs), especially gated recurrent units (GRUs), are widely adopted due to their effectiveness in modeling temporal relationships with relatively lightweight architectures (Cho et al. 2014). Leveraging the

universal function approximation capability of fully connected neural networks (FCNNs) (Hornik 1991), the latent representations produced by GRUs can be mapped to interpretable, low-dimensional state predictions of dynamic systems, bridging the gap between high-level perception and actionable insights required for optimal control.

Conventional deep reinforcement learning (RL) algorithms, such as Deep Q-Networks (DQN), often assume full access to the environment's true state vector. While this assumption simplifies policy learning, it does not reflect real-world conditions, where direct measurement of all states is typically expensive or challenging. In such systems, the true state values must instead be estimated through processing one or multiple sensor inputs.

This research addresses these challenges by proposing a perception-based controller design framework (as shown in Figure 1) that integrates spatiotemporal feature extraction with learned state estimation. Our framework employs a CNN to extract spatial features from individual image frames. These features, combined with corresponding actions, are processed through a GRU to capture temporal dependencies. An FCNN subsequently maps these learned representations to physically meaningful, low-dimensional state estimates. By processing sequences of frames and incorporating action history, the model implicitly captures latent dynamics like velocity, filtering out irrelevant information and emphasizing task-relevant features.

The estimated states serve as inputs to a reinforcement learning (RL) agent to train an optimal policy. To the best of our knowledge, this framework represents the first approach explicitly designed to extract interpretable state information for optimal controller design from raw visual inputs. Our key contributions are summarized as follows:

- We develop a perception-based reinforcement learning framework<sup>1</sup> that extracts interpretable state predictions from high-dimensional image data.
- We demonstrate that an RL agent trained on these predicted states achieves comparable performance to an RL agent trained using full state information.
- We present a quantitative evaluation methodology to independently assess state prediction accuracy and controller performance.

<sup>1</sup><https://github.com/arasul42/cartpoleStatePred>

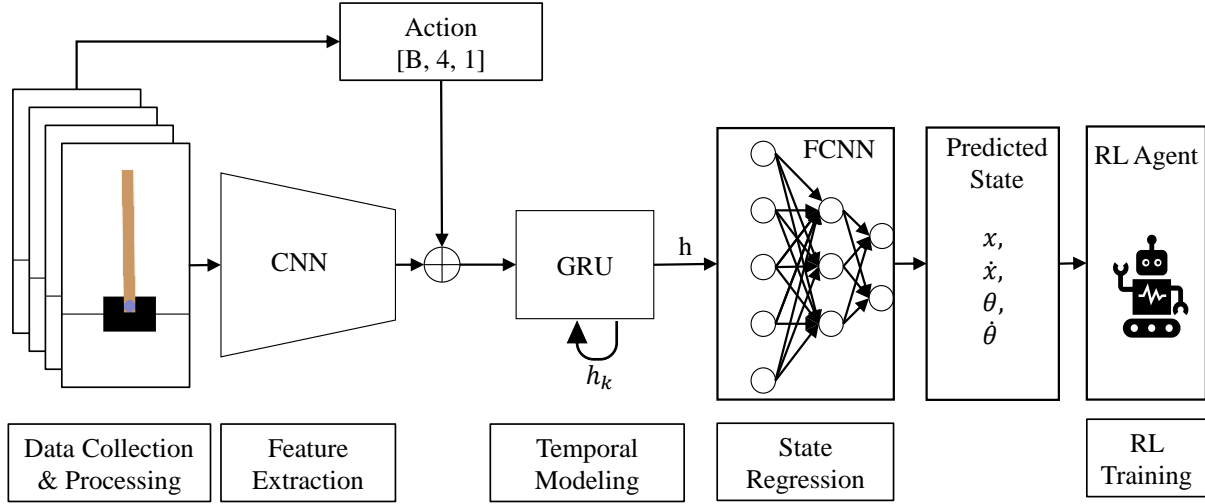


Figure 1: Proposed framework for dynamic system state prediction and RL training

## Background

Perception-based control relies heavily on compressing high-dimensional observations into low-dimensional representations that preserve task-critical temporal features. To process grid-structured data such as images, Convolutional Neural Networks (CNNs) are most widely used. They utilize convolutional layers with learnable filters to extract local and hierarchical features from raw input data, followed by non-linear activation functions like ReLU to introduce complexity into the model. CNNs with backpropagation were successfully applied to identify handwritten ZIP codes (LeCun et al. 1998). However, when the number of layers in the neural network is increased, traditional backpropagation becomes challenging. They face problems such as local optima, gradient vanishing, gradient exploding or overfitting. The introduction of ImageNet and multi-hidden-layer pre-training (Krizhevsky, Sutskever, and Hinton 2012) shows the capability of CNNs in large scale image classification tasks. However, despite their effectiveness in capturing spatial patterns, standard CNNs are limited in modeling long-range temporal dependencies in sequential data, as they lack an explicit mechanism to capture the order or temporal dynamics inherent in time-series. This shortcoming motivates the use of recurrent architectures for tasks where temporal context plays a critical role.

Traditional Fully Connected Neural Networks (FCNNs) demonstrated high accuracy in regression tasks (Chemali et al. 2018) but suffer from an explosion in the number of parameters when processing long sequences. Recurrent Neural Networks (RNNs) inherently handle sequential data better and have demonstrated superior performance compared to FCNNs. However, they face gradient vanishing and exploding issues for long input sequences (Bengio, Simard, and Frasconi 1994). To overcome these limitations, Gated RNNs, such as Long Short-Term Memory

(LSTM) and Gated Recurrent Units (GRU), introduce gating mechanisms to regulate information flow and capture long-term dependencies. LSTMs have shown exceptional performance in various domains, including time series forecasting (Siami-Namini, Tavakoli, and Namin 2019), machine translation (Sutskever, Vinyals, and Le 2014), and healthcare diagnostics (Balaji et al. 2021). However, their complex architecture and high computational cost hinder real-time deployment. To balance computational efficiency and accuracy, GRUs were developed, offering comparable performance with a simpler structure (Yang et al. 2019), making them more suitable for real-time applications.

Reinforcement learning (RL) is a branch of machine learning in which agents learn to make decisions by interacting with an environment to maximize cumulative rewards (Sutton, Barto et al. 1998). Q-learning is a widely used RL algorithm that enables agents to learn a Q-function, which estimates the expected return of taking a particular action in a given state. The introduction of the Deep Q-Network (DQN) (Mnih et al. 2015) combined Q-learning with deep neural networks, allowing agents to learn control policies directly from high-dimensional sensory inputs such as raw pixel images. DQN incorporates key innovations such as experience replay—where the agent’s experiences are stored and randomly sampled during training—and a target network, which is updated periodically to stabilize learning and prevent divergence. Conventional implementations of Deep Q-Networks (DQN) typically assume access to full state information (Raffin et al. 2021). However, in real-world scenarios, such complete observability is often unattainable due to sensor limitations or occlusions. To address this, reinforcement learning based on low-dimensional latent representations has gained significant traction (Lee et al. 2020). Despite their success, the lack of interpretability in these latent-based approaches poses a challenge for deployment in safety-critical or high-stakes applications.

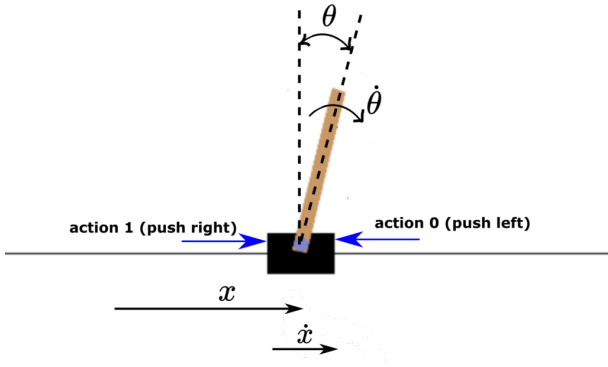


Figure 2: CartPole Environment in OpenAI Gym (Haber 2023)

### Proposed Framework

Our proposed framework as illustrated in Figure 1 has following components:

#### Data Acquisition & Preprocessing

We use the OpenAI Gym simulation environment (Brockman et al. 2016) as the data generation source for our framework. The system consists of four state variables ( $s_k$ ) at timestep  $k$ : the cart's position ( $x$ ), the cart's linear velocity ( $\dot{x}$ ), the pole's angular position ( $\theta$ ), and the pole's angular velocity ( $\dot{\theta}$ ) as shown in Figure 2. The action space comprises two discrete actions: action 0 moves the cart to the left, while action 1 moves it to the right. The agent's objective is to take actions that keep the pole within a bounded angular range  $([-0.21, 0.21])$  and the cart within a bounded displacement  $([-2.4, 2.4])$  as illustrated in Figure 2. The agent collects reward ( $r = 1$ ) for each timestep. Each episode is truncated at 500 timesteps if not terminated earlier by going out of bounds defined for  $x$  and  $\theta$ .

To generate the training samples, the agent applies random actions on the environment. To ensure balanced representation across the state space, we discretize the continuous CartPole state dimensions into bins and perform environment resets to generate samples from each bin as shown in Figure 3. The dataset is collected at a frame rate of 30 FPS with an image resolution of  $128 \times 128$  pixels using the RGB rendering mode of the simulator. Each rendered frame is re-sized and normalized by scaling pixel values to the range  $[0, 1]$  through division by 255. These preprocessed RGB image sequences, along with the corresponding action labels, are used as input for training our framework.

#### Convolutional Feature Extraction with CNN

We feed a sequence of 4 RGB image frames  $\{x_k\}_{k=1}^4$  through the CNN encoder to extract spatial features. Each image frame  $x_k \in \mathbb{R}^{3 \times 64 \times 64}$  is passed through two convolutional layers with ReLU activations and downsampling by a factor of 2 at each layer. This produces a compact feature map, which is then flattened and passed through a fully connected layer to produce a 128-dimensional feature vector:

$$f_k = \text{CNN}_\phi(x_k), \quad f_k \in \mathbb{R}^{128}, \quad (1)$$

where  $\phi$  denotes the parameters of the frame encoder.

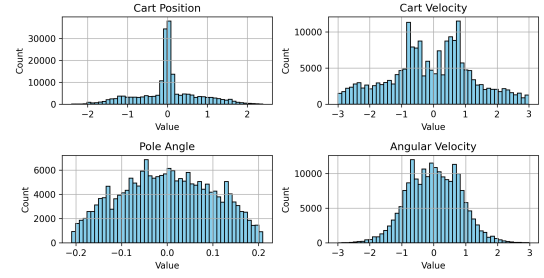


Figure 3: Dataset distribution across the state space

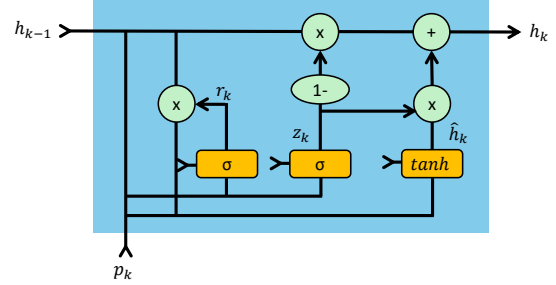


Figure 4: Gating mechanism of the GRU

### Temporal Modeling with GRU

We use a Gated Recurrent Unit (GRU) to capture temporal dependencies across the sequence of image frames and corresponding control actions. The GRU receives a sequence of 4 inputs  $\{p_k\}_{k=1}^4$ , where each  $p_k = [f_k; a_k]$  is a concatenation of the convolutionally encoded image feature  $f_k \in \mathbb{R}^{128}$  and the scalar action  $a_k \in \mathbb{R}$  at timestep  $k$ .

Following the formulation of (Chung et al. 2014), the hidden state  $h_k \in \mathbb{R}^{d_h}$  at each timestep is computed as a convex combination of the previous hidden state  $h_{k-1}$  and the candidate hidden state  $\tilde{h}_k$ , modulated by the update gate  $z_k$ :

$$h_k = (1 - z_k) \odot h_{k-1} + z_k \odot \tilde{h}_k \quad (2)$$

The update and reset gates are computed as:

$$z_k = \sigma(W_z[h_{k-1}, p_k]) \quad (3)$$

$$r_k = \sigma(W_r[h_{k-1}, p_k]) \quad (4)$$

The candidate hidden state  $\tilde{h}_k$  is defined by:

$$\tilde{h}_k = \tanh(W_h[r_k \odot h_{k-1}, p_k]) \quad (5)$$

Here,  $\sigma(\cdot)$  denotes the sigmoid activation function,  $\tanh(\cdot)$  is the hyperbolic tangent function, and  $\odot$  represents element-wise multiplication. The weight matrices  $W_z, W_r, W_h$  are learnable parameters of the GRU. The gating mechanism of GRU is illustrated in Figure 4. After processing the 4-timestep sequence  $\{p_1, p_2, p_3, p_4\}$ , the final hidden state  $h_4$  serves as a compact representation of the spatiotemporal history of observations and actions.

## State Prediction with FCNN

The final hidden state  $h_4$  is passed through a fully connected neural network (FCNN) to predict the system's physical state  $\hat{s}_{k+4} = [x, \dot{x}, \theta, \dot{\theta}]$ :

$$\hat{s}_{k+4} = \phi_{\text{fcnn}}(h_4), \quad (6)$$

where  $\phi_{\text{fcnn}}$  denotes the parameters of the regression head. The model is trained end-to-end to minimize the mean squared error (MSE) between the predicted and ground truth states:

$$\mathcal{L} = \|\hat{s}_{k+4} - s_{k+4}\|^2 \quad (7)$$

## Reinforcement Learning with DQN

If the estimated state is close to the true state, then the next state depends only on the current estimated state and action. Given data of  $(\hat{s}_k, r_k, \hat{s}_{k+1}, a_k)$ , we can estimate the action-value function. The Bellman optimality equation for the action-value function is:

$$Q^*(\hat{s}_k, a_k) = r(\hat{s}_k, a_k) + \gamma \mathbb{E} \left[ \max_{a'} Q^*(\hat{s}_{k+1}, a') \right] \quad (8)$$

where  $Q^*(\hat{s}_k, a_k)$  is the optimal action-value function,  $r$  denotes the reward,  $\gamma$  is the discount factor, and  $\hat{s}_{k+1}$  is the next estimated state.

## Evaluation Metrics

Evaluation focuses on both state estimation accuracy and RL state tracking performance with respect to a reference trajectory.

**State Estimation Accuracy:** Given the true state  $s_k = [x_k, \dot{x}_k, \theta_k, \dot{\theta}_k]$  and the estimated state  $\hat{s}_k$ , prediction accuracy is evaluated using the Root Mean Squared Error (RMSE) over  $T$  timesteps:

$$\text{RMSE} = \sqrt{\frac{1}{T} \sum_{k=1}^T \|s_k - \hat{s}_k\|^2} \quad (9)$$

To report relative accuracy, the RMSE values are normalized by the respective state variable bounds and expressed as percentages.

**RL Policy Tracking Error:** Let the desired reference state be  $s_{\text{ref}} = [0, 0, 0, 0]$ . The Mean Absolute Error (MAE) between the actual controlled state  $s_k$  and the reference trajectory is computed as:

$$\text{MAE} = \frac{1}{T} \sum_{k=1}^T |s_k - s_{\text{ref}}| \quad (10)$$

As with RMSE, the MAE values are also normalized by the respective state variable bounds and reported as percentages.

## Experiments and Results

### Training of State Prediction Model

We collect and preprocess a dataset of size 200,000. The dataset is randomly split into training and validation sets with an 80:20 ratio. We use a batch size of 32 and train the

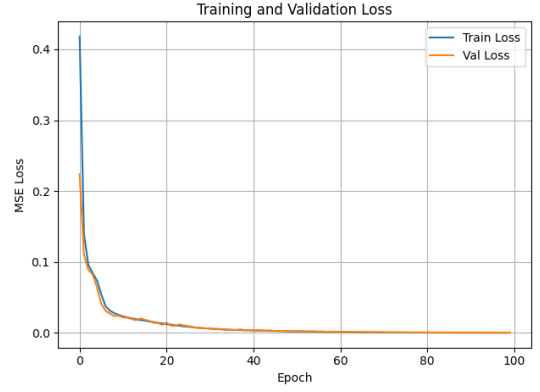


Figure 5: Training and validation loss of the prediction model

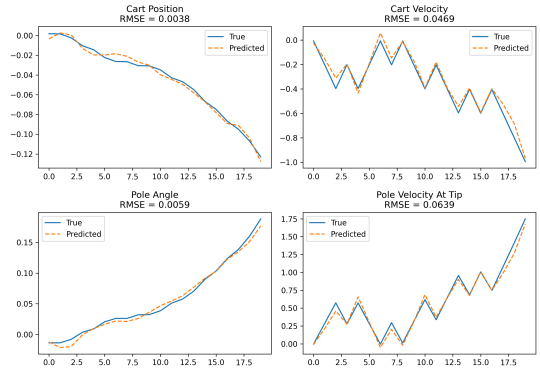


Figure 6: Evaluation of the state prediction model on a single episode

model for 100 epochs with a learning rate of  $\text{lr} = 1 \times 10^{-3}$ . We observe that both training and validation losses converge after 60 epochs, as shown in Figure 5.

To test the learned model, we generate test image frames by applying random actions in the CartPole environment. The image frames are recorded until the episode terminates due to a high pole angle ( $\theta = 0.2$ ). Although initial prediction errors are low as shown in Figure 6, the simulation timespan is too short to draw definitive conclusions. To evaluate the prediction model over a longer time horizon and measure the tracking error, we train two separate reinforcement learning (RL) models: one using full state access, and the other using the predicted state from our model.

### Reinforcement Learning with Full State Observation

We first establish a reinforcement learning (RL) baseline assuming full observability of the system. Under this assumption, the agent has access to the true state vector  $s_k = [x_k, \dot{x}_k, \theta_k, \dot{\theta}_k]$  at each timestep. We use the Deep Q-Network (DQN) algorithm, where the agent learns an approximation  $Q(s, a; \theta)$  as defined in Equation 8. The

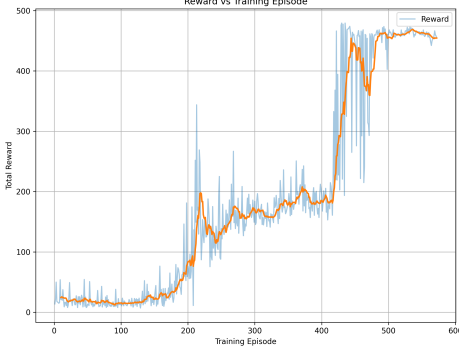


Figure 7: DQN training with full state observation

temporal-difference (TD) error is given by:

$$\delta = r_k + \gamma \max_{a'} Q(s_{k+1}, a'; \theta') - Q(s_k, a_k; \theta) \quad (11)$$

We minimize the Huber loss over a batch of experiences:

$$\mathcal{L} = \frac{1}{B} \sum_{i=1}^B \mathcal{L}(\delta_i), \quad \text{where } \mathcal{L}(\delta) = \begin{cases} \frac{1}{2} \delta^2 & \text{if } |\delta| \leq 1 \\ |\delta| - \frac{1}{2} & \text{otherwise} \end{cases} \quad (12)$$

We adopt an  $\epsilon = 2\%$ -greedy exploration strategy, where the action is selected randomly with probability  $\epsilon$  and greedily with probability  $1 - \epsilon$ . The exploration rate decays by 10% over time to encourage early exploration and later exploitation.

To enhance data efficiency, we use an experience replay buffer of size 100,000, which stores past transitions and samples minibatches uniformly during training.

The default reward function does not encourage the cart to be at the center or pole angle to be at the minimum. To encourage smoother and centered control, we augment the default reward with penalties on cart displacement, pole angle deviation, and action switching (jerk), using weights  $\lambda_1 = 0.1$ ,  $\lambda_2 = 1$ , and  $\lambda_3 = 0.35$ , respectively. The updated reward function becomes:

$$r_k = 1 - \lambda_1 x_k - \lambda_2 \theta_k - \lambda_3 \Delta u_k \quad (13)$$

We train the RL model with a learning rate  $\text{lr} = 1 \times 10^{-4}$ , discount factor  $\gamma = 0.99$ , batch size of 64, and for 100,000 timesteps. Each episode is truncated at 500 timesteps if not terminated earlier. We observe steady reward convergence after 450 episodes, as illustrated in Figure 7.

We evaluate the trained DQN agent in the same environment with access to the full state. The policy achieves stable control with low tracking errors. This evaluation serves as a performance benchmark for the dynamic system. The trajectory tracking is shown in Figure 8, an illustrative video is available here<sup>2</sup>.

<sup>2</sup><https://youtu.be/7gsOLwh1Ei0>

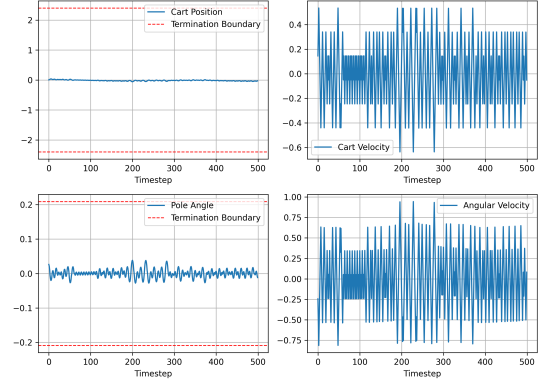


Figure 8: DQN performance with true state observation

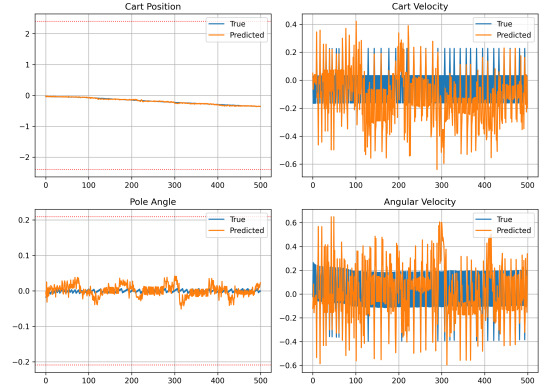


Figure 9: DQN performance with predicted state

## Reinforcement Learning with State Prediction

We then train the second DQN agent using the same hyperparameters, but with access only to the predicted state from the learned model. During evaluation, the agent is given access solely to the estimated state for decision making. We observe that the pole is controlled effectively; however, there is slight drift in cart position away from the center, as illustrated in Figure 9.

## Evaluation Results

As shown in Table 1, the prediction model achieves low RMSE across all state variables, with the cart position estimated most accurately (0.24%). Pole angle and angular velocity errors remain below 4%, indicating reliable encoding of rotational dynamics. The highest error occurs in cart velocity (3.80%), reflecting the challenge of inferring motion from visual input. Overall, the model demonstrates effective state reconstruction from image sequences.

Table 1: State Prediction Error (RMSE %)

State	$x$	$\dot{x}$	$\theta$	$\dot{\theta}$
RMSE (%)	0.24	3.80	3.89	3.28

Table 2: Tracking Error (MAE %) by RL Agents

Agent	$x$	$\dot{x}$	$\theta$	$\dot{\theta}$
Full State RL	0.53	3.99	1.60	3.90
Pred State RL	5.30	3.04	1.19	2.70

Table 2 compares tracking performance of DQN agents using full versus predicted states. As expected, the full-state agent achieves minimal error, particularly in cart position (0.53%). The predicted-state agent performs comparably on pole angle (1.19% vs. 1.60%) and even outperforms in angular velocity. However, its cart position error increases to 5.30%, aligning with the estimation gap in cart velocity. We observe the model maintains stable control, validating the use of predicted states for RL in perception based control setting.

## Conclusion and Future Work

One of the primary challenges in deploying machine learning-based control policies in real-world systems is their lack of interpretability, stemming from the black-box nature of deep models. In contrast, traditional optimal control approaches based on state estimation are both theoretically grounded and widely adopted in safety-critical applications. This work demonstrates the feasibility of bridging this gap by learning reliable and interpretable reinforcement learning (RL) policies directly from high-dimensional visual observations of dynamic systems.

A promising direction for future work is to extend the current framework using a Dynamic Autoencoder (DAE) to explicitly model the next observation frame, enabling RL training directly in the learned latent space and reducing reliance on full state prediction. Additionally, incorporating a particle filter into the inference pipeline could help account for process and observation noise, improving robustness under real-world uncertainty.

## Use of AI in Writing

In this document, AI tools were used to check grammar, to ensure L<sup>A</sup>T<sub>E</sub>X formatting, to check sentence coherence and to understand the implementation better. However, the methodology development, literature review, experiment design and evaluation plan were manually developed based on the review of academic sources.

## References

Balaji, E.; Brindha, D.; Elumalai, V. K.; and Vikrama, R. 2021. Automatic and non-invasive Parkinson’s disease diagnosis and severity rating using LSTM network. *Applied Soft Computing*, 108: 107463.

Bengio, Y.; Simard, P.; and Frasconi, P. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2): 157–166.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. Openai gym. *arXiv preprint arXiv:1606.01540*.

Chemali, E.; Kollmeyer, P. J.; Preindl, M.; and Emadi, A. 2018. State-of-charge estimation of Li-ion batteries using deep neural networks: A machine learning approach. *Journal of Power Sources*, 400: 242–255.

Cho, K.; Van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; and Bengio, Y. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Chung, J.; Gulcehre, C.; Cho, K.; and Bengio, Y. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Grigorescu, S.; Trasnea, B.; Cocias, T.; and Macesanu, G. 2020. A survey of deep learning techniques for autonomous driving. *Journal of field robotics*, 37(3): 362–386.

Haber, A. 2023. Cart-Pole Control Environment in OpenAI Gym/Gymnasium – Introduction to OpenAI Gym. Blog post. Accessed: Feb 24, 2025.

Hornik, K. 1991. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2): 251–257.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 25, 1097–1105.

LeCun, Y.; Bengio, Y.; and Hinton, G. 2015. Deep learning. *Nature*, 521(7553): 436–444.

LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278–2324.

Lee, A. X.; Nagabandi, A.; Abbeel, P.; and Levine, S. 2020. Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model. *Advances in Neural Information Processing Systems*, 33: 741–752.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533.

Raffin, A.; Hill, A.; Gleave, A.; Kanervisto, A.; Ernestus, M.; and Dormann, N. 2021. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*, 22(268): 1–8.

Siarni-Namini, S.; Tavakoli, N.; and Namin, A. S. 2019. The performance of LSTM and BiLSTM in forecasting time series. In *2019 IEEE International conference on big data (Big Data)*, 3285–3292. IEEE.

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27.

Sutton, R. S.; Barto, A. G.; et al. 1998. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Yang, F.; Li, W.; Li, C.; and Miao, Q. 2019. State-of-charge estimation of lithium-ion batteries based on gated recurrent neural network. *Energy*, 175: 66–75.