# Policy Learning for an Unbalanced Disc

Design Assignment , 5SC28 Machine Learning for Systems and Control

Sadra Moosavi Lar
*Dept. of Electrical Engineering*
*Eindhoven University of Technology*
Eindhoven, NL
s.s.moosavi.lar@student.tue.nl

***Abstract*— We study data-driven modeling and policy learning for the unbalanced disk, an underactuated inverted-pendulum benchmark with saturated actuation (±3 V). First, we identify the dynamics from input–output data using two families of models: (i) sparse variational Gaussian processes (SVGP) with ARD-RBF kernels and sin/cos angle features to handle periodicity, and (ii) artificial neural networks in NARX (MLP) and sequence (LSTM) forms. Models are trained for one-step prediction and validated by free-run simulation with a short warm-up; hyperparameters are tuned with Optuna. Across the benchmark, advanced LSTM models achieved the lowest errors (e.g., prediction RMSE ≈ $2.85\times10^{-3}$ rad; simulation RMSE ≈ $2.23\times10^{-2}$ rad), with SVGP competitive in one-step prediction but somewhat behind in long-horizon rollouts. Second, we address control via reinforcement learning. We implement a value-based baseline (DQN with discretized torques) for swing-up and an actor–critic approach (SAC) for continuous control. To avoid hand-crafted rewards, we learn an inverse-RL reward by logistic classification over hand-engineered state–action features from expert demonstrations; this reward is then used to train two SAC policies: one for upright stabilization and a second for reference tracking around the top equilibrium. Policies and models are selected by validation performance and exported in checker-compatible formats. Results demonstrate that modern sequence models provide strong identification accuracy and that SAC trained with learned rewards yields robust swing-up, stabilization, and reference tracking on this nonlinear benchmark.**

***Keywords—reinforcement Learning, Unbalanced Disc***

## I. Introduction

The unbalanced disc is a canonical inverted-pendulum system: a disk with an off-center mass that behaves like a pendulum driven by a DC motor. The design objective is twofold: first, to swing the disc from its stable bottom position to upright; second, to keep it balanced at the top while rejecting disturbances—tasks that capture the essence of many self-balancing devices (e.g., Segways and hoverboards)

We model the unbalanced disc as a planar pendulum driven by a voltage-limited DC motor. The continuous-time dynamics used throughout identification and control are:

$$\dot{\theta}(t) = \omega(t),$$

$$\dot{\omega}(t) = -\omega_0^2 \sin(\theta(t)) - \gamma\,\omega(t) - F_c\,\mathrm{sign}(\omega(t)) + K_u\,u(t)$$

Here, omega_0 denotes the natural frequency, gamma the viscous-friction coefficient, Fc the Coulomb-friction level, and Ku the input gain. For data-driven identification we use NARX or sequence mappings that rely only on the input u and past angle measurements theta:

1. a Gaussian-process model (sparse variational GP with ARD-RBF kernel) that maps past inputs and sin/cos angle features to the next angle, and
2. neural baselines—an MLP-NARX and an LSTM—trained for one-step prediction and validated by free-run simulation after a short warm-up.

Policy learning treats control as reinforcement learning with full-state feedback (theta and omega). We implement a value-based method (DQN with discretized actions) for swing-up and an actor–critic method (SAC) for continuous control. To avoid hand-crafted reward shaping, we also learn a reward via inverse RL: a logistic classifier over hand-crafted features of theta, omega, and u trained on expert trajectories. The learned reward replaces the native swing-up reward and is used to train two SAC policies: one specialized for upright stabilization and one for reference tracking.

Hyperparameters for models and policies are tuned with Optuna, and performance is reported using one-step prediction error, free-run simulation RMSE, and episodic return of the learned controllers.

## II. Gaussian Processes

Gaussian Processes (GPs) are Bayesian nonparametric models that place a prior over functions and use a covariance (kernel) to encode smoothness, scales, and correlations in the data. Conditioning this prior on observed input–output pairs yields a posterior whose mean serves as a predictor and whose variance quantifies epistemic uncertainty. In our setting, the "inputs" are compact histories of the disc—past voltage commands and past angles—and the "output" is the next angle. Concretely, we adopt a NARX map with sin/cos angle embeddings (to respect periodicity) and standardize all regressors. For scalability we use a Sparse Variational GP (SVGP): inducing points are initialized by k-means on the training inputs; the kernel is ARD-RBF to learn separate relevance/length-scales per regressor; and the model is trained by maximizing the variational ELBO with Adam.

Evaluation follows two complementary tasks. First, one-step prediction on a held-out split, scored by a wrapped-angle RMSE (to avoid $2\pi$ discontinuities). Second, free-run simulation: the GP iteratively feeds back its own angle predictions (after a short warm-up from ground truth) to assess closed-loop rollouts. The results are presented in Table 1 and 2.

TABLE I. Simulation Results for Gaussian Processes

| *Type* | *RMSE (Rad)* | *RMSE (deg)* |
|---|---|---|
| Lower Bound | 0.0195 | 1.12 |
| Good NN model | 0.0271 | 1.55 |
| **Sparce GP Approach** | **0.0509** | **2.91** |
| Linear Model | 0.255 | 14.6 |

TABLE II. Prediction Results for Gaussian Processes

| *Type* | *RMSE (Rad)* | *RMSE (deg)* |
|---|---|---|
| Good NN model | 0.00382 | 0.219 |
| **Sparce GP Approach** | 0.00451 | 0.258 |
| Linear Model | 0.00665 | 0.381 |

In simulation (Table I), lower is better. The "Lower Bound" line—likely a noise floor/oracle reference—sets the best achievable RMSE (0.0195 rad). Among learned models, the Good NN is the most stable in free-run (0.0271 rad), the Sparse GP (SVGP) is clearly behind (0.0509 rad), and the Linear model degrades severely (0.255 rad). This gap highlights two points: (i) long-horizon rollouts amplify small one-step biases; (ii) the dynamics are strongly nonlinear, so linear IO models are inadequate for closed-loop use.

In one-step prediction (Table II), the ordering is the same but the margins shrink: Good NN (0.00382 rad) ≈ best, Sparse GP close (0.00451 rad), Linear worst but still reasonable (0.00665 rad). That the Sparse GP is near-parity in prediction yet lags in simulation suggests mild underfitting/over-smoothing (e.g., too few inducing points or a stationary kernel) and exposure bias: it was trained on ground-truth histories but evaluated by feeding back its own outputs.

Overall: both NN and SVGP capture next-step behavior well, but for control-relevant rollouts the NN is currently more robust. Likely GP improvements include more inducing points, non-stationary/periodic kernels, sin–cos output parametrization, or training with a multi-step/rollout loss; any of these should narrow the simulation gap without sacrificing GP's uncertainty advantages.

We tune key hyperparameters—history lengths (na, nb), number of inducing points (M), learning rate, and epochs—using Optuna, selecting the configuration that minimizes validation wrapped-RMSE (and cross-checking free-run error). This GP pipeline delivers a data-efficient, uncertainty-aware model of the unbalanced disc suitable for both prediction and downstream control design. {'na': 10, 'nb': 8, 'M': 100, 'lr': 0.0144, 'epochs': 100, 'batch_size': 512}

## III. Artificial Neural Networks

The goal of our ANN models is to learn a data-driven surrogate of the disk's input–output dynamics that maps recent inputs/outputs to the next angle, then remain stable when rolled out in free-run simulation. We focus on two complementary architectures. NARX (Nonlinear AutoRegressive with eXogenous inputs) uses a feed-forward MLP on a fixed window of lagged signals (past motor voltages and past angles, typically encoded as sin/cos to avoid angle wrap issues). This makes NARX simple, fast to train, and easy to tune, with clear control over the memory length via the lag orders. LSTM (Long Short-Term Memory) treats the trajectory as a sequence and learns its own internal state, which can capture longer and variable-length dependencies beyond a fixed lag window. In practice, we train both for one-step prediction and assess closed-loop robustness by free-run simulation; NARX offers strong performance with low complexity, while LSTM can better model longer-range effects at the cost of more parameters and training time.

### *A. Simple Approach: MLP + NARX*

Our MLP + NARX approach treats the unbalanced disk as a nonlinear autoregressive system with exogenous input. Concretely, the regressor stacks a fixed window of past motor voltages and past angles; to mitigate wrap-around discontinuities we primarily encode the angle with $\sin\theta$ and $\cos\theta$ (converted back to $\theta$ for evaluation/exports). Inputs are standardized with train statistics, and a feed-forward MLP (hidden layers/units and activation tuned) maps the regressor to the next output. We train on one-step prediction using PyTorch with an angle-aware loss (MSE on $[\sin\theta,\cos\theta]$ plus a small unit-norm penalty); early stopping and Optuna are used to tune na, nb (lag orders), depth/width, learning rate, and batch size.

Closed-loop fidelity is assessed by free-run simulation with a 50-sample warm-up, in addition to one-step RMSE on held-out data, and both metrics are reported in the submission-checker format. Implementation relies on NumPy/PyTorch (modeling and simulation) and Optuna (hyperparameter search); [1]a linear IO model and an LSTM serve as baselines. As summarized in the table 3 and 4, MLP–NARX attains low one-step errors and stable rollouts, clearly outperforming the linear baseline and remaining competitive with sequence models at substantially lower computational cost (see the ANN results table).Here we explain our approach MLP + NARX is, we explain our approach tools we used and refer to the results in table.

TABLE III. Simulation Results for Simple Approach

| *Type* | *RMSE (Rad)* | *RMSE (deg)* |
|---|---|---|
| Lower Bound | 0.0195 | 1.12 |
| Good NN model | 0.0271 | 1.55 |
| **Simple NN Approach** | **0.0546** | **3.13** |
| Linear Model | 0.255 | 14.6 |

TABLE IV. Prediction Results for Simple Approach

| *Type* | *RMSE (Rad)* | *RMSE (deg)* |
|---|---|---|
| **Simple NN Approach** | **0.00322** | **0.184** |
| Good NN model | 0.00382 | 0.219 |
| Linear Model | 0.00665 | 0.381 |

In one-step prediction (Table IV), the Simple NN Approach attains the best RMSE (0.00322 rad), improving on the "Good NN model" (0.00382 rad) by ≈16% and clearly beating the linear baseline (0.00665 rad). This indicates the simple regressor captures the local input–output map very well.

In free-run simulation (Table III), the ordering flips: the Good NN model (0.0271 rad) is close to the Lower Bound (0.0195 rad), while the Simple NN degrades to 0.0546 rad—about 2× the Good NN's error—though it still outperforms the linear model by ~79% (0.255 → 0.0546 rad). The gap reflects

[1] Optuna training best config for simple approach: na=8, nb=8, hl=2, hn=32, act=relu, lr=0.00069272, bs=128, epochs=160

error compounding: models tuned purely for one-step accuracy can drift when rolled out, whereas architectures/regularization in the Good NN (e.g., better lag choice, angle encoding, capacity, or rollout-aware validation) yield more stable closed-loop behavior.

## *B. Advanced Approach: LSTM + NARX*

A Long Short-Term Memory (LSTM) network is a recurrent neural architecture designed to model sequences with long-range dependencies via gated state updates. Unlike feed-forward NARX, which consumes a fixed window of regressors, an LSTM maintains an internal memory that can propagate information across many time steps—well-suited to nonlinear, history-dependent dynamics such as the unbalanced disk.

We train a sequence-to-one LSTM that ingests a window of past inputs/outputs and predicts the next angle. Inputs are the time series [uk,θk] the target is θk+1]

. Models are fitted on one-step prediction loss and then evaluated both as (i) one-step predictors and (ii) free-run simulators by rolling predictions forward from a short warm-up. We tune sequence length, hidden width, number of layers, and bidirectionality via a small grid, selecting the configuration that minimizes validation error while remaining stable in rollout: seq_len=15, hs=64, nl=2, bi=True, lr=0.00077154, bs=256, epochs=120

TABLE V. SIMULATION RESULTS FOR ADVANCED APPROACH

| *Type* | *RMSE (Rad)* | *RMSE (deg)* |
|---|---|---|
| Lower Bound | 0.0195 | 1.12 |
| **Advanced NN Approach** | **0.0223** | **1.28** |
| Good NN model | 0.0271 | 1.55 |
| Linear Model | 0.255 | 14.6 |

TABLE VI. PREDICTION RESULTS FOR ADVANCED APPROACH

| *Type* | *RMSE (Rad)* | *RMSE (deg)* |
|---|---|---|
| **Advanced NN Approach** | **0.00285** | **0.163** |
| Good NN model | 0.00382 | 0.219 |
| Linear Model | 0.00665 | 0.381 |

The LSTM attains the best one-step accuracy and nearly reaches the simulation lower bound, outperforming both the strong feed-forward "Good NN" and the linear baseline. The improvement in simulation (0.0223 rad vs 0.0271 rad) indicates reduced error compounding during rollout—consistent with an LSTM's ability to internalize multi-step dynamics beyond a fixed regressor horizon. Linear models remain inadequate given the system's strong nonlinearity.

## IV. Q-LEARNING

Deep Q-Networks replace the table with a neural network Qθ (s,a). Two stabilizers are key: (i) experience replay, which breaks temporal correlations by training on a uniformly (or prioritised) sampled buffer of past transitions, and (ii) a target network Qθ whose parameters are updated slowly (hard copy every target_update_interval). In our unbalanced-disk setup, the state is (θ,ω) and the true action is a continuous voltage u ∈[−3,3]. To use DQN we discretize the torque into N evenly spaced bins and learn Q (θ,ω,ai) for each bin ai

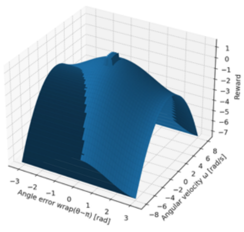
We trained an MLP (ReLU, 2–3 hidden layers) with Adam, ε-greedy exploration scheduled via exploration_fraction and exploration_final_eps, a replay buffer, and periodic evaluation through an EvalCallback. We wrapped the environment with TimeLimit (episode cap) and Monitor (logging) and tuned key hyperparameters (learning rate, batch size, target update, number of bins, network depth/width) with Optuna.

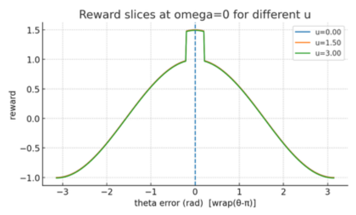
## *A. Reward Design*

For swing-up and stabilization we used a physically motivated shaped reward centered on the upright equilibrium. The core term is a smooth potential bonus based on the wrapped angle error to the top, complemented by penalties on angular speed (to discourage persistent spin) and on control effort (to reduce chattering).

$$r(\theta,\omega,u) = \cos\big(\mathrm{wrap}(\theta-\pi)\big) \;-\; k_\omega\,\omega^2 \;-\; k_u\Big(\frac{u}{u_{\max}}\Big)^2 \;+\; b\,\mathbf{1}\big(\,|\mathrm{wrap}(\theta-\pi)| < \theta_{\mathrm{tol}} \;\wedge\; |\omega| < \omega_{\mathrm{tol}}\,\big).$$

A small upright bonus within a few degrees of the top encourages settling. This design yields a dense, informative signal across the state space, avoids discontinuities at ±π by wrapping the angle, and balances the trade-off between energy injection during swing-up and minimal actuation near the top. (For separate experiments we also learned an IRL reward from expert logs and swapped it into the same training loop, but the DQN experiments reported here use the native shaped reward.)





Similar to previous parts, We framed tuning as maximizing the evaluation return reported by EvalCallback. Optuna sampled: learning rate (log-scale), exploration schedule (exploration_fraction, exploration_final_eps), discount factor, target-update interval, batch size, number of action bins, and network depth/width ("medium/large/huge" templates). A MedianPruner stopped trials that underperformed the running median at intermediate checkpoints, freeing compute for promising configurations. This produced a compact search (dozens, not hundreds, of trials) that quickly isolated a stable region of hyperparameters; notably, learning rate and target-update cadence were the most sensitive, followed by action-bin count. We retained per-trial checkpoints, selected the best by validation return, and exported the final policy artifact for downstream evaluation. Best hyperparameters: {'learning_rate': 0.00193, 'exploration_fraction': 0.279, 'exploration_final_eps': 0.0011, 'discount_factor': 0.962, 'target_update_interval': 1000, 'batch_size': 512, 'n_actions': 21, 'net_arch_style': 'large'}

## V. ACTOR CRITIC

We adopt Soft Actor–Critic (SAC) for policy learning because it directly handles the continuous motor command

and encourages smooth control via maximum-entropy RL. SAC parameterizes a stochastic Gaussian policy (actor) and two action-value functions (twin critics), updates critics by TD targets, and adjusts the entropy weight to balance exploration and exploitation. Observations are the full state (θ,ω); the action is a clipped scalar torque $u \in [-3,3]$ V. Episodes are time-limited (600 steps, $\Delta t=0.025$ s). Hyperparameters (learning rate, $\tau$, $\gamma$, buffer/batch sizes, network width) are tuned with Optuna using short studies (4 trials, large eval_freq) to respect wall-clock constraints.

## A. *No Reference Tracking*

To remove reward-shaping bias, we learn the reward by inverse RL from a single expert log containing (t,θ,ω,u). The log were created using a game in which we could adjust the input voltage using arrow keys and try to balance the disc. We fit a linear logistic discriminator on hand-crafted features of (θ,ω,u), normalized by expert statistics; the shaped reward used by SAC is the discriminator logit:

$$r(s,a) = \mathbf{w}^{\top}\,\widehat{\boldsymbol{\phi}}(s,a) + b$$

This reward captures the swing-up and balancing behavior visible in the demonstrations without us hand-tuning penalties on angle, spin, or control effort. SAC trained on this learned reward yields a single "upright" policy (sac-model-upright.zip) that consistently swings up from near-bottom and maintains the disc near the upright with modest torque usage. Optuna searches were kept small (4 trials; $6\times10^4$ steps/trial; evaluation every $2\times10^4$ steps) to finish under the time budget, and still provided stable policies.

## B. *With reference tracking*

In the last part of the assignment, we repeat the same IRL →SAC pipeline using the expert log recorded under reference tracking, which includes columns (t,θ,ω,u,θref,track_err). In our implementation the IRL features use (θ,ω,u); the distinction comes from the different state–action distribution in the tracking demonstrations, so the learned reward implicitly favors trajectories that oscillate +-15 degree around the upright as in the expert.

Training SAC on this reward produces a "ref" policy (sac-model-ref.zip) that reaches the top and is optimized to follow the reference within the +- 15 deg band around upright, trading slightly higher control activity for improved tracking. As with the upright case, hyperparameters are selected by a short Optuna study to keep runtime low while preserving reliability.

## VI. Acknowledgment

We have used various GenAI tools to improve the learning rate of various topics and optimize the workflow in code review, commenting, summarizing and so on. Our usage was in accordance with TU/e AI regulations (released on November 2024) and in full compliance with the learning objectives of this assignment.