# Exploring Unbalanced Disk Dynamics: Model Identification and Non-Linear Control Policy Learning

1st Wenyu Song
*dept. name of organization (of Aff.)*
*name of organization (of Aff.)*
Eindhoven, Netherlands
w.song@student.tue.nl

2nd Kevin Laemers
*dept. name of organization (of Aff.)*
*name of organization (of Aff.)*
City, Country
email address or ORCID

3rd Ryo Sugimura
*dept. name of organization (of Aff.)*
*name of organization (of Aff.)*
City, Country
email address or ORCID

4th Marten Hanegraaf
*dept. name of organization (of Aff.)*
*name of organization (of Aff.)*
City, Country
email address or ORCID

***Abstract*—This document is a model and instructions for LaTeX. This and the IEEEtran.cls file define the components of your paper [title, text, heads, etc.]. *CRITICAL: Do Not Use Symbols, Special Characters, Footnotes, or Math in Paper Title or Abstract.**

***Index Terms*—component, formatting, style, styling, insert**

## I. Introduction

The study of the inverted pendulum system has garnered significant attention due to its wide-ranging applications in various fields such as unicycles and hoverboards. The primary challenge of this system lies in its inherent instability, necessitating precise control to maintain equilibrium. This assignment paper focuses on an analogous system represented by an unbalanced disc, as shown in Fig.1, which serves as a practical model for the inverted pendulum system.

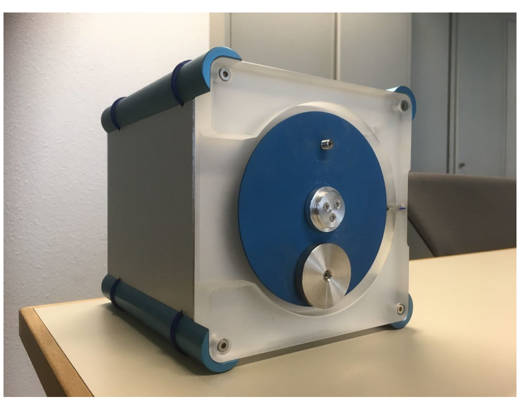

Fig. 1. Unbalanced disc set up

The unbalanced disc system is actuated by voltage, with the disc's angular position, $\theta$, being a function of the applied voltage, $u$. The relationship between $u$ and $\theta$ is governed by the following equations:

$$\dot{\theta}(t) = \omega(t) \tag{1}$$

$$\dot{\omega}(t) = -{\omega_0}^2 sin(\theta(0)) - \gamma\omega(t) - F_c \text{sign}(\omega(t)) + K_u u(t) \tag{2}$$

This paper aims to explore the system dynamics of the unbalanced disc system using a combination of Artificial Neural Networks (ANN) and Gaussian Processes (GP) in conjunction with Nonlinear AutoRegressive with eXogenous inputs (NARX) models. The objective is to identify the system dynamics accurately and subsequently develop an effective policy for system control.

## II. System Identification

Two methods are introduced in order to identify the unbalanced disk system from the given dataset of voltage input and angle output. These two methods include Gaussian Processes (GP) and artificial neural networks (ANN).

### A. Gaussian Process

Besides form ANN, a probability-based non-parametric model called Gaussian Process can also be used to model functions.

*1) Gaussian Process:* The Gaussian Process (GP) assumes that the relationship between data points obeys a multivariate Gaussian distribution, and the core idea of GP is to describe the distribution of the function through the mean function and the covariance function. Specifically, a Gaussian process is defined by the following two parts:

$$\begin{aligned} m(x) &= \mathbb{E}\{f(x)\} \\ k(x,\tilde{x}) &= \mathbb{E}\{(f(x)-m(x))(f(\tilde{x})-m(\tilde{x}))\} \end{aligned} \tag{3}$$

The covariance function $k(x, \tilde{x})$, e.g. kernel function, is the most crucial part in GP. Different kernel function can capture different types of relationships. Given a set of observed data, Gaussian Process can be used for inference to obtain the conditional distribution of the target function. By calculating the posterior distribution, we can obtain predictions of function values and corresponding uncertainties given the observed data.

*2) NARX model:* NARX (Nonlinear AutoRegressive with eXogenous inputs) is a non-linear autoregressive model commonly used for time series modelling and prediction. The NARX model captures the non-linear relationship in time series data while considering the influence of exogenous inputs.

The general form of the NARX model can be expressed as

$$y(t) = f(y(t-1), y(t-2), ..., y(t-n), u(t), u(t-1), ..., u(t-m)) \tag{4}$$

The key to the NARX model is to define the function $f$, which describes the relationship between the independent variable (output) and past observations and external inputs. The function $f$ can be any nonlinear function, such as a polynomial function, neural network, etc. Choosing an appropriate function form and model structure is the key to building an effective NARX model.

By combining NARX and Gaussian Process regression, NARX GP can capture non-linear dynamics and temporal dependencies using the NARX structure while providing probabilistic predictions and uncertainty estimates using the Gaussian Process regression framework. This hybrid model is particularly useful when dealing with time series data with complex non-linear relationships and uncertain dynamics.

*3) Algorithm:* In this situation, we used NARX GP and managed to identify the system dynamics according to the given data of input voltage $u$ and output angle $\theta$. The basic logic is shown in Algorithm 1 below .

**Algorithm 1** NARX GP

0: Load training data from file.
0: Set the voltage data ($X_N$) as the input (**ulist**) and the motor speed data ($Y_N$) as the target output (**ylist**).
0: Define a non-linear function $f$ that takes past inputs and outputs as parameters and returns the predicted output.
0: Define a function which uses the NARX model to simulate the target output sequence based on the given input sequence and the function $f$.
0: Define a function, which creates the training data set from the given input and target output sequences. This function combines past input and output values into feature vectors and uses the current target output as labels.
0: Create a Gaussian Process regression model (**reg**) using **GaussianProcessRegressor** and the specified kernel function.
0: Fit the training set to the model and estimate the parameters of the Gaussian Process model by calling function **reg.fit**.
0: Use the fitted model to predict the training set and validation set, and calculate the standard deviation of the predicted results. =0

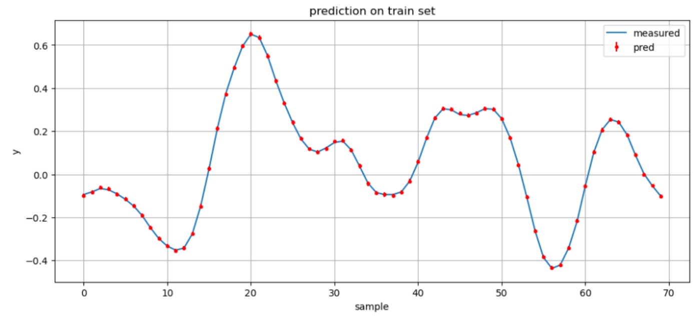


Fig. 2. Prediction on training set

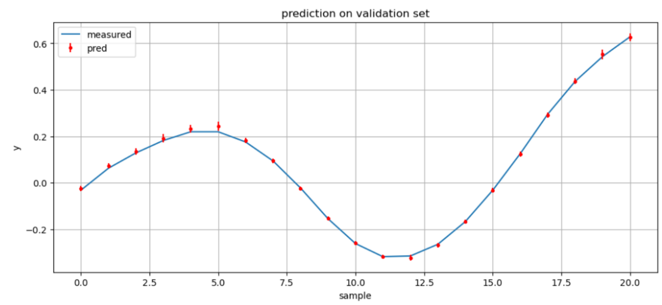


Fig. 3. Prediction on validation set

*4) Result:* As shown in Fig2 and 3, a small uncertainty indicates a successful Gaussian process, implies that the GP model fits the data well and provides reliable predictions.

### *B. Artificial Neural Network*

*1) Neural Network:* An artificial neural network (ANN) is a black box system that can be used for approximation of both linear and non-linear dynamics of systems. An artificial neural network is connected by nodes which calculate based on the input, weight and biases. The output from this is fed into the non-linearity function. The equation for a node is shown in the equation below.

$$y = \sigma(W^T x + b) \tag{5}$$

The inputs are multiplied by the weights and bias is added. The output of that is fed into a function. For the structure, a multi-layer perceptron with the non-linearity function was used. The base structure of the model was decided on further research of system identification using neural networks. However, since the system dynamics are different to the one in the paper, further hyperparameter research was done. The relation between the number of hidden layers and identification loss was further researched to finalize the structure of the model. The Nonlinear Autoregressive with exogenous inputs (NARX), non-linear state space and recurrent neural networks (RNN) were considered using the ANN methods.

As mentioned before II-A2, the NARX model can be used in conjunction with ANN where the functions can be approximated using the nodes and hidden layers. For the NARX model, the $n$ and $m$ mentioned previously are hyperparameters that can be tuned. These hyperparameters are looked further into in the form of a grid search. The combination of the parameters in a certain range is taken to see how it affects the loss.

Along with the NARX model, we incorporated a non-linear state-space model which is another way of describing the dynamics of a system. The state-space models are given by the equations below.

$$\dot{x} = f(x, u) \tag{6}$$

$$y = g(x, u) \tag{7}$$

The y, x and u correspond to the output state, input state and action respectively. The $f$ and $g$ are the non-linear functions that express the relation between the given input state/action to each output. The number of states and actions that are the input for state-space models is another kind of hyperparameter and has to be optimized.

*2) Results:*

## III. Swing-up policy

This section outlines the methodology used to solve the pendulum swing-up problem using a Deep Q-Network (DQN) approach and an Advantage Actor Critic (A2C) approach. Both DQN and A2C agent's architecture, learning process, and the environment setup are described in detail.

### A. Deep Q-Network Architecture

The DQN agent comprises two primary components: the Q-Network and the target network, both of which are implemented as neural networks. The Q-Network is responsible for approximating the Q-value function, which represents the expected future rewards for each action in each state. The target network, on the other hand, is used to generate the target Q-values for training the Q-Network. The architecture of these networks includes two fully connected layers with 64 neurons each, followed by a dropout layer to prevent overfitting. The output layer of the network has a dimension equal to the number of possible actions. The Rectified Linear Unit (ReLU) function is used as the activation function for the neurons.

### B. Learning Process

The DQN agent learns by interacting with the environment and storing the experiences in a replay buffer. Each experience is a tuple that includes the current state, the action taken, the reward received, and the next state. The agent samples a batch of experiences from the replay buffer and learns by adjusting its Q-Network parameters to minimize the difference between the predicted Q-values and the target Q-values. This difference, known as the temporal difference error, is calculated using the mean squared error loss function. The learning process also involves a technique known as soft updates to update the parameters of the target network, which helps to stabilize the learning process.

**Algorithm 2** Enhanced Deep Q-Network with Experience Replay and Dropout

Initialize replay memory $D$ with capacity $N$
Initialize Q-network $Q$ with weights $\theta$, adding dropout layers
  for prevention of overfitting
Initialize target network $\hat{Q}$ with weights $\theta^- = \theta$
**for** episode = 1 to $M$ **do**
  Initialize state $s$
  Set $\epsilon$ according to decay strategy
  **for** time-step = 1 to $T$ **do**
    With prob. $\epsilon$ select random action $a$, else $a = \arg\max_a Q(s, a; \theta)$
    Execute action $a$ in emulator and observe reward $r$, next state $s'$
    Store experience tuple $(s, a, r, s')$ in $D$
    Sample random mini-batch of tuples $(s, a, r, s')$ from $D$
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(s', a'; \theta^-) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(s, a; \theta))^2$
      with respect to $\theta$, using mixed precision training
    Every $C$ steps perform a soft update on $\hat{Q}$ with
      $\tau * Q + (1 - \tau) * \hat{Q}$
    $s = s'$
  **end for**
**end for**=0

### C. Advantage Actor Critic

The actor and critic are implemented as separate neural networks, each with two hidden layers of 64 neurons and a dropout layer for regularization. The actor network outputs a probability distribution over actions, while the critic network outputs a value estimate.

The agent uses the actor network to select actions and the critic network to evaluate them. The agent learns from its experiences by storing them in a replay buffer and sampling mini-batches to update the networks. The actor network is updated to maximize the expected return, and the critic network is updated to minimize the difference between its value estimates and the actual returns.

The agent uses a soft update strategy to update the target networks, which involves slowly blending the weights of the target networks with the weights of the main networks. This strategy helps to stabilize learning by making the target values for the updates more consistent.

The agent is trained for a maximum of 5000 episodes, with each episode lasting up to 3000 time steps. The environment is considered solved when the agent achieves an average score of 2560 over 100 consecutive episodes.

### D. Environment Setup

The agent is trained in a simulated environment that models the dynamics of the pendulum. The state space of this environment includes the angle and angular velocity of the pendulum. Using these parameters, a reward function is constructed as follows:

**Algorithm 3** Advantage Actor-Critic (A2C)

```
Initialize actor network with weights θ and critic network with weights φ
Initialize memory D with capacity N
Initialize batch size B, discount factor γ, and learning rate α
Initialize optimizer for actor and critic with learning rate α
for each episode do
    Initialize state s
    for each time step do
        Select action a from actor network given state s
        Execute action a in the environment and observe reward r and next state s'
        Store transition (s, a, r, s') in D
        if |D| > B then
            Sample random mini-batch of transitions (s, a, r, s') from D
            Compute critic loss: L_c = (r + γV(s'; φ) − V(s; φ))^2
            Compute actor loss: L_a = −log π_θ(s, a)(r + γV(s'; φ) − V(s; φ))
            Update φ by minimizing L_c using optimizer and GradScaler
            Update θ by minimizing L_a using optimizer and GradScaler
        end if
        s = s'
    end for
end for=0
```

$$
\begin{aligned}
\theta &= \arctan\left(\frac{\sin(\theta)}{\cos(\theta)}\right) \\
r_{\text{angle}} &= \left(3\exp\left(-\frac{(cos(\theta)-\mu)^2}{2\sigma_{\text{angle}}^2}\right)\right)^2 \\
r_{\text{peak}} &= 3\exp\left(-\frac{(cos(\theta)-\mu)^2}{2\sigma_{\text{peak}}^2}\right) \\
p_{\text{action}} &= 0.001\cdot|u|^2 \\
p_{\text{velocity}} &= \left(3\exp\left(-\frac{(cos(\theta)-\mu)^2}{2\sigma_{\text{velocity}}^2}\right)\right)\cdot\left(0.001\cdot|\omega|^2\right) \\
r &= r_{\text{angle}} + r_{\text{peak}} - p_{\text{action}} - p_{\text{velocity}}
\end{aligned}
\tag{8}
$$

where:

- $\theta$ is the angle of the pendulum, calculated from the sine and cosine of the angle.
- $\mu$ is the mean of the Gaussian functions used in the reward function, set to $-1$ as the goal is to swing the pendulum up to the upright position where $\cos(\theta) = -1$.
- $\sigma_{\text{angle}}$, $\sigma_{\text{velocity}}$, and $\sigma_{\text{peak}}$ are the standard deviations of the Gaussian functions used in the reward function, determining the width of the Gaussian functions. Their values are chosen to reflect the spread or tolerance around the desired mean, with a smaller value leading to a narrower peak and thus a stricter reward or penalty. The values were chosen as follows:

$$
\begin{aligned}
\sigma_{\text{angle}} &= 0.6 \\
\sigma_{\text{velocity}} &= 0.1 \\
\sigma_{\text{peak}} &= 0.001
\end{aligned}
\tag{9}
$$

- $r_{\text{angle}}$ is the angle reward, which encourages the agent to swing the pendulum up to the upright position. It's a Gaussian function of the cosine of the pendulum's angle, peaking at $\cos(\theta) = -1$.
- $r_{\text{peak}}$ is the peak reward, a Gaussian function providing a high reward when the pendulum is very close to the upright position. It serves as a fine-tuning reward to give the agent an extra incentive to hit the exact upright position.
- $p_{\text{action}}$ is the action penalty, proportional to the square of the action, discouraging the agent from taking large actions.
- $p_{\text{velocity}}$ is the velocity penalty, a Gaussian function of the cosine of the pendulum's angle, multiplied by the square of the angular velocity $omega$, discouraging high angular velocity near the upright position. This is important to prevent wild swings and to achieve more smooth and controlled movements.
- $r$ is the total reward, the sum of the angle reward and the peak reward, minus the action penalty and the velocity penalty. The goal of the agent is to maximize this total reward, which would mean swinging the pendulum up to the upright position in a controlled manner and stopping there.
- $u$ represents the input voltage that powers the internal motor. Given that $u$ lies within the range $[-3, 3]$, and considering the squaring of the action in the penalty term, smaller input values are incentivized, thus promoting energy efficiency and smoother control actions.

To sum it up, the applied approach consists of training a Deep Q-Network (DQN) agent to derive an optimal control policy for the swing-up problem of a pendulum, based on interactions with a simulated environment. The agent's performance is measured in terms of average reward accumulated over numerous episodes, and the learned policy is subsequently implemented in a real-world scenario. Designing the reward function is an intricate task, often involving a process of iterative refinement and adjustments. One potential enhancement to the reward function could be the incorporation of a sparsity reward that accounts for the number of input actions instead of their magnitude. However, this enhancement was not implemented anymore due to the late discovery of this idea.

### *E. Mitigating the Simulation-to-Reality gap*

The discrepancy between simulation and reality often renders DQN and A2C models, trained in simulation, ineffective when transferred directly to physical systems. However, the introduction of parameter perturbations serves as a viable strategy to bridge this gap. By introducing variations into the simulation parameters, an environmental augmentation process analogous to domain randomization is enacted, which effectively diversifies the learning scenarios. Such a methodology improves the development of a model that is conditioned to function efficiently under a wide range of conditions, thus also on the set of available physical systems, which all slightly vary.

At the onset of each episode, a distinct set of parameters is constructed. Parameters subjected to this randomization process includes the base frequency $\omega_0$ of the pendulum in rad/s, dynamics friction coefficient $\gamma$, the input proportionality constant $K_u$, Coulomb friction coefficient $F_c$, and speed dependent friction $\omega_c$. A random perturbation of 10% is added around the base value of each parameter, creating a uniform distribution within this specified range.

### *F. Hyperparameter Optimization*

The hyperparameters of the agent, including the learning rate, dropout rate, discount factor, soft update factor, batch size, maximum number of time steps per episode, and size of the replay buffer, are optimized using the Optuna library. The objective function for the optimization is the average score over the last 100 episodes. The optimization is run for 20 trials, and the best set of hyperparameters is used to train the final agent.