ME 599 Final Project

Solving Cart-Pole Swing-Up System Using MPC and RL

William Lin, Joslyn Chen, John Cheng, Thomas Lin

Abstract ---

The Cart-Pole Swing-Up problem is a classic challenge in the field of physical control. In this project, we employed contemporary methods to tackle the 4-dimensional model. Our primary approaches involved Model Predictive Control (MPC) and Deep Q-Network (DQN) from Reinforcement Learning (RL). Python, SciPy, OpenAI Gym, and TensorFlow Keras serve as the main frameworks and tools for our implementation. The source code and the animation result can be found in our GitHub repository GitHub link.

Index Terms --- Model Predictive Control, Reinforcement Learning, Deep Q-network

1. INTRODUCTION

Cart-Pole is a common system frequently used in control problems. It consists of a cart that can move along a frictionless track with a pole attached to it. The objective of this system is to maintain the pole in an upright position by controlling the cart's movement along the cart track direction. This seemingly simple problem poses a considerable challenge due to the nonlinear dynamic models as well as the inherent instability of the system. To tackle this problem, we aim to stabilize the Cart-Pole system with two ways: Model Predictive Control and Reinforcement Learning.

Model Predictive Control (MPC) is an advanced control technique that predicts the system's behavior in a finite time horizon with optimal control inputs calculated by minimizing the formulated cost functions while satisfying constraints on the system's states or inputs. The process is repeated until reaching the goal state. This predictive capability enables MPC to handle constraints and adapt to changing system conditions, making it an ideal choice of addressing the stability challenges to the Cart-Pole system.

Reinforcement Learning (RL), on the other hand, is a machine learning paradigm that focuses on learning optimal control policies through interaction with the environment. RL agents learn from trial and error by receiving feedback in the form of rewards or penalties based on their actions. By exploring different control strategies and gradually improving their performance, RL agents can discover effective policies of stabilizing complex and dynamic systems such as the Cart-Pole system.

In our experiment, we implemented both MPC and RL methods using Python, utilizing a variety of libraries to support our implementation. In the MPC framework, we use Scipy as our optimal problem solver and tuned values for best performance. Scipy is a python library that is able to solve non-linear and non-convex optimization problems, making it well-suited for addressing the complex control challenges encountered in the Cart-Pole system. The while loop implementation allowed us to simulate the system's dynamics over small horizons, enabling iterative optimization and control actions in real-time.

In parallel, we utilized the environment provided by OpenAI Gym for implementing RL-based approaches. This framework allows us to simulate the system with built-in functions calculating steps, reward, and even physical models and animation. Within this framework, we utilized the Deep Q-Network (DQN) algorithm with Tensorflow for the neural-network structured dynamics model training the self-balancing agent by interacting with the environment and learning from rewards.

By harnessing the strengths of both MPC and RL techniques, our aim was to effectively address the challenges of stabilizing the Cart-Pole system. These comprehensive approaches enabled us to compare and evaluate their performance, while also providing valuable hands-on experience in formulating and implementing both methods.

2. RELATED WORK

We drew inspiration from a compilation of works that have guided our approach to solving the Cart-Pole swing-up system. These historical studies have played a vital role in constructing the models, frameworks, and environment necessary for our research. Our focus has been on exploring approaches related to Model Predictive Control (MPC) and Reinforcement Learning (RL).

2.1. Optimal Control and Model Predictive Control

The inverted pendulum has been extensively researched in the field of control and robotics. In addition to the MPC method, other control methods have been employed to stabilize the system's inherent instability. Specifically, PID controllers and LQR controllers, proposed by BAKARÁČ, Peter, et al. [1] (2018), are commonly used. However, these methods require a linear model of the dynamic system, and for nonlinear systems like the Cart-Pole system, linearization is often performed to approximate the system as linear.

In the realm of control systems, systems can be broadly classified as either open-loop or closed-loop control systems. Open-loop control, also known as feedforward control, relies on a presumed perfect model and a sequence of control inputs to achieve a desired output without considering the current system states or any feedback. However, it lacks adaptability and robustness as it cannot handle disturbances or uncertainties through feedback.

Closed-loop control, also known as feedback control, incorporates an additional error term that compares the system's actual response to a reference signal or target. It calculates an error signal to modify the control input and drive the system towards the desired state. By considering feedback, it exhibits good adaptability to disturbances and uncertainties encountered in real-world scenarios. However, it lacks the

ability to predict future system behaviors like MPC does and cannot effectively handle constraints on system states or control inputs.

CAMACHO, Eduardo F., et al. [2] (2013) demonstrated that MPC is a control strategy that combines elements of open-loop and closed-loop control techniques to address complex control problems. At its core, MPC breaks down the control problem into several small horizon sequences as open-loop controls and formulates each sequence as an optimal control problem, taking into account the dynamic model, constraints, and desired goal state. Within each short horizon, MPC calculates the optimal control sequence and feeds only the first control input to the plant as it progresses. MPC stands out from traditional closed-loop control by implicitly incorporating feedback error terms between iterations of the short horizons. It continuously recalculates control actions at each time step, adapting them based on new measurements and the updated system state. This feedback-driven adjustment enhances MPC's ability to adapt to changing conditions and improve system performance, even with imperfect dynamic models.

By combining the predictive capabilities of open-loop control with the feedback-driven adjustments of closed-loop control, MPC offers a powerful control strategy that excels in handling complex systems, nonlinear dynamics, and constraints. Its ability to anticipate future behavior and optimize control actions accordingly makes it an effective approach for various control applications, including stabilization, trajectory tracking, and energy optimization.

In the realm of the Cart-Pole systems, a real-time MPC framework was proposed by ASKARI, Masood, et al. [3] (2009). To ensure rapid response times, the framework utilized quadratic programming (QP) to formulate the cost function. By incorporating quadratic terms, QP effectively handles linear constraints, as well as convex and non-convex problems. In our chosen model, we considered the angular position between the pole and the cart, necessitating the implementation of sine and cosine functions within the dynamics model. Hence, we opted to express the cost function using quadratic terms, leveraging their advantageous properties for our specific requirements.

MILLS, Adam, et al. [4] (2009) conducted a real-time experiment aimed at stabilizing the inverted pendulum system. Instead of using Quadratic Programming (QP) as mentioned earlier, they opted for a more advanced technique called Sequential Quadratic Programming (SQP). SQP is an iterative optimization algorithm specifically designed for solving nonlinear-constrained optimization problems. It builds upon the principles of QP but expands its capabilities to handle nonlinear objective functions and constraints. Unlike directly solving the optimization problem, SQP approximates the nonlinear functions using quadratic models at each iteration, enabling a step-by-step refinement of the solution.

2.2. Reinforcement Learning Frameworks

The concept of utilizing machine learning and reinforcement learning (RL) for strategic tasks was proposed by Samuel, Arthur L. [6] (1959). The author demonstrated the improvement of strategies for the game of checkers. Inspired by this work, we aim to apply RL to solve the strategic challenge of the Cart-Pole swing-up task. RL is a method that enables an agent to learn and improve its decision-making abilities through interactions with the environment. By constructing a reward function and weighing the policy, the agent optimizes its decisions to reach the goal state when properly set up.

Q-learning, introduced in the article "Q-learning" by Watkins and Dayan [7] (1992), published in Machine Learning, is one of the most widely used models in RL. The concept of Q-learning involves building a quality table to analyze the rewards associated with each action. With a finite number of states and actions, the Q-table is expected to converge after a certain number of steps. The core idea of Q-learning is to use the Bellman equation to update the Q-values, which describes how the Q-value of the current state can be updated based on the current reward and the maximum Q-value of the next state. Through iterative updates of the Q-values, Q-learning gradually converges towards the optimal policy.

Q-learning is a highly effective model for discrete problems. Since all Q-table values are updated in each step, the training result is likely to converge to the optimal solution. However, as problems become more complex, the limitations of Q-learning become apparent. If the input of the physical model is continuous or in multiple dimensions, the Q-table can become extremely large, leading to a significant decrease in computational efficiency. Consequently, adjustments or improvements are necessary to extend the concept of Q-learning to advanced challenges.

The concept of Deep Q-network (DQN) was proposed by MNIH, Volodymyr, et al. [8] (2015). DQN is a notable deep reinforcement learning model that combines the conceptual framework of Q-learning with Neural Networks. The article showcases the application of DQN in advanced games like Atari 2600 and sports video games, demonstrating how it overcomes the limitations of Q-learning through extension algorithms and sampling policies. By incorporating DQN into our model, we can address the challenges of the Cart-Pole system more effectively.

DQN introduces several key upgrades compared to Q-learning. Firstly, it leverages Deep Neural Networks to approximate the Q-value function, enabling a more efficient representation of Q-values in high-dimensional and continuous state spaces. Secondly, it incorporates Experience Replay, where the agent's experiences are stored in a replay buffer and randomly sampled during training. This breaks the correlation between consecutive experiences, improving sample efficiency and stabilizing learning. Additionally, DQN incorporates constructs such as Exploration-Exploitation Tradeoffs, Fixed Target Networks, and Gradient-Based Optimization to further enhance its performance and power. These advancements make DQN a more powerful and effective model for reinforcement learning tasks.

OpenAI Gym, founded by BROCKMAN, Greg, et al. [9] (2016) is a Python-based introductory framework for RL. It offers a diverse set of standardized RL environments, including popular models like Pendulum and Cart-Pole. The framework also provides a render visualization function that allows designers to assess the state of the physical model during operation, facilitating effective adjustment of training parameters. OpenAI Gym has become a widely used tool in the RL community for developing and evaluating RL algorithms.

OpenAI Gym demonstrates robust compatibility with popular machine learning frameworks, such as Keras and PyTorch, facilitating seamless integration with advanced models like DQN. Gym maintains exceptional performance even when handling complex models like DQN while providing extensive flexibility for parameter adjustments. Furthermore, when combined with prevalent visualization libraries like Pyplot or Matplotlib, Gym enables the effective representation of RL training results.

The Inverted Pendulum and Fixed Pendulum are commonly studied RL tasks, while the Cart-Pole Swing-Up problem is considered more challenging due to its multi-dimensional states involving swinging and maintaining movement. MANRIQUE ESCOBAR et al. [10] (2020) employed advanced training models such as Deep Deterministic Policy Gradient (DDPG) to tackle this problem. In contrast, our approach involves utilizing DQN to address the challenge, aiming for training outcomes comparable to those achieved by MPC.

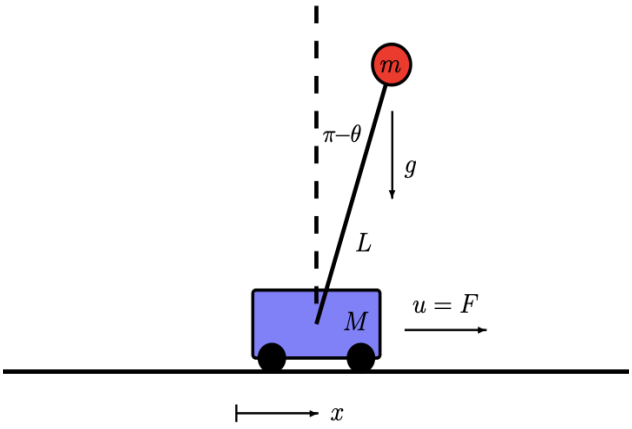

Fig. 1: Schematic of inverted pendulum on a cart, adapted from Brunton and Kutz (2022) in "Data-driven science and engineering: Machine learning, dynamical systems, and control" (p. 352) [10].

3. PROPOSED METHOD

In this section, our proposed method is presented for addressing the Cart-Pole system with approaches of MPC and RL. Our objective is to implement and compare the advantages of each model to effectively tackle the challenges posed by the Cart-Pole system.

3.1. Problem Formulation

The dynamics model of the Cart-Pole swing-up system depicted in Figure 1 is referenced from the book "Learning, Dynamical Systems, and Control" published by Cambridge University Press in 2022 [10]. The full nonlinear dynamics are listed by the following equations:

$$\dot{x} = v$$

$$\dot{v} = \frac{-m^2L^2g\cos(\theta)\sin(\theta) + mL^2(mL\omega^2\sin(\theta) - \delta v) + mL^2u}{mL^2(M + m(1 - \cos(\theta)^2))}$$

$$\dot{\theta} = \omega$$

$$\dot{\omega} = \frac{(m+M)mgL\sin(\theta) - mL\cos(\theta)(mL\omega^2\sin(\theta) - \delta v) + mL\cos(\theta)u}{mL^2(M + m(1 - \cos(\theta)^2))}$$

x is the cart position, v is the velocity, θ is the pendulum angle, ω is the angular velocity, m is the pendulum mass, M is the cart mass. L is the pendulum arm, g is the gravitational acceleration, δ is a friction damping on the dart, and u is the control force applied to the cart. The parameter values used in this project are as follows: g = 9.81 m/s, m = 0.5 kg, M = 1.0 kg, L = 0.5 m, and δ = 0.

Based on the dynamics model, the state of the system consists of the cart's position, velocity, pendulum angle, and angular velocity, while the control input represents the force applied to the cart. The objective of this project is to swing up the pendulum from its initial downward position (θ=0) by manipulating the force applied to the cart.

Therefore, we plan to leverage both MPC and RL techniques to tackle this problem. Given the relatively low-dimensional nature of the system, we will directly incorporate the nonlinear dynamics into the MPC framework to generate the control sequences required for swinging up the pendulum effectively. In addition, we will utilize the OpenAI Gym environment for implementing the RL approach, which will facilitate the training and evaluation of RL agents in this specific setting. This integration with OpenAI Gym will provide a convenient platform for training RL agents and evaluating their performance in solving the Cart-Pole swing-up task.

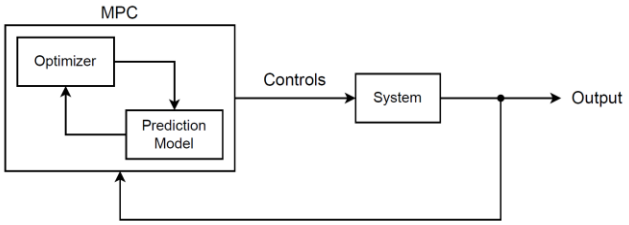

Fig. 2: MPC Flow Diagram

3.2. MPC Formation

Model Predictive Control (MPC) is an advanced control strategy widely used in engineering and industrial processes. It leverages a dynamic mathematical model to compute precise control actions based on the current system states, constraints, and objectives. MPC's strength lies in its ability to accurately predict the system's future behavior, enabling effective control by simulating and forecasting responses to control inputs.

In our project, we aim to design a model predictive controller for an inverted pendulum system with an adjustable cart. The main objective is to swing up the inverted pendulum from its initial downward position to an upright position by manipulating the force applied to the cart. To achieve this, we will incorporate the nonlinear dynamics of the system into the MPC framework.

To implement MPC, we will create an objective function, represented by the 'cost_scipy' function, which captures the deviation between the current state of the system and the desired goal state. We will use the Python package scipy.optimize.minimize to solve the optimization problem. The system's states, including the cart position, velocity, pendulum angle, and angular velocity, will be considered, while the control input will be the force applied to the cart.

The initial state of the system will be set to [0, 0, 0, 0], and the goal state will be [1, 0, π, 0], indicating that we want the inverted pendulum to end up at the top of the cart.

To determine the optimal control actions, MPC operates in a receding horizon manner. We define a horizon, which represents the time steps over which MPC solves the optimization problem. This allows MPC to consider the predicted behavior of the system over the entire horizon while determining the optimal control actions for each time step.

The objective function in MPC incorporates a cost matrix, which assigns penalties to both the state variables (Q) and the control input (R). The cost matrix Q captures the significance of each state variable in the objective function, with higher penalties assigned to states that require more precise control or have greater consequences if deviating from the desired goal. The cost matrix R reflects the effort or cost associated with manipulating the control input.

By defining these penalty matrices, the objective function in MPC aims to minimize the cumulative cost over the prediction horizon. To solve the optimization problem, we utilize sequential least squares programming, setting the method as 'SLSQP' in scipy.optimize.minimize. This algorithm generates control actions that minimize the objective function while satisfying the system dynamics and constraints over the prediction horizon.

However, it's important to note that MPC only implements the first control action from the computed sequence, which corresponds to the action to be applied in the current time step. The remaining control actions are discarded, and the process is repeated at the next time step.

By iterating this process until the desired goal state is achieved, MPC will output the control sequence required to swing up the inverted pendulum.

In summary, our project focuses on utilizing MPC to control an inverted pendulum system by generating a control input sequence that effectively swings up the pendulum. By formulating an objective function, considering the system's dynamics, and optimizing control actions over a finite time horizon, we aim to achieve precise and efficient control of the system using the MPC approach.

3.3. Cart-Pole Model Deep Q-Learning

The second model is using the Deep Q-Learning (DQN) model to solve this Cart-Pole problem. We created an environment with boundaries for the system. Then we define the reward function, which is our output of each of our "step()" functions, then feed it to our Sequential NN model; finally, let the model learn the system, to achieve the swing-up motion.

To begin, we used OpenAI Gym for the environment. There is a pre-defined class called "CartPole-v1" in the default library; however, it did not match our specific requirements, thus we modified the class to make the pole point downwards in our zero state. The system has a continuous, four-dimensional output for its state, representing the cart's position, cart's velocity, pole's angle, and pole's angular velocity. The action space consists of two discrete actions: moving the cart left or right.

Then we defined the reward function. Initially, we thought of using the minus of the sum of x, x_dot, theta, and theta_dot as the reward function, however, this function is then vulnerable to the change of direction and the swings. Eventually, we came to the conclusion of letting the sum of the kinetic energy and the potential energy be as close to the final state. This modified reward function has a greater penalty for either too large or too little of total energy generated.

Next, we built the DQN model using a sequential architecture from Keras. The model consisted of a series of fully connected (dense) layers. We used the ReLU activation function for the hidden layers and a sigmoid activation function for the output layer to estimate the Q-values for each action. The DQN model was then compiled with the Adam optimizer and the mean squared error loss function. We chose the mean squared error loss function to minimize the difference between the predicted Q-values and the target Q-values during training.

DQN is typically useful when we have a continuous observation state space, unlike the Q table from the general Q-learning method, DQN utilizes a neural network that can generalize across similar states and actions and could handle higher dimensional input spaces.
To train the DQN model, we implemented an iterative process. For each episode, we reset the environment and obtained the initial state. Then, we executed the following steps until the episode terminated, shown on the graph below:

First, we applied an epsilon-greedy exploration strategy to determine the action to take. With a certain exploration rate (epsilon), we either selected a random action with high probability or chose the action with the maximum predicted Q-value from the DQN model. We then performed the selected action in the environment and received the next state, reward, and termination flag. The reward is bigger if the pole is going upwards, and the episode terminated if the pole angle exceeded a threshold or the cart moved out of bounds.

Next, we stored the experience tuple (state, action, reward, next_state, done) in the DataPlotter class as lists.Then sampled a mini-batch of experiences from the replay memory and used it to update the DQN model's weights through backpropagation. The target Q-values were calculated using a target network, which was periodically updated with the main DQN network's weights to stabilize the training process. Finally, we updated the current state to the next state for the next iteration.

The results from our two reward functions have a great difference, for our naive approach, the model nearly did not go over 90 degrees for the first 10 episodes, and for our final, modified reward function, it did not take the model for more than 3 or 4 episode for it to learn to swing and increase its potential energy.

We repeated this training process for a specified number of episodes, gradually decreasing the exploration rate (epsilon) over time to shift from exploration to exploitation. After training, we evaluated the performance of the trained DQN model by running several episodes in the environment with the exploration rate set to zero. We recorded the total rewards obtained in each episode to assess the model's effectiveness in swinging up the pole. Finally, we visualized the results using Matplotlib, plotting the max rewards obtained per episode, as well as the scores for each episode. We also recorded the maximum angle for each episode and plotted them side by side. This allowed us to observe the learning progress of the DQN model and determine if it successfully solved the Cart-Pole problem.

In summary, the method involved building a DQN model using TensorFlow.Keras, training it on the Cart-Pole environment from OpenAI Gym, and evaluating its performance. The exploration-exploitation trade-off was managed through epsilon-greedy exploration, and the training progress was visualized using Matplotlib and Pygame.

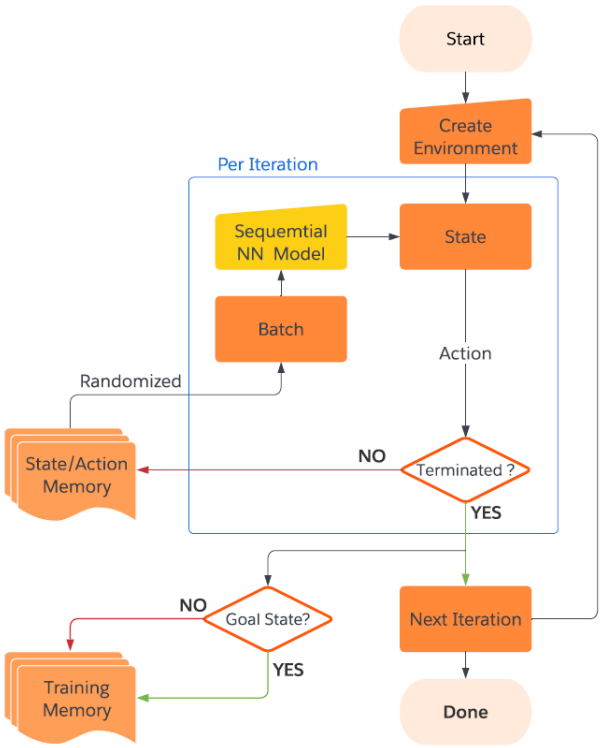

Fig. 3: DQN Flow Chart

4. EXPERIMENTS

The experiments chapter presents the running results of our MPC and RL/DQN models. It includes the training results, parameter settings, comparisons, and discussions. The diagrams and source code can be accessed in our GitHub repository.

4.1. MPC Setup and Experimental Results

In our experiment, we began by formulating the Cart-Pole system's physical model in state-space form and implemented an animation using matplotlib. With the given dynamics model, we were able to simulate the motion by providing a random control input sequence and integrating from continuous time dynamics to discrete time dynamics using the Euler Integration Method. However, we observed that the simulated outcome exhibited continuous rotation even with zero control inputs or extremely small angles. After further investigation, we identified that this issue was caused by the error introduced by the Euler integrator. To address this problem, we replaced the Euler integrator with the more accurate Runge-Kutta integrator. With this modification, the system successfully reached its downward stabilization point with zero control inputs every time, enabling us to proceed with our experiments.

In our previous works, we gained experience in implementing MPC with a 3D point mass, which is a linear model, in a target-reaching problem. In that particular problem, we formulated the optimization problem using cvxPy, leveraging convex cost terms. However, when considering the dynamics model of the Cart-Pole system, which involves sine and cosine terms, the resulting cost function becomes non-convex. Consequently, we had to adapt our approach and formulate the objective function in a manner that would enable us to utilize the SciPy optimization library for solving the problem. This allowed us to overcome the non-convexity challenge and proceed with implementing MPC for the Cart-Pole system.

In the previous chapter, we discussed the importance of the cost matrices Q and R in penalizing the state output and control input, respectively, within the MPC framework. By setting these cost matrices, we can determine the system's aggressiveness in driving toward the desired goal. In this experiment, we assume that we have unlimited control inputs available, which leads us to assign a value of 0.1 for the R matrix. On the other hand, our primary objective is to achieve an upright position for the beam, while placing less emphasis on the precise location of the cart at the end of each iteration. Therefore, we assign a higher cost to the pendulum angle and angular velocity, reflecting the importance of reaching the goal of balancing the pendulum in an upright position.

Initially, our approach showed promise as we successfully achieved the desired goal of stabilizing the Cart-Pole system within a limited total MPC horizon. However, we soon encountered challenges related to the sensitivity of our model to the assigned cost values. It became evident that even a slight adjustment in the weight assigned to penalize the angle term would cause the model to fail. Furthermore, in our experiment, we opted for a penalized mechanism rather than directly constraining the position of the cart. This decision posed additional difficulties as we struggled to strike the right balance between the parameters. We faced the challenge of simultaneously maintaining the beam in an upright position while effectively limiting the final position of the cart. Finding this delicate balance proved to be a complex task. To illustrate our results, we have included a plot showcasing the outcomes of our MPC-based Cart-Pole stabilization efforts. In addition, we have made the completed source code and further results available in the provided GitHub repository.

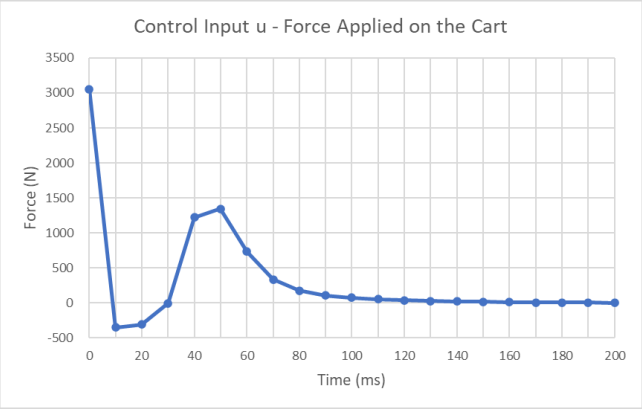

Fig. 4(a): Input Force Chart

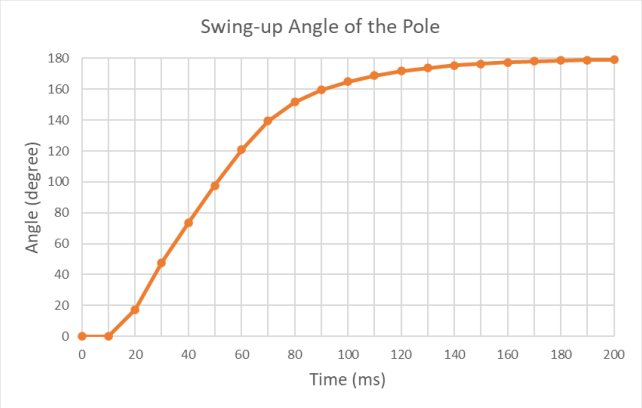

Fig. 4(b): Swing-up Angle Chart

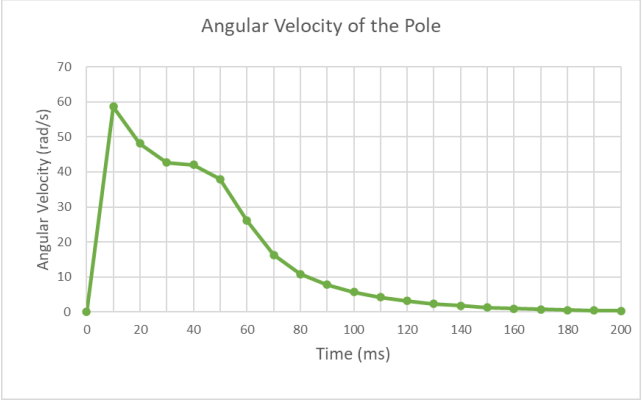

Fig. 4(c): Angular Velocity Chart

4.2. DQN Setup and Experimental Result

After modeling the environment in Gym and designing the neural network model for our DQN, we proceeded to test our model. Initially, we implemented the model in a single Python function, where we plotted the graph using a single command. However, we soon realized a design flaw in our approach. We were unable to redraw the graph and compare results from different experiments, particularly when using different sets of parameters. To address this issue, we decided to redesign our model. We introduced a new feature to store all the data locally in CSV files and restructured our code into two Python classes. One class was responsible for training the model, while the other class focused on plotting and saving the data to CSV. This redesign allowed us not only to compare data points between different parameter sets but also to access various methods within the same class, which temporarily stored our data points in class properties. This improvement significantly accelerated our debugging and testing process, making it more efficient to analyze and compare results from different experiments.

We tested our reward function, there are three variables that we considered, the theta, theta dot and the position x, the naive approach that we came up is to reward the theta that is closest to 180 degrees, reward the theta dot that is closest to zero, and reward the x position that is closest to zero (starting point).Our naive approach is then summing up the total of these three regulations and form our reward function. We fed this reward function to our model; the result is presented in Fig. 5. We found that there should be a modification of our reward function. The swing-up motion cannot be instigated through the current reward. In order for the model to swing up the pole, there is an initial energy that should be generated, after some research, we decided to use the total energy approach, that is the potential energy of the pole with the kinetic energy of the pole.

We discovered that if we are at the goal position, there is potential energy but little kinetic energy, and when the pole is at its starting position, it has zero potential energy, so it should have more kinetic energy in this case. Eventually, we came up with a squared equation of the total energy being as close as that of the goal state, penalizing either too large or too small of total energy. We also have another penalty called the boundary penalty, basically penalizing the model if it gets too close to the boundary. We tested our new reward function with the model. The result is presented below. As we could see, the new reward function has a faster converging speed, typically before the first half of our episodes, the new model was able to learn the motion a little faster than the former model.

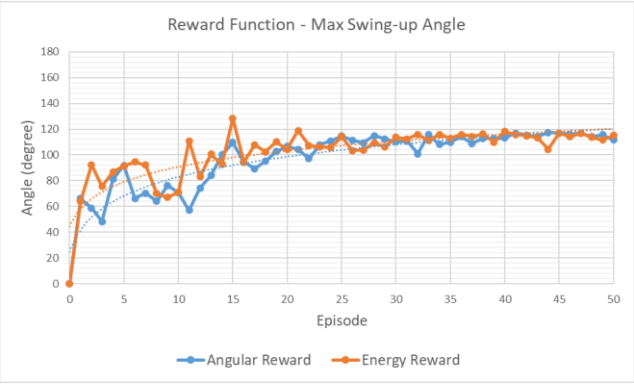

Fig. 5(a): Reward Function Comparison

For our DQN algorithm, in order to achieve the state-of-the-art result. We have multiple hyperparameters to tune, the exploration decay, batch size, and gamma. We also have the episode and iteration number to follow up with.

The exploration decay rate determines the rate at which the agent transitions from exploration to exploitation during training, for Q-learning type of algorithms, the model typically starts from a higher exploration rate and gradually decreases until it finds the best policy and sticks with exploiting that policy. We tested out the exploration rate of 0.995, 0.997, and 0.999, as the results are shown below, we could see that the model converges faster on the former one, and slowest on the 0.999 one.

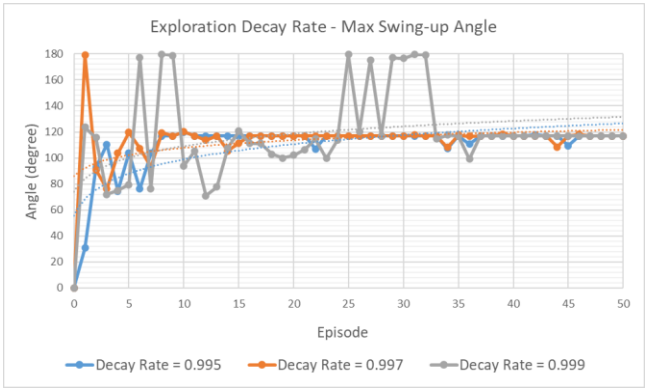

Fig. 5(b): Explorational Decay Comparison

The batch size refers to the width of the model's discovery. Whenever the model takes one step, it is storing the next state after that step in its memory. And the batch size determines how many options the model selects from its memory to generate the next optimal move. The reason for us to not simply use a Q-table and Arg-maxing as the next best step is because of the continuity of our observation space, it complicates the system and makes the Q-table too large to store in this case. We selected batch size = 5, 10, 20. The result is shown below, however, there is not really a great difference in the sense of these three batch sizes, the reason being that there might not be enough episodes and iterations that we would discuss later on. The larger the batch size is, the more time the model is spending on each iteration, generally, we try to limit the batch size to 20.

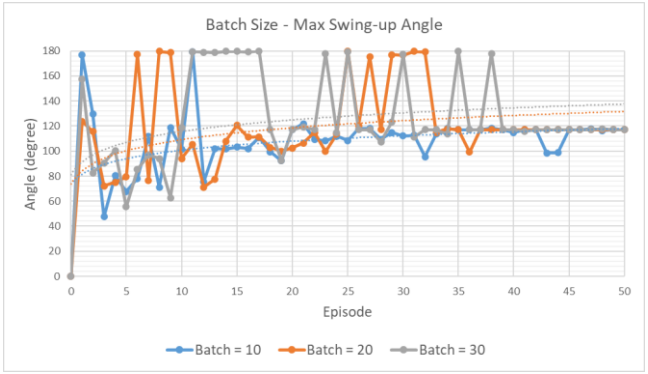

Fig. 5(c): Batch Size Comparison

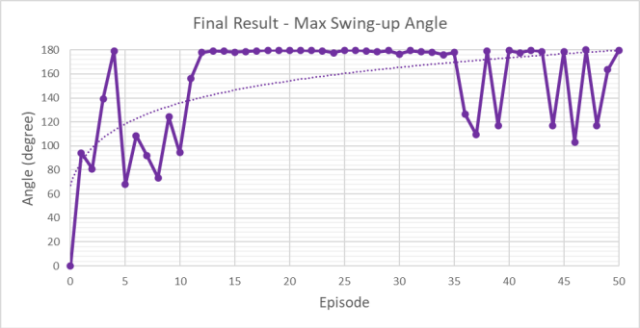

Fig. 5(d): DQN Final Result

For the final result, we have our batch size = 20, episodes = 50, max iteration = 200, decay = 0.999, as shown in Fig. 5(d). We could see there is an improvement in max angle during the first half of the episodes, however, there is still not a substantially stable result coming from this DQN model.

We suspect that the main reason is the insufficient computational power we have, it is said that it requires more than 1000 episodes to teach basic motions to machines, whereas we were only able to perform around 50 episodes, with higher-spaced hardware, we might see a better bump in our result. The same reason also limited the hyper-parameter tuning. One other potential improvement is to fine grain the reward function, we are currently using only the velocity and the y-position of the pole, another factor is to add the velocity approaching or leaving the goal position as another reward, adding more rewarding factors is certainly a good starting point for the next steps of this research.

4.3. Discussion and Comparison to MPC and RL

MPC is a powerful model that can effectively solve multi-dimensional physical control problems when the goal state is well-defined and the system is free from external disturbances. Based on our experimental experience, MPC offers a high degree of freedom in parameter design and provides visible feedback when tuning variables. The program execution is fast and runs in seconds. In general, MPC is a suitable method for the Cart-Pole Swing-Up problem.

During the experiments, we observed certain limitations of the MPC model. For instance, the parameters have mutual interdependence. When some parameters are excessively large, they tend to overshadow the others. Another issue is that when the time step increases, the program's execution time will significantly raise. This phenomenon may render the model unsuitable for real-time control systems. As a result, exploring supportive methods or algorithms to improve efficiency may be necessary for high-dimensional problems.

In comparison, the DQN model incurs significantly higher computational costs than MPC. Under identical conditions, the execution time of DQN is at least 1000 times higher than that of MPC, and the randomness of the DQN model is higher. Based on our experience, adjusting parameters in DQN is less intuitive. Furthermore, due to efficiency limitations, the optimization process of the model is relatively slow. Overall, in low-dimensional problems with well-defined objectives, the advantages of DQN are less pronounced. In contrast, control models like MPC would be a better choice.

Despite the limitations, we were able to improve the performance of the model through multiple attempts under constrained conditions. When the parameter proportions were appropriate, the DQN model exhibited a certain degree of correct convergence. For further exploration, by incorporating advanced features such as Exploration-Exploitation Tradeoff and Fixed Target Network, we anticipate that DQN can achieve comparable performance to MPC, and be even available to extend its applicability to higher-dimensional challenges.

5. CONCLUSION

Both the MPC and DQN models demonstrated their effectiveness in addressing the Cart-Pole swing-up challenge. Our MPC model achieved the swing-up task efficiently, showcasing its ability to control the Cart-Pole system. On the other hand, the DQN model, while not reaching the optimal solution, exhibited a degree of stable convergence after multiple attempts.

This project presented a great opportunity for learning about real-world problems. During the process, we successfully acquired the skills to construct the environment, framework, and models while exploring suitable algorithms to achieve the objective. If given the chance to further extend this project in the future, we expect to produce better results and expand our models to tackle higher-dimensional problems and applications. In conclusion, this project successfully employed MPC and DQN to solve the Cart-Pole swing-up problem and accomplished analyzing model efficiency and improvement strategies.

REFERENCES

[1] BAKARÁČ, Peter; KLAUČO, Martin; FIKAR, Miroslav. Comparison of inverted pendulum stabilization with PID, LQ, and MPC control. In: *2018 Cybernetics & Informatics (K&I)*. IEEE, 2018. p. 1-6.

[2] CAMACHO, Eduardo F.; ALBA, Carlos Bordons. *Model predictive control*. Springer science & business media, 2013.

[3] ASKARI, Masood, et al. Model predictive control of an inverted pendulum. In: *2009 International Conference for Technical Postgraduates (TECHPOS)*. IEEE, 2009. p. 1-4.

[4] MILLS, Adam; WILLS, Adrian; NINNESS, Brett. Nonlinear model predictive control of an inverted pendulum. In: *2009 American control conference*. IEEE, 2009. p. 2335-2340.

[5] SAMUEL, Arthur L. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 1959, 3.3: 210-229.

[6] WATKINS, Christopher JCH; DAYAN, Peter. Q-learning. *Machine learning*, 1992, 8: 279-292.

[7] MNIH, Volodymyr, et al. Human-level control through deep reinforcement learning. *nature*, 2015, 518.7540: 529-533.

[8] BROCKMAN, Greg, et al. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[9] MANRIQUE ESCOBAR, Camilo Andrés; PAPPALARDO, Carmine Maria; GUIDA, Domenico. A parametric study of a deep reinforcement learning control system applied to the swing-up problem of the cart-pole. *Applied Sciences*, 2020, 10.24: 9013.

[10] BRUNTON, Steven L.; KUTZ, J. Nathan. *Data-driven science and engineering: Machine learning, dynamical systems, and control*. Cambridge University Press, 2022, p. 352-353.