Dynamics and Control of an Inverted Pendulum on a Cart via Optimal Linear Quadratic Regulator and Reinforcement Learning

a project presented to
The Faculty of the Department of Aerospace Engineering
San José State University

in partial fulfillment of the requirements for the degree
Master of Science in Aerospace Engineering

by

Grace Y. Feng

December 2024

approved by

Professor Jeanine Hunter
Faculty Advisor

ABSTRACT

Dynamics and Control of an Inverted Pendulum on a Cart via Optimal Linear Quadratic Regulator and Reinforcement Learning

Grace Y. Feng

With the rapid growth of artificial intelligence and machine learning (AI/ML) in recent decades, businesses are increasingly leveraging these technologies to enhance their platforms. When implemented effectively, AI/ML can deliver transformative outcomes, improving efficiency and reducing operational costs. Significant successes have been seen in areas such as object recognition, gaming, and robotics, among others. This project focuses on a specific subset of AI/ML, known as reinforcement learning (RL), and examines RL-based algorithms for controlling a 2-D nonlinear inverted pendulum on a cart. While RL encompasses the training and testing of various RL agents, it also includes a specialized area called deep reinforcement learning (DRL). Specifically, this project explores two DRL-based approaches from Stable Baselines3: the Advantage Actor-critic (A2C) and Proximal Policy Optimization (PPO). For the purpose of this project, RL and DRL are used interchangeably. The performance of these DRL agents is evaluated and compared with that of a baseline optimal linear quadratic regulator (LQR) controller. Evaluation metrics for the DRL agents include mean episode length, mean episode reward, and value loss. For comparison with the LQR controller, the metrics considered are overshoot/undershoot displacements, settling time, and stability convergence. The results indicate that the proposed DRL-based methods generally outperform the LQR controller in terms of overshoot/undershoot displacements and settling time. Although the inverted pendulum system is successfully balanced, the DRL results display neutrally stable system responses with sustained oscillations and exhibit non-zero stability convergence in the cart's motion. Future work aims to further optimize and enhance the stability of the DRL-based solutions.

Acknowledgments

I extend my sincere gratitude to the faculty of the Department of Aerospace Engineering at San José State University. In particular, I wish to express my heartfelt thanks to my project advisor, Professor Jeanine Hunter, for believing in me and giving me the opportunity to explore the cutting-edge field of artificial intelligence and machine learning (AI/ML). Your kindness, understanding, and appreciation of my work have been invaluable and kept me motivated. I am also deeply grateful to our Department Head, Dr. Nikos Mourtos, for welcoming me into the program and providing guidance throughout my academic journey in aerospace engineering. Finally, I would also like to thank my family and friends for their unwavering support and encouragement throughout my graduate studies.

Table of Contents

List of Symbols

Symbol	Definition	Units (SI)
x	Cart position	m
$\dot{x}$	Cart velocity	$\frac{m}{s}$
$\ddot{x}$	Cart acceleration	$\frac{m}{s^2}$
θ	Pendulum angle	deg or rad
$\dot{\theta}$	Angular velocity of the pendulum	$\frac{deg}{s}$ or $\frac{rad}{s}$
$\ddot{\theta}$	Angular acceleration of the pendulum	$\frac{deg}{s^2}$ or $\frac{rad}{s^2}$
m_A	Mass of the cart	kg
m_B	Mass of the pendulum rod	kg
g	Gravitational acceleration	$\frac{m}{s^2}$
B_{cm}	Pendulum rod's center of mass	m
I_{zz}	Pendulum rod's mass moment of inertia about B_{cm}	kgm^2
F_c	Control force applied to the cart	N
L	Distance between cart frame and B_{cm}	m
Ⓐ	Cart's reference frame by $\hat{a}x$, $\hat{a}y$, and $\hat{a}z$ (non-Newtonian)	——
Ⓑ	Pendulum rod's body frame by $\hat{b}x$, $\hat{b}y$, and $\hat{b}z$ (non-Newtonian)	——
Ⓝ	Newtonian reference frame by $\hat{n}x$, $\hat{n}y$, and $\hat{n}z$	——

List of Tables

List of Figures

1. Introduction

1.1 Motivation

The demand for robust, low-cost optimal control systems used to stabilize and control nonlinear dynamical systems such as aircraft and spacecraft have risen due to the bloom of artificial intelligence and machine learning (AI/ML). Typically, over 90% of the controllers used today are proportional-integral-derivative (PID) controllers [1]. These industry-standard controllers are well developed by classical control methods and do the job well with guaranteed performance. However, they have limitations, as do most things in real life. PID controllers, for example, are only suitable for single-input, single-output (SISO) systems, require parameter tuning, and are often constrained under certain conditions. In aircraft dynamics and control, for instance, a PID controller designed to satisfy a specific flight regime is generally not applicable to all flight envelopes. This is why multiple linear controllers at different trim points are often required in flight control design [2]. For multiple-input, multiple-output (MIMO) systems, state space models, which are built by modern control theory, are used instead. State space models are designed to solve optimization control problems. Controllers designed for this type of problem are often developed efficiently by computers. However, problems that are multidimensional and nonlinear or contain highly coupled state equations require increased computational power [3]. Furthermore, classical and modern control methods often require the derivation of the exact mathematical model for the system, which are not always readily available in practice due to unmodeled perturbations and modeling errors. In light of the limitations discussed, there is a need to examine alternative methods that are more robust and economical in time as well as computational effort. Machine learning (ML), according to [4], is best defined by Arthur Samuel in 1959 as "the field of study that gives computers the ability to learn without being explicitly programmed." AI/ML is the state-of-the-art technology that has gained popularity in recent years due to its success in employing data-driven models to solve complex problems with proven results. For example, handwritten digit recognition through the use of Deep Neural Networks (DNN) has provided results with over 98% precision [5]. AI/ML consists of three main paradigms: supervised learning, unsupervised learning, and reinforcement learning. Of those, reinforcement learning is often used to solve optimal control problems due to its ability to learn and improve on its own, even in the presence of uncertainties. The goal of this project was to conduct a comparative study of two popular types of reinforcement learning algorithms from Stable Baselines3 - Advantage Actor-critic (A2C) and Proximal Policy Optimization (PPO) - with the classic optimal linear quadratic regulator (LQR) to evaluate their performance in an inverted-pendulum-on-a-cart problem.

1.2 Literature Review

An inverted pendulum on a cart is a system that is commonly used in control studies and research. It is a two-dimensional (2-D), nonlinear, second-order system that is easy to model and experiment with, making it a teaching aid favored by many students, scholars, and researchers across the scientific communities. Even though it is inherently unstable, it can be stabilized through various control methods. The goal for any inverted pendulum control problem is to balance the inverted pendulum by applying a force to the cart to which the pendulum is attached, as shown in

figure 1.1 [6].

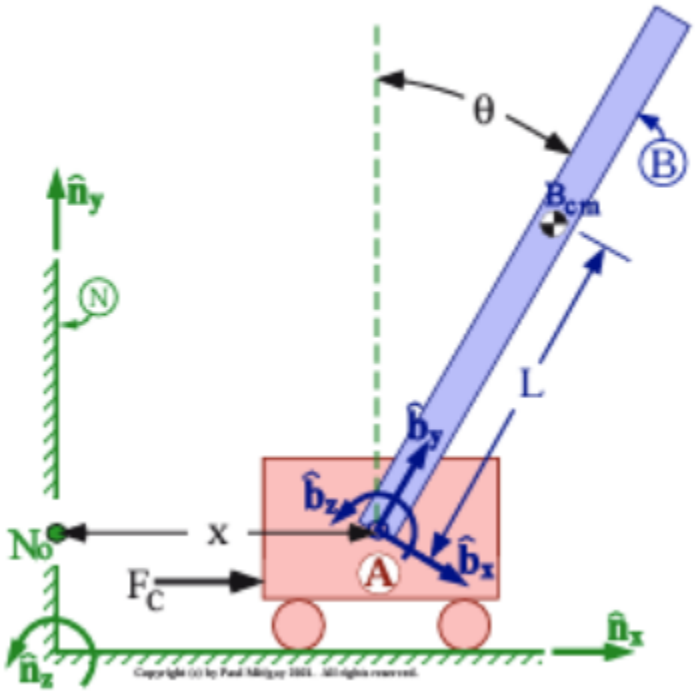


Figure 1.1: Schematic of an inverted pendulum on a cart [6].

Many believe that the inverted pendulum systems have been used as a classical control tool in the laboratories since the 1950s [7, 8]. However, findings showed that credited work was first documented in 1960 by a student named James K. Roberge, who applied classical control methods to stabilize and control an inverted pendulum system for his bachelor's thesis at the Massachusetts Institute of Technology (M.I.T.) [8]. Roberge's work and many other inspired versions of inverted pendulum systems suggest that to successfully control and stabilize an inverted pendulum system, both the dynamic motion of the pendulum pole and the cart must be considered carefully [8, 9]. Assuming that the pendulum pole could rotate freely about a hinge point that is fixed to a cart and connects the pole, and the cart is free to translate horizontally. If the cart moves too drastically to reach one end on the horizontal direction, then the pendulum pole would submit to a large angle (e.g. greater than ± 24° from the vertical axis) on the vertical plane, where the pole would not be able to recover in time to balance itself. On the other hand, if the cart were stationary, then as time goes by the pendulum pole would simply fall over by the pull of gravity without the help of any control mechanism.

From a mechanical standpoint, an inverted pendulum on a cart is a system that could be easily understood using intuition. The intuition part comes in when one imagines physically balancing a rod on a hand [8, 10]. A perfectly upright rod is naturally unstable because any deviations from this equilibrium will not converge back to this point. To displace the rod to the right, the hand must first move to the left. Subsequently, to balance and control the rod as it tips to the right, the hand must catch it by swiftly moving toward the same direction (in this case right) as the rod and so

on and so forth. Beyond the physical senses, an inverted pendulum system can also be analyzed theoretically using fundamental calculus-based physics and rigid bodies dynamics, as shown by the work of Åström and Furuta [7], Blitzer [11], and Cannon [12].

From a control perspective, an inverted pendulum on a cart is a system that could be properly controlled via various methods. Two of the most common ones are classical control and modern control methods. Many studies related to inverted pendulum system control assume linearization to simplify the nonlinear system [8, 9, 13]. This reasonable assumption makes it possible for engineers to use linear classical control strategies since for a small range of angles, the results of a linearized system can often accurately approximate those of a nonlinear system. Furthermore, in the days of paper and pen, linear systems were more desired as they could be solved by hand and analyzed graphically. In the 1960s, the advent of digital computers transitioned classical control practices to modern control strategies, which enable linear time-invariant (LTI) systems in state space form [14]. State space models, however, still require linearization. The issue with linearization methods, as mentioned before, is that they are not generalized and do not function well when uncertainties/disturbances are presented.

The inverted pendulum on a cart has inspired many real-world applications. Some of the well-known ones are shown in figure 1.2 [15]. In robotics and control, self-balancing scooters, also known as Segways, and single-wheeled electric unicycles often used for personal transportation are all systems that stem from the concept of the inverted pendulum systems [16]. In addition, the inverted pendulum systems are systems developed based on human anatomy as human beings stabilizing themselves with their feet in the upright position and performing routine movements through their pivoting joints require the skill of balancing [16]. Further engineering discoveries in robotics based on the idea of an inverted pendulum system has led to the development of quadrupedal and bipedal humanoid robots [17, 18]. It is shown by [17] that the design of quadruped walking robots could be used to perform complicated tasks on rough and dangerous terrains beyond the reach of humans. Furthermore, [18] illustrated that the study of the inverted pendulum systems could be used to improve stability as well as performance on bipedal humanoid robots with minimal energy requirement. Other real-world applications that share similar characteristics as an inverted pendulum system are space launch vehicles and aircraft. It is shown by [19, 20] that attitude control designed to balance an inverted pendulum system can also be utilized to stabilize a rocket booster at takeoff and assist aircraft landing.

Figure 1.2: Inverted pendulum system applications [15].

Of course, common optimal control methods such as linear quadratic regulator (LQR) and sliding mode control (SMC) could be used to control most of the aforementioned systems. However, with the integration of AIML, overall system performance can be improved more efficiently despite the presence of uncertainties and nonlinearity through machine learning algorithms. For example, for the typical inverted pendulum on a cart problem, it takes barely 1.5 seconds to stabilize the cart position and "the pendulum angle swiftly converges to its desired trajectory when the SMC based Neural Networks (SMCNN) is applied" [20]. Many related studies have also shown that RL-based algorithms such as A2C and PPO can be implemented to control and balance an ideal inverted pendulum system, also known as the cart pole, with fast training time and low computational resource requirements [21–24]. In recent years, unmanned aerial vehicles (UAV) and drones have been widely used as testbeds to train reinforcement learning agents. For instance, combining adaptive control with a deep reinforcement learning approach, Caltech scientists designed autonomous quadrotor drones that learn to fly on their own in different wind conditions [25]. It is shown by [25] that the drones were able to learn in real time given only 12 minutes of flying data and subsequently map out the defined flight paths with great accuracy. Similarly, a robust reinforcement learning algorithm was developed for an autonomous UAV to perform vertical landing on a ship despite various difficult wind conditions [26]. The results demonstrated that the reinforcement learning algorithm was able to outperform a standard PID controller in recovering the UAV from

deviated positions as well as safe landing [26].

1.3 Project Proposal

This project proposed to stabilize and control an inverted pendulum on a cart in a simulated environment using reinforcement learning (RL) algorithms. Traditionally, system stability and control could be done by using different conventional control methods such as PID, LQR, and model predictive control (MPC), just to name a few. However, such processes require the development of exact mathematical models, which can present complexity and demand substantial computational effort for higher-order nonlinear systems. Moreover, the results generated by the classical methods can become unreliable when experiencing uncertain disturbances. Optimizing controllers in the traditional process can also be a difficult and time-consuming task. Therefore, two model-free RL-based algorithms called Advantage Actor-critic (A2C) and Proximal Policy Optimization (PPO) are examined for an inverted-pendulum-on-a-cart problem. The system proposed is an ideal frictionless system consisting of a cart with a pole attached to it. The pole is balanced by moving the cart underneath it with a force in the horizontal direction. Typically, the performance of the RL algorithms can be determined by various metrics such as the mean episode length, mean episode reward, and value loss. In addition, the results generated by the RL algorithms are then used to compare with those generated by a LQR controller to verify the robustness of the controllers in balancing the inverted pendulum system.

1.4 Methodology

The remainder of this project is organized as follows. Section 2 introduces the general concepts and the common terminologies of reinforcement learning. Section 3 sets up the problem and model, including deriving the governing equations of motion for the inverted pendulum on a cart. Section 4 develops the state space model and analyzes the open-loop stability and controllability of the inverted pendulum on a cart. Section 5 constructs the LQR controller and presents simulation results. Section 6 defines the reinforcement learning process using different reinforcement learning libraries. Section 7 presents and discusses the reinforcement learning results, and section 8 proposes future work and concludes the overall project.

2. Reinforcement Learning Concepts

2.1 Introduction to Reinforcement Learning

Reinforcement learning is a subset of ML that is often used to build learning agents that focus on achieving set goals through a heuristic approach. This means that a qualified agent can learn to solve problems, perform tasks, and improve performance on its own over time. According to Sutton and Barto [27], who were the two pioneers for reinforcement learning since the late 1970s, the premise of reinforcement learning is not to tell the agent what or what not to do but rather to allow it to learn from experience. Through trial-and-error interaction with its environment, the agent learns to choose actions that bring it the greatest profit. Often, one of the determining factors that aid the agent in making decisions is the reward received. The reward is a scalar value and can be positive or negative. Another term for negative reward is penalty. Both forms of reward are used to, as the name suggested, reinforce the behaviors of the agent as it learns. Teaching a dog a new trick and the idea of how children learn about the world on their own are two common examples that are often used to describe the basic idea of reinforcement learning [27, 28]. These two examples illustrate the concept of exploitation vs. exploration, which is one of the distinguishable features of reinforcement learning [27]. Teaching a dog to perform a new trick conditions it to do a task (e.g. go fetch a stick). Over time, the dog learns to exploit the act of fetching a stick when instructed to do so every time as it realizes that this is how it will be rewarded [27]. On the other hand, the idea of how children learn about the world on their own demonstrates the idea of exploration. Using their senses such as touching, tasting, and feeling to explore is how they first learn about their surrounding environments [27].

Reinforcement learning is different from other major types of ML methods such as supervised and unsupervised learning, as shown by figure 2.1 [29]. Unlike supervised learning, reinforcement learning does not learn from examples provided by a teacher [30]. Also different from supervised learning agents, reinforcement learning agents are not designed to recognize or classify patterns from well-defined data, nor do they try to match desired responses by tuning weighing parameters [30]. Another ML approach that also does not require its agents to learn from a teacher is unsupervised learning. However, unsupervised learning agents are designed to cluster information from well-known data and hence are less likely to improve on their own especially when challenged by unknown environments [30]. Reinforcement learning agents, on the other hand, are designed to learn and discover actions on their own through interaction with their environments, and they can do so without the help of a teacher. Reinforcement learning agents are also capable of making decisions and improving performance on their own even under uncertain, challenging environments, thus making them good candidates for any learning problems [30]. Due to the rise of AI/ML, these capabilities of reinforcement learning agents have recently brought widespread interest to engineers and researchers in the scientific community.

Figure 2.1: Three major types of machine learning [29].

The design architecture of a reinforcement learning algorithm mainly consists of the *agent*, the *environment*, and the *set of rules that governs the interactions between the agent and the environment*, as shown by figure 2.2 [31].

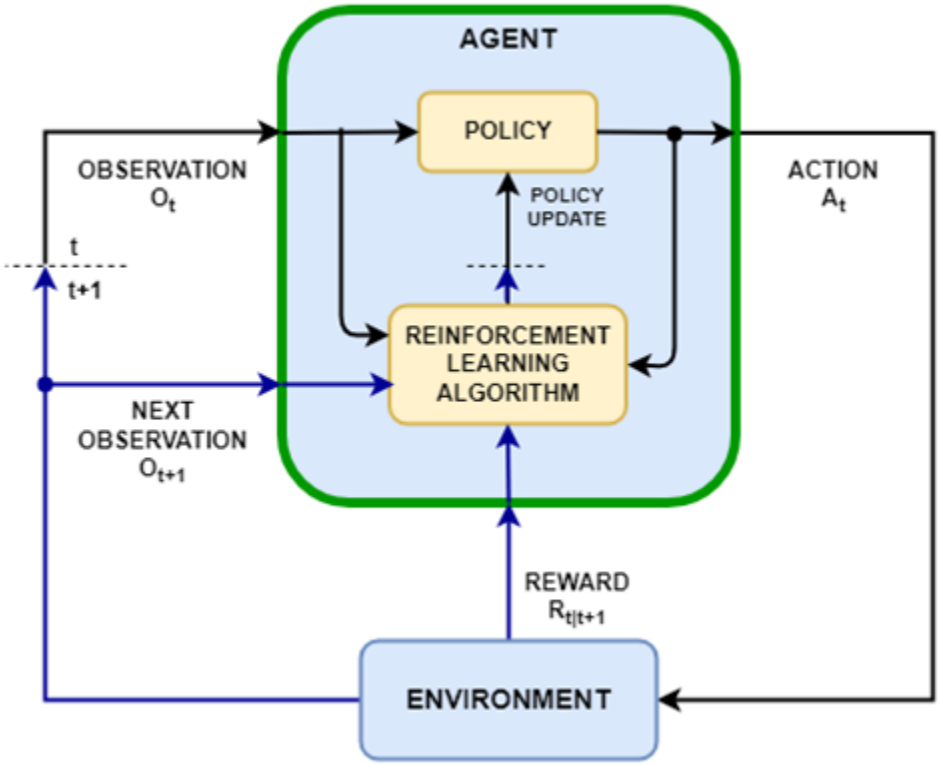


Figure 2.2: Reinforcement learning design architecture [31].

Figure 2.2 shows that given a certain number of training episodes, the environment takes in an action as input and generates observations and rewards as outputs. On the other hand, the input action passing to the environment becomes the output of the agent whereas the output components (observations and reward) generated by the environment become the inputs for the agent. Prior to setting up the inverted pendulum system, it is necessary to first introduce some of the key, relevant terms in order to grasp a better understanding of the framework of a reinforcement learning algorithm [27, 32].

2.1.1 Environment

The *environment* defines the initial states of the agent and the world in which the agent resides. For example, the states of an agent could be its coordinates, and they can be discrete or continuous. The environment can be designed based on experience or numerical models, and its job is to interact with the agent by sending observations to the agent followed by a reward signal to evaluate the actions taken by the agent. When the environment determines its next set of states based on a complete history of past occurrences, then the relationship can be defined by the following expression [27]:

$$P_r = (s_{t+1} = s', r_{t+1} = r \,|s_t, a_t, r_t, s_{t-1}, a_{t-1}, ..., r_1, s_0, a_0) \tag{2.1}$$

This indicates the probability of the environment ending up in the state s' with a reward r given all its past states s and actions a at each time step t [27]. Conversely, when the environment's response at $t+1$ is solely based on the action taken at t, then equation 2.1 can be simplified to the following [27]:

$$P_r = (s_{t+1} = s', r_{t+1} = r \,|s_t, a_t) \tag{2.2}$$

2.1.2 Agent

The *agent*, or sometimes referred to as the *algorithm*, is designed to choose actions that would help it maximize the cumulative reward [27]. It plays a key role as it is just like the brain where the process of decision-making takes place. The information that contains the observation states and action space is what enables the agent to map states to actions. The action space defines a series of possible input actions the agent can take to achieve a certain task. Similar to observation state space, action space can be discrete or continuous. The agent makes random actions initially and then adjusts according to the subsequent updates provided by the environment.

2.1.3 Policy

In general, the *policy* in reinforcement learning governs the agent's behavior [27]. Think of the policy as a strategy that highlights the optimal solutions for the agent. The policy is developed and can be updated by the agent according to the consequences of its actions and the information it receives from the environment. Over time through trial and error, the agent learns the best policy as it explores and exploits different actions. By continuously complying with its policy, the agent is capable of making good decisions and improving on its own. It is stated by [33] that if a policy π can be used to map a perceived state s to an action a that is to be taken when in that state, then the policy can be expressed as follows:

$$a = \pi(s) \tag{2.3}$$

2.1.4 Reward

The *reward or penalty* is a scalar indication that provides a positive or negative feedback to the agent. This number is provided by the environment, and it allows the agent to obtain feedback and evaluate its behavior from previous actions. The agent gets rewarded by taking good actions or gets penalized by taking bad ones. The goal of a reinforcement learning agent is always to seek maximum cumulative positive rewards in the long run, also known as value [27]. Note that the reward can be expressed as follow [33]:

$$R_k = r_{k+1} + \gamma r_{k+2} + \gamma^2 r_{k+3} + ... = \sum_{n=1}^{\infty} \gamma^n r_{k+n+1} \tag{2.4}$$

The sum of the reward can be scaled by multiplying the future reward by a discount factor γ, where $0 \leq \gamma \leq 1$. As γ approaches 1, the closer the future rewards are to the immediate rewards. If $\gamma = 1$, then future rewards weigh the same as immediate rewards. Conversely, if γ approaches 0, immediate rewards completely outweigh future rewards [33].

2.1.5 Value

The biggest difference between reward and *value* is that the former is related to immediate returns while the latter focuses on cumulative, long-term rewards. A good reinforcement learning agent seeks to take actions that would help accumulate the most rewards in the long run, even if it means to choose actions that result in low return initially. To obtain value, the reinforcement learning agent must be able to approximate all the rewards (i.e. current and future) for all the states. It is stated by [27] that value in reinforcement learning can be approximated by the following expression:

$$V(s) \leftarrow V(s) + \alpha[V(s') - V(s)] \tag{2.5}$$

Equation 2.5 estimates the value of a state based on the initial state s and the next state s'. α is called the step-size parameter, which is a small positive fraction used to define the learning rate of the agent [27]. This value function can be found by scaling the difference between the estimated values of two states in two different periods of time by a factor of α and adding the result to the estimated value of the initial state.

3. Inverted Pendulum System

3.1 System Overview

An inverted pendulum system mentioned in this project is shorthand for an inverted pendulum on a cart. It is a two-dimensional (2-D), nonlinear second-order system commonly used to demonstrate the concept of various modern control analysis techniques. Intuitively, control of an inverted pendulum system can be better understood by simply considering the act of balancing a broomstick in a hand. To balance the broomstick in a perfectly upright angle due to a disturbance, the hand underneath the broomstick has to catch the broomstick by moving toward the direction where the broomstick is falling. This indicates that the inverted pendulum system requires control effort similar to that of the broomstick task. Without the required control effort, the inverted pendulum system tends to immediately fall over due to a disturbance or naturally fall toward the ground due to the effect of gravity over time. Besides intuition, the motion of the inverted pendulum system can be analyzed and modeled by mathematical formulation and numerical simulation. Prior to developing the mathematical model of the system, proper assumptions need to be made. For details of the derivation, consider a two degree-of-freedom (DOF) inverted pendulum system as shown in figure 3.1.

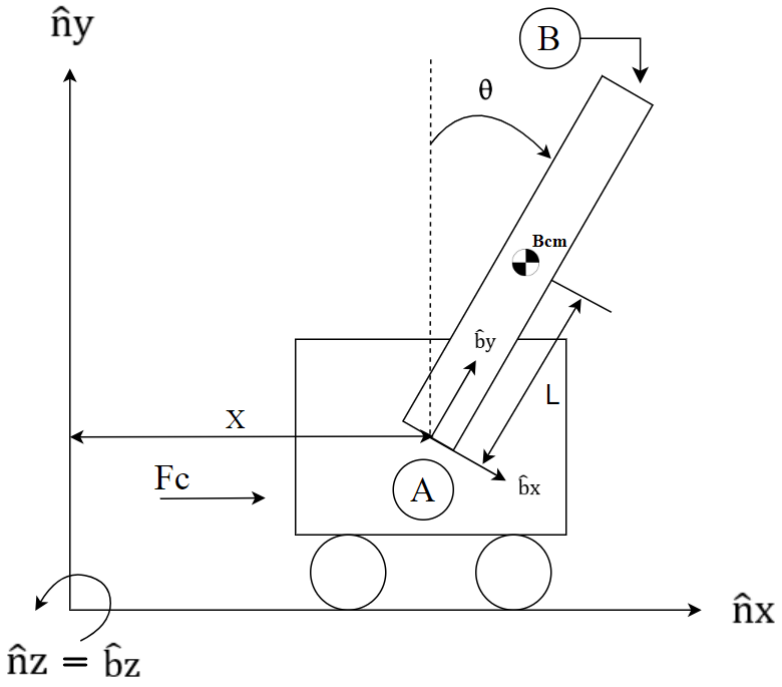


Figure 3.1: Two DOF inverted pendulum system.

3.2 Problem Definition and Assumptions

In this project, the inverted pendulum system consists of a cart (modeled as a particle or point mass) attached to a rigid rod. The rod is assumed to have a fixed length, and one of its ends is free

to rotate about a frictionless joint connecting the cart and the base of the rod. The cart controlled by a constant force in the horizontal direction can move left or right along the track without friction. Further assumptions are established for the inverted pendulum system as follows:

- Treat cart A as a particle fixed in $\hat{a}x$, $\hat{a}y$, and $\hat{a}z$.
- Reference frame B is fixed in the rod.
- Assume the mass of the cart stays constant.
- Assume the moment of inertia of the pendulum rod stays constant.
- Assume friction between the cart and the ground is negligible.
- Assume friction at the revolute joint connecting the cart and the rod is also negligible.

3.3 Coordinate System Reference Frames

Direction cosine matrix can be used to transformed different reference frames from the Newtonian frame (N-frame) to the body frame (B-frame), as shown by figure 3.2 and table 3.1:

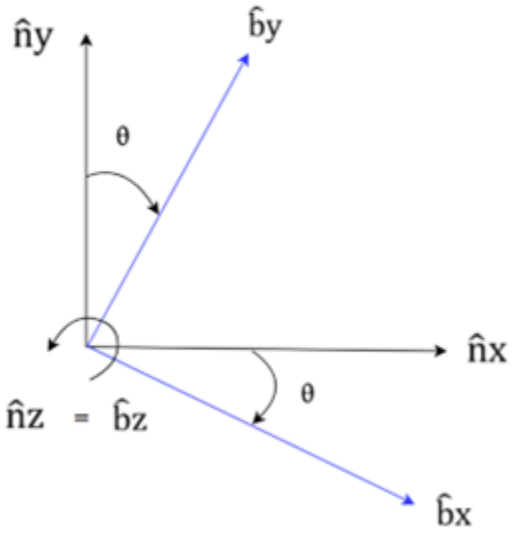


Figure 3.2: Direction cosine matrix from N-frame to B-frame.

Table 3.1: Transformation from N-frame to B-frame.

${}^{B}R^{N}$	$\hat{n}x$	$\hat{n}y$	$\hat{n}z$
$\hat{b}x$	$\cos\theta$	$-\sin\theta$	0
$\hat{b}y$	$\sin\theta$	$\cos\theta$	0
$\hat{b}z$	0	0	1

3.4 Governing Equations of Motion of the Inverted Pendulum System

According to figure 3.2, the angular velocity is:

$$^{N}\vec{\omega}^{B} = -\dot{\theta}\,\hat{b}_z \tag{3.1}$$

For detailed derivation of the motion of the inverted pendulum system, refer to Appendix A at the end.
The governing equations of motion of the inverted pendulum system in the horizontal and vertical direction are:

$$F_C = (m_A + m_B)\ddot{x} + m_B L(\ddot{\theta}\cos\theta - \dot{\theta}^2\sin\theta) \tag{3.2}$$

$$(m_A + m_B)g - N_c - m_B L(\ddot{\theta}\sin\theta + \dot{\theta}^2\cos\theta) = 0 \tag{3.3}$$

For the rotational motion of the inverted pendulum, the governing equation of motion is:

$$m_B g L\sin\theta = (I_{zz} + m_B L^2)\ddot{\theta} + m_B L\cos\theta\ddot{x} \tag{3.4}$$

Note that equation 3.3 represents the motion of the inverted pendulum system in the vertical direction. However, the system is not moving nor accelerating up or down in the vertical direction. Therefore, equation 3.3 can be dismissed in this case. Therefore, the complete governing equations of motion for the inverted pendulum system can be fully represented by equations 3.2 and 3.4, which are rewritten as follow:

$$F_C = (m_A + m_B)\ddot{x} + m_B L\ddot{\theta}\cos\theta - m_B L\dot{\theta}^2\sin\theta \tag{3.5}$$

$$m_B g L\sin\theta = (I_{zz} + m_B L^2)\ddot{\theta} + m_B L\cos\theta\ddot{x} \tag{3.6}$$

Observation can be further made to see that the derived governing equations of motion are nonlinear, second-order systems of differential equations.
Manipulating the final equations of motion in terms of x and θ on one side gives the following equations:

$$\ddot{\theta} = \frac{g\sin\theta + \cos\theta\left(\frac{-F_c - m_B L\dot{\theta}^2\sin\theta}{m_A + m_B}\right)}{L\left(\frac{4}{3} - \frac{m_B\cos^2\theta}{m_A + m_B}\right)} \tag{3.7}$$

$$\ddot{x} = \frac{F_C + m_B L(\dot{\theta}^2\sin\theta - \ddot{\theta}\cos\theta)}{m_A + m_B} \tag{3.8}$$

4. Open-Loop Stability and Controllability Analysis of the Inverted Pendulum System

4.1 Open-Loop System Stability and Controllability Overview

The open-loop system defines the bare frame/geometry of a system without any equipped feedback control effort. It is stated by [34] that open-loop control systems are the simplest yet imprecise form of controlling devices. Their lack of precision is due to the fact that the system inputs do not measure and adjust themselves based on the feedback provided by the output. In other words, the output and the input signals do not interact with each other to improve performance. Common examples of open-loop systems include washing machines, toasters, fans, microwaves, and traffic lights, just to name a few. The inverted pendulum system with any control effort is also another good example of an open-loop system, and it can be represented graphically by a block diagram, as depicted by figure 4.1 [35]:

Figure 4.1: Open-loop inverted pendulum system block diagram [35].

Figure 4.1 depicts a basic block diagram of an open-loop system that consists of an input, a process, and an output. Consider the inverted pendulum system. The input can be represented by the force exerted by a hand, while the process denotes the inverted pendulum system itself. The system responds based on the input signal, and the response is known as the output. An open-loop system is used to evaluate the inherent response of a system given any input. Besides the standard configuration shown by figure 4.1, another common open-loop system configuration used is illustrated by figure 4.2 [35]. Unlike the simplified version, this configuration includes a controller and actuator, and it is known as an open-loop control system. The controller receives an input and generates a controlled signal to direct the actuator to move the process. Although an open-loop control system includes a controller, it neither adjusts the output based on the input nor improves performance due to the absence of feedback control.

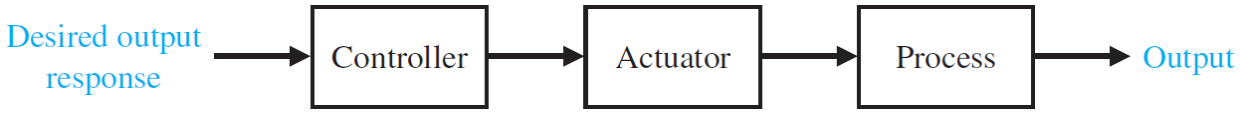


Figure 4.2: Open-loop control system for the inverted pendulum system [35].

Open-loop stability and controllability are important indicators used to determine the intrinsic characteristics of a system. They play key roles in developing fully stable and controllable systems and can be assessed both analytically as well as numerically through simulation. If an open-loop system is determined to be stable, then control effort is to be seen as supplementary.

There are essentially two types of open-loop stability. Static stability is the initial tendency of the system's response to disturbances, while dynamic stability defines the system's response to disturbances over time [36]. Since open-loop systems are designed to perform specific tasks, they are not calibrated to handle unknown disturbances [14]. Consequently, disturbances can cause an open-loop control system to transition from stable to unstable rapidly and indefinitely.

Typically, to address such issues, a controller can be designed to measure the difference between the desired and actual response resulting from a disturbance. This difference, known as error deviation, is fed back to the controller, allowing for corrections to both the error and the system's response. Prior to designing a controller, the open-loop controllability of a system must also be analyzed to ensure that the desired system states are indeed controllable.

An open-loop control system designed with a feedback control loop is referred to as a closed-loop control system, which will be further discussed in section 5. The scope of this chapter is to further develop the plant through linearization of the system so that the stability and controllability of the open-loop system can be evaluated.

4.2 Finding the Equilibrium Points for the Inverted Pendulum System

To linearize a nonlinear system, an equilibrium point must be identified about which the system can be linearized. Typically, in static stability, there are three types of equilibrium points: stable, unstable, and neutrally stable, as shown in figure 4.3 [36].

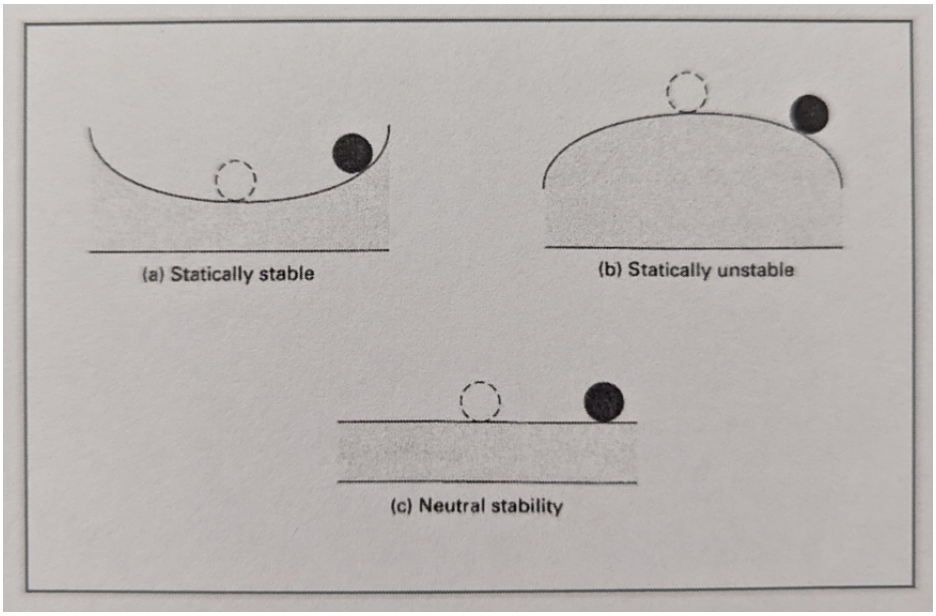


Figure 4.3: Classification of equilibrium points. (a) statically stable equilibrium, (b) statically unstable equilibrium, (c) neutrally stable equilibrium [36].

An equilibrium point is defined as stable if system motion can converge and return to its initial

equilibrium point after being perturbed. Conversely, if a system tends to diverge from the initial equilibrium point immediately and indefinitely once perturbed, then the initial equilibrium point is considered unstable. In contrast to stable and unstable equilibria, systems that are neutrally stable tend to remain in a certain state (e.g. sustained oscillation) after being perturbed, as displayed in figure 4.3c.

A good example of a statically stable system is a simple pendulum hanging directly downward, as shown in figure 4.4a. The simple pendulum remains at rest until it experiences a disturbance. When slightly perturbed, it tends to swing left or right as it seeks to return to its equilibrium point, which is directly downward.

In contrast, an inverted pendulum that is perfectly upright, as shown in figure 4.4b, is a statically unstable system because when it is slightly perturbed, it tends to roll away from the initial equilibrium point and does not return. However, the statically unstable point for an inverted pendulum is still considered as an equilibrium point; it represents an unstable equilibrium, while the statically stable point of a simple pendulum represents a stable equilibrium.

Therefore, naturally it makes sense to linearize the intended inverted pendulum system about the unstable equilibrium point defined at $\theta = 0^\circ$, where the inverted pendulum is perfectly upright.

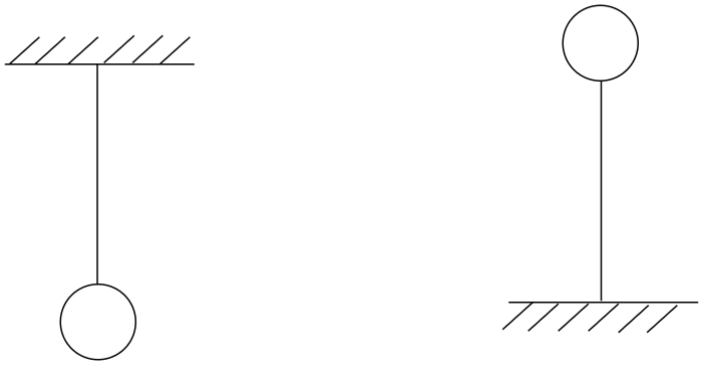


Figure 4.4: (a) Stable quilibrium point of a simple pendulum (left), (b) unstable equilibrium point of an inverted pendulum system (right).

Sometimes, an equilibrium point of a dynamic system is referred to as a steady-state condition. Given any control input $\vec{u}(t)$, the solution to the steady-state condition can be defined as follows [37]:

$$\begin{aligned} \vec{0} &= \vec{f}(\vec{x}, \vec{u}) \\ \vec{y} &= \vec{h}(\vec{x}, \vec{u}) \end{aligned} \tag{4.1}$$

To verify whether $\theta = 0^\circ$ is indeed an equilibrium point for the inverted pendulum system, substitute $\theta = 0$ in equations 3.5 and 3.6 to check if the results satisfy definition 4.1. Recall that equations 3.5 and 3.6, derived in section 3, are:

$$F_C = (m_A + m_B)\ddot{x} + m_B L\ddot{\theta}\cos\theta - m_B L\dot{\theta}^2 \sin\theta \tag{4.2}$$

$$m_B g L \sin\theta = (I_{zz} + m_B L^2)\ddot{\theta} + m_B L \cos\theta \ddot{x} \tag{4.3}$$

Equations 4.2 and 4.3 represent a system of nonlinear, coupled second-order ordinary differential equations, with $\ddot{x}$ and $\ddot{\theta}$ representing the highest derivatives. However, equations 4.2 and 4.3 can be decoupled and expressed in explicit forms. To do so, first solve equation 4.3 for $\ddot{\theta}$ to obtain:

$$\ddot{\theta} = \frac{m_B g L \sin\theta - m_B L \cos\theta \ddot{x}}{(I_{zz} + m_B L^2)} \tag{4.4}$$

Next, substitute equation 4.4 into equation 4.2, perform the necessary algebraic manipulations to solve for $\ddot{x}$ explicitly, and obtain:

$$\begin{aligned} \ddot{x} = &\frac{(I_{zz} + m_B L^2) m_B L \sin\theta}{I_{zz}(m_A + m_B) + m_A m_B L^2 + m_B^2 L^2 \sin\theta^2} \dot{\theta}^2 - \frac{m_B^2 L^2 \cos\theta \sin\theta g}{I_{zz}(m_A + m_B) + m_A m_B L^2 + m_B^2 L^2 \sin\theta^2} \\ &+ \frac{(I_{zz} + m_B L^2)}{I_{zz}(m_A + m_B) + m_A m_B L^2 + m_B^2 L^2 \sin\theta^2} F_C \end{aligned} \tag{4.5}$$

Equation 4.5 thus becomes a function of θ, $\dot{\theta}$, and F_C only. A similar procedure can be used to find $\ddot{\theta}$, with the result shown as follow:

$$\begin{aligned} \ddot{\theta} = &\frac{-m_B^2 L^2 \sin\theta \cos\theta}{I_{zz}(m_A + m_B) + m_A m_B L^2 + m_B^2 L^2 \sin\theta^2} \dot{\theta}^2 + \frac{(m_A + m_B) m_B g L \sin\theta}{I_{zz}(m_A + m_B) + m_A m_B L^2 + m_B^2 L^2 \sin\theta^2} \\ &- \frac{m_B L \cos\theta}{I_{zz}(m_A + m_B) + m_A m_B L^2 + m_B^2 L^2 \sin\theta^2} F_C \end{aligned} \tag{4.6}$$

Subsequently, let $\theta = 0^\circ$ and the control input $F_C = 0$ and substitute these values in. Then it can be seen that equations 4.5 and 4.6 indeed yield zero, satisfying definition 4.1. Therefore, $\theta = 0^\circ$ is a solution to the steady-state equations. In other words, it is an equilibrium point about which the system can be linearized.

4.3 Linearization of the Inverted Pendulum System

Equations 4.2 and 4.3, corresponding to equations 3.5 and 3.6 derived in section 3, represent the exact nonlinear equations of motion, which can be linearized using the perturbation method. This method involves redefining the varying parameters "as the sum of a steady-state value and a perturbation quantity" [36]. The varying parameters in the system of equations are u, $\ddot{x}$, θ, $\dot{\theta}$, and $\ddot{\theta}$, where u is the force control input applied to control the inverted pendulum system, $\ddot{x}$ is the second derivative of x with respect to time, θ is the angular displacement of the pole, $\dot{\theta}$ is first derivative of θ with respect to time, and $\ddot{\theta}$ is the second derivative of θ with respect to time. Each can be rewritten as follow:

$$\begin{aligned} U &= U_1 + u \\ \ddot{x} &= \ddot{X}_1 + \ddot{x} \\ \theta &= \Theta_1 + \theta \\ \dot{\theta} &= \dot{\Theta}_1 + \dot{\theta} \\ \ddot{\theta} &= \ddot{\Theta}_1 + \ddot{\theta} \end{aligned} \tag{4.7}$$

Letters with a subscript "1" represent the steady-state values, while those without subscripts denote the perturbation values. The redefined parameters can then be substituted into the exact nonlinear

system of equations accordingly. The detailed linearization process for the equations of motion of the inverted pendulum system is documented in Appendix B.
The linearized equations of motion for the inverted pendulum system, as derived in Appendix B, are thus:

$$u = (m_A + m_B)\ddot{x} + m_B L\cos\Theta_1\ddot{\theta} \tag{4.8}$$

$$m_B g L\cos\Theta_1\theta = (I_{zz} + m_B L^2)\ddot{\theta} + m_B L\cos\Theta_1\ddot{x} \tag{4.9}$$

Since the system of equations can be linearized about the steady-state condition $\Theta_1 = 0^\circ$, equations 4.8 and 4.9 can be further simplified to:

$$\cos(\Theta_1) = \cos(0) = 1 \tag{4.10}$$

$$u = (m_A + m_B)\ddot{x} + m_B L\ddot{\theta} \tag{4.11}$$

$$m_B g L\theta = (I_{zz} + m_B L^2)\ddot{\theta} + m_B L\ddot{x} \tag{4.12}$$

Equations 4.11 and 4.12 thus represent the governing equations of motion for the inverted pendulum system linearized about a steady-state condition at $\Theta_1 = 0^\circ$.

4.4 State Space System Definition and Representation

Now that the nonlinear equations of motion for the inverted pendulum system have been linearized, the state space model can be defined for the system. However, before diving in, a brief background of the state space system is discussed. Given a single-input, multiple-output (SIMO) system, such as the inverted pendulum system in this project, the nonlinear state space model can be expressed in this form [36]:

$$\dot{\vec{x}}(t) = \vec{f}(\vec{x}(t), \vec{u}(t)) \tag{4.13}$$

$$\vec{y}(t) = \vec{h}(\vec{x}(t), \vec{u}(t)) \tag{4.14}$$

where $\vec{x}(t)$ and $\vec{u}(t)$ are n-dimensional vector-valued functions that represent the states and control input changing with respect to time. In general, equations 4.13 and 4.14 represent the dynamics of the system evolving over time. In particular, the former defines the time derivative of the state and the latter defines the output at time t as a function of $\vec{x}(t)$ and $\vec{u}(t)$. If the differential equations $\vec{f}$ and $\vec{h}$ are linear, time-invariant (LTI) functions, then equation 4.13 and 4.14 can instead be rewritten in a LTI state space form shown as follows [36]:

$$\dot{\vec{x}} = A\vec{x} + B\vec{u} \tag{4.15}$$

$$\vec{y} = C\vec{x} + D\vec{u} \tag{4.16}$$

where A, B, C, and D are LTI matrices corresponding to the state, input, output, and feedforward characteristics of the system, respectively. D is often a zero matrix because the sensor is usually not directly linked to the control input. Lowercase $\vec{u}$ represents the control input variable, which by convention symbolizes F_c and will be used in place of F_c henceforth. Equations 4.15 and 4.16 are thus the LTI state space representation of equations 4.13 and 4.14, respectively. The state space model can also be graphically represented by a block diagram, as shown in figure 4.5:

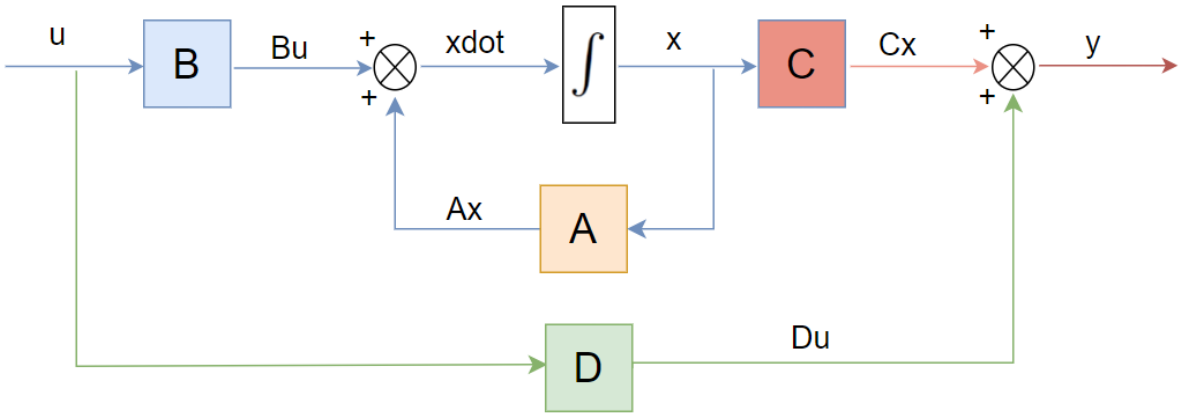


Figure 4.5: State space block diagram.

To develop equations 4.11 and 4.12 into state space form, they must first be expressed in explicit forms. The procedures are similar to those used to derive equations 4.5 and 4.6 in subsection 4.2. Recall that equations 4.11 and 4.12 can first be decoupled through algebraic manipulation, and the explicit form of equation 4.12 can be obtained as follows:

$$m_B g L\theta - m_B L\ddot{x} = (I_{zz} + m_B L^2)\ddot{\theta} \tag{4.17}$$

$$\frac{m_B g L\theta - m_B L\ddot{x}}{(I_{zz} + m_B L^2)} = \ddot{\theta} \tag{4.18}$$

Subsequently, substitute equation 4.18 into equation 4.11, perform necessary algebraic manipulations to solve for $\ddot{x}$, and express $\ddot{x}$ as a function of θ and u:

$$\ddot{x} = \frac{-m_B^2 L^2 g}{I_{zz}(m_A + m_B) + m_A m_B L^2}\theta + \frac{(I_{zz} + m_B L^2)}{I_{zz}(m_A + m_B) + m_A m_B L^2}u \tag{4.19}$$

Similarly, to solve for $\ddot{\theta}$ as a function of θ and u, follow the same procedure and obtain:

$$\ddot{\theta} = \frac{(m_A + m_B)m_B g L}{I_{zz}(m_A + m_B) + m_A m_B L^2}\theta + \frac{-m_B L}{I_{zz}(m_A + m_B) + m_A m_B L^2}u \tag{4.20}$$

Equations 4.19 and 4.20 explicitly represent the cart acceleration and the angular rate of the inverted pendulum system as they evolve over time. They define the states of the inverted pendulum system by linearizing about a steady state condition at $\Theta_1 = 0$.
In state space representation,

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{-m_B^2 L^2 g}{I_{zz}(m_A+m_B)+m_A m_B L^2} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{(m_A+m_B)m_B g L}{I_{zz}(m_A+m_B)+m_A m_B L^2} & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{(I_{zz}+m_B L^2)}{I_{zz}(m_A+m_B)+m_A m_B L^2} \\ 0 \\ \frac{-m_B L}{I_{zz}(m_A+m_B)+m_A m_B L^2} \end{bmatrix} [u] \tag{4.21}$$

The desired outputs of interest are x, $\dot{x}$, θ, and $\dot{\theta}$; therefore, the C and D matrices are:

$$\begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} u \end{bmatrix} \tag{4.22}$$

Equations 4.19 and 4.20 model the linearized dynamics of the inverted pendulum system and can be used to verify with the nonlinear equations 4.5 and 4.6 to see whether the former is a good representation of the latter. To do so, numerical integration algorithms can be applied. One of the most accurate numerical integration algorithms used to solve systems of ordinary differential equations (ODEs) is MATLAB's ODE 45 integration function, or the Python equivalent solve_ivp function. Figures 4.6 and 4.7 present the Python simulation results, comparing the responses of x and θ of the nonlinear and linearized system given a control input u and the defined initial conditions x_0. The Python program is documented in Appendix C.

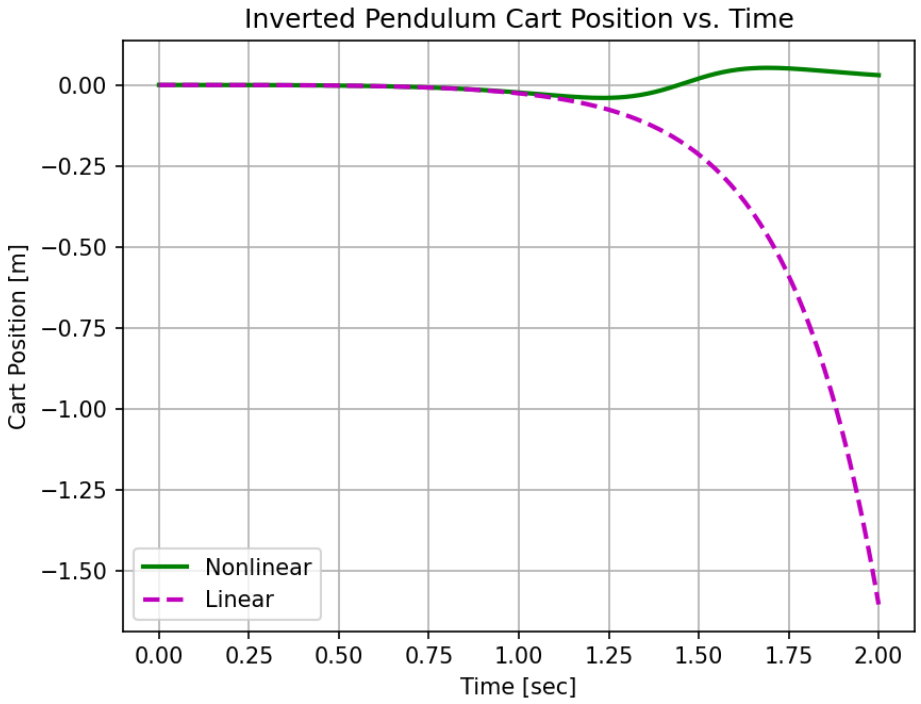


Figure 4.6: Cart position response of nonlinear vs. linearized inverted pendulum system.

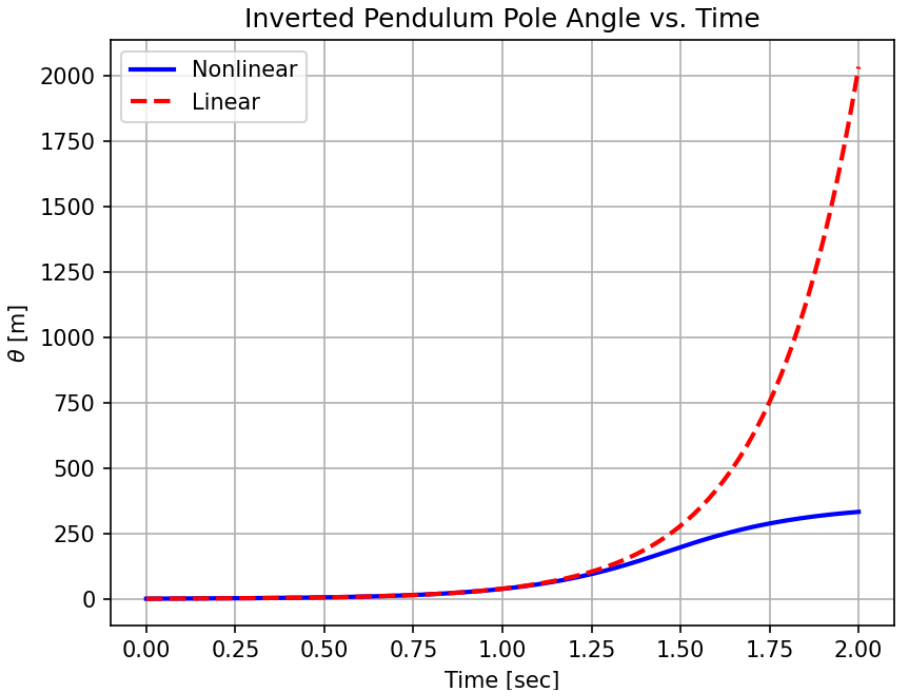


Figure 4.7: Pole angle response of nonlinear vs. linearized inverted pendulum system.

Figures 4.6 and 4.7 illustrate the response of an inverted pendulum system governed solely by natural forces, without any control input. Both the nonlinear and linearized models assume ideal conditions, neglecting factors such as friction and air drag. The only external force affecting both systems is gravity. To further simplify the analysis, the initial conditions for the four states x, $\dot{x}$, θ, $\dot{\theta}$ are set to 0, 0, 0, and 0.1, respectively. In addition, figures 4.6 and 4.7 demonstrate that the linearized state space model accurately approximates the nonlinear model for a short time interval. This behavior is expected because the inverted pendulum system begins from an unstable equilibrium.

4.5 Open-Loop Stability and Controllability Analysis

After the linearized equations of motion are derived with respect to $\theta = 0°$ in state space form, real values, as shown by table 4.1, can be assigned to the variables to determine the stability and controllability of the open-loop system.

Table 4.1: Inverted pendulum system parameters

Parameters	Description	Value
m_A	Mass of the cart	1 $[kg]$
m_B	Mass of the pendulum	0.1 $[kg]$
g	Gravitational acceleration	9.8 $[\frac{m}{s^2}]$
L	Half of the length of the pendulum	0.5 $[m]$
I_{zz}	Mass moment of inertia of the pendulum about B_{cm}	$\frac{1}{3}m_B L^2$ $[kgm^2]$

Subsequently, the A, B, C, and D matrices, as well as the state space model, can be constructed using the Python Control Systems Library. The results are shown as follow:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -0.7171 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 15.7756 & 0 \end{bmatrix} \tag{4.23}$$

$$B = \begin{bmatrix} 0 \\ 0.9756 \\ 0 \\ -1.4634 \end{bmatrix} \tag{4.24}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.25}$$

$$D = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.26}$$

4.5.1 System Stability

The open-loop stability of the inverted pendulum system can be determined by finding the pole locations using the damp function provided by the Python Control Systems Library. The result is shown in table 4.2.

Table 4.2: Open-loop system pole locations and damping characteristics

Eigenvalue (pole)	Damping	Frequency (rad/seconds)
0	1	0
0	1	0
0 + 3.972j	-0	3.972
0 - 3.972j	-0	3.972

A control system is stable when it has poles with negative real part, unstable when it has poles with positive real part, and neutrally stable when it has poles on the imaginary axis. This is because poles with negative real part signify that they are located in the left half-plane (LHP), while poles with positive real part are in the right half-plane (RHP). Zero poles, as shown by table 4.2, indicate that the poles are located at the origin. This type of poles are neutrally stable, but they are on the verge of becoming unstable and can transition to the scenario shown in figure 4.3b when a slight perturbation is introduced. Since the open-loop system can diverge from its equilibrium with even minor perturbations, a controller is required to stabilize the system, and its design is further discussed in section 5.

In addition to pole locations, the open-loop stability of the inverted pendulum system can also be determined graphically. Consider figures 4.8 through 4.11 shown as follows:

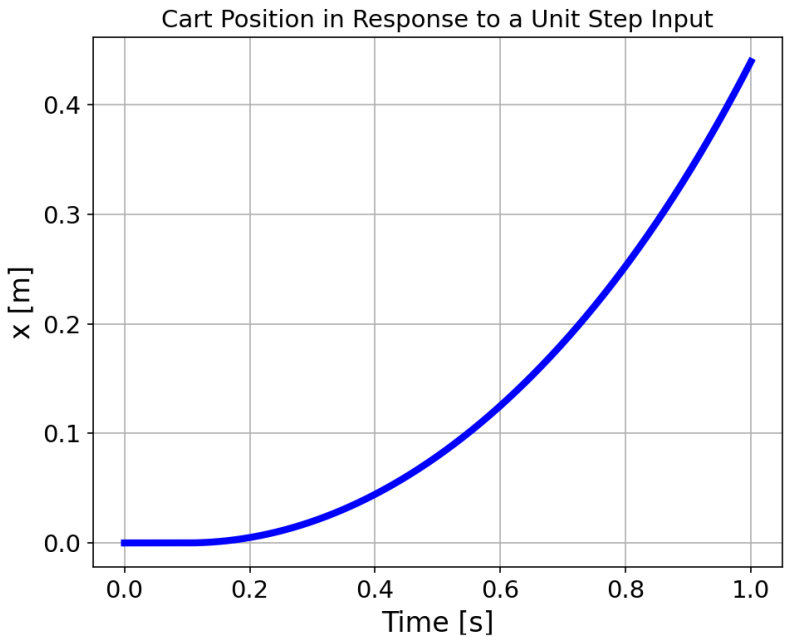


Figure 4.8: Cart position in response to a unit step input

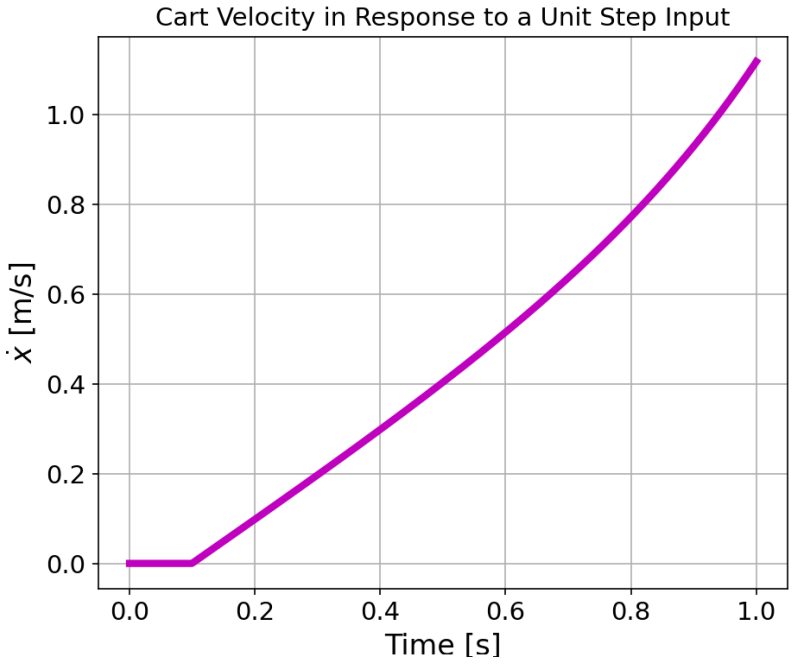


Figure 4.9: Cart velocity in response to a unit step input

Figures 4.8 and 4.9 demonstrate the dynamics of the cart's motion and velocity in response to a unit step input shown by figure 4.12. It is clear to see that the responses of x and $\dot{x}$ tend to diverge immediately without any control effort.

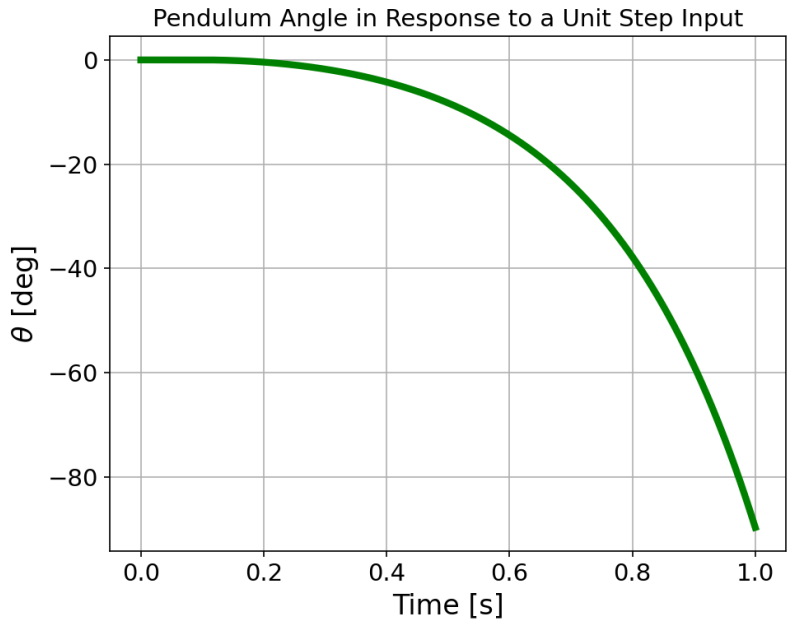


Figure 4.10: Pendulum angle in response to a unit step input

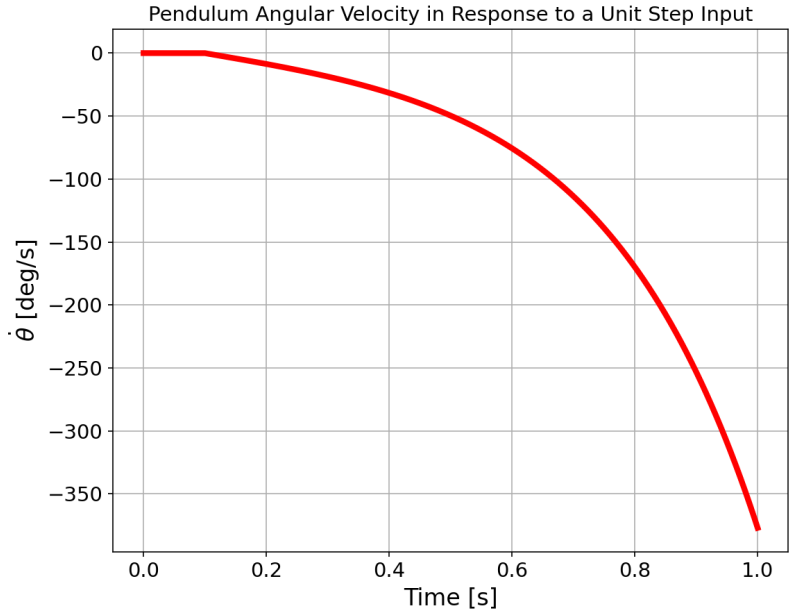


Figure 4.11: Pendulum angle in response to a unit step input

Similarly, figures 4.10 and 4.11 show that the open-loop responses of the pole angle (θ) and pole rate ($\dot{\theta}$) also tend to diverge immediately given a unit step input. These results are expected because the open-loop inverted pendulum system is neutrally stable at rest but tends to respond poorly to slight perturbation.

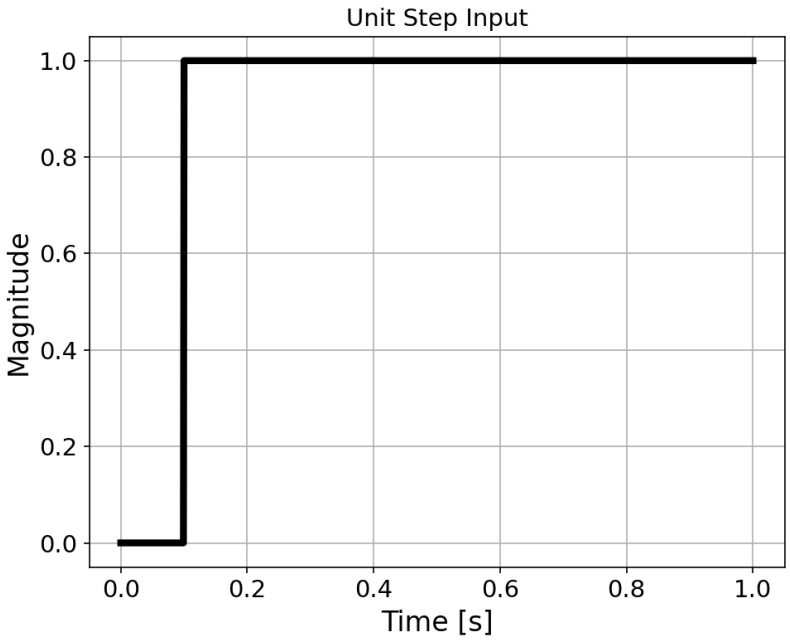


Figure 4.12: Unit step input

4.5.2 System Controllability

The controllability of a system is defined as the proof of existence of a finite control input signal that can transfer the system from any initial states to any final states in a finite amount of time [36]. To determine whether the inverted pendulum system is controllable, the Kalman controllability matrix is defined and implemented as follows:

$$Co = \begin{bmatrix} B & AB & A^2B & \dots & A^{n-1}B \end{bmatrix} \tag{4.27}$$

where n is the number of states. Given four states, Co thus becomes:

$$Co = \begin{bmatrix} B & AB & A^2B & A^3B \end{bmatrix} = \begin{bmatrix} 0 & 0.9756 & 0 & 1.0494 \\ 0.9756 & 0 & 1.0494 & 0 \\ 0 & -1.4634 & 0 & -23.0863 \\ -1.4634 & 0 & -23.0863 & 0 \end{bmatrix} \tag{4.28}$$

A system is fully controllable if the rank of Co equals n. In other words, the matrix Co is full rank. Since the rank of Co is equal to $n = 4$, the system is indeed fully controllable. Subsequently, control effort can be considered. Appendix D documents the Python code used to analyze the open-loop stability and controllability of the inverted pendulum system.

5. Inverted Pendulum System Control by Linear Quadratic Regulator

5.1 Closed-Loop Feedback Control System

Determined to be neutrally stable yet fully controllable in section 4, the inverted pendulum system is on the verge of becoming unstable given any perturbation, and as a result, a control mechanism is required to balance the system. In control engineering, the inverted pendulum system has been commonly used to test different control methodologies, which are often implemented by closed-loop control systems, as the one shown in figure 5.1 [35].

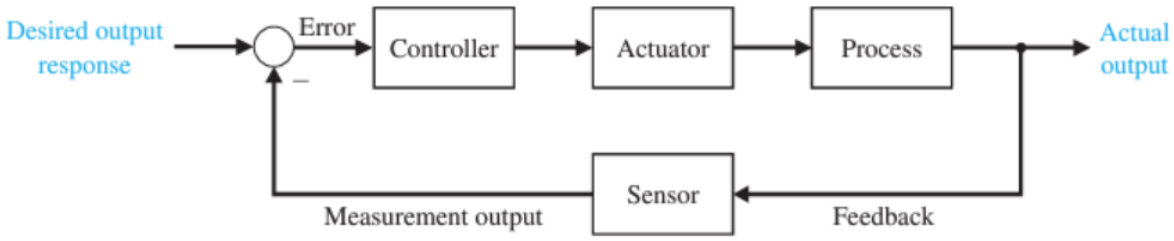


Figure 5.1: Closed-loop feedback control system [35].

Note that this is only one configuration. There are other variations of block diagrams that can be used to represent closed-loop feedback control systems.

Unlike the open-loop control systems discussed in section 4, closed-loop control systems are designed to measure the output through a sensor feedback signal, which is then used to compare with the desired signal and form an error signal as input for the controller, ultimately enabling the controller to adjust the actuator and achieve system stability [35]. Therefore, closed-loop control systems are also known as feedback control systems. Although open-loop control systems have simpler constructions than feedback control systems, the latter are less susceptible to unpredictable internal and external disturbances and can thus ensure desired system performance [14, 35]. This section introduced the inverted pendulum system modeled as a SIMO system that outputs x, $\dot{x}$, θ, and $\dot{\theta}$, and a state feedback controller called the linear quadratic regulator (LQR) was designed to stabilize the system.

5.2 Overview of the LQR Control Methodology

LQR is an optimal control system designed to stabilize LTI state space systems in the time domain. For nonlinear systems, such as the inverted pendulum system, LQR is also applicable as long as the system can be linearized about an equilibrium point [38]. The process to design a LQR controller begins with an understanding of it. First, recall a LTI system of the form:

$$\begin{aligned} \dot{\vec{x}} &= A\vec{x} + B\vec{u} \\ \vec{y} &= C\vec{x} + D\vec{u} \end{aligned} \tag{5.1}$$

where the control input $\vec{u}(t)$ can be defined by a linear, optimal control law of the form:

$$\vec{u} = -K\vec{x} \tag{5.2}$$

The goal is to find a gain matrix K to minimize the cost function (J), also known as performance index, of the form:

$$J = \int_0^\infty (x^T Qx + u^T Ru)\, dt \tag{5.3}$$

Mathematically, Q is a positive semi-definite state-weighting matrix, and R is a positive definite control-weighting matrix. In addition, both Q and R are symmetric matrices. $x^T Qx$ is related to the transient energy cost while $u^T Ru$ represents the control energy cost. $K = R^{-1}B^T P$ is the optimal feedback control gain matrix, where P is a real, positive, and symmetric matrix defined by solving the algebraic Ricatti equation, which has the form:

$$A^T P + PA - PBR^{-1}B^T P + Q = 0 \tag{5.4}$$

In control engineering, the Q and R matrices define the weights associated with the state variables and control effort, respectively. Together, the Q and R matrices are also known as the weighting matrices and are used to minimize the cost function J. It is stated by [39] that a large value of Q and R will penalize the control system in response to any small changes on the specified state variables $\vec{x}(t)$ and control effort $\vec{u}(t)$, respectively. These acts are intended to stabilize the state variables of interest and reduce control cost. The relationship between the state space system and the LQR controller can be represented by the block diagram in figure 5.2 [39]:

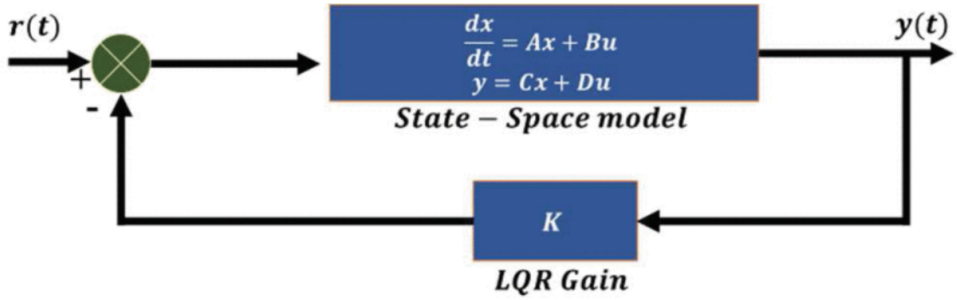


Figure 5.2: Block diagram of an LQR feedback controller [39].

Similar to figure 5.1, figure 5.2 depicts a closed-loop feedback control system. Indeed, LQR is a form of state feedback control system. However, in contrast to other types of closed-loop feedback control systems such as a PID, which relies on the tuning of a gain matrix K only such that the closed-loop poles can be placed anywhere in the left half plane, LQR focuses on the specification of the Q and R weighting matrices. Proper selection of Q and R is important because it can generates the best K gain matrix that will minimize the cost function J. It is stated by [38] that the K gain matrix of LQR serves to reinforce the controller to stabilize the system, and it can be easily computed using the A, B, Q, and R matrices and MATLAB. Even though LQR involves the tuning of two weighting matrices, Q and R, it is an intuitive and systematic control authority that guarantees optimal control. Designed for stability only and often used to optimize

control systems, LQR demonstrates a delicate balancing act between maximizing state benefits and minimizing control effort [39]. As mentioned earlier, one of the few notable caveats is that LQR, as its name implies, only applies to LTI state space systems or nonlinear state space systems that can be linearized about a trim point [38].

5.3 Design of LQR for the Inverted Pendulum System Using MATLAB/SIMULINK

To design an LQR controller for the inverted pendulum system, a linearized mathematical model of the system is required. This state space model of the inverted pendulum system was developed by using the same A, B, C, and D matrices found in section 4, and after plugging in the values defined in table 4.1, it can be expressed in the form of equation 5.1, as shown:

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -0.7171 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 15.7756 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ 0.9756 \\ 0 \\ -1.4634 \end{bmatrix} \begin{bmatrix} u \end{bmatrix} \tag{5.5}$$

$$\begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} u \end{bmatrix} \tag{5.6}$$

Once the state space model is developed, the controllability of the system must be determined. Since the inverted pendulum system was confirmed to be neutrally stable and fully controllable in section 4, the LQR controller is designed in SIMULINK, as shown in figure 5.3:

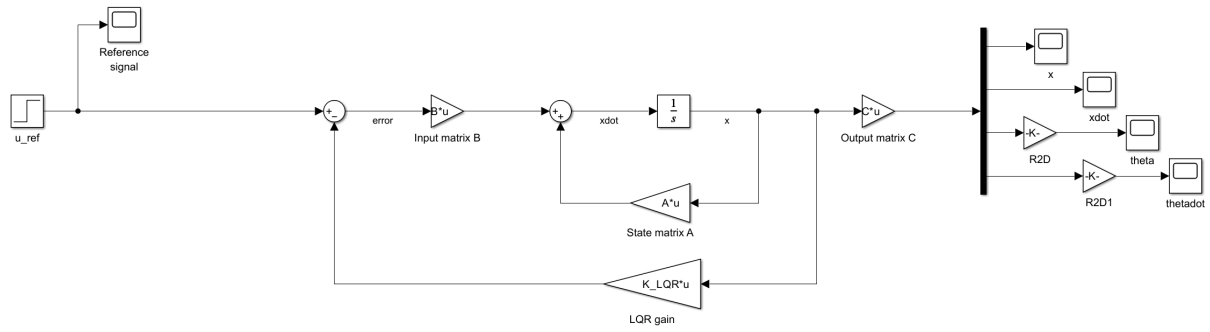


Figure 5.3: SIMULINK model of the LQR controller for the inverted pendulum system.

Figure 5.3 displays the block diagram of an LQR controller that is used to improve the stability and performance of the inverted pendulum system. It consists of a negative feedback control loop scaled by an optimal controller gain matrix K_{LQR} and used to feed the full states back to compare with the reference signal and drive the control input. To design for stability, the reference signal (u_{ref}) is set to zero. The system was simulated for 10 seconds with a fixed-step size of 0.01 and initial conditions $x_0 = \begin{bmatrix} 0 & 0 & \frac{\pi}{4} & 0 \end{bmatrix}^T$. Furthermore, the states of the inverted pendulum system were constrained by four design criteria defined in table 5.1, which were further used to validate the performance of the LQR controller.

Table 5.1: Design criteria for LQR controller

Design Parameters	Min	Max
x	-2.4 m	2.4 m
$\dot{x}$	$-\infty$	∞
θ	-12° ($\sim$ -0.2094 rad)	12° ($\sim$ 0.2094 rad)
$\dot{\theta}$	$-\infty$	∞

These values were chosen based on the studies achieved by [30], where the authors specified the thresholds for x, $\dot{x}$, θ, and $\dot{\theta}$ to provide realism to the control problem. Note that $\dot{x}$ and $\dot{\theta}$ are less constrained than x and θ because the goal of the LQR controller is to stabilize the inverted pendulum system, and this stability can be emphasized solely by the results of the pendulum angle and cart displacement. A successful LQR controller is capable of controlling these states to remain within their corresponding bounds. Based on the design criteria, a Q matrix is constructed as follows:

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.7}$$

where each diagonal entry in the Q matrix represents a weight assigned to each state, and out of the four states, θ is given a weight ten times more than the other states. This instructs the LQR controller to focus on maintaining the pendulum in a perfectly upright position at $\theta = 0^\circ$. As mentioned earlier, a large value of weight in the Q matrix will penalize the controller given any small changes in the specified state, which is θ in this case. Since there is only one control input u, R is a scalar and is set as follows:

$$R = 0.001 \tag{5.8}$$

Similar to the values in the Q matrix, a large value of R is intended to penalize the control effort in order to reduce energy cost and vice versa. The optimal K_{LQR} gain matrix can then be obtained by inputting the A, B, Q and R matrices into the MATLAB lqr() command, and the results are presented as follows:

$$K_{LQR} = \begin{bmatrix} -31.6628 & -53.0327 & -291.6252 & -77.9484 \end{bmatrix} \tag{5.9}$$

The optimal gain matrix K_{LQR} serves as a weighting factor, which is used to scale the full states and feed them back to compare with the reference signal. Subsequently, the error difference between the feedback and reference signal is supplied to the control input to determine the ideal amount of control effort/energy needed to stabilize the system. Note that since θ was given a greater weight in the Q matrix, the result was reflected by the greatest scalar magnitude 291.6252 in the K_{LQR} matrix.

5.4 Simulations and Results

According to the SIMULINK model shown in figure 5.3, the simulation was carried out using a MATLAB script documented in Appendix E, and the results were presented in figure 5.4:

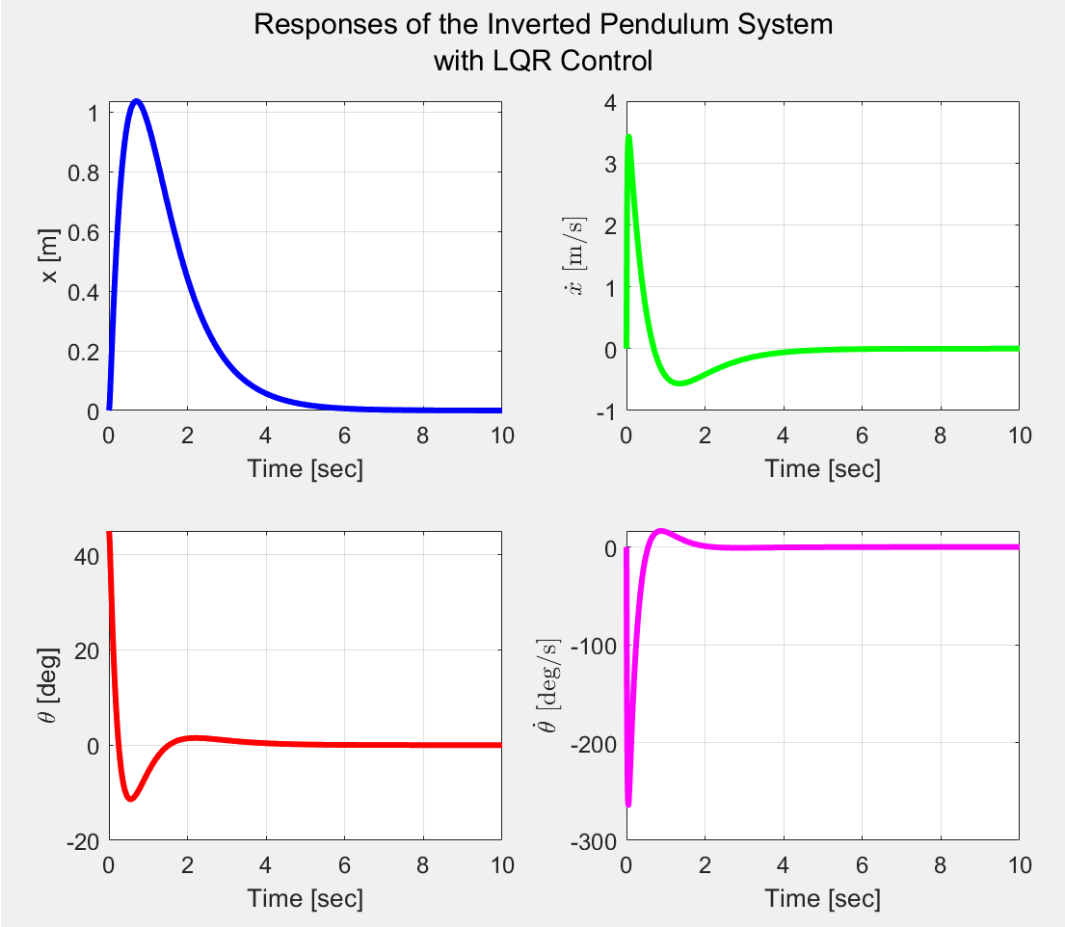


Figure 5.4: Simulation responses of the inverted pendulum system via LQR control.

The performance of the inverted pendulum system controlled by a LQR can be simply and quickly evaluated through visual interpretation of the simulation results. For example, it can be observed by figure 5.4 that the full-state closed-loop responses of the inverted pendulum system controlled by the designed LQR were able to reach stability with a settling time of less than 10 seconds. More specifically, $\dot{x}$ and $\dot{\theta}$ converged quickly in about 4 and 2 seconds after initially experiencing some overshoots/undershoots. But more importantly, the responses of x and θ were able to reach steady-state in about 6 and 4 seconds, respectively. Even though the response of x experienced an overshoot of approximately 1.04 meters at around 0.7 seconds, it was still well within the limits defined by the design criteria specified in table 5.1. Similarly, the response θ experienced an undershoot of about -11.38° at around 0.5 seconds without exceeding the design criteria imposed.

Besides graphical representation, the poles of the closed-loop system were checked to verify the system stability, and the outcomes are illustrated in table 5.2:

Table 5.2: Closed-loop pole locations and system characteristics.

Pole	Damping	Frequency(rad/seconds)	Time Constants (seconds)
-1.07	1	1.07	9.35e-01
-2.75 + 1.62e-01i	9.98e-01	2.76	3.63e-01
-2.75 - 1.62e-01i	9.98e-01	2.76	3.63e-01
-5.58e+01	1	5.58e+01	1.79e-02

Since all the poles, including the complex conjugate pairs, contain negative real values, the closed-loop inverted pendulum system designed with an LQR controller is thus stable. The simulation results and the closed-loop characteristics verify that the LQR controller indeed improved the stability and performance of the inverted pendulum system, ultimately proving the robustness of the LQR controller.

6. Development of Inverted Pendulum System Control by Reinforcement Learning (RL)

6.1 Steps and Best Practices for Implementing RL

The implementation of RL can be an intricate process. However, a general, high-level process to design and deploy a RL algorithm can be followed by the flow chart shown by figure 6.1 [40]. Each step of the process is further elaborated in the following subsections.

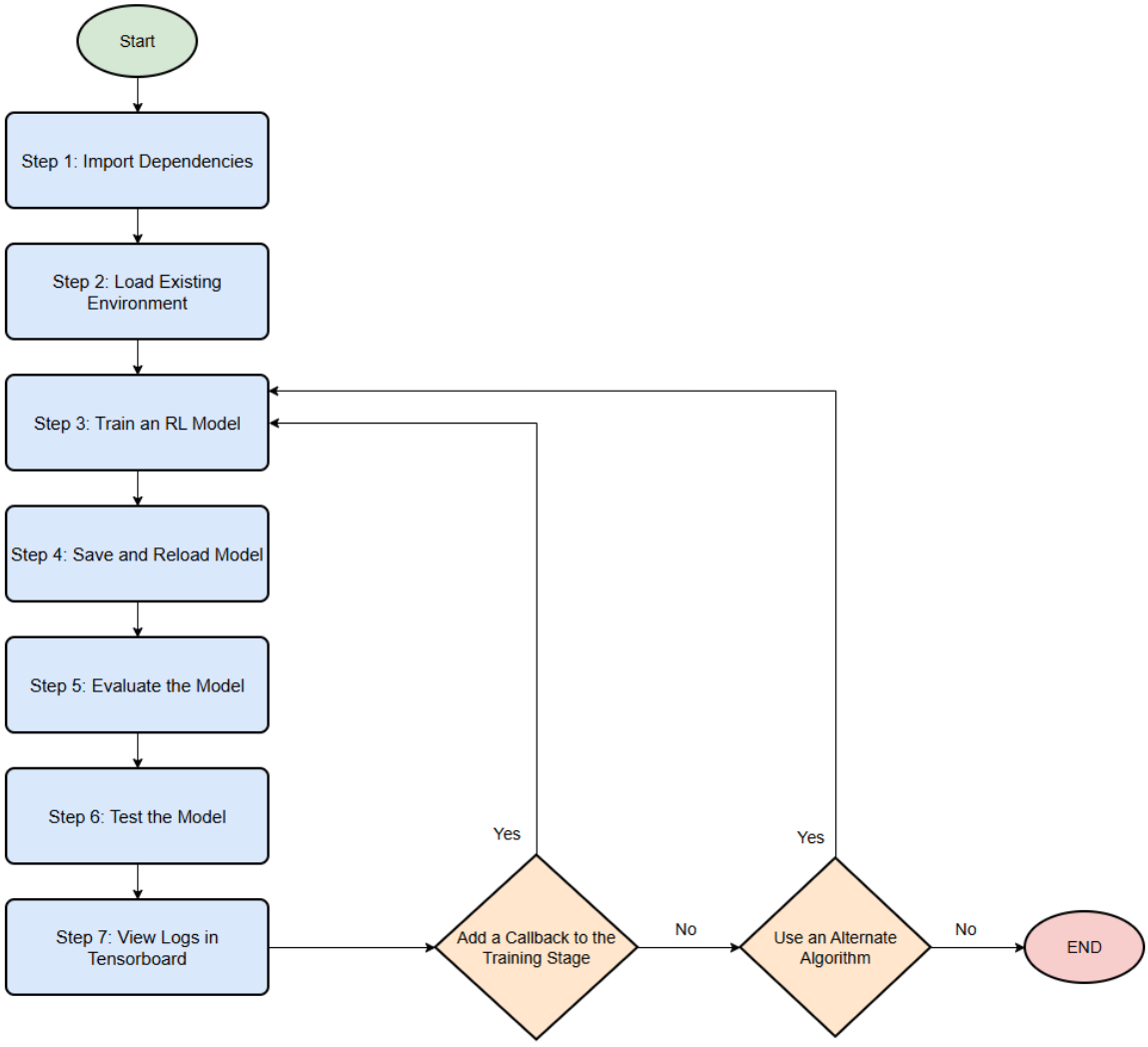


Figure 6.1: RL training process [40].

6.1.1 Getting Started

The operating system used to kick off this part of the project has the following device specifications:

Table 6.1: Hardware specifications.

Operating System	Microsoft Windows 11
Processor	11th Gen Intel(R) Core(TM) i5-11300H
CPU core	4
CPU frequency	3.10 GHz
Random Access Memory (RAM)	16.0 GB

For the code-related tasks, Microsoft Visual Studio (VS) code editor and Python were installed and used. The former is a code compiler that provides various coding features that can be easily leveraged to support code editing and completion. Python is used as the programming language because it is versatile and human-readable. It is also the preferred programming language for machine learning practices because it provides many reliable libraries and packages that can be easily leveraged through merely a few lines of code. Python can be installed and managed through Anaconda.

6.1.2 Import Libraries/Dependencies

After all the necessary software is obtained, the next step is to import the required RL dependencies. The first one to consider is Stable Baselines3. Stable Baselines is a RL library that enables users to interface with many different well-established model-free RL algorithms [40]. Stable Baselines has gone through multiple iterations, and the most up-to-date version as of this writing is Stable Baselines3. Also note that according to the documentation [41], Stable Baselines3 requires Python version 3.9+ and Pytorch version 2.3 or above. Stable Baselines3 was chosen for this project because it is an easy-to-use open-source library that provides many useful features and functions. In addition, it provides well-supported documentation, pretrained models, and high test coverage compared to the other RL libraries, as shown by figure 6.2 [42]:

	SB3	OAI Baselines	PFRL	RLlib	Tianshou	Acme	Tensorforce
Backend	PyTorch	TF	PyTorch	PyTorch/TF	PyTorch	Jax/TF	TF
User Guide / Tutorials	✓/ ✓	✗/ –	–/ ✓	✓/ ✓	–/ ✓	–/ ✓	✓/ –
API Documentation	✓	✗	✓	✓	✓	✗	✓
Benchmark	✓	✓	✓	✓	–	–	–
Pretrained models	✓	✗	✓	✗	✗	✗	✗
Test Coverage	95%	49%	?	?	94%	74%	81%
Type Checking	✓	✗	✗	✓	✓	✓	✗
Issue / PR Template	✓	✗	✗	✓	✓	✗	✗
Last Commit (age)	< 1 week	> 6 months	< 1 month	< 1 week	< 1 month	< 1 week	< 1 month
Approved PRs (6 mo.)	75	0	13	222	85	5	7

Figure 6.2: Comparison of Stable Baselines3 and other RL libraries. The blue bar means that the feature is only partially present [42].

Besides Stable Baselines3, another important dependency used is OpenAI gymnasium. OpenAI gymnasium provides various pre-built RL environments for researchers to use to quickly test different learning agents. Indeed, one of the commonly used environments for classic RL control practices provided by OpenAI gymnasium is called the cart pole, which was used to implement the RL portion of this project. Stable Baselines3 is designed to work with OpenAI gymnasium. The former provides different learning agents while the latter supplies the environments as testbeds. Together, they can be used to build a basic RL framework.

6.1.3 Load the Environment

The environment is the world in which the learning agent resides or operates. With the help of OpenAI gymnasium, the cart pole environment can be easily instantiated by using only one line of code, as shown by figure 6.3:

```
env = gym.make("CartPole-v1", render_mode='rgb_array')
```

Figure 6.3: Instantiate the cart pole environment from OpenAI gymnasium.

The cart pole environment was originally developed by Sutton, Barto, and Anderson from their studies in [30]. The physical system consists of a pole being attached to a cart through an unactuated joint without any friction, and starting out from an upright position, the goal is to apply forces to the cart in the horizontal directions to keep the pole balanced [43]. Typically, in a RL environment, there exists an action and observation space. The action space defines the set of possible actions the agent can take while the observation space contains the state variables used to describe the system. For the cart pole, the state variables used to represent the dynamics of the system are x, $\dot{x}$, θ, and $\dot{\theta}$. The default cart pole environment contains discrete action space and continuous observation space, with their possible range of values listed in tables 6.2 and 6.3 [43]:

Table 6.2: Possible action space [43].

Num	Action
0	Push cart to the left
1	Push cart to the right

Table 6.3: Possible and actual observation space [43].

Num	Observation	Possible Min & Max	Actual Min & Max
0	x	$\pm$ 4.8 m	$\pm$ 2.4 m
1	$\dot{x}$	$\pm \infty$	$\pm \infty$
2	θ	$\sim \pm$ 24° ($\pm$ 0.418 rad)	$\sim \pm$ 12° ($\pm$ 0.2095 rad)
3	$\dot{x}$	$\pm \infty$	$\pm \infty$

In the cart pole environment, it is clear to see that the agent can only take two discrete actions, which is either pushing the cart to the left (0) or to the right (1). The type of the action space in a RL environment is very important as it helps to determine the types of RL algorithms to use

to solve the problem. The detail is further discussed in subsection 6.1.4. On the other hand, the continuous observation space provides a wide range of possible values. These values include the conditions in which an episode ends. An episode in RL is defined as a story point or trajectory that captures the sequence of occurrences from start to finish [44]. For instance, in a video game, each episode is considered as a game play, and it ends when the player wins or loses. In the cart pole environment, although the available range of values to observe for the cart position is from $+4.8$ to -4.8 m, the episode actually terminates when it moves past $+2.4$ and -2.4 m in either direction [43]. Similarly, the angle of the pole can be observed in between $+24^\circ$ and -24°, but the episode terminates when it rotates beyond $+12^\circ$ and -12° [43]. Besides the observation thresholds, another way an episode in RL can end is when the maximum number of timesteps is reached. This is known as truncation. Timesteps in RL refer to "how many times the RL agent interacts with the environment and performs an action, receives a reward, and then changes the state" [21]. In the OpenAI cart pole environment, each episode is designed to truncate when 500 time steps are reached, provided that the observation thresholds have not yet been breached [43]. Furthermore, equations 3.7 and 3.8 were used as the governing equations of motion along with the data provided by table 4.1. To simulate the dynamics of the cart pole, OpenAI gymnasium adopted Euler's numerical integration method. This method is sufficient to approximate the responses of the nonlinear cart pole without demanding too much simulation run time.

Another key component that is part of the RL architecture and serves as an important evaluation metrics is the reward. If an episode has not been terminated or truncated in the cart pole environment, then the agent accumulates $+1$ point as reward; otherwise, it obtains no point [43]. Reward in RL is paramount because it is a form of feedback that connects the agent and the environment. It allows the environment to send a signal to tell the agent whether the previous action chosen was good or bad. As expected, an untrained agent sampling random actions for five episodes in the cart pole environment accumulates arbitrary rewards, as shown by figure 6.4:

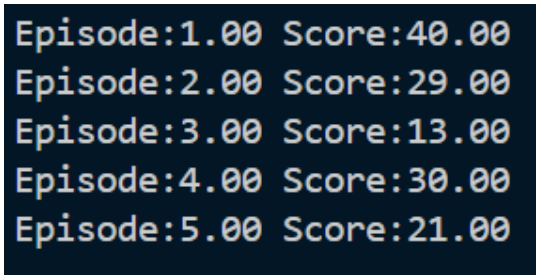


Figure 6.4: Rewards obtained by an untrained agent in the cart pole environment.

It is clear to see from figure 6.4 that episode 1 had the best run out of all the episodes. This means that the untrained agent, despite sampling actions randomly, had accumulated the most rewards in the first episode. However, as the environment gets equipped with a trained RL agent, readers can expect the reward values to increase drastically.

6.1.4 Train RL Agents

Prior to training, it is paramount to first decide on which RL algorithms to use. Although there are many options to choose, most of them tend to fall under two main categories - model-free and model-based RL - as illustrated by figure 6.5 [45]:

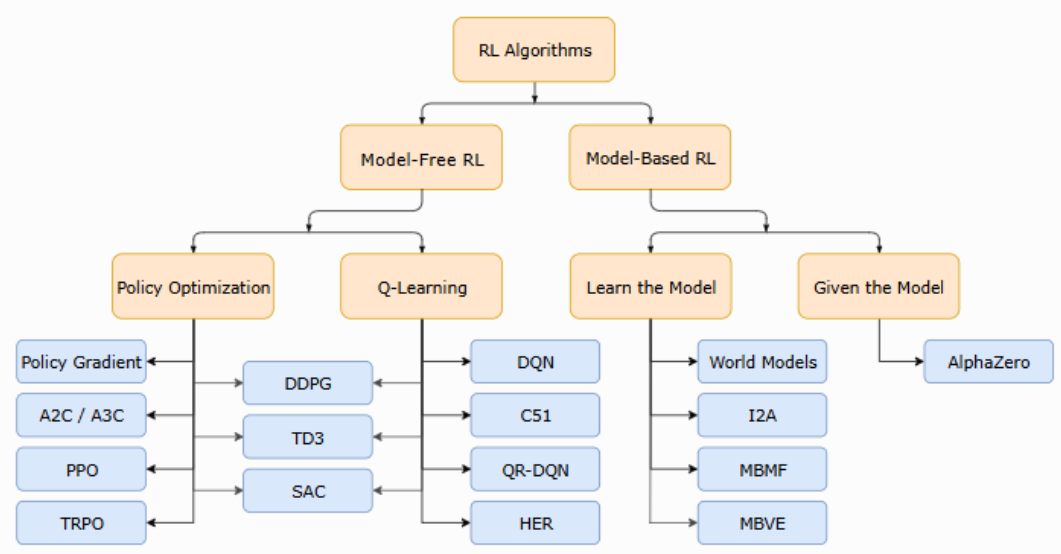


Figure 6.5: Types of RL algorithms [45].

Model-free RL agents learn to make decisions based on the current state only whereas model-based RL agents can learn from past experience. Since the exact mathematical models for most real-world systems are not always readily available, model-based RL as a result can pose challenges. In the recent years, model-free RL agents have become the focus for development in the industry because they exemplify the potential of data-driven models. Consequently, the focus for this part of the project is to investigate the model-free RL agents.

The key to further narrow it down and choose the proper model-free RL agents ultimately is dependent upon the goal of the project or mission. For the cart pole problem, the goal is to drive the cart, which is controlled by a trained RL agent, to balance the pole to stay upright for as long as possible. After the goal of the problem is decided, the observation and action spaces can be determined. Recall that the cart pole environment provided by the OpenAI gymnasium has a discrete action space. This type of action space can then be used to help determine what type of RL agents to choose. More specifically, Stable Baselines3 provides a look-up table that helps associate the type of action space with the RL agents, as shown by figure 6.6 [46]:

Name	Box	Discrete	MultiDiscrete	MultiBinary	Multi Processing
ARS [1]	✔	✔	✘	✘	✔
A2C	✔	✔	✔	✔	✔
CrossQ [1]	✔	✘	✘	✘	✔
DDPG	✔	✘	✘	✘	✔
DQN	✘	✔	✘	✘	✔
HER	✔	✔	✘	✘	✔
PPO	✔	✔	✔	✔	✔
QR-DQN [1]	✘	✔	✘	✘	✔
RecurrentPPO [1]	✔	✔	✔	✔	✔
SAC	✔	✘	✘	✘	✔
TD3	✔	✘	✘	✘	✔
TQC [1]	✔	✘	✘	✘	✔
TRPO [1]	✔	✔	✔	✔	✔
Maskable PPO [1]	✘	✔	✔	✔	✔

Figure 6.6: RL algorithms based on action space [46].

Since the cart pole system has discrete action space, it further narrows it down to the discrete column, and within it, it is clear to see that there are many applicable options. For this project, Advantage Actor-critic (A2C) and Proximal Policy Optimization (PPO) are chosen to control the cart pole.

As the name itself implies, A2C employs an actor and a critic, which are represented by two neural networks (NNs) [47, 48]. The actor is designed to choose an action given a set of observations while the critic's job is to evaluate the value of that action [47, 48]. Subsequently, the actor updates the policy based on the critic's evaluation [47, 48].

Another actor-critic-based algorithm used for this project is PPO. Similar to the cost function of the LQR, PPO relies on a gradient objective function to update the policy [47, 49]. However, different from LQR, PPO is known to implement a clipping parameter to prevent the policy from being updated too drastically, causing inaccurate solutions [47, 49].

Both A2C and PPO can be imported from the Stable Baselines3 library. A snippet of code is provided in figure 6.7 to show how easy it is to import and train an agent in the cart pole environment [50].

```python
import os
import gymnasium as gym
import numpy as np

from stable_baselines3 import A2C
from stable_baselines3.common.evaluation import evaluate_policy

# Instantiate the Environment
env = gym.make("CartPole-v1", render_mode="rgb_array")

# Train the RL agent
model = A2C("MlpPolicy", env, verbose=1)
model.learn(total_timesteps=10_000)
```

Figure 6.7: Training A2C with Stable Baselines3 [50].

The results for training the A2C and PPO agents for 100,000 timesteps are further discussed in subsection 6.1.6. Note from figure 6.7 that besides the environment, another important input argument that is required to pass to the model is the policy. The policy of a RL agent is essential as it governs the action of the agent. According to [21], the most commonly-used policy is the "MlpPolicy", which stands for Multi-Layer Perceptron (MLP). MLP uses neural networks (NNs) to learn patterns in data and leverage actions based on the policies to maximize rewards. Another well-known policy design is called "CnnPolicy", where CNN indicates Convolutional NNs [21]. CNN is not explored in this project since it is mostly used for the development of image-related type problems [21].

After the policy is specified and the environment is passed to the model, the model can be trained by the function model.learn(). Here, the total number of time steps, which is customizable by the users, must be passed to the function to specify the required training time. Verbose specifies the type of outputs that are being generated. A verbose of 0 indicates that no outputs will be printed to view while a verbose of 1 displays basic outputs such as progress updates, metrics, and additional diagnostic information.

6.1.5 Save and Reload the Model

After a model is trained, it can be saved and reloaded through Stable Baselines3. Saving the model enables users to save the progress made and if necessary continue training from the latest checkpoint rather than starting over from scratch. Saving is also necessary for any future modification and deployment. To save and reload a trained agent, a path to the folder/log directory is required. The OS dependency allows users to create or access existing directories on their operating systems. After the folder directory is created or located, the trained model can then be saved and reloaded as necessary through Stable Baselines3. For details of operation, refer to the code in Appendix F.

6.1.6 Evaluate the Model

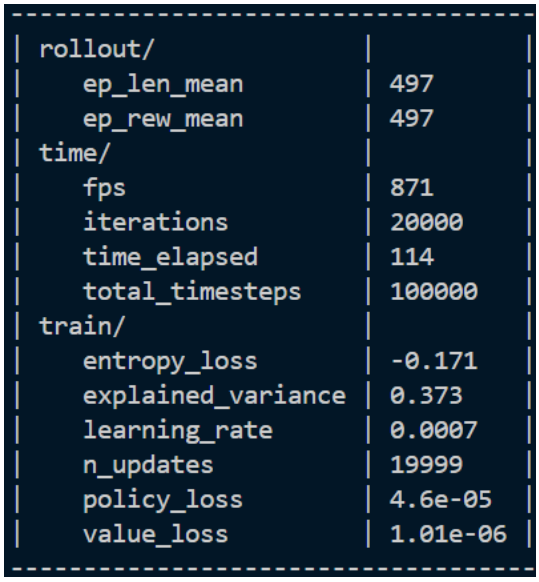
There are a couple of ways to evaluate a model. The simplest way is to use an evaluation policy function developed by Stable Baselines3. It is designed to evaluate the policy used to govern the actions of the selected RL agents. Details about the function can be found in the Stable Baselines3 documentation [51]. This function, expressible by one-line of code, as shown in figure 6.8, takes in various input arguments such as the model, the environment, and the number of episodes to evaluate and returns the mean reward and standard deviation as outputs. Note that prior to calling the function, the evaluate policy dependency must be imported.

```
# Evaluate the trained RL agent
mean_reward, std_reward = evaluate_policy(model, env, n_eval_episodes=10, render=True)
print(f"mean_reward: {mean_reward:.2f} +/- {std_reward:.2f}")
```

Figure 6.8: Evaluation policy function by Stable Baselines3 [51].

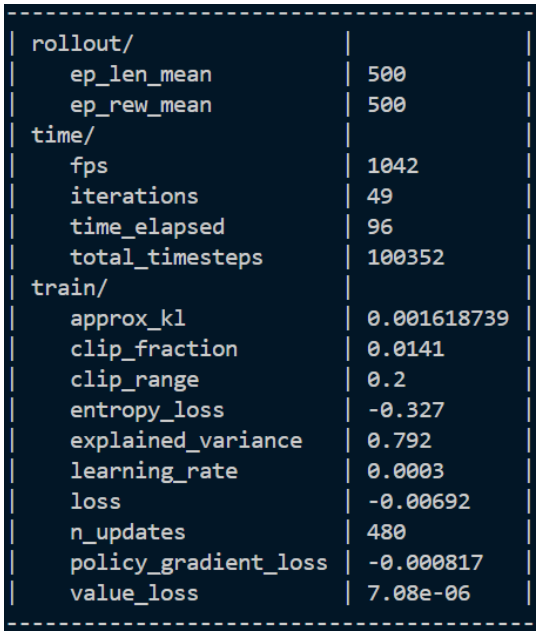
Passing the model and environment along with the number of episodes required for evaluation to the evaluate_policy function ultimately generates a mean reward followed by a standard deviation value. The mean reward is computed by summing up the total number of rewards accumulated over a period of time and then dividing it by the total number of time steps taken. According to the design of the OpenAI cart pole environment, when a maximum average reward of 475 is reached, the cart pole problem is considered solved [43]. Through the evaluate_policy function, both A2C and PPO were able to accumulate a mean reward of 500 with zero deviation.

Another way to evaluate the performance of an RL agent is to observe the evolution of the evaluation metrics from the rollout tables during training. An example of a rollout table for A2C and PPO are shown by figures 6.9 and 6.10. Under the rollout parameter, two variables - mean episode length (ep_len_mean) and mean episode reward (ep_rew_mean) - provide important insight for each training episode. The former denotes on average how long the agents survive in a particular episode while the latter shows the average rewards the agents accumulated in each episode [40]. Recall from the design of the cart pole environment that as long as the episode is not terminated or truncated, the agent accumulates a $+1$ reward; otherwise, no reward is given [43]. So the goal is to maximize the mean episode length and reward since the longer the agents survive, the more rewards they can accumulate. These rollout parameters for each training episode can be monitored and generated in real-time from the terminal window.



rollout/	
ep_len_mean	497
ep_rew_mean	497
time/	
fps	871
iterations	20000
time_elapsed	114
total_timesteps	100000
train/	
entropy_loss	-0.171
explained_variance	0.373
learning_rate	0.0007
n_updates	19999
policy_loss	4.6e-05
value_loss	1.01e-06

Figure 6.9: Evaluation metrics of the last episode for A2C.



rollout/	
ep_len_mean	500
ep_rew_mean	500
time/	
fps	1042
iterations	49
time_elapsed	96
total_timesteps	100352
train/	
approx_kl	0.001618739
clip_fraction	0.0141
clip_range	0.2
entropy_loss	-0.327
explained_variance	0.792
learning_rate	0.0003
loss	-0.00692
n_updates	480
policy_gradient_loss	-0.000817
value_loss	7.08e-06

Figure 6.10: Evaluation metrics of the last episode for PPO.

Notice how the two rollout parameters have the same values and seem to be directly proportional to each other. This is due to how the reward function was set up in the environment. Both the mean episode length and mean episode reward get incremented by 1 while the episode is not finished until they reach a mean reward of 500. It is also clear to see that A2C and PPO were able to accumulate a ep_len_mean and ep_rew_mean of 497 and 500, respectively, in their last episodes. This indicates good performance as it shows that the agents were learning to survive and maximize rewards. By observing the evolution of the evaluation metrics, researchers may also be able to identify

interesting results from the training data and get a rough sense of whether or not the agents are being trained properly. More importantly, the evaluation metrics log the hyperparameters that are used to train the agents. These hyperparameters can be used to customize training and learning. Different model-free RL agents can have different sets of hyperparameters, as demonstrated by figure 6.9 and 6.10. This project used the default Stable Baselines3 hyperparameters to facilitate learning.

6.1.7 Test the Model

After training of the A2C and PPO agents is completed, testing can be conducted through simulation to see how well they perform. To do so, use the model.predict() function. Detailed code provided by Stable Baselines3 are shown in figure 6.11 [50].

```python
import gymnasium as gym

from stable_baselines3 import A2C

env = gym.make("CartPole-v1", render_mode="rgb_array")

model = A2C("MlpPolicy", env, verbose=1)
model.learn(total_timesteps=10_000)

vec_env = model.get_env()
obs = vec_env.reset()
for i in range(1000):
    action, _state = model.predict(obs, deterministic=True)
    obs, reward, done, info = vec_env.step(action)
    vec_env.render("human")
    # VecEnv resets automatically
    # if done:
    #   obs = vec_env.reset()
```

Figure 6.11: Sample code to test trained RL agents by Stable Baselines3 [50].

One way to see the cart pole in action is to render an animation. To do so, follow figure 6.11, change the render mode from “rgb_array” to “human”, and call the .render() function. When the animation is enabled, sample images such as one shown by figure 6.12 are rendered.

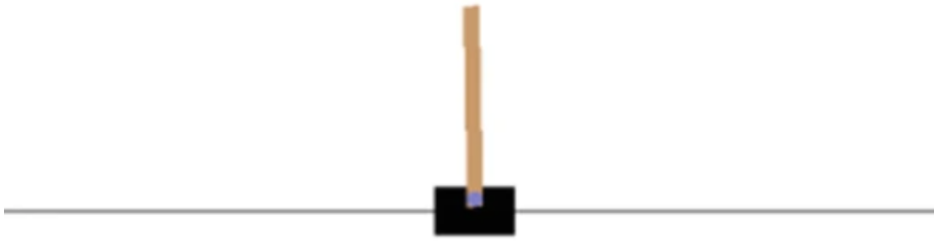


Figure 6.12: A screenshot captured from the cart pole animation.

Besides animation, the performance of the trained RL agents can be verified by the rewards accumulated in each episode. For instance, figure 6.13 shows that rather than sampling random actions, the trained agents were able to take correct course of actions to keep the cart pole system balanced for as long as possible and maximize a mean reward of 500 for each episode. The Python script used to simulate and test the model is attached at the end in Appendix H. It is clear to see from figure 6.13 that the mean rewards accumulated by the trained agents increased drastically and were more consistent than those shown by the untrained agent in figure 6.4.

```
Episode:1 Score:[500.]
Episode:2 Score:[500.]
Episode:3 Score:[500.]
Episode:4 Score:[500.]
Episode:5 Score:[500.]
Episode:4 Score:[500.]
Episode:5 Score:[500.]
Episode:6 Score:[500.]
Episode:7 Score:[500.]
Episode:8 Score:[500.]
Episode:9 Score:[500.]
Episode:10 Score:[500.]
```

Figure 6.13: Rewards obtained by trained A2C and PPO after testing.

6.1.8 Tensorboard Logging

The integration of Tensorboard with Stable Baselines3 enables real-time monitoring and logging of the training evaluation metrics. This is especially important for sophisticated RL algorithms that require a long time to train and develop. Identifying and correcting problems early on rather than waiting until the end of training can save time and resources. Often used for benchmarking different types of RL algorithms, Tensorboard can be set up by passing the argument tensorboard_log = "a log path" to the model when initializing the RL algorithm. The log path is different from the model directory used to save and reload the model. Nevertheless, it can be created the same way. In the meantime, tb_log_name = "a log name" and reset_num_timesteps = False need to be specified in the model.learn() function. The process is documented in detail in the Stable Baselines3 library [52]. Shown in figure 6.14 is an outlook of Tensorboard logging of the training evaluation metrics.

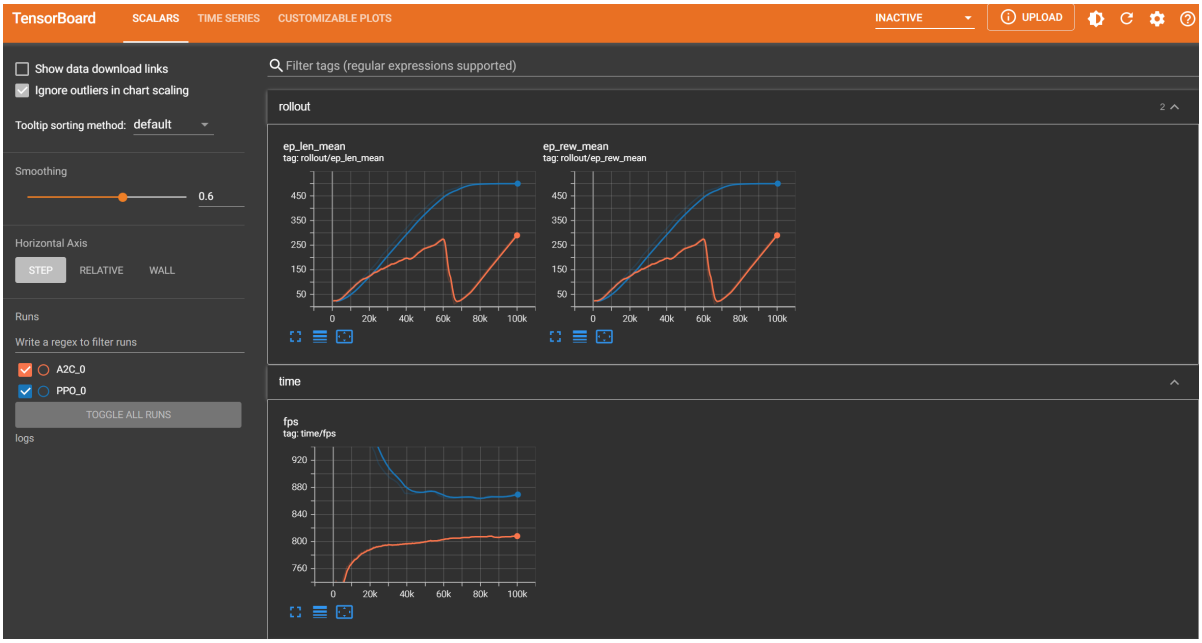


Figure 6.14: Tensorboard view.

6.1.9 Callbacks and Alternative Algorithms

Callbacks are optional features offered by Stable Baselines3 and can be used to customize logging. For example, while in training, a callback can be applied to deliberately monitor and display certain evaluation metrics in Tensorboard. Other model-free RL algorithms were not tested in this project, but it is certainly possible to experiment with the other applicable ones provided by Stable Baselines3.

7. Results and Discussion

This section showcases the simulation results developed by Stable Baselines3's A2C and PPO agents. OpenAI gymnasium's cart pole environment with slight changes made was used to train these agents. For detail, see Appendix H. The training was designed by five cases of trial run, and each contained a different number of timesteps. The total number of timesteps used for training and the run time consumed for each case run are documented in table 7.1.

Table 7.1: Information for each trial run

Trial No.	Number of Timesteps Used	Run Time for A2C	Run Time for PPO
1	50,000	3 mins	2 mins
2	100,000	4 mins	4 mins
3	500,000	16 mins	18 mins
4	1 Million	30 mins	38 mins
5	5 Million	180 mins	171 mins

The results in terms of the mean episode length, mean episode reward, and value loss for each trial run were captured in Tensorboard and displayed as follows.

7.1 Case I for 50,000 Timesteps

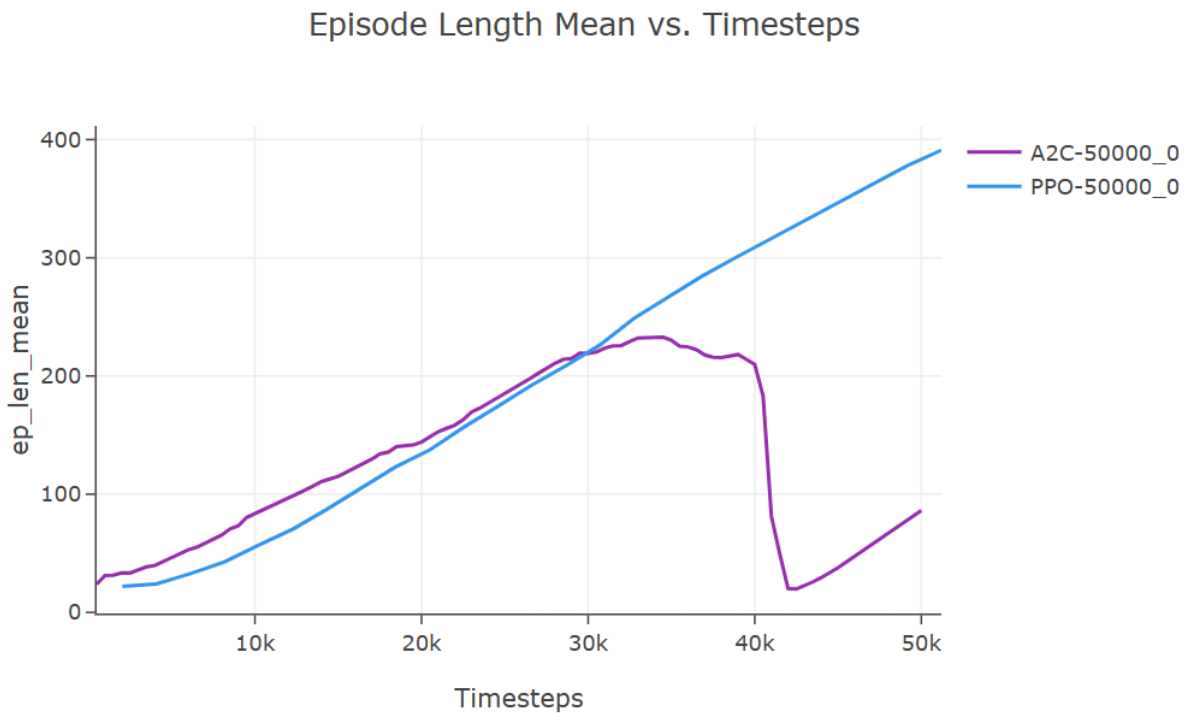


Figure 7.1: History of mean episode length for 50,000 timesteps

Figure 7.1 shows the survival time for each agent on average over a specific number of timesteps. It is clear to see that PPO's mean episode length increased linearly as the number of timesteps increased while A2C's mean episode length increased up to a point and then took a deep dive at around 40,000 timesteps before trying to recover. The longest A2C survived on average was around 232 timesteps during training over approximately 32,000 timesteps.

Another key rollout parameter to look at is the mean episode reward, as shown by figure 7.2:

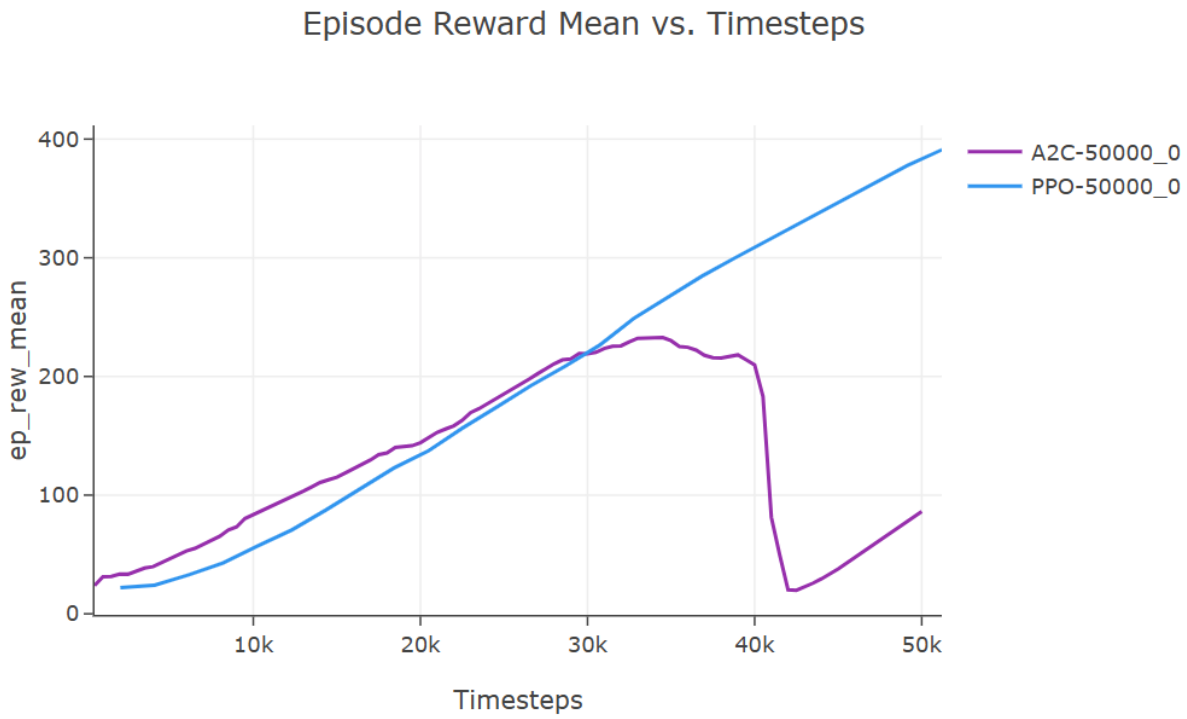


Figure 7.2: History of mean episode reward for 50,000 timesteps

Note that the mean episode length and mean episode reward have identical plots. This is mainly because of how the cart pole environment was designed to correlate rewards with the number of timesteps. As long as the episode is not finished, the agents accumulate $+1$ point for reward and the number of timesteps also gets incremented by 1. Similar to the mean episode length, as the number of training timesteps increased, PPO's mean episode reward also increased linearly. This means that during training, PPO was actively seeking to maximize rewards on average over the specific number of timesteps. The more timesteps PPO were trained, the better it performed. However, this is not necessarily true for all RL agents, as one shall see from the results followed. Since both mean episode length and mean episode reward have identical results, from this point on, only plots of the mean episode reward are presented for the other cases of run.

Besides mean episode length and mean episode reward, value loss is another key training metric that is often used to evaluate performance and compare different RL algorithms. In RL, value is defined as the expected long-term return accumulated by following a particular policy [53]. Value is related to not just a single reward of one episode but the overall rewards throughout all the episodes, and as a result, value is often more desired than reward. Since value is good, value loss is bad and should be avoided or minimized if possible. The result of the value loss for both A2C

and PPO is shown by figure 7.3:

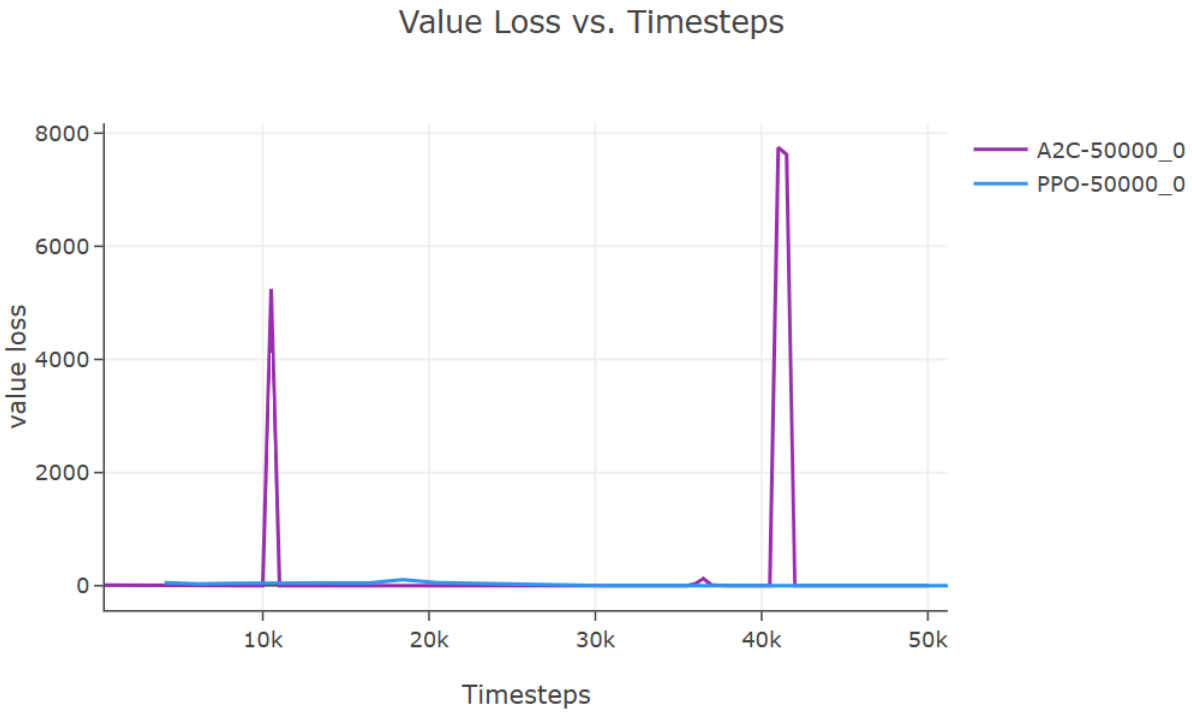


Figure 7.3: History of value loss for 50,000 timesteps

It can be observed that the value loss of PPO had very few fluctuations and was quickly minimized. On the other hand, even though the value loss of A2C was eventually minimized just like PPO, it experienced fluctuations from time to time. This shows that A2C is not as consistent and stable as PPO.

7.2 Case II for 100,000 Timesteps

Results of the mean episode reward are shown by figure 7.4:

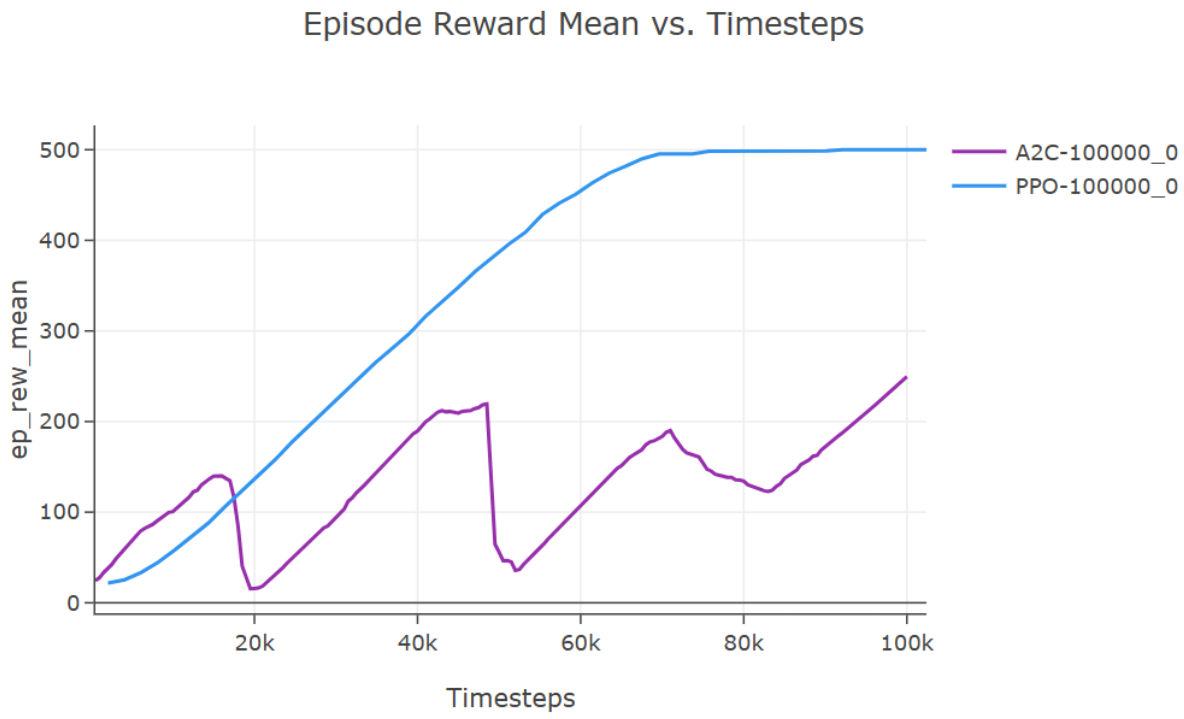


Figure 7.4: History of mean episode reward for 100,000 timesteps

Figure 7.4 shows that training both agents for more timesteps contributed to better results. This is especially true for PPO as it was able to reach and exceed the reward threshold of 475 and converged to a mean episode reward of 500 at around 80,000 timesteps. A2C also performed better as it was able to accumulate more rewards on average than before as training increased. However, it is clear that A2C was still not as stable as PPO.

As far as the value loss goes, figure 7.5 once again shows PPO outperformed A2C as there were barely any value losses in PPO compared to A2C. On the other hand, although A2C's value loss was periodically minimized, it showed more volatility than before.

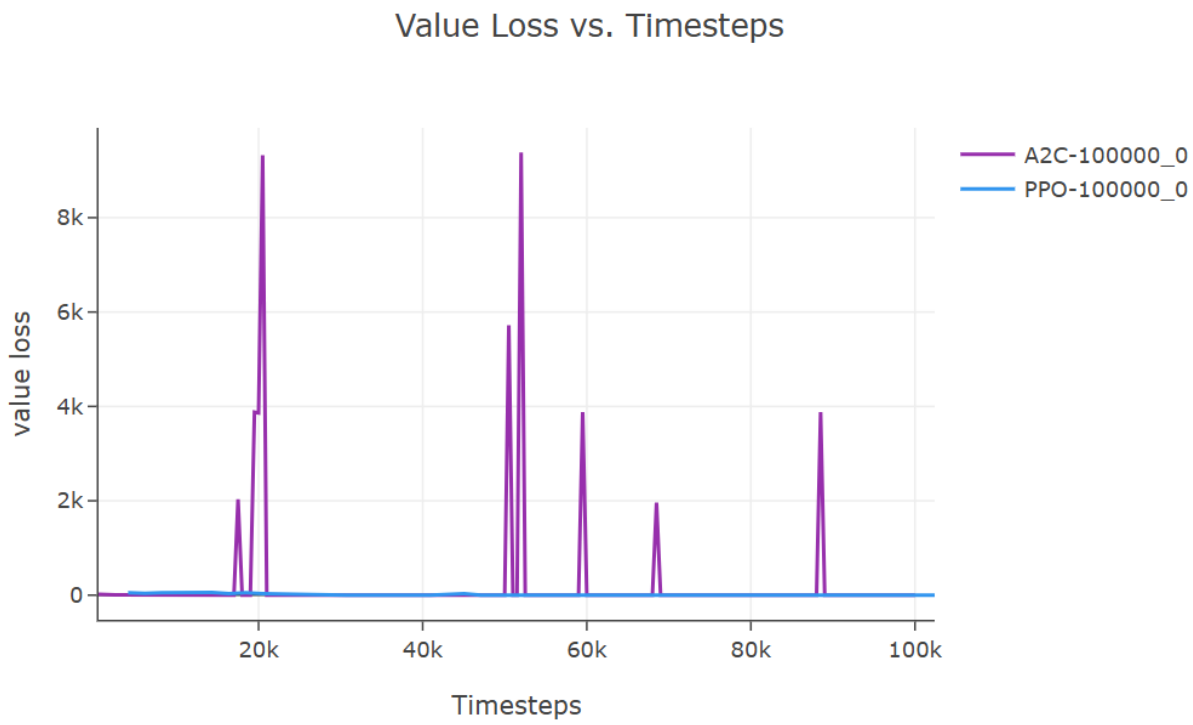


Figure 7.5: History of value loss for 100,000 timesteps

7.3 Case III for 500,000 Timesteps

In the third case run, it can be observed that the mean episode rewards of A2C and PPO both reached and exceeded the reward threshold of 475. Ultimately, the results converged to a mean episode reward of 500, as shown by figure 7.6. Although both agents were capable of reaching a mean episode reward of 500, it clearly took A2C 20,000 more training timesteps to get there compared to PPO. In addition, even though A2C reached a mean episode reward of 500 after 100,000 timesteps, unlike PPO, A2C did not maintain this mean episode reward throughout. This further shows that A2C was not as stable and reliable as PPO as training continued.

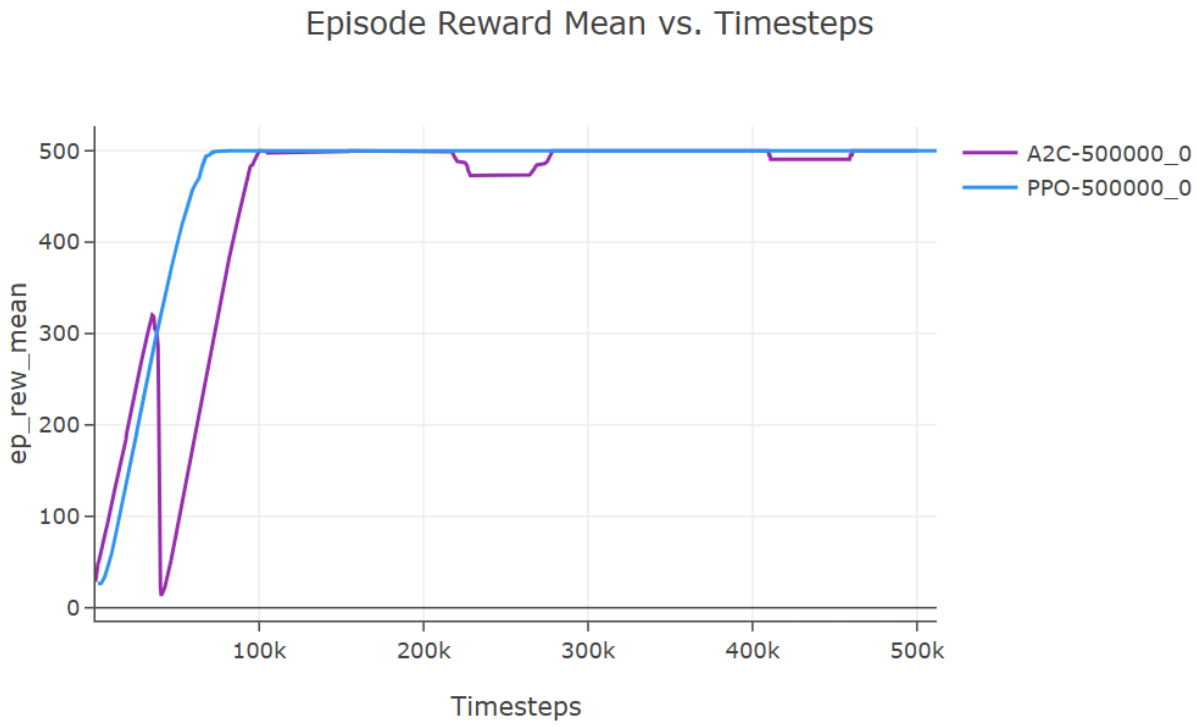


Figure 7.6: History of mean episode reward for 500,000 timesteps

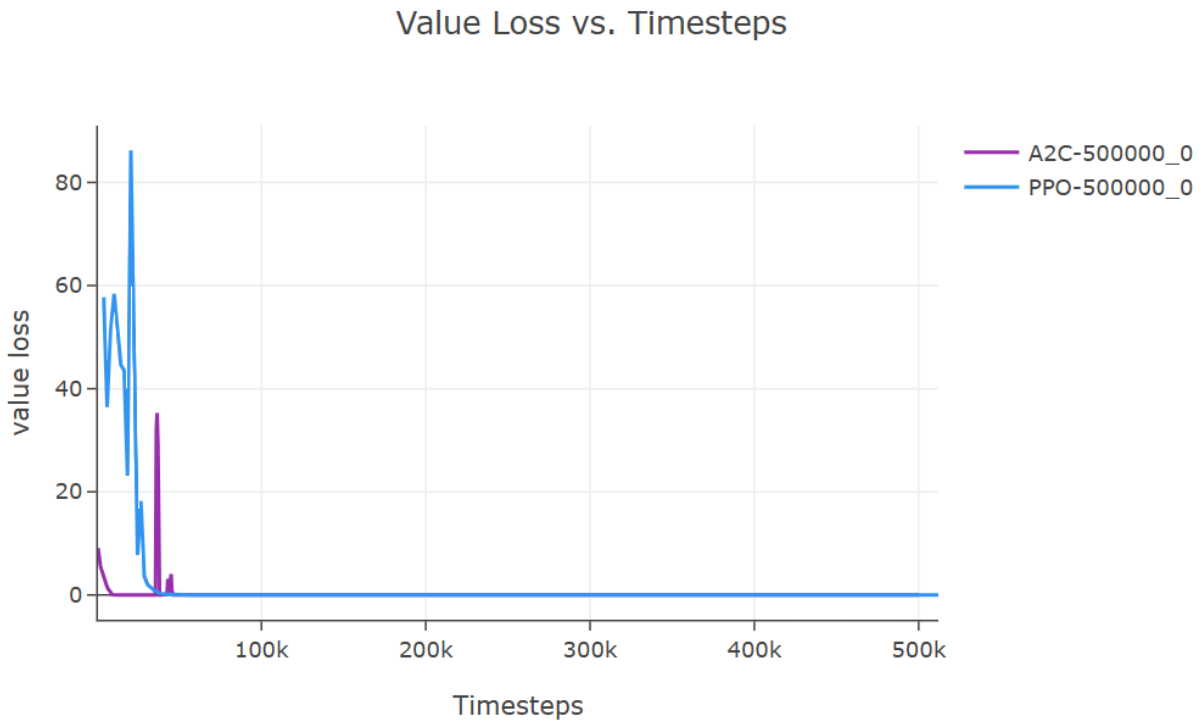


Figure 7.7: History of value loss for 500,000 timesteps

Despite initially experiencing some volatility, the value loss for both A2C and PPO were minimized and stabilized at zero as the number of training timesteps increased.

7.4 Case IV for One Million Timesteps

A case run with one million timesteps was conducted for both A2C and PPO, and the results are shown by figures 7.8 and 7.9. As both agents were trained for one million timesteps, A2C did worse than before as it took almost one-fourth of the timesteps to reach a mean episode reward of 500, and even then the mean episode rewards that followed were not as stable compared to PPO. This further illustrates that the training performance of a RL agent is not necessarily guaranteed by the number of timesteps used for training.

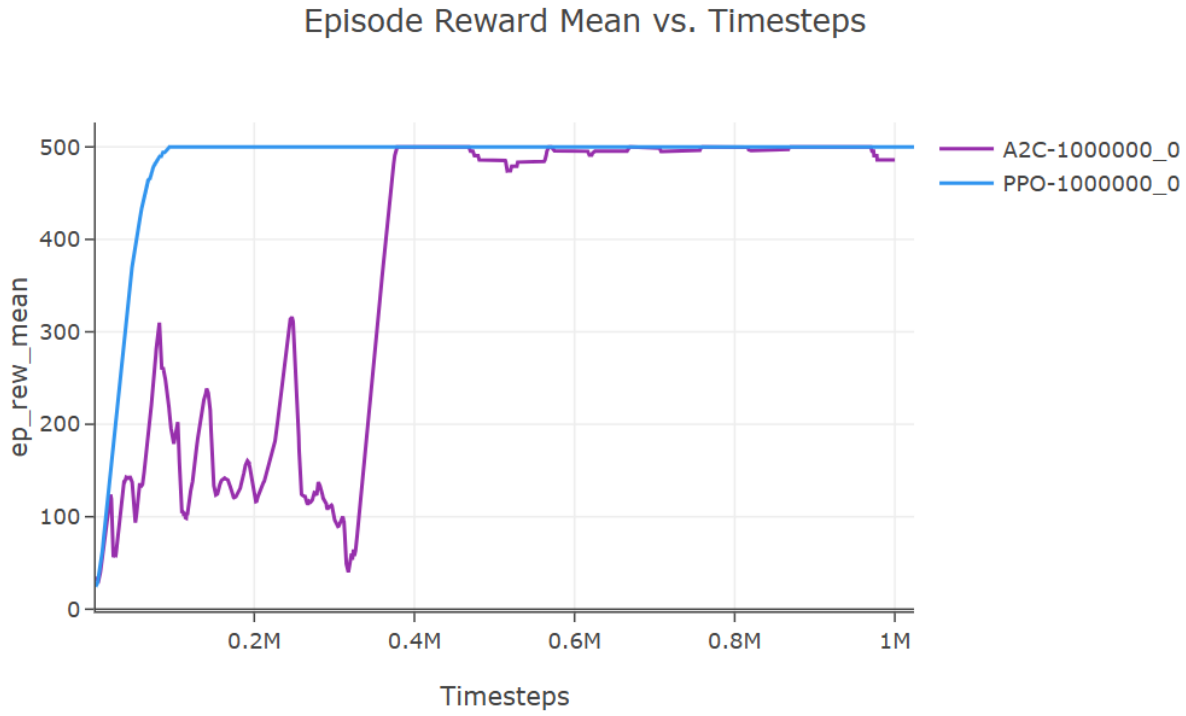


Figure 7.8: History of mean episode reward for one million timesteps

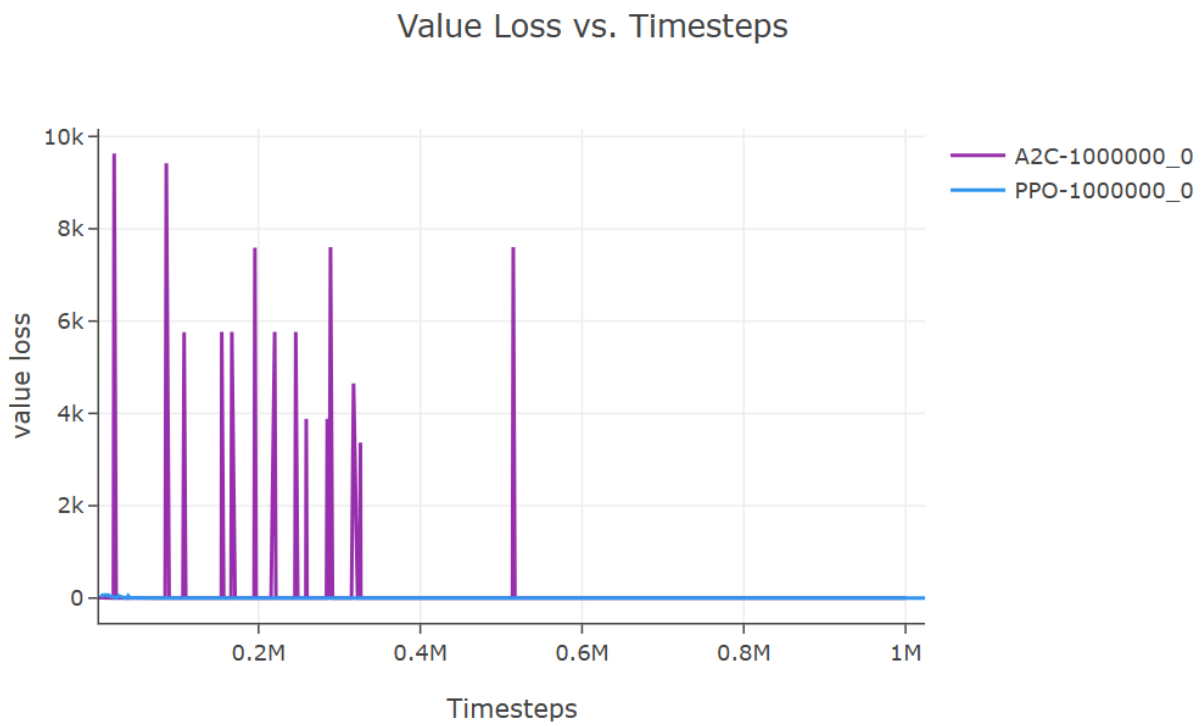


Figure 7.9: History of value loss for one million timesteps

Figure 7.9 shows that the A2C's value loss once again experienced a lot of volatility initially and all the way up to about half way into training. In addition, it can be observed that A2C at one point had reached a peak value loss that was as high as around 9,000. On the other hand, although PPO also experienced some value losses at the beginning, they were insignificant compared to the A2C's value losses.

7.5 Case V for Five Million Timesteps

The last case was designed to train the A2C and PPO agents for five million timesteps. This case required the greatest computational resource to support, as shown in table 7.1. Overall, A2C exhibited the most volatile mean episode rewards throughout. This volatility lasted all the way up to around 3.8 million timesteps. This once again shows that training a RL agent for longer does not necessarily guarantee better training performance. However, the mean episode rewards for PPO, after they reached the reward threshold, stayed smooth and stable as always.

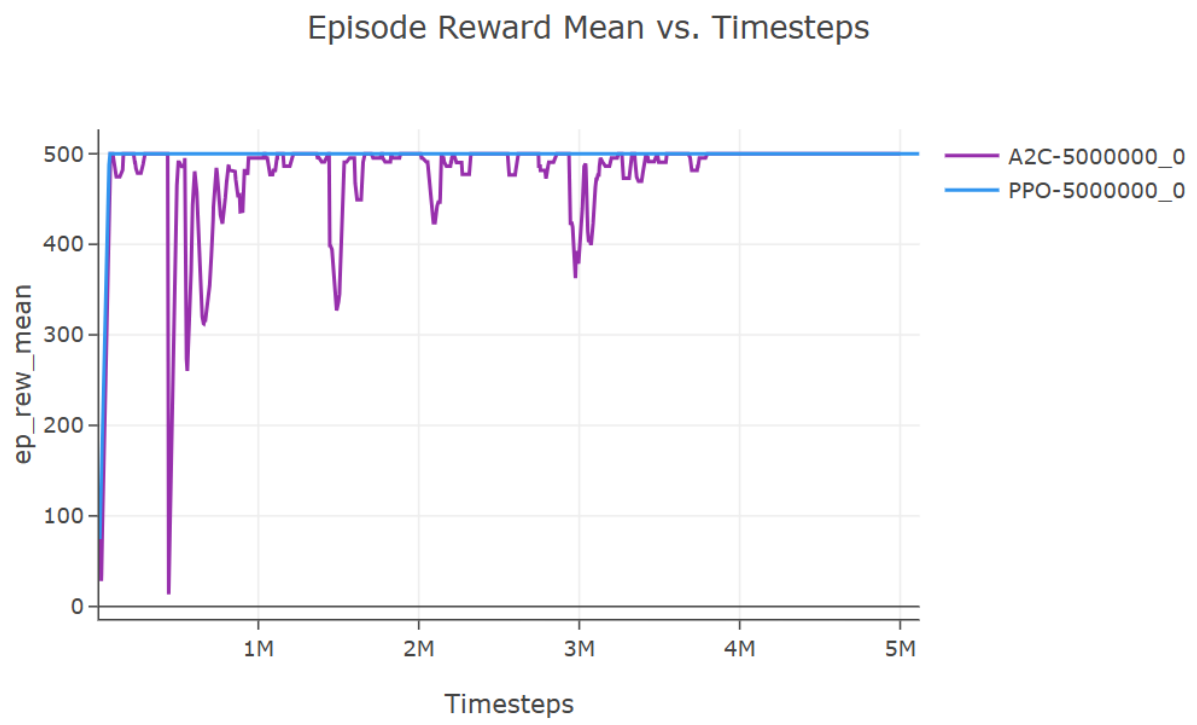


Figure 7.10: History of mean episode reward for five million timesteps

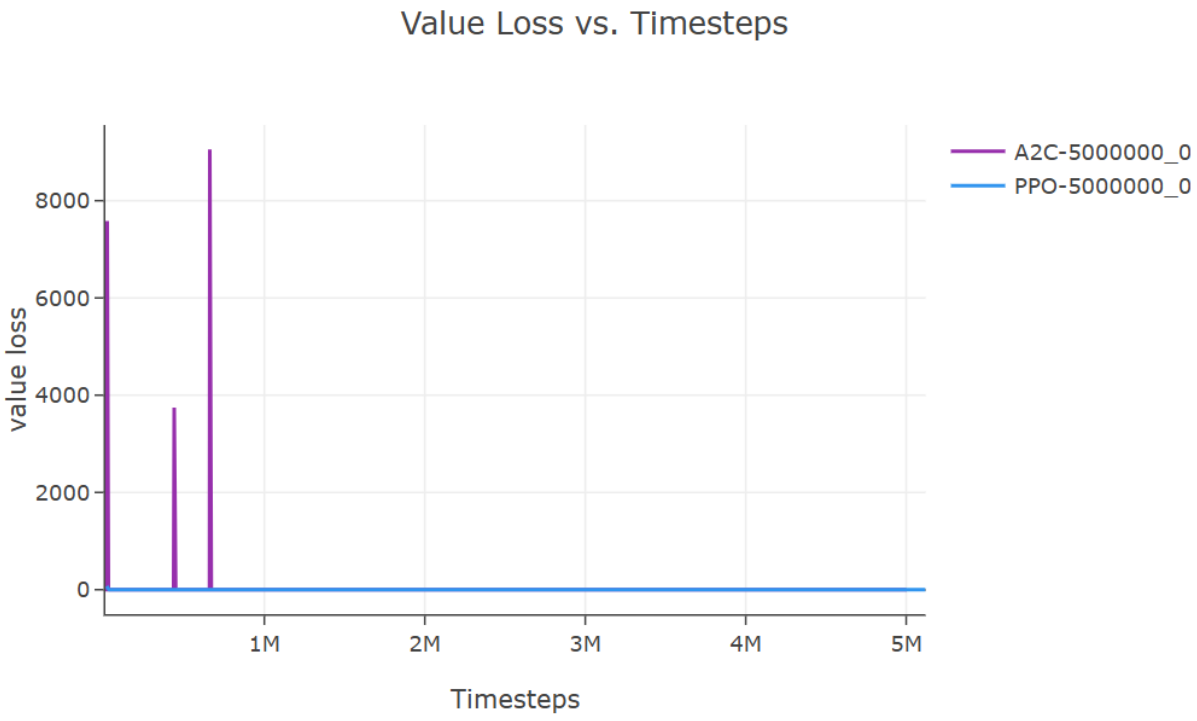


Figure 7.11: History of value loss for five million timesteps

Even though the value loss of A2C was minimized after approximately 800,000 timesteps of train-

ing, it still exhibited initial volatility. However, compared to the value losses observed in some of the other A2C cases, such as case II and case IV, the amount of volatility was noticeably reduced. This reduction might be attributed to the use of a larger timesteps scale in the value loss graph, which caused fluctuations that were relatively close to one another to appear as a single impulse signal. Nevertheless, A2C's value loss reached a peak of over 8,000 at one point. In contrast, similar to PPO's mean episode reward curve, its value loss was just as smooth and stable as always.

Based on the results exhibited by the mean episode length mean, episode reward mean, and value loss, it can be concluded that PPO is better than A2C because PPO can provide more stable and reliable results during training.

7.6 Responses of the Cart Pole by A2C and PPO

After training the agents for five case runs, the results of the best case can be analyzed using Tensorboard. The best case is identified based on the maximum mean episode reward and the minimum value loss. Once the best case is selected, testing can proceed. For example, the results at one million timesteps, out of the five-million-timesteps case run, met the requirements and were thus chosen for simulation. The data obtained from this process was then used to generate the closed-loop responses of the cart-pole system and the results were shown as follows. The script used for post-processing the data is provided in Appendix I.

7.6.1 A2C Results

The results from the A2C controller are shown by figures 7.12 through 7.15:

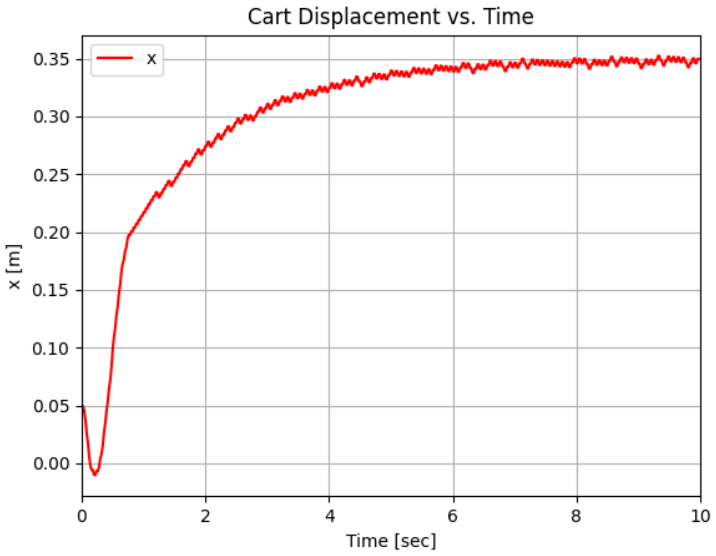


Figure 7.12: Closed-loop response of cart position over time by A2C

Figure 7.12 demonstrates that the A2C controller successfully balanced the cart's motion and the response of x asymptotically approaching 0.35 m. In other words, instead of converging to zero, the response of x governed by the A2C controller stabilized at a different equilibrium point of 0.35 m. It can also be observed that the A2C response of x exhibits small, sustained oscillations, indicating that the system is neutrally stable with respect to x. Furthermore, the closed-loop responses

of x by A2C and LQR, illustrated in figure 5.4, show that they both achieved a settling time of approximately 7 seconds.

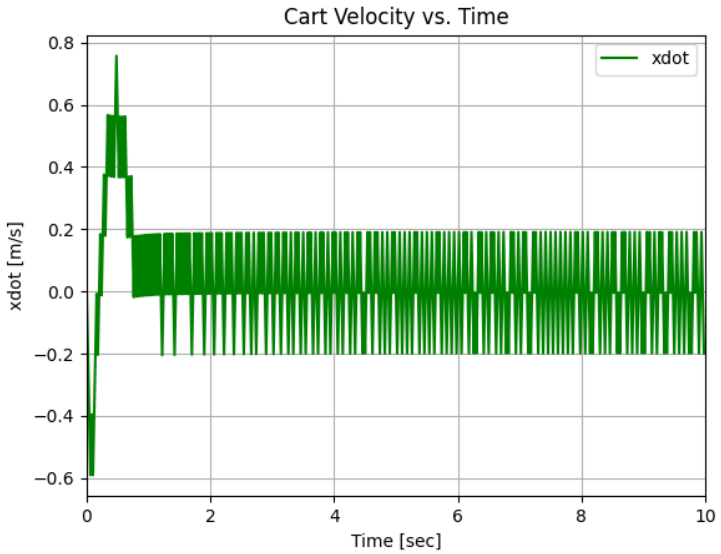


Figure 7.13: Closed-loop response of cart velocity over time by A2C

As shown in figure 7.13, the response of the cart velocity ($\dot{x}$), controlled by the A2C algorithm, exhibits a significantly smaller overshoot displacement of approximately 0.78 $\frac{m}{s}$ and a faster settling time of approximately 1 second compared to the LQR controller, which demonstrates an overshoot of around 3.5 $\frac{m}{s}$ and a settling time of about 5 seconds (illustrated in figure 5.4). While figure 7.13 clearly shows that the response of $\dot{x}$ stabilized around zero, it also reveals sustained oscillations ranging from -0.2 $\frac{m}{s}$ to 0.2 $\frac{m}{s}$. Similar to x, this response indicates neutral stability for $\dot{x}$.

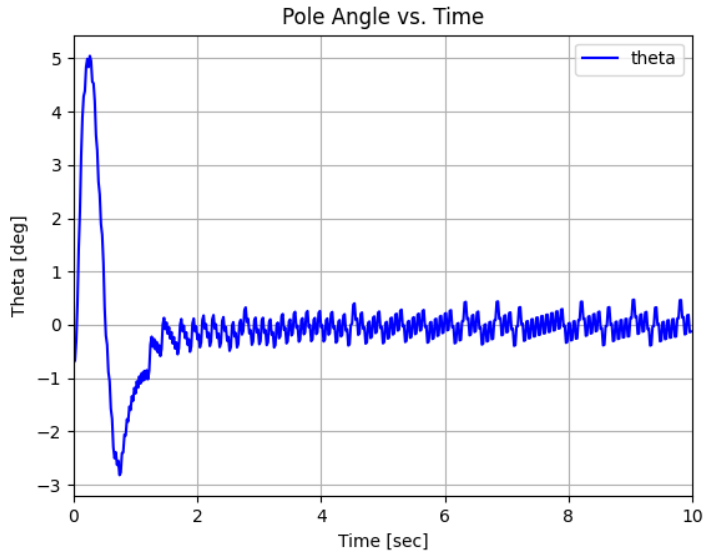


Figure 7.14: Closed-loop response of pole angle over time by A2C

Figure 7.14 further highlights the effectiveness of the A2C algorithm compared to the LQR controller in terms of overshoot control and settling time. Specifically, the pole angle response (θ) under A2C control exhibited a 5° overshoot, meaning that the angle deviated by at most 5° from the perfectly upright position of 0°. In contrast, the LQR response of θ showed an undershoot of approximately -11.4°. Furthermore, the A2C-controlled response reached steady-state about zero in approximately 2 seconds, whereas the LQR controller required over 3 seconds to stabilize the response. However, the A2C response for θ still demonstrates minor sustained oscillations.

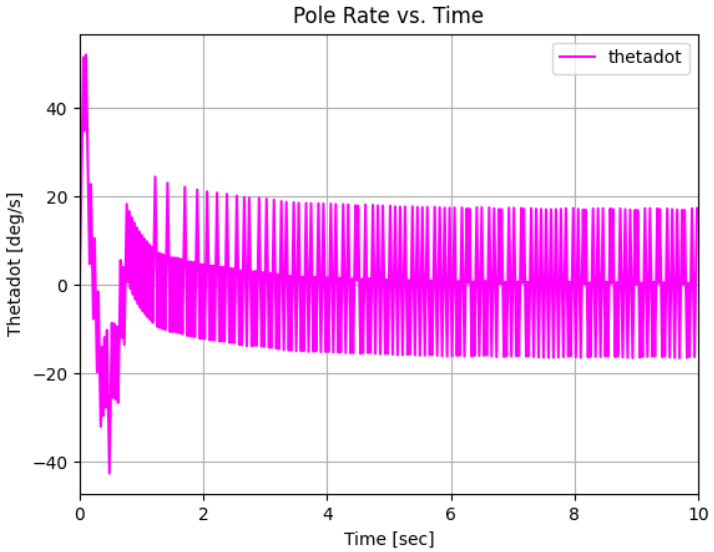


Figure 7.15: Closed-loop response of pole rate over time by A2C

It can be observed by figure 7.15 that the response of $\dot{\theta}$ exhibited the greatest fluctuations compared to the other A2C responses. While the A2C response experienced an undershoot of approximately $-40°$/sec, this displacement was nearly 85% smaller than the undershoot observed in the LQR-controlled response, which reached about $-260°$/sec. Despite this, the A2C-controlled response of $\dot{\theta}$ achieved stability with a settling time of approximately 4 seconds, whereas the LQR-controlled response stabilized more quickly, within 2-3 seconds. Similar to the responses of all other system states, the response of $\dot{\theta}$ also exhibited sustained oscillations, ranging between -20 to $20°$/sec.

A possible explanation for the sustained oscillations could be the absence of saturation limits. Specifically, $\dot{x}$ and $\dot{\theta}$ were not restricted in the same manner as x and θ, as dictated by the conditions specified in table 6.3. Without these limits, they are free to vary from $-\infty$ to $+\infty$, effectively prioritizing the balancing of the x and θ.

7.6.2 PPO Results

The results simulated by the PPO controller are shown by figure 7.16 through 7.19:

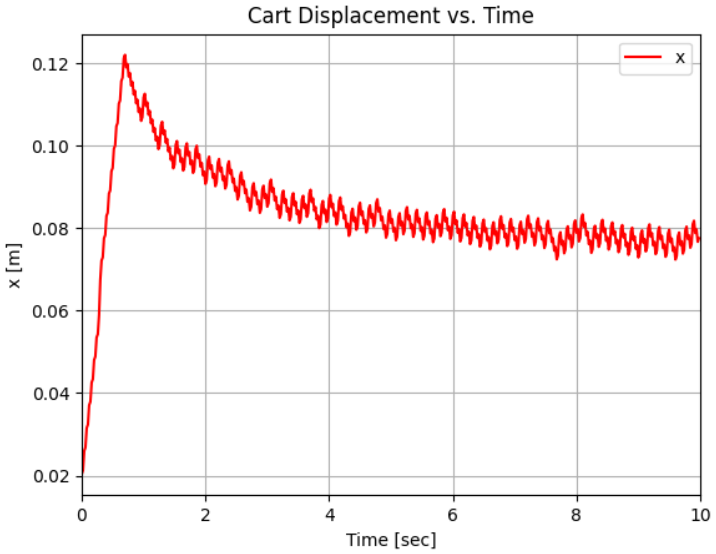


Figure 7.16: Closed-loop response of cart position over time by PPO

Figure 7.16 shows that the PPO response of x achieved a small overshoot of approximately 0.12 m, with a settling time of around 5-6 seconds. Compared to the A2C result, however, the PPO response of x reached an equilibrium point closer to zero stability. However, similar to the A2C response, the overshoot displacement was significantly smaller than that of the LQR response. In addition, like the A2C result, the PPO response of x exhibits minor sustained oscillations.

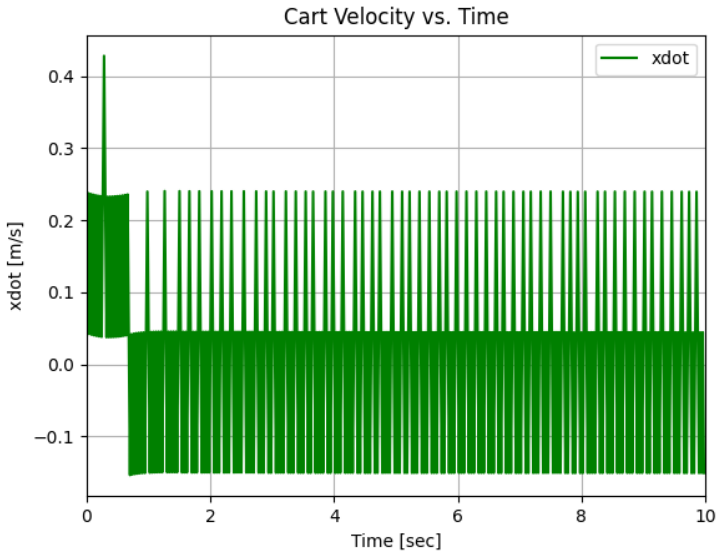


Figure 7.17: Closed-loop response of cart velocity over time by PPO

From figure 7.17, it is evident that the PPO response for $\dot{x}$ achieved a smaller overshoot displacement of approximately 0.45 $\frac{m}{s}$ and the fastest settling time of less than 1 seconds compared to A2C and LQR. However, sustained oscillations persisted, varying between approximately -0.18 $\frac{m}{s}$ and 0.24 $\frac{m}{s}$.

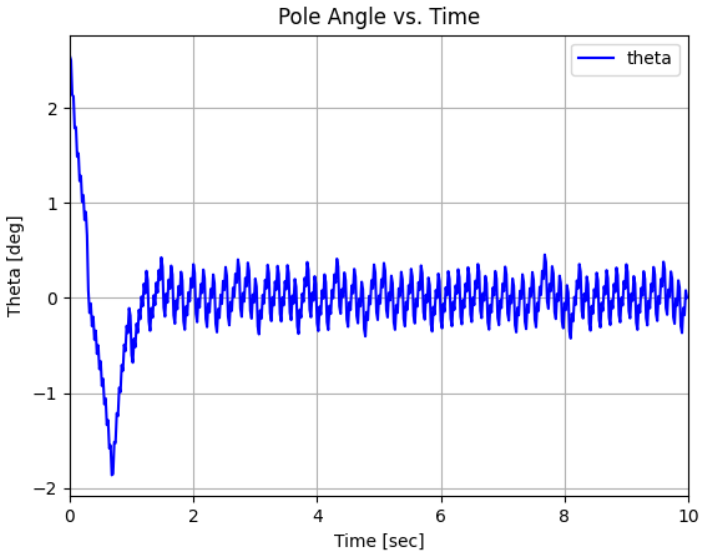


Figure 7.18: Closed-loop response of pole angle over time by PPO

Figure 7.18 shows that the PPO response of θ outperforms those of A2C and LQR in terms of undershoot control and settling time. Specifically, the PPO response of θ exhibited a -1.9° undershoot and achieved a settling time of less than 2 seconds. However, similar to the response of

A2C, the PPO response demonstrated small sustained oscillations.

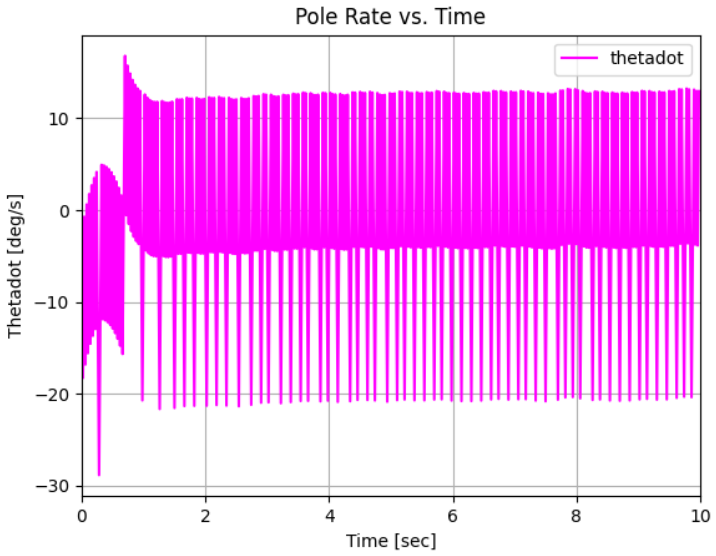


Figure 7.19: Closed-loop response of pole rate over time by PPO

Compared to the A2C and LQR responses of $\dot{\theta}$, the PPO response from figure 7.19 exhibited a smaller undershoot displacement of approximately $-29°$/sec and a faster settling time of less than 2 seconds. Additionally, similar to the A2C response, the PPO response displayed sustained oscillations, bounded between approximately $-20°$/sec to $12°$/sec. Since saturation limits were not imposed, the PPO responses of $\dot{x}$ and $\dot{\theta}$ were expected to behave similarly to the A2C responses, exhibiting more radical fluctuations compared to the responses of x and θ.

Given the definition of the cart-pole RL problem, sustained oscillation may not necessarily be problematic, as the cart-pole task has been successfully solved, with each RL agent achieving maximum rewards. However, further optimization is recommended to enhance the results and ensure full stability and convergence. Experiments were conducted to modify the default reward calculation, as shown in figure 7.20, in the cart pole environment developed by the OpenAI gymnasium [43]. This reward design clearly states that if the episode is neither terminated nor truncated, the agent will accumulate $+1$ point; otherwise, it will receive 0 points. Several different reward functions were designed and tested, but none of them succeeded in eliminating the sustained oscillation phenomenon. For details regarding the modified reward functions that were tried, refer to Appendix J.

```
if not terminated:
    reward = 1.0
elif self.steps_beyond_terminated is None:
    # Pole just fell!
    self.steps_beyond_terminated = 0
    reward = 1.0
else:
    if self.steps_beyond_terminated == 0:
        logger.warn(
            "You are calling 'step()' even though this "
            "environment has already returned terminated = True. You "
            "should always call 'reset()' once you receive 'terminated = "
            "True' -- any further steps are undefined behavior."
        )
    self.steps_beyond_terminated += 1
    reward = 0.0
```

Figure 7.20: Default reward calculation defined by OpenAI gymnasium.

Besides redesigning the reward calculation, the action space was also changed from discrete to continuous in the environment. In other words, instead of allowing the agent to move the cart to either left (0) or right (1), the action space changed to allow the agent to take a wider range of actions from $[-1, 1]$. After the change was made, simulations were performed by using PPO as PPO works for both discrete and continuous action spaces, but the sustained oscillation remained.

8. Conclusion and Future Work

This paper examined two types of control frameworks used to balance a 2-D, nonlinear second-order inverted pendulum system, also commonly known as the cart pole. The first framework was implemented by a classic optimal LQR control, and the simulation results showed that the responses of the full states of the inverted pendulum system achieved stability well within 10 seconds without exceeding the design criteria imposed. In addition, despite experiencing an overshoot and undershoot of 1.04 m and -11.38° in about 0.7 and 0.5 seconds, respectively, the responses of x and θ reached a zero steady-state in about 6 and 4 seconds, respectively. The closed-loop system stability was also demonstrated by the negative pole characteristics for all four states of the inverted pendulum system, further verifying that the system is indeed fully stable.

Another control framework analyzed and applied to control the inverted pendulum system was developed based on a subset of artificial intelligence machine learning (AI/ML) called reinforcement learning (RL). Stable Baselines3's A2C and PPO algorithms were chosen to perform this task. The results of A2C and PPO showed that the RL-based controllers can often outperform LQR in terms of overshoot/undershoot control and settling time, but not in achieving zero stability convergence. Although the inverted pendulum system was balanced, the A2C and PPO responses of x stabilized at non-zero equilibrium points - around 0.35 m and 0.08 m, respectively. Furthermore, the results from A2C and PPO showed sustained oscillations, indicating that the system is neutrally stable. Similarly, the responses of θ, governed by A2C and PPO, also displayed sustained oscillations. However, these oscillations were relatively small and centered about 0°. Besides benchmarking the RL-based controllers with LQR, the A2C and PPO were also evaluated against each other based on several valuable metrics, namely, the mean episode length, mean episode reward, and the value loss. It was found that although both RL algorithms were able to achieve a mean episode reward of 500 after approximately 500,000 timesteps, PPO not only reached the maximum reward faster but also provided reliable results consistently. Although the results generated by the RL-based control frameworks showed sustained oscillations, the rendered animation revealed that the cart pole was successfully balanced and both RL algorithms were able to maximize a mean reward of 500 as training timesteps increased.

Although the results of the two proposed RL-based control frameworks, A2C and PPO, require further improvement, they still demonstrated the potential of data-driven learning algorithms. These algorithms eliminate the need for complex mathematical formulations and system linearization, enabling them to learn and improve independently, even in the presence of uncertainties. In addition to optimizing results, future work involves analyzing and tuning the hyperparameters to understand their impact on the training of the RL agents, integrating RL-based and traditional control frameworks to enhance performance in controlling the inverted pendulum system or other applications, and designing custom environments from scratch with more efficient numerical integration methods to address more complex systems.

References

[1] Åström, K. J., and Hägglund, T., "The Future of PID Control," *Control Engineering Practice*, Vol. 9, No. 11, 2001, pp. 1163–1175. https://doi.org/10.1016/S0967-0661(01)00062-4.

[2] Blue, P., Odenthal, D., and Muhler, M., "Designing Robust Large Envelope Flight Controllers for High-Performance Aircraft," *AIAA Paper 2002–4450*, August 2002. https://doi.org/10.2514/6.2002-4450.

[3] Boyd, S., Baratt, C., and Norman, S., ""Linear Controller Design: Limits of Performance Via Convex Optimization," *Proceedings of the IEEE*, Vol. 78, No. 3, 1990, pp. 529–574. https://doi.org/10.1109/5.52229.

[4] Awad, M., and Khanna, R., *Efficient Learning Machines Theories,Concepts, and Applications for Engineers and System Designers*, 1st ed., Springer nature, California, 2015, Chap. 1.

[5] Géron, A., *Hands-on Machine Learning With Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 3rd ed., O'Reilly, California, 2022, Chap. 1.

[6] Mitiguy, P., *Dynamics of Mechanical, Aerospace, and Bio/Robotic Systems*, student ed., MotionGenesis, California, 2022, Chap. 25.

[7] Åström, K. J., and Furuta, K., "Swinging Up A Pendulum by Energy Control," *Automatica*, Vol. 36, No. 2, 2000, pp. 287–295. https://doi.org/10.1016/S0005-1098(99)00140-5.

[8] Lundberg, K. H., and Barton, T. W., "History of Inverted-Pendulum Systems," *IFAC Proceedings Volumes*, Vol. 42, No. 24, 2010, pp. 131–135. https://doi.org/10.3182/20091021-3-JP-2009.00025.

[9] Barton, T. W., "Stabilizing the Dual Inverted Pendulum," *IFAC Proceedings Volumes*, Vol. 42, No. 24, 2010, pp. 113–118. https://doi.org/10.3182/20091021-3-JP-2009.00022.

[10] Siebert, W. M., "The Dynamics of Feedback Systems," *Circuits, Signals, and Systems*, MIT Press, Massachusetts, 1986, pp. 177–182.

[11] Blitzer, L., "Inverted Pendulum," *American Journal of Physics*, Vol. 33, No. 12, 1965, pp. 1076–1078. https://doi.org/10.1119/1.1971158.

[12] Cannon, R. H., "Some Case Studies in Automatic Control," *Dynamics of Physical Systems*, McGraw-Hill, New York, 1967, pp. 703–710.

[13] Kalmus, H. P., "The Inverted Pendulum," *American Journal of Physics*, Vol. 38, No. 7, 1970, pp. 874–878. https://doi.org/10.1119/1.1976486.

[14] Ogata, K., "Introduction to Control Systems," *Modern Control Engineering*, Prentice-Hall, Boston, 1970, pp. 1–3.

[15] Acurio, E., "Mechanical Redesign and Control With A PLC of An Inverted Pendulum," *ResearchGate*, June 2015. Retrieved 14 April 2024 from https://www.researchgate.net/publication/316923344_Mechanical_redesign_and_control_with_a_PLC_of_an_inverted_pendulum.

[16] Kaheman, K., Fasel, U., Bramburger, J. J., Strom, B., Kutz, J. N., and Brunton, S. L., "The Experimental Multi-Arm Pendulum on A Cart: A Benchmark System for Chaos, Learning, and Control," *HardwareX*, Vol. 15, 2023, pp. e00465–e00465. https://doi.org/10.1016/j.ohx.2023.e00465.

[17] Han, K.-C., and Kim, J.-Y., "Posture Stabilizing Control of Quadruped Robot Based on Cart-Inverted Pendulum Model," *Intelligent Service Robotics*, Vol. 16, No. 5, 2023, pp. 521–536. https://doi.org/10.1007/s11370-023-00480-8.

[18] Chang, L., Piao, S., Leng, X., He, Z., and Zhu, Z., "Inverted Pendulum Model for Turn-Planning for Biped Robot," *Physical Communication*, Vol. 42, 2020, pp. 101168–. https://doi.org/10.1016/j.phycom.2020.101168.

[19] Pei, J., and Rothhaar, P., "Demonstration of the Space Launch System Augmenting Adaptive Control Algorithm on Pole-Cart Platform," *AIAA Paper 2018–0608*, January 2018. https://doi.org/10.2514/6.2018-0608.

[20] Irfan, S., Zhao, L., Ullah, S., Mehmood, A., and Fasih Uddin Butt, M., "Control Strategies for Inverted Pendulum: A Comparative Analysis of Linear, Nonlinear, and Artificial Intelligence Approaches," *Plos one*, Vol. 19, No. 3, 2024, pp. e0298093–e0298093. https://doi.org/10.1371/journal.pone.0298093.

[21] Kumar, Y., and Kumar, P., "Empirical Study of Deep Reinforcement Learning Algorithms for CartPole Problem," *2024 11th International Conference on Signal Processing and Integrated Networks (SPIN)*, 2024, pp. 78–83. https://doi.org/10.1109/SPIN60856.2024.10512107.

[22] Jo, E., and Kim, Y., "Performance Comparison of Reinforcement Learning Algorithms in the Cartpole Game using Unity ML-Agents," *Journal of Theoretical and Applied Information Technology*, Vol. 102, No. 16, 2024. Retrieved 14 April 2024 from https://www.jatit.org/volumes/Vol102No16/7Vol102No16.pdf.

[23] Rio, A., Jimenez, D., and Serrano, J., "Comparative Analysis of A3C and PPO Algorithms in Reinforcement Learning: A Survey on General Environments," *IEEE Access*, Vol. 12, 2024, pp. 146795–146806. https://doi.org/10.1109/ACCESS.2024.3472473.

[24] Abdusamadov, A., "Design and Implementation of an Inverted Pendulum Control System using FPGA and Reinforcement Learning," 2023. Retrieved 14 April 2024 from https://webthesis.biblio.polito.it/27644/1/tesi.pdf.

[25] Perkins, R., "Rapid Adaptation of Deep Learning Teaches Drones to Survive Any Weather," *Caltech*, 4 May 2022. Retrieved 14 April 2024 from https://www.caltech.edu/about/news/rapid-adaptation-of-deep-learning-teaches-drones-to-survive-any-weather.

[26] Saj, V., Lee, B., Kalathil, D., and Benedict, M., "Robust Reinforcement Learning Algorithm for Vision-based Ship Landing of UAVs," *arXiv (Cornell University)*, 2022. https://doi.org/10.48550/arxiv.2209.08381.

[27] Sutton, R. S., and Barto, A. G., *Reinforcement Learning An Introduction*, MIT Press, Massachusetts, 1998.

[28] Murphy, K., "A Brief Introduction to Reinforcement Learning," *University of British Columbia*, 1998. Retrieved 14 April 2024 from https://www.cs.ubc.ca/~murphyk/Bayes/pomdp.html.

[29] Siddhardhan, S., "1.2. Supervised vs Unsupervised vs Reinforcement Learning — Types of Machine Learning," 2021. Retrieved 14 April 2024 from https://www.youtube.com/watch?v=Atg-Sl32vOo.

[30] Barto, A. G., Sutton, R. S., and Anderson, C. W., "Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 13, No. 5, 1983, pp. 843–846. https://doi.org/10.1109/TSMC.1983.6313077.

[31] "Reinforcement Learning Agents," , MathWorks, Retrieved 14 April 2024 from https://www.mathworks.com/help/reinforcement-learning/ug/create-agents-for-reinforcement-learning.html.

[32] Ravichandiran, S., *Hands-on Reinforcement Learning with Python: Master Reinforcement Learning and Deep Reinforcement Learning by Building Interlligent App*, 1st ed., Packt Publishing Ltd, London, 2018.

[33] "Knowledge-Based Control Systems Summary," , Aerostudents, Retrieved 14 April 2024 from http://www.aerostudents.com/courses/knowledge-based-control-systems/knowledgeBasedControlSystemsFullVersion.pdf.

[34] Shinners, S. M., *Modern Control System Theory and Design*, 2nd ed., Wiley, New York, 1998.

[35] Dorf, R. C., and Bishop, R. H., *Modern Control Systems*, 13th ed., Pearson, Boston, 2016.

[36] Hunter, J., *Aerospace Engineering 168 Course Reader: Aerospace Vehicle Dynamics and Control*, Maple Press, California, 2023.

[37] Raginsky, M., Liberzon, D., and Seiler, P., "Linearization of Nonlinear Models," *University of Illinois Urbana-Champaign*, 2021. Retrieved 14 April 2024 from https://courses.grainger.illinois.edu/ECE486/fa2021/documentation/lectures/.

[38] Kuśmierz, B., Gromaszek, K., and Kryk, K., "Inverted Pendulum Model Linear–Quadratic Regulator (LQR)," *SPIE*, Vol. 10808, 2018, pp. 1921–1928. https://doi.org/10.1117/12.2501686.

[39] Mojumder, M. R. H., and Roy, N. K., "PID, LQR, and LQG Controllers to Maintain the Stability of an AVR System at Varied Model Parameters," *2021 5th International Conference on Electrical Engineering and Information Communication Technology (ICEEICT)*, 2021, pp. 1–6. https://doi.org/10.1109/ICEEICT53905.2021.9667897.

[40] "Reinforcement Learning in 3 Hours — Full Course Using Python," , Retrieved 14 April 2024 from https://www.youtube.com/watch?v=Mut_u40Sqz4&list=PL3uHAJ2WXdN0k4Ml-D2agB7bjySFyCRqZ.

[41] "Installation," , Stable Baselines3, Retrieved 14 April 2024 from https://www.stable-baselines3.readthedocs.io/en/master/guide/install.html.

[42] Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N., "Stable-Baselines3: Reliable Reinforcement Learning Implementations," *Journal of Machine Learning Research*, Vol. 22, No. 268, 2021, pp. 1–8. Retrieved 14 April 2024 from https://jmlr.org/papers/v22/20-1364.html.

[43] "Classic Cart-Pole System," , OpenAI Gym GitHub Repository, Retrieved 14 April 2024 from https://github.com/openai/gym/blob/master/gym/envs/classic_control/cartpole.py#L96.

[44] "Experiment Outputs," , OpenAI Spinning Up, Retrieved 14 April 2024 from https://spinningup.openai.com/en/latest/user/saving_and_loading.html?highlight=episode.

[45] "Part 2: Kinds of RL Algorithms," , OpenAI Spinning Up, Retrieved 14 April 2024 from https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html#citations-below.

[46] "RL Algorithms," , Stable Baselines3, Retrieved 14 April 2024 from https://stable-baselines3.readthedocs.io/en/master/guide/algos.html.

[47] Majumder, A., "Deep Reinforcement Learning," 2021, pp. 305–447. https://doi.org/10.1007/978-1-4842-6503-1_5.

[48] "Reinforcement Learning Onramp," , MathWorks, Retrieved 14 April 2024 from https://matlabacademy.mathworks.com/details/reinforcement-learning-onramp/reinforcementlearning.

[49] Raju, A. M., and Vanschoren, J., "Analyzing Policy Gradient Approaches Towards Rapid Policy Transfer," 202. Retrieved 14 April 2024 from https://pure.tue.nl/ws/portalfiles/portal/167952556/Raju_A..pdf.

[50] "Getting Started," , Stable Baselines3, Retrieved 14 April 2024 from https://stable-baselines3.readthedocs.io/en/master/guide/quickstart.html.

[51] "Evaluation Helper," , Stable Baselines3, Retrieved 14 April 2024 from https://stable-baselines.readthedocs.io/en/master/common/evaluation.html.

[52] "Tensorboard Integration," , Stable Baselines3, Retrieved 14 April 2024 from https://stable-baselines3.readthedocs.io/en/master/guide/tensorboard.html.

[53] "Part 1: Key Concepts in RL," , OpenAI Spinning Up, Retrieved 14 April 2024 from https://spinningup.openai.com/en/latest/spinningup/rl_intro.html#key-concepts-and-terminology.

Appendix A: Governing Equations of Motion of the Inverted Pendulum System

The methodology throughout this Appendix follows that developed by Dr. Paul Mitiguy in [5]. Consider the free body diagram that shows the translational (horizontal and vertical) motion of an inverted pendulum system below:

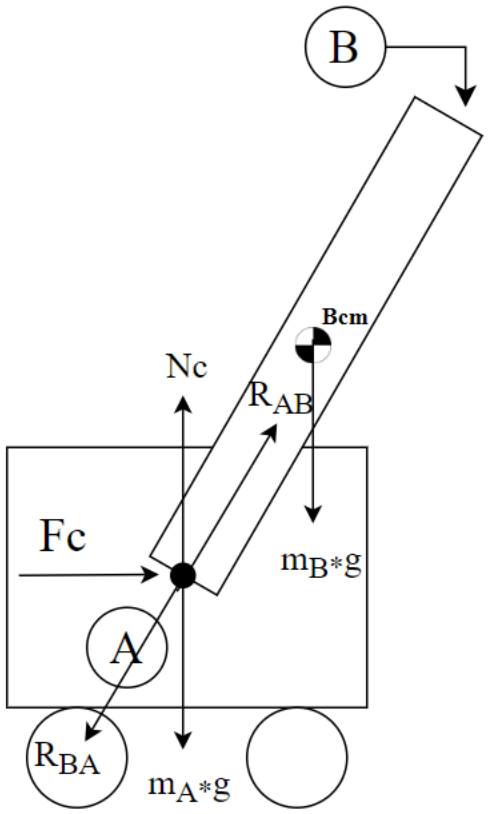


Figure A.1: Translational motion of the inverted pendulum system.

Since cart A and pole B have equal and opposite reaction forces (R_{AB} and R_{BA}), $|R_{AB}| = |R_{BA}|$ and they cancel each other out according to Newton's Third Law of Motion. Through figure A.1, it is clear to see that the only force in the horizontal direction is the control force applied to the cart (F_c), whereas the forces applied to the cart in the vertical direction are Earth's gravitational forces on cart A and pole B as well as the normal force (N_c) on cart A from N.
The schematic of the inverted pendulum system shown in figure 3.1 is used to find the position (${}^N\vec{r}^{Bcm}$), velocity (${}^N\vec{v}^{Bcm}$), and acceleration (${}^N\vec{a}^{Bcm}$) of the inverted pendulum system with respect to the N-frame:

$$ {}^N\vec{r}^{Bcm} = {}^N\vec{r}^{A} + {}^A\vec{r}^{Bcm} \tag{A.1} $$

$$ {}^N\vec{r}^{Bcm} = x\hat{n}x + L\hat{b}y \tag{A.2} $$

$$^{N}\vec{v}^{Bcm} = {}^{N}\frac{d}{dt}{}^{N}\vec{r}^{A} + {}^{N}\frac{d}{dt}{}^{A}\vec{r}^{BCM} \tag{A.3}$$

$$^{N}\vec{v}^{Bcm} = {}^{N}\vec{v}^{A} + {}^{B}\frac{d}{dt}{}^{A}\vec{r}^{Bcm} + {}^{N}\vec{\omega}^{B} \times {}^{A}\vec{r}^{Bcm} \tag{A.4}$$

$$^{N}\vec{v}^{Bcm} = \dot{x}\,\hat{n}x - \dot{\theta}\,\hat{b}z \times L\hat{b}y \tag{A.5}$$

$$^{N}\vec{v}^{Bcm} = \dot{x}\,\hat{n}x + L\dot{\theta}\,\hat{b}x \tag{A.6}$$

$$^{N}\vec{a}^{Bcm} = {}^{N}\frac{d}{dt}{}^{N}\vec{v}^{A} + {}^{N}\frac{d}{dt}{}^{A}\vec{v}^{Bcm} \tag{A.7}$$

$$^{N}\vec{a}^{Bcm} = {}^{N}\vec{a}^{A} + {}^{B}\frac{d}{dt}L\dot{\theta}\,\hat{b}x + {}^{N}\vec{\omega}^{B} \times L\dot{\theta}\,\hat{b}x \tag{A.8}$$

$$^{N}\vec{a}^{Bcm} = \ddot{x}\,\hat{n}x + L\ddot{\theta}\,\hat{b}x - \dot{\theta}\,\hat{b}z \times L\dot{\theta}\,\hat{b}x \tag{A.9}$$

$$^{N}\vec{a}^{Bcm} = \ddot{x}\,\hat{n}x + L\ddot{\theta}\,\hat{b}x - L\dot{\theta}^{2}\,\hat{b}y \tag{A.10}$$

A.0.1 Translational Equations of Motion

The free body diagram in figure A.1 shows the sum of forces applied to the inverted pendulum system in both the horizontal and vertical directions. The governing equations of motion in the horizontal and vertical directions, developed by Newton's Second Law of Motion, are thus:

$$\sum \vec{F}^{S} = m^{S\,N}\vec{a}^{Scm} \tag{A.11}$$

$$\sum \vec{F}^{S} = m_{A}\,{}^{N}\vec{a}^{A} + m_{B}\,{}^{N}\vec{a}^{Bcm} \tag{A.12}$$

$$F_{C}\,\hat{n}x - m_{A}\,g\,\hat{n}y + N_{c}\,\hat{n}y - m_{B}\,g\,\hat{n}y = m_{A}\,\ddot{x}\hat{n}x + m_{B}\,(\ddot{x}\hat{n}x + L\ddot{\theta}\,\hat{b}x - L\dot{\theta}^{2}\,\hat{b}y) \tag{A.13}$$

where

$$\hat{b}x = \cos\theta\,\hat{n}x - \sin\theta\,\hat{n}y \tag{A.14}$$

$$\hat{b}y = \sin\theta\,\hat{n}x + \cos\theta\,\hat{n}y \tag{A.15}$$

Substitute eq. A.14 and eq. A.15 into eq. A.13 and rearrange to combine like terms:

$$(\hat{n}x) : F_{C} = (m_{A} + m_{B})\ddot{x} + m_{B}L(\ddot{\theta}\cos\theta - \dot{\theta}^{2}\sin\theta) \tag{A.16}$$

$$(\hat{n}y) : (m_{A} + m_{B})g - N_{c} = m_{B}L(\ddot{\theta}\sin\theta + \dot{\theta}^{2}\cos\theta) \tag{A.17}$$

A.0.2 Rotational Equations of Motion

The inverted pendulum system also has rotational motion, as shown in figure A.2:

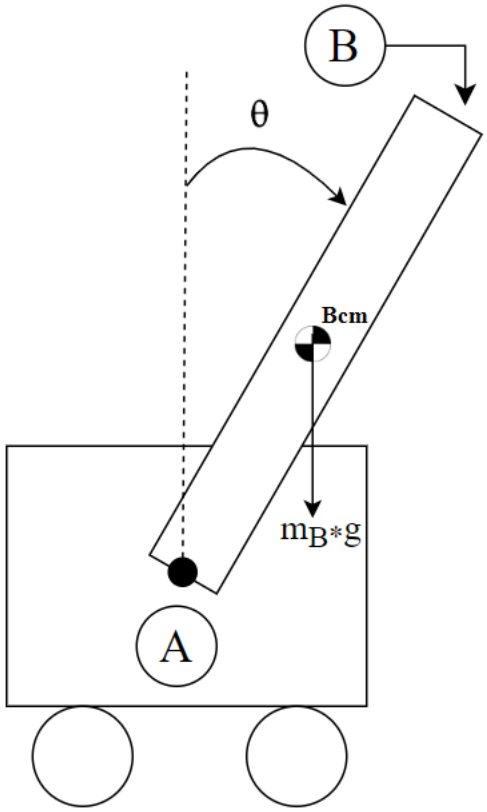


Figure A.2: Rotational motion of the inverted pendulum system.

The about point chosen in this case is the point at which cart A and pole B are connected, and the equations of motion are modeled with respect to Newton-Euler's Laws of Rotational Motion designed for rigid body dynamics:

$$\sum \vec{M}^{B/A} =^{N} \frac{d}{dt}\,{}^{N}\vec{H}^{B/A} \tag{A.18}$$

$${}^{N}\vec{H}^{B/A} = I_{zz}^{B/A} \cdot {}^{N}\vec{\omega}^{B} +^{A} \vec{r}^{Bcm} \times m_B\,{}^{N}\vec{v}^{A} \tag{A.19}$$

where

$$I_{zz}^{B/A} = I_{zz}^{B/Bcm} + I_{zz}^{Bcm/A} = (I_{zz} + mL^2)\,\hat{b}z \tag{A.20}$$

$${}^{A}\vec{r}^{Bcm} \times \vec{F}^{B} =^{N} \frac{d}{dt}\,(I_{zz}^{B/A} \cdot {}^{N}\vec{\omega}^{B} +^{A} \vec{r}^{Bcm} \times m_B\,{}^{N}\vec{v}^{A}) \tag{A.21}$$

$$L\hat{b}y \times (-m_B g\hat{n}y) = I_{zz}^{B/A} \cdot {}^{N}\frac{d}{dt}\,{}^{N}\vec{\omega}^{B} +^{A} \vec{r}^{Bcm} \times m_B\,{}^{N}\frac{d}{dt}\,{}^{N}\vec{v}^{A} \tag{A.22}$$

$$(L\sin\theta\,\hat{n}x + L\cos\theta\,\hat{n}y) \times (-m_B g\,\hat{n}y) = I_{zz}^{B/A} \cdot {}^{N}\vec{\alpha}^{B} +^{A} \vec{r}^{Bcm} \times m_B\,{}^{N}\vec{a}^{A} \tag{A.23}$$

$$-m_B gL\sin\theta\,\hat{n}z = (I_{zz} + mL^2)\,\hat{b}z \cdot (-\ddot{\theta}\,\hat{b}z) + (L\hat{b}y \times m_B\ddot{x}\hat{n}x) \tag{A.24}$$

Since $\hat{n}z = \hat{b}z$ and $\hat{n}x$ can be converted to $\hat{b}x$ and $\hat{b}y$,

$$-m_B gL\sin\theta\,\hat{b}z = (I_{zz} + mL^2)\,\hat{b}z \cdot (-\ddot{\theta}\,\hat{b}z) + (L\hat{b}y \times (m_B\ddot{x}\cos\theta\,\hat{b}x + m_B\ddot{x}\sin\theta\,\hat{b}y)) \tag{A.25}$$

$$-m_B gL\sin\theta = -(I_{zz} + mL^2)\ddot{\theta} - m_B L\cos\theta\ddot{x} \tag{A.26}$$

$$m_B gL\sin\theta = (I_{zz} + mL^2)\ddot{\theta} + m_B L\cos\theta\ddot{x} \tag{A.27}$$

Note that $L =$ half of the pole length, thus

$$I_{zz} = \frac{1}{12}m_B(2L)^2 = \frac{1}{3}m_B L^2 \tag{A.28}$$

Eq. A.27 thus becomes:

$$m_B gL\sin\theta = (\frac{1}{3}m_B L^2 + m_B L^2)\ddot{\theta} + m_B L\cos\theta\ddot{x} \tag{A.29}$$

$$m_B gL\sin\theta = \frac{4}{3}m_B L^2\ddot{\theta} + m_B L\cos\theta\ddot{x} \tag{A.30}$$

Isolate and solve for $\ddot{x}$:

$$\ddot{x} = \frac{m_B L(g\sin\theta - \frac{4}{3}L\ddot{\theta})}{m_B L\cos\theta} \tag{A.31}$$

$$\ddot{x} = \frac{g\sin\theta - \frac{4}{3}L\ddot{\theta}}{\cos\theta} \tag{A.32}$$

Substitute eq. A.32 into eq. A.16 and solve for $\ddot{\theta}$:

$$F_c - (m_A + m_B)\left(\frac{g\sin\theta - \frac{4}{3}L\ddot{\theta}}{\cos\theta}\right) - m_B L\cos\theta\,\ddot{\theta} + m_B L\sin\theta\,\dot{\theta}^2 = 0 \tag{A.33}$$

$$\ddot{\theta}L\left(\frac{4}{3}-\frac{m_B\cos^2\theta}{m_A+m_B}\right)=g\sin\theta+\cos\theta\left(\frac{-F_c-m_BL\dot{\theta}^2\sin\theta}{m_A+m_B}\right) \tag{A.34}$$

$$\ddot{\theta}=\frac{g\sin\theta+\cos\theta\left(\frac{-F_c-m_BL\dot{\theta}^2\sin\theta}{m_A+m_B}\right)}{L\left(\frac{4}{3}-\frac{m_B\cos^2\theta}{m_A+m_B}\right)} \tag{A.35}$$

$\ddot{x}$ can also be found using eq. A.16:

$$F_C=(m_A+m_B)\ddot{x}+m_BL(\ddot{\theta}\cos\theta-\dot{\theta}^2\sin\theta) \tag{A.36}$$

$$F_C-m_BL(\ddot{\theta}\cos\theta-\dot{\theta}^2\sin\theta)=(m_A+m_B)\ddot{x} \tag{A.37}$$

$$\ddot{x}=\frac{F_C+m_BL(\dot{\theta}^2\sin\theta-\ddot{\theta}\cos\theta)}{m_A+m_B} \tag{A.38}$$

Appendix B: Linearization Process for the Inverted Pendulum System

This process, using perturbation method, follows the steps developed by [36]. It begins with equations 3.5 and 3.6 derived in Chapter 3:

$$F_C = (m_A + m_B)\ddot{x} + m_B L\ddot{\theta}\cos\theta - m_B L\dot{\theta}^2 \sin\theta \tag{B.1}$$

$$m_B g L \sin\theta = (I_{zz} + m_B L^2)\ddot{\theta} + m_B L \cos\theta \ddot{x} \tag{B.2}$$

Let

$$\begin{aligned} F_C &= U \\ U &= U_1 + u \\ \ddot{X} &= \ddot{X}_1 + \ddot{x} \\ \theta &= \Theta_1 + \theta \\ \dot{\theta} &= \dot{\Theta}_1 + \dot{\theta} \\ \ddot{\theta} &= \ddot{\Theta}_1 + \ddot{\theta} \end{aligned} \tag{B.3}$$

The process to linearize the exact equations of motion begins by substituting the steady-state and perturbation values into the exact equations:

$$U_1 + u = (m_A + m_B)(\ddot{X}_1 + \ddot{x}) + m_B L(\ddot{\Theta}_1 + \ddot{\theta})\cos(\Theta_1 + \theta) - m_B L(\dot{\Theta}_1 + \dot{\theta})^2 \sin(\Theta_1 + \theta) \tag{B.4}$$

Apply trigonometric identities and multiply all the terms out:

$$\cos(\Theta_1 + \theta) = \cos\Theta_1 \cos\theta - \sin\Theta_1 \sin\theta \tag{B.5}$$

$$\sin(\Theta_1 + \theta) = \sin\Theta_1 \cos\theta + \cos\Theta_1 \sin\theta \tag{B.6}$$

$$\begin{aligned} U_1 + u = &(m_A + m_B)\ddot{X}_1 + (m_A + m_B)\ddot{x} + m_B L\ddot{\Theta}_1 \cos\Theta_1 \cos\theta - m_B L\ddot{\Theta}_1 \sin\Theta_1 \sin\theta \\ &+ m_B L\ddot{\theta}\cos\Theta_1 \cos\theta - m_B L\ddot{\theta}\sin\Theta_1 \sin\theta - m_B L\dot{\Theta}_1^2 \sin\Theta_1 \cos\theta \\ &- m_B L\dot{\Theta}_1^2 \cos\Theta_1 \sin\theta - 2m_B L\dot{\Theta}_1\dot{\theta}\sin\Theta_1 \cos\theta - 2m_B L\dot{\Theta}_1\dot{\theta}\cos\Theta_1 \sin\theta \\ &- m_B L\dot{\theta}^2 \sin\Theta_1 \cos\theta - m_B L\dot{\theta}^2 \cos\Theta_1 \sin\theta \end{aligned} \tag{B.7}$$

For small angle (θ less than 13°) [36], assume small angle approximation:

$$\cos\theta \to 1 \tag{B.8}$$

$$\sin\theta \to \theta \tag{B.9}$$

$$\begin{aligned} U_1 + u = &(m_A + m_B)\ddot{X}_1 + (m_A + m_B)\ddot{x} + m_B L\ddot{\Theta}_1 \cos\Theta_1 \cancel{\cos\theta}^{\,1} - m_B L\ddot{\Theta}_1 \sin\Theta_1 \cancel{\sin\theta}^{\,\theta} \\ &+ m_B L\ddot{\theta}\cos\Theta_1 \cancel{\cos\theta}^{\,1} - m_B L\ddot{\theta}\sin\Theta_1 \cancel{\sin\theta}^{\,\theta} - m_B L\dot{\Theta}_1^2 \sin\Theta_1 \cancel{\cos\theta}^{\,1} \\ &- m_B L\dot{\Theta}_1^2 \cos\Theta_1 \cancel{\sin\theta}^{\,\theta} - 2m_B L\dot{\Theta}_1\dot{\theta}\sin\Theta_1 \cancel{\cos\theta}^{\,1} - 2m_B L\dot{\Theta}_1\dot{\theta}\cos\Theta_1 \cancel{\sin\theta}^{\,\theta} \\ &- m_B L\dot{\theta}^2 \sin\Theta_1 \cancel{\cos\theta}^{\,1} - m_B L\dot{\theta}^2 \cos\Theta_1 \cancel{\sin\theta}^{\,\theta} \end{aligned} \tag{B.10}$$

Equation B.9 thus becomes:

$$\begin{aligned} U_1 + u =& (m_A + m_B)\ddot{X}_1 + (m_A + m_B)\ddot{x} + m_B L\ddot{\Theta}_1 \cos\Theta_1 - m_B L\ddot{\Theta}_1 \sin\Theta_1 \theta \\ &+ m_B L\ddot{\theta}\cos\Theta_1 - m_B L\ddot{\theta}\sin\Theta_1\theta - m_B L\dot{\Theta}_1^2 \sin\Theta_1 \\ &- m_B L\dot{\Theta}_1^2 \cos\Theta_1\theta - 2m_B L\dot{\Theta}_1\dot{\theta}\sin\Theta_1 - 2m_B L\dot{\Theta}_1\dot{\theta}\cos\Theta_1\theta \\ &- m_B L\dot{\theta}^2\sin\Theta_1 - m_B L\dot{\theta}^2\cos\Theta_1\theta \end{aligned} \tag{B.11}$$

Subtract the steady-state equation:

$$\begin{aligned} \cancel{U_1} + u =& \cancel{(m_A + m_B)\ddot{X}_1} + (m_A + m_B)\ddot{x} + \cancel{m_B L\ddot{\Theta}_1 \cos\Theta_1} - m_B L\ddot{\Theta}_1 \sin\Theta_1 \theta \\ &+ m_B L\ddot{\theta}\cos\Theta_1 - m_B L\ddot{\theta}\sin\Theta_1\theta - \cancel{m_B L\dot{\Theta}_1^2 \sin\Theta_1} \\ &- m_B L\dot{\Theta}_1^2 \cos\Theta_1\theta - 2m_B L\dot{\Theta}_1\dot{\theta}\sin\Theta_1 - 2m_B L\dot{\Theta}_1\dot{\theta}\cos\Theta_1\theta \\ &- m_B L\dot{\theta}^2\sin\Theta_1 - m_B L\dot{\theta}^2\cos\Theta_1\theta \end{aligned} \tag{B.12}$$

Neglect higher-order perturbation terms (H.O.T):

$$\begin{aligned} u =& (m_A + m_B)\ddot{x} - m_B L\ddot{\Theta}_1 \sin\Theta_1\theta + m_B L\ddot{\theta}\cos\Theta_1 - m_B L\ddot{\theta}\sin\Theta_1\theta \xrightarrow{H.O.T} - m_B L\dot{\Theta}_1^2\cos\Theta_1\theta \\ &- 2m_B L\dot{\Theta}_1\dot{\theta}\sin\Theta_1 \xrightarrow{H.O.T} - 2m_B L\dot{\Theta}_1\dot{\theta}\cos\Theta_1\theta - m_B L\dot{\theta}^2\sin\Theta_1 \xrightarrow{H.O.T} - m_B L\dot{\theta}^2\cos\Theta_1\theta \xrightarrow{H.O.T} \end{aligned} \tag{B.13}$$

Equation B.12 thus becomes:

$$u = (m_A + m_B)\ddot{x} - m_B L\ddot{\Theta}_1 \sin\Theta_1\theta + m_B L\ddot{\theta}\cos\Theta_1 - m_B L\dot{\Theta}_1^2\cos\Theta_1\theta - 2m_B L\dot{\Theta}_1\dot{\theta}\sin\Theta_1 \tag{B.14}$$

Assume no steady-state angular velocities and angular acceleration:

$$\dot{\Theta}_1 = 0 \tag{B.15}$$

$$\ddot{\Theta}_1 = 0 \tag{B.16}$$

$$u = (m_A + m_B)\ddot{x} - \cancel{m_B L\ddot{\Theta}_1 \sin\Theta_1\theta} + m_B L\ddot{\theta}\cos\Theta_1 - \cancel{m_B L\dot{\Theta}_1^2\cos\Theta_1\theta} - \cancel{2m_B L\dot{\Theta}_1\dot{\theta}\sin\Theta_1} \tag{B.17}$$

Therefore, the linearized forced equation becomes:

$$u = (m_A + m_B)\ddot{x} + m_B L\cos\Theta_1\ddot{\theta} \tag{B.18}$$

Similar process was used to linearize equation B.2:

$$m_B g L\sin\theta = (I_{zz} + m_B L^2)\ddot{\theta} + m_B L\cos\theta\ddot{x} \tag{B.19}$$

Apply trigonometric identities:

$$m_B g L\sin(\Theta_1 + \theta) = (I_{zz} + m_B L^2)(\ddot{\Theta}_1 + \ddot{\theta}) + m_B L(\ddot{X}_1 + \ddot{x})\cos(\Theta_1 + \theta) \tag{B.20}$$

Multiply all the terms out:

$$\begin{aligned} m_B g L\sin\Theta_1\cos\theta + m_B g L\cos\Theta_1\sin\theta =& (I_{zz} + m_B L^2)\ddot{\Theta}_1 + (I_{zz} + m_B L^2)\ddot{\theta} \\ &+ m_B L\ddot{X}_1\cos\Theta_1\cos\theta - m_B L\ddot{X}_1\sin\Theta_1\sin\theta \\ &+ m_B L\ddot{x}\cos\Theta_1\cos\theta - m_B L\ddot{x}\sin\Theta_1\sin\theta \end{aligned} \tag{B.21}$$

Per small angle approximation,

$$\begin{aligned} m_B g L \sin\Theta_1 \cancelto{1}{\cos\theta} + m_B g L \cos\Theta_1 \cancelto{\theta}{\sin\theta} =& (I_{zz} + m_B L^2)\ddot{\Theta}_1 + (I_{zz} + m_B L^2)\ddot{\theta} \\ &+ m_B L \ddot{X}_1 \cos\Theta_1 \cancelto{1}{\cos\theta} - m_B L \ddot{X}_1 \sin\Theta_1 \cancelto{\theta}{\sin\theta} \\ &+ m_B L \ddot{x} \cos\Theta_1 \cancelto{1}{\cos\theta} - m_B L \ddot{x} \sin\Theta_1 \cancelto{\theta}{\sin\theta} \end{aligned} \tag{B.22}$$

$$\begin{aligned} m_B g L \sin\Theta_1 + m_B g L \cos\Theta_1 \theta =& (I_{zz} + m_B L^2)\ddot{\Theta}_1 + (I_{zz} + m_B L^2)\ddot{\theta} \\ &+ m_B L \ddot{X}_1 \cos\Theta_1 - m_B L \ddot{X}_1 \sin\Theta_1 \theta \\ &+ m_B L \ddot{x} \cos\Theta_1 - m_B L \ddot{x} \sin\Theta_1 \theta \end{aligned} \tag{B.23}$$

Subtract steady-state equation and neglect H.O.T:

$$\begin{aligned} \cancel{m_B g L \sin\Theta_1} + m_B g L \cos\Theta_1 \theta =& \cancel{(I_{zz} + m_B L^2)\ddot{\Theta}_1} + (I_{zz} + m_B L^2)\ddot{\theta} \\ &+ \cancel{m_B L \ddot{X}_1 \cos\Theta_1} - m_B L \ddot{X}_1 \sin\Theta_1 \theta \\ &+ m_B L \ddot{x} \cos\Theta_1 - \cancel{m_B L \ddot{x} \sin\Theta_1 \theta}^{H.O.T} \end{aligned} \tag{B.24}$$

Assume no steady-state horizontal acceleration:

$$\ddot{X}_1 = 0 \tag{B.25}$$

$$m_B g L \cos\Theta_1 \theta = (I_{zz} + m_B L^2)\ddot{\theta} - \cancel{m_B L \ddot{X}_1 \sin\Theta_1 \theta} + m_B L \ddot{x} \cos\Theta_1 \tag{B.26}$$

Thus, the second linearized equation is:

$$m_B g L \cos\Theta_1 \theta = (I_{zz} + m_B L^2)\ddot{\theta} + m_B L \cos\Theta_1 \ddot{x} \tag{B.27}$$

The linearized equations corresponding to B.1 and B.2, respectively, are:

$$u = (m_A + m_B)\ddot{x} + m_B L \cos\Theta_1 \ddot{\theta} \tag{B.28}$$

$$m_B g L \cos\Theta_1 \theta = (I_{zz} + m_B L^2)\ddot{\theta} + m_B L \cos\Theta_1 \ddot{x} \tag{B.29}$$

Appendix C: Python Code for the Comparison of Nonlinear versus Linearized Inverted Pendulum Systems

```
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt

# Define variables
m_B = 0.1 # [kg] pole mass
m_A = 1 # [kg] cart mass
L = 0.5 # [m] half of the pole length
g = 9.8 # [m/s^2] gravitational acceleration on Earth
u = 0 # [N] control input/force applied
I_zz = (1/12)*m_B*(2*L)**2 # mass moment of inertia about B_cm

# Define functions
def nonlin_ips(t1, S1):
    x1, v1, theta1, w1 = S1
    D1 = I_zz*(m_A+m_B)+m_A*m_B*L**2+m_B**2*L**2*(np.sin(theta1))**2
    f1 = v1
    f2 = (1/D1)*((I_zz+m_B*L**2)*m_B*L*np.sin(theta1)*w1**2 - m_B**2*L**2*
                                        np.cos(theta1)*np.sin(theta1)*g +
                                         (I_zz+m_B*L**2)*u)
    f3 = w1
    f4 = (1/D1)*(-m_B**2*L**2*np.sin(theta1)*np.cos(theta1)*w1**2 + (m_A+
                                        m_B)*m_B*g*L*np.sin(theta1) - m_B
                                        *L*np.cos(theta1)*u)
    return [f1, f2, f3, f4]

def lin_ips(t2, S2):
    x2, v2, theta2, w2 = S2
    D2 = I_zz*(m_A+m_B)+m_A*m_B*L**2
    g1 = v2
    g2 = (1/D2)*(-m_B**2*L**2*g*theta2 + (I_zz+m_B*L**2)*u)
    g3 = w2
    g4 = (1/D2)*((m_A+m_B)*m_B*g*L*theta2 + (-m_B*L)*u)
    return [g1, g2, g3, g4]

# Define initial conditions
x_0 = 0
v_0 = 0
theta_0 = 0
w_0 = 0.1
S_0 = np.array([x_0, v_0, theta_0, w_0])

t = np.linspace(0, 2, 200)
sol1 = solve_ivp(nonlin_ips, t_span=(0, max(t)), y0=S_0, t_eval=t)
sol2 = solve_ivp(lin_ips, t_span=(0, max(t)), y0=S_0, t_eval=t)

x_sol1 = sol1.y[0]
theta_sol1 = sol1.y[2]
```

```
x_sol2 = sol2.y[0]
theta_sol2 = sol2.y[2]

p1 = plt.figure(1)
plt.plot(t, x_sol1, 'g-', label='Nonlinear', linewidth=2)
plt.plot(t, x_sol2, 'm--', label='Linear', linewidth=2)
plt.xlabel('Time [sec]')
plt.ylabel('Cart Position [m]')
plt.title('Inverted Pendulum Cart Position vs. Time')
plt.legend()
p1.show()

p2 = plt.figure(2)
plt.plot(t, theta_sol1*(180/np.pi), 'b-', label='Nonlinear', linewidth=2)
plt.plot(t, theta_sol2*(180/np.pi), 'r--', label='Linear', linewidth=2)
plt.xlabel('Time [sec]')
plt.ylabel(r'$\theta$ [m]')
plt.title('Inverted Pendulum Pole Angle vs. Time')
plt.legend()
p2.show()

input()
```

Appendix D: Python Code for the Open-Loop Stability and Controllability Analysis of The Inverted Pendulum System

```
import matplotlib.pyplot as plt
import numpy as np
import control as ct

def plotData(x, y, color, title, outputlabel):
    plt.figure(figsize=(8,4))
    plt.plot(x, y, color=color, linewidth=4,label='x')
    plt.xlabel('Time [s]',fontsize=16)
    plt.ylabel(outputlabel,fontsize=16)
    plt.title(title, fontsize=14)
    plt.tick_params(axis='both',which='major',labelsize=14)
    plt.grid()
    result = plt.show()
    return result

# Define constants and variables
m_A = 1 # [kg] mass of the cart
m_B = 0.1 # [kg] mass of the pole
L = 0.5 # half of the pole length
g = 9.8 # [m/s^2] gravitational acceleration constant
I_zz = (1/12)*m_B*(2*L)**2 # [kgm^2] moment of inertia about B_cm

# Denominator
D = (m_A*m_B*L**2) + (I_zz*(m_A+m_B))

# Define the state space model
A = np.array([[0, 1, 0, 0],
              [0, 0, (1/D)*(-m_B**2*L**2*g), 0],
              [0, 0, 0, 1],
              [0, 0, (1/D)*((m_A+m_B)*m_B*g*L), 0]])
print('A= ', A)

B = np.array([[0],
              [(1/D)*(I_zz+m_B*L**2)],
              [0],
              [(1/D)*(-m_B*L)]])
print('B= ', B)

C = np.eye(4)
D = np.zeros((C.shape[0], B.shape[1]))

sys = ct.ss(A, B, C, D)

# Check stability by finding the location of the poles
ct.damp(sys, doprint=True)
print('Even though the poles are non-negative real part, the system is
                                    still neutrally stable since the
                                    poles are zeros.')
```

```
# Check controllability
Co = ct.ctrb(A, B)
print(Co)
rankCo = np.linalg.matrix_rank(Co)
print('Since the rank of Co = the number of states = {}, the system is
                                        fully controllable.'.format(rankCo))

# Define simulation time and force input
startTime = 0
endTime = 1
timeSteps = 1000
simulationTime = np.linspace(startTime, endTime, timeSteps)
forceInput = np.zeros(len(simulationTime))
amplitude = 1
forceInput[100:] = amplitude

# Plot
plotData(simulationTime, forceInput, 'k', 'Unit Step Input', 'Magnitude')
T, yout = ct.forced_response(sys, T=simulationTime, U=forceInput, squeeze=
                                        True)
plotData(T,yout[0,:],'b','Cart Position in Response to a Unit Step Input',
                                         'x [m]')
plotData(T,yout[1,:],'m','Cart Velocity in Response to a Unit Step Input',
                                         r'$\dot{x}$ [m/s]')
plotData(T,yout[2,:]*(180/np.pi),'g','Pendulum Angle in Response to a Unit
                                         Step Input', r'$\theta$ [deg]')
plotData(T,yout[3,:]*(180/np.pi),'r','Pendulum Angular Velocity in
                                        Response to a Unit Step Input', r'$\
                                        dot{\theta}$ [deg/s]')
```

Appendix E: MATLAB/SIMULINK Code for the LQR Design and Closed-Loop Stability Analysis of The Inverted Pendulum System

```
clear all, clc, close all;

% Define constants and variables
m = 0.1;
M = 1;
L = 0.5;
g = 9.8;
I = (1/12)*m*(2*L)^2;

D = ((M*m*L^2) + (I*(M+m)));

% Define the state space model
A = [0 1 0 0;
     0 0 (1/D)*(-m^2*L^2*g) 0;
     0 0 0 1;
     0 0 (1/D)*((M+m)*m*g*L) 0];
B = [0;
    (1/D)*(I+m*L^2);
     0;
     (1/D)*(-m*L)];
C = eye(4);

D = zeros(size(C,1),size(B,2));

% Check the stability of the open-loop system for x and theta
sys = ss(A,B,C,D);
damp(sys)
disp('Even though the poles are non-negative real part, it is
   stable due to poles being zero.')

% Check the controllability of the open-loop system
Co = ctrb(A,B);
rankCo = rank(Co);
disp('Since the rank of Co = n = 4, the system is fully
   controllable.')

% Define simulation parameters
t0 = 0; % [s] initial sim time
tf = 20; % [s] final sim time
```

```
dt = 0.01; % [s] time step

% Tune Q and R matrices for the LQR controller
Q = diag([1 1 10 1]);
R = 0.001;

% Determine the LQR gain
[K_LQR,S,P] = lqr(A,B,Q,R)

% Simulate the Closed-loop system
sys_CL = ss(A-B*K_LQR,B,C,D);
damp(sys_CL)

% Start the simulation
open_system('CL_LQR.slx');
sim("CL_LQR.slx");

% Plot the results
figure,
subplot(2,2,1)
plot(ans.x(:,1),ans.x(:,2),'b')
xlabel('Time [sec]');
ylabel('x [m]');
set(gca,'fontsize',12);
set(findall(gcf,'type','line'),'linewidth',3);
grid on
subplot(2,2,2)
plot(ans.xdot(:,1),ans.xdot(:,2),'g')
xlabel('Time [sec]');
ylabel('$\dot{x}$ [m/s]','Interpreter','latex');
set(gca,'fontsize',12);
set(findall(gcf,'type','line'),'linewidth',3);
grid on
subplot(2,2,3)
plot(ans.theta(:,1),ans.theta(:,2),'r')
xlabel('Time [sec]');
ylabel('\theta [deg]');
set(gca,'fontsize',12);
set(findall(gcf,'type','line'),'linewidth',3);
grid on
subplot(2,2,4)
plot(ans.thetadot(:,1),ans.thetadot(:,2),'m')
xlabel('Time [sec]');
ylabel('$\dot{\theta}$ [deg/s]','Interpreter','latex');
set(gca,'fontsize',12);
```

```
set(findall(gcf,'type','line'),'linewidth',3);
grid on
sgtitle({"Responses of the Inverted Pendulum System", "with LQR
    Control"})
```

Appendix F: Python Script to Train and Save Model

These lines of code were developed using the Stable Baselines3 library.

```
# Importing Dependencies
import os
import gymnasium as gym
from stable_baselines3 import A2C, PPO
from stable_baselines3.common.monitor import Monitor
from stable_baselines3.common.env_checker import check_env
from stable_baselines3.common.callbacks import BaseCallback
from stable_baselines3.common.logger import HParam

# Create Log Directory
models_dir = "models/PPO"
logdir = "logs"

os.makedirs(models_dir, exist_ok=True) # If models directory doesn't exist
                                       , create one
os.makedirs(logdir, exist_ok=True) # If the log doesn't exist, create one

# Instantiate the Environment
env = gym.make("CustomCartPole")

# Train the RL agent
class HParamCallback(BaseCallback):
    """
    Saves the hyperparameters and metrics at the start of the training,
                                           and logs them to TensorBoard.
    """

    def _on_training_start(self) -> None:
        hparam_dict = {
            "algorithm": self.model.__class__.__name__,
            "learning rate": self.model.learning_rate,
            "gamma": self.model.gamma,
        }
        # define the metrics that will appear in the `HPARAMS` Tensorboard
                                                tab by referencing their tag
        # Tensorbaord will find & display metrics from the `SCALARS` tab
        metric_dict = {
            "rollout/ep_len_mean": 0,
            "train/value_loss": 0.0,
        }
        self.logger.record(
            "hparams",
            HParam(hparam_dict, metric_dict),
            exclude=("stdout", "log", "json", "csv"),
        )

    def _on_step(self) -> bool:
        return True
```

```
check_env(env)
model = PPO("MlpPolicy", env, verbose=1, tensorboard_log=logdir)
env = Monitor(env)

TIMESTEPS = 10000
for i in range(1,2):
    model.learn(total_timesteps=TIMESTEPS, tb_log_name='PPO',
                                           reset_num_timesteps=False,
                                           callback=HParamCallback())
    model.save(f"{models_dir}/{TIMESTEPS*i}")
```

Appendix G: Python Script to Evaluate and Test Model

These lines of code were developed using the Stable Baselines3 library.

```
import gymnasium as gym
from stable_baselines3 import A2C, PPO
from stable_baselines3.common.evaluation import evaluate_policy

models_dir = "models/A2C"

# Instantiate the Environment
env = gym.make("CustomCartPole")

# Load the trained/saved model
model = A2C.load(f"{models_dir}/1000000", env=env)

# Evaluate the trained RL agent
mean_reward, std_reward = evaluate_policy(model, model.get_env(),
                                          n_eval_episodes=10)
print(f"mean_reward: {mean_reward:.2f} +/- {std_reward:.2f}")

# Test run the trained/save model
vec_env = model.get_env()
obs = vec_env.reset()
for i in range(1000):
    action, _state = model.predict(obs, deterministic=True)
    obs, reward, done, info = vec_env.step(action)
```

Appendix H: Python Script for Generating CSV Output from the OpenAI Gymnasium's Cart Pole Environment

This cart pole script was obtained from OpenAI gymnasium [43] and revised to add the write-to-csv feature with the help from Connor Miholovich.

```
import math
from typing import Optional, Tuple, Union

import numpy as np

import csv
import os
import datetime

import gymnasium as gym
from gymnasium import logger, spaces
from gymnasium.envs.classic_control import utils
from gymnasium.error import DependencyNotInstalled
from gymnasium.experimental.vector import VectorEnv
from gymnasium.vector.utils import batch_space

timestamp = datetime.datetime.now().strftime("%Y%m%d_%H%M%S")
file_path = f'cart_pole_test_{timestamp}.csv'


class CustomCartPoleEnv(gym.Env[np.ndarray, Union[int, np.ndarray]]):
    """
    ### Description

    This environment corresponds to the version of the cart-pole problem
                                        described by Barto, Sutton, and
                                        Anderson in
    ["Neuronlike Adaptive Elements That Can Solve Difficult Learning
                                        Control Problem"](https://
                                        ieeexplore.ieee.org/document/
                                        6313077).
    A pole is attached by an un-actuated joint to a cart, which moves
                                        along a frictionless track.
    The pendulum is placed upright on the cart and the goal is to balance
                                        the pole by applying forces
     in the left and right direction on the cart.

    ### Action Space

    The action is a `ndarray` with shape `(1,)` which can take values `{0,
                                         1}` indicating the direction
     of the fixed force the cart is pushed with.

    | Num | Action                 |
    |-----|------------------------|
```

```
| 0   | Push cart to the left  |
| 1   | Push cart to the right |

**Note**: The velocity that is reduced or increased by the applied
                                    force is not fixed and it depends
                                     on the angle
 the pole is pointing. The center of gravity of the pole varies the
                                     amount of energy needed to move
                                     the cart underneath it

### Observation Space

The observation is a `ndarray` with shape `(4,)` with the values
                                    corresponding to the following
                                    positions and velocities:

| Num | Observation           | Min                   | Max
                                                      |
|-----|-----------------------|-----------------------|
                                                      -------------------|
| 0   | Cart Position         | -4.8                   | 4.8
                                                     |
| 1   | Cart Velocity         | -Inf                   | Inf
                                                     |
| 2   | Pole Angle            | ~ -0.418 rad (-24 ) | ~ 0.418 rad (
                                                      24 ) |
| 3   | Pole Angular Velocity | -Inf                   | Inf
                                                     |

**Note:** While the ranges above denote the possible values for
                                    observation space of each element
                                    ,
    it is not reflective of the allowed values of the state space in
                                        an unterminated episode.
                                        Particularly:
-  The cart x-position (index 0) can be take values between `(-4.8, 4.
                                    8)`, but the episode terminates
   if the cart leaves the `(-2.4, 2.4)` range.
-  The pole angle can be observed between  `(-.418, .418)` radians (or
                                     **  24  **), but the episode
                                    terminates
   if the pole angle is not in the range `(-.2095, .2095)` (or **
                                        12  **)

### Rewards

Since the goal is to keep the pole upright for as long as possible, a
                                    reward of `+1` for every step
                                    taken,
including the termination step, is allotted. The threshold for rewards
                                     is 475 for v1.

### Starting State
```

```
    All observations are assigned a uniformly random value in `(-0.05, 0.
                                        05)`

    ### Episode End

    The episode ends if any one of the following occurs:

    1. Termination: Pole Angle is greater than   12
    2. Termination: Cart Position is greater than  2 .4 (center of the
                                        cart reaches the edge of the
                                        display)
    3. Truncation: Episode length is greater than 500 (200 for v0)

    ### Arguments

    ```
    gym.make('CartPole-v1')
    ```

    No additional arguments are currently supported.
    """

    metadata = {
        "render_modes": ["human", "rgb_array"],
        "render_fps": 50,
    }

    def __init__(self, render_mode: Optional[str] = None):
        self.gravity = 9.8
        self.masscart = 1.0
        self.masspole = 0.1
        self.total_mass = self.masspole + self.masscart
        self.length = 0.5 # actually half the pole's length
        self.polemass_length = self.masspole * self.length
        self.force_mag = 10.0
        self.tau = 0.02 # seconds between state updates
        self.kinematics_integrator = "euler"

        # Angle at which to fail the episode
        self.theta_threshold_radians = 12 * 2 * math.pi / 360
        self.x_threshold = 2.4

        # Angle limit set to 2 * theta_threshold_radians so failing
                                            observation
        # is still within bounds.
        high = np.array(
            [
                self.x_threshold * 2,
                np.finfo(np.float32).max,
                self.theta_threshold_radians * 2,
                np.finfo(np.float32).max,
            ],
            dtype=np.float32,
)
```

```
        self.action_space = spaces.Discrete(2)
        self.observation_space = spaces.Box(-high, high, dtype=np.float32)

        self.render_mode = render_mode

        self.screen_width = 600
        self.screen_height = 400
        self.screen = None
        self.clock = None
        self.isopen = True
        self.state = None

        self.steps_beyond_terminated = None

    def step(self, action):
        err_msg = f"{action!r} ({type(action)}) invalid"
        assert self.action_space.contains(action), err_msg
        assert self.state is not None, "Call reset before using step
                                            method."
        x, x_dot, theta, theta_dot = self.state
        data = [x, x_dot, theta, theta_dot]
        with open(file_path, 'a', newline='') as csvfile:
            writer = csv.writer(csvfile)
            writer.writerow(data)  # Write a single row of state data

        force = self.force_mag if action == 1 else -self.force_mag
        costheta = math.cos(theta)
        sintheta = math.sin(theta)

        # For the interested reader:
        # https://coneural.org/florian/papers/05_cart_pole.pdf
        temp = (
            force + self.polemass_length * theta_dot**2 * sintheta
        ) / self.total_mass
        thetaacc = (self.gravity * sintheta - costheta * temp) / (
            self.length * (4.0 / 3.0 - self.masspole * costheta**2 / self.
                                                total_mass)
        )
        xacc = temp - self.polemass_length * thetaacc * costheta / self.
                                            total_mass

        if self.kinematics_integrator == "euler":
            x = x + self.tau * x_dot
            x_dot = x_dot + self.tau * xacc
            theta = theta + self.tau * theta_dot
            theta_dot = theta_dot + self.tau * thetaacc
        else:  # semi-implicit euler
            x_dot = x_dot + self.tau * xacc
            x = x + self.tau * x_dot
            theta_dot = theta_dot + self.tau * thetaacc
            theta = theta + self.tau * theta_dot

        self.state = (x, x_dot, theta, theta_dot)
```

```
        terminated = bool(
            x < -self.x_threshold
            or x > self.x_threshold
            or theta < -self.theta_threshold_radians
            or theta > self.theta_threshold_radians
        )

        if not terminated:
            reward = 1.0
        elif self.steps_beyond_terminated is None:
            # Pole just fell!
            self.steps_beyond_terminated = 0
            reward = 1.0
        else:
            if self.steps_beyond_terminated == 0:
                logger.warn(
                    "You are calling 'step()' even though this "
                    "environment has already returned terminated = True.
                                                            You "
                    "should always call 'reset()' once you receive '
                                                            terminated = "
                    "True' -- any further steps are undefined behavior."
                )
            self.steps_beyond_terminated += 1
            reward = 0.0

        if self.render_mode == "human":
            self.render()
        return np.array(self.state, dtype=np.float32), reward, terminated,
                                                    False, {}

    def reset(
        self,
        *,
        seed: Optional[int] = None,
        options: Optional[dict] = None,
    ):
        super().reset(seed=seed)
        # Note that if you use custom reset bounds, it may lead to out-of-
                                                    bound
        # state/observations.
        low, high = utils.maybe_parse_reset_bounds(
            options, -0.05, 0.05  # default low
        )  # default high
        self.state = self.np_random.uniform(low=low, high=high, size=(4,))
        self.steps_beyond_terminated = None

        if self.render_mode == "human":
            self.render()
        return np.array(self.state, dtype=np.float32), {}

    def render(self):
        if self.render_mode is None:
```

```
        gym.logger.warn(
            "You are calling render method without specifying any
                                                render mode. "
            "You can specify the render_mode at initialization, "
            f'e.g. gym("{self.spec.id}", render_mode="rgb_array")'
        )
        return

    try:
        import pygame
        from pygame import gfxdraw
    except ImportError:
        raise DependencyNotInstalled(
            "pygame is not installed, run `pip install gym[
                                                classic_control]`"
        )

    if self.screen is None:
        pygame.init()
        if self.render_mode == "human":
            pygame.display.init()
            self.screen = pygame.display.set_mode(
                (self.screen_width, self.screen_height)
            )
        else:  # mode == "rgb_array"
            self.screen = pygame.Surface((self.screen_width, self.
                                                screen_height))
    if self.clock is None:
        self.clock = pygame.time.Clock()

    world_width = self.x_threshold * 2
    scale = self.screen_width / world_width
    polewidth = 10.0
    polelen = scale * (2 * self.length)
    cartwidth = 50.0
    cartheight = 30.0

    if self.state is None:
        return None

    x = self.state

    self.surf = pygame.Surface((self.screen_width, self.screen_height)
                                            )
    self.surf.fill((255, 255, 255))

    l, r, t, b = -cartwidth / 2, cartwidth / 2, cartheight / 2, -
                                        cartheight / 2
    axleoffset = cartheight / 4.0
    cartx = x[0] * scale + self.screen_width / 2.0  # MIDDLE OF CART
    carty = 100  # TOP OF CART
    cart_coords = [(l, b), (l, t), (r, t), (r, b)]
    cart_coords = [(c[0] + cartx, c[1] + carty) for c in cart_coords]
    gfxdraw.aapolygon(self.surf, cart_coords, (0, 0, 0))
```

```
        gfxdraw.filled_polygon(self.surf, cart_coords, (0, 0, 0))

        l, r, t, b = (
            -polewidth / 2,
            polewidth / 2,
            polelen - polewidth / 2,
            -polewidth / 2,
        )

        pole_coords = []
        for coord in [(l, b), (l, t), (r, t), (r, b)]:
            coord = pygame.math.Vector2(coord).rotate_rad(-x[2])
            coord = (coord[0] + cartx, coord[1] + carty + axleoffset)
            pole_coords.append(coord)
        gfxdraw.aapolygon(self.surf, pole_coords, (202, 152, 101))
        gfxdraw.filled_polygon(self.surf, pole_coords, (202, 152, 101))

        gfxdraw.aacircle(
            self.surf,
            int(cartx),
            int(carty + axleoffset),
            int(polewidth / 2),
            (129, 132, 203),
        )
        gfxdraw.filled_circle(
            self.surf,
            int(cartx),
            int(carty + axleoffset),
            int(polewidth / 2),
            (129, 132, 203),
        )

        gfxdraw.hline(self.surf, 0, self.screen_width, carty, (0, 0, 0))

        self.surf = pygame.transform.flip(self.surf, False, True)
        self.screen.blit(self.surf, (0, 0))
        if self.render_mode == "human":
            pygame.event.pump()
            self.clock.tick(self.metadata["render_fps"])
            pygame.display.flip()

        elif self.render_mode == "rgb_array":
            return np.transpose(
                np.array(pygame.surfarray.pixels3d(self.screen)), axes=(1,
                                                                        0, 2)
            )

    def close(self):
        if self.screen is not None:
            import pygame

            pygame.display.quit()
            pygame.quit()
            self.isopen = False
```

Appendix I: Python Script for Data Post-Processing

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

t = np.arange(0,120,0.02)
df = pd.read_csv('cart_pole_test_20241107_165335_revised.csv')
df['theta'] = np.degrees(df['theta'])
df['thetadot'] = np.degrees(df['thetadot'])
df2 = df.assign(time = t)
df3 = df2.iloc[0:500]

p1 = plt.figure(1)
ax = plt.gca()
df3.plot(kind='line', x='time',y='x',color='red',ax=ax)
plt.xlabel("Time [sec]")
plt.ylabel("x [m]")
plt.xlim(0,10)
plt.grid()
plt.title('Cart Displacement vs. Time')
p1.show()

p2 = plt.figure(2)
ax = plt.gca()
df3.plot(kind='line', x='time',y='xdot',color='green',ax=ax)
plt.xlabel("Time [sec]")
plt.ylabel("xdot [m/s]")
plt.xlim(0,10)
plt.grid()
plt.title('Cart Velocity vs. Time')
p2.show()

p3 = plt.figure(3)
ax = plt.gca()
df3.plot(kind='line', x='time',y='theta',color='blue',ax=ax)
plt.xlabel("Time [sec]")
plt.ylabel("Theta [deg]")
plt.xlim(0,10)
plt.grid()
plt.title('Pole Angle vs. Time')
p3.show()

p4 = plt.figure(4)
ax = plt.gca()
df3.plot(kind='line', x='time',y='thetadot',color='magenta',ax=ax)
plt.xlabel("Time [sec]")
plt.ylabel("Thetadot [deg/s]")
plt.xlim(0,10)
plt.grid()
plt.title('Pole Rate vs. Time')
```

```
p4.show()

input()
```

Appendix J: Python Script for Reward Functions Modification

```
from __future__ import annotations

from typing import Callable, SupportsFloat

import numpy as np

import os

from gymnasium import logger
import gymnasium as gym
from stable_baselines3 import A2C, PPO
from gymnasium.core import ActType, ObsType
from stable_baselines3.common.callbacks import BaseCallback
from stable_baselines3.common.logger import HParam
from stable_baselines3.common.env_checker import check_env
from stable_baselines3.common.monitor import Monitor

__all__ = ["TransformReward"]

# Modify the Reward Function
class TransformReward(gym.RewardWrapper[ObsType, ActType], gym.utils.
                                        RecordConstructorArgs):
    def __init__(
        self,
        env: gym.Env[ObsType, ActType],
        func: Callable[[SupportsFloat], SupportsFloat],
    ):
        gym.utils.RecordConstructorArgs.__init__(self, func=func)
        gym.RewardWrapper.__init__(self, env)

        self.func = func
        self.steps_beyond_terminated = None

    def reward(self, reward: SupportsFloat) -> SupportsFloat:
        terminated = False
        # epsilon = 1e-6 # Small value to avoid division by zero
        set_point = 0 #target value
        # n = 3
        x, x_dot, theta, theta_dot = self.state #getting current state
        if not terminated:

            # reward = -np.abs(theta - set_point) # attempt 6
             if theta <= 1e-3 and theta >= -1e-3: # attempt 5
                reward = 1/abs(set_point-theta)
            #     reward = (1e-3 - abs(theta))**n/(1e-3)**n # attempt 3
            #reward = 0.001*(x**2) + 0.1*(theta**2) # attempt 4
            # reward = 1/(abs(set_point - theta) + 1) # attempt 2
        elif self.steps_beyond_terminated is None:
```

```
            self.steps_beyond_terminated = 0 # Pole just fell
            reward = 0.0 #sets reward equal to zero, pole fell, we are
                                                entering terminal state
            #reward = 1/ ((0 - reward) + 1 + epsilon) # attempt 1
        else:

            if self.steps_beyond_terminated == 0:

                logger.warn(
                    "You are calling 'step()' even though this "
                    "environment has already returned terminated = True.
                                                            You "
                    "should always call 'reset()' once you receive '
                                                            terminated = "
                    "True' -- any further steps are undefined behavior."
                )
            self.steps_beyond_terminated += 1
            reward = 0.0 #simple case to learn, similar to terminal state
                                                of pole falling
            #reward = -1.0 #previous trial
        return self.func(reward)

def custom_reward_transform(reward: SupportsFloat) -> SupportsFloat:
    return reward

# Create Log Directory
models_dir = "models/PPO"
logdir = "logs"

os.makedirs(models_dir, exist_ok=True) # If models directory doesn't exist
                                    , create one
os.makedirs(logdir, exist_ok=True) # If the log doesn't exist, create one

# Train the RL agent
env = gym.make("CustomCartPole")
env = TransformReward(env, custom_reward_transform)
class HParamCallback(BaseCallback):
    """
    Saves the hyperparameters and metrics at the start of the training,
                                        and logs them to TensorBoard.
    """

    def _on_training_start(self) -> None:
        hparam_dict = {
            "algorithm": self.model.__class__.__name__,
            "learning rate": self.model.learning_rate,
            "gamma": self.model.gamma,
        }
        # define the metrics that will appear in the `HPARAMS` Tensorboard
                                                tab by referencing their tag
        # Tensorbaord will find & display metrics from the `SCALARS` tab
        metric_dict = {
            "rollout/ep_len_mean": 0,
            "train/value_loss": 0.0,
```

```
        }
        self.logger.record(
            "hparams",
            HParam(hparam_dict, metric_dict),
            exclude=("stdout", "log", "json", "csv"),
        )

    def _on_step(self) -> bool:
        return True
check_env(env)
model = PPO("MlpPolicy", env, verbose=1, tensorboard_log=logdir)
env = Monitor(env)

TIMESTEPS = 10000
for i in range(1,11):
    model.learn(total_timesteps=TIMESTEPS, tb_log_name='PPO',
                                           reset_num_timesteps=False,
                                           callback=HParamCallback())
    model.save(f"{models_dir}/{TIMESTEPS*i}")
```