Towards Green Steel: A Dynamic Programming Approach to Slag-Basicity Control in HIsarna

Master Thesis Report

R. Agarwal

Technische Universiteit Delft

TU Delft

Towards Green Steel: A Dynamic Programming Approach to Slag-Basicity Control in Hlsarna

Master Thesis Report

by

R. Agarwal

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended on Wednesday August 30, 2023 at 10:00 AM.

Student number:	5546419	
Thesis committee:	Prof. P. M. Esfahani ,	TU Delft, supervisor
	Prof. A. Dabiri ,	TU Delft
	Ir. E. Feenstra,	Tata Steel Europe
	Ir. G. Max,	TU Delft

This thesis is confidential and cannot be made public until August 30, 2025.

TU Delft

Preface

The HIsarna iron-making process presents a significant step forward in steel production, chiefly for its potential to substantially reduce CO_2 emissions, paving the way for greener steel production. However, for HIsarna to be robustly adopted at an industrial scale, optimising various facets is essential, with slag-basicity control emerging as a focal point. Current control methods, anchored in affine calculations, have not yielded the desired efficiency in maintaining slag-basicity near a target value.

This thesis introduces a novel approach to this challenge. Drawing upon the high-fidelity HIsarna process simulator, we formulate an MDP model. In conjunction with the Bellman optimality principle, we use this model to design an optimal control strategy as an optimal controller table. Simulations substantiate the effectiveness of our proposed control methodology compared to existing practices.

Beyond the core strategy, this work explores alternate design parameters and control methods. The aim is to balance superior reference tracking and computational efficiency. These investigative undertakings either extend or abstract the primary control strategy and provide insights to refine and justify our design decisions.

Given the safety-critical nature of the HIsarna process, transitioning directly to fully automated control is cautiously approached. In light of this, we propose the incorporation of decision trees to render our advanced control strategy more accessible to human operators. Positioned between the prevailing suboptimal controls and a fully automated optimal control system, these decision trees, backed by simulations, offer a promising and implementable solution. Their potential role in facilitating consistent HIsarna operations is underscored, marking a step towards greener steel production.

R. Agarwal
Delft, August 2023

Contents

Acknowledgement

I would like to express my sincere gratitude to my supervisors. Firstly, to Prof. P. M. Esfahani for introducing me to this captivating thesis topic and for his invaluable and timely feedback throughout my work. My appreciation also extends to Ir. E. Feenstra. His expertise was pivotal in helping me navigate and comprehend the complexities of the HIsarna process. His insights and guidance have been indispensable in shaping the objectives and outcomes of this project. Lastly, I want to appreciate the time and dedication shown by my daily supervisor Ir. G. Max, who helped me navigate this work and come up with exciting ideas for potential solutions and improvements. Their combined support and mentorship have been instrumental in the successful completion of this work.

I would also like to thank my family, who have been incredibly supportive throughout my master's. I would also like to thank all my friends who have been a part of this journey and a source of inspiration, support and encouragement.

I would also like to thank Tata Steel Europe B.V. for supporting and providing resources for this thesis work. This research is part of the project Digital Twin with project number P18-03 of the research programme TTW Perspective which is (partly) financed by the Dutch Research Council (NWO).

1 Introduction

There's a need for immediate changes in the steel industry, making it sustainable and reducing the environmental burden. Steel manufacturing-related greenhouse gas emissions contribute 7-9% of all human-made greenhouse gas emissions. In 2020, 1.86 billion metric tons of steel were produced worldwide, producing a staggering 3 billion tons of CO_2. By 2050, the global steel demand is expected to rise to 2.5 billion tons per year. This will also lead to an increase in emissions if the current steel manufacturing process is continued in the future. The current manufacturing process is highly dependent on heavy infrastructures, such as blast furnaces which require massive investment to be built.

It is estimated that the iron production step, a part of the steel manufacturing process, contributes to 80% of the total CO_2 emissions during the entire steel-making process. The Blast Furnace is the dominating technology for producing iron from iron ore; 60% of iron production happens through blast furnaces. This process is expensive both economically and environmentally. Pre-processing steps such as preheating air consumes much energy, and pre-heating coke is also a very energy-intensive process requiring the crushing of coal and heating it to 1100°C. One more drawback of this process is related to the quality of ore. If the iron ore has low iron content, it requires energy-intensive pre-processing. Therefore there is a need to develop new innovative iron-making procedures that are economical and environment friendly.

The HIsarna iron-making process is a new and innovative method of producing iron that aims to be more efficient and environmentally friendly than traditional methods. Developed by Tata Steel, this process involves two main steps: iron ore is reduced to iron in a cyclone reactor, and then the iron is used to produce steel in a separate vessel. This is a significant departure from traditional blast furnace steelmaking. This innovative process also allows skipping the energy intensive pre-processing steps and is also not affected by the quality of ore.

In this thesis, we help Tata Steel make the HIsarna process easier to control. We describe the HIsarna process and the problem and define the goals of this thesis in this chapter.

1.1. HIsarna: Smelting Process

HIsarna combines two processes – cyclone converter furnace (CCF) and the HIsmelt vessel. Fig. 1.1 depicts the HIsarna setup. CCF was developed by Hoogovens (now Tata Steel) in IJmuiden and was used for melting and partially reducing iron ores. The process involved the injection of pure oxygen to help reach the required melting temperatures and to enable the separation of fines from the gas through the centrifugal flow of gas. This process also involved the injection of ore and oxygen into the converter furnace in the presence of the hot smelter gas. CCF and HIsmelt were brought together to produce the HIsarna steel manufacturing process.

The HIsarna process involves two stages of contact between the iron and gas, and both are operated above melting temperatures. The process has been highly integrated and occurs in a single smelting furnace. CCF is a pre-reduction vessel to melt the iron-ore particles, and the final reduction occurs in

the HIsmelt vessel. To integrate these, the CCF was placed above the HIsmelt vessel. Crushed ore, oxygen and coal dust are injected into the CCF. The oxygen produces the required heat to melt the crushed ore. The melted ore then falls into the HIsmelt vessel containing the molten iron bath. At this point, powdered coal is injected into the smelting vessel, which helps reduce iron and helps produce pure liquid iron, which is then tapped and then can be transported to a separate container for steel production. On top of the molten iron produced, a molten layer of slag is created, containing all the impurities in the ore. The hot gases released during iron reduction releases provide the heat for the reactions to occur in the CCF. As a result of this highly integrated process, the requirement of coke, sinter and pellets was eliminated, essential requirements of the traditional blast furnace process. To produce each of coke, sinter and pellet different types of infrastructure are required and are all energy-intensive processes. Coke is produced at extremely high-temperatures in coking ovens, sintering is done in sintering plants and is an energy-intensive process as well. HIsarna eliminates the use of these processes giving both economical and environmental benifits.

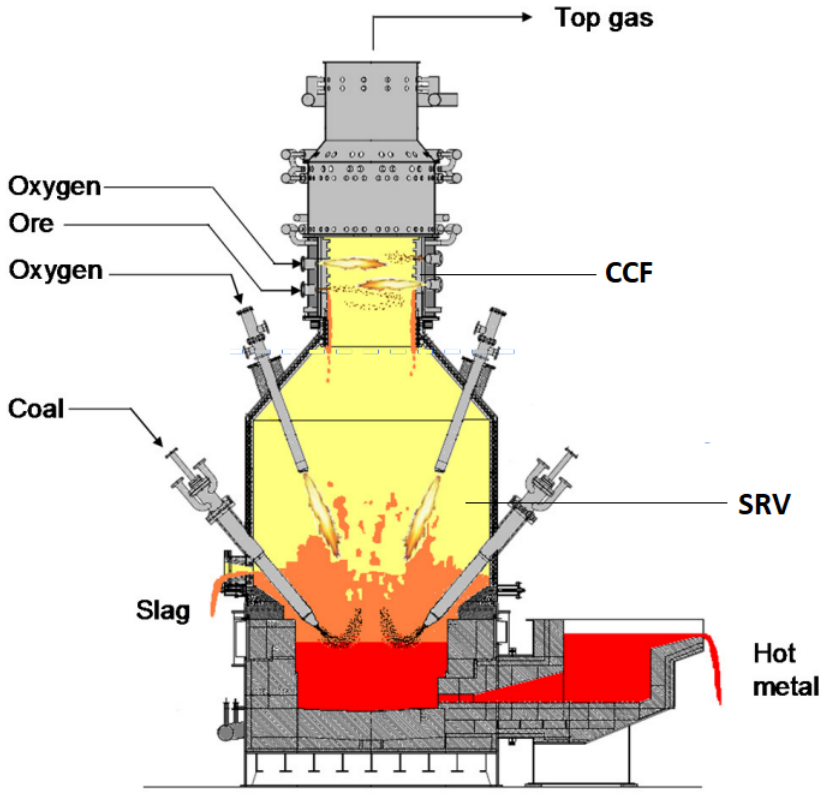

Figure 1.1: HIsarna Setup.

The process was 30% more energy efficient and 2% less carbon-intensive. The CO_2 produced is almost pure, therefore, suitable for capture. CO_2 capture, if integrated with the HIsarna process, can reduce between 80%- 100% emissions from the steel manufacturing process and substantially reduce sulfur and nitrogen dioxide emissions. Over the years, various experimental runs at the prototype plant have been carried out to prove the various theoretical claims about the HIsarna steel-making process. These experimentations proved that liquid iron could be produced without pre-processing the raw materials and that the entire process was safe. The concepts of carbon capture and storage were also tested during the first experimentation phase. After these experimentations, sustained production and production using different raw materials were tested, and the results were positive. The results showed that 53% of the materials used for iron production could be scrap steel. A different mix of raw materials was also tested to find the ideal combination.

1.2. Current Focus and Problems - Slag Basicity

Currently, the focus is designing an industrial-scale plant on the HIsarna steel-making process. For this, the process needs to be made commercially viable and to enable that process control needs to be improved. One area of particular importance in achieving the above-mentioned targets is the control of slag basicity. The slag serves a crucial function in the smelting process. Primarily composed of silicon dioxide and calcium oxide, it also contains various impurities and non-iron metal alloys, deriving from the coal and ore mixture. Acting as a heat transfer medium between the heating zone in the upper part of the SRV (Smelting Reduction Vessel) and the molten iron, the slag ensures optimal heat flow and enhanced iron and coal mixing. Coal is introduced at high pressure and angled into the slag, promoting thorough mixing with the pre-reduced iron. This process generates a slag fountain, where slag droplets circulate in the SRV's upper section, absorbing heat produced by coal combustion. Maintaining the slag's viscosity within a specific range is vital for an effective slag fountain. A reliable indicator of slag viscosity is its basicity, known as the B2 value, representing the mass ratio of the slag's constituent materials.

$$B2 = \frac{\text{CaO Mass}}{\text{SiO}_2\text{ Mass}} \tag{1.1}$$

Maintaining appropriate slag basicity in the HIsarna steelmaking process is essential for process efficiency and also for preventing potential damage to the equipment. A high basicity level can trigger slag foaming, wherein the slag mixture starts to ascend. This upward movement of the hot slag mixture could cause substantial physical harm to the HIsarna plant, necessitating significant repair costs.

Moreover, any incidence of slag foaming disrupts the operational continuity of the plant, requiring an immediate cessation of the ongoing run. The subsequent steps include cooling the furnace, cleaning it, and preparing it anew for restart. Thus, an episode of slag foaming is not only economically taxing but also leads to considerable time loss.

The furnace's extreme temperatures introduce a significant degree of measurement noise, further complicating the process. Moreover, the inherent variability in the composition of iron ore and coal introduces an element of unpredictability to the process, which is no longer strictly deterministic. Also, the measurement of the slag basicity is only available at slag-tap events, which makes the measurement sparse. Many factors contribute to this system's randomness, making choosing an optimal lime input rate a complex issue. This process becomes an exercise in sequential decision-making amidst uncertainty.

1.3. Current Control for Slag Basicity

Avoiding slag foaming is of paramount importance, underscoring the need to monitor and manage slag basicity meticulously during the HIsarna process. Ideally, it is wanted that the B2 value remains at a desired constant throughout the process. This B2 value can be controlled by changing the rate of lime (CaO) being put into the smelting vessel. Due to the variance in the composition of ore and coal, it is difficult to maintain this B2 value at the desired level.

Presently, the control of slag basicity in the HIsarna steelmaking process falls under the supervision of a human operator. This operator, engaged in constant surveillance of various parameters, is responsible for adjusting the lime input rate. Before each operational run of the plant, an analysis of the composition of iron ore and coal to be utilised is conducted. The outcome of this analysis informs the establishment of a simple control rule for the lime-input rate, based upon the ore-input rate. However, the efficacy of this control rule is poor due to the severity of the conditions within the furnace, making the control of slag basicity based on an affine rule challenging.

Therefore, the current method of manually controlling slag basicity warrants reconsideration. There's a need to develop a better control strategy for maintaining the slag basicity, considering more factors than just the ore rate.

1.4. Conclusion

In the HIsarna steel-making process, managing slag basicity is crucial for achieving reliable and continuous steel production. The control of the lime-input rate is a key factor in maintaining slag basicity,

yet simple affine rules have proven insufficient for maintaining consistent basicity levels. A more sophisticated control mechanism, which considers the numerous factors affecting slag composition and the inherent randomness of the system, is necessary.

This project builds on the prior work of a former Master's student, G. Vitanov. Vitanov successfully developed an optimal control strategy based on a data-driven model he created. The outcomes generated by this controller proved superior based on the simulations carried out on the developed process model, to those of the manually operated process.

However, during Vitanov's tenure on the project, the high-fidelity process model was still under development at Tata Steel. Consequently, he had to construct his model utilising data from actual plant operations. Since then, Tata Steel has developed a HIsarna process model which can serve as a simulator for the actual process, thus providing a valuable tool for further research. This thesis is aimed at improving the work done by Vitanov, building on the knowledge gained from his work. **This thesis aims to develop an optimal control strategy for slag-basicity based on the high-fidelity simulator developed by Tata Steel engineers.**

In the next chapter, we discuss relevant literature in this area and also the work done by Vitanov and set the goals for this thesis.

2

Literature Survey

Slag is a topic of much discussion and research, especially in the steel-making and civil industry. Slag as we discussed above is a by-product of the iron-making process and is not directly useful for the iron-producers. However, slag finds its use in the cement industry and is used as a raw material for cement production. Since HIsarna is a novel technique being developed, there's no prior research done on controlling this process's slag, but there are some works available which talk about the control of slag produced from blast furnaces.

In this chapter, we first look at more general works about the control of slag composition, its impact and its importance. Then we look at the use of dynamic programming algorithms in controlling such uncertain environments. Finally, we discuss the previously proposed solution by a Master's student working on the same problem.

2.1. Slag Composition: Prediction & Control

Controlling of slag-composition is beneficial not only to maintain the operational safety of the plant but also to maintain the quality of the steel being produced from it. [1] discusses the importance of maintaining slag-basicity in terms of making clean-pure steel. Among other things, the authors talk about controlling the slag basicity and its role in presenting the re-oxidation of molten steel and propose a new deoxidation method.

There has been a lot of interest in utilising the by-products of traditional iron-making procedure which uses blast-furnace. Blast furnace-produced slag is also used as an addition to Portland cement. This reduces the need for clinker addition, which in itself is a carbon-intensive and expensive process. To facilitate the use of blast-furnace slag in Portland cement manufacture, it is important to maintain slag-basicity. [2] proposes an Artificial Neural Network model to predict the composition of slag basicity based on the input parameters of the blast furnace and the output parameters being the slag composition. The proposed approach is accurate for predicting the slag composition and offers the potential to develop a controller based on this prediction model.

A similar prediction model has been developed for slag-viscosity in [3]. In this work, the role of slag properties in optimising blast furnace reactions is highlighted. They propose incorporating the phase-equilibrium equations in the prediction of slag viscosity and based on historical data demonstrate the effectiveness of this model.

After the production of pig iron from the blast furnace, the molten iron still contains impurities in the form of carbon. To purify iron further basic oxygen furnaces are used. [4] discusses the factors, online monitoring and prediction methods to control slag foaming and slopping in the basic oxygen furnace. They conclude that Relative Stability potential diagrams are a good tool to predict the uncertainties inside a basic oxygen furnace.

Steel is used for diverse applications. One of the areas where steel cords find their use is in the manufacture of radial ply tires. [5] discusses the role of maintaining the slag-basicity at a low-constant

value to achieve good plasticity in the steel cords. Through laboratory tests, it is proven that maintaining this low basicity helps the steel cords achieve plastic deformation.

The above works demonstrate a relevant interest in academia in the prediction and control of slag compositions and their potential advantages. While most of the discussed results demonstrate possible effective models to predict the slag composition and the advantages of controlling the slag composition, a clear idea about the methodology to control the basicity is not proposed.

2.2. Dynamic Programming in Steel Manufacturing

From the previously done work on this problem in [6], we saw the potential of using Dynamic Programming to develop a control law. Since there are so many factors which add uncertainty to the HIsarna process, developing a stochastic controller is a good choice. Dynamic Programming (DP) has been used extensively to develop controllers for such stochastic environments. In this section, we see some examples of the use of DP in the steel industry.

DP and stochastic control principles are widely researched in the steel industry for various applications. These concepts have been researched for many years, and due to their potential of optimising the manufacturing process find great application in the industry.

DP has proven to be a valuable tool for allocating by-product gas systems used in steel manufacturing [7] & [8]. By-product gas is produced mainly by blast furnaces and is consumed by other processes such as sintering, pelleting, steel making and rolling. The demand and supply of the by-product gas are dynamic and varies based on equipment, time and other factors. DP can help in the dynamic allocation of this gas and maximise energy efficiency. Adaptive dynamic programming (ADP) has also been used to propose a solution to this problem [9].

Another application area where DP has found its potential use is the logistics of the heavy slabs used in steel manufacturing. This problem arises between the continuous casting stage and the hot rolling mill in the slab-yard storage stage. The problem is about minimising the efforts of the cranes in moving these slabs around the yard and reaching them to the desired location. [10] develops an integer programming model for the problem and uses DP for solving the problem on a small scale and uses a segmented dynamic programming approach for the full-scale problem. The approach is found to be effective in reducing the workload of cranes. Similar work in [11] concerning finished heavy steel plate logistics has been done.

Energy and resource allocation are also areas where optimisation is necessary for the steel industry. [12] proposes using approximate dynamic programming (ADP) to allocate energy to various production lines based on an online time-varying environment. This method is found to be more efficient when compared to the static allocation methods. Similar work is done in [13] concerning allocating molten iron to different steel production facilities. This problem is again related to the efficiency of the steel production process, and the goal is to avoid reheating the molten iron. The work uses an integer programming model and a dynamic programming algorithm for the optimal solution.

The above discussion shows that DP finds applications in various problems within the steel industry. Mostly DP has been used to optimise the logistics inside the factories or for optimal allocation of products, energy or resources.

2.3. Previous work on HIsarna Slag Basicity control

Previously this problem was worked on by a former Master's student, George Vitanov [6]. Vitanov developed an optimal control strategy for the data based model constructed by him. He used value iteration (VI) to solve the dynamic programming problem. This approach has found applications in many fields such as economics [14], management [15], inventory mangement [16], scheduling [17], load shifting [18] etc. This section discusses his work and identifies areas that can be improved upon.

2.3.1. Modelling the System

Vitanov developed his model based on the mass balance in the furnace and using the recorded data made available by Tata Steel. A discrete-time mass-balanced model was used to develop the parametric system model. This was best suited to the problem due to the discrete availibility of measurement

data. Slag mass, SiO_2 mass, and CaO mass were chosen to be the state variables, and the inputs were coal, ore and lime mass put into the system between slag tap intervals. The A matrix was chosen to be an identity matrix, assuming there will be no change in the system's state if no inputs. Another assumption is made about the event of a slag tap, where it is assumed that in the case of a slag tap, the slag extracted will be homogeneous and hence the composition or basicity will not be affected by a slag tap. The above assumptions and first principles lead to the following state equation:

$$x(k+1) = Ax(k) + Bu(k) - D(x(k))s_{tap}(k) \tag{2.1}$$

The B matrix parameters were derived from existing plant-run data. Parameter estimation for the B matrix was carried out using a least-squares technique. Model evaluation results based on the available data was done and it showed good results. further analysis of the error in SiO_2 and CaO states was carried out. This analysis provided an idea about the model errors.

.

2.3.2. Dynamic Programming and Controller Synthesis

The problem is an infinite horizon decision making problem. Dynamic Programming was used to solve this due to its proven effectiveness in solving such problems. The developed model is a Markov Decision Process (MDP) model, so Dynamic Programming (DP) could be applied to this model. Due to the large number of computaions involved, parallel computaion was used to reduce the convergence time of the DP algorithm.

After defining all the design parameters including cost-function, discretisation of states and action, and the interpolation technique the solution to the DP was computed and stored as a look-up-table, mapping the states to the optimal actions.

2.3.3. Controller Evaluation and Results

Evaluation of the developed controller was carried out using simulations on the developed process model. The controller's performance was compared to the human operator's performance using the operator input data present in the database. Input lime rates and the evolution of B2 in the system were plotted and compared.

Similar trends for lime-inputs are noticed, suggesting that the developed controller's suggested inputs are reasonable. It was concluded that although the controller outperforms the human-operator, the performance of the controller has a significant amount of uncertainity. This conclusion was derived based on quantative analysis of the controller's simulation using the running cost-function.

2.3.4. Conclusion

This works laid a solid foundation for the control of plant basicity, providing a compelling comparison between the performance of an operator and an automated controller. Using a running cost function to evaluate the two, the simulations indicate that the controller outperforms the operator. Despite this, it is crucial to acknowledge that in the worst-case scenarios, the controller's performance aligns closely with that of the operator, thus emphasising the need to consider the variability in performance outcomes due to factors such as model errors.

2.3.5. Controller Performance on HIOM

Post Vitanov's work, Tata-Steel engineers developed a more sophisticated plant model, the HIOM, based on mass and thermodynamic balance equations. This model includes a greater number of variables compared to the one developed by Vitanov. Upon applying Vitanov's controller to the HIOM model, Fig.2.1, the controller maintained plant basicity around the target value during initial events, but its performance deteriorated over time. This decline can be attributed to a combination of the controller's simpler model origin and the selected range of the state space values, which did not match the actual range of values involved in the process.

The work discussed herein offers a valuable launchpad for further research. We plan to focus on the development of a new controller based on the HIOM model. This high-fidelity model presents a unique opportunity to develop an optimal controller.

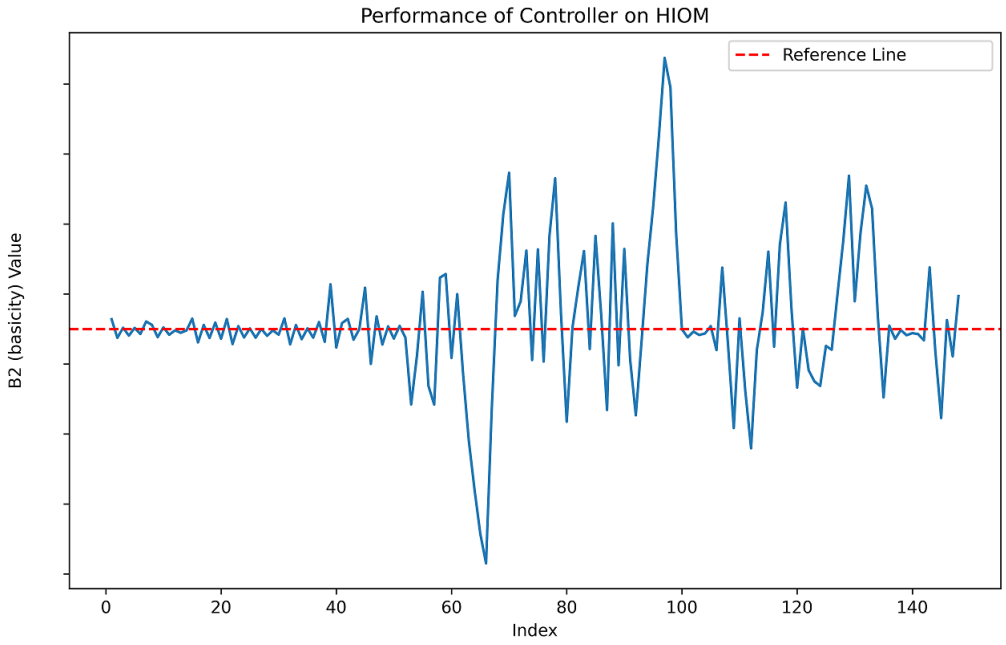

Figure 2.1: Previously Developed Controller run on HIOM

Our objective is to build on the knowledge gained from Vitanov's work and the subsequent application of the controller to the HIOM model to create a controller that delivers better performance. Through this, we also aim to close the loop between the actual plant and the developed controller and hence develop a roadmap for the implementation of the developed controller on the actual plant. This will lead to automating the lime-input and reduce the efforts of the human operator and also contribute to reliable and continous iron production from this process.

3

Modelling the System

Building on the discussions from the previous chapter, it is evident that dynamic programming (DP) and value iteration (VI) have shown significant potential in addressing this problem. The controller's performance was not as expected primarily due to modelling errors and model simplicity. Therefore, the focus is on using the high-fidelity process model, HIOM, to accommodate Dynamic Programming. The goal is to morph the system into a Markov Decision Process (MDP), a mathematical construct for decision-making where outcomes are partially random and partially under the decision-makers control [19]. A continuous-time Markov Decision Process is represented as a 4-tuple (S, A, P, R), where:

- $S \subset \mathbb{R}^n$ is a set of states
- $A \subset \mathbb{R}^m$ is a set of actions
- $P : S \times A \times S \to \mathbb{R}$ is the state transition density function.
- $R : S \times A \times S \to \mathbb{R}$ is the reward function.

This chapter defines each (S, A, P, R). We start by first understanding the process model provided to us by Tata Steel. HIOM is a process simulator which simulates the HIsarna iron-making process. It functions as a rate-based system, meaning all inputs and outputs are considered in terms of rates. Specifically, the inputs to HIOM comprise the rates and compositions of materials fed into the furnace, including iron ore, coal, and lime. Leveraging thermodynamic and mass-balance equations, HIOM predicts the output. For our purposes, the primary output of interest is the rate and composition of the slag produced.

Figure 3.1 presents a flowchart of our modelling process. The blocks highlighted in grey represent components provided to us, which are beyond our control. This chapter delves into a detailed description of each block

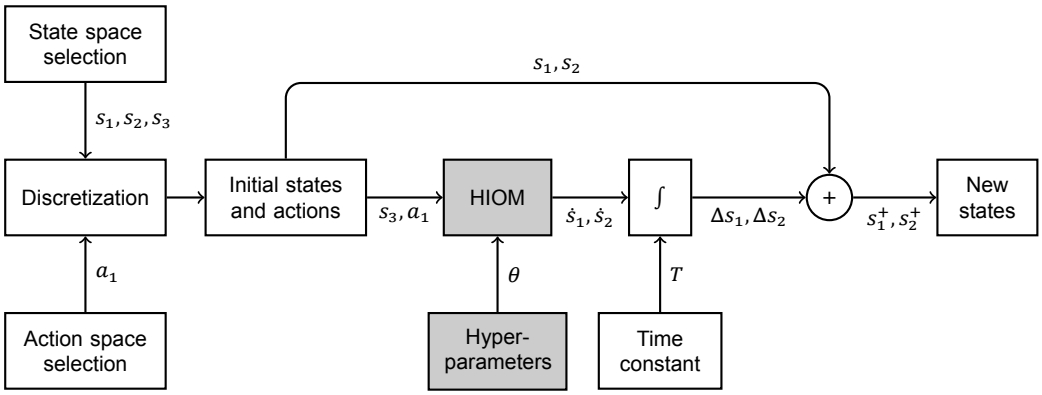

Figure 3.1: Modelling Flowchart

3.1. Relevant Dynamics

In our prior discussions, we established the goal of maintaining a constant slag-basicity value. Basicity is defined by the ratio of masses between two key slag constituents: CaO and SiO_2. Maintaining a constant slag-basicity in real time requires monitoring the masses within the furnace. However, this is a challenging endeavour. We can only estimate these masses in actual scenarios since measurements are taken during slag taps, which occur at irregular intervals to prevent overflow. Conversely, molten iron is tapped continuously. It's worth noting that the process model doesn't inherently track the masses inside the furnace.

To address this challenge, we aim to develop a virtual sensor based on the process model. This sensor is designed to capture the state of our target variables. The output from the process model is given in rates (kg/hr). A time constant is essential to transform this into absolute masses (kg). By utilizing this time constant, we can construct our MDP model based on these masses.

The HIsarna process is complex, characterised by the interaction of numerous materials at high temperatures. Iron ore, coal, and lime input rates don't solely determine its output. Factors like temperature, gas flow rate, and furnace design also play significant roles. These variables are encapsulated within the HIOM model and are provided as inputs defined by experts at Tata Steel. In Fig. 3.1, we refer to these as hyperparameters, denoted as θ.

Using the above approach, we can accurately capture the dynamics of the HIsarna process pertinent to our control objectives. As this chapter progresses, we will delve deeper into state and action space selection, discretisation, and time constant selection.

3.2. State and Action Space

The problem at hand is a continuous space problem. The states concerning our control target are ore-rate, coal-rate, SiO_2 mass and CaO mass. Ore rate and coal rates are, in reality, inputs to the system. These rates define the most important output of the process, the hot metal production rate. In the actual scenario, the required hot metal production rate defines the input rate of iron ore and coal. Hence, changing these inputs to control basicity is not ideal, so we define them as states rather than inputs.

An analysis of the previous run's recorded data showed a linear relationship between iron ore and coal. A discussion with engineers at Tata Steel also confirmed this observation. Based on the composition of iron-ore and coal being used for iron production, a ratio between iron-ore and coal is precomputed and maintained for those compositions. This fact helps us substitute the coal rate with a linear relationship with iron ore, reducing dimensionality and immensely boosting the computation time, as discussed in Chap.4.

As discussed previously, the other two states, SiO_2 mass and CaO mass, define our control target. From HIOM, we get the rate (kg/hr) of the produced SiO_2 and CaO under given rates and compositions of ore, coal and lime referred as hyperparameters or θ in Fig 3.1. To convert this into masses, we integrate these rates over time and add them with the initial masses inside the furnace. This gives us the expected next state if we take a certain action at an initial state. Eq. 3.1 depicts this

$$P(s_{k+1}|s_k, a_k) \tag{3.1}$$

From the above discussion, we can define our state and action spaces as:

- $S = \{SiO_2 \text{ Mass}, \text{CaO Mass}, \text{Ore-rate}\} = \{s_1, s_2, s_3\}$
- $A = \{\text{Lime-rate}\} = \{a_1\}$

3.3. Discretization

We make use of the previous plant run data for modelling our system. As defined above, we are interested in four variables: ore rate, lime rate, SiO_2 mass and CaO mass. We need to discretise our state and action space so that the solving DP is computationally possible.

Our states and action spaces are bounded because of the physical limitations of the plant. The smelting vessel has a limited capacity, so the SiO_2 and CaO mass must be bounded. Similarly, there's a limit on the amount of lime and ore that can be injected into the system so these quantities are also bounded.

We look at the input rates and masses of the concerned variables, in the database and define the range of the grid for these variables. We also discussed with experts at Tata Steel if the ranges selected were reasonable. Based on these discussions and the available data the selected range and discretization of these ranges are defined in table 3.1. To protect the intellectual property of Tata Steel we hide the actual numbers for the range of data.

State/ Action	Range	Discrete Points
Ore Rate	??kg/hr – ?? kg/hr	10
SiO_2 Mass	??kg – ??kg	15
CaO Mass	??kg – ??kg	25
Lime Rate	??kg/hr – ??kg/hr	20

Table 3.1: Discretised State and Action Space

The ore rate decides the hot metal output of the process, and the minimum is defined based on that. Based on this ore's composition and the coal's composition, a ratio is decided, which we use as a design parameter in our model. The mass range for SiO_2 and CaO depends on the smelting vessel's capacity. Since the basicity is a ratio between the CaO and SiO_2, always maintained at a value above one, that justifies the selection of the ranges of CaO and SiO_2 masses.

Since the discretised state and action space are now defined, we can use the HIOM simulator to create a data dictionary of our next states. As shown in Fig. 3.1, we input all the possible combinations of ore and lime rates in the simulator. Then we integrate the simulator output over a fixed time interval and add it up with the starting states for CaO and SiO_2 masses. This way, we map the possible next states, starting from our initial states and taking all possible actions. This is the transition probability matrix P

3.4. Selection of Time constant

Firstly, to determine an appropriate time constant for integrating the simulator result, the interval between slag taps was analyzed. It was observed that the average interval between slag taps ranges between two and three hours. Using this duration would allow our model to predict the optimal action up to approximately the next slag tap. However, a significant drawback is the potential accumulation of model error over this extended period, possibly leading to substantial discrepancies between anticipated and actual next states.

Additionally, there's the potential for changes in input ore rates between two slag taps. Consider a scenario where the controller table determines the optimal lime input for a two-hour window. If there's a set point change after just one hour, the controller's performance will deviate from the desired outcome. Given these considerations, a time constant of one hour was chosen. This decision enables set point adjustments every hour, minimizing integration errors. Subsequent testing validated this choice, as controllers based on this one-hour interval demonstrated superior performance on the simulator.

3.5. Cost Function

We aim to employ Dynamic Programming, and a key component of this approach is the reward or cost function. Our primary objective is to identify the best action that minimizes deviation from the target basicity value. The controller's goal is to consistently maintain this target basicity. To do so, we introduce lime into the system. Importantly, there's no associated cost with this action. As a result, our cost function is exclusively dependent on the system's state, particularly focusing on the basicity value. Larger deviations carry higher penalties, as significant departures from desired levels can result in adverse, potentially irreversible plant conditions like slag foaming. To counteract this, our strategy involves minimizing costs using a quadratic cost function.

Based on the above discussion, the cost function is defined by Eq. 3.2. In the equation, λ is a positive constant. Fig. 3.2 depicts the plotted shape of the cost function. This defines the R and completes our

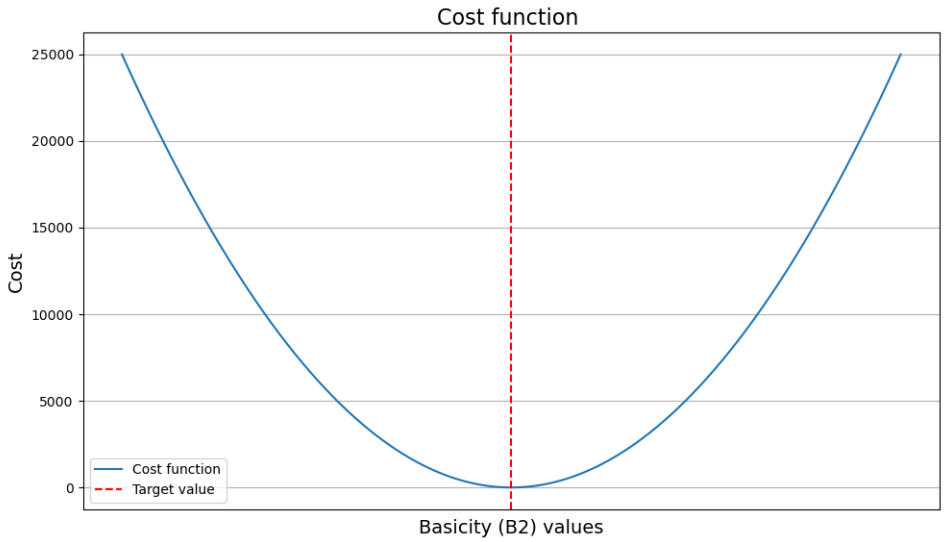

Figure 3.2: Cost - Function

definition of this MDP.

$$R(s) = \lambda * (B2_{target} - B2)^2 = \lambda * (B2_{target} - s_2/s_1)^2 \tag{3.2}$$

3.6. Consideration of Model Errors- HIOM$^+$

The HIOM process model, developed by Tata Steel, is a high-fidelity representation, meaning it seeks to emulate the HIsarna plant's operations as accurately as possible. However, it's crucial to understand that no model is completely free from errors, and HIOM is no exception. The iron smelting process, characterized by extreme temperatures and large volumes of materials undergoing thermochemical reactions within the smelting furnace, is inherently complex. Given these high temperatures and the myriad variables at play, the process exhibits natural variability; thus, predicted outputs can be influenced by any of these factors.

While HIOM is rooted in thermodynamic and mass-balance equations, it doesn't capture all these variabilities. For a truly realistic HIsarna process model, we must incorporate model errors into the predicted HIOM outputs. Ideally, HIOM should undergo rigorous model evaluation, and based on those results, model errors should be quantified and integrated into its predictions. However, at the time of this work, a comprehensive model evaluation of HIOM hadn't been conducted due to data limitations.

Consequently, in this work, we adopt a certain assumption about the model error. We propose a normally distributed error on the HIOM outputs—specifically, the SiO_2 and CaO mass rates—with a zero mean and a standard deviation equal to 20% of the potential output mass-rate range. This assumption was informed by discussions with experts at Tata Steel, and we further seek to validate this supposition through subsequent simulations. For clarity, we'll refer to this enhanced, non-deterministic model as HIOM$^+$ throughout this document.

3.7. Conclusion

From the above discussion, we have successfully developed an MDP model capturing the relevant dynamics. We have also identified and eliminated one state, which reduces dimensionality and should reduce computational time. Discretisation of the state and the action space was also performed, and also the state transition matrix was defined using the HIOM simulator. We also defined a state-based cost function. We have successfully defined all the requirements to set up Dynamic Programming, and in the next chapter, we look at the DP algorithm.

4

Dynamic Programming

From the discussion in the previous chapter, we have now developed a discrete MDP model for our control problem. We want to optimally control the state of our system infinitely. This leads to an infinite horizon sequential decision-making problem. To solve this, developed MDP model means finding an optimal mapping between the states and the actions [19]. Given any point in the state space, we want to choose the optimal action that takes us closer to our objective, which is the target basicity value. This mapping can be represented by the Eq. 4.1

$$\pi^*(s) : S \rightarrow A \tag{4.1}$$

Where π^* represents the optimal decision mapping. Finding the optimal action is not the same as finding a greedy policy because we also want to keep the costs in future states in check. Optimising only based on immediate rewards will ignore the current action's effects on the system's future state. Therefore, selecting a greedy policy will lead to sub-optimal control, which will be more aggressive and lead to large overshoots. As discussed in earlier chapters, overshoots are not ideal for plant safety and reliability.

In the context of sequential decision-making problems, the principal aim is to minimise the infinite-horizon expected cumulative cost, defined as:

$$V^{\pi} := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right], \text{s.t. } a_t = \pi(s_t). \tag{4.2}$$

where s_t and a_t denote the state and action at time t, and γ is the discount factor and $R(s_t, a_t)$ is the running cost as defined in Chapter 3. This objective poses computational challenges due to the infinite dimensionality of the action sequence.

The Bellman optimality principle provides a recursive solution to the aforementioned problem. This principle posits that an optimal policy, irrespective of the initial state and action, will yield an optimal policy for the remaining sequence of states and actions resulting from the initial decision [20].

Based on this principle, the infinite-horizon expected cumulative cost can be represented more compactly via a fixed-point equation for the value function V using the Bellman operator Γ:

$$\Gamma(V)(s) := \min_a \left(R(s,a) + \gamma \sum_{s'} P(s'|s,a) V(s') \right) \tag{4.3}$$

The fixed point of the Bellman operator, $V^* = \Gamma(V^*)$, signifies the optimal solution, whereby the value function at the optimal policy corresponds to this fixed point.

Therefore to solve this problem, we make use of the value-iteration algorithm. It is a much-researched algorithm in stochastic control and reinforcement learning [21] [22] [23] [24]. Value iteration in Dynamic Programming (DP) literature is also known as successive approximation [20]. Value Iteration (VI) emerges as an effective solution methodology to identify V^*. This recursive algorithm employs the Bellman operator and assures convergence to the optimal value function as the number of iterations k progresses to infinity. The recursive update mechanism is defined as:

$$V_{k+1} = \Gamma(V_k) \tag{4.4}$$

When applying this theoretical solution in practice, several important considerations arise (i) the transition probabilities P might not be perfectly known,(ii) the state and action spaces are discretised for computational viability, (iii) infinite iterations may be unachievable due to finite computational resources, and (iv) interpolation techniques are required to handle continuous variables. We have already defined how we describe the P matrix in the previous chapter. The other issues will be further explored and addressed in the following sections.

4.1. Selection of state and action space

Since we are using a discrete approach to our continuous state space problem, the approximation of the actual value function can be affected by our selection of grid points. The grid point selection is done considering the trade-off between the performance or accuracy of the approximate solution and the computational spend on calculating the approximate solution.

Choosing a sparse grid will reduce computation time significantly but at the expense of a potentially inaccurate approximation for the value function. We have already described our state and action space's selection and defined the grids. We further justify our grids' choice when we analyse our controller's performance.

Another problem with the value-function approximation is the curse of dimensionality. If we want a good optimal control law, we should include all the states which affect our control objective. While doing this is ideal for the accuracy of the control policy, it incurs enormous costs on the computational time required to compute the optimal control policy. The relation between the number of dimensions and the computational time required is exponential. To deal with this, we already have reduced one dimension from our state-space as explained in Chapter 3, where we eliminate coal-rate as a state-space element as it had a linear relation with ore-rate. This helps us to compute the optimal controller in a reasonable time.

4.2. Selection of Discount Factor

For problems with an infinite horizon, the cumulative future costs can tend toward infinity. Minimizing over such an expansive sum becomes untenable. To circumvent this, the discount factor is introduced. As previously highlighted, the discount factor, represented by γ, plays a crucial role in DP design decisions. The magnitude of γ determines the relative weighting of future costs when optimizing present actions. As depicted in Eq. 4.5, each added future cost is multiplicatively discounted by γ. This equation shows that only the immediate cost is wholly incorporated, while all subsequent costs undergo progressive discounting. Further, it's inferable from Eq. 4.5 that as t approaches infinity, $\gamma^t r_{t+1}$ converges to 0, where $r_{t+1} = R(s_{t+1})$.

$$\sum_{t=0}^{\infty} \gamma^t r_{t+1} \tag{4.5}$$

The choice of γ presents a classic trade-off: weighing the significance of future costs against computational efficiency. A γ near 1 emphasizes future costs, necessitating more iterations for convergence. Conversely, a γ close to 0 prioritizes immediate rewards, often overlooking the long-term consequences of actions. Though this reduces computational time, it may jeopardize the optimality of the solution.

In our work, we determined a discount factor of $\gamma = 0.95$ to strike a balance between controller performance and computational efficiency. This choice was apt for the deterministic case. However, when examining a controller based on a non-deterministic process model, due to heightened computational demands, we opted for a smaller γ. This decision will be elaborated upon in a subsequent chapter.

4.3. Selection of Convergence Threshold

Value iteration is an iterative algorithm, and without a predefined stopping condition, it would continue for infinite iterations. This stopping condition is set based on the desired accuracy of the solution. The condition essentially represents the maximum permissible difference between the value function across two consecutive iterations, as shown in Eq. 4.6.

$$\max_s \left|V_{k+1}(s) - V_k(s)\right| < \epsilon \tag{4.6}$$

Here, ϵ is a positive constant and serves as a design parameter, analogous in its role to γ. The smaller the value of ϵ, the more accurate the resultant value function is likely to be, implying that the current iteration's value function is very close to the next iteration's. However, a smaller ϵ typically necessitates more iterations, leading to increased computational overhead. As such, choosing an appropriate ϵ is a balance between computational feasibility and solution accuracy.

Given our cost function R and our control objectives, we've chosen $\epsilon = 1 \times 10^{-5}$. A lower ϵ should bring our value function closer to the optimal, suggesting that the controller will perform better. The implications of this ϵ choice will be further evaluated during our discussion of the results.

4.4. Interpolation of the Discrete Value Function

As we use a discretised state space for our problem, the value function $V(s)$ is only defined at the selected discrete state points. While evaluating the Eq. 4.3 it might happen that the resulting next state s' is not a part of the discretised state space set S. To evaluate the value function at such a s', an approximation technique is required.

Many linearly or non-linearly parameterised techniques can be used for the approximation of the value function. Non-linearly parameterised solutions include neural-network-based approximators which are more efficient, but they add complexity to the problem. Given the complexity and the dimensionality of the problem at hand, linearly parameterised approximations are considered a good choice. Different techniques can be used for linear approximation of the value function, such as crisp discretisation [25], use of radial basis functions [26] and multilinear interpolation [27] [28]. Multi-linear interpolation is one of the simplest techniques for approximating the value function. Given our problem's dimensionality and complexity level, it is also the best-suited method.

Python's scipy library offers a 'Regular Grid Interpolator' function, which was initially considered for the interpolation. While the accuracy of this interpolator is commendable, profiling of the Python script revealed that a significant amount of computational time was expended on this library function. This could be attributed to various factors, including the grid size and the complexity of the Dynamic Programming (DP) algorithm.

Given the computational inefficiencies associated with the library function, defining a custom trilinear interpolation function tailored to the problem was deemed necessary. This approach not only ensures computational efficiency but also maintains the accuracy of the solution. The custom trilinear interpolation function is defined as follows:

Given a point $P(x, y, z)$ and a 3D array V, the trilinear interpolation $I(x, y, z)$ can be calculated as follows:

1. Find the indices x_0, y_0, z_0 of the grid points in V that are closest and smaller to x, y, z respectively. Also find the indices x_1, y_1, z_1 of the next grid points in V.
2. Calculate the eight corner values $c_{000}, c_{001}, c_{010}, c_{011}, c_{100}, c_{101}, c_{110}, c_{111}$ from V. These are the values of V at the cube's eight corners containing the point (x, y, z). Specifically, $c_{ijk} = V[x_i, y_j, z_k]$ for $i, j, k \in \{0, 1\}$.

3. Calculate the relative distances x_d, y_d, z_d of x, y, z from the grid points x_0, y_0, z_0.
4. Interpolate along the x-axis to get $c_{00} = c_{000}(1 - x_d) + c_{100}x_d$, $c_{01} = c_{001}(1 - x_d) + c_{101}x_d$, $c_{10} = c_{010}(1 - x_d) + c_{110}x_d$, $c_{11} = c_{011}(1 - x_d) + c_{111}x_d$.
5. Interpolate along the y-axis to get $c_0 = c_{00}(1 - y_d) + c_{10}y_d$, $c_1 = c_{01}(1 - y_d) + c_{11}y_d$.
6. Interpolate along the z-axis to get the final interpolated value $I(x, y, z) = c_0(1 - z_d) + c_1 z_d$.

The code for this can be found in Appendix A.4.

4.5. Conculsion

From the discussions in this chapter, we have defined the discrete DP algorithm and all the design choices made for solving the DP. Using the MDP model developed in 3 and the above discussion, we have fulfilled all the modelling and design requirements and can now compute the optimal controller for the problem. In the next chapter, we go into the implementation details of the MDP and the DP code.

5

Implementation of MDP and DP

This chapter discusses the practical implementation of the topics discussed in Chapters 3 & 4. First, we look into the practical implementation of the modelling process and then discuss the DP algorithm's implementation. We use one Python script to build the MDP model, and perform DP. This script accepts an XML file with all the hyper-parameters defined, the linear constant between the iron-ore rate and coal rate, c, and the path to the installed HIOM simulator. The output of this code is a look-up table in the form of a CSV and Python readable `pickle` file.

5.1. Developing the MDP model

5.1.1. Discretising the state & action space

In Fig. 3.1 we saw the steps involved in developing the MDP model. The first steps involved the selection of the state and action spaces. After selecting the state and action variables, we discretize the action and state space to develop a discrete MDP model. Table 3.1 shows the range and discretisation points for the action and state variables. We select equidistant points in the defined range. This code is in Appendix A.1. Fig. 5.1 depicts the discretised state space. The axes have been scaled to protect the IP of Tata Steel.

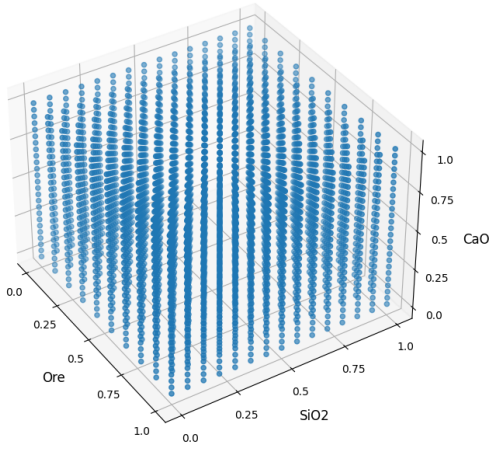

Figure 5.1: Discretised state space

5.1.2. Capturing Dynamics and creating P matrix

As discussed earlier, we use the high-fidelity process model developed by Tata Steel to build our MDP model. HIOM's inputs are in the form of XML files where the rate and composition of input materials are present. Usually for a run the composition of all the different input materials is pre-computed at Tata Steel. We then make use of these pre-computed composition values and values for other involved variables, stored in an XML format compatible with the HIOM simulator. These values are the hyper-parameters, refer Fig.3.1, defined by the Engineers at Tata Steel.

From our state and action space, only ore-rate s_3 and lime rate a_1, are inputs to the HIOM simulator. The coal rate is also an input, but we assume a linear relationship between it and the ore rate and define it as a user-defined input c. We use the discretised ore and lime points, the XML file, and the linear constant c and pass it on to a function. This function parses the input XML file and identifies the tags related to the injection points of ore and lime. It then changes the input rate of ore, coal and lime and creates a new XML file. This function then calls the HIOM simulator executable and passes this newly created XML as input to the simulator. The HIOM simulator updates the slag and hot metal output value in the same XML. The function parses this updated XML and reads the newly created SiO_2 and CaO rate in the slag. It makes a mapping of the input rates of ore and lime to output rates of SiO_2 and CaO or $\{s_3, a_1\} \rightarrow \{\dot{s}_1, \dot{s}_2\}$. This is repeated for all combinations of ore and lime rates in the defined grid, and a data dictionary of the above-defined mapping is created.

The function then loops over all possible combinations of our state-action pairs, including the discretised SiO_2 and CaO masses. Using the defined time constant and the above-defined data dictionary, we convert the rate-to-rate mapping to a mass-to-mass mapping, $\{(s_1, s_2, s_3), a_1\} \rightarrow \{s_1^+, s_2^+, s_3^+\}$ where $s_3^+ = s_3$. This now defines our $P(s_{k+1}|s_k, a_k)$ matrix. The code for these steps is shows in the Appendix A.2.

5.2. Implementing Dynamic Programming

From the above, we have developed our MDP model and can now perform dynamic programming. The data dictionary or P matrix is now passed on to a function which performs the Value Iteration (VI). After each iteration, we check the convergence condition as defined by Eq. 4.6. If the condition is satisfied we say that the algorithm has converged. The convergence threshold, ϵ, as described before is a design parameter. We then define the initial value table and the optimal action table to store the optimal control values. We initialise these tables with 0.

We aim to find the best action which can be taken at all possible combinations of the states. We must loop over all the possible state values and actions for this. Simple loops for all states and actions will be computationally costly. To make a saving on the computation time, we make use of vectorisation made possible by the use of NumPy Python library. Using vectorisation makes use of hardware-level parallelism and accelerates the computations. For all possible state pairs, we calculate the immediate cost of taking one action from our set of possible actions. We store these costs as a vector.

We use the same vectorization technique as above to calculate the interpolated value in the V table at all the possible next states and store them in another vector. This gives the future reward. Using the immediate reward vector and multiplying the discount factor γ with the future reward vector, we get a new vector which gives us the total cost of taking all the different actions. We then find the minimum value in this vector and also action which incurred the minimum cost and store these values to the updated value table and action table, respectively.

We repeat the above process and update the value table until convergence. Once the solution has converged, the optimal action table is stored in two formats, one CSV and the other as serialised file known as a pickle file. We use this pickle file later to run simulations on the HIOM simulator. The code for performing the steps involved in performing DP is present in Appendix A.3. Fig. 5.2 depicts a visualisation of the optimal action table obtained.

5.3. Simulation

We now have the optimal controller table. We implement the developed optimal controller table on the HIOM process simulator. For simulations, we use the recorded input data from previous runs, which

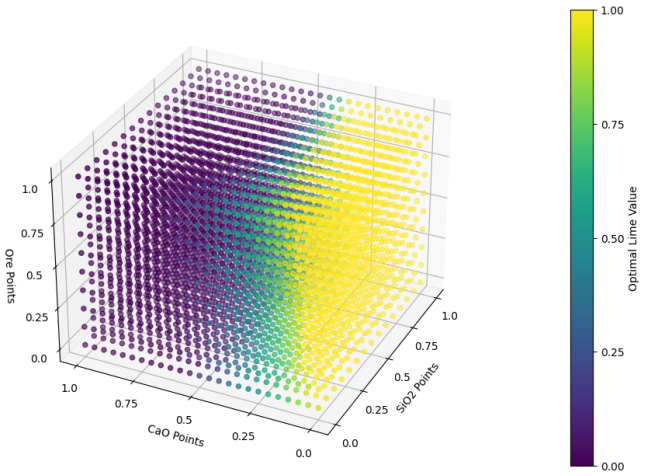

Figure 5.2: Optimal Action Table Visualised

we have in the form of a database. From this database, we want to use the actual input values of iron ore and coal rates and the initial approximate masses of the SiO_2 and CaO masses in the vessel.

We use a Python script to read the previous run database and find the data corresponding to input iron ore and coal. This database contains the one-hour, 15-minute and 1-minute average input rates for ore and coal. The python script then, reads the input XML file, which contains all the hyperparameters, parses it and finds the tags for the injection points of ore and coal. Using the data from the database, it changes the injection rates for ore and coal and creates a new XML file. It repeats the process for all the entries in the database.

We can run the simulations after creating the XML files for all inputs. We must keep track of our state variables, as the HIOM simulator does not do that. We define variables to store the initial masses of SiO_2 and CaO in the vessel and then update these values as we simulate the inputs. One more variable which we need to keep track of is the total slag mass inside the smelting vessel. This needs to be done because the slag is tapped as a whole in the real system, not only the SiO_2 and CaO mass. We find the mass percentage of SiO_2 and CaO in the slag, tap the slag using the values from the database, and then convert the mass percentage back to actual masses using the updated slag mass after the tap. This is where we use our assumption of homogeneous slag tap.

After defining all the required initial points, we start our simulations. First, the script parses the created XML containing the ore and coal input values. We find the ore's injection rate value and use the SiO_2 and CaO mass estimate to query the optimal action table. By this, we get the optimal lime action at our current state. We use this value to update the lime injection value in the current XML. We then call the HIOM executable from the code and use the updated XML as input. The HIOM updates the simulation output in the same XML, and then we parse the XML again. This XML now contains the rates and composition of the output slag, we read the rates of SiO_2 and CaO, which are present in kg/hr. We simulate three time intervals, 1-hour, 15-minute and 1-minute, depending on the database from which the XML was created. We use the time intervals and the mass-rate output for SiO_2 and CaO to update the masses. We then calculate and record the $B2$ value, the ratio between CaO and SiO_2. We then update the slag mass value and deduct the slag's mass if there was a slag tap performed in the database. We repeat the above process for all the XML files created from one database and store the values of $B2$, ore input, and optimal lime input to analyse the simulations.

We also simulate the actual lime inputs in the database to compare the optimal controller's simulation with the human operator-controlled simulation input. For this, only a slight modification in the above-defined process is required. Instead of querying the optimal controller table for the lime input value,

we use the database to change the lime input injection value. The actual lime inputs are not exact as the input to the real system for various reasons such as limited hardware accuracy and measurement noise. In situations where multiple control strategies have been evaluated, we use parallel simulations and a discretised error set. The code for simulations and cost-analysis is present in Appendix A.5.

6

Results

In this chapter, we look at the simulation results of the developed optimal controller table. We use plant run data from two different runs for our simulations. We have 13 datasets from these runs, 11 from the first run and two from the second run. We use datasets from two different runs to analyse the performance of the developed control strategy over different set of hyperparameters which are defined by the experts at Tata Steel. In these two runs, the composition of the input material, iron ore, coal and lime used differed and based on this the obtained controller table would also differ.

We use the given hyperparameters and implement the strategy we have defined to obtain the controller table for both these datasets. We then perform simulations on the HIOM simulator and compare the performance on the simulator between the actual operator actions present in the database and the controller table advice. The current control strategy at Tata Steel is based on an affine relationship between iron-ore input and lime input, derived based on the hyperparameters for that run. The operator inputs in the database are based on this relationship.

All the simulation results presented in this chapter are based on Eq. 6.1. We vary different parameters of this equation to show the robustness and adaptibility of the proposed solution. In order to compare the performance of the controller to the human operator we replace $\pi^*_\theta(s(k))$ with $\pi(\text{operator})$. To show that the controller performs well with different time intervals of simulation we vary Δt between 1-min, 15-min and 1-hour intervals. This shows the ability of the controller to react to changes in the input set points, as they can happen at any moment. Different Δt values simply show the results of querying the controller at different time-intervals. We also want to evaluate the performance of the controller on different set of hyperparameters θ, this is done by using data from two different runs as defined above. We also want to evaluate the performance of our controller on HIOM$^+$ model as defined in Chapter 3. So we use two scenarios first where we don't add model errors or simply $w(k) = 0$ and the non-deterministic scenario where we add the model errors as shown in Eq. 6.1.

$$s(k+1) = s(k) + \int_{k\Delta t}^{(k+1)\Delta t} \text{HIOM}(\theta, s(k), \pi^*_\theta(s(k))) + w(k)dt \tag{6.1}$$

where

$s(k)$: discretised set of initial states,

θ : hyperparameters,

$\pi^*_\theta(s(k))$: optimal action table,

$w(k)$: error percentage,

Δt : time interval of simulation

We divide our results into two further sections. The first section analyses our controller's performance on the deterministic model. We use different values of Δt here to show the adaptibility of the controller.

In the second section, we evaluate on HIOM$^+$ but we do not make any changes to tackle these errors in our control strategy. In both sections we vary the hyperparameters, θ, and compare with the operator actions , π(operator). Table 6.1 presents an overview of figures in this section. We use the data from the actual runs on HIsarna. It was not possible to analyse the performance of the developed control strategy on the actual HIsarna plant due to safety concerns.

Table 6.1: Overview of Figures in this chapter

	Model		
	HIOM	**HIOM$^+$**	**HIsarna**
Operator Inputs	Fig. 6.1-6.10	Fig. 6.12,6.14,6.16	Fig. 6.12,6.14,6.16
Optimal Controller Inputs	Fig. 6.1-6.10	Fig. 6.11,6.13,6.15	N.A

6.1. Simulation results on Deterministic Model

In this section, we assume the model is deterministic, i.e. $w(k) = 0$, and perform our simulations. We use the recorded lime-input set points in the data to compare the human operator with the developed optimal control strategy. It is important to note that the set point is not always the actual input to the smelting vessel due to the limited accuracy of the equipment. The confidence in the set point being equal to the actual input was higher for the second run's data than the first run's. One additional advantage of the second run's data is the availability of the actual slag-tap mass, which in the previous data is simulated through an approximation as this data is not available for the previous dataset.

6.1.1. Simulation Results on Run-1's data

We present the results of the controller's performance using the database available for Run-1.In this database, we have 11 datasets where the initial plant states differed. Also, most of these datasets were recorded for long periods. We query the controller at 15-minute intervals and 1-hour intervals, i.e. $\Delta t_1 = 15$ mins & $\Delta t_2 = 1$ hour. We avoid the simulations where we query the controller table every minute, i.e. $\Delta t = 1$ min, as the computational time for doing this for long-duration simulations is very high. We do present that strategy for the next database. For the operator input simulation, we use the per-hour averaged lime-input set point rate present in the database.

Depicting the performance of all 11 datasets through graphs will be redundant. We present the cases where the initial $B2$ values were very different from the target $B2$ value. We also plot the lime-input rates used by the controller and the operator. Fig. 6.1 - Fig. 6.6 presents the result of these simulations. The values for the Y-Axis are hidden to protect the IP of Tata Steel.

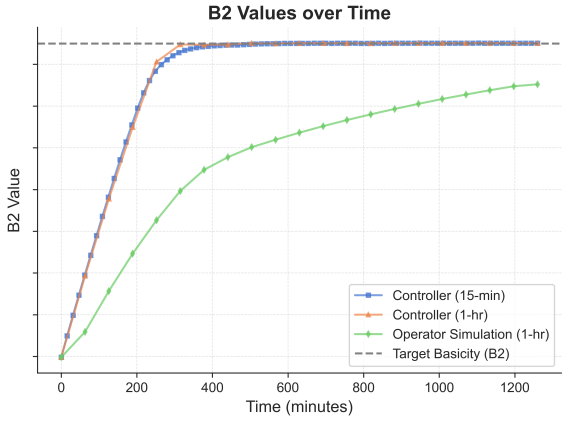

Figure 6.1: Run-1 Dataset-1 Simulated B2 values

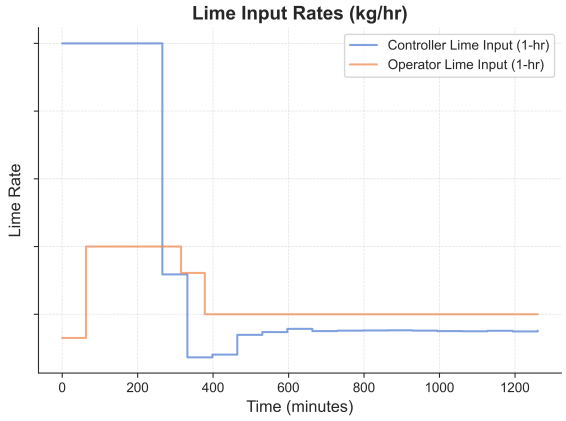

Figure 6.2: Run-1 Dataset-1 Operator vs Controller Input

Fig. 6.1 & 6.3 depict the scenarios where the initial $B2$ values were much lower than the target. We simulate three scenarios, the operator inputs averaged over an hour, the controller input every hour and the controller input every 15 minutes. A stark difference between the performance in the operator-controlled and optimal controller-controlled simulations can be observed. In Fig.6.1 the controller's

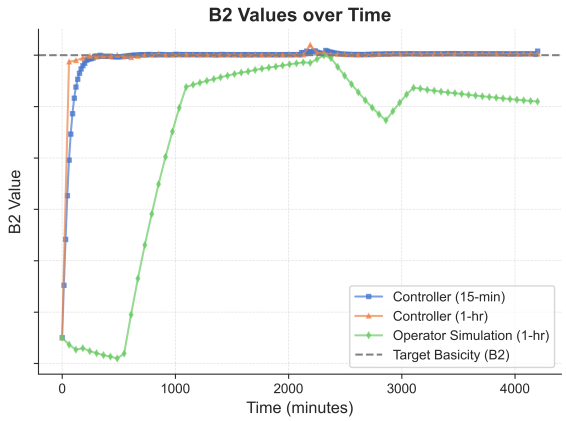

Figure 6.3: Run-1 Dataset-5 Simulated B2 values

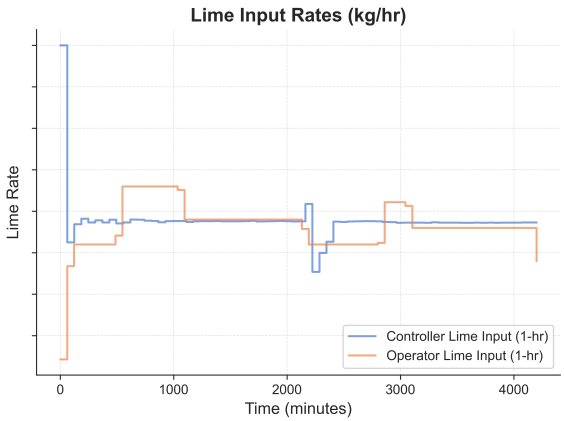

Figure 6.4: Run-1 Dataset-5 Operator vs Controller Input

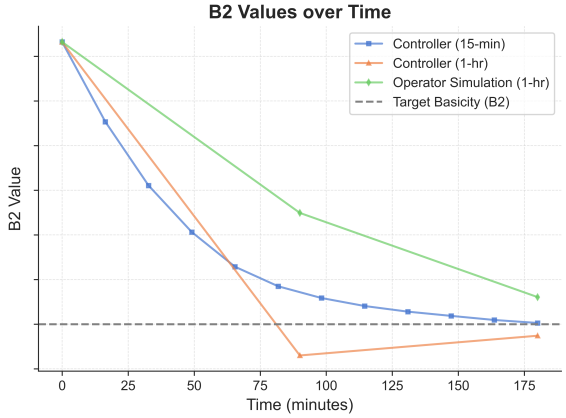

Figure 6.5: Run-1 Dataset-9 Simulated B2 values

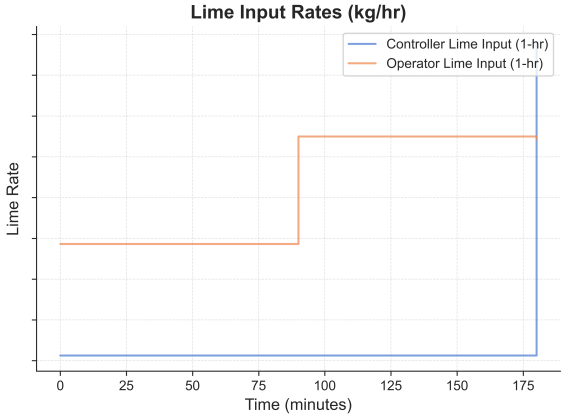

Figure 6.6: Run-1 Dataset-9 Operator vs Controller Input

input drives the plant basicity towards the target much quicker than the operator inputs. The controller inputs take nearly three hours to drive the plant basicity close to the target, but this is because the input is constrained at a certain value, which can be seen in Fig. 6.2. After reaching the target value, the optimal control strategy can also maintain the basicity at the target constantly. In Fig. 6.2, it is seen that the lime-input rate drops for the optimal control strategy as the plant basicity reaches its target. The same trend is also seen in the plot for operator actions, but that can also be a reaction to the change in iron-ore input. The operator-controlled simulation rises towards the target basicity value but never reaches the target value. It is also seen that after 400 minutes, there's only a small mismatch between the operator inputs and the optimal controller inputs, it can be said that the operator is a little cautious with the lime input rate and if the operator actions were higher in the initial 400 minutes, the operator could have attained and maintained the target basicity value. Comparing the responses of the two different Δt used for optimal controller inputs, it is observed that the response is pretty similar in both cases. The one-hour interval querying performs slightly better, reaching the target in less time.

In Fig. 6.3 a similar scenario is depicted where the initial $B2$ is lower than the target. Here the difference between the initial and the target basicity is $\frac{1}{3}$rd of the difference in the previous case, Table 6.2. Hence, the optimal controller table is able to reach the target quickly. It is again seen that the one-hour querying is able to reach the target basicity in slightly less time, other than that, the performance of both the optimal control methods is similar. The operator inputs-based simulations are much slower in reaching the target basicity. Fig. 6.3, and after reaching the target value, due to a change in the ore input, the operator adjusts the lime input to a lower value but does not revert back to the original lime input once the ore-input is changed to the original value Fig. 6.4. This slight difference in rates for a few hours adversely affects the $B2$ value inside the furnace. Again a similar trend of the initial input rates being very different between the optimal control strategy and the operator can be seen, which supports the

notion that the operator is cautious with the lime-input rate.

Fig.6.5 & 6.6 depicts a scenario where the target basicity is higher than the target. The shorter simulation duration is another difference between this scenario and the others above. This scenario only presents data for a three-hour simulation. Here again, a similar comparison between the two different Δt can be drawn as the one-hour simulation reaches the target basicity in lesser time. Still, the one-hour querying strategy undershoots, whereas the 15-minute strategy settles smoothly. The operator-controlled simulation does not reach the target within the three-hour interval. This can be attributed to the difference in the input rates between optimal control inputs and the operator inputs Fig. 6.6. Here the operator inputs are seen to be higher than the optimal controller inputs, this is also a cautious reaction and is largely governed by the input-ore rate.

The plots above show the controller performance is much better than the simulated operator actions. The controller is able to drive the basicity of the system to the target value in a much quicker time, and once at the target, the controller maintains the basicity at the target. To quantitatively compare the optimal control and the operator action-based simulation, we calculate the cost at each state of our different scenarios using the cost function defined by Eq. 3.2. For the 11 datasets available for this run, the costs of both the operator control and optimal control table are presented in Table 6.2. For the optimal control table, we only use the costs obtained by using the 1-hour querying strategy. It can be seen that the optimal controller outperforms the operator in all of the available scenarios.

Table 6.2: Performance Comparison Operator vs Optimal Controller on HIOM

Data Set Number	$B2_{initial}$-$B2_{target}$	Operator Control Costs	Controller Table Costs
1	-0.1504	30 827	**7858**
2	-0.0077	371	**2**
3	0.003	10 249	**2**
4	-0.0091	182 719	**66**
5	-0.055	16 191	**4**
6	-0.0022	15 475	**61**
7	-0.031	10 897	**4**
8	0.39	184 969	**133**
9	0.0158	16	**1**
10	-0.0108	681	**2**
11	-0.0262	5751	**2**

6.1.2. Simulations on Run-2's Data

In the second run, we have two datasets, but the duration of these two datasets was around 7 hours, so comparing 1-minute interval simulation is also viable. So we run simulations where we query our controller table with new feedback every one 1-minute interval, 15-minute intervals and one-hour intervals.

Figures 6.7 & 6.9 show the $B2$ evolution for run-2's dataset. For the first data, 6.7, we see that the initial plant basicity is far away from the target value and hence the input rates for both operator and the controller table are seen to be very high. Again it can be noticed that the operator is cautious as he doesn't input lime at the maximum possible rate, but chooses a slightly lesser rate Fig. 6.8. The effect of this slight difference in rates can be seen in the $B2$ trajectory, as all the optimal control table-based strategies are able to drive the basicity towards the target in lesser time. Fig. 6.8 shows a similar trend for lime-input rates, so both the controller and operator respond to changes in the plant states but the operator actions are much more cautious and lead to a poorer control of plant basicity. There is not much to separate the three different optimal control strategies, as the trajectories for plant basicity are very similar. At the end of the simulation, it is observed that the one-hour time-interval-based simulation achieves a slightly higher basicity,

In the second data, Fig. 6.9, the initial basicity is closer to the target value than the previous data. The optimal control table outperforms the operator in these simulations as the operator-controlled simulation goes further below the initial basicity. The cause of this can be found in Fig. 6.10, where initially the operator input is seen to be lesser than required, but between 100 and 200 minutes interval, the

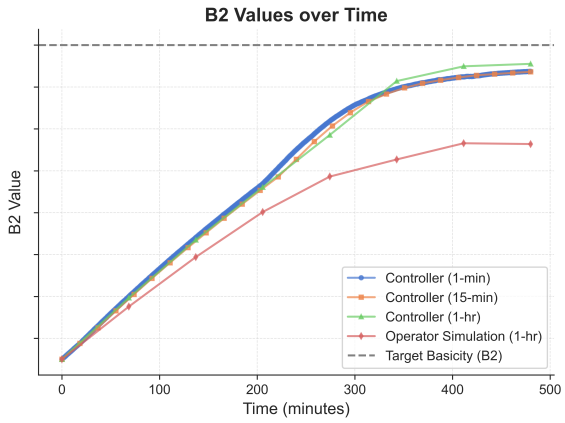

Figure 6.7: Run-2 Dataset-1 Simulated B2 values

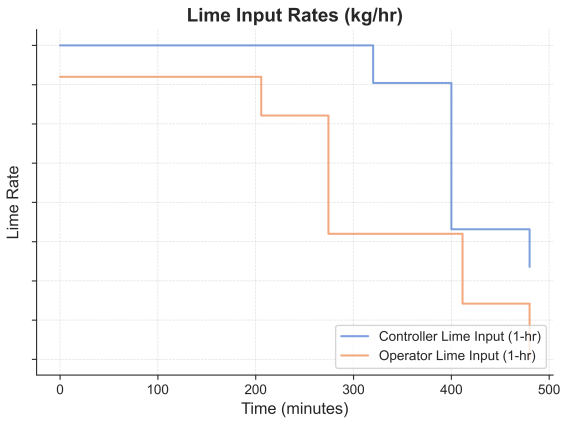

Figure 6.8: Run-2 Dataset-1 Operator vs Controller Input

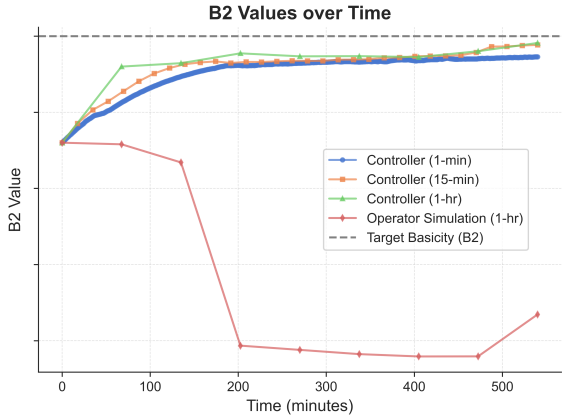

Figure 6.9: Run-2 Dataset-2 Simulated B2 values

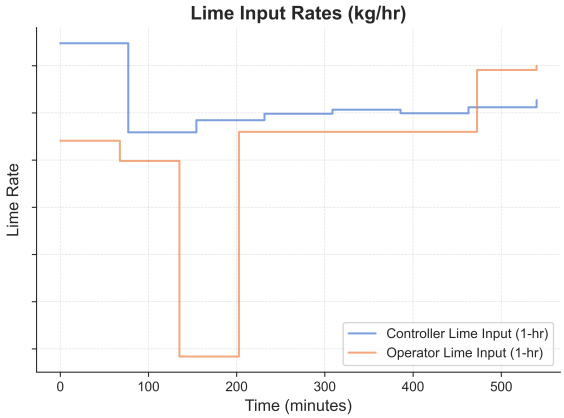

Figure 6.10: Run-2 Dataset-2 Operator vs Controller Input

operator lime input goes to 0. This drives the plant basicity to a much lower value and when the operator corrects the lime-input, it is insufficient to increase the system's basicity. As for the comparison between different querying times, there is not much difference in the trajectories for all three time periods, and the one-hour interval-based querying is slightly better.

Table 6.3 presents the quantitative performance comparison between the operator input simulation and the one-hour time-interval-based optimal controller table simulations. Similar to the previous case the optimal controller table-based simulations are better as compared to the operator input control.

Table 6.3: Performance Comparison Operator vs Optimal Controller on HIOM Run2

Data Set Number	$B2_{initial}$-$B2_{target}$	Operator Control Costs	Controller Table Costs
1	0.075	4533	**3013**
2	0.014	4178	**24**

6.2. Simulation results on HIOM$^+$

So far we have considered that the model or the dynamics that govern the HIOM simulator are deterministic. This is probably not true, as many sources of uncertainties add errors to the model dynamics. There can be many sources of this uncertainty such as measurement noise, actuator noise and other factors. In this section, we evaluate the performance of our optimal controller on the HIOM+ model as defined in Chapter 3.

We draw errors from two normal distributions with 0 mean and standard deviation equalling 20% of the range of possible SiO_2 and CaO mass rate predictions from the HIOM simulator. We then simulate each

scenario with each control strategy under study 100 times. From this, we get 100 possible trajectories $B2$. We find the mean and standard deviation of these 100 trajectories and plot the mean and 2 standard deviations. By doing this, we want to compare the performance of the optimal control strategy to the operator inputs. This can be considered an unfair comparison as the optimal controller reacts to the disturbances, whereas the operator inputs stay the same. Since we know that the operator inputs are only based on the input-ore rates, and the ore rates stay the same, this can be considered a fair comparison.

We simulate the same scenarios as we did for the deterministic model. Since we already know that the performance of the different time-interval control strategies is similar, we only use the 1-hour interval-based control strategy. We also don't plot the input-lime rates, as they wouldn't change for the operator inputs. We plot the results for the optimal controller and the operator inputs in separate plots for clarity. We plot the actual recorded $B2$ present in the dataset along with the simulated operator actions to validate our model error estimation. If the average of the $B2$ trajectories is close to the actual recorded $B2$ we can say that the assumption was valid.

6.2.1. Simulations on Run-1's data

We first simulate the Run-1's dataset on our HIOM+ model. We use the same scenarios as used in the deterministic model. The average of all the trajectories is plotted using the blue dots, and the dark and light-shaded regions represent the 1 & 2 standard deviations from the mean. In Figures 6.11 - 6.14 we consider the cases where the initial $B2$ was lower than the target. In the optimal controller-based simulations, Figs. 6.11 & 6.13, we see that the average $B2$ trajectory is similar to the one seen in the deterministic case in the previous section. It can also be observed that the $B2$ trajectory is much closer to the target for the optimal controller's simulation as compared to the operator input's simulation Fig. 6.12. The trajectory trends are similar for the $B2$ plots compared to the deterministic case. Due to the hidden Y-axis labels and different scaling of the Y-axis, the plots appear very close to each other regarding the possible $B2$ values, but that is not the case. The quantitative results presented later also prove this point, Table 6.4. Still, it can also be observed in the plot for the operator input's simulation in Fig. 6.14, that the $B2$ value can be lower than the initial $B2$ value, which is not the case for controller input based simulations Fig. 6.13.

The same observations can be made from the Figures 6.15 & 6.16. Since the initial $B2$ is higher than the target value, the desired trajectory is to reduce the $B2$. In the controller-based simulations, the probability of reducing the $B2$ from the initial value is higher than in the operator input-based simulations. Also, the actual recorded $B2$ values from HIsarna, are plotted in the operator input-based simulation plots. All the recorded actual $B2$ trajectories are within the one standard deviation region for the possible $B2$ trajectories. From this, we can say that the 20% model error is a generous over approximation of the model error.

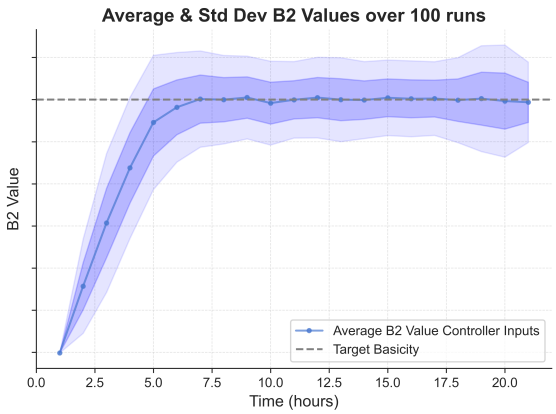

Figure 6.11: Run-1 Dataset-1 Controller Input Simulation with Errors

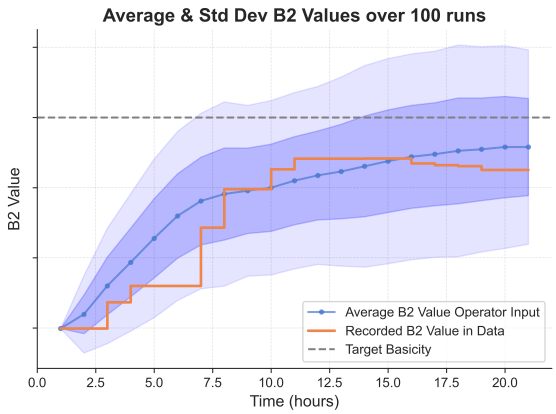

Figure 6.12: Run-1 Dataset-1 Operator Input Simulation with Errors

In table 6.4, we present a quantitative analysis of the performance of the operator inputs and the con-

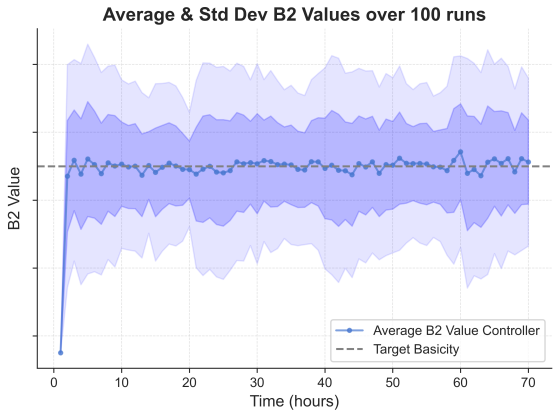

Figure 6.13: Run-1 Dataset-5 Controller Input Simulation with Errors

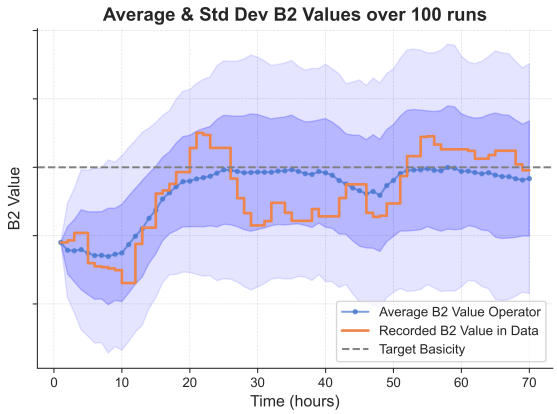

Figure 6.14: Run-1 Dataset-5 Operator Input Simulation with Errors

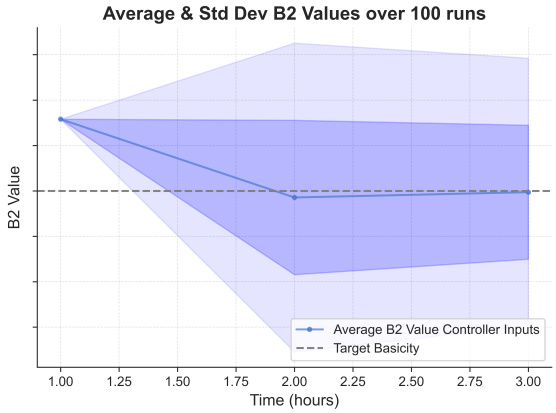

Figure 6.15: Run-1 Dataset-9 Controller Input Simulation with Errors

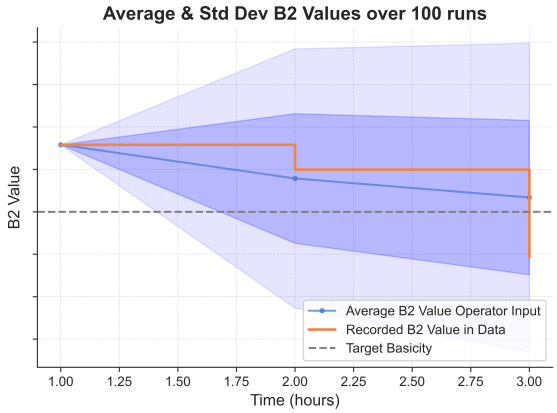

Figure 6.16: Run-1 Dataset-9 Operator Input Simulation with Errors

troller table. We calculate the cost of each of the 100 $B2$ trajectories for both strategies. Then we find the mean and standard deviations of these costs, which are presented for all the data available under this run. Like the deterministic case, the controller input-based simulations easily outperform the operator input based simulations. The average cost and the standard deviations for all the scenarios are lower for the controller-based simulations.

6.2.2. Simulations on Run-2's data

We also simulate the run-2's available data using both control strategies. We use the same methods as described above to perform the performance analysis. Figures 6.17 - 6.20 present the plots for these simulations. Similar trends as observed for the run-1's data are seen. In Fig. 6.17 it is seen that the probability of the $B2$ value reaching the target basicity is higher than that observed in Fig. 6.18. It can also be said that the controller-based simulations drive the system basicity towards the target basicity faster, as in the deterministic model.

Similarly, in Figures 6.19 & 6.20, the performance of the controller inputs is observed to be better. As observed in the deterministic model simulation for the operator inputs, the possibility of the $B2$ value decreasing and going further away from the target basicity is higher than it increasing. The controller-based simulations have a much better probability of driving the plant basicity towards the target and a much lesser probability of reducing the plant's basicity. Also it is observed from these plots that the actual recorded $B2$ values fall within the one standard deviation of the possible $B2$ trajectories.

Table 6.5 presents the results of the quantitative analysis on this run's data. As seen for the previous run

Table 6.4: Performance Comparison Operator vs Controller on HIOM$^+$

Data Set Number	Operator Inputs		Controller Inputs	
	Average Cost	Standard Deviation	Average Cost	Standard Deviation
1	40 089	22 561	9453	2985
2	8639	8680	1346	458
3	36 998	20 825	2994	836
4	274 867	144 207	5685	1043
5	62 780	39 025	5127	1015
6	33 610	30 152	2581	724
7	42 386	28 016	3267	805
8	220 719	89 811	3108	820
9	254	414	203	207
10	6485	6595	1131	446
11	16 346	12 918	1442	577

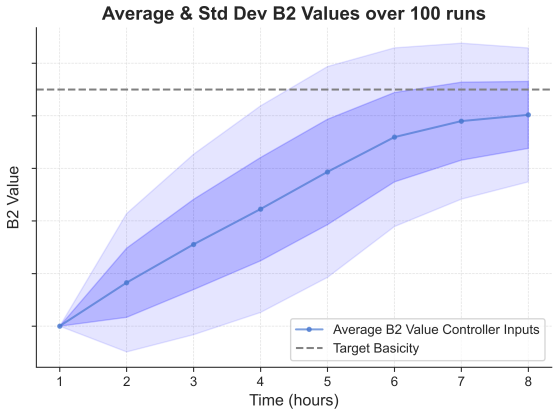

Figure 6.17: Run-2 Dataset-1 Controller Input Simulation with Errors

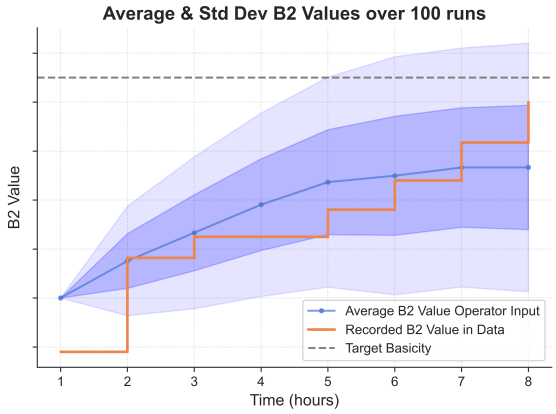

Figure 6.18: Run-2 Dataset-1 Operator Input Simulation with Errors

and also for the deterministic case results the controller-based costs are much lower when compared to the operator inputs-based costs. The same can be observed for the standard deviations, and that is expected as the operator input based simulations are more affected by errors as they don't react to these errors and their responses are fixed.

Table 6.5: Performance Comparison Operator vs Controller on HIOM$^+$

Data Set Number	Operator Inputs		Controller Inputs	
	Average Cost	Standard Deviation	Average Cost	Standard Deviation
1	8579	4422	6462	3805
2	5327	4000	666	458

6.3. Conclusion

From the above discussion, we have shown the results of the developed control strategy on the HIOM and HIOM$^+$ simulators. We now present the conclusions from the above simulations on the deterministic HIOM model and the non-deterministic HIOM$^+$ model.

6.3.1. HIOM Model

The above simulations show that the developed control strategy is better than the human operator inputs on the deterministic model. The first conclusion from the human operator's inputs is that the human operator is very cautious while feeding lime into the system. Another point to note is that the trends for most of the scenarios for lime input, match between the operator and the optimal controller,

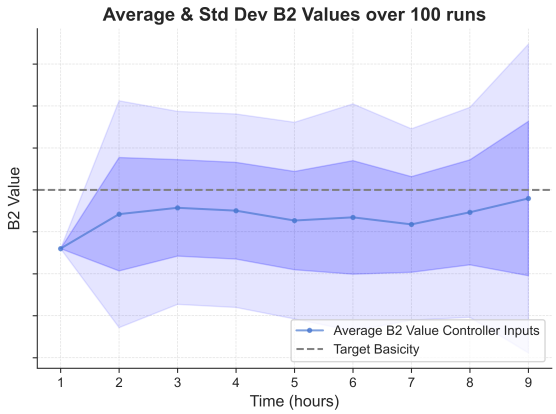

Figure 6.19: Run-2 Dataset-2 Controller Input Simulation with Errors

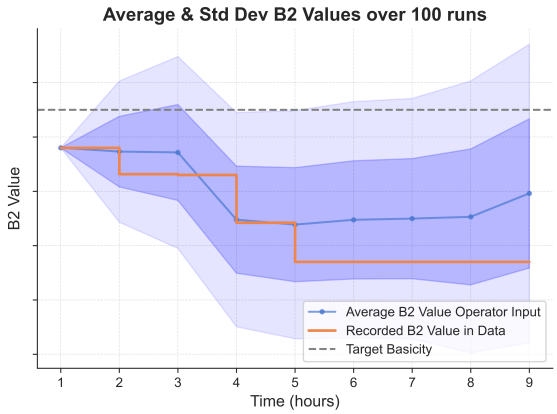

Figure 6.20: Run-2 Dataset-2 Operator Input Simulation with Errors

in most scenarios, it is seen that the difference between the initial actions causes the main differences between the $B2$ trajectories. It is known that the operator's actions are based solely on the iron-ore input rates and do not consider the slag's mass or composition inside the smelting furnace. This results in an inadequate control strategy as shown by Figures 6.4 & 6.10, where it can be seen that there is small or no difference between the actions of the operator and the optimal controller, but the difference in $B2$ plots is much higher, Figs. 6.3 & 6.9. This is caused because both the control strategies respond to the same input iron ore rate, but the operator controller strategy does not respond to the slag mass and composition inside the system. Hence, a one-dimensional control law is ineffective for controlling the HIsarna process.

In comparing the three different time-interval querying of the optimal controller table, there was not much difference in the resulting $B2$ trajectories. Since the model predicts one hour ahead, the one-hour interval-based querying is expected to perform better, as seen in the simulations above. Hence the developed control strategy will perform well even if there is an abrupt change in the system feedback or the ore-input rate and is not bound to strict time-interval-based usage. Also, it is seen that the developed control strategy performs well on different datasets with different hyperparameters. This shows that the controller is robust and adapts well to different plant dynamics.

6.3.2. HIOM$^+$ Model

The controller-based simulations produce better control of the plant's basicity even in the presence of model errors. These simulations give a more realistic performance evaluation of the developed control strategy's performance on the HIsarna plant. In the worst cases, the controller's performance comes close to the average performance of the recorded operator inputs. Still, the probability that the developed controller drives the plant basicity in the correct direction is higher. The assumption about the model error is also found to be generous as the recorded $B2$ trajectories in data fall within one standard deviation of the possible trajectories.

7

Alternative Control Methods

In this chapter, we discuss some modifications of the above-discussed control strategy. The goal behind these alternative control techniques is to improve the performance of the developed control strategy. Improvement of the current technique can be quantified by reducing running costs, reducing computation time, and improving the readability of the DP output. We propose and examine three alternative control techniques and compare them with the existing technique discussed so far. These techniques are not entirely different from the above discussion but are only extensions and abstractions of the already developed control strategy. By evaluating these alternative strategies, we also have an opportunity to analyse the design choices we made while developing the current control technique.

The first technique is simply an extension of the above-discussed strategy. So far, we have developed our controller based on a deterministic model and so solving the DP was much simpler due to the lesser computations required. In the previous chapter, we saw the results of taking into account the model errors and the consideration of these model errors on the HIOM simulator drives the model closer to reality. In this chapter, we discuss a strategy where we consider model errors in our DP algorithm, so we develop a control technique that considers the model uncertainty. We compare the performance of the current and proposed new techniques using simulations.

The second alternative technique is based on altering the grid sizes we considered for our states in Table 3.1. We try different grid sizes and compare if we gain performance by sacrificing computation time. Currently, we have a grid of 3750 states, and the computation time to solve DP is less than 4 minutes. Since this is not a lot, we use a finer grid to discretise our states and check if the algorithm's performance improves. We also try a few smaller grids to see the impact of the grid sizes on the controller's performance.

We use the developed controller table in the third technique to train a decision tree model. We do this because the current output of the DP algorithm, the optimal lime actions at different state value pairs, are stored in two formats a *'.pkl'* file format and a *'.csv'* file format, which is difficult to read. By making use of decision trees we can classify our controller table into a series of simple decision rules which can be visualized and are also human interpretable. We compare the performance of the decision tree-based controller with the developed controller technique and see how this abstraction performs.

We divide this chapter into three sections each discussing the three alternate methods discussed above. In each section, we also present the results of the simulation on the HIOM simulator. Since in the previous chapter, we already saw that the control strategy performs well on different hyperparameters and different time-interval querying scenarios, we only use one-hour interval querying and also use data from only one run to perform the simulations.

7.1. DP on non-deterministic model

As described above, in this technique, we define a controller that accounts for the model uncertainty. The major difference between this technique and the one we have used so far is the change in the defi-

nition of the state transition probabilities or in the definition of P matrix. So far, while solving the Bellman equation 7.3, it was always the case that $P(s'|s,a) = 1$. Since the model is no longer deterministic and the model outputs are affected by uncertainty, the definition of P matrix is no longer that simple. Due to the uncertainty, a particular action taken at a particular system may have multiple possible next states. Therefore,

$$P(s'|s,a) < 1 \tag{7.1}$$

but,

$$\sum_{s'} P(s'|s,a) = 1 \tag{7.2}$$

So, in the case for the previous controller while we only considered one possible future state, now we consider the possibility of multiple future states and evaluate the Bellman equation.

$$\Gamma(V)(s) := \min_{a} \left(R(s,a) + \gamma \sum_{s'} P(s'|s,a) V(s') \right) \tag{7.3}$$

In the previous chapter we assumed a normally distributed error with 0 mean and a standard deviation equalling 20% of the possible outputs. It was observed that this was good assumption as the recorded $B2$ values were seen to lie in the region of one standard deviation of the predicted $B2$ values. We use the same error approximation to construct our P matrix for this case. Using HIOM we find out the possible range of output SiO_2 and CaO rates. We find the highest and lowest rates for these two outputs and take the mean of those values. We then calculate the 20% value of this mean. We then select 15 equidistant points between the negative of this mean to the positive of this mean value for both outputs. We then create all possible combinations of the possible errors. In total, we create a combination of 225 possible next states from every possible state-action pair in our discretised state and action space.

To define the state transition probabilities or the P matrix, we need to find the joint probability distribution of the selected noise combinations. We make use of Eq. 7.4

$$P(x,y) = \mathcal{N}_1(x;\mu_1,\sigma_1) \times \mathcal{N}_2(y;\mu_2,\sigma_2) \tag{7.4}$$

where:

$P(x,y)$**:** Represents the joint probability of the values x and y.

$\mathcal{N}_1(x;\mu_1,\sigma_1)$**:** The probability density function of the first Gaussian distribution evaluated at x.

μ_1**:** The mean of the first Gaussian distribution.

σ_1**:** The standard deviation of the first Gaussian distribution.

$\mathcal{N}_2(y;\mu_2,\sigma_2)$**:** The probability density function of the second Gaussian distribution evaluated at y.

μ_2**:** The mean of the second Gaussian distribution.

σ_2**:** The standard deviation of the second Gaussian distribution.

We then normalise these joint probabilities, Eq. 7.5, ensuring the probabilities sum up to 1.

$$\text{normalized } P(x,y) = \frac{P(x,y)}{\sum_{\forall(x,y)} P(x,y)} \tag{7.5}$$

In our case, $P(x,y) = P(s_1,s_2)$, as we only consider model errors on s_1 & s_2. We have successfully constructed the new P matrix from the above-defined procedure. We now use this new matrix in the Bellman update rule 7.3. We are solving the same equation as we did for the deterministic case, but previously we didn't need to sum up our expected costs over all possible future states as only one future state was possible. Now number of possible future states is 225, so a sum of all the future costs

and probabilities needs to be considered. Adding 225 possible states per action significantly increases the number of calculations, potentially amplifying the computational cost by a factor of approximately 225. To tackle the problem, we alter the convergence threshold ϵ and the discount factor γ. In the deterministic case, we used a convergence threshold of $1x10^{-}5$, whereas in this case, we used a convergence threshold of 1. Similarly, in this case, we use a discount factor γ of 0.9 compared to 0.95 used earlier.

We now present the results of the simulations comparing the deterministic model-based controller and the non-deterministic model-based controller. To ensure a fair comparison, we compute a new deterministic model-based controller with the same convergence threshold and discount factor used for the non-deterministic one. We also present the results of the deterministic controller with the original convergence threshold and discount factor.

7.1.1. Simulation Results

In table 7.1, we present the average and standard deviation of total costs over 100 iterations of running the scenarios present under Run-1's data. We compare the three controller tables as described above. On comparison of performance between the determinitic and non-deterministic controller with the same γ and ϵ, it is seen that in all the scenarios the determinitic controller incurs lower average cost. Comparing standard deviation it is seen that performance between the non-deterministic controller and the deterministic controller is very close. Therefore, it can be concluded that consideration of the model uncertainities in the DP algorithm does not improve the performance of the controller table in our case. While accounting for uncertainities adds nothing in terms of performance, it adds a lot of expense in terms of the computational time as the non-deterministic model based controller requires in excess of 7200 seconds to be computed, the deterministic model based controller requires approximately 50 seconds.

Since we also compared the performance of the original deterministic controller table, an interesting result is obtained. For all the scenarios simulated it is seen that both the deterministic controllers with different γ and ϵ, give exactly same performance. For this same performance the original control strategy costs nearly 250 seconds of computational time whereas the determinitic control with a higher ϵ and lower γ costs 50 seconds. Since there is no performance gain it is better to use the controller which is computationally less expensive.

Table 7.1: Non-Determintic vs Deterministic Model based controller

Data Set No.	**Non-Det. Controller** $\gamma = 0.9\ \epsilon = 1$		**Det. Controller** $\gamma = 0.9\ \epsilon = 1$		**Det. Controller** $\gamma = 0.95\ \epsilon = 10^{-}5$	
	Mean	**S.D.**	**Mean**	**S.D.**	**Mean**	**S.D.**
1	9575	2832	**9556**	2834	9556	2834
2	1272	420	**1258**	418	1258	418
3	3071	788	**3036**	778	3036	778
4	5765	1067	**5677**	1072	5677	1072
5	6959	1551	**6951**	1552	6951	1552
6	3362	914	**2609**	720	2609	720
7	3343	768	**3320**	774	3320	774
8	3283	905	**3271**	902	9271	902
9	254	236	**245**	224	245	224
10	1080	420	**1073**	414	1073	414
11	1556	490	**1551**	503	1551	503

7.2. Controllers based on different grid sizes

In Table 3.1, we defined the size of our discretised state space grid. We selected a coarse grid to keep the computational costs in check. Due to the selection of such a coarse grid, we sacrifice the performance of our control strategy. This section analyses the trade-off between coarse and fine grids regarding performance and computational costs. We define four controllers based on the deterministic model, using different grid sizes as defined in Table 7.2. This table also mentions the computational

time required to compute these controller tables. We then simulate run's on the HIOM$^+$ model and compare the performances of these grid sizes based on the running-cost function R.

Table 7.2: Grid Sizes & Computational Time for different controllers

States	Grid Sizes			
	Ultra-small Grid	Small Grid	Default Grid	Large Grid
SiO_2	5	8	15	38
CaO	5	13	25	63
Ore	5	5	10	25
Total	**125**	**520**	**3750**	**59850**
Avg. Computational Time (seconds)	**50**	**150**	**300**	**8000**

7.2.1. Simulation Results

Based on the grid sizes defined above, we calculate the controllers and simulate the run-1's data on HIOM. In table 7.3, we present the mean and standard deviation of costs over 100 runs on the HIOM$^+$ model. It is seen that for all of the cases the default grid size based controller outperforms the controllers based on coarser grids. In most of the cases simulated it is also seen that the standard deviation for the costs is also higher for the coarser grids. For the coarser grids the computational time is lower as compared to the default grid's computational time, but the performance is comparatively poor.

For the default discretisation the required computational time is approximately 300 seconds which is not a lot. Therefore, we create a controller based on a finer discretisation. The average costs for this controller are found to be lesser than the average costs for the default controller. The difference between these two controllers, in average costs and in standard deviation is not that high. The performance gain is not that significant when compared to the difference in computation time for the two controllers, Table 7.3. Therefore, using a finer grid would increase the performance of our control startegy as in doing so we reduce the approximation we need to carry out through interpolation. While this results in better performance, the computaional costs incurred are also much higher. Fig. 7.1 shows the computational time, and mean and standard deviation of costs for simulation on data 1, for the different grid sizes. We need to find the balance between performance and computational time and the selected default grid provides the best balance.

Table 7.3: Mean and Standard Deviations of total costs for different controllers

Data Set Number	Ultra Coarse Grid		Coarse Grid		Default Grid		Fine Grid	
	Mean	S.D.	Mean	S.D.	Mean	S.D.	Mean	S.D.
1	28 351	4930	11 002	3741	10 032	3812	**10021**	3812
2	22 210	3927	2738	1065	**1836**	738	1837	738
3	53 097	6872	6275	1538	4360	1179	**4351**	1174
4	97 448	8404	11 808	2104	7641	1399	**7597**	1386
5	88 865	7107	10 349	1932	7208	1468	**7198**	1458
6	39 850	5257	6984	2537	**3287**	945	3299	960
7	55 615	5482	6524	1574	4423	1021	**4405**	1013
8	52 667	5843	6051	1618	4111	1173	**4103**	1168
9	2408	1248	339	408	298	332	**298**	331
10	16 700	3398	1946	866	1526	773	**1525**	769
11	24 504	4220	3002	991	2141	665	**2133**	660

7.3. Decision Trees based Control

The main aim of this thesis work is to automate the lime input in the HIsarna plant. We have not been able to verify the performance of the developed control strategy on the actual plant due to time constraints and other issues with the plant run. Before the lime is automated completely, the developed control strategy should be implemented as advice to the human operator to prove the effectiveness of the control strategy. Currently, the output of the developed controller is in the form of a look-up-table

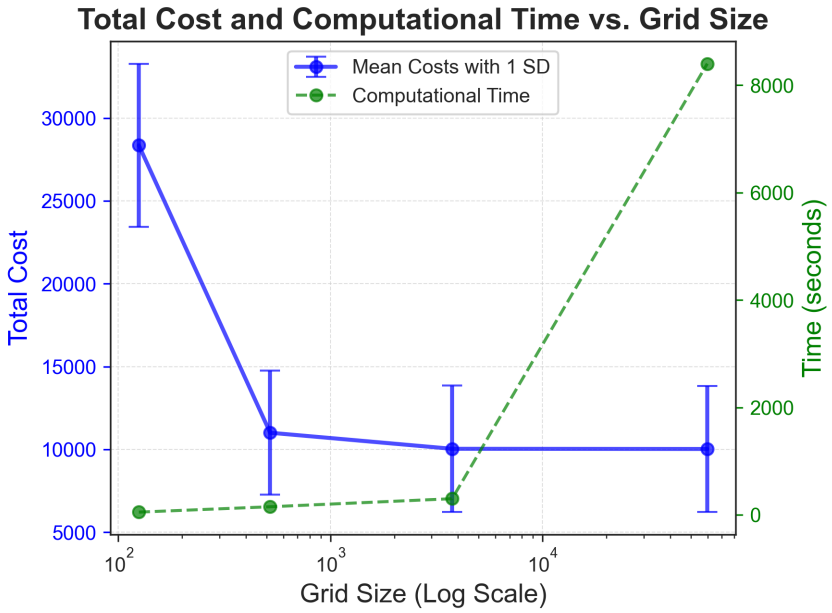

Figure 7.1: Running Costs & Computational Time vs Grid Size for Run-1 Data-1

or a .csv file which is not convenient to read.

We employed decision trees as an instrument in our analysis, primarily to translate the controller table into a format that's both visually comprehensible and human-readable. Decision trees, at their core, function by segmenting the data space into distinct regions. This segmentation is achieved using a series of decision rules, where each rule is based on a simple comparison of one of the state values. As we traverse down the tree, these rules become more specific, allowing for finer categorizations.

In our context, the decision tree methodically analyzes the state values from the controller table. Depending on the outcomes of these analyses, the tree delineates paths leading to specific recommendations. For our application, this translates to suggesting an optimal input value rate for lime. One of the standout benefits of using decision trees is the transparency it offers. Unlike many other machine learning models which operate as 'black boxes', decision trees provide a clear and logical sequence of decisions. This characteristic aids in interpretability.

Furthermore, by representing the controller table through decision trees, we aim to bridge the gap between complex numerical data and actionable insights. This bridge ensures that the derived knowledge is not confined to those with a deep technical understanding but is accessible to a wider audience, promoting informed decision-making.

We use our obtained optimal controller table to train the decision tree model. Decision trees work by making binary decisions based on features. In our case features are our states. Each internal node of the tree evaluates a condition on one of the states, and depending on the outcome, the path traverses to the left or right node. The tree was constrained to a maximum depth of 4 to ensure simplicity and better interpretability. This depth fixes the maximum depth between the intial node and the final node, i.e, in any scenario a maximum of 4 binary decisions will give the optimal lime value at that state. The decision at each node is made to minimize a certain cost function.

In the case of regression trees, like the one used in our implementation, the cost function is typically the total variance across the target values within a node. The goal is to find the state and the threshold that will result in the biggest reduction in variance. Given a set $\mathcal{D}$ of training samples, the variance $\text{Var}(\mathcal{D})$ is defined as:

$$\text{Var}(\mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} (y_i - \bar{y})^2 \tag{7.6}$$

where y_i is the target value of the i^{th} sample and $\bar{y}$ is the average target value of the samples in $\mathcal{D}$.

When a split is considered at node t on state s with threshold θ, the dataset is divided into two subsets $\mathcal{D}_{left}$ and $\mathcal{D}_{right}$:

$$\mathcal{D}_{left} = \{s|s \leq \theta\} \quad \text{and} \quad \mathcal{D}_{right} = \{s|s > \theta\} \tag{7.7}$$

The variance reduction resulting from the split, also called the cost reduction, is calculated as:

$$\Delta\text{Var}(t) = \text{Var}(\mathcal{D}) - \left(\frac{|\mathcal{D}_{left}|}{|\mathcal{D}|}\text{Var}(\mathcal{D}_{left}) + \frac{|\mathcal{D}_{right}|}{|\mathcal{D}|}\text{Var}(\mathcal{D}_{right})\right) \tag{7.8}$$

The decision tree algorithm will evaluate many possible splits and choose the one that maximizes the variance reduction. This iterative process ensures that the constructed tree is optimal with respect to the chosen cost function.

The decision tree implementation efficiently translates the controller table into a structured and intuitive model. Fig. 7.2 shows a section of the trained decision tree. Here the actual values have been hidden to protect the IP of Tata Steel. In each node we can see the state value being compared, for simiplicity we divide the CaO mass by SiO_2 mass, so that we can have a direct comparison for the current system basicity. We also show the total squared error and the number of samples for each node. The final nodes give the optimal lime value for any state pair. We simulate and compare this decision tree based control to our optimal controller table

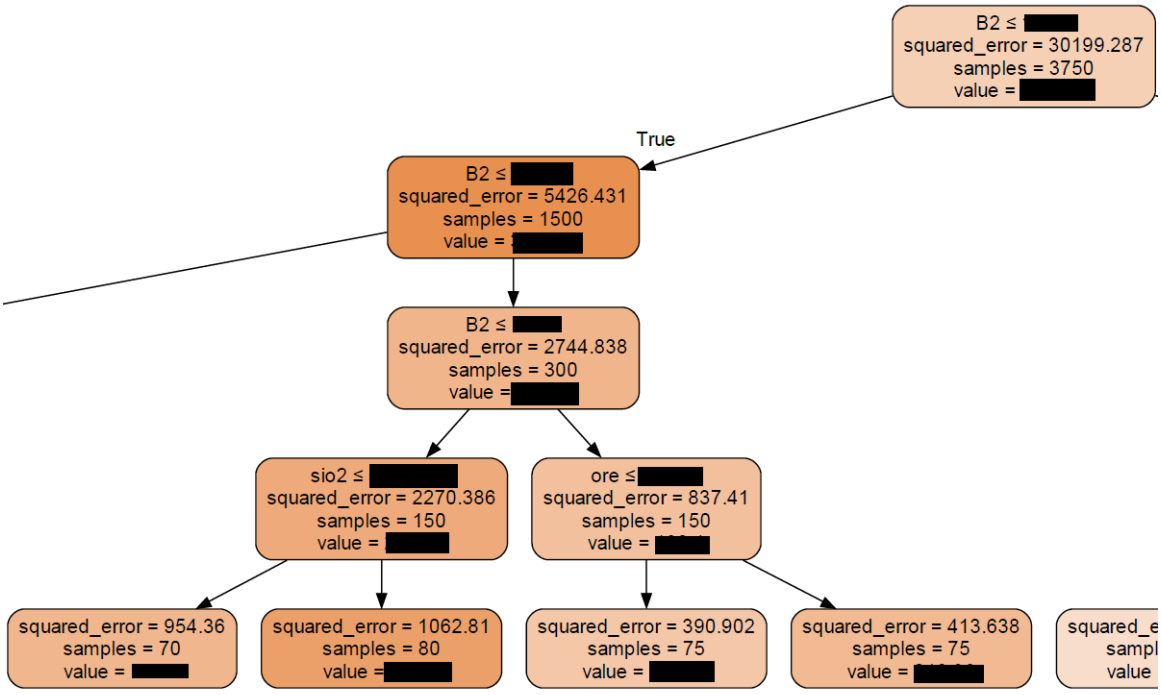

Figure 7.2: A section of the trained Decision Tree

7.3.1. Simulation Results

Fig. 7.3 & 7.4 show the plots for B2 values over time using the optimal controller table and the decision tree-based control respectively. It can be seen that the optimal controller is able to track the reference B2 value more accurately than the decision tree based control.

Table 7.4 shows the average and standard deviation of total costs calculated in the same way as done in the previous sections. For all the scenarios simulated we see that the optimal control rule performs better in terms of mean costs, and this is expected as the optimal controller provides a finer control law. However, the costs and standard deviations do not differ a lot between the controller table-based

simulation and the decision tree-based simulation. This shows that even abstraction or simplification of the optimal controller table shows promising results on the HIOM$^+$ model.

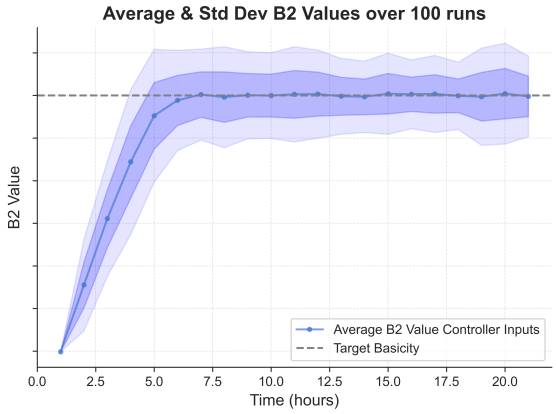

Figure 7.3: Run-1 Dataset-1 Controller Table Simulation

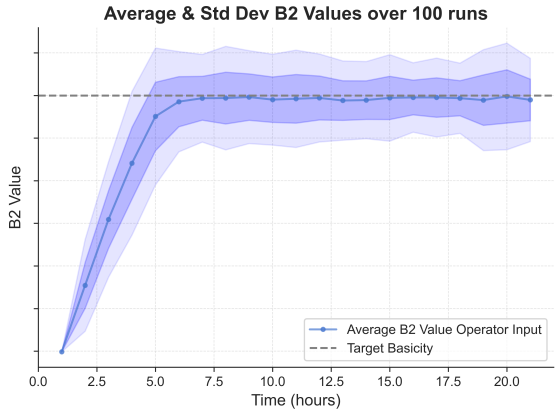

Figure 7.4: Run-1 Dataset-1 Decision Tree Simulation

Table 7.4: Performance Comparison Decision Tree vs Optimal Controller

Data Set Number	Decision Tree		Optimal Controller	
	Average Cost	Standard Deviation	Average Cost	Standard Deviation
1	9831	2984	**9653**	2935
2	1404	495	**1304**	459
3	3245	811	**2989**	784
4	6108	1170	**5656**	1078
5	5550	1039	**5075**	916
6	2854	1116	**2516**	723
7	3872	895	**3567**	816
8	3642	984	**3392**	882
9	198	165	**192**	165
10	1147	442	**1068**	391
11	1693	512	**1536**	525

7.4. Conclusion

In this chapter we considered and evaluated three alternate control techniques to the one proposed in the previous chapter. For the first technique we considered model errors for controller synthesis and we made some alternate design choices to have a reasonable computational time. Although the error based controller was not better than the deterministic model based controller, we found that an alternate design choice in terms of discount factor, γ and discount factor for the deterministic controller leads to a reduced computational time while having the same performance in terms of running cost.

In the second technique, we analysed coarse and finer discretisation of the state space for our controller. As expected the coarser discretisation based controller's performance was worse and it didn't improve much in terms of computational time. Similarly for the finer grid, the performance improved in terms of reduction of running costs, but the computational time increase for this improvement was too high. It can be concluded that the current discretisation offers a balance between the performance of the controller and the computaional time required to compute the controller table.

Finally, we propose the use of decision trees to make the obtained controller table simpler so that it is interpretable by a wider audience. We made of decision trees to classify the output of the controller table as simple state dependent binary decision which makes the control law human interpretable. This is an abstraction of the controller table. On simulation using this controller table it is seen that the performance of the decision tree is comparable to that of the optimal controller table. Therefore, before

complete automation of the lime-input is done, decison tree can be used as a technique to verify the potential of the developed control startegy on the actual HIsarna plant.

From the above alternate techniques, we have a better idea about the selection of design choices which will lead to a reduced computational time. We also were able to verify that the design choices made ensure balance between computational time and controller performance. We have also proposed a potential path to investigate the potential of the developed control strategy using decision trees to give advice to the human-operator. If decison tree based control performs well on the HIsarna plant, the complete automation using the optimal controller table should perform even better.

8

Conclusion

HIsarna represents a pivotal technology in the production of green steel. For it to be widely adopted in industrial-scale steel production, its reliability and capacity for sustained long-duration operation must be demonstrated. A critical aspect of this is controlling the chemical composition of the slag mixture within the smelting furnace. Presently, control methodologies rely on affine calculations based control law, which have proven suboptimal. The work in this thesis introduces a dynamic programming-based optimal strategy for controlling the plant's basicity. Simulations, based on the HIOM simulator, reveal the merits of this optimal control strategy, underscoring its potential in enhancing slag-basicity control. This, in turn, has the potential to improve the reliability of the process, paving the way for uninterrupted steel production.

Our approach hinges on the use of the HIOM simulator, which we treat as a black box with immutable hyperparameters, to develop our deterministic MDP model. Employing this MDP model, we advocate applying the Bellman optimality principle to navigate the challenges of infinite horizon decision-making using the Value-Iteration algorithm. Recognizing the complex nature of the system, certain design choices are made in parameter selection. This methodology enables the creation of a look-up table pairing states with their optimal actions. Subsequent simulations on HIOM model compare our optimal control strategy and the prevailing affine control rule. From the simulation results, we see that the control strategy we developed performs significantly better than the current control method on the deterministic model.

Given the intricacies of the HIsarna process, we acknowledge that the HIOM process model won't be perfectly accurate due to inherent uncertainties. To better gauge the performance of our developed control strategy in a more realistic scenario, we introduced uncertainties into the model predictions. This modified version is referred to as HIOM$^+$. When comparing the optimal control strategy and the affine control law using the HIOM$^+$ model, the optimal control strategy consistently outperforms across varied hyperparameters, simulation durations, and initial basicity values.

Building upon our initial control strategy, we put forth three additional control strategies, each serving as an extension or abstraction of the optimal control strategy.

Firstly, instead of relying on the HIOM model, we crafted a control strategy that employs the HIOM$^+$ model to formulate the optimal control table. By doing so, we integrate the stochastic elements of the plant model directly into our control law. When we compare the performance of this updated control table with its predecessor on the HIOM$^+$ model, a surprising finding emerges: the controller rooted in the deterministic model exhibits superior performance across various simulated scenarios.

One potential explanation lies in the probable overestimation of the model error. The absence of ample data prevented us from thoroughly evaluating the HIOM model. Consequently, based on discussions, we approximated model uncertainties to be around 20%. A comparison of the simulation results from HIOM$^+$ and actual recorded data suggests we might have overshot our model error estimates. This overestimation has ripple effects, such as the future costs being averaged over an overly broad error

set, ultimately culminating in a less-than-optimal control rule.

In our analysis, we also stumbled upon an intriguing observation concerning the design parameters for the controller based on the deterministic model. Specifically, we found that a reduced discount factor, γ, coupled with a heightened threshold, ϵ, offers equivalent performance in terms of slag-basicity control while being more computationally efficient. This phenomenon can be rationalised by understanding that a minute deviation in the lime-input rate might not translate to a discernible change in the furnace's basicity, determined based on the masses of SiO_2 and CaO. Consequently, an ultra-precise control law doesn't necessarily yield a noticeable enhancement in the controller's overall efficacy.

This insight became even more apparent when examining the second alternative control approach. In this exploration, we aimed to discern the impact of discretisation levels on the efficacy of the control strategy. The challenge in selecting a discretisation level lies in finding the right balance between performance optimisation and computational cost. Consequently, our starting point was a coarser grid. Subsequently, we employed finer and coarser discretisation scales to assess their implications on performance and computational demands. Notably, in all scenarios simulated, our default discretisation proved superior to the coarser grids. While the finer discretisation did exhibit enhanced performance in numerous scenarios, the marginal performance boost did not justify the associated surge in computational costs. This further substantiates that an exceedingly detailed control law might not substantially increase the controller's overall proficiency.

We employed decision trees to make the developed optimal control law more understandable to humans. These trees can translate the optimal controller table into a series of straightforward logical decisions, which are intuitively easier to execute. Simulations were used to evaluate the efficacy of the decision tree-derived control strategy. As observed, the performance of this strategy, while not surpassing, closely mirrored that of the optimal controller table. Such an outcome was anticipated, given that we were essentially leveraging a simplified representation of the optimal control strategy, making it inherently suboptimal. These results further underline that an abstracted strategy can deliver competent results. Aiming for a solution with even greater precision might not significantly amplify the overall performance.

In summary, we have presented an optimal control strategy for the slag-basicity control challenge, exhibiting promising potential in simulation tests. A deeper dive into the strategy allowed us to discern and clarify the key design choices concerning discretisation, the discount factor, and the convergence threshold. We introduce decision trees to bridge the gap between the prevalent affine control law and a fully automated lime input. These can serve as guides for human operators. Should the decision tree-based control demonstrate effective performance in the HIsarna plant, it would pave the way for total automation of lime input, grounded in the optimal control law.

An added stipulation for the optimal controller we developed is the necessity for frequent state feedback, which is currently available only at sporadic intervals. To counter this, we suggest running the HIOM simulator concurrently when implementing the controller on the actual HIsarna plant, aiming to estimate state evolution. Conversations with Tata Steel have confirmed that the HIOM is already being run in tandem to predict the outputs of HIsarna, implying that meeting this requirement should be straightforward.

There's also merit in fully exploring the capabilities of the control strategy grounded in the non-deterministic $HIOM^+$ model. Such an exploration mandates a comprehensive assessment of the HIOM model. Based on the outcomes of this assessment, we can determine the model's errors. A more accurate depiction of these errors could potentially enhance controller performance. If investigations reveal that the non-deterministic model-based controller yields a markedly improved performance, strategies like parallel processing can be employed to curb computational costs.

A

Appendix

Listing A.1: Code for Discretization

```
import numpy as np


ore_points = np.round(np.linspace(??, ??, 10), 2)
sio2_points = np.linspace(??, ??, 15)
cao_points = np.linspace(??, ??, 25)
lime_points = np.round(np.linspace(??, ??, 20), 2)
```

Listing A.2: Transition Probability Matrix Creation

```
def parse_data(xml_file, ratio_constant, exec_path):
    with open(xml_file, 'rb') as f:
        xml_template = f.read()

    # Parse the XML template
    root = etree.fromstring(xml_template)

    # Coal points calculated based on the ratio constant
    coal_points = ore_points / (ratio_constant * 2)

    # Define the dictionary to store the data
    data_dict = {}

    for i in range(len(ore_points)):
        for v235_value in v235_points:
            # Update rate values based on the injection points
            for material in root.findall('.//Material'):
                update_rate_value(material, "V212", coal_points[i])
                update_rate_value(material, "V222", coal_points[i])
                update_rate_value(material, "V140", ore_points[i])
                update_rate_value(material, "V235", v235_value)

            # Create a new XML file with the updated rate values
            file_name = f'file_C_{coal_points[i]*2:.0f}_O_{ore_points[i]:.0f}
                _V235_{v235_value:.0f}.xml'
            file_path = os.path.join(output_dir, file_name)
            with open(file_path, 'wb') as f:
                f.write(etree.tostring(root, pretty_print=True, encoding='utf-8',
                    xml_declaration=True))

            # Run the executable on the newly created file
```

```
            subprocess.run([exec_path, '/f', file_path])


            # Parse the processed XML file
            tree = etree.parse(file_path)
            root = tree.getroot()

            # Extract the relevant data from the XML file
            v140_rate = float(root.find(".//Material[InjectionPoint='V140']/Rate")
                .attrib['value'])
            v235_rate = float(root.find(".//Material[InjectionPoint='V235']/Rate")
                .attrib['value'])

            slag_elem = root.find(".//Slag")
            # Find the <Component> elements with name='SiO2' and 'CaO'
            for component_elem in slag_elem.findall(".//Component"):
                if component_elem.attrib['name'] == 'SiO2':
                    sio2 = float(component_elem.attrib['value'])
                elif component_elem.attrib['name'] == 'CaO':
                    cao = float(component_elem.attrib['value'])

            # Add the data to the dictionary
            data_dict[(v140_rate, v235_rate)] = (sio2, cao)

    # Post-processing of data_dict to create pred_states
    new_dict = {(round(k[0], 2), round(k[1], 2)): v for k, v in data_dict.items()}


    pred_states = {}
    for i in sio2_points:
        for j in cao_points:
            for k in ore_points:
                for l in v235_points:
                    sio2_mass = i + new_dict[(k, l)][0]
                    cao_mass = j * i + new_dict[(k, l)][1]
                    pred_states[(i, j, k, l)] = (sio2_mass, cao_mass / sio2_mass)
    return pred_states
```

Listing A.3: Dynamic Programming

```
import numpy as np
from utils import nearest_index, trilinear_interpolation
from settings import ore_points, sio2_points, cao_points, v235_points

def perform_dynamic_programming(data_dict):
    # Define the convergence threshold
    threshold = 1
    max_iterations = 1000

    V = np.zeros((len(sio2_points), len(cao_points), len(ore_points)), dtype=float
        )
    optimal_actions_1 = np.zeros((len(sio2_points), len(cao_points), len(
        ore_points)), dtype=int)

    for iteration in range(max_iterations):
        V_prev = np.copy(V)

        for i, sio2 in enumerate(sio2_points):
            for j, cao in enumerate(cao_points):
                for k, ore in enumerate(ore_points):
                    valid_keys = [(sio2, cao, ore, lime) for lime in v235_points
```

```
                        if (sio2, cao, ore, lime) in data_dict]
                    if not valid_keys:
                        continue

                    update_value_function(V, V_prev, optimal_actions_1, i, j, k,
                        ore, valid_keys, data_dict)

        if np.all(np.abs(V_prev - V) < threshold):
            break
    print (iteration)

    return V, optimal_actions_1

def update_value_function(V, V_prev, optimal_actions_1, i, j, k, ore, valid_keys,
    data_dict):
    next_states = np.array([data_dict[key] for key in valid_keys])
    next_sio2_values, next_cao_values = next_states[:, 0], next_states[:, 1]

    # Compute costs using broadcasting
    costa = np.where(next_cao_values < ??, 400000 * (?? - next_cao_values)**2,
                    np.where(next_cao_values == ??, 0, 400000 * (?? -
                        next_cao_values)**2))

    # Estimate the value at the next state using interpolation
    next_values = np.array([trilinear_interpolation([next_sio2, next_cao, ore],
        V_prev) for next_sio2, next_cao in zip(next_sio2_values, next_cao_values)])
    costs = costa + 0.95 * next_values

    # Find the minimum cost and the corresponding action
    min_cost_idx = np.argmin(costs)
    min_cost = costs[min_cost_idx]

    # Update the value function and optimal action
    V[i, j, k] = min_cost
    optimal_actions_1[i, j, k] = valid_keys[min_cost_idx][3]
```

Listing A.4: Trilinear Interpolation

```
def trilinear_interpolation(point, V):
    x, y, z = point
    x0, y0, z0 = map(nearest_index, (sio2_points, cao_points, ore_points), point)
    x1, y1, z1 = x0 + 1, y0 + 1, z0 + 1

    # Ensure the indices are within bounds
    x1 = min(x1, len(sio2_points) - 1)
    y1 = min(y1, len(cao_points) - 1)
    z1 = min(z1, len(ore_points) - 1)

    c000 = V[x0, y0, z0]
    c001 = V[x0, y0, z1]
    c010 = V[x0, y1, z0]
    c011 = V[x0, y1, z1]
    c100 = V[x1, y0, z0]
    c101 = V[x1, y0, z1]
    c110 = V[x1, y1, z0]
    c111 = V[x1, y1, z1]

    xd = (x - sio2_points[x0]) / (sio2_points[x1] - sio2_points[x0]) if x0 != x1
        else 0
    yd = (y - cao_points[y0]) / (cao_points[y1] - cao_points[y0]) if y0 != y1 else
         0
```

```
    zd = (z - ore_points[z0]) / (ore_points[z1] - ore_points[z0]) if z0 != z1 else
         0

    c00 = c000*(1-xd) + c100*xd
    c01 = c001*(1-xd) + c101*xd
    c10 = c010*(1-xd) + c110*xd
    c11 = c011*(1-xd) + c111*xd

    c0 = c00*(1-yd) + c10*yd
    c1 = c01*(1-yd) + c11*yd

    c = c0*(1-zd) + c1*zd

    return c
```

Listing A.5: Parallel Simulation

```
def parallel_simulation(args):
    pkl_file, folder_name, initial_sio2_val, initial_b_val, initial_slag_val,
        noise_matrix_sio2, noise_matrix_cao = args
    xml_files = [os.path.join(folder_name, file) for file in os.listdir(
        folder_name) if file.endswith('.xml')]
    xml_files.sort(key=extract_index)

    # Rest of the simulation code, similar to your original run_simulation
        function
    with open(pkl_file, "rb") as f:
        A_interpolator = pickle.load(f)

    total_costs = []
    results= []
    for a in range(100):
        initial_sio2 = initial_sio2_val
        initial_b = initial_b_val
        initial_slag = initial_slag_val
        total_cost=0
        b_values = []
        b_values.append(initial_b)
        for i, xml_file in enumerate(xml_files):
            sio2 = initial_sio2
            ore_rate = read_rates_from_xml(xml_file)[3]
            if isinstance(A_interpolator, DecisionTreeRegressor):
                optimal_lime_rate = A_interpolator.predict(np.array([[sio2,
                    initial_b, ore_rate]]))[0]
            else:
                optimal_lime_rate = A_interpolator((sio2, initial_b, ore_rate))
            update_xml_file(xml_file, optimal_lime_rate)
            subprocess.run(['C:\\Users\\ritik\\AppData\\Local\\HIsarna.Operation.
                Model\\HIsarna.Operation.Model.GUI.exe', '/f', xml_file])
            sio2_rate, cao_rate, slag_rate, _ = read_rates_from_xml(xml_file)
            simulation_duration = ??
            initial_slag += slag_rate * simulation_duration
            sio2_noise = noise_matrix_sio2[i][a]
            cao_noise = noise_matrix_cao[i][a]
            initial_sio2 = sio2 + (sio2_rate + sio2_noise) * simulation_duration
            cao_new = initial_b * sio2 + (cao_rate + cao_noise) *
                simulation_duration
            sio2_conc = initial_sio2 / initial_slag
            cao_conc = cao_new / initial_slag
            initial_slag -= ??
            if initial_sio2 > ??:
```

```
                initial_slag = ??
            initial_sio2 = initial_slag * sio2_conc
            cao_new = initial_slag * cao_conc
            initial_b = next_cao_values = cao_new / initial_sio2
            b_values.append(initial_b)
             #next_cao_values
            cost = np.where(next_cao_values < ??, ?? * (?? - next_cao_values)**2,
                            np.where(next_cao_values == ??, 0, ?? * (?? -
                                next_cao_values)**2))
            total_cost += cost
        total_costs.append(total_cost)
        results.append(b_values)
        # ... [rest of your loop here] ...

    pkl_name = os.path.basename(pkl_file).split('.')[0]
    return pkl_name, total_costs, results  # Return results so they can be
        collected later

if __name__ == '__main__':

    for dataset_folder, info in datasets_info.items():
        folder_name = dataset_folder
        xml_files = [os.path.join(folder_name, file) for file in os.listdir(
            folder_name) if file.endswith('.xml')]
        xml_files.sort(key=extract_index)

        n = len(xml_files)  # number of XML files
        m = 100  # number of runs

        # Generate noise matrices once for each dataset
        noise_matrix_sio2 = [[np.random.normal(0, ??) for _ in range(m)] for _ in
            range(n)]
        noise_matrix_cao = [[np.random.normal(0, ??) for _ in range(m)] for _ in
            range(n)]

        args_list = [(pkl_files[key], folder, info["initial_sio2_value"], info["
            initial_b_value"], info["initial_slag_value"], noise_matrix_sio2,
            noise_matrix_cao)
                     for key, folder in zip(pkl_files.keys(), info["copies"])]

        with Pool(cpu_count()) as p:
            all_results = p.map(parallel_simulation, args_list)

        simulation_results = {result[0]: (result[1], result[2]) for result in
            all_results}

        # Print the results for each dataset
        for key, result in simulation_results.items():
            total_costs = result[0]
            mu = np.mean(total_costs)
            sigma = np.std(total_costs)
```

Bibliography

[1] B.-H. Yoon, K.-H. Heo, J.-S. Kim, and H.-S. Sohn, "Improvement of steel cleanliness by controlling slag composition," *Ironmaking & steelmaking*, vol. 29, no. 3, pp. 214–217, 2002.

[2] W. Cardoso, D. Barros, R. Baptista, and R. di Felice, "Mathematical modelling to control the chemical composition of blast furnace slag using artificial neural networks and empirical correlation," *IOP Conference Series: Materials Science and Engineering*, vol. 1203, no. 3, p. 032 096, Nov. 2021. DOI: 10.1088/1757-899X/1203/3/032096. [Online]. Available: https://dx.doi.org/10.1088/1757-899X/1203/3/032096.

[3] J. Muller and M. Erwee, "Blast furnace control using slag viscosities and liquidus temperatures with phase equilibria calculations," in *SOUTHERN AFRICAN PYROMETALLURGY CONFERENCE*, Southern African Institute of Mining and Metallurgy Johannesburg, 2011, pp. 309–326.

[4] B. Deo, A. Overbosch, B. Snoeijer, D. Das, and K. Srinivas, "Control of slag formation, foaming, slopping, and chaos in bof," *Transactions of the Indian Institute of Metals*, vol. 66, pp. 543–554, 2013.

[5] S.-h. Chen, M. Jiang, X.-f. He, and X.-h. Wang, "Top slag refining for inclusion composition transform control in tire cord steel," *International Journal of Minerals, Metallurgy, and Materials*, vol. 19, pp. 490–498, 2012.

[6] G. Vitanov, "Dynamic programming based basicity control of an experimental smelting furnace prototype," 2022.

[7] W. Sun, Y. Wang, F. Zhang, and Y. Zhao, "Dynamic allocation of surplus by-product gas in a steel plant by dynamic programming with a reduced state space algorithm," *Engineering Optimization*, vol. 50, no. 9, pp. 1578–1592, 2018.

[8] J. Zhao, T. Wang, W. Pedrycz, and W. Wang, "Granular prediction and dynamic scheduling based on adaptive dynamic programming for the blast furnace gas system," *IEEE Transactions on Cybernetics*, vol. 51, no. 4, pp. 2201–2214, 2019.

[9] T. Wang, L. Wang, J. Zhao, W. Wang, and Y. Liu, "Evolutionary adaptive dynamic programming algorithm for converter gas scheduling of steel industry," in *2017 6th International Symposium on Advanced Control of Industrial Processes (AdCONIP)*, 2017, pp. 137–142. DOI: 10.1109/ADCONIP.2017.7983769.

[10] L. Tang and H. Ren, "Modelling and a segmented dynamic programming-based heuristic approach for the slab stack shuffling problem," *Computers Operations Research*, vol. 37, no. 2, pp. 368–375, 2010, ISSN: 0305-0548. DOI: https://doi.org/10.1016/j.cor.2009.05.011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0305054809001531.

[11] R. Zhao and L. Tang, "Integer programming model and dynamic programming based heuristic algorithm for the heavy plate shuffling problem in the iron and steel industry," in *2010 International Conference on Logistics Systems and Intelligent Management (ICLSIM)*, vol. 3, 2010, pp. 1381–1385. DOI: 10.1109/ICLSIM.2010.5461192.

[12] Y. Zhang, Q. Guo, and L. Tang, "On-line energy allocation based on approximate dynamic programming for iron and steel industry," *ISIJ International*, vol. 56, no. 12, pp. 2214–2223, 2016.

[13] L. Tang, G. Wang, and J. Liu, "A branch-and-price algorithm to solve the molten iron allocation problem in iron and steel industry," *Computers & Operations Research*, vol. 34, no. 10, pp. 3001–3015, 2007.

[14] M. S. Santos and J. Vigo-Aguiar, "Analysis of a numerical dynamic programming algorithm applied to economic models," *Econometrica*, pp. 409–426, 1998.

[15] C. Shoemaker, “Applications of dynamic programming and other optimization methods in pest management,” *IEEE Transactions on Automatic Control*, vol. 26, no. 5, pp. 1125–1132, 1981.

[16] J. R. Stedinger, B. F. Sule, and D. P. Loucks, “Stochastic dynamic programming models for reservoir operation optimization,” *Water resources research*, vol. 20, no. 11, pp. 1499–1505, 1984.

[17] H. Ko, T. Koseki, and M. Miyatake, “Application of dynamic programming to the optimization of the running profile of a train,” *WIT Transactions on The Built Environment*, vol. 74, 2004.

[18] B. Favre and B. Peuportier, “Application of dynamic programming to study load shifting in buildings,” *Energy and Buildings*, vol. 82, pp. 57–64, 2014.

[19] L. P. Martin, *Markov Decision Processes: Discrete Stochastic Dynamic Programming* (Wiley Series in Probability and Statistics). John Wiley & Sons, 1994, ISBN: 9780470316887. [Online]. Available: `https://onlinelibrary.wiley.com/doi/book/10.1002/9780470316887`.

[20] R. Bellman, *Dynamic programming.* Princeton University Press, 1972, p. 342, ISBN: 069107951X.

[21] D. P. Bertsekas, “A new value iteration method for the average cost dynamic programming problem,” *SIAM journal on control and optimization*, vol. 36, no. 2, pp. 742–759, 1998.

[22] D. Bertsekas, “Multiagent value iteration algorithms in dynamic programming and reinforcement learning,” *Results in Control and Optimization*, vol. 1, p. 100 003, 2020.

[23] T. Bian and Z.-P. Jiang, “Value iteration and adaptive dynamic programming for data-driven adaptive optimal control design,” *Automatica*, vol. 71, pp. 348–360, 2016.

[24] Q. Wei, F. L. Lewis, D. Liu, R. Song, and H. Lin, “Discrete-time local value iteration adaptive dynamic programming: Convergence analysis,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 48, no. 6, pp. 875–891, 2016.

[25] J. N. Tsitsiklis and B. Van Roy, “Feature-based methods for large scale dynamic programming,” *Machine Learning*, vol. 22, no. 1-3, pp. 59–94, 1996.

[26] I. Menache, S. Mannor, and N. Shimkin, “Basis function adaptation in temporal difference reinforcement learning,” *Annals of Operations Research*, vol. 134, no. 1, pp. 215–238, 2005.

[27] S. A. Johnson, J. R. Stedinger, C. A. Shoemaker, Y. Li, and J. A. Tejada-Guibert, “Numerical solution of continuous-state dynamic programs using linear and spline interpolation,” *Operations Research*, vol. 41, no. 3, pp. 484–500, 1993.

[28] L. Busoniu, D. Ernst, B. De Schutter, and R. Babuska, “Consistency of fuzzy model-based reinforcement learning,” in *2008 IEEE International Conference on Fuzzy Systems (IEEE World Congress on Computational Intelligence)*, IEEE, 2008, pp. 518–524.