Implementation of PID Controller using Deep Reinforcement Learning in Edge computing node for Industry 4.0

Lavish Thomas

M.Sc. in Computing
Big Data Analytics
and Artificial
Intelligence

2020

Computing Department, Letterkenny Institute of Technology, Port Road,
Letterkenny, Co. Donegal, Ireland.

Implementation of PID Controller using Deep Reinforcement Learning in Edge computing node for Industry 4.0

Author: Lavish Thomas

Supervised by: Dr. Eoghan Furey

A thesis submitted in partial fulfilment of the requirements for the
Master of Science in Computing in Big Data Analytics and Artificial Intelligence

Submitted to Quality and Qualifications Ireland (QQI)
Dearbhú Cáilíochta agus Cáilíochtaí Éireann January 2020

Declaration

I hereby certify that the material, which I now submit for assessment on the programmes of study leading to the award of Master of Science in Computing in L00150445, is entirely my own work and has not been taken from the work of others except to the extent that such work has been cited and acknowledged within the text of my own work. No portion of the work contained in this thesis has been submitted in support of an application for another degree or qualification to this or any other institution. I understand that it is my responsibility to ensure that I have adhered to LYIT's rules and regulations.

I hereby certify that the material on which I have relied on for the purpose of my assessment is not deemed as personal data under the GDPR Regulations. Personal data is any data from living people that can be identified. Any personal data used for the purpose of my assessment has been pseudonymised and the data set and identifiers are not held by LYIT. Alternatively, personal data has been anonymised in line with the Data Protection Commissioners Guidelines on Anonymisation.

I consent that my work will be held for the purposes of educational assistance to future students and will be shared on the LYIT Computing website (www.lyitcomputing.com) and Research THEA website (https://research.thea.ie/). I understand that documents once uploaded onto the website can be viewed throughout the world and not just in Ireland. Consent can be withdrawn for the publishing of material online by emailing Thomas Dowling; Head of Department at thomas.dowling@lyit.ie to remove items from the LYIT Computing website and by email emailing Denise McCaul; Systems Librarian at denise.mccaul@lyit.ie to remove items from the Research THEA website. The material will continue to appear in printed formats once published, and as websites are public medium, LYIT cannot guarantee that the material has not been saved or downloaded.

LAVISH THOMAS

Signature of Candidate   Date : 5th Jan 2021

Acknowledgements

I would like to thank everyone who helped me in this journey of the master degree. To start with, I thank the Letterkenny Institute of Technology for giving admission for the course of Big Data Analytics and Artificial Intelligence. I would also like to thank Thomas Dowling, Head of Computing Department for providing all the support and guidance for completion of MSc degree.

I thank Dr. Eoghan Furey for the supervision of this dissertation. I would also like to thank my lecturers throughout the course: Dr. Shagufta Henna, Dr. James Connolly, Dr. Nigel McKelvey, Ms. Karen Quinn, Dr. Kevin Meehan, Lect. Edwina Sweeney and Dr. Mark Leeney; for the knowledge I gained from this course.

Finally, I would like to thank my classmates, friends and family for supporting me throughout my journey.

Abstract

The Automation Industry is one of the core benefactors of Artificial Intelligence applications. Deep Reinforcement learning is the most promising frontier in achieving the human-level controls in the physical world. This research is carried out to evaluate the maturity level of the deep reinforcement learning framework and agents for the implementation of control blocks in the industrial automation domain. As the edge computing technologies have advanced, the project focuses on the robustness, adaptability and portability of the solution to edge computing devices. As part of this dissertation, two new types of environment are developed based on OpenAI Gym which is used to test the Keras-rl agents. Five agents were initially selected for the evaluation out of which three were discrete value-based and two were of continuous value-based. The continuous values were not able to converge the error rate given the parameters of the continuous environment. The discrete environment based agents were able to exhibit comparable performance with respect to the PLC-based PID controller logic blocks. This is a promising result for the implementation of AI-based edge controllers for the process automation in the industry. This will lead to more optimized performance of the automation system. The auto-tuning capability achieved by neural networks will reduce the commissioning effort of the process plants. These advantages align directly proportional to the goals of edge computing and Industry 4.0.

Acronyms

Acronym	Definition
AI	Artificial Intelligence
API	Application Programming Interface
CEM	Cross-Entropy Method
CISC	Complex Instruction Set Computer
CNN	Convolutional Neural Network
CSV	Comma Separated Values
CV	Controlled Variable
DC	Direct Current
DDPG	Deep Deterministic Policy Gradient
DL	Deep Learning
DNN	Deep Neural Network
DQN	Deep Q-Networks
DRL	Deep Reinforcement Learning
FORL	Function Optimization by Reinforcement Learning
GPU	Graphical Processing Unit
GUI	Graphical User Interface
IIOT	Industrial Internet Of Things
IO	Input/Output
IOT	Internet Of Things
JSON	JavaScript Object Notation
LTL	Linear Temporal Logic
MDP	Markov Decision Process
MQTT	Message Queuing Telemetry Transport
NAF	Normalized Advantage Function
NOC	No Objection Certificate
OOPS	Object-Oriented Programming System
OPC	Object Linking and Embedding for Process Control
OS	Operating System
OSI	Open Systems Interconnection
PI	Process Instrumentation
PID	Proportional Integral Derivative
PLC	Programmable Logic Controller
PV	Process Value
REST	Representational State Transfer
RISC	Reduced Instruction Set Computer
RL	Reinforcement Learning
SARSA	State–action–reward–state–action
SCADA	Supervisory Control And Data Acquisition
SP	Set-Point
SQL	Structured Query Language

SV	Set-Point Variable
TF	Tensor Flow
TPU	Tensor Processing Unit

Table of Contents

Table of Figures

Table of Tables

Table of Code Listings

1. Introduction

A proportional–integral–derivative (PID) controller is the core of the industrial automation systems which are currently based on mathematical calculus formulations (Likins, 2016). This research is an attempt to prove that deep reinforcement learning has high potential in the field of industrial automation. This dissertation concentrates on the implementation of the most used PLC logic block named PID using a deep reinforcement learning agent which is capable of self-tuning using experience. The benefit of this implementation is that the conventional PID blocks are time-consuming in terms of tuning requirements. It is also prone to errors as it is a human-centred process. This experiment has tried to align the trending topic of industry 4.0 and its requirements for a faster and automated process using the digitalisation. This is achieved by optimising the deep reinforcement framework for the IoT and Edge Computing arena.

1.1. Background

The latest trending topic of artificial intelligence is deep reinforcement learning, a combination of deep neural networks and reinforcement learning. A PID controller is a widely used but complicated control block used in industry automation usually implemented using PLC logic blocks. As part of industry 4.0 revolution, the companies are trying to incorporate the more cloud technologies without compromising on latency and security which have led to the new products range in edge computing for industry automation (Devices similar to Raspberry Pi). However, the implementation of deep reinforcement learning is currently limited to gaming playing and autonomous tasks. There is a considerable knowledge gap in the field of industrial automation for deep reinforcement learning. Along with the requirement for edge computing (Zou et al., 2019)., this research will be a breakthrough demonstration of artificial intelligence in the field of industrial automation.

1.2. Motivation

The deep reinforcement learning is considered as the ultimate path to achieve the human level like efficiency for machines where the two of the most successful branches of the artificial intelligence are combined, that is, deep learning and reinforcement learning (Mnih et al., 2015). Industry automation is one the early investors in the domain of artificial intelligence, but the inefficiency of the infrastructure in place before the AI-winter era had paused the work in domain much like all others domains in the AI-winter (Sharp et al., 2018). The criticality of the industrial automation domain makes it highly susceptible to a high amount of monetary loss in case of an error (Chen et al., 2018). This creates a high demand for accurate models than the typical applications of normal machine learning algorithms where errors such as false negatives do not create high losses for the stakeholders. In recent years, the increase in the edge computing technologies has greatly reduced the latency and increased the computing capabilities near the key decision-making points (Schulz et al., 2018). This has encouraged the industry to experiment with the AI-based decision-making process, which is mission-critical (Zou et al., 2019).

1.3. Purpose

The motivation behind this research is to demonstrate the implementation of AI technology in the field of industrial automation. Currently, all control systems are programmed explicitly using various mathematical functions which have to be fine-tuned after iterations of testing (Shang et al., 2013). Implementation of a reinforcement learning agent will reduce the time-lapse by auto-adjusting to the requirement. Integration of deep learning framework has increased the reinforcement learning agent's performance (Qin et al., 2018). Therefore, this project will be an attempt to create a deep reinforcement learning agent (DL + RL) which can imitate the behaviour of a PID controller.

1.4. Research Question (s)

Primary Research Question:

Can modern deep reinforcement learning frameworks be used to implement controller logic in an edge computing device for industry automation?

Secondary research questions:

1. Which type of state representation is optimal: discrete or continuous space?
2. Are the agents provided by the Keras-RL framework applicable in an industrial environment

1.5. Methodology

The research will follow an experimental approach where the inputs for various scenarios will be generated for the PID controller based on the empirically devised scenarios. The outputs will be compared to a simulated PID controller in a simulated environment. As depicted in Figure 1, this research has five stages. In the initial stage, the requirements are gathered by researching the current literature work in the domain. Afterwards, two parallel steps are taken which goes hand in hand for developing an environment which adheres to the OpenAI Gym standards. Another process component and data logging system which is used to collect data about the agent behaviour and environmental states are also created in parallel. In the third step, the agents provided by the Keras-RL is compiled to run in the previously designed environments. Subsequentially, the execution of the agents in the environment is carried out, and the data is collected using the data logging system developed in the second stage. Finally, the data is analysed in the Jupyter notebooks using various data analytics methods.

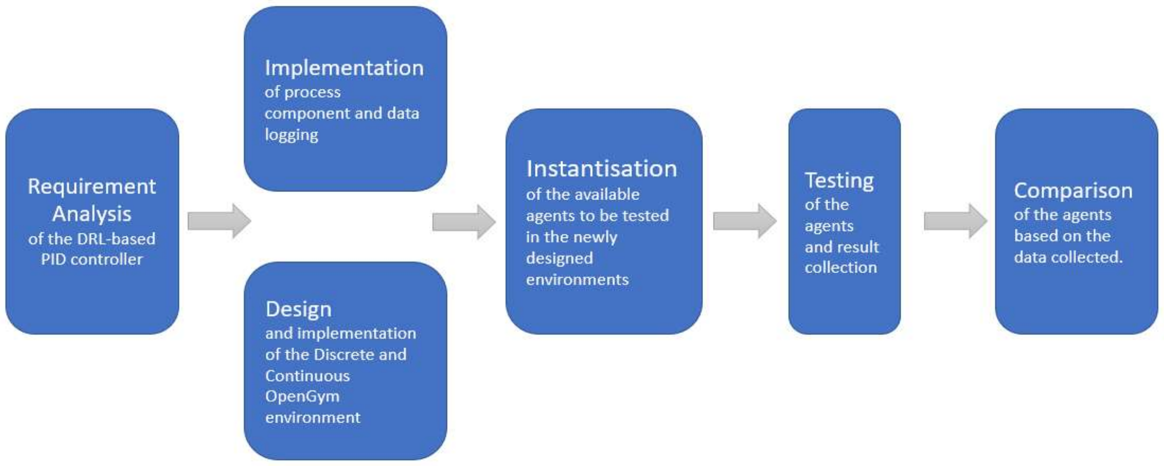
Requirement Analysis of the DRL-based PID controller
Implementation of process component and data logging
Design and implementation of the Discrete and Continuous OpenGym environment
Instantisation of the available agents to be tested in the newly designed environments
Testing of the agents and result collection
Comparison of the agents based on the data collected.

Figure 1. Methodology Stages

1.6. Research contributions

- This most significant contribution of this research is that this proves the technology has developed for the successful application of artificial intelligence in the industry.
- As far as we have researched, this is the first implementation of a deep reinforcement learning using the Keras framework to demonstrate the capability in the realization of controller logic.

- The two environments developed part of this research will be published back to the OpenAI Gym community after the successful completion of this dissertation, and NOC are cleared.
- The process artefact and data-logging system have been developed as individual components which can be published to the open-source community who would like to simulate the process behaviour in the software environments.
- The sub-research question answered by this project is an addition to the knowledge base about the agent's behaviour in the industry domain problem-statements.

1.7. Thesis Organization

This thesis is organized into six chapters of which the first chapter gives a brief introduction of the topic, afterwards which it describes the background, motivation, purpose and the research question in order. And finally, this chapter concludes with the scope and limitations.

The second chapter is aimed at giving the theoretical background on the different concepts such as deep learning, reinforcement learning, edge computing and various topics related to the design, implementation and analysis of the results. This chapter concludes with the brief description of previous attempts to implement a deep reinforcement learning agent for performing PID operation and drawback of each.

The third chapter is dedicated to the design requirements for the experiment. Initially, a system design is presented by explaining the modules interact with each other. Afterwards, the expectations from each module are detailed, after which the interaction points between the modules are defined. Additionally, the chapter also presents the expected environmental behaviours in the form of correlation equations.

The fourth chapter is used to explain the implementation of each module in detail. The rationale behind the code blocks is explained with reference to the design requirements. The chapter gives a detailed explanation of the environment implementation and compilation of the deep reinforcement learning agent. Finally, the PLC-based PID controller logic is explained, which is used to benchmark the performance of the new deep reinforcement approach proposed in this research.

In the fifth chapter, the execution of the experiment is detailed initially after which the results are evaluated. Subsequently, the results are compared and contrasted between each agent, along with its respective execution environments.

In the final chapter, the summary of the thesis in the lines of the analyses of the results is presented first. To conclude, the future direction which can be taken up in order to improve upon this approach is listed.

Chapter 2

2. Literature Review

2.1. Introduction

This chapter is aimed at reviewing the current literature work in this field which will give supporting evidence on the feasibility of this dissertation work. Initially, the core topics of the dissertation, such as Artificial Intelligence, Reinforcement Learning, Deep Learning, is discussed in detail. Further, the application arena of this dissertation, that is, Industry automation, Control systems and PLC are introduced. Additionally, the various software components used in this project are listed and described in brief. Finally, in the related works section, the previous attempts of this solution are presented in terms of application, technology stack and implementation.

2.2. Artificial Intelligence

"The science and engineering of making intelligent machines" – John McCarthy

This is the modern definition of "Artificial intelligence" by John McCarthy, who coined the term. The definitions of AI systems have been attempted by various scientists from different perspectives, as shown in Table 1, Russel et al. (2010) have defined the AI in two scopes, thought-process and the behaviour for the same. The relevance of the first scope is that some machines can be taught to mimic human behaviour, for example drilling a hole of proportional size. As this can be achieved using conventional programming techniques using the logical representation of the problem, this does not be qualified as a thinking machine process. The second perspective deals with the usability of the AI systems unless the systems which are designed to think like a human cannot perform an action in the real world will hold no value despite its cognitive capabilities. Another aspect which is needed to be considered while evaluating the AI is whether the thought process and behaviour was humane or rational. Human behaviour cannot be explained entirely with logical reasoning all the time, but rationality is defined as what is the right action with the provided knowledge—often used to achieve ideal performance. Table 1 gives some popular definition and its categorical position based on the previously discussed dimensions (Russell et al., 2010). The various categories have been tried and blended to achieve the unified goal of the AI paradigm. Firstly, Acting

Humanly is the concept stating that a program which can simulate human behaviour is considered as AI. In other words, which pass the Turning test. Thinking humanly or cognitive modelling states that, for a machine to be considered Intelligent, it should be able to think like human. This is a difficult task to exact as the cognitive science itself is not matured enough to give a clear explanation of how humans think. Thinking rationally is basically representing all knowledge and its relations in precise notations and in-theory could solve any solvable problem represented in logical notation. Finally, Acting Rationally is a notion in which an agent can perform autonomously by perceiving its environment, continuously learning, adapting and overcoming the challenges.

Table 1. Categorization of popular definitions of AI-based on based on whether the action and thought process was humane or rational (Russell et al., 2010).

Thinking Humanly "The exciting new effort to make computers think ... machines with minds, in the full and literal sense." (Haugeland, 1989) "The automation of] activities that we associate with human thinking, activities such as decision-making, problem-solving, learning ." (Bellman, 1978)	**Thinking Rationally** "The study of mental faculties through the use of computational models." (Charniak and McDermott, 1985) "The study of the computations that make it possible to perceive, reason, and act." (Winston, 1992)
Acting Humanly "The art of creating machines that perform functions that require intelligence when performed by people." (Kurzweil, 1990) "The study of how to make computers do things at which, at the moment, people are better." (Rich and Knight, 1990)	**Acting Rationally** "Computational Intelligence is the study of the design of intelligent agents." (Poole et al., 1998) "AI is concerned with intelligent behaviour in artefacts." (Nilsson, 1998)

2.3. Markov Decision Process

A Markov process is a method in which the entire problem statement can be represented as a probability distribution of possible actions on a current state and its rewards (Li, 2009). The central idea of Markov Decision Process is that the given the current state of a system, the future state is the solemn product of the current state and sequence of actions. Therefore, the information of the current state is a summation of the historical events, which is conclusive enough to predict any future state (Ortner, 2013). This can be represented mathematically as:

$$P[St+1\,|St\,] = P\,[\,St+1\,|\,S1, \ldots .St\,]$$

The foremost challenge in MDP is the representation of the real-world problems in terms of Markov process and to derive the exact relations between states in adherence to Markov property. This process will require a tedious amount of manual work and indigenous knowledge in each field of application (Lavaei et al., 2020). Other drawbacks with MDP's are that even though the most real-world problems can be represented in this form, usually the state space grows in polynomial-way for most scenarios which make the computational requirements costly (Ortner, 2013) and it is called the curse of dimensionality. Another downside to MDP is that the exploitation of the state space in terms of similarities and symmetries are limited unlike the human learning process of an environment (Li, 2009).

2.4. Reinforcement Learning

Reinforcement learning can be described as a process of imitating human learning pattern in order to achieve a specified goal (Sutton and Barto, 1998). The basics of this method are action space and reward systems attached to it. The primary goal is to increase the overall reward collected in the progression of an agent from the initial state to the goal state. This is achieved by a combination of exploitation and exploration of the state space and actions available to the agent. These parameters can be controlled by a supervisor or an environment variable based on the application domain or desired efficiency (Haesaert et al., 2017). In the case of exploration, random actions are chosen, and the rewards returns are recorded for future references. For exploitation, the experience is used in order to select a particular action such that rewards are higher in the progression to the final goal (Mnih et al., 2015).

2.5. Deep Learning

The deep learning is a subset of the field of artificial intelligence, and it is inspired by the human cognitive system. Machine learning techniques are used to create most decision-making systems which learn from the data rather than explicit programming. The deep learning has advanced in recent years surpassing most of the other machine learning algorithms for the classification problems, especially in the field of computer vision (Cha et al., 2017). The following sections will present how a neural network is created and how it achieves the data analytic capabilities.

2.5.1. Neuron

The basic building block of the deep neural network is the artificial neuron which is designed from the inspiration of the human brain cells called neurons. The neuron acts as a mathematical functional block which produces an output based on the input inputs provided. As shown in Figure 2, the multiple inputs are connected via weighted inputs, and a sum is calculated, which is then propagated through an activation function.

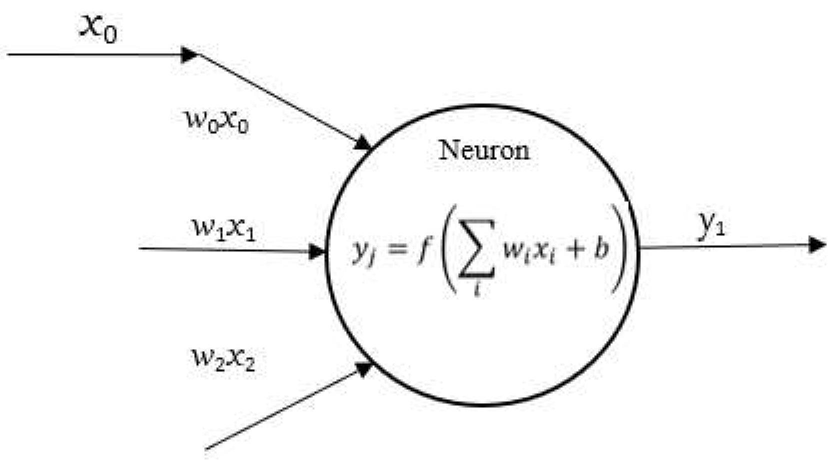

Figure 2. Mathematical representation of an artificial neuron (Zhou et al., 2019)

Based on the use case, there are several activation functions such as Sigmoid, TanH, ReLu (Araki and Nakamura, 2017). More than one type of activation function can be assigned to different layers of a neural network.

2.5.2. Activation functions

The output of the individual neurons is calculated using activation functions. This function could output discrete values like 0 and 1, or continuous values based upon the activation function used.

2.5.2.1. Binary Step function

In the binary step function, the output can assume only two, which is usually 0 or 1. The output is based on the sum of all input multiplied their respective weights which need to surpass a particular threshold value which is called the bias of the neuron (Gullì and Pal, 2017). This type of activation functions is not used widely. Mathematically it can be represented as,

$$f(x) \quad = \quad \begin{matrix} 1, & if\ x \geq threshold\ 0, \\ 0, & if\ x < threshold \end{matrix}$$

The graph of the function will be in the approximate form as plotted in Figure 2.

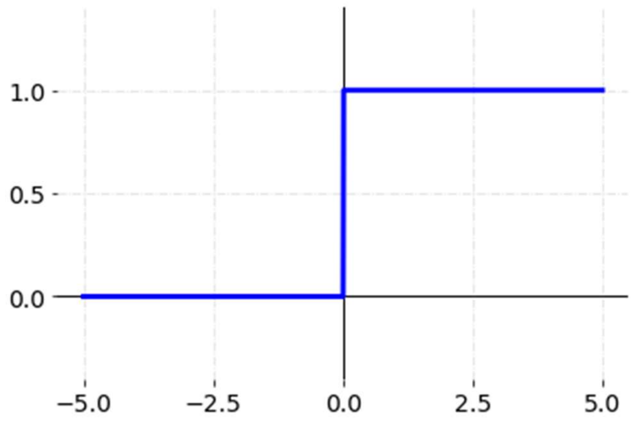

Figure 3. Binary Step Function

2.5.2.2. Linear

In linear activation function, the value is propagated linearly multiplied by a constant c which is usually 1 (Gullì and Pal, 2017). As shown in Figure 4, the function can be represented as a straight line in the Cartesian plane. Due to the constant derivate of the linear function, the error correction done in backpropagation is constant and do not depend on the input values.

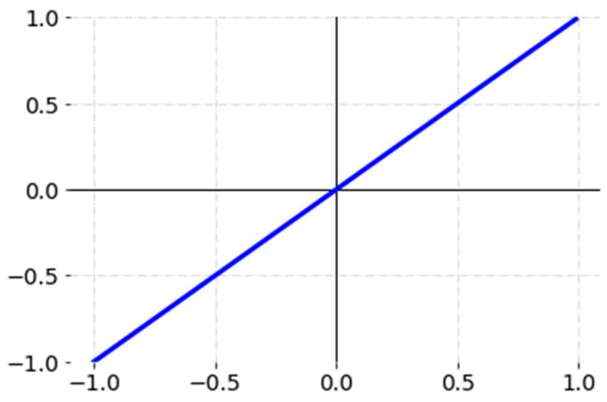

Figure 4. Linear activation function

2.5.2.3. Sigmoid

The sigmoid function was designed to have a smoother transition between values in the output side of the neuron (Gullì and Pal, 2017). However, as depicted in Figure 5, the function has a steep curve between the values -2 and 2, whereas a diminishing value transition is exhibited towards higher values. This vanishing gradient issue of the sigmoid can cause a slower learning rate if the input value range is high. The sigmoid function is represented as

$$f(x) = \frac{1}{1 + e^{-x}}$$

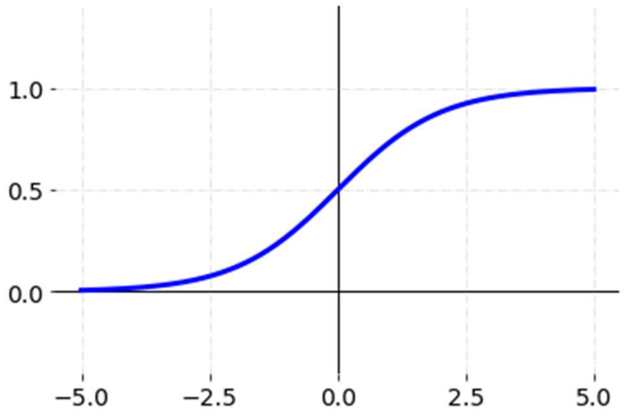

Figure 5. Sigmoid activation function

2.5.2.4. Tan H

To increase the range of higher variability of the graph, Tan H is introduced. The advantage of the tan h over sigmoid is the gradient strength (Gullì and Pal, 2017). It is a scaled version of sigmoid function as shown in the equation below

$$f(x) = \frac{2}{1 + e^{-x}} - 1$$

Or in terms of the sigmoid function

$$\tan \mathrm{h}(x) = 2\, sigmoid\ (2x) - 1$$

Figure 6 illustrates the tan h function where the output ranges between -1 to 1.

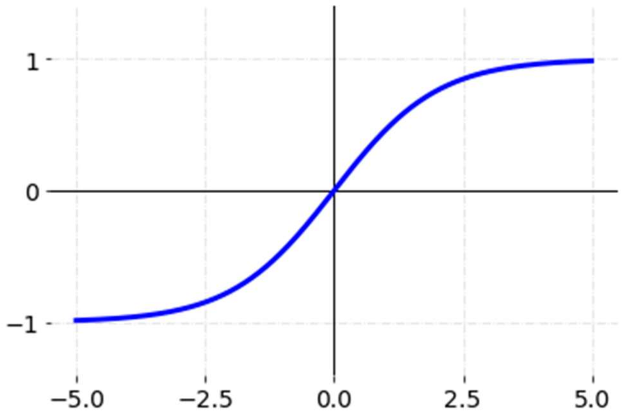

Figure 6. Tan H activation function

2.5.2.5. ReLu

ReLu is a modified version of a linear function in which the negative values are zeroed out in the output side (Gullì and Pal, 2017). This function has the same fault of the linear as it can blowout the activation function towards higher positive values. Nevertheless, the advantage over the sigmoid and tanh is that due to the chance of having output as 0 for 50% of the input range, the neural network has a lighter computational requirement. As shown in Figure 7, the positive values have a constant gradient like the linear function.

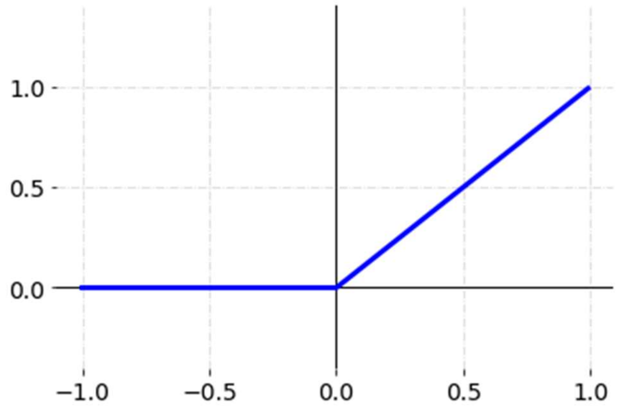

Figure 7. ReLu activation function

2.5.3. Multilayer perceptron and deep neural networks

Following the design of the human brain, the neurons are connected with each other to create a network of neurons. As shown in Figure 8, multiple neurons are arranged in layers which are interconnected via weights. In a training mode, these weights are randomly selected then are adjusted using the process of backpropagation to reduce the error, which will make the deep neural networks intelligent.

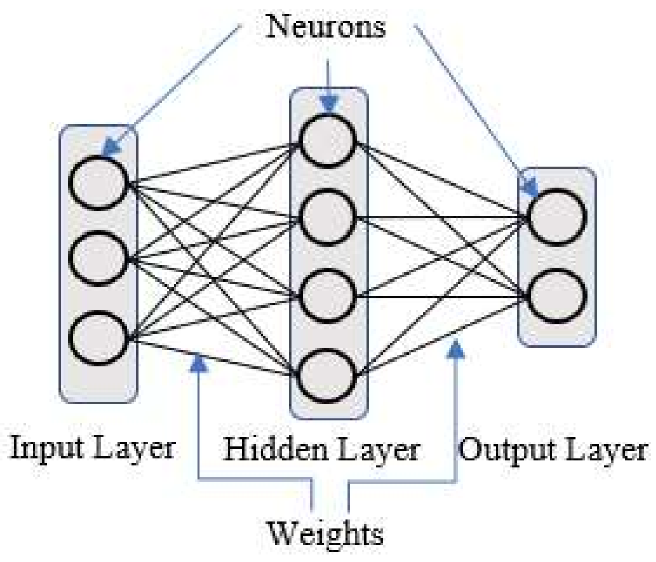

Figure 8. Multilayer Perceptron (Zhou et al., 2019)

There are mainly three layers: an input layer, which is used to feed data into the network; an output layer, it is used to extract a decision or an answer from the neural network; one or more hidden layers, which holds the information of the deep neural network training. There have been several disputes over the ethical and moral issues created by the behaviour of the hidden layers as unlike other machine learning algorithms, and the exact mathematical reasoning is not available.

2.6. Deep Reinforcement Learning

Deep learning has helped in resolving the sensor inputs to output actions pathways in a more complex intertwined dataset where features are not individually separable like in a conventional machine learning algorithms such as Support vector machines (Krizhevsky et al., 2012). This has gained popularity in resolving the real-world problems, especially in a game-playing scenario which can be used to analyse the environment which produces a continuous range of input values. This area has been combined with the reinforcement learning concepts to increase the accuracy of agent actions in the actor-critic model (Mnih et al., 2015).

There are several reinforcement agents designed with deep neural networks as the core decision-makers of the agent. Of these, five methods which are available under the Keras-rl library is discussed here.

2.6.1. Cross-Entropy Method

Cross-entropy method (CEM) is a technique which is the approximation algorithm to optimize the global cost value in the gradient plane of the deep neural network (Szita and Lörincz, 2006). The task which can be solved using CEM can be represented in the form:

$$w^{*} = arg_{w}\, max\, S(w)$$

where S is a general real-valued objective function, with an optimum value $\gamma^{*} = S(w^{*})$.
The approach of the CEM can be described as an elimination process of the plausible solutions in each iteration (Hu et al., 2019). Initially, a probability distribution of the actions and rewards are created. Then at each iteration, the gamma value is updated for each item in probability distribution as it progresses, based on the accuracy and training speed requirements items with lower gamma values can be dropped. After all the iteration is completed, the item with the most optimal gamma is selected.

2.6.2. Deep Q-Network (DQN)

A deep Q-network is a type of multi-layered neural network which can determine an action from a predefined action space based on the current state (of the agent) and parameters (observed values of the environment) associated with it. This method is a novel method presented by Mnih et al. (2015) to achieve human-level performance in stochastic environments. As per the author, animals harmonically blend their hierarchical sensory systems with the neural system to achieve optimal performance in the reinforcement learning scenarios. Therefore, by merging the reinforcement learning systems with current deep learning systems into an agent, the authors created a deep Q-network. The system is designed to learn successful policies in the-end-to-end reinforcement learning scenario by extracting information directly from sensor deployed to observe the environment. This deep Q-network has been tested in the game environment with direct pixel information and outperformed the existing algorithms, which uses an intermediate layer for the observation sampling (van Hasselt et al., 2015).

2.6.3. Deep Deterministic Policy Gradient (DDPG)

The deep deterministic policy gradient is introduced in order to extend the DQN method for reinforcement learning for continuous action space. This is an off-policy algorithm which deterministic in nature which works in two entities: the actor which learn the policy and the critic which evaluates the Q-function for each step. According to Q-learning convention, the Q-function is updated as per the Bellman equation (Lillicrap et al., 2019). As per Silver et al. (2015), the stochastic algorithms for policy gradient behave differently than the deterministic method in the integration over both action spaces and state. Therefore, the stochastic policy gradient will be resource-intensive in terms of training; an increased number of iterations will be required for convergence of the policy. In conclusion, the DDPG algorithms create a perfect candidate for continuous state-space problems, but the resource intensiveness will be a critical factor in deciding the fitness for particular applications.

2.6.4. Normalized Advantage Function (NAF)

The major drawback of the DDPG was the high delta between the iterations, which makes it suspectable to breaking changes in the controlled environment. This inability of the DDPG to create a smoother gradient for the target path is addressed by the Normalized Advantage function (Gu et al., 2016). The NAF algorithm is also optimized for the continuous action space with an actor-critic model for the policy evaluation. The NAF algorithm performs better when the solution space is narrower in nature, and multi-target optimal solutions are absent. The solution uses a Quadratic Approximation on the outputs given by the multiple dense layers. This approximation is further passed through an activation function to derive the final action Q value ("Normalized Advantage Functions — Reinforcement Learning Coach 0.12.0 documentation", 2020). This solution can be accelerated using transfer learning techniques to derive a base setting of the neural networks which have been already trained in the domain using the DQN or DDPG algorithms. In conclusion, this method has added the advantages of a model-based reinforcement learning to the exiting the model-free reinforcement learning agent used to create the DRL agents.

2.6.5. State–action–reward–state–action (SARSA)

SARSA is an on-policy algorithm which is an acronym based on the significant factors which drive the agent's behaviour. That is, based on the current state and the action chosen by the agent leads to the state change (Siebra and Neto, 2014). Afterwards, based on the reward observed by the agent from the environment and the state the agene have reached by the previous action, the new action is derived. This can be mathematically represented as (Siebra and Neto, 2014):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)$$

It is an iterative algorithm which updates its policies, having a low initial value for the first cycle will increase the chance of exploitation over the exploration. This property is highly favoured in the application domains where the probability distribution is spread over a range of values.

2.7. Edge Computing

The cloud systems have become the norm for storing data and big data analytics. The flexibility and rapid elasticity of the cloud systems have made it lucrative for prototyping and batch-processing. However, when the use of cases involving real-time analytics was implemented, the cloud systems were bottlenecked in terms of latency. This has led to the shift of the computing core more towards the edges of the internet called edge nodes. The technologies which blends cloud computing with IoT devices to bring the power of analytics while reducing the latency for the same is called edge computing (Liu et al., 2019). As depicted in Figure 9, The medium-level processors or PC-based gateways in the edge of the network close to the data generation points constitutes the edge computing area.

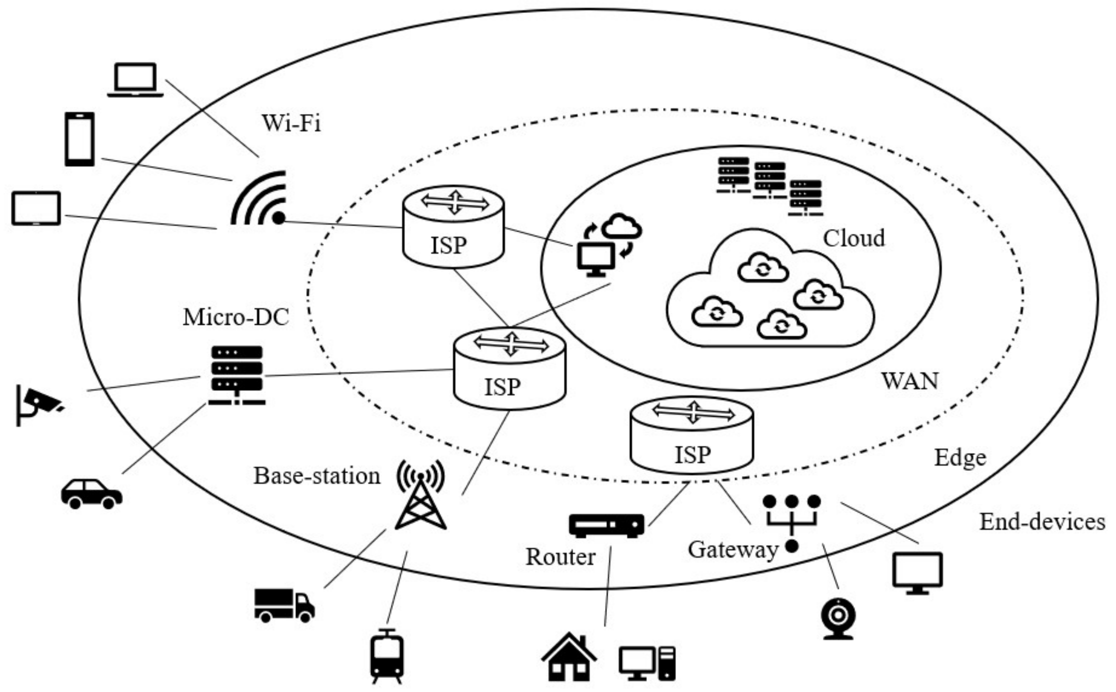

Figure 9. Edge Computing Arena in an end-to-end data flow of an Information technology ecosystem. (Liu et al., 2019)

The edge computing technologies have evolved in recent years as the number of end devices has increased. This has led to rapid adaptability of the field protocols into the edge devices (Calo et al., 2017). One of the most popular protocols which gained traction is Message Queuing Telemetry Transport (MQTT) which is a publish-subscribe method of a machine to machine communication (Kawaguchi and Bandai, 2020). Another popular method which has gained popularity is the Representational state transfer (REST) which is widely used for request-response mode of communication (Rykowski and Wilusz, 2014).

2.8. Industrial automation and Automation systems

Industrial automation is the elimination of human dependency on the manufacturing process using computer systems. The automation systems were designed using an electrical relay in the initial years, which was later replaced by PLC-based automation systems (Heddy et al., 2015). As shown in Figure 10, a process is monitored using a sensor, and an actuator is used to control the process. The logic to implement this mechanism is embedded inside a PLC-controller.

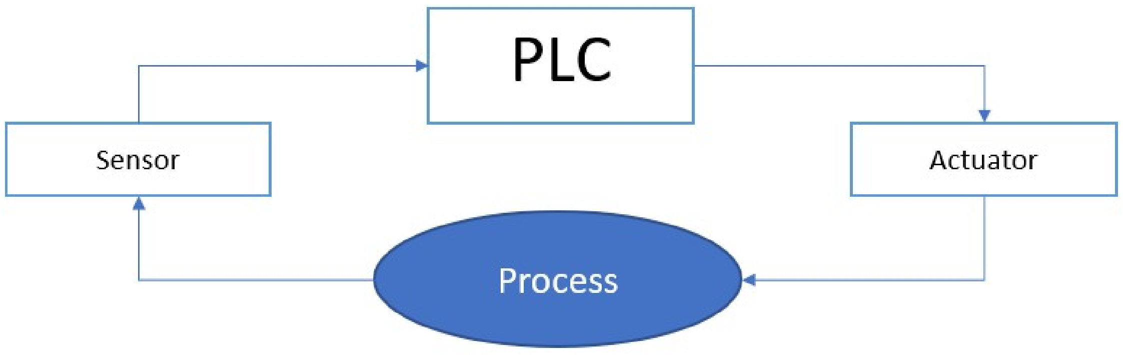

Figure 10. Data flow in a PLC-based Automation system (Shang et al., 2013)

2.8.1. Process

The process is the constant state of flux which is influenced by the energy variation applied to it. The process can have chemical, physical, electrical or mechanical forces acting upon it, which is usually applied to achieve the desired state (Iyer, 2018, p. 2). Process automation has been the core of the industrial research because such systems are incredibly complicated and optimization of any parameter had brought in a considerable amount of monetary savings. One of the foremost reasons for the complexity is the batch processing criteria: once a process is started if any metric is disrupted beyond the tolerance level, the entire set of raw materials are wasted. This requirement constitutes the core reason for process automation.

2.8.2. Programmable Logic Controller (PLC)

Programmable logic controllers are specialised computer systems based on Reduced instruction set computers (RISC) architecture (Kim and Sung, 2017). In RISC based computers, the microprocessor accepts only a small set of optimised instructions which increases the performance for some particular tasks. Most of the PLC's work with a mechanism called cycle times, where a set of instruction is set to execute at regular intervals (Hooper, 2006). The PLC design incorporates a broad set of registers in contrast to the conventional CISC architecture. The higher number of registers allows the system to swap between process with minimum reloading of the variables to the memory.

2.8.3. Sensors

Devices designed to capture the environment state metrics are called sensors. They are designed to detect changes and convert into electrical signals which are communicated to the computer for further analysis of the system and take necessary actions if required. The sensor can be categorised into discrete and continuous signal processing (Heddy et al., 2015). For example, an application like flow meter, pressure sensor are examples of continuous process variables where the measurements can take values in a continuous stream. Nevertheless, in reality, the values are modulated to values between 4 to 20 milliamperes for transmission purposes. The field of study which deals with the sensor is called process instrumentation.

2.8.4. Actuators

When an action has to be taken, humans or computer make use of the devices called actuators which induces changes to the environment based on the electricals signals calibrated for the same. The actuators have multiple domains of applications which utilises its own set of physical properties to achieve the results (Heddy et al., 2015). For example, in mechanics applications, the screw jack is tightened or loosened to control the speed of the rotatory component. The best example of electrical actuators is the DC motor where the voltage is used as a degree of targeted speed modulated based on the desired output values.

2.8.5. Closed-loop control systems

Closed-loop control systems are an electromechanical system designed to control a process based on a predefined logic without the manual inputs after the initial set up (Casado-Vara et al., 2019). As shown in the figure, the system interacts with the process and the auto-adjust it setting based on the user requirement.

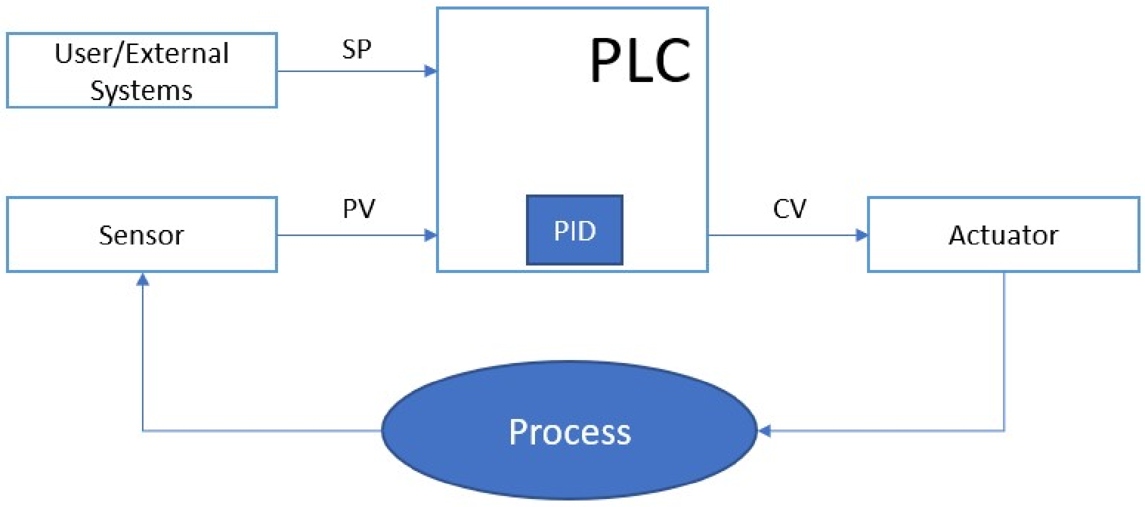

Figure 11. The variables in a closed-loop control *system (Shang et al., 2013)*

2.8.5.1. Process Variable (PV)

Measurable quantities associated with the process is called process variables (Schulz et al., 2018). Examples are temperature, pressure, viscosity, which represent a state representation to evaluate an environment in terms of reinforcement learning. These values are captured by sensors deployed in the environment and passed on to PLC for further processing.

2.8.5.2. Set-point Variable (SV)

The desired values for the process in the environment are called the set-point variables. The controlled systems will try to match PV to this value by manipulating the CV (Schulz et al., 2018). The human input is used to set this value. Usually, this value is constant in most control systems. However, in modern, flexible process controllers, these can be dynamically changed in runtime.

2.8.5.3. Controlled Variable (CV)

The controlled variable is the value which is used by the control system to keep the process under the environment constraints (Schulz et al., 2018). The CV is sent to actuators which converts the electricals signals to physical actions such as motor-starter, flow-control, valve-control, which will regulate the environment. The PV is impacted by the changes in the CV of which the correlation could be linear, inverse, quadratic or polynomial. Derivation of this correlation is the most laborious task in the PID controller tuning process(Shang et al., 2013).

2.8.6. Proportional Integral Derivative (PID) controller,

Control loops are simplified by using the PID controller as the core of the process control operations. The performance of PID is consistent and reliable because of which 95% of the controls in the industry have integrated PID blocks in the controller logic (Likins, 2016). Three elementary components of the PID controller are proportional, integral, and derivative which works in harmony to keep the PV as close as possible to the SV by controlling the CV. To achieve the desired equilibrium between these parameters, a mathematical model is created based on the behaviour of the process, which consist of three parameters:

- Kp – The value for the proportional gain Kp
- Ki – The value for the integral gain Ki
- Kd – The value for the derivative gain Kd

These parameters are tuned in a test environment during the commissioning phase. Currently, these tuning processes are designed to identify the correlation equations of the process which is set to control by the PID. As per Likins (Likins, 2016), the PID variables tuning is a complex process but essential in order to achieve optimal throughput while reducing the risk of process failures.

2.8.7. Industry Data Flow Pyramid

In manufacturing industries, the volume of the devices used, and data generated is enormous. To tackle this vast dataset, the dataflow is conceptualized into a pyramid where the segregation of the information is achieved for each layer (Sharp et al., 2018). This reduces the network bandwidth considerably, which in practice is reduced from n^2 to n.log (n) by implementing a star scheme network wherever applicable. For example, at the bottom level, the set of PLC, sensors and actuator associated with one control loop is kept in isolation in the network, and only a single point of contact is exposed for the outside nodes. Then these nodes are connected together to a SCADA system in a master-slave configuration, which in turn works in a similar way for the higher order of the dataflow pyramid, as shown in Figure 12.

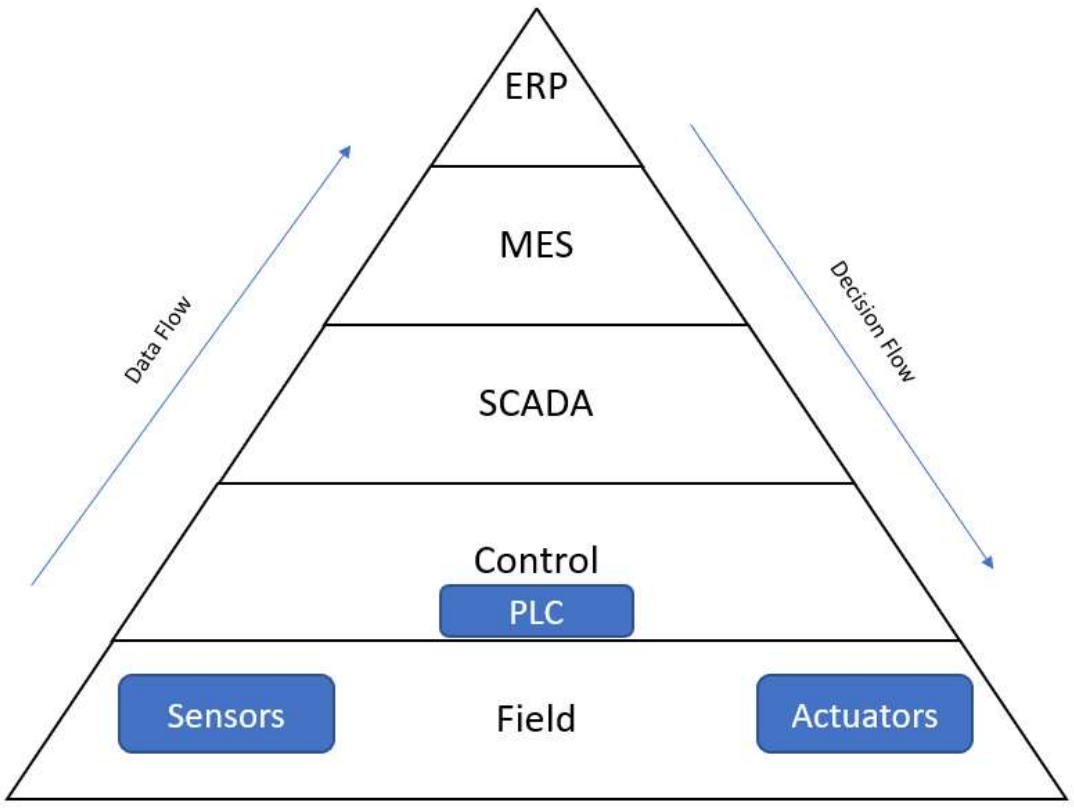

Figure 12. Dataflow pyramid (Sharp et al., 2018)

2.8.8. Industry 4.0

Industry 4.0 is an initiative by the German government to promote digitisation across the manufacturing industry. The core objective is to increase the agility of the manufacturing systems to cater to dynamically changing requirements in the market as well as to increase efficiency and to reduce the carbon print (Zhong et al., 2017). The significant contributions to this initiative are expected by increasing digitalisation of the manufacturing industry and to drive the industry towards a data-based decision-making process. As per Zhong, Industry 4.0 can be categorised into three types. Intelligent manufacturing, IoT-enable manufacturing, Cloud manufacturing. In intelligence manufacturing, the core is integration AI-based decision making. This is expected to be used along with big data analytics and connectivity services in seamless human-machine integration. AI-based systems are of more importance as the requirement for flexible manufacturing is introduced with faster customer feedback. The industry orientation towards more agile-based product development is also a significant contribution to the drive towards industry 4.0 (Rykowski and Wilusz, 2014, p. 0). The intelligent systems are not possible without the use of the IoT enables manufacturing systems. The IoT technologies empower the intelligent systems by providing a capability for real-time data collections and traceability of the issues to the atomic levels of the manufacturing plants.

2.8.9. IoT-based control systems

In conventional automation systems, the PLC-based control systems are preferred as it ensures high reliability. The PLC was designed and implemented in the late ’70s; therefore, familiarity with the PLC systems is better among the workforce in the industry domain (Bolton, 2006). The industry-grade Windows or Linux machines was not available for another three decades, and the earlier version was not stable enough to trust it with process automation. Nevertheless, in the last decade, the ecosystem has changed with more stable software packages and field-tested operating systems such as Ubuntu. With the introduction of the powerful IoT devices, the interest in reducing PLC dependency has gained much traction in recent years. Companies such as Siemens, Honeywell and ABB have started new product lines with a portfolio of Industrial Internet of Things (IIoT). Siemens IoT2020 is an IoT Gateway which has a Linux operating systems with direct IO connections (“SIMATIC IOT2000 | SIMATIC IOT gateways | Siemens Global,” 2020). Honeywell has designed a complete platform IIoT systems to utilize the full power of edge devices and data analytics (“Industrial Internet of Things by Honeywell,” 2020). A study by ABB (“ABB Industrial IoT applications | Internet of Things, Services and People - What is new,” 2020) on the IIoT use cases describes how the IoT and edge analytics can make monetary savings for the manufacturers.

2.9. Software components

This section lists the software which was used to implement this project.

2.9.1. Python 3

Python is a prevalent programming language with OSI approved open source licensing rights which allows the implementation of the software products even for commercial purposes (Phillips, 2015). The open-source nature has attracted a lot of communities to actively contribute to language modules which have increased the pre-built modules and libraries for almost every domain. The reason Python3 is selected in this project is that the OpenAI Gym library which is standard for testing the reinforcement learning algorithm is implemented in python which makes it a flawless in terms of API integration and compatibility of the library (Brockman et al., 2016). This is also supplemented by the fact that the Keras deep learning framework is also developed in python (Team, 2020).

2.9.2. IPython and Jupyter Notebooks

IPython is an interactive command shell for running python programs in a non-sequential execution mode while retaining the program data in the memory for cross-referencing the variable and function definitions (Rossant, 2018). IPython provides the inherent support for parallel execution for programs and native libraries for data visualization, including the support for MATLAB. Combined with the Jupyter notebooks GUI based program made the development of the prototypes and hypothesis became faster ("Project Jupyter," 2020). In this project, the Jupyter notebooks are used with IPython kernel to perform data analysis on the data collected about the learning experience of various agents and environmental observations (Perez and Granger, 2007).

2.9.3. NumPy

The NumPy is a module designed to simplify the intensive mathematical operation in python. It helped the more accessible adaptation of the python programming language into the scientific community ("NumPy," 2020). The NumPy is a library specifically designed to increase the effectiveness of the array operations in the python programming framework. In this project, the NumPy is used both in data generation as well as for data analysis in the later stages.

2.9.4. Pandas

Pandas is a library designed for data manipulation and analysis in python frameworks. It has native functions for reading data files in various file formats such as CSV, JSON, Excel and SQL ('pandas - Python Data Analysis Library' 2020). It reduces the custom programming overheads for handling tabular data structures. Pandas is helpful in the cleansing of the data by null value eliminations, filtering the time-series data and linear transformations using lambda functions. In this project, the observational data generated by the process artefact is analysed to analyse the environmental behaviour and evaluate the agent performance.

2.9.5. MATLAB

MATLAB is a technical computing software package created for engineering and scientific purposes (“MATLAB Documentation,” 2020). The MATLAB provides scalability and for higher mathematical modelling using the software counterparts. Python programming has a module of MATLAB by which the graphical representation of the data is faster and accurate without additional custom programming for the visualization. In this research, MATLAB has helped in giving insights towards the behaviour of the agents in a different environment with various parameter settings.

2.9.6. Simple PID controller

Simple PID controller is a python package which is created using the conventional integral derivative logic (*simple-PID: A simple, easy to use PID controller* 2019). This particular package is designed based on the PID equation, which is described as per the Brett Beauregards guide (“Improving the Beginner’s PID – Introduction « Project Blog,” 2018). The initialization of the PID is done with a pre-set value for the parameters Kp, Ki and Kd. Then an infinite while loop is run in order which continuously fed a value for process variable and the PID control in turn provides the process with a value for the CV. The process library which is described in the section 3.4 is used to emulate the process, this same library is used by the DRL-based PID so that the results are consistent.

2.9.7. TensorFlow

The process of creating deep neural networks using conventional programming was hectic and error-prone. So, the requirement to develop a framework for neural networks became inevitable. In the year 2011, the Google Brain project conducted extensive research into neural network creation and computations, which can handle a vast number of neurons (Dean et al., 2011). The product of this first phase was the DistBelief (Le, 2013), also called as first-generation scalable distributed training and inference system. Several applications such as Object Detection (Frome et al., 2015.), Video classification (Szegedy et al., 2014) and speech recognition (Mikolov et al., 2013; Vinyals et al., 2015) made several advances using this network, and a second-generation framework was created for these extensive deep neural network system called TensorFlow (Abadi et al., 2015). The base philosophy behind this improved framework was to create a harmonious data flow across varying hardware systems such as android, iOS, high powered server systems and dedicated GPU cards in the personal computer systems. This simplified the development and deployment of the deep neural networks the underlying hardware become less of a concern to the developer, which was abstracted to neural networking components in the TensorFlow framework. The framework has been designed to be robust so that, it is suitable for prototyping and deployment for production systems without much optimizations from the developer. The scaling and performance tuning, according to the hardware, is abstracted inside the TensorFlow runtime itself. As per Author of TensorFlow (Abadi et al., 2015), it is optimized for parallel processing which have shown increased efficiency in the runtime with respect to its predecessor such as DistBelief. The TensorFlow is created out of the requirement for deep neural network systems in the modern digitalisation era of the human technical advancements. This open source platform has nurtured the deep learning research and development community has contributed vastly in the boosting of the performance (Abadi et al., 2015).

2.9.8. Keras

Keras is an extension of TensorFlow framework in which the tensor operations are further abstracted to direct deep neural network components and operations such as layers, models, training and classification (‘Keras Documentation’ 2019). The high-level API interfaces provided by Keras decreased the boilerplate code and increased the readability of the deep learning projects. Keras is supported with multiple back ends such as Theano and TensorFlow, which is easily interchangeable with zero changes to the source code for the project (Gullì and Pal, 2017). It has incorporated the optimization algorithm to reduce the cognitive load in computational resource consumption. It has been ranked the best framework for deep learning in the year 2019 based on the data of Kaggle competitions (Team, 2020). The Keras supports direct deployment into runtime on the latest TPU hardware, which is the latest and fastest hardware specially designed for tensor operations. Along with the support for distributed GPU computational clusters, it makes the lucrative choice for any large deep neural network implementations. Auxiliary frameworks such as AutoKeras (Jin et al., 2019), TF Cloud (*TF Cloud*, 2020), Keras Tuner (*Keras-tuner*, 2020) are developed by the community which is further boosting the developer acceptance circle for the Keras.

2.9.9. OpenAI Gym

The primary aim of the OpenAI Gym project is to consolidate the various reinforcement learning environment and unify the benchmarking criteria for the solutions. Currently, prevalent environments such as cart pole, pendulum are provided as part of the package. The user can create different agents which can solve the problems and compare the performance in terms of iterations and rewards. In OpenAI Gym, the reinforcement learning is achieved by breaking the iteration into a series of episodes which add to the agent's experience in a discrete incremental manner (Brockman et al., 2016).

2.9.10. Keras-rl

Keras-rl is a python framework which makes use of Keras library to create deep reinforcement learning agents ("Core - Keras-RL Documentation," 2020). These agents which are created using the Keras-rl framework can be tested against the OpenAI gym environments. For this project, there are five agents under consideration, as described in section 2.6.

2.10. Related works

The concept of deep reinforcement learning has been used in the domain of industrial automation before also. The following sections will summarise the works related to PID controllers and deep reinforcement learning.

2.10.1. Parameter Optimisation of PID Controllers by RL.

The initial application of reinforcement learning was to optimize the parameters of the PID, as described in section **Error! Reference source not found.**. Shang et al. (Shang et al., 2013), have successfully used reinforcement learning to reduce the convergence time over the existing population-based optimization algorithms of the PID parameters. The method is the direct implementation of Function Optimization by Reinforcement Learning (FORL) which has outperformed the previous algorithms used to optimize higher order functionalities (Liao and Wu, 2010).

2.10.2. Improve the PID controller through RL

The static tuning of the PID variables worked well in a passive process where the set-point is not expected to change without a downtime. Therefore, methods such as FORL can be used only in a linear control process. To solve, this, Qin et al. (Qin et al., 2018) have used deep reinforcement learning which individually tunes each PID parameters in the control process.

A deep deterministic policy gradient (DDPG) algorithm is used to train the agent in the control process environment. This allows for a continuous state space search for the agent to set the parameters. The study presents a comparative study between the conventional PID, DQN and the new DDPG-PID. The experiment was based on the OpenAI Gym environment: Inverted pendulum simulation.

2.10.3. RL and DNN for PI Controller Tuning

In a study by Shipman and Coetzee (Shipman and Coetzee, 2019), a first-order controller system was analysed by implementing both discrete and continuous action spaces. However, the environment is a theoretical representation using mathematical formulations. The system had closed-loop control environment with dynamically changing SP with was implemented using OpenAI Baseline library and Python 3.6 of Anaconda distribution. The duration of the experiment was segregated into phases based on the SP dynamicity, and the performance of both the discrete and continuous agents was compared in terms of reward collection, error over time, and controller gains. An input layer of dimension 120x5 was used, which was further flattened into 600 element vectors. Tan H activation function was used for all the layer in the deep neural network. The study concluded that deep reinforcement learning is a highly desirable candidate for implementing PID controllers in a dynamic control loop.

2.10.4. Performance Analysis of Deep Q Networks for PID controllers

A self-tuning PID controller is the ultimate use case of deep reinforcement learning in the field of control system automation. Currently there two popular methodologies which are used to design self-tuning PID controllers: Deep Q network and advantage Actor-Critic algorithm. An experiment was conducted by Mukhopadhyay *et al.* (Mukhopadhyay et al., 2019), in a simulated environment for the servo position control system to compare the previously stated algorithms for designing self-tuning PID controllers. The system was benchmarked against a PLC-based PID controller in both static and dynamic environment. The authors concluded that when the parameter variability is high, the Deep Q-networks are better than the actor-critic model. In the case of tracking of trajectory-based applications, the A2C models perform better.

2.11. Summary

The literature review has covered the previous works related to the application of artificial intelligence in the domain of automation industry. This chapter had given in details explanations of the core concepts related to this research, that is, the deep learning and reinforcement learning. Afterwards, the deep reinforcement learning agents under consideration is discussed with its advantages and disadvantages. Furthermore, the reinforcement learning environments to be designed is presented, that is, discrete and continuous environments. To give a context of the application area, the industrial concepts are explained with examples after which the key terms used in the project related to the domain are described. Additionally, the various software components used to realize this experiment is introduced with their usage highlights. Finally, the related works are explained to highlight the importance of this research.

Chapter 3

3. Design

3.1. Introduction

The design of the experiment is the crucial phase of this project. The design of the project is divided into system design, internal module designs, interfaces for interaction between the modules, physical-world emulation requirements and framework capabilities. Initially, a system design is drafted, after which, each module in it designed as per the interface requirements. This chapter also includes details about the framework capabilities to be considered, such as OpenAI Gym interfaces, Keras interfaces and Process PID interfaces. Another important discussed in the section is the restriction of the neural network in terms of size and activation functions while implemented in the edge computing devices.

3.2. System Design

The system is broken to various modules for easy maintenance and extensibility. The interaction between the modules of the system is based on the API interfaces. High cohesion in modules is achieved using inheritance and encapsulation. The coupling between modules is reduced by following the concept of abstraction and library like the exposure of the functionalities developed. As shown in Figure 13, the system is divided into multiple modules such as Process, Environments, Agents and a parallel conventional PID controller. Each component is described in detail in different sections.

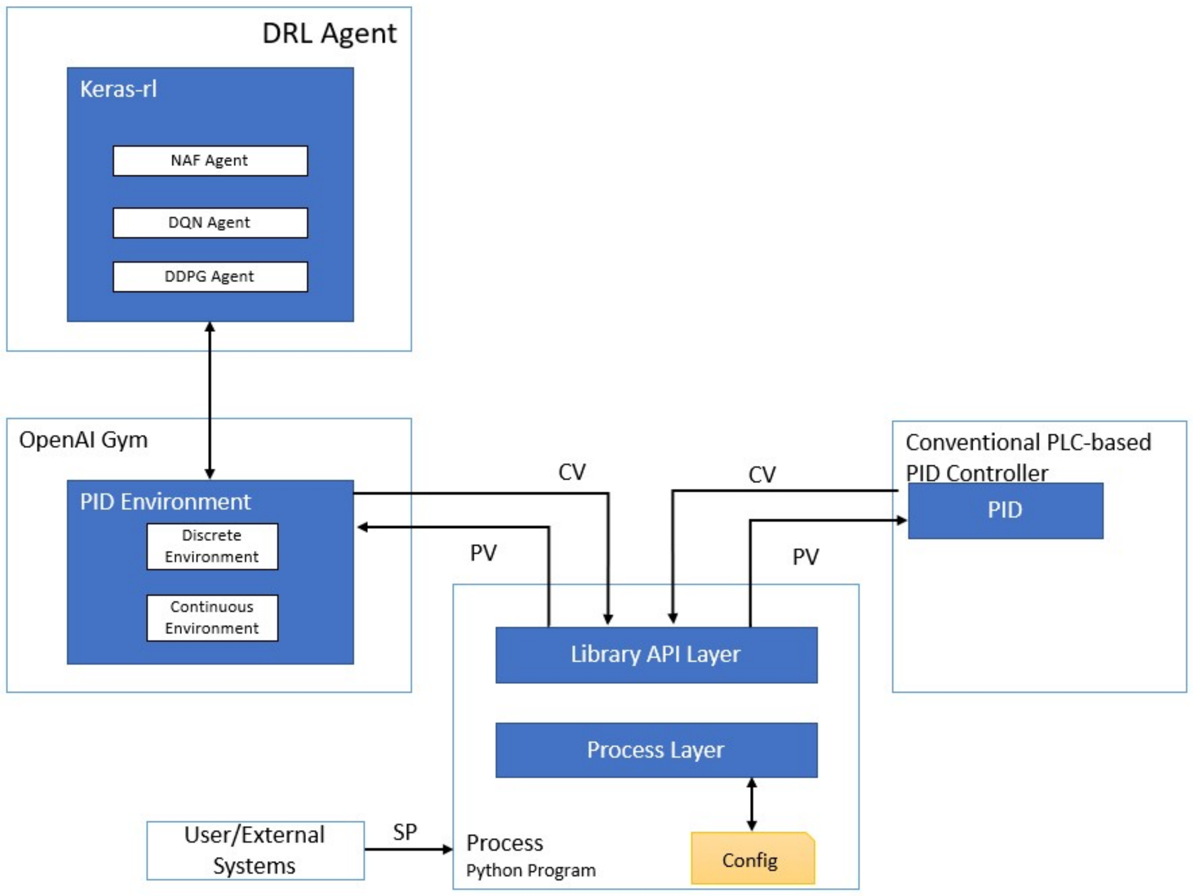

Figure 13. The overall system design of the experiment set up

3.2.1. Environment-process interaction

The process module is developed to emulate the industrial process which behaves the based on the equations sent in the process config file. The process outputs should be based on the correlation equations explained later in section 3.5. Figure 14 illustrates the interaction between the environment definition in the OpenAI gym standard and the process component. In each step of the agent-environment interaction, a new controlled variable is calculated and the feed into the process component, which in turn provides the environment with a new process value.

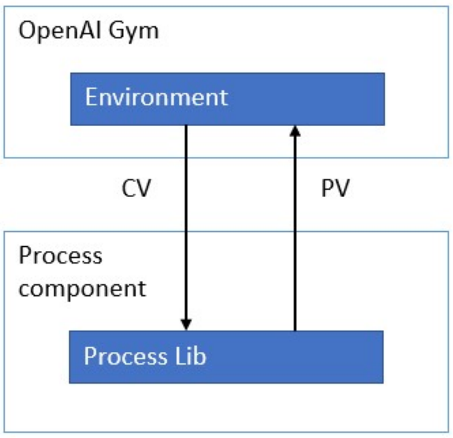

Figure 14. Interface definition between OpenAI Gym environment and Process component

3.2.2. Agent-Environment interaction

The environment is designed to give feedback for the agent actions based on the process behaviour. The environment definitions standardize the interfaces needed for the evaluation of the agent performance in OpenAI Gym framework. The significant interactions between the agent the environment will be based on the action chosen by the agent and reward calculated by the state change of the environment. The agent and environment interaction is based on action and reward. As depicted in Figure 15, the agent is given a reward based on the action chosen in the current iteration.

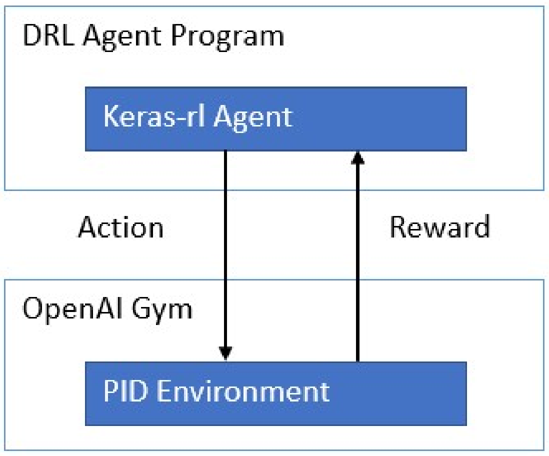

Figure 15. Agent and environment interaction

3.2.3. Benchmarking System interaction

A parallel PLC-based PID controller is setup parallelly which interacts with the exact same process in order to evaluate the process. This PLC-based PID controller has the exact same parameters configuration as the DRL-based PID controller. This is one of the reasons why the process behaviour was modularised so that the same interfaces could be programmed in both controllers. As shown in Figure 16, the interaction is as same and the OpenAI Gym environment.

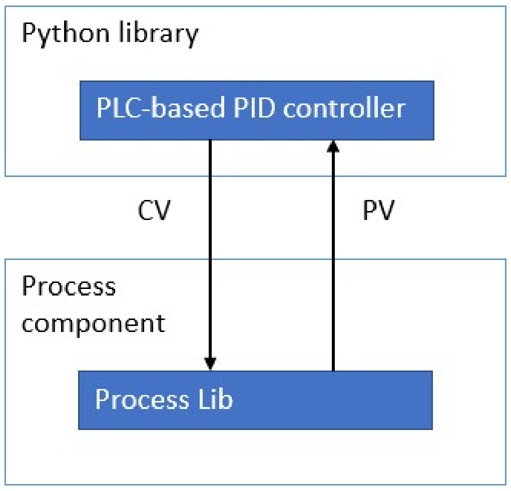

Figure 16. Interaction between the process library and the benchmarking system

3.2.4. Logging and data collection

The environment variables are captured at the process library, which is not contextual to the Mode of the implementation of the PID logic. This allowed an unbiased data representation for both conventional and the DRL-based controllers. The logging of environmental data such as the PV, SP, CV is analysed further in the processing of the results. The precision of the values is set to 5 decimals points which are handled by the process class itself. Each instantiation of the process creates a new data log file with a creation timestamp as the filename. CSV format is used for the data collection as it is unambiguous and straightforward in the analysis phase to read from Jupyter notebooks.

3.3. Environment design

The environment in OpenAI Gym is represented as a set of observables values which are influenced by the agent’s actions (Brockman et al., 2016). Each action is presented with a reward based on the changes in the environment and how much it is closer to the goal state. It is based on Figure 17 depicted below.

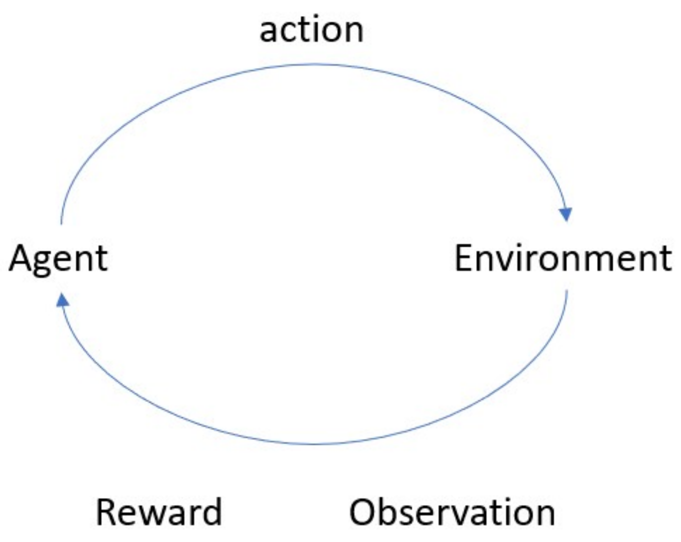

Figure 17. Agent and Environment Interaction (Arulkumaran et al., 2017)

Agent interaction with the environment is using the predefined interfaces defined by the OpenAI Gym framework. The rewards are based on the observation in the environment, which is custom-defined for the process control scenario using PID blocks.

3.3.1. Space Class

The space class is an abstract class of Gym package, which is written in order to enforce specific properties for all the observations and actions which will occur in the environment (*Gym Spaces*, 2018). The class consist of methods such as “sample”, “seed” and “contains” which enables to perform environmental changes such as initialization, observation and manipulation.

3.3.2. Observation space

The Observation space is a quantized representation of the environment variables which describes the state of the system in terms of numerical values (*Gym Wrappers*, 2019). It is a subclass of Space objects in the OpenAI Gym environment.

3.3.3. Action space

The Action space is also a quantized representation but which represents the values which can be fed into the environment as the action taken by the agent in a particular instinct (*Gym Wrappers*, 2019). These actions taken by the agent and the rewards returned are the key elements which decide the future actions of the agents for a particular situation. This is also a subclass of the Space class.

3.3.4. Rewards

Rewards are the key evaluation metric used in analysing the agent performance for a given environment (Mnih et al., 2016). The rewards functions are a vital component to be designed in step function as it determines the convergence accuracy of the agent to the goal state. The rewards function in this project will be based on much closer the current PV has moved to SP from the previous state. If the previous PV was close to SP than present PV, a negative reward would have to be awarded and if a current PV is closer to the SP than the previous state a positive reward should be given. If the PV has not changed, then the rewards shall be zero.

3.3.5. Step function

This is the essential function which determines the agent interactions with the environment. In each iteration of the step function, random action is selected from the action space and applied to the environment. Then the environment should process this action which is the process artefact: a library which CV take as input and outputs PV based on the correlation equation specified in the config file of the process artefact.

3.4. Process requirements

The Process Library is one of the crucial artefacts of this project. It defines how the environment is responding to the input given by the agent. The process is implemented as a class with properties such as set point variable, controlled variable, process variable, correlation equation, degree of the correlation equation, set point change frequency (variables detailed in section 2.8.5). The primary function of this module is equation evaluator which gives a value for process variable based on the controlled variable value provided. This function is the emulating the process which usually happens in an industrial process automation scenario. The implementation of the process artefact is detailed in section 4.2.

3.5. Correlation equations

The correlation equations are the behavioural plotting of the CV against PV (Shang et al. 2013). The deep reinforcement learning agent is expected to derive this relationship from experience by adjusting the weight over the episodes in their respective environment. In this project, the correlation equation is of four types:

3.5.1. Positive linear correlation

In this type, as shown in Figure 18, the correlation is linear. The value of PV increases as the CV increases. The process variables like motor speed which directly depends on the voltage across the electrodes.

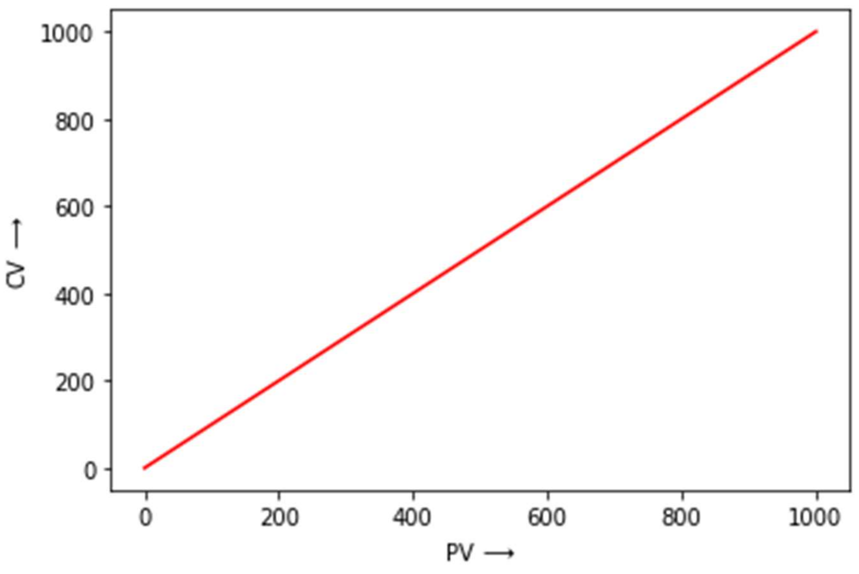

Figure 18. Positive linear correlation

3.5.2. Negative linear correlation

As shown in Figure 19, in negative linear correlation PV decreases as the CV increases. This type of correlation exists in pressure chambers, where the pressure drops the flow of liquid increases.

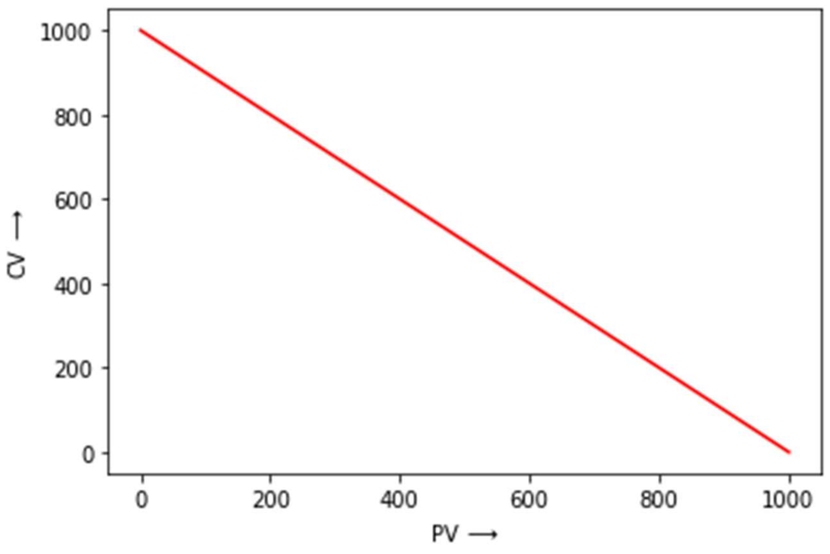

Figure 19. Negative linear correlation

3.5.3. Quadratic correlation

In quadratic correlation, PV increases nonlinearly against the CV. In the heating process of metals, this type of correlation is common.

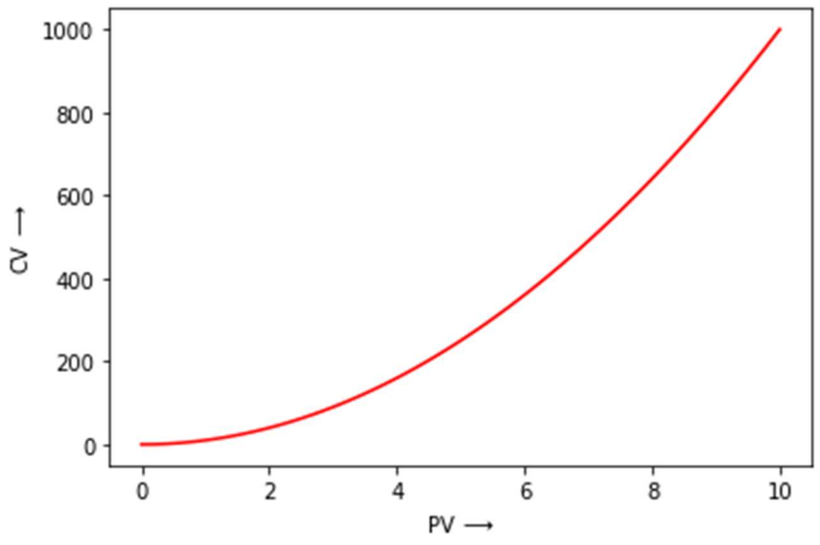

Figure 20. Quadratic correlation

3.5.4. Polynomial correlation

This type of correlations is very complex in nature; at different point of the CV, the PV could change at varying degree. This kind of correlations occurs when a chemical reaction is in progress.

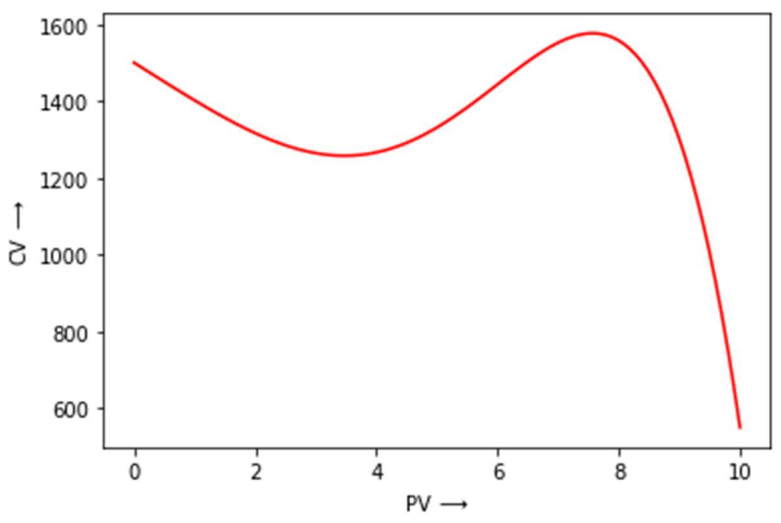

Figure 21. Polynomial correlation

3.6. Agent requirements

The agent should have the ability to decide the action to perform given a particular PV. This can be to increment CV, decrement the CV or keep the CV constant based upon the previous experience. The agent should be able to interpret the rewards and correct its actions at each stage. The deep reinforcement learning agent should store the data about the experience in its neural network, which is explained in section 3.7. The agent is expected to interact with the environment through the interfaces described in section 3.2. The agent can have the state space and action space in discrete or continuous values based on the agent type described in section 2.6. The ultimate aim of the agent is the keep the PV as close as possible to the set value in a minimum number of iterations. The agent is expected to reach the goal faster as the experience grows.

3.7. Selection of environments based on agents

The agents which are described in section 2.6 are implemented to either discrete or continuous output. Table 2 lists the agent type the respective action space output type as per the design document of Keras-RL agents (“Agents and Environments Keras-RL Documentation,” n.d.). Based on this information, the five agents provided by the libraries are coupled with two types of environments, which is created as part of this research.

Table 2. Agents and the action space type

Agent Type	Action Space
DQN	Discrete
DDPG	Continuous
NAF	Continuous
CEM	Discrete
SARSA	Discrete

3.8. Neural network requirements

Most of the neural networks are enormous in terms of the neurons, which is expected to increase accuracy (Zhou et al., 2019). The size of the neural network is not a significant concern when the system is comprised of high-end computing resources whereas in the realm of edge computing where the processing power and memory space is limited the network has to be optimized in order reduce the latency in the control logic which is crucial to the process continuity.

3.8.1. Size

The number of neurons of the neural network should be as minimal as possible. In order to make sure the accuracy is not reduced considerably, ideally, the experiment should be conducted with two sets of neural network sizes. This would have allowed determination of the optimal neural network size for the edge computing devices (Calo et al., 2017). Due to the computing restraints for the project, this experiment will be restricted to the minimum network size scenarios.

3.8.2. Activation functions

The activation functions are essential as they can have a significant influence on the computing requirements. The activation functions are chosen based on the domain behaviours. As described in section 2.5.2, the binary function is not suitable for real-world applications. Due to the continuous complex nature of the Sigmoid and Tan H, the computation resource usage will be high, which is not affordable in the edge computing domain (Gullì and Pal, 2017). Therefore, this research will be concentrated on two types of activation functions, that is, ReLu and Linear.

3.9. Result Analysis

The agents are run for 100 times with exactly same parameters. This allows reducing the effect of the skewed data points in the agent performance. The statistical measurements such as mean and median are used in result analysis. As part of the data collection program mentioned in 3.2.4, the data collected includes the current set-point and attained PV. Based on these two values, the error factor is calculated for each instance. Then the median of these error values is calculated for all the collected data. Based on this median is calculated for the first transition edge and last transition edge. After that, the count of mean which has values above this value is calculated for each agent.

3.10. Summary

The design chapter has discussed the system design and the requirements on a module level. These requirements have to be satisfied while the modules are implemented. This design has ensured that the system works as a whole without any disruptions causing the data to be corrupted while the training is carried out. The interfaces discussed for the interactions have made the modules independent and reusable, which has reduced the code base for the experiment optimized, scalable and portable. The interface definition would be helpful in future if the experiment needed to be implemented using a new framework for deep neural network or reinforcement learning.

4. Implementation

4.1. Introduction

The implementation section explains how the design principles described in Chapter 3 is realized in the project. The project is built around the object-oriented concepts wherever applicable. Process Library is the class which is used by both DRL-agents, and the PLC-PID controller is the core behavioural drive of the experiment. Process library is segregated and detailed in section 4.2, after which the environmental design is introduced. Out of two new environments designed for this experiment, only the discrete environment design is detailed as the continuous environment also follows a similar design pattern. Similarly, out of the five agents under considerations, the implementation of the DQN agent is explained in detail, and the only delta in the agents are described in the dissertation. Finally, the method of result analysis is presented, and the code used for the same is explained.

4.2. Process artefact

The process artefact is designed as an independent python module which can be imported in the other programs as a library function. The library is developed as a class as per OOPS standard with private variable and public function to access the same. The Process module is designed in a way that the behaviour of the process can be changed just by modifying the configuration. In this section, the component Process will be implemented, as shown in Figure 22.

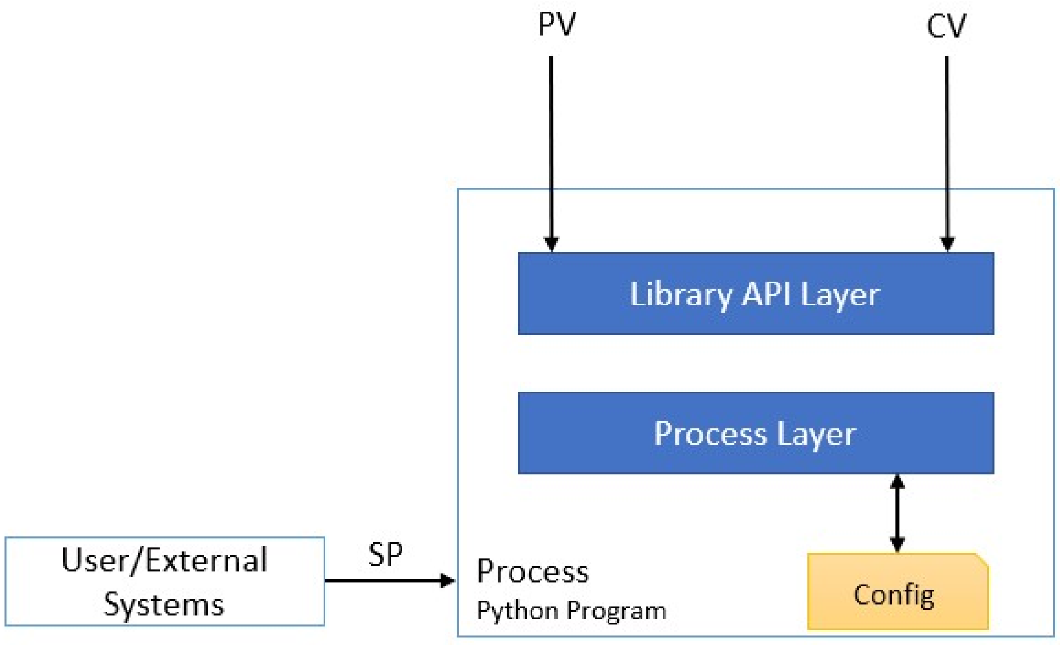

Figure 22. Process Artifact design

4.2.1. Config file

The config file consists of parameters which define the environment in a JSON format. The config file has set of arrays with keys 1 to 8 which represent polynomial equations from as shown in the Code Listing 1. For example, the key 1 represents an equation with degree one and **“1”: [8, 9]** represents **$y = 8x + 9$**. Likewise, **“2”: [7, 8, 9]** represents **$y = 7x^2 + 8x + 9$.**

```
"1": [
    8,
    9
],
"2": [
    7,
    8,
    9
……………………………….
……………………………….
……………………………….
……………………………….
……………………………….

],
"8": [
    1,
        2,
        3,
        4,
        5,
        6,
        7,
        8,
        9
    ]
```

Code Listing 1. Polynomial Arrays

The config file is designed to be versatile in nature so that various correlation behaviour can be emulated with just a change of a variable named degree. The values of PV, SP and CV are the initial values of the environment which will be modified in the run time. As shown in Code Listing 2, environment variables are stored as key-value pair in the JSON object.

```
    "sp": "10000",
    "degree": 1,
    "cv": 1,
    "pv": 0,
    "sp_change_frequency": 1000,
```

Code Listing 2. Environment variables

4.2.2. Reading the config file

The configuration file is read as JSON and loaded into a python object called polynomials as shown in the Code Listing 3. The polynomials are stored as array values which are in turn used to determine the degree in later stages.

```
with open('../config.json', 'r') as config:
       self.polynomials = json.load(config)
```

Code Listing 3. Reading the configuration

4.2.3. Equation evaluator

This code is the core of the process module, which calculates the PV based on the cv. Initially, the variable name total is declared with value 0. Then based on the degree, the corresponding coefficient array is loaded. The PV is evaluated by accumulating the value raised to its power multiplied by its corresponding coefficient while iterating through the array. In the end, the total is assigned to the PV value of the process class.

```
# Resetting the vale
    total = 0

# Iterating through the data
    for index, coefficient in enumerate
                (reversed(self.polynomials[str(self.degree)])):
        #print (index, ' ', coefficient)
        total = total + (coefficient * (x_value ** (index)))

# Assigning the PV value
self.pv = total
```

Code Listing 4. Equation Evaluator

4.2.4. Saving the data

When an instance, the process class is initialized, a corresponding file is created with the start time for the execution as the file name. The variables captured in the file are CV, SP, PV and timestamp as shown in the Code Listing 5.

```
    now = datetime.now()
    dt_string = now.strftime("%d_%m_%Y_%H_%M_%S")
    print("date and time =", dt_string)
    dt_string = 'data/' + dt_string + '.csv'
    self.f = open(dt_string, "w")
    self.f.write('cv,sp,pv\n')
```

Code Listing 5. Creation of data file

Each call to equation evaluator triggers a data write to the file which acts as the dataset for further analysis as depicted in Code Listing 6.

```python
# Creation of JSON Data
data = {'sp': round (self.sp, 5),
        'pv': round (self.pv, 5),
        'cv': round (self.cv, 5)}

# Converting to comma separated values
w_data = str(self.cv) + ','
         + str(self.sp) + ','
         + str(self.pv) + '\n'

#  Writing to the file
self.f.write(w_data)
```

Code Listing 6. Writing the data to the file

4.2.5. Dynamic SP change

To test the adaptability of the agent in a dynamic setting of SP. The SP is changed at the frequency set in the configuration file as shown in Code Listing 7.

```python
# SP changer
self.counter += 1
if self.counter > self.sp_change_frequency:
    self.counter = 0
    self.sp = self.sp * uniform (0.8, 1.2)
```

Code Listing 7. Dynamic SP value change

4.3. Discrete environment

The discrete environment will give provide the agent with predefined sets of actions. As describes in section 2.4, the environment is easy to be manipulated by the agent as the actions are bound to specific predefined outcomes. This reduces the computing requirements for both agent and the environment set up.

4.3.1. Initialization of the environment

Initialization of the object of this class creates an instance of the class which have the properties described in section 3.4. Afterwards, as per the requirement of the OpenAI gym environment design, as mentioned in section 2.9.9 which initializes the variables viewer, state, and steps_beyond_done. The parameters previous_error and current_error of the discrete environment are used to compare the difference between PV and SP in each iteration.

```
        ######################################
        # Initializing Process
        ######################################

        self.process = Process()

        ######################################

        self.seed()
        self.viewer = None
        self.state = None
        self.steps_beyond_done = None

        ######################################

        self.previous_error = 0
        self.current_error = 0

        ######################################
```

Code Listing 8. Initialization of the process environment

4.3.2. Observation space

The observation space is created using the Box class mentioned in section 0. An array with limit set PV value and SP value is initialized and fed into the constructor of the Box class to create the observation space object as shown in the Code Listing 9.

```
        high = np.array([self.process.pv, self.process.sp],  dtype=np.float32)
       self.observation_space = spaces.Box(-high, high, dtype=np.float32)
```

Code Listing 9. Initialization of the observation space

4.3.3. Action space

The action space is limited in number in the case of the discrete environment implementation. The help of spaces class as described in section 3.3.1 is taken to create the action space type which complies with the actions accepted by the environment-agent interaction of the

OpenAI Gym framework. As shown in Code Listing 10, the action is initialized with five possible actions.

```
# 5 possible actions 0 to 4
self.action_space = spaces.Discrete(5)
```

Code Listing 10. Initialization of the action space

4.3.4. Increment calculation

The increment calculation for the step is calculated based on the current delta between the SP and PV. The change factor represents how fast the convergence is expected out of the system, which is set on the config file of the process. The faster the convergence, the less the agene ability to converge to an optimal state of performance.

```
increment = (self.previous_error/self.process.sp) * self.process.cv * \
        (self.process.cv_change_percent ** self.process.cv_change_factor)
increment = round(increment, 5)
```

Code Listing 11. Increment calculation and rounding off the values

4.3.5. Step function

The step function is the crux of reinforcement learning, in which the complete cycle of agent action and environment feedback is fed back to the agent for learning, as described in section 3.2. The step function has three major responsibilities: find the change to be injected based on the action chosen by the agent, obtain the new environmental values based on the feedback from the process library and to determine the rewards for the action taken by the agent. As shown in the function written in the Code Listing 12, the step function accepts one argument other than its own reference, that is, action chosen by the agent for that step.

```
def step(self, action):
  ……………
  ……………
  ……………
  # Changing the cv based on action chose by the agent
  if action == 0:
```

```python
    pass
elif action == 1:
    self.process.cv += increment
    print('increment action')
elif action == 2:
    self.process.cv -= increment
    print('decrement action')
elif action == 3:
    self.process.cv += increment*2
    print('increment high action')
elif action == 4:
    self.process.cv -= increment*2
else:
    print('unidentified action')
```

Code Listing 12. Function signature for the step function

The step function also has the responsibility to update the new CV value based on the agent action. As listed in Code Listing 12, the change in the CV is calculated with respect to the value of the increment calculated in the 4.3.4

4.3.6. The trigger of equation evaluator

The discrete environment uses the equation evaluator function of the process-module to assign the new values of the process variables. As depicted in Code Listing 13, the new process values are assigned using the values calculated by the processing module.

```python
# New PV is calculated by the process
new_values = self.process.eq_evaluator(self.process.cv)

# New sp value based on the equation
self.process.sp = new_values['sp']

# New pv value based on the equation
self.process.pv = new_values['pv']

# Calculating error
self.current_error = abs(self.process.sp - self.process.pv)
```

Code Listing 13. Evaluation of the new process values using process module

4.3.7. Reward calculation

In the discrete environment, the rewards are binary. The agent is awarded a positive reward if the error of the PV from the SP is reduced by the action of the agent in the current step. As shown in Code Listing 14, the same is achieved using a simple if statement in the environment module.

```
# Reward calculator
if self.current_error < self.previous_error:
    reward = 1
else:
    reward = -1
```

Code Listing 14. Reward calculation

4.4. Continuous environment

The continuous environment follows the same design pattern as the discrete environment described in section 4.3, in all other aspects except the action space and rewards. Unlike in the discrete environment, action space uses Box class to initialize the action space, as shown in Code Listing 15. The rewards value is converted to the proportional value of the current value.

```
high = np.array([self.process.pv, self.process.sp],
                         dtype=np.float32)

self.action_space = spaces.Box(
    low=0,
    high=1,
    shape=(1,),
    dtype=np.float32
)
self.observation_space = spaces.Box(
    low=-high,
    high=high,
    dtype=np.float32
)
```

Code Listing 15. Initialization of the action space

4.5. DQN agent

The implementation of an agent is split into three steps. First, a set of the initial condition such as the environment, seed and action space dimension is initialized. Afterwards, the core processing setup of the agent is defined, that is, the deep neural network. Then the agent class DQN is initialized with the parameters mentioned in the first two steps. Code Listing 16 shows the environment declaration for the DQN agent using the discrete environment explained in section 4.3. Afterwards, the action space dimension is captured, which is 1 in the case of the current experiment.

```
ENV_NAME = 'drl_envs:discrete-pid-v0'

# Get the environment and extract the number of actions.
env = gym.make(ENV_NAME)
np.random.seed(123)
env.seed(123)
nb_actions = env.action_space.n
```

Code Listing 16. Initialization of the agent parameters

The next step is to create the neuron layers for the deep neural network. For the DQN agent, a sequential model is created using the Keras inbuilt functions as listed in Code Listing 17. The network has three hidden layers in addition to the input and output layer. The input layer is consists of a four neuron interface which is fed by the four possible values for the actions which can be taken by the agent. Afterwards, three dense layers with ReLu activation function are introduced. The output layer is fitted with an additional linear layer to enable the smooth transition for values.

```
# Deep nueral network layers
model = Sequential()
model.add(Flatten(input_shape=(1,) + env.observation_space.shape))
model.add(Input(shape=(1, 1, 4)))
model.add(Dense(16))
model.add(Activation('relu'))
model.add(Dense(16))
model.add(Activation('relu'))
model.add(Dense(16))
model.add(Dense(nb_actions))
model.add(Activation('relu'))
```

```
            model.add(Activation('linear'))
```

Code Listing 17. Composition of the neural network

Based on the neural network designed and the environmental parameters set in the agent is compiled. In the case of the DQN agent, the Boltzman policy is applied in order to choose the actions within the specified list of the environment parameters. The agent is providing the actions space as shown in the Code Listing 18, which in turn selects within those while the agent is running.

```
           # Compiling of the agent
           memory = SequentialMemory(limit=1000000, window_length=1)
           policy = BoltzmannQPolicy()
           dqn = DQNAgent(model=model, nb_actions=nb_actions, memory=memory,
                          nb_steps_warmup=100,
                          target_model_update=1e-2, policy=policy)
           Adam._name = 'DQN'
           dqn.compile(Adam(lr=1e-3), metrics=['mae'])
```

Code Listing 18. Compilation of the agent with policy

4.6. DDPG agent

The DDPG agent follows the same initialization process as the DQN agent but in terms of initialization. In contrast to the DQN, the DDPG is an actor-critic model. Therefore, there are two neural networks defined one for each actor and critic, respectively. The difference is mainly in the compilation of the agent where the agent class used is different. Additional parameters are set, such as critic as the DDPG is an actor-critic model of reinforcement learning. The details parameters are described in Code Listing 19.

```
           # Compiling of the agent
           memory = SequentialMemory(limit=100000, window_length=1)
           random_process = OrnsteinUhlenbeckProcess
                            (size=nb_actions, theta=.15, mu=0., sigma=.3)
           agent = DDPGAgent(nb_actions=nb_actions, actor=actor,
                             critic=critic, critic_action_input=action_input,
                             memory=memory, nb_steps_warmup_critic=100,
                             nb_steps_warmup_actor=100,
                             random_process=random_process,
                             gamma=.99, target_model_update=1e-3)
```

```
        agent.compile(Adam(lr=.001, clipnorm=1.), metrics=['mae'])
```

Code Listing 19. Compilation of the DDPG agent

4.7. NAF

In NAF, we have three neural networks in place which is called as V, mu and L networks. These are configured similar to the other neural network discussed previously with a similar number of neurons and layers. The agent is compiled by assigning the respective neural network to the parameters required by the NAFagent class for initialization, as shown in Code Listing 20. The OrnsteinUhlenbeckProcess is the process used for the choosing of the actions for exploration.

```
        # Compiling of the agent
        processor = PIDProcessor()
        memory = SequentialMemory(limit=20000, window_length=1)
        random_process = OrnsteinUhlenbeckProcess
                          (theta=.15, mu=0., sigma=.3, size=nb_actions)
        agent = NAFAgent(nb_actions=nb_actions,
                         V_model=V_model, L_model=L_model,
                         mu_model=mu_model, memory=memory,
                         nb_steps_warmup=100,
                         random_process=random_process, gamma=.99,
                         target_model_update=1e-3, processor=processor)
        agent.compile(Adam(lr=.001, clipnorm=1.), metrics=['mae'])
```

Code Listing 20. Compilation of the NAF agent

4.8. CEM

The CEM follow the same configuration as the DQN in the initialization and the neural networks. But the CEM agent is much simpler in terms of the agent configuration. As represented in Code Listing 21, the memory is used for storing the reward history for the action taken. EpisodeParameterMemory is used for the same in the CEM agent, which stores defined state-action-reward mapping.

```
        # Compiling of the agent
        memory = EpisodeParameterMemory(limit=1000, window_length=1)
```

```
    cem = CEMAgent(model=model, nb_actions=nb_actions,
                   memory=memory, batch_size=50, nb_steps_warmup=2000,
                   train_interval=50, elite_frac=0.05)
    cem.compile()
```

Code Listing 21 Compilation of the CEM agent

4.9. SARSA

SARSA model is one of the simplest models. As Code Listing 22 depicts, only a handful of parameters are set for the same. The policy is set to BoltzmannQPolicy similar to the DQN agent. The SARSA agent does not require memory as described in the section 2.6.5 therefore, the same is not included in the compilation of the agent.

```
    # Compiling of the agent
    # SARSA does not require memory.
    policy = BoltzmannQPolicy()
    sarsa = SARSAAgent(model=model, nb_actions=nb_actions,
                       nb_steps_warmup=10, policy=policy)
    sarsa.compile(Adam(lr=1e-3), metrics=['mae'])
```

Code Listing 22 Compilation of the SARSA agent

4.10. Simple PID controller

The PID Controller based on the conventional PID integral-derivative method is used for benchmarking the DRL agent performances in balancing the PV value as close to the SP in the same given conditions. Code Listing 23 depicts the initialization of the process library, which is commonly used by the DRL agents also. This enforces the fairness of the performance metrics from both types of PID controllers. The initialization is similar to the DRL agents depicted before.

```
    ######################################
    # Initializing Process
    ######################################
    process = Process()

    # New PV is calculated by the process
    new_values = process.eq_evaluator(process.cv)
    # New sp value based on the equation
    process.sp = new_values['sp']
    # New pv value based on the equation
    process.pv = new_values['pv']
```

```
pid = PID(.005, 1, .0005, setpoint=process.sp, sample_time=0.01)
```

Code Listing 23. Initializing the Process Object

The for loop depicted in the Code Listing 24 runs the PID controller through 10000 iterations similar to the DRL agents. The code to record the environmental variables are captured through the Process library itself and collected for 100 runs using automated PowerShell script.

```
for i in range(0, 10000):
    process.cv = pid(process.pv)
    process.cv = round(process.cv, 5)
    new_values = process.eq_evaluator(process.cv)
    process.pv = new_values['pv']
    pid.setpoint = process.sp
    print('cv', process.cv, 'pv', process.pv)
    time.sleep(.01)
```

Code Listing 24. Running the PID controller

4.11. Results Analysis

This is result analyses start with reading all the files which are created to collect the system state, that is, the set-point and the process value. The data for each agent is segregated into a separate folder for reading the data separately. Initially, a sample file from each agent is loaded, and the general behaviour is analysed. After which, the first and last transition edges for each agent is isolated to get a better behavioural model of the agent performance. These steps will not have concluding evidence to analyse the performance.

4.11.1. Median calculation

As shown in Code Listing 25, the entire data is read and loaded into a single data frame for further analysis. This is achieved by reading the files for all the three agents one by one and loading the data into a previously defined data frame named mean_df. After which, the mean for first transition and last transition is captured for each dataset. This mean is loaded into the mean_df data frame mentioned earlier.

```
path = 'C:/pid_env/drl_pid/dsc/*/*'
# * means all if need specific format then *.csv
list_of_files = glob.glob(path)

mean_df = []

for file in list_of_files:
    a = pd.read_csv(file)
    a['error'] = abs(a['sp'] - a['pv'])

    first_transition = a.head(1500).tail(1000)
    first_transition_mean = first_transition["error"].mean()
    #print("first transition mean : " , mean)

    last_transition = a.tail(1000)
    last_transition_mean = last_transition["error"].mean()
    #print("last transition mean : " , mean)

    mean_df.append([first_transition_mean, last_transition_mean])

mean_df = pd.DataFrame(
    mean_df, columns=['first_transition_mean',
                      'last_transition_mean'])
```

Code Listing 25. Creation of the mean data frame of the entire dataset.

After this, the mean, max and min of the entire data set is calculated as shown in the Code Listing 26. This calculated for both first and last transition. This helps in understanding the range of the first transition and last transition mean for the entire dataset.

```
print("first_transition_mean : ",
      mean_df['first_transition_mean'].mean())
print("last_transition_mean : ",
      mean_df['last_transition_mean'].mean())
print("max_of_first_transition_mean : ",
      mean_df['first_transition_mean'].max())
print("max_of_last_transition_mean : ",
      mean_df['last_transition_mean'].max())
print("min_of_first_transition_mean : ",
      mean_df['first_transition_mean'].min())
print("min_of_last_transition_mean : ",
      mean_df['last_transition_mean'].min())
```

Code Listing 26. Calculation of min, max and mean of the entire dataset.

The median of all agents is calculated as a single dataset for both the first transition and the last transition as in Code Listing 27. This is a significant variable as this median value acts as the benchmark for the evaluation of individual agent performance. The stability of the agent is calculated by counting the instances that the agent was able to keep the mean-variance of PV from SP under the median for the transition period.

```
ft_median = mean_df['first_transition_mean'].median()
lt_median = mean_df['last_transition_mean'].median()
print("first_transition_median : ", ft_median)
print("last_transition_median : ", lt_median)
```

Code Listing 27. Medium calculation of the mean data frame.

4.11.2. Agent stability factor calculation

The median values calculated for the entire dataset is then used to count the number of agent execution which have mean above the value. These counts will be used further to compare the stability of the agents. The aggregated mean, min, max of the mean of all the executions of the agent is calculated for both the first and the last transition similar to the calculation of the same metrics for the entire dataset as described in 4.11.1. As shown in the Code Listing 28, the count of the instances of agent execution where the first and last transition was above or below the median is calculated respectively. The same process is repeated for the agents. These values are captured in the comparison table in section 5.2.5.

```
    print("count_of_ft_mean_above_median : ",
    mean_df[mean_df['first_transition_mean']
    .gt(ft_median)]['first_transition_mean'].count() )
    print("count_of_ft_mean_below_median : ",
    mean_df[mean_df['first_transition_mean']
    .lt(ft_median)]['first_transition_mean'].count() )

    print("count_of_lt_mean_above_median : ",
    mean_df[mean_df['last_transition_mean']
    .gt(lt_median)]['last_transition_mean'].count() )
    print("count_of_lt_mean_below_median : ",
    mean_df[mean_df['last_transition_mean']
    .lt(lt_median)]['last_transition_mean'].count() )
```

Code Listing 28. Counting the number of instances the mean of the iteration was below and above the median.

4.12. Summary

This chapter was used to explain the implementation of the experiment. Initially, the process of artefact implementation was detailed. After which the environmental behaviour realization was explained. Furthermore, the agent's compilation and execution details were presented. Finally, the method of result analysis using the automated code was justified.

5. Results and Evaluation

5.1. Introduction

This chapter presents the execution results of the agents. A brief explanation of the execution strategy is presented. Then the overall execution pattern of each agent is plotted. Afterwards, an isolated transition of each agent for both first and last instances for the sample run is discussed in detail to provide some insights to the behaviour. Finally, a comparison report between the agents and a brief discussion on the same is presented.

5.2. Execution of the Agents

As described in Table 2, the agents are executed in the respective environment, and the general behaviour is captured in this section. The execution of agent is carried out by a PowerShell script which automatically runs a hundred iterations of the agent sequentially as shown in the Code Listing 29. The agents are executed a hundred times in order to analyse the general behaviour of the agent and derive an approximation.

```
for ($i = 0; $i -lt 100; $i++) {
    try { python dqn_pid_discrete.py }
    catch { continue }
}

for ($i = 0; $i -lt 100; $i++) {
    try { python sarsa_pid_discrete.py }
    catch { continue }
}

for ($i = 0; $i -lt 100; $i++) {
    try { python cem_deep.py }
    catch { continue }
}

for ($i = 0; $i -lt 100; $i++) {
    try { python pid.py }
    catch { continue }
}
```

Code Listing 29. PowerShell script to run the agents sequentially.

5.2.1. DQN agent in a discrete environment

This section provides the analysis of a random sample of the agent execution. As shown in Figure 23, the agent had done a negative exploration before learning to balance the PV with CV. In the ten transitions, every edge has a spike of the PV as the value of the SP changes, and the agent had to readjust its learning parameters accordingly. But a return of the SP value towards the range previously it had encountered the stability is higher as expected.

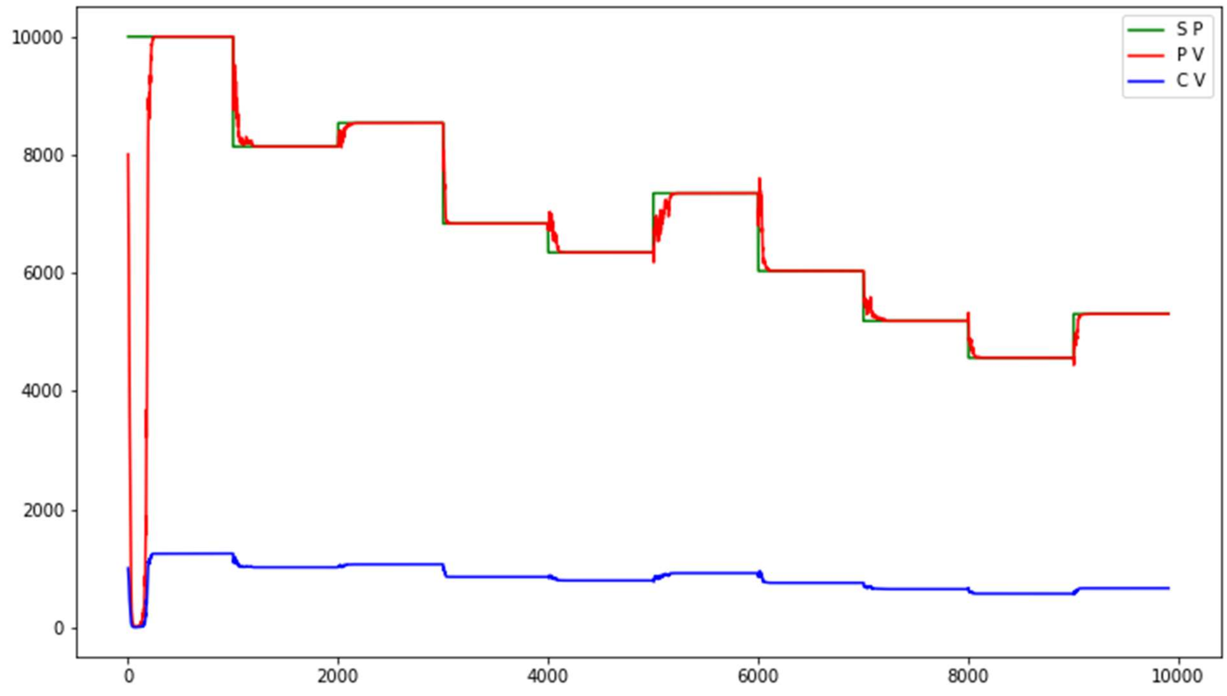

Figure 23. DQN agent execution sample

Now the first transition of the same execution is isolated in order to analyse it deeper, as shown in Figure 24. As observed in the figure, the DQN converges the error of PV from the SP towards 0 in around 50 steps. The mean of the error over the first 100 steps is taken into account for calculating the mean error of the first transition in the comparison report.

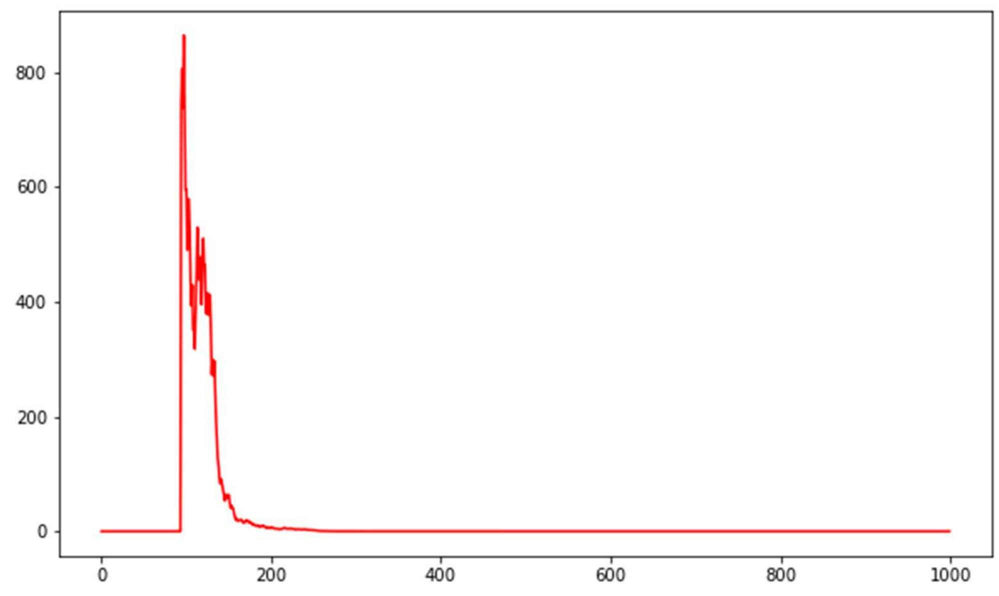

Figure 24. First transition sample for DQN Execution

Figure 25 is the detailed plot of the SP-PV error value for the last transition in the sample taken from the agent execution depicted in Figure 23. This last transition values will be analysed to derive the mean value, which in turn used to compare the mean error between the agent to measure the learning performance.

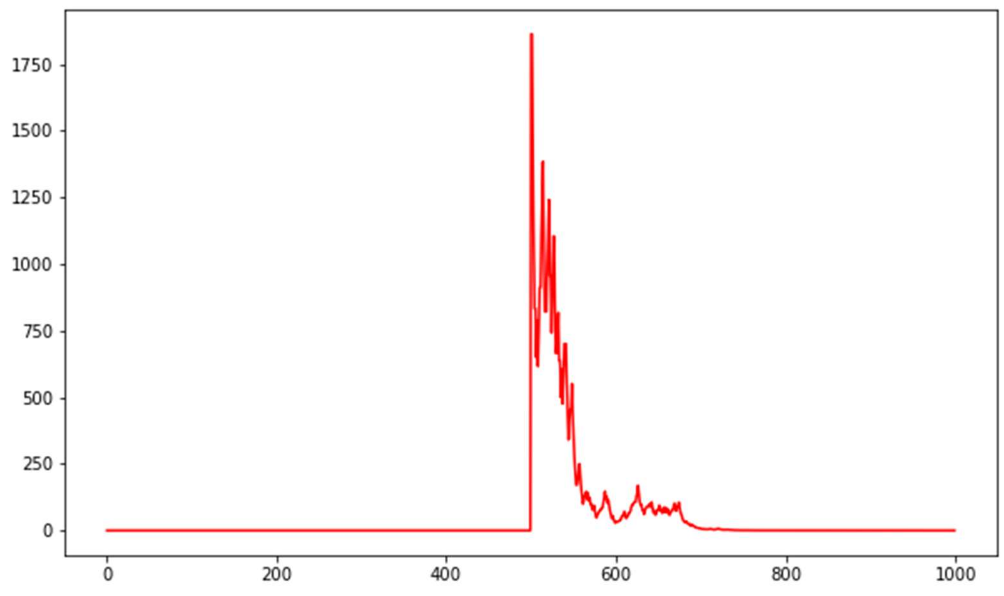

Figure 25. Last transition sample for DQN Execution

5.2.2. SARSA agent in a discrete environment

SARSA agent also is plotted similarly like the DQN agent. Coincidently the sample which is plotted playouts the best the case scenarios. But on further analysis of the data for 100 execution, the stability of the agent is lower than DQN, which will be further discussed in the comparison report.

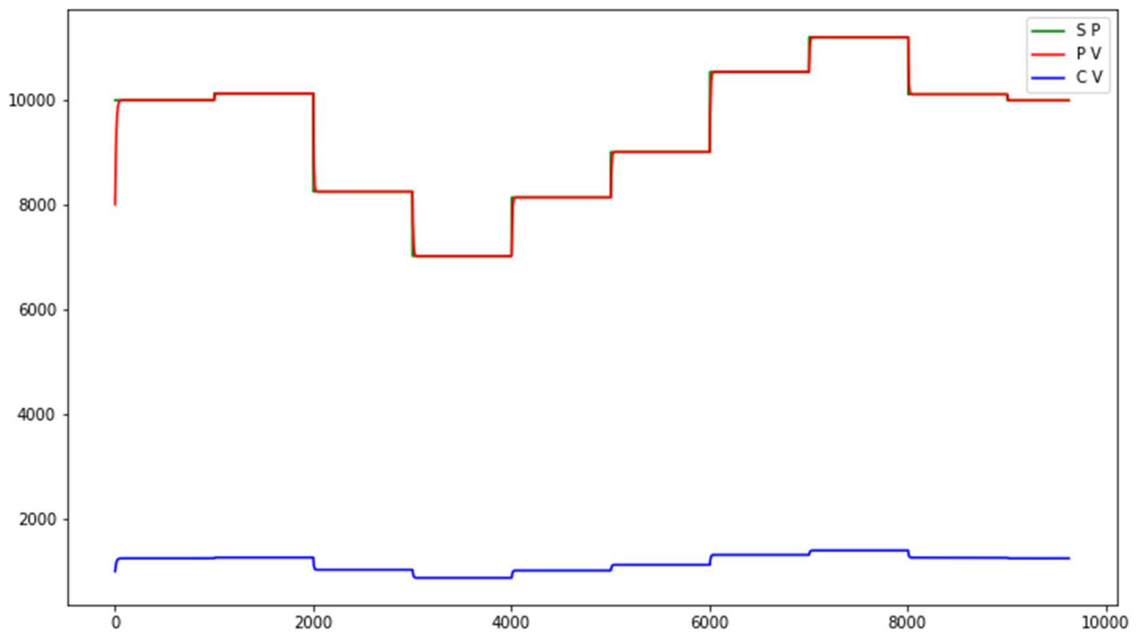

Figure 26. SARSA agent execution sample

Similarly to the DQN agent Figure 27 and Figure 28 represents the first and last transition of the SARSA agent, respectively. As mentioned earlier, this sample represents the best case scenarios of the SARSA where the transition are smooth. The SP-PV error converges to zero very fast in a few numbers of steps. This is true for both the first and last transition of the SARSA agent.

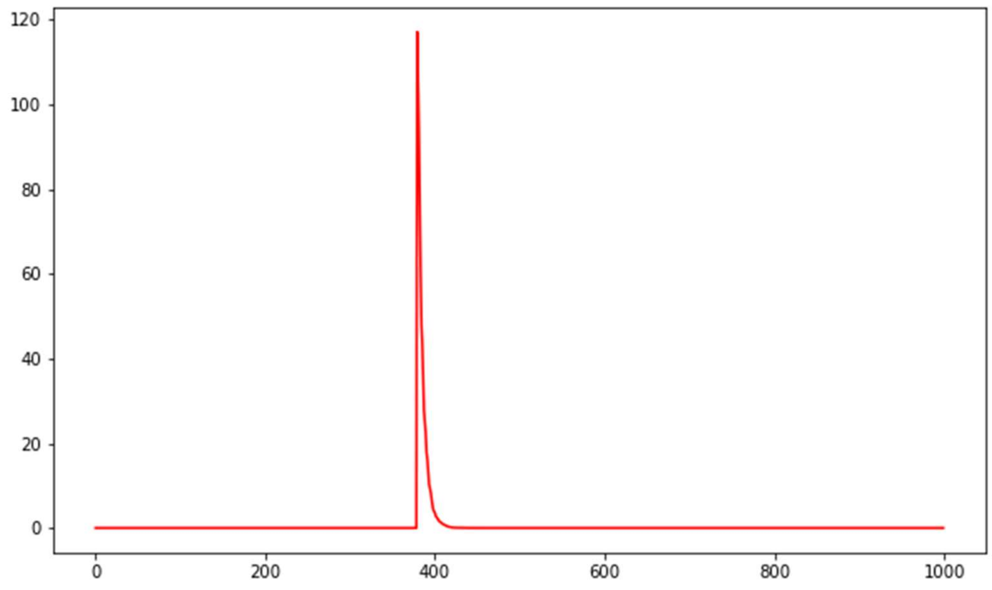

Figure 27. First transition sample for SARSA Execution

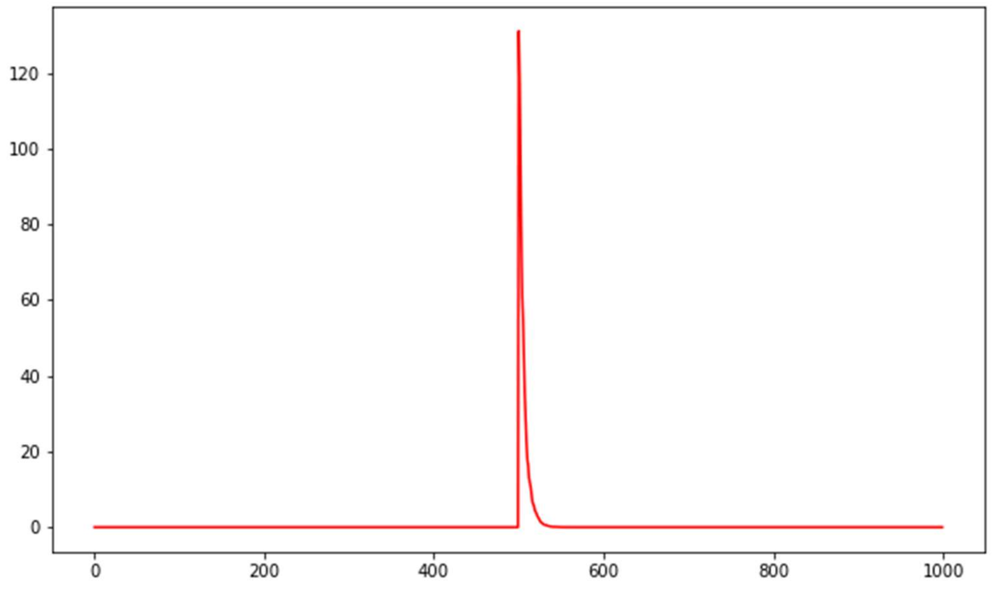

Figure 28. Last transition sample for SARSA Execution

5.2.3. CEM agent in a discrete environment

CEM agent is one of the primitive agents available in the Keras_RL library. Figure 29 is the data plot of one of the executions of the CEM agents for the entire 10000 steps. As observable in the plot, the agent has a high variance in the initial transition. But over time, the agent has stabilised which have learned to reduce the SP-PV error significantly in the later transitions.

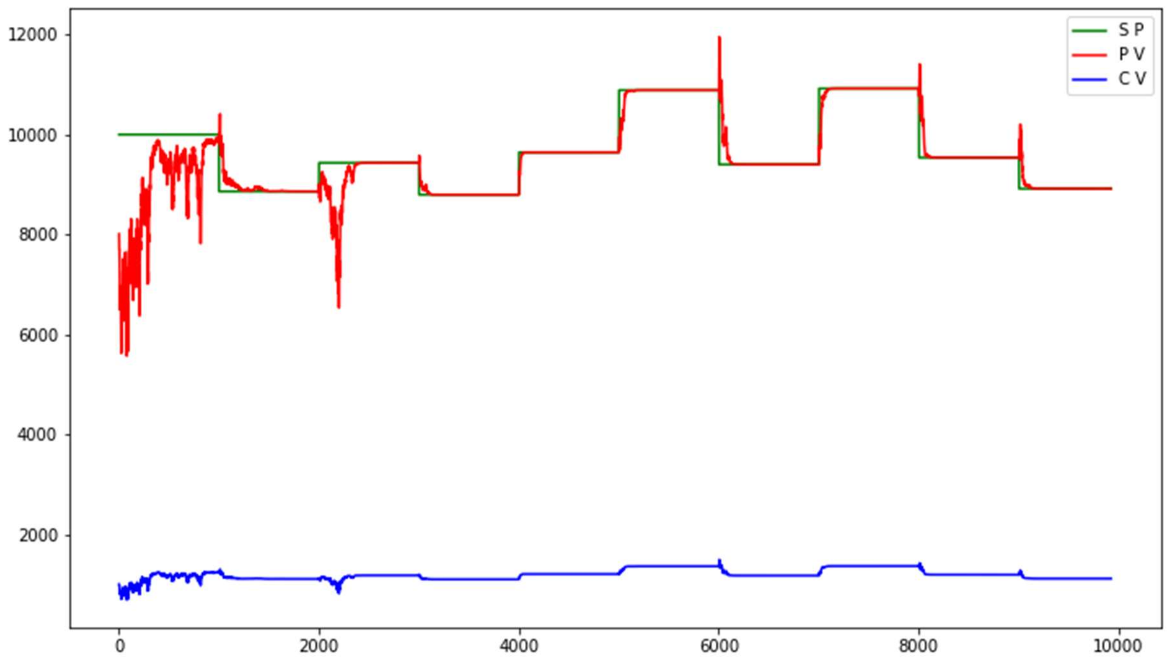

Figure 29. CEM agent execution sample

To analyse the behaviour of the CEM for the first and last transitions, Figure 30 and Figure 31 are plotted. As clearly visible in Figure 30, the CEM agent was not able to converge the error to zero in the first 1000 steps. Compared to the DQN and SARSA, the mean error also has remained high in the first transition. But in the last transition, the situation is ameliorated drastically.

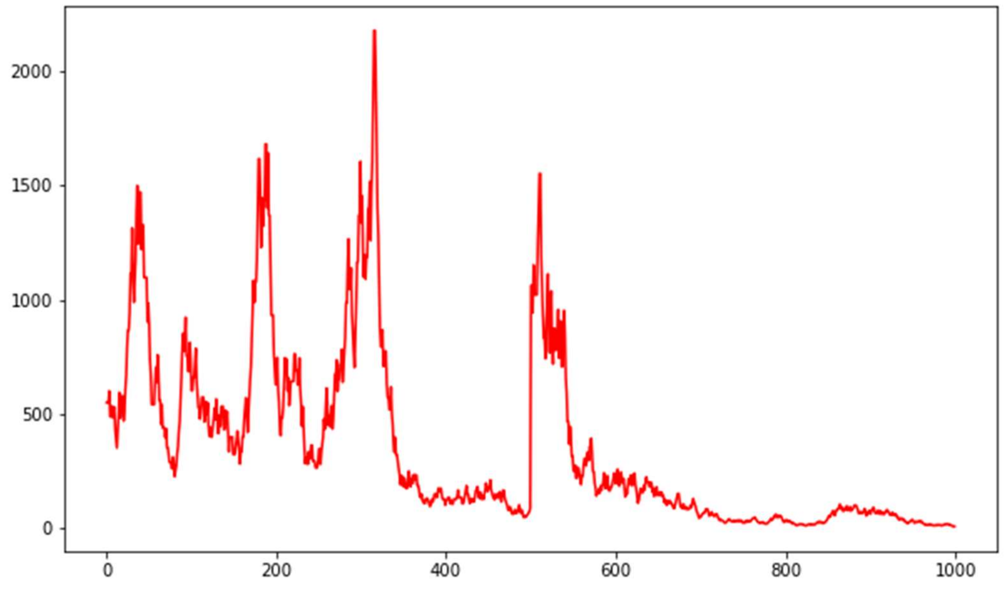

Figure 30. First transition sample for CEM Execution

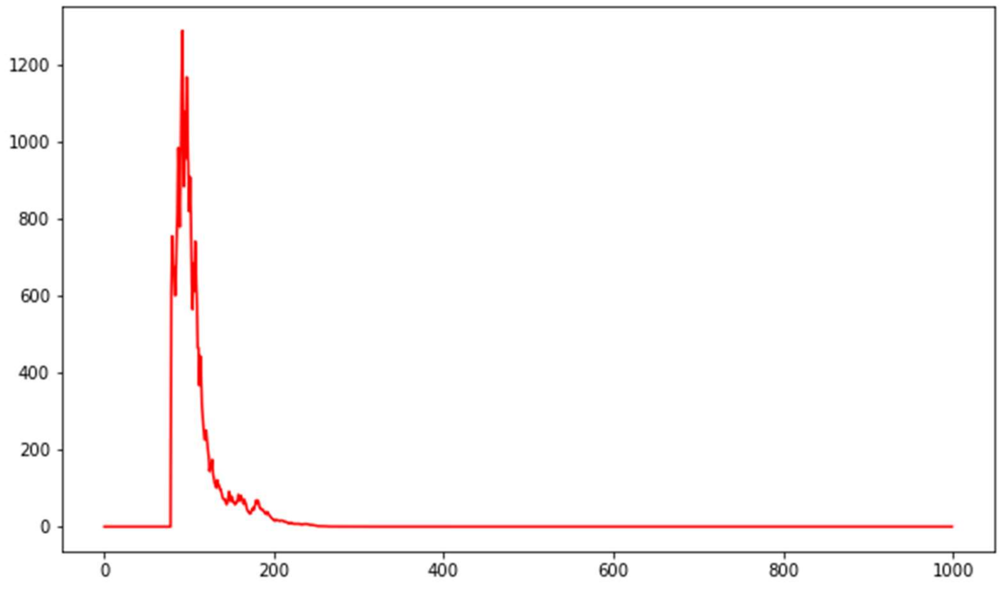

Figure 31. Last transition sample for DQN Execution

5.2.4. DDPG agent and NAF agent in a continuous environment

The DDPG and NAF agents were not able to converge the SP-PV error with the given environmental parameters. Figure 32 is the plot of the DDPG environment variables for the entire execution. As noticeable, the PV value remains unchanged irrespective of the SP value. This may be caused by the action and reward map is too huge for the current neural network to learn. The continuous values of the action space will have an infinite number of values to map to the reward function. This makes the DDPG and NAF agents fail to converge in the continuous environment.

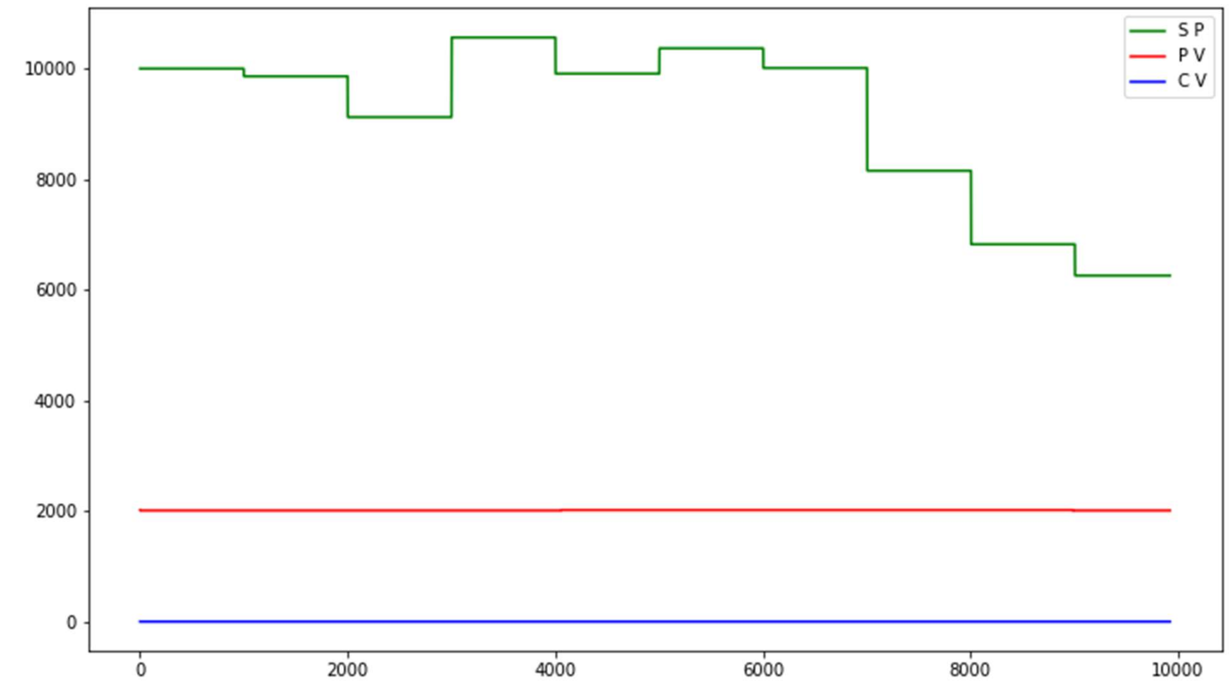

Figure 32. Execution of the DDPG agent in a continuous environment.-

5.2.5. PLC-PID Controller execution for benchmarking

The PLC-PID controller with same process state was executed to have a comparison of the convergence rate and stability of the agents. As observable in Figure 33, thePLC-PID controller exhibits high stability and reliable convergence rate.

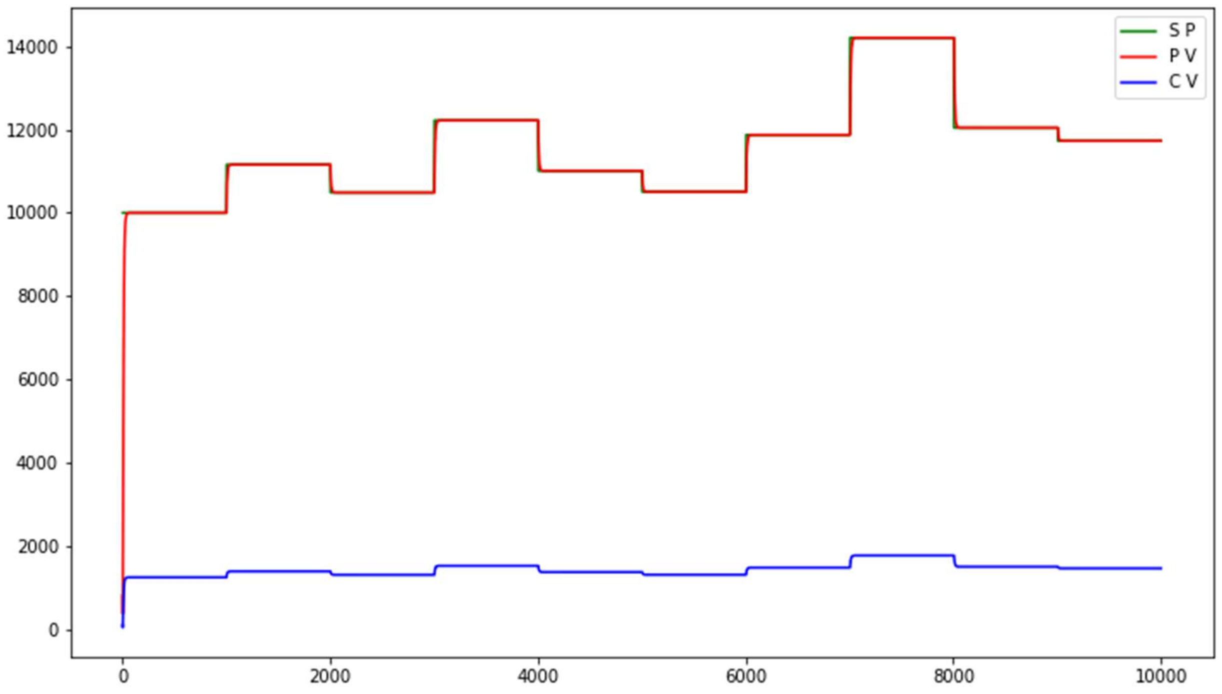

Figure 33. PLC-PID controller execution

In contrast to the DRL agents, the first transition plotted in Figure 34 is very similar to the last transition represented in Figure 35. This because the PID logic is based on direct mathematical equations which are set in advance. This leads to a calibration of the PID parameters every time there is a change in the range of any of the environmental variables.

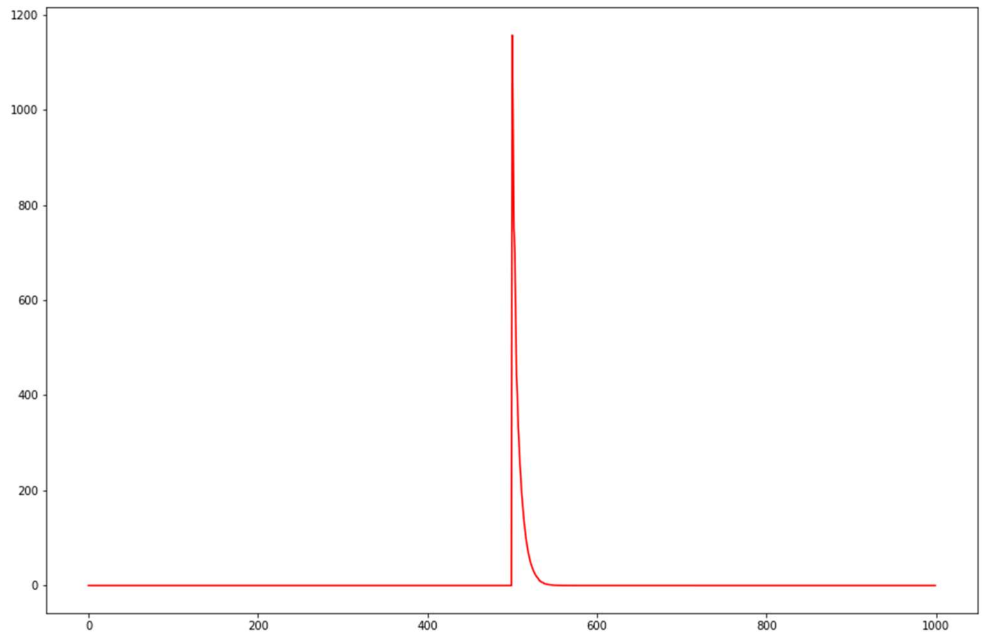

Figure 34. First transition of the PLC-PID logic controller

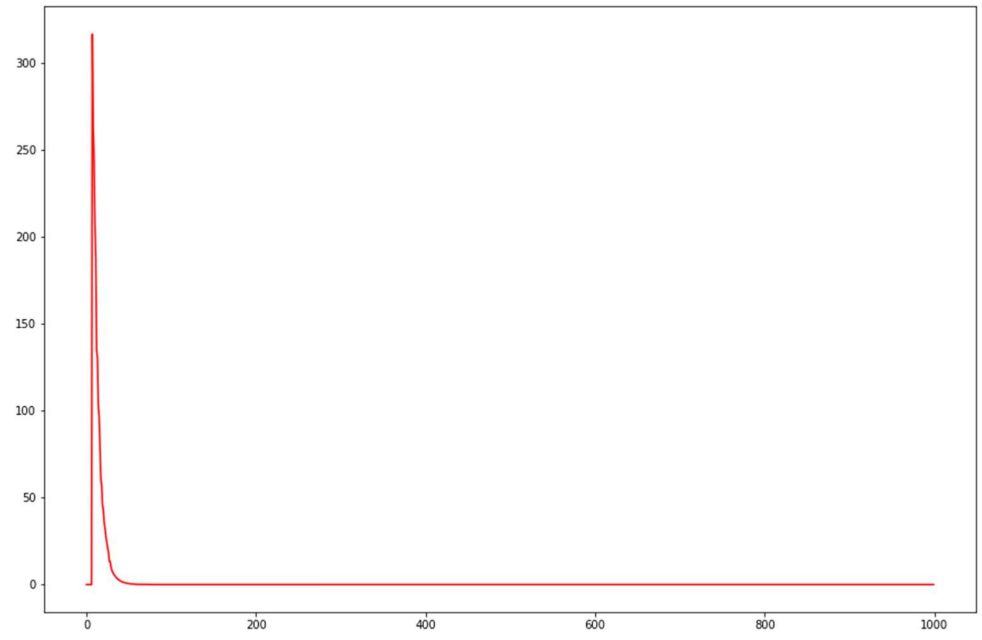

Figure 35. Last transition of the PLC-PID logic controller

5.3. Comparison of execution data logs

This comparison section is used to compare the DRL-agent performance with respect to the PLC-based PID logic. This also has comparative analysis between the three successful agents which were able to converge to SP-PV error towards zero. As shown in Table 3, the agent value for varies metrics such mean, median, max, min and counts of instances over the median are listed. The columns are segregated for each type of agent, and one column is used to specify if the value is for the first or the last transition. As expected, the first transition values of the DRL-PID agents will be higher than the last transitions. The data in this table is further discussed in detail under different sections.

Table 3. Comparison Report for the agents

Agent Metrics	First/Last Transition	PLC	Aggregate of all DRL agents	DQN	SARSA	CEM
Mean	FT	6.53	3103.61	1004.97	7931.88	373.96
Mean	LT	7.57	1805.40	24.26	4568.74	823.19
Median	FT	6.32	317.06	19.20	1717.60	369.06
Median	LT	6.41	38.65	21.61	1399.09	61.36
Max	FT	15.63	388932.94	10840.10	388932.94	496.65
Max	LT	26.41	17430.41	71.16	14942.37	17430.41
Min	FT	0.17	0.18	0.31	0.18	267.50
Min	LT	.104	0.04	0.65	0.56	0.04
Counts above median	FT	NA	NA	14	55	81
Counts below median	FT	NA	NA	86	45	19
Counts above median	LT	NA	NA	20	60	70
Counts below median	LT	NA	NA	80	40	30

5.3.1. Mean analysis

In terms of the mean of the SP-PV error, as expected, the first transition has a higher value for the DRL agents than the PLC. But in the last transitions, the DQN have achieved a comparable value with respect to the PLC-PID controller. SARSA have exhibited the highest error rate for the mean value of both first and last transition. CEM agent maintains a medium performance compared to the other two DRL-PID agents.

5.3.2. Min and Max analysis

Similar to the mean analysis, the SARSA agent shows a high error rate for the maximum values which have created a skew in the dataset for the same. The PLC-PID maintain the best case values for the maximum mean rate. In comparison, the DQN and CEM hold the respective next positions in terms of maximum error rates. But the case of the minimum is unexpected as the SARSA have shown a better result than the PLC-PID controller. This is because of the high exploration rate of the SARSA agent. But with the high exploration rate, the risk factor of having extreme values increases as demonstrated in the max value analysis.

5.3.3. Median and stability analysis

The median is calculated for all DRL-PID agent data. This is to used further to determine the stability of each agent. As mentioned in section 4.11.2, the count of instances of each agent which have values higher than the median and below the median is calculated. As the last transition represents the agent's performance after the learning process, the values for the same holds significance than the first transition. As captured in Table 3, the DQN has 80 values below the median and 20 above. Therefore the DQN agent exhibits the highest stability out of all the three DRL-PID agents. The CEM agent has 70 and 30 counts of instances for above and below the median, respectively. This indicates that the CEM has least the stability out of three agents under comparison. SARSA agent has 60 instances above the median, which is less than DQN but better than the CEM.

5.4. Summary

The agents were executed in an automated manner as described initially in the chapter. First, the discrete agents were run and the data was collected for a hundred runs. Then one instance of each agent was critically analysed in terms of the general behaviour on the convergence. Afterwards, the first and last transition of the same instance was further analysed in detail. A brief comparison is presented while each agents behaviour is analysed. Further, the continuous environment based agents were executed. However as described in detail in section 5.2.4, the agents were not able to converge. For the benchmarking purposes, the PLC-based PID block was executed and the performance was recorded a brief discussion was presented. Finally, a comparison of the performance was presented in terms of the mean, min, max, median and stability factors of the agents. As a conclusion, the CEM agents performance was not incomparable range with the other two agents. DQN have better stability in maintaining the PV values consistently than SARSA. But the SARSA shows a better perspective in terms of lowering the error rate in some instances.

6. Conclusion

This chapter is focused on providing a summary of the experiment and useful insights for further research in this field. The research summary provides the insights derived from this experiment. Research summary also provides integration details to the existing industry 4.0 initiatives on the application of artificial intelligence. Further, the limitations of this research are listed in the next section. Finally, the enhancements which are possible to the current project and promising research fields for the future are briefly discussed.

6.1. Research summary

The objective of this research was to evaluate the current maturity level of the deep reinforcement learning framework for the implementation of the critical logic controller blocks of industrial automation. Recent advancements in deep learning and reinforcement learning has enabled the implementation of various deep reinforcement learning in an efficient manner than the previous generation of deep reinforcement learning agent solutions. Edge computing capabilities have drastically matured in the last decade with the arrival of IoT devices with higher resilience. This has allowed the engineer to develop solution platform closer to the data collection and control points of the industrial automation systems. There were two types of environmental representation of the PID logic which was created: discrete and continuous. The environments were created according to the Open AI Gym framework standards. A deep reinforcement learning framework named Keras-RL was used to create three discrete data processing agents and two continuous values processing agent. An additional module of Process Simulation was created to mimic the real-world process values such as SP, CV and PV. In this experiment with the given constraints of the Edge computing capabilities, only the discrete type agents were able to converge in the solution space of the PID logic. The continuous agents DDPG and NAF was either saturated at a low value or exploded to a value out of range. Out of the discrete three agents which were able to converge the SP-PV error value close to zero, SARSA had the least mean value over a hundred iterations but the stability was better for the DQN agent. These results were compared with the conventional PID block and a comparative analysis was presented. The overall conclusion is the DRL agents had performed in the comparable range of accuracy and stability of PLC PID controllers at some instances but overall stability was better for the PLC-PID logic.

As described in the previous sections, the previous PID controllers were designed and modulated case by case. The autonomous ability was limited to the core function of tuning parameters and normalization of the sensor input was mandated. This project is an implementation of a self-tuning PID controller without explicit normalization of the sensors. Another distinction is that a complete deep reinforcement learning package is used for the implementation in this research. As a by-product, two new environments were created, which can be added to the of the OpenAI Gym community. These environments can be used for benchmarking future DRL agents which are used design self-tuning PID controllers.

This dissertation has presented a novel combination of both deep neural network and reinforcement learning tools to implement one of the core logic controllers in the field of industrial automation: PID controller. Even though there has been a previous attempt at this goal, it was not able to achieve the same level of accuracy as the new frameworks such as Keras and OpenAI gym. Keras framework enables the adaptability of the codebase into the edge nodes without any changes, making it close to platform-independent implementation with the python execution engine. OpenAI gym provides the standard set of rules to develop the environment; therefore, it can be used to test the various agents developed without any bias based on the technology stack. The solution presented in the dissertation uses the agent class provided by the library Keras-RL which is developed using the common interfaces with OpenAI Gym, which gives seamless integration of the two independent frameworks.

The comparison report of the agents set a benchmark on the standards of the DRL-PID controllers and provides insights on how we can develop new agents more efficiently. The continuous environment was challenging to solve. The complexity in handling the continuous observation and action space leads to infinite mapping. This prevented the DDPG agent from converging to the desired value and caused the NAF agent to explode out of range. The dataset median value was used in determining the stability factor for the discrete agents, that is, DQN, SARSA and CEM. DQN came out as the best in terms of overall mean values and stability whereas the SARSA showed better performance in isolated instances which indicates that the if we can bring in more stability to SARSA it can exhibit comparable performance to the PLC-PID controller.

The primary research question of this dissertation was “Can modern deep reinforcement learning frameworks be used to implement controller logic in an edge computing device for industry automation?”. The successful implementation of DRL agents for the PID controller logic directly answers the primary research question. The modern deep reinforcement learning framework possess excellent candidature for the implementation of logic controllers in edge computing devices.

One of the sub-question of the research was “Which type of state representation is optimal: discrete or continuous space?” As mentioned earlier, the discrete space is more optimal for the state representation in the resource constraint environment.

Another sub-question was “Are the agents provided by the Keras-RL framework applicable in an industrial environment”. Keras-rl agents with its platform independency and seamless integration with the OpenAI gym framework makes it easy to implement and test the logic in industrial environments.

Therefore, deep reinforcement learning is a promising frontier to achieve human-level controller behaviour in the industrial automation field. The auto-tuning capability, which is an advantage of DRL-PID over PLC-PID creates a unique opportunity to reduce the risk of human error in automation projects for Industry 4.0.

6.2. Limitations

The currently identified the limitations of this dissertation are:

- The environment design is based on empirical experimentation of agent capabilities and behaviour. This may need to be modified to make it robust enough to accept other types of agents.
- The deep neural network size is limited in order to adapt to the current capabilities of edge computing devices; these may need to be re-evaluated based on the progression of the technology.
- The agents implemented in this thesis are limited to the agent classes provided by the library Keras-rl.

- With the current scope and budget of the project, testing on a real edge node and comparing the results with a real PLC was not feasible.

6.3. Future Works

6.3.1. CNN networks to handle the continuous environment

As mentioned in the 5.2.4, the DDPG and NAF agents were not able to converge in the continuous environment. One of the reasons for this was the continuous values of the action space and rewards. This can be tackled by using the convolutional operations to reduce the value range in which the neural network needs to be calculated. Another possibility is to use the CNN network to manage the degree of the polynomial as it increases. This will allow the agent to achieve a better convergence pattern and eventually leading to better stability. This will enable the application of the Transfer learning methods to transfer the convolutional filters based on the application area.

6.3.2. A second agent to determine the degree of the process complexity

Currently, the agent has a single neural network to handle the environment variables. This leads to complex intertwined relationships resulting in less accuracy. But if the environmental variables are segregated for the various behaviour, such as the degree of the process complexity, then the distinguishable features in the various dimension will have better performance. This segregation can decrease the count of instances of the agents which have a higher error rate for the SP-PV difference.

6.3.3. Visualization of the state parameters and agent actions

Currently, this project relies on the data collection to analyse the agent behaviour, real-time visual feedback to the user is not implemented in the current environment. Similar to the other environment available in the OpenAI Gym, a visualization of the PID controller loop can be implemented which will enhance the user experience of the agent testing. This could be achieved using a MATLAB plot lib as the native python execution program.

6.3.4. Testing on the physical machines

This experiment was run on the windows based python compilers, but the platform independence python programs enable it to run on Linux or embedded systems alike without any other modifications. However, the performance of the agents was compared in terms of the turnaround time and convergence speed. The deep learning agent's performance can have slight variations when run with limited memory. Another aspect is the latency between the decision matrix and control point. Currently, the PLC-PID logic runs on specialized hardware with high response time but on a RISC architecture. But this experiment proposes the use of CISC based architecture, computer to replace the PLC's for control solutions. But actual execution time on the CISC may not affect the performance capability of the DRL agents. Therefore, a test of this solution in comparison with a physical PLC will deliver more accurate results on the maturity level of the Edge-based control solutions.

6.3.5. Industry to consumer products

Current solutions concentrate on the edge computing based DRL agents for the industrial PID problem space. But the PID logic is also used in the consumer products like Air conditioners, Heating systems and auto navigational transport vehicles. This solution can be adapted for such products given the platform supports a python execution engine. This can highly useful as the customization in consumer markets is much higher than the industrial applications.

Appendix A: References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mane, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viegas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X., 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems 19.

ABB Industrial IoT applications | Internet of Things, Services and People - What is new [WWW Document], 2020. URL https://new.abb.com/control-systems/features/industrial-IoT-services-people-use-cases (accessed 7.5.20).

Agents and Environments Keras-RL Documentation [WWW Document], n.d. URL https://keras-rl.readthedocs.io/en/latest/agents/overview/ (accessed 10.20.20).
Araki, T., Nakamura, Y., 2017. Future trend of deep learning frameworks — From the perspective of big data analytics and HPC, in: 2017 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC). Presented at the 2017 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), IEEE, Kuala Lumpur, pp. 696–703. https://doi.org/10.1109/APSIPA.2017.8282122

Arulkumaran, K., Deisenroth, M.P., Brundage, M., Bharath, A.A., 2017. Deep Reinforcement Learning: A Brief Survey. IEEE Signal Process. Mag. 34, 26–38. https://doi.org/10.1109/MSP.2017.2743240

Bellman, R.E., 1978. An Introduction to Artificial Intelligence: Can Computers Think?, illustrated edition edition. ed. Boyd & Fraser Pub. Co, San Francisco.

Bolton, W., 2006. Programmable Logic Controllers, 4 edition. ed. Newnes, Amsterdam.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W., 2016. OpenAI Gym. arXiv:1606.01540 [cs].

Calo, S.B., Touna, M., Verma, D.C., Cullen, A., 2017. Edge computing architecture for applying AI to IoT, in: 2017 IEEE International Conference on Big Data (Big Data). Presented at the 2017 IEEE International Conference on Big Data (Big Data), IEEE, Boston, MA, pp. 3012–3016. https://doi.org/10.1109/BigData.2017.8258272

Casado-Vara, R., Chamoso, P., De la Prieta, F., Prieto, J., Corchado, J.M., 2019. Non-linear adaptive closed-loop control system for improved efficiency in IoT-blockchain management. Information Fusion 49, 227–239. https://doi.org/10.1016/j.inffus.2018.12.007

Cha, Y.-J., Choi, W., Buyukozturk, O., 2017. Deep Learning-Based Crack Damage Detection Using Convolutional Neural Networks. Computer-Aided Civil and Infrastructure Engineering 32, 361–378. https://doi.org/10.1111/mice.12263

Charniak, E., McDermott, D., 1985. Introduction to Artificial Intelligence: Addison-Wesley Series in Computer Science. Addison Wesley Longman Publishing Co, Reading, Mass.

Chen, B., Wan, J., Shu, L., Li, P., Mukherjee, M., Yin, B., 2018. Smart Factory of Industry 4.0: Key Technologies, Application Case, and Challenges. IEEE Access 6, 6505–6519. https://doi.org/10.1109/ACCESS.2017.2783682

Core - Keras-RL Documentation [WWW Document], 2020. URL https://keras-rl.readthedocs.io/en/latest/core/ (accessed 4.3.20).

Dean, J., Corrado, G.S., Monga, R., Chen, K., Devin, M., Le, Q.V., Mao, M.Z., Ranzato, M., Senior, A., Tucker, P., Yang, K., Ng, A.Y., 2011. Large Scale Distributed Deep Networks 11.

Frome, A., Corrado, G.S., Shlens, J., Bengio, S., Dean, J., Ranzato, M., Mikolov, T., n.d. DeViSE: A Deep Visual-Semantic Embedding Model 11.

Gu, S., Lillicrap, T., Sutskever, I., Levine, S., 2016. Continuous Deep Q-Learning with Model-based Acceleration. arXiv:1603.00748 [cs].

Gullì, A., Pal, S., 2017. Deep learning with Keras: implement neural networks with Keras on Theano and TensorFlow. Packt, Birmingham Mumbai.

Gym Spaces, 2018.

Gym Wrappers, 2019.

Haesaert, S., Esmaeil, S., Soudjani, S., Abate, A., 2017. Verification of general Markov decision processes by approximate similarity relations and policy refinement. SIAM Journal on Control and Optimization 55, 2333–2367. https://doi.org/10.1137/16M1079397

Haugeland, J., 1989. Artificial Intelligence: The Very Idea. MIT Press.

Heddy, G., Huzaifa, U., Beling, P., Haimes, Y., Marvel, J., Weiss, B., Minnick, A.E.L., 2015. Linear Temporal Logic (LTL) based monitoring of smart manufacturing systems, in: PHM 2015 - Proceedings of the Annual Conference of the Prognostics and Health Management Society

2015. Presented at the 2015 Annual Conference of the Prognostics and Health Management Society, PHM 2015, Prognostics and Health Management Society, pp. 640–649.

Hooper, J.F., 2006. Introduction to PLCs, Second Edition, 2 edition. ed. Carolina Academic Press, Durham, N.C.

Hu, T., Wu, Q., Zhou, D.-X., 2019. Distributed kernel gradient descent algorithm for minimum error entropy principle. Applied and Computational Harmonic Analysis S1063520318300514. https://doi.org/10.1016/j.acha.2019.01.002

Improving the Beginner's PID – Introduction « Project Blog, 2018. URL http://brettbeauregard.com/blog/2011/04/improving-the-beginners-pid-introduction/ (accessed 7.8.20).

Industrial Internet of Things by Honeywell [WWW Document], 2020. URL https://www.honeywellprocess.com/en-US/online_campaigns/IIOT/Pages/index1.html (accessed 7.5.20).

Iyer, A., 2018. Moving from Industry 2.0 to Industry 4.0: A case study from India on leapfrogging in smart manufacturing. Procedia Manufacturing 21, 663–670. https://doi.org/10.1016/j.promfg.2018.02.169

Jin, H., Song, Q., Hu, X., 2019. Auto-Keras: An Efficient Neural Architecture Search System, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD '19. Association for Computing Machinery, Anchorage, AK, USA, pp. 1946–1956. https://doi.org/10.1145/3292500.3330648

Kawaguchi, R., Bandai, M., 2020. Edge Based MQTT Broker Architecture for Geographical IoT Applications. 2020 International Conference on Information Networking (ICOIN), Information

Networking (ICOIN), 2020 International Conference on 232–235. https://doi.org/10.1109/ICOIN48656.2020.9016528

Keras Documentation [WWW Document], 2020. URL https://keras.io/ (accessed 12.22.19).
Keras-tuner, 2020. . Keras.

Kim, W., Sung, M., 2017. Poster Abstract: OPC-UA Communication Framework for PLC-Based Industrial IoT Applications, in: 2017 IEEE/ACM Second International Conference on Internet-of-Things Design and Implementation (IoTDI). Presented at the 2017 IEEE/ACM Second International Conference on Internet-of-Things Design and Implementation (IoTDI), pp. 327–328.

Krizhevsky, A., Sutskever, I., Hinton, G., 2012. ImageNet Classification with Deep Convolutional Neural Networks. Neural Information Processing Systems 25. https://doi.org/10.1145/3065386

Kurzweil, R., 1990. The Age of Intelligent Machines. MIT Press, Cambridge, Mass.

Lavaei, A., Somenzi, F., Soudjani, S., Trivedi, A., Zamani, M., 2020. Formal Controller Synthesis for Continuous-Space MDPs via Model-Free Reinforcement Learning, in: 2020 ACM/IEEE 11th International Conference on Cyber-Physical Systems (ICCPS). Presented at the 2020 ACM/IEEE 11th International Conference on Cyber-Physical Systems (ICCPS), IEEE, Sydney, Australia, pp. 98–107. https://doi.org/10.1109/ICCPS48487.2020.00017

Le, Q.V., 2013. Building high-level features using large scale unsupervised learning, in: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing. Presented at the ICASSP 2013 - 2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, Vancouver, BC, Canada, pp. 8595–8598. https://doi.org/10.1109/ICASSP.2013.6639343

Li, L., 2009. A unifying framework for computational reinforcement learning theory.

Liao, H.L., Wu, Q.H., 2010. Multi-objective optimisation by reinforcement learning, in: IEEE Congress on Evolutionary Computation. Presented at the IEEE Congress on Evolutionary Computation, pp. 1–8. https://doi.org/10.1109/CEC.2010.5585972

Likins, M., 2016. Improve Process Efficiency Through Regulatory Control Stabilization 11.

Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D., 2019. Continuous control with deep reinforcement learning. arXiv:1509.02971 [cs, stat].

Liu, F., Tang, G., Li, Y., Cai, Z., Zhang, X., Zhou, T., 2019. A Survey on Edge Computing Systems and Tools. Proc. IEEE 107, 1537–1562. https://doi.org/10.1109/JPROC.2019.2920341

MATLAB Documentation [WWW Document], 2020. URL https://uk.mathworks.com/help/matlab/index.html (accessed 9.29.20).

Mikolov, T., Chen, K., Corrado, G., Dean, J., 2013. Efficient Estimation of Word Representations in Vector Space. arXiv:1301.3781 [cs].

Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T.P., Harley, T., Silver, D., Kavukcuoglu, K., 2016. Asynchronous Methods for Deep Reinforcement Learning. arXiv:1602.01783 [cs].

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D., 2015. Human-level control through deep reinforcement learning. Nature 518, 529–533. https://doi.org/10.1038/nature14236

Mukhopadhyay, R., Bandyopadhyay, S., Sutradhar, A., Chattopadhyay, P., 2019. Performance Analysis of Deep Q Networks and Advantage Actor Critic Algorithms in Designing Reinforcement Learning-based Self-tuning PID Controllers, in: 2019 IEEE Bombay Section Signature Conference (IBSSC). Presented at the 2019 IEEE Bombay Section Signature Conference (IBSSC), IEEE, Mumbai, India, pp. 1–6. https://doi.org/10.1109/IBSSC47189.2019.8973068

Nilsson, N.J., 1998. Artificial Intelligence: A New Synthesis, 1 edition. ed. Morgan Kaufmann Publishers, Inc., San Francisco, Calif.

Normalized Advantage Functions — Reinforcement Learning Coach 0.12.0 documentation [WWW Document], 2020. . Normalized Advantage Function. URL https://nervanasystems.github.io/coach/components/agents/value_optimization/naf.html#choosing-an-action (accessed 7.2.20).

NumPy [WWW Document], 2020. URL https://numpy.org/ (accessed 9.29.20).

Ortner, R., 2013. Adaptive aggregation for reinforcement learning in average reward Markov decision processes. Ann Oper Res 208, 321–336. https://doi.org/10.1007/s10479-012-1064-y

Perez, F., Granger, B.E., 2007. IPython: A System for Interactive Scientific Computing. Comput. Sci. Eng. 9, 21–29. https://doi.org/10.1109/MCSE.2007.53

Phillips, D., 2015. Python 3 Object-oriented Programming - Second Edition, Community Experience Distilled. Packt Publishing, Birmingham.

Poole, D., Mackworth, A., Goebel, R., 1998. Computational Intelligence: A Logical Approach, 1 edition. ed. Oxford University Press, New York.

Project Jupyter [WWW Document], 2020. URL https://www.jupyter.org (accessed 2.22.20).

Qin, Y., Zhang, W., Shi, J., Liu, J., 2018. Improve PID controller through reinforcement learning, in: 2018 IEEE CSAA Guidance, Navigation and Control Conference (CGNCC). Presented at the 2018 IEEE CSAA Guidance, Navigation and Control Conference (GNCC), IEEE, Xiamen, China, pp. 1–6. https://doi.org/10.1109/GNCC42960.2018.9019095

Rich, E., Knight, K., 1990. Artificial Intelligence, Subsequent edition. ed. McGraw-Hill College, New York.

Rossant, C., 2018. IPython Interactive Computing and Visualization Cookbook : Over 100 Hands-on Recipes to Sharpen Your Skills in High-performance Numerical Computing and Data Science in the Jupyter Notebook, 2nd Edition, Community Experience Distilled. Packt Publishing, Birmingham.

Russell, S.J., Norvig, P., Davis, E., 2010. Artificial intelligence: a modern approach, 3rd ed. ed, Prentice Hall series in artificial intelligence. Prentice Hall, Upper Saddle River.

Rykowski, J., Wilusz, D., 2014. Comparison of architectures for service management in IoT and sensor networks by means of OSGi and REST services, in: 2014 Federated Conference on Computer Science and Information Systems. Presented at the 2014 Federated Conference on Computer Science and Information Systems, pp. 1207–1214. https://doi.org/10.15439/2014F324

Schulz, J., Popp, R.S.-H., Scharmer, V.M., Zaeh, M.F., 2018. An IoT Based Approach for Energy Flexible Control of Production Systems. Procedia CIRP 69, 650.

Shang, X.Y., Ji, T.Y., Li, M.S., Wu, P.Z., Wu, Q.H., 2013. Parameter optimization of PID controllers by reinforcement learning, in: 2013 5th Computer Science and Electronic Engineering Conference (CEEC). Presented at the 2013 5th Computer Science and Electronic Engineering Conference (CEEC), IEEE, Colchester, United Kingdom, pp. 77–81. https://doi.org/10.1109/CEEC.2013.6659449

Sharp, M., Ak, R., Hedberg, T., 2018. A survey of the advancing use and development of machine learning in smart manufacturing. Journal of Manufacturing Systems 48, 170–179. https://doi.org/10.1016/j.jmsy.2018.02.004

Shipman, W.J., Coetzee, L.C., 2019. Reinforcement Learning and Deep Neural Networks for PI Controller Tuning. IFAC-PapersOnLine 52, 111–116. https://doi.org/10.1016/j.ifacol.2019.09.173

Siebra, C. de A., Neto, G.P.B., 2014. Evolving the Behavior of Autonomous Agents in Strategic Combat Scenarios via SARSA Reinforcement Learning, in: 2014 Brazilian Symposium on Computer Games and Digital Entertainment. Presented at the 2014 Brazilian Symposium on Computer Games and Digital Entertainment, pp. 115–122. https://doi.org/10.1109/SBGAMES.2014.36

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., Riedmiller, M., 2015. Deterministic Policy Gradient Algorithms 9.

SIMATIC IOT2000 | SIMATIC IOT gateways | Siemens Global [WWW Document], 2020. URL https://new.siemens.com/global/en/products/automation/pc-based/iot-gateways/iot2000.html (accessed 7.5.20).

simple-pid: A simple, easy to use PID controller, 2019.

Sutton, R.S., Barto, A.G., 1998. Reinforcement Learning: An Introduction 352.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A., 2014. Going Deeper with Convolutions. arXiv:1409.4842 [cs].

Szita, I., Lörincz, A., 2006. Learning Tetris Using the Noisy Cross-Entropy Method. Neural Computation 18, 2936–2941. https://doi.org/10.1162/neco.2006.18.12.2936

Team, K., 2020. Keras documentation: Why choose Keras? [WWW Document]. URL https://keras.io/why_keras/ (accessed 6.25.20).

TF Cloud, 2020. . tensorflow.

van Hasselt, H., Guez, A., Silver, D., 2015. Deep Reinforcement Learning with Double Q-learning. arXiv:1509.06461 [cs].

Vinyals, O., Kaiser, L., Koo, T., Petrov, S., Sutskever, I., Hinton, G., 2015. Grammar as a Foreign Language. arXiv:1412.7449 [cs, stat].

Winston, P.H., 1992. Artificial Intelligence, Third edition. ed. WINSTON.

Zhong, R.Y., Xu, X., Klotz, E., Newman, S.T., 2017. Intelligent Manufacturing in the Context of Industry 4.0: A Review. Engineering 3, 616–630. https://doi.org/10.1016/J.ENG.2017.05.015

Zhou, Z., Chen, X., Li, E., Zeng, L., Luo, K., Zhang, J., 2019. Edge Intelligence: Paving the Last Mile of Artificial Intelligence with Edge Computing. arXiv:1905.10083 [cs].

Zou, Z., Jin, Y., Nevalainen, P., Huan, Y., Heikkonen, J., Westerlund, T., 2019. Edge and Fog Computing Enabled AI for IoT-An Overview 6.

Appendix B: Code Base

The code base developed for this project is huge, Therefore the same is uploaded into GitHub.

https://github.com/lavishthomas/PID_ENV

The code is divided into modules as described in the system design.

1. Process module
2. Environments
3. Agents
4. PLC-Agent
5. Results

Details of each module is given in the Chapter 4