Control-Theoretical Stress-Testing for Cross-Platform Compliance

A thesis

submitted in partial fulfilment

of the requirements for the Degree

of

Masters of Research in Computer Science

at

The University of Waikato

by

Harry McCarthy

THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

2020

Abstract

Modern systems are required more and more to be able to provide performance reliability qualities for not just one system but cross platform as well. Stress testing applications allow for performance testing of a system, however, when applied across different platforms the performance of the application is not a one to one comparison. This thesis introduces the idea of applying control theory to provide consistent resource percentage utilisation when stress testing cross platform. Proportional Integral Derivative (PID) control, Linear Quadratic Regulator (LQR) control and a PID Assisted LQR control were utilised as three control system in this thesis. While the PID controller provided the best results across multiple platforms, its inability to control all the variables in the system along with multiple different state objectives (CPU and memory) leads to the need for a system model approach. LQR controllers provide this model based approach but at the cost of being unable to adapt across different platforms and to unpredictability in the system. Finally, a combined approach PID Assisted LQR control. PID Assisted LQR control gave the best of both PID and LQR control approach and had improvements in the case of where the system was most unpredictable but lacked any significant improvements in areas where the model gave an accurate representation of the system.

Acknowledgements

The author thanks Panos Patros for guidance in producing this thesis. The University of Waikato and Oceania Researchers in Cloud and Adaptive-systems *te Reo, Ohu Rangahau Kapua Aunoa* lab for providing the facilities and required systems in order to carry out research into this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Software engineering as a discipline has designed and introduced many systems and applications many of which have only grown in complexity. Along with our continuing dependability on these systems and applications have created an impossible challenge for modern software engineers. These systems and applications are only growing in complexity hence software engineers are required to come up with new ways of managing and optimising these systems, one of the growing ideas for managing these systems came in an IBM white paper [10] that laid out the ideas of technology managing technology. IBM's white paper, initiated the idea of self adaptive systems that could manage the running of the everyday challenges that came with mundane but specialist maintenance of these systems. With the introduction of cloud platforms and continual growth of managing virtualized servers [16], these environments have led to the furthering development and research of self adaptation in all kinds of systems both in localised applications [11] and remote cloud management [15]. Part of providing cloud applications services is being able to supply reliable service level objectives, and knowing what performance the cloud service can provide [12] especially when running applications across multiple platforms. One way to provide this is through stress testing the cloud performance, allowing the cloud provider to give realistic service level objectives and understand how certain applications and

systems are going to perform in the real world. However, stress testing rarely represents the real world running of a system or application, as cloud providers often have to run multiple virtualized machines and services all running on the same server, and being able to simulate the real world running conditions can be important to providing reliability.

1.1 Cross Platform Reliability

Part of providing reliability is being able to take the same system or application and being able to provide the same level of resource consumption no matter what system the application is running on. However, stress testing can often only provide performance reliability on the platform that it was tested due to the differences in resources and software when moving from one platform to another. Restricting resource utilisation of an application or system can allow for simulation of realistic conditions when running applications in the real world, as understanding the performance of stress testing cross platform while only utilising 30% CPU can give an idea on how an application is going to scale or preform when under stress on a resource capped system. Stress testing applications running on multiple platforms can also utilise different percentages of resources, or have an increased variation on one platform that is not present on the other. This inconsistency in resource percentage utilisation makes comparing the performance of this stress testing application unrealistic as one system may have utilised more CPU than the other removing the ability to make a one to one comparison of the performance on either system. Looking at Figure 1.1 we can see that the CPU utilisation of two machines, Donner and Narwhal on the same benchmarking program is around 10% above Narwhal, although Donner is the better system it is difficult to make any assertions on the results while 10% more CPU is being utilised over Narwhal.

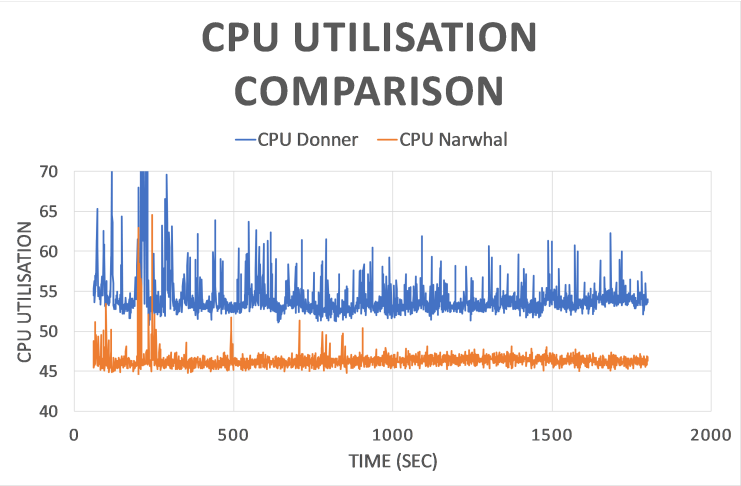

Figure 1.1: Cross Platform Resource Utilisation

1.2 Control Theoretical Systems and Resource Control

In software engineering control theory is still a relatively new discipline but can generally be split into two broad categories, open and closed loop control (Figures 2.1 and 2.2). This thesis focuses on closed loop control system as open loop control tends to rely on passive systems with predictable input and output. This thesis focuses on the utilisation of PID and LQR control feedback loops, PID controllers make use of the different between the current point in the system and the desired setpoint and uses this error to scale up and down the parameters in the system in an attempt to meet these setpoints. LQR controllers on the other hand take a more grounded mathematical approach to control and rely on the model of the system along side the error as a way to make decisions on how to move the parameters up and down to meet the desired setpoint. However, this reliability on the model means that if the model has any inaccuracies present, the control of the system will be off as well, unlike PID controllers which rely on the actual measurements taken from the system in order to control the parameters. These two control approaches have their benefits such as PID control is more likely to be able to adapt to

changes in the system model whereas LQR will strictly follow the model and is unable to make these changes as the system runs, however, LQR controls reliance on a model means that instead of being reactive like PID control LQR control can make changes to the system before these variations occur.

1.3 Design and Layout

This thesis attempts to utilise control theory through a feedback controller to try to maintain a constant resource utilisation percentage when running on multiple platforms. For benchmarking this thesis makes use of a program called Graphical user interface Garbage Collection (GUIGC) [21] which was designed to test the effect of the Garbage Collection (GC) on Graphical user interfaces (GUIs) by testing the time take to create display and destroy a pseudorandomly generated interface (Sections 3.1, 4.2.3, and 4.2.3.1). However, the ideas that are displayed in this thesis can be applied to any benchmarking or stress testing application. This thesis takes three different approaches to controlling the benchmarking application shown in Figure 1.2, the idea is to test each control approach and understand how well each one achieves three of the main goals, setpoint accuracy, low variation, and controllability of the system. Software engineering systems can be unpredictable and due to the lack of any solid equations, such as the ones found in other fields of engineering, building and design an accurate model for LQR control required leveraging Machine Learning (ML). ML and more specifically linear regression was utilised in this thesis as a way of building a set of equations that can predict the benchmarking system. However, even with the introduction of ML in the model design process there is still some inaccuracy in the model, hence the introduction of a PID Assisted LQR controller. By trying to utilise the strengths of a system model with the adaptability to uncertainty in PID control the PID assisted LQR controller attempted to make the use of both these properties of control found in the

respective feedback loops.

Figure 1.2: Three Control Approaches

Chapter 2

Background

Increasing complexity of administering and optimising systems has increased to a level at which software engineers are challenged to keep up with system complexity and performance [10, 11]. Along with these increasing complexities is the demand of dependable guarantees and performance of applications within and across multi platform environments [18, 9, 17], involving some variety of optimisation in order for this to be provided. One of the many areas of software engineering that exhibit promise at helping achieve these goals and optimisation, is the idea of self adaptive systems or technology managing technology [10]. IBM [10] lays out the goals of a self adaptive system that can self configure, heal, and, optimise laying out a blueprint for self adaptive systems to manage many of the complexities of optimising a modern day system. Weyns [34] lays out six waves of research in self adaptive systems with the current focus on waves V (guarantees under uncertainties) and VI (control based adaptation) which could be leveraged to provide cross platform adaptation while also providing dependable guarantees of an application across different systems.

2.1 Control Theory

The complexity of software is growing, along with our dependence on software to run in a multitude of different dynamic and complex

environments [3, 38]. Dependency of software to run in these complex environments has created a need for self-adaptive software, either to meet service level objectives or to be optimised to run under certain conditions. Control theory employs feedback loops shown in Figure 2.2 as a way of adapting to a given set point within a system in a mathematically predictable way. By employing control theory this allows users to make guarantees about the software's performance. In this thesis, this project investigates two different forms of control theory, PID controllers and LQR controllers. Datacentres face this problem in many aspects of their operation, but importantly software must be able to adapt to changes in workload [38] in order to utilise available resources to meet service level objectives.

Figure 2.1: Open Loop System

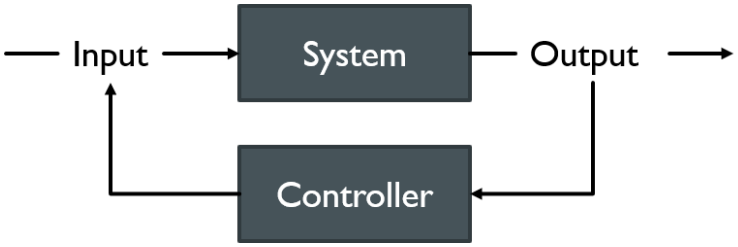

Figure 2.2: Closed Loop System

2.1.1 Proportional Control

Proportional control is defined by the equation shown below and takes the error between the set point and the current point, with the idea that the magnitude of the change depends on the magnitude of the error [13].

$$u_P = K_P \cdot e(t) \tag{2.1}$$

However, proportional control alone is not enough to provide a stable control, when using proportional control, the output of y will always be lower than the desired set point due to a phenomenon called the proportional droop [13]. Proportional droop is caused by the controller requiring a non-zero input in order to produce a non-zero output, however, if a zero input is produced then the output signal will vanish. The proportional droop can be reduced by increasing the value of K_P however if K_P is increased, it could cause some instability in the system.

2.1.2 Integral Control

Integral control defined by the equation shown below is one of the ways to reduce the proportional droop and fix the steady-state error [13].

$$u_I(t) = K_I \int_0^t e(t)\, dt \tag{2.2}$$

The proportional droop is corrected by adding the accumulated error to the proportional error to a PI controller. Although integral control will fix the proportional droop in a PI controller, it does, however, introduce an issue where unwanted oscillations can occur due to previous high errors at the beginning. By having the integral controller accumulate errors over time, it is likely that high errors, in the beginning, are going to occur, having a knock-on effect later in the controller. The knock-on effect caused by high errors at the beginning is generally fixed in one of two ways, either wipe the integral and start accumulating again, create a fixed window where values are added and removed as time continues, removing high errors earlier while keeping stable integral control.

2.1.3 Derivative Control

Derivative control defined by the equation shown below attempts to anticipate how the error is going to change over time.

$$u_D(t) = K_D \frac{de(t)}{dt}$$

For example, if the rate of change is positive this gives an indication that the error is growing overtime. Because we can see that the error is growing overtime through a positive value in the derivative control, this allows the controller to apply corrective action immediately even if the tracking error is small. Although the derivative controller has the potential to correct errors before they occur it also has the potential to enhance the effect of noise provided to the system input [13]. Because derivative controllers can amplify the noise of a system, it often becomes necessary to smooth the noise of the input however this leads to more complexity in the controlled system. Derivative controllers also have sudden spikes that occur when the set point of the controller changes suddenly: this is known as the derivative kickback. Because of amplifying noise and derivative kickbacks, it is common for the derivative controller to not be used in production systems.

2.1.4 Proportional Integral Derivative Controllers

The combination of all three of these components, proportional, integral and derivative control is known as a 3-term or PID controller and is represented by the equation shown below and shown in Figure 2.3.

$$u_{PID}(t) = K_e e(t) + K_i \int_0^t e(t)\, d\tau + K_d \frac{de(t)}{dt} \tag{2.3}$$

PID controllers provide an easy and simplistic way to build and implement feedback control, where little knowledge is needed of how the

system behaves. Although little knowledge of the system is required, it can be helpful when implementing PID controllers to have some understanding of how the different parameters affect the overall system. Proportional control is central to a feedback system, as it allows keeping track of the error between the set point and the current point of the system. Integral control is required to eliminate the proportional droop mentioned earlier. However, what is less required is the derivative control, due to its effect of amplifying noise and derivative kickback, because of this, it is more common to utilise PI control over a PID controller when used in production systems. PID controllers utilise three gain values K_e, K_i, and , K_d for the proportional, integral and derivative sections of the controller respectively, these values can be used to amplify or reduce different parts of the controller and in most production systems K_d is set to 0 effectively turning off the derivative section of the controller. Although beyond the scope of this thesis one of the benefits of the gain values is that they can be adaptively controlled to amplify and scale down different sections during various points in the control process. It is possible that at the beginning the integral and derivative sections are better at the control process but find these values hinder the controller later on. Therefore, begin with K_e, K_i, and, K_d values of 1, 5, and, 3 respectively then slowly adapt them over time of the controller to a point where the error starts having more effect in the later stages of control.

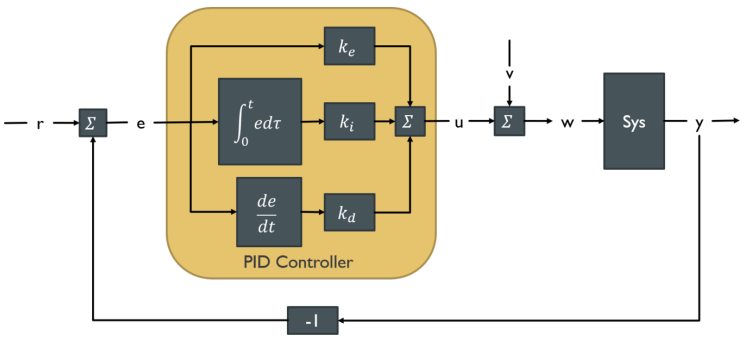

Figure 2.3: PID Controller

2.1.5 Optimum Control

When designing control systems the goal is to find an optimum performance while also satisfying specific requirements [32]. PID controllers attempt to achieve the control section of the system by utilising the error between the setpoint and the measured values in the desired system. However, throughout the control process, PID controllers provide no guaranty that control of the system is going to be achieved in the most optimum way. Optimum control attempts to take a model of the desired system and design the best way to achieve the goals in the most optimum way. This thesis focuses on an optimum controller design called a Linear Quadratic Regulator.

2.1.6 LQR

LQR controllers utilise a model of the system through the use of matrix equations that describe how the system behaves and reacts to different parameter changes. LQR controllers also utilise a K matrix based off of a cost function J is shown below that describes an optimum control solution for the LQR controller. In the state space equation $\dot{x}$ there are two vectors x and u which are the state and control vectors respectively.

$$\dot{x} = Ax + Bu \tag{2.4}$$

The state vector takes parameters that are measured from the system and represent the states that are being monitored within the system. The control vector represents different parameters of the system that allow the controller to assert control over the state vector x. Two other matrices A and B are the state and control matrix respectively, A and B matrices together describe the overall model of the system with A describing how the state vectors change over time, while B describes how the control vectors will influence the state vector over time. This thesis looked at 2 different types of LQR controllers, use

state error (Figure 2.5) and scale reference (Figure 2.4) feedback control, with use state error being the main feedback controller utilised in the final results. Use state error controllers (Figure 2.5), like proportional control (Section 2.1.1) utilises the difference between the setpoints and the measured values. With r in Figure 2.5 representing a vector of setpoints that are a one to one match of the state vector, take the state vector x and subtract the values producing a new vector of the error between the setpoints and the current state, giving a modified system equation both are shown below with the original system equations first.

$$\dot{x} = Ax + B(-Kx) \tag{2.5}$$

$$\dot{x} = Ax + B(-K(r - x)) \tag{2.6}$$

Scaled Reference controllers (Figure 2.4), take a vector K_r and scales the setpoint r to match what is required in order to control the system. In this thesis, the use state error was found to be the best way to model LQR control of GUIGC. Whether using the scale reference or use state error controllers, LQR optimises the control of the system through the use of a cost function shown below that is defined by manipulating the values in the matrices Q and R.

$$J = \int_0^\infty \left(\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}^T \begin{bmatrix} Q_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & Q_n \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix}^T \begin{bmatrix} R_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & R_n \end{bmatrix} \begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix} \right) dt \tag{2.7}$$

The matrix R defines the cost values for the control vector u, where higher values indicate a greater cost for changing that parameter in the system. R and Q are both diagonal matrices that define the cost for the control vector u and state vector x respectively, and once calculated produce

a gain matrix K that will optimise the control of the system base off of the model and requirements set out by the A, B , Q , and R matrices.

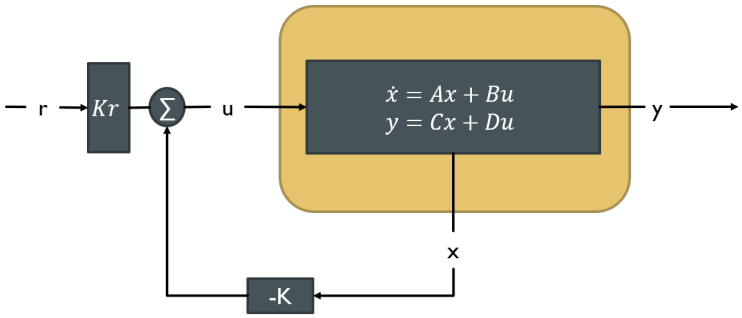

Figure 2.4: Scale Reference Feedback Controller

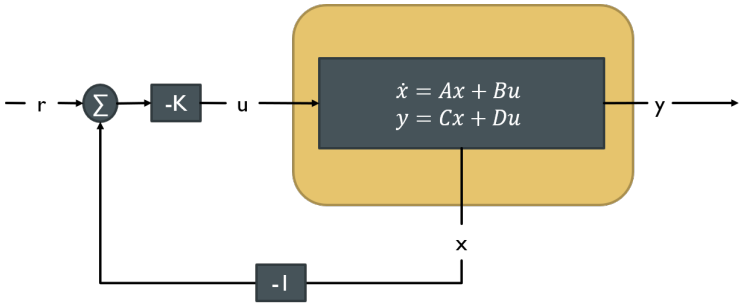

Figure 2.5: Use State Error Feedback Controller

2.1.7 Kalman Filter

A Kalman filter is a recursive filter, meaning that it uses the previous state obtained in order to predict the next state in the system. The process of evaluating a Kalman filter is done in a four-step computation: predict state and error covariance, compute Kalman gain, compute the estimate and compute the error covariance. Throughout this four-step computation, the Kalman filter takes one measurement vector and outputs one estimate vector, with the background noise from the system filtered out providing better accuracy from the uncertainty in the measurements. The prediction process takes the previously calculated estimate and attempts to predict what the next state in the system using the pre-calculated system model will be. The Kalman gain is a calculated matrix that is used as a weighting matrix in order to

compute the estimate. Once the estimate is calculated, the Kalman filter will then take all the previous steps and a measurement vector, using this to calculate the estimated next state of the system. Finally, the error covariance is used to indicate the difference between the estimated value from the Kalman filter and the true unknown values of the system. Kalman filters are used in a variety of areas in computer science and engineering [19, 15, 16]. Kalman filters are mostly used in order to provide more accurate results under uncertainty, and due to being able to execute quickly this makes them great for real-time applications.

2.2 Machine Learning

ML is being increasingly used across the social and computer sciences, as a way of addressing the complex and global challenges of the developing world [7]. Along with the developing world ML is also being heavily utilised in the world of business, advertising, object recognition and a multitude of other complex real-world problems [5]. ML utilises a range of different statistical techniques in order to find relationships and patterns in data, building mathematical models that then describe the relationships between this data as a whole [5]. Machine learning can be separated into two broad categories of supervised learning and unsupervised learning [4]. Supervised learning takes a labelled data set and produce a model of the system that can accurately predict the data set and any other future data provided. Unsupervised learning is where the data provided is unlabelled and it is up to the algorithm used to classify and model the system from the data provided. In this thesis an accurate model of the system is required in order to implement a working LQR controller, however, due to the limitations of a linear model being required along with the need for little CPU and memory utilisation, this thesis only focused on linear regression as the chosen machine learning technique.

2.2.1 Linear Regression

Linear regression is a popular machine learning algorithm, that attempts to create a model of the system using a linear combination of features from the input [4]. Linear regression produces a simple mathematical equation in the format of equation shown below, where the goal is to find the best value for m and c that accurately predicts future data.

$$y = mx + c$$

One of the justifications for using linear regression is that it uses a simple machine learning algorithm, which reduces the complexity of the model compared to other machine learning techniques. Linear regression will rarely produce over fitting results, where the model produced predicts the training data so accurately but is unable to correctly predict new data. However, because a linear equation is produced, it tends to cause errors when predicting data that is not following a linear trend.

2.3 Garbage Collection

Memory management for programs has been a common issue, especially as programs are becoming more complex. In lower-level languages, such as C or C++, the programmer must manually manage the memory space. Manual management of memory can lead to common problems, such as dangling pointers and memory leaks, within an application [36]. Such issues have given rise to modern programming languages, such as Java, Python, and Ruby, that implement automatic memory management. Automatic memory management takes away the responsibility of the programmer to manage objects that are created during the execution of an application, using a special program called GC. Garbage Collection is used to keep track and

manage memory during the program languages runtime, by allocating and deallocating objects to and from the heap. However, the added simplicity of GC does come at a cost, for portions of the programming language runtime the garbage collector must implement stop-the-world events, which cause delays and impact the performance of the running application [21, 22, 26, 25, 23, 6, 14, 2, 1]. Over the years, different GC methods and algorithms have been utilised in order to minimize the performance impact of the garbage collector. In the JVM, there are many different GC algorithms each with their own benefits and ideas on how to manage memory [21, 29], one such idea is the generational hypothesis, which states the objects tend to die quickly after their allocation, so the heap spaces separated appropriately into a nursery space and a tenure space. Generational GC algorithms work great if the application creates and deallocates objects over a short period of time, however, if the application does not follow the generational hypothesis, objects get moved to the tenured space and all benefit from these algorithms is lost, hence why optimisation of the JVM And pre-selecting the GC algorithm before execution can lead to performance increases [14, 1].

Chapter 3

Related Work

Control theory has been utilised in many aspects of software engineering from Imes et al. **Co**ntrol **P**erformance with **P**ow**er** (CoPPer) [11] to the theories and ideas discussed by Filieri et al. [8]. The idea of managing system through control theory is not new and has been laid out in IBM's white paper on self adaptive system [10], where IBM introduces the idea of having technology manage technology as a way of getting around the complexity of managing modern systems. Unfortunately, to the author's knowledge utilising control theory for stress testing is a limited field of research, the majority of control theory approaches are around combatting non linearity such is the case with Copper [11], communication networks [33], utilisation of control theory in production systems [37] and autoscaling [27, 24], or discussions of theoretical applications of control theory in software engineering [8, 3]. The lack of research in this area leaves many areas of this research open to interpretation, however, papers such as [30, 37] provide practical uses of LQR controllers in software engineering systems but fail to introduce any form of software modelling of systems where equations are yet to be found.

3.1 Benchmarking

Graphical User Interfaces Effect on Garbage Collection (GUIGC) [21] is a benchmarking program that was used in the paper GC Delays on Java GUIs,

in order to compare the performance of the GC on GUIs in JavaFX, and specifically the GC delays when used in low resource environments. The paper GC Delays on Java GUI focused on analysing the performance or four different GC policies of the IBM J9 Java runtime, Gencon, Balanced, Optthrouput, and Optavgpause. GUIGC achieves this by creating a tree structure of different GUI panes using the two parameters depth and breadth, then fills each of these panes with N number of buttons which when clicked computes a simple MD5 hash from a pseudo-random string. GUIGC then repeats the process of creating and closing this form x number of times while gathering different metrics including the length of time to show the form, click the buttons, and then close the form along with different GC metrics gathered during the execution. By repeatedly creating and destroying objects GUIGC can ascertain the delay that garbage collection has on the creation and execution of GUIs in JavaFX, which is important as long delays can often lead to poor response time in GUIs. In order to better make use of GUIGC some extensions were introduced giving more control parameters to utilise in the controller, these extensions are described in Section 4.2.3.1.

3.2 Control Theoretical Systems

The paper laid out by Weyns [34] describes the struggles that are placed on the management and optimisation of complex systems and how the introduction of the IBM manifesto [10] introduced this complexity crises. Weyns [34] also present six waves of self adaptive systems automating tasks, architecture-based adaptation, runtime models, goal-driven adaptation, guarantees under uncertainties, and, control-based adaptation. Control theory has been utilised in many aspects of software engineering and in the paper Software Engineering Meets Control Theory [8], Filieri et al. propose that there are two main problems with developing control theoretical systems

in software engineering, the first, is a lack of any software engineering methodologies for pursuing controllability, while secondly, is the difficulty in developing mathematical models of systems. Proportional controllers (Section 2.1.1) are the simplest form of control which Imes et al. [11] utilises for power capping in the paper CoPPer, to combat the relationship between power and performance being non-linear. Performance requirements with power capping on its own is harder to adapt too so to overcome this relationship Imes et al. proposes an adaptive controller as a way to approximate the non-linearities present in the model. Zhang et la. [37] on the other had is utilising an LQR controller in order to provide guarantees of an uncertain inverted pendulum system, using control theory to provide something different to Imes et al. [11], where the focus for Imes et al. is control of a non linear system. Filieri et al. also goes on to describe the different approaches between software and control engineers, where the former is primarily focused on having software react to changing environmental and contextual conditions through adaptation of the software's behaviour (self-adaptive systems), while the latter, is more focused on providing formal guarantees on the controllability of a system, and self-adaptation is one way of providing this guarantee. Filieri et al. propose that the merging of these two approaches could enhance the software engineering process, providing more mathematically grounded self-adaptive processes.

3.3 Control Theoretical Methodology

Filieri et al.[8] layouts methodologies for implementing control theory into software engineering. Synthetic design is the first methodology, where little information is required, however, because it relies on pre-designed control blocks a specialist in the desired system is required in order to decide what blocks are needed to control the system. Parameter Optimisation is the

second methodology and is a variation of the first, however, it differs in the way that the parameters are chosen based on optimisation strategies. Finally, the last methodology is an analytical design where the system is based on an equation that describes a model of the desired system. However, both of these methodologies rely on heavy knowledge of the system and already existing control blocks that have been designed for the required systems that control is going to be applied. Filieri et al. [8] also note that the system's model should take into account the time scales of each parameter e.g. spinning up a virtual machine may take some time for the desired effect to take place, hence the controller must make allowances for this time in the model of the system. This aspect of taking into account time scales introduces another complexity when running control theoretical models across different platforms. Software Engineering Meets Control Theory also expands to other aspects of control theory in software engineering, such as an external control system. External control systems, allow for a system or systems to take advantage of an external control system, provided that all the required endpoints are in place and the systems can provide the required measurements. Filieri et al. [8] do note, however, that there is a potential performance penalty with an external controller due to either interference or limited resources such as is the case with embedded devices.

3.4 Control Theoretical Model Design

Design and building of a control model takes two approaches, either utilise a control theoretical approach that does not require a model such is the case with Imes et al. [11] or utilise control blocks that are discussed by Filieri et al. [8] and used in the papers Raj et al. and Zhang et al. [30, 37]. Zhang et al. [37] uses a mathematical model in the form of matrix equations (Control blocks) to represent an inverted pendulum, however, these models already existed and required someone with specialist knowledge on these equations to

place them in to control blocks (stated by Filieri et al. [8]). Raj et al. [30] dose the same as Zhang et al. [37] except the mathematical equations utilised in the LQR controller are based on the law of magnetism in order to model magnetic levitation. Both of Zhang et al. and Raj et al. [37, 30] both make use of the code blocks mentioned by Filieri et al. [8] and require specialists in the respective areas to build and place these equations in to a model that can be used by the LQR controller. In other areas of software engineering and more broadly in the real world where these mathematical equations either change or are too complex to be built and designed by hand the utilisation of ML is applied. Chaoji et al. [5] describe ML as something that researchers and practitioners must have a conceptual and practical understanding and argued that ML has become a mature technology that is utilised in most aspects of the real world. Outside of software engineering ML is being utilised to build complex models of population dynamics, weather patterns, distribute health care supply and solve complex problems [5, 7]. Prasad et al. [28] attempted to utilise the model of a pendulum, however, instead of a singular controller a combination of PID and LQR controllers were used. Prasad et al. [28] takes three approaches to the inverted pendulum, one is using two PID controllers one for the position of the cart and the other for angle, the second is two PID combined with an LQR controller and lastly is a single PID controller with a single LQR. However, there is very little in the way of concrete conclusions about the performance of this approach in the paper, leaving it undecided about whether dual controllers are a valid approach to the inverted pendulum.

Chapter 4

Research Method and Objectives

Java utilises GC when it comes to memory management making it a resource in the system that is much harder to control through conventional means, such as PID control. Along with the JVM and interference from other programs on the system that utilises the JVM, this makes controlling the CPU and memory on a system difficult for a single process. The difficulty in controlling system resources becomes much harder when these controlled states have control parameters that influence both states, meaning that there is the potential to require more model-based approaches to control in order to be able to adequately meet both of the desired set-points for the states.

4.0.1 Motivation

When looking at Figures 4.1, 4.3, 4.2 and 4.4 we can see the cross-platform CPU and memory utilisation for GUIGC when no control is applied to the system, where both Narwhal and VirtualBox used the same parameters shown in Table 4.1. The CPU utilised for both Narwhal and VirtualBox shows to be around 100% to 120% in both Figures 4.1 and 4.2 however, the variation shown in Figure 4.2 is far greater than that shown in Figure 4.1. The variation in the CPU utilised across both systems (Narwhal and

VirtualBox) indicates to us that there are resource utilisation differences when analysing the same benchmarking program (GUIGC) across multiple systems. Similarities in the point where both systems hover around during execution give an indication that both systems are capable of meeting the same CPU set point, however, what is different is that Narwhal is better at maintaining a consistent CPU utilisation while VirtualBox shows heavy variation. The same effect shown in the CPU utilisation is also present in the memory utilisation as well, where the memory utilised for Narwhal remains consistent starting at just above 75% and ending at just below 85% (Figure 4.3) whereas the memory utilisation for VirtualBox has a greater variation and starting and ending utilisation. Looking at Figure 4.4 we can see the greater variation and starting and ending utilisation points, where VirtualBox starts at just above 45% and ends at just below 70%, all while the system oscillates more than Narwhal's memory utilisation. Due to the differences in starting and ending utilisation for both Narwhal and VirtualBox, along with the challenges that are proposed with controlling the memory utilisation of GC controlled languages, there is the possibility that controlling memory utilisation to the same point in both systems can not be achieved. However, even though it may not be possible to control the memory utilisation of both systems this thesis will attempt through control theory to produce a consistent oscillation of memory utilisation across both systems.

Parameter	Value
Hash	1
Sleep	1
Buttons	1
Depth	8
Breadth	2

Table 4.1: Base Test Parameter Settings

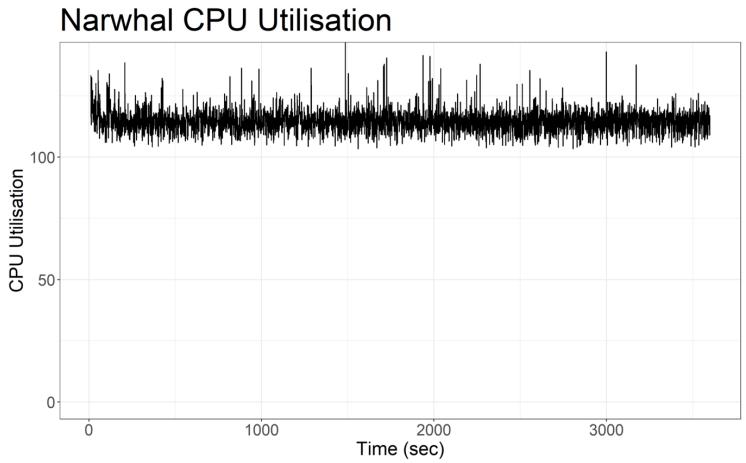

Figure 4.1: Narwhal - Zero System Control (CPU)

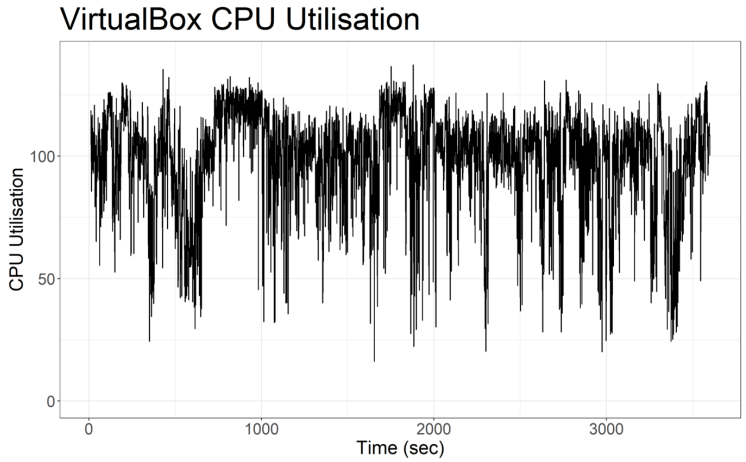

Figure 4.2: VirtualBox - Zero System Control (CPU)

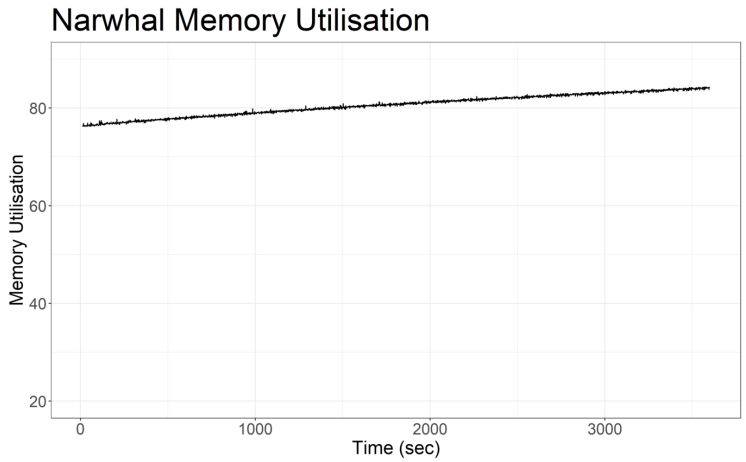

Figure 4.3: Narwhal - Zero System Control (Memory)

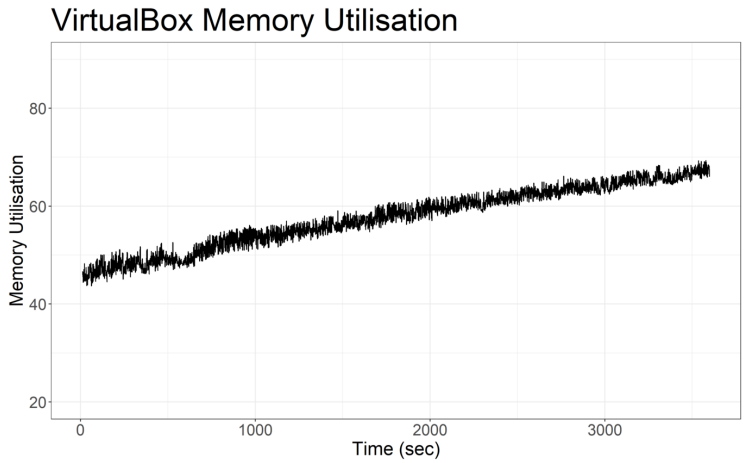

Figure 4.4: VirtualBox - Zero System Control (Memory)

4.1 Requirements

The difficulty in a multi-state system control is what the thesis is analysing and attempting to address existing issues with such controllers, while still attempting to provide mathematical guarantees about the performance of the system. Proportional Integral Derivative controllers are currently well-utilised ways of controlling a system in software engineering, due to the simplicity of implementation and formal guarantees provided by this approach. However, when dealing with the software engineering systems the lack of formal models utilised in PID controllers can hold back its usability in a system. This thesis analyses the ability of a PID controller to control resource utilisation for a single and multi-state system and attempts to utilise the forefront of software engineering based control theory. This Thesis hypothesises that by utilising Wave VI (Control Based Adaptation) as mentioned in the paper by Weyns [34] better control can be applied to a system. LQR controllers utilise models of systems in order to control and optimise a system, usually, these models are pre-existing and have been well tested. However, because the benchmarking system in this thesis is more abstract and harder to determine equations this thesis leveraged machine learning in order to provide the system guarantees required in Wave V for

LQR control based adaptation. By utilising Wave VI along with an LQR feedback controller and machine learning this thesis is hoping to provide solutions to the issues that are apparent in the PID control approach of software engineering when attempting to control the multi-state systems. For this thesis the experiments are carried out using GUIGC, however, the results can be generalised to other benchmarking software.

4.2 Proposed Approach and Design

Implementation of optimum control in software engineering is a developing field, where PID controllers have been utilised to control many systems, however, LQR controllers are less utilised due to the requirement of formal models. This thesis proposes that not only can LQR controllers be utilised in more software engineering systems but that LQR controllers can also provide more control in systems with greater uncertainty than the conventional PID controllers. This thesis also looks at the cross-platform state control of a system as well by providing the same utilisation of resource percentage two different systems across systems.

4.2.1 Systems Used for Experiments

Narwhal was the first machine that testing was conducted on and is the machine that the original LQR model was built for. Narwhal is running Ubuntu with an Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz with hyperthreading enabled, as well as 12288k L3, 256k L2 and 2×32k L1 caches, 15 GiB of memory. **VirtualBox** was run on top of Narwhal using Oracle's VirtualBox software so the same CPU was used for both VirtualBox and Narwhal. VirtualBox also used the Ubuntu operating system, however, VirtualBox was limited to 4 cores on the CPU and 4 GiB of memory.

4.2.2 Java Control Library

There are many ways that control can be introduced into a system but in order to implement PID and LQR control a simple library would be required in order to achieve the desired result of this thesis. One of the main requirements of a controller is that the controller works in the background of the original program (GUIGC) allowing the program to continue running while control is being applied. Second requirement was that it needed to be simple to switch between the two controllers so that as much or the same code was running as possible including the way the parameters were implemented and utilising the same control timer. By utilising as much of the same code between the two controllers meant that both controllers would run much the same just changing the way that the control was applied. Finally, although the controller does not necessarily need to be written in the same programming language as GUIGC, it was found that gaining access to the required states (CPU and memory) and making changes to the parameters was challenging in other languages. Due to the difficulty in and reading the states and changing parameters it was decided that keeping the controller in the same languages (Java) as GUIGC was the best way to move forward. For this thesis a separate library was created and ran in a separate thread in the same Java program as GUIGC, however, this thesis proposes future research could be implementing two individually running programs in order to reduce some interference from the controller during execution.

4.2.2.1 Reading the States (CPU and Memory)

When looking for a way to analyse the CPU and memory that were utilised throughout the execution of GUIGC, the first and most obvious way in Java is to utilise MXBeans to get these states. However, due to the controller being a part of the same program as GUIGC there were issues that arose as the controller would produce noise within the system give a reading that included

both the controller and GUIGC. This thesis attempted to implement a way of calculating the CPU utilisation by taking the running time that the GUIGC program was on the thread and dividing it by the total CPU run time, however, this method lead to inaccuracies when moving the program across platforms. There was also no way to calculate the memory utilised by just GUIGC as memory is not separated into threads so for consistency JavaFX's version of CPU utilisation and memory utilisation was used in this project. Lastly, this method of measuring CPU utilisation scaled the CPU by percentage based on the number of cores hence if a 5-core machine the max CPU utilised would be 500%, because of this the number of cores on both systems tested had to be above the max CPU set point that was used to get a good comparison at this CPU utilisation.

4.2.3 GUIGC

This thesis takes a benchmarking program called GUIGC and utilises it as a system that will be controlled under both PID and LQR control. GUIGC is a simple Java program that utilised JavaFX as a way of testing the effects of GC on JavaFX GUI while also analysing other performance metrics such as cycles pre-second and time taken to create, display and close the GUI. GUIGC was chosen for several factors. One being the simplicity of reducing the CPU through the use of the sleep parameter, this gives a great starting point for testing the control ability of a system. However, when attempting to control higher CPU values GUIGC has an effect where graphical elements such as buttons and panes cause spikes in the CPU introducing unpredictability into the system. The spikes in CPU also prevent parameters such as buttons, depth and breadth from increasing beyond a set point 50, 5 and 10 respectively, as the spikes in the system can last far too long and introduce out of memory errors. GUIGC also happens to have parameters that introduce complexity into the system, the parameters buttons, breadth and depth all have an effect on both the CPU and the memory in the system hence making it harder to control

such a system and giving the two controllers (LQR and PID) another element to consider when making decisions on how best to change the parameters.

4.2.3.1 Expansion

In order to allow for better control of the benchmarking program GUIGC some expansions to the parameters were required [20], hence the expansions make to GUIGC for this thesis have been laid out below. Version of the GUIGC program does make some changes to the list of parameters that are set by the program. These include being able to set the number of hashing repetitions. Along with these programmable set additions, there are also changes to the user set parameters as `--preset=[A|B|C|D]` was replaced with `--seed=<Integer>`, which configures all the knobs of GUIGC pseudo-randomly based on the given seed. This was done to enable testing of numerous combinations of different parameters as well as recording various performance indicators measured per configuration. For this thesis a seed of 1 was utilised for all the tests so that the same combination of parameters were used for testing.

Parameter	Lowest	Highest
Depth	1	5
Breadth	1	10
Buttons	1	50
Sleep	1	10000
Hash	1	10000

Table 4.2: Ranges for Parameters

The parameters described in Table 4.2 were chosen to fit the constraints and environment that the program is running in. By having depth set to a max of 5 and breadth set to a maximum of 10 along with the number of buttons set to a maximum of 50, this gives the highest number of potential buttons generated while keeping out of memory errors to a minimum.

4.2.3.2 Buttons Performing Tasks

Having a button perform some sort of task when clicked is also a newly introduced feature of version 2 GUIGC and introduced a new parameter hash. When a button was clicked we needed a way of ensuring that the same task was going to be repeated each time the program is run with the same seed. To ensure that the same task was repeated, it was decided that hashing a random string multiple times was suitable to achieve this; the hashing repetitions are decided based on the parameter hashes. The length of the string and the string used in the hashing were chosen pseudo-randomly so that it was not the same string used every time. The string was randomly set as a way of preventing the Java runtime from optimizing the button press. If the same string was used, the Java runtime may optimize this to skip the hashing loop as it would always be the same result. Having a button perform a task was decided as a way of getting the program to run more closely to a real GUI application but also keeping it so that the task could be controllable and repeated.

4.2.4 PID Control

PID controllers have been utilised many times and experimented with in multiple different papers [11, 35, 31]. Hence, along with a simplistic design and no requirement of any formal model makes a PID feedback controller (Figure 2.3) a good starting point when analysing resource control across systems. However, due to the design of PID controllers, a lack of any formal model being utilised this thesis proposes that better controllers be utilised when moving towards controlling multiple different states. This thesis will start out attempting to control one state (CPU) through the use of a PID controller and then adapt the system to start controlling multiple states (CPU and Memory) giving a good baseline for what is currently the best for both single and multi-state control in the space of software engineering.

4.2.5 LQR Control

LQR controllers utilise models of the system to implement optimum control, and although a very simplistic form of optimum control exists in PID controllers, it has little formal mathematical guarantees and still suffers from a scaling issue due to no system model. Because of the simplicity of the actual feedback loop in LQR controllers (Figure 2.4) and the mathematically grounded optimisation techniques, this makes LQR control a good starting point for exploring the possibilities of utilising more formal control methods in software engineering. The only issue with LQR is that in software engineering there is a lack of mathematical equations that can represent systems, historically LQR controllers would utilise well tested and defined equations such as Newtons equations on motion or in electronics there is Maxwell's equation that can represent a system, however, in the software engineering space, these equations do not exist. Hence, this thesis proposes the utilisation of ML as a way of creating usable equations that represent the complex system of GUIGC. Future work from this would be the next step of using ML to continuously adapt and learn as the system is running, by taking the control parameters and the states produced run through ML and then adapt the B and A matrix accordingly.

4.2.6 Hypothesis

This thesis aims to introduce better resource controllability for cross-platform stress testing, where there are three main aims to maintain, cross-platform controllability, reduction in variation, and setpoint accuracy. The limitations present in this thesis lead us to some conclusions about how the system is going to perform during testing concerning the three aims.

4.2.6.1 PID Controller Accuracy and Controllability

Due to the increasing complexity of controlling the CPU at higher levels and the complex relationship between CPU and memory in the system, I believe that PID controllers will only be able to confidently control either memory or CPU but not both. It is hypothesised that PID controllers will fail to control the CPU to higher values as the spikes in the CPU will produce high error numbers in the PID controller shifting it away from the desired set-points.

4.2.6.2 Controlling More Than One Parameter

The LQR controller, will due to the model be able to better control both CPU and memory at the same time while also being able to control a better array of control parameters, whereas PID controllers I believe will start to fail when attempting to control a higher number of parameters. However, the effect that the LQR controller has on the system will be completely reliant on the quality of the model and how accurately the model can predict the system as this is the primary driver of the LQR controller.

4.2.6.3 Cross Platform Reliability and Controllability

The aim is to introduce cross-platform resource reliability through control theory, this thesis hypothesises that not all variation in the systems will be reduced, hence, this thesis hypothesises that throughout the experiments there will most likely be slightly more variation in VirtualBox than Narwhal for the same experiments. The reason for the hypothesised variation is due to the presence of increased variation in Figure 4.2 for VirtualBox while there is significantly less for the same base experiment in Narwhal (Figure 4.1).

4.2.6.4 The Issue of Memory Managed Languages

One of the issues with memory managed languages is that unlike the CPU which reacts to changes almost straight away, memory is only released once the GC decides to run. This thesis adds to the implementation of a GC call

in the controller upon every cycle (One second for this thesis) however, just because a GC called does not mean that the GC is going to be run right away. The discrepancy in calling the GC and running the GC leads to complexity's in controlling the memory utilised, leading this thesis to conclude that controlling something as complex as memory in a memory managed system is going to be limited by the system and produce results that will display varied control of memory at best.

4.3 Experimental Methodology

This thesis was carried out in three broad tests, the first being the testing of the system and development of the over all model, along with single state (CPU) control used to test the PID controller, and finally, the multi state control used for both the LQR and PID Assisted LQR control. All the tests were run for 8 repetitions to attempt to produce more accurate results for the tests, along with the 8 repetitions all the test were carried out on the same utilisation percentages, 30%, 60%, 90% and 150% or in the case of PID control 120%. LQR controllers were tested on higher CPU utilisation of 210% and 310%, for memory that values of 68%, 74%, 77%, 83%, and 86% these values were chosen arbitrarily to attempt to cover a broad range of testing scenarios. PID control was tested differently to LQR in a few ways one of which the memory utilisation was not controlled in the PID controller tests, this was a decision made in order to test the reliability of the PID controller on a single controllable state (CPU). PID controller tests also did not cover CPU utilisation values above 120%, this 120% CPU utilisation was an arbitrary mark that was chosen due to the CPU utilisation results from Figure 4.1 and 4.2 to see the effectiveness of the PID controller in this range. LQR controllers were utilised as a way to attempt to go beyond the capability of PID control so more setpoints were added along with two control states (CPU and memory) which introduced the setpoints mentioned above. LQR control also attempted to utilise more

parameters specifically Buttons, Breadth, and Depth, and due to the increased CPU spikes that are present in these parameters it was decided that testing higher CPU setpoints for LQR was a good way to access the usability of a model based control approach.

4.3.1 Calculating Stable Time

One of the statistics that is compared in this thesis is the time taken for a system to stabilize, however, judging how long it takes a system to stabilize can be a difficult thing to do as there are many factors and contribute to what can be considered a stable system. One theory to calculate this was to take the first time that the system crosses the set point and consider this the stable point, however, as mentioned in this thesis there is a difference between meeting a set point and having a controllable system. Second theory was to utilize the median as the point and the first value to cross would be the stable point, however, as shown in Figure 4.1 at the beginning the first point to pass the median is not where the system is stable. Lastly, the final idea was to introduce a range between the median and plus or minus the calculated standard deviation and when 20% of the data in a row was within this range the beginning value was considered the stable point, and all values preceding this were removed from the set when calculating the final data.

Chapter 5

PID Controller Evaluation

Proportional Integral Derivative control is utilised in many areas of software engineering such as power capping with CoPPer, and under particular conditions can implement control over a simple system. In this thesis a PID controller utilises the equation shown below to control the CPU of both systems, it was found that PID controllers can control the CPU of a system and hold it at the desired set-point.

$$u_{PID}(t) = K_e \cdot e(t) + K_i \int_0^t e(t)\, d\tau + K_d \frac{de(t)}{dt}$$

However, in order to do this, the idea of how the system would behave was required and due to the inability to utilise the depth, breadth and buttons parameters in the system correctly, it leads to the PID controller failing to control CPU values higher than 100%. The PID controller was also unable to control memory in the system again due to the inability to utilise the required parameters and Java's GC being unpredictable in when the GC was run, this also meant that controlling both states at the same time was also beyond the scope of the PID controller.

5.1 Methodology

When analysing the performance of a PID controller it was found that only CPU set points below the threshold of 100% were controllable to any significant extent, so for the PID controller experiments only 30%, 60%, 90%, and 120% CPU utilisation were used in the experiments. The reason for the 100% threshold as shown by Figure 5.1 is that 100% is the limit of the system where the CPU is unable to be raised any higher under the limits of the current system. To raise the CPU higher, the PID controller would be required to control three more parameters in the system buttons, breadth, and depth. However, due to the more complex nature of these parameters, a PID controller as it stands in this thesis would require a model of how these parameters behave to have any significant control. Each of the experiments below was carried out on their respective machines (Narwhal and VirtualBox) and ran 60 minutes for each of the tested percentages (30%, 60%, 90%, and 120%) with a repeat of 8 times. Figures 5.4 and 5.5 have been normalized against the set point used to control the system, meaning that when looking at the figures the closer to 1 the more accurately the set point was met when controlling the system. The figures also display the upper and lower quartile ranges giving a visual indication on the controllability of the system, e.g. when looking at the 30% utilisation in Figure 5.4 we can see that the mean is close to 1 showing that the system met the desired set point, we can also see that the upper and lower quartile ranges are close together and within a significant distance from 1 show good controllability in the system.

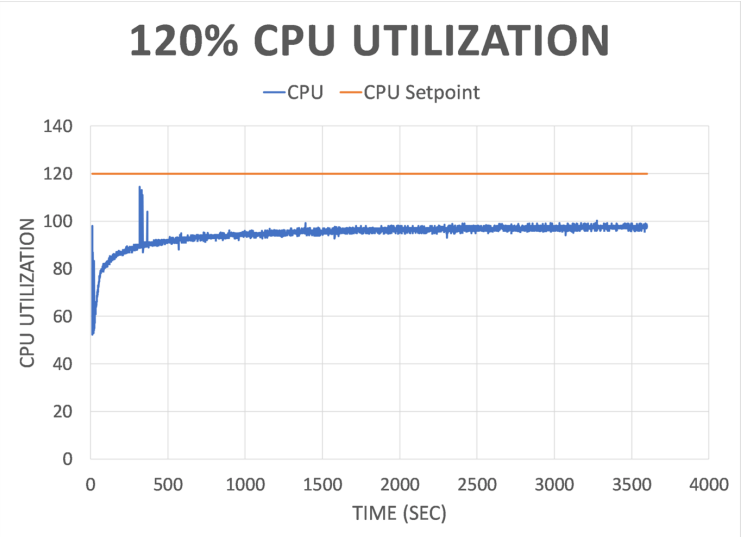

Figure 5.1: Narwhal 120% CPU Utilisation Time Graph

5.1.1 PID Controller Optimisation

One of the challenges of any controller is determining how to optimise it through the use of either gain value in a PID controller or a gain matrix in LQR controllers, whether its gain values or a matrix the result can alter the performance of the controller if not optimised correctly. There are two general ways that PID controller gain values can be optimised either through manual trial and error or self adaptive gain values that change with in the controller. For this thesis manual trial and error was used of the PID controller with some results of this process shown in Figure 5.2, looking at the Mean Squared Error (MSE) first (188.86) and last (430.82) values show what there can be vast differences in the controller and indicated that smaller values benefit GUIGC's control than the larger values. The benefit of smaller gain values for GUIGC is likely due to large over corrections produced in the controller, however, it is likely that with more testing different combinations for the Proportional, Integral, and, Derivative gains would provide more fine control and may even require adaptation of the gain over time to give better control at different stages of the PID controller. For this thesis the values in Table 5.1 were used as the final static gain values in

the PID controller tests, although the values chosen are the second best shown in Figure 5.2, when analysing the time graph in Figure 5.3 we can see that the differences between the Two best are small and for GUIGC a conservative system was used to prevent out of memory errors and hence the smaller of the two values were utilised for the final tests.

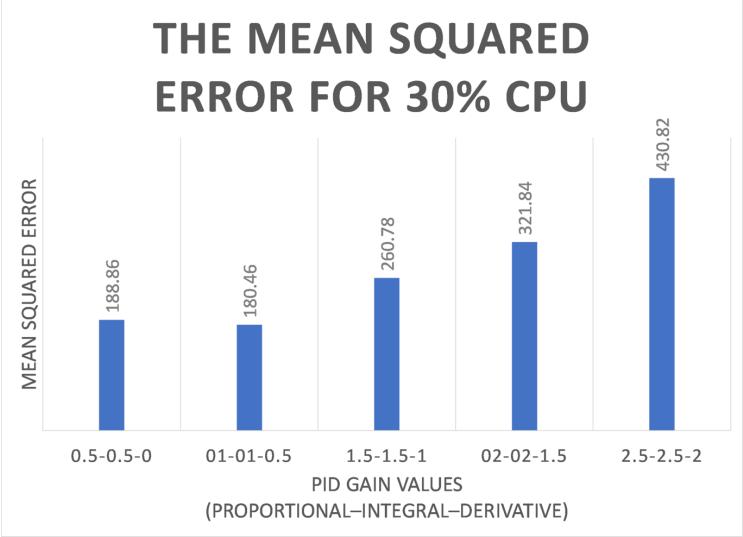

Figure 5.2: PID Gain Values Mean Squared Error for 30% CPU

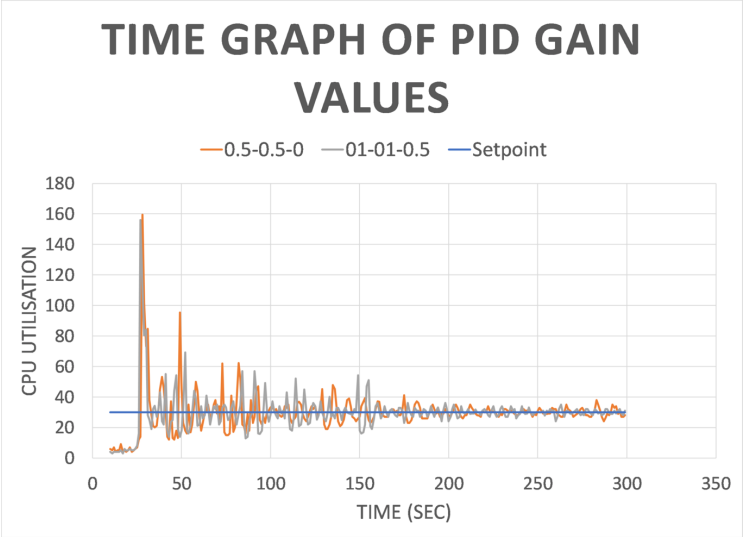

Figure 5.3: Time Graph for the Two Best Gain Values

	Chosen Gain Value
Proportional	0.5
Integral	0.5
Derivative	0

Table 5.1: Used PID Gain Values

5.2 Results

For the majority of the experiments, the CPU utilisation set points did manage to be met in both Narwhal and VirtualBox except for 120%, which met the requirements of controllability but was unable to meet the desired set point. 120% failed to meet the set point due to the limitations of the system described above (Section 5) where the system will throttle at 100%, although there is a slightly more variation in VirtualBox than Narwhal shown in Figures 5.4 and 5.5. This variation in 120% CPU utilisation is likely due to the already increased variation that existed in the base tests shown in Figure 4.2. There are also indications that with a PID controller Narwhal has better success at controlling CPU utilisation at and above 90%, this can be seen when comparing the 90% utilisation for Figures 5.4 and 5.5 where the last two show increased variation for VirtualBox. Where both Narwhal and VirtualBox succeeded was with the 30% utilisation which are shown to be similar with slight differences in the outliers although this is to be expected across multiple systems. The mean and the upper and lower quartiles for 30% CPU utilisation are almost the same for both Narwhal and VirtualBox, although as hypothesised there is more variation shown in Table 5.3 for VirtualBox than Narwhal due to the high number of outliers. However, differences between the median and having the majority of the data being in similar ranges for the upper and lower quartile shown in Figures 5.4 and 5.5 give a good indication that the PID controller is sufficiently controlling the CPU around the desired set point across Narwhal and VirtualBox. The middle two set points 60% and 90% also exhibits control around the desired set points with the medians for Narwhal at

60 and 92 respectively shown in Table 5.2. Figure 5.4 shows that the 60% CPU utilisation has both control and met the required set point with little variation, however, 60% for VirtualBox in Table 5.3 shows a that the set point was met but there is more variation in the CPU. Although there is more variation in the CPU for VirtualBox we can see from Figure 5.5 that this variation is in the outliers with the upper and lower quartiles being significantly closer to indicate control around the desired set point in the system. 90% CPU utilisation for both systems is where the accuracy of the median starts to decrease with the median for Narwhal being 92 with a variation of 26 and for VirtualBox the median is 77 with a variation of 417. However, for both Narwhal and VirtualBox their upper and lower quartiles are centred around the desired set point with Narwhal being slightly higher and VirtualBox being slightly lower, there is also a greater range in the upper and lower quartiles indicating that the CPU oscillates within this range. Finally, when looking at Tables 5.2 and 5.3 the variation in the CPU shown increases for both cases with 30% being the lowest and 90% the highest for both systems, with the exception for 120% however, this is where the system throttles so a decrease in variation is expected. The reason for this increased variation is likely due to the difference in effect for hashes and sleeps, where sleeps have much greater effect on the system meaning it is much easier to lower the CPU of the system than raise the CPU.

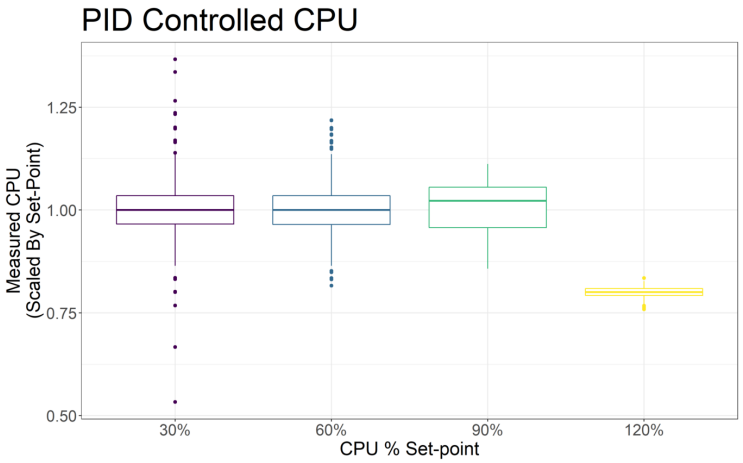

Figure 5.4: Narwhal PID Controlled CPU Utilisation

CPU	Time To Stabalize	Median	Variation
30	910	30	3.74
60	1879	60	15.58
90	368	92	26.38
120	571	96.08	2.38

Table 5.2: Narwhal PID Stabilisation Time

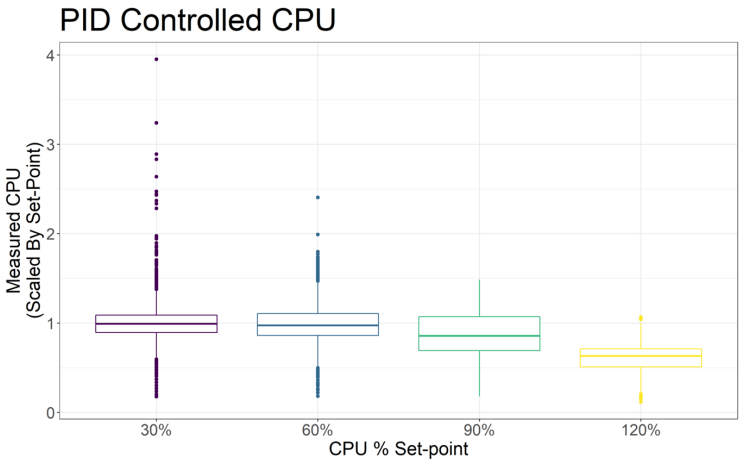

Figure 5.5: VirtualBox PID Controlled CPU Utilisation

CPU	Time To Stabalize	Median	Variation
30	20	29.74	41.98
60	20	58.51	179.56
90	20	77.01	417.63
120	20	75.76	315.8

Table 5.3: VirtualBox PID Stablization Time

5.3 Cross-Platform

One of the requirements for this thesis is reliable resource utilisation across different systems, and looking at Figures 5.4 and 5.5 we can see that the PID controller achieves this requirement for CPU values lower that 90% but fails for both Narwhal and VirtualBox when the CPU set point is set beyond 120%. For both 30% and 60% CPU set points the PID controller manages to maintain the desired set point and control the system, however, as hypothesised in Section **??** there is an increased variation in VirtualBox than Narwhal. Both systems also start to stop meeting the set point at around 90% CPU utilisation with the median for 90% and 120% for VirtualBox being 77 and 75 respectively (Table 5.3) and Narwhal 92 and 96 respectively (Table 5.2). It is also worth noting that VirtualBox fails to meed the set point in the median before Narwhal and there is a difference of 21 between Narwhal and VirtualBox for the median in 120% CPU utilisation. The divergence in variation between the two systems is considerable as VirtualBox leans towards having a difference of 40 to 100 when compared to Narwhal and this only increases as the CPU set point dose. However, it is worth noting that when comparing the upper and lower quartiles for both Narwhal and VirtualBox that there is minimal difference for 30% and 60% although there is significant difference in variation. Because the upper and lower quartiles are similar this indicates that the interquartile range for the CPU values is similar across both the systems and shows a good cross platform control for both Narwhal and VirtualBox when the set point is at 30% or 60%.

Chapter 6

System Model Identification

LQR controllers take advantage of a model for the system, allowing them to make better predictions and react faster to disturbances than PID controllers. Because of this model-based approach to utilise an LQR controller, an accurate model of how the CPU and memory behaved in a running instance of GUIGC was required. To build a model of GUIGC, a testing method that would provide a good range of different combinations of hashes, sleep, buttons, depth, and breadth while also being able to complete the testing in a reasonable amount of time would be required. 3 different things were set that the tests needed to accomplish. Firstly, all the tests needed an adequate amount of time in order to stabilise, allowing the result for the change in CPU and memory to be as close to 0 as possible. To achieve this stabilisation, a runtime of 2 minutes was used for each chosen combination. Secondly, during testing for the PID controller, it was found that the first few results could be inaccurate, as shown by the variation at the beginning of Figure 6, so in order to prevent these from skewing the averages that were calculated at the end, the first 20 values were skipped. Thirdly, to ensure that these tests were accurate and not just produced due to disturbances that occurred during the time they were running, each chosen combination was repeated 8 times and then averaged to calculate the final result that would be used to evaluate the overall model through linear regression.

6.1 Data Point Sampling

One of the biggest challenges in evaluating the system was the different ranges that were involved with the data, e.g. Sleep and hash ranged from 0-10000, while depth was 0-5, breadth was 0-10 and buttons were 0-100. Hash and sleep have little effect on the overall system hence why their values have such a large range, whereas if depth, buttons, and breadth are set too high it can start to cause out of memory errors. In order to accurately evaluate the model, as many of these different combinations would need to be tested while also running in a reasonable amount of time. In order to meet these criteria, it was decided that a random sample of 2000 for each of the values would be used, producing a wide range of different combinations that could be run during testing. By doing this we can control the values that are run, while also being able to control the runtime for the overall system tests.

6.2 Identifying the Model

Once all the data points have been sampled and run 8 times, they were averaged together to produce a result that would be used in the evaluation of the system. Using the data produced, linear regression was then applied to give a final model for the system that can be used in the control matrix B.

$$B = \begin{bmatrix} a_{CPU} & b_{CPU} & c_{CPU} & d_{CPU} & e_{CPU} \\ a_{MEM} & b_{MEM} & c_{MEM} & d_{MEM} & e_{MEM} \end{bmatrix} \tag{6.1}$$

Using the Equation below, where values a, b, c, d, e, and m were found through the use of linear regression the results of which are shown in Table 6.1, we could then start to predict CPU and memory values for other points

in the system.

$$\begin{aligned}\ln(z) - m = &\ln(hash) \times a + \ln(sleep) \times b + \ln(buttons) \times c \\ &+ \ln(depth) \times d + \ln(breadth) \times e\end{aligned} \tag{6.2}$$

When evaluating the system there were some limitations in how the model could be presented, for the equation to be applicable to the LQR controller, then a linear model is required. However, due to the way the system works there are many aspects that are not linear, so to retrieve better results from the model it was decided through trial and error that using the natural logarithm for each of the control vector parameters and the state vector was a better approach when using linear regression. The system matrix A informs the LQR controller of how the system would evolve over time without any type of control. However, due to the lack of any current equations to represent this for CPU and memory, it was decided the use a simple proportional differential equation where the change in CPU and memory is their current values multiplied by some negative constant. The equations shown below represent the change equations that were used in the A to predict how the CPU and memory would change over time.

$$\frac{dc(t)}{dt} = -\frac{1}{150}c(t) + u(t)$$

$$\frac{dm(t)}{dt} = -\frac{1}{200}m(t) + u(t)$$

In both equations shown above the value $u(t)$ is assumed to be 0 indicating that there is no interaction between the CPU and memory, this does however introduce issues such as the A matrix not representing the full interactions between CPU and memory. However, it was decided that in order to simplify both the equations and the A matrix that the interactions between the CPU and memory would be excluding from this thesis and expanded upon in future work. Both the CPU and memory stabilise at different rates and were decided

based on where the system stabilised over time when left with no interaction from the parameters in the control vector. Finally, in order to find the values for the system matrix A shown below, it was required to find the point where the system would stabilise under no control being applied.

$$A = \begin{bmatrix} -0.007 & 0 \\ 0 & -0.005 \end{bmatrix} \tag{6.3}$$

The system was tested for 10 minutes then averaged across 8 repetitions, taking the point where the system matrix would stabilise as the constant value used in the change equations. The u and x vectors hold the control parameter and start of the system, both x and u use the natural logarithm in order to compensate for the non-linear trends that were found in the system model.

$$u = \begin{bmatrix} \ln(hash) \\ \ln(sleep) \\ \ln(buttons) \\ \ln(depth) \\ \ln(breadth) \end{bmatrix} \tag{6.4}$$

$$x = \begin{bmatrix} \ln(CPU) - m_{CPU} \\ \ln(MEM) - m_{MEM} \end{bmatrix} \tag{6.5}$$

The state vector x in order to include the intercept in to the LQR matrix it was required to modify the state vector so that the value of m was subtracted from the state calculated by the LQR controller. The modification of the state vector required that when reading the state outside the controller we were needed to make a slight change to the state before taking the exponential shown in the equation below.

$$y = \begin{bmatrix} \exp(CPU + m_{CPU}) \\ \exp(MEM + m_{MEM}) \end{bmatrix} \tag{6.6}$$

	CPU	MEMORY
a	0.1 ±0.01	0 ±0.002
b	-0.26 ±0.01	-0.002 ±0.002
c	0.41 ±0.01	0.005 ±0.003
d	1.89 ±0.02	0.046 ±0.004
e	0.83 ±0.01	0.019 ±0.003
m	0.4 ±0.1	4.23 ±0.03

Table 6.1: Linear Regression Model Values

6.2.1 Model Optimisation

Unlike PID controllers which utilise gain values, the optimisation of an LQR controller utilises a gain matrix K, while the gain values for a PID controller can be difficult to decipher their effect on the system, a gain matrix is calculated in an easier to understand manner. The matrix K is calculated by making use of two other matrices Q and R, where Q is the state cost weighted matrix and R is the control parameter cost weighted matrix along with a cost function J.

$$J = \int_0^\infty \left(\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}^T \begin{bmatrix} Q_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & Q_n \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix}^T \begin{bmatrix} R_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & R_n \end{bmatrix} \begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix} \right) dt \tag{6.7}$$

Q and R represent easy to understand ways to manipulate the optimisation of an LQR controller to the desired effect of something like the cost of increasing a control parameter or maintaining a state. Although the Q and R matrices can be adaptive throughout the LQR process for this thesis static Q and R values were used (shown below) for both the Static LQR controller (Section 7.2) and the PID Assisted LQR Controller

(Section 7.3).

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, R = \begin{bmatrix} 0.1 & 0 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 4 \end{bmatrix} \tag{6.8}$$

The values for Q were set to the identity matrix for the LQR experiments and the R values were chosen based on the cost to bringing the system close to out of memory errors, where Hashes and Sleeps do nothing to cause this so their values are set low, however, Buttons, Breadth, and, Depth to cause this effect so their values are set accordingly. These values were set arbitrarily in order to provide a goal driven adaptation (Wave IV [34]) by penalising values in R

6.3 Limitations of Model

Currently, there are many limitations of this model, the major one is that LQR can only model a linear system, however, GUIGC is non-linear, this could cause some inaccuracies with the actual results and the predicted values from the model. LQR has previously been used in engineering where there are equations that have been tried and tested to accurately model a system, however, when modelling CPU and memory there is no simple way to capture all the parameters of a system that are going to affect how they behave, causing some variation between what can accurately be predicted by the model and the variation that is produced within the system. Currently, as it stands in this project the system matrix (A) assumes that there is no interaction between the CPU and memory, however, we know this to not be true due to how Java controls memory, e.g. GC. Future work in this area would be to analyse and implement the interaction between CPU and memory into the equations

for the system matrix (A) and accurately portraying the relationship between states. The system is limited by the amount of testing that has been done, due to only a select range of values for the control vector being tested this leaves unpredictability in the system when going above these values, due to assuming a linear connection that may not exist outside the tested range. To provide better guarantees beyond this range either more testing is required or more adaptive modelling of the system during runtime, the system is also limited by the available control vector parameters each one having an effect on both the CPU and memory. The influence that the control vectors have on both CPU and memory means that there is no guarantee that the system will stabilise to the selected set points for both CPU and memory. Finally, the JVM also limits the ability of the system to be controlled, due to Java being an automatic memory managed language this introduced some unpredictability in the measurements, as in the current model of the system there is no way to separate other processes running on the JVM in MXBeans. Due to the limitation of MXBeans, future work could better the guaranties by improving the accuracy of the observability into the system.

	Residual Standard Error on 1994 DoF
CPU	111
log(CPU)	0.59
Memory	9.07
log(Memory)	0.13

Table 6.2: System Model Comparison

Chapter 7

LQR Evaluation

PID controllers require little knowledge of the desired system while PID controllers provide some guarantees of stability when controlling a single state. However, when attempting to control multiple states the guarantees provided by the controllers become less predictable, especially when these multiple states have conflicting control parameters. A more effective way of controlling multiple states while providing better guarantees of the controller's performance is to utilise a Linear Quadratic Regulator controller instead of Proportional Integral Derivative controllers.

7.1 LQR Controllers

LQR controllers provide many advantages over PID control, one of them is the ability to utilise not just control but optimisation of the control parameters in a system. Unlike PID controllers, LQR controllers make use of a K matrix (shown in the equation below), which allows for control of a system through the use of optimisation.

$$\dot{x} = Ax + B(-K(r - x))$$

However, one of the negatives of multiple state control is that there are two or more different set-points present in the controller making it difficult to exactly

meet these desired set-points in the system. Due to multiple set points, it means that the results from the LQR controller in this thesis may never meet their set points however, LQR should still provide stability in the system even if the set-points are never meet. In this thesis the LQR controller optimises the control of the system through the use of the two matrices Q and R shown below, the values of these matrices were used for both Static LQR Control (Section 7.2) and PID Assisted LQR Control (Section 7.3).

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, R = \begin{bmatrix} 0.1 & 0 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 4 \end{bmatrix} \tag{7.1}$$

Q and R were chosen based on the relative cost of how related the parameters are to the states of the system where parameters such as buttons breadth and depth have higher values than hash and sleep. Also, note that buttons have a value of 5 which is higher than depth and breadth, this is due to the effect of when buttons would reach higher values it would cause the GUI interface to require longer times to build and would even seem to cause the system to freeze, so to prevent this from occurring in the system a higher value was set for buttons. The other values, 0.1 for hash and sleep were chosen because they cost little to increase and have a smaller effect on CPU and memory, whereas Depth has a value of 4 because if the depth is increased to high it has the potential to cause out of memory errors. Finally, breadth with a value of 3, while can still cause out of memory errors if set too high it can be raised higher than depth. Future work for LQR controllers would be to expand on better ways to choose optimisation values for Q and R, although in a different system optimisation values can be chosen based on some real-world cost such as money or some cost to the utilisation of the system. However, for other systems, there is a little way to know how

to optimise the system other than through educated guesses for the values in Q and R. Each of the two sections 7.2 and 7.3 the tests were run on the same system under two different sets of resource criteria, the first was run on Narwhal and the second running in a VirtualBox that was limited in the number of CPU cores and memory. For this system the LQR controller was trained on Narwhal this model was then used on both Narwhal and VirtualBox.

7.2 Static LQR Controller

For this thesis, a static LQR controller utilising the full-scale reference system equation $\dot{x}$ as a way of controlling the system states (CPU and Memory), where the r vector is a static vector representing a one to one set point for the state vector.

$$\dot{x} = Ax + B(-K(r - x))$$

The aim of this is to assess how just a basic LQR controller will perform when being utilised to control two states where the control parameters have the potential to influence both of the monitored states. While the experiments ran did show some attempt at control in the system especially with the CPU it also showed some flaws with utilising an LQR controller and showed that more fine-tuning around the optimisation of the Q and R matrices are required to provide more varied results and a less conservative system. The static LQR experiments also revealed the difficulty in controlling the memory utilisation of GUIGC as the results produced show little to no control over the memory that was utilised in the overall runtime of the system.

7.2.1 Narwhal Memory Utilisation

When analysing Figures 7.1, 7.2, 7.3 and 7.4, it becomes apparent that the majority of the means across all the graphs remain focused within similar ranges of memory utilisation. Along with the similarities in the lower and

upper quartile range leads us to conclude that the controller is having very little influence over the memory utilisation within the system. We can draw this conclusion of control over the memory due to the spread of the memory utilisation across all the tests if the controller was having some influence over the memory utilised the lower and upper quartile ranges would be much closer together like what is shown in Figure 7.1 for 150%, 210%, and 310% CPU.

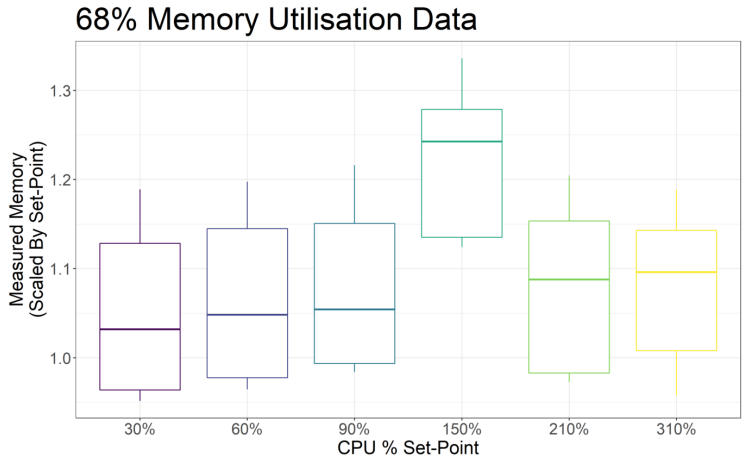

Figure 7.1: Narwhal 68% Static LQR Memory Utilisation

We know the lower and upper quartile ranges should be close together because that is where the controller should be shifting the majority of the memory utilised towards in order to control the system to a specific point. Figures 7.3 and 7.4 shows that the mean of the data is around the desired set-point of the system, however, due to the near similarities in the box plots across all the figures we can conclude that this is nothing significant and it is just that the memory utilisation set-point was placed where the mean of the data would have been rather than this being a cause of the controller. We can also conclude that the means in Figures 7.3 and 7.4 are not significant by also looking at the spread of the data that was mentioned before.

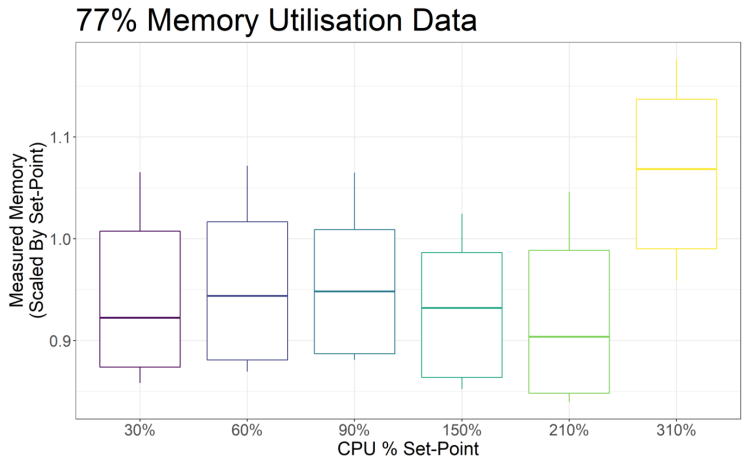

Figure 7.2: Narwhal 77% Static LQR Memory Utilisation

Finally, the data shows that the memory utilisation starts at or close to 68% and ends at or close to 95%, and this is consistent across all the figures for the memory runtime data (Figures 7.1, 7.2, 7.3 and 7.4). Because of the consistency in the min and max of the starting and ending memory utilisation across all figures along with the near symmetry between the box plots we can conclude that across all the experiments the memory is following the same trend providing further evidence that there is little control being shown in the utilisation of memory.

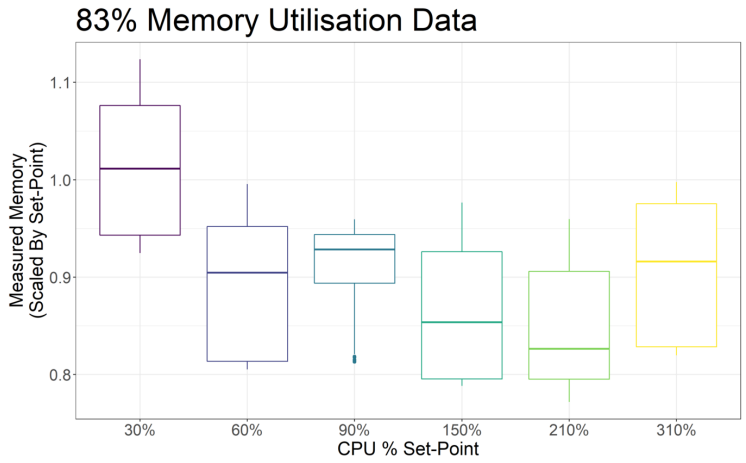

Figure 7.3: Narwhal 83% Static LQR Memory Utilisation

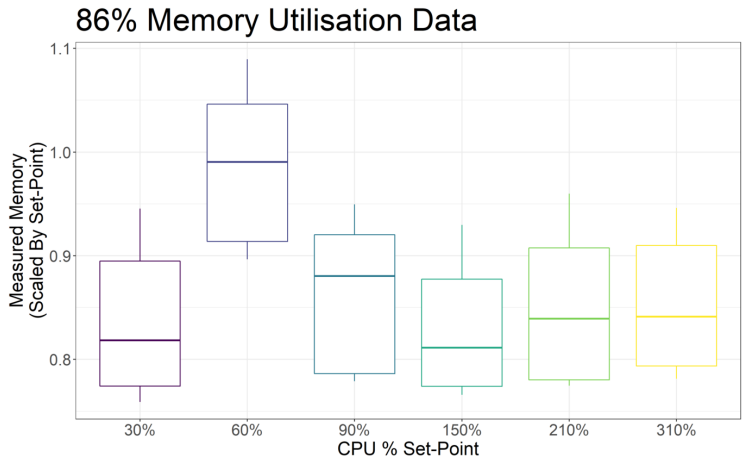

Figure 7.4: Narwhal 86% Static LQR Memory Utilisation

7.2.2 Narwhal CPU Utilisation

The Q1 and Q3 ranges in Figures 7.6, 7.7 and 7.5 are close together to the point where they are almost the same value, this gives a strong indication that the static LQR controller is controlling the CPU utilised in the system quite well. However, just having the Q1 and Q3 ranges being close together is only half of what is required to show that the system is being controlled correctly towards the desired set-point, we also need the mean of CPU utilised to be near or as close to the set-point as possible. Although it is unlikely that the CPU utilised will match the set-point exactly, we can see from Figures 7.9 and 7.8 that the mean of the CPU utilised is failing to get close to the desired set-points.

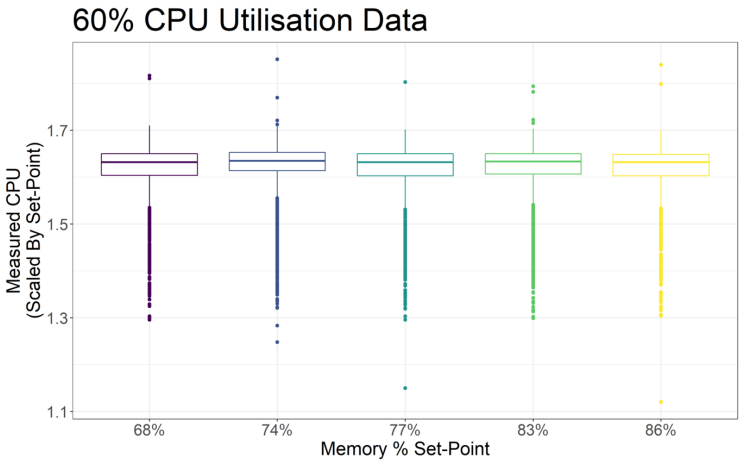

Figure 7.5: Narwhal 60% CPU Static LQR CPU Utilisation

Although, it looks from Figures 7.6, 7.7 and 7.5 that the mean of the CPU utilised is significantly close to the desired set-points and being controlled by the static LQR controller. It appears that Figures 7.6, 7.7 and 7.5 prove to be significant we can conclude that this is not the case and is a matter of the set-point being set to a value that the CPU would have been controlled to whether the set-points were set to 90%, 150%, and 60% or not, and the controller has no control over meeting the desired set-points. We know that the controller is not meeting the desired set-points because looking across all Figures 7.6, 7.7 and 7.5 we can see that the CPU utilised remains very consistent, leading us to believe that the result in Figures 7.6, 7.7 and 7.5 would have occurred even if the set-point was not set the shown values. Finally, because of all the Figures 7.6, 7.7 and 7.5 being similar to each other although we can show control of the CPU utilised, there is no variation among the graphs leading this thesis to conclude that the CPU utilised is not changing as different set-points are used.

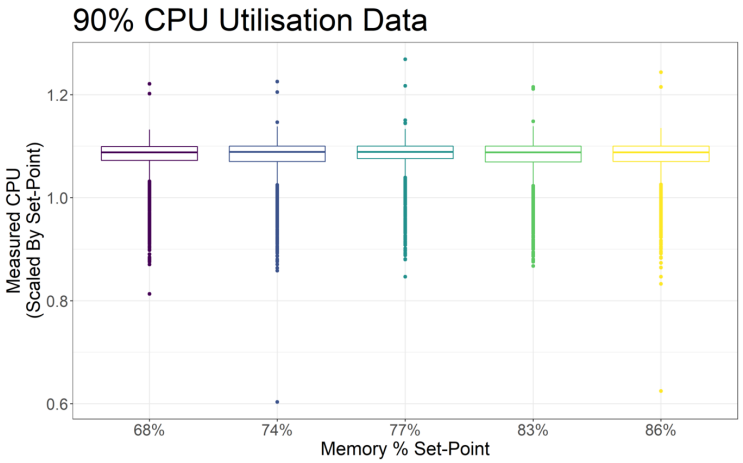

Figure 7.6: Narwhal 90% CPU Static LQR Utilisation

The inability for the controller to change the CPU utilised over different set-points is due to the conservative nature of the Q and R matrices where the values set for buttons, breadth, and depth are set too high preventing the LQR controller from making any significant changes over different set-points. Looking at Figure 7.19 which utilised a different R matrix equation shown below while keeping the Q matrix the same, we can see that by providing am more liberal approach to optimisation we can move the mean of the CPU utilisation closer to the desired set-point.

$$R = \begin{bmatrix} 0.1 & 0 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{7.2}$$

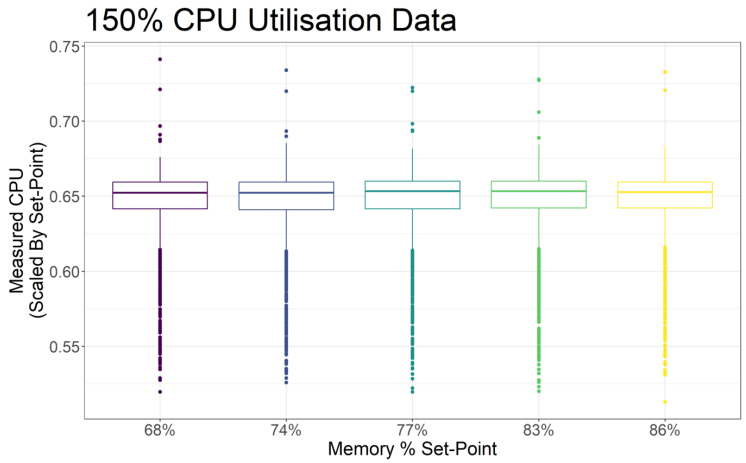

Figure 7.7: Narwhal 150% CPU Static LQR Utilisation

However, in giving more freedom to the R matrix we have lost some control ability in the CPU utilisation with a much larger range between the lower and upper quartile. The major differences between meeting the set-point in Figure 7.19 and having far better control ability in Figure 7.9 show the need for much better ways of fine-tuning the optimisation of the system in order to provide the best of both control ability and meeting the desired set-point.

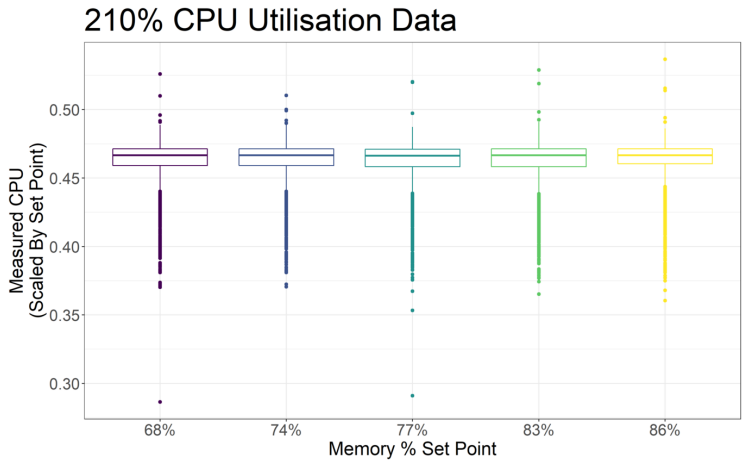

Figure 7.8: Narwhal 210% CPU Static LQR Utilisation

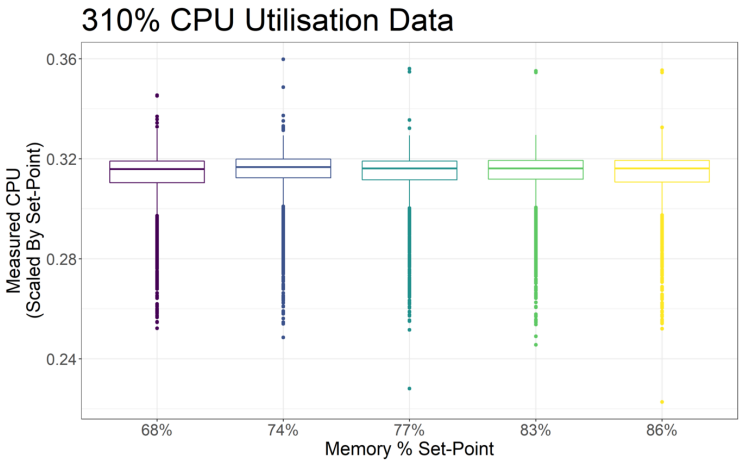

Figure 7.9: Narwhal 310% CPU Static LQR Utilisation

7.2.3 VirtualBox Memory Utilisation

When looking at the memory utilisation of the Static LQR controller its clear that there is more variation between the CPU set points. Looking at Figure 7.1 and 7.10 we can see that for Narwhal the memory utilisation remains consistent across different CPU set points, except for 150% utilisation, however, when looking at VirtualBox we can see that the different CPU set points utilise different amounts of memory across different CPU set points. The second thing that stands out in the data is that the memory utilised remains consistent when compared with the same CPU set points across different memory set points.

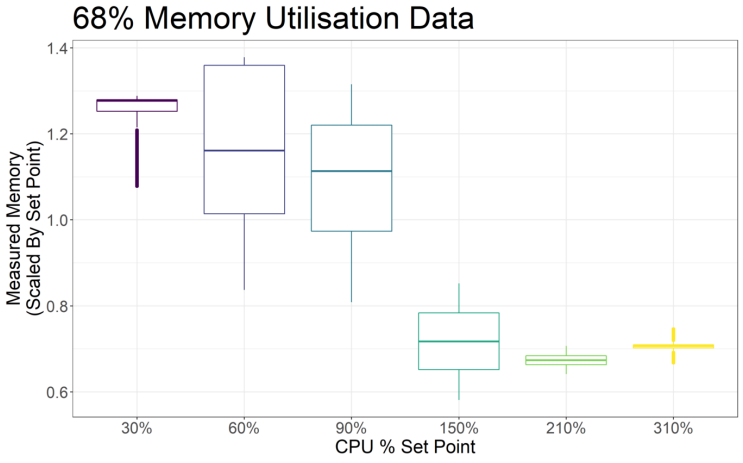

Figure 7.10: VirtualBox 68% Memory Static LQR Utilisation

Looking at Figures 7.10, 7.12, and, 7.13 for all the CPU set points we can see that the variation is minimal e.g. 150% CPU utilisation is consistently has a mean around 0.7 and 0.8 with the same distribution on the upper and lower quartiles. This gives the indication that unlike Narwhal we are changing the utilisation of memory across CPU set points, however, there is little to no variation between changing the memory set point. Figures 7.12 and 7.11 for 150% CPU utilisation shows to be consistently maintaining a median that is close to or matching the desired set point for memory, however, the other CPU set points consistently do not maintain a significant median at the desired set point except Figures 7.11 and 7.12 for 30% and 60% CPU respectively.

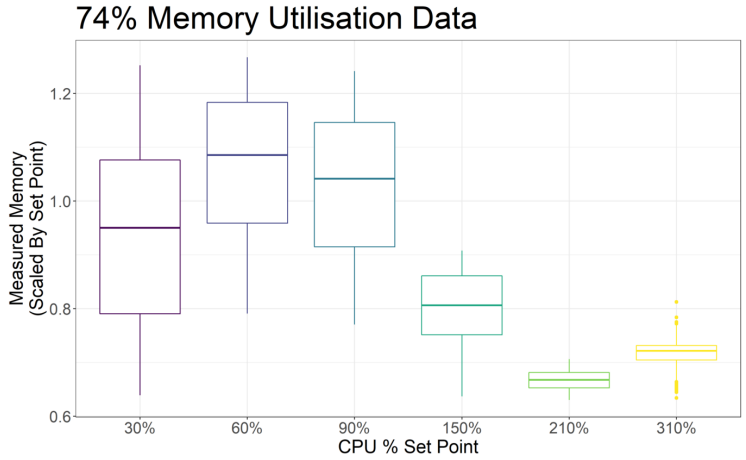

Figure 7.11: VirtualBox 74% Memory Static LQR Utilisation

The similarities between the same CPU utilisation for each of the different memory set points give indication that the static LQR controller is capable of maintaining a consistent utilisation of memory, however, just not able to meet the desired set points. It is also worth mentioning that as the CPU utilisation set point increases the variation in the memory decreases, this is likely due to more freedom to raise the parameters that control memory and CPU as the set point for the CPU is higher and not hit as quickly.

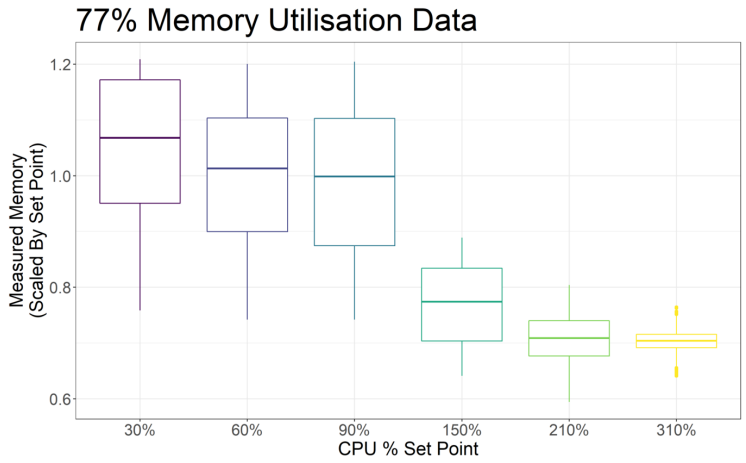

Figure 7.12: VirtualBox 77% Memory Static LQR Utilisation

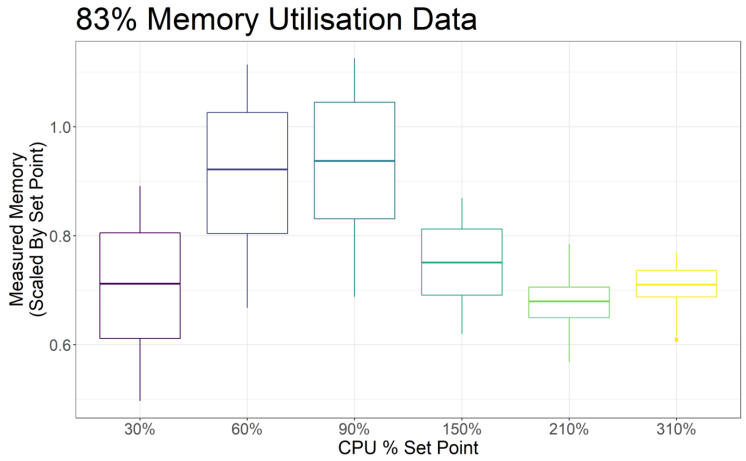

Figure 7.13: VirtualBox 83% Memory Static LQR Utilisation

7.2.4 VirtualBox CPU Utilisation

When comparing the CPU utilisation for Figures 7.6 and 7.15 we can clearly see that 90% CPU is the easiest set point for the Static LQR controller to control, as both figures consistently have the median hit 90%. The main cause for this is that looking at Figures 4.1 and 4.2 which show the system under zero control the CPU utilised is around 100% and since the CPU natural is around this point with no control we can conclude that the 90% to 100% will be the easiest set point to control.

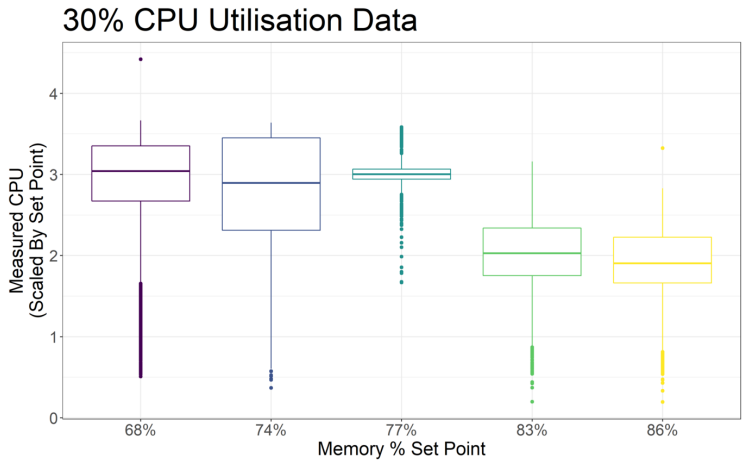

Figure 7.14: VirtualBox 30% CPU Static LQR Utilisation

Apart for some slight differences in Figure 7.14 the rest of the Figures 7.15, 7.16, and, 7.17 remain very consistent across the different memory utilisations indicating the Static controller is controlling to the same point for each of the set points. In Figures 7.15, 7.16, and, 7.17 will notice that only Figure 7.15 is meeting the desired set point the others show good control but do not meet the required set points. Finally, due to the LQR model being trained on Narwhal and then that same model being utilised for VirtualBox this leaves room for more in accuracies because although they may run the same program it is highly likely that a different model will be required in order to fully control VirtualBox.

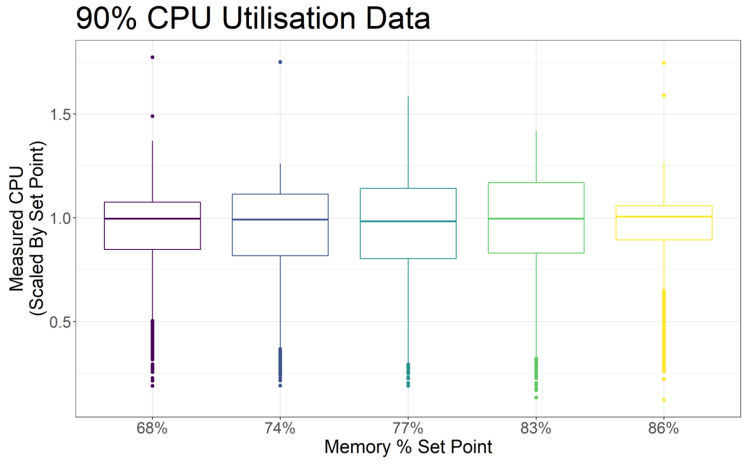

Figure 7.15: VirtualBox 90% CPU Static LQR Utilisation

The utilisation of the same model for narwhal in VirtualBox will also be a contributor to the increased variation specifically for Figures 7.15 and 7.14, and future work may find that more adaptive system models may produce better control across systems.

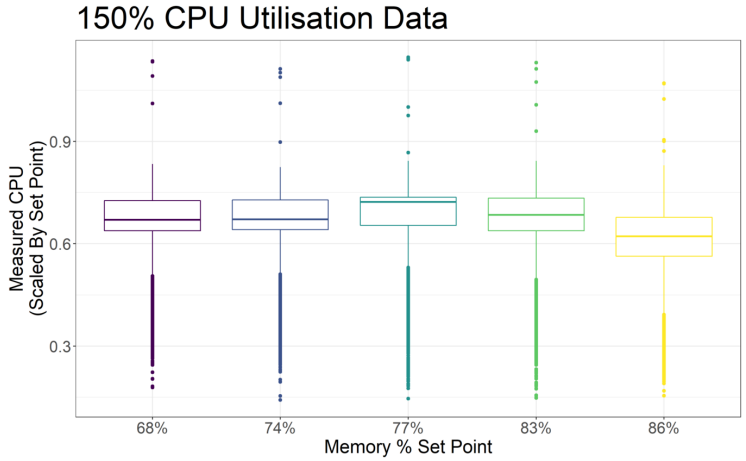

Figure 7.16: VirtualBox 150% CPU Static LQR Utilisation

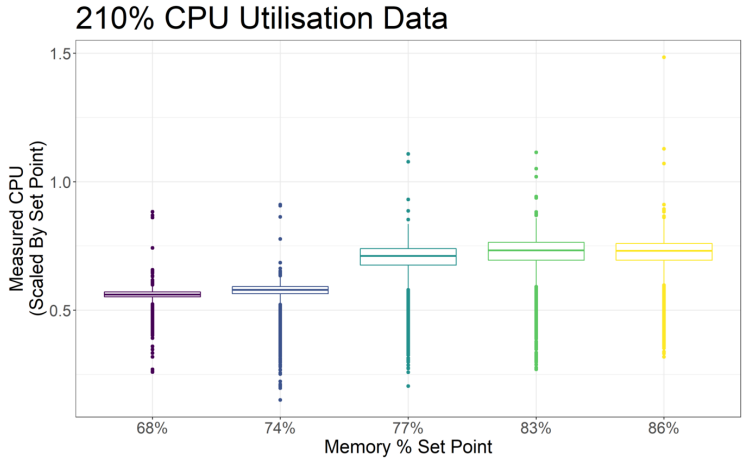

Figure 7.17: VirtualBox 210% CPU Static LQR Utilisation

Control Parameters	Average	Median	Standard Deviation
Hash	1	1	0
Sleep	14.0	14	0.05
Buttons	1	1	0
Depth	5	5	0
Breadth	2	2	0

Table 7.1: 30% CPU and 86% Memory Utilisation Control Parameter Statistics

Contol Parameters	Average	Median	Standard Deviation
Hash	80.74	81	3.69
Sleep	1	1	0
Buttons	1	1	0
Depth	5	5	0
Breadth	5.0	5	0.04

Table 7.2: 210% CPU and 86% Memory Utilisation Control Parameter Statistics

7.3 PID Assisted LQR Controller

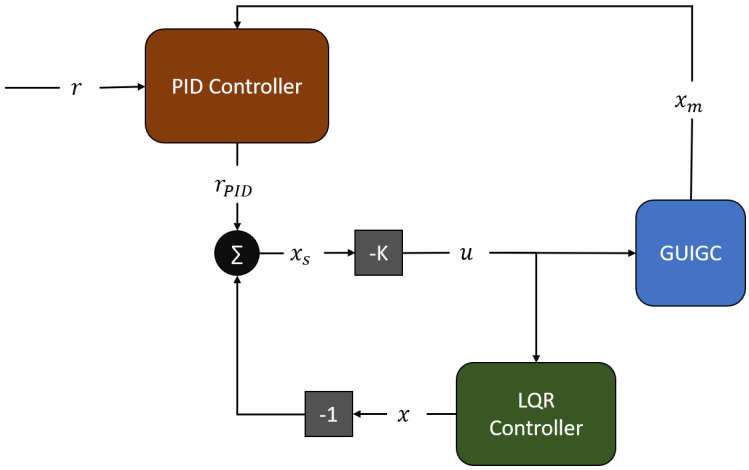

Figure 7.18: PID Assisted LQR System Diagram

PID Assisted LQR has been done before in the paper Raj et al. [30], however, the implimentation in this thesis is different. Raj et al. [30] combined the output of PID and LQR controllers into one where as this thesis is using PID controller to manipulate the setpoint values used in the LQR controller. PID Assisted LQR control was introduced as a way of attempting to combine a system that is conservative e.g. utilises the same Q and R matrices, while also getting the system to meet the set point rather than stay at the same CPU utilisation across all tested set points. An issue with a LQR controller is that it relies on a model which keeps track of where the states are internally, this causes a situation in unpredictable systems where the internal states don't match the external measured ones from GUIGC. Hence, the idea behind a PID

Assisted LQR controller is that the PID controller will monitor the external states (CPU and memory) of GUIGC and then update the set points feed into the LQR controller accordingly. Figure 7.18 displays a simplified version of the PID Assisted LQR Controller, where r is feed into the PID controller along with x_m (GUIGC Java measured states), the PID controller will then output r_{PID}.

	Proportional Gain	Integral Gain	Derivative Gain
CPU	0.007	0.007	0.007
Memory	0.005	0.005	0.005

Table 7.3: PID Assisted LQR Gain Values

The matrix r_{PID} represents the new set of set points that are based off the error between the measured and the desired set points for the system. PID gain values are shown in Table 7.3 and were required to be small otherwise the LQR controller would not have enough time to adapt to the required changes in the set points. For this thesis it was decided that the absolute values from the A matrix would be used as a starting point and future work could be to find optimum ways of selecting these values for a system. The next step in the system is the scaled reference LQR feedback controller which works the same as described in Section 2.1.6 where the difference between the calculated x matrix and the r_{PID} matrix are taken and feed into the K matrix to calculate u. The u matrix is then utilised in the next step of the LQR controller and the control parameters are updated in GUIGC to reflect this new u matrix.

7.3.1 Methodology

When looking at Figures 7.1, 7.2, 7.3 and 7.4 we can see that there is control in the memory that is utilised by the system when under a static LQR control, while the CPU utilised (Figures 7.6, 7.7 and 7.5) show control of the CPU utilised, however, fails to meet the desires set-points of the system. When looking at Figures 7.19 and 7.20, we can see from Figure 7.19 that we can

meet the set point better, but we also lose control over the CPU utilised, and when looking at the memory utilised in Figures 7.20 we can see that although we have got better control over the memory utilised we have lost control in the CPU utilised (Figure 7.19). Due to the loss in control over the CPU utilised and the similar patterns being followed between the memory, this thesis analysed a different LQR technique that utilised a PID controller as a way of changing the system slightly to give better control over the two desired achievements of the system (better meeting of the set-point and finer control over the system states).

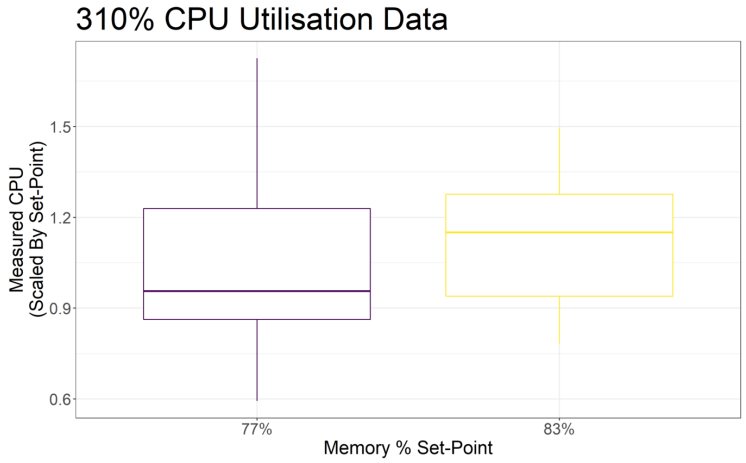

Figure 7.19: Narwhal 310% Static LQR CPU Utilisation with Alternative R Matrix

To achieve the two desired points, this thesis introduces a Proportional Integral Derivative assisted Linear Quadratic Regulator controller, to achieve this a PID controller has attached to the r (shown below) matrix where the desired set-points were scaled up and down through the PID controller depending on how good the set points were meet. However, this approach does have its drawbacks that were needed to be overcome before it could be utilised properly in the LQR controller the major of which was a continually increasing PID controller. The continual increasing PID controller was caused by the effect of the LQR controller where because the controller had two set-points it is common for the controller to find the middle ground

where it finds a point that satisfies both set-points while not meeting either one exactly. Due to the LQR controller effect, this meant that the desired set-point may never actually be met, and because the PID controller was assessing whether it should increase the set-point based off how well the controller meet this set-point in the running system it would then lead to a controller that would continue to increase moving all the other states away from their desired points while attempting to meet this one state.

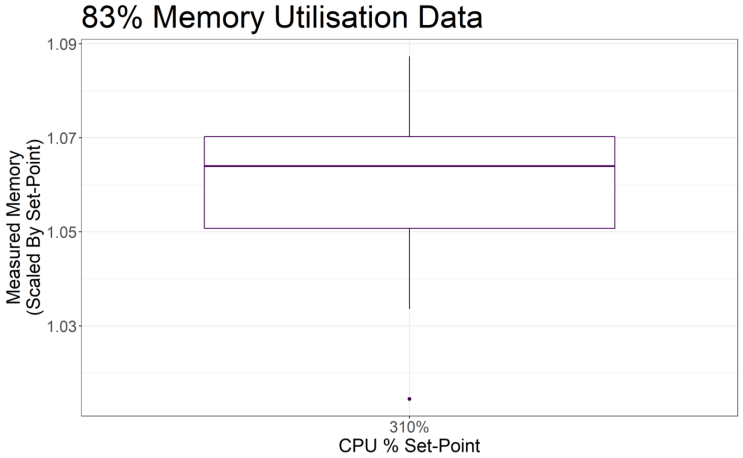

Figure 7.20: Narwhal 83% Static LQR Memory Utilisation with Alternative R Matrix

To correct the continuously increasing PID controller a toggle switch was introduced that would prevent the PID controller being run if it looks like the system is going above the desired set-point of any state (CPU or Memory), to detect this a running average of five steps back was introduced, so we know the system is trending towards the set-point and it was not just a spike in the measurements. However, by using the average it became apparent that we could overshoot the desired set-point leading an increase across all the states, so to correct this error a decay was introduced once the PID controller was stopped to bring the set-point back down. By introducing a PID assisted LQR controller it introduced more variation into the system, however, in some cases with the memory in introduced not only much greater control over the memory utilised it also in some cased achieved better control of the system while also

meeting the set-point that was desired for both states as shown in Figures 7.22 and 7.21 for 68% memory utilisation and 150% CPU utilisation. In the system, there were also far more set-points for the CPU utilisation being meet and a reduced similarity in CPU utilisation across different set-points.

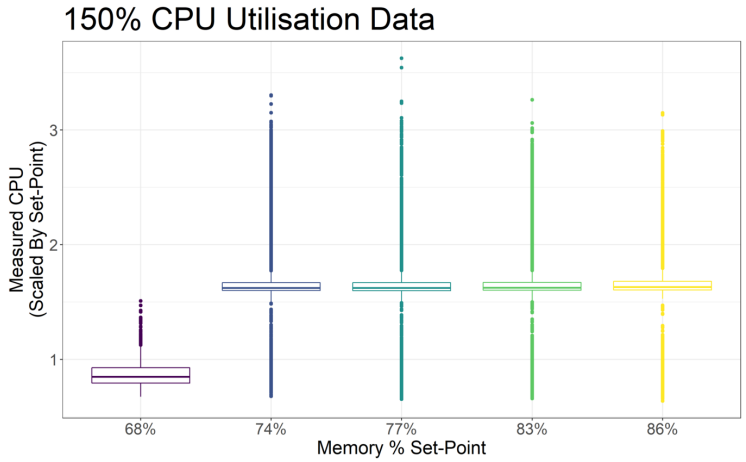

Figure 7.21: Narwhal 150% PID Assisted LQR CPU Utilisation

7.3.2 Narwhal Memory Utilisation

When analysing the memory utilisation shown in Figures 7.22, 7.23, 7.24, and 7.25 it shows far better control and meeting the desired set-points that the previous static LQR controller (Figures 7.1, 7.2, 7.3 and 7.4) there is also greater variation between the different set-points giving an indication that the changing set-points are attempting to be met by the system. Looking at Figures 7.22, 7.23, 7.24, and 7.25 the CPU utilisation increases beyond 90% there are better results for both meeting of the set-point and control of the memory utilised. An effect of how the PID controller is turned on and off once a state is above the set-point given by the user because when the CPU utilisation is increased the state that is reached first is memory utilised which caused better control around the memory set-point, whereas when the CPU utilisation is below 150% the PID controller is turned off before the set-point for the memory is reached.

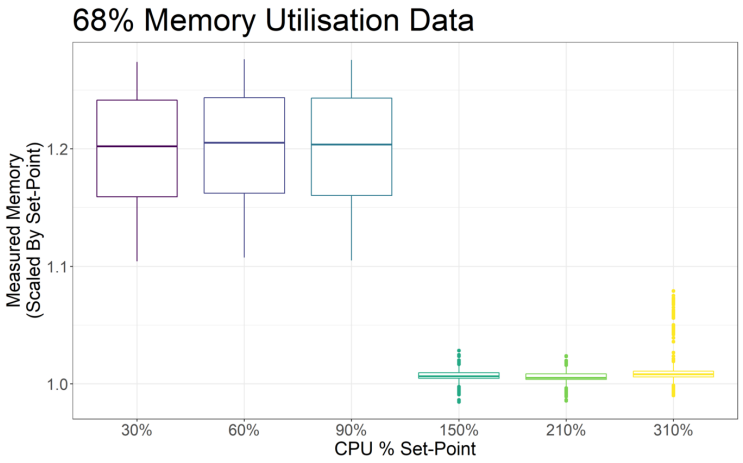

Figure 7.22: Narwhal 68% PID Assisted LQR Memory Utilisation

The low CPU utilisation effect is greatly shown in Figure 7.22 where we can see that the experiments with 30%, 60% and 90% CPU utilisation have far less control over the memory than the ones where the CPU utilised is 150% and above. When looking at Figures 7.22 and 7.23 we can see that at least three (150%, 210% and 310%) experiments are meeting or significantly close to the desired set-point however, when we look at Figures 7.24 and 7.25 we can see that there are no experiments that are meeting or significantly close to the desired set-point. There are two possible reasons for this, one as mentioned before, the memory utilisation set-point is too high and the PID controller is being turned off because it has reached the CPU utilisation set-point.

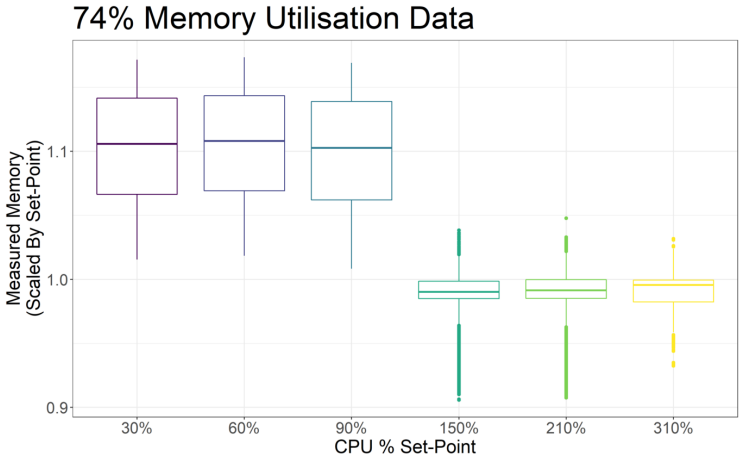

Figure 7.23: Narwhal 74% PID Assisted LQR Memory Utilisation

The second reason is that we could be approaching the limits of controlling the memory utilisation, this is hinted at but more testing would be required to confirm, by the similarities in the two figures (Figures 7.24 and 7.25) and how even though two different set-points are used the experiments have produced similar results in the Q1 and Q3 ranges. It is still worth mentioning that we do maintain significantly improved control over the experiments where the CPU utilisation is above 150% and less control when below 150%, even if we never get close to the desired set-point.

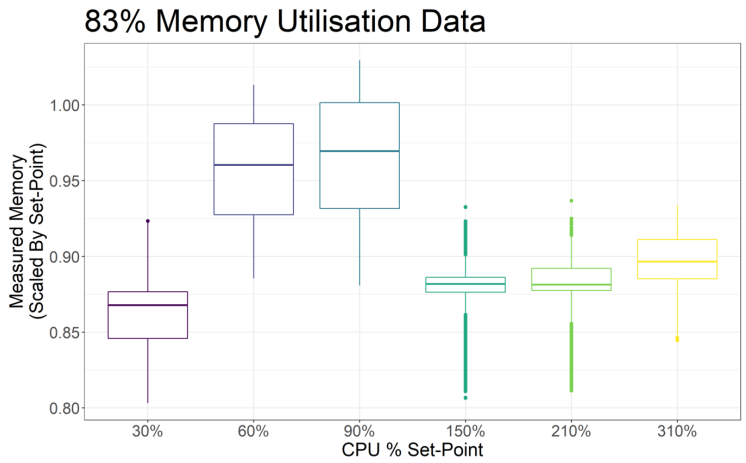

Figure 7.24: Narwhal 83% PID Assisted LQR Memory Utilisation

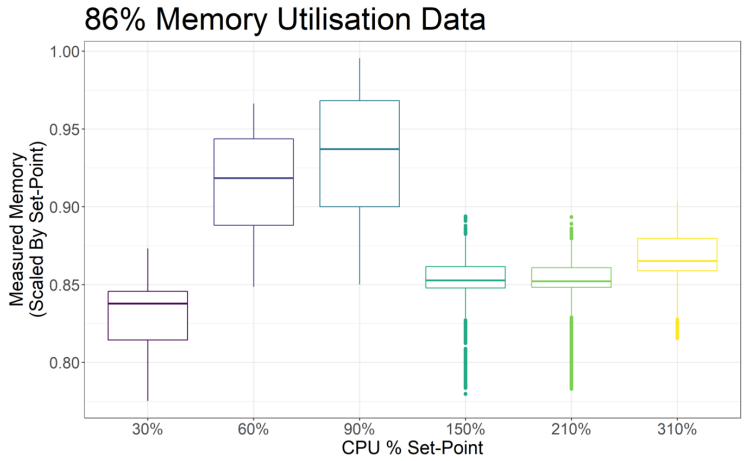

Figure 7.25: Narwhal 86% PID Assisted LQR Memory Utilisation

7.3.3 Narwhal CPU Utilisation

When utilising a PID assisted LQR controller for the CPU we can see that there is much greater variation in the different set-points (Figures 7.26, 7.27, 7.28, and 7.29) when compared to the static LQR controller (Figures 7.26, 7.27, 7.28, and 7.29), showing that the CPU is changing to attempt to meet the desired set-points. We can also see that in majority of the cases we still maintain good control over the CPU utilisation, except for 77%, 83% and 86% in Figure 7.29, 83% and 86% in Figure 7.26 and 86% in Figure 7.27, the cause of this is likely an effect of the bench-marking program utilised in that there is less control in the CPU when attempting to meet higher memory utilisation set-points due to having to render more objects on to the GUI which causes longer spikes in the CPU.

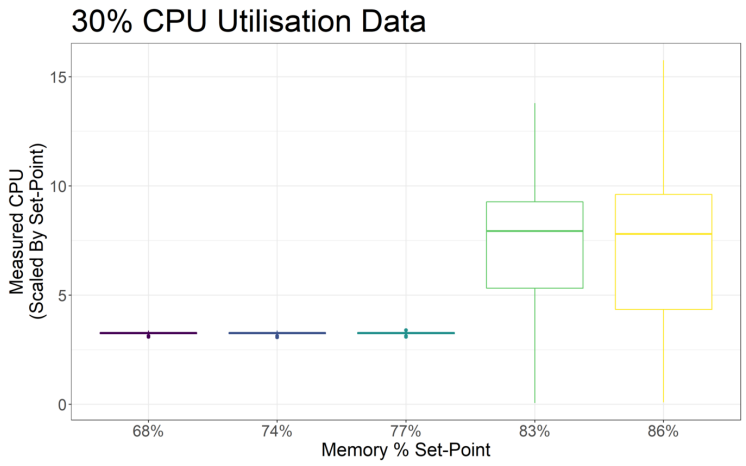

Figure 7.26: Narwhal 30% PID Assisted LQR CPU Utilisation

In the case of meeting the desired set points, we can see an increase in the CPU utilisation meeting or be significantly close to these desired set-points specifically with the 210% and 310% CPU utilisation shown in Figures 7.28 and 7.29, where for Figure 7.28 the set-points are significantly closer to the desired set-point while maintaining good control over the CPU utilised and for Figure 7.29 we are much closer to the desired set-point except for 68% memory utilisation, however, although we are starting to meet the desired set-point we

have lost some control in the CPU utilised in the system with more variation in the CPU, although as mentioned before this is likely an effect of controlling higher memory utilisation set points.

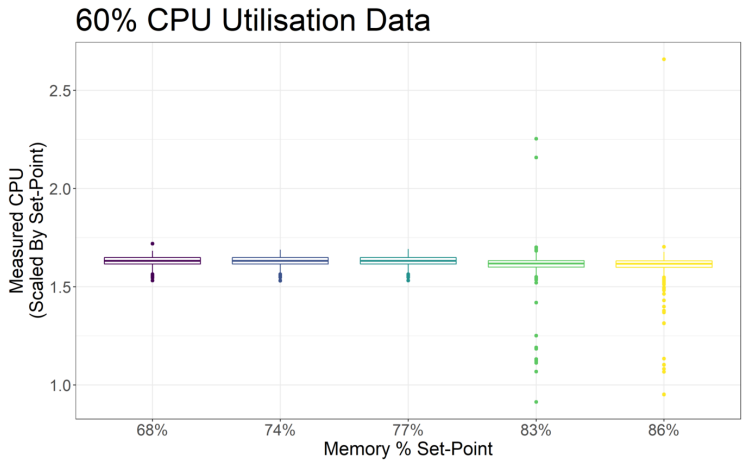

Figure 7.27: Narwhal 60% PID Assisted LQR CPU Utilisation

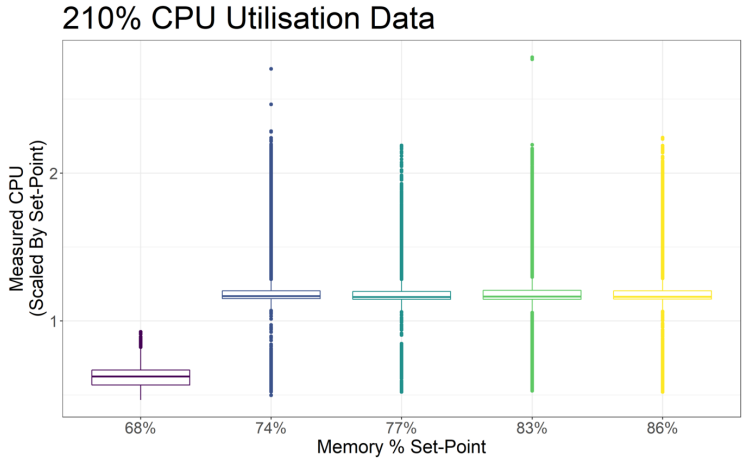

Figure 7.28: Narwhal 210% PID Assisted LQR CPU Utilisation

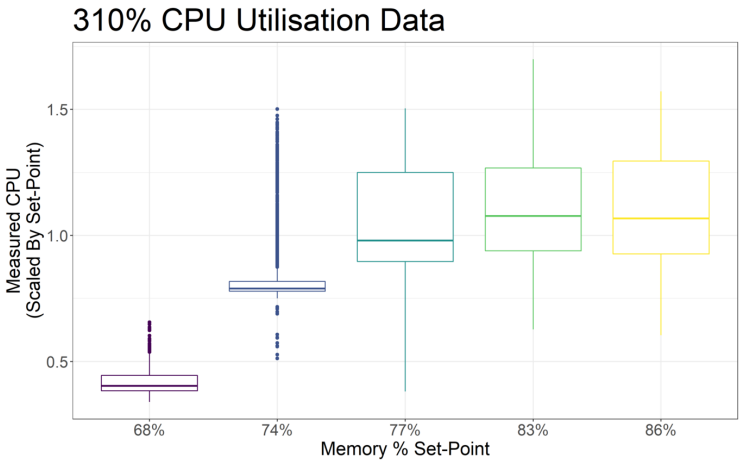

Figure 7.29: Narwhal 310% PID Assisted LQR CPU Utilisation

7.3.4 VirtualBox Memory Utilisation

The Static LQR controller did a good job of keeping consistent control across different CPU set points, due to the conservative nature. Hence, the introduction of a PID assisted LQR controller to attempt two things one was a better set point accuracy and the second was to try to reduce the variation that can not be modelled by the LQR controller equations. Memory utilisation is challenging at best to both control and model in memory managed systems as the garbage collector makes the decisions about when objects are removed from memory, hence why it is not unexpected that only Figure 7.30 shows the median being significantly close to the set point. However, it is worth noting that even though Figures 7.31, 7.32, and, 7.33 do not meet the set point as closely as Figure 7.30 they do remain consistently within the range or 0.8 to 1 except for Figure 7.33 being slightly lower than this range.

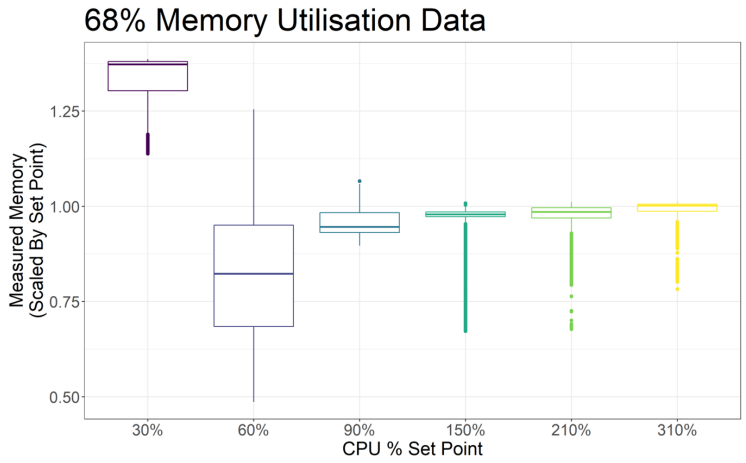

Figure 7.30: VirtualBox 68% PID Assisted LQR Memory Utilisation

Figure 7.30 shows the best results when looking at meeting the desired set point except for 30% and 60% CPU utilisation, the reason for these two CPU set points being the exception is due to the better control given to the LQR controller at higher CPU utilisations. When comparing Figure 7.30 to the Static LQR controller at the same memory set point (Figure 7.10), shows significant improvement in not only meeting the set point but also the variation. PID assisted LQR control shows better consistency in CPU utilisation above 90% showing better control in the memory utilisation across changing CPU set points that Figure 7.10.

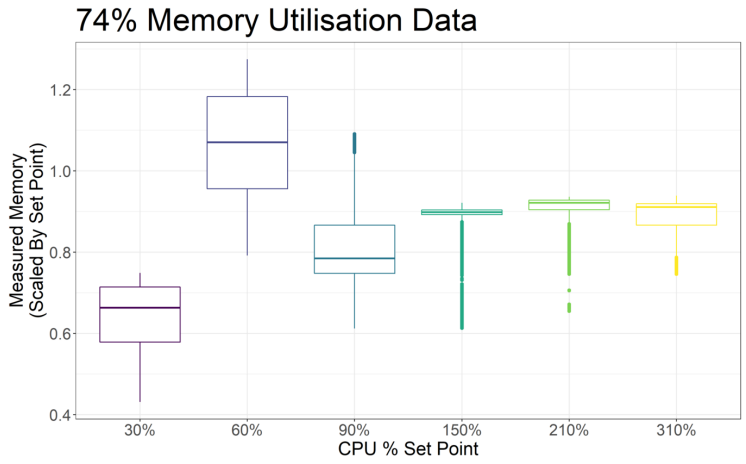

Figure 7.31: VirtualBox 74% PID Assisted LQR Memory Utilisation

When analysing Figures 7.31, 7.32, and, 7.30 we see further evidence of

better memory control in higher CPU set points, supporting the idea that there is more freedom to move the memory utilisation when the CPU set point is set high. 30% CPU utilisation as expected is much lower than all other CPU set points in Figures 7.31, 7.32, and, 7.33, however, in Figure 7.30 the Memory utilisation for the 30% CPU set point is much higher than the other CPU set points. Figure 7.30 being much higher is likely due to the same reason that Figures 7.31, 7.32, and, 7.33 are lower, except in the case of Figure 7.30 there is little freedom to stop the memory utilisation increasing.

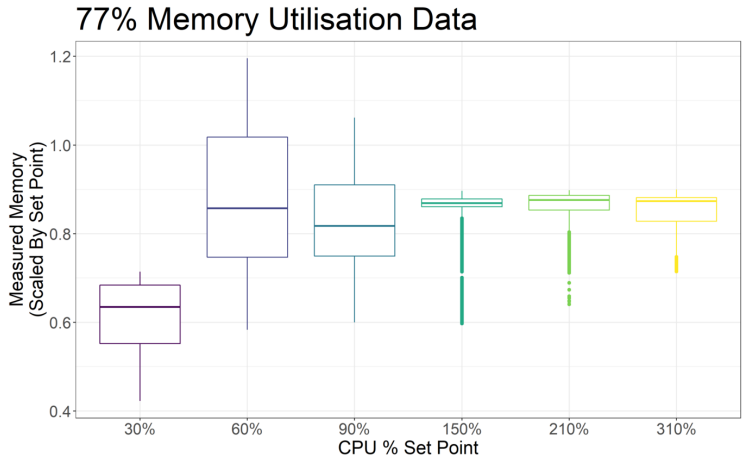

Figure 7.32: VirtualBox 77% PID Assisted LQR Memory Utilisation

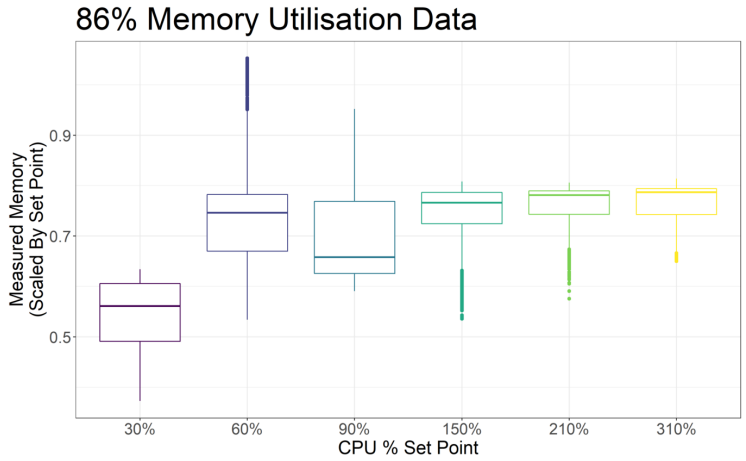

Figure 7.33: VirtualBox 86% PID Assisted LQR Memory Utilisation

7.3.5 VirtualBox CPU Utilisation

CPU utilisation is much easier to control in the system as the changes made to alter the CPU take effect in a predictable manner making it easier to put into the A matrix the system changes. However, even though CPU is easier to predict there still exists some unpredictability in the LQR model and the conservative nature chosen shown in Figures 7.14, 7.15, 7.16, and, 7.17 displays the inability to meet the desired set point for the CPU.

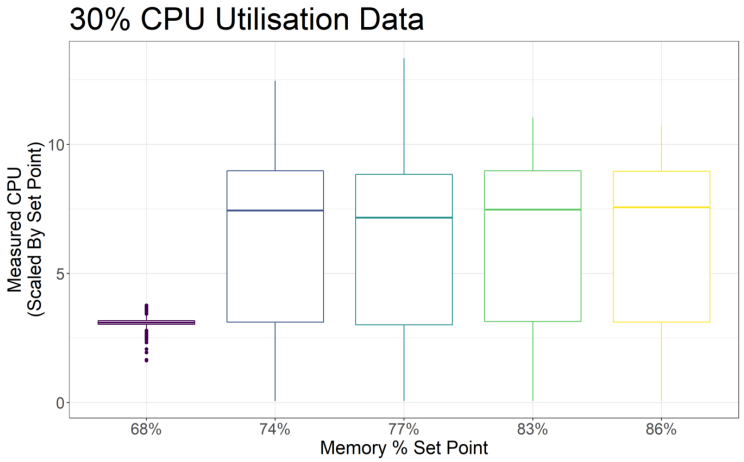

Figure 7.34: VirtualBox 30% PID Assisted LQR CPU Utilisation

Figure 7.34 shows decrease in both reducing variation and meeting the set point when compared to the same 30% CPU utilisation in Figure 7.14, where the upper and lower quartiles have gone from a range of 2.5 and 3.5 to around 2.5 and 8. The increased variation is likely due to the already increased variation in the VirtualBox system shown in Figure 4.2, and the introduction of a PID control miss reads the spike causing more variation in the system than the Static LQR controller. As mentioned in the Static LQR controller some of this variation could be introduced due to the same model as Narwhal being used, there are also differences that should be taken into account in the A matrix. Because VirtualBox is a different system the stable time used for Narwhal will also be different and give rise to the LQR controller adapting and making changes too early in the control cycle.

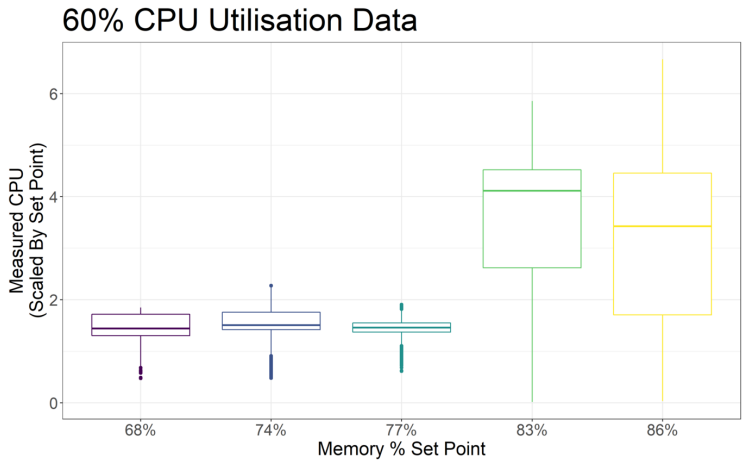

Figure 7.35: VirtualBox 60% PID Assisted LQR CPU Utilisation

Figures 7.35 and 7.36 show that controlling lower memory set points is easier than controlling the higher ones, for 68%, 74%, and 77% show the best when looking at meeting the set point and the variation in the system, however, Figure 7.36 shows significantly more outliers for 68%, 74%, and 77% memory set points. The cause of these outliers are likely due to spikes in the CPU utilisation of VirtualBox, for the PID Assisted LQR control 150% CPU utilisation 7.36 displays much better control in the system than any of the other Figures. For the PID Assisted LQR control the higher CPU utilisations are showing to be the better controlled than lower CPU utilisations which continues to follow the trend shown in the CPU utilisation for Narwhal (Figures 7.26 to 7.29) which is the opposite to the Static LQR control where more control is applied to lower CPU values.

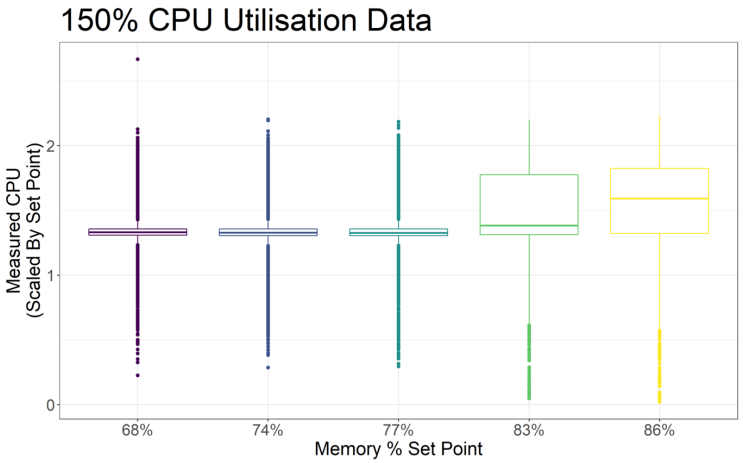

Figure 7.36: VirtualBox 150% PID Assisted LQR CPU Utilisation

The cause of PID Assisted control having better control at higher CPU utilisation is due to the way that the LQR controller decided to increase the parameters shown in Tables 7.4 and 7.5 for 30% and 210% CPU and 86% Memory. The Tables 7.4 and 7.5 show us how the LQR portion of the controller decided to change the parameters, and looking at Tables 7.4 and 7.5 we can see that the same values were used for Buttons, Breadth, and Depth for both 30% and 210% CPU utilisation. Values for Buttons, Breadth, and Depth being the same leaves just Hashes and Sleeps to control the system and from Tables 7.4 and 7.5 we can see that Sleeps was used for 30% and Hashes was used for 210% as expected. However, by utilising the same high values for Buttons, Breadth, and Depth which cause increases in the CPU it makes sense that the system would be easier to control at higher CPU set points than lower CPU set points. In Contrast, when looking at Tables 7.1 and 7.2 for the Static LQR controller we can see that two different Breadth values are utilised this slight change in the way the control parameters are utilised is the reason that the Static controller can control lower CPU set points. GUIGC is unique in that the parameters are separated into two groups Hash and Sleep allow better control at lower CPU set points whereas Buttons, Breadth, and Depth are much better and required to control the CPU at higher values.

Control Parameter	Average	Median	Standard Deviation
Hash	1	1	0
Sleep	1440.95	1274	852.54
Buttons	1	1	1
Depth	5	5	0
Breadth	9.83	10	1.22

Table 7.4: 30% CPU and 86% Memory Utilisation Control Parameter Statistics

Control Parameter	Average	Median	Standard Deviation
Hash	123.01	1	303.94
Sleep	1.01	1	0.13
Buttons	1	1	0
Depth	5.0	5	0.06
Breadth	9.98	10	0.29

Table 7.5: 210% CPU and 86% Memory Utilisation Control Parameter Statistics

7.4 PID Assisted and Static LQR

Part of this thesis is analysing the controllability of stress testing cross platform, and whether by implementing a PID Assisted LQR controller provides any benefit over the Static LQR controller in achieving all three of the goals in this thesis (set point accuracy, reduced variation, and cross platform controllability). Looking at Figures 7.39 and 7.40 we can see that when it comes to cross platform controllability VirtualBox lacked the any of the control that Narwhal got from utilising either the Static LQR controller or the PID Assisted LQR controller. The same model was utilised for both VirtualBox and Narwhal which was tested and build using the Narwhal machine, hence it is no surprise that Narwhal gained the better performance from the LQR controller.

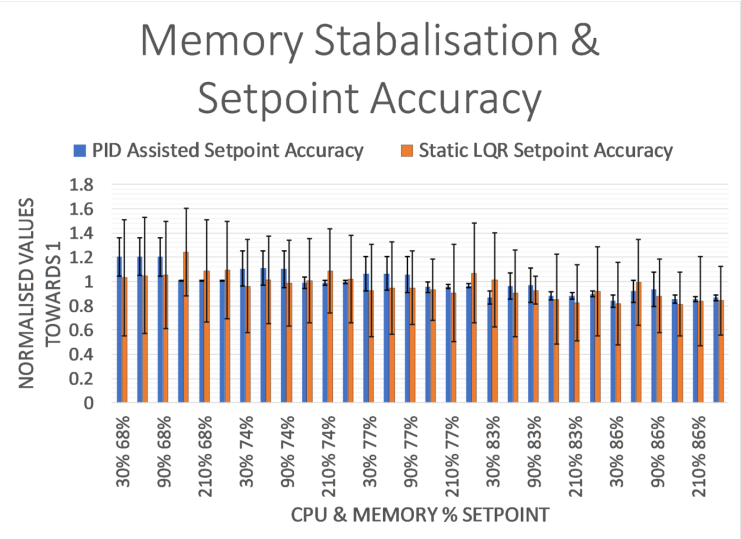

Figure 7.37: Narwhal Stabilisation and Variation of Memory Utilisation

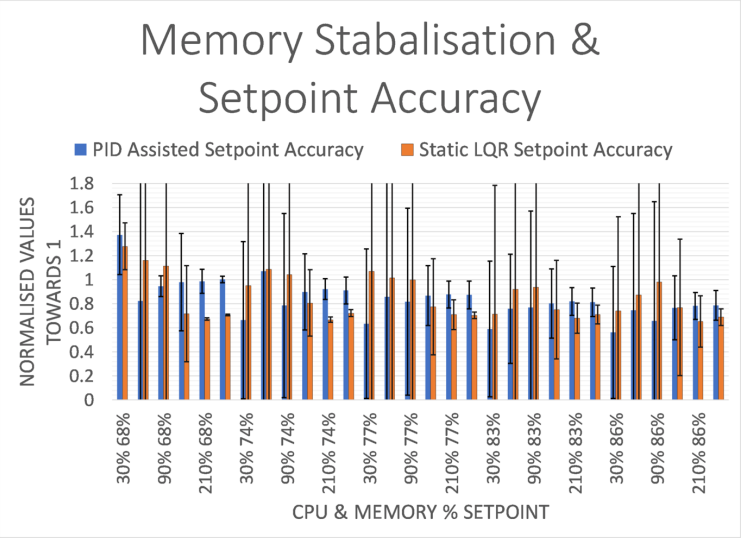

Figure 7.38: VirtualBox Stabilisation and Variation of Memory Utilisation

However, just because Narwhal benefited more from the utilised model does not mean that VirtualBox displayed no control it just means that better performance could be obtained from a more machine specific model. We can see from Figure 7.38 that VirtualBox show good control in memory for some set points with PID Assisted control displaying improvements in the higher memory set points than the lower ones where from 74% memory there are improvements in set point accuracy and decreased variation with these two

goals only getting better as the memory set point increases. The Static LQR controller for Figure 7.38 does not seem to benefit memory all that much with only minimal improvements shown for 30% CPU and 68% memory, and smaller variation for 210% and 310% CPU on 77% memory utilisation. CPU utilisation for VirtualBox shown in Figure 7.40 suffers under PID Assisted LQR control with both higher variation and the same as or worse that Static LQR in set point accuracy, except for the CPU set points higher than 210%. CPU set points above 210% show a better set point accuracy than the Static LQR controller but at the cost of a higher variation for all the tested set points. These Figures 7.38 and 7.40 lead us to conclude that a PID assisted LQR controller when the Narwhal LQR model is utilised only shows significant benefits when utilised for memory likely due to the ability to react better to the unpredictable nature or memory. However, CPU utilisation tends to suffer when both Static and PID Assisted LQR control is used indicating that in order to reliably control CPU an accurate model needed to be used for the system it is running on.

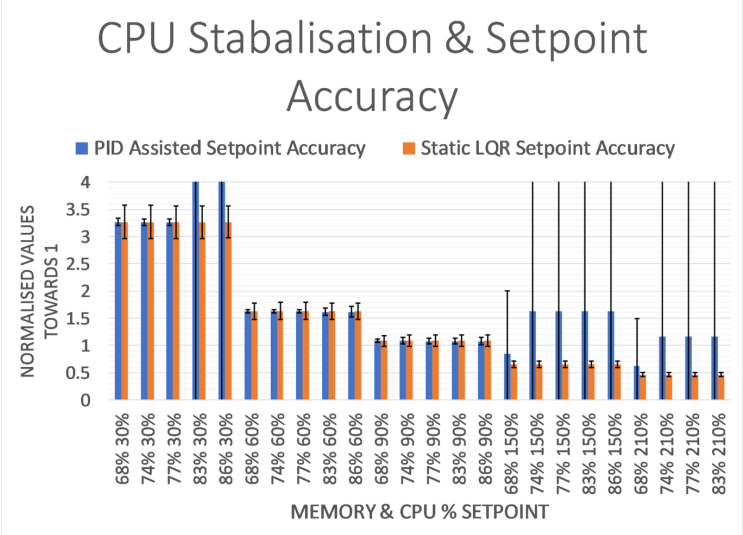

Figure 7.39: Narwhal Stabilisation and Variation of CPU Utilisation

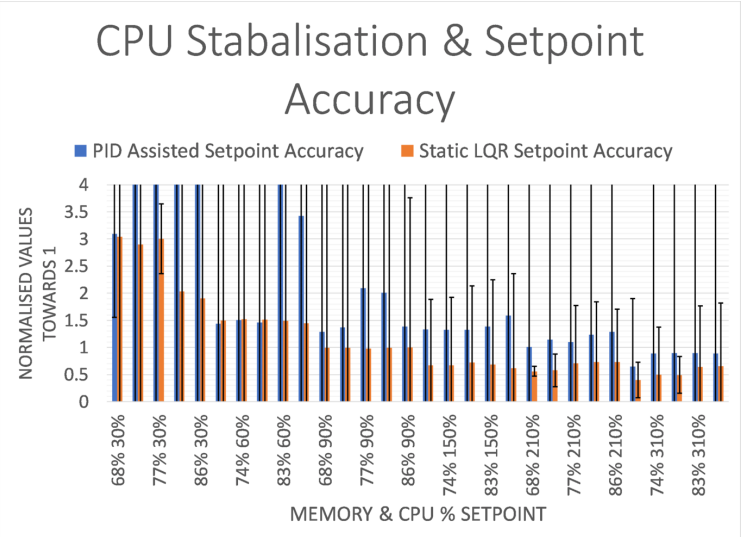

Figure 7.40: VirtualBox Stabilisation and Variation or CPU Utilisation

Narwhal for memory utilisation (Figure 7.37) benefits greatly from the utilisation of a PID Assisted LQR controller with significant drops in variation for the majority of the tested cases and for CPU set points 150% and above there is also improved set point accuracy for the memory. Static LQR controller displays better set point accuracy for CPU set points lower than 150% but across majority of the tested cases displays a much higher variation in the memory utilised. CPU on the other had (shown in Figure 7.39) does show improvements for PID Assisted LQR for CPU set points lower than 150% with a similar set point accuracy and slightly lower variation than the Static LQR controller. Except for two cases 83% and 86% memory utilisation in the 30% CPU range where both variation and set point accuracy are drastically worse than the Static LQR controller for some unknown reason that requires further research. CPU set points at and greater than 150% display significantly worse results under the PID Assisted LQR controller, which is due to the PID controller attempting to push the LQR set point to high causing far greater variation and worse set point accuracy.

Chapter 8

Conclusion

System testing with zero control showed that VirtualBox gave significantly more variation in the running system than narwhal for both CPU and memory with the zero control for narwhal remaining quite consistent (Figures 4.1, 4.3, 4.2, and 4.4). It is therefore not surprising to find that during the testing of the two system under PID and LQR control presented with more variation in the system shown in Tables 5.2 and 5.3, Figures 7.37, 7.38, 7.39, and 7.40 as hypothesised in Section 4.2.6.3.

8.0.1 PID Control

PID control for both Narwhal and VirtualBox displayed control of both the variation in the system and the set point accuracy. The low variation and good set point accuracy shows that when controlling a single variable with linear and easy to scale parameters PID controllers can out preform LQR on both implementation and control as no model of the system is required to run a PID controller. PID controllers also under the same conditions can provide the best cross platform implementation for a controller that will just run with little configuration when moving to another platform Figures 5.4 and 5.5. However, PID controllers lack the ability to control CPU set point 120% due to the inability to make use of non-linear control parameters Buttons, Breadth, and Depth making it hard to implement PID control for non-linear

control parameters leaving out a whole set of parameters that provides control for the higher CPU set points. This section leads into a hypothesised point laid out in Section 4.2.6.2, where it was hypothesised that as the number of parameters increases the PID controller will start displaying less control when trying to control more than a few parameters at one time.

8.1 LQR Control

The implementation of LQR controllers are easy and simple to implement ($\dot{x} = Ax + B(-K(r - x))$), however, the challenging section of an LQR controller is the ability to implement an adequate linear model that will accurately show the desired system. The model is what the LQR controller relies on when making decisions in how to control the system so designing an accurate model is important. Software engineering is a field where there is a lack of good equations that can represent how systems behave and specifically how the chosen parameters are going to affect the system. In engineering these equations are well tested such as Newton's laws of motion, making the challenge being to put these equations together to represent the desired system. For software engineering to find the patters in the control parameters utilised this thesis found that machine learning provided a good way to model the system for an LQR controller.

8.1.1 Model Analysis

By using linear regression this thesis found an accurate set of equations that could be produced and utilised to predict how the GUIGC program was going to behave under different combination of parameters. However, the downside to LQR controllers is that it does not provide good cross platform performance due to the heavy reliance on the model produced by linear regression, causing the need for a model to be rebuilt whenever the program is moved to a different system. This reliance on a model makes utilising an LQR controller difficult

in software engineering especially since this PID controllers do not suffer from the same cross platform model dependency. Finally, the implementation of a PID Assisted LQR controller provides a benefit to control states such as memory that have a hard to predict and model performance. However, control states such as CPU that can be well predicted and are less unpredictable than memory do not benefit from the added PID controller in the LQR system.

8.1.2 Performance Impact of Memory Manages Language

In cases where the model was designed for the running system the issue of memory being harder to control as hypothesised in Section 4.2.6.4 holds true as the CPU of the system displayed far better control than memory Figures 7.37 and 7.39. However, when moving to a system that the model was not designed for this hypothesis does not hold true as there is far more variation in the CPU than memory Figures 7.38 and 7.40.

8.2 Limitations

This thesis faces many limitations in the range of tests run and the model utilised, the major of which is the lack of a complete model of the system and the range of benchmarks that this thesis has tested. GUIGC is the only benchmark that has been tested in this thesis meaning that the effectiveness of controlling resource utilisation across other benchmarks and programs still needed to be determined. The ideas behind the testing run on GUIGC should be applicable to other programs but the extent of how well they preform is currently unknown and needs further testing. LQR controllers ability to successfully control a system depends on on the accuracy of the model, however, for brevity a major part of the system model was left out of the A matrix. The A matrix modelled the time to rise or rate of change for both the CPU and memory, unfortunately what was left out of this model

was the interactions between memory and CPU which we know exists. In this thesis a single model was used to test both Narwhal and VirtualBox using the same model for both systems has the potential to lead to some inaccuracies in VirtualBox. A singular model for both Narwhal and VirtualBox prevents VirtualBox being controlled optimally as there are many aspects that could have changed such as the control parameters effect on the system or the A matrix rise time. LQR also utilises a static model meaning that the model of the system does not change overtime or with the system, this static model can lead to inaccuracies in the system as changes in the GUIGC benchmarking program later on could introduce new unpredictability that are not present during testing of the model.

8.3 Future Work

Much of the research in self adaptive systems and PID/LQR controllers is relatively new and still evolving, this leaves many gaps in this thesis that are yet to be researched in order to provide better guarantees, faster controllers, and, better modelling techniques. Like Weyns [34] description of the six waves this thesis concludes that there are yet more waves left to be discovered, and machine learning could play a key role in the adaptive modelling of these systems. This thesis draws this conclusion based on the results produced through LQR control and the model produced through ML.

8.3.1 PID Controllers

PID controllers are beneficial in many ways one of which being that no model of a system is required unlike LQR controllers, hence, why the performance of a PID controller is similar across different systems. However, PID controller tuning does have the potential to give better performance when different gain values are used on different systems, but the effect of this is currently to the author's knowledge unknown and could benefit from further research. PID

controllers also struggle to adapt when controlling two or more conflicting states, however, the little model approach could be beneficial in continually changing systems where building complex models for a continually changing system is unrealistic.

8.3.2 LQR Controllers

When moving in to digital system based adaptation LQR controller optimisation still have much research needed in order to find the best ways to optimise the system as some cost based models of engineering may not apply when utilised for digital system. Sleeps and Hashes in this thesis are good examples of an abstract cost based approach in digital systems as there are no real cost based approaches that can be applied for these control parameters. In LQR controllers the design of the model decides how well the controllers are going to preform when applied to the system, also LQR controllers remain very static and any changes to the current system could require the rebuilding of a whole new model which is not cheap in time to design and build. Future work and expansion into self adaptive models, continues machine learning could help to provide better predictions for the adaptation of the control parameters, however this does not help with the building of the A matrix. One of the limiting factors of this thesis is the design of the A matrix in that the thesis had no way of calculating the effect of the states with each other, hence, research into ways to calculate these interactions in a more complex manner would benefit the self adaptive behaviour of the LQR controller.

Bibliography

[1] Stephen M. Blackburn, Perry Cheng, and Kathryn S. McKinley. Myths and realities. In *Proceedings of the joint international conference on Measurement and modeling of computer systems - SIGMETRICS 2004/PERFORMANCE 2004*. ACM Press, 2004.

[2] Tim Brecht, Eshrat Arjomandi, Chang Li, and Hang Pham. Controlling garbage collection and heap growth to reduce the execution time of Java applications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 28(5):908–941, sep 2006.

[3] Yuriy Brun, Giovanna Di Marzo Serugendo, Cristina Gacek, Holger Giese, Holger Kienle, Marin Litoiu, Hausi Müller, Mauro Pezzè, and Mary Shaw. Engineering Self-Adaptive Systems through Feedback Loops. In *Software Engineering for Self-Adaptive Systems*, pages 48–70. Springer Berlin Heidelberg, 2009.

[4] Andriy Burkov. *The hundred-page machine learning book*. 2019. OCLC: 1089445188.

[5] Vineet Chaoji, Rajeev Rastogi, and Gourav Roy. Machine learning in the real world. *Proceedings of the VLDB Endowment*, 9(13):1597–1600, sep 2016.

[6] G. Chen, R. Shetty, M. Kandemir, N. Vijaykrishnan, M.J. Irwin, and M. Wolczko. Tuning garbage collection in an embedded Java environment. In *Proceedings Eighth International Symposium on High Performance Computer Architecture*. IEEE Computer. Soc.

[7] Maria De-Arteaga, William Herlands, Daniel B. Neill, and Artur Dubrawski. Machine Learning for the Developing World. *ACM Transactions on Management Information Systems*, 9(2):1–14, aug 2018.

[8] Antonio Filieri, Martina Maggio, Konstantinos Angelopoulos, Nicolas D'Ippolito, Ilias Gerostathopoulos, Andreas Berndt Hempel, Henry Hoffmann, Pooyan Jamshidi, Evangelia Kalyvianaki, Cristian Klein, Filip Krikava, Sasa Misailovic, Alessandro Vittorio Papadopoulos, Suprio Ray, Amir M. Sharifloo, Stepan Shevtsov, Mateusz Ujma, and Thomas Vogel. Software Engineering Meets Control Theory. In *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE, may 2015.

[9] Anshul Gandhi, Parijat Dube, Alexei Karve, Andrzej Piotr Kochut, and Li Zhang. Providing Performance Guarantees for Cloud-deployed Applications. *IEEE Transactions on Cloud Computing*, pages 1–1, 2019.

[10] IBM. An architectural blueprint for autonomic computing, 2005.

[11] Connor Imes, Huazhe Zhang, Kevin Zhao, and Henry Hoffmann. CoPPer: Soft Real-Time Application Performance Using Hardware Power Capping. In *2019 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE, jun 2019.

[12] Kugathasan Janarthanan, P.R.L.C Peramune, A.T Ranaweera, Theviyanthan Krishnamohan, Lakmal Rupasinghe, Kalpa Kalhara Sampath, and Chethana Liyanapathirana. Policies Based Container Migration Using Cross-Cloud Management Platform. In *2018 IEEE International Conference on Information and Automation for Sustainability (ICIAfS)*. IEEE, dec 2018.

[13] Philipp K. Janert. *Feedback control for computer systems*. O'Reilly, first edition edition. OCLC: ocn824605038.

[14] Sanath Jayasena, Milinda Fernando, Tharindu Rusira, Chalitha Perera, and Chamara Philips. Auto-Tuning the Java Virtual Machine. In *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*. IEEE, may 2015.

[15] Evangelia Kalyvianaki, Themistoklis Charalambous, and Steven Hand. Self-adaptive and self-configured CPU resource provisioning for virtualized servers using Kalman filters. In *Proceedings of the 6th international conference on Autonomic computing - ICAC '09*. ACM Press, 2009.

[16] Evangelia Kalyvianaki, Themistoklis Charalambous, and Steven Hand. Adaptive Resource Provisioning for Virtualized Servers Using Kalman Filters. *ACM Transactions on Autonomous and Adaptive Systems*, 9(2):1–35, jul 2014.

[17] Rouven Krebs, Alexander Wert, and Samuel Kounev. Multi-tenancy Performance Benchmark for Web Application Platforms. In *Lecture Notes in Computer Science*, pages 424–438. Springer Berlin Heidelberg, 2013.

[18] Mayuresh Kunjir. Guided Bayesian Optimization to AutoTune Memory-Based Analytics. In *2019 IEEE 35th International Conference on Data Engineering Workshops (ICDEW)*. IEEE, apr 2019.

[19] Qiang Li, Ranyang Li, Kaifan Ji, and Wei Dai. Kalman Filter and Its Application. In *2015 8th International Conference on Intelligent Networks and Intelligent Systems (ICINIS)*. IEEE, nov 2015.

[20] Harry McCarthy, Abigail Koay, Michael Dawson, Kenneth B Kent, and Panos Patros. Benchmarking garbage collection delays for resource-restricted graphical user interfaces. *Software: Practice and Experience (Under Review)*, 2020.

[21] Panagiotis Patros, Michael Dawson, and Kenneth B. Kent. Garbage Collections Delays on Java GUIs. In *2018 Eleventh International*

Conference on Mobile Computing and Ubiquitous Network (ICMU). IEEE, oct 2018.

[22] Panagiotis Patros, Michael Dawson, and Kenneth B. Kent. Why is garbage collection causing my service level objectives to fail? *International Journal of Cloud Computing*, 7(3/4):282, 2018.

[23] Panagiotis Patros, Kenneth B Kent, and Michael Dawson. Investigating the effect of garbage collection on service level objectives of clouds. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 633–634. IEEE, 2017.

[24] Panagiotis Patros, Kenneth B Kent, and Michael Dawson. Slo request modeling, reordering and scaling. In *Proceedings of the 27th Annual International Conference on Computer Science and Software Engineering*, pages 180–191, 2017.

[25] Panagiotis Patros, Kenneth B Kent, and Michael Dawson. Mitigating garbage collection interference on containerized clouds. In *2018 IEEE 12th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages 168–173. IEEE, 2018.

[26] Panagiotis Panos Patros. *Modeling and Improving the Performance of Cloud Systems*. PhD thesis, UNIVERSITY OF NEW BRUNSWICK, 2018.

[27] Vladimir Podolskiy, Michael Mayo, Abigail Koey, Michael Gerndt, and Panos Patros. Maintaining slos of cloud-native applications via self-adaptive resource sharing. In *2019 IEEE 13th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages 72–81. IEEE, 2019.

[28] Lal Bahadur Prasad, Barjeev Tyagi, and Hari Om Gupta. Optimal control of nonlinear inverted pendulum dynamical system with disturbance input

using PID controller &amp LQR. In *2011 IEEE International Conference on Control System Computing and Engineering*. IEEE, nov 2011.

[29] P. Pufek, H. Grgic, and B. Mihaljevic. Analysis of Garbage Collection Algorithms and Memory Management in Java. In *2019 42nd International Convention on Information and Communication Technology Electronics and Microelectronics (MIPRO)*. IEEE, may 2019.

[30] Ritesh Raj, Subrat Kumar Swain, and Sudhansu Kumar Mishra. Optimal Control for Magnetic Levitation System Using H-J-B Equation Based LQR. In *2018 2nd International Conference on Power Energy and Environment: Towards Smart Technology (ICEPE)*. IEEE, jun 2018.

[31] Gilberto Reynoso-Meza, Leandro dos Santos Coelho, and Roberto Z. Freite. Efficient Sampling of PI Controllers in Evolutionary Multiobjective Optimization. In *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference - GECCO '15*. ACM Press, 2015.

[32] Laurent Simon. Optimum Control and Design. In *Control of Biological and Drug-Delivery Systems for Chemical, Biomedical, and Pharmaceutical Engineering*, pages 347–360. John Wiley & Sons, Ltd, 2013.

[33] Felix Tarasenko, Vladimir Kozlov, Violetta Volkova, and Arina Kudriavtceva. On future development of the control theory of automated complexes in the information-communication technologies implementation. In *Proceedings of the XI International Scientific Conference Communicative Strategies of the Information Society*. ACM, oct 2019.

[34] Danny Weyns. Engineering Self-Adaptive Software Systems – An Organized Tour. In *2018 IEEE 3rd International Workshops on Foundations and Applications of Self∗Systems(FAS∗W).IEEE, sep*2018.

[35] Zhenglong Xiang, Daomin Ji, Heng Zhang, Hongrun Wu, and Yuanxiang Li. A simple PID-based strategy for particle swarm optimization algorithm. Information Sciences*, 502:558–574, oct 2019.*

[36] Jingfeng Xue, Changzhen Hu, Hongyu Ren, Rui Ma, and Jian Li. Memory errors prevention technology for C/C program based on probability. In 2011 7th International Conference on Natural Language Processing and Knowledge Engineering*. IEEE, nov 2011.*

[37] Kai Zhang, Jun Yao, Liming Zhang, Xia Yan, and Huanhuan Qian. Optimal Control for Reservoir Production Working System Using Gradient-Based Methods. In 2010 2nd International Workshop on Intelligent Systems and Applications*. IEEE, may 2010.*

[38] Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang, Sharad Singhal, Arif Merchant, Pradeep Padala, and Kang Shin. What does control theory bring to systems research? ACM SIGOPS Operating Systems Review*, 43(1):62, jan 2009.*