Autonomous Excavation Using Reinforcement Learning with Proximal Policy Optimization

Master's thesis in Complex Adaptive Systems

MÅRTEN SANDERÖD, OSKAR TRYGGVASON

DEPARTMENT OF MECHANICS & MARITIME SCIENCES

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Master's thesis 2025

Autonomous Excavation Using Reinforcement Learning with Proximal Policy Optimization

MÅRTEN SANDERÖD
OSKAR TRYGGVASON

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mechanics & Maritime Sciences
Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
Gothenburg, Sweden 2025

Autonomous Excavation Using Reinforcement Learning with
Proximal Policy Optimization
MÅRTEN SANDERÖD, OSKAR TRYGGVASON

Supervisor 1: Marcus Carlson, CPAC Systems AB
Supervisor 2: Malte Landgren, CPAC Systems AB
Examiner: Peter Forsberg, Department of Mechanics and Maritime Sciences

Master's Thesis 2025
Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Visualization of the excavator model Volvo EC550E which is used in the project.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2025

Autonomous Excavation Using Reinforcement Learning with
Proximal Policy Optimization
MÅRTEN SANDERÖD, OSKAR TRYGGVASON
Department of Mechanics and Maritime Sciences
Chalmers University of Technology

Abstract

This thesis presents a reinforcement learning based approach for grading, applied in the Volvo excavator EC550E. The work was based around a simulation model of the excavator which facilitated easy training of the algorithm. A hydraulic controller was trained using proximal policy optimization. Together with the hydraulic controller, a PID was implemented as a positional controller for a complete system capable of performing grading tasks.
Training was conducted by testing different reward functions and parameter choices to improve policy performance. The results showcases hyperparameter evaluation, velocity tracking accuracy for the hydraulic controller as well as grading accuracy of the complete system. The implemented solution had an accuracy of $\pm$ 4 cm during grading. However, the hydraulic controller was not able to consistently follow the target velocities in the cylinders, particularly for the bucket. In future works the hydraulic controller needs to be retrained for better precision before being deployed in a real machine. This thesis shows the potential and possibility of replacing traditional control policies with an machine-learning driven approach.

Keywords: autonomous excavation, excavator, hydraulics, IMVT, machine learning, proximal policy optimization, reinforcement learning.

Acknowledgements

First and foremost we would like to thank our examiner, Peter Forsberg, his insights and support made that pointed us in the right direction and made it so we could push the boundaries for the work. A special thanks to our supervisors, Marcus Carlsson for providing us with knowledge about excavators and helpful insights when designing the system, and Malte Landgren for his knowledge about machine learning that were valuable for creating a good model. We would also like to thank the entire Active Control team for their interest and valuable insights to the project. We are also grateful for CPAC Systems AB, that provided a workspace, equipment and coffee which was invaluable for the project work.

Mårten Sanderöd & Oskar Tryggvason, Gothenburg, June 2025

Thesis advisors: Marcus Carlson & Malte Landgren, CPAC Systems AB
Thesis examiner: Peter Forsberg, Mechanics and Maritime Sciences

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

AI	Artificial intelligence
CPAC	CPAC systems AB
DOF	Degrees of freedom
DLL	Dynamic linking library
EHPV	Electro-hydraulic poppet valve
FFNN	Feed forward neural network
GAE	Generalized advantage estimation
GNSS	Global navigation satellite system
IMVT	Idependent meetering valve technology
IMU	Inertial measurement unit
LSTM	Long short term memory
ML	Machine learning
MSE	Mean square error
PID	Proportional integral derivative
PPO	Proximal policy optimization
RL	Reinforcement learning
RNN	Recurrent neural network
SB3	Stable baselines3
TRPO	Trust region policy optimization
VAC	Volvo active control

Contents

List of Acronyms **ix**

List of Figures **xv**

List of Tables **xix**

1 Introduction **1**
1.1 Background . . . 1
1.1.1 The excavator . . . 3
1.2 Related works . . . 4
1.3 Purpose and goals . . . 6
1.4 Limitations and current platform . . . 7

2 Theory **9**
2.1 Hydraulics . . . 9
2.1.1 Excavator hydraulics . . . 9
2.1.2 Electro-hydraulic poppet valves (EHPV) . . . 10
2.1.3 EC550E system . . . 10
2.1.3.1 Extension of the cylinder . . . 12
2.1.3.2 Retraction of the cylinder . . . 12
2.1.3.3 Regenerative movement . . . 12
2.1.3.4 Remarks on safety and control . . . 13
2.2 PID-controller . . . 13
2.3 Recurrent neural networks (RNNs) . . . 13
2.3.1 Long short-term memory (LSTM) . . . 14
2.4 Reinforcement learning (RL) . . . 15
2.4.1 Branches and applications . . . 16
2.4.1.1 Model-based methods . . . 16
2.4.1.2 Model-free methods . . . 17
2.4.2 Policy gradient methods . . . 17
2.4.3 Proximal policy optimization (PPO) . . . 18
2.4.3.1 Clipped surrogate objective . . . 19
2.4.4 Generalized advantage estimation (GAE) . . . 19

3 Method **21**
3.1 Mechanical model . . . 21
3.1.1 Coordinate systems and naming conventions . . . 22

3.1.2 Inverse kinematics . . . 23
3.2 Positional controller . . . 24
3.3 Simulation model . . . 26
3.3.1 Modifications to the simulation model . . . 26
3.3.2 Verification of provided model . . . 27
3.3.2.1 Verification of python bridge . . . 28
3.3.3 Training environment . . . 28
3.3.3.1 Observations and actions . . . 30
3.3.3.2 Episode initialization . . . 31
3.3.3.3 Command sampling . . . 31
3.4 Hydraulic control . . . 32
3.4.1 Actor and critic . . . 32
3.4.2 Algorithm optimization . . . 33
3.4.2.1 GAE . . . 33
3.4.2.2 Minibatching . . . 33
3.4.2.3 Entropy regularization . . . 33
3.4.2.4 Learning rate decay . . . 34
3.4.2.5 Gradient clipping . . . 34
3.4.3 Algorithm . . . 34
3.4.4 Reward function . . . 35
3.4.4.1 Velocity reward . . . 36
3.4.4.2 Direction reward . . . 36
3.4.4.3 Fuel and engine penalty . . . 37
3.4.4.4 Total reward . . . 37
3.5 Training the hydraulic controller . . . 37
3.5.1 Comparing parameters . . . 38
3.5.1.1 Episode length . . . 39
3.5.1.2 Batch size . . . 39
3.5.1.3 Error/reward scaling . . . 39
3.5.1.4 Policy network . . . 40
3.5.1.5 Fuel penalty . . . 41
3.5.2 System complexity . . . 41
3.5.2.1 Single actuator policy . . . 41
3.5.2.2 Combined actuator policy . . . 41
3.5.3 Final policy . . . 42
3.5.3.1 Varying target velocities . . . 42
3.6 The complete system . . . 42
3.7 Evaluation . . . 43
3.7.1 Hydraulic controller . . . 43
3.7.2 Grading performance . . . 43

4 Results **45**
4.1 Training parameters . . . 45
4.1.1 Episode length . . . 45
4.1.2 Batch size . . . 46
4.1.3 Reward scaling . . . 47

4.1.4 Policy network . . . 48
4.1.5 Fuel penalty . . . 49
4.2 Hydraulic control . . . 50
4.2.1 Velocity tracking . . . 51
4.2.2 Oscillatory behavior . . . 53
4.2.3 Final policy . . . 54
4.3 Grading performance . . . 55
4.3.1 Sloping trajectory . . . 55
4.3.2 Horizontal trajectory . . . 57

5 Discussion **61**
5.1 Training parameters . . . 61
5.2 Hydraulic control . . . 62
5.2.1 Separate cylinder policies . . . 62
5.2.2 Velocity tracking . . . 63
5.2.3 Oscillations . . . 64
5.3 Grading performance . . . 64
5.4 Simulation platform and time constraints . . . 65
5.5 Application on a real excavator . . . 66

6 Conclusion **67**
6.1 Future works . . . 67

Bibliography **69**

A Model comparison between Matlab and Python **I**
A.1 Cylinder displacement . . . II
A.2 IMU signals . . . III

B Excavator simulation inputs and outputs **V**
B.1 Inputs . . . V
B.2 Outputs . . . VI

C Grading evaluation results **XI**
C.1 Horizontal and slope towards . . . XI
C.2 Horizontal towards . . . XIII

List of Figures

1.1 The Volvo EC550E excavator which is investigated in this thesis. [1] . 3
1.2 Main structural and functional components of a hydraulic excavator. Adapted from [2]. 3

2.1 Simplified model of the hydraulic system used in EC550E. It shows how all three cylinders are connected to the pumps together with bypass valves. 11
2.2 Model of the hydraulic system for one cylinder used in the excavator using IMVT with two pumps. All valves are EHPVs. All cylinders share bypass valves. 11
2.3 Basic concept of RNN, with self connection through time where x denotes the input to the network, y the output of the network, h the hidden state, v the self-connection, U and W are weights between the layers. 14
2.4 Structure of the hidden layer in an LSTM network at timestep t. With cell state C, hidden state h, sigmoid (σ) and tanh activation function. 15
2.5 Visualization of RL system. The agent takes actions based on the observations and rewards calculated from the environment. 16

3.1 Overview of the sub-problems and the information required by each controller. 21
3.2 Excavator model with points of interest labeled as well as important coordinate systems highlighted. Reproduced with permission. Source: [2] 22
3.3 Flowchart of the PID control loop for excavator motion. The controller receives reference pose and velocity inputs to generate control actions. 25
3.4 Illustration of the data being shared between the Simulink model and Python environment with the most important inputs and outputs shown 27
3.5 Comparison of the three different actuators on a real machine compared to the Simulink model. 28
3.6 Architecture of the training environment, exposing a Gymnasium-compatible interface which internally interacts with the Simulink model. 29
3.7 Comparison of error type and scaling. 40
3.8 Flow of the system used to perform the task of grading. 43
3.9 Grading trajectories used for evaluation. 44

4.1 Rolling average of the combined velocity error for different episode lengths. . . . 46
4.2 Rolling average of the velocity tracking error per simulation step for different batch sizes. . . . 47
4.3 Rolling average of the cylinder velocity error per epoch for different batch sizes. . . . 47
4.4 Cylinder velocity error for different reward scalings and error calculations. . . . 48
4.5 Combined velocity tracking error during training for different network configurations. . . . 49
4.6 Combined velocity error for all cylinders with different reward components enabled 50
4.7 Fuel consumption during one evaluation episode for policies with different reward components enabled. . . . 50
4.8 Comparison of velocity errors for single-cylinder control, between the policy for all cylinder control and the separate cylinder control. . . . 51
4.9 Velocity tracking for the boom cylinder over an episode, for separate boom and combined policy. . . . 51
4.10 Velocity tracking for arm cylinder over an episode, for separate arm and combined policy. . . . 52
4.11 Velocity tracking for the bucket cylinder over an episode, for separate bucket and combined policy. . . . 52
4.12 Velocity tracking for all cylinders with a wanted velocity $\neq 0$ m/s for the arm cylinder. . . . 53
4.13 Cylinder velocity errors during training for the final policy (constant velocity) and varying velocity. . . . 54
4.14 Evaluation of velocity tracking performance on all cylinders between the final policy (constant velocity) and a varying velocity policy. . . . 55
4.15 Grading performance on a sloping trajectory, with the start of grading to the right. . . . 56
4.16 Grading error during evaluation of the sloping trajectory. . . . 56
4.17 Cylinder velocities during grading evaluation of the sloping trajectory. 57
4.18 Grading performance on a flat trajectory, with the start of grading to the right. . . . 58
4.19 Grading error during evaluation of the flat trajectory. . . . 58
4.20 Cylinder velocities during grading evaluation on the flat trajectory. . 59

A.1 Comparison between Matlab and Python cylinder displacement for different test cases. . . . II
A.2 Comparison between Matlab and Python IMU signals for different test cases. . . . III

C.1 Grading result on a sloping and horizontal trajectory where the linkage moves towards the excavator house. . . . XI
C.2 Grading error on a sloping and horizontal trajectory where the linkage moves towards the excavator house. . . . XII

C.3 Wanted and actual velocities in each cylinder for a sloping and horizontal trajectory where the linkage moves towards the excavator house. XII
C.4 Grading result on a horizontal trajectory where the linkage moves towards the excavator house. XIII
C.5 Grading error on a horizontal trajectory where the linkage moves towards the excavator house. XIV
C.6 Wanted and actual velocities in each cylinder for a horizontal trajectory where the linkage moves towards the excavator house. XIV

List of Tables

3.1 Observation signals used as input to the control policy. 30
3.2 List of all action signals controlled by the agent. 30
3.3 Valve layout for a single hydraulic cylinder. 31
3.4 Critic neural network Architecture. 33
3.5 Parameters used during training, with parameters to be evaluated marked. 38
3.6 Evaluated neural network architectures for the policy. 41

4.1 Initial evaluated parameters used during training 45
4.2 Training configurations with varied episode time and update parameters 45
4.3 Training configurations with varying reward function and scaling . . . 48
4.4 Training configurations with engine penalty and fuel penalty activated in separate configurations 49
4.5 Evaluated parameters used during the final training. 54

B.1 Excavator Control Inputs V
B.1 Excavator Control Inputs VI
B.2 Excavator Control Outputs VI
B.2 Excavator Control Outputs VII
B.2 Excavator Control Outputs VIII
B.2 Excavator Control Outputs IX
B.2 Excavator Control Outputs X

1
Introduction

This thesis was carried out in collaboration with CPAC Systems AB (CPAC), with the objective of investigating how machine learning (ML), specifically reinforcement learning (RL), can be used to improve the control of hydraulic systems in an excavator. Excavators are one of the most widely used heavy machinery at construction sites, as they are good at performing a number of different tasks, all from digging, offloading, grading and destruction. However, the increasing shortage of skilled operators, combined with safety concerns and the need for higher efficiency, has motivated the industry to explore autonomous solutions.
Autonomous excavation has become a promising research field, driven by advancements in robotics and RL connected to excavator control [3, 4, 5, 6, 7, 8]. These technologies offer the potential to reduce operating costs, improve site safety, and support operators by automating complex movements. Despite the progress, achieving the level of control and precision of a skilled human operator remains a major technical challenge, particularly due to the complexity of the hydraulic system.
The focus of this thesis is on both automating a specific excavation task, grading, as well as developing an intelligent control policy of the underlying hydraulic system, which is fundamental for enabling efficient and adaptable machine behavior. Instead of relying on traditional control strategies, which often struggle with nonlinearities and inter-dependencies of hydraulic cylinders, this work investigates a simulation-driven approach using RL. The aim is to develop a control policy that can handle dynamic operating conditions while optimizing both speed and accuracy of movement.

1.1 Background

Hydraulic control plays a central role in the operation of excavators, as it controls the movement of actuators and ultimately dictating the precision and responsiveness of the machine. However, controlling hydraulics effectively is challenging due to a range of physical phenomena such as the initial forces required to overcome static friction, nonlinear relationships between linkage geometry and actuator lengths, and the challenge of coordinating pump flows and valve settings to avoid bypass or instability. These complexities limit the effectiveness of traditional control systems in real-world applications.
One repetitive task where precise hydraulic control is needed is grading. Grading is an essential process in excavation and typically demands a high level of precision and smooth coordination across multiple actuators. It consists of leveling the

ground to create an even and stable surface for buildings or landscaping. The procedure requires a skilled operator and sometimes aids such as construction lasers or a traditional cord to help the operator.

In more advanced excavators, several systems are employed to assist the operator. For example, Volvo's dig assist system that utilizes a combination of sensors on the arm with global navigation satellite system (GNSS) units to determine position and orientation of the bucket [9]. This can help the grading process by displaying the current angle of the bucket on a screen to the operator and thus helping the operator to always keep the right angle. Another solution is the "Volvo Active Control" (VAC) [10] which provides multiple ways for the operator to operate the excavator in semi-autonomous movements. These movements includes grading movements which further helps the operators by autonomously controlling two actuators while the operator has to control the boom actuator. This semi-autonomous solution provides some help to the operator, but lacks the precision needed to be a truly competitive solution.

To achieve truly adaptable and precise motion, a shift toward learning-based control is needed. RL offers a promising alternative, as it enables the agent to learn control strategies directly from interaction with the system. Previous research has shown promising results in applying RL to robotic excavation and hydraulic control [5, 4, 7, 11, 12, 13]. These studies highlight the potential of RL in handling the nonlinear dynamics of hydraulic machinery, but also reveal gaps in terms of accuracy, robustness, and training stability.

This thesis contributes to that direction by designing and evaluating RL agents that learn to control the hydraulic system of an excavator from scratch. The control problem is approached in increasing levels of complexity, from single actuator scenarios to full multi-cylinder coordination under varying conditions to analyze how the agent generalizes and maintains control performance. The goal is to develop a proof of concept RL-based hydraulic controller which can be used in grading tasks and can match or exceed human-like control.

1.1.1 The excavator

Figure 1.1: The Volvo EC550E excavator which is investigated in this thesis. [1]

Excavators are among the most commonly utilized types of heavy machinery in construction, mining, and landscaping operations due to their versatility and powerful digging capabilities. A standard hydraulic excavator comprises a cab, commonly referred to as the house, mounted on a rotating platform, which allows for 360-degree movement. Attached to the house is a mechanical arm, composed of several interlinked components including the boom, arm, yoke, c-link, and an end-effector such as a bucket, grapple, or hydraulic breaker, depending on the task at hand. In this thesis, a bucket is used as the end effector. An illustration can be seen in Figure 1.2.

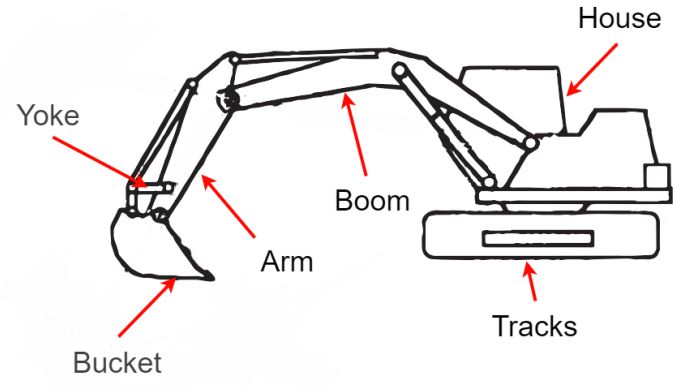


Figure 1.2: Main structural and functional components of a hydraulic excavator. Adapted from [2].

The mechanical linkage composed of the boom, arm and bucket is actuated by a series of hydraulic cylinders. While the exact number and configuration of actuators may vary between models, the primary ones typically include actuators for the boom, arm, and bucket. These actuators enable the complex motions required for digging, lifting, and grading tasks.

Excavator operation is generally performed via a pair of joysticks within the operator's cabin. It requires substantial experience and training to master the machine's controls. In particular, performing precise or complex operations, such as leveling a surface at an incline, demands a highly skilled operator with several hundred hours of practical experience.
The focus of this thesis is on the Volvo excavator EC550E, shown in Figure 1.1. The excavator has an operating weight around 55 metric tonnes and reach around 12 meters [1]. The Volvo EC550E features a electro-hydraulic system branded as independent metering valve technology (IMVT). This is an electro-hydraulic "closed-center" architecture which uses poppet valves with electronic actuation instead of conventional spool valves. According to Volvo, this IMVT system "produces up to a 25% improvement in fuel efficiency" by supplying exactly the hydraulic flow needed and stopping flow when no actuator demand exists [1, 14]. In effect, IMVT replaces the hydraulic system of a traditional main control valve with multiple independently controlled poppet valves, eliminating the constant neutral flow of an open-center circuit and reducing throttling losses.
Combined with the IMVT technology, the Volvo EC550E also contains a new valve, referred to as the regeneration valve, which connects the two chambers in the cylinder. This enables the hydraulic system to preserve the fluid already present in the system while still performing certain operations. This contributes to a better fuel efficiency as the engine does not need to drive the hydraulic pumps as much.

1.2 Related works

There has been extensive research into the automation of excavators. Previous research at CPAC investigates the possibilities of achieving perfect grading at both angles and horizontally, utilizing control theory [2]. One large problem that had to be tackled was the nonlinear behavior and long dead-times in the model as well as operator input during the execution of the algorithm. The excavator could be operated in two different ways, either by using USB-joysticks or by hydraulic joysticks directly connected to the pilot valves. The USB-joysticks introduced dead-time while the hydraulic joysticks provided no way to measure the output from the movements. Using this implementation, better precision was achieved than that of a skilled operator, proving that autonomous solutions could be better than operators. However, speed had to be sacrificed over precision, making the autonomous solution slower than a manually operated excavator.
In conventional excavators, a hydraulic system using an open center solution is usually used. An open center hydraulic system supplies a constant flow of hydraulic fluid which means that the pressure can vary. Instead, closed center systems can be used where the pump flow is cut of, ensuring that the pressure inside the system remains constant. Research into the different benefits between the two systems has shown that closed loop systems are more energy efficient compared to open center solutions [15, 16]. As the pumps, valves and pipes in the hydraulic system can account for around 50% of the energy lost in these systems [17], better solutions are needed to reduce the energy loss. Studies has shown that by using electro-hydraulic poppet valves (EHPV), the fuel consumption can be reduced by 25% compared

to conventional systems [18] and thus making EHPVs a great choice for excavator hydraulics.

Currently, Volvo construction equipment implements the VAC system [10] which provides solutions for most problems mentioned in this thesis. However, the autonomous solutions used in VAC are based on control theory methods with multiple nested controllers. Despite its sophistication, the system encounters limitations consistent with those outlined in the problem statement, particularly in its reliance on conventional control methods to coordinate multiple actuators effectively.

Recent research has shifted towards integrating machine learning techniques, such as RL, and advanced computer vision to enhance excavator automation. One example is the work by Yoo et al. [5], which investigates the application of reinforcement learning together with image processing to create an adaptive and intelligent control system. Their approach involves designing a reward-based mechanism for individual joints rather than treating the system as a whole. This enables more control and optimization of the excavator's movements. Additionally, the research expands the traditional two-dimensional (2D) operational model to a three-dimensional (3D) one by incorporating cabin swing rotation, effectively increasing the bucket's workspace.

Their solution also integrates environmental awareness through an independent camera system that surveys the construction site. This allows the excavator to dynamically respond to changes in its surroundings, enhancing adaptability and reducing the need for pre-programmed interventions. While this research aligns closely with the objectives of the proposed thesis, it falls short in addressing the complexity of certain movements, such as the straight-line bucket motion problem described earlier. This challenge remains a critical barrier to achieving fully autonomous excavator operation in complicated scenarios.

Another paper investigated the application of reinforcement learning algorithms for excavators, trying to achieve perfect horizontal grading by controlling the joystick movements [4]. The authors identified TRPO (trust region policy optimization) as the best method [19], outclassifying the other algorithms by far (Hill climb, cross entropy method and random). Together with the TRPO, they applied a eight-pole infinite impulse response filter to smooth out the excessive inputs to the controllers, mimicking an operators behavior while still performing better. The model was validated using the simulation platform Dynasty and OpenAI Gym framework, showing that dynamic models can be simulated and exposed to machine learning in Python.

In the work of Egli, Pascal and Hutter [7] a trajectory tracking controller for a excavator arm with highly nonlinear hydraulics was developed. They used the RL method proximal policy optimization (PPO) instead of TRPO because of faster training with similar performance. By using observations of the pose and velocities of the excavator along with engine and valve readings, the PPO model was used to directly control the valves of the excavator instead of the operator controls to overcome nonlinearities exhibited when controlling on a joint level. By also sending a velocity command to the PPO the model could execute tasks at a requested speed. The system developed showed great accuracy at both grading and trajectory tracking in both simulation and deployment.

Another project investigated the possibility to utilize two powerful tools for reinforcement learning: Python and Matlab's Simulink [20]. Simulink is a powerful tool

which enables developers to create complex simulations quite easily and visualize them. However, applying complex machine learning to the models leaves much to ask for. Python has been one of the major programming languages for developing machine learning methods and has many different frameworks, making it easier for the developer. To get the best from both worlds, the authors created a bridge between Python and Simulink utilizing Simulink's Codegen abilities, compiling the models to C code which then can be imported into Python. This enabled the authors to utilize the OpenAI Gym framework for seamless simulation and evaluation of reinforcement learning networks.
The complexity and non-linearities of electro-hydraulics can make it difficult to implement an accurate control system. In the research of Filo [11] the current field of AI/ML methods for hydraulic control is investigated and presented. Filo claims that AI based methods can improve both operational efficiency and energy consumption of the hydraulic system. The main categories used in contemporary research are neural networks, evolutionary algorithms and fuzzy logic. The broadest are artificial neural networks, which is used for prediction, estimation and control. The implementation of such is heavily dependent on the system it was deployed on. Further in the works of Khater and Fekry [13] a hybrid system with a PI controller and deep Q-learning networks was used for control of a electro-hydraulic servo system.
In summary, while significant achievements have been made in automating excavator operations, current solutions are constrained by trade-offs between speed and precision, limitations of control theory-based approaches, and gaps in addressing specific movement challenges. These limitations present an opportunity to further explore advanced methods, such as reinforcement learning and enhanced perception systems, to overcome these problems and achieve a more robust and capable autonomous excavator system.

1.3 Purpose and goals

The purpose of this thesis is to investigate the application of machine learning, particularly reinforcement learning, in the control of electro-hydraulic systems in construction equipment. The primary focus is on the Volvo EC550E excavator, with the aim of evaluating whether a machine learning-based hydraulic control system using IMVT can outperform traditional controllers in terms of accuracy and speed. Modern hydraulic control systems in excavators face significant challenges due to the inherent nonlinearities and complex dynamics of hydraulic actuators. These issues often lead to suboptimal performance, such as oscillatory behavior and decreased precision during fine manipulation tasks like grading. This thesis proposes the development and evaluation of a RL-based controller capable of mitigating such nonlinear effects while improving fuel efficiency and accuracy.
More specifically, the project explores the feasibility of replacing or augmenting the existing control architecture with a hybrid or fully machine learning-driven approach. The goal is to create an RL-based controller that can generalize across varying task conditions and adaptively compensate for system dynamics, thereby achieving smoother and more energy-efficient operation. This includes evaluating

the performance of the proposed controller on grading tasks, where precision and stability are of critical importance.

1.4 Limitations and current platform

Due to the excavator only being available in Volvo's branch in South Korea, the work will be based around a simulation model developed by the same branch. A reworked version of this simulation model was used, removing the current control system to only incorporate the physical excavator model along with inertial measurement systems (IMUs) and other sensors of the system.
The simulation model is simultaneously under development in CPAC, meaning that the current version of the control system might not align perfectly with later iterations of the simulation model.
The simulation model also imposes major limitations on how realistic the simulation can become. The interaction between the excavator's surroundings and the excavator does not exist in the simulation, meaning that there is no soil to interact with. This means that there are perfect conditions in the model, no friction or force resistance and thus making the control problem more simple as well as unrealistic. The excavator is also static in the environment, which means that oscillations of driving the machine will not be taken into account. The tracks are also always perfectly angled on the horizontal plane, which is usually not the case in normal excavator operations. All grading and trajectory following was conducted in 2D with focus on the hydraulic dynamics of the boom, arm, and bucket.
Due to the work being solely conducted on the simulation model it will not perfectly reflect the actual excavator dynamics. Implementation on a real machine would require reconstruction of the control system and more focus on safety aspects. This thesis will serve more as a proof of concept for control of the IMVT system using RL.

2
Theory

This chapter provides the theoretical foundation necessary to understand the methodology and systems of this thesis. A deeper background in excavator hydraulics is provided to better grasp the problem. Following that, sections about control theory, AI and RL that were relevant for constructing the proposed system are presented.

2.1 Hydraulics

Hydraulics is the branch of engineering that deals with the mechanical properties and use of liquids, particularly water or oil, to transmit power. It involves the study of fluid dynamics, pressure, flow, and how these can be controlled and applied to perform work. The delivered power is highly dependent on oil temperature and pressure, as changes in viscosity affect system performance. Additionally, fluid flows from regions of high pressure to low pressure, which can lead to energy losses if not properly managed.
In excavators with electro-hydraulic systems, hydraulic fluid is managed through a series of poppet valves and control systems that direct pressurized fluid to different actuators to perform tasks such as lifting, rotating, and moving. These systems play a vital role in the functionality of the machine.

2.1.1 Excavator hydraulics

The excavator relies on pressurized hydraulic fluid to convert hydraulic energy into mechanical force, enabling the machine to lift heavy loads and perform intricate movements with precision. The key components of an excavator's hydraulic system include:

- **Pumps:** These convert mechanical energy from the engine into hydraulic energy by generating pressurized fluid. The efficiency of the pump significantly influences the overall system performance and fuel consumption.
- **Cylinders:** These actuators convert hydraulic energy into linear mechanical motion. The extension and retraction of the cylinders enable arm, boom, and bucket movements.
- **Control Valves:** These regulate the flow and direction of hydraulic fluid. Acting as the system's "brain," they ensure fluid is directed appropriately to various actuators based on operator input.
- **Hydraulic Fluid:** This fluid serves as the medium for energy transmission. Its properties such as viscosity, thermal stability, and lubricity are important

for system performance and efficiency. Typically, specialized oils referred to as hydraulic oil are used to ensure minimal wear and efficient flow.

The hydraulic system in an excavator starts at the pumps, which will pressurize the hydraulic oil, causing flow from the pumps towards the cylinders. The flow is regulated with control valves which open or close depending on which chamber in the cylinder should be filled. Oil displaced from the cylinder returns to a reservoir, where it can be filtered, cooled, and reused.

A large portion of the excavator's engine power is used by the hydraulic system, but much of the power is lost during the way. When converting mechanical energy to hydraulic energy, losses occur in both the engine as well as the pumps. Throttling losses also contribute to the overall loss which occur when valves are used to control where the hydraulic power is heading. Other vital functions, such as steering, cooling and breaking are also considered losses as they do not contribute to the execution required task [21].

2.1.2 Electro-hydraulic poppet valves (EHPV)

An EHPV (electro-hydraulic poppet valve) is a type of proportional control valve that is widely employed for motion control in hydraulic applications and are described in [22, 23]. They are more advantageous than spool type valves due to a higher resistance to contamination, fewer failure modes, larger effective flow areas, and superior sealing performance.

An EHPV regulates the flow of hydraulic fluid by modulating the valve's flow conductance coefficient, K_v, through a pulse-width-modulated electric current supplied to a solenoid actuator. This solenoid acts on a pilot-poppet assembly that opens or closes the orifice depending on the current magnitude. The relationship between the flow conductance and hydraulic parameters can be expressed as:

$$K_v = |Q|/\sqrt{|\Delta P|} \tag{2.1}$$

where Q is the flow through the valve and ΔP is the pressure difference between the side flow and nose flow [24].

This relation implies that the conductance K_v increases with higher flow rates or lower pressure drops, and it serves as a measure of how easily the fluid passes through the valve. The EHPV can be modeled as a system with an input current and an output conductance, where the internal mechanical dynamics are governed by a three degree-of-freedom (DOF) structure. These degrees of freedom, subject to mechanical constraints, account for the interactions between the solenoid, pilot, and poppet mechanisms. Notably, EHPVs include a pressure compensation feature that minimizes the variation in actuation current required to initiate valve opening, helping to stabilize performance under changing load conditions. The valve is designed to operate bidirectionally, supporting both forward and reverse flow, and is characterized by minimal internal leakage and low hysteresis.

2.1.3 EC550E system

The hydraulic system in the excavator can be divided into a separate hydraulic system for the boom, arm and bucket and is visualized in Figure 2.1. All three

hydraulic systems are fed from the same two pumps which both has bypass valves, enabling the hydraulic oil to bypass the cylinders and directly flow back to the return reservoir.

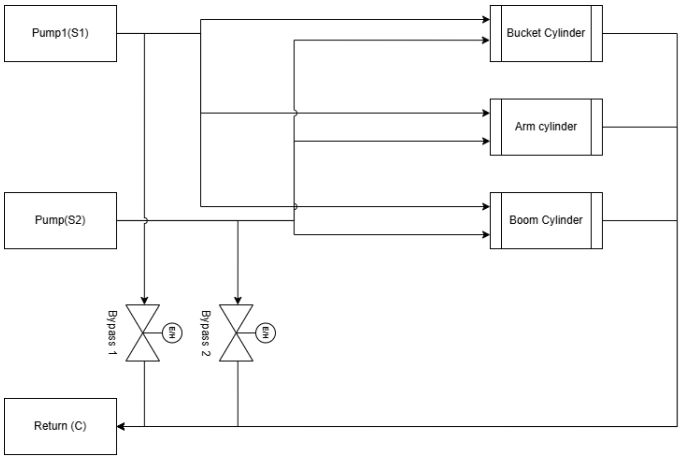


Figure 2.1: Simplified model of the hydraulic system used in EC550E. It shows how all three cylinders are connected to the pumps together with bypass valves.

The flow in the cylinder is regulated through the pumps and several EHPVs set up in a Wheatstone bridge layout. This is referred to as Independent Metering Valve Technology. Research has shown that IMVT setups can improve energy efficiency in the hydraulic system [15, 25] and is beneficial to use in excavators which historically suffer from large energy losses in the hydraulic system.
These valves can be configured in many ways to achieve the wanted flow, and displays a non-linear behavior in many cases. A schematic of the setup used in the simulation platform and real excavator is shown in Figure 2.2. Each valve is controlled by an electric current which ranges from 150-750 mA where an increase in current opens the valve as described in subsection 2.1.2. The system depends on the amount of hydraulic oil that is currently in the pipes and cylinder, which valves are open and which pumps are operating to regulate the movement of the cylinders.

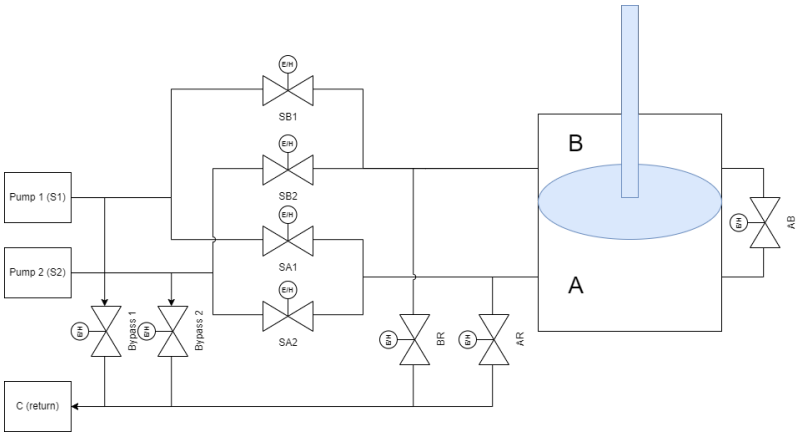


Figure 2.2: Model of the hydraulic system for one cylinder used in the excavator using IMVT with two pumps. All valves are EHPVs. All cylinders share bypass valves.

A key modification from the conventional Wheatstone bridge layout is the inclusion of the **AB valve**, also referred to as the *regeneration valve*. This valve enables hydraulic fluid to flow directly between the two cylinder chambers (from A to B or B to A), depending on the pressure differential. When the pressure in one chamber exceeds the other, the valve can be opened to recycle internal fluid, reducing the volume of oil that must be supplied by the pump. This mode of operation is beneficial in scenarios where the cylinder is absorbing energy, such as during gravitational retraction, and contributes to increased system efficiency.
Another modification exists in the boom cylinder, where pump 1 is not contributing to the B chamber, meaning that the SB1 valve is not present. This is due to the gravitational force which helps the cylinder to retract, making the need for two pumps redundant.
The cylinder can operate in different modes, depending on the desired direction of movement and the pressure conditions in the system.

2.1.3.1 Extension of the cylinder

In the extension mode, the piston rod is driven outward to lengthen the excavator arm. This is achieved by activating both pumps to deliver pressurized oil into chamber A (the piston side of the cylinder). Simultaneously, the EHPVs connected to the A inlet (**SA1** and **SA2**) are opened to admit fluid, while the EHPV connected to the B outlet (**BR**) is opened to allow return flow from chamber B to the tank. The AB valve remains closed during this process in order to prevent undesired cross-flow between the chambers. This configuration ensures that chamber A is pressurized while chamber B is depressurized in a controlled manner.

2.1.3.2 Retraction of the cylinder

During retraction, the piston rod is pulled back into the cylinder, shortening the excavator arm. In this mode, both pumps are activated to supply pressure to chamber B (the rod side) in all cylinders except the boom cylinder, where only pump 1 is used. Through opening EHPVs **SB1** and **SB2**, fluid is allowed into chamber B, while EHPV **AR** is opened to release oil from chamber A back to the tank.

2.1.3.3 Regenerative movement

Regenerative movement is a special case of the hydraulic control where the **AB** valve plays a central role. When the cylinder is operating under an external load, such as gravity acting on the excavator linkage, one of the chambers may become highly pressurized in one or more of the cylinders. In such cases, the **AB** valve is opened to allow oil to flow from chamber A to chamber B or vice versa. This internal recycling of hydraulic fluid reduces the need for the pumps to supply oil to the chamber with low pressure, thereby minimizing energy consumption. The A or B return valve (**AR**/**BR**) may remain closed or partially opened depending on the pressure conditions. Although efficient, this mode requires careful regulation to ensure that the highly pressurized chamber does not become depressurized below

the other chambers, as the oil would equalize the pressure between chamber A and B.

2.1.3.4 Remarks on safety and control

Across all operating modes, it is important to prevent vacuum conditions within the cylinder chambers or connecting pipes, as these can lead to cavitation and irreversible damage to hydraulic components. The control system must therefore continuously monitor chamber pressures, valve positions, and pump activity to ensure stable and safe operation, especially during transitions between modes or when the AB valve is actively engaged.

2.2 PID-controller

A proportional-integral-derivative (PID) controller is a widely used feedback control strategy for systems that require continuous regulation [26, 2]. It is often considered a good starting point in control design due to its simplicity and effectiveness for many linear systems.

The PID controller operates based on the error signal $e(t) = r(t) - Y(t)$ which represents the difference between the desired output $r(t)$ and the actual system output $Y(t)$. It combines three components:

- **Proportional (P)**: Reacts to the current error.
- **Integral (I)**: Accumulates past errors to eliminate steady-state error.
- **Derivative (D)**: Anticipates future error based on its rate of change.

The standard continuous-time PID control law is expressed as:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \tag{2.2}$$

where $u(t)$ is the control signal, $e(t)$ is the error, and K_p, K_i and K_d are the proportional, integral, and derivative gains, respectively.

2.3 Recurrent neural networks (RNNs)

Recurrent neural networks (RNN), are a branch of neural networks designed to handle sequential data by retaining a memory from previous inputs [27]. This is accomplished in the network by incorporating a self connection to process information from previous time-steps. An illustration of a recurrent layer can be seen in Figure 2.3.

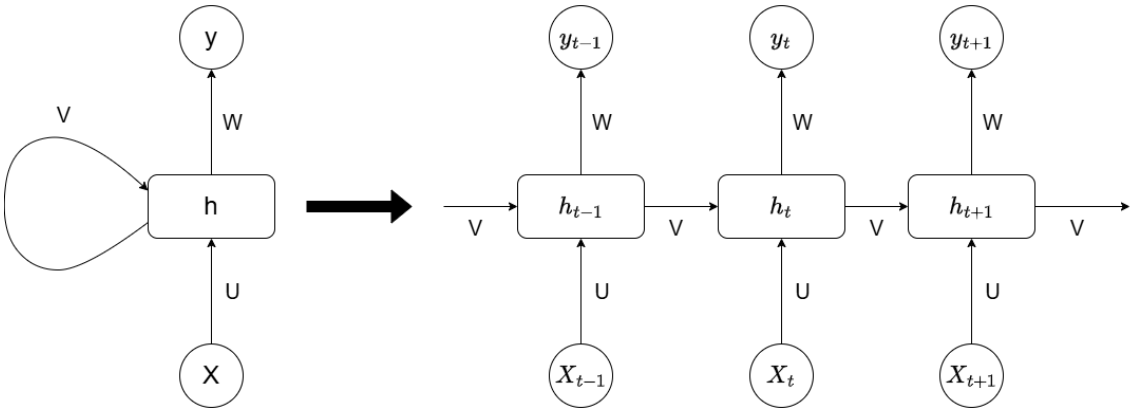


Figure 2.3: Basic concept of RNN, with self connection through time where x denotes the input to the network, y the output of the network, h the hidden state, v the self-connection, U and W are weights between the layers.

The RNN processes data sequentially while maintaining the hidden state which contains information about previous inputs. The hidden state is updated at each step based on the input to the network for the current timestep and the previous hidden state. This recurrence helps the network keep a memory across several time steps. However, the RNNs processing of sequential data can lead to two common problems during training: vanishing gradients or exploding gradients [28]. These issues arises from the backpropagation in the training where the derivatives of the network's time layers accumulate to either very large or very small numbers. If the gradient vanishes, the network has a hard time to establish a long term interdependence and if the gradient grows very large, it can lead to a highly unstable training.

2.3.1 Long short-term memory (LSTM)

One of the more common approaches to enhance the standard RNN is called long short-term memory (LSTM), which uses cell state along with the hidden state to process information between time steps, and a gated unit in the layer to handle which information to keep and discard based on the previous hidden state and current input to the layer [28].

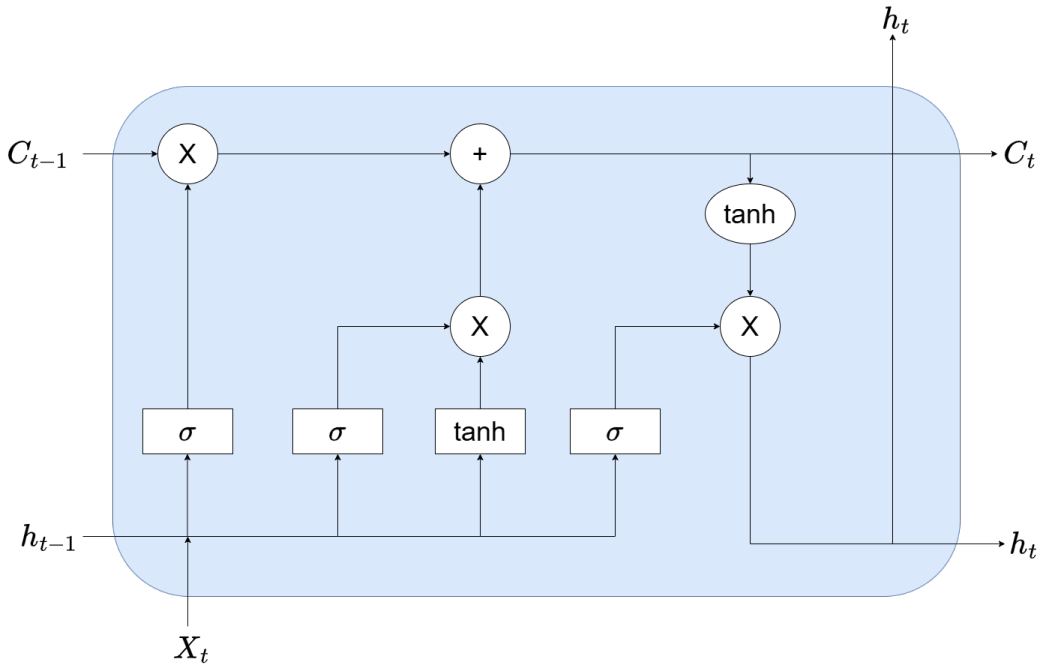


Figure 2.4: Structure of the hidden layer in an LSTM network at timestep t. With cell state C, hidden state h, sigmoid (σ) and tanh activation function.

The gated unit consists of three gates that takes the input of the network along with the previous cell state and hidden state to process which information should be passed on.
The three gates from left to right in Figure 2.4 are:

- **Forget gate:** Decides which information should be discarded from the cell state. It applies a sigmoid activation function to the previous hidden state and current input, producing a value between 1 and 0 which indicates how much of the previous information to keep.
- **Input gate:** Determines how much of the new information from the previous hidden state and current input should be added to the cell state. Uses a sigmoid function to regulate value updates and a tanh function to create new candidate values.
- **Output gate:** controls the final output from the layer based on the cell state. Uses a sigmoid function to determine what part of the cell state should be outputted followed by a tanh function.

2.4 Reinforcement learning (RL)

Reinforcement learning is a subfield of machine learning in which an agent learns to interact with an environment by performing actions and receiving feedback in the form of rewards. The goal of the agent is to learn a policy that maximizes the cumulative reward over time [29, 30]. RL is particularly suited for control problems where the dynamics of the system are unknown or highly nonlinear, as is the case with hydraulic systems in heavy machinery.

Unlike supervised learning, RL does not require labeled inputs and outputs. Instead, the agent learns from the consequences of its actions through trial and error. The key components of an RL system are:

- **Agent:** The algorithm that learns and make decisions
- **Environment:** External system which the agent interacts with
- **Observation/state (S):** Representation of the current situation the Agent is in the environment
- **Action (A):** A set of possible decisions the agent can take from the current state
- **Reward (R):** A scalar value given as feedback to the agent, indicating the success of the current action
- **Policy (π):** A strategy the agent follows to determine the actions based on the states

This interaction can be visualized as a feedback loop, shown in Figure 2.5, where the agent observes the state of the environment, takes an action, receives a reward and then transitions to a new state.

Figure 2.5: Visualization of RL system. The agent takes actions based on the observations and rewards calculated from the environment.

2.4.1 Branches and applications

RL methods are generally categorized into two main branches: model-based and model-free approaches [30].

2.4.1.1 Model-based methods

Model-based methods attempt to learn an explicit model of the environment's dynamics. This model is then used by the agent to simulate and evaluate future trajectories before taking an action, making the learning process more sample efficient. These methods are well suited for tasks where accurate modeling is feasible and interactions with the environment are costly. However, their performance heavily depends on the accuracy of the learned model, and they can struggle in environments with high complexity or stochastic behavior [31].

2.4.1.2 Model-free methods

Model-free methods, in contrast, do not learn an explicit model of the environment. Instead, they learn directly from experience by interacting with the environment and updating their behavior based on observed rewards. These methods are more robust to complex or partially observable environments but tend to require more data due to their lack of internal environmental simulation.
Model-free methods can be further divided into two subcategories:

- **Value-based methods**: These methods, such as Q-learning, estimate value functions that quantify how good it is to be in a given state or to take a particular action. Q-learning uses a tabular approach or a function approximation to estimate action-values and selects actions that maximize expected rewards. While foundational and conceptually simple, value-based methods face scalability issues when applied to high-dimensional or continuous spaces.
- **Policy-based methods:** These learn a parametrized policy directly, without estimating value functions. They are well suited for high dimensional spaces and allow for stochastic policies. The core idea is to optimize the parameters of the policy to maximize the expected rewards by using gradient descent methods.

In practice, actor-critic methods combine the strengths of both value-based and policy-based approaches by using a value function (the critic) to guide the policy updates (the actor) [30, 32]. These hybrid methods form the foundation for many modern RL algorithms, especially when combined with deep neural networks to handle complex environments and high-dimensional data.

2.4.2 Policy gradient methods

Policy gradient methods performs stochastic gradient ascent on an estimator of the policy gradient to maximize the reward [33, 34]. The objective of policy based RL agents is to maximize the expected reward following a policy, π, by defining a set of parameters, Θ, to parametrize the policy π_Θ. In policy gradient methods the goal is to directly learn a policy function using stochastic gradient ascent on an estimator of the policy gradient to maximize the reward. The policy is typically represented as a parametrized function: $\pi_\Theta(a|s)$ where π is the policy, Θ are the parameters, a is the action taken and s is the state. The goal is to maximize the sum of future rewards for the policy:

$$J(\Theta) = E_{\pi_\Theta}[R] = E_{\pi_\Theta}[\sum_{t=0}^{T} \gamma^t r_t] \tag{2.3}$$

- E_{π_Θ} is the expectation over the policy
- r_t is the reward at timestep t
- γ is a discount factor
- T is the episode length

The policy is improved by updating the parameters Θ in the direction that increases expected return. This is done by computing the gradient of $J(\Theta)$ with regards to Θ:

$$\nabla_\Theta J(\Theta) = E_{\pi_\Theta}[\nabla_\Theta \log \pi_\Theta(a|s) Q^\pi(s,a)] \tag{2.4}$$

- $\nabla_\Theta log\pi_\Theta(a|s)$ is the gradient of log probability of taken action a given the state s under the policy π_Θ
- $Q^\pi(s,a)$ is the action-value function that gives the expected return of action a given the state s.

The gradient estimate can be noisy. This can be avoided in policy gradient methods by using stochastic gradient ascent, through an actor critic based method. To reduce the variance, a value function (critic), $V^\pi(s)$ is used to estimate the mean rewards from the current state over all possible actions. The value is then subtracted from the actual reward $Q^\pi(s,a)$ from the action taken by the policy (actor), resulting in the advantage function:

$$\hat{A}^\pi(s,a) = Q^\pi(s,a) - V^\pi(s) \tag{2.5}$$

The advantage can be seen as an estimate of how much better an action is compared to the average in a specific state. This leads to a more stable and efficient gradient estimate:

$$\nabla_\Theta J(\Theta) \approx \hat{E}_{\pi_\Theta}[\nabla_\Theta log\pi_\Theta(a|s)\hat{A}^\pi(s,a)] \tag{2.6}$$

2.4.3 Proximal policy optimization (PPO)

Proximal policy optimization (PPO), introduced by Schulman et al. [34], is a widely used policy gradient method in reinforcement learning, known for its balance of performance and implementation simplicity. PPO has been successfully applied in various domains, including robotics and game environments [7, 35, 36, 37].

PPO addresses a key challenge in policy gradient methods: determining how much to update the policy in response to new data. If updates are too small, learning progresses slowly and inefficiently. If updates are too large, the policy can diverge, degrading performance. PPO improves upon the earlier trust region policy optimization (TRPO) algorithm [19] by enforcing constraints through a simpler and computationally efficient surrogate objective function.

The method operates iteratively. An initial policy is used to collect experience from the environment. This data is then used to compute an improved version of the policy, and the process is repeated. Central to PPO is the use of a probability ratio that compares the new policy to the old policy:

$$r_t(\Theta) = \frac{\pi_\Theta(a_t|s_t)}{\pi_{\Theta_{old}}(a_t|s_t)} \tag{2.7}$$

By weighting the new policy against the old, the policy is updated to increase the probability for profitable actions. It also ensures that the updates are larger for actions where the new policy differs significantly from the old. With the probability ratio, the gradient objective (objective function) is defined as:

$$L_{CPI}(\Theta) = \hat{E}_t[r_t(\Theta)\hat{A}_t] \tag{2.8}$$

where $\hat{A}_t$ is the advantage estimate at time step t. This objective encourages the policy to increase the likelihood of actions that yield higher than expected rewards (positive advantage) and decrease it for actions with negative advantage.

2.4.3.1 Clipped surrogate objective

To overcome the challenge of computational infeasibility in complex environments, surrogate objectives are introduced [33, 34]. This is an alternative objective that approximates the true goal of maximizing the future rewards. It captures the aspects of the goal, while still being able to guide the agent to improved performance. The clipped surrogate objective is defined as:

$$L^{CLIP}(\Theta) = \hat{E}_t[\min(L_{CPI}(\Theta), \text{clip}(r_t(\Theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{2.9}$$

Where ϵ is a hyperparameter, usually 0.2. The term $\text{clip}(r_t(\Theta), 1-\epsilon, 1+\epsilon)A_t$ is the clipped surrogate objective, used to clip the probability ratio, which hinders moving r_t outside of the interval $[1-\epsilon, 1+\epsilon]$. Then by taking the minimum of the clipped and unclipped objective, the change in probability ratio is ignored if the objective improves. The probability ratio is clipped at $1-\epsilon$ or $1+\epsilon$ depending on if the advantage is positive or negative.

2.4.4 Generalized advantage estimation (GAE)

Generalized advantage estimation (GAE) is a RL technique to calculate more accurate advantage estimations with the primary purpose being improving stability and efficiency of policy optimization algorithms [38]. The standard way to estimate the advantage is through the temporal difference error:

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{2.10}$$

However, high variance can occur in this calculation if only one timestep is used. GAE reduces that variance by introducing a trade-off between variance and bias. It uses the temporal difference error over multiple time steps to create an average estimate of the advantage.

$$\hat{A}_t^{GAE(\lambda)} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l} \tag{2.11}$$

where γ is the discount factor, determining how future rewards are weighted compared to the immediate. λ is the GAE controlling the trade of between bias and variance and smooths out the temporal difference error over time. This leads to a smoother and more stable estimate of the advantage function, which results in improved learning stability.

3
Method

To simplify the overall problem, the task of achieving effective grading with an excavator is divided into smaller, more manageable sub-problems each with its own solution approach seen in Figure 3.1. The overall control system can be broken down into two main components:

- **Positional Controller** – This component is responsible for determining the desired cylinder velocities needed to achieve the target motion. It relies on understanding the excavator's current position in the world and calculating where it needs to go.
- **Hydraulic Controller** – This part translates the desired cylinder velocities into commands for the hydraulic system, including pump demands, engine requirements, and valve positions.

By developing robust solutions for each of these sub-problems, an effective system for automated grading can be achieved. In this chapter, control theory is evaluated for positional control, backwards kinematics for computing desired cylinder velocities and construction of a machine learning approach for controlling the hydraulics. All implementations are tested within the provided simulation platform.

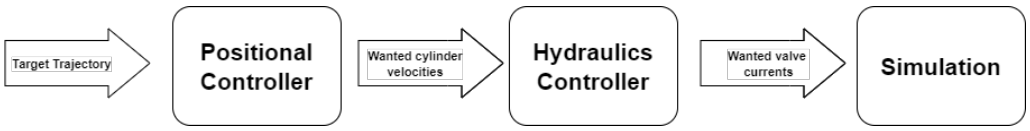


Figure 3.1: Overview of the sub-problems and the information required by each controller.

The work starts with detailing the development of the Positional controller which outputs a target bucket pose and is then translated into actuator velocities through backward kinematics in a mechanical model of the excavator. The simulation platform is also described, as well as its conversion from Simulink to Python and any modifications made during the process.

3.1 Mechanical model

A fundamental understanding of the excavator's physical behavior is essential for building a control system. To that end, a simplified mechanical model is developed to compute both forward and inverse kinematics calculations for the excavator's movement.

3.1.1 Coordinate systems and naming conventions

Because of the multiple DOF, there will be several coordinate frames. The name of joints, point of interest and coordinate frames of the excavator can be seen in Figure 3.2.

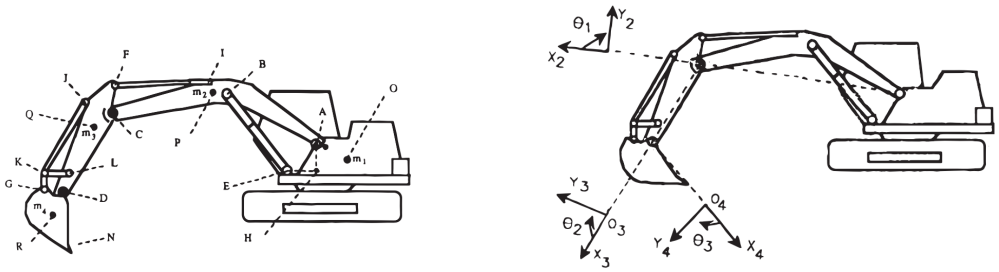


Figure 3.2: Excavator model with points of interest labeled as well as important coordinate systems highlighted. Reproduced with permission. Source: [2]

A coordinate in frame n from a point A will be referred to as $\boldsymbol{C}_{An} = \begin{bmatrix} x_{An} \\ y_{An} \end{bmatrix}$. A vector from a point A to another point B in coordinate frame n will be referred to as $\boldsymbol{C}_{ABn} = \boldsymbol{C}_{Bn} - \boldsymbol{C}_{An}$. The angle between two vectors $\boldsymbol{C}_{ABn}$ and $\boldsymbol{C}_{BCn}$ will be referred to as $\boldsymbol{\theta}_{ABC}$. The angle between a vector $\boldsymbol{C}_{ABn}$ and the x-axis in coordinate frame n will be referred to as $\boldsymbol{\theta}_{ABXn}$. The distance between two points will be referred by $d_{AB} = \sqrt{(x_{An} - x_{Bn})^2 + (y_{An} - y_{Bn})^2}$. When referring to any of the indices in a full set of parameters the subscript notation $(*)$ is used.
As seen in Figure 3.2 there are three coordinate systems of interest. System 1 that belongs to the boom and has its center in the joint between boom and house (A). System 2 for the arm that has a center-point in the joint between arm and boom (C). Lastly system 3 that belongs to the bucket and has a center point in the joint between bucket and arm (D).
The orientation of the excavator will often be described as joint positions and joint angles. The absolute angles of the excavator is denoted as:

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix} = \begin{bmatrix} \theta_{ACX0} \\ \theta_{CDX0} \\ \theta_{DNX0} \end{bmatrix} \tag{3.1}$$

The lengths of the links will be denoted as

$$\boldsymbol{L} = \begin{bmatrix} L_1 \\ L_2 \\ L_3 \end{bmatrix} = \begin{bmatrix} d_{AC} \\ d_{CD} \\ d_{ND} \end{bmatrix} \tag{3.2}$$

To follow the bucket movement the end effector for the bucket tip is defined as the pose

$$\boldsymbol{p} = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} x_{N0} \\ y_{N0} \\ \theta_3 \end{bmatrix} \tag{3.3}$$

Angles and angle-velocities of the system are as defined positive for counterclockwise movement. Cylinder lengths are defined as the length of the extension, with fully retracted being 0 and fully extended being the maximum length. Cylinder velocities are defined as positive when extending and negative when retracting.

3.1.2 Inverse kinematics

The positional controller calculates the target pose, and by using inverse kinematics in the mechanical model the pose can be translated to target actuator velocities, which will be used as input for the hydraulic controller. For the inverse kinematics there are two major mappings to find, from angle to pose, and angle to actuator length:

$$F(\boldsymbol{\theta}) = \boldsymbol{p} \tag{3.4}$$

$$G(\boldsymbol{\theta}) = \boldsymbol{l} \tag{3.5}$$

Firstly, the position of the bucket tip is related to the angles of the joints through forward kinematics:

$$p_1 = L_1\cos(\theta_1) + L_2\cos(\theta_2) + L_3\cos(\theta_3) \tag{3.6}$$

$$p_2 = L_1\sin(\theta_1) + L_2\sin(\theta_2) + L_3\sin(\theta_3) \tag{3.7}$$

$$p_3 = \theta_3 \tag{3.8}$$

The bucket velocity can be derived by taking the derivative of the positions:

$$\dot{p}_1 = -L_1\sin(\theta_1)\dot{\theta_1} - L_2\sin(\theta_2)(\dot{\theta_2}) - L_3\sin(\theta_3)(\dot{\theta_3}) \tag{3.9}$$

$$\dot{p}_2 = L_1\cos(\theta_1)\dot{\theta_1} + L_2\cos(\theta_2)(\dot{\theta_2}) + L_3\cos(\theta_3)(\dot{\theta_3}) \tag{3.10}$$

$$\dot{p}_3 = \dot{\theta}_3 \tag{3.11}$$

In vector form, the relationship between the pose velocities and the angular velocities can be expressed as:

$$\dot{\boldsymbol{p}} = J(\boldsymbol{\theta})\dot{\boldsymbol{\theta}} \implies \dot{\boldsymbol{\theta}} = \dot{\boldsymbol{p}}J(\boldsymbol{\theta})^{-1} \tag{3.12}$$

Where the Jacobian is defined as:

$$J(\boldsymbol{\theta}) = \begin{bmatrix} -L_1\sin(\theta_1) & -L_2\sin(\theta_2) & -L_3\sin(\theta_3) \\ L_1\cos(\theta_1) & L_2\cos(\theta_2) & L_3\cos(\theta_3) \\ 0 & 0 & 1 \end{bmatrix} \tag{3.13}$$

From the vectorized angular velocity around the z-axis ($\theta_{(*)}$), and the positions of the pins connecting the cylinders to the links (B, F and K in Figure 3.2) the cylinder velocity can be calculated by taking velocity of the cylinder pin relative to the link

$(\dot{\boldsymbol{\theta}}_{(*)} \times \boldsymbol{C}_{cylpin})$, and then computing the speed component in the direction of the cylinder movement $(\hat{\boldsymbol{C}}_{cyl})$:

$$\dot{l} = (\dot{\boldsymbol{\theta}}_{(*)} \times \boldsymbol{C}_{cylpin}) \cdot \hat{\boldsymbol{C}}_{cyl} \tag{3.14}$$

Because the distance between joint position and cylinder pin is constant for the boom and arm, this equation can be used directly to calculating the cylinder velocity. However, the bucket cylinder is not directly connected to the bucket, but rather connected on the yoke (K) which is connected to the bucket through a 4 bar linkage. The 4 bar linkage consists of joints L, K, G and D as seen in Figure 3.2, where L is attached to the arm of the excavator and D is a joint between the arm and the bucket.
The speed vector $\dot{\boldsymbol{C}}_{G0}$ can be calculated by using the angular velocity from the bucket $\dot{\theta_3}$ and crossing it with the vector $\boldsymbol{C}_{DG0}$.

$$\dot{\boldsymbol{C}}_{G0} = \dot{\boldsymbol{\theta}}_{\boldsymbol{3}} \times \boldsymbol{C}_{DG0} \tag{3.15}$$

To find speed vector in the direction of $\boldsymbol{C}_{KG0}$, the calculated speed vector can be multiplied with the unit vector in the corresponding direction, resulting in:

$$\dot{\boldsymbol{C}}_{KG0} = (\dot{\boldsymbol{C}}_{G0} \cdot \hat{\boldsymbol{C}}_{KG0})\hat{\boldsymbol{C}}_{KG0} \tag{3.16}$$

To find the resulting velocity of the bucket cylinder $\dot{l}_3$, $\dot{\boldsymbol{C}}_{KG0}$ can be projected with the unit vector $\hat{\boldsymbol{C}}_{JK0}$ resulting in:

$$\dot{l}_3 = \dot{C}_{JK0} = \dot{\boldsymbol{C}}_{KG0} \cdot \hat{\boldsymbol{C}}_{JK0}. \tag{3.17}$$

The velocity for the three different cylinders can now be expressed as

$$\dot{\boldsymbol{l}} = \begin{bmatrix} \dot{l}_1 \\ \dot{l}_2 \\ \dot{l}_3 \end{bmatrix} = \begin{bmatrix} (\dot{\boldsymbol{\theta}}_1 \times \boldsymbol{C}_{AB0}) \cdot \hat{\boldsymbol{C}}_{EB0} \\ (\dot{\boldsymbol{\theta}}_2 \times \boldsymbol{C}_{CF0}) \cdot \hat{\boldsymbol{C}}_{IF0} \\ (((\dot{\boldsymbol{\theta}}_3 \times \boldsymbol{C}_{DG0}) \cdot \hat{\boldsymbol{C}}_{KG0})\hat{\boldsymbol{C}}_{KG0}) \cdot \hat{\boldsymbol{C}}_{JK0} \end{bmatrix} \tag{3.18}$$

3.2 Positional controller

To serve as a positional controller in the grading system, a PID-controller is implemented. While effective in many settings, PID controllers are inherently linear and symmetric, making them less suited for complex systems with nonlinear dynamics, such as those encountered in an excavator. However, much of the non linear behavior of the excavator is due to the hydraulic system. With the proposed solution, the hydraulic controller is expected to remove much of the nonlinear behavior, making the system linear enough to rely on a simple PID for positional control. In this application, the controller error is defined as the difference between the desired pose and the measured pose of the excavator:

$$\boldsymbol{e}_p = \boldsymbol{p}_{ref} - \boldsymbol{p}_{measured} \tag{3.19}$$

The controller only tries to minimizes pose error, and lacks awareness of what velocity the system should have to achieve the desired pose. Since the error signal

only considers current position, the controller has no means to regulate the system's velocity during motion. This can lead to movement that is either too slow or too fast for the task at hand.

To address this limitation of the standard PID, a feed-forward term is introduced, based on the reference pose velocity:

$$\dot{\boldsymbol{p}}_{ref} = \begin{bmatrix} \dot{x}_{ref} \\ \dot{y}_{ref} \\ 0 \end{bmatrix} \tag{3.20}$$

The reference velocity is composed of an x-component ($\dot{x}_{ref}$) and y-component ($\dot{y}_{ref}$), in the direction of the trajectory. The angle velocity component is controlled by the other terms of the PID and is set to zero. The feed-forward term consists of the reference velocity multiplied by a gain K_{ff}. This is introduced to provide the system with a desired velocity when performing the grading task, guiding both the magnitude and direction of the velocity. While the other terms in the controller tries to minimize the pose error, the feed-forward term tries to keep a constant velocity in the direction of the trajectory. With this velocity reference, the controller can be constrained to follow a target speed, both for linear motion and the bucket's rotation, enhancing task-specific behavior.

By augmenting the original PID formulation in Equation (2.2) with the feed-forward term, the extended control law becomes:

$$\dot{\boldsymbol{p}}_{pref}(t) = K_p \boldsymbol{e}(t)_p + K_i \int_0^t \boldsymbol{e}(\tau)_p d\tau + K_d \frac{d\boldsymbol{e}(t)_p}{dt} + K_{ff} \dot{\boldsymbol{p}}_{ref} \tag{3.21}$$

Here K_{ff} is the gain associated with the feed-forward term, which scales the influence of the desired velocity on the control signal. This combination allows the controller to not only correct deviations in position but also follow a velocity trajectory suitable for the grading task. The complete control structure is illustrated in Figure 3.3:

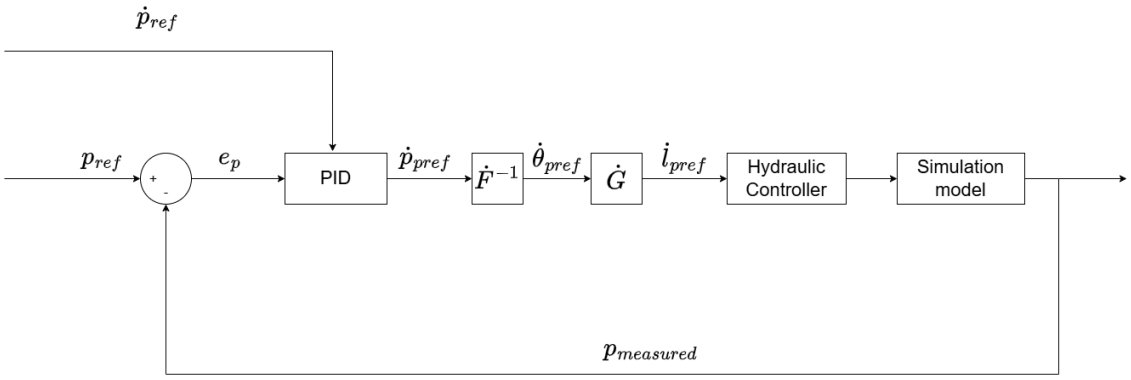


Figure 3.3: Flowchart of the PID control loop for excavator motion. The controller receives reference pose and velocity inputs to generate control actions.

3.3 Simulation model

To facilitate easy and safe testing, a simulation model is used as a benchmark, verification tool, and training environment for the RL agent. The simulation model is provided by the excavator manufacturer and is based on the model EC550E. The simulation of the excavator is modeled in a dynamic way, enabling the user to select how the excavator should be controlled, either by joystick input, or manual control of each cylinder. The model is created using Matlab's Simulink, and together with Simscape a full multibody system can be simulated accurately [39].

To facilitate the needs in the project, an interface to the simulation model is created. This interface can be accessed through the programming language Python which provides open source libraries for easy training and verification of machine learning models. Simulink provides a way to facilitate this, through their embedded coder which is able to convert the simulation model to C code which then can be interfaced in Python [40]. Previous research projects have used the same approach when trying to train machine learning models on Simulink models [20]. The embedded coder exposes top level inputs and outputs, a function to initialize the model, a function to take one step in the simulation as well as a terminate function. To be able to use the model in a Python environment, the following steps have to be taken:

- Generate code in C using the embedded coder
- Compile the generated code into a dynamic linking library (DLL)
- Import the DLL into Python using the **ctypes** library
- Create a training environment for the model

This includes modifying the existing Simulink model to get the wanted inputs and outputs exposed to the Python interface to be able to control the excavator and calculate rewards.

3.3.1 Modifications to the simulation model

The original simulation model consisted of several interconnected subsystems, including operator input, control logic, an environment block (partially implemented), and a detailed physical model of the excavator. For the purpose of deploying a fully autonomous RL-based control system, only the physical model was retained. The excavator was decoupled from the remaining system components, and its input/output interfaces were modified to be compatible with the generated C-code. This reconfiguration allowed the control signals to bypass the traditional control block and instead directly interface with the hydraulic valve solenoids.

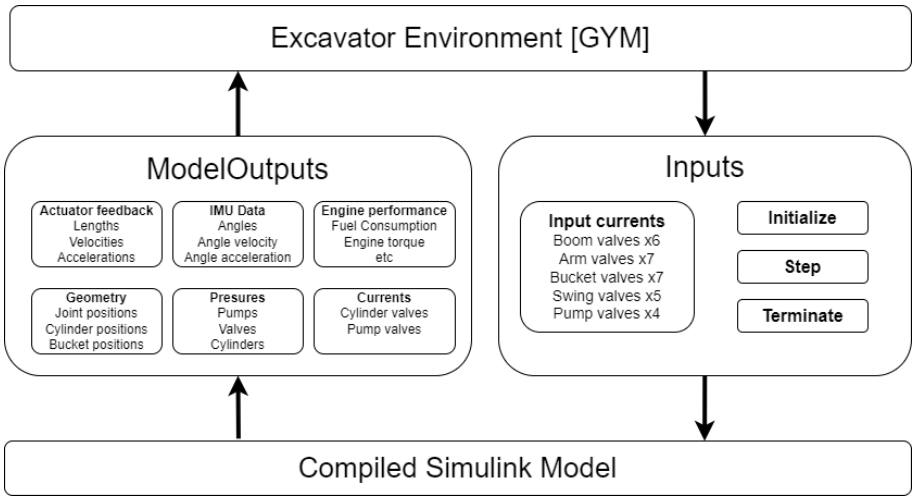


Figure 3.4: Illustration of the data being shared between the Simulink model and Python environment with the most important inputs and outputs shown

In this setup, the control inputs are defined as electrical currents applied directly to the hydraulic valves responsible for actuating the excavator's mechanical subsystems, as illustrated in Figure 3.4. These include the boom, arm, bucket, and swing as well as pump control.

To support learning and analysis, the model also outputs a rich set of signals. These outputs are used both as observations for the RL agent and for monitoring and debugging via a Python-based interface. The output signals provide a detailed view of the excavator's physical, mechanical, and hydraulic states and the most important ones are shown in Figure 3.4.

A full specification of all input and output signals, including units and descriptions, is provided in Appendix B.

3.3.2 Verification of provided model

As of the time the simulation model was provided, no official verification of the model had yet been performed. However, since test data from the actual machine could be collected through CAN-data, a comparison could be made between the two environments. The result of an episode over 200 seconds can be seen in Figure 3.5. The simulation model performs accurately in most scenarios, with the exception when the actual machine interacts with the soil. This can be observed at around 160 seconds as minor oscillations occur in the real machine but not in the simulation model.

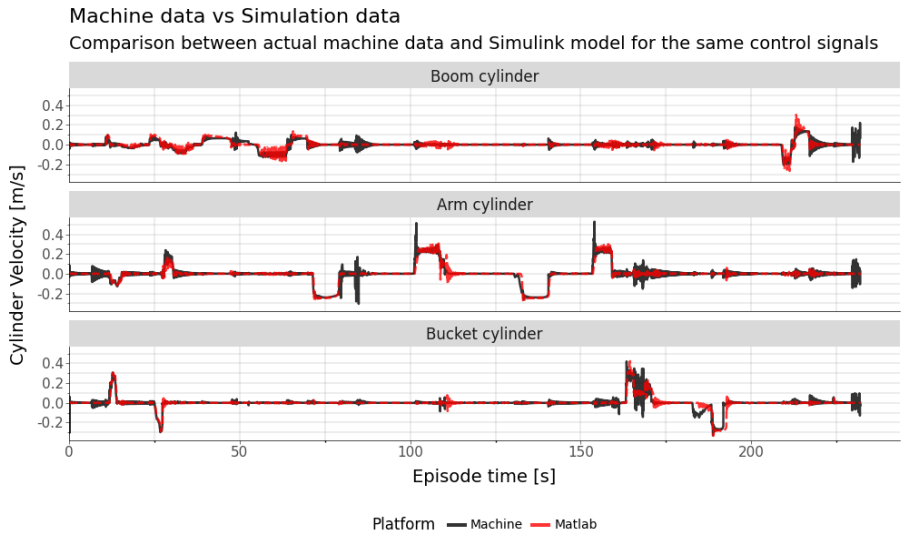


Figure 3.5: Comparison of the three different actuators on a real machine compared to the Simulink model.

3.3.2.1 Verification of python bridge

As the model has been compiled into C code and exported into Python, ensuring that identical inputs in Simulink and Python produce identical results is important to be able to trust the results. To ensure this, logs were collected from both the Simulink and Python simulation models where they were initialized to the same starting positions and the same commands were sent to the models and evaluated. The cylinder displacement tracking performs well for the bucket and boom, while there are some small differences for the arm cylinder displacement, but only when moving the bucket. However, since the differences are small and rare, this is not an issue for the scope of this project. A summary of the comparison is shown in Figure A.1. Using the same test cases, the IMU values are also compared between the models and can be seen in Figure A.2.

3.3.3 Training environment

The training environment follows the conventions of the open source library Gymnasium which implements a standard for training reinforcement learning models [41]. The environment consists of several key concepts which is derived from the base environment class implemented in Gymnasium. The environment defines an observation space as well as an action space which is used to interact with the simulation model. The observation space is a structured datatype that defines which signals can be extracted and fed to the agent deciding which action to take. The actions are also clearly defined by the action space, setting bounds and datatype.
The environment implements four important methods:

1. Initialize: The underlying model is initialized and set to its initial state, observation space and action space is defined.
2. Step: Receives an action and passes it to the model and steps the model for a configurable amount of time steps. An observation is then made and a reward

is calculated. Returns the observation, reward and if the model terminated or truncated.

3. Reset: Resets the underlying model by terminating it and initializing it again. Removes any saved state and observation. Returns a fresh observation.
4. Render: Optionally displays a GUI of the excavator and its movements.

The training environment is designed with a modular architecture that allows it to be flexibly wrapped by any control algorithm, see Figure 3.6. This design enables consistent evaluation across multiple settings, enabling both the testing of different environments using the same control algorithm and the comparison of various algorithms within a single environment configuration.

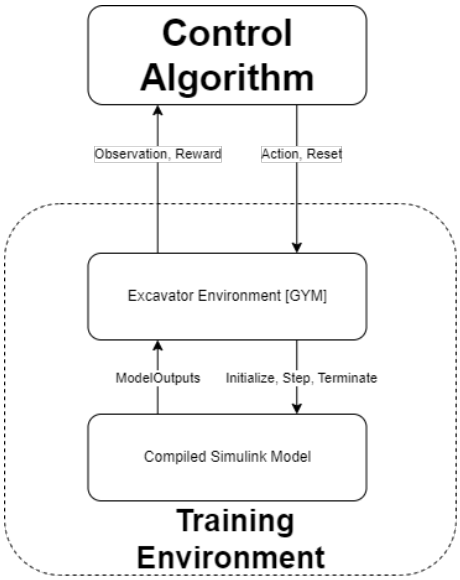


Figure 3.6: Architecture of the training environment, exposing a Gymnasium-compatible interface which internally interacts with the Simulink model.

To accelerate training and enhance sample efficiency, the environment implementation supports parallel execution, allowing multiple instances to run simultaneously. However, due to limitations of the `ctypes` library used to interface with the Simulink DLL, each environment instance must maintain an isolated memory space. As a result, separate copies of the DLL are required, and each environment must be initialized in its own thread.

Although Gymnasium offers basic support for vectorized environments through its wrappers, it lacks some essential features required for efficient logging and inspection during training. To address this, the vectorized environment implementation from Stable Baselines3 (SB3) was adopted [42]. This implementation provides a more robust interface for managing multiple environments concurrently and aligns better with the requirements of this project. It should be noted that there are some minor differences between the Gymnasium and SB3 implementations, particularly in the handling of truncation and termination signals.

3.3.3.1 Observations and actions

The performance of the RL agent depends heavily on the structure and content of the observation and action spaces. These serve as the interface between the agent and the simulation model, defining what the agent can perceive and how it can interact with the environment. The observations include both raw simulation outputs and engineered features, such as error signals between desired and actual values. Choosing which signals to include is a matter of balancing relevance and complexity. For low-level control (e.g., regulating cylinder velocities), detailed signals such as actuator states and hydraulic pressures are essential. Higher-level control would instead benefit from global pose or trajectory information. The signals used as observations for the specific task of controlling cylinder velocity can be seen in Table 3.1.

Table 3.1: Observation signals used as input to the control policy.

Observation Signal	**Dimensions/Unit**	
Cylinder velocity error	3×1	[m/s]
Cylinder velocity	3×1	[m/s]
Cylinder lengths	3×1	[m]
Engine angular velocity	1×1	[rad/s]
Cylinder pressures	6×1	[bar]
Pump pressures	2×1	[bar]
Previous valve currents	24×1	[mA]

Actions represent the control commands sent to the simulation model and is in this case valve currents. The environment expects action values normalized in the range [-1,1], which are later rescaled to match their physical ranges before being applied in the simulation. The complete list of actions is shown in Table 3.2.

Table 3.2: List of all action signals controlled by the agent.

Action Signal	**Dimensions/Unit**	
Boom valve currents	6×1	[mA]
Arm cylinder valve currents	7×1	[mA]
Bucket cylinder valve currents	7×1	[mA]
Pump 1 valve current	1×1	[mA]
Pump 1 bypass valve current	1×1	[mA]
Pump 2 valve current	1×1	[mA]
Pump 2 bypass valve current	1×1	[mA]

The boom, arm, and bucket cylinders all share the same valve layout, see Table 3.3, with one minor difference. The boom cylinder lacks the **SB1** valve (Pump 1 to

chamber B), as it has only one B-side inlet. Since the swing is not taken into account in this project, those signals and relevant observations were disregarded and always kept constant.

Table 3.3: Valve layout for a single hydraulic cylinder.

Valve Function	**Abbreviaton**
Inlet A from Pump 1	SA1
Inlet A from Pump 2	SA2
Inlet B from Pump 1	SB1
Inlet B from Pump 2	SB2
Return A	AR
Return B	BR
Regen between A and B	AB

All observations were normalized by dividing them by the highest value possible defined in the environment. This ensures that the observations are all below 1, but are able to be negative for the observations which require it

3.3.3.2 Episode initialization

To make learning generalizable for the agent, the environment is initialized with 16 positions within the excavator workspace for each episode. By giving the agent different initial observations for each run it will learn to be adaptable to all positions in the workspace. The optimal solution for learning would be to randomly initialize the start position for each episode but due to incompatibilities between the Simulink model and code generation, the initial conditions of the C-code generated model can't be changed. To not give the model the same starting position for every run, C-code was generated for 16 starting positions in different areas of the workspace. This is not an optimal solution but gives the agent a learning opportunity from most areas of the workspace.
This is essential when training, when all the different episodes are run using the same policy. If the environment would start with the same initial conditions, there would be no variance in the data, and the agent would overfit on the current initial positions.

3.3.3.3 Command sampling

Desired cylinder velocities are sampled in the beginning of each training episode, for each of the boom, arm, and bucket. The velocities are randomly initialized in the range [-0.4,0.4] m/s and are unchanged throughout the episode. If any of the cylinders reach their extension/retraction limit, the wanted velocity will be set to zero to not encourage the algorithm to extend past the limits physical limits of the cylinder.

The operation frequency of the environment will be dependent on the IMUs, which has a update frequency of 100 Hz compared to the simulation model which run in 400 Hz. This means that between every IMU update, the simulation takes 4 steps. Due to the fact that the cylinder velocities and lengths are calculated solely from the angles and velocities of the IMUs, the command sampling frequency should be equal or lower than the IMUs update frequency. It is inefficient to send the same observations to the agent several times because it will give the same control signals with more computation.

3.4 Hydraulic control

The high complexity of the hydraulic system makes it difficult to achieve a solution with both high efficiency and precision. There are several approaches to solve this problem; however, as discussed in section 1.2, machine learning has been seen as the most promising in recent research.
Instead of calculating the setpoints for the valves and the required pump flow, one can utilize machine learning techniques to train an agent which will output the desired action for any given observation. Since there is no large dataset of sensor data available for different movements and optimal control of the valves, unsupervised learning is the best approach, specifically reinforcement learning which is explained in section 2.4. For the given problem, the most suitable training algorithm is the PPO algorithm, thoroughly explained in subsection 2.4.3.

3.4.1 Actor and critic

The PPO algorithm is implemented with an actor-critic framework, where two neural networks are used to represent the policy (actor) and value function (critic). Choosing an appropriate architecture for both the actor and critic networks is crucial for achieving stable and efficient learning. If the architecture is too simple, the policy may fail to converge due to an insufficient representational capacity. On the other hand, overly complex architectures risk overfitting and can lead to extended training times.
Since both the action and observation spaces in this application are continuous, it is beneficial for the policy network to incorporate some form of memory. This is due to optimal actions often depending on not only the current observation but also on past states and actions. To address this, recurrent architectures such as RNNs and LSTMs are explored in the actor network. These are implemented using the PyTorch library [43]. The critic network, by contrast, is implemented as a standard fully connected feedforward neural network (FFNN) and its structure is showcased in Table 3.4.
The networks used in the policy (actor) utilized ReLU activation functions in the hidden layers and a tanh activation function in the output layer. The tanh function ensures that the network outputs are bounded within the range of -1 to 1, which are then rescaled to match the appropriate action ranges. This choice simplifies the learning process by mitigating the need for precise weight initialization and enabling the agent to quickly discover meaningful control behaviors. For the value function,

ReLU activations are used in the hidden layers, with a linear output to produce scalar value estimates.
The critic network only had the purpose of accurately predicting the expected rewards. This is a much simpler task compared to the policy and has less impact on the overall training as long as performance is sufficient. It was concluded that a FFNN performed well enough to estimate the rewards, with the structure seen in Table 3.4.

Table 3.4: Critic neural network Architecture.

Critic type	Architecture	Trainable parameters	Activation Functions
FFNN	Input → Linear(256 → 256) → Linear(256 → 256) → Linear(256 →256) → Linear(256 →128) → Linear(128→Output size) → Output	109,825	ReLU

3.4.2 Algorithm optimization

For efficient training, several techniques were utilized to ensure stable and correct learning.

3.4.2.1 GAE

GAE is a method for calculating the advantage. By using GAE, an unbiased estimate can be created with lower variance. GAE was used to lower the temporal difference error by applying the estimate, with its dependencies, over the entire episode, weighing future rewards against current to see how the action affects later rewards. The advantage estimate for a timestep t in an episode with length n is then calculated as:

$$\hat{A}_t = \sum_{l=0}^{n-1-t} (\gamma\lambda)^l \delta_{t+l} \tag{3.22}$$

3.4.2.2 Minibatching

Minibatching was used to divide a batch of episodes into smaller subsets. Each subset is randomly sampled from the list of episodes present in the batch. When sampling, the selected episode is not removed from the list of available episodes, meaning that the same episode could occur in multiple minibatches. Minibatching was used to speed up the training and enhancing the policy's ability to generalize. By updating the policy from a random subset of trajectories, the network will both learn to distinguish the differences between the trajectories and generalize across trajectories. If one instead would perform one update for the entire batch, the network will create a solution which is too generic and is unable to perform well in all scenarios.

3.4.2.3 Entropy regularization

To provide a balance between exploration and exploitation of the agent, entropy regularization is used. Entropy refers to the uncertainty or randomness of the policy.

Entropy regularization is implemented by adding a penalty to the objective function, encouraging a higher entropy which helps the algorithm to explore and make less deterministic choices. The entropy for a given policy is defined as:

$$H(\pi_t) = -\sum_a \pi_t(a) log(\pi_t(a)) \tag{3.23}$$

where $\pi_t(a)$ is the probability to pick action a in at time t. The actor loss calculation is updated to include the entropy regularization term as well:

$$L(\Theta)_\pi = L(\Theta)_\pi - \lambda H(\pi_\Theta) \tag{3.24}$$

3.4.2.4 Learning rate decay

A dynamic learning rate is a common technique where the learning rate is changed throughout the training, usually by decaying the learning rate with the number of epochs or simulation time. Dynamic learning rate scheduling was implemented using a cosine decay function over the course of simulation time in the model, utilizing the built in learning rate scheduler in PyTorch.

3.4.2.5 Gradient clipping

To ensure that the loss calculated by the surrogate objective does not contain exceptionally large gradients, another clipping is employed on the loss. If the gradients of the loss are to large, it can lead to unstable training and hinder convergence. By clipping the gradient it mitigates the risk of exploding gradients in the actor and critic.

3.4.3 Algorithm

The implemented algorithm starts by performing a rollout over a batch, where the current policy and value function is used to collect actions, rewards, and values over several episodes. With the collected rewards and values, the advantage is calculated using GAE for the trajectories of the batch and the discounted rewards is calculated as:

$$R_t(\Theta) = \sum_{k=0}^{T-t-1} \gamma^k r_{t+k} \tag{3.25}$$

Discounted rewards are used to give the agent a sense of the relationship between current and future rewards. This encourages the agent to optimize for long-term benefits rather than focusing solely on immediate gains. After collecting data through rollouts, the learning phase begins. The algorithm evaluates the probability ratio between the action probabilities under the current policy and the old policy (used during data collection). The new policy is simulated by sampling actions from a distribution centered around the mean actions observed during the rollout. The actor network is updated by minimizing the clipped objective (3.26). To make the most of the collected data, the network is updated multiple times per batch. The number of these updates is treated as a hyperparameter, referred to as *updates per iteration*.

$$L_\pi(\Theta) = \frac{\sum_{t=0}^{T} \hat{E}_t[\min(E_t[r_t(\Theta)\hat{A}_t], \text{clip}(r_t(\Theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)]}{T} \tag{3.26}$$

The goal of the critic is to accurately predict the mean rewards and is updated using MSE (mean squared error) loss between the current values and the discounted rewards:

$$L_V(\Theta) = \frac{\sum_{t=0}^{T}(R_t(\Theta) - V_t(\Theta))^2}{T} \tag{3.27}$$

The following algorithm details the complete implementation of the PPO:

Algorithm 1 Proximal policy optimization (PPO) implementation

Require: $N \in \mathbb{N}$ (updates per iteration), $T \in \mathbb{N}$ (total timesteps), $M \in \mathbb{N}$ (mini-batch size)

Initialize: π (actor network parameters) and V (critic network parameters)
$t = 0$
while $t < T$ **do**
  $\Theta \leftarrow$ Perform rollout to collect trajectories from simulation model using the current policy π and value function V.
  $t \leftarrow$ Update t with elapsed simulation time in rollout
  $\hat{A}(\Theta) \leftarrow$ Calculate normalized advantage estimates using Θ
  $R(\Theta) \leftarrow$ Collect discounted rewards from Θ
  for N **do**
    Shuffle trajectories in Θ
    for M **do**
      D $\leftarrow$ Collect subset of trajectories form Θ
      $r(D) \leftarrow$ Evaluate new policy $\pi_{D_{new}}$ using V and calculate probability ratio against π_D
      $L_\pi \leftarrow$ Calculate policy loss using $r(D)$ and $\hat{A}(D)$.
      $L_V \leftarrow$ Calculate critic loss as MSE between $R(D)$ and $V(D)$
      $\pi \leftarrow$ Update policy using L_π.
      $V \leftarrow$ Update critic using L_V.
    end for
  end for
  $lr \leftarrow$ update learning rate based on $\frac{t}{T}$
end while

3.4.4 Reward function

For the agent to learn effectively, it's important that there is a clear and consistent link between what it observes, the actions it takes, and the rewards it receives. Using a hybrid environment helps simplify this learning process by narrowing the observation space to focus solely on information relevant to the hydraulic system with the ability to scale the complexity of the environment when required.
Given that the agent's primary objective is to track the desired cylinder velocities, the velocity tracking error becomes an important metric for evaluating performance.

This error is defined as the difference between the desired (preferred) and actual cylinder velocities:

$$\dot{\boldsymbol{l}}_{error} = \dot{\boldsymbol{l}}_{pref} - \dot{\boldsymbol{l}} = \begin{bmatrix} \dot{l}_{1_{pref}} - \dot{l}_1 \\ \dot{l}_{2_{pref}} - \dot{l}_2 \\ \dot{l}_{3_{pref}} - \dot{l}_3 \end{bmatrix} \tag{3.28}$$

The reward function itself is modular, composed of several sub-rewards designed to capture different aspects of task performance. This decomposition simplifies the tuning process, allowing each component to be scaled independently to ensure a balanced contribution to the overall reward signal.
To give the agent a clear bound on the rewards and stabilize the reward distribution, negative exponential was used for all reward components. This was used to bind rewards to a span of $0 < r_{(*)} < 1$, ensuring that rewards with high variance have less impact on training and ensuring all components are in the same range. If the component was used as a penalty for unwanted behavior it would be in the range $-1 \leq r_{(*)} \leq 0$.

3.4.4.1 Velocity reward

The first part of the reward function is only based on the error for the cylinder velocities to encourage the policy to follow the target velocities. The error is scaled with the k_1 parameter to create the wanted relationship between the reward and error:

$$\boldsymbol{r}_1 = \begin{bmatrix} r_{11} \\ r_{12} \\ r_{13} \end{bmatrix} = \begin{cases} e^{-k \cdot |\dot{\boldsymbol{l}}_{error}|}, & \text{raw error} \\ e^{-k \cdot (\dot{\boldsymbol{l}}_{error})^2}, & \text{squared error} \end{cases} \tag{3.29}$$

During the evaluation phase, two different formulations of this reward was tested. These differed in whether the velocity error, $\dot{l}_{error}$, was squared. In the initial phase of experimentation, the squared error was used for the reward, before comparing it with the reward utilizing the raw error.

3.4.4.2 Direction reward

To further reinforce the importance of changing the velocity in the right direction, cylinder acceleration $\ddot{l}$ is used along with the velocity error. When the velocity error is large, it is important for the agent to accelerate in the correct direction. However, as it approaches the target, the importance of the acceleration direction decreases. The desired direction of acceleration will always be the same as the direction as the velocity error. The agent is penalized by using the magnitude of the error and checking if the error and acceleration has the same sign:

$$r_2 = (1 - \boldsymbol{r_1}) \cdot \frac{(sign(\dot{\boldsymbol{l}}_{error} \circ \ddot{\boldsymbol{l}}) - 1)}{2} \tag{3.30}$$

If $sign(\ddot{l}) = sign(\dot{l}_{error})$ the agent will be rewarded based on the magnitude of $\dot{l}_{error}$, and penalized by the magnitude if $sign(\ddot{l}) \neq sign(\dot{l}_{error})$. The reward component is used as a penalty and is only applied when the velocity changes in the wrong

direction. To penalize instead of reward, the term $(1 - \boldsymbol{r}_1)$ is added to make the values negative, in the range $-1 \leq r_2 \leq 0$.

3.4.4.3 Fuel and engine penalty

To achieve a lower fuel consumption, two different components of the reward function included engine parameters, specifically the engine torque τ_{engine} and the fuel consumption, $fuel_{engine}$. The model is able to track the fuel consumption of the machine in any given moment, making it a good metric to use as one of the goals of the project is to make the machine as fuel efficient as possible. The fuel consumption is not passed to the model as an observation, but by rewarding it when the consumption is low, the model can learn to behave differently. The reward is calculated as

$$r_3 = e^{-\frac{fuel_{engine}}{20{,}000}} \tag{3.31}$$

where the fuel is normalized with a chosen scaling and measured in milligrams per second. The other parameters used in the reward calculations are already normalized as they are part of observations. This means that the fuel parameter must be scaled while calculating the reward. The other component only used the engine torque with a similar setup

$$r_4 = e^{-\tau_{engine}} \tag{3.32}$$

3.4.4.4 Total reward

The total reward which the PPO tries to maximize consists of each sub-reward multiplied with a scaling factor $\lambda_{(*)}$ and then added together:

$$R = \boldsymbol{\lambda_1} \cdot \boldsymbol{r}_1 + \lambda_2 r_2 + \lambda_3 r_3 + \lambda_4 r_4 \tag{3.33}$$

$\boldsymbol{\lambda_1}$ is a vector representing scaling for the velocity error of each actuator:

$$\boldsymbol{\lambda}_1 = \begin{bmatrix} \lambda_{11} \\ \lambda_{12} \\ \lambda_{13} \end{bmatrix} \tag{3.34}$$

3.5 Training the hydraulic controller

The hydraulic controller was trained through iterative testing of training parameters, network architectures, and reward structures. A central focus was placed on methodically exploring the impact of key hyperparameters, while holding others constant to isolate their influence. Parameters such as batch size, episode length, and reward scaling were individually evaluated to identify optimal configurations. This controlled evaluation allowed for a clearer understanding of their respective contributions to learning stability and control precision.
Certain hyperparameters, such as the learning rate, PPO clip, GAE factor, discount factor, entropy coefficient, and gradient clipping threshold were not tested due to time constraints and were instead chosen based on established best practices and literature guidance.

A key aspect of the controller is pump control, which is important for achieving fuel efficiency but introduces additional complexity for the learning agent. Instead of simply controlling valve positions to drive actuators, the agent must also learn to regulate pump flow, ensuring that sufficient hydraulic fluid is available without generating excess flow that would lead to inefficient bypassing through valves. This dual responsibility forces the agent to balance function-specific flow demands with overall system efficiency, making learning more difficult but also more realistic.

A complete list of parameters used in training can be seen below. Parameters where multiple values was tested are marked as **Evaluated** in Table 3.5.

Table 3.5: Parameters used during training, with parameters to be evaluated marked.

Parameter	**Value**
Simulation time	50,000 seconds
Actor	**Evaluated**
Critic	FFNN
Learning rate	0.001
Discount factor (γ)	0.95
GAE factor	0.95
PPO Clip	0.2
Gradient Clip	0.5
Entropy coefficient	0.005
Decaying learning rate	True
Batch size	**Evaluated**
Episode length	**Evaluated**
Mini-batch size	4
Updates per iteration	**Evaluated**
Entropy Coefficient	0.005
Command sample frequency	100 Hz
Max cylinder velocity	0.4 m/s
Velocity Error scaling $\dot{l}_{error}$	**Evaluated**
Reward components	$\lambda_{(*)}$ = **Evaluated**, k = **Evaluated**

3.5.1 Comparing parameters

Some parameter choices could greatly impact the training and performance of the agent. This section shows the performance of the training when using different parameter choices. Due to time aspects in regards to the slow training, not all parameters were evaluated and focus was be on the parameters which was believed to have a major impact on training outcome. All experiments were conducted on 50,000 seconds in simulation time, and only one parameter was evaluated at a time, leaving the others constant. The performance was evaluated using the sum of cylinder velocity errors.

3.5.1.1 Episode length

The episode length defines the duration for which the agent interacts with the environment before a reset is triggered. A balance is required between achieving high performance and maintaining reasonably short episodes. If the episodes are excessively long, the algorithm observes fewer trajectories over the same number of time steps, resulting in reduced data diversity. Therefore, shorter episodes are preferred once sufficient performance is achieved.

Due to the training setup, where a constant target velocity is set for each episode, overly long episodes proved to be suboptimal. The agent tended to reach the endpoint of the cylinder rapidly and remain stationary thereafter. Consequently, training was conducted using episode lengths of 1, 2, 3, and 4 seconds. To ensure that the network updates remained proportionate across different episode durations, the number of updates per iteration was adjusted accordingly.

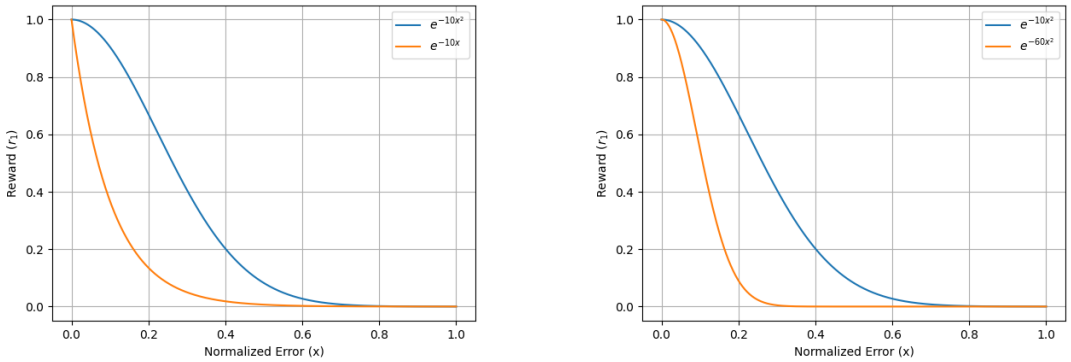
3.5.1.2 Batch size

Batch size can have a large impact on the learning and convergence of the algorithm. By having a small batch size, the system will not learn because of the frequent updates to a small subset of trajectories; in other words it will overfit when only updating on a few trajectories and then update completely different when encountering new trajectories. This makes training unstable with high fluctuation in reward and performance between batches. If the batch size on the other hand is large, the algorithm will generalize over all trajectories in the batch and not being sufficiently good on any specific trajectory.

By testing different batch sizes, the aim is to find the sweet spot where the algorithm does not get unstable in training or generalize too much. Batch sizes 16, 32, 64 and 128 were tested.

3.5.1.3 Error/reward scaling

Scaling of the rewards can greatly impact training, especially in the context of trying to minimize the velocity error. Two velocity errors was proposed for the velocity reward, either the raw error or the squared error given by Equations 3.29. The relationship between the error and velocity reward will be affected by type of error used and the scaling, k, of the errors.



(a) Raw and squared errors relationship to the velocity reward.

(b) Changes in reward characteristics due to scaling of the error.

Figure 3.7: Comparison of error type and scaling.

A comparison between the raw and squared error is showed in Figure 3.7a. When applying the squared error, a more linear slope is created compared to the exponential growth of the raw error. This means that the derivative of the reward will be high for the raw error, even if the error is close to 0. This encourages the model to keep improving even if it has a small error.
Figure 3.7b on the other hand shows how the reward is shifted in x-direction based on the k parameter. If the scaling is increased, the curve will be pushed toward the y-axis, decreasing the rewards for large errors and increasing the derivative of the slope, making smaller changes in error in certain spans have a bigger impact on the reward.
The errors was compared with different scalings to see how learning is affected in training, depending on the type and scale of the error. This was done to see if the algorithm can be pushed to further minimize the error by scaling the rewards differently. For the squared error, k was set to 10, 60, 120 and for the raw error k was set to 15 and 25.

3.5.1.4 Policy network

The size and layers of the policy network can affect the computation and backpropagation time when training. By adding more layers and connections, the complexity of the policy is expanded and it can learn more advanced relationships. One major aspect of this project is the ability to be aware of what has happened previously, through time-dependency of the vale setpoints. This can heavily affect the performance of the policy, depending on if the algorithm learns how the valve-setpoint affects the cylinder velocities over several inputs. By testing different network architectures for the policy, an optimal network structure can be chosen which is able to capture the characteristics of the cylinder dynamics based on current and previous inputs and be efficient. Due to time-aspects, all possible configurations are not able to be tested. Instead, the number of layers and number of neurons in each layer is not investigated, but rather different network types, such as RNN, LSTM and FFNN and is showcased in Table 3.6.

Table 3.6: Evaluated neural network architectures for the policy.

Policy Class	Architecture	Trainable Parameters	Activation Functions
FFNN	Input → Linear(256 → 256) → Linear(256 → 256) → Linear(256 →256) → Linear(256 →128) → Linear(128→Output size) → Output	178,608	ReLU, tanh (output)
RNN	Input → RNN(Input → 256) → Linear(Input size → 256) → Linear(256 → 256) → Linear(256 →256) → Linear(256 →128) → Linear(128→Output size) → Output	310,192	ReLU, tanh (output)
LSTM	Input → LSTM(Input → 256) → Linear(256 → 256) → Linear(256 → 256) → Linear(256 →256) → Linear(256 →128) → Linear(128→Output size) → Output	540,592	ReLU, tanh (output)

3.5.1.5 Fuel penalty

To examine the possibilities to make the policy more fuel efficient, the reward components for engine (3.32) and fuel penalty (3.31) were evaluated. This was evaluated to examine how adding a penalty would affect the overall performance of the system, and if the fuel efficiency could be optimized without sacrificing precision in the system. The reward functions compared was the base reward $r_1 + r_2$, to the rewards with either the added engine penalty r_3 or fuel penalty r_4. The evaluation was carried out by comparing mean fuel consumption in an episode.

3.5.2 System complexity

After selecting which final hyperparameters to use, the focus was shifted towards evaluating the complete system and increasing the system complexity. Policies for single actuators were investigated as well as a combined actuator policy.

3.5.2.1 Single actuator policy

The agent was first trained in simplified environments, where each model controlled only one actuator at a time. This setup aimed to determine the upper bound of control precision under minimal complexity. While this scenario is unrealistic for real excavator operations, it provided valuable insight for evaluation purposes.
To isolate control, both observations and actions were limited to the selected actuator, and the reward was based solely on its velocity error. The valves in the two remaining actuators remained closed with target velocities fixed at zero, while the controlled actuator received a random but constant velocity command throughout each episode.

3.5.2.2 Combined actuator policy

To compare performance between isolated and combined control strategies, the agent was next trained to control all actuators simultaneously. The same set of optimized hyperparameters, identified during hyperparameter evaluation training was reused, but the simulation time was doubled to ensure model convergence, as some previous runs suggested incomplete learning within the shorter time frame.
In this configuration, the agent received full observations and produced full action outputs, targeting simultaneous control of all cylinders. This setup more closely

reflects the real operational environment of an excavator and allows for interaction between actuators and shared dynamics within the system.

Training a unified policy for all actuators, including the hydraulic pumps, is important for capturing system wide dependencies. Coordinated control enables the agent to learn how actuators influence one another during complex tasks, interactions that isolated training would fail to capture. This approach also serves as a foundation for implementing grading maneuvers later.

3.5.3 Final policy

During the hyperparameter selection and evaluation of the combined actuator policy, it was observed that the boom cylinder had major issues while extending and would oscillate around 0 m/s. As it is crucial to be able to lift the boom, and inherently lift all links, more attention was given to the boom by increasing $\boldsymbol{\lambda}_{11}$.

3.5.3.1 Varying target velocities

To further enhance the model, the final policy was retrained with the best actor and a significantly lower learning rate in the hopes that some finetuning could be done to the model. The model was finetuned by introducing an environment where the target velocities would vary during the episode. The target velocity was perturbed by a small increment, sampled uniformly from the interval $[0, 0.0025]$ m/s at each control decision. This continuous adjustment introduced gradual changes to the control target, simulating more realistic and challenging conditions. The direction of the perturbation was decided at the start of the episode by randomly sampling of a binary number. This ensured that the model would have a constant velocity direction.

Furthermore, the behavior of the model near the mechanical limits of cylinder extension or retraction was revised. In the initial implementation, when a cylinder reached its physical limit, the corresponding target velocity was simply set to zero, effectively halting motion. This approach limited the learning opportunities near joint boundaries. To address this, a new strategy was introduced: upon reaching an endpoint, a new target velocity was sampled with a sign opposite to the previous direction, using a uniform distribution ranging from 0 to the maximum allowable cylinder velocity. This enabled the model to learn to reverse motion at mechanical limits rather than simply stopping, thereby increasing both control complexity and learning opportunities near boundary conditions.

3.6 The complete system

The complete system (Figure 3.8) utilized to complete the task of grading consists of four main components, the PID-controller, mechanical backwards calculations, the hydraulic controller and the simulation model.

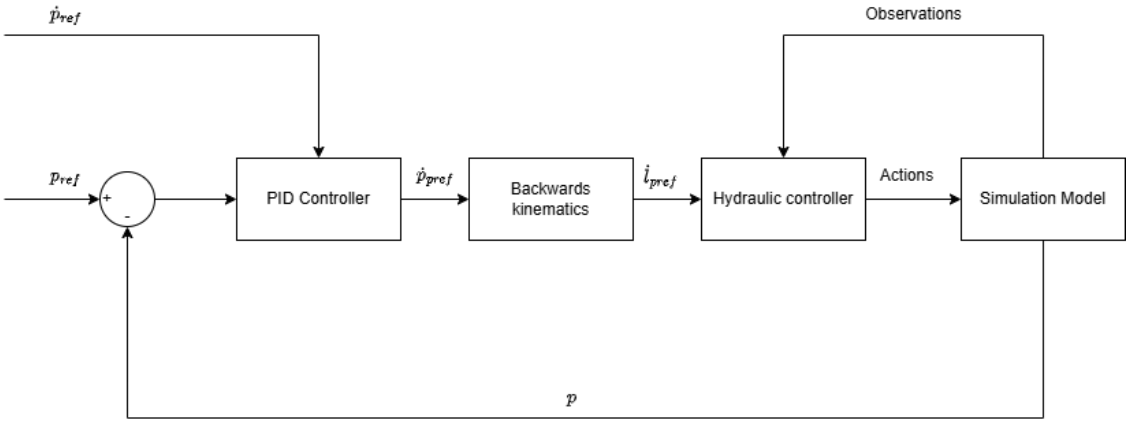


Figure 3.8: Flow of the system used to perform the task of grading.

The system flow for performing the task of grading on a given trajectory can be broken down to the following parts: The current target bucket tip pose on the trajectory p_{ref} is subtracted from the actual bucket pose p and the error is sent to the PID controller along with a target velocity $\dot{p}_{ref}$ for the bucket tip. The PID calculates the wanted change in pose $\dot{p}_{pref}$, and through the backwards kinematics (subsection 3.1.2) the wanted change in cylinder velocities $\dot{l}_{pref}$ is derived. The cylinder velocities are then inputted to the hydraulic controller along with the observations from the simulation model (Table 3.1). The controller derives the actions (currents) for controlling the valves and the signals are sent to the simulation model after being rescaled to the proper range. The model performs a step with the new input values and then the loop starts over.

3.7 Evaluation

Evaluation was performed in different ways, depending on what the goal of the test was. The metrics generated during training could be investigated, such as how the raw velocity difference on the different cylinders varied with time and configurations.

3.7.1 Hydraulic controller

During evaluation and result generation, a predefined seed was used to ensure that the model was initialized in the same way and have the same target velocities, ensuring that a fair comparison between the evaluated models could be achieved. The hydraulic controller was evaluated in two different ways, either by investigating performance during training, or performance during evaluation. Evaluation was carried out by initializing the model at a fixed starting location and running the environment for 3 seconds.

3.7.2 Grading performance

After evaluating the hydraulic controller, trained by the PPO algorithm, the controller was used during grading evaluation together with the positional controller

for a complete system test. The system was evaluated on different trajectories in the excavator's workspace, with both flat and sloping trajectories, as illustrated in Figure 3.9.

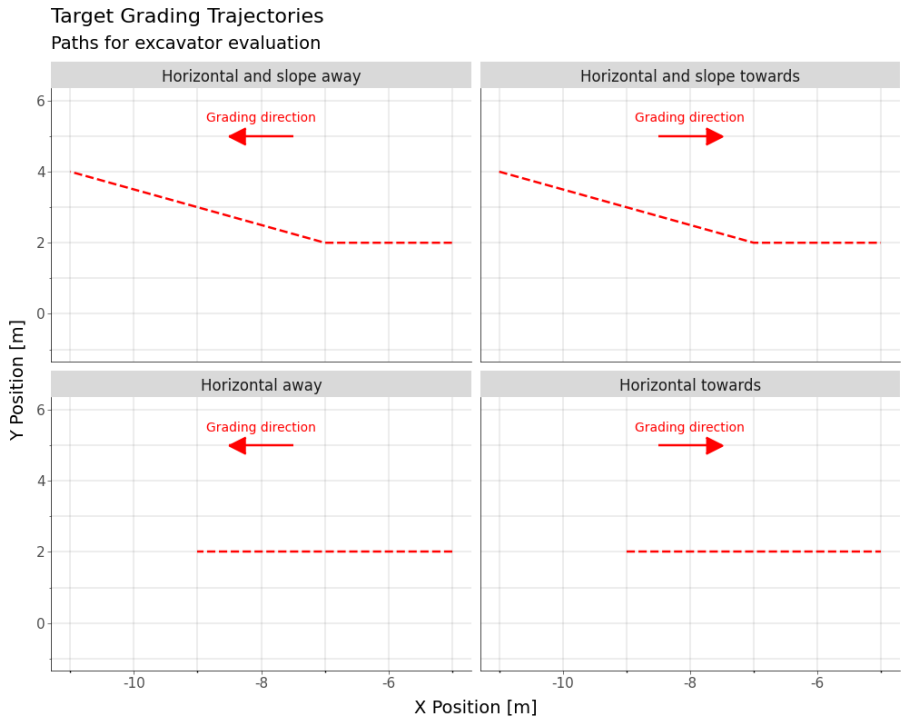


Figure 3.9: Grading trajectories used for evaluation.

The magnitude of the reference pose velocity $|\dot{p}_{ref}| = 0.5$, was constant during the test episodes while the target angle of the bucket was changed during the episode, based on horizontal distance of the tip of the bucket to the machine. The angle was scaled between the maximum and minimum distance of the workspace into the range of $\theta_3 \in [-\frac{\pi}{2}, \frac{\pi}{2}]$. This was done to ensure that the excavator could reach as far as possible in the workspace. However, to ensure that the grading is not impacted by the target angle, a minimum value which is based on the angle of the slope is introduced to ensure that other parts of the machine would not cross the target trajectory.

$$\theta_3 = min(\theta_{slope}, \theta_3) \tag{3.35}$$

Grading performance was measured by the smallest distance from the bucket tip to the given trajectory, with the sign of the error indicated if the bucket tip was above or below the target trajectory. Together with the performance, the time it took to perform the grading was also taken into account.

4

Results

The following chapter will detail the results of parameter evaluation when training the different policies. The resulting performance is then displayed by showcasing the velocity tracking and grading precision for the trained policies.

4.1 Training parameters

The core architecture of the training model remained unchanged throughout the evaluation of the different parameters. However, minor modifications were introduced to the environment between individual training runs; these changes will be detailed in the corresponding chapter.
The initial values for the evaluated parameter are presented in Table 3.5:

Table 4.1: Initial evaluated parameters used during training

Parameter	**Value**
Actor	LSTM
Batch size	16
Episode length	2
Velocity Error scaling $\dot{l}_{error}$	Squared
Reward components	$\lambda_1 = [1,1,1]^T$, $\lambda_2 = 1$, $\lambda_3 = 0$, $\lambda_4 = 0$, $k = 10$

4.1.1 Episode length

Table 4.2 outlines the configurations used when evaluating the episode length.

Table 4.2: Training configurations with varied episode time and update parameters

Configuration	**Episode Length (s)**	**Updates per Iteration**
1	1	2
2	2	4
3	3	6
4	4	8

Following training with the earlier mentioned configurations, one can see that an episode length of at least two seconds was necessary to achieve similar learning outcomes, as illustrated in Figure 4.1. No substantial performance differences were observed between episode lengths of 2, 3, or 4 seconds. As a result, an episode length of three seconds was selected for subsequent experiments.

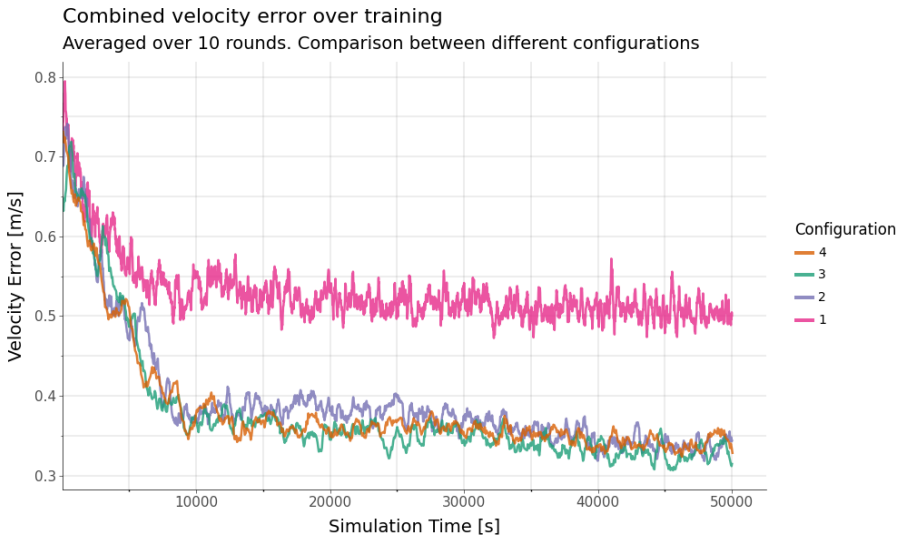


Figure 4.1: Rolling average of the combined velocity error for different episode lengths.

4.1.2 Batch size

Following the last experiment, the episode time is set to 3 seconds and 4 different batch sizes was tested. The sizes were 16, 32, 64 and 128. From Figure 4.2 one can see that higher batch size means more stable training, but in regards to simulation steps it takes longer for the model to converge.

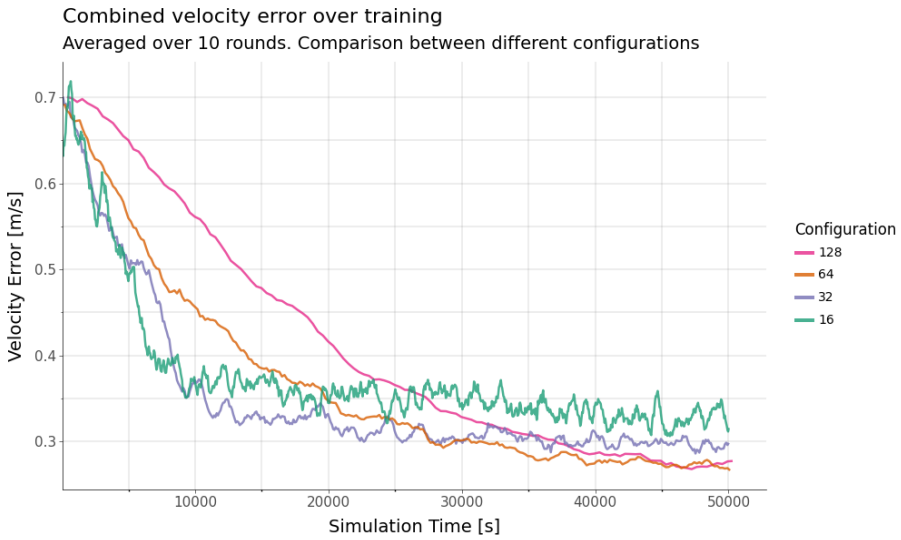


Figure 4.2: Rolling average of the velocity tracking error per simulation step for different batch sizes.

Another interesting aspect can be seen in Figure 4.3 where the number of updates of the network is significantly lower the higher the batch size. This would be more important if the simulation model wasn't the bottleneck in the system, because we would learn more with less backpropagation. After investigating the results, it was decided that a batch size of 128 would be used for future training evaluations.

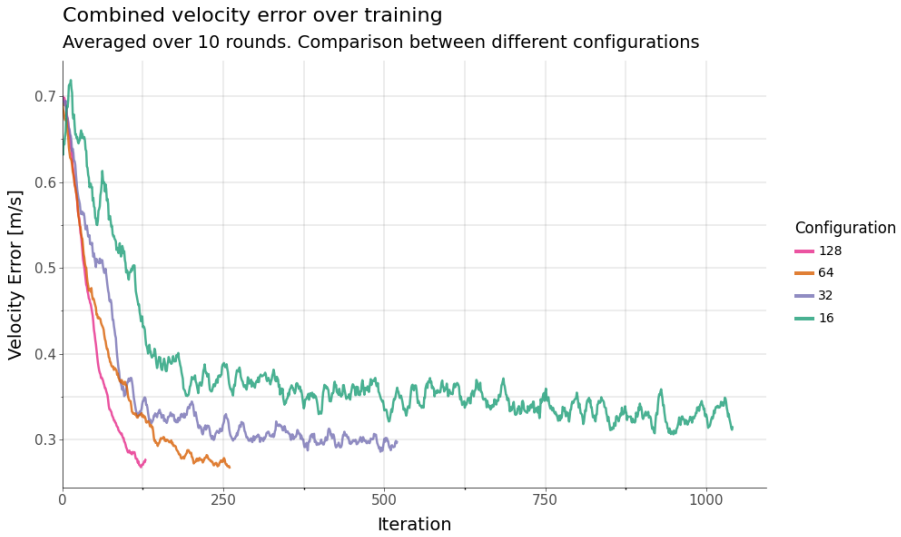


Figure 4.3: Rolling average of the cylinder velocity error per epoch for different batch sizes.

4.1.3 Reward scaling

Different reward scalings was tested, as showcased in Table 4.3.

Table 4.3: Training configurations with varying reward function and scaling

Configuration	Error	$\boldsymbol{\lambda}_1$	λ_2	λ_3	λ_4	k
1	Squared	$[1,1,1]^T$	1	0	0	10
2	Raw	$[1,1,1]^T$	1	0	0	15
3	Raw	$[1,1,1]^T$	1	0	0	25
4	Squared	$[1,1,1]^T$	1	0	0	60
5	Squared	$[1,1,1]^T$	1	0	0	120

From the previous experiments the batch size and episode iteration was tested and selected to 128 and 3 respectively for all training evaluations.

From the training results one can see that the original reward function (Configuration 1) starts to flatten while configuration 2 and 3 still continues to decrease, which is shown in Figure 4.4. To provide a conclusive result the training time would've needed to be longer, but due to time constraints this was not possible.

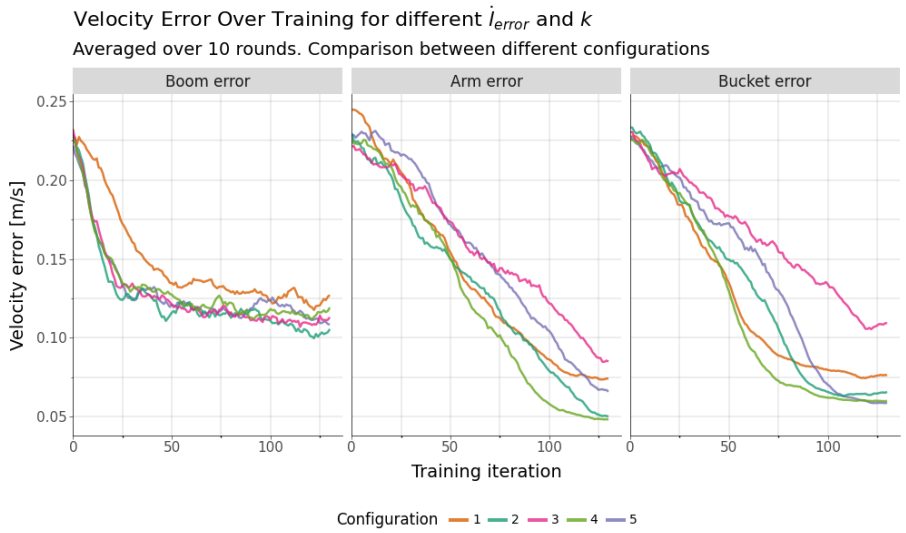


Figure 4.4: Cylinder velocity error for different reward scalings and error calculations.

4.1.4 Policy network

Three different network configurations was tested, with each network being of a different type. The networks tested was an RNN, LSTM and FFNN. Figure 4.5 shows that the LSTM has the best tracking performance and should be chosen as the final network. RNN, which LSTM is based on, did not show a similar performance while the FFNN without time dependency performed the worst.

Figure 4.5: Combined velocity tracking error during training for different network configurations.

4.1.5 Fuel penalty

To test the ability of the model to optimize energy consumption, the fuel and engine penalty was added to the reward function as separate components, with one being active for each run. The previous experiment showed that the raw error calculations together with $k = 15$ was the best setup and was used as a baseline in this experiment. Configurations tested can be seen in Table 4.4

Table 4.4: Training configurations with engine penalty and fuel penalty activated in separate configurations

Configuration	**Error**	$\boldsymbol{\lambda}_1$	λ_2	λ_3	λ_4	k
Fuel penalty	Raw	$[1, 1, 1]^T$	1	1	0	15
Engine penalty	Raw	$[1, 1, 1]^T$	1	0	1	15
Baseline	Raw	$[1, 1, 1]^T$	1	0	0	15

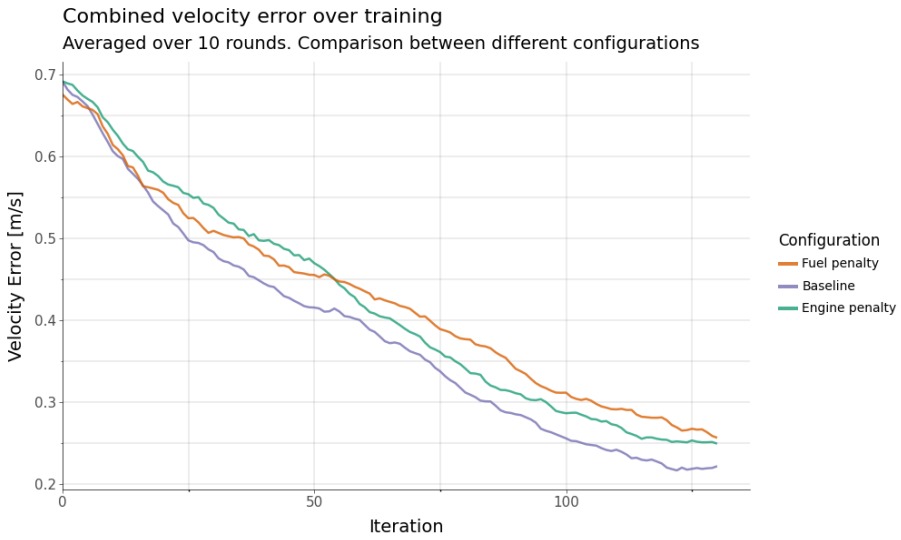


Figure 4.6: Combined velocity error for all cylinders with different reward components enabled

From Figure 4.6 minor differences between the baseline and the two evaluated policies can be observed from the combined velocity error, with the baseline having the best performance. During evaluation the differences between the fuel consumption are minimal between the fuel reward and baseline. The engine reward had the highest fuel consumption, as seen in Figure 4.7.

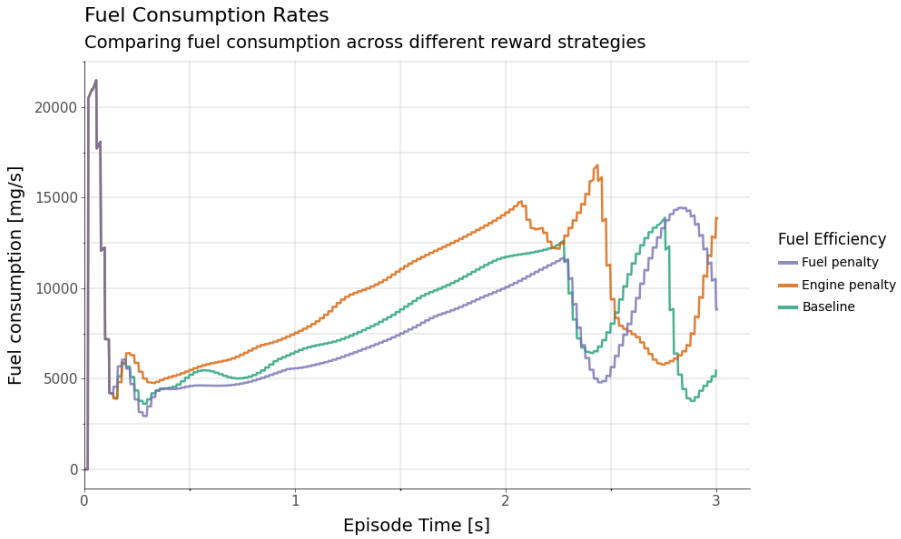


Figure 4.7: Fuel consumption during one evaluation episode for policies with different reward components enabled.

4.2 Hydraulic control

The three single-cylinder policies were evaluated against a policy where all cylinders were controlled simultaneously. The separate policies showed superior performance

in raw velocity tracking errors for each cylinder. Each of the links achieved an velocity error lower than 0.05 m/s, compared to the combined control policy where only the arm achieved that precision. The highest error measured during training was the boom cylinder error for the combined policy. This is showcased in Figure 4.8.

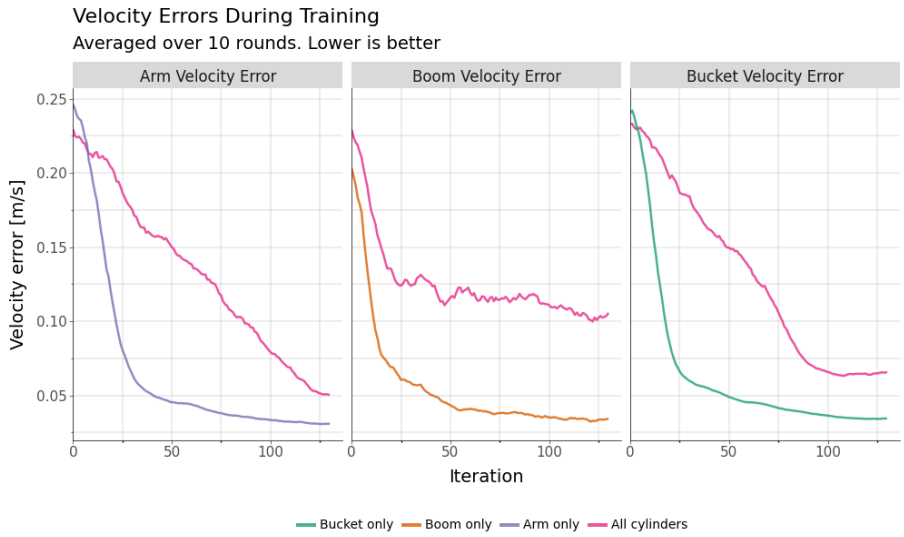


Figure 4.8: Comparison of velocity errors for single-cylinder control, between the policy for all cylinder control and the separate cylinder control.

4.2.1 Velocity tracking

The separate cylinder policies were then evaluated during an episode of 3 seconds against the model with full cylinder control. To represent a fair comparison between the two policies, the combined policy set the wanted velocity of the two other cylinders not relevant for the corresponding evaluation to 0.

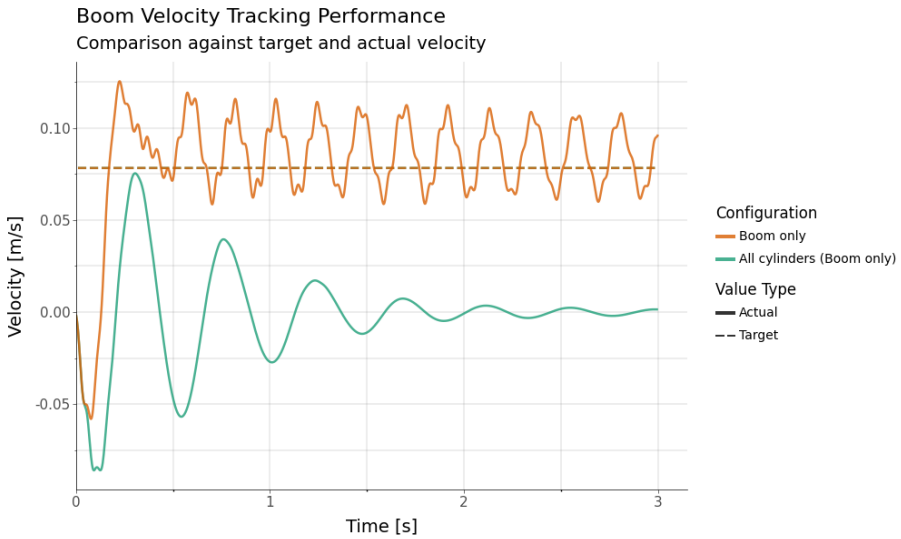


Figure 4.9: Velocity tracking for the boom cylinder over an episode, for separate boom and combined policy.

From Figure 4.9 it is clear that the boom policy is able to lift the boom and oscillate around the target velocity while the combined actuator policy is not able to lift the boom and instead oscillates around 0 m/s.

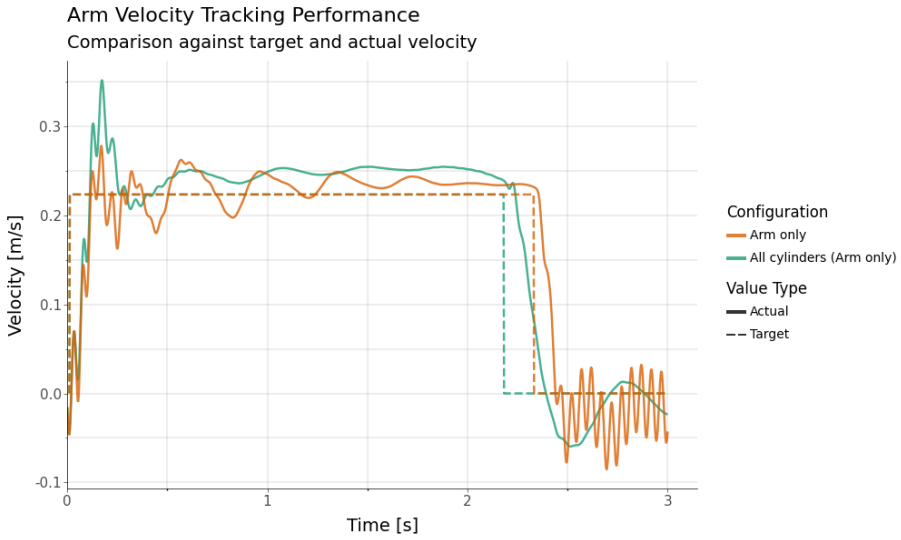


Figure 4.10: Velocity tracking for arm cylinder over an episode, for separate arm and combined policy.

Showcased in Figure 4.10, the wanted arm cylinder velocity is followed by both policies. Both the policies hit the maximum extension of the cylinder at different times, indicated when the target velocity drops to 0. After hitting the maximum extension, the single actuator policy oscillates with a higher amplitude and higher frequency compared to the combined policy.

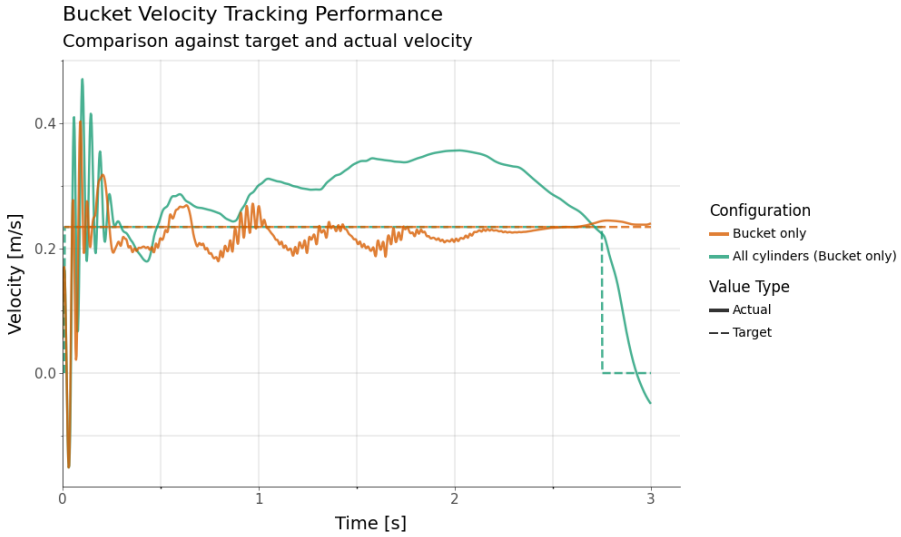


Figure 4.11: Velocity tracking for the bucket cylinder over an episode, for separate bucket and combined policy.

From Figure 4.11 one can see that the bucket showcases major oscillations in the

transient beginning of the episode. However, both policies are able to cancel out the oscillations to some extent. The single actuator policy on the bucket is able to accurately track the wanted velocity with minor deviations and small oscillations. The combined policy shows issues with tracking the wanted velocity and hits the maximum extension at around 2.8 seconds.

4.2.2 Oscillatory behavior

To compare oscillations in the system, the single actuator policy on the arm was compared against the combined policy and the velocity dynamics of all links was observed and is showcased in Figure 4.12.

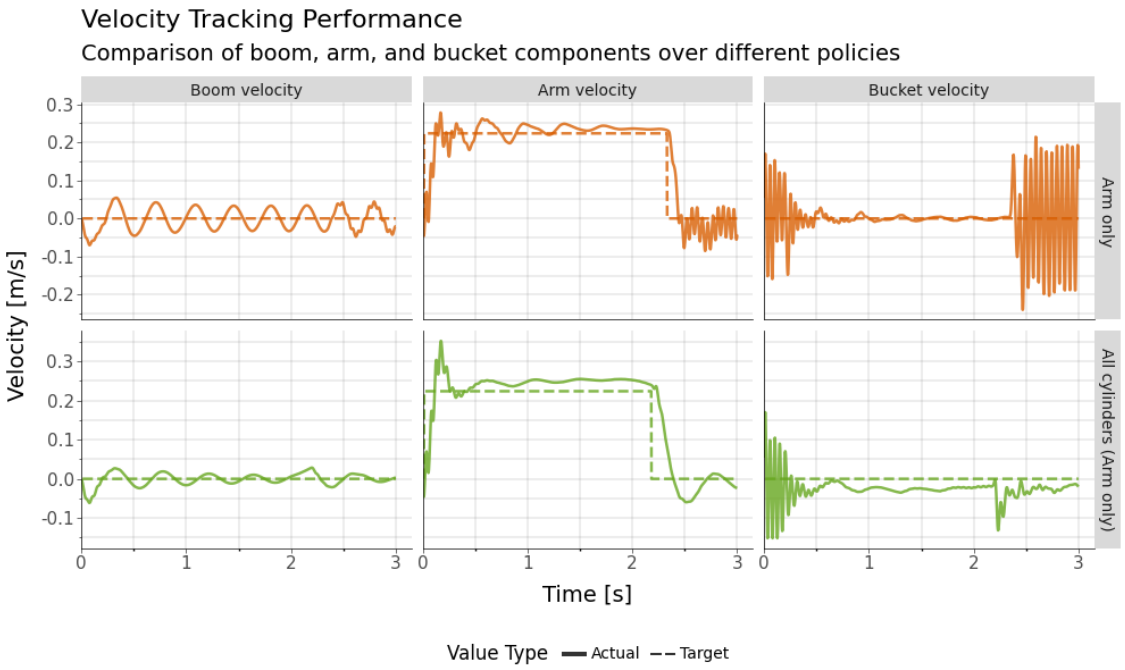


Figure 4.12: Velocity tracking for all cylinders with a wanted velocity $\neq 0$ m/s for the arm cylinder.

The separate policy exhibit oscillations in all links, with the bucket showing the largest magnitude and highest frequency of the oscillations. The boom velocity oscillates throughout the episode, but for the combined policy the oscillations are hardly noticeable. The combined policy is able to mitigate the oscillations since the policy is able to control the valves in the boom, even if the wanted velocity is set to 0. The highest magnitude of oscillations can be seen for the bucket when the arm reaches its cylinder limit, when the target velocity goes to 0. For the separate arm policy this creates large oscillations in the bucket which continues for the reminder of the episode. The combined policy hardly oscillates when the arm reaches its limit since it is able to compensate for the change in dynamics.

4.2.3 Final policy

The final policy used for grading was trained with the a larger velocity reward for the boom cylinder, to encourage the policy to learn both extension and retraction of the boom. The policy was trained for 100,000 simulation seconds. Table 4.5 outlines the parameters used.

Table 4.5: Evaluated parameters used during the final training.

Parameter	Value
Actor	LSTM
Batch size	128
Episode length	3
Velocity Error scaling $\dot{l}_{error}$	Raw
Reward components	$\boldsymbol{\lambda_1} = [2, 1, 1]^T$, $\lambda_2 = 1$, $\lambda_3 = 0$, $\lambda_4 = 0$, $k = 15$

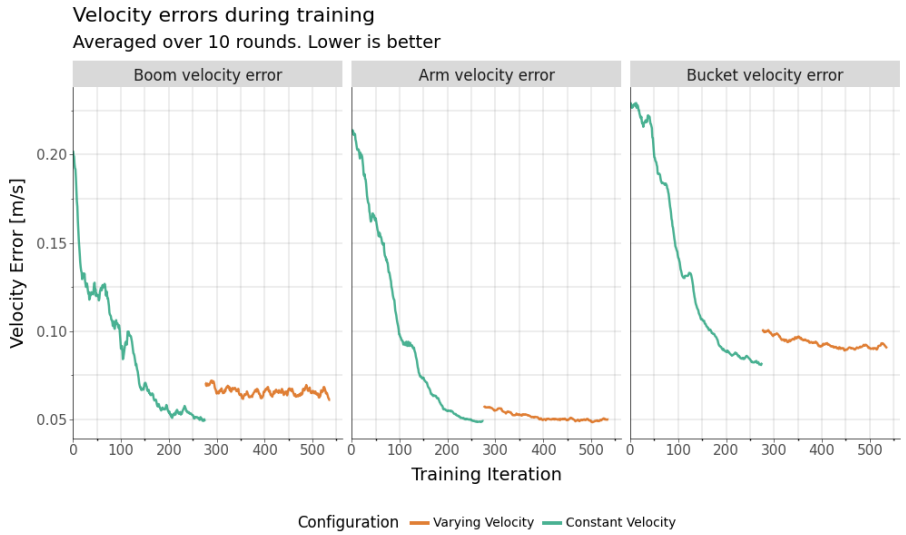


Figure 4.13: Cylinder velocity errors during training for the final policy (constant velocity) and varying velocity.

From Figure 4.13 it is clear that the velocity error for the boom cylinder has decreased significantly compared to previous experiments. However, the bucket cylinder performed worse than previous experiments, indicating a tradeoff between boom and bucket accuracy.
After training the final policy, the actor and critic were trained for another 100,000 simulation seconds with varying velocities throughout the episodes. The second training started with a significantly lower learning rate of $\lambda = 0.0005$. As Figure 4.13 illustrates, an small initial reduction in performance is observed due to the increased complexity of the task. During training, no significant improvement or decrease in performance was observed, indicating that the policy is able to adapt to the more complex environment.

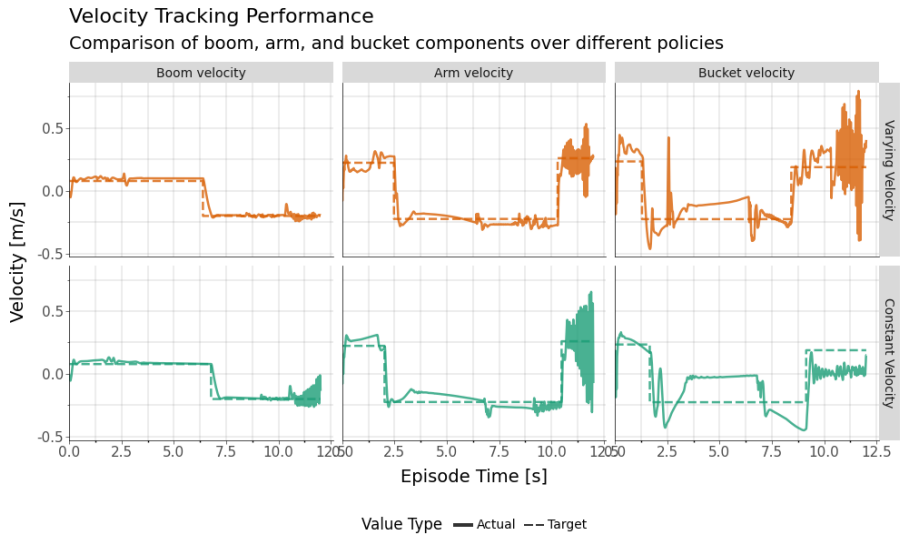


Figure 4.14: Evaluation of velocity tracking performance on all cylinders between the final policy (constant velocity) and a varying velocity policy.

When evaluating the velocity tracking performance on an episode, no major differences can be observed between the two different policies, as can be seen in Figure 4.14. Both policies manages to track the wanted boom and arm velocity accurately, while the bucket velocity performs worse. The varying velocity policy is able to mitigate the oscillations occurring in the boom cylinder when the arm and bucket reaches their end joints better than the constant velocity policy. However, the opposite is true for the bucket velocity, with the varying velocity policy showing major oscillations towards the end of the episode.

4.3 Grading performance

After completing the hyperparameter evaluation, the model was trained for 100,000 simulation seconds. The varying velocity configuration was applied after training this model for another 100,000 simulation seconds, with a significantly lower learning rate. The results showcased in this section only highlights trajectories where the grading direction is away from the machine. Results for the opposite direction has also been obtained and can be seen in Appendix C.

4.3.1 Sloping trajectory

The systen was evaluated on a combined horizontal and sloping trajectory, shown in Figure 4.15.

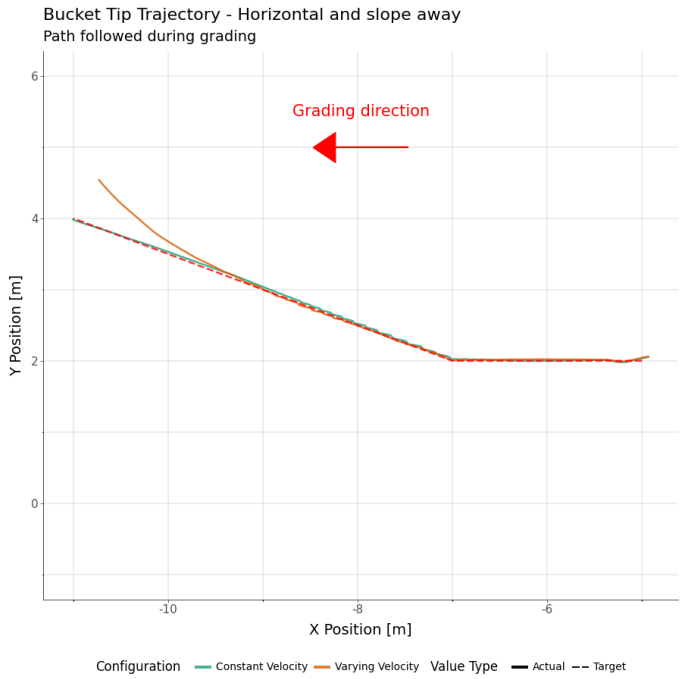


Figure 4.15: Grading performance on a sloping trajectory, with the start of grading to the right.

From Figure 4.15 one can see that both models perform well on the chosen grading task, with the largest error being around 5 centimeters for the constant velocity experiment, showcased in Figure 4.16. The varying velocity experiment is unable to lower the boom at the end of the target trajectory which results in a bad tracking performance at the end, even if it is relatively similar during the episode.

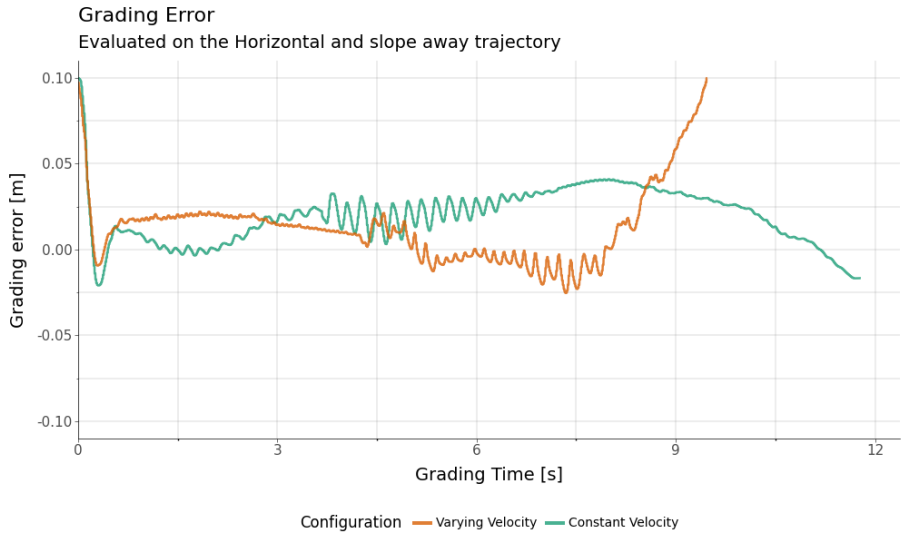


Figure 4.16: Grading error during evaluation of the sloping trajectory.

Both experiments show some minor oscillatory behavior. However, the oscillations are small and can be neglected when evaluating the grading performance as a whole.

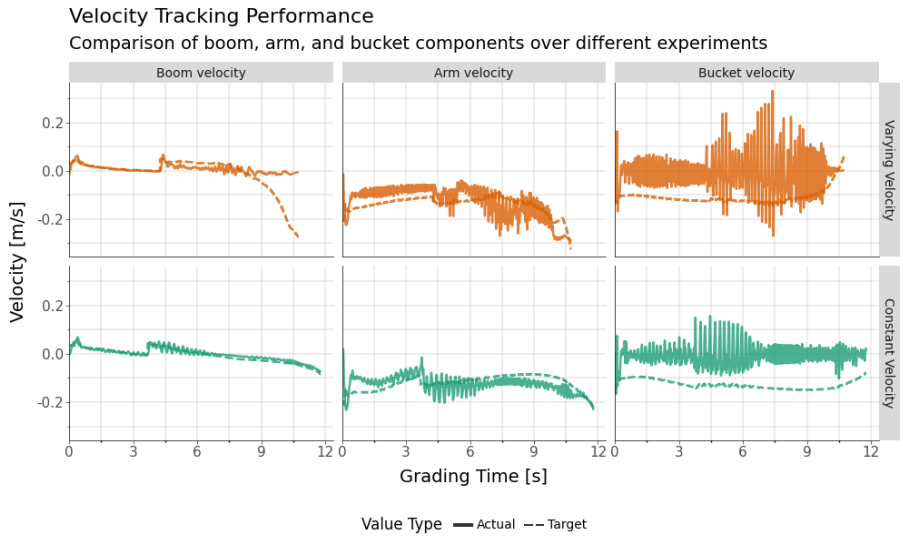


Figure 4.17: Cylinder velocities during grading evaluation of the sloping trajectory.

From Figure 4.17 one can see that the PID-Controller outputs reasonable velocity demands, increasing the boom velocity demand when the hydraulic controller is unable to follow the wanted velocity. Both the arm and the bucket shows oscillatory behavior and where the bucket has a significantly higher magnitude. One can also see that the wanted bucket velocity is not accurately met for both experiments, while the arm shows minor deviations.

4.3.2 Horizontal trajectory

Grading was also performed on a horizontal trajectory, shown in Figure 4.18.

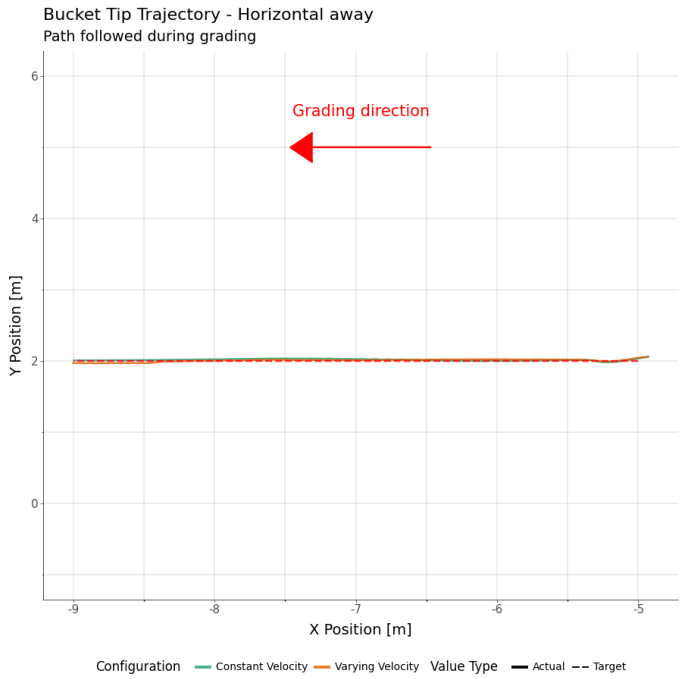


Figure 4.18: Grading performance on a flat trajectory, with the start of grading to the right.

From Figure 4.18 one can see that both policies perform well when grading on a flat trajectory. The largest error for the constant velocity policy was around 3 centimeters and -3 centimeters for the policy trained with varying velocities, as seen in Figure 4.19. From the figure it is also clear that the policy trained on a varying velocity executed the task faster than the policy trained on a constant velocity.

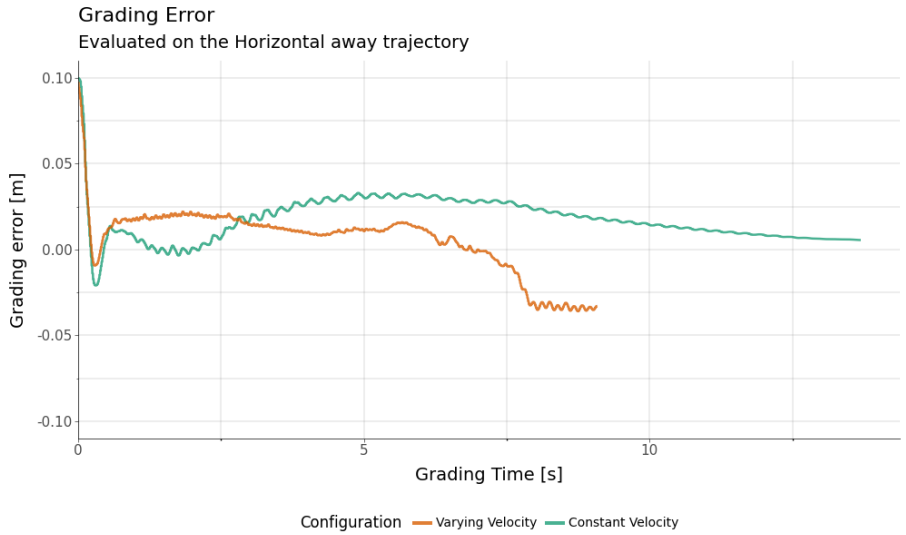


Figure 4.19: Grading error during evaluation of the flat trajectory.

Figure 4.20 shows that the velocity tracking for both policies performed well for the boom and arm. For the bucket velocity, the error was large for both models, even

though both policies performed well on the grading. The varying velocity policy displayed a larger magnitude for oscillatory behavior than the constant velocity model.

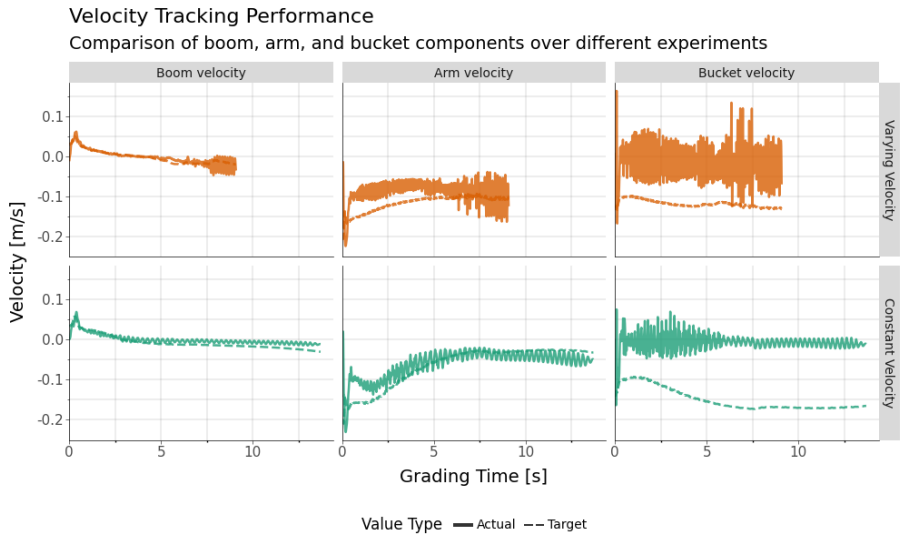


Figure 4.20: Cylinder velocities during grading evaluation on the flat trajectory.

5
Discussion

This chapter presents a detailed discussion of the algorithm's training process, as well as the performance evaluation of velocity tracking and grading. Based on the results, further analysis is conducted with regard of time constraints around the simulation platform, and the potential of the system for deployment in a real excavator.

5.1 Training parameters

Throughout the development of the control policy, various training parameters were systematically evaluated, including episode length, batch size, reward function formulation and scaling, as well as network architectures. It was observed that these parameters influenced two primary aspects of the learning process: training stability and policy performance. While hyperparameters such as batch size and episode length primarily affected training stability, elements like reward structure and network architecture had a larger impact on the final policy quality.

A key observation was that episodes could not be too short in duration. During the early stages of each episode, the simulation model showed oscillatory transients, which stemmed from an initial imbalance in the system dynamics. If the episode terminated before the model stabilized, the reinforcement learning agent tended to focus solely on suppressing these transient behaviors, thereby neglecting meaningful control objectives. By extending the episode length beyond this transient period and providing well-structured rewards for post-transient behavior, the agent was able to learn more effective and generalizable policies.

Batch size also played a critical role in ensuring stable convergence. Initial experiments used a batch size of 16, which resulted in high variance during training, as illustrated in Figure 4.3. Given that the model was trained across 16 predefined starting configurations, a batch size of 16 yielded only one sample per configuration per training step, insufficient for effective generalization. Increasing the batch size to 128 allowed for eight samples per configuration, leading to a more robust and stable learning process. It is reasonable to expect that a fully randomized initialization of starting positions could have further improved generalization. However, due to platform limitations, the initial conditions had to be defined prior to model export. As a result, 16 separate models with varied initial conditions were loaded and evaluated during training as the training was parallelized on 16 CPU cores. If the simulation platform had supported dynamic initialization at runtime, a broader and more diverse set of scenarios could have been explored more efficiently.

Among the most influential components was the reward function and its associated

scaling. Initial training runs used a squared error formulation as defined in Equation 3.29, which caused diminishing reward gradients for small errors, effectively plateauing improvements when velocity errors fell below 0.1 m/s. This led to premature convergence of the agent's performance. By reformulating the reward to use the linear error term in Equation 3.28, the agent was encouraged to continue minimizing errors beyond this threshold. The change significantly improved performance, particularly during extended training cycles. Additionally, the reward scaling function had a considerable impact. If rewards were given for actions with large velocity errors convergence was slowed. Instead, more effective results were achieved by shifting the reward curve to reward only lower velocity errors, as shown in Figure 3.7. This adjustment encouraged more rapid and consistent learning progress.
Finally, the selection of network architecture, particularly the inclusion of time dependency, was shown to be important. Given the continuous nature of the simulation environment, it was important for the agent to have access to temporal context in order to make informed decisions. This is shown in Figure 4.5, where three different architectures were evaluated, all with the same amount of layers. Recurrent or time-aware network structures consistently outperformed feedforward counterparts, showing the importance of modeling time-dependent behavior in sequential control tasks.

5.2 Hydraulic control

Hydraulic control was divided into two parts, separate and combined policies. The purpose of the separate policies was to see the potential performance of the different links, while the combined policy was used to implement the complete system. The velocity tracking precision achieved with the separate policies was not quite reached for the final combined policy, but performance was deemed sufficient.

5.2.1 Separate cylinder policies

Performance of the velocity tracking for the separate cylinders policies achieved an better accuracy when training with separate networks. The most noticeable increase in performance was for the boom that went from an average error of 0.1 m/s to 0.034 m/s when training a separate policy for the boom cylinder compared to the combined actuator policy. There was also an increase in precision when using separate policy for the arm where error decreased from 0.05 m/s to 0.03 m/s, and bucket with a decrease from 0.06 to 0.03 m/s on average.
While the separate policies achieved better precision when evaluating on one moving cylinder, it can not be concluded as the superior solution. This is due to that much of the system complexity comes from how the links of the excavator affects each other. When only moving one link at a time, much of the dynamics the policy needs to learn are removed, for example the flow and pressures will only be affected by the link to be moved and not all three. When moving more than one link at a time, the dynamics of the system completely changes, thus affecting the precision of the algorithm. This is highlighted in Figure 4.12, where the separate policy does not remove the oscillatory behavior of the other links that causes oscillations throughout

the episode. Compare this to the combined policy where that type of oscillatory behavior is only present in the very start of the episode and is then canceled out.
When training separate policies, the pump control is implemented in each of the policies meaning that the policy learns to fit the pump flow to the specific function. To accurately implement a solution with separate policies for the links, it would also require a policy for pump control, so the flow wouldn't solely be fitted to one function. All the separate policies would also have to be trained with the other links moving to learn how the dynamics of the other links affect the current link. This could make for a more precise solution, because each policy has a much more clear goal to optimize. When training one policy for all cylinders with pump control the policy can't determine which link is affected by the action as it only sees if the sum of rewards go up or down. However, only using one network requires much less computational power and is much more data efficient to train. If time wouldn't be a factor, the solution with separate policies could be tested, but with how the RL structures are currently implemented this solution would require too much time to test.

5.2.2 Velocity tracking

After the initial phases of hyperparameter selection, the combined policy had major difficulties extending the boom cylinder as seen in Figure 4.9. If the other functions requested a velocity at the same time, raising the boom became almost impossible, while the performance of the other functions was not affected.
The reason for this type of behavior from the policy could be a result of the boom being the hardest link to control. From the training it can be seen that the velocity errors for the arm and bucket steadily goes down throughout the training (Figure 4.8), while the boom suddenly stops at an error of around 0.13 m/s. This could indicate that the task of learning the dynamics for the arm and bucket is easier than controlling the boom. Because the algorithm can't distinguished the different link and only sees the total reward, it derives a solution that focuses on optimizing performance for the arm and bucket but in the process seems to hinder learning of the boom which stagnates. When compared to the separate policy for the boom, training resulted in an much lower error of 0.03 m/s, so the boom has potential to achieve better performance on velocity tracking, but is most likely held back by the policy focusing on an optimal solution for the arm and bucket.
A big improvement compared to the separate policies was that the oscillations of the system was reduced. The oscillatory behavior in the separate policies originates from the movement of the other links that which is not controlled. With the combined policy that can derive relationships between the links, oscillatory behavior is automatically removed when the other links are controlled to keep still thus reducing oscillations.
After training the final policy with the increased velocity reward for the boom, it performed well in both extending and retracting the cylinder but at the cost of bucket performance. Even if accuracy was sacrificed to increase boom performance the policy would perform much better at the grading task due to the bucket velocity impacting the position of the bucket tip much less then the boom.

5.2.3 Oscillations

When looking at oscillations in the system, the combined policy actively works to reduce their occurrence. As seen in Figure 4.12, where the biggest oscillations for the separate policy are almost completely removed when using the combined policy. This is likely due to the combined policy keeping a relationship between the links and concludes how they affect each other. The transient state in the beginning of each episode in which the system suffers from self-oscillation due to pressure errors is also much less impactful when using the combined policy.

However, small oscillations in velocity are not harmful for the complete system, due to the grading being conducted in the position domain. As seen in the performance of the grading, where it operated with good precision even if the velocity tracking was not perfect. The reason is that the policy oscillates around the target velocity instead of the target position, which is more advantageous than using length tracking for hydraulic control that would create oscillations around the target position instead.

5.3 Grading performance

When implementing the final policy in the complete system for grading, the velocity differences that seemed alarming when evaluating the controller seemed to have less impact on grading. Because the bucket stood for the largest error in velocity tracking, the system could still perform well when the velocities of the boom and arm were accurate. As seen in Figure 4.17, the velocity tracking of the bucket had large errors but still was able to achieve a grading performance below 4 centimeters on most trajectories. The largest errors achieved were mainly due to bucket performance. Especially when the bucket cylinder was to be retracted, the controller could not deliver the target velocity, which got the bucket tip off trajectory when the other links moved correctly.

However, on most trajectories the performance was beyond expectations when compared to the velocity tracking precision. This seems to indicate that if the complexity of hydraulic control is handled adequately, the problem of grading becomes quite simple. Only a simple PID controller was implemented for the actual grading task, which cannot handle nonlinearities that well, but it still completes most tasks. This may be because most of the nonlinearities of the system was removed when the hydraulic controller handled those nonlinearities.

The use of target velocities instead of cylinder lengths for the hydraulic controller reduced the oscillations of the system. In Figure 4.17 oscillatory behavior in the velocities can seem alarming for the grading performance. However when looking at the grading error in Figure 4.16, the magnitude of oscillations never exceed 1 cm, which is extremely small for the system. The controller oscillates around velocities instead of position which is very favorable for the system and can even help reduce oscillations created by the other links.

5.4 Simulation platform and time constraints

A significant bottleneck throughout the project was the simulation platform itself, which had a direct impact on the overall training time and experimentation capabilities. The simulation model, exported and interfaced with via Python, was inherently CPU-bound and computationally intensive. The model was large and complex, containing a substantial amount of embedded logic representing mechanical, hydraulic, and control system behaviors. As a result, each iteration during reinforcement learning took a considerable amount of time, with a single policy evaluation requiring up to 24 hours of processing.

While multi-threading and parallelization strategies were employed to combat this issue, resulting in a 16x speedup, the simulation still remained a major limiting factor. The training pipeline was effectively constrained by CPU throughput, as the simulation engine did not utilize GPU acceleration. Meanwhile, the machine learning operations running on the GPU had a relatively small computational footprint in comparison, indicating a significant under-utilization of available hardware resources such as CUDA-compatible GPUs.

These performance limitations imposed strict restrictions on the scope of the experiments. Only the most important parameters and configurations could be tested, significantly reducing the total number of policy evaluations and hyperparameter sweeps that could be reasonably conducted within the available time frame. This likely constrained the quality of the resulting models and left many potential improvements unexplored.

In retrospect, alternative strategies for system identification and training could have yielded more efficient outcomes. One such possibility would have been to train either the complete reinforcement learning agent directly on the physical machine, or have the reinforcement learning agent interact with an AI-based plant model, trained from an physical machine. This would have not only removed the CPU-bound simulation bottleneck but also allowed for near real-time interaction using GPU-accelerated components and sensor feedback. However, this approach was not feasible for this project due to logistical constraints as no physical machines were accessible during the development period, and the prototypes were under production in Asia. It also contains safety aspects, as the model can be unpredictable and might employ invalid configurations which can hurt the machine or its surroundings. This illustrates a broader challenge in simulation-based reinforcement learning for heavy machinery, where model complexity and compute limitations can significantly hinder development speed and solution quality.

Future iterations of this work would benefit from transitioning to simulation platforms that support GPU acceleration or real time co-simulation frameworks. Moreover, access to physical prototypes or digital twins running on cloud infrastructure could allow for faster and more scalable training pipelines, ultimately leading to more optimized and robust control policies.

5.5 Application on a real excavator

A continuation of this project is the potential deployment of the developed hydraulic and positional controllers onto a real excavator. While the simulation-based results demonstrate promising behavior, several critical limitations within the current training and simulation environment must be addressed before such real world implementation becomes feasible.
First and foremost, the current simulation model lacks realistic interaction between the excavator and its environment. Specifically, the bucket is not subjected to external forces such as soil resistance or inertial feedback during digging or grading operations. This constitutes a significant simplification, as in practical scenarios, the act of grading inherently involves displacing material with complex physical properties, including resistance, weight distribution, and terrain deformation. Without these dynamics modeled during training, the learned policy is unlikely to generalize effectively to real-world conditions. Future iterations of this work must therefore incorporate a physically realistic environment model, including dynamic soil interaction and terrain response, in order to develop policies that are both transferable and robust.
Additionally, the simulation assumes a flat, level operating surface and disables the swing mechanism of the excavator. These constraints further limit the trained controller's ability to generalize. In real-world scenarios, excavators often operate on uneven or sloped terrain, and such variations can significantly affect the dynamics of the hydraulic actuators, particularly because the controller is dependent on velocity feedback from the hydraulic cylinders. When operating on a slope, gravity will introduce bias into the velocity profiles, and the existing controller, trained solely on flat-ground dynamics, may fail to compensate appropriately. One possible mitigation strategy is to enhance the positional controller with pre-evaluation mechanisms that determine whether a desired slope or surface contour is physically achievable given the machine's current orientation.
The omission of the swing function was a deliberate simplification intended to reduce model complexity and focus the training scope. However, the integration of swing control is essential for enabling more advanced operations such as diagonal grading. The authors are confident that incorporation of the swing hydraulics and the ability to perform more complex grading maneuvers is entirely feasible, given more time. The training environment would need to be modified to support multiple degrees of freedom and the rewards would need to be updated accordingly, but the possibility exists.
In summary, while the results of this project provide a solid foundation, several enhancements are necessary before real world deployment is possible. These include realistic environment modeling with force feedback, support for inclined terrain, and the integration of swing dynamics into the control framework. Addressing these challenges would enable the system to function reliably in the diverse and unstructured conditions encountered in practical excavation tasks.

6
Conclusion

The goal of this thesis was to examine the possibilities of using reinforcement learning to control an IMVT hydraulic system and use it to autonomously perform grading. This was solved by implementing a proximal policy optimization algorithm for training a policy used as the hydraulic controller. The results show that the policy was able to perform all the grading tasks presented to the system with a average error of under 4 cm. However, the velocity tracking of the controller is not perfect, especially for the bucket cylinder. This did not have any major impacts on the grading performance, showing that the proposed hydraulic controller is able to handle the nonlinearities present in the system. This resulted in that the positional controller could be implemented as a simple PID controller with satisfactory performance.
Even though grading can be performed accurately, the hydraulic controller is not ready to be used in other operation modes, such as manual control or semi-autonomous control. As discussed in the previous chapter, the proposed system is not ready to be implemented on a real excavator. Instead, further development and training of the hydraulic controller is needed to achieve better velocity tracking.

6.1 Future works

The project can be continued by further developing the training environment of the hydraulic controller, either by new reward functions, different observations, or more complex environments, it could be possible to develop a policy with increased accuracy in both grading and velocity tracking. One of the most significant impacts on the performance of the policy was the scaling and design of the reward function. By further developing and tuning of the rewards, it might lead to a policy where all cylinders are able to achieve an more accurate velocity tracking. The current model completes the grading task, but has varying performance on velocity tracking, especially for the bucket, where errors could be large with oscillating behavior. For a more robust system that could be implemented in other solutions, the precision of velocity tracking has to be increased.
The surrounding system for positional control has a fairly simple implementation, with many safety aspects currently ignored. For example, no check for extension limits in the actuators are currently implemented, if the cylinder reaches its endpoint a velocity can still be requested in that direction. This could be dangerous as pressure could build up which might damage the machine, and invalid positions of the links could be requested to reach the target point. To avoid this, a more advanced controller is needed which is aware of the environment around itself and

what positions is reachable to mitigate dangerous velocity requests. As the controller currently receives the wanted trajectory in a simulation environment, no interaction with the operator is needed. However, if the system was to be deployed onto a real machine, the positional controller would need to expose some form of interface for the operator to interact with, where the operator can request the grading trajectory. This could for example be integrated to the already existing VAC or Dig Assist when the hydraulic controller is sufficiently good.
Another aspect which has not been investigated thoroughly is the fuel efficiency of the machine. As there was little to no data available on the current fuel efficiency of the machine, there existed no real benchmark to compare against. A fuel reward was introduced into the reward function during hyperparameter testing but showed no major impact. As training of the model was slow, the accuracy of the velocity tracking was prioritized over fuel efficiency as a fuel efficient system which is unable to solve the task is useless. With more time, more focus should be given to the reward scaling of the fuel reward and further development of other rewards which encourage fuel efficiency. This could for example be penalties for discarding hydraulic fluid in the system by penalizing how much each return valve is open. However, this might introduce other unwanted behaviors in the system which has to be analyzed and regulated.

Bibliography

[1] Volvo-CE, "Ec550e | excavators | overview | volvo construction equipment." https://www.volvoce.com/europe/en/products/excavators/ec550e/#overview. Accessed: 2025-02-11.

[2] M. Tulldahl and O. Valdemarsson, "Intuitive grading assist for excavator - a control system for a linear motion of an excavator arm," 2016.

[3] L. Zhang, J. Zhao, P. Long, L. Wang, L. Qian, F. Lu, X. Song, and D. Manocha, "An autonomous excavator system for material loading tasks," *Science Robotics*, vol. 6, no. 55, p. eabc3164, 2021.

[4] B. J. Hodel, "Learning to operate an excavator via policy optimization," *Procedia Computer Science*, vol. 140, pp. 376–382, 2018. Cyber Physical Systems and Deep Learning Chicago, Illinois November 5-7, 2018.

[5] Y. Yoo, D. Jung, and S.-W. Kim, "3d operation of autonomous excavator based on reinforcement learning through independent reward for individual joints," 2024.

[6] D. Jud, G. Hottiger, P. Leemann, and M. Hutter, "Planning and control for autonomous excavation," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2151–2158, 2017.

[7] P. Egli and M. Hutter, "A general approach for the automation of hydraulic excavator arms using reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 5679–5686, 2022.

[8] B. Son, C. Kim, C. Kim, and D. Lee, "Expert-emulating excavation trajectory planning for autonomous robotic industrial excavator," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2656–2662, 2020.

[9] Volvo-CE, "Dig assist | volvo construction equipment." https://www.volvoce.com/sverige/sv-se/volvo-services/dig-assist/. Accessed: 2025-05-16.

[10] Volvo-CE, "Volvo active control." https://www.volvoce.com/sverige/sv-se/volvo-services/productivity-services/dig-assist/volvo-active-control/, 2024. Accessed: 2024-11-28.

[11] G. Filo, "Artificial intelligence methods in hydraulic system design," *Energies*, vol. 16, 2023.

[12] Z. Yao, X. Liang, G. P. Jiang, and J. Yao, "Model-based reinforcement learning control of electrohydraulic position servo systems," *IEEE/ASME Transactions on Mechatronics*, vol. 28, 2023.

[13] A. Khater, M. Fekry, M. El-Bardini, *et al.*, "Deep reinforcement learning-based adaptive fuzzy control for electro-hydraulic servo system," *Neural Computing and Applications*, vol. 2025, 2025.

[14] VolvoCE, "Electro-hydraulics may change the future of excavator design - the scoop | volvo ce," 6 2024.

[15] K. Abuowda, I. Okhotnikov, S. Noroozi, P. Godfrey, and M. Dupac, "A review of electrohydraulic independent metering technology," *ISA Transactions*, vol. 98, pp. 364–381, 2020.

[16] P. Casoli, F. Scolari, C. M. Vescovini, and M. Rundo, "Energy comparison between a load sensing system and electro-hydraulic solutions applied to a 9-ton excavator," 2022.

[17] R. Ding, J. Zhang, and B. Xu, "Advanced energy management of a novel independent metering meter-out control system: A case study of an excavator," *IEEE Access*, vol. 6, pp. 45782–45795, 2018.

[18] T. H. Nguyen, T. C. Do, and K. K. Ahn, "Research on a new independent metering system for boom excavator," in *2021 24th International Conference on Mechatronics Technology (ICMT)*, pp. 1–5, 2021.

[19] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," *CoRR*, vol. abs/1502.05477, 2015.

[20] G. Schäfer, M. Schirl, J. Rehrl, S. Huber, and S. Hirlaender, "Python-based reinforcement learning on simulink models," 2024.

[21] M. Vukovic, R. Leifeld, and H. Murrenhoff, "Reducing fuel consumption in hydraulic excavators—a comprehensive analysis," *Energies*, vol. 10, p. 687, 05 2017.

[22] Y. XIAOLONG, P. M. J, and P. J. L, "Pilot operated control valve having a poppet with integral pressure compensating mechanism."

[23] P. Opdenbosch, N. Sadegh, W. Book, and A. Enes, "Auto-calibration based control for independent metering of hydraulic actuators," *2011 IEEE International Conference on Robotics and Automation, Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 153–158, 2011.

[24] P. Opdenbosch, N. Sadegh, and W. Book, "Intelligent controls for electro-hydraulic poppet valves," *Control Engineering Practice*, vol. 21, no. 6, pp. 789–796, 2013.

[25] K. Choi, J. Seo, Y. Nam, and K. U. Kim, "Energy-saving in excavators with application of independent metering valve," *Journal of Mechanical Science and Technology*, vol. 29, pp. 387–395, 1 2015.

[26] R. A. Paz, "The design of the pid controller," *Computer Engineering*, 2001.

[27] R. M. Schmidt, "Recurrent neural networks (rnns): A gentle introduction and overview," 2019.

[28] R. C. Staudemeyer and E. R. Morris, "Understanding lstm – a tutorial into long short-term memory recurrent neural networks," 2019.

[29] A. G. Collins, *Reinforcement Learning*. MIT Press, dec 3 2024. https://oecs.mit.edu/pub/k2ek981x.

[30] M. Ghasemi and D. Ebrahimi, "Introduction to reinforcement learning," 2024.

[31] K. Murphy, "Reinforcement learning: A comprehensive overview," 2025.

[32] S. Garcin, T. McInroe, P. S. Castro, P. Panangaden, C. G. Lucas, D. Abel, and S. V. Albrecht, "Studying the interplay between the actor and critic representations in reinforcement learning," 2025.

[33] M. Lehmann, "The definitive guide to policy gradients in deep reinforcement learning: Theory, algorithms and implementations," 2024.

[34] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017.

[35] H. Wei, X. Liu, L. Mashayekhy, and K. Decker, "Mixed-autonomy traffic control with proximal policy optimization," in *2019 IEEE Vehicular Networking Conference (VNC)*, pp. 1–8, Dec 2019.

[36] B. Zhang, X. Lu, R. Diao, H. Li, T. Lan, D. Shi, and Z. Wang, "Real-time autonomous line flow control using proximal policy optimization," in *2020 IEEE Power and Energy Society General Meeting (PESGM)*, pp. 1–5, Aug 2020.

[37] B. Li, Y. Cui, Y. Xiao, S. Fu, J. Choi, and C. Zheng, "An improved energy management strategy of fuel cell hybrid vehicles based on proximal policy optimization algorithm," *Energy*, vol. 317, p. 134585, 2025.

[38] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," 2018.

[39] Mathworks, "Simscape - matlab." `https://se.mathworks.com/products/simscape.html`. Accessed: 2025-02-11.

[40] Mathworks, "Embedded coder - matlab." `https://se.mathworks.com/products/embedded-coder.html`. Accessed: 2025-02-11.

[41] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. D. Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, H. Tan, and O. G. Younis, "Gymnasium: A standard interface for reinforcement learning environments," 2024.

[42] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.

[43] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," 2019.

A

Model comparison between Matlab and Python

A.1 Cylinder displacement

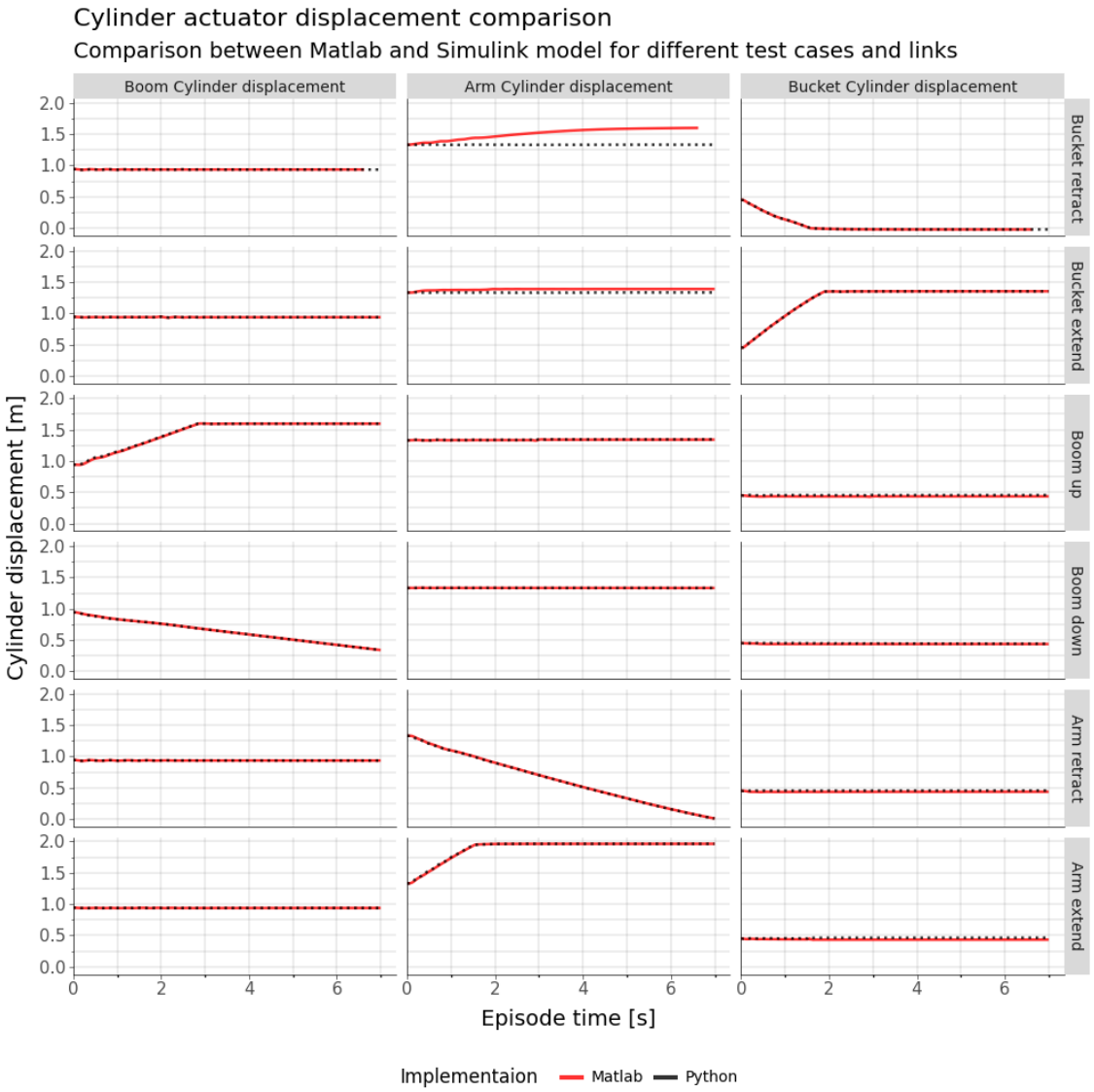


Figure A.1: Comparison between Matlab and Python cylinder displacement for different test cases.

A.2 IMU signals

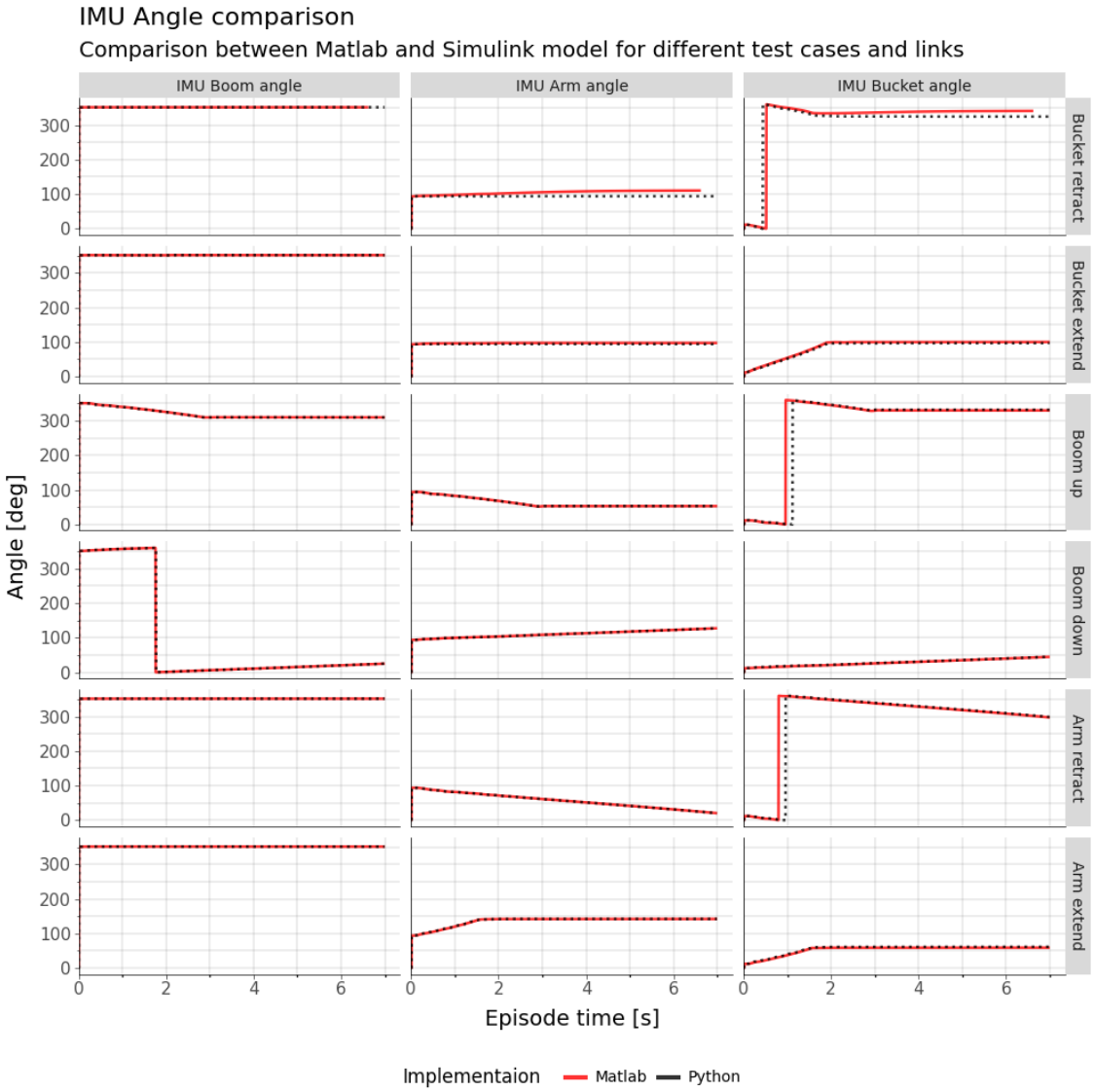


Figure A.2: Comparison between Matlab and Python IMU signals for different test cases.

B

Excavator simulation inputs and outputs

B.1 Inputs

Table B.1: Excavator Control Inputs

Description	Min	Max	Unit
Boom Control Valves			
Boom SA1 EHPV current	0	1000	mA
Boom SA2 EHPV current	0	1000	mA
Boom SB2 EHPV current	0	1000	mA
Boom AR EHPV current	0	1000	mA
Boom BR EHPV current	0	1000	mA
Boom AB EHPV current	0	1000	mA
Boom RiHRV EHPV current (Unused)	0	1000	mA
Boom LeHRV EHPV current (Unused)	0	1000	mA
Arm Control Valves			
Arm SA1 EHPV current	0	1000	mA
Arm SA2 EHPV current	0	1000	mA
Arm SB1 EHPV current	0	1000	mA
Arm SB2 EHPV current	0	1000	mA
Arm AR EHPV current	0	1000	mA
Arm BR EHPV current	0	1000	mA
Arm AB EHPV current	0	1000	mA
Arm HRV EHPV current (unused)	0	1000	mA
Bucket Control Valves			
Bucket SA1 EHPV current	0	1000	mA
Bucket SA2 EHPV current	0	1000	mA
Bucket SB1 EHPV current	0	1000	mA
Bucket SB2 EHPV current	0	1000	mA
Bucket AR EHPV current	0	1000	mA
Bucket BR EHPV current	0	1000	mA
Bucket AB EHPV current	0	1000	mA
Swing Control Valves			

Continued on next page

Table B.1: Excavator Control Inputs

Description	Min	Max	Unit
Swing SA1 EHPV current (unused)	0	1000	mA
Swing SB1 EHPV current (unused)	0	1000	mA
Swing AR EHPV current (unused)	0	1000	mA
Swing BR EHPV current (unused)	0	1000	mA
Swing time delay solenoid current (unused)	0	1000	mA
Pump Control			
Pump 1 EHPV current	0	1000	mA
Pump 2 EHPV current	0	1000	mA
Pump 1 bypass EHPV current	0	1000	mA
Pump 2 bypass EHPV current	0	1000	mA

B.2 Outputs

Table B.2: Excavator Control Outputs

Description	Min	Max	Unit
Position Information			
Boom cylinder extension	0	1.6	m
Arm cylinder extension	0	1.9	m
Bucket cylinder extension	0	1.3	m
Swing angle	-	-	rad
Bucket teeth X position	-	-	m
Bucket teeth Y position	-	-	m
Bucket teeth Z position	-	-	m
End of linear bucket, creates horizontal plane with bktTip1, X position	-	-	m
End of linear bucket, creates horizontal plane with bktTip1, Y position	-	-	m
End of linear bucket, creates horizontal plane with bktTip1, Z position	-	-	m
Back of bucket, X position	-	-	m
Back of bucket, Y position	-	-	m
Back of bucket, Z position	-	-	m
Back of bucket, X position	-	-	m
Back of bucket, Y position	-	-	m
Back of bucket, Z position	-	-	m
Back joint, connects yoke and bucket, X position	-	-	m
Back joint, connects yoke and bucket, Y position	-	-	m
Back joint, connects yoke and bucket, Z position	-	-	m
Joint which connects bucket with arm, X position	-	-	m
Joint which connects bucket with arm, Y position	-	-	m

Continued on next page

Table B.2: Excavator Control Outputs

Description	**Min**	**Max**	**Unit**
Joint which connects bucket with arm, Z position	-	-	m
IMU - Angle Information			
Boom angle	0	360	deg
Boom angular velocity	-	-	deg/s
Boom angular acceleration	-	-	deg/s^2
Arm angle	0	360	deg
Arm angular velocity	-	-	deg/s
Arm angular acceleration	-	-	deg/s^2
Yoke angle	0	360	deg
Yoke angular velocity	-	-	deg/s
Yoke angular acceleration	-	-	deg/s^2
Network Control - CAN signals			
Engine rotation speed	0	-	rpm
Actual engine percentage torque	0	100	%
Driver demanded engine percentage torque	0	100	%
Nominal friction percentage torque	0	100	%
Vehicle input demanded torque	-	-	Nm
Engine boost pressure	0	-	kPa
Engine torque	0	-	Nm
Engine friction torque	0	-	Nm
Reference engine torque	0	-	Nm
Requested motor speed	0	-	rpm
Requested motor torque	0	-	Nm
Requested override control mode	-	-	-
Requested override control mode priority	-	-	-
Requested speed control condition	-	-	-
Network Control - Pressures			
Pump 1 pressure	0	410	bar
Pump 2 pressure	0	410	bar
Boom cylinder A-side pressure	0	400	bar
Boom cylinder B-side pressure	0	400	bar
Arm cylinder A-side pressure	0	400	bar
Arm cylinder B-side pressure	0	400	bar
Bucket cylinder A-side pressure	0	400	bar
Bucket cylinder B-side pressure	0	400	bar
Swing motor A-side pressure	0	400	bar
Swing motor B-side pressure	0	400	bar
Engine Data			
Engine inertia	0	-	kg $\cdot$ m^2
Engine fuel consumption	0	-	mg/s
Available fuel	0	-	kg
Engine torque	0	-	Nm

Continued on next page

Table B.2: Excavator Control Outputs

Description	**Min**	**Max**	**Unit**
Engine urea consumption	0	-	g/s
Engine total urea consumed	0	-	kg
Engine total urea consumed	0	-	L
Engine boost pressure	0	-	kPa
Engine angular velocity	0	-	rad/s
Engine boost torque offset	-	-	Nm
Chassis Data			
Boom cylinder position	0	1.6	m
Boom cylinder velocity	-2	2	m/s
Boom cylinder force	-	-	N
Arm cylinder position	0	1.9	m
Arm cylinder velocity	-2	2	m/s
Arm cylinder force	-	-	N
Bucket cylinder position	0	1.3	m
Bucket cylinder velocity	-2	2	m/s
Bucket cylinder force	-	-	N
Swing angle	-	-	rad
Swing angular velocity	-	-	rad/s
Swing torque	-	-	Nm
Track left height	-	-	m
Track right side	-	-	m
Global positions			
Bucket teeth X position	-	-	m
Bucket teeth Y position	-	-	m
Bucket teeth Z position	-	-	m
End of linear bucket, creates horizontal plane with bkt-Tip1, X position	-	-	m
End of linear bucket, creates horizontal plane with bkt-Tip1, Y position	-	-	m
End of linear bucket, creates horizontal plane with bkt-Tip1, Z position	-	-	m
Back of bucket, X position	-	-	m
Back of bucket, Y position	-	-	m
Back of bucket, Z position	-	-	m
Back of bucket 2, X position	-	-	m
Back of bucket 2, Y position	-	-	m
Back of bucket 2, Z position	-	-	m
Back joint, connects yoke and bucket, X position	-	-	m
Back joint, connects yoke and bucket, Y position	-	-	m
Back joint, connects yoke and bucket, Z position	-	-	m
Joint which connects bucket with arm, X position	-	-	m
Joint which connects bucket with arm, Y position	-	-	m

Continued on next page

Table B.2: Excavator Control Outputs

Description	**Min**	**Max**	**Unit**
Joint which connects bucket with arm, Z position	-	-	m
Joint which connects arm with bucket, X position	-	-	m
Joint which connects arm with bucket, Y position	-	-	m
Joint which connects arm with bucket, Z position	-	-	m
Start position of arm cylinder on the boom, X position	-	-	m
Start position of arm cylinder on the boom, Y position	-	-	m
Start position of arm cylinder on the boom, Z position	-	-	m
Joint where the boom connects to the machine frame, X position	-	-	m
Joint where the boom connects to the machine frame, Y position	-	-	m
Joint where the boom connects to the machine frame, Z position	-	-	m
Second joint on the arm which connects the 4-bar linkage with bkt_arm, X position	-	-	m
Second joint on the arm which connects the 4-bar linkage with bkt_arm, Y position	-	-	m
Second joint on the arm which connects the 4-bar linkage with bkt_arm, Z position	-	-	m
Cylinder Position Information - Boom			
Start joint on machine frame, X position	-	-	m
Start joint on machine frame, Y position	-	-	m
Start joint on machine frame, Z position	-	-	m
End joint on boom, X position	-	-	m
End joint on boom, Y position	-	-	m
End joint on boom, Z position	-	-	m
Cylinder Position Information - Arm			
Start joint on boom, X position	-	-	m
Start joint on boom, Y position	-	-	m
Start joint on boom, Z position	-	-	m
End joint on the arm, X position	-	-	m
End joint on the arm, Y position	-	-	m
End joint on the arm, Z position	-	-	m
Cylinder Position Information - Bucket			
Start joint on the arm, X position	-	-	m
Start joint on the arm, Y position	-	-	m
Start joint on the arm, Z position	-	-	m
End position on the yoke, X position	-	-	m
End position on the yoke, Y position	-	-	m
End position on the yoke, Z position	-	-	m
Hydraulic Data			
Bucket SB1 valve flow rate	-	-	L/min

Continued on next page

Table B.2: Excavator Control Outputs

Description	Min	Max	Unit
Bucket SB1 valve pressure differential	-	-	bar
Bucket SA1 valve flow rate	-	-	L/min
Bucket SA1 valve pressure differential	-	-	bar
Bucket BR valve flow rate	-	-	L/min
Bucket BR valve pressure differential	-	-	bar
Bucket AR valve flow rate	-	-	L/min
Bucket AR valve pressure differential	-	-	bar
Bucket SB2 valve flow rate	-	-	L/min
Bucket SB2 valve pressure differential	-	-	bar
Bucket SA2 valve flow rate	-	-	L/min
Bucket SA2 valve pressure differential	-	-	bar
Bucket BA valve flow rate	-	-	L/min
Bucket AB valve pressure differential	-	-	bar
Arm SB1 valve flow rate	-	-	L/min
Arm SB1 valve pressure differential	-	-	bar
Arm SA1 valve flow rate	-	-	L/min
Arm SA1 valve pressure differential	-	-	bar
Arm BR valve flow rate	-	-	L/min
Arm BR valve pressure differential	-	-	bar
Arm AR valve flow rate	-	-	L/min
Arm AR valve pressure differential	-	-	bar
Arm SB2 valve flow rate	-	-	L/min
Arm SB2 valve pressure differential	-	-	bar
Arm SA2 valve flow rate	-	-	L/min
Arm SA2 valve pressure differential	-	-	bar
Arm AB valve flow rate	-	-	L/min
Arm AB valve pressure differential	-	-	bar
Boom SA1 valve flow rate	-	-	L/min
Boom SA1 valve pressure differential	-	-	bar
Boom BR valve flow rate	-	-	L/min
Boom BR valve pressure differential	-	-	bar
Boom AR valve flow rate	-	-	L/min
Boom AR valve pressure differential	-	-	bar
Boom SB2 valve flow rate	-	-	L/min
Boom SB2 valve pressure differential	-	-	bar
Boom SA2 valve flow rate	-	-	L/min
Boom SA2 valve pressure differential	-	-	bar
Boom AB valve flow rate	-	-	L/min
Boom AB valve pressure differential	-	-	bar

C

Grading evaluation results

C.1 Horizontal and slope towards

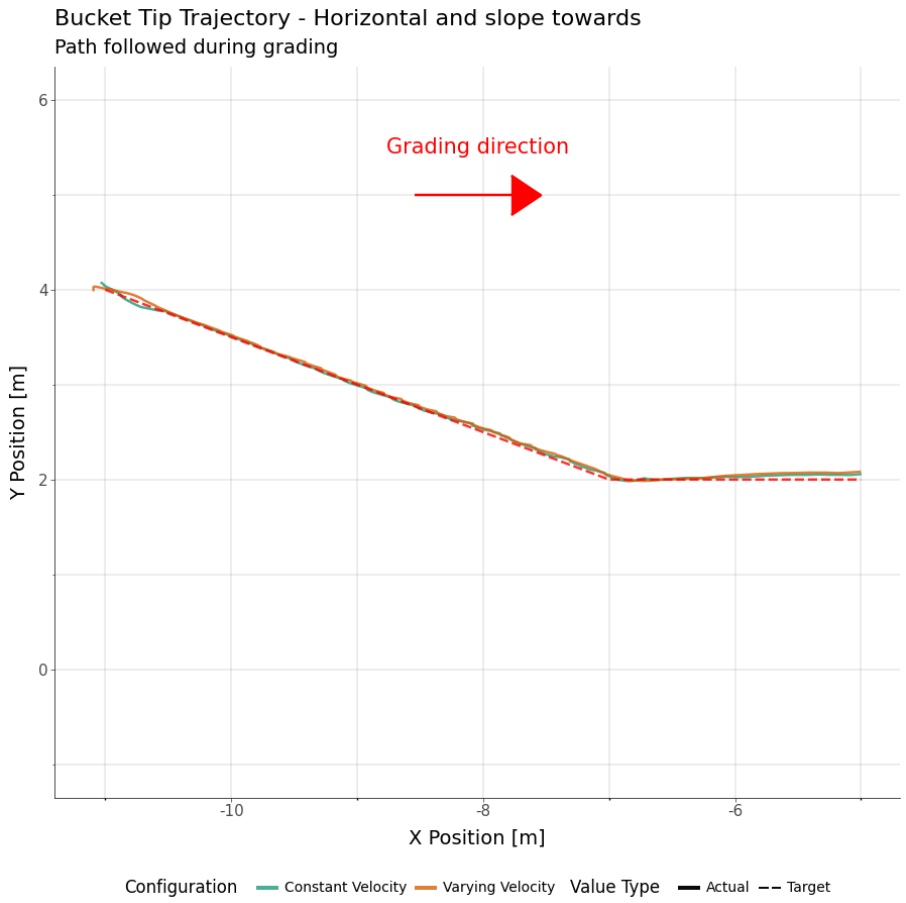


Figure C.1: Grading result on a sloping and horizontal trajectory where the linkage moves towards the excavator house.

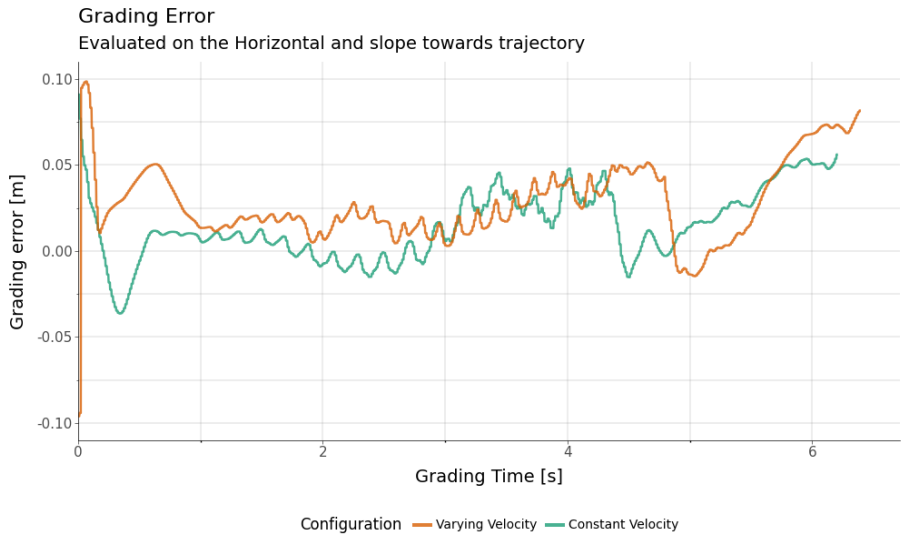


Figure C.2: Grading error on a sloping and horizontal trajectory where the linkage moves towards the excavator house.

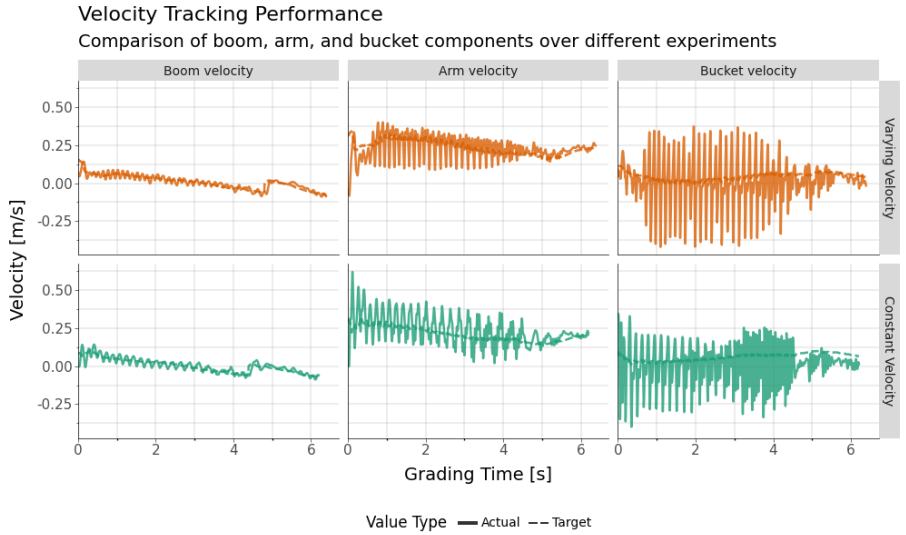


Figure C.3: Wanted and actual velocities in each cylinder for a sloping and horizontal trajectory where the linkage moves towards the excavator house.

C.2 Horizontal towards

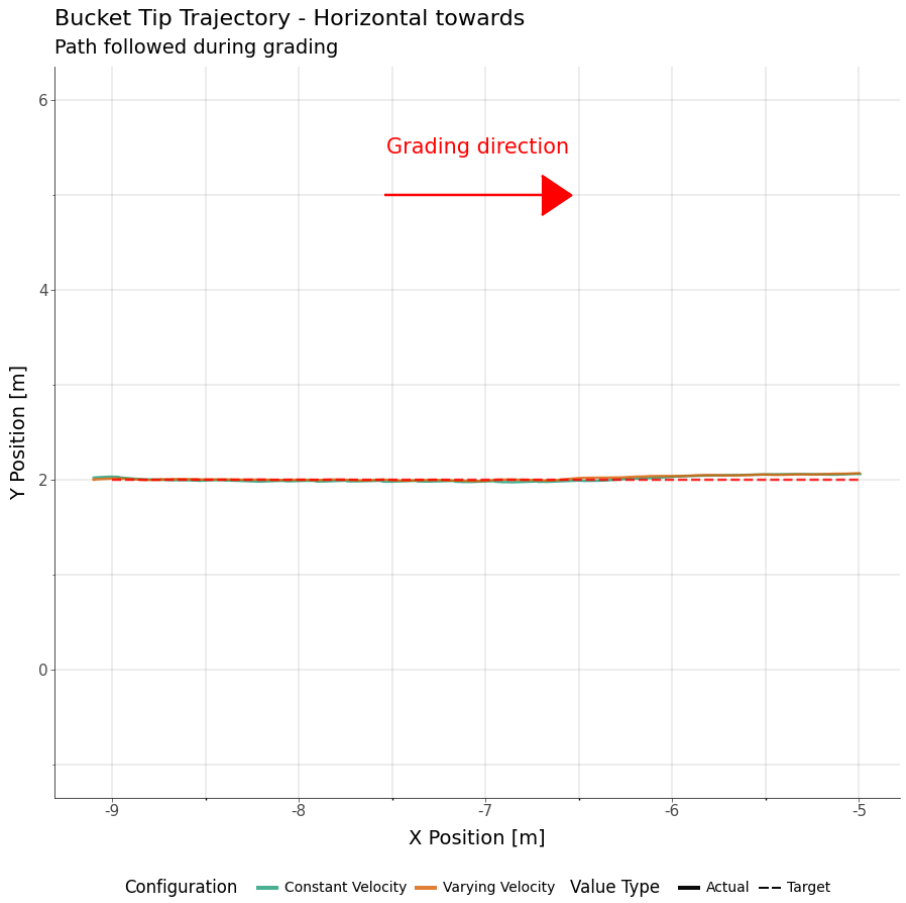


Figure C.4: Grading result on a horizontal trajectory where the linkage moves towards the excavator house.

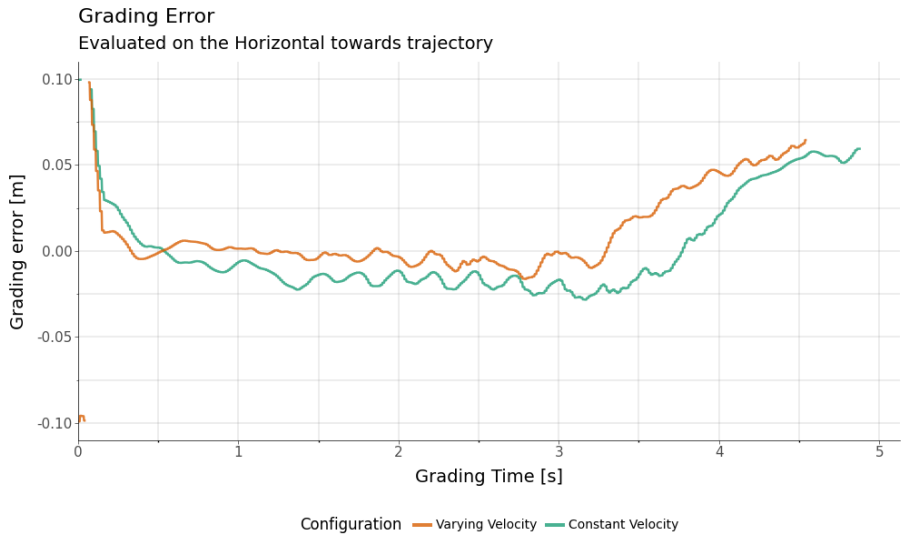


Figure C.5: Grading error on a horizontal trajectory where the linkage moves towards the excavator house.

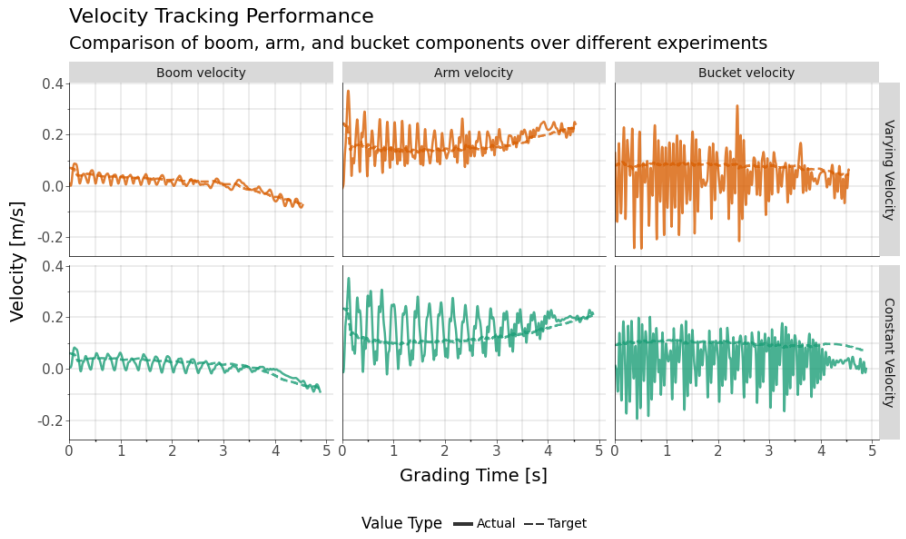


Figure C.6: Wanted and actual velocities in each cylinder for a horizontal trajectory where the linkage moves towards the excavator house.

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY