Contents lists available at ScienceDirect

Automation in Construction

journal homepage: www.elsevier.com/locate/autcon

Optimizing bucket-filling strategies for wheel loaders inside a dream environment

Daniel Eriksson[a,*], Reza Ghabcheloo[a], Marcus Geimer[b]

[a] *Automation and Mechanical Engineering, Faculty of Engineering and Natural Sciences, Tampere University, P.O. Box 1001, Tampere 33014, Finland*
[b] *Institute Mobile Machines, Karlsruhe Institute of Technology, Rintheimer Querallee, Karlsruhe 76131, Germany*

ARTICLE INFO

Keywords:
Automatic bucket filling
Wheel loaders
World models
Reinforcement learning
Neural networks

ABSTRACT

Reinforcement Learning (RL) requires many interactions with the environment to converge to an optimal strategy, which makes it unfeasible to apply to wheel loaders and the bucket filling problem without using simulators. However, it is difficult to model the pile dynamics in the simulator because of unknown parameters, which results in poor transferability from the simulation to the real environment. Instead, this paper uses world models, serving as a fast surrogate simulator, creating a dream environment where a reinforcement learning (RL) agent explores and optimizes its bucket-filling behavior. The trained agent is then deployed on a full-size wheel loader without modifications, demonstrating its ability to outperform the previous benchmark controller, which was synthesized using imitation learning. Additionally, the same performance was observed as that of a controller pre-trained with imitation learning and optimized on the test pile using RL.

1. Introduction

Heavy-duty Mobile Machines (HDDMs) such as wheel loaders are getting increasingly more assistance and autonomous functions, enabled by technical advancements in computer power and Artificial Intelligence (AI) and motivated by the upcoming labor shortage in the construction industry [1]. Assistance systems can reduce the barrier of entry for new operators in this area and thus reduce the need for expert operators, who need several years of experience to become effective workers. They also have the potential to increase the productivity and save costs by reducing fuel consumption. Another potential cost-saving area is reduced maintenance costs because an automatic system could reduce the overall wear and tear on the machine [2–4]. Automating the bucket filling problem can also improve efficiency on a worksite in combination with teleoperation, where one operator can control many machines at the same time and switch between them when necessary for intervention or giving new commands [5].

Finding a high-performing and robust automatic bucket filling system for wheel loaders is a challenging task because of unknown interaction forces between the bucket and the pile material, which depends on the properties of the material: kernel size, moisture, shape, and density. These properties make it difficult to create general and effective controllers [5]. Data-driven methods, on the other hand, do not require any modeling of the pile or the interaction forces, which is why most recent research for the bucket filling problem is based on these approaches.

Imitation Learning (IL), also known as learning from demonstrations, is a data-driven method, and it has been applied to the bucket filling problem in Dadhich et al. [6], Halbach et al. [7], Yang et al. [8], Yang et al. [9], Eriksson and Ghabcheloo [10]. All of them synthesize a Neural Network (NN) bucket filling controller by collecting expert demonstrations of the bucket filling task, and then evaluating the performance on the same pile used for data collection.

Another approach is to use Reinforcement Learning (RL) instead, and learn the optimal bucket filling behavior through trial and error by digging the pile repeatedly. This approach has previously been explored in Dadhich et al. [11], Backman et al. [12], Strokina et al. [13], Eriksson et al. [14], to train bucket filling controllers for wheel loaders, and in Egli and Hutter [15], Egli et al. [16], to train a digging strategy for excavators. RL has the potential to surpass human operators and discover unknown bucket filling strategies. The downside of RL is that it requires a significant amount of interactions with the environment to learn a good bucket filling strategy. Interacting with the environment is costly and unsafe for wheel loaders and other HDMMs, since it requires a lot of time and fuel, and has the potential risk of damaging the machine.

Simulators are powerful tools for testing and developing control strategies and algorithms without the limitations of the real world, and provide a safe alternative environment for training RL algorithms and general testing of different control strategies. Highly accurate

* Corresponding author.
E-mail address: daniel.eriksson@tuni.fi (D. Eriksson).

https://doi.org/10.1016/j.autcon.2024.105804
Received 30 March 2024; Received in revised form 23 September 2024; Accepted 26 September 2024
Available online 4 October 2024

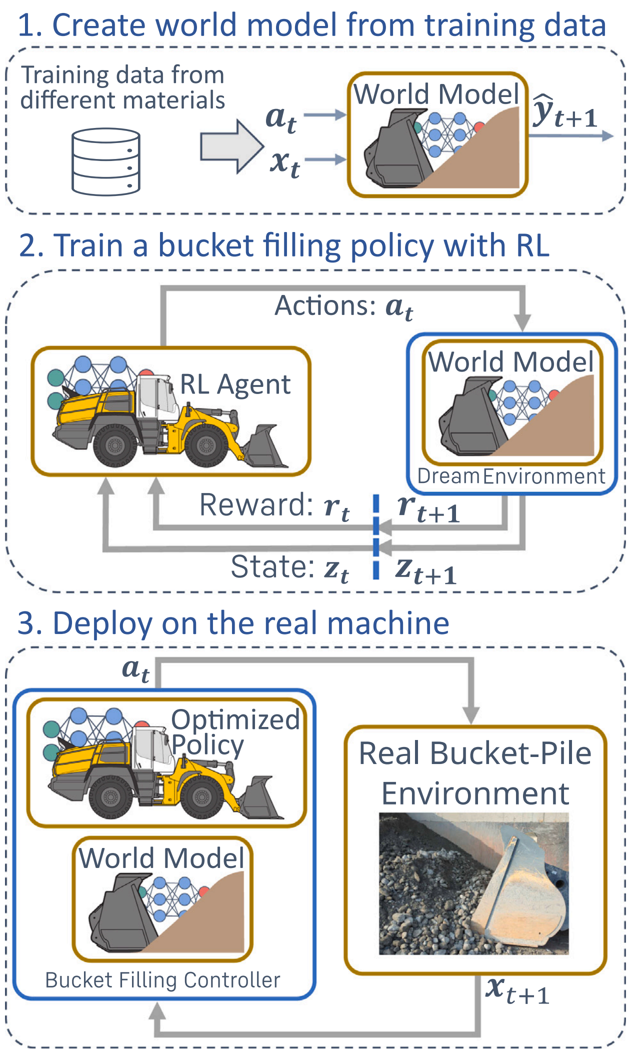

1. Create world model from training data
Training data from different materials
a_t
x_t
World Model
$\hat{y}_{t+1}$
2. Train a bucket filling policy with RL
Actions: a_t
RL Agent
World Model
Dream Environment
Reward: r_t
r_{t+1}
State: z_t
z_{t+1}
3. Deploy on the real machine
a_t
Optimized Policy
World Model
Bucket Filling Controller
Real Bucket-Pile Environment
x_{t+1}

Fig. 1. Overview of the method used for creating the bucket filling controller.

Discrete Element Method (DEM) simulations have been used in Filla [17], Filla and Frank [18], to find optimal bucket filling trajectories. DEM has the downside of being computationally intensive, and one scooping trajectory usually takes several hours to compute on consumer hardware.

Multibody simulation (MBS) is an alternative type of simulation that is able to run in real time on consumer hardware, thus making RL algorithms feasible to run and train. These simulators have been used in Backman et al. [12], Kurinov et al. [19], to find digging strategies for both wheel loaders and excavators. However, all simulators have the same fundamental problem: they are typically not accurate enough for a learned RL policy to be effectively transferred directly to the real world. Techniques such as domain randomization [20] and transfer learning from sim-to-real [21] can reduce this issue. These approaches are not researched or explored in this paper.

We explore alternative methods in this paper by creating a world model—a surrogate simulator. The world model is created with deep learning, trained on recorded bucket fillings from expert operators, and simulates the dynamics of the bucket-pile environment. It is then used to train an RL agent that is transferred to the real world, as shown in Fig. 1. The main advantage of world models compared to physics-based simulators is that a world model simulator [22] is faster and can run faster than real-time. Another advantage of world models is that they do not require domain adaptation or sim-to-real transfer methods; they can instead be directly deployed to the real system. The downside is that world models instead require a lot of training data to be varied enough for different scenarios.

In this paper, we also demonstrated the performance and robustness of the RL agent trained in the surrogate simulator and compared it to two other NN controllers. The first NN controller is synthesized with IL, and the second controller is pre-trained with IL and fine-tuned with RL on the target material used for evaluation. Finally, we make the following contributions in this paper:

- A bucket filling dream environment created using world models trained from data consisting of normal work cycles recorded in various and unknown worksites, and from different operators.
- A high-performing controller was synthesized using RL in the dream environment and transferred to the real world directly. This is the first work to create a bucket filling controller using world models in combination with RL.
- We demonstrated the performance and robustness of the controller on a full-scale wheel loader in a field test and compared it to benchmark methods.

2. Related work

The methods used in this paper are mostly inspired by the world models paper in Ha and Schmidhuber [22], where they trained a world model on a car racing and Doom game environment. They combined a Long Short-Term Memory (LSTM) [23], which is a type of Recurrent Neural Network (RNN), with a Mixture Density Network (MDN) [24] to predict the next vector in latent space. They used RGB images as inputs to the system that was first compressed to latent space with a Variational AutoEncoder (VAE). The training data for the world model was collected by a random agent for 10 000 episodes. They showed that it was possible to successfully train a policy completely inside the world model environment—a dream environment where the agent imagines the episode in the latent space without needing access to the real one. The trained agent was then redeployed to the original environment with high performance. Our environment only uses low-dimensional sensory inputs instead of high-dimensional images, and therefore it is not necessary for us to encode the observations into latent space using a VAE, and we can thus predict the next observations directly.

The LSTM in combination with an MDN network was also used in Graves [25], for generating handwriting sequences. The network was trained on a dataset of handwriting collected from different college professors, and they showed that it was possible to predict the next letter stroke and synthesize words in a specific handwriting style.

A related approach to the world models described earlier was used in Hafner et al. [26,27], Wu et al. [28], where the authors created an algorithm called Dreamer and applied it firstly to different Atari games. They also applied it to real-world robots, such as a quadruped robot and a pick-and-place robotic arm. The dreamer algorithm learns a world model using Recurrent State-Space Model (RSSM) [29], modeled using Gate Recurrent Units (GRU) (an RNN similar to LSTM), and leverages it to imagine future outcomes based on the given actions.

The dynamics model consists of three modules: a transition model that predicts the state of the Markov Decision Process (MDP), an observation model that predicts the observed state, and a reward model that predicts the rewards of the current state. It is then used to learn a policy and value function, which are also parametrised using neural networks. The learning algorithm is iteratively updating the dynamics and behavior models, and collecting new data from the environment by interacting with it using the latest policy combined with exploration noise, until the process has converged. It learns in the latent imagination for a certain prediction horizon and not the entire episode.

The authors showed that it is feasible to learn a policy directly in the real world without simulators for selected robots. However, the training time varies significantly depending on the robot and the problem to solve. They could learn a quadruped robot to walk within one hour, but learning the pick-and-place robots to grasp different objects took between 8–10 h [28]. These robots and environments are relatively easy to reset after an episode, either manually or automatically. But it is more challenging with wheel loaders since the material in the bucket has to be weighed, resetting the pile, and driving the machine into the correct starting position. There are also safety concerns when starting with a random bucket filling policy, where the wrong action might damage the machine.

World models can be considered as a type of non-linear Kalman filters. Kalman filters [30] also predicts the next state given a prior observation and the command signals. Traditional Kalman filters combined with deep neural networks are an alternative to LSTM and other RNN architectures. A few alternatives are the Recurrent Kalman Network (RKN) [31] which learns a latent space representation and linearization, and Backprop KF (BKF) [32] which combines a feedforward network with a Kalman filter. A similar idea to BKF was used for model adaptation and few shot sim-to-real transfer in Arndt et al. [33]. The authors in these papers demonstrated that their variant of the Kalman filter performed slightly better than LSTMs on the selected tasks. However, they have so far only been used on a limited number of problems, while LSTMs are more popular, tested on various tasks, and easier to implement using existing libraries.

3. Background

In this section, we provide background information about world models, and their underlying architectures, such as LSTMs and MDNs, as well as the preliminaries for Reinforcement Learning (RL).

3.1. World models

A world model is a representation of how an agent views the world and is created with data-driven methods, more specifically, with data consisting of interactions with the world. After training, the world models are a compact representation of the environment, which is, in our case, the dynamics between the bucket and the pile during the loading phase. This model of the world is, by design, limited to a specific part of the environment and not the entirety of it. Just like humans have a mental model of the surrounding world that is limited and constrained to what is useful for us [22].

The world model outputs the probability distribution of the next observation and the approximate material weight as a mixture of Gaussian given the previous observation and actions taken. In other words, it will model $\Pr(\boldsymbol{y}_{t+1}|\boldsymbol{x}_t, \boldsymbol{a}_t, \boldsymbol{h}_t)$, where $\boldsymbol{y}_{t+1} = (\boldsymbol{x}_{t+1}, m_T)$ is the next observation and final material weight, $\boldsymbol{x}_t$ and $\boldsymbol{a}_t$ are the input observations and actions at time t, and $\boldsymbol{h}_t$ is the hidden state of the LSTM layer.

The LSTM is a type of RNN but with additional logic to reduce problems, such as the vanishing gradient problem. This issue appears during backpropagation through time when the gradient becomes too small or too big for long sequences of data, resulting in underflow or overflows computational errors. The LSTM has two additional internal state variables: the hidden state h_t, and the cell state c_t. The hidden state, h_t is the working memory, i.e., the short-term memory, and the cell state, c_t, represents the long-term memory. The cell state distinguishes the LSTM from the standard RNN and gives it the capability to choose which points in the past data to remember and which to forget [23].

The neural network architecture for the world model is shown in Fig. 2 and consists of one layer of LSTM units, where the hidden state is connected to an MDN as the second layer. The network predicts the probability π^i, mean $\boldsymbol{\mu}^i$, and standard deviation $\boldsymbol{\sigma}^i$, for a mixture of Gaussians distribution, where the predicted next observation and material weight are sampled from. The distribution consists of M mixtures, and the mean, $\boldsymbol{\mu}^i$, and standard deviation, $\boldsymbol{\sigma}^i$, vectors are the

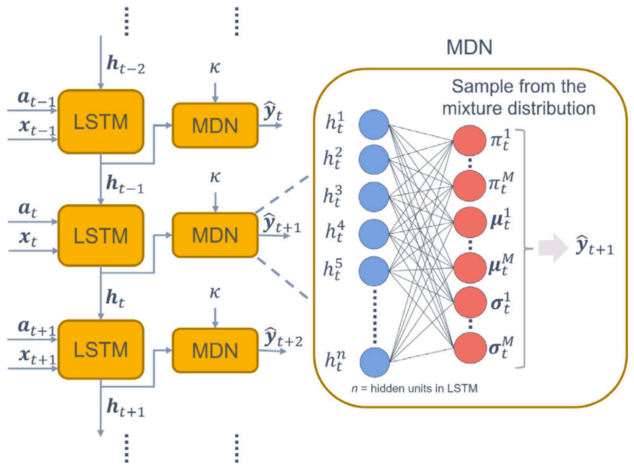


Fig. 2. Architecture of the world model with an LSTM and MDN layer.

same length as $\hat{y}$—the number of features to predict. Furthermore, the MDN network also takes the temperature κ as an input, which adjusts the overall variance of the sampled prediction: $\kappa = 0$ results in the most probable mean output, $\kappa = 1$ has no effect, and $\kappa > 1$ results in a higher variance of predictions [22].

The three output variables from the world model: π^i, $\boldsymbol{\mu}^i$, $\boldsymbol{\sigma}^i$ are activated with Eqs. (1), (2), and (3) respectively, to force the values into correctly defined ranges, where $\hat{\pi}_t^i$, $\hat{\boldsymbol{\mu}}_t^i$, and $\hat{\boldsymbol{\sigma}}_t^i$, are the outputs from the last neurons.

$$\pi_t^i = \frac{\exp(\hat{\pi}_t^i)}{\sum_{i=1}^{M} \exp(\hat{\pi}_t^i)}, \quad \pi_t^i \in (0,1), \sum_{i=1}^{M} \pi_t^i = 1 \tag{1}$$

$$\boldsymbol{\mu}_t^i = \hat{\boldsymbol{\mu}}_t^i, \quad \boldsymbol{\mu}_t^i \in \mathbb{R} \tag{2}$$

$$\boldsymbol{\sigma}_t^i = \exp(\hat{\boldsymbol{\sigma}}_t^i), \quad \boldsymbol{\sigma}_t^i > 0. \tag{3}$$

We use an MDN layer because the output from conventional NN results in the mean value of the target data. This leads to poor performance in multimodal situations [24], such as the bucket filling task, where different materials results in different modalities.

The output from the network is then interpreted as a probability density function in the form of a linear combination of Gaussian kernel functions:

$$\Pr(\boldsymbol{y}_{t+1}|\boldsymbol{x}_t, \boldsymbol{a}_t, \boldsymbol{h}_t) = \sum_{i=1}^{M} \pi_t^i \mathcal{N}(\boldsymbol{y}_{t+1}|\boldsymbol{\mu}_t^i, \boldsymbol{\sigma}_t^i) \tag{4}$$

where π^t are the mixing weights, and $\mathcal{N}(\boldsymbol{y}_{t+1}|\boldsymbol{\mu}_t^i, \boldsymbol{\sigma}_t^i)$ is the multivariate Gaussian of the form:

$$\mathcal{N}(\boldsymbol{y}|\boldsymbol{\mu}, \boldsymbol{\sigma}) = \frac{1}{\sqrt{(2\pi)^k |\boldsymbol{\Sigma}|}} \exp\left[\frac{-1}{2}(\boldsymbol{y}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{y}-\boldsymbol{\mu})\right] \tag{5}$$

$\boldsymbol{\Sigma}$ is the covariance matrix, $|\boldsymbol{\Sigma}|$ is the determinant, and k is the rank of the covariance matrix.

We assume that the features are statistically independent to simplify the calculations and to reduce the number of output neurons in the network. The Gaussian mixture model is capable of approximating any density function given enough mixture components, and when the means and standard deviations are correctly chosen [24]. This results in a simplified covariance matrix $\boldsymbol{\Sigma}$, on a diagonal form with the variances on the diagonal and with the rank equal to the number of features in $\boldsymbol{y}_t$. The covariance matrix $\boldsymbol{\Sigma}$ and its inverse $\boldsymbol{\Sigma}^{-1}$ becomes:

$$\boldsymbol{\Sigma} = \begin{pmatrix} \sigma_1^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & \sigma_k^2 \end{pmatrix} \tag{6}$$

$$\Sigma^{-1} = \begin{pmatrix} \frac{1}{\sigma_1^2} & 0 & \cdots & 0 \\ 0 & \frac{1}{\sigma_2^2} & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & \frac{1}{\sigma_k^2} \end{pmatrix} \tag{7}$$

Eq. (5) is simplified with the diagonal covariance matrix to:

$$\mathcal{N}(\boldsymbol{y}|\boldsymbol{\mu},\boldsymbol{\sigma}) = \frac{1}{\sqrt{(2\pi)^k}\prod_{j=1}^{k}\sigma_j^2}\exp\left[\frac{-1}{2}\sum_{j=1}^{k}\frac{(y_j-\mu_j)^2}{\sigma_j^2}\right] \tag{8}$$

3.2. Reinforcement learning

We formalize the learning problem using the Reinforcement Learning (RL) context. The base component of any RL problem is the Markov Decision Process (MDP), which consists of four main components: observed state space $\boldsymbol{x}_t \in \mathcal{O}$, action space $\boldsymbol{a}_t \in \mathcal{A}$, transition dynamics probability function $p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t,\boldsymbol{a}_t)$, and reward $r_t(\boldsymbol{x}_{t+1},\boldsymbol{x}_t,\boldsymbol{a}_t) \in \mathcal{R}$. At each time step, the agent receives the observed state, $\boldsymbol{x}_t$, and reward r_t, then acts according to a stochastic policy $\pi(\boldsymbol{x}_t|\boldsymbol{a}_t)$ with the goal of maximizing the accumulated reward during the entire episode $R_T = \sum_{k=t}^{T}\gamma^{k-t}r_t(\boldsymbol{x}_{t+1},\boldsymbol{x}_t,\boldsymbol{a}_t)$. γ is the discount factor, and T is the terminal state [34].

A common and popular deep RL algorithm is Proximal Policy Optimization (PPO) [35], which is an actor–critic algorithm where the actor is the policy function $\pi(\boldsymbol{x}_t|\boldsymbol{a}_t)$, and the critic evaluates the actions taken by the actor. Both of them are parametrised as separate neural networks. The PPO algorithm is model-free and requires little hyperparameter tuning compared to other deep RL algorithms.

The reward function, $r_t(\boldsymbol{x}_{t+1},\boldsymbol{x}_t,\boldsymbol{a}_t)$, is defined as a scalar input to the agent at each time step, and the definition of it defines how fast the agent learns and how well it achieves our goals. A badly designed reward function can result in the agent not acting as intended by finding ways to exploit it and earn high rewards without achieving the desired behavior. An example of this was explained in Ng et al. [36], where a football agent learned to "vibrate" the football by touching the ball with a very high frequency. The reward function was, in this case, possession—the amount of time a player has control of the ball. While this is a useful in the overall game, it is not sufficient to describe the true goal of football. A good reward function should communicate what you want to achieve and not how to do it [34].

Remark. Training an RL agent on a real wheel loader requires a well-engineered reward, as in our previous work [14], to keep the required experiment manageable, which is typically a hard job. In the dream world model, we only provide a sparse reward, consisting of the final material weight. Although this requires more episodes, it is not a problem in the dream environment because of the fast execution time.

4. Method

Here, we describe the input and output signals for the system, as well as the bucket filling algorithm. The method of creating a bucket filling controller using world models and RL is also described in Section 4.3.

4.1. Signals and commands

The input signals used for the automatic bucket filling are visualized in Fig. 3, and are the following: θ_{tilt}, θ_{lift}, F_{tilt}, F_{lift}, v, τ_{pull}. The angles: θ_{tilt} and θ_{lift} are both 0 when the bucket lies flat on the ground and increasing in the direction of the arrows. The forces, F_{tilt} and F_{lift}, are calculated by measuring the pressure at the rod and bottom side of the respective cylinders as well as the area. The velocity of the wheel loader, v, is measured from the rotational speed of the tires, and therefore the true velocity over ground is not measured. The last signal,

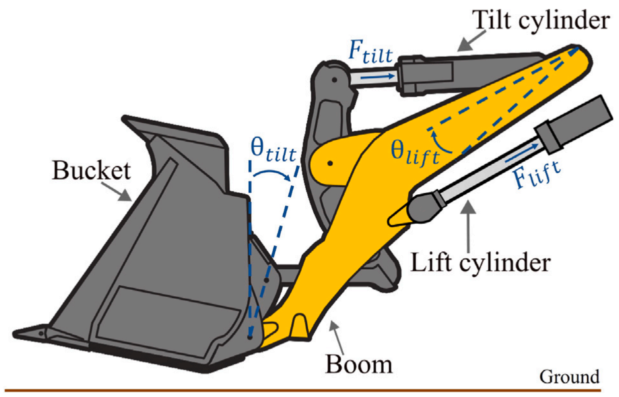


Fig. 3. Definition of the joint angles and forces.

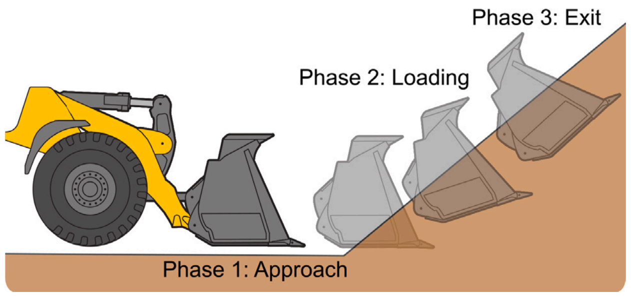


Fig. 4. Three phases of the bucket filling algorithm. The first phase brings the wheel loader into the digging position in the pile. The bucket filling controller is active in the second phase and diggs through the pile.

τ_{pull} is the rimpull torque, which relates to the traction force between the wheels and the ground.

The command signals used by the operator and the bucket filling controller are: u_{tilt}, u_{lift}, $u_{throttle}$. The two command signals, u_{tilt} and u_{lift}, control the bucket, the end effector of the wheel loader, by increasing or decreasing the angles θ_{tilt}, and θ_{lift}, respectively. The commands are in the range $[-1, 1]$, where a positive signal indicates a command to increase the angles and a negative signal indicates the opposite. The last command signal, $u_{throttle}$, controls the throttle of the engine and is on the interval $[0, 1]$.

The sensors for measuring these signals are already equipped on the base machine. Therefore, no modifications were needed on the wheel loaders used in data collection and the field test.

4.2. Bucket filling algorithm

The bucket filling algorithm used in this paper is the same as in our previous work [10,14,37], which is similar to the approach in Dadhich et al. [6,11]. The steps are summarized in Fig. 4, and consists of 3 phases: approach, loading, and exit.

1. *Approach:* The bucket is placed flat on the ground, and then the wheel loader is accelerated with 50% throttle command towards the pile.
2. *Loading* The loading phase starts when the force in the lift cylinder exceeds a pre-defined threshold. The force in the lift cylinder increases as the bucket starts to penetrate the pile. The neural network controller is outputting the control commands u_{tilt}, u_{lift}, $u_{throttle}$.
3. *Exit* The exit phase, tilts the bucket completely in to prevent material from spilling out, marks the end of the bucket filling process.

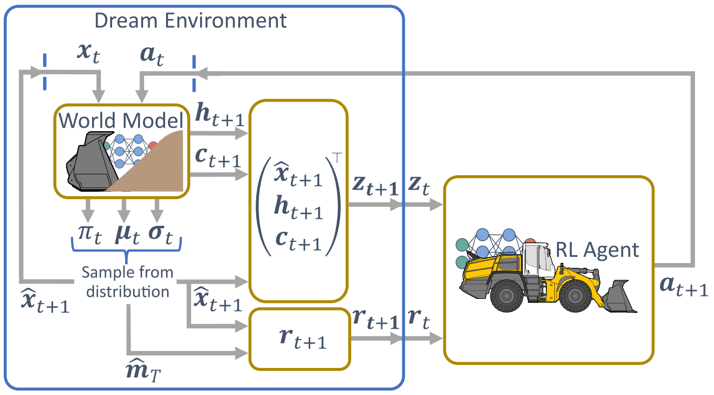


Fig. 5. Overview of the gym environment and the training of the RL agent.

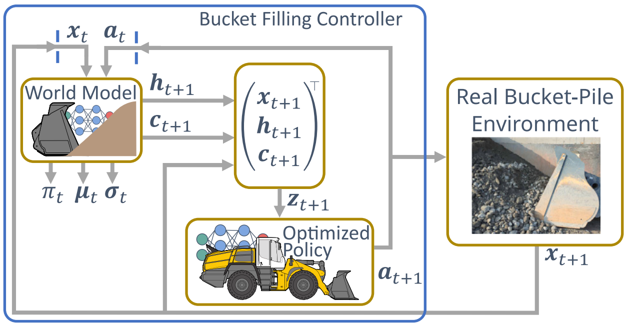


Fig. 6. Overview of the *wm_controller* used in the field test.

The bucket filling process is divided into phases to simplify the problem, and the algorithm assumes that the wheel loader is already positioned in front of the digging spot since no sensors for detecting the pile are used. Furthermore, the neural network controller is only controlling the u_{tilt}, u_{lift}, $u_{throttle}$ in phase 2 when the bucket is in the digging position.

4.3. Synthesizing a controller

We synthesize the bucket filling controller in three steps, as shown in Fig. 1, and described in the following sections.

4.3.1. Creating a world model

We create a world model simulator by collecting a large amount of bucket filling data from expert operators. The training data consists of the input signals and operator commands described in Section 4.1.

We use the probability density from Eq. (4) where $\boldsymbol{y} = (\boldsymbol{x}, m_T)$, $\boldsymbol{x} = (\theta_{tilt}, \theta_{lift}, F_{tilt}, F_{lift}, v, \tau_{pull})$, m_T is the final material weight, and $\boldsymbol{a} = (u_{tilt}, u_{lift}, u_{throttle})$.

The probability given by the world model for predicting the observation sequence $\boldsymbol{y}$ is given by:

$$\Pr(\boldsymbol{y}) = \prod_{t=1}^{T} \Pr(\boldsymbol{y}_{t+1} | \boldsymbol{x}_t, \boldsymbol{a}_t, \boldsymbol{h}_t) \tag{9}$$

The loss function, $\mathcal{L}(\boldsymbol{x}_t, \boldsymbol{a}_t, \boldsymbol{h}_t)$, for the observation sequence used in training the world model is then the negative logarithm of $\Pr(\boldsymbol{y})$. The loss is calculated for all the predicted observations in the bucket filling sequence, and the material weight is predicted only once at the last time step T using the entire data trajectories: $\boldsymbol{x}$, $\boldsymbol{a}$, and $\boldsymbol{h}$. This gives the following loss function:

$$\mathcal{L}(\boldsymbol{x}) = -\log\left[\Pr(m_T | \boldsymbol{x}, \boldsymbol{a}, \boldsymbol{h})\right] - \sum_{t=1}^{T-1} \log\left[\Pr(\boldsymbol{x}_{t+1} | \boldsymbol{x}_t, \boldsymbol{a}_t, \boldsymbol{h}_t)\right] \tag{10}$$

The log of the probability density function in Eq. (4) can be written as:

$$\log\left[\Pr(\boldsymbol{y}_{t+1} | \boldsymbol{x}_t, \boldsymbol{a}_t, \boldsymbol{h}_t)\right] = \log\left[\sum_{i=1}^{M} \exp\left(\log\left[\pi_t^i\right] + \log\left[\mathcal{N}(\boldsymbol{y}_{t+1} | \boldsymbol{\mu}_t^i, \boldsymbol{\sigma}_t^i)\right]\right)\right] \tag{11}$$

Similarly, Eq. (8) can be rewritten as:

$$\log\left[\mathcal{N}(\boldsymbol{x} | \boldsymbol{\mu}, \boldsymbol{\sigma})\right] = \frac{-1}{2}\sum_{j=1}^{k} \frac{(x_j - \mu_j)^2}{\sigma_j^2} - \frac{k}{2}\log(2\pi) - 2\sum_{j=1}^{k}\log(\boldsymbol{\sigma}_j) \tag{12}$$

4.3.2. Training a bucket filling policy with RL

The world model simulator is used in this step as a fast surrogate simulator (replacing the role of a traditional physics-based simulator) to create a dream environment where an RL agent can explore and optimize its bucket filling behavior. The implemented dream environment follows the gym API [38]. At each time step, the environment receives the action from the agent as input and returns the next observation, reward, and state of the episode termination. An overview image is shown in Fig. 5.

The input to the agent is defined as $\boldsymbol{z}_t = (\hat{\boldsymbol{x}}_t, \boldsymbol{h}_t, \boldsymbol{c}_t)$, where $\hat{\boldsymbol{x}}_t$ is the predicted observation from the world model, $\boldsymbol{h}_t$ is the hidden state of the LSTM, and $\boldsymbol{c}_t$ is the cell state of the LSTM. Including the hidden and cell states from the LSTM results in the agent receiving temporal information about the bucket filling process instead of only the instant observation.

The gym environment takes the temperature κ, and a dataset with bucket fillings as parameters before starting the training. The temperature, κ, controls the variance in the world model. $\kappa = 0$ results in the outputs being deterministic, and only the most likely output is always chosen, while $\kappa > 1.0$, makes the world model increasingly more unpredictable. This results in a more difficult environment and makes it more challenging for the RL agent, since it must consider the randomness of the world model while optimizing its policy. The dataset with bucket fillings is used to prime the world model to generate observations in the style of a particular pile. Priming was used for a similar NN model in Graves [25], where they used it for generating handwriting in the style of a certain person. The target handwriting had to be in the dataset during training for priming to work; therefore, we can only simulate piles that are already in the dataset.

The world model was primed by feeding it observations and actions from a random episode with sample lengths of 10% of the total length and starting at the beginning of the bucket filling.

The RL agent was initialized with a completely random policy and critic, and the maximum episode length was set to 22.5 s. If an episode

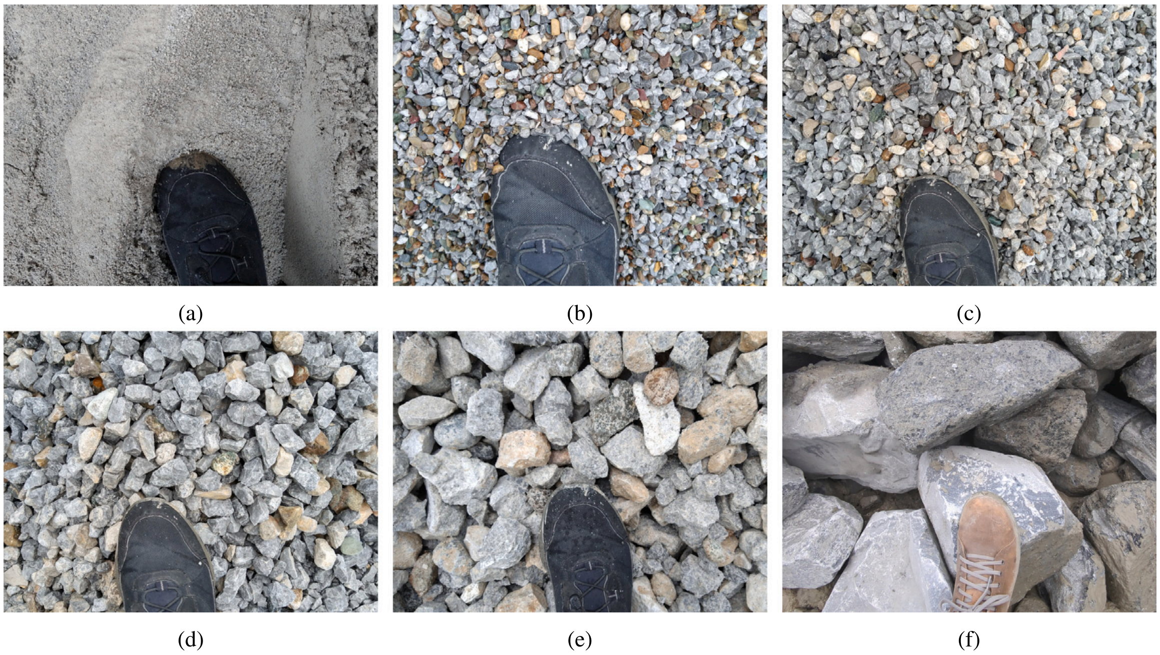


Fig. 7. Selected sample of the materials collected in the training data. (a) sand 0–4 mm (b) gravel 3–10 mm (c) gravel 8–16 mm (d) gravel 16–32 mm (e) gravel 32–63 mm (f) blasted rock 0–400 mm.

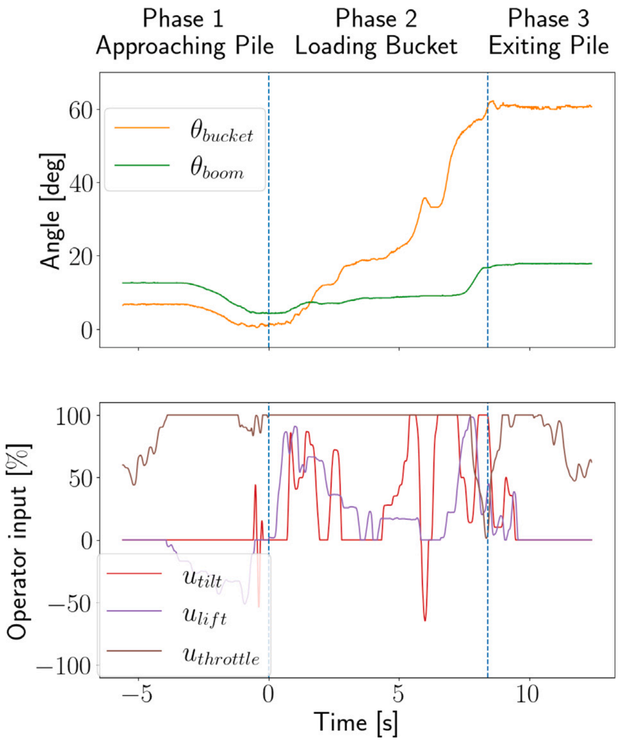


Fig. 8. Example of the operator commands during a bucket filling and the corresponding joint angles. The three phases described in Section 4.2 are also marked out on the plot.

took longer than that, it would be marked as truncated, and the agent would not receive any terminal reward.

Our previous work in Eriksson et al. [14] showed that the RL training was improved if the time between each step in the RL algorithm was matched to the delays of the hydraulic system, which is in our case 225 ms. The same modifications were used in this paper, but the world model updated its state at the default sampling time of 15 ms.

The main goal of the bucket filling is to achieve a high fill grade, which can be calculated approximately using the material weight as a proxy. Therefore, we chose the following reward function:

$$r_t(\boldsymbol{y}_t, \boldsymbol{a}_t) = 0 \tag{13}$$

$$r_T(\boldsymbol{y}_T, \boldsymbol{a}_T) = \min\left(\frac{m}{w_1}, 1.0\right) \tag{14}$$

Eq. (13) is the step reward, and Eq. (14) is the terminal reward at the end of the episode at time-step T, m is the weight of the material in the bucket in kg at the end of the episode. w_1 is the target loaded material weight that the agent should achieve. This number is determined according to the specific wheel loader, bucket, and pile combination. It should be selected so that the bucket is full when $\frac{m}{w_1} = 1$.

We chose sparse rewards calculated only with the material weight because it is the most important goal for the bucket filling controller to achieve. A fast bucket filling time is also important, and by setting the $\gamma < 1.0$, we can incentivize the agent to load the bucket as fast as possible when our step reward is 0.

A dense and more complicated reward function could speed up the training at the risk of the agent finding behaviors to abuse the reward function, which are most likely not beneficial for us. The trade-off for a simpler reward function against training time is not an issue with the surrogate simulator, since it runs faster than real time. It is thus capable of completing a large amount of episodes in a feasible amount of wall clock time.

4.3.3. Deployment on the real machine

The trained bucket filling controller, denoted in this paper as *wm_controller*, is deployed on the real machine without any modifications or optimization of the weights. The controller assumes $\boldsymbol{z}_t$ as input, and therefore the world model is running in the background to produce the necessary inputs h_t, and c_t, for the controller, as shown in Fig. 6. In this stage, the outputs from the MDN: π, $\boldsymbol{\mu}$, and $\boldsymbol{\sigma}$, are ignored and discarded.

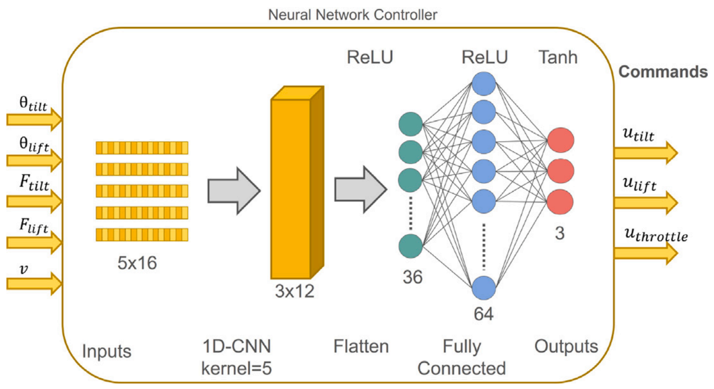


Fig. 9. Architecture of the *baseline* and *baseline_rl* controller from [14].

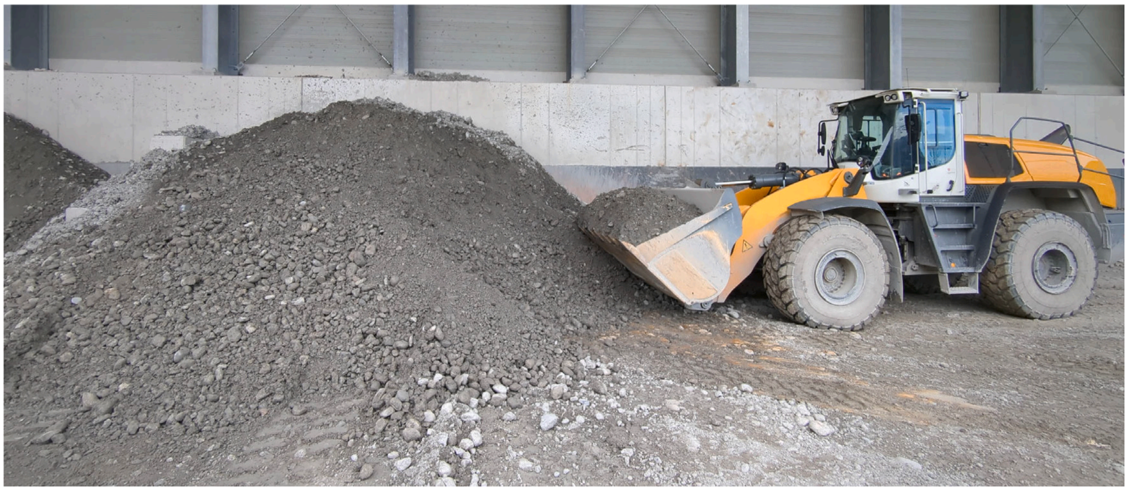


Fig. 10. Wheel loader and pile used for testing the neural network controller.

5. Experiment setup

5.1. Training data

The training data used to create the world model explained in Section 3.1 was collected from two different worksites and from two different types of wheel loaders: a 24-tonnes Liebherr L576, and a 33-tonnes Liebherr L586. The two worksites were quarries, which blast rock from the mountain and crush it into different sizes. At *worksite 1* the wheel loader handles both the blasted material for loading into a crusher and the crushed material for loading trucks. The wheel loader at *worksite 2*, only handles the blasted rock. Therefore, the two wheel loaders handle a wide range of materials with different kernel sizes, shapes and densities. It loads, for example, different fine-grained sand, gravel, and blasted rock. A selected sample of the materials that were collected is shown in Fig. 7.

The data was recorded in the background during normal operation, where the operators carried out their normal tasks without any specific instructions on how to load the different materials. This results in a natural and varied dataset with different pile shapes, and driving and weather conditions, as well as different operator loading strategies.

The recordings resulted in long time series sampled at 15 ms, which contained bucket fillings as well as other tasks irrelevant to us. We used the same automatic labeling technique developed in our previous work in Eriksson and Ghabcheloo [10], to extract the useful bucket fillings from the training data. The data was also filtered to exclude bucket fillings that took longer than 15 s and with less material than 6 000 kg. The detected bucket fillings outside this range have a higher risk of being false positives, or the operator might have a reason for slow loading times or loading less than half-full buckets. This is impossible to know by looking at the data, so it is better to discard them. Fig. 8 shows an example of the operator commands during a digging and the corresponding joint angles θ_{tilt}, and θ_{lift}.

We collected data during 5 months of operation and used the automatic labeling technique from [10] to extract the bucket fillings from the time-series data. This resulted in approximately 20 000 bucket fillings from the different materials.

Another dataset was also recorded using a third machine and a different operator. This dataset was used for validating the world model and was not part of the training data. Furthermore, it was collected from the same material and pile that was used for the real-world tests.

5.2. Training details

The world model was trained with the training data from the previous Section, and 95% of the data was used for training and the remaining 5% as test data. The world model network was implemented using PyTorch [40], consisting of 1 LSTM layer with 512 hidden units, and $M = 3$ mixtures for the MDN layer. In total, the world model has 1 096 237 trainable parameters and was optimized using the ADAM algorithm [41].

We used the PPO algorithm included in the Stable-Baselines3 package [42] using the hyperparameters from Table 1. The actor and critic were modeled as separate networks with the same architecture; a fully connected feedforward network with one hidden layer of 64 neurons activated by the hyperbolic tangent function. The policy network was chosen as a simple NN because all the complexity is incorporated in the world model that produces the inputs for the policy network.

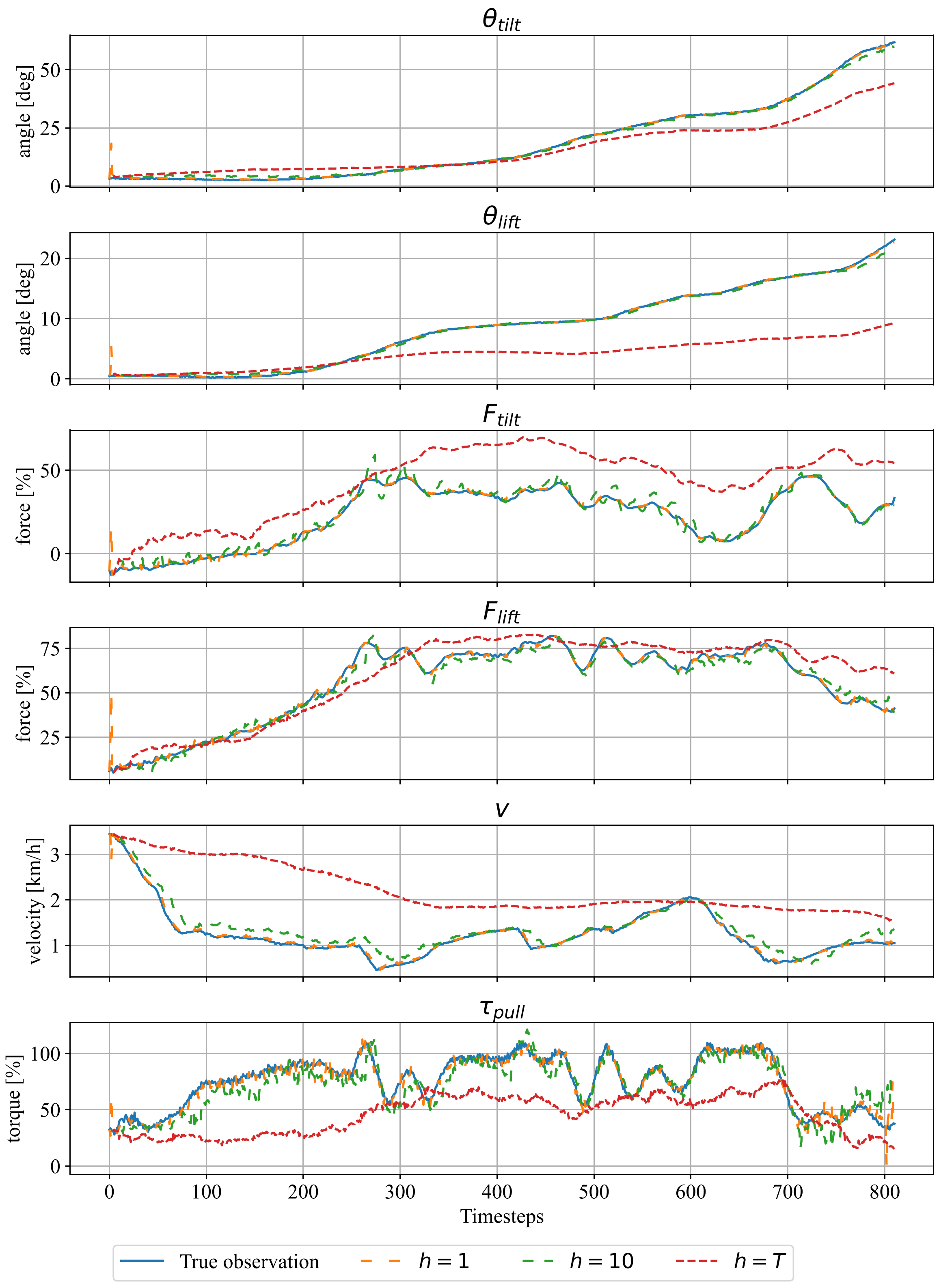


Fig. 11. Predicted observations from the world model using different prediction horizons.

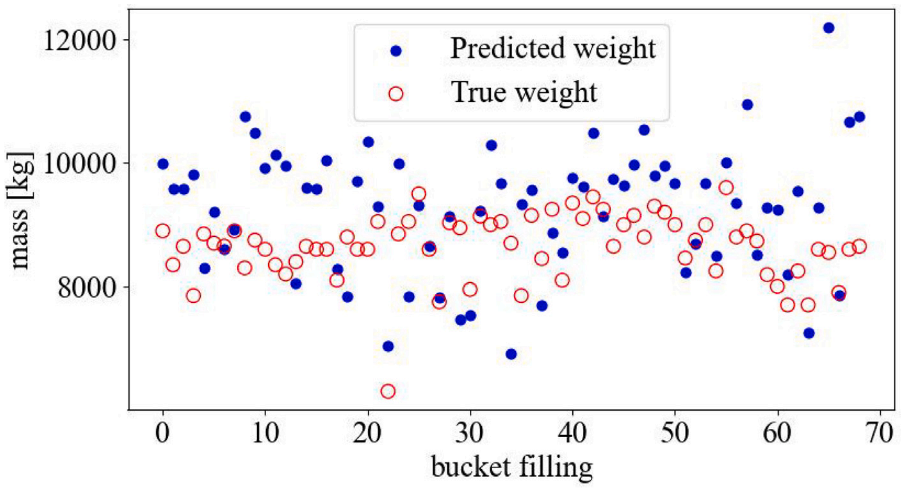


Fig. 12. True and predicted weights from the world model.

Table 1
Hyperparameters for the PPO algorithm and the gym environment.

Hyperparameter	Value	Description
γ	0.99	Discount factor
α	1e−05	Learning rate for the optimizer
λ_{gae}	0.95	Factor for trade-off of bias vs variance for GAE [39]
clip_range	0.2	Clipping parameter
ent_coef	0.01	Entropy coefficient for the loss calculation
n_epochs	10	Number of epochs when optimizing the surrogate loss
batch_size	64	Minibatch size
n_steps	512	Size of the rollout buffer
vf_coef	0.5	Value function coefficient for the loss calculation
max_grad	0.5	The maximum value for the gradient clipping
τ	1.1	Temperature of the world model
w_1	10 000	Target value for the loaded material weight

The RL agent was trained for 30 000 episodes, and we used a separate environment for testing and reporting the rewards and episode lengths. The test environment was set up in the same way as the training environment, but the agent does not use any exploration noise.

The world model and RL agent were both trained on a consumer PC with an Intel i7 12 cores CPU, 16 GB of RAM, and an Nvidia GeForce RTX 2070 Mobile GPU with 8 GB of VRAM.

The training time for the world model was approximately 10 h, and the training time for the RL agent was approximately 3 h.

5.3. Baseline controllers

The *wm_controller* is compared against two baseline controllers as well as a human expert operator. The first baseline controller was developed in our previous work in Eriksson and Ghabcheloo [10],Eriksson et al. [37], where it showed the overall best performance and robustness. It was synthesized with IL from a dataset consisting of about 100 bucket fillings collected from an expert operator at a worksite loading blasted rock (0–200 mm). The baseline controllers have five inputs: θ_{tilt}, θ_{lift}, F_{tilt}, F_{lift}, v, and it takes a time window of 16 samples of each input. The inputs are passed through a 1-D Convolutional Neural Network (CNN) with three output channels, which are then flattened and sent to a fully connected layer, which outputs the command signals: u_{tilt}, u_{lift}, $u_{throttle}$. The full architecture is shown in Fig. 9.

The second baseline controller, *baseline_rl*, uses the same architecture and imitation learning technique as the baseline controller, but it was optimized on the testing pile using RL. This is the same model and method that were used in our previous work in Eriksson et al. [14]. The controller was fine-tuned for 20 episodes on the target pile.

5.4. Evaluation

We evaluate and test the controllers in a field test using a Liebherr L576 24-tonnes wheel loader and a gravel-like material (0–64 mm), as shown in Fig. 10. The performance and robustness are compared between the three controllers by measuring the loaded material weight in the bucket, the loading time, and the number of successful bucket fillings. The loaded material weight in the bucket is weighed after each bucket filling using the internal weighing system, and the loading time is defined as the time from pile penetration to a certain tilt angle, θ_{tilt}, is reached. A bucket filling is considered successful if the controller manages to load the bucket without getting stuck in the pile.

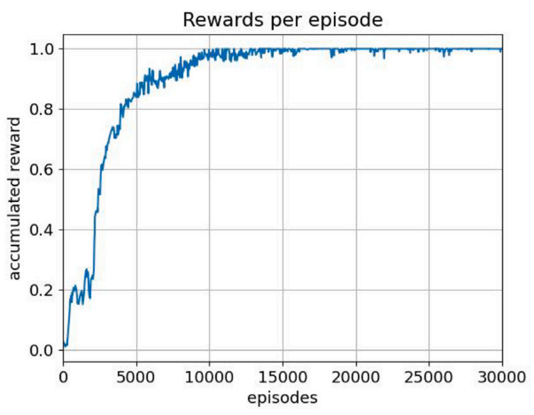
The bucket filling controller runs as a node using the Robotic Operating System 2 (ROS 2) [43], on a consumer PC connected via the CAN-bus to the machine's internal controller for receiving input signals from the sensors and transmitting commands. Each of the three controllers was evaluated 10 times on the same pile.

6. Results and discussion

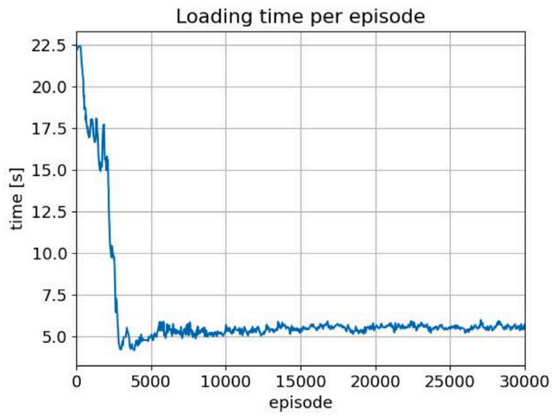
This section reports the results from the training of the world model, training the RL agent, and the final performance of the controller deployed on the real machine during the field test.

6.1. World model

The world model was trained using the training data from Section 5.1, and the prediction capability was evaluated on a second validation dataset that was not used during training or testing. As mentioned in Section 5.1 this dataset was collected from a third machine and from the same pile used in the field test.



(a)



(b)

Fig. 13. Rewards and loading time for the RL-agent during training (a) Accumulated rewards from each episode (b) Average loading time for each episode.

The prediction capabilities of the world model were evaluated using three different prediction horizons: $h = 1$, $h = 10$, and $h = T$, which are shown in Fig. 11. For the first horizon, $h = 1$, the world model predicts the observation for the next time step, which is 15 ms, given the last true observation and the commands from the operator. Similarly, $h = 10$, predicts the next 10 time steps, which is 150 ms into the future, given one true observation and the commands from the operator during the prediction horizon. After 10 time steps, the next true observation is fed to the model, and the prediction cycle repeats. For the last prediction horizon, $h = T$, the world model predicts the entire bucket filling starting with a true observation and then only using the commands from the operator and the previously predicted observation.

For the two longer prediction horizons, $h = 10$, and $h = T$, the world model is first warmed up using the first four true observations to get more reliable results because of the high uncertainties in the state of the model. This is illustrated in Fig. 11, where the orange curve has a high deviation from the true observation for the first few time steps.

We can see from Fig. 11, that with a next-step prediction horizon, $h = 1$, the world model is very capable of predicting the observed state. This capability decreases the longer prediction horizon we test with. Using a prediction horizon of $h = 10$, the world model is still very close to the true values, except for some oscillations in the tilt cylinder force F_{tilt}. The performance degrades when predicting the entire bucket filling sequence, but they are still close to the true values, and the predicted values follow the general shape of the true observations but with an offset.

The world model was also designed to be able to predict the weight of the material at the end of a bucket filling, which is used in calculating the reward function in step 2. The predicted weights from the world model are shown in Fig. 12, with a Mean Absolute Percentage Error (MAPE) of 10.5% for the validation dataset.

The results in Fig. 12 shows that the world model is capable of predicting reasonable weights close to the true value but overestimates it more often than underestimates it. The figure also indicates that the spread of the predicted loaded material weights is higher than the true one. One reason for this is that predicting the material weight using only the data during the loading phase is challenging because the forces observed during digging are also affected by the pile properties. These properties are different depending on the material, and the shape and size of the pile. The true weight can only be estimated after the bucket filling has been completed and is free from the pile, which we do not have in the training data.

6.2. RL agent

The result from the second step, training an RL-agent using the world model as a fast surrogate simulator, is shown in Fig. 13, as the accumulated reward per episode as well as the loading time of each bucket filling episode.

Early tests indicated that it was necessary to prime the world model before each episode during training to have satisfactory bucket filling performance in the field test. Without priming, the RL agent can cheat the world model by forcing it to predict easy-to-load piles, such as a sand pile, to earn high rewards without being able to load more challenging materials, such as blasted rock. With priming, we set the state of the world model to output a certain pile from the training data and its properties. Another strategy to avoid this issue is to divide the training data into different piles, train one world model for each material, and then train the RL agent on all the different world models. But the problem with this strategy is that then you also have to select the correct world model for the material that is being loaded when using it in the field.

The RL training converges after about 10 000 episodes, as shown in Fig. 13, starting from a random policy. This would not be feasible to do in the real world, but using the world model as a fast surrogate simulator, we can do it in approximately 3 h.

We can see in Fig. 13 (b), that the RL-agent is capable of optimizing the loading time even though the loading time was not explicitly part of the reward function in Eq. (14). As mentioned earlier, this is due to the discount factor $\gamma = 0.99$, and thus future rewards are valued less than immediate rewards. Furthermore, since we only have a non-zero reward at $t = T$, the agent is optimizing to reach that goal as quickly as possible. Another thing to consider, is the properties of the bucket filling problem. A slow bucket filling strategy does not necessarily mean a high material weight, and a quick loading time can also yield a high material weight.

There is a risk that the RL agent learns behaviors that are not possible in the real world but only in the world model simulator because it will be incomplete—the range of possible predictions is limited by the training data. Therefore, it is possible for the agent to cheat the world model and find a behavior that accumulates high rewards in the simulator but is nonsense in the real world. The authors from [22] found that tuning $\kappa > 1.0$ makes it harder for the agent to exploit the world model. This will limit this risk, but not completely remove it. It is necessary to inspect the agent's behavior manually to see if it is reasonable.

6.3. Field test

Lastly, the *wm_controller* was deployed to the real machine and compared in a field test against the two baseline controllers, and a human expert operator. Each of the three controllers was tested 10 times on the same pile on the same day.

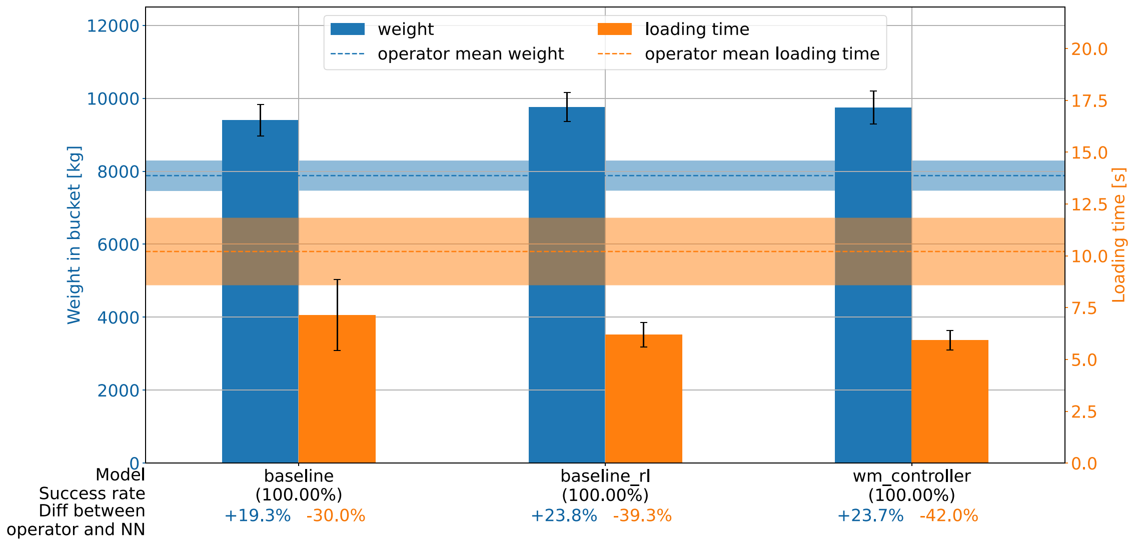


Fig. 14. Performance and robustness of the bucket filling controllers averaged over 10 trials. The horizontal bar represents the performance of the human operator.

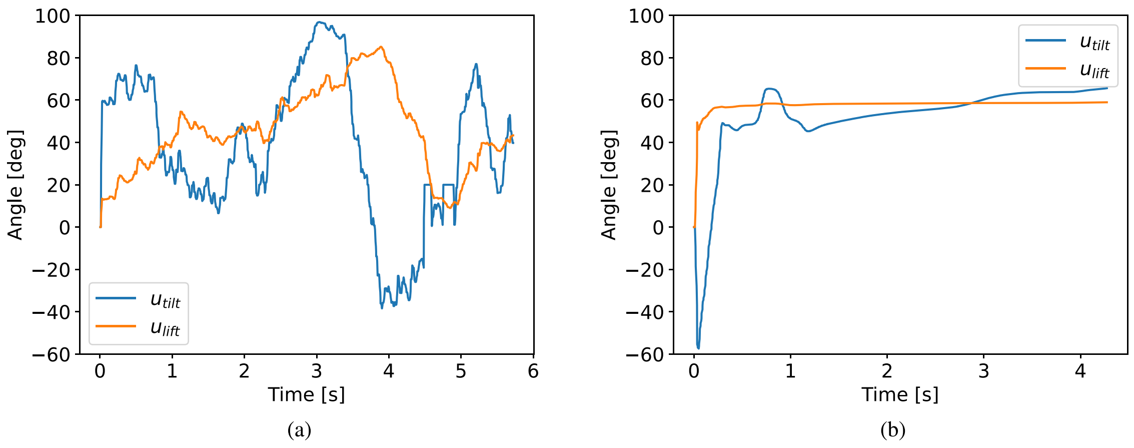


Fig. 15. u_{tilt}, and u_{lift} commands from (a) the *baseline* controller and (b) the *wm_controller*.

The performance of the three controllers is shown in Fig. 14. All of them could successfully load the bucket and could beat the human expert operator.

The *wm_controller* and *baseline_rl* have about the same performance and could get higher material weight and faster loading times than the *baseline* controller. They also have a more consistent performance compared to the operator and the *baseline* controller.

The output commands of the *wm_controller* and the *baseline* are shown in Fig. 15, where the two behaviors are very different from each other and the *wm_controller* commands are smoother and less noisy than the *baseline* controller.

The *baseline* controller commands are noisier because the controller tries to mimic the operator behavior, which usually has a variance in the commands during the bucket filling, as shown in Fig. 8. The RL-agent, on the other hand, is free to optimize its behavior without any prior human bias and is capable of finding an optimal digging behavior different from the optimal human digging behavior. The experiments show that smooth command signals outperform high-variance ones.

Interestingly, the *wm_controller* learns to tilt down at the beginning of the episode while lifting at the same time. It only does it for less than half a second, so the tilt angle, θ_{tilt}, does not change considerably during this time.

7. Conclusions

We have collected training data from different worksites, created a surrogate simulator using world models, and then trained an RL agent from scratch to solve the bucket filling problem. The RL-agent was transferred to and tested on a full-size wheel loader in a field test. It also had higher performance than our previous best bucket filling controller and was on the same level as a controller optimized for the target pile. The experiments also demonstrated that the RL agent was able to learn a very different digging behavior than human expert operators.

We have demonstrated the feasibility of our approach applied to the bucket filling problem for wheel loaders with a gravel-type material. But it was only possible to test the controller on this material, so the next step will be to test the controller on different materials and compare it to the previous best controllers. Furthermore, implementing a similar approach as the Dreamer algorithm in Wu et al. [28], which continuously adapts to the pile environment and updates the world model with new interactions, is also interesting to research in the future.

The world model simulator developed in this paper can also be used for testing the performance of other bucket filling controllers, such as those created by IL, and evaluating their performance. As we showed in Eriksson and Ghabcheloo [10], it is difficult to evaluate the performance of the controller by only comparing the predicting performance of the operator commands because this does not translate directly to closed-loop performance in a field test. With the world model, it is possible to test the closed-loop performance of a synthesized NN controller with different pile environments using priming, instead of testing it on the real machine.

Another interesting direction for future research is to investigate how the same approach can translate to other earth-moving HDMMs, such as excavators.

CRediT authorship contribution statement

Daniel Eriksson: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Reza Ghabcheloo:** Writing – review & editing, Supervision. **Marcus Geimer:** Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The authors do not have permission to share data.

Acknowledgments

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 858101.

References

[1] B. Brucker Juricic, M. Galic, S. Marenjak, Review of the construction labour demand and shortages in the EU, Buildings 11 (1) (2021) 17, http://dx.doi.org/10.3390/buildings11010017.
[2] B. Frank, L. Skogh, R. Filla, A. Froberg, M. Alaküla, On increasing fuel efficiency by operator assistance systems in a wheel loader, 2012, http://dx.doi.org/10.13140/RG.2.1.3129.1362, Unpublished.
[3] V. Nezhadali, B. Frank, L. Eriksson, Wheel loader operation—Optimal control compared to real drive experience, Control Eng. Pract. 48 (2016) 1–9, http://dx.doi.org/10.1016/j.conengprac.2015.12.015.
[4] B. Frank, J. Kleinert, R. Filla, Optimal control of wheel loader actuators in gravel applications, Autom. Constr. 91 (2018) 1–14, http://dx.doi.org/10.1016/j.autcon.2018.03.005.
[5] S. Dadhich, U. Bodin, U. Andersson, Key challenges in automation of earth-moving machines, Autom. Constr. 68 (2016) 212–222, http://dx.doi.org/10.1016/j.autcon.2016.05.009.
[6] S. Dadhich, F. Sandin, U. Bodin, U. Andersson, T. Martinsson, Field test of neural-network based automatic bucket-filling algorithm for wheel-loaders, Autom. Constr. 97 (2019) 1–12, http://dx.doi.org/10.1016/j.autcon.2018.10.013.
[7] E. Halbach, J. Kämäräinen, R. Ghabcheloo, Neural network pile loading controller trained by demonstration, in: 2019 International Conference on Robotics and Automation, ICRA, 2019, pp. 980–986, http://dx.doi.org/10.1109/ICRA.2019.8793468.
[8] W. Yang, N. Strokina, N. Serbenyuk, R. Ghabcheloo, J. Kamarainen, Learning a pile loading controller from demonstrations, in: 2020 IEEE International Conference on Robotics and Automation, ICRA, IEEE, 2020, pp. 4427–4433, http://dx.doi.org/10.1109/ICRA40945.2020.9196907.
[9] W. Yang, N. Strokina, N. Serbenyuk, J. Pajarinen, R. Ghabcheloo, J. Vihonen, M.M. Aref, J.-K. Kamarainen, Neural network controller for autonomous pile loading revised, in: 2021 IEEE International Conference on Robotics and Automation, ICRA, IEEE, 2021, pp. 2198–2204, http://dx.doi.org/10.1109/ICRA48506.2021.9561804.
[10] D. Eriksson, R. Ghabcheloo, Comparison of machine learning methods for automatic bucket filling: An imitation learning approach, Autom. Constr. 150 (2023) 104843, http://dx.doi.org/10.1016/j.autcon.2023.104843.
[11] S. Dadhich, F. Sandin, U. Bodin, U. Andersson, T. Martinsson, Adaptation of a wheel loader automatic bucket filling neural network using reinforcement learning, in: 2020 International Joint Conference on Neural Networks, IJCNN, IEEE, 2020, pp. 1–9, http://dx.doi.org/10.1109/IJCNN48605.2020.9206849.
[12] S. Backman, D. Lindmark, K. Bodin, M. Servin, J. Mörk, H. Löfgren, Continuous control of an underground loader using deep reinforcement learning, Machines 9 (10) (2021) 216, http://dx.doi.org/10.3390/machines9100216.
[13] N. Strokina, W. Yang, J. Pajarinen, N. Serbenyuk, J. Kämäräinen, R. Ghabcheloo, Visual rewards from observation for sequential tasks: Autonomous pile loading, Front. Robotics AI 9 (2022) http://dx.doi.org/10.3389/frobt.2022.838059.
[14] D. Eriksson, R. Ghabcheloo, M. Geimer, Automatic loading of unknown material with a wheel loader using reinforcement learning, in: 2024 IEEE International Conference on Robotics and Automation, ICRA, 2024, pp. 3646–3652, http://dx.doi.org/10.1109/ICRA57147.2024.10610221.
[15] P. Egli, M. Hutter, A general approach for the automation of hydraulic excavator arms using reinforcement learning, 2021, http://dx.doi.org/10.3929/ethz-b-000487440.
[16] P. Egli, D. Gaschen, S. Kerscher, D. Jud, M. Hutter, Soil-adaptive excavation using reinforcement learning, IEEE Robot. Autom. Lett. 7 (4) (2022) 9778–9785, http://dx.doi.org/10.1109/LRA.2022.3189834.
[17] R. Filla, Evaluating the efficiency of wheel loader bucket designs and bucket filling strategies with non-coupled DEM simulations and simple performance indicators, 2015, http://dx.doi.org/10.13140/RG.2.1.1507.1201, Unpublished.
[18] R. Filla, B. Frank, Towards finding the optimal bucket filling strategy through simulation, in: Proceedings of 15:Th Scandinavian International Conference on Fluid Power, 15th Scandinavian International Conference on Fluid Power, Fluid Power in the Digital Age, SICFP'17, June 7–9 2017 - Linköping, Sweden, in: Linköping Electronic Conference Proceedings, Linköping University Electronic Press, 2017, pp. 402–417, http://dx.doi.org/10.3384/ecp17144402.
[19] I. Kurinov, G. Orzechowski, P. Hamalainen, A. Mikkola, Automated excavator based on reinforcement learning and multibody system dynamics, IEEE Access 8 (2020) 213998–214006, http://dx.doi.org/10.1109/ACCESS.2020.3040246.
[20] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, P. Abbeel, Domain randomization for transferring deep neural networks from simulation to the real world, 2017, URL: http://arxiv.org/pdf/1703.06907v1.
[21] H. Ju, R. Juan, R. Gomez, K. Nakamura, G. Li, Transferring policy of deep reinforcement learning from simulation to reality for robotics, Nat. Mach. Intell. 4 (12) (2022) 1077–1087, http://dx.doi.org/10.1038/s42256-022-00573-6.
[22] D. Ha, J. Schmidhuber, World models, 2018, CoRR abs/1803.10122, URL: http://arxiv.org/abs/1803.10122.
[23] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780, http://dx.doi.org/10.1162/neco.1997.9.8.1735.
[24] C.M. Bishop, Mixture density networks, 1994, URL: https://research.aston.ac.uk/en/publications/mixture-density-networks.
[25] A. Graves, Generating sequences with recurrent neural networks, 2014, URL: https://arxiv.org/abs/1308.0850.
[26] D. Hafner, T. Lillicrap, J. Ba, M. Norouzi, Dream to control: Learning behaviors by latent imagination, 2019, URL: http://arxiv.org/pdf/1912.01603v3.
[27] D. Hafner, T. Lillicrap, M. Norouzi, J. Ba, Mastering atari with discrete world models, 2020, URL: http://arxiv.org/pdf/2010.02193v4.
[28] P. Wu, A. Escontrela, D. Hafner, K. Goldberg, P. Abbeel, DayDreamer: World models for physical robot learning, 2022, URL: http://arxiv.org/pdf/2206.14176v1.
[29] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, J. Davidson, Learning latent dynamics for planning from pixels, in: K. Chaudhuri, R. Salakhutdinov (Eds.), Proceedings of the 36th International Conference on Machine Learning, in: Proceedings of Machine Learning Research, vol. 97, PMLR, 2019, pp. 2555–2565, URL: https://proceedings.mlr.press/v97/hafner19a.html.
[30] R.E. Kalman, A new approach to linear filtering and prediction problems, J. Basic Eng. 82 (1) (1960) 35–45, http://dx.doi.org/10.1115/1.3662552.
[31] P. Becker, H. Pandya, G. Gebhardt, C. Zhao, J. Taylor, G. Neumann, Recurrent Kalman networks: Factorized inference in high-dimensional deep feature spaces, 2019, URL: http://arxiv.org/pdf/1905.07357v1.
[32] T. Haarnoja, A. Ajay, S. Levine, P. Abbeel, Backprop KF: Learning discriminative deterministic state estimators, in: D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, R. Garnett (Eds.), Advances in Neural Information Processing Systems, 29, Curran Associates, Inc, 2016, URL: https://proceedings.neurips.cc/paperfiles/paper/2016/file/697e382cfd25b07a3e62275d3ee132b3-Paper.pdf.
[33] K. Arndt, A. Ghadirzadeh, M. Hazara, V. Kyrki, Few-shot model-based adaptation in noisy conditions, IEEE Robot. Autom. Lett. 6 (2) (2021) 4193–4200, http://dx.doi.org/10.1109/LRA.2021.3068104.
[34] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, A Bradford Book, Cambridge, MA, USA, ISBN: 0262039249, 2018.
[35] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, 2017, URL: http://arxiv.org/pdf/1707.06347v2.
[36] A.Y. Ng, D. Harada, S. Russell, Policy invariance under reward transformations: Theory and application to reward shaping, in: Icml, vol. 99, 1999, pp. 278–287.

[37] D. Eriksson, R. Ghabcheloo, M. Geimer, Towards multiple material loading for wheel loaders using transfer learning, in: Proceedings of 18th Scandinavian International Conference on Fluid Power, SICFP23, 2023, URL: https://urn.fi/URN:NBN:fi:tuni-202404023220.

[38] M. Towers, J.K. Terry, A. Kwiatkowski, J.U. Balis, G. de Cola, T. Deleu, M. Goulão, A. Kallinteris, K. Arjun, M. Krimmel, R. Perez-Vicente, A. Pierré, S. Schulhoff, J.J. Tai, A.T.J. Shen, O.G. Younis, Gymnasium, Zenodo, 2023, http://dx.doi.org/10.5281/zenodo.8127026, URL: https://zenodo.org/record/8127025.

[39] J. Schulman, P. Moritz, S. Levine, M. Jordan, P. Abbeel, High-dimensional continuous control using generalized advantage estimation, 2015, URL: http://arxiv.org/pdf/1506.02438v6.

[40] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, PyTorch: An imperative style, high-performance deep learning library, in: H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems 32, Curran Associates, Inc, 2019, pp. 8024–8035, URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[41] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, URL: http://arxiv.org/pdf/1412.6980v9.

[42] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, N. Dormann, Stable-Baselines3: Reliable reinforcement learning implementations, J. Mach. Learn. Res. 22 (268) (2021) 1–8, URL: http://jmlr.org/papers/v22/20-1364.html.

[43] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, W. Woodall, Robot operating system 2: Design, architecture, and uses in the wild, Sci. Robotics 7 (66) (2022) eabm6074, http://dx.doi.org/10.1126/scirobotics.abm6074.