Modeling weakly-instrumented excavator arm dynamics with stacked-input LSTM

Nicolas Hoffmann[1,2], Max Cohen[1], Marius Preda[1], and Titus Zaharia[1]

[1] Télécom SudParis, France
[2] Heracles Robotics, France

Abstract. The application of machine learning for modeling complex dynamic systems, such as excavators, is gaining momentum as it offers flexibility beyond traditional mathematical models. Recent advances leverage neural networks to create data-driven models that can handle the non-linear and intricate dynamics of real machinery. However, these models often depend on expensive sensors and controlled environments. This study presents a cost-effective approach to modeling the dynamics of a weakly-instrumented 25-ton CAT 323 excavator arm using stacked-input Long Short-Term Memory (LSTM) networks. We evaluate the performance of Multi-Layer Perceptron (MLP) and LSTM architectures, both with and without input stacking, to accurately simulate excavator arm motion. Our results show that combining LSTM with stacked inputs significantly improves the model's predictive capabilities, challenging the notion that LSTM and input stacking are redundant. These findings highlight the potential of data-driven neural network models to provide accurate and efficient solutions for dynamics modeling in complex, real-world settings, paving the way for advanced AI-based strategies in the earthworks and construction industries.

Keywords: Robotics · System identification · Weak instrumentation · Construction 4.0 · Excavator dynamics modelling · Deep learning · MLP and LSTM neural networks

1 Introduction

1.1 Excavator Modelling

The application of machine learning and AI in modeling complex dynamic systems, such as hydraulic excavators, has gained significant attention in recent years. Traditional methods for dynamic modeling often rely on physical models or simplified assumptions, which are limited to specific use-cases and controlled environments. These approaches require significant expertise and can involve expensive modifications, making them less adaptable for diverse real-world applications.

The dynamics of an excavator arm are inherently non-linear and complex due to coupled hydraulic joints and shared pumps. Accurately modeling these

dynamics is crucial for various applications, including control, simulation, and optimization. Data-driven approaches, particularly using deep learning models, offer a promising alternative by learning directly from input-output data rather than requiring detailed physical modeling.

Most current AI-based approaches, such as Non-linear Model-Based Control (NMBC), depend on accurate and fast dynamic models to operate effectively, especially for simulating diverse trajectories or predicting long-term behavior in real-time [15]. Fast simulation speeds are essential not only for training controllers offline but also for real-time applications where the simulator must be run online [7].

In this study, we use a weakly instrumented 25-ton CAT 323 excavator, a commonly used model in the earthworks industry. Our focus is on data-driven dynamic modeling using neural networks to capture the complexities of excavator arm movements in real operational environments. By leveraging low-cost sensors and real-world data, we aim to develop robust models that generalize well across different scenarios without relying on costly instrumentation or controlled settings.

1.2 Problem Formulation

We define the state of the excavator arm at time t as a vector of articular positions and velocities:

$$x_t := (\alpha, \beta, \gamma, \dot{\alpha}, \dot{\beta}, \dot{\gamma}) \tag{1}$$

where α, β, γ are the angular positions of the boom, stick, bucket measured relative to the horizontal plane, assuming the excavator is on level ground, and $\dot{\alpha}$, $\dot{\beta}$, $\dot{\gamma}$ are the angular velocities of the boom, stick, and bucket. The action at time t, u_t, represents joystick commands normalized within $[-1, 1]$:

$$u_t = [u_t^{boom}, u_t^{stick}, u_t^{bucket}], \quad u_t \in [-1, 1]^3 \tag{2}$$

We want to predict the next state x_{t+1} from the current state x_t and action u_t:

$$x_{t+1} = f(x_t, u_t) \tag{3}$$

This involves simulating a trajectory by auto-regressively predicting each state, which is challenging due to the accumulation of prediction errors over time. We adopt a data-driven approach using neural networks trained on data collected from the physical machine.

1.3 Related work

Modeling the dynamics of hydraulic excavators using mathematical models or multi-body models, called white box models, has been successful and extensively applied to excavator control [15]. However, such methods require specific skills

and considerable expert knowledge to explicitly construct a physical model of the system, with all the related parameters involved. In addition, they need to be regularly maintained for heavy-duty machines like excavators, whose dynamics change with the age and degree of utilization. Finally, a white box model is dedicated to a singular system and needs to be crafted again for each new machine.

To overcome such constraints, the data-driven black box models learn solely from input-output data [8]. As representative of this family of approaches, let us first mention the Multi-Layer Perceptron (MLP) techniques, which have been successfully employed to estimate machine consumption [9] and to evaluate machine productivity using Original Equipment Manufacturer (OEM) telematics data [10].

The Non-linear Auto-Regressive with eXogenous inputs (NARX) formulation enables the model to compute temporal features by predicting outputs based on a stack of the n past actions and m past states of the system [16]:

$$\hat{x_t} = f(u_{t-1}, \ldots, u_{t-n}, x_{t-1}, \ldots, x_{t-m}) \tag{4}$$

A NARX model can be trained by minimizing its prediction error over a transition or a batch of transitions. This approach, known as Series-Parallel (Fig. 1), is theoretically robust and allows for faster training. In contrast, Parallel training minimizes the prediction error over an entire free-run simulation. By feeding the predicted next state back into the input of the network, parallel training ensures that the model is trained on the actual task on which it is intended to perform. Our experimental results bellow show that both training methods are valuable.

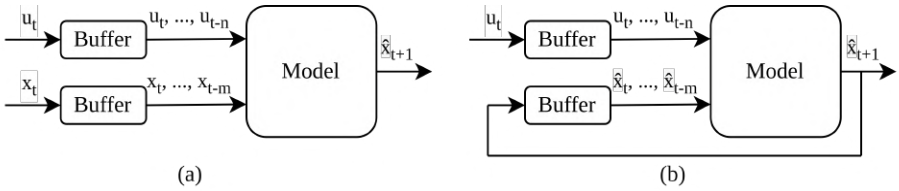

Fig. 1: Series-Parallel (a) and Parallel (b) training of a NARX.

NARX with MLP backbones have been effectively used to model the dynamics of excavator arms and to train position and speed controllers [4, 5, 12]. However, such approaches depend on measurements such as cylinder pressure or motor RPM, necessitating the installation and maintenance of expensive sensors, particularly when deployed across a fleet of machines. Additionally, the finite length of the input stack imposes a constraint on the network, limiting its

ability to capture information beyond the specified time window.

Recurrent Neural Network (RNN), such as Long Short-Term Memory (LSTM) networks [6], offer an alternative to NARX for capturing temporal features. Unlike NARX, which stacks the network inputs over time, LSTM maintain a hidden state throughout time, allowing them to handle complex temporal dependencies more effectively and without a fixed time horizon. For example, LSTM have been used to model the activity of a Komatsu PC220LC excavator with 99.78% accuracy using only a few cellphone sensors and a Convolutional LSTM (Conv-LSTM) network [14].

The review of the state of the art highlights the successful application of neural networks for modeling various aspects of an excavator. To incorporate temporal features into these models, researchers either stack inputs over time using MLP or switch entirely to LSTM architectures. The prevailing assumption is that LSTM, with their inherent ability to manage temporal dependencies, render the practice of stacking inputs redundant.

1.4 Contribution

This study evaluates and compares the performances of MLP and LSTM architectures in modeling the dynamics of a weakly-instrumented excavator arm, focusing on the influence of input stacking. Our results demonstrate that combining LSTM with stacked inputs significantly enhances the modeling capabilities of the network. Our approach challenges the common assumption that LSTM networks and input stacking are redundant in their ability to capture temporal features, resulting in a both accurate and cost-effective model.

2 Dataset creation

2.1 State measure

We collected data from a weakly instrumented CAT323 excavator by measuring the arm's articular positions (α, β, γ) using Inertial Measurement Unit (IMU) sensors installed by the OEM. The positions were low-pass filtered to remove vibrations, and velocities were estimated via numerical differentiation followed by another low-pass filter.

To ensure real-time applicability, we avoided non-causal filters, resulting in a delay of about 30 ms for position measurements and 50 ms for velocities. The excavator is termed "weakly" instrumented since our state formulation is limited to articular coordinates, unlike other methods requiring additional, expensive sensors for cylinder pressure, forces, engine RPM, or temperature [5, 12].

2.2 Data collection

We recorded the excavator moving only in the air by excavating in front of the machine and positioning it on leveled ground, to make sure that we model the dynamics of the arm of our excavator, without ground interaction effects. We also aimed to sample the entire articular space densely. The boom, stick, and bucket were oscillated between their angular limits, with oscillation periods independently sampled between 4 and 60 seconds. A manually tuned Proportional Integral Derivative (PID) [1] controller was used to follow these trajectories, computing actions u_t based on position error e_t, its derivative, and integral:

$$u_t = K_p e_t + K_i \sum e_t + K_d \dot{e}_t \tag{5}$$

To improve the dataset, we also recorded data with immobile articulations at various positions, ensuring the model learns that articulations without commands remain stationary. We achieved this by imposing zero commands for 10-second intervals every 10 seconds on randomly selected articulations.

We collected 20 trajectories in total, each about 5 minutes long and recorded at 100 Hz, yielding around one million transitions. We segmented the trajectories into non-overlapping 500-step windows (5 seconds each) and split them split into training (80%) and validation (20%) sets. States and actions were normalized to $[-1, 1]$ for neural network processing.

3 Network Architectures

We have retained the state of the art, both MLP and LSTM architectures, each in stacked and not stacked variants, as illustrated in Fig. 2:

1. A MLP network that inputs a concatenation of the current state and action and outputs the next state.
2. A Stack MLP network that adds previous states and actions to its input to provide temporal features.
3. A LSTM network that only inputs the current state and action but propagates a hidden state through time.
4. A Stack LSTM network that stacks the inputs and propagates a hidden state.

We parameterize all network architectures by the number of hidden layers and the size of each hidden layer, as detailed in Tab. 1, and use the networks to compute the following state. For instance, in the series-parallel setting:

$$\hat{x}_t = NN(u_{t-1}, \ldots, u_{t-n}, x_{t-1}, \ldots, x_{t-m}) \tag{6}$$

And in the parallel setting:

$$\hat{x}_t = NN(u_{t-1}, \ldots, u_{t-n}, \hat{x}_{t-1}, \ldots, \hat{x}_{t-m}) \tag{7}$$

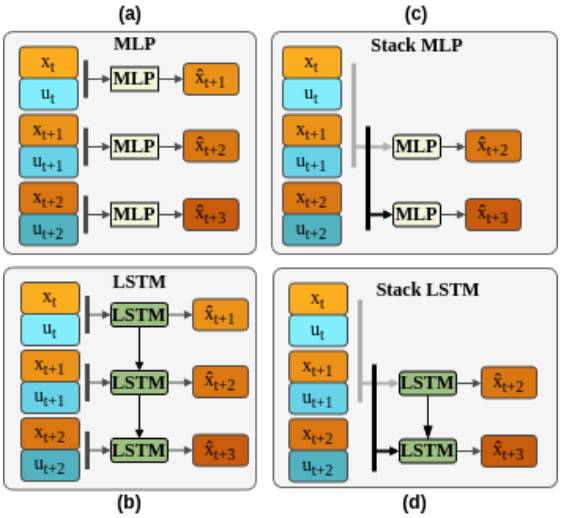

Fig. 2: The four network architectures adopted in our study. (a) MLP predicts the next state from the previous state and action. (b) LSTM passes a hidden state through time. (c) 2-stack MLP concatenates the 2 last states and actions. (d) 2-Stack LSTM also passes the hidden state.

4 Training

We train our networks to minimize the prediction Mean Squared Error (MSE) over a batch of training windows with the Adam optimizer [11]. To allow the hidden state sufficient time to initialize, the first 20 predictions of each window are excluded from the loss calculation. All training sessions are conducted on an NVIDIA GeForce GTX 1080 Ti GPU, with each session lasting up to 6 hours. We use early stopping to prevent over-fitting.

We have experimented with varying proportions of parallel and series-parallel training by introducing a new hyper-parameter that represents the fraction of the training window allocated to series-parallel training. Training always begins in series-parallel and switches to parallel at the specified point within the window. However, validation and test performances are measured almost exclusively in free-runs, with 96% parallel: we provide the ground truth to each model for the first 20 steps (4%) of each window to allow for proper initialization of the hidden state.

We also experimented with predicting only the variation between the current state and the next state to stationarize the distribution [13]. In the series-parallel setting, this approach is implemented as follows:

$$\hat{x}_t = \hat{x}_{t-1} + NN(u_{t-1}, \ldots, u_{t-n}, \hat{x}_{t-1}, \ldots, \hat{x}_{t-m}) \tag{8}$$

We parameterize the input stack of a network by the size of the stack, *i.e.* how many inputs are concatenated, and the interval between time-steps. For example,

a stack of size 3 and interval 5 would contain $[s_{t-1}, a_{t-1}, s_{t-6}, a_{t-6}, s_{t-11}, a_{t-11}]$. This sparse stacking method enables the network to capture dependencies from the distant past while maintaining a manageable input size.

Table 1: Hyper-parameters and their optimization ranges.

	Hyper-parameter	**Range**
Network	# Hidden layers	1 to 8
	Hidden layers size	16 to 2048
	Stack size	1 to 15
	Stack interval	1 to 10
Training	Learning rate	10^{-5} to 10^{-1}
	Dropout rate	0 to 0.75
	Batch size	8 to 512
	Predict variation	True or false
	Series-Parallel %age	0% to 100%

5 Experimental results

We have optimized all hyper-parameters listed in Tab. 1 using the Tree-structured Parzen Estimator (TPE) optimization method [2]. The objective is to minimize the validation loss achieved during training while maximizing the inference speed of the network. We applied this optimization process to each of the four network architectures shown in Fig. 2 by changing the backbone and adapting the stack size and stack interval ranges. We trained hundreds of networks per architecture: 1136 MLP, 740 Stack MLP, 490 LSTM and 408 Stack LSTM.

This multi-architecture, multi-objective optimization yields one Pareto front of training runs per network architecture, highlighting the networks that outperform others in terms of lower validation loss, faster inference speed, or both, as illustrated in Fig. 3.

5.1 Prediction objective quality

The obtained results reveal a clear hierarchy between network architectures: Stack LSTM networks perform the best, followed by LSTM networks, with Stack MLP and MLP networks showing progressively lower performances. The differences in terms of Pareto fronts are highly significant, especially given the autoregressive nature of the regression task considered, which is susceptible to lead to an accumulation of prediction errors over time.

Table 2 compares the best prediction quality among all trainings shown in Fig. 3 as measured by the validation MSE. The reported MSE values confirm here again the superiority of LSTM-based approaches.

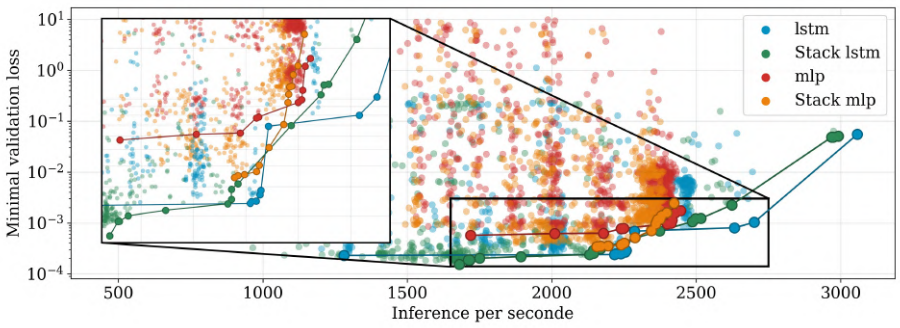

Fig. 3: Hyper-parameter optimization of the four network architectures. Each dot represents one training, the Pareto front is highlighted for all architectures.

Table 2: Lowest validation loss obtained by network architecture.

Architecture	Lowest validation loss
MLP	5.69e-4
Stack MLP	3.55e-4
LSTM	2.29e-4
Stack LSTM	**1.76e-4**

5.2 Prediction speed

The inference speed values presented in Fig. 3 (between 1500 and 2500 inferences per second for most networks), do not reveal a significant difference between network architectures. While the best Stack LSTM networks are slower than most MLP or Stack MLP networks, this is due to the number of layers rather than the stacking or LSTM components themselves. Evidence of this is that at a given inference speed, such as where MLP performs best, there are also LSTM or Stack networks with comparable or superior prediction performance. Stack LSTM networks scale better with larger networks but are not intrinsically slower.
Due to the lack of significant difference in inference speed we choose the best predicting network, *i.e.*, the one with the lowest validation loss, for further analysis.

5.3 Best hyper-parameters

Table 3 details the hyper-parameters of the best predicting network for each architecture. We can drive the following conclusions:

– MLP-based networks do not seem to benefit from additional or larger hidden layers, unlike LSTM-based networks. Our results closely corroborate the work of [4], which used a Stack MLP with 3 hidden layers and 128 nodes per layer.

– All networks converged toward predicting the state variation instead of the state itself, a behavior that we observed early in the training process.
– The input stacks for the Stack MLP are about half a second, and for the Stack LSTM, about a quarter second, indicating that older information does not help predict future motion. The Stack MLP requires a longer input window than the Stack LSTM, aligning with the intuition that LSTM networks can compute temporal features internally.

Table 3: Best hyper-parameters found for each network architecture.

	Hyper-parameter	**MLP**	**Stack MLP**	**LSTM**	**Stack LSTM**
Network	# Hidden layers	3	3	6	4
	Hidden layers size	115	116	189	545
	Stack size	1	12	1	5
	Stack interval	1	6	1	5
Training	Learning rate	4.2e-3	8.87e-3	1.3e-3	1.65e-3
	Dropout rate	2.9e-3	1.35e-2	2.49e-1	3.8e-2
	Batch size	25	16	17	21
	Predict variation	True	True	True	True
	Series-Parallel %age	68%	50%	51%	50%

5.4 Free-run experiment

We conducted a free-run experiment using a human-operated, real-life trajectory, significantly different from the oscillating trajectories of our training dataset. Starting from a stationary position, which allowed us to initialize the hidden states with 20 identical static states, the operator moved the excavator arms for one minute. After initialization, we fed the actions of the operator actions into our networks and compared the predicted positions to the ground truth measurements from the actual machine. The results are presented in Fig. 4.

In this free-run, MLP-based networks quickly rotated the arm upward, likely due to a slight positive bias accumulating over time. In contrast, LSTM-based networks closely followed the ground truth, demonstrating their ability to accurately simulate novel trajectories and generalize beyond the training data. Although the impact of input stacking with the LSTM network is not apparent in this test, Tab. 2 shows that adding stacked inputs to an LSTM network reduces its validation loss by 23%. This provides a trade-off between predictive quality and implementation simplicity. Figure 5 illustrates the ability of the best stacked LSTM to closely follow the ground truth for 60 seconds.

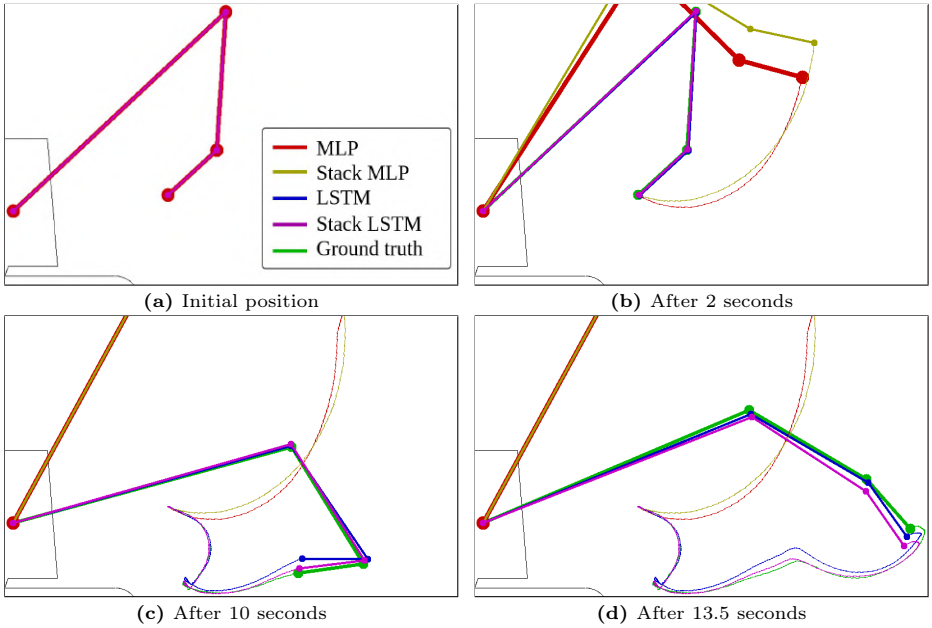

Fig. 4: Free-run simulation of the best network from each architecture over 1,350 predictions (13.5 seconds). All simulations begin from the same initial position (a). The MLP (red) and Stack MLP (yellow) networks quickly accumulate errors and overlap near the upper limit (b), constrained by the machine's mechanical stops (c). In contrast, the LSTM (purple) and Stack LSTM (blue) networks closely follow the operator's ground truth trajectory (green), with minimal drift (c, d).

6 Discussion

The goal of this work was to evaluate the effectiveness of MLP and LSTM networks in modeling the dynamics of a weakly-instrumented excavator arm, both with and without input stacking. Our results indicate that LSTM models can effectively simulate machine dynamics and consistently outperform MLP networks, with input stacking further enhancing LSTM performance. This challenges the assumption that LSTM and input stacking are redundant for capturing temporal features.

We suggest two reasons for this outcome: (1) While LSTM networks handle long-term dependencies, their performance decreases over longer timescales. Input stacking, however, provides temporal features from any chosen time frame, making it adaptable to different sampling rates, unlike LSTM. (2) Input stacking may offer additional temporal features for LSTM to process, improving performance.

We trained networks to predict state variations rather than states, improving free-run performance for oscillating trajectories. However, this may not work well for abrupt trajectories involving sharp vibrations or stops at mechanical limits.

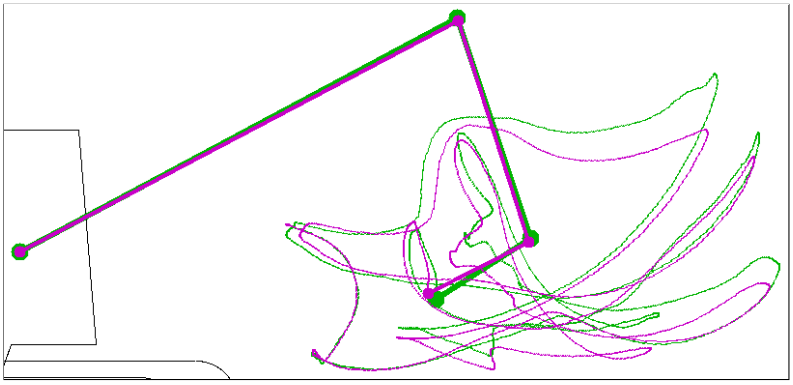

Fig. 5: 60 seconds free-run of the best stack LSTM network.

The poor free-run performance of our MLP networks, unlike results reported in [4], could be due to: (1) our lack of specific data like oil temperature and engine RPM; (2) the complexity or difference of our free-run trajectory; (3) different machine types—our 25-ton versus their 12-ton excavator; or (4) unoptimized hyper-parameters.

7 Future work

Future work includes using the simulator for control purposes via model predictive control, reinforcement learning-based training, or inverse model computation. The proposed approach is suitable for online and offline simulation of various earthwork machines, facilitating cost-effective control strategies without additional investments, because most machines already have basic articulation sensors.

Promising directions also involve enhancing model adaptation to abrupt trajectories, comparing them with filtering-based techniques, and developing data-driven soil interaction models. Current models rely on hand-crafted simulations for ground interactions [3]. Modeling ground interactions is challenging due to the variability in ground properties and shapes, which could be better addressed with stochastic models. Advancements in these areas could reduce simulator costs and enhance the realism and effectiveness of autonomous systems.

References

1. Ang, K.H., Chong, G., Li, Y.: Pid control system analysis, design, and technology. IEEE Transactions on Control Systems Technology **13**(4), 559–576 (2005). https://doi.org/10.1109/TCST.2005.847331
2. Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., Weinberger, K. (eds.) Advances in Neural Information Processing Systems. vol. 24. Curran Associates, Inc. (2011)

3. Egli, P., Gaschen, D., Kerscher, S., Jud, D., Hutter, M.: Soil-adaptive excavation using reinforcement learning. IEEE Robotics and Automation Letters **7**, 1–8 (10 2022). https://doi.org/10.1109/LRA.2022.3189834
4. Egli, P., Hutter, M.: Towards rl-based hydraulic excavator automation. In: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 2692–2697 (2020). https://doi.org/10.1109/IROS45743.2020.9341598
5. Egli, P., Hutter, M.: A general approach for the automation of hydraulic excavator arms using reinforcement learning. IEEE Robotics and Automation Letters **7**(2), 5679–5686 (2022). https://doi.org/10.1109/LRA.2022.3152865
6. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**, 1735–80 (12 1997). https://doi.org/10.1162/neco.1997.9.8.1735
7. Hoffmann, N., Cohen, M., Roullier, L., Preda, M., Zaharia, T.: Data-driven control of a weakly-instrumented excavator with deep learning. IEEE International Conference on Intelligent Reality (ICIR) (2024)
8. Hou, Z.S., Wang, Z.: From model-based control to data-driven control: Survey, classification and perspective. Information Sciences **235**, 3–35 (2013). https://doi.org/https://doi.org/10.1016/j.ins.2012.07.014, https://www.sciencedirect.com/science/article/pii/S0020025512004781, data-based Control, Decision, Scheduling and Fault Diagnostics
9. Jassim, H.S.H., Lu, W., Olofsson, T.: Predicting energy consumption and co2 emissions of excavators in earthwork operations: An artificial neural network model. Sustainability **9**(7) (2017). https://doi.org/10.3390/su9071257, https://www.mdpi.com/2071-1050/9/7/1257
10. Kassem, M., Mahamedi, E., Rogage, K., Duffy, K., Huntingdon, J.: Measuring and benchmarking the productivity of excavators in infrastructure projects: A deep neural network approach. Automation in Construction **124**, 103532 (2021). https://doi.org/https://doi.org/10.1016/j.autcon.2020.103532, https://www.sciencedirect.com/science/article/pii/S0926580520311122
11. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. CoRR **abs/1412.6980** (2014), https://api.semanticscholar.org/CorpusID:6628106
12. Lee, M., Choi, H., Kim, C., Moon, J., Kim, D., Lee, D.: Precision motion control of robotized industrial hydraulic excavators via data-driven model inversion. IEEE Robotics and Automation Letters **7**(2), 1912–1919 (2022). https://doi.org/10.1109/LRA.2022.3142389
13. Livieris, I.E., Stavroyiannis, S., Iliadis, L., Pintelas, P.: Smoothing and stationarity enforcement framework for deep learning time-series forecasting. Neural Computing and Applications **33**(20), 14021–14035 (2021)
14. Mahamedi, E., Rogage, K., Doukari, O., Kassem, M.: Automating excavator productivity measurement using deep learning. Proceedings of the Institution of Civil Engineers - Smart Infrastructure and Construction **174**, 1–13 (04 2022). https://doi.org/10.1680/jsmic.21.00031
15. Mattila, J., Koivumäki, J., Caldwell, D.G., Semini, C.: A survey on control of hydraulic robotic manipulators with projection to future trends. IEEE/ASME Transactions on Mechatronics **22**(2), 669–680 (2017). https://doi.org/10.1109/TMECH.2017.2668604
16. Narendra, K., Parthasarathy, K.: Identification and control of dynamical systems using neural networks. IEEE Transactions on Neural Networks **1**(1), 4–27 (1990). https://doi.org/10.1109/72.80202