Citation: J. Fañanás-Anaya, G. López-Nicolás and C. Sagüés, "Robot Modeling With Autoregressive Transformers," 2024 7th Iberian Robotics Conference (ROBOT), Madrid, Spain, 2024, pp. 1-6

Robot Modeling with Autoregressive Transformers

Javier Fañanás-Anaya, Gonzalo López-Nicolás, Carlos Sagüés
Instituto de Investigación en Ingeniería de Aragón (I3A),
Universidad de Zaragoza, Spain
javierfa@unizar.es, gonlopez@unizar.es, csagues@unizar.es

***Abstract*—Many applications require accurate models of the dynamic behaviour of robots, such as realistic simulation, state prediction, and control. Traditionally, the modeling of these systems is based on analytical solutions that are complicated to develop. This is why in the state of the art some solutions are based on neural networks. We present a novel transformer-based architecture, called AR-Transformer, designed to estimate the model of complex nonlinear robotic systems. Our goal is to predict the behavior of the robot knowing only its initial state at any time step for any prediction horizon, knowing the external inputs or the control inputs. The AR-Transformer has an autoregressive approach, which allows the architecture to iterate and estimate states up to the time step for which the control inputs are known. This autoregressive approach works correctly with large prediction horizons thanks to training the model without teacher forcing. We have positively validated the architecture with a public dataset of real experiments of a 3 DOF robotic arm.**

I. Introduction

Modeling the dynamic behavior of robots presents a significant challenge due to the complex and unpredictable nature of their environments. Precise and accurate models are essential for various applications, such as realistic simulation, state prediction and control [1]–[3]. Traditional methods are based on physical models, which can be difficult to obtain in dynamic or complex environments. Deep learning techniques have emerged as a powerful alternative to address these challenges by enabling robots to be modeled directly from experimental data, adapting to complex systems autonomously.

Physical models are widely used, however they are often based on simplifications that do not capture all the dynamics of the system. These simplifications may produce very small errors in predictions of a few time steps, however, they can greatly affect model performance in predictions of many time steps [4]. Therefore, there has been growing interest in using neural networks (NN) to model robotic systems because of their ability to learn complex dynamics from data obtained directly from experimental observations.

The problem addressed in this paper is to obtain an NN-based model of a dynamic system to predict its behavior up to any prediction horizon from known control inputs. For example, to model the dynamics of the motion of a robot from the power supplied to its motors. In modeling robotic systems with NN, two main trends are observed in the state of the art: NN architectures and hybrid physics models with NN. The work of [5] integrates an architecture based on Recurrent Neural Networks (RNN) and FeedForward NNs as a predictor in a control scheme of a robotic arm whose function is to cut fruits and vegetables. Another example proposes a RNN applied to real-time motion control for mobile robots [6]. In the case of hybrid models, one approach uses this method to predict the state of a quadrotor [4]. Its proposed architecture consists of a motion model and RNN. Another example uses an hybrid model to estimate the aerodynamic forces and torques of a quadrotor [7]. Specifically, the architecture consists of a motor model and a rotor model, which combined with the estimations of a temporal-convolutional NN, predict the aerodynamic forces and torques.

One solution in the state of the art of NN is to work with transformers. The original transformer architecture, as well as the attention mechanisms, were designed mainly for natural language processing problems [8]. In addition, the use of transformers has been successfully extended to problems such as computer vision [9] or time series forecasting [10]–[12]. In robotics, transformers have been applied to a variety of problems, including vision-guided locomotion of a quadruped robot [13], path planning [14] and imitation learning [15]. The work of [16] presents a transformer-based model capable of interpreting instructions from text and performing the corresponding actions. Although transformer-based methods have been applied in various fields of robotics, the state of the art in terms of modeling, simulating or predicting robot dynamics is still based on RNN or hybrid physics models with NN.

The context of our paper is the modeling of dynamic robotic systems from external control inputs. The proposed transformer-based architecture models the system by receiving control inputs in real-time or using as input a time sequence of control inputs of any length. Our main contributions are:

- A novel architecture, called AR-Transformer, based on autoregressive transformers for modeling dynamic robotic systems with any prediction horizon.
- The proposed architecture not only correctly models the dynamics of robotic systems with the conditions and control policies observed during training, but also demonstrates high generalization and transferability.
- We evaluate the performance of the proposed architecture with a public robotic dataset of real experiments.

The code of the proposed architecture, pre-trained models and a tutorial for adding new datasets are publicly available at `https://github.com/javierfa98/AR_Transformer`

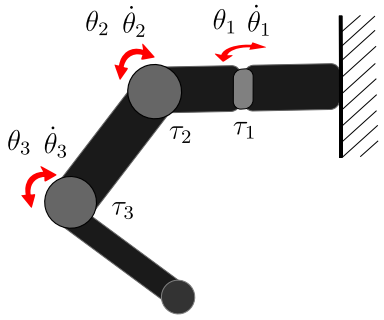

Fig. 1. Schematic of a 3 DOF robotic arm system. At a time step t, the control inputs $u_t = [\tau_1, \tau_2, \tau_3]$ represent the desired torques (Nm) applied to the motors. The system state $y_t = [\theta_1, \theta_2, \theta_3, \dot{\theta}_1, \dot{\theta}_2, \dot{\theta}_3]$ includes the joint angles (rad) and angular velocities (rad/s).

II. Problem Formulation

This section defines the problem of modeling dynamical systems. Consider the state of a system at time step t as $y_t \in \mathbb{R}^m$ and the control inputs at t as $u_t \in \mathbb{R}^n$. Consider a function f that estimates the state of the system at the next time instant $\hat{y}_{t+1}$ from u_t and $\hat{y}_t$:

$$\hat{y}_{t+1} = f(u_t, \hat{y}_t). \tag{1}$$

Modeling dynamical systems aims to calculate over a time period T its state evolution $Y_{1,T} \in \mathbb{R}^{m \times T}$

$$Y_{1,T} = \begin{bmatrix} y_1 & y_2 & \cdots & y_T \end{bmatrix}. \tag{2}$$

Knowing the initial state y_0, and the control inputs over the entire time period $U_{0,T-1} \in \mathbb{R}^{n \times T}$

$$U_{0,T-1} = \begin{bmatrix} u_0 & u_1 & \cdots & u_{T-1} \end{bmatrix}, \tag{3}$$

function f can be iterated T times until getting an estimate $\hat{Y}_{1,T} \in \mathbb{R}^{m \times T}$

$$\hat{Y}_{1,T} = \begin{bmatrix} \hat{y}_1 & \hat{y}_2 & \cdots & \hat{y}_T \end{bmatrix}. \tag{4}$$

Note that y_0 can be any initial state, not necessarily at rest, that needs to be known. The value of T is not fixed, and will depend on how many time instants will be simulated or how many future control inputs are known. The main problem to be solved in this paper is to obtain a function f using transformers that minimizes the error between $Y_{1,T}$ and $\hat{Y}_{1,T}$.

Fig. 1 shows a schematic of the 3 DOF robotic arm system used during the experiments of this paper.

III. Transformers in our framework

This section defines concepts of transformer architectures and motivates how to apply these architectures to the presented problem. Transformers are applied to problems where both the input and output are sequences, and due to the nature of these problems, the encoder-decoder architecture is typically used in this context. For our problem we have considered keeping this approach, since the attention mechanisms benefit from this.

To exploit the full potential of this approach, we propose to use as input to the transformer both the current time step t and the last h estimated states and control inputs, stored in a matrix $X_{t,t-h} \in \mathbb{R}^{(n+m)\times(h+1)}$:

$$X_{t,t-h} = \begin{bmatrix} u_t & u_{t-1} & \cdots & u_{t-h} \\ \hat{y}_t & \hat{y}_{t-1} & \cdots & \hat{y}_{t-h} \end{bmatrix} \tag{5}$$

where each column represents a time instant. With this input we can modify (1) as follows:

$$\hat{y}_{t+1} = f(X_{t,t-h}). \tag{6}$$

The motivation for including the last h states is to apply attention on them. Attention mechanisms are a fundamental part of transformers. The main goal of attention in NN is to enable it to focus selectively on certain parts of a sequence, assigning different levels of importance to each element.

Attention calculation has three components: queries Q, keys K, and values V. Obtaining these matrices involves linearly transforming the input matrix X:

$$Q = W_q X, \quad K = W_k X, \quad V = W_v X \tag{7}$$

where W_q, W_k, and W_v are weight matrices. Once Q, K, and V are obtained, attention is calculated by taking a weighted sum of the values, with the weights being derived from the similarity between the queries and the keys:

$$\text{Attention}(Q, K, V) = V \ \text{softmax}\left(\frac{Q^T K}{\sqrt{d_k}}\right) \tag{8}$$

where d_k is the dimension of the key vector. Note that we calculate Q, K, and V from the same input sequence X. This attention approach is referred as self-attention, allowing the architecture to weigh the importance of different elements within the same sequence.

Cross-attention is another approach, where Q is derived from the input sequence, but K and V are calculated from another sequence. Generally, in transformer architectures, self-attention is used in the encoder to obtain context from the input vectors. This context, along with the decoder input vector, is then used with cross-attention to generate the output vector in the decoder. Commonly transformer architectures use Multi-Head Attention, which can be defined as an extension of Attention that uses m_h sets of queries, keys and values:

$$\begin{aligned} \text{MultiHead}(X) &= W_o \text{Concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_{m_h}) \\ \text{where head}_i &= \text{Attention}(W_{qi}X, W_{ki}X, W_{vi}X) \end{aligned} \tag{9}$$

where W_{qi}, W_{ki} and W_{vi} are weight matrices of the i-th head. The output of the Multi-Head attention is calculated concatenating the results of each head and linearly projecting them with W_o. Multi-Head Attention aims to focus on different sequence parts with respect to distinct aspects, enhancing its ability to capture diverse data relationships.

The final architecture will depend on the nature of the problem to be solved. The following section details all the layers and operations performed by our proposed architecture.

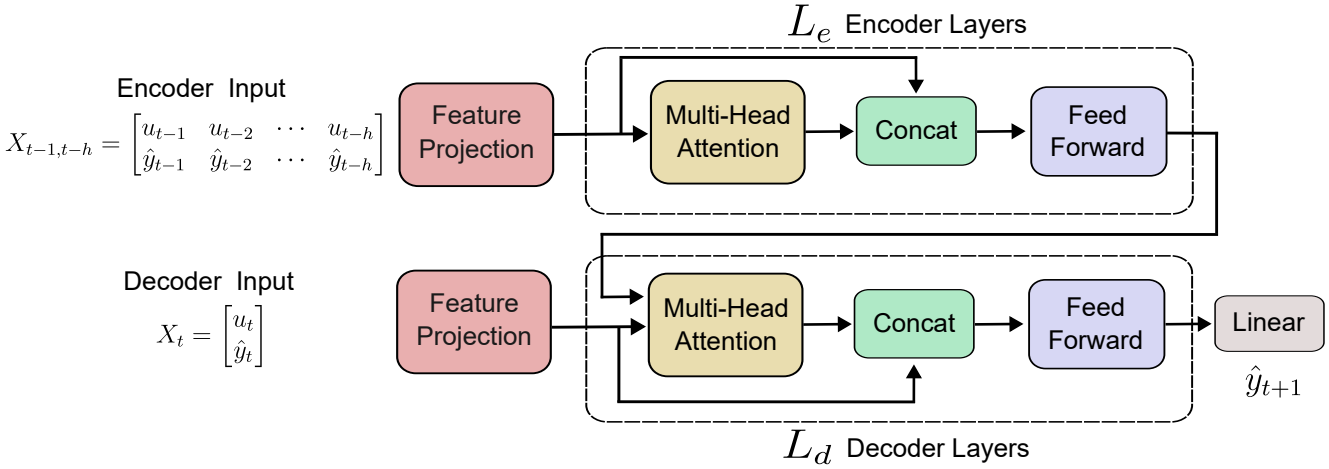

Fig. 2. Diagram of the AR-Transformer architecture. The encoder calculates the context from the last h control inputs and estimated system states $X_{t-1,t-h}$. The decoder estimates the next state $\hat{y}_{t+1}$ of the system from the output of the encoder, and the current control inputs u_t and state $\hat{y}_t$. The model can be used in autoregressive way to predict any prediction horizon knowing the control inputs.

IV. Proposed architecture: AR-Transformer

This section will present the proposed architecture, AR-Transformer. Fig. 2 shows a diagram of the architecture. As motivated in the previous section, the encoder-decoder approach is used. In our case, the encoder input matrix is $X_{t-1,t-h} \in \mathbb{R}^{(n+m)\times h}$:

$$X_{t-1,t-h} = \begin{bmatrix} u_{t-1} & u_{t-2} & \cdots & u_{t-h} \\ \hat{y}_{t-1} & \hat{y}_{t-2} & \cdots & \hat{y}_{t-h} \end{bmatrix} \tag{10}$$

where the last h control inputs and estimated states of the system are stored. The decoder input matrix is $X_t \in \mathbb{R}^{(n+m)}$:

$$X_t = \begin{bmatrix} u_t \\ \hat{y}_t \end{bmatrix} \tag{11}$$

where the current control input u_t and the last estimated state $\hat{y}_t$ are used as input.

The encoder architecture will now be detailed. The dimension of the model d_m is a hyperparameter with a large impact on the size and performance of the architecture. We define d_m as the number of features at a time step. Considering the encoder input matrix, we find that $d_m = m + n$. Remember that $u_t \in \mathbb{R}^n$ and $y_t \in \mathbb{R}^m$. However, $n+m$ may not be very large, limiting the abstraction and learning capability of the architecture, since d_m has a direct impact on the number of learning parameters. As an alternative with $d_m > n + m$ we propose to include a layer before the encoder input, called Feature Projection that projects the input matrix $X_{t-1,t-h}$ from $\mathbb{R}^{(n+m)\times h}$ to $\mathbb{R}^{d_m \times h}$. This projection increases the number of parameters at each time step, but decreases the interpretability of the input matrix. The Feature Projection layer consists on a FeedForward NN layer with d_m neurons and ReLU as activation function.

The proposed architecture continues with L_e encoder layers, each consisting of a Multi-Head Attention layer, a concatenation layer, and a FeedForward layer. In the first layer, the input is the output of the Feature Projection layer. In the subsequent layers, the input is the output of the previous FeedForward layer.

In the encoder, the Multi-Head Attention layer performs self-attention on the input matrix. The output of the Attention layer is concatenated with its input matrix, obtaining a matrix of size $\mathbb{R}^{(2\cdot d_m)\times h}$. In transformer models, a residual connection is typically used instead of concatenation to add the output of a Multi-Head Attention to its input. This avoids doubling the size of d_m, which is usually large. However, in our scenario, we have found that concatenating the output with the input, and retaining the original information, leads to better results. This approach is particularly beneficial because the next state of a robotic system highly depends on previous states, and preserving this information improves the estimations.

In the encoder, the dimensions of $Q \in \mathbb{R}^{d_q \times h}$, $K \in \mathbb{R}^{d_k \times h}$, and $V \in \mathbb{R}^{d_v \times h}$ have been defined as follows for consistency:

$$d_q = d_k = d_v = d_m / m_h \tag{12}$$

where m_h is the number of heads. The encoder ends with a FeedForward layer. This layer has d_m neurons with a ReLU activation function. The decoder has a similar composition. The input matrix X_t is projected with a Feature Projection layer if $d_m > n + m$. The matrix is then used as input for the decoder.

The decoder is formed with L_d decoder layers, where each layer is again formed by a Multi-Head Attention layer, a concatenation layer and a FeedForward layer. The main difference is that in the Multi-Head Attention layer there is cross-attention, where $Q \in \mathbb{R}^{d_q \times 1}$, $K \in \mathbb{R}^{d_k \times h}$, and $V \in \mathbb{R}^{d_v \times h}$, since the decoder attends to the sequence provided by the encoder. Therefore, with the output of the decoder it is predicted the next state of the system based on the context of the previous h states.

As described, with one iteration of the AR-Transformer only the state of the system at the next time instant is estimated.

TABLE I
HYPERPARAMETERS OF THE AR-TRANSFORMER ARCHITECTURE ALONG WITH THE BEST VALUES FOUND DURING EXPERIMENTATION

Symbol	Value	Description
h	30	Previous time steps used by the encoder
L_e	3	Number of encoder layers
L_d	1	Number of decoder layers
m_h	8	Number of heads in the Attention layer
d_m	128	Number of features at each time step
e	200	Number of epochs
B	7	Batch size
η	0.0001	Learning rate
J	MSE	Cost function

Thus, note that it is possible to use this architecture in an autoregressive way, and estimate the evolution of the state of a dynamical system for any prediction horizon T as long as the control inputs are known.

V. TRAINING ALGORITHM AND HYPERPARAMETERS

Each iteration of the AR-Transformer not only depends on the h previous control inputs, but also uses the h last estimated states of the system. This leads to an autoregressive approach, where the model estimations are used to make the next predictions. To train an autoregressive NN there are two options, training with teacher forcing (TF) or training without teacher forcing (no-TF) [17].

The TF training consists in using during the training phase the actual states of the system y_t, instead of the estimations $\hat{y}_t$ made by the architecture. This approach has some advantages, such as shorter convergence times, simpler training algorithm and better parallelization of the training. However, the main drawback is that the NNs trained with this approach present more error with long prediction lengths [18].

In the no-TF approach the architecture is trained using as input in each iteration the previous h estimations of the state. It is trained in an autoregressive way, similar to when the architecture is executed in the inference phase. In our scenario, $X_{t,t-h}$ (5) is used as input in each iteration. In our experimentation we have observed that the no-TF approach, although it is a slower training process, allows us to obtain a much more robust architecture at long prediction times when we have an autoregressive architecture.

The AR-Transformer architecture has a number of hyperparameters that need to be adjusted during the training phase. Table I defines the hyperparameters that have the greatest impact on the performance and size of the architecture together with the best values found after a gridsearch in the example used during experimentation.

VI. RESULTS

In this section we will validate the proposed architecture with experiments. We have selected the dataset proposed in [19]. In that paper, a dataset based on real experiments of a 3 DOF robotic arm system is presented, similar to the one shown in Fig. 1. The robot arm has three joints, each providing one degree of freedom. It is controlled by sending desired torque commands to the motors at each joint. The setup includes a variety of controllers to assess the transferability of learned dynamics models. In particular, we have used the dataset that employs feedback policies, which generates the control inputs with a PD controller. The dataset consists of 50 series of experiments, where each series consists of 14 seconds. The control and observation rate is 1000 Hz, meaning that from each series we have 14000 time steps. At each time step t, we have as control input u_t the three desired torques (Nm) sent to the motors, where $n = 3$. The state y_t is six-dimensional, where $m = 6$, containing measured angles (rad) and measured velocities (rad/s). The 50 series of experiments are divided into 38 for training, 3 for validation and 9 for testing.

In addition to the 9 test sets, which we will call D_0, the dataset has tests to measure the transferability of the trained architecture, with 3 datasets designed to evaluate the performance of the architecture in conditions different from those seen during training:

- D_1: Trajectories recorded with a closed-loop controller tracking sine waves of high angular frequency, constrained to a restricted area within the robot's entire range of movement.
- D_2 Trajectories with low-frequency sine waves but utilizes the full range of safe joint movements.
- D_3: High-frequency sine waves across the full range of joint movements, presenting a challenging scenario where both the input frequency and the spatial constraints are different from the training conditions.

Note that in the training series and at D_0, trajectories are recorded with low frequency sine waves restricted to an area within the entire range of motion of the robot.

The metrics used during training and testing are listed below. The objective of the metrics is to evaluate the estimations of the architecture during N time steps $\hat{Y}_{1,N}$ with respect to the real values $Y_{1,N}$. The chosen metrics are Mean Square Error (MSE) and Mean Absolute Error (MAE).

$$\text{MSE}(Y_{1,N}, \hat{Y}_{1,N}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{13}$$

$$\text{MAE}(Y_{1,N}, \hat{Y}_{1,N}) = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{14}$$

MSE is a metric widely used as a cost function in NN training. MAE is a metric that respects the original units of the data, and is less sensitive to outliers, being easier to interpret.

To optimize the hyperparameters of the AR-Transformer, a gridsearch was performed. The best results on the training and validation data were obtained with the hyperparameters shown in Table I. The Adam algorithm has been used as optimizer [20]. The prediction horizon during training, validation and testing has been set at 1000 time steps.

In addition to calculating the average error as a validation metric, we calculate the average ranking in the same way proposed in [19]. This metric is the average of the error of the angles and velocities, losing the physical interpretability but obtaining a single metric to evaluate and compare architectures

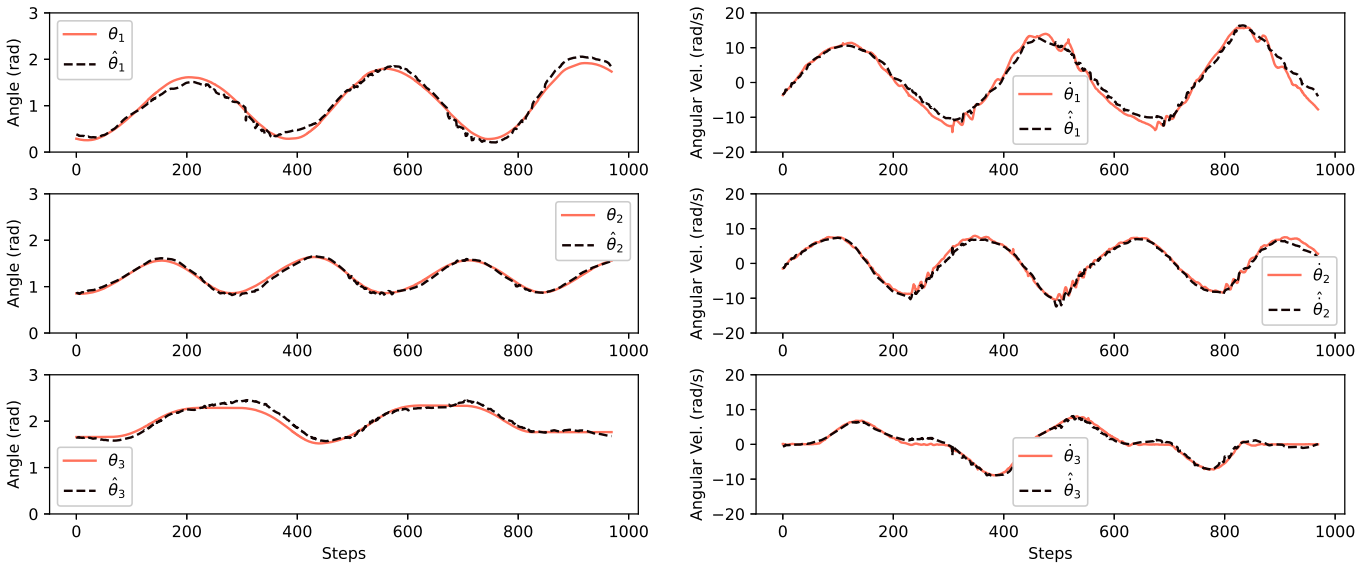

Fig. 3. Actual angles θ and angular velocities $\dot{\theta}$ with their estimations $\hat{\theta}$, $\hat{\dot{\theta}}$ with negligible error using the AR-Transformer in the first 1000 time steps of D_0, estimated from 30 known time steps. The sampling frequency was 1000 Hz, so each time step corresponds to 0.001s. The plots on the left correspond to the 3 joint angles and those on the right correspond to their 3 angular velocities.

TABLE II
AR-TRANSFORMER MAE AND AVERAGE RANKING IN THE TESTS

Architecture	Test	Angle MAE (rad)	Velocity MAE (rad/s)	Average Ranking
AR-Transformer	D_0	0.11	0.86	0.48
AR-Transformer	D_1	0.16	1.98	1.07
AR-Transformer	D_2	0.35	2.43	1.39
AR-Transformer	D_3	0.32	3.35	1.83
NN-hist1 [19]	D_0	-	-	≈ 2
NN-hist1 [19]	D_3	-	-	≈ 2.5

in a simple way, where lower is better. Table II shows the MAE of the angles and velocities, and the average ranking, for the 4 datasets of test data. Using AR-Transfromer, in the case of D_0 the error is very small, and as expected, in the transferability tests (D_1, D_2, D_3) this error increases slightly. In [19] is evaluated as a benchmark 14 methods, obtaining the best results with a NN with an average ranking of approximately 2 for D_0 and 2.5 for D_3. In our case, AR-Transformer outperforms these metrics obtaining 0.48 and 1.83 average ranking for D_0 and D_3 respectively.

Fig. 3 shows the angle and velocity estimations of the AR-Transformer with negligible error in the first 1000 time steps of D_0. Fig. 4 shows the angular velocity estimation at the third joint in order to observe how the AR-Transformer performs in the transferability tests. The reason for showing this joint is that, in the transferability tests, it is where the behavior varies the most, going from a range of values from -12 to 12 rad/s in the training data to a range of -30 to 30 rad/s in D_3. As can be seen, in D_1 and D_2 the estimation error is still small. In D_3, although more noise is present, the estimation is good. The results in the transferability tests are very good considering that the conditions and ranges of values are different from

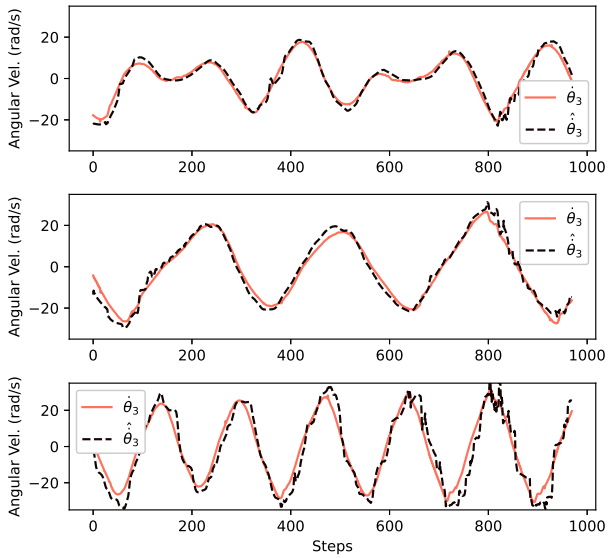

Fig. 4. True angular velocity $\dot{\theta}_3$ and AR-Transformer estimation $\hat{\dot{\theta}}_3$ at the third joint. First and second plots correspond to 1000 time steps of D_1 and D_2 respectively, showing a small error. These results are good considering that the conditions and ranges shown in these tests are different from those observed during training. The third plot corresponds to D_3 where, although the results are good, more noise is observed.

those observed during training.

In addition to the results shown, we have measured the time of the AR-Transformer in estimating 1000 time steps on an NVIDIA GeForce RTX 2060 GPU. We obtained 18.36 s, which is 18.36 ms per time step, or 54.46 Hz. These results are positive and show a low computational cost.

The architecture shows good results and generalization capability in the transferability tests, however, the robustness

of the model to perturbations is not measured in these tests. For example, cases where the robot is pushed or the robotic arm picks up an object are not tested. In these situations, the model estimation would eventually diverge. A possible solution would be to consider these situations by adding more input parameters to the model. For example, in the case of picking up an object, the input parameters indicating when an object is picked up and its characteristics could be useful. Another simpler solution would be to periodically measure the actual state of the robot, to correct the perturbations, and continue the estimation from that instant.

VII. Conclusion

We propose AR-Transformer, a new architecture based on autoregressive transformers. In the state of the art of modeling robotic systems with NN we found solutions based on RNN. However, we detected a lack of proposals with Transformers, despite being the state of the art in NN-based solutions for many problems. The AR-Transformer allows modeling robotic systems and predict their state evolution from any number of time steps of control inputs. Our proposal uses an autoregressive state estimation approach, that works for large prediction horizons by using the no-TF training algorithm.

Our AR-Transformer architecture has several novelties compared to a basic transformer. We keep the encoder-decoder structure, where the encoder calculates the context from the last h control inputs and previous estimated states. The decoder estimates the next state from the context, current control inputs, and the previous estimated state. We propose processing the input data matrices in a FeedForward layer, called Feature Projection, in order to increase the feature dimension and obtain more complex and rich features. The main novelty in the encoder and decoder layers is that, instead of using a residual connection, we use a concatenation to maintain the original information since the next state of a robotic system is highly dependent on the previous one.

Our approach has been evaluated using a public dataset of real experiments of a 3 DOF robotic arm, and with the AR-Transformer we improved the results presented in the original paper associated to the dataset. We have shown the potential of the AR-Transformer in the modeling of robotic systems. In view of the good performance of this architecture with the dataset based on real experiments, we consider as future work to test this architecture in real applications, including modeling, identification and control of robotic systems. Additionally, a more extensive comparison with other datasets and NNs is also of interest. It would also be interesting to analyze the generalization of the proposed architecture to other different robotic systems and dynamical systems.

Acknowledgment

This work was funded by the Government of Aragón, group T45_23R, and by projects CPP2021-008938, PID2021-124137OB-I00 and TED2021-130224B-I00, funded by MCIN/AEI/10.13039/501100011033 and by the European Union-NextGenerationEU/PRTR.

References

[1] C. Armanini, F. Boyer, A. T. Mathew, C. Duriez, and F. Renda, "Soft Robots Modeling: A Structured Overview," *IEEE Transactions on Robotics*, vol. 39, no. 3, pp. 1728–1748, 2023.

[2] C. Mavrogiannis, F. Baldini, A. Wang, D. Zhao, P. Trautman, A. Steinfeld, and J. Oh, "Core Challenges of Social Robot Navigation: A Survey," *ACM Transactions on Human-Robot Interaction*, vol. 12, no. 3, pp. 36:1–36:39, 2023.

[3] T. Zhang, G. Kahn, S. Levine, and P. Abbeel, "Learning deep control policies for autonomous aerial vehicles with MPC-guided policy search," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 528–535.

[4] N. Mohajerin and S. L. Waslander, "Multistep Prediction of Dynamic Systems With Recurrent Neural Networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 11, pp. 3370–3383, 2019.

[5] I. Lenz, R. Knepper, and A. Saxena, "DeepMPC: Learning Deep Latent Features for Model Predictive Control," in *Robotics: Science and Systems XI*, 2015.

[6] D. Chen, S. Li, and L. Liao, "A recurrent neural network applied to optimal motion control of mobile robots with physical constraints," *Applied Soft Computing*, vol. 85, p. 105880, 2019.

[7] L. Bauersfeld, E. Kaufmann, P. Foehn, S. Sun, and D. Scaramuzza, "NeuroBEM: Hybrid Aerodynamic Quadrotor Model," in *Robotics: Science and Systems XVII*, 2021.

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is All you Need," in *Advances in Neural Information Processing Systems*, vol. 30, 2017, pp. 6000–6010.

[9] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah, "Transformers in Vision: A Survey," *ACM Computing Surveys*, vol. 54, pp. 200:1–200:41, 2022.

[10] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang, "Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 12, pp. 11 106–11 115, 2021.

[11] H. Wu, J. Xu, J. Wang, and M. Long, "Autoformer: Decomposition Transformers with Auto-Correlation for Long-Term Series Forecasting," in *Advances in Neural Information Processing Systems*, vol. 34, 2021, pp. 22 419–22 430.

[12] T. Zhou, Z. Ma, Q. Wen, X. Wang, L. Sun, and R. Jin, "FEDformer: Frequency Enhanced Decomposed Transformer for Long-term Series Forecasting," in *Proceedings of the 39th International Conference on Machine Learning*, 2022, pp. 27 268–27 286.

[13] R. Yang, M. Zhang, N. Hansen, H. Xu, and X. Wang, "Learning Vision-Guided Quadrupedal Locomotion End-to-End with Cross-Modal Transformers," 2022. [Online]. Available: 10.48550/arXiv.2107.03996

[14] D. S. Chaplot, D. Pathak, and J. Malik, "Differentiable Spatial Planning using Transformers," in *Proceedings of the 38th International Conference on Machine Learning*, 2021, pp. 1484–1495.

[15] H. Kim, Y. Ohmura, and Y. Kuniyoshi, "Transformer-based deep imitation learning for dual-arm robot manipulation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 8965–8972.

[16] M. Shridhar, L. Manuelli, and D. Fox, "Perceiver-Actor: A Multi-Task Transformer for Robotic Manipulation," in *Proceedings of The 6th Conference on Robot Learning*, 2023, pp. 785–799.

[17] R. J. Williams and D. Zipser, "A Learning Algorithm for Continually Running Fully Recurrent Neural Networks," *Neural Computation*, vol. 1, no. 2, pp. 270–280, 1989.

[18] M. Sangiorgio, F. Dercole, and G. Guariso, "Forecasting of noisy chaotic systems with deep neural networks," *Chaos, Solitons & Fractals*, vol. 153, p. 111570, 2021.

[19] D. Agudelo-Espana, A. Zadaianchuk, P. Wenk, A. Garg, J. Akpo, F. Grimminger, J. Viereck, M. Naveau, L. Righetti, G. Martius, A. Krause, B. Scholkopf, S. Bauer, and M. Wuthrich, "A Real-Robot Dataset for Assessing Transferability of Learned Dynamics Models," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 8151–8157.

[20] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," 2017. [Online]. Available: 10.48550/arXiv.1412.6980