Learning of Multi-Context Models for Autonomous Underwater Vehicles

Bilal Wehbe[1,2], Octavio Arriaga[1], Mario Michael Krell[2], and Frank Kirchner[1,2]

***Abstract*— Multi-context model learning is crucial for marine robotics where several factors can cause disturbances to the system's dynamics. This work addresses the problem of identifying multiple contexts of an AUV model. We build a simulation model of the robot from experimental data, and use it to fill in the missing data and generate different model contexts. We implement an architecture based on long-short-term-memory (LSTM) networks to learn the different contexts directly from the data. We show that the LSTM network can achieve high classification accuracy compared to baseline methods, showing robustness against noise and scaling efficiently on large datasets.**

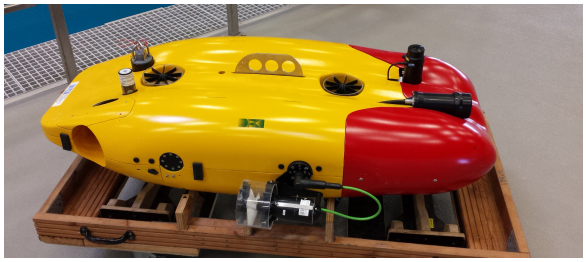


Fig. 1. The AUV Dagon used in our experiment.

I. INTRODUCTION

A robotic model is an essential tool for control, action and path planning, and several other applications. Classically, models were manually engineered by humans for specific robotic designs and applications, which restrict their usability when the environmental conditions or the robot mechanics are non-stationary. This issue as well presents itself critically in marine robotics, where robots have to operate persistently in harsh and unpredictable environments for weeks or even months. Machine learning methods can avoid the manual hand crafting of robotic models, and instead learn these models directly from the data streams acquired by the robot during operation. Furthermore, machine learning can generalize better on larger state space of the model and take into account nonlinearities that are most of the times neglected by classical physics-based approaches [1].

Model learning has been proven to be an efficient methodology in various robotic domains such as inverse kinematics and dynamics control, robot manipulation, locomotion or navigation. However learning these models may not always be straightforward and is still being faced with several challenges [1]. For most applications, model learning is regarded as a regression problem mapping the robot's states and actions. However, in cases when the robot's dynamics or operating environment are non-stationary, standard regression techniques cannot be used since they are not able to represent the full state space of the model. This problem is commonly known as multi-context learning [2]. The common challenges that face multi-context learning is discovering the correct number of contexts present in the data space, and identifying the current context the robot is in at a certain time. Furthermore, the incomplete state space of the sampled data, poses one of the major problems for learning algorithms. Generally any learning method requires a large and rich enough dataset to be able to generalize properly. In such cases, learning from simulation can help improve the model by providing an alternative for missing data.

In this work, we study the case of an autonomous underwater vehicle (AUV) where many factors such as salinity or density fluctuations, biofouling, or body damage can cause a change of the robot's model context. We aim to classify the different contexts of an AUV model resulting from disturbances in its dynamics. First, we generate a simulation model of the AUV Dagon (Fig. 1) that is learned from real data collected from experiments. We then induce several faults into the simulated model and generate a sufficiently rich dataset which contains different model contexts and covers a large area of the model's state space. We regard the multi-context learning as a multi-class classification problem, where each context of the robot model is assigned to a unique label. Using the generated dataset, we build a gating network that classifies the correct model context seen by the robot at a certain time. We build the gating network using a long-short-term-memory (LSTM); therefore, modeling the data as a time series. We show that the LSTM network can achieve a better performance when compared to standard classification methods such as support vector machines (SVM), random forests (RF), and multi-layer perceptrons (MLP).

A. Related Work

Multi-context learning has been recently a quite active topic for robotic model learning, most frequently in the area of inverse dynamics modeling for manipulator arms [2], [3]. The mixture of experts (ME) approach [4] is a frequently used method for learning multi-context models, where the data is clustered into smaller groups and subsequently a local model (or an expert) is built for each cluster. An infinite mixture of linear experts was used in [2] to capture the different contexts of an inverse dynamic model for a humanoid arm manipulating objects with different weights. However, since the linear experts only model the system locally, this causes the number of experts to increase quickly as the system perceives new contexts. This method results in the number of

[1] DFKI - Robotic Innovation Center, Bremen, Germany.
{first name.last name}@dfki.de
[2] Robotics Research Group, University of Bremen, Germany.

experts not representing the correct number of contexts, for example around 60 experts were needed to represent only two contexts in [2]. A mixture of Gaussian process (GP) experts was used in [3] to model different contact models for a humanoid manipulator arm. An SVM was used as a gating network to select between the different contexts, which achieved an accuracy comparable to a manually tuned heuristic method described in the same paper. A multi-context model of a wheeled mobile robot was learned using an infinite mixture of Gaussian process experts in [5]. Here, a Dirichlet process (DP) gating network was used instead, which allows the classification of different contexts in an unsupervised manner. One disadvantage of this method is that clustering via the DP may not always predict the true context, but can easily get confused depending on the density distribution of the training data. For example, two batches from the same context would be classified as two different contexts if they have different densities. Another drawback is that all training samples have to be used for a prediction.

While GPs are the state of the art methods for robotic model learning, it is a well known fact that their computational complexity scales cubically with the number of training samples $O(n^3)$. This fact makes GPs unlikely to benefit from having big datasets that cover larger regions of the model's state space. A recent study [6] shows that LSTM networks with a training time complexity of only $O(n)$, can outperform GPs for learning the inverse dynamics of a manipulator arm. The fact that LSTMs exploit the temporal correlations in the data makes this method suitable for learning dynamic models. Furthermore, their good computational efficiency and their scalability on big datasets is a major advantage for long-term learning settings.

In the context of marine robotics, the application of machine learning is still relatively scarce. For instance, convolutional neural networks were used for sonar image recognition in [7], moreover, the classification of autonomous underwater vehicle (AUVs) trajectories was studied in [8]. Yet model learning for underwater vehicles is still understudied in this field. Locally weighted projection regression was used to identify the mismatch between the physics based model and the data output from the vehicle's navigation sensors in [9]. In [10], a nonlinear auto-regressive network with a gating network based on a genetic algorithm was used to identify a simulated model of an AUV with variable mass. In previous work [11], [12] we showed that support vector regression can be a good candidate for learning AUV dynamic models.

In this work, we build upon our previous findings, focusing our efforts on finding a good candidate for a gating network which can provide an accurate classification of different model contexts while at the same time being able to scale on big datasets.

II. Problem Statement

We start with stating the formulation of the dynamic model of AUVs first. Next, we describe the functionality of the gating network as a classifier for different model contexts, followed by a description of the methods being evaluated.

A. The Dynamic Model

Following the notation of [13], the nonlinear 6-degrees-of-freedom (DOF) equations of motion of an underwater vehicle are generally described by

$$\dot{\boldsymbol{\eta}} = J(\boldsymbol{\eta})\boldsymbol{\nu}\,, \tag{1}$$

$$M\dot{\boldsymbol{\nu}} + C(\boldsymbol{\nu})\boldsymbol{\nu} + d(\boldsymbol{\nu}) + g(\boldsymbol{\eta}) = \tau + \zeta(\boldsymbol{\eta}, \boldsymbol{\nu}, \boldsymbol{\tau})\,. \tag{2}$$

Eq. (1) represents the kinematics equation, where $\boldsymbol{\eta} = [x\ y\ z\ \phi\ \theta\ \psi]^T$ is the pose of the vehicle in a fixed coordinate frame, and $\boldsymbol{\nu} = [u\ v\ w\ p\ q\ r]^T$ is the velocity of the vehicle expressed in a body-attached frame. The term $J(\boldsymbol{\eta}) \in \mathbb{R}^{6\times6}$ represents the nonlinear Euler angles transformation matrix. The dynamic equations of motion are represented in Eq. (2), where M is the total mass (dry + added mass) of the submerged vehicle, and $C(\boldsymbol{\nu})$ is the Coriolis and centripetal forces and moments. $d(\nu)$ represents the hydrodynamic damping, and $g(\boldsymbol{\eta})$ accounts for the buoyancy and gravitational efforts. τ is a vector denoting the actuators forces and moments, and ζ is a term representing all unmodeled dynamics and sensor noise. We follow the definition of [13] for all terms except for the damping term $d(\boldsymbol{\nu})$, where we use the formulation of McFarland and Whitcomb [14],

$$d(\boldsymbol{\nu}) = \left(\sum_{i=1}^{6} |\boldsymbol{\nu_i}| D_i\right)\boldsymbol{\nu}\,, \tag{3}$$

where $D_i \in \mathbb{R}^{6\times6}, i = 1, ..., 6$ are six matrices representing the fully coupled quadratic damping. The choice of this damping model is based upon our findings in [11], where we showed that the McFarland-Whitcomb model achieves a decent performance amongst physics-based models. We rewrite Eq. (2) as a forward model denoted by

$$\dot{\boldsymbol{\nu}} = \mathcal{F}(\boldsymbol{\eta}, \boldsymbol{\nu}, \boldsymbol{\tau})\,, \tag{4}$$

where $\mathcal{F}$ represents the dynamics function resulting from rearranging Eq. (2).

We define a context $(\boldsymbol{c})$ of the model as a unique set of the hydrodynamic parameters underlying the function $\mathcal{F}^{\boldsymbol{c}}$. In other words, any changes in the set of the model parameters (mass, coriolis, damping, buoyancy, ...) will result in a different model context. For practicality reasons, we will constrain the set of possible contexts to a finite set $\{\mathcal{F}^{\boldsymbol{c}}, \boldsymbol{c} = 1, ..., n\}$. Accordingly, we will denote a dataset sampled from a model context $\mathcal{F}^{\boldsymbol{c}}$ as the vector formed by the model's states and control inputs, $\mathfrak{D}^{\boldsymbol{c}} = \langle \dot{\boldsymbol{\nu}}, \boldsymbol{\nu}, \boldsymbol{\eta}, \boldsymbol{\tau} \rangle^{\boldsymbol{c}}$.

B. The Gating Network

The goal of the gating network is to be able to infer from an observation $\mathfrak{D} = \langle \dot{\boldsymbol{\nu}}, \boldsymbol{\nu}, \boldsymbol{\eta}, \boldsymbol{\tau} \rangle$, the true underlying model context currently in action. The gating network (Fig. 2) can be seen as a decision layer that determines the active context. We implement the gating network in this work as a multi-class classifier taking the state and control input observations as a feature inputs, denoted as $Clf\,(\dot{\boldsymbol{\nu}}, \boldsymbol{\nu}, \boldsymbol{\eta}, \boldsymbol{\tau})$.

We differentiate between two types of gating networks, temporal and non-temporal. As their name infers, temporal

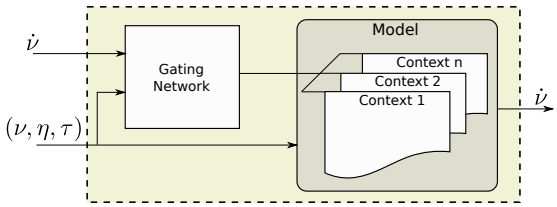


Fig. 2. The gating network classifier which acts as a decision layer on which context to select.

learning algorithms can model the temporal relations in the data if they exist, taking in data in the form of a time-series of shape $(n,\ l,\ f)$. Here, n is the number of samples, l is the series length (or time-steps), and f is the number of features. Non-temporal algorithms do not model any time dependency in the data, and thus take a feature vector as input, of shape $(N,\ f)$, where in this case $N = n \times l$.

C. Non-Temporal Learning

We use three standard classifiers as our baseline namely SVM, MLP, and random forest (RF) classifier. We won't consider any method involving GPs in this study due to their cubic time complexity, which makes such method impractical to train on a datasets larger than 10K samples. In the following, we describe briefly the non-temporal baseline algorithms we use to classify the model contexts.

1) Support Vector Machine: SVMs are binary classifiers that map the input vectors into a higher dimensional feature space using a kernel transformation, and then fit a decision hyperplane linearly onto this space [15]. For classifying multiple classes with SVM, the problem is split into multiple binary classification problems. In our implementation, we distinguish between every pair of classes, known as the one-versus-one (ovo) approach, to avoid class imbalance problem that would come with the one-vs-all approach. As kernel, we use the radial-basis-function (RBF). The hyperparameters to be optimized for the SVM are given as (C, γ), where C is a regularization parameter and γ is the kernel's length scale.

2) Multi-layer Perceptron: The basic element of an MLP is a single neuron, which is a linear weighted sum of several inputs with a non-linear activation function at its output [16]. Several neurons can be stacked to form one layer, and thereafter several layers are connected to form an MLP. MLPs are trained using the back-propagation method. To perform classification tasks, MLPs minimize a cross-entropy loss function, with a softmax activation at the last layer to handle multi-class problems [16]. For our application, we use a standard architecture with three hidden fully-connected layers with a size of [256, 512, 128] respectively. Every layer uses a ReLU [17] activation function, with and a dropout [18] of 50% to reduce overfitting. The output layer uses a softmax activation as mentioned earlier. The total number of parameters used is 201,098.

3) Random Forest: RF is an ensemble method which fits a set of decision trees classifiers by randomly sampling from the training set with replacement. The decision of the RF is computed by averaging the prediction of the individual classifiers [19]. Additionally, RFs can naturally handle multiple classes. We optimize two hyperparameters for this method, namely the number of individual trees used and the maximum depth of a each tree.

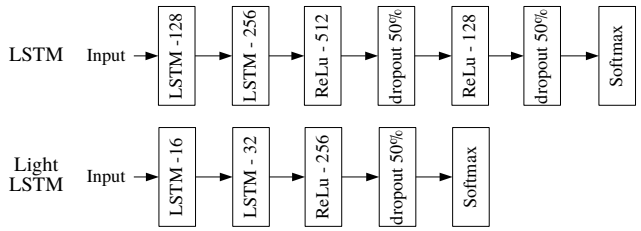


Fig. 3. Architectures of LSTM networks. **Top:** standard LSTM network with two LSTM layers, two Dense layers with dropouts and a softmax output layer. **Bot.:** Light LSTM architecture with two small LSTM layers, one Dense layer and a softmax output.

D. Temporal Learning with LSTM Networks

LSTM networks are recurrent neural networks used for modeling long-term dependencies in time series. LSTMs avoid the vanishing gradient problem that classical recurrent networks suffer from, by adding special types of forgetting and remembering gates [20]. LSTMs have a chain like structure with a repeating module. The repeating module consists of four networks (also called gates) namely, input and output gates, a forget-gate and a state-update-gate. A detailed explanation can be found in [20].

We implement two architectures of LSTM networks Fig. (3) as follows. The first network has a bigger architecture, with an LSTM input layer of 128 nodes, followed by another hidden LSTM layer with 256 nodes, then followed by two fully-connected (dense) layers with 512 and 128 nodes, respectively. Both dense layers have a dropout of 50% each to avoid overfitting. The output is a softmax layer, and the total number of parameters is 662,531. We denote the second network as "Light LSTM" due to its smaller configuration, including two LSTM layers with 16 and 32 nodes respectively, one dense layer with 256 nodes and 50% dropout, and a softmax output layer. The overall number of parameters of light LSTM is 17,155.

III. Evaluation and Results

First, we briefly describe the robotic platform used for our experiment, and the data collection procedure. Next, we describe the simulation model we fit using the collected dataset, and the generation of the synthetic datasets. We report afterwards, the training and model evaluation procedures, and compare the testing results of each model.

A. The Robotic Platform

We use the AUV Dagon (Fig. 1) as a testing platform to collect the data needed to fit the model described in Eq. (2). Dagon is a hovering type AUV that can be actuated in five DOFs (roll is passive). A detailed description of the vehicle can be found in [21]. To reduce the dimensionality of the problem, we stabilize Dagon in the pitch and heave DOFs, and let it run freely in surge, sway and yaw DOFs. We actuate

TABLE I

DESCRIPTION OF DIFFERENT MODEL CONTEXTS

label	context description
class 0	nominal model
class 1	thruster 1 damage
class 2	damping change - surge
class 3	thruster 1 broken
class 4	damping change - yaw
class 5	random damping change 1
class 6	random damping change 2
class 7	random damping change 3
class 8	random thrusters configuration 1
class 9	random thrusters configuration 2

the three lateral thrusters with a sinusoidal signal of varying frequencies in order to cover as much range as possible of the model's state space. A more detailed description of the data collection procedure can be found in [11]. Using this dataset with a total of 2063 samples, we identify the parameters of Eq. (4) by minimizing the sum of squared errors.

B. Synthetic Data Generation

We generate our datasets by running the resulting model through an ordinary-differential-equation (ODE) solver. As an input signal, we give the rotational velocity of each thruster in the form of a sinusoidal signal with a period randomly selected between 20 and 70 seconds. For each model context, we run the simulation for 40K seconds, randomly changing the value of the sine periods every 1000 seconds. We sample the simulations with a frequency of 1 Hz resulting in a dataset with 40K samples per context. The choice of these values was made in order to have a large enough dataset that covers as much area as possible of the model's state space. We assume the disturbances representing different contexts are in the robot frame. By inducing disturbances in the simulation model, we generate data for 10 different model contexts described in Table I. Disturbances in classes 1 to 4 were manually selected, whereas for the rest a we applied a random disturbance to the damping and thrusters coefficients using a uniform distribution bounded to [0.5, 3]. We follow this approach to ensure having physically realistic models that simulates random disturbances that might happen in a real world scenario.

C. Training the Gating Networks

In this section, we describe the training procedure of the different methods. Moreover, we study the effect of incrementally adding more classes on the performance of each classifier. From the generated dataset described in Sec. III-B we construct three sets, $\left\{\mathfrak{D}^{\mathbf{3}}, \mathfrak{D}^{\mathbf{6}}, \mathfrak{D}^{\mathbf{10}}\right\}$, where $\mathfrak{D}^{\mathbf{3}}$ contains classes 0 to 2, $\mathfrak{D}^{\mathbf{6}}$ contains classes 0 to 5, and $\mathfrak{D}^{\mathbf{10}}$ contains all 10 classes.

We split the generated datasets into 3 consecutive (no shuffle) subsets: training, validation, and testing. The training and validation sets are used to run a grid-search to find the best hyper parameters for the classifiers. The classifier with the best validation accuracy is then evaluated on the testing set. To keep the classes balanced, we use a stratified split with a ratio of $60/20/20$ % for training, validation and testing respectively. We train the LSTM networks using backpropagation-through-time method described in [20], and run several passes over the whole training set (epochs). The learning rate is reduced by a factor of 0.9 after 10 epochs if no improvement in the validation loss is observed. The training is stopped after 20 epochs with no improvement in the validation loss or if the improvement is less than 10^{-4}. We run this process as a grid-search for different time-series lengths. Fig. 4 reports the validation accuracy plotted against the number of epochs for the two LSTM networks, where the standard LSTM requires less epochs to converge than the ligh LSTM architecture. Although the time per epoch for the light LSTM is less than that of the standard LSTM, the total time required by the light version is slightly higher.

TABLE II

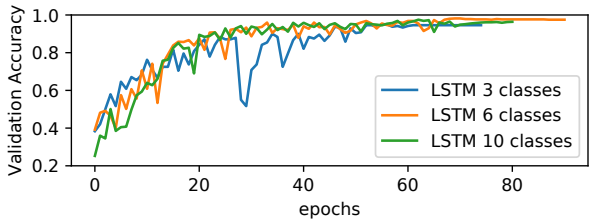
CROSS-VALIDATION RESULTS SHOWING BEST VALIDATION ACCURACIES WITH THE CORRESPONDING TRAINING ACCURACIES

Classifier	dataset	3 classes	6 classes	10 classes
LSTM	training	0.966	0.998	0.998
	validation	0.946	0.977	0.973
Light LSTM	training	0.979	0.981	0.995
	validation	0.947	0.956	0.964
SVM	training	0.854	0.844	0.847
	validation	0.783	0.779	0.788
MLP	training	0.784	0.751	0.688
	validation	0.790	0.768	0.745
RF	training	0.970	0.936	0.867
	validation	0.578	0.553	0.504

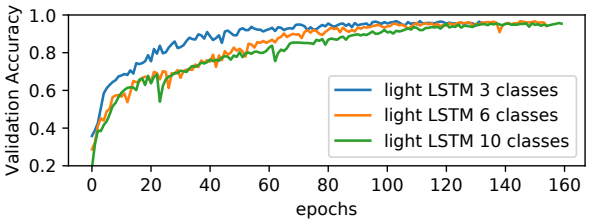
In Fig. 5 we report the validation accuracies of both LSTMs versus the length of the time-series. Both architectures achieve an accuracy higher the 95% with a time-series length of 80 and 100 samples.

We use the same mechanism to train the MLP network, except we only optimize for the number of epochs since the MLP is a non-temporal method which takes in data in the form of a feature vector. For the SVM and RF, we run a grid search over the hyperparameters described in Sec. II-C.1 & II-C.3. Consequently, the hyperparameters resulting in the best validation accuracy are selected for each classifier. Table II reports the best validation accuracy for each method. We report also the corresponding training accuracies to determine if a classifier overfits the data. An overfit can be clearly noticed with the RF classifier, where the training accuracy is much higher than the validation accuracy. Contrarily, all other methods show close values of the training and validation accuracies.

Furthermore, we compare the time complexity of the methods being evaluated. According to [20], an LSTM unit is local in space and time, meaning that the time complexity does not depend on the network size and the storage requirement is independent of the time-series length. These factors render an LSTM to scale linearly with respect to the number of training samples. On the other hand, the time complexity of an MLP with fully connected layers, scales with the number number of neurons and hidden layers. Note that the

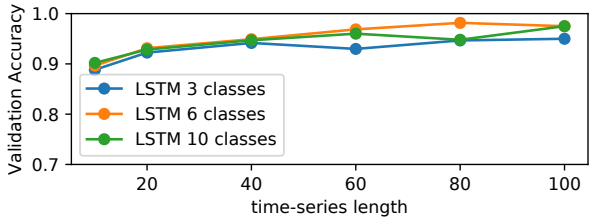


(a) Grid search vs. no. of epochs length for LSTM

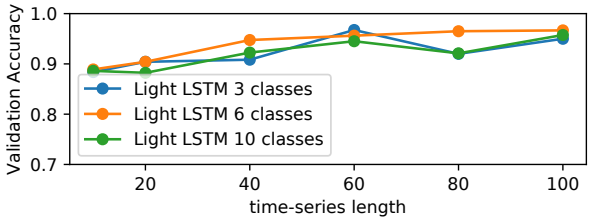


(b) Grid search vs. no. of epochs length for Light LSTM

Fig. 4. Validation accuracies with different epochs. A saturation in accuracy is observed after 60 epochs for the LSTM network, whereas the Light LSTM required around 120 epochs.



(a) Grid search vs. time-series length for LSTM



(b) Grid search vs. time-series length for Light LSTM

Fig. 5. Validation accuracies LSTM with different series lengths. Results show a saturation in accuracy of the LSTM for a series length of 100 samples. The Light LSTM shows similar results except with the 3 classes dataset.

LSTM and MLP networks were trained on a GPU operating at 1 GHz, whereas the SVM and RF where trained on a 10-core CPU operating at 3.3 GHz. To give a fair evaluation, we only compare methods trained on the same kind of processor. The processing times are depicted in in Fig. 6, where the LSTM networks show much more computational efficiency as compared to the MLP. For example in the 10 classes dataset, the MLP required an average of 39 seconds per epoch with 230 epochs to reach the best validation accuracy, whereas the LSTM required only an average of 16 seconds per epoch with 80 epochs for the validation accuracy to saturate. On the other hand, the time complexity of RFs are proportional to the number of trees in the forest, which in this case is still more efficient than the SVM which scales quadratically with the number of samples.

D. Test Results and Discussion

After selecting the classifiers with the best performances on the validation set, we evaluate how well can each classifier generalize on an unseen testing dataset that was left completely independent from the hyperparameter optimization process. Moreover, we test the effect of noise on the performance of the classifiers. For this purpose we train the classifiers with the same datasets described, but this time adding Gaussian noise similar to the natural noise observed on the robot's sensors. The comparison of testing results is reported numerically in Table III and graphically in Fig. 7. Both LSTM networks show a prediction accuracy higher than 90% in all cases, whereas none of the other baseline classifiers achieves more than 80% accuracy. With an increasing number of classes and therefore increasing number of samples, the LSTM networks maintain a consistent performance. It can also be noticed that the light

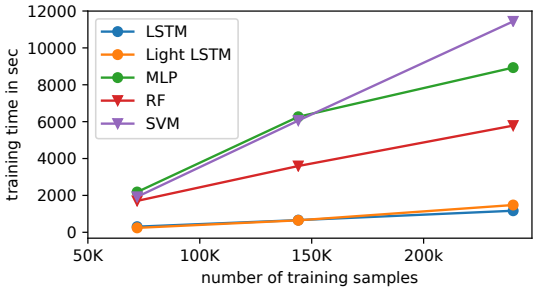


Fig. 6. Comparison of computational times for different classifiers used. Dot labels represent methods trained on GPU and triangle labels represent methods trained on a CPU.

LSTM network performs even better than the larger LSTM for the 3 classes case. This result is not surprising a large network tends more to overfit and less generalize on small training datasets. Moreover, the SVM shows a consistent performance with increasing the number of classes, whereas the performance of both the MLP and the RF degrades with additional classes. A decrease of performance is usually expected because the classification problem becomes more challenging with more classes. Eventually, with more classes come with more data, and if a classifier can take advantage of that and obtain a better overall understanding of the data, performance might increase. In addition, we analyse the effect of noisy data on the performance of the methods being tested. The LSTM networks show a high robustness against noise with only a slight drop in the accuracy with the noisy datasets. The RF shows also an invariance in the results when evaluated with noisy data, although its overall performance is the worst amongst the other methods. On the contrary, the SVM and MLP classifiers show a significant drop in performance with noisy data.

TABLE III

TEST ACCURACIES OF DIFFERENT CLASSIFIERS WITH THE EFFECT OF ADDING NOISE

Classifier	noise	3 classes	6 classes	10 classes
LSTM	no noise	0.903	**0.960**	0.949
	noise	0.900	**0.950**	**0.939**
Light LSTM	no noise	**0.950**	**0.960**	**0.958**
	noise	**0.945**	0.931	0.936
SVM	no noise	0.792	0.777	0.781
	noise	0.746	0.722	0.704
MLP	no noise	0.784	0.751	0.688
	noise	0.729	0.687	0.634
RF	no noise	0.612	0.561	0.500
	noise	0.606	0.552	0.488

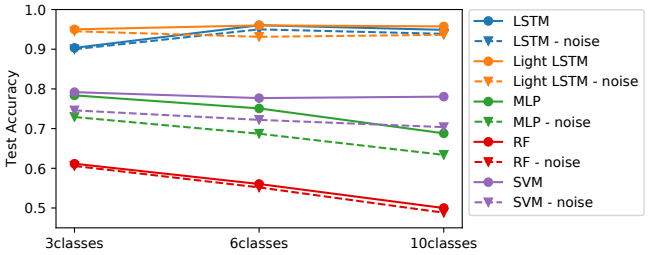


Fig. 7. Testing accuracies of the different classifiers. Dot labels represent noiseless data and triangle labels represent noisy data.

TABLE IV

TEST ACCURACIES AND TRAINING TIMES OF LSTMS WITH 100 CLASSES

Classifier	noise	Test Accuracy	Training time
LSTM	no noise	**0.994**	12865 sec
	noise	**0.992**	12798 sec
Light LSTM	no noise	0.988	14080 sec
	noise	0.985	14400 sec

Finally, we present an extreme case where we generate 90 more classes on top of the original $\mathfrak{D}^{10}$ dataset, resulting in a dataset of 100 classes with a total of 4 million samples. The additional classes were generated by inducing random disturbances onto the damping and thrusters parameters, in a similar fashion as classes 5 to 9. We evaluate only the LSTM networks on this dataset, since training the other algorithms would have taken too much processing resources. Table IV shows the test results, where both networks maintain a very high classification accuracy (98-99%). These results show clearly the capability of LSTM networks to generalize with very high accuracy on datasets that are considered extremely large in robotic applications.

IV. CONCLUSIONS

In this work, we demonstrated the capability of LSTM networks to classify accurately different contexts of an AUV model. The LSTM showed high robustness when dealing with increasing number of classes as well as the ability to generalize on noisy data, outperforming all other baseline classifiers tested in this paper. Another advantage of LSTMs is their scalability on big datasets (up to 4M samples). As future work, we aim to perform transfer learning with an LSTM network trained with simulation onto real data.

ACKNOWLEDGMENT

This work is part of the Europa-Explorer project (grant No. 50NA1704) funded by the German Federal Ministry of Economics and Technology (BMWi).

REFERENCES

[1] D. Nguyen-Tuong and J. Peters, "Model learning for robot control: a survey," *Cognitive processing*, vol. 12, no. 4, pp. 319–340, 2011.

[2] L. Jamone, B. Damas, and J. Santos-Victor, "Incremental learning of context-dependent dynamic internal models for robot control," in *Intelligent Control (ISIC), 2014 IEEE International Symposium on*. IEEE, 2014, pp. 1336–1341.

[3] R. Calandra, S. Ivaldi, M. P. Deisenroth, E. Rueckert, and J. Peters, "Learning inverse dynamics models with contacts," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 3186–3191.

[4] S. E. Yuksel, J. N. Wilson, and P. D. Gader, "Twenty years of mixture of experts," *IEEE transactions on neural networks and learning systems*, vol. 23, no. 8, pp. 1177–1193, 2012.

[5] C. D. McKinnon and A. P. Schoellig, "Learning multimodal models for robot dynamics online with a mixture of gaussian process experts," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 322–328.

[6] E. Rueckert, M. Nakatenus, S. Tosatto, and J. Peters, "Learning inverse dynamics models in o(n) time with lstm networks," in *Proceedings of the International Conference on Humanoid Robots (HUMANOIDS)*, 2017. [Online]. Available: https://ai-lab.science/wp/Humanoids2017Rueckert.pdf, Article File

[7] M. Valdenegro-Toro, "Best practices in convolutional networks for forward-looking sonar image recognition," in *OCEANS 2017-Aberdeen*. IEEE, 2017, pp. 1–9.

[8] M. D. L. Alvarez, H. Hastie, and D. Lane, "Navigation-based learning for survey trajectory classification in autonomous underwater vehicles," in *2017 IEEE 27th International Workshop on Machine Learning for Signal Processing (MLSP)*, Sept 2017, pp. 1–6.

[9] G. Fagogenis, D. Flynn, and D. M. Lane, "Improving underwater vehicle navigation state estimation using locally weighted projection regression," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 6549–6554.

[10] M. Shafiei and T. Binazadeh, "Application of neural network and genetic algorithm in identification of a model of a variable mass underwater vehicle," *Ocean Engineering*, vol. 96, pp. 173–180, 2015.

[11] B. Wehbe and M. M. Krell, "Learning coupled dynamic models of underwater vehicles using support vector regression," in *OCEANS 2017 - Aberdeen*, June 2017, pp. 1–7.

[12] B. Wehbe, A. Fabisch, and M. M. Krell, "Online model identification for underwater vehicles through incremental support vector regression," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sept 2017, pp. 4173–4180.

[13] T. I. Fossen, *Marine control systems: guidance, navigation and control of ships, rigs and underwater vehicles*, 2002.

[14] C. J. McFarland and L. L. Whitcomb, "Comparative experimental evaluation of a new adaptive identifier for underwater vehicles," in *ICRA*. IEEE, 2013, pp. 4614–4620.

[15] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, Sep 1995. [Online]. Available: https://doi.org/10.1007/BF00994018

[16] E. Alpaydin, *Introduction to machine learning*. MIT press, 2014.

[17] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 2011, pp. 315–323.

[18] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[19] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[20] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[21] M. Hildebrandt and J. Hilljegerdes, "Design of a versatile auv for high precision visual mapping and algorithm evaluation," in *2010 IEEE/OES Autonomous Underwater Vehicles*, Sept 2010, pp. 1–6.