Importance Sampling for Deep System Identification

Antoine Mahé, Antoine Richard, Benjamin Mouscadet, Cedric Pradalier, Matthieu Geist

Importance Sampling for Deep System Identification

Antoine Mahé[1]*, Antoine Richard[2]*, Benjamin Mouscadet[3], Cédric Pradalier[4], Matthieu Geist[5]

***Abstract*— This paper revisit the methodology of system identification and shows how new paradigms from machine learning can be used to improve the model identification performance in the case of non-linear systems observed with noisy and unbalanced dataset. We prove that using importance sampling schemes in system identification can provide significant performance boost on a wide variety of systems, in particular when some of the system dynamic is only exhibited by relatively rare events. The performance of the approaches is evaluated on a real and simulated drone and two standard datasets from real robotic systems. Our approach consistently outperforms baseline approaches on these datasets, all the more when the datasets are noisy and unbalanced.**

I. Introduction

Model identification is often the first step in designing the command of a dynamical system. Nowadays, the most commonly used method is Auto Regressive Moving Average (ARMA). Its simple implementation and light computational weight made of ARMA the *de facto* standard in the field of system identification for the last fifty years.

Yet, recently, building on the boom of deep learning, and dedicated hardware capable of inferring simple models more than a thousand times per second, Neural Networks (NNs) have made a remarked entrance in this field. They have produced impressive results with their high versatility, and their capacity to learn continuously over time [1]. Unfortunately, these methods require a massive amount of data to converge properly.

Regrettably, collecting data from a robotic system naturally leads to the construction of a dataset with command distribution issues [2]. The distribution of the commands on real systems can be strongly unbalanced, as we experienced in our data acquisition where most of the samples are generated with commands close to zero. Most commands sent during data acquisition comes from states where the operator (or the controller) is comfortable. This leads to some areas of the model's action space being visited only a few times while others are continually experienced. This results in limited generalization capabilities of the network.

*These authors contributed equally

[1] Antoine Mahé is with CentraleSupélec, Université de Lorraine, CNRS, LORIA, France antoine-robin.mahe@centralesupelec.fr

[2] Antoine Richard is with Georgia Institute of Technology, Atlanta, Georgia 30332–0250 arichard@georgiatech-metz.fr

[3] Benjamin Mouscadet is with CentraleSupelec, France benjamin.mouscadet@supelec.fr

[4] Cédric Pradalier is with UMI2958 GT-CNRS, France cedric.pradalier@georgiatech-metz.fr

[5] Matthieu Geist is with Google Brain, Paris, France mfgeist@google.com

To cope with those problems there has been some recent research proposing to improve the quality of deep neural networks model training by sampling a subset of the data based on how the network perform on them [3], [4], [5]. The common idea is to evaluate how useful samples are for the training, then increase or decrease their weight based on their importance.

In order to improve the ability to train NNs on unbalanced datasets we use two different sampling mechanisms. Those methods evaluate which samples are most suited to improve the model performances at a given time during training.

In this paper, prioritization using the upper-bound gradient form [6] is applied to system identification. The result from [7] are extended to standard and real world datasets. We demonstrate the method on different conditions and systems and compare the different methods : ARMA, standard NN training, prioritized experienced replay NN training, and gradient upper-bound NN training.

II. Related Work

Since the 1970's ARMA [8] has been the way to go for most black-box system identification. Such models are particularly simple to understand and implement. They rely on linear difference equations (their transfer functions are rational fractions), based on the past states and inputs of the system. The linearity of these equations enables the computation of the system parameters with a least-squares method. As a result, the fitting of an ARMA model is not computationally expensive; this makes it possible to perform efficient grid-search over the orders of the filter (orders of the numerator and denominator of the transfer function).

The last years have seen Deep Learning achieving outstanding results in a large number of tasks, over a very large field of applications ranging from computer vision to natural language processing. Control and system identification are no exceptions.

Recent attempts at black-box system identification using deep learning have shown great results, on both linear and non-linear systems [9], [10], [11]. Those works rely on architectures such as Multi Layer Perceptron (MLP), and Long Short-Term Memory (LSTM) [12].

However, to be able to train these networks a lot of data is necessary which implies an important amount of demonstration of the system in its environment. The sample inefficiency of NNs leads to the need to generate large demonstration datasets. These are often very unbalanced [2] and the training is saturated with common samples while interesting data points do not have any impact on the learning

process. All in all, properly learning hard cases is hindered by the imbalance of the data.

The capacity of NNs to keep improving over new data by continuously training has been used to alleviate this problem [1]. However, this implies that we keep training the network on very well-known situations while new pieces of information are only seen once in a while. Identifying the informative samples on which to focus the learning makes it more efficient as we will show later on.

As far as we are aware, there is only one prior attempt to perform importance sampling in neural-network-based model identification [7]. This work relies on the Prioritized Experience Replay [3] which demonstrated great results in Reinforcement Learning (RL). This method is used in the Double DQN algorithm [13] and has recently been extended to support distributed architectures [14].

Yet prioritized experience replay suffers from a major drawback. It requires the tuning of multiple parameters: alpha and beta parameter but also the number of replays to perform. It additionally relies on re-evaluating the sample sampling distribution regularly. This makes [3]'s work unpractical, as a large grid search has to be performed to acquire optimal prioritization parameters. This implies that, as the network keeps on training, the samples importance is not updated even though their usefulness evolves. This results in an efficiency loss where the network keeps on focusing on sample that used to be hard but are now properly learned. However, one of the main advantages of this method is that it permits a fine control over how the prioritization is done. This is particularly interesting as it allows to put a low amount of prioritization at the beginning of the training to grasp the general dynamic of the model, then increase the prioritization as the training reaches its end to maximize the learning of hard cases.

More recently, similar studies have been conducted on the computer vision classification task. In particular, [4] showed that the loss of the network on a given sample could be used as an indicator of the sample's importance. However, [6] also outlines that using the loss can results in degraded learning performances in some specific cases. Fortunately, they show an interesting mechanism that alleviates both the tedious parameter tuning present in the prioritized experience replay and the need to update the sample importance based on the most recent network state. In their experiments the loss is no longer used as an estimate of a sample importance. Instead, they rely on an estimation of the gradient norm. Note that one could use the real gradient, but the computational cost is prohibitive. The prioritization weights are no longer saved for the time of the epoch but recomputed at every iteration on a super-batch[1]. This has two advantages: first as the super-batch is sampled uniformly, it naturally reduces the over-fitting risks; second, it ensures that the scores are accurate for this instance of the network, thus preventing the risk to keep giving a high importance to samples that have become easy and further mitigating the over-fitting risk. Nevertheless, these advantages come at a cost: because the scores have to be recomputed at every iteration, the training process is slower than before. To get over that problem, a trigger is introduced to perform importance sampling only when we estimate that gradient acceleration is possible. This trigger is computed for free when the backward pass is performed. All in all, they showed that their approach out-performed all previous prioritization methods on the classification task.

In this work, we apply various importance sampling methods to system identification. Those approaches are evaluated on standard datasets [15] and custom datasets that exhibit various degrees of non-linearity, unbalancing and noise. Building on the promising simulated result of [7] we expand the prioritize experience replay method to real world data and show its applicability on standard datasets. Moreover, we propose to use an other prioritization scheme that alleviate the hyperparameter complexity of the previous algorithm.

III. Method

In this section, we detail the different approaches to system identification that we compare in our experiments. From the standard linear system identification and methods using Multi-Layer-Perceptron (MLP), we show how prioritized sampling can be adapted to the system identification task.

A. Model identification

Linear system identification of the ARMA family have been used for decades with success. When $u(t)$ and $x(t)$ respectively denote the system's input and output at time t, ARMA's model of the system is given by the following discrete-time linear difference equation:

$$x(t) + \sum_{k=1}^{p} a_k x(t-k) = \sum_{k=1}^{q} b_k u(t-k) \tag{1}$$

It is more intuitive to consider this equation as a way to determine the next output value given previous observations and a set $\theta = \{a_1, ..., a_p, b_1, ...b_q\}$ of parameters:

$$x(t) = -\sum_{k=1}^{p} a_k x(t-k) + \sum_{k=1}^{q} b_k u(t-k) \tag{2}$$

The linearity of the model makes it easy to compute the optimal parameters θ^* using the linear least-square method.

More recently, the ARMA methods have been challenged by NNs, as they expanded the range of system that can easily be modelled from data. In particular, non-linear systems are no longer an issue using a NN [16].

In this study we chose to use a standard MLP, with 2 hidden layers. Not only has this kind of networks been extensively used and studied in the last decades [1], but their simplicity also highlights the performances of our algorithms and their impact on the final results.

To properly learn the dynamic of the system, the historic of the twelve previous commands and states is used. This time horizon gives the MLP a memory of the previous events which proved to be sufficient for the considered systems. Thus, it made sense not to use LSTM as the required time

[1]A super-batch is a batch n times larger than the batch size used for training. In their work, it is suggested to chose a super-batch 3 times larger than the batch size

horizon can easily be known. Additionally, LSTMs add unnecessary complexity to this study where our focus is on the data and the optimization, not the architecture of the networks.

To learn the model, a multi-parameter regression is computed. The cost-function used is the Mean Squared Error (MSE), but its implementation within TensorFlow [17] has been slightly modified to allow easier computation of the network's gradient.

B. Prioritizing sample

Learning a dynamic which is poorly represented in the dataset is hard. This is particularly true on datasets which have not been acquired for the specific purpose of model identification. Indeed, large model identification datasets tend to be filled with redundant information. To cope with this issue, we apply importance sampling mechanisms to the training process.

Some dynamic systems are more difficult to identify than others. For instance, the velocity variations of the drone used in our experiments are way easier to identify in straight line than during a 180-degree turn. Moreover, as operators, we tend to demonstrate systems in conditions where we already have a good understanding of the system behavior. Thus, the interesting data where the network should learn the most is the rarest. This leads traditional learning approaches to fail to train on those seldom seen events.

1) Prioritize experience replay: To answer this problem, we propose to adapt the prioritization scheme introduced in [3] for reinforcement learning to the context of system identification. Indeed, prioritization forces the training on harder samples even if they are scarce. The adaptation of this sampling strategy to system identification yielded encouraging results illustrated in [7]. In practice, we use the loss of the network prediction to estimate the training value of a sample. The samples that lead to the highest errors are the one where the network has the most learning to do. Hence, the network prediction errors are collected to compute a probability distribution over the samples, which is then used for sampling the dataset for the next training session. This process is detailed in the algorithm 1.

One of the limitations of this approach is that it focuses on a small subset of samples. Although that focus improves its data efficiency, it also increases the risk of over-fitting. Noisy datasets are also hard to learn from as it is harder to make the distinction between complex cases and outliers. To mitigate those problems and make the method practical, hyper-parameters are introduced. Those parameters allow to choose how much the training should focus on hard cases. The probability of choosing the sample i during the sampling is given in eq. (3) where δ_i is the score of sample i, in our case the error between the NN prediction and the actual observation.

$$P(i) = \frac{\delta_i^\alpha}{\sum_k \delta_k^\alpha} \tag{3}$$

Algorithm 1 Prioritized Experience Replay

Require: data, K number of trials, MLP neural network model

$trainingData \leftarrow data$
$sampleWeight \leftarrow \emptyset$
for $k = 0$ to K **do**
  $MLP \leftarrow Train(trainingData)$
  N number of samples in $data$
  for $i = 0$ to N **do**
    $\delta_i \leftarrow \left\| Y_i - MLP(X_i, U_i) \right\|$
    $P(i) \leftarrow \frac{\delta_i^\alpha}{\sum_k \delta_k^\alpha}$
    $w_i \leftarrow \left(\frac{1}{N} \frac{1}{P(i)} \right)^\beta$
    $sampleWeighted \leftarrow \{w_i\}_{0 \leq i \leq N}$
    $trainingData \leftarrow$ sample data $d_i \sim P(i)$
  end for
end for

However, the sensibility to the hyper-parameters make the approach difficult to apply and makes a systematic tedious grid search mandatory to find optimal values for these parameters.

2) Gradient upper-bound: Another way to prioritize samples is to use the gradient upper-bound as explained in [6]. As its name implies, the gradient upper-bound method relies on an approximation of the norm of the network's gradient. [5], [18] showed that the gradient norm represents what a network can learn from a data-point. In comparison, the loss of the network on which the prioritized experience replays relies is a poor approximation of it. As a result, drawing from a loss-based distribution is less efficient than using a distribution homogeneous to the norm of the gradient.

Yet, computing the gradient norm is prohibitively expensive. In order to alleviate this issue [6] introduce an accurate and computationally inexpensive estimation of it, that is the gradient upper-bound. This so-called gradient upper-bound is obtained by computing the norm of the gradient between the loss and the last activation layer of the network. Furthermore, this approach, which constantly updates the probabilities of drawing the samples, has less hyper-parameters than the prioritized experience replay. Indeed, instead of updating the weights at some arbitrary training step, or epoch, this technique samples *super-batches*, i.e. n-time larger batches than the standard training batches. From these *super-batches*, a distribution based on the gradient upper-bound is computed and a standard batch is sampled from it. This as two implications: because the super-batches are sampled uniformly it reduces both the risk of over-fitting and the risk of focusing on outliers. However, depending on the size of the super-batches, the training time can be significantly extended. Algorithm 2 shows our implementation of the gradient prioritization scheme.

Algorithm 2 Gradient Prioritization

Require: *data*, N number of steps, *ssbs* super-batch size, *bs* batch size

$trainingData \leftarrow data$
for $i = 0$ to N **do**
  $super\_batch \xleftarrow{\text{ssbs}} \mathscr{U}(trainingData)$
  $g = get\_gradient\_upper\_bound(super\_batch)$
  $\mathscr{G} \leftarrow distribution\ from\ g$
  $weights \leftarrow \frac{1}{Bg}$

  $batch \xleftarrow{\text{bs}} \mathscr{G}(super\_batch)$
  $train\_step(batch, weights)$
end for

IV. Experiments

A. Evaluation

To evaluate the model, we consider two metrics. The first one, which will be referred as single step accuracy, expresses the next-point prediction accuracy. Its value is computed using the MSE of all the prediction made on the test set. This metric is used to evaluate the instantaneous prediction accuracy. The second one is a trajectory prediction accuracy, later it will be referred as multi-step accuracy. In this case a time horizon is selected (15 samples for the drone, 30 for DaISy), and the network iterates over its own predictions. The model is run over 60 different trajectories from our test set. The final metric is the MSE of those trajectories. The latter evaluation score is the one we use to select our hyper parameters during the grid search.

B. Datasets

After validating the concept on a simulated drone, we test our approach on standard system identification dataset collected from real systems. Finally, we move onto a more challenging dataset consisting of a real drone fitted with an Real Time Kinematic (RTK) GPS.

1) Parrot Bebop Drone: In order to test the validity of the new optimization method and the importance sampling, we first evaluate our approach using a simulated drone. The simulation is done in Gazebo [19], using Robotic Operating System (ROS) [20] and a drone emulated by the rospackage tum simulator[2]. The system includes the simulated drone and its low-level controller. The dataset created for this experiment is a combination of two trajectory generation algorithms. The first one only moves the drone in the plan in straight line, there is no vertical and no angular command sent. The second algorithm move the drone such as it uniformly explores the action space. while not crashing. The aggregation of samples from both data generation schemes produce an unbalanced dataset that we use to train our model. On the real drone, the data is acquired by the uniform exploration algorithm detailed earlier on. As such this dataset is balanced. To acquire the position of the drone an RTK GPS has been set up on it. The acquisition was made using the RTK mode, and not Post Processing Kinematic (PPK)[3] as this would not be representative of a real time system. Also, we rarely reached a fix solution state but were most of the time in a good float solution state. Hence the precision is roughly 20cm and we can see some discontinuities in the data. Moreover, the drone crashed into some trees. This implies that the data contains strong outliers.

2) DaISy: DaISy [15] is a large open source system identification database of real and simulated systems. Because our scope is robotics, we chose, to study two robotic systems, namely the flexible robot-arm and the CD-player arm. Those datasets are based on data from real systems.

- Flexible robot arm is a single-input single-output (SISO) system, the input command is the reaction torque of the structure, while the measured state is the acceleration of the arm. This dataset is balanced.
- For the CD-player arm, a multiple-input multiple-output (MIMO) system, the inputs commands are the actuators forces, while the output is the tracking accuracy of the arm over the x, y coordinates. This dataset is balanced.

The size of these datasets is small: about a thousand samples. They have been selected to illustrate that our methods work at least as well as previous methods even when the data is well balanced and contains few samples.

C. ARMA

In order to enable comparisons between our algorithm and a more standard system identification method, we modeled our system with an ARMA filter. The AR and MA orders are found through a grid search. The goal is to find the lowest orders providing the most accurate results.

D. Neural Networks

As described earlier we chose to use MLPs to learn the model of the different systems. To emphasize the versatility of our method, we used the same architecture to fit all of the models depicted in this paper. This architecture is similar to the one used in Auto-Rally [1]: it has two hidden layers with a width of 32. The input is made of a state vector X, and a command vector U, for all of the datasets the commands and states history are the same length. It has been set to twelve samples. In the case of the drone this represents 2.4 seconds, enough to catch the dynamics of the model. All of the models are trained using TensorFlow [17] with the ADAM [21] optimizer and a learning rate of 0.001. All the networks trained for a total of 25.000 iterations with a batchsize of 16. Finally, the input data are normalized individually, that means that each command input and state input have been normalized independently. Thus, the networks predict normalized states.

E. Prioritization

In this paper, we evaluate two prioritization schemes: first we try the sampling based on the prioritized experience replay which relies on the loss of the samples to assess the

[2] `http://http://wiki.ros.2org/tum_simulator`

[3] The PPK mode is computed offline after that the system has run.

importance of a sample and then we use the gradient upper-bound to estimate the importance of a sample.

1) Prioritized experience replay: In all of our experiments we retrain our networks five times while sampling on a loss-based distribution obtained with the previous iteration of the network. In order to choose the hyper-parameters, we perform a grid search where we look for the combination of α, β that yields the best accuracy.

α is comprised between 0 and 1, this parameter allows to increase the prioritization of hard cases. At the same time, we search for the β parameter also included within 0 and 1. Its goal is to compensate for the bias introduce by the prioritization. High values of alpha rapidly lead to over-fitting and focusing on outliers. Details about the roles of α and β can be found in [3].

2) Gradient prioritization: When training using the gradient upper-bound, the gradients are computed between the loss and the last activation layer. To do so the original implementation of TensorFlows MSE is modified to allow access to the per-example MSE instead of the batch MSE. The only parameter to tune is the super-batch size: to select it we perform a grid search were the superbatch size varies between 60 and 10000 samples per super-batches.

F. Grid Searches

To appropriately select the hyper parameters on all of our approaches we use a k-fold-cross-validation methodology. The dataset is split in five equal parts, four for training and validation, one for testing. The data allocated to the training is split into five more sets, four for training and one to validate the hyper parameters. Furthermore, the datasets are normalized: the mean and the variance of each command and state are extracted from the training set and used to normalize both on the input data and output data. Finally, each experiment is run ten times to average the results. The best hyper-parameters for a given test set are selected using the trajectory-based accuracy.

V. Results

A. Drone Simulations

The simulated dataset has the particularity to have little noise but is unbalanced. As such we expect our prioritization scheme to perform better than the regular training. The results of the experiments on the simulated drone can be seen in table I. It shows that the importance sampling scheme bring a 5 to 10% on single step accuracy. The gradient prioritization scheme noted GRAD in table I consistently outperforms the baseline, *i.e.* the training without any form of prioritization noted as STD. We can see that the Prioritized Experience Replay results are less constant, this is most probably due to the complexity of properly choosing the hyper-parameters. In multistep accuracy we can draw similar conclusion as the Gradient outperforms the non-prioritized approach. Here again the edge provided by our method allows for a 5 to 10% increase in overall performance. Please note that those results do not rely on hand picked hard cases but rather on trajectories taken at random in the test set. As such this shows that our methods improve the performance of the networks in general. Additionally, we can see that the traditional method: ARMA is nowhere close to the performances of the NNs.

Test set	0	1	2	3	4
	Single Step Accuracy				
ARMA	0.836	0.873	0.850	0.855	0.883
STD	0.311	0.346	0.109	0.329	0.481
PER	**0.298**	**0.328**	0.122	0.316	0.492
GRAD	**0.298**	0.331	**0.100**	**0.315**	**0.473**
	Multi-Step Accuracy				
ARMA	1.003	0.986	1.056	0.983	0.958
STD	0.214	**0.263**	0.166	0.257	0.709
PER	0.227	0.271	0.179	0.250	**0.685**
GRAD	**0.202**	0.271	**0.155**	**0.239**	0.705

TABLE I

Simulated Drone k-fold cross validation results. Lower is better.

B. DaISy

Those datasets are short, and their command distributions are balanced. As such our methods should have little impact on the training results, but more importantly, we are interested in assessing that in the case of a small and/or balanced dataset they do not end up degrading the training performances.

1) SISO: Robotic Flexible Arm: As shown in table II the training using Prioritized Experience Replay (PER), gives better results on both single-step accuracy and multi-step accuracy. This result is interesting as it shows that even on small balanced datasets our methods can increase the performance. Only on 1 out of the 5 test sets the training without any prioritization (STD) gave better results. On the other hand, the gradient based sampling scheme (GRAD) consistently made marginally worse predictions than the two others.

Test set	0	1	2	3	4
	Single Step Accuracy				
ARMA	0.0454	0.0364	0.0671	0.0432	0.0485
STD	0.00031	0.00162	**0.00503**	0.00610	0.00144
PER	**0.00012**	**0.00135**	0.00518	**0.00565**	**0.00038**
GRAD	0.00052	0.00188	0.00623	0.00660	0.00123
	Multi Step Accuracy				
ARMA	2.220	3.379	1.359	1.921	3.173
STD	0.00065	0.00422	**0.0104**	0.0141	0.00239
PER	**0.00033**	**0.00347**	0.0115	**0.0117**	**0.00074**
GRAD	0.00097	0.00489	0.0138	0.0148	0.00214

TABLE II

Flexible robotic arm k-fold cross validation results. Lower is better.

2) MIMO: CD Player Arm: As table III shows, the results here are blurrier, the PER still outperforms its counterparts in single-step accuracy. Yet, in multi-step accuracy, we can only conclude that the methods are equivalent as we cannot pick a method that consistently performs better than the other. However, this clearly shows that despite the small dataset

and the fact that the data are balanced our method do not decrease the training performances.

Test set	0	1	2	3	4
	Single Step Accuracy				
STD	0.0197	0.0185	0.0149	**0.0173**	0.0173
PER	**0.0191**	**0.0181**	**0.0148**	0.0180	**0.0171**
GRAD	0.0199	0.0184	0.0154	0.0175	0.0183
	Multi Step Accuracy				
STD	**0.0933**	0.0845	**0.0685**	0.0773	0.0896
PER	0.0948	**0.0825**	0.0717	0.0799	0.0850
GRAD	0.0961	0.0834	0.0751	**0.0710**	**0.0842**

TABLE III

CD PLAYER ARM K-FOLD CROSS VALIDATION RESULTS. LOWER IS BETTER.

C. Real Drone

Test set	0	1	2	3
	Single Step Accuracy			
ARMA	0.778	0.812	0.783	0.804
STD	0.179	**0.549**	0.144	0.116
PER	0.190	0.563	0.162	0.134
GRAD	**0.165**	0.560	**0.137**	**0.108**
	Multi Step Accuracy			
ARMA	1.054	1.012	1.046	1.014
STD	0.863	0.738	0.543	0.804
PER	**0.861**	**0.720**	**0.519**	**0.783**
GRAD	0.890	0.728	0.534	0.822

TABLE IV

REAL DRONE K-FOLD CROSS VALIDATION RESULTS. LOWER IS BETTER.

On the real system, our model and training methods are put to the test. The acquisition periods have to be short due to limited battery life. The weather and in particular the wind make proper model identification complicated. Additionally, we recall that we never achieved RTK fix only a good float[4] resolution. Yet, despite the aforementioned issues, our approaches do not degrade performances on this balanced dataset. To the contrary, they slightly improve the performance showing that those methods are resilient to outliers and do not over-fit on noisy data.

VI. CONCLUSION

In this paper we propose to use the gradient-upper bound method as an alternative priorization scheme for system identification. We show that, even without complex hyperparameter fine tuning, our approach achieves comparable result to previous methods on unbalanced dataset of a simulated drone. Furthermore, we expand the evaluation of these methods on standard datasets as well as on a dataset we collected with a real drone.

We provide a thorough comparison of different identification methods: ARMA, standard NNs training, prioritized experienced replay NNs training, and gradient upper-bound NNs training.

Furthermore, we show that even on small datasets, our approaches do not degrade the performances. In future research, we would like to extend this work to other real systems and to use them in combination withModel Predictive Control (MPC), and Model Predictive Path Integral (MPPI).

[4]In fix mode the RTK GPS as an accuracy bellow 5cm, in float the localization is comprised between 1m and 5cm, in our case the localization was around 20cm

ACKNOWLEDGMENTS

We would like to thank Oksana Riou and Corentin Godeau, for their preliminary work on the DaISy datasets using ARMA.

REFERENCES

[1] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou, "Information theoretic mpc for model-based reinforcement learning."

[2] S. Schaal, C. G. Atkeson, and S. Vijayakumar, "Real-time robot learning with locally weighted statistical learning," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 1. IEEE, 2000, pp. 288–293.

[3] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[4] A. Katharopoulos and F. Fleuret, "Biased importance sampling for deep neural network training," *CoRR*, vol. abs/1706.00043, 2017. [Online]. Available: http://arxiv.org/abs/1706.00043

[5] G. Alain, A. Lamb, C. Sankar, A. Courville, and Y. Bengio, "Variance reduction in sgd by distributed importance sampling," *arXiv preprint arXiv:1511.06481*, 2015.

[6] A. Katharopoulos and F. Fleuret, "Not all samples are created equal: Deep learning with importance sampling," *CoRR*, vol. abs/1803.00942, 2018. [Online]. Available: http://arxiv.org/abs/1803.00942

[7] A. Mahé, C. Pradalier, and M. Geist, "Trajectory-control using deep system identication and model predictive control for drone control under uncertain load." in *2018 22nd International Conference on System Theory, Control and Computing (ICSTCC)*, Oct 2018, pp. 753–758.

[8] L. Ljung, "System identification," in *Signal analysis and prediction*. Springer, 1998, pp. 163–173.

[9] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096–2103, 2017.

[10] T. Zhang, G. Kahn, S. Levine, and P. Abbeel, "Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search," in *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2016, pp. 528–535.

[11] J. Gonzalez and W. Yu, "Non-linear system modeling using lstm neural networks," *IFAC-PapersOnLine*, vol. 51, no. 13, pp. 485–489, 2018.

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

[13] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2015. [Online]. Available: http://arxiv.org/abs/1509.02971

[14] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. Van Hasselt, and D. Silver, "Distributed prioritized experience replay," *arXiv preprint arXiv:1803.00933*, 2018.

[15] B. De Moor, P. De Gersem, B. De Schutter, and W. Favoreel, "Daisy: A database for identification of systems," *JOURNAL A*, vol. 38, pp. 4–5, 1997.

[16] V. A. Akpan and G. D. Hassapis, "Nonlinear model identification and adaptive model predictive control using neural networks," *ISA transactions*, vol. 50, no. 2, pp. 177–194, 2011.

[17] M. A. et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[18] I. Loshchilov and F. Hutter, "Online batch selection for faster training of neural networks," *arXiv preprint arXiv:1511.06343*, 2015.

[19] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, vol. 3. IEEE, pp. 2149–2154.

[20] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.
[21] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.