Trajectory-control using deep system identification and model predictive control for drone control under uncertain load

Antoine Mahé, Cédric Pradalier, Matthieu Geist

Trajectory-control using deep system identification and model predictive control for drone control under uncertain load.

Antoine Mahé
CentraleSupélec, Université de Lorraine, CNRS, LORIA
F-57000 Metz, France
antoine-robin.mahe@centralsupelec.fr

Cédric Pradalier
UMI2958 GT-CNRS
F-57000 Metz, France
cedric.pradalier@georgiatech-metz.fr

Matthieu Geist
Université de Lorraine, CNRS, LIEC
F-57000 Metz, France
matthieu.geist@univ-lorraine.fr

***Abstract*—Machine learning allows to create complex models if provided with enough data, hence challenging more traditional system identification methods. We compare the quality of neural networks and an ARX model when use in an model predictive control to command a drone in a simulated environment. The training of neural networks can be challenging when the data is scarce or datasets are unbalanced. We propose an adaptation of prioritized replay to system identification in order to mitigate these problems. We illustrate the advantages and limits of this training method on the control task of a simulated drone.**

***Index Terms*—identification, model predictive control, neural networks, learning**

I. Introduction

The use of Unmanned Aerial Vehicles (UAV) for various inspection tasks is more and more appealing. Tasks such as precision agriculture or building inspection are benefiting greatly from the improvement of aerial robotic capabilities. However, most drone applications require an expert to accomplish their mission. This requirement can be a serious limitation in many situations. For example, underground mines inspection doesn't allow for real time communications. In such case autonomous control of the aerial robots becomes a necessity.

Trajectory planification is an active area of research. Model Predictive Control (MPC) is a control algorithm that is widely used in this context [9], [11], [15]. MPC is built on top of a model of the controlled system. The design of the model can be achieved by different methods. Model identification is one of them, which works well on a wide variety of systems. Recently machine learning and neural networks have challenged the traditional system identification tools.

We study here the advantages and limitations of using neural networks as the system model in the control algorithm. We discuss the challenge of data generation as neural networks require a large quantity of data to be trained. We investigate the importance of the data quality and explore the possibility of prioritizing data samples during model training to alleviate these difficulties.

After presenting the control algorithm used, we compare three methods for identifying the system: a traditional linear approach using the ARX algorithm, a standard machine learning approach where we train a neural network on a dataset without preprocessing and the same neural network which we trained using a new method inspired from [12]. The dynamic model is then used by the MPC to sample trajectories as in [13].

We show that the system model obtained by the neural network, especially using prioritized sampling, performs better in term of multistep errors. We test such model for the control of a simulated drone in two different situations. First, we use the drone in normal conditions then we add a suspended mass to the drone to disturb its dynamics. We discuss the gain of the neural network model in regard to its computational cost.

II. Related Work

To allow for more autonomy, control algorithms are continuously developed and improved, in particular MPC has been used for the control of drones in various settings with success [3], [10]. MPC are based on the online optimization of cost functions. In our implementation we use a version of that algorithm called Model Predictive Path Integral (MPPI) described in [13] which is able to take into account complex cost functions. The flexibility given by the design of the cost function allows to implement both objectives and constraints which is very useful in drones control as mission and security often are in competition.

Machine learning has been used with great results in recent years in several control settings [4], [15]. Neural network have shown capabilities to model complex systems which make them a tools of choice for the implementation of predictive systems. 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.

A way to lessen this burden is to collect data from the system while the algorithm is running as demonstrated in [14].

This work is done under the Grande Region rObotique aerienNE (GRoNe) project, funded by a European Union Grant thought the FEDER INTERREG VA initiative and the french "Grand Est" Region.

The model is first only trained with minimal demonstrations. The accuracy of the neural network model is then improved by collecting more data as the system operates and reuses it in a training session.

The data collection problem is often encountered in learning settings. For example the Deep Q-Networks (DQN) algorithm that solves Atari games [8] needs millions of examples to be effective. An interesting way to alleviate this problem has been developed in [12] where the samples are given importance depending on how much they are supposed to help the learning process. This is closely linked to importance sampling which also uses weighted samples [5].

In our case we propose an improvement on the retraining done in [14] by using prioritized sampling. This allows to improve sample efficiency and to counter the negative impact of unbalanced datasets. To alleviate the bias induced by our prioritization we combine it with importance sampling as in [12].

III. METHOD

A. Control

MPC uses a model of the system dynamic to predict the behavior of the system.

$$X_{t+dt} = F(X_t, U_t). \tag{1}$$

Given the state X_t and a command U_t, the model F provides the next state X_{t+dt}. More specifically, in our drone settings our state X_t is composed of a position p_t and a velocity v_t. The position update is done using simple kinematic update while the velocity update is done using the identified model f for the dynamic :

$$\begin{cases} p_{t+1} = p_t + v_t dt \\ v_{t+1} = v_t + f(v_t, u_t)dt \end{cases} \tag{2}$$

This prediction is used to optimize a cost function over a receding horizon. The first command calculated is the only one applied from the optimization before optimizing again from the new step. This lets the controller take into account future events while computing the next command. It is also possible to consider the n-first commands instead of the first one. This allows for a more flexible time window for the processing of the next command. In our implementation, we use a command buffer to ensure the rate of the control. The impact on the performance for using n-step predictions instead of one step prediction is far less than the impact of uneven rate control.

We use here the MPPI version developed in [14] which proposes to calculate the desired control from the evaluation of sampled trajectories. The approach is described in algorithm 1.

The prediction of the next step state as described here is done using the last step state and command. This can be generalized to using the N last step states and commands. The cost of a trajectory $\phi(S_k)$ is computed as the sum of the cost of the states of the trajectory. In our case, each state cost is calculated as the euclidean distance between the actual drone pose and the desire pose. It could also be derived from a cost map or be a combination of different factors.

Algorithm 1 Model Predictive Path Integral

```
Require: $F$ : Dynamic model
  $T$ : number of timesteps
  $K$ : number of sampled trajectories
  $\phi$ : cost function
  $u_t$ : commands sent at step $t$
  $s_t$ : state at step t
  $U = \boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_T$ : initial control sequence

  Sample $\epsilon_k = \epsilon_k^1, \epsilon_k^2, \ldots, \epsilon_k^T \sim \mathcal{N}(\mu, \sigma^2)$
  for $k = 0$ to $K - 1$ do
      for $t = 1$ to $T$ do
          $u_t = \boldsymbol{u}_t + \epsilon_k^t$
          $s_{t+dt} \leftarrow F(s_t, u_t)$
      end for
      $S_k = \{s_t$ for $t$ in $[0, T]\}$
      $C_k \leftarrow \phi(S_k)$
  end for
  $\beta \leftarrow min_k[C_k]$

  $\eta \leftarrow \sum_{k=0}^{K-1} exp(-(C_k - \beta))$

  for $k = 0$ to $K - 1$ do
      $w_k \leftarrow \frac{1}{\eta} exp(-(C_k - \beta))$
  end for
  for $t = 1$ to $T$ do
      $\boldsymbol{u}_t = \boldsymbol{u}_t + \sum_{k=1}^{K} w_k \epsilon_k^t$
  end for
  return $U$
```

B. Model Identification

System identification is a central problem of every MPC implementation. This problem is usually solved using standard algorithms such as ARX [7]. Machine learning algorithms, that are used with a lot of success in vision, are also very powerful to achieve system modeling. By using neural network in a regression setting, it is possible to model with relative ease even non-linear systems.

One of the main problem of these learning methods for identification is that they are very dependent on the quantity and quality of the data used during the training of the model.

In our case, we want a model of the dynamic of the system, the f function in the equation (2). To be able to do so we need to gather data of the drone in as many situations as possible. In simulation, data can be gathered by recording the drone flying while sending the right command to generate any data needed. However, it is not always the case. Indeed there can be a lot of constraints in experiments, particularly when trying to solve real world problems. Battery charge being limited shorten the time available for data gathering. Security and regulation are also to be taken into consideration. Moreover the cost of crashing the drone may limit the range of actions available.

All these limitations have two main consequences for dataset generation. First the data is more scarce, second the action space (command send to the drone) is less explored. Exploration, especially when guided by an expert, can lead to unbalanced dataset. This means that some dynamic will be very well covered (moving in a given plane) while other will be rare (altitude variation for instance).

In [14], it is shown that a way to solve the problem is to retrain the model as the experiment is being conducted, training the model on a growing dataset of samples relevant to the task. However in that setting most of the information contained in the newer dataset are already samples that are correctly handled by the previous model. Going in straight line is the most common behavior for a drone and we keep spending time learning to do that, which is not very efficient.

The unbalanced dataset problem has been encountered in the Reinforcement Learning (RL) settings by the DQN algorithm [8]. While trying to solve the Atari game, it uses a buffer for training. However the information in that buffer is not evenly distributed among the samples. By prioritizing some samples over others, it was shown [12] that it is possible to accelerate the training, using less data while conserving the same performance. We propose to use a similar approach here.

To prioritize samples, a measure of their importance is needed. How much can be learned from a given sample is a rather hard question. In the RL settings the temporal difference error is what seems to be the most logical choice. In our context, we use the distance δ_i between the prediction of our model and the actual observation (of sample i), following the idea that our model as more to learn from samples where its prediction fails the most. We then use equation (3) to draw a new collection of sample from our original dataset by picking event i with probability $P(i)$.

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

The α hyper-parameter allows to soften the prioritization. Choosing $\alpha = 0$ gives the uniform distribution, in that case there is no prioritization at all. While a higher α value may encourage learning on edge cases.

One problem is that we are now trying to learn from a different distribution than the one we had before prioritization. We correct the bias induced by the prioritization by using importance-sampling [5] weights :

$$w_i = \left(\frac{1}{N}\frac{1}{P(i)}\right)^\beta \tag{4}$$

With $\beta = 1$ the prioritized sampling bias is completely corrected but it also slow down the learning. The α parameter increases the aggressiveness of the prioritization while the β parameter increase the correction thus, there is an equilibrium to find between both.

IV. Results

A. Experimental settings

Our test are conducted in a simulated environment using the gazebo[1] simulator. The UAV is simulated using the

[1] http://gazebosim.org/

Algorithm 2 Prioritized Sampling

```
Require: data, K number of trial and Task
  $trainingData \leftarrow data$
  $sampleWeight \leftarrow \emptyset$
  for $k = 0$ to $K$ do
    $F \leftarrow Train(trainingData)$
    $newdata \leftarrow Task(controller(F))$
    $data \leftarrow data \cup newdata$
    N number of sample in $data$
    for $i = 0$ to $N$ do
      $\delta_i \leftarrow \big\| Y_i - F(X_i, U_i) \big\|$
      $P(i) \leftarrow \frac{\delta_i^\alpha}{\sum_k \delta_k^\alpha}$
      $w_i \leftarrow \left(\frac{1}{N}\frac{1}{P(i)}\right)^\beta$
      $sampleWeighted \leftarrow \{w_i\}_{0 \le i \le N}$
      $trainingData \leftarrow$ sample data $d_i \sim P(i)$
    end for
  end for
```

tum_simulator[2] which provides the same interface as the real Parrot drone[3]. The implementation makes extensive use of the Robotic Operating System (ROS)[4] framework of which we used the kinetic version. Controllers run at 5 Hz as it is the rate at which the drone driver publishes the odometry.

The experiment are done on a Linux 4.13 using the Ubuntu distribution on an Intel i5-6200U CPU with 8GB of RAM in DDR3.

The system we are trying to identify is the drone using the low level Parrot drivers. Regarding data generation, three different approaches are taken.

First a simple Proportional Integral Derivative (PID) is used to fly the drone in a square defined by four destination points. This provides an easy way to generate data of the system in closed loop but does not explore all the dynamic of the drone as discussed in section IV-B2.

Secondly, in order to generate a dataset that better explores the action space, we design a dummy planner that goes in a random direction with random speed. For stability, constraints preventing the drone from crashing were included. The planner is designed for uniformly sampling the action space. Due to these constraints, the vertical dynamics might be a little more represented: to prevent crashing, the drone is asked to regain altitude if its current one is too low. This generate data in open loop as there is no feedback in this controller.

Lastly, we also collect data of the system in closed loop with the MPPI controller while it performs the task. The task consists in the following of a trajectory that goes around a cube in order to solicit dynamics on the three dimensions of space.

[2] http://http://wiki.ros.org/tum_simulator
[3]Copyright ©2016 Parrot Drones SA. All Rights Reserved
[4] http://www.ros.org/

B. Model identification

We are modeling the dynamic of the system. The input used are the velocity v_x, v_y, v_z, v_{rz} and command u_x, u_y, u_z, u_{rz} of the two previous timestep, t and $t - dt$ if we want to predict the $t + dt$ step. Here dt is the timestep. The output of the model F is the velocity at next timestep :

$$\begin{cases} v_x, v_y, v_z, v_{rz}(t-dt) \\ u_x, u_y, u_z, u_{rz}(t-dt) \\ v_x, v_y, v_z, v_{rz}(t) \\ u_x, u_y, u_z, u_{rz}(t) \end{cases} \xrightarrow{F} v_x, v_y, v_z, v_{rz}(t+dt) \quad (5)$$

1) ARX and Neural Network: For the ARX model, we use a second order model. The neural network used is composed of a succession of an input dense layer, two hidden dense layer and an output dense layer. The input layer is composed of 16 nodes corresponding to the input variable described in (5). The hidden layers have 32 nodes each and the output layer have 4, corresponding to the desired state output. Both hidden layer use Rectified Linear Unit (ReLU) activation. The output layer uses a linear activation. The loss used for the training is the mean square error, and the optimizer is ADAM [6].

That network is implemented using the Keras [2] python framework on top of the TensorFlow library [1].

First we compare our neural network and an ARX network. Both the ARX model and the neural network model are trained with the same data. Here, we use a balanced dataset for which all the dynamics are explored with similar frequencies. The dataset is obtained using the “dummy planner” previously mentioned. To evaluate their performance, we measure the error between their prediction and the actual observation.

In the figure 1, we see that they both perform very well on the vx dynamic. As this dynamic is linear this is expected. The average error on the test set for the neural network is $0.064\,\mathrm{m/s}$ for the velocity along the x axis. This is twice as good as the ARX performance that has an average error on the same dataset of $0.146\,\mathrm{m/s}$ for this axis. This is reasonable as such error in our condition would translate for the controller into a position error for one step of $3\,\mathrm{cm}$.

In the figure 2 we can see that for a more complex dynamics such as the vertical one in which the low level controller has to counteract the gravity, the gap between the neural network and the ARX model is more important. Along that axis the average error of the neural network is $0.045\,\mathrm{m/s}$ where it is $0.107\,\mathrm{m/s}$ for the ARX model. Notice that the range of speed for vz (between $-0.6\,\mathrm{m/s}$ to $0.6\,\mathrm{m/s}$) is more narrow than for vx (between $-1.5\,\mathrm{m/s}$ to $2\,\mathrm{m/s}$).

2) Neural Network with prioritized sampling: In the previous section, neural networks were shown to be able to perform a more precise model identification (in terms of one-step prediction error) than the ARX algorithm. To achieve this, a rich dataset of 60468 samples, equivalent to about 3 hours and 20 minutes of navigation data, was used. This data was generated in order to obtain a balanced dataset containing all the required dynamic. This implies important limitations for the applicability of neural network for system identification. To

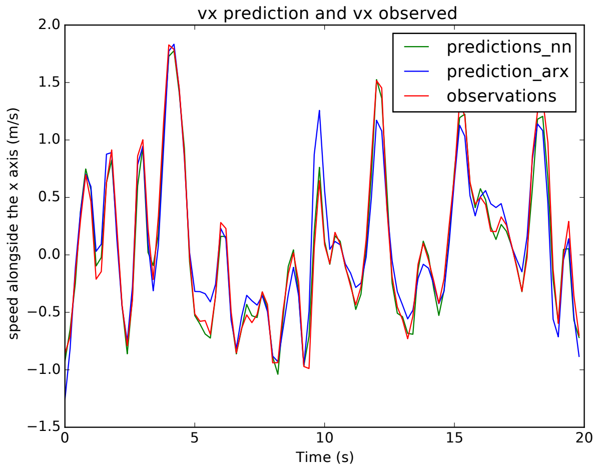

Figure 1. comparison of ARX and neural network model on linear dynamic.

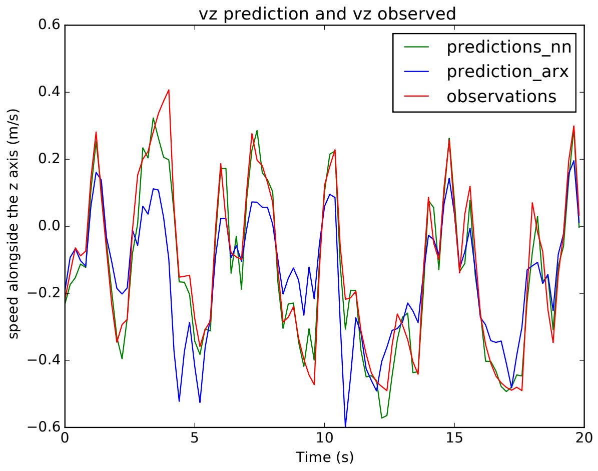

Figure 2. comparison of ARX and neural network model on non linear dynamic.

alleviate this burden we use the prioritized retraining method described in III-B.

To highlight the interest of prioritized sampling, we compare the different models in terms of multistep error. Each model is used to simulate the trajectory of the drone over 20 steps $(4\,\mathrm{s})$ while a constant command is applied. The applied speed command ($u_x = 0.5\,\mathrm{m/s}$, $u_y = 0.5\,\mathrm{m/s}$, $u_z = 0.1\,\mathrm{m/s}$, $u_{rz} = 1\,\mathrm{rad/s}$) is chosen such that all the dimensions of the dynamics are affected.

We first train the networks on a very unbalanced dataset generated with a PID which does not explore all the action space. Then we progressively add samples with more dynamic features. We construct the dataset such as the first couple of training exposes the network to only vx and vy commands and then progressively add rotation and vertical commands. The choice of hyperparameters is important here as it affects the performance. In this case we obtain the result with $\alpha = 0.6$ and

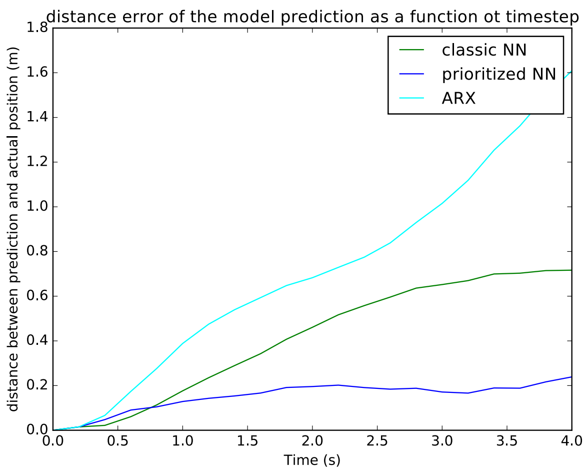

Figure 3. comparison of Neural network with and without prioritized replay after 8 successive training run in term of multistep prediction error

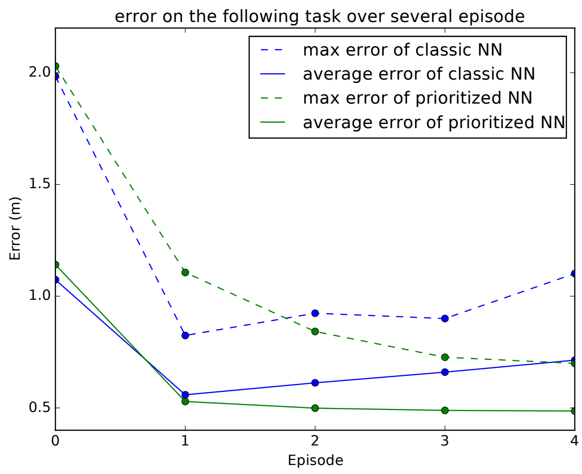

Figure 4. comparison of neural networks with or without prioritized replay in term of maximum and average error.

$\beta = 0.4$. We show that the neural network using prioritized sampling is able to faster use the new data and learn a better model in terms of multistep error in figure 3. The two networks are evaluated after having both been retrained eight times on the growing dataset. ARX is shown for comparison.

C. *Control*

To evaluate the quality of the models in term of control we use them as part of an MPPI controller. We evaluate the controller in a trajectory following task. The trajectory used is the following of the edges of a cube such that all dynamics (vx, vy and vz) are used. In our experiment the target moves at constant speed.

First, we consider the models we previously evaluated on multistep prediction and evaluate them. To this end, we use the distance between the targeted pose from the trajectory and the effective pose of the drone as the error. The result of this evaluation is shown in table I. Both neural networks perform better than the ARX model. It is important to note that for this experiment all the rate where set to $5\,\mathrm{Hz}$. It is possible to increase ARX performance by increasing the controller rate. It is harder to do so for the neural networks because this would require to change the hardware on which they run. Indeed while the computational cost of the ARX model prediction is negligible compare to the overall MPPI controller, the neural network forward propagation pushes the whole controller computational time just below the $200\,\mathrm{ms}$ that are available.

Table I
MAXIMUM AND AVERAGE ERROR

	ARX	*classical NN*	*prioritized NN*
max error [m]	1.789	0.882	0.819
average error [m]	1.272	0.562	0.508

One advantage of the neural networks is their capacity to keep improving as new data are available for them to train on. In order to compare the capability of the prioritized training to the normal training, we evaluate the controller on the control task over several episodes. Between each episode, the drone land and the network is trained again on a dataset combining the data used to train the model initially to witch we add the data collected during the previous episodes. The result of that experiment is depicted in figure 4. The neural network using resampling prioritization on the dataset is able to achieve better performance both in average and maximum error. As the dataset grows from the task, it also becomes more unbalanced as the task presents a lot of straight line and only occasional changes of direction. This has a negative influence on the classical training of the neural networks but not on the prioritized version.

The main advantage of using neural networks is there ability to model very complex systems. In the previous experiment, the controller is implemented on top of a low level driver. Thus, the system model needed by the MPC is almost linear. In order to test our implementation on a more challenging system, we add a suspended mass below our drone. This makes the dynamic of the drone much more complex. Our drone mass is $1.477\,\mathrm{kg}$ and we add a mass of $150\,\mathrm{g}$. The distance between the drone and the mass is constant and we constrain the mass to stay in a cone of $\frac{\pi}{3}\mathrm{rad}$ below the drone. In order for our system to be able to model the system, we increase the history it has access to. Until now only the two previous step were considered (i.e. $400\,\mathrm{ms}$ of history). In order to capture the effect of the mass we increase this history to ten steps (i.e. $2\,\mathrm{s}$ of history).

First, we compare an ARX model and a neural network model (without retraining) for the same task as previously. The neural networks performs better if we keep both controller running at the same rate.

To evaluate the interest of the prioritized retraining we use

Table II
MAXIMUM AND AVERAGE ERROR.

	ARX	*Neural Network*
max error [m]	2.250	1.536
average error [m]	1.220	0.946

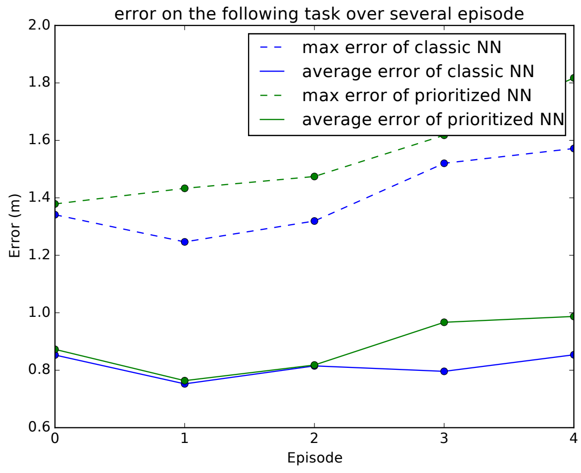

Figure 5. comparison of neural networks with or without prioritized replay in term of maximum and average error for a drone with uncertain load.

the same method as before, retraining the neural network between episodes. The result are shown in the figure 5. In this case none of the network training yield a better result. There might be several factor contributing to this. First, the dynamic being much more complex might require an improvement on the architecture of the network. Alternatively, for this experiment, the meta-parameters of the prioritized replay α and β have been kept from the previous settings but as the problem differs a new parameter search might be needed. Finally, the low level controller was calibrated for the normal drone and wasn't modified when the mass was added; it is possible that the interaction between the mass and the driver makes the dynamic much more stochastic than dynamic, which is not something system identification will handle easily.

V. Conclusions

We proposed a method to efficiently train neural networks for the purpose of system identification based on sample prioritization. We have compared the results of this modeling with the standard identification method ARX. We then tested those models on a drone in a simulated environment and discussed the limitations of the neural network based approach.

Studying the prioritizing meta-parameters is an interesting subject for future considerations as they have an influence on the results. However, an exhaustive grid search is too expensive to be practical. An automatic method for choosing these parameters would be of great value and might be helpful for improving the unbalanced drone problem.

Moreover, there is room for improvement in terms of neural network design. For example, in order to better take into account the state history, the use of Recurrent Neural Networks (RNN) might be interesting.

Another avenue of investigation about the unbalanced drone is to wonder about the deterministic nature of the problem. Indeed, the perturbation produced by the mass might make the problem stochastic, which would require a different approach.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] François Chollet et al. Keras. https://keras.io, 2015.

[3] Jan Dentler, Somasundar Kannan, Miguel Angel Olivares Mendez, and Holger Voos. A tracking error control approach for model predictive position control of a quadrotor with time varying reference. In *Robotics and Biomimetics (ROBIO), 2016 IEEE International Conference on*, pages 2051–2056. IEEE, 2016.

[4] Jemin Hwangbo, Inkyu Sa, Roland Siegwart, and Marco Hutter. Control of a quadrotor with reinforcement learning. *IEEE Robotics and Automation Letters*, 2(4):2096–2103, 2017.

[5] Angelos Katharopoulos and François Fleuret. Not all samples are created equal: Deep learning with importance sampling. *arXiv preprint arXiv:1803.00942*, 2018.

[6] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[7] Lennart Ljung et al. Theory for the user. *Prentice Hall*, 1987.

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[9] Mark W Mueller and Raffaello D'Andrea. A model predictive controller for quadrocopter state interception. In *Control Conference (ECC), 2013 European*, pages 1383–1389. IEEE, 2013.

[10] T. Naegeli, J. Alonso-Mora, A. Domahidi, D. Rus, and O. Hilliges. Real-time motion planning for aerial videography with dynamic obstacle avoidance and viewpoint optimization. *IEEE Robotics and Automation Letters*, 2(3):1696–1703, July 2017.

[11] Gabriele Pannocchia. Offset-free tracking mpc: A tutorial review and comparison of different formulations. In *Control Conference (ECC), 2015 European*, pages 527–532. IEEE, 2015.

[12] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[13] Grady Williams, Paul Drews, Brian Goldfain, James M Rehg, and Evangelos A Theodorou. Aggressive driving with model predictive path integral control. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 1433–1440. IEEE, 2016.

[14] Grady Williams, Nolan Wagener, Brian Goldfain, Paul Drews, James M Rehg, Byron Boots, and Evangelos A Theodorou. Information theoretic mpc for model-based reinforcement learning.

[15] T. Zhang, G. Kahn, S. Levine, and P. Abbeel. Learning Deep Control Policies for Autonomous Aerial Vehicles with MPC-Guided Policy Search. *ArXiv e-prints*, September 2015.