UNIVERSITY OF
CAMBRIDGE
Department of Computer
Science and Technology

Towards Sim-to-Real via Real-to-Sim: An end-to-end Automated Sampling and Dynamics modelling Framework for data-driven simulations

Ziyi Liu

Christ's College

May 2022

Submitted in partial fulfillment of the requirements for the
Computer Science Tripos, Part III

Total page count: 50

Main chapters (excluding front-matter, references and appendix): 39 pages (pp 7–45)

Main chapters word count: 11325

Methodology used to generate that word count:

[

```
$ make wordcount
gs -q -dSAFER -sDEVICE=txtwrite -o - \
   -dFirstPage=6 -dLastPage=11 report-submission.pdf | \
egrep '[A-Za-z]{3}' | wc -w
```

]

Abstract

Data-driven approaches for multi-robot control problems have proven very promising with a variety of model architectures, including deep reinforcement learning, GNN, and Gaussian Processes. So far, simulation data has been the main sampling source, where typically a large amount of training data is required. The model trained from the simulation, when deployed onto the real world (sim-to-real transfer), can lead to a significant reality gap due to the limitations of simulation assumptions, real-world noises, and robot-robot interactions. Alternatively, we propose that a direct mapping of current states and reference states to next states can be learnt as the dynamics simulator, such that the real action policies for different tasks can be learnt with RL methods. We also propose a general framework for such real-world data-driven models on single- and multi-robot systems with an automated and efficient data sampling mechanism.

Acknowledgements

Above all, I would like to express my sincerest gratitude to my supervisors Amanda and Jan, who have offered incredible support to me throughout the project. Every time I asked for suggestions, Jan was most patient and helpful, and I believe this project will appear much paler without him. Amanda always gives the most sound and inspiring advice and pointed to clear directions for me, and offered tremendous insight for this dissertation. It is indeed very fortunate of me to have both of them as my supervisors this year.

The project would also be impossible without the unconditional and unwavering support from my parents, who, though may not know what the project is about, are behind me the whole time no matter what I do. No words can express my gratitude to them enough.

I also want to give special thanks to the wonderful people in Prorok Lab, who are all so kind and friendly, and they are always there to help when I approach them. I felt like a part of the community because of them.

Having been in Christ's college for four years, I feel a strong sense of belongingness to it thanks to all the people around me, particularly my tutor Elena and my DoS Richard, who have been very supportive throughout these years.

Last but not least, I am thankful to all of my friends who made my life so much easier and more joyful this year, so that I can complete the project in a cheerful and bright spirit!

Contents

Chapter 1

Introduction

In a wide range of robotics tasks, gathering data, training policies and then transferring them into the real world (sim-to-real) has been a popular approach, and the reality gap [1] - the performance disparity between simulations and real world - as a result of this transfer has been handled with many approaches, such as domain adaptation and domain randomization. However, instead of relying on simplified simulation assumptions or devising extremely complicated analytical simulations for training action policies, we can directly train an end-to-end neural network that maps the current state to control parameters in deployment. This data-driven approach has shown great potential in many tasks for multi-robot systems, but each trained model is isolated and only specific to one task with poor generalisability. See more details on dynamics modelling in chapter 5.

Another two main considerations for the data-driven approach are the large size of data required and the sensitivity of performance to the quality and coverage of the data we feed into it. One intuitive method would be collecting samples that have not been covered by our training data so far independent of specific tasks. However, it is not pragmatic to sample the whole action space. An alternative view is to leverage active learning methods for continuous data sampling that minimizes model uncertainty via active information acquisition [2, 3] by strategically picking the most informative samples. Therefore, one key objective of this project is to explore the adaptive sampling mechanism for better and more efficient data collection, resulting in a data collection library as a deliverable. See more details on automated sampling in chapter 6.

The main objective of the project is to learn a model that maps the current state and desired action to the next state with neural networks [4], which serves as the simulator (real-to-sim transfer); and this trained model will then be used for learning the actual action policy. This approach follows the same intuition from VR-Goggles real-to-sim transfer [5], and much better action generations for the sim-to-real deployment should be expected. In the multi-robot case, GNN-based methods will also be employed for modelling robot-robot interactions [6]. We will first consider a single-robot system (RoboMas-

ter) with only robot-environment interactions, and then the problem can be extended to either a multi-RoboMaster or a single-drone system. Ideally, the trained model will be task-agnostic, such that specific tasks - such as static or dynamic targets, evader pursuit problems [7], narrow passage problem for a robot formation [6], and collision avoidance - would use a subset of dynamics that have been captured by the learnt model. The final outcome for the real-to-sim adaptation approach will be compared to the simple simulation results on different tasks as another performance indicator. The full exploration of sim-to-real transfer is elaborated in chapter 8.

Therefore, the main contributions of this paper can be summarised below:

1. Implementing a dynamics modelling framework with built-in model library and evaluation metrics
2. Building a library of automated sampling and integrating the online sampling with the existing ROS2 framework while implementing an offline active learning framework
3. Evaluating various models on a large dynamics dataset collected with the online sampling framework and demonstrating their capability in capturing real-robot behaviour
4. Performing RL training using modelled simulator on the multi-agent passage task with GNN-enabled inter-robot communications and transferring the RL policy to real-world scenario.

Once a general-purpose framework with dynamics modelling and active sampling has been fully implemented and tested, several extensions to this project can be further developed: (1) Generalizing the adaptation techniques to other robotic systems (such as a drone swarm) that involve significant robot-robot interactions(such as down-wash effects), (2) embedding physical knowledge priors into the dynamics models and comparing the hybrid modelling with our main framework [8] and (3) implementing a safe reinforcement learning approach, which concerns maximizing total reward within the safety constraints [9]. More challenges and future work can be found in chapter 9.

Chapter 2

Background

2.1 Nonlinear System Identification

In a perfect Newtonian dynamic system with ideal conditions, we can expect a linear response from the agent, which can be easily modelled. In fact, we can make this assumption in many scenarios by approximating the local steady dynamics as linear when the local increment is well-behaved. Therefore, we can regard this as the theoretical lower bound for our dynamics learner, such that the least expressive model can only extract the linear relationship.

However, in real-life robotic systems, non-linearity dominates the response domain, so that models of higher complexity with non-linear properties are required to learn the dynamics of the robot. Some common causes include the following:

- External noise in the environment, such as uneven ground, obstacles in the path, and blind spots for sensors, etc.
- Response lags to the reference state commands and stochasticity in controller behaviour
- Non-rigidity under high-velocity/high-frequency circumstances. For instance, at a critically high speed, the wheels of a robot could slip with respect to the ground, which can severely deviate the robot from the determined path.
- Inter-agent interference due to unwanted interactions. One prominent example is the down-wash effect seen in a swarm of quadrotors, such that when one quadrotor is directly and closely above another, the rotor-induced wind can effectively "push" down the bottom one, derailing it from the ideal dynamics.

In order to learn these effects and include them in the models, non-linear modelling methods have been proposed [10], and many of them are data-driven, where the models are trained empirically using velocity and/or position data collected from the real agent.

A brief overview can be found in section 3.1.

2.2 Uncertainty Estimation

In applied ML practices, making an inference from the trained model may not be sufficient for many scenarios, such as diagnostic radiography and other tasks that require very certain results. Therefore, having uncertainty estimates alongside the predictions is crucial under such circumstances. In general, the ML community divides the uncertainty into two main categories - aleatoric (data) uncertainty and epistemic (model) uncertainty [11]. Data uncertainty can be generally attributed to the randomness and variability in the experiment results, while model uncertainty refers to the uncertainty incurred by lack of knowledge of the model of the underlying information. Therefore, the full uncertainty of a predictive process is a simple sum of the two parts [12], where the data uncertainty is often irreducible while we aim at cutting the model uncertainty.

In robotic systems, both uncertainties are prevalent and non-negligible for applications due to both precision and security concerns. Taking an autonomous driving car as an example, in order to make the learned dynamics system robust to external disturbances, such as adversarial visual attacks or a sudden flash of light, on the one hand, we need to collect sufficient data covering a wide range of scenarios to reduce the data uncertainty as much as possible, and on the other hand, we also need to construct a model with high enough capacity and complexity to learn the road rules and dealing with accidents to push down the model uncertainty.

Methods of quantifying these uncertainties have been extensively studied due to their importance in making predictive inferences more reliable. In traditional ML practices, for instance, both Bayesian [13] and frequentist [14] approaches are developed and tested for Gaussian process (GP) models [15]. Many recent studies, however, focused on uncertainty estimation for DL models [12], and the two main categories are Bayesian techniques [16, 17] and ensemble techniques [18]. We will explore some of these approaches in the robotic dynamics learner context in chapter 6.

2.3 Active Information Acquisition

When placed in an unknown realm and asked to traverse the space, an agent can acquire information by stochastically moving around for a significantly long period with the expectation of capturing all of the important information. This approach, however, is not only vague in the procedure itself but also unclear about the ultimate goal, since we need to define more carefully what "important information" really entails. In response to the former, we can come up with a self-monitored method that involves a feedback loop to facilitate the information gathering process. Active information acquisition (AIA) [2] was

first proposed as a general-purpose framework for prediction tasks, comprising of two key components: task predictor and information selector.

One ubiquitous challenge for many robotic systems is the adaptability to novel environments and how robust the agent is when faced with border cases and unseen situations. When predictions are made, the uncertainty evaluation (see section 2.2) helps us understand in what subsets of the state space the model fails, so that more data can be collected from those ranges to boost the model performance. This procedure can be accomplished both offline and online, and the latter approach is much more flexible and efficient, where uncertainty estimation during training can directly help the robot collect the "useful" data.

The offline method, nevertheless, can also help us understand the model's weaknesses and analyze the collected data. In our implementation of the offline active learning, we also aim at training the models more efficiently - that is, using as little data as possible to achieve the same level of competence as that trained with the full dataset.

2.4 GNN for multi-agent RL

Graph Neural Networks (GNN) are powerful models for learning graph representations based on the graph structure and node information. Three essential types of GNNs - convolutional [19], attentional [20], and message-passing [21] - can be seen in almost all GNN layers, which characterise different mechanisms for aggregating information from neighboring nodes. A GNN layer can be interpreted as a permutation-equivariant function $\mathbf{F}(\mathbf{X}, \mathbf{A})$, and the general structure of a convolutional GNN can be formalised as

$$\mathbf{h_i} = \phi\big(\mathbf{x_i}, \oplus_{j \in \mathcal{N}_i} c_{i,j} \psi(\mathbf{x_j})\big), \tag{2.1}$$

where ϕ is the permutation-invariant propagation function, $\oplus$ some aggregation function, c_{ij} some normalization factor, and ψ a MLP function [22].

In this work, sim-to-real transfer demonstrations are performed by training RL policies with a ModGNN [23] framework, where a 2-hop 1-layer message passing GNN was used to aggregate data from the kth-hop neighborhood. The message aggregation module gathers local observations from each of the neighbors and compresses them before transmitting out again, and the transmission outputs from each hop are concatenated together as the node input. Subsequently, the node update module aggregates the node inputs and passes it through a 3-step multi-layer MLP that maps to the agent's action space as the node output. See fig. 2 and 3 in [23] for detailed illustration.

Chapter 3

Related work

3.1 General Dynamics Learning

As described in section 2.1, data-driven system identification can be most effective in a complex environment with a intricate control system. On top of straightforwardly applying deep learning models to randomly collected data for non-linear system identification, a variety of recent works have taken many more factors into account. In Gen2Real [24], general robot dynamics (GRD) was proposed in response to dynamics randomisation: instead of repeatedly sampling and modelling when a slightly different environment and robot configuration comes up, a large general model can be pre-trained on a massive dataset generated with different dynamics parameters, topology configurations, and model dimensions. A modified GPT model was used to learn GRD given its demonstrated power in tasks involving a series of coherent variables, such as natural language processing [25].

Several works explored non-parametric models, such as Gaussian Processes Regression (GPR) [8, 26, 27]. However, one common issue to GP-based methods is the dimensionality curse, such that the regression model becomes overly cumbersome as the computation cost grows linearly with the sample size. Gaussian Mixture Models (GMM), alternatively, maintains a small number of parameters and strikes a balance between modelling complexity and training data variations [28].

While the aforementioned works have assumed a black-box controller such that the internal mechanisms are completely unknown, KNode-MPC [29] proposed a knowledge-based NN for dynamics learning - effectively combining first-principle analytical modelling with data-driven NN modelling approach, where neural ordinary differential equations (NODE) [30] was used as the model. Intuitively, this mechanism assumes some prior knowledge of the controller by tapping into the controller dynamics, and the NN model predicts the error from this analytical output, giving rise to a final prediction.

Similarly, NeuralBEM [31] also assumes prior knowledge of the motor and rotor models

for quadrotors, but the learning-based NN models take in the past states as well to make the residual prediction. This breakaway from Markov assumption is also used in Gen2Real modelling [24].

3.2 Sim-to-Real Transfer

Although methods that train on real world data directly, such as imitation learning [32], can explicitly interact with the environment and acquire the real-world information, the shortcomings are also outstanding: (1) for behavioral cloning (BC) methods [33], they perform poorly as the environment becomes more complicated. Since the learning procedure relies solely on trajectories collected beforehand, it is most likely that expert demonstrations are not uniformly sampled across the state-action space. Thus, during inference, the BC model will not be able to generalize well in states unseen to the model in training, resulting in compounding errors. (2) For inverse reinforcement learning (IRL) methods [34], they tend not to scale well with comparatively large environments, as the RL-IRL loop quickly becomes very costly. Therefore, It is more practical to train RL policies in a simulation environment where data acquisition and rollout is much more efficient; but meanwhile, sim-to-real transfer would be required.

Sim-to-real transfer is a very broad concept referring to transferring models trained in simulation to the real world [35]. Under the assumption that underlying dynamics are fundamentally similar and the learned representations are still significantly aligned with those in the new domain, many transfer methods have been developed to counter the reality gap, including domain randomization [36], inverse dynamics learning [37], domain adaptation [38], etc.

In the Multi-robot coordination task detailed in [6], a sim-to-real gap evaluation framework was established for benchmarking the makespans (time taken for the last agent to reach the proximity of its goal within a given error) and success rates (the fraction of episodes where no collision with the wall or between agents occurred) for both sim and real, as well as visualising the trajectories and distributions of positions in both scenarios.

Chapter 4

Experimental Setup

The entire framework consists of three major divisions - sampling, modelling, and sim-to-real transferring. The sampling module not only constructs the fundamental infrastructure for running the whole robotic system and passing reference states and collecting robotic state data, but it also implements a framework that evaluates uncertainty given trained models as well as providing active sampling methods during data collection (see chapter 6). Given the collected data, the modelling module trains and evaluates a diverse collection of models with a set of metrics for different benchmarking purposes. By utilising the dynamics models as simulator environments, the sim-to-real framework then trains the RL policies and deploy them in real world. The full framework structure is shown in fig. 4.2.

4.1 Robotic System Setup

For most of our data collection and experiments, a DJI RoboMaster-S1 was used as the navigator in the real world for data collection, and it is the agent for multi-agent setups as well for evaluation. With the Freyja multi-rotor controller [39], we can obtain the velocity and position data from the `Current State` topic subscription estimated by 10 capture cameras, while motion commands can be passed into the `Reference State` topic. A simplified visualisation of the setup is shown in fig. 4.1.

ROS2 [40] system was used for all tasks involving real-world interactions as well as some simulation environments. The full implementation for this project can be found in this repository. Based on the infrastructure built for the multi-agent passage problem [6], data collection toolbox package was added into it. The `passage_simple_gnn` package contains three main methods for the multi-agent narrow passage task - RVO-only, centralised GNN policy, and decentralised GNN policy, and the centralised policy will be used in section 8.

To train the RL policies, the `rl_multi_agent_passage` repo was modified to enable an NN-based simulation environment, in which the linear velocity "incrementer" was replaced

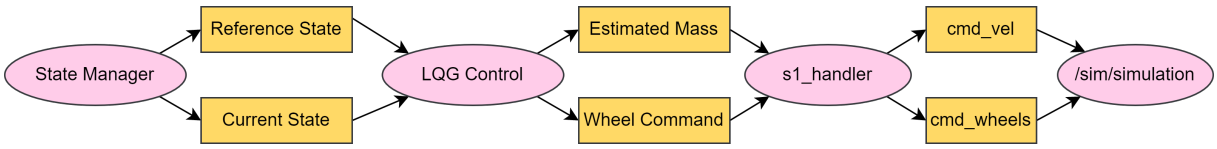


Figure 4.1: One-agent ros2 setup visualisation. Pink ellipses represent nodes, each of which is responsible for a single module purpose, and the orange rectangles are topics, which can be subscribed to for message exchanges, linking nodes together. All but the last node are under the workspace `robomaster_0`, associated with the single agent in the simulation. LQG control node takes in the relevant states and outputs the desired wheel command to reach the reference state, which are processed and passed into `cmd_wheel` topic for individual wheel motions.

by the data-driven model as the de-facto simulator.

4.2 Data Collection Pipeline

To ensure the robot security, (in the robomaster case, avoiding collision with walls), collision avoidance was performed for all sampling methods, which works according to the following procedure:

1. Check whether collision will occur (running outside the bounding box) with the reference velocity at the next timestep for each velocity component using the timestep length `dt`
2. Reverse each "will-collide" component velocity direction
3. if step 2 fails, reduce the concerned velocity by a fixed percentage repeatedly until the no-collision check passes.

In fact, in order to avoid looping in ROS2 programs, step 3 was only repeated at most twice in the actual data collection experiments, and as long as the maximum velocity is bounded by $3ms^{-1}$, the fixed perimeters will not be breached.

Having recorded Frejya messages from the `Reference State` and `Current State` topics, we parse them into a readable format with further processing. Finally, data alignment was performed to unify the frequency of the two different data sources, and to match each current state with its corresponding reference.

A similar pipeline was used for the `simple_simulator` data collection scheme as a baseline for the real-world data, where we simply passed and recorded `Twist` messages into `cmd_vel` and from `/sim/robomaster_0_state`.

4.3 Modelling and Sampling Framework

The integrated framework of dynamics modelling and data sampling is independently developed in this repository. For the modelling infrastructure, as shown in section 5, a

collection of models and training scripts are included, in which many tricks and model hyperparameters are invented. Evaluation, trajectory plotting, test metrics, and other auxiliary functions are all supported in the modelling framework. For sampling, two main active sampling methods and two active learning approaches are implemented, along with a number of proposed uncertainty metrics and visualisation tools. More details can be found in fig. 4.2.

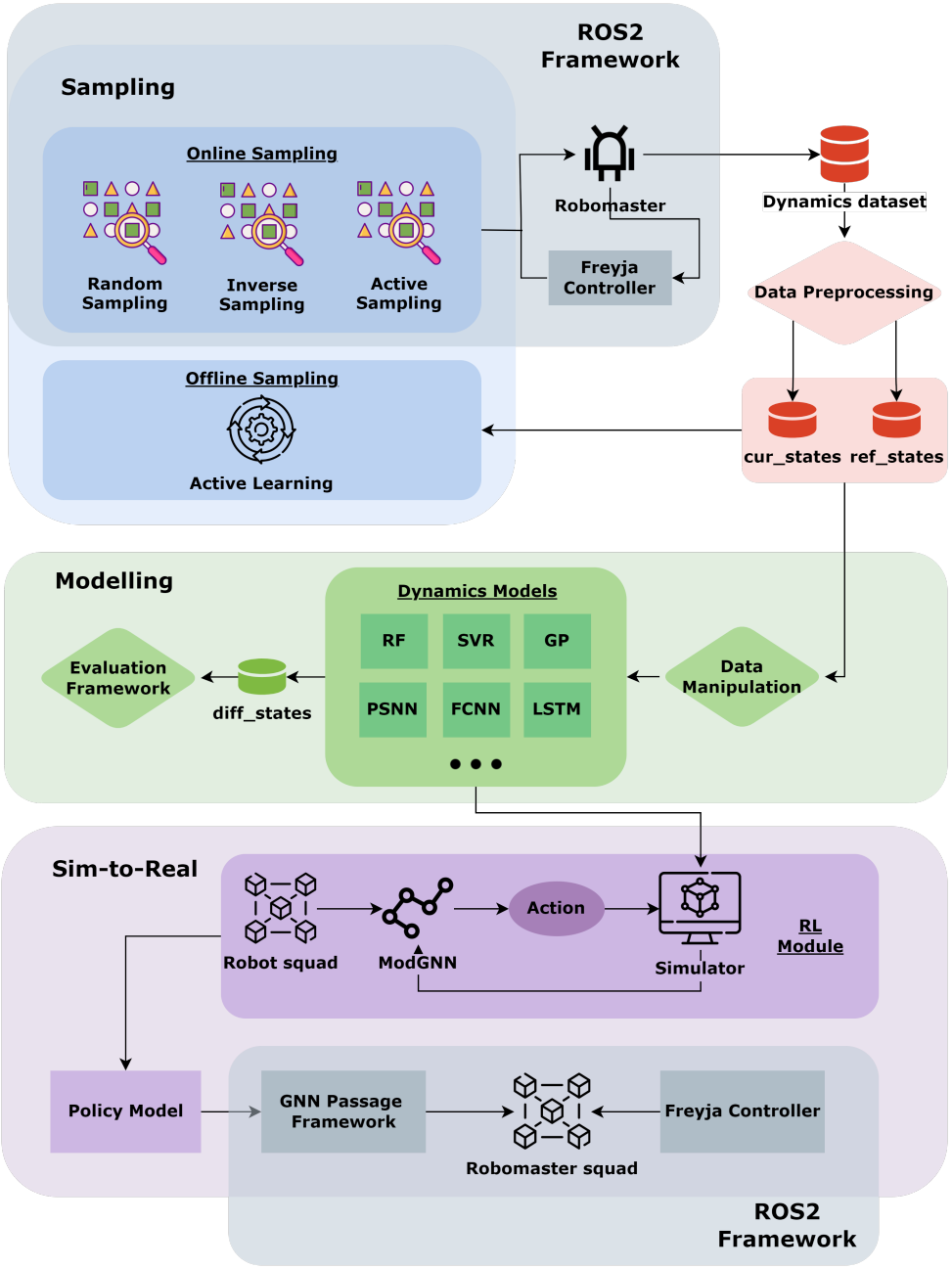


Figure 4.2: Diagram of the full framework implementation. Three main components - Sampling, Modelling, and Sim-to-real are shown together with their connection with each other. ROS2 integration with the sampling and sim-to-real components is also displayed.

Chapter 5

Sim-to-Real Modelling

In sim-to-real modelling, we built a framework that applies to an extensive collection of scenarios. Essentially, the differential state $\mathbf{v}_{diff,i}$ can be expressed as $\mathbf{v}_{diff,i} = f_\theta(\mathbf{v}_{cur,i}, \mathbf{v}_{ref,i})$, where $f_\theta : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$ denotes the dynamics model parameterised with θ, such that the next state is simply $\mathbf{v}_{cur,i+1} = \mathbf{v}_{cur,i} + \mathbf{v}_{diff,i}$. Given this relationship, numerous modelling methods are explicated in the upcoming sections.

5.1 Baseline Methods

For a regression problem, several commonly applied methods, such as `SVM` [41], `Gaussian Process (GP)` [15], `Random Forest` [42], etc, were integrated with our baseline training framework. One notable adjustment is scaling up the output dimension since each state consists of at least two variables. While `Random Forest` from `sklearn` has built-in support for multi-dimensional targets, external wrappers were required for other methods. For instance, `MultiOutputRegressor` was used for `SVM`, while independence between the two components in the velocity state was assumed for the GP regressor and N independent GP models were used for one-target regressions.

5.2 Neural Networks

As shown earlier, the lower bound of the relationship the model needs to learn is a simple linear relationship: $f(\mathbf{v}_{cur,i}, \mathbf{v}_{ref,i}) = \mathbf{v}_{ref,i} = \mathbf{v}_{cur,i+1}$. This can be easily learnt with a linear fully-connected neural network given a fixed system frequency, and this assumption was demonstrated empirically from a synthetic dataset collected with an ideal simulator (see section 7.2).

Nevertheless, since the ultimate goal is to emulate and reproduce the nonlinear controller behaviour reflected in the real-world dataset, an NN modelling framework with different architectures, non-linearities, and input/output processing methods was built and trained.

A list of the main approaches are listed below in table 5.1.

Method Name	**Explanation**	**Input Dim**
`SimplestPredictor`	minimum model with $2N^2$ weights	$2N$
`SimplePredictor Full`	vanilla model predicting full states at next timestep	$2N$
`SimplePredictor`	differential output (difference from the previous timestep)	$2N$
`PSNN-TK` (abbr. `PSNN`)	differential output with K past states visible	$(K+1) \times N$
`Stacked-Independent PSNN`	a set of independent models for each state component	$(K+1)$
`PSNN-TK Res`	replacing the differential target with residual difference from the reference state	$(K+1) \times N$
`Vanilla RNN`	past K states modelled temporally with a simple RNN	$(K+1) \times N$
`LSTM`	past K states modelled temporally with LSTM	$(K+1) \times N$
`Attentional RNN`	experimental implementation without full evaluations	$(K+1) \times N$

Table 5.1: List of main approaches in the NN-modelling framework with brief explanations for the model adjustments from the base. Input dimensions are included to indicate the amount of information each model takes in at each timestep, where N refers to the size of the state vector(e.g. 2 for robomaster, v_x and v_y).

Apart from some experiments used for comparison, all prediction targets are processed as increments from the current state instead of the next state itself, which can effectively reduce the model weight magnitude and render the model more robust.

5.2.1 Markov NN models

Theoretically, the next state $\mathbf{v}_{n+1}$ is only dependent on the immediate previous state $\mathbf{v}_t$, and therefore the Markov assumption $P(X_n = x_n \mid X_{n-1} = x_{n-1}, \ldots, X_0 = x_0) = P(X_n = x_n \mid X_{n-1} = x_{n-1})$. can be assumed. The family of such models, `SimplePredictor`, is trained, and the results are displayed in table 7.1.

5.2.2 lag factor δ_{lag}

One significant observation from the animations of the trajectories rollout is a short but considerable lag in the real agent behind the reference states. This statement can be further corroborated in fig. 5.1, where the real velocity tends to experience strong

oscillations upon a step-wise change of reference state without responding to it. To estimate the average time lag, the `current_states` array was shifted forward by a finite timestep $\delta_{lag} \in [0, 100]$, and the mean velocity difference was computed, shown in fig. 5.2. Consistent with the rough observation from the frame-by-frame estimate, the minimum overall difference occurs with $\delta_{lag} \sim 50$. Using this information, the training data can also be shifted to counter the lag effect.

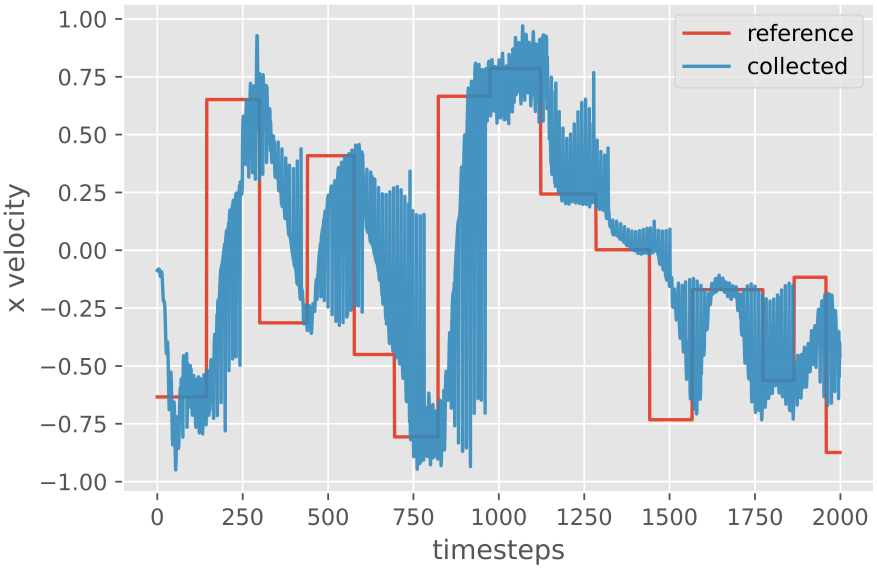


Figure 5.1: Timeseries plot of reference states and real-world states estimated with the Freyja framework. Strong noises are seen between state transitions, and significant lags can be observed of the real-world state behind the reference state.

In plot 5.3a, validation errors of shifting the current state forward by different intervals are shown. The general observation is that the larger the shift, the better the results. This demonstrates that adjusting the gap between the matching of reference state and current state against the lag direction can effectively minimize the gap in evaluation.

5.2.3 Past-states NN models

Although simplifying the system identification with Markov assumption to a tractable and memory-less problem has produced decent results for the robomaster data, the disparity from the real robot is still considerable, and the unexpected phenomena - such as skidding and erratic movements - cannot be modelled with only one current state; moreover, manually estimating the lag factor can bring in an additional sub-optimal hyperparameter. Therefore, in order to incorporate more information from the previous states, we make use of an MLP backbone and increase the input dimensionality.

However, the most fitting number of past states cannot be known a priori, as too few would be incapable of capturing the nonlinearities described above, and too many would

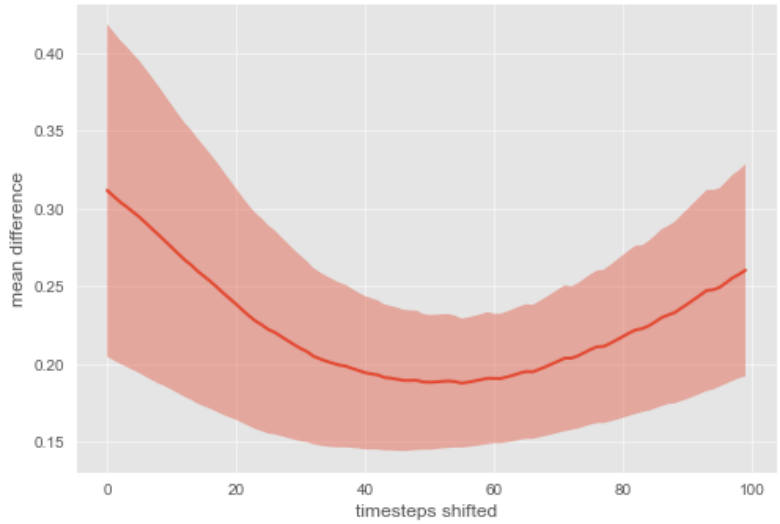
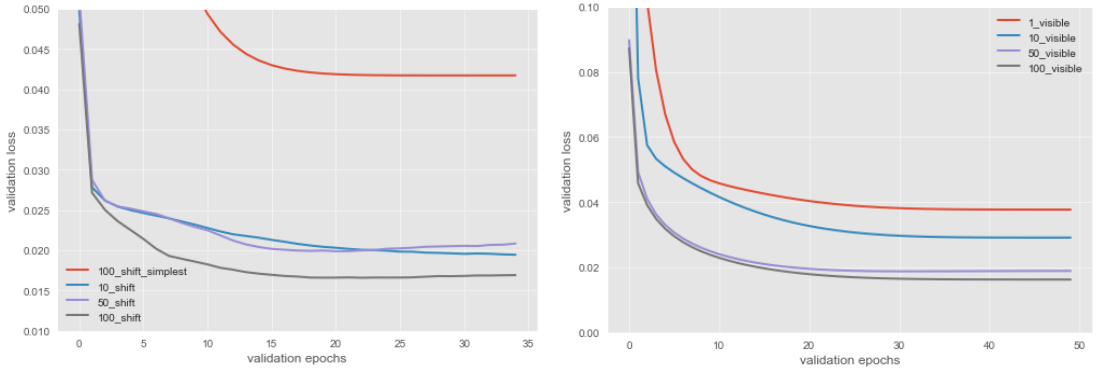


Figure 5.2: The mean difference between the shifted current states and reference states for shift step $\delta_{lag} \in [0, 100]$, with standard deviation displayed. The minimal difference can be found around $\delta_{lag} = 50$.

confuse the model and lead to potential overfitting. Our framework enables flexible tuning of the number of past states, and some training results are shown in fig. 5.3b.



(a) Time-step shifts

(b) visibility ranges

Figure 5.3: Validation MSE loss curves for PSNN and Markov-NN model with different shift steps and visibility ranges.

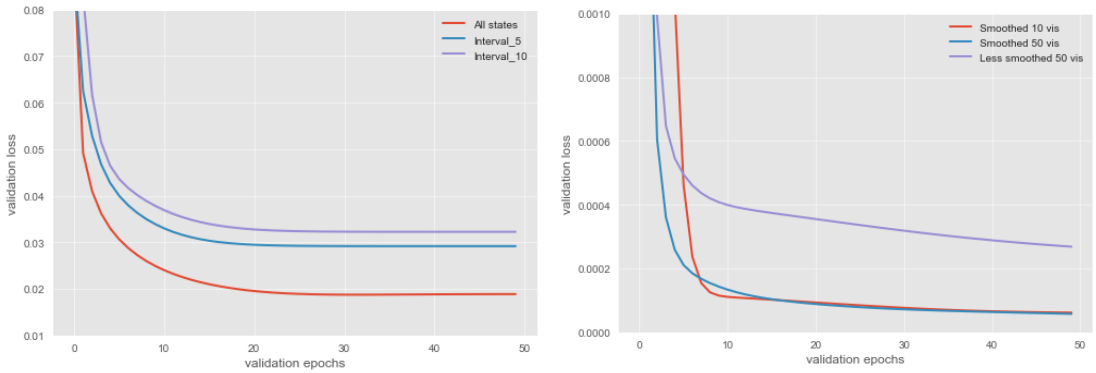
From the plot, 1-state was provided as the baseline benchmark, while we vary the number of past states visible to the PSNN model in the range of $[10, 100]$. A monotonic improvement trend can be observed as the amount of information increases with more states included, further illustrating that simplifying the problem with Markov assumption could hurt the model performance. Nonetheless, the improvement provided by 100-states modelling is very minimal compared with the 50-states, which signals that no new and/or useful information is added beyond a certain number of past states included. In fact, since it has been shown that the timestep lag is approximately 50, one explanation for the diminishing marginal improvement is that the lag and other significant real-world noises present in the nearer past are weighted much more in terms of the impact on the current prediction than those earlier.

5.2.4 RNN models

As shown in the last section, incorporating past information from previous timesteps can effectively enhance model performances, and since the nominal timestep is fixed, uni-directional RNN methods can be employed to leverage the information from previous states better. Both the vanilla RNN and LSTM implementations were integrated with the training framework. In our experiments, 3-layer RNN/LSTM with an additional projection layer and ReLU activation was used.

5.2.5 Data Gleaning

One pitfall identified from the `PSNN` modelling was that with the number of past states growing, it is not only more challenging for the model to learn the importance of the more recent states but also much more inefficient in training and evaluating. For the purpose of including more information from the past without compromising the model compactness, the data gleaning processing technique was invented and incorporated. For one specific data point, every k-th state from s_{t-L} to s_t is included, so that the total input dimensionality is reduced down to $\lceil \frac{L}{k} \rceil \times N$.



(a) Data gleaning

(b) Data smoothing

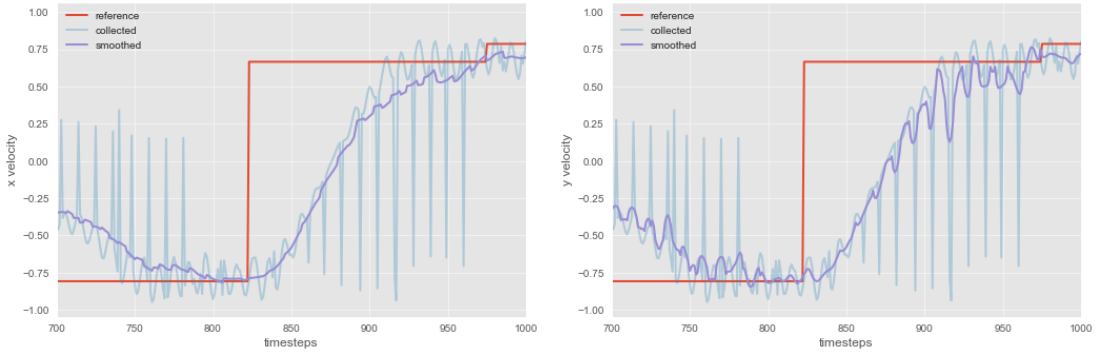
Figure 5.4: Validation MSE loss curves for PSNN and Markov-NN model with data gleaning and data smoothing with different gleaning intervals and visibility ranges.

With this approach in place, the models are tested with visibility scope of 50 past states but different sampling intervals. The baseline (red curve) trains on all visible states, while the others use the sub-sampling procedure. Notably, when more states are included, even though the same range is covered, the test results are better. However, this does not necessarily negate the utility of data gleaning, as it can substantially reduce the memory and training time needed without seriously compromising the model performance.

5.2.6 Data Smoothing

Zooming in on the visualisation seen on fig. 5.1, the timeseries plot of the collected data exhibits spikes of velocities periodically (see fig. 5.5a), which could adversely affect the

training due to many such outliers even though these spikes were not visibly observed during data collection. Therefore, a data smoothing method - Savitzky-Golay filter [43] - was introduced to the modelling framework, so that the drastic variations are reduced to the expected level, as indicated by the purple curve in fig. 5.5a. For a more realistic representation of the real agent, the SG-filter window length and polynomial order parameters can be tuned to include more oscillatory effects, shown in fig. 5.5b.



(a) window size=50, poly order=3 (b) window size=30, poly order=6

Figure 5.5: Illustration of severe periodic state disruption due to frequency mismatch in data collection at a high data collection frequency (∼200Hz). Data filters with different degree of smoothing were applied with curves overlaid on the original data.

The cause of the spikes were identified by comparing the raw `current_state` data with the post-processed states matched with the reference states. In fact, in the raw data, no such spikes were observed; however, when we sort the data on the timestamp in ascending order (which should not rearrange the data significantly), there were some data sequences embedded in the full time-series with a much lower frequency and often out-of-place, and replacing the points back to the correct spots gives rise to the sudden and periodic spikes. The solution above, data smoothing, essentially ignored these spikes at an expense of a very small deviation from the actual trend. Another solution we attempted was re-processing the dataset without re-ordering and adjusting the reference state matching.

From fig. 5.4b, we can eventually find the validation error at a much lower level than all previous experiments, with a factor of $\sim 10^{-2}$. Furthermore, the less smoothed data did not produce a better result as expected, which shows that the perturbations in the dataset can indeed adversely affect the learnability of the underlying task. Moreover, although both 10-visible and 50-visible models produced comparable results in the end, the latter converged much faster, illustrating the training efficiency due to more information.

5.2.7 Stacked Single-output NNs

Since the theoretic correlation between v_x and v_y of the system is negligible, as the two reference velocities are sampled independently (corroborated in fig. 5.6), it is possible to further reduce the number of input variables (confounding variables due to their lack of

information contribution) by only keeping the relevant component states for each model, and D (state dimensionality), while model backbone architectures are maintained.

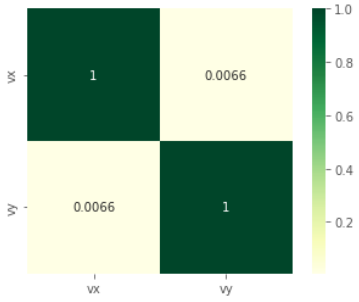
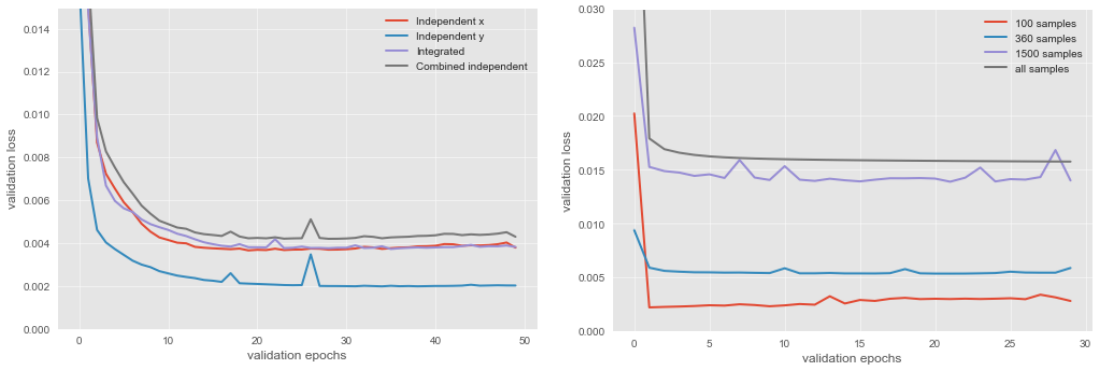


Figure 5.6: Correlation matrix of the 2D data collected and processed from the random collection scheme. The off-diagonal entries are very close to zero, effectively signifying the independence between the two components.



(a) Independent models

(b) Smaller datasets

Figure 5.7: Validation MSE loss curves for stacked single-output independent models (for x- and y-component predictions separately. A combined prediction loss is benchmarked against the full model) and training with less samples (for correctness demonstration).

Shown in fig. 5.7a, although the independently trained components performed about as well as the integrated model trained with all velocity components, the overall RMSE was slightly worse than its integrated counterpart, which in part validated that the two velocity components are not fully independent of each other.

As explored in previous sections, many trained models with the unsmoothed data led to unsatisfactory results for a longer-period evaluation. We then varied the amount of data to include to verify the model correctness. From the animation, it can be seen that the first reference state wasn't changed until $\sim$ 360 timesteps, so that this period was included along with other much smaller subsets. In fig. 5.7b, first k samples ($k \in \{100, 360, 1000,$ full-length$\}$) of the full trajectory were selected and trained. We can observe the monotonously decreasing validation loss as the number of samples lowers from the full set to only 100, as expected, proving the validity of the model.

Chapter 6

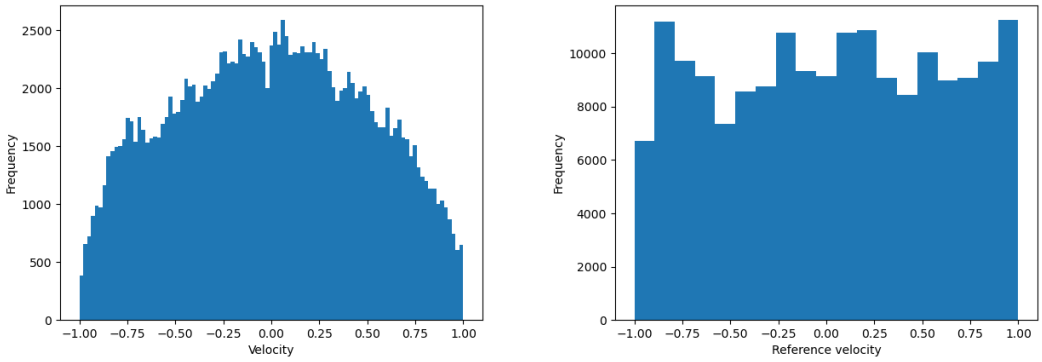
Sim-to-Real Sampling

Thus far, all training and evaluations have been performed on data collected with the simple random-walk approach described in section 4. However, there are several caveats that render this sampling approach both inefficient and unfulfilling. Firstly, data redundancy, in particular when the state space is small, could seriously burden the training procedure. Secondly, due to unexpected behaviour at high velocities, robot-wall collision avoidance script, and finite acceleration constraints, low-velocity states are much more common than those in the high-velocity range realistically. This data imbalance could hurt model performances at higher velocities. Finally, with a very large dataset collected at high frequency, training on the full dataset could be rather wasteful.

These issues, therefore, all prompt an adaptive sampling loop with feedbacks from collected data so that less represented, less certain, and more informative states should be sampled. Two distinct methods, offline active learning and online active sampling, were implemented and experimented with, which addressed the ineffectiveness of random-walk from different perspectives using uncertainty estimation during data collection. In addition, a fast and distribution-based method, inverse sampling, was also invented as a lightweight approach without repeated data uncertainty estimation.

6.1 Inverse sampling

Above all, distributions of states from random-walk are shown in fig. 6.1. As expected, states on both ends are less represented, while some bins at low velocities do not have good coverage either. One counter measure to this non-uniform distribution is computing a distribution with inverse "weights" for drawing reference states, where $p_{inv}(\mathbf{x}_{ref}) = \frac{1}{p(\mathbf{x}_{ref})}$, followed by normalization such that $\sum p_{inv} = 1$. Similarly, inverting the distribution by subtracting the current one from a certain level of uniform probability value can also achieve the same effect. One critical factor that could affect the inverse sampling efficacy is the bin count, which can be adjusted in accordance with the distribution shape. An

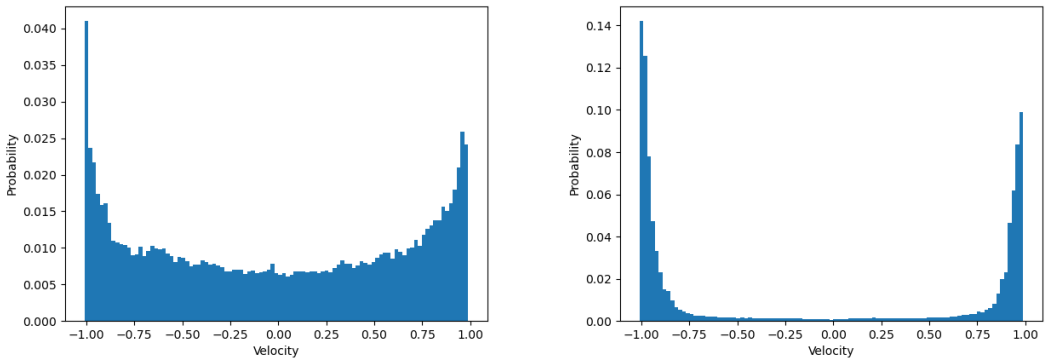


(a) current states $v_{cur,x}$ (b) reference states $v_{ref,x}$

(c) current states $v_{cur,y}$ (d) reference states $v_{ref,y}$

Figure 6.1: Histograms of current state and reference state distributions in both directions. While reference state random variables showed uniform distributions, the current state followed a tome-shaped distribution with a sharp decrease of state frequencies at higher-velocity range.

example of inverse sampling is shown in fig. 6.2



(a) upcoming $v_{ref,x}$ sampling distribution (b) upcoming $v_{ref,y}$ sampling distribution

Figure 6.2: Illustration of an inverse sampling example for a non-uniform current state distribution at a given timestep, where the two ends are given much higher weights than the middle to counter the non-uniformity.

For an online integration, inverse sampling can be invoked periodically to alter the weights for sampling reference states, while for offline learning, dataset pruning can replace active

sampling by setting the velocity state bin with the lowest frequency as the benchmark, and randomly discard extra samples from each bin until it reaches the same level as the benchmark, hard-correcting the dataset balance.

6.2 Uncertainty Quantification

Out of the uncertainty quantification methods outlined in section 2.2, MCDropout [16] and Ensemble [18] methods were implemented. The central idea shared by both methods is to produce a fixed number of predictions for each data point, and with all predictions, a heuristic is applied to estimate the uncertainty. Some metrics are discussed in section 6.2.1. MCDropout makes the stochastic predictions by randomly masking a subset of model weights with a dropout probability (by default p=0.2) during inference, while Ensemble method keeps the model architecture unchanged but model weights are randomly initiated before training, so that the inference results can still vary.

6.2.1 Active Acquisition Heuristics

Based on the two methods described above, one common heuristic for quantizing the uncertainty in regression tasks is computing the empirical variance of the ensemble of predictions, such that the model uncertainty can be simply estimated as $\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(pred_i - \overline{pred})^2$. There are, however, a number of heuristics that can be put to use for different purposes. As the original codebase [44] only implemented uncertainty heuristics for discrete values for classification, adaptions were made to work with continuous variables.

Shannon entropy, formulated as $\mathrm{H}(X) = -\sum_{i=1}^{n}\mathrm{P}(x_i)\log\mathrm{P}(x_i)$, is a measure of the level of information or uncertainty, and a higher entropy indicates more uncertainty in the dataset [45]. However, this metric was designed for discrete random variables, so that in our first experiments, most entropy estimates for the continuous regression outputs were either infinite or `NaN`. This problem was resolved with differential entropy [46], estimated with the Van Es method for a small sample size [47]. A closely related metric, mutual information [48], incorporating the entropy concept, concerns the mutual dependence between two (or more) random variables, which in turn determines how much information can be gained from one variable by observing another. Hence, data points with higher mutual information upon inference indicates a higher potential information gain for the model when we draw more observations from this neighborhood [49].

Finally, a random acquisition baseline, with which samples drawn with a uniform distribution, was provided to benchmark the performance of the aforementioned techniques.

6.3 Offline Active Learning (OFAL)

When a large dataset has been collected, an active learning procedure can be applied to iteratively select "most relevant" data points to enhance the model performance. The full procedure can be found in fig. 6.3, and summarised as below:

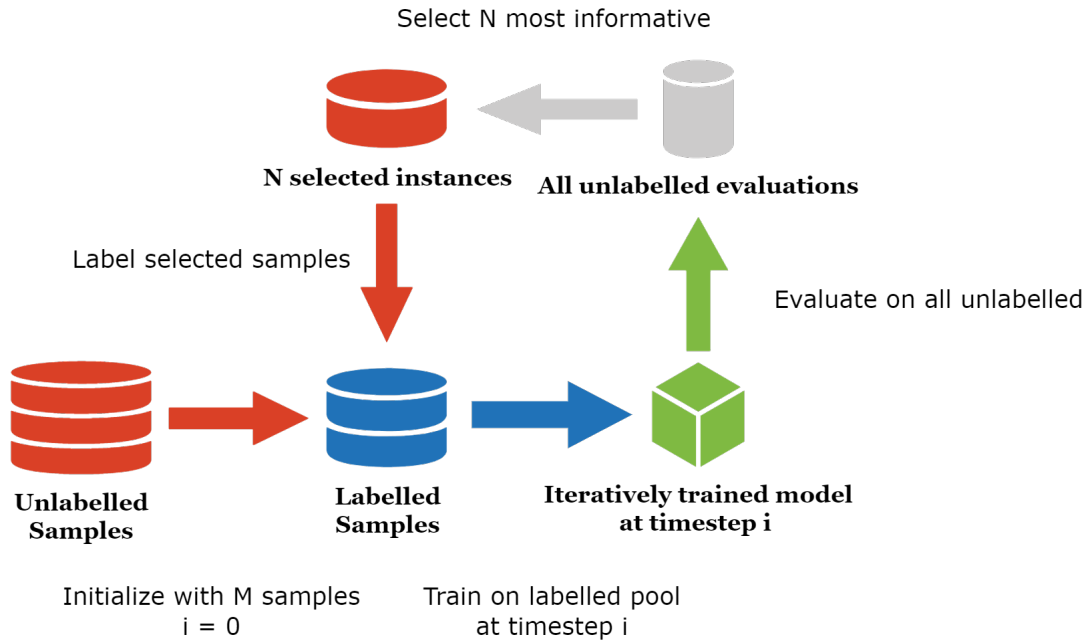


Figure 6.3: General offline active learning procedure illustrated. The sample pools and labelling processes are maintained with the `BAAL` library. In this example, only one model is used throughout, which corresponds to the MCDropout method.

1. Randomly label a small number of samples (M) in the unlabelled pool to initialize the procedure
2. For some heuristic h, uncertainties are calculated for all unlabelled data in the pool
3. Top n samples in the pool with highest uncertainties are selected and labelled
4. Repeat steps 2-3 until the pool is empty or a the stopping criterion has been met

The two main purposes of applying active learning - higher training efficiency and better model performance - are tested in the experiments shown on fig. 6.4 and 6.5. For the following experiments, a subset of 1000 samples are selected, and 700 of which are stored in the unlabelled pool. The training pool is initialised with 75 samples and 40 new samples are added to it from the unlabelled pool each round. The ensemble size is set as 5 for both uncertainty methods, where 5 models are initialised in the ensemble experiments while only one model is maintained but evaluated independently 5 times for MCDropout. To facilitate the model convergence given restricted training resources, model weights are carried over active steps (model weight resetting option can be turned on in the framework as well).

To analyze the step-wise progress of active learning, we evaluate the trained model on the

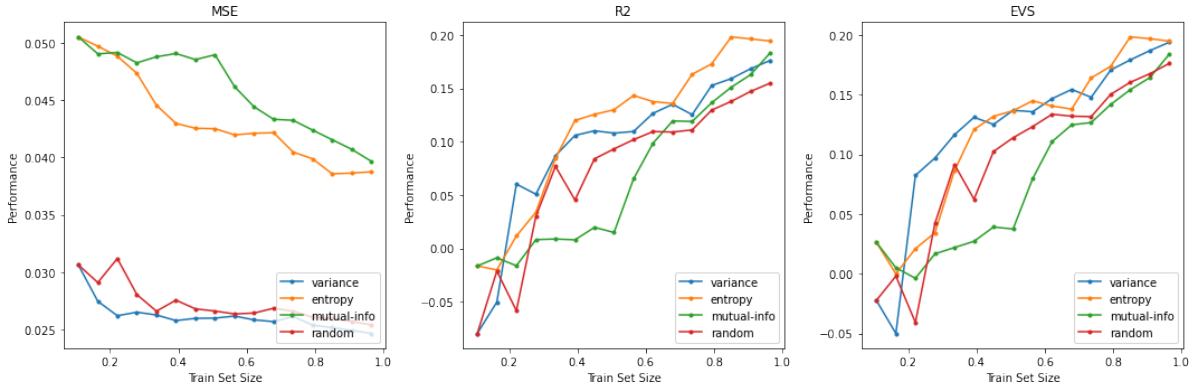


Figure 6.4: Active learning step-wise performance using PSNN model and Ensemble uncertainty method. Different uncertainty estimation metrics are benchmarked against the random selection baseline with different evaluation scores (MSE, R2, EVS, respectively) with a cumulative training sample pool.

randomly selected test dataset at the end of each iteration with various metrics. Since the original implementation was only used for classification, we added a few regression metrics for the dynamics learning task, such as MSE, R2, and Explained Variance Score(EVS). As shown in fig. 6.4, for the ensemble uncertainty estimation, three acquisition heuristics were benchmarked against the `Random` selection baseline. As expected, all active procedures are improving with increasing dataset size and more training iterations. In the MSE plot, we find the random baseline performing almost as well as the variance method, but the latter saw a much higher training efficiency in the early active steps, where the MSE dropped very fast. The other two approaches, however, initiated with a higher error and therefore did not reach the same level above, but rapidly decreasing trends are seen in both experiments, signifying their potential in producing much better results given more data and training. On the other hand, in the R-squared and EVS plots, all three methods exhibited a higher efficiency in early rounds and outperformed the random baseline by the end of the AL procedures.

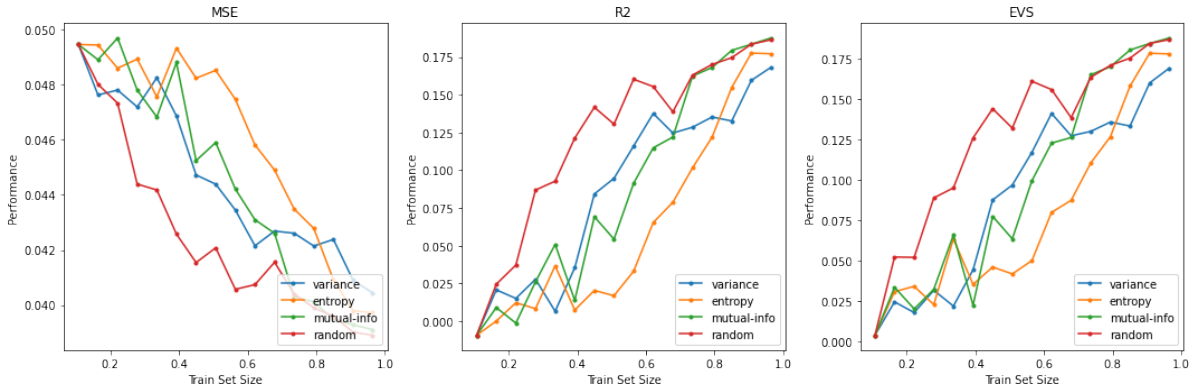


Figure 6.5: Active learning step-wise performance using PSNN model and MCDropout uncertainty method. Different uncertainty estimation metrics are benchmarked against the random selection baseline with different evaluation scores (MSE, R2, EVS, respectively) with a cumulative training sample pool.

The experiments with the MCDropout method are conducted while keeping all other factors constant. In fig. 6.5, we can see the random baseline performing at least as well as the other three methods in all regression metrics, with a consistently higher training efficiency. This can be partly attributed to the shortcoming of the dropout approach itself: as the model has not yet converged, applying dropout during evaluation can exert a strongly perturbing effect on the results without producing reliable uncertainty estimations.

One common challenge confronted by both methods is the accumulating active step time cost. Although the MCDropout method takes significantly shorter time as only one model is required, the later steps still take increasingly more resources to train due to the expanding training pool, and the full AL procedure scales with $O(n^2)$, where n is the size of the full dataset.

6.4 Online Active Sampling(OAS)

As a combination of online inverse sampling and offline active learning, the online active sampling (OAS) framework can serve a powerful tool for many robotic tasks that require interaction with the environment and data collection, primarily collecting data at an ad hoc basis. The invented procedure is detailed below:

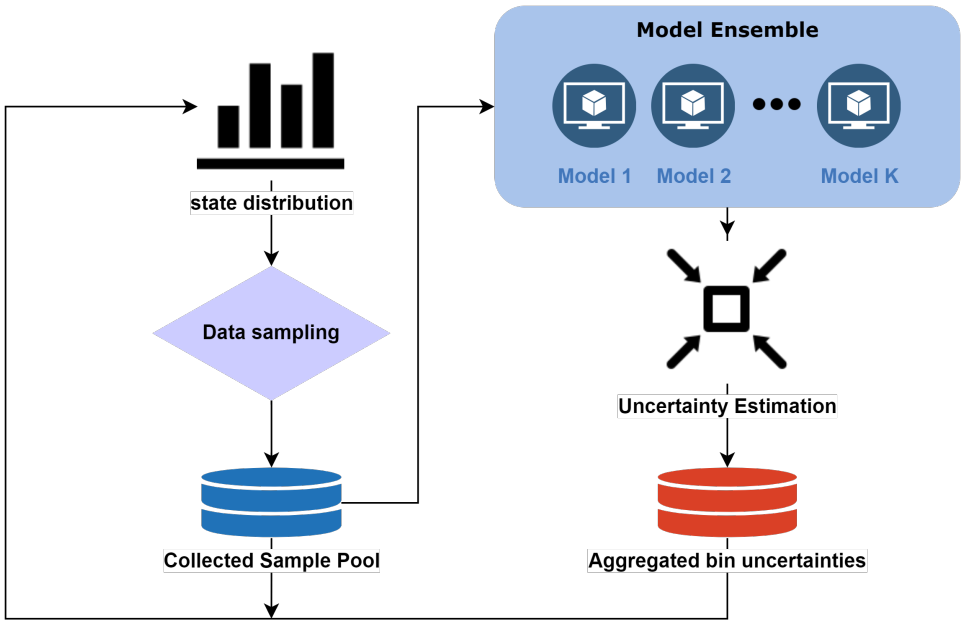


Figure 6.6: Diagram of OAS procedure with Ensemble method. The state distribution is initialised as uniform and dynamically updated in each iteration by applying higher weights to the more uncertain bins. A slight variation applies to the MCDropout method by replacing the model ensemble with only one model but producing a prediction ensemble instead.

1. Initialize the sampling distribution with a uniform distribution $\mathcal{U}$, based on which select random reference states, and initialise k simple dynamics learning models
2. Collect the real-world states estimated by the control framework on the run

3. Upon reaching the resampling threshold, train all k models separately

4. Evaluate the trained models on a set of synthetic data (where each bin has l randomly generated samples), and obtain an uncertainty estimate for each sample

5. Aggregate the uncertainties for each bin and produce a new state distribution based on the bin uncertainties

6. Repeat steps 1-5 until either (1) prompted to stop or (2) a stopping criterion is met.

Similar to uncertainty estimation methods applied in OFAL, an ensemble of fully independent models are trained, but the output for each velocity component is now two-dimensional: one for the expected value of the prediction μ and the other for the standard deviation σ. The learning of both variables with one model can be interpreted in a probabilistic approach. We can formulate our regression problem in a Gaussian expression

$$Y_i \sim \mathcal{N}(\mu_\theta(x_i), \sigma^2),$$

where μ_θ is the neural network parameterised by θ, and both θ and σ are to be learned.

In this paper [18], the ensemble prediction was further approximated with a gaussian, where the mean and variance are assumed to be those of the mixture model, where $\hat{\mu}(x) = M^{-1}\sum_m \mu_{\theta_m}(x)$ and $\hat{\sigma}^2(x) = M^{-1}\sum_m(\sigma^2_{\theta_m}(x) + \mu^2_{\theta_m}(x)) - \hat{\mu}^2(x)$. The log likelihood of the whole dataset [50] can be expressed as

$$\log Pr(y; \theta, \sigma) = -\frac{n}{2} log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \mu_\theta(x_i))^2,$$

and we need to minimize the negative loss likelihood over θ and σ. The whole procedure was implemented with the built-in support of negative log-likelihood loss and self-defined `predict_with_uncertainty` function.

By applying steps 3-5, we are effectively identifying the state ranges with higher uncertainty so that selecting instances from these ranges are most beneficial towards learning a good model. The model chosen for OAS does not necessarily align with the dynamics learner, as long as its capacity is high enough to approximate the system behaviour, so as to reduce the uncertainty estimation running overhead. A few issues stood out during the agent exploration phase. Firstly, as more states are collected, the cost of training increases at least linearly, which led to a state-passing congestion, and this increased queue suspension period in turn results in a longer `dt` than the collision avoidance script expected, causing robot collisions with the boundaries. Even with more aggressive avoidance mechanisms, the training overhead could still significantly disturb the state distribution. Secondly, the updated distribution cannot be effected timely after each acquisition step, as the choices made with the new distribution can only be roughly following it given a sufficiently long period of time, which is not plausible with the relatively high update

frequency.

Although uniformity is not necessarily the primary metric that determines the effectiveness of the sampling technique, it can still be used to evaluate how data distributions evolve with time quantitatively. The non-uniformity can be formalised as

$$non\text{-}uniformity = \frac{norm \times \sqrt{d} - 1}{\sqrt{d} - 1},$$

where $norm$ represents the L2-norm of the normalised distribution array, and d the dimension of the array. The non-uniformity scales between 0-1, where 0 corresponds to a purely uniform distribution and 1 the one-hot array. To visualise the change of uniformity, the value is computed for every k samples cumulatively, as shown in fig. 6.7. Log scale was applied to the uncertainties on y-axis to clearly illustrate the relative non-uniformity relations. Although the random baseline started with a lower value, the inverse method saw a faster dip, and as the number of samples increase, the inverse-sampling method saw a higher uniformity of the samples collected. Though the active-sampling method had a non-uniformity value more than twice as high as the other methods over 30000 samples, it is irrational to judge the validity based on this metric alone, as active sampling takes the data uncertainty into account, which is intrinsically not uniform due to uncertainty level disparity among different velocity ranges.

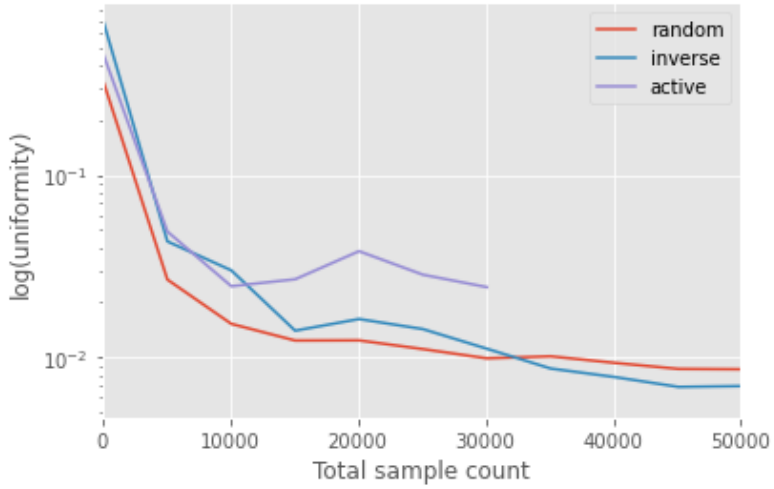


Figure 6.7: log-non-uniformity plot of different sampling strategies in progress with cumulative instances. The lower the value, the more uniform the overall distribution is.

Chapter 7

Evaluation

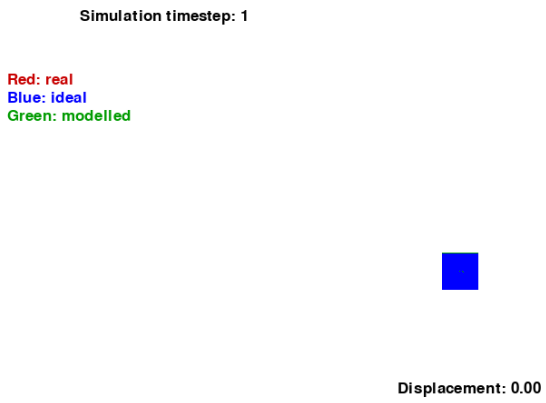
Although demonstrations of the models and sampling methods have been elaborated in the previous sections based on their empirical test performances and efficiency, it is most straightforward to visualise the learned model running through a full trajectory given the same sequence of reference states. Pygame was used to produce the trajectory animations. For easier comparison and intuitive understanding of how well the learned model could adhere to the true dynamics, a real agent and a reference agent are added to the animation as well, where both baseline agents have pre-recorded states. All three agents have the same starting position, and both the absolute and the relative positions are saved during the run.

As discussed earlier, the lower bound model performance should exhibit the same behaviour as the reference state agent - that is, the ideal dynamics. This would translate to a trajectory very close to that of the reference agent, which can be seen in the `SimplestNN` rollout.

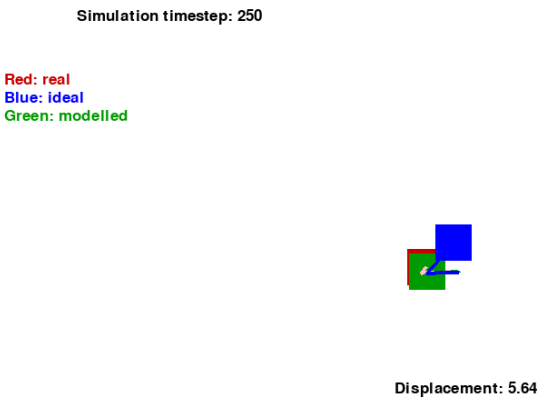
7.1 Evaluation metrics

On top of eyeballing the disparity in the rollout trajectories and to assess the quality of the learned model, a few metrics are devised for a quantitative analysis.

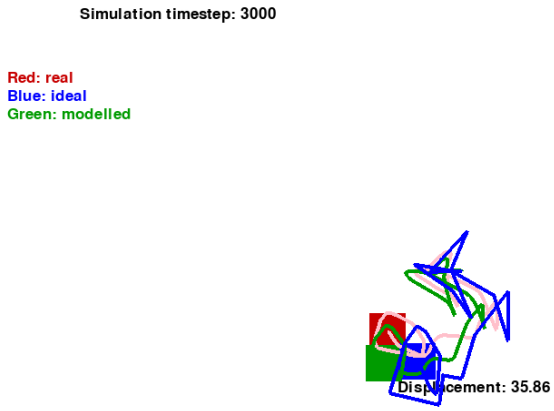
Displacement For rollout trajectories S_1 and S_2, the displacement between two trajectory points s_{1i} and s_{2i} at an instant t_i is simply the euclidean distance between them: $d(s_{1i}, s_{2i}) = \sqrt{\sum_k^D (s_{1ik} - s_{2ik})^2}$, where D is the number of dimensions, with default $D = 2$ in our following evaluations. An average displacement - simple mean of displacements across all timesteps - is also computed to illustrate how close two trajectories are intuitively. Notably, if the average displacement between real and modelled agents is less than that between modelled and ideal agent, then the learned dynamics model can at least replace the ideal simulator as a more realistic environment.



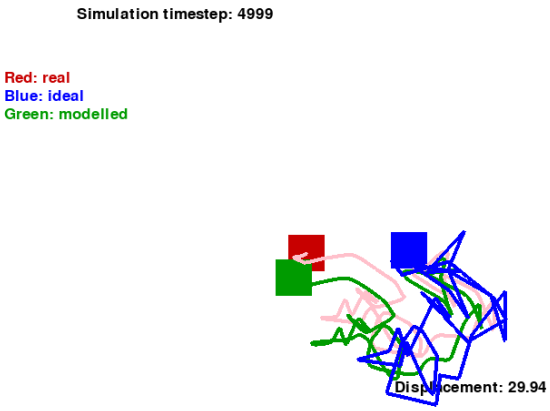
(a) Timestep: 1



(b) Timestep: 250



(c) Timestep: 3000



(d) Timestep: 4999

Figure 7.1: Animation of agents with random reference states at different timestep signatures for PSNN-10 model trained on smoothed rollout data. Three agents are shown simultaneously: blue the raw reference states representing the lower bound of learning, red the real trajectory mapped out by the agent, and green the one we learned, following the real-world agent closely in this scenario. Displacement between the real and learned agents is displayed on the screen.

Orientation Alignment Another vital aspect of correctly emulating a robot's behaviour is aligning the direction of movement at a certain timestep given the reference state. Cosine similarity, a normalised metric between -1 and 1, clearly indicates the degree of angle deviation of one robot from another (formalised as $S_C(\mathbf{v}_1, \mathbf{v}_2) := \cos(\theta) = \frac{\mathbf{v}_1 \cdot \mathbf{v}_2}{\|\mathbf{v}_1\| \|\mathbf{v}_2\|}$). Similar to the displacement metric, the alignment score can be averaged over the whole trajectory for global comparisons.

Propagated Error Since the model is agnostic of any information of the real robot other than the reference state it received, the error may accumulate with time. So as to keep track of the robustness of the model towards error accumulation, `propagated error` metric was invented to describe how quickly the modelled robot deviates from the real one, formalised as

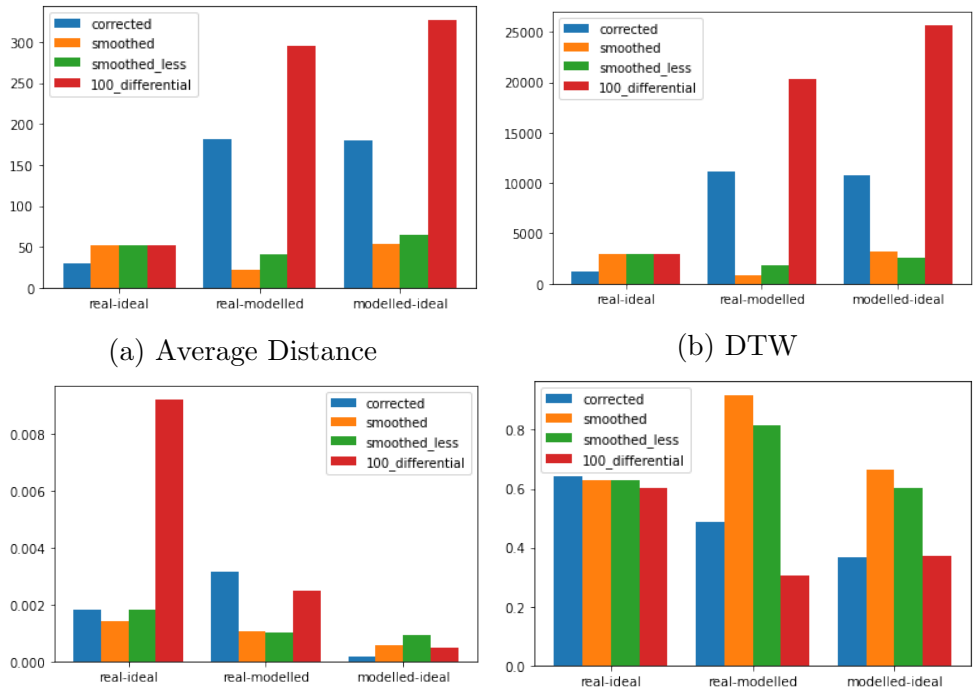
$$\frac{1}{N} \sum_{i=0}^{N-k} \frac{d(s_{1,i+k}, s_{2,i+k}) - d(s_{1,i}, s_{2,i})}{d(s_{1,i}, s_{2,i})},$$

where k is the interval parameter that specify the timestep difference between two points of interest. This effective position deviation percentage change can quantify the propagated difference averaged across time.

Dynamic Time Warping (DTW) distance DTW [51] is a sequence matching algorithm that looks for the minimal distance between two trajectories by attempting all possible alignments. In this framework, the implementation from [52] was adopted and applied to the rollout trajectories. Unlike previously explained metrics, DTW has a quadratic time complexity $O(n^2)$, such that some speed-up and pruning methods needed to be applied.

7.2 Evaluation Results

Before applying the evaluation metrics described above, trajectory rollouts were performed, where the initial state(s) and reference states are given, and the next state produced by the model would repeatedly replace or update the current state. It is expected that this approach would lead to growing divergence between different trajectories due to error accumulation, and the key results are shown in fig. 7.2 and 7.3:



(a) Average Distance

(b) DTW

(c) Prpagated Error

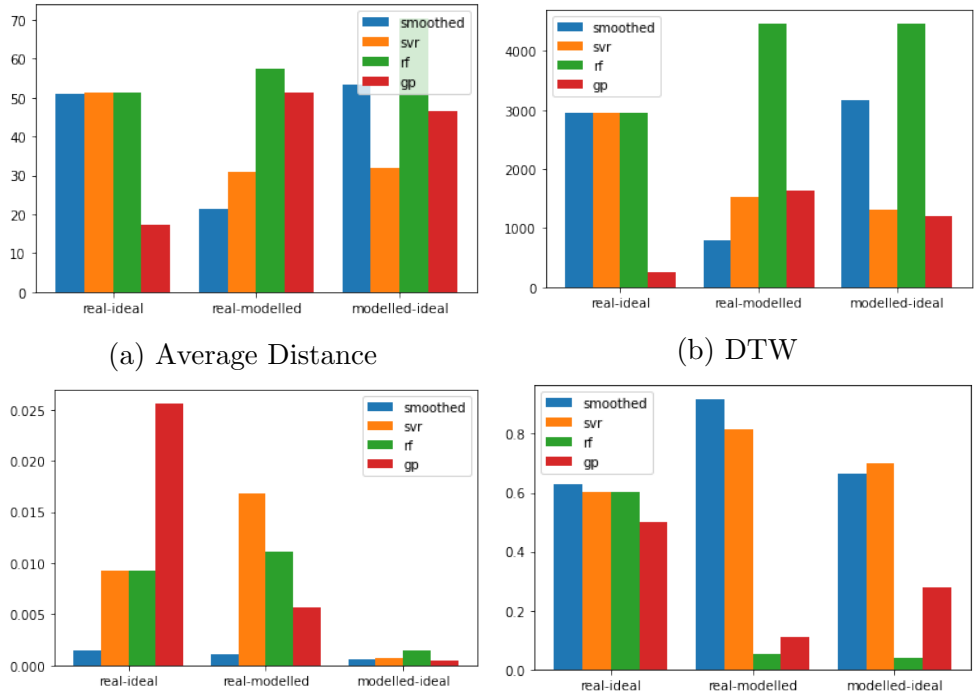
(d) Parallel alignment

Figure 7.2: Evaluation performance metrics between the three rollout trajectories in pairs for PSNN models trained with different data pre-processing tricks (frequency correction and data smoothing).

Above all, we can observe that the average distance metric (AD) results are highly consistent with those from DTW, so that AD will be used to analyze the proximity between

trajectories. In fig. 7.2a, only the two smoothed methods had a real-modelled AD lower than that of real-ideal, signifying that the modelled trajectory is, on average, closer to the real robot than the linear default. This is also reflected in all other evaluations as well. Particularly, we find that the parallel alignment for the real-modelled pair of smoothed model approaching 0.9, significantly higher than the 0.6 score obtained from the baseline real-ideal, which indicates a high level of direction synchronisation between the learned model and the real agent.

The "corrected" method shown in fig. 7.2 refers to the model trained on the naturally smooth but out-of-order data described in section 5. It can be validated that the frequency mismatch, though not visibly displaced, can be detrimental to the dynamics learning, as the distance and errors tend to be significantly higher than the smoothed counterparts, where the parallel alignment score is lower. The unsmoothed PSNN-100 baseline method was included, which exhibited a much worse performance.



(a) Average Distance

(b) DTW

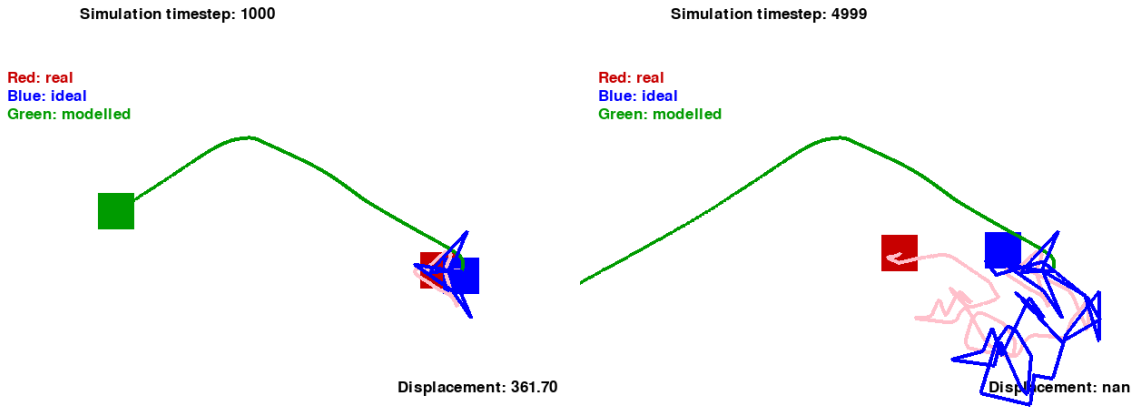
(c) Propagated Error

(d) Parallel alignment

Figure 7.3: Evaluation performance metrics between the three rollout trajectories in pairs for ML baseline models in contrast to the best-performing PSNN-smoothed model.

The ML baseline evaluation results are also shown in fig. 7.3. The smoothed method results were displayed to help compare the ML methods with the best of NN-modelling. Firstly, one general observation is that the smoothed-NN approach outperforms all chosen classic ML methods on numerous metrics, as seen from the real-modelled measures. Secondly, it is almost always the case that $SVR > GP > RF$ in terms of model performance, and the real-modelled metrics of ML rollouts are not significantly superior over the modelled-ideal pair, suggesting a mediocre performance that lies between the lower bound and the real agent. Thirdly, the GP method shows a different level of real-ideal

values from the rest of the ML methods due to the issue that GP models are difficult to scale with a large dataset, so that in order to reduce the rollout cost, only 1000 steps were executed in the trajectory.



(a) Timestep: 1000 (b) Timestep: 4999

Figure 7.4: Rollout animations example for exceptional cases in which the prediction error builds up till `NaN` and the regular metrics can not apply. The excerpts are taken from the evaluation trajectories of vanilla PSNN-20 model.

	Real-Modelled				Ideal-Modelled			
Models	AD	DTW	PE [1]	PA	AD	DTW	PE	PA
V-PSNN-10	300.73	17974	**1.0**	−0.021	307.69	17420	0.36	−0.077
V-PSNN-50	2.94e6	1.73e9	**1.0**	−0.051	2.94e6	1.73e9	0.13	−0.15
V-PSNN-100	295	20347	2.4	0.30	327.31	25686	0.49	0.37
S-PSNN-10	**21.38**	**797**	**1.0**	**0.92**	53.27	3162	0.56	0.67
less-S-PSNN-10	40.09	1872	1.0	0.81	63.80	2568	0.93	0.60
C-PSNN-10	182.01	11137	3.2	0.49	179.09	10823	0.16	0.37
S-RNN-10	216.89	17004	2.3	0.60	200.13	15550	0.36	0.96
S-LSTM-10	272.08	20107	9.2	0.38	268.93	19917	0.33	0.020
S-SP-10	28.84	1557	21	0.84	30.41	1193	0.64	0.66
S-SP-50	28.24	1571	22	0.71	30.15	1335	1.0	0.92
SP-10	31.28	1433	15	0.81	35.56	1529	0.60	0.70
SP-shift-40	33.10	1927	16	0.69	27.96	1394	1.3	0.91
SVR	**30.86**	**1520**	17	**0.82**	31.92	1315.63	0.66	0.70
RF	57.5	4457	11	0.051	70.40	4469	1.4	0.037
GP	51.1	1627.4	**5.6**	0.11	46.55	1209	0.53	0.28

Table 7.1: Evaluation results on rollout trajectory for representative models outlined in section 5. The real-modelled metrics demonstrates the learning power of the dynamics model, compared to the lower bound of ideal-modelled metrics. To clarify on the model acronyms, the numerical suffix, if any, represents the number of past states visible to the model at any training step. An 'S' prefix refers to the model trained on smoothed dataset, 'V' for vanilla, and 'C' for corrected.

[1] The values displayed are x1000 magnified

Many other evaluation experiments were performed and some metric results for benchmarking different approaches are given in table 7.1. For some models with high complexity, such as PSNN with a large visibility range and more hidden layers, divergence of the modelled agent from the current system has been observed. One example can be found in `PSNN-20`, where by the end of the rollout, the real-modelled displacement has diverged to `NaN`, shown in fig. 7.4. Since this behaviour did not appear in the smoothed-data approach under otherwise identical conditions, we can further confirm that the spikes due to irregular data in the original data set are critically harmful to dynamics modelling. And as the complex model tends to overfit to the current velocity range without seeing higher velocities during training, the accumulated error could grow exponentially. In such cases, the quantitative metrics no longer apply, and these models are deemed futile.

Chapter 8

Real2Sim Transfer

The ultimate goal of the full modelling and sampling framework described above is to provide a gateway towards a most realistic representation of the real-robot behaviour so that the simple linear simulator can be replaced by the more complex and expressive learned model. We choose a multi-agent passage task in this work, the default environment of which entails N robomasters and one narrow passage that allows one robomaster through at one time, and the task is to coordinate all robomasters, either through centralised or decentralised policies, to move across the passage and reach their respective destination as fast as possible without colliding into the barrier or each other. The implementation details of this environment can be found in this codebase.

Replacing the existing simulator that takes the reference velocity as the next velocity and computes the position change, it is expected that the new simulator will provide essential information about the real-robot dynamics as well as external environment so that when a policy is trained, optimizations will be made with respect to this refined dynamics system. All other setups and training configurations were preserved from [6], and training was performed on a GPU, as CPU training was no longer feasible.

8.1 Training

In order to achieve the objective of the coordination-passage task described above, the original setup outlined a reward function $r^i_{total} = r^i_d + r^i_c + r^i_g$ at a known timestep t. r^i_d, the guidance reward, can be written as

$$r^i_d = \left(\frac{\mathbf{d}^i}{||\mathbf{d}^i||} \cdot \frac{\mathbf{v}^i_m}{||\mathbf{v}^i_m||} \right) ||\mathbf{v}^i_m||,$$

where $\mathbf{d}^i$ is the vector pointing towards the next waypoint and $\mathbf{v}_m^i$ the current measured velocity, while the collision reward, r_c^i, states that

$$r_c^i = \begin{cases} -1.5 & \text{if collide with another agent} \\ -0.25 & \text{if collide with obstacle} \\ 0 & \text{otherwise} \end{cases}$$

and the goal-reached reward r_g^i is defined as

$$r_g^i = \begin{cases} 10 & \text{if reaches its own goal} \\ 0 & \text{otherwise} \end{cases}$$

For our experiments, most training hyperparameters were kept as default, while we conducted with either 2 or 5 agents (2 agents for faster training convergence and relatively more straightforward evaluation). Also, since the dynamics simulation models are trained with a implicit time interval 0f 0.005s, we modified `dt` parameter to align with the simulation model, and trained both baseline policies using simple additions for velocity states and various modelled policies. The maximum number of policy learning iterations was also varied to suit the update rate `dt`. For each training episode, the agent locations and goal locations are initialised randomly while maintaining their relative positions. One example setup with five agents can be seen in the RViz2 visualisation on fig. 8.3a, along with the real-world setup on fig. 8.3b.

To illustrate the training process, we plotted the mean training rewards in progress to demonstrate the viability of the GNN method in RL with both ideal-simulation and modelled-simulation methods. For the latter, PSNN-10-visible was used, where all elements in the past-state vector are initialised to be 0. In fig. 8.1, we first observe that with the default `dt`=0.01s, both simulators had very similar performances by the end of the training, certifying the correctness of the psnn-10 model under this context. However, having adjusted `dt` to 0.005s, both simulators exhibited significantly higher performances, where the baseline model generates a mean reward about twice as high as the psnn-10 model. This "boost" is seen due to a lower chance of colliding into each other and barriers as the movement is a lot slower while the episode length is fixed, resulting in an effectively shorter training episode, where the models did not necessarily reach the barrier at all. From the animations, a lag behind the reference states (actions produced by the trained policy) in the PSNN-10 simulator was clearly visible. This could potentially incur more clashes and penalties.

However, worse training rewards with learned model simulators are not necessarily reflected in real-world evaluation results; on the contrary, it should be expected that PSNN-10 would have a worse simulation rollout performance, since the evaluations are not done in the real-world, and the model would be over-correcting the non-existent lags and noises.

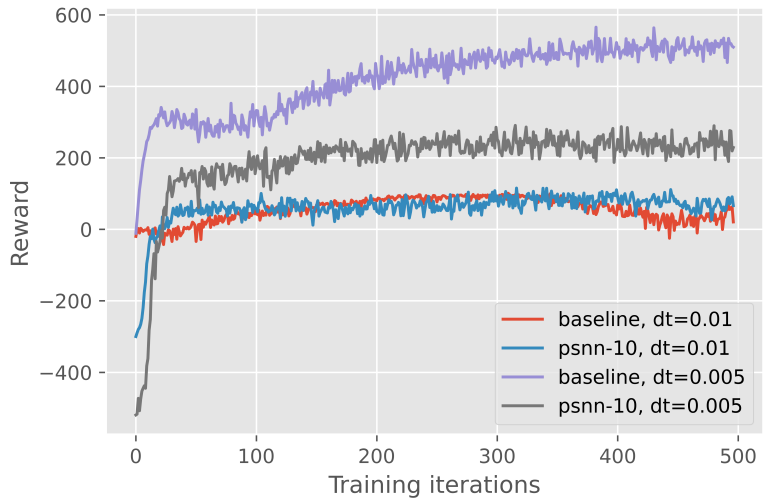


Figure 8.1: RL reward evolution within 500 iterations for PSNN-10 model and simple-simulator baseline implementation with different update rates dt=0.01 and dt=0.005.

Subsequently, various dynamics models were experimented with, and the most representative examples are shown in fig. 8.2. The untrained PSNN model was included as the random baseline, and the trained PSNN, in contrast, produced consistently better rewards. The simplest linear model with only $4 \times (K + 1)$ weights (K being the number of visible past states) pegged onto the ideal simulation baseline as expected.

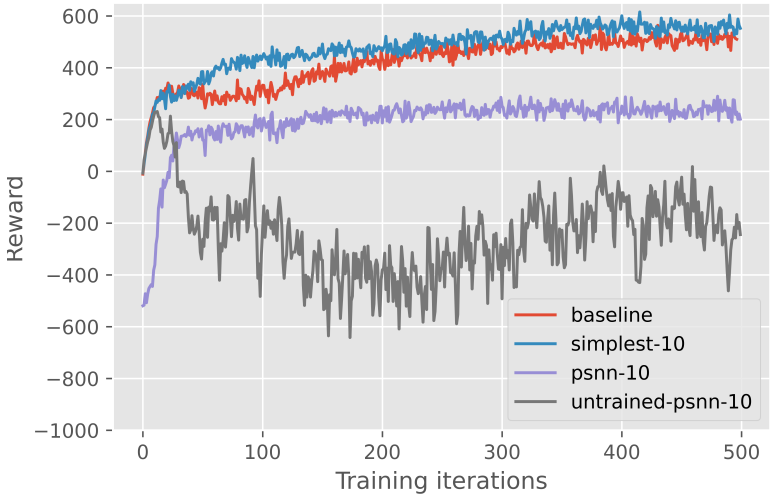
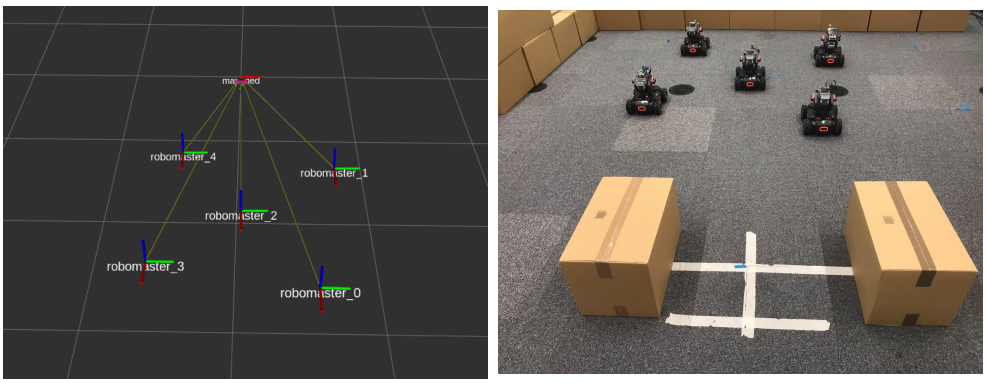


Figure 8.2: RL reward evolution within 500 iterations for representative models seen in chapter 5 benchmarked on the simple-simulator baseline implementation.

In order to draw easier comparisons with the original experiments and potentially expedite the training convergence, more policies are trained with the full 5-agent setup. More results are shown in table 8.1.

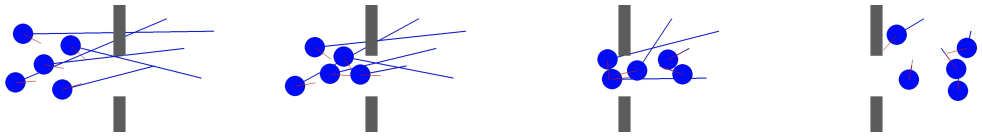


(a) (b)

Figure 8.3: 5-agent starting configuration in a cross-shape with a narrow passage marked by two boxes. (a) real-time monitor of the relative positions of the five agents using RViz2. (b) real-life photo of the 5 robots in their starting positions.

8.2 Learned Policy Evaluation

Upon obtaining working policies trained with finetuned parameters and simulation dynamics models, we evaluated the learned policy first in the simulation space. In this framework, animation clips are produced to visualise the evaluation episodes for each independently initialised environment, one of which is shown in fig. 8.4.



(a) (b) (c) (d)

Figure 8.4: Trained policy evaluation rollout on 5 agents. Each blue circle represents an agent, with the extending blue line indicating the goal vector, and the red stick pointing in the direction of reward vector. (a)-(d) illustrates a time-series evolution for the SimplePredictor simulator trained for 1000 iterations.

In this particular episode, although the objective of the experiment can be met, where each of the agent navigates to their respective post, it is sub-optimal in terms of both the longer makespan and the larger number of collisions.

Finally, we exported the learned policies and transferred them to the real-world setting. However, unlike the baseline experiments shown in the previous works, even though the actions generated by the policies led to intuitively plausible results, where the agents eventually passed through to the other side of the passage with few collisions, both the real-life makespan and the fluctuations in the actions are markedly higher than the baseline. As a result, it is infeasible to compare the performances quantitatively with the previous metrics, yet there are several qualitative takeaways from the current experimental results: (1) The data-driven simulations demonstrated the potential of replacing the simple simulator with a non-linear model and still learning the task to some extent. (2) The disturbances observed in the simulations are greatly diminished when deployed to

	Model	Max Len/dt	Reward		Episode Length	
			Mean	STD	Mean	STD
5 Agents	SPSNN-10	500/0.05	-491.7	273.0	500	0
	SPSNN-50	500/0.05	−281.7	327.3	500	0
	SP-10	500/0.05	220.1	91.8	500	0
	SP	500/0.05	**407.3**	47.5	**273.9**	105.2
	SP	1000/0.05	161.0	142.3	**589.7**	260.4
	SP	1000/0.005	**886.5**	435.2	1000	0
2 Agents	SPSNN-10	500/0.005	**201.4**	318.4	500	0
	SPSNN-10	2000/0.005	203.5	395.7	2000	0
	SPSNN-10	500/0.01	66.3	163.7	500	0
	SP-10	500/0.005	-244.4	1054	500	0
	SP	2000/0.005	**455.7**	252.9	2000	0
	Ideal-baseline	500/0.005	507.7	389.1	500	0

Table 8.1: Evaluation results of selected experiments with various model structures and setup configurations. The max-len indicates the The zero value in the episode length STD indicates a failure of all agents reaching their respective goals.

the real robots, which confirms the signal delay and wheel slipping issues, and that the learned dynamics models can take such effects into account.

Nevertheless, the sub-optimal performance prompted a few challenges and directions for future work with the framework. Firstly, although most RL policies are ostensibly trained till convergence, they still cannot match the reward level of the baseline. The lack of lag factor in the previous reward function may lead the policy optimizer to misconstrue it as the undesired actions and therefore over-correct for it, while the sheer complexity of the simulators can also confuse the PPO optimizer without further constraints. Secondly, a factor of f in computing actions during real-world evaluation, an inherited issue from the previous work, could significantly affect the actions taken by the robots. In extreme cases, the robots may remain static or diverge when the f factor is not properly tuned manually and specifically for the model, and it is highly impractical to pin down the optimal f. Lastly, the issue of update period `dt` inconsistency can be found in many experiments. In the sim-to-real policy training, maintaining a low `dt` leads to a very slow convergence, if at all, and increasing the maximum number of iterations per episode resulted in fatal crash due to computation divergence.

Chapter 9

Future work and Summary

9.1 Future Work

9.1.1 Modelling

The current modelling framework can be applied to learn any underlying controller module with a black box assumption. However, in some studies [29, 31], a more tailored approach takes advantage of the internal mechanism of the response to the reference states, and the task becomes predicting the difference between the actual state and the state calculated by tapping into the model. Despite the reportedly improved performance, our framework aims at generalising the dynamics modelling to a wide range of systems.

Although the experiments so far have been extensively performed on robomasters, the framework can be further evaluated on other robots. Systems with a higher degree of freedom in dynamics modelling, such as quadrotors and multi-joint robotic arms, can be modelled by only changing the number of input parameters. Positional information can be likewise included to account for the position-dependent effects (e.g. drone down-washing).

In terms of model parameters, only models on the lighter scale with at most 10k weights have been experimented with, while those with higher capacity, such as transformer-based models dealing with time-series data are not included here. Although we have shown for more noisy data, more complicated models are associated with a higher probability of overfitting, it is still worth experimenting with the large-scale pretrained attentional models.

9.1.2 Sampling

Throughout the sampling experiments, positional data have been omitted for better generalisation; however, locality-aware active sampling methods can be further developed to account for a specified and complex topology. For instance, although the ground on which the robomaster navigates for all the experiments above is relatively homogeneous,

there are small circular patches with a lower friction coefficient, where the probability of skidding significantly increases around these areas, and thus, a higher model uncertainty arises on these regions. By incorporating positions into the state space, these patches can be located and sampled more.

For offline active learning experiments, we have only demonstrated the efficiency of active learning for a smaller dataset. To completely replace training with the full dataset, a stopping rule can be developed and applied to the current procedure, so that only a relatively small subset is needed for even better results.

Although the online active sampling has the most theoretical edge in terms of collecting informative samples efficiently, one major issue with the current implementation is in-the-loop model training. With the current uncertainty quantification techniques, the estimates can only be made after training the model on the existing dataset, which grows in size linearly with time, resulting in an exponential overall training time. Since the ROS2 running loop can be significantly impacted by inner iterations, it is infeasible to perform the training while collection script is running. Hence, two potential solutions have been conceived for future work:

1. Training the model in parallel. Instead of interrupting the data collection scheme, the model can be updated in separate processes, while the distribution can be either resumed to default or preserved from the previous OAS iteration.

2. Discarding previous states. Rather than keep the full dataset of all previous results, we can train the model iteratively on the freshly collected data only, keeping the training cost constant.

Finally, for robomaster experiments, independence between v_x and v_y has been assumed, and in some modelling methods, each variable is predicted independently from the previous and reference states of only that component. Meanwhile, all sampling methods conduct data re-sampling with the current distribution and data uncertainty of that component only as well. This method generalises well in this simpler setup as the two states are almost independent, but a multi-dimensional sampling should be explored to better represent the less certain and more prone-to-error states.

9.2 Summary

Towards the goal of establishing a full framework for reducing sim-to-real gap through a data-driven approach, three components have been implemented and detailed so far: modelling, sampling, and transferring. In modelling framework, there are a diverse selection of models, ranging from simple linear functions to ML models, then to complex nonlinear neural networks, the performances of which are systematically analyzed qualitatively through trajectory animations and quantitatively with a collection of metrics. The full

data-processing, training, evaluation, and result visualisation pipeline is incorporated in the framework. Furthermore, we successfully emulated the environment through several data-driven models and empirically proved that several factors, most significantly data smoothness, past-states visibility, and model linearity, can improve the vanilla regression approaches when tuned properly.

Independent from the modelling framework to a large extent, the implemented sampling architecture also proved effective in multiple scenarios based on an uncertainty estimation scripts. For the online branch, inverse sampling and OAS methods were invented and their validity and efficacy were demonstrated, while for the offline branch, active learning using both MCDropout and Ensemble schemes was tailored for the dynamics learning task. The results obtained from the few-shot setup indicated the superiority of AL approach over training on randomly selected examples.

Having established the whole framework, the GNN-enabled RL policy for multi-agent passing problem was trained and evaluated in the data-driven "simulation" first, which produced well-performing policies that could guide the agents to their goal points in the same simulation during evaluation. The transfer to real-world evaluation, however, was more challenging potentially due to fatal mismatches of update frequency and insufficient training when deployed on the hardware. In addition to the qualitative analysis of the robot formation behaviour and explanation for the sub-optimal performance, as the full transfer framework is in place, we can easily evaluate the real-life performance quantitatively by plugging in new dynamics models once the aforementioned issues are resolved.

Bibliography

[1] Nick Jakobi, Phil Husbands, and Inman Harvey. Noise and the reality gap: The use of simulation in evolutionary robotics. In Federico Morán, Alvaro Moreno, Juan Julián Merelo, and Pablo Chacón, editors, *Advances in Artificial Life*, pages 704–720, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

[2] Yiannis Kantaros and George J. Pappas. Scalable active information acquisition for multi-robot systems. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7987–7993, 2021.

[3] Gautam Salhotra, Christopher E. Denniston, David A. Caron, and Gaurav S. Sukhatme. Adaptive sampling using pomdps with domain-specific considerations. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2385–2391, 2021.

[4] Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning, 2017.

[5] Jingwei Zhang, Lei Tai, Peng Yun, Yufeng Xiong, Ming Liu, Joschka Boedecker, and Wolfram Burgard. Vr-goggles for robots: Real-to-sim domain adaptation for visual control, 2019.

[6] Jan Blumenkamp, Steven Morad, Jennifer Gielis, Qingbiao Li, and Amanda Prorok. A framework for real-world multi-robot systems running decentralized gnn-based policies, 2021.

[7] Sumeet Batra, Zhehui Huang, Aleksei Petrenko, Tushar Kumar, Artem Molchanov, and Gaurav Sukhatme. Decentralized control of quadrotor swarms with end-to-end deep reinforcement learning, 2021.

[8] Lucas Rath, Andreas René Geist, and Sebastian Trimpe. Using physics knowledge for learning rigid-body forward dynamics with gaussian process force priors. In *5th Annual Conference on Robot Learning*, 2021.

[9] Javier Garcıa and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

[10] G.P. Liu. Nonlinear identification and control: A neural network approach [book review]. *Control Systems Magazine, IEEE*, 22:103– 103, 11 2002.

[11] Eyke Hüllermeier and Willem Waegeman. Aleatoric and epistemic uncertainty in machine learning: an introduction to concepts and methods. *Machine Learning*, 110(3):457–506, mar 2021.

[12] Moloud Abdar, Farhad Pourpanah, Sadiq Hussain, Dana Rezazadegan, Li Liu, Mohammad Ghavamzadeh, Paul Fieguth, Xiaochun Cao, Abbas Khosravi, U. Rajendra Acharya, Vladimir Makarenkov, and Saeid Nahavandi. A review of uncertainty quantification in deep learning: Techniques, applications and challenges. *Information Fusion*, 76:243–297, 2021.

[13] Ilias Bilionis and Nicholas Zabaras. Multi-output local gaussian process regression: Applications to uncertainty quantification. *Journal of Computational Physics*, 231(17):5718–5746, 2012.

[14] Christian Fiedler, Carsten W. Scherer, and Sebastian Trimpe. Practical and rigorous uncertainty bounds for gaussian process regression, 2021.

[15] Carl Edward Rasmussen. *Gaussian Processes in Machine Learning*, pages 63–71. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[16] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning, 2015.

[17] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, apr 2017.

[18] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles, 2016.

[19] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2016.

[20] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2017.

[21] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry, 2017.

[22] Ziyi Liu. Scaffold-based self-supervised learning for molecular graphs, 2022.

[23] Ryan Kortvelesy and Amanda Prorok. Modgnn: Expert policy approximation in multi-agent systems with a modular graph neural network architecture, 2021.

[24] Dengpeng Xing, Jiale Li, Yiming Yang, and Bo Xu. General robot dynamics learning and gen2real, 2021.

[25] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[26] Guillem Torrente, Elia Kaufmann, Philipp Foehn, and Davide Scaramuzza. Data-driven mpc for quadrotors. 2021.

[27] Jonathan Ko and Dieter Fox. Gp-bayesfilters: Bayesian filtering using gaussian process prediction and observation models. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3471–3476, 2008.

[28] S. Mohammad Khansari-Zadeh and Aude Billard. Learning stable nonlinear dynamical systems with gaussian mixture models. *IEEE Transactions on Robotics*, 27(5):943–957, 2011.

[29] Kong Yao Chee, Tom Z. Jiahao, and M. Ani Hsieh. Knode-mpc: A knowledge-based data-driven predictive control framework for aerial robots, 2021.

[30] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations, 2018.

[31] Leonard Bauersfeld, Elia Kaufmann, Philipp Foehn, Sihao Sun, and Davide Scaramuzza. NeuroBEM: Hybrid aerodynamic quadrotor model. In *Robotics: Science and Systems XVII*. Robotics: Science and Systems Foundation, jul 2021.

[32] Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J. Andrew Bagnell, Pieter Abbeel, and Jan Peters. An algorithmic perspective on imitation learning. *Foundations and Trends in Robotics*, 7(1–2):1–179, 2018.

[33] Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 4950–4957. International Joint Conferences on Artificial Intelligence Organization, 7 2018.

[34] Andrew Y. Ng and Stuart J. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, page 663–670, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[35] Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. *CoRR*, abs/2009.13303, 2020.

[36] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, may 2018.

[37] Paul Christiano, Zain Shah, Igor Mordatch, Jonas Schneider, Trevor Blackwell, Joshua Tobin, Pieter Abbeel, and Wojciech Zaremba. Transfer from simulation to real world through learning deep inverse dynamics model, 2016.

[38] Konstantinos Bousmalis, Nathan Silberman, David Dohan, Dumitru Erhan, and Dilip Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks, 2016.

[39] Ajay Shankar, Sebastian Elbaum, and Carrick Detweiler. Freyja: A full multirotor system for agile & precise outdoor flights. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 217–223. IEEE, 2021.

[40] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074, 2022.

[41] M.A. Hearst, S.T. Dumais, E. Osuna, J. Platt, and B. Scholkopf. Support vector machines. *IEEE Intelligent Systems and their Applications*, 13(4):18–28, 1998.

[42] L Breiman. Random forests. *Machine Learning*, 45:5–32, 10 2001.

[43] Abraham. Savitzky and M. J. E. Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical Chemistry*, 36(8):1627–1639, 1964.

[44] Ram Popat. Embracing the uncertainty of deep learning models to support clinical decision making for mental health conditions. Intend to publish soon, 6 2021.

[45] Sriram Vajapeyam. Understanding shannon's entropy metric for information, 2014.

[46] Oldrich Vasicek. A test for normality based on sample entropy. *Journal of the Royal Statistical Society. Series B (Methodological)*, 38(1):54–59, 1976.

[47] Bert van Es. Estimating functionals related to a density by a class of statistics based on spacings. *Scandinavian Journal of Statistics*, 19(1):61–72, 1992.

[48] Jason Brownlee. Information gain and mutual information for machine learning, 2020.

[49] Yarin Gal, Riashat Islam, and Zoubin Ghahramani. Deep bayesian active learning with image data, 2017.

[50] Damon Wischik. Probabilistic machine learning notes, 2021.

[51] Han Su, Shuncheng Liu, Bolong Zheng, Xiaofang Zhou, and Kai Zheng. A survey of trajectory distance measures and performance evaluation. *The VLDB Journal*, 29(1):3–32, jan 2020.

[52] wannesm, khendrickx, Aras Yurtman, Pieter Robberechts, Dany Vohl, Eric Ma, Gust Verbruggen, Marco Rossi, Mazhar Shaikh, Muhammad Yasirroni, Todd, Wojciech Zieliński, Toon Van Craenendonck, and Sai Wu. wannesm/dtaidistance: v2.3.5, January 2022.