Protocol of PRISM 2023 summer term

Thomas Kob
Friedrich-Alexander-Universität
Erlangen, Germany
thomas.kob@fau.de

Marcel Schöckel
Friedrich-Alexander-Universität
Erlangen, Germany
marcel.schoeckel@fau.de

***Abstract*—Inertial Motion Tracking (IMT) allows measuring the orientation of objects without external references. This usually requires many 9D-Inertial Measurement Units (IMUs), measuring acceleration, rotation and the magnetic field. Recently, a machine-learning based approach has been applied, achieving successful IMT with sparse 6D-IMU-setups, thus decreasing the system complexity. In this protocol, we propose a training technique that includes rigid (motionless) phases in training data. We showcase that this augmentation can increase the performance of the inference, as adding few short or many very short stops can halve the mean angle error (mae) between prediction and truth, compared to classic training.**
Additionally, we implemented functionality to export systems that represent these setups from Python objects to the XML format.

I. Introduction

Recent advances have made Inertial Measurement Units (IMUs) smaller and more affordable, increasing their use in Inertial Motion Tracking (IMT). Typically, 9D-IMUs are used for sensor fusion in kinematic chains. However, attaching an IMU to each segment can be cumbersome and expensive. Additionally, IMT applications can be affected by magnetic disturbances. These issues can be mitigated by using sparse magnetometer-free (6D) sensor fusion. The `x_xy`[1] package provides a solution for this, by utilizing a Recurrent Neural Network observer (RNNo) to predict the orientation of sparse IMU kinematic chains. This does a great job on continuous movements, but diverges whenever the movement halts. In this report, we aim at reducing the divergence and enhancing the overall accuracy by incorporating rigid phases into the generated data.

II. Simulation Setup

In this section, we explain our experimental setup by illustrating our data generation process including rigid phases, how we train the RNNo and run inference on the network.
The body of interest is a kinematic chain, consisting of three chain segments, numbered `seg1`, `seg2` and `seg3`. All segments are connected via hinge joints. `seg1` to `seg2` in y-direction. `seg3` to `seg2` in z-direction. On both `seg1` and `seg3` a fixed IMU is attached. `seg2` is the center part of our three-segment-chain without any IMUs. The goal is to predict the orientation of `seg2` and `seg3` using the IMUs attached to `seg1` and `seg3`.
To train the RNNo for our specific problem, it is first required that we generate training data for it, since the amount of available experimental data is very limited. For this, we use the already existing `random_angle_over_time` function, that creates a series of angles for n discrete frames. We overlay the angular differences with a mask, to incorporate transitions and rigid phases. For the creation of this mask, we implement a `motion_amplifier` function, which apart from time and sampling rate, takes the number of halts as an argument. To control the duration and transition from a *motion* to a *rigid* phase, two corresponding variances $\sigma_{r,i}, \sigma_{tr,i}$ for each *rigid phase* are also passed. Consequently, each rigid phase is described by a rigid duration $t_{r,i}$ and a transition duration $t_{tr,i}$, which are normal distributed, so that $t_{r,i} \sim \mathcal{N}(0, \sigma_{r,i}^2), t_{tr,i} \sim \mathcal{N}(0, \sigma_{tr,i}^2)$. The mask values are set to 1 during motion phases, set to 0 during rigid phases and linearly interpolated in transition phases. To support fast calculation times, the entire function can be compiled just-in-time.
This data generator is then registered as a special hinge joint type, as in `x_xy`, joint types include a random generator function, which can be used during training and inference.
In order to simulate a more realistic scenario, for each sequence during training, the length and position of the segments and the IMU positioning is altered slightly, since experimental data will most likely contain inaccuracies. The length of $\mathcal{S}_2$ is increased by moving $\mathcal{S}_1$ and $\mathcal{S}_3$ away by a uniformly drawn distance between 0.05 and 0.3 meters along $\mathcal{S}_{2,x}$. On the two other axes, $\mathcal{S}_1$ and $\mathcal{S}_3$ are moved by a uniformly drawn distance between -0.02 and 0.02 meters. The IMUs are also moved analogously by $d_x \in [0.05, 0.25]m$ on $\mathcal{S}_{2,x}$ and $d_y, d_z \in [-0.05, 0.05]m$ on the other respective axes in S_2.
We then train the network for the *configurations* we describe in Section III and obtain a corresponding parameter set for each. A data set of sufficient quality requires large batch sizes, in our case 80 and a very high number of epochs (1500) are necessary for the verification to reach stable results. To perform inference, we provide the network with the parameters retrieved from training, a system description of our kinematic chain and the IMU data, generated or experimental. The network will then output the estimated orientations $\hat{y}$.

III. Evaluation

In this section, we evaluate the behaviour of the RNNo concerning the angular errors for different problem configurations, by measuring the difference between the predicted and the actual orientations.

[1] https://github.com/SimiPixel/x_xy_v2

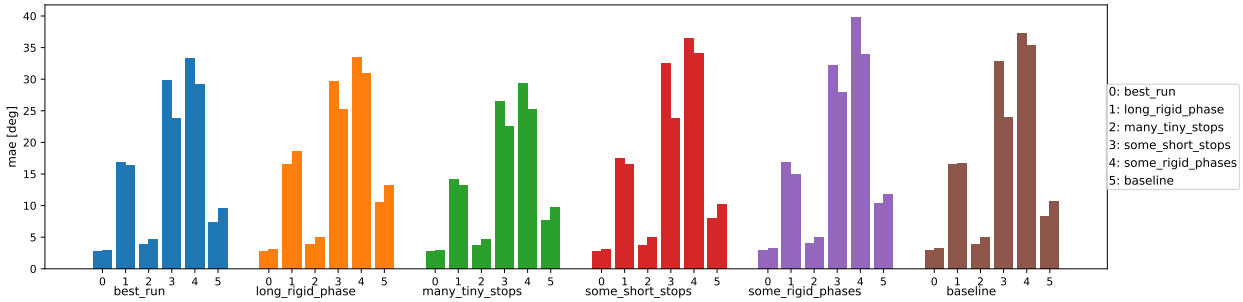

Fig. 1. Mean angle error by performing inference on generated data. Bars of equal color used the same problem configuration for inference.

TABLE I
CONFIGURATIONS FOR PROBLEMS

Configuration no.	(0)	(1)	(2)	(3)	(4)	(5)
Rigid phases (n)	1	1	30	5	3	0
$\sigma_{r,i}$	0.02	0.25	0.001	0.02	0.05	-
$\sigma_{tr,i}$	0.1	0.1	10^{-4}	0.005	0.01	-

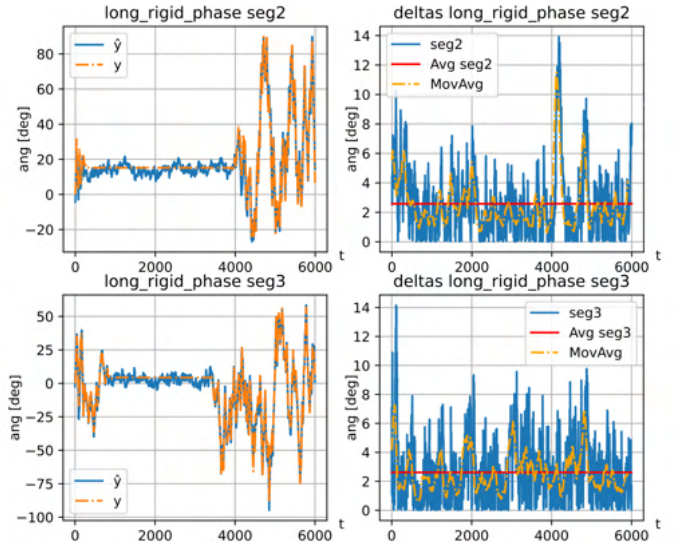

Fig. 2. Inference with parameters trained on (0) on a series generated by (1)

The problem definitions are displayed in Table I. These definitions consist of one short (0), one longer (1), and a series of many very short stops (2). To evaluate the number of stops, we also consider a series with five short (3) and three slightly longer stops (4). For the sake of comparability, we also consider a data series without any halts (5). Therefore, we have trained the network on all the problems mentioned above. In the following, we will first evaluate the results for generated, then for experimental data, acquired by OMC.

A. Generated data

For each configuration we trained on, we generate five unique data series per configuration, resulting in a total of 30 data series. We then run inference on all 30 data series.

Each data series contains data of two chain segments, *seg2* and *seg3*. For each segment, y is the true angle of the joint axis and $\hat{y}$ the prediction made by the RNNo. An example data series, trained on configuration (0) and inferred on (1) is shown in Fig. 2. The upper row refers to segment 2, the lower row to segment 3. On the right side, the absolute difference between y and $\hat{y}$ are displayed in the blue lines, as well as a moving average and the average over the differences in the series. This average is used for the calculation of the mean angle error (mae) in Fig. 1. Averaging over the five unique series, we end up with one mae per trained configuration per inferred configuration. Each configuration (0)-(5) corresponds to a certain naming which can be taken from the legend on the right. Each numbered *tick* on the x-axis refers to the configuration for training, while each *naming* and coloring refers to the configuration for inference. For each tick there are two bars, representing the mae of segment 2 (left) and 3 (right) independently. All data series used for the creation of Fig. 1 can be taken from Appendix A.

Using the baseline training without rigid phases (tick 5) as a reference, we can quickly discard the configurations (1), (3) and (4) as all of them consistently perform worse. Looking at ticks 0 and 2 however they outperform the baseline training by halving the mae.

Fig. 2 displays very well how the configuration `long_rigid_phase`/(1) determines this data series, as the long pause is clearly visible. The prediction of the RNNo is almost correct throughout, with an average angular error of less than 3 degrees. Instabilities are especially noticeable when there are very strong variations, where the prediction is slightly shifted in time, for example at segment 2 around frame 4200, which causes high angular errors for a short time. During the rigid phase, the prediction is also slightly unstable, as it is not able to detect that the system is not moving at all. Nevertheless, it does not diverge in either case.

Adding more and longer rigid phases however greatly reduces the stability of the inferred data. As can be seen in Fig. 1, for a long rigid phase (1) and for 5 short stops (3), the mea gets progressively worse. In both cases, the prediction does not completely diverge from the truth, as can be taken from

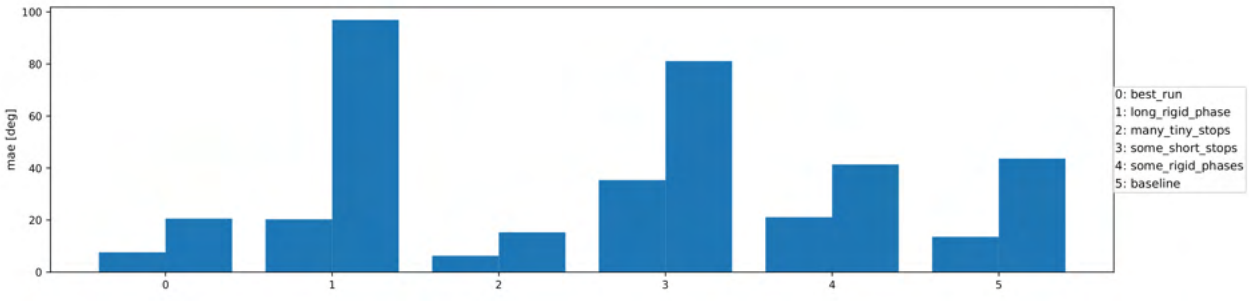

Fig. 3. Mean angle error by performing inference on experimental data.

all data series of Appendix A. For 3 longer stops however, this statement does not hold true anymore, as the data series oscillate between positive and negative values. It is also noticeable that in all cases, *seg3* has a higher mae as *seg2*, which is due to the fact, that *seg3* is attached to *seg2* and therefore dependent on its orientation.

B. Experimental data

In Fig. 3, the results of the experimental data are displayed. The x-axis contains the training configurations (0)-(5) for both segment 2 and 3. The bars, similar to Fig. 1, represent the mae of inference on actual IMU data, recorded with OMC.
The statements about the generated data made in Section III-A still hold true for experimental data. Surprisingly the configuration `many_tiny_stops`/(2) outperforms the configuration `best_run`/(0) which prevailed on generated data. Looking at the raw data series provided in Appendix B, the reason for this is the 5. batch of the experimental data, which the RNNo trained on configuration (0) does not handle very well. The prediction error is about 7 times higher compared to the other batches. We deem an unfortunate seed chosen for the random generator in training as the cause, since the data of this batch does not differ much from the other batches. To generate the data, pseudo-random-generator (PRNG) keys are used. For inference, the RNNo was trained using 0 as a seed for the key. With other keys, inference by (0) is more stable in the series and achieves a lower mae than (2), making this an exception. To fully determine why this exception arises, more research is necessary.
Looking at the results, it turns out that it can be beneficial to include rigid phases during training of the network. It is however imperative to find a good balance, since too many and too long halts have a negative impact on the outcome of the training, far more beyond than not using any rigid phases at all. Including one short ore several very short stops can however greatly improve the accuracy of the trained RNNo.

IV. Further additions to x_xy

A. Notes on the "System to XML" Functionality

In addition to the previous work on enhancing model accuracy, we also worked directly on the `x_xy` code-base. Parsing XML files and strings to systems, Python objects that describe the structure of kinematic chains, is already a core feature, and used every time a system is loaded. Contrary, exporting a system to an XML String or File was not yet possible, but proves to be useful. Now, with `sys2xml`, when modifying properties of a system in Python, one can easily export and save it in an XML format.
The algorithm starts by creating an XML-string with constant XML elements like *"x_xy"* or *"worldbody"*. It then loops over all links found in the system and recursively processes their children. For each element found it extracts their properties and injects them into the XML-string. The resulting XML-string can also be directly saved as a file.

V. Conclusion

In this report we presented a method for training an RNNo for a sparse IMU kinematic chain by generating random angles with rigid phase in order to enhance the prediction of the orientation for each segment. We provide functionality that allows for precise determination on how this random data may look like in a configuration, to optimize inference.
We successfully train two configurations, that yield solid results and outperform the classic training by a factor of two. While too many short or too long rigid phases negatively affect results, leading to a divergence of actual and predicted orientations, many very short or little short rigid phases have a positive effect on the inferred results. These statements hold true for both generated and actual data, acquried by OMC. Even though large batch sizes were used during training, the impact of PRNG keys are still visible on some data series, meaning that the result is still dependent on the key choice before training.
The `sys2xml` function is a powerful tool when working with systems in code, as it enables to quickly merge and export two smaller systems, without having to manually create a large XML file, and contributes to the already extensive system composing capabilities of *x_xy*.

Acknowledgment

We gratefully acknowledge the quick, helpful feedback and support of Simon Bachhuber and the compute resources provided by the AIBE department of the FAU.