Where's WALL-E? A Comparison of the Extended Kalman Filter and Hybrid Inference for Pose Estimation in MAVs

Roger Milroy
Department of Computer Science
Royal Holloway, University of London
Egham, UK
roger.milroy.2016@live.rhul.ac.uk

***Abstract*—Pose estimation is a core competence for cyber-physical systems and is all the more important where there is any element of autonomy.**

In the context of Micro Air Vehicles (MAVs) this task is more challenging due to weight and cost restrictions. These restrictions dictate that MAVs usually have noisy sensors and limited computational capacity. There are many different approaches to solving this problem but the standard approach is to use the Kalman Filter (KF) [1], or it's nonlinear variant the Extended Kalman Filter (EKF) [2], in order to fuse sensor data and provide optimal estimates of state.

While the KF is an optimal estimator of linear systems given some assumptions, most systems are non-linear so the EKF is used. In either case the estimates rely on assumptions that may not always hold. This allows room for improvement. This paper implements the newly proposed technique of Hybrid Inference (HI) [3] on a model of an MAV simulated in Gazebo [4] and explores its performance as compared to the EKF which is used as the standard.

HI is a framework for combining graphical models, like the KF, with inverse models which are learned with a Recurrent Message Passing Neural Network (MPNN) [5][6].

This paper evaluates the technique in a more challenging domain than has previously been implemented. It explores the challenges of implementing the technique, analyses its computational performance and discusses its suitability for use at this time with a strong practical focus.

The main findings are that it is too challenging to implement correctly to take full advantage of its proposed benefits. And that it is too computationally inefficient in its current form for it to be suitable for use in real time systems with current technology.

***Index Terms*—Neural nets, Neural models, Graph Neural Networks, Filtering, MAV, UAV**

I. Introduction

Pose estimation is a fundamental competence for cyber-physical systems such as Micro Air Vehicles (MAVs) which includes devices such as quadcopters. It is essential for navigation, which enables many of the potential uses of this class of vehicle.

The problem of pose estimation can be formulated as that of inferring a set of latent, unobserved variables that represent the true state of the system (the MAV) from a set of noisy observations. The solution to this problem is the Kalman Filter (KF). For linear dynamic systems a correctly specified KF gives optimal estimates of state from noisy observations.

For the case of MAVs, we cannot use a KF because the system is non-linear. This means that we must make use of a technique that is able to deal with non-linearities such as the Extended Kalman Filter (EKF) [2].

A. Hybrid Inference

While the KF is proven to be an optimal estimator of state for linear dynamic systems given some assumptions the same is not true for the EKF. It is still used widely because despite not having proven optimality it often performs well in practice. There is room for us to improve the KF, as the property of being an optimal estimator is only true if the noise follows certain parameters and we characterise it well, which is not always easy. There is even more room for improvement with the EKF as the linearisation process is imperfect and can introduce error.

Hybrid Inference (HI) is a general framework for combining mathematical models and Deep Learning models. The original paper used the Kalman Filter as the example. They reformulated the problem solved by the KF to be a Maximum Likelihood problem [3], where the states are $\bar{\mathbf{x}} = \{x_0, x_1 ... x_k\}$ and the observations are $\bar{\mathbf{y}} = \{y_0, y_1 ... y_k\}$. In this context, the task is to predict the optimal estimate of $\bar{\mathbf{x}}$, $\hat{\mathbf{x}}$ which is defined as

$$\hat{\mathbf{x}} = \underset{x}{\operatorname{argmax}}\ p(\bar{\mathbf{x}}|\bar{\mathbf{y}}) \tag{1}$$

They assume a Markov process and that the transition is stationary so this can be expressed as

$$p(\bar{\mathbf{x}}, \bar{\mathbf{y}}) = p(x_0) \prod_{t=1}^{T} p(x_t|x_{t-1}) p(y_t|x_t) \tag{2}$$

They model this as an iterative optimization process to arrive at $\hat{\mathbf{x}}$ Specifically they define a recursive update operation for the general case and then formulate it for the Hidden Markov case

$$x_t^{(i+1)} = x_t^{(i)} + \gamma M_t \tag{3}$$

Where M_t represents the sum of matrix products, which they call messages, from $x_t - 1$ to x_t from $x_t + 1$ to x_t and from y_t to x_t.

In our case these messages turn out to be

$$x_{t-1} \to x_t = -Q^{-1}(x_t - Fx_{t-1}) \tag{4}$$
$$x_{t+1} \to x_t = F^T Q^{-1}(x_{t+1} - Fx_t) \tag{5}$$
$$y_t \to x_t = H^T R^{-1}(y_t - Hx_t) \tag{6}$$

The graphical interpretation is of the xs and ys at each time step forming nodes in a graph. The edges are the same as the direction of the messages, that is from $x_{t-1} \to x_t$, $x_{t+1} \to x_t$ and $y_t \to x_t$. The messages passed over these edges iteratively update the xs and after some iterations they converge to a best estimate.

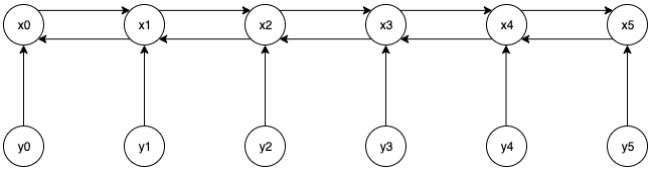

Fig. 1. Graphical Representation of the Kalman Filter

This graphical interpretation allows us to define an equivalent graph with different dimension nodes but the same edges. These are known as h_x and h_y which stands for hidden nodes relating to x and y respectively.

The key part of the paper is to define a Message Passing Neural Network (MPNN) [5], a generalisation of a Graph Neural Network (GNN), that operates over this graph with a message passing routine over the nodes that connect edges and the messages passed in the original graph. More specifically for each type of edge, for example $y_t \to x_t$, we define a feedforward neural network that takes the source node, the target node and the message and outputs an encoding of the edge, I will refer to these as edge models. This corresponds to the Message function of the MPNN. These encodings are summed according to their source node and then passed through a separate feedforward network which I will call the node model. This corresponds to the Update function.

The output of this is passed through a GRU along with the previous h_x in order to produce a new estimate of h_x. The interpretation of this is that edge feedforward networks compute the residual error over the edges, the node model computes the residual error left in the h_x and the GRU allows some of the past residual error to propagate into the current estimate. The final step passes the new h_x through a decoding step to produce an additional corrective factor in addition to M_t called ϵ.

This gives us the final general recursive update rule

$$x_t^{(i+1)} = x_t^{(i)} + \gamma(M_t + \epsilon_t) \tag{7}$$

II. Implementation

To fully evaluate HI and the process that a practitioner would go through to use it in their application it was decided to reimplement the technique with reference to the paper and the original implementation. Due to this a much fuller account of the practical aspects is possible and can therefore better inform decision making around the use of HI in industry.

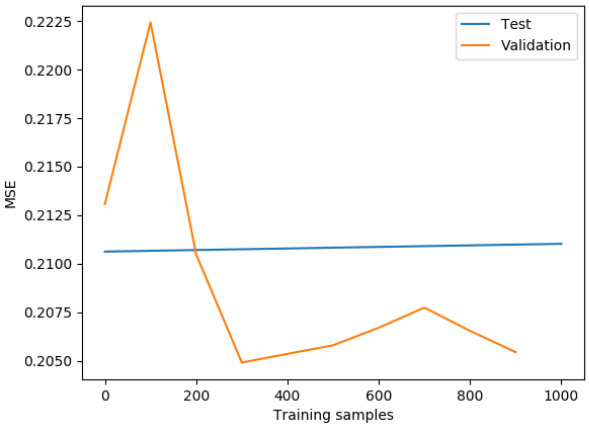

Fig. 2. HI training.

The first step was to understand the structure of the existing implementation and then re-engineer it. It was decided to separate more completely the graphical model and the GNN model. This allowed for reusability and had some advantages for testing. The original design made use of some more exotic functions in PyTorch to enable the most faithful implementation of the theory. It was decided to focus on ensuring that the code was functionally equivalent while being easier to interpret and understand. This also had the desire of being slightly more computationally efficient. It also simplified the inputs to the model considerably. The original implementation needed a specification of the edges and nodes as well as the measurements. This increased the cognitive load for any practical applications as well as being fairly redundant for a system designed only for KF type applications, as was the case for this paper.

III. Evaluation

A. Validation of HI over a linear data set

The first step was to validate the technique on a similar dataset to those in the original paper. To this end a similar synthetic datasest was created of state space size 4 with position and velocity in 2 dimensions.

The graphical model was found to be very sensitive to implementation differences, with even very minor changes resulting in extremely poor results. However once correctly implemented it did result in significantly better accuracy than the reference implementation of the KF using Rauch-Tung-Striebel Smoothing (RTS Kalman Smoother).

For Figures 2 and 3, the Mean Squared Error (MSE) for the reference RTS Kalman Smoother was 0.9232, approximately 4 times the best results achieved by the untrained HI.

Training HI was not simple. The ability to predict generalisation performance from the validation error was very

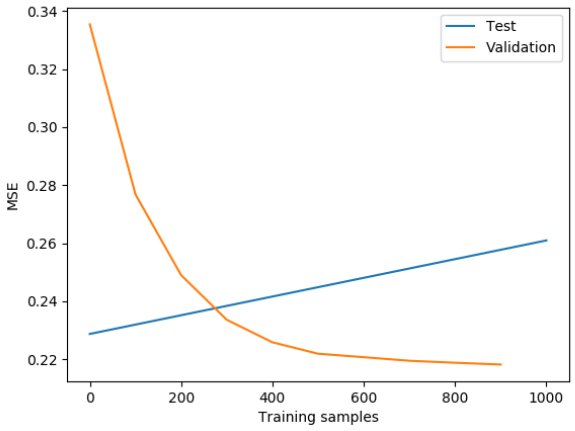

Fig. 3. HI training.

limited, with many training runs resulting in increased test set error compared with no training. Even with a very promising validation curve as shown in Figure 3 the error on the test set became worse. These training graphs are representative of the experience of training HI on this dataset. There appeared to be little difference between the training of perfectly parameterised HI and less well parameterised HI. The learning rates made some difference but it was not able to generalise to the test set at any point.

This raises some questions about the technique as the expected behaviour is that an equally parameterised KF and untrained HI should have very similar MSE. And that a slightly misparameterised HI should be able to approach the error of a perfectly parameterised KF with training.

I should mention that this is not meant to question the results of [3]. Their code is available and does indeed reproduce the results they presented. Instead it is meant to highlight the extent to which implementation affects the techniques effectiveness.

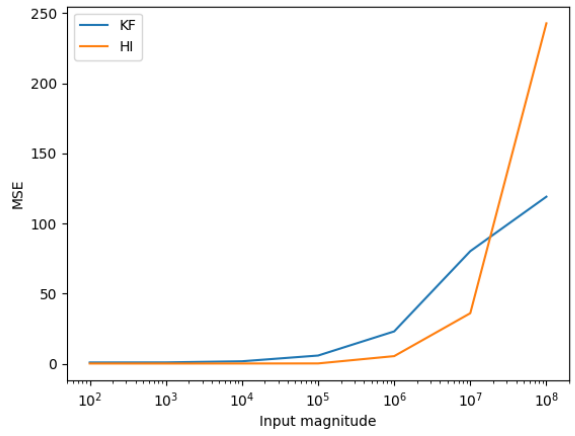

Fig. 4. Error vs input magnitude.

One interesting feature that was noted during testing was the different behaviour of the two techniques with regard to large input values. When considering untrained versions of HI, I found that for values up to 10^7 it had significantly lower MSE than the standard Kalman Smoother. Beyond 10^8 however, the MSE of HI outstripped the Kalman Smoother. This is not the focus of this paper and so no further work has been done to explain it however it is worth noting and perhaps further investigation into the causes.

B. Accuracy and Training

1) MAV simulation dataset: Here we move on to evaluate performance in one of these more challenging domains, that of an MAV, in particular a simulated quadcopter. This has a state space of size 15, position, velocity, acceleration, orientation and rate, each in three dimesions.

For testing the EKF and non linear version of the HI, which I will refer to as EKHI from now on, data was recorded from the simulation software Gazebo [4]. The hector suite of packages provided by TU Darmstadt [7] was used to simulate a prototypical quadcopter. Samples were taken at 100Hz with simulated IMU, barometer and GPS measurements. Ground truth values were collected from Gazebo itself. Trasition matrices were calculated by a module in the hector suite.

The data consisted of trajectories generated by the quadcopter performing a number of manoeuvres between randomly selected points. This ensured a wide variety of trajectories for the model to train on.

2) Accuracy comparison with EKF: There was some modification necessary to the linear version of HI in order to operate over non linear datasets. The main one was to operate with one transition matrix per timestep. It was also necessary for generalisability to operate with one measurement matrix per timestep. In the model evaluated this was not necessary due to the linear nature of the measurements taken. From here on the non-linear version of HI will be referred to as the EKHI.

The EKF used for comparison was from filterpy, a pip package [8]. The untrained EKHI was compared against a reference implementation of the EKF which was implemented by filterpy. The EKHI had an MSE approximately double the EKF before training. With training this was brought down to be below that of the EKF. This was much more successful than the KF however this was still not the expected behaviour which was to equal the EKF without training but improve steadily with training. It does however support the hypothesis that implementational issues are at the heart of the difference in performance.

It is important to note the difference in MSE between the validation set and the test set. This technique appears to be quite sensitive to dataset distributions which reinforces the need for representative data to train on.

3) A note on training: As is the case for most deep learning techniques, training is not a trivial process and cannot be assumed to run smoothly first time. While it is certainly possible to achieve good results from training EKHI, it is quite sensitive to certain parameters.

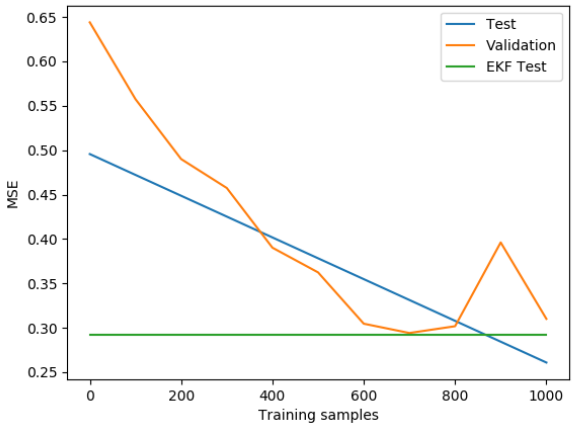

Fig. 5. EKHI training.

Before starting training, it is important that the underlying graphical model is tuned with respect to two parameters. γ which is the weighting of each iterations update to the estimate, and the number of iterations that HI runs. If these are not chosen well to begin with, the training is unlikely to improve the model very much and often makes the model worse than with no training at all.

Training is fairly unpredictable and some experimentation is needed to find a reasonable learning rate, as the optimal varies from problem to problem. Even with a good learning rate the process can vary a lot and the greatest challenges come with training over larger datasets ($>$1000 samples). In general however lower learning rates are better. One of the reasons for the challenging nature of training is the recurrent nature of the GNN. This leaves it with many of the challenges encountered by vanilla Recurrent Neural Networks (RNNs), the exploding and dissapearing gradient problems. The techniques used to allow training on traditional RNNs may be helpful for HI as well.

In the original paper the objective function used for training is a weighted version of MSE loss where early iterations have less impact on the loss and thus the modification of the GNN weights. The experiences of training for this paper have shown that the weighted loss is harder to tune and generally produces worse results than optimising for MSE loss directly over the final estimate. It is possible that this is due to implementation differences.

C. Performance - Speed

One of the main findings of this paper is that the technique has significant computational overheads. When compared to the KF and EKF, HI performs two to three orders of magnitude slower. This is very dependant on the number of iterations of the model.

Both HI and EKHI have similar behaviour with error decreasing monotonically with increasing number of iterations. HI is much more sensitive to low numbers of iterations where EKHI is far less so. Speed of execution drops by nearly two orders of magnitude. This has been replicated on both consumer and cloud infrastruture. One interesting feature of performance is that GPU acceleration makes little to no difference to wall clock performance which demonstrates that matrix multiplication is not the bottleneck here.

TABLE I
IMPACT OF ITERATIONS ON EXECUTION SPEED AND ERROR

	Linear HI		EKHI	
iterations	it/s	MSE	it/s	MSE
5	91.51	35.176	72.19	0.481
10	47.07	11.213	37.89	0.479
20	23.61	5.857	18.92	0.477
50	9.52	1.840	7.64	0.474
75	6.47	0.816	5.18	0.472
100	4.92	0.429	3.68	0.470
150	3.26	0.221	2.66	0.468
200	2.35	0.189	1.87	0.466
250	1.93	0.184	1.59	0.464
300	1.62	0.183	1.33	0.463

The fact that it is based around an iterative optimisation procedure that is serial in nature is a fundamental challenge to improving real time execution speeds as it resists parallelisation in the way that has accelerated the excecution of Deep Neural Networks.

This leads to the interesting situation where the computational complexity is less a function of the state space, or number of samples and is instead dominated by the number of iterations of optimization.

D. Usability - Formulation

This paper has previously highlighted some of the issues of implementing this technique. Here we will explore some other scenarios for anyone attempting to use HI.

If the potential application is a Kalman Filtering problem then modifying existing modules, preferably the one from the original paper, is the preferred approach and is not overly challenging. It is the same basic process as that for the EKF, modelling the state transition model and the measurement model and then computing the Jacobians at each time step. Along with additional implementation or context specific work such as getting the data into a useable format.

If seeking to use HI with a different class of problem then it will be necessary to reformulate the problem as a Maximum Likelihood problem with a graphcial interpretation. It is necessary to derive the appropriate messages that will be passed between nodes in this graph and also define an equivalent GNN. Getting this right is not easy.

In addition, the implementation is quite sensitive. There are many minor changes that will yield poor results or a model that is basically untrainable.

To contrast this with the EKF, a large number of EKF implementations are written from scratch to cater to the context they will be used in. While it is considered one of the more demanding state estimation techniques, there is a large amount of material and tooling that supports its development either from scratch or using existing modules.

E. Usability - Data Gathering

As with other Machine Learning techniques, data acquisition is a key consideration. Getting high quality, representative

data with labels is very important. One benefit to this technique is that the quantity of data for good results is far lower than alternatives such as learning the mapping directly with Neural Networks. Gathering labels is likely to be the hardest part of the data collection problem as the measurements are likely to be fairly easy to collect.

IV. Discussion

While HI is a promising and interesting technique, this paper has shown that it is challenging to implement and unless implemented perfectly, falls short of it's promise in some areas. In addition it is currently too computationally expensive to be useful for the vast majority of real time systems and certainly for MAVs.

The technique does still hold real potential for practical improvements that could have impacts across multiple industries and so is worth further study. One effort that could help to deliver on the promise of the technique and make it available to end users is to package a well engineered and verified version of HI that is shown to be effective and trainable so that potential users would have a reduced development burden to adopt it.

It would also be helpful to potential users to have a sense of whether training could offset the need for many iterations. This seems to be a reasonable approach for the EKHI as the error decreases little with increased iterations. There would need to be some study of the relationship between the number of iterations and the amount of benefit gained through training, to optimise for different situations.

Any reformulation that removes iterative optimisation, such as alternative graphical formulations that are still able to be augmented by Neural Network approaches would be an improvement.

Due to the serial nature of optimisation removing it could allow the parallelisation of the technique. This could enable GPU acceleration to reduce wall clock execution time which in turn could enable real time applications to take advantage of it.

Other important areas of work would be to improve the robustness of the technique. This could improve the usability and open up the technique to more problem domains without needing an expert to formulate it first as well as to more less sophisticated users.

Finally it would also be interesting to explore the reason for the different performance on large inputs between KF and HI. This may be implementation specific or down to numerical issues however it would be useful to understand the reason.

References

[1] R. E. Kálmán, "A new approach to linear filtering and prediction," 1960.

[2] G. L. Smith, S. F. Schmidt, and L. A. McGee, "Application of state-space methods to navigation problems," NASA Technical Report R-135, 1962.

[3] V. G. Satorras, Z. Akata, and M. Welling, "Combining generative and discriminative models for hybrid inference," *ArXiv*, vol. abs/1906.02547, 2019.

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

[5] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," *CoRR*, vol. abs/1704.01212, 2017. [Online]. Available: http://arxiv.org/abs/1704.01212

[6] Y. Li, D. Tarlow, M. Brockschmidt, and R. S. Zemel, "Gated graph sequence neural networks," *CoRR*, vol. abs/1511.05493, 2016.

[7] J. Meyer, A. Sendobry, S. Kohlbrecher, U. Klingauf, and O. von Stryk, "Comprehensive simulation of quadrotor uavs using ros and gazebo," in *3rd Int. Conf. on Simulation, Modeling and Programming for Autonomous Robots (SIMPAR)*, 2012, p. to appear.

[8] R. R. Labbe, "rlabbe on GitHub kalman-and-bayesian-filters-in-python," https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python.