SESAME
SECURE AND SAFE MULTI-ROBOT SYSTEMS

Project Number 101017258

D2.2 Model Identification

Version 1.0
22 December 2021
Final

Public Distribution

University of Luxembourg

Project Partners: **Aero41, ATB, AVL, Bonn-Rhein-Sieg University, Cyprus Civil Defence, Domaine Kox, FORTH, Fraunhofer IESE, KIOS, KUKA Assembly & Test, Locomotec, Luxsense, The Open Group, Technology Transfer Systems, University of Hull, University of Luxembourg, University of York**

Project Partner Contact Information

Aero41 Frédéric Hemmeler Chemin de Mornex 3 1003 Lausanne Switzerland E-mail: frederic.hemmeler@aero41.ch	**ATB** Sebastian Scholze Wiener Strasse 1 28359 Bremen Germany E-mail: scholze@atb-bremen.de
AVL Martin Weinzerl Hans-List-Platz 1 8020 Graz Austria E-mail: martin.weinzerl@avl.com	**Bonn-Rhein-Sieg University** Nico Hochgeschwender Grantham-Allee 20 53757 Sankt Augustin Germany E-mail: nico.hochgeschwender@h-brs.de
Cyprus Civil Defence Eftychia Stokkou Cyprus Ministry of Interior 1453 Lefkosia Cyprus E-mail: estokkou@cd.moi.gov.cy	**Domaine Kox** Corinne Kox 6 Rue des Prés 5561 Remich Luxembourg E-mail: corinne@domainekox.lu
FORTH Sotiris Ioannidis N Plastira Str 100 70013 Heraklion Greece E-mail: sotiris@ics.forth.gr	**Fraunhofer IESE** Daniel Schneider Fraunhofer-Platz 1 67663 Kaiserslautern Germany E-mail: daniel.schneider@iese.fraunhofer.de
KIOS Panayiotis Kolios 1 Panepistimiou Avenue 2109 Aglatzia, Nicosia Cyprus E-mail: kolios.panayiotis@ucy.ac.cy	**KUKA Assembly & Test** Michael Laackmann Uhthoffstrasse 1 28757 Bremen Germany E-mail: michael.laackmann@kuka.com
Locomotec Sebastian Blumenthal Bergiusstrasse 15 86199 Augsburg Germany E-mail: blumenthal@locomotec.com	**Luxsense** Gilles Rock 85-87 Parc d'Activités 8303 Luxembourg Luxembourg E-mail: gilles.rock@luxsense.lu
The Open Group Scott Hansen Rond Point Schuman 6, 5th Floor 1040 Brussels Belgium E-mail: s.hansen@opengroup.org	**Technology Transfer Systems** Paolo Pedrazzoli Via Francesco d'Ovidio, 3 20131 Milano Italy E-mail: pedrazzoli@ttsnetwork.com
University of Hull Yiannis Papadopoulos Cottingham Road Hull HU6 7TQ United Kingdom E-mail: y.i.papadopoulos@hull.ac.uk	**University of Luxembourg** Miguel Olivares Mendez 2 Avenue de l'Universite 4365 Esch-sur-Alzette Luxembourg E-mail: miguel.olivaresmendez@uni.lu
University of York Simos Gerasimou & Nicholas Matragkas Deramore Lane York YO10 5GH United Kingdom E-mail: simos.gerasimou@york.ac.uk nicholas.matragkas@york.ac.uk	

Document Control

Version	Status	Date
0.1	Document outline	01 October 2021
0.2	First draft	01 November 2021
0.4	Internal review	01 December 2021
0.6	First full draft for partners review	16 December 2021
0.8	Further editing draft	20 December 2021
1.0	QA review	22 December 2021

Table of Contents

1 Introduction **1**
1.1 Overview . . . 1
1.2 Intentions . . . 1
1.2.1 Benefits to the project . . . 1
1.2.2 Benefits to the industry . . . 2
1.3 Outcome . . . 2
1.4 Deliverable structure . . . 2

2 Rationale **3**

3 Methodology **5**

4 Generic System Identification Theory **6**
4.1 Notation . . . 6
4.2 Model identification techniques . . . 6
4.2.1 Extended Kalman Filter (EKF) . . . 7
4.2.2 Multi-State Constraint Kalman Filter: MSCKF . . . 8
4.2.3 Unscented Kalman Filter: UKF . . . 9
4.2.4 Sliding Window Least Square: SWLS . . . 9
4.2.5 Data Driven Approach . . . 10

5 System Identification Application: UAV simulation case **15**
5.1 Notation . . . 15
5.2 Nominal quadrotor dynamic model with variable mass . . . 15
5.3 Dataset . . . 17

6 Results **18**
6.1 Implementation details . . . 18
6.2 Results of model-based methods . . . 18
6.3 Results of offline ELM and OSELM . . . 25

7 Discussion **32**

8 Supplementary Study on Environmental Interference **32**
8.1 Introduction . . . 32
8.2 Problem Formulation . . . 33
8.2.1 Notation . . . 33
8.2.2 State Vector . . . 34
8.2.3 Propagation . . . 34

8.2.4 Accelerometer Update 35
8.2.5 Visual Measurement Update 35
8.3 Real-world Experiments 35
8.4 Conclusion 37

List of Figures

1 Model-based approach scheme. . . . 3
2 Model-based approach scheme with the online model identification component. . . . 3
3 Model identification techniques studied in this deliverable. EKF: Extended Kalman Filter; MSCKF: Multi-State Constraint Kalman Filter; UKF: Unscented Kalman Filter; SWLS: Sliding Window Least Square method; ELM: Extreme Learning Machine; OSELM: Online Sequential Extreme Learning Machine. . . . 6
4 Dataset generation framework . . . 17
5 Left: Horizontal view (x-y) of the simulated trajectories; Right: Mass behaviour (y-axis) during the time (x-axis) simulated for each trajectory . . . 17
6 Z direction of the simulated random trajectories . . . 18
7 Mass estimation (y-axis [kg]) vs time (x-axis [s]) for loop trajectory . . . 20
8 Mass estimation (y-axis [kg]) vs time (x-axis [s]) for zig-zag trajectory . . . 21
9 Mass estimation (y-axis [kg]) vs time (x-axis [s]) for square trajectory . . . 22
10 Mass estimation (y-axis [kg]) vs time (x-axis [s]) for random trajectory . . . 23
11 Mass estimation (y-axis [kg]) vs time (x-axis [s]) for lemniscate trajectory . . . 24
12 Results for lemniscate trajectory . . . 25
13 Offline ELM with lemniscate trajectory for (a) 1^{th}, (b) 2^{nd}, (c) 3^{rd}, (d) 4^{th}, and (e) 5^{th} type of mass variation. Vertical axis represents the mass variation in kilogram. . . . 27
14 Offline ELM with random trajectory for (a) 1^{th}, (b) 2^{nd}, (c) 3^{rd}, (d) 4^{th}, and (e) 5^{th} type of mass variation. Vertical axis represents the mass variation in kilogram. . . . 28
15 Offline ELM with zigzag trajectory for (a) 1^{th}, (b) 2^{nd}, (c) 3^{rd}, (d) 4^{th}, and (e) 5^{th} type of mass variation. Vertical axis represents the mass variation in kilogram. . . . 29
16 OSELM with lemniscate trajectory for (a) 1^{th}, (b) 2^{nd}, (c) 3^{rd}, (d) 4^{th}, and (e) 5^{th} type of mass variation. Vertical axis represents the mass variation in kilogram. . . . 30
17 OSELM with zigzag trajectory for (a) 1^{th}, (b) 2^{nd}, (c) 3^{rd}, (d) 4^{th}, and (e) 5^{th} type of mass variation. Vertical axis represents the mass variation in kilogram. . . . 31
18 Environmental interference, taken from [1]. . . . 32
19 Forces of UAV. . . . 33
20 Left: Picture of experimental platform, taken from [2]; Right: Abstract diagram of experimental platform. . . . 35
21 Left: External force estimation results for sequence 17; Right: External force estimation results for sequence 18. . . . 36
22 Left: Different views of the aligned trajectories for sequence 17; Right: Different views of the aligned trajectories for sequence 18. . . . 36

List of Tables

1 Offline ELM and RELM results for $\lambda = 0.0000001$ and 5 randomly selected datasets. 26
2 RELM for various λ for $\tilde{N} = 500$ and 5 randomly selected datasets. 26
3 RELM for various number training datasets with $\tilde{N} = 500$ and $\lambda = 0.00001$. 26
4 External force estimation RMSE (m/s^2) of different algorithms (VID-Fusion, Ours) was evaluated with different sequences of VID-Dataset. 36
5 ATE (m) of different algorithms (VID-Fusion, Ours) was evaluated with different sequences of VID-Dataset. 36

Acronyms

ANFIS ..... Adaptive Network-based Fuzzy Inference System

ARMAX ..... Auto-Regressive Moving Average with Exogenous Input

ARX ..... Auto-regressive Exogenous Input

EKF ..... Extended Kalman Filter

ELM ..... Extreme Learning Machine

IMU ..... Inertial Measurement Unit

MRS ..... Multi-Robot Systems

MSCKF ..... Multi-State Constraint Kalman Filter

OSELM ..... Online Sequential Extreme Learning Machine

RELM ..... Regularized Extreme Learning Machine

SESAME ..... Secure and Safe Multi-Robot Systems

SWLS ..... Sliding Window Least Squares

UAV ..... Unmanned Aerial Vehicle

UKF ..... Unscented Kalman Filter

Nomenclature

N	Number of Samples
x_i	i^{th} measurement of states vector x
t_i	i^{th} target value
$g(.)$	Activation function
β	Network weights vector
$\tilde{N}$	Number of hidden nodes
λ	Regularization parameter

Executive Summary

In this report, several model identification techniques are developed to be used at run-time by a robot to identify the dynamic and kinematic model of itself or other robots in Multi-Robot Systems (MRS). This reflects the robot capabilities, which improves the transparency, i.e., what a robot can do. This deliverable will present a comparison of different model identification techniques ranging from model-based approaches like those one based on dynamic observers (EKF,UKF) or mathematical fitting (SWLS) to data-driven approaches (ELM, OSELM). It should be noted that these techniques are to be integrated into Tasks 2.3 and 2.4, i.e., to estimate the state of an MRS at run-time and generate and track feasible trajectories, respectively. As a result, for instance, the robot mispositioning is avoided and, thus, the safety is not compromised.

The model identification methods are developed in a generic manner to be applicable to any dynamic system. Furthermore, the developed methods range from model-based approaches to data-driven ones, considering the different use-case partners with different practical restrictions. More importantly, the on/offline variations are derived, considering various types of operations. Finally, the methods are numerically evaluated on a high-fidelity Unmanned Aerial Vehicle (UAV) model with time-varying mass. Finally, the concluding remarks are presented, addressing the applicability of this example to the other types of MRS operational scenarios.

1 Introduction

1.1 Overview

In this report, we present and document several model identification techniques for run-time detection of dynamic and kinematic models of robots developed in the context of Task 2.2. The original purpose is to have these methods documented and analyze different aspects, aiming for a wide range of applicability to different use-case partners. More importantly, the developed methods are presented in a generic manner to cover different robot models. Also, the methods are derived mathematically, and the implementation algorithms are provided.

In terms of the technical development of the SESAME project each robot of a Multi-Robot Systems (MRS) should be mathematically modeled with the coupled dynamics to tackle planning, tracking, and estimation tasks. More importantly, considering the computational burden, the model must be as simple as possible, e.g., linear model. However, regardless of the inexpensive computational burden of simple models, this imposes model uncertainty and mismatch in practice. Accordingly, "*kinematics of the MRS given the robot types as well as the dynamics of the robot, with identification of the model parameters*" is needed. Furthermore, as mentioned in the proposal, it is required to develop identification techniques, which can be used at run-time by a robot to identify the dynamic and kinematic model of itself or other robots in the team, to reflect capabilities and capture changes in the kinematic and dynamic systems while operating. Regarding the project objective, we use a simple model that can be extended to a more complex one and focus on the higher-level task and corresponding requirements.

1.2 Intentions

This document is intended to be used as a guideline for model identification of robotic systems. In the context of the SESAME project, the document will contribute to the SESAME Knowledge Base to improve the robustness and the capabilities of the MRS. Proper identification of the robotic system will increase the system's safety and security. It will also improve the overall performance of the robot and its capabilities to work close to its optimal point as a consequence of a better self-awareness. The model identification component will be adequately integrated with the rest of the task within the WP2 to exploit this capability in real-time, but also the rest of the components of SESAME could be fed with the model information if needed.

1.2.1 Benefits to the project

Model identification techniques are to be used by the MRS to identify the robot kinematics and dynamics, either online or offline. The necessity of various methods to be developed stems from different use-case partners' needs, with different robot types, to be carefully addressed. Accordingly, the versatile and generic presentation of the methods enables different partners to choose a suitable technique considering the corresponding needs and limitations. More importantly, the mathematical derivation of each method makes any potential modification an easy task.

We seek novel as well as practical model identification techniques to be used and implemented in real-time to identify variable robot dynamics. This presents the dynamic model with reflecting changes in the model while operating. For instance, this happens in the vineyard, and the battery innovation center SESAME use cases. In both, the system changes its mass and consequently its dynamic characteristics when it is spraying the pesticide (mass loss) or accomplishing manipulation activities (mass gain and loss), respectively. Furthermore, model information is essential in cases where the model-based approaches are adopted, e.g., trajectory planning, control design and system monitoring. Therefore, this technique will, eventually, feed the model-based trajectory planning and tracking and the joint sensor fusion developed along with the SESAME project.

1.2.2 Benefits to the industry

The most significant benefit of the presented model identification techniques to the robotic industry can be entitled as the "*fast and accurate identification of the robot kinematics and dynamics, with time-varying parameters*". This, consequently, enhances the navigation performance, as it requires information on the model. Moreover, accurate model identification guarantees reliable system monitoring schemes. This is needed because much control, navigation, monitoring, and operation methods are model-based. However, these methods are usually designed based on the nominal model of robots and, thus, fail to function satisfactorily if the robot model is time-varying. Therefore, the model is required to be identified constantly and precisely. For example, consider a spraying UAV, whose mass is time-dependent and variable. On the other hand, the mass considerably affects the dynamic model of UAVs. So, it is required to have the mass variation identified.

1.3 Outcome

Considering the SESAME objectives, the final outcomes that this report delivers are summarised below.

i) Generic model: This is used to let the designed methods be easily modifiable for different applications.

ii) Variety of methods: Different model identification methods are designed and compared, which might be used in different scenarios.

iii) Mathematical derivation: The designed methods are derived mathematically, which eases the application on different use cases.

iv) Algorithm: The model identification methods are presented in an algorithmic manner.

v) Case-study analysis: The methods are implemented numerically to a case-study application, and the results are analyzed. Also, the implementation codes are presented.

1.4 Deliverable structure

This report initially presents the rationale behind the need for model identification in Section 2. Moreover, a brief taxonomy of model identification methods is given in Section 3. Accordingly, satiable model identification methods are derived, considering the generic system representation, in Section 4. On this basis, the designed procedure is given for all the methods. Then, as a case of study, a UAV model is elaborated in Section 5 with time-varying mass. It is aimed to estimate the mass variation using the presented methods. The results are given and analyzed in Section 6 to investigate the accuracy of each model identification method and its computational cost. Finally, the concluding remarks are summarized in Section 7.

2 Rationale

In robotics, state-of-the-art controllers, trajectory planners, and sensor fusion techniques use the kinematic and dynamic information of the robot to optimize its behavior and maximize its performance (see Figure 1). This figure shows the classical model-based architecture used in a robotics. In which, the controller, the sensor fusion and the trajectory planner consider the dynamic model of the system to optimize system performance. This is a widely extended architecture, for instance, Model Predictive Control (MPC) has gained broad interest in the robotics community as a tool for motion control of complex and dynamic systems. The ability to deal with nonlinearities and constraints has popularized this technique for many robotic applications, such as quadrotor control [3], [4], autonomous racing [5], and legged locomotion [6], [7], [8]. These strategies enable the control action optimization for a given cost function over a time horizon. MPC approaches are usually combined with a trajectory generation phase. In [9], the authors studied the problem of finding dynamically feasible trajectories and controllers that drive a quadrotor to the desired state in state space. A similar problem was reformulated in [10]. In this deliverable, the idea is to solve the control and the planning problems together using the model information.

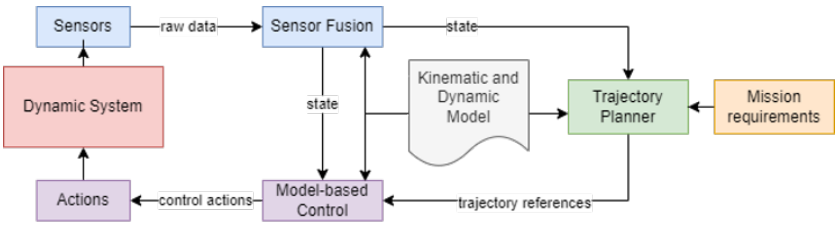


Figure 1: Model-based approach scheme.

However, although model-based techniques provide optimal results when the used model is accurate, those techniques could be very unsafe, unstable, and, for sure, not optimal if the used model is not properly formulated, or inaccurate. In more complex systems or system that changes their mass or topology in the middle of the operation, such as the spraying drone of Aero41, a robotic arm when grasping or releasing an object, a morphing robot or even an aerial manipulator, the kinematic and the dynamic characteristics could change during the operation. In these cases, an online model identification that can capture those changes in the dynamic model is needed to maintain operational safety.

In this deliverable, different model identification approaches have been adapted or reformulated to capture this specific phenomenon. In order to generalize the different methodologies, it is essential to remark that the presented techniques as comprehensive and generic as possible, considering a variety of dynamic systems. The critical aspect is to identify which is the level of abstraction required for the specific use case. This will mainly depend on the robotic system and the inputs/outputs available. In some cases, we could model the system with differential equations, for instance, following the classical equations of a rigid solid, or assume that the system can be modeled as a first-order system, a second-order system, or others. This concept allows the generalization of the model identification component (see Figure 2).

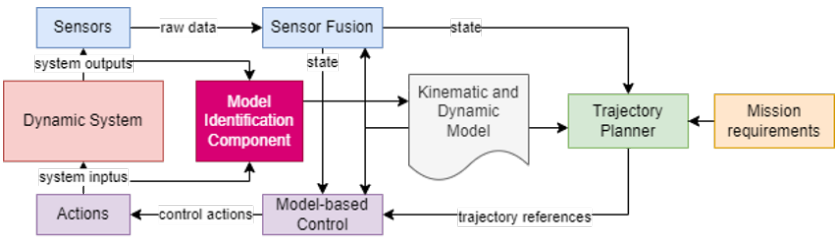


Figure 2: Model-based approach scheme with the online model identification component.

This document evaluates different model identification techniques that have been adapted to capture changes in the dynamics online, calculating their maximum errors and their computational costs at run-time. It aims to be used as a guideline by the robotics owners of SESAME to improve operational safety without losing the optimality of model-based approaches.

3 Methodology

The problem of model identification can be stated as identifying unknown parameters of the system or the system model, given several samples. When *a priori* knowledge on system characteristics is available, then the identification procedure can be enhanced. This knowledge acts as a set of constraints of solution space, in which the problem in this new space is more tractable. This approach is successfully applied to estimate algebraic and dynamic affine systems from noisy samples by using reasonable assumptions on the noise properties and the sampling process.

Dynamic systems identification depends on techniques developed by mathematicians, considering the mandatory association of a unique model to every available dataset, contaminated by noise, by introducing additional information unrelated to the data, i.e. of prejudices. While such prejudices can be convenient in some practical cases, it is, of course, very important to evaluate the family of solutions that can be found without introducing prejudices or, at least, by introducing only mild ones.

To tackle identification, we generally have two approaches, namely, model-based and data-driven approaches. Model-based approaches can also be subdivided into two sub-categories. On the one hand, a nominal model derived from physics equations could be assumed as the equations that govern the state of the system. On the other hand, system engineers usually use simplified nominal models like a first or a second-order model with a time delay to represent the system dynamics with a higher level of abstraction. In both cases, we can use the assumed dynamic model to observe the dynamics of the system to infer the dynamic variables or use a classical mathematical fitting technique.

Indeed, a perfect model identification technique does not exist. However, although there might be some discrepancies between the identified model response and the actual one, these techniques will allow us to have a preliminary mathematical nominal model that behaves similar to our system. The well-known approaches are a step-response approximation, extended and unscented Kalman filter, and various least square methods [11].

In contrast, in data-driven approaches, such as those on the basis of machine learning algorithms, the mentioned issues are avoided, just using the measurement data and fitting a regression model. The corresponding well-known methods include Auto-regressive Exogenous Input (ARX), Auto-Regressive Moving Average with Exogenous Input (ARMAX) [12], and Frisch Scheme for noisy measurement [13]. These methods generally utilize the least square minimization and involve the matrix inversion operation, which might be numerically expensive for the system with a large number of states and measurements. Moreover, there have been some novel algorithms developed in the last decade to identify highly nonlinear systems. For instance, Extreme Learning Machine (ELM) [14] is an approach for real-time fast prototyping of dynamic systems. The speed of identification is a main characteristic of this approach. However, it might suffer from a heavy computational burden. Furthermore, the training data is required to cover almost the whole operational region of the system to have accurate identification. On the other hand, some other approaches, e.g., Fuzzy Takagi Sugano [15] and Adaptive Neuro-Fuzzy Inference System (ANFIS) [16] have shown salient features. The complexity of algorithms is one main drawback of these approaches.

To this end, in order to capture different solutions for use-cases with different requirements, we present several approaches covering both model-based and data-driven ones for online system identification problems in a generic way for a generic dynamic system. The results are compared on a case study, that is, the estimation of the time-varying mass of a UAV. Finally, all the results are discussed according to the performance of the different algorithms in each specific situation.

4 Generic System Identification Theory

4.1 Notation

A generic system is usually be formulated with differential equations as:

$$\dot{x} = f\left(x, u, p\right) + n_x \tag{1}$$

The measured variables follow the equation as:

$$y = h\left(x\right) + n_y \tag{2}$$

In previous equations, x is the state vector, u is control input, p is an unknown time-varying parameter to be identified, y is the measured variables, f represents the dynamic of the system, h is the observation model, and n_x and n_y represent zero-mean Gaussian noise.

In general, f is a function of the kinematic and dynamic characteristics of the system. However, in (1), f is evidently a function of the parameter p, which varies with time. This, in turn, makes the implementation of the system dynamics (1) impossible. Consequently, the model-based techniques, e.g., controllers and planners, require the information of the variation of p in their derivations to perform satisfactorily. Otherwise, in those use-cases in which the system changes its dynamic properties while operating, the safety of the system could be compromised, and the assumed optimal process will not be optimal at all.

4.2 Model identification techniques

This section presents the formulation of suitable model identification techniques, categorized in Figure 3.

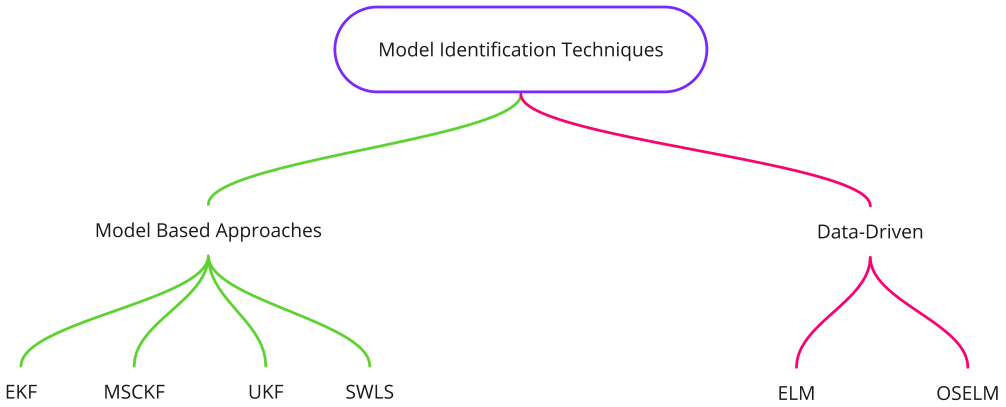


Figure 3: Model identification techniques studied in this deliverable. EKF: Extended Kalman Filter; MSCKF: Multi-State Constraint Kalman Filter; UKF: Unscented Kalman Filter; SWLS: Sliding Window Least Square method; ELM: Extreme Learning Machine; OSELM: Online Sequential Extreme Learning Machine.

An Extended Kalman Filter (EKF), a Multi-State Constraint Kalman Filter (MSCKF), and an Unscented Kalman Filter (UKF) have been implemented to evaluate how good are the classical observers capturing changes in the dynamic. To do so, we have included the dynamic variables as unknowns in the state vector as indirectly observable variables. Furthermore, the sliding Window Least Square (SWLS) method has also been adapted to evaluate the performance of this purely mathematical fitting method to capture changes in the

dynamic variables. Last, two data-driven approaches (Extreme Learning Machine (ELM) and Online Sequential Extreme Learning Machine (OSELM)) have been trained with the same purpose. The final objective is to benchmark them and evaluate how these techniques perform the model identification task in different contexts to help future robotics owners and developers decide which techniques they should use depending on their use case and the behavior expected in their robots.

4.2.1 Extended Kalman Filter (EKF)

In estimation theory, the Extended Kalman Filter (EKF) is the nonlinear version of the Kalman filter, which linearizes about an estimate of the current mean and covariance. Kalman filtering is an algorithm that provides estimates of some unknown variables given the measurements observed over time. It has been demonstrating its usefulness in various applications. This filter has a relatively simple form and requires small computational power.This filter can also indirectly estimate observable variables over time. The main advantage of this filter is the minimization of uncertainty effects on the estimated variables.

This filter can be adapted to observe the dynamics of the system. To do so, we assume that we obtain the state vector x_k at the time k and the corresponding covariance P_k. In the state vector, we could include indirectly observable variables like the mass to estimate their values.

In the standard estimation process, the Kalman Filter takes the sensor measurements and a motion model as input to calculate the state of the system. In the EKF's case, the process evaluates the variables in the model, so the EKF takes the measurements and control actions as the input to predict the system's behavior. In the next step-time, the algorithm will compare the prediction with the real input to recalculate the dynamic variables and their uncertainty. In this way, it is able to estimate the dynamic of the system online.

So, according to the control input u_k and the assumed dynamic equations (1), we could get the state vector x_{k+1} at the time $k+1$ propagating the model as follows:

$$\begin{aligned} x &\leftarrow x + \dot{x}dt \\ x_{k+1} &= x_k + f\left(x_k, u_k\right) dt \end{aligned} \tag{3}$$

Next, we need to derive the error state dynamic model:

$$\begin{aligned} J &= \partial f / \partial x \\ \delta\dot{x} &= J\delta x \end{aligned} \tag{4}$$

J is the jacobian matrix of f with respect to x. After obtaining the matrix J , we can predict the covariance P_{k+1} at the time $k+1$:

$$\begin{aligned} F &= \exp\left(Jdt\right) \\ P_{k+1} &= FP_kF^T + Qdt \end{aligned} \tag{5}$$

Where, Q is the preset covariance of state noise.

$$Q = \begin{bmatrix} \sigma_{x_0}^2 & & \\ & \ddots & \\ & & \sigma_{x_{\dim(x)-1}}^2 \end{bmatrix} \tag{6}$$

The i-th element of the main diagonal corresponds to the noise variance of the i-th state. All non-diagonal elements are 0.

After receiving the measurement, we need to update the state vector and covariance:

$$
\begin{aligned}
&\hat{y} = h\left(x\right) \\
&H = \partial h / \partial x \\
&S = HPH^T + R \\
&K = PH^T S^{-1} \\
&dx = K\left(y - \hat{y}\right) \\
&x \leftarrow x + dx \\
&P \leftarrow \left(I - KH\right) P \left(I - KH\right)^T + KRK^T
\end{aligned} \tag{7}
$$

H is the jacobian matrix of h with respect to x. R is the preset covariance of measurement noise.

$$
R = \begin{bmatrix} \sigma_{y_0}^2 & & \\ & \ddots & \\ & & \sigma_{y_{\dim(y)-1}}^2 \end{bmatrix} \tag{8}
$$

The i-th element of the main diagonal corresponds to the noise variance of the i-th measurement. All non-diagonal elements are 0.

4.2.2 Multi-State Constraint Kalman Filter: MSCKF

The idea of this part draws lessons from [17]. First, define the following state:

$$
\begin{aligned}
x &= \begin{pmatrix} x_I & x_C \end{pmatrix} \\
x_C &= \begin{pmatrix} x_{c_1} & \cdots & x_{c_N} \end{pmatrix}
\end{aligned} \tag{9}
$$

Where, x_I represents the head state,N represents the window size. We get x_{c_i} through the clone of x_I at different times when we get measurements, and the corresponding covariance is:

$$
P = \begin{pmatrix} P_{II} & P_{IC} \\ P_{IC}^T & P_{CC} \end{pmatrix} \tag{10}
$$

P is the covariance matrix of x. P_{II} is the variance of x_I. P_{CC} is the variance of x_C. P_{IC} is the covariance between x_I and x_C. How to get P will be further described below. Similar to EKF, according to the control input u_k and dynamic model, we can get the state vector x_{k+1} at the time $k+1$:

$$
x_I \leftarrow x_I + \dot{x}_I dt \tag{11}
$$

Covariance P_{k+1} at the time $k+1$:

$$
\begin{aligned}
&F = \exp\left(Jdt\right) \\
&P_{II} \leftarrow F P_{II} F^T + Qdt \\
&P \leftarrow \begin{pmatrix} P_{II} & F P_{IC} \\ P_{IC}^T F^T & P_{CC} \end{pmatrix}
\end{aligned} \tag{12}
$$

where, Q is the preset covariance of state noise. J and Q have the same meaning as in EKF (see Section 4.2.1).

When receiving the measurement, we can do state augmentation, and the covariance needs to change accordingly:

$$
P = \begin{pmatrix} I \\ \frac{\partial x_c}{\partial x} \end{pmatrix} P \begin{pmatrix} I \\ \frac{\partial x_c}{\partial x} \end{pmatrix}^T \tag{13}
$$

Measurement update is similar to EKF, except that we need to select the measurement in the whole window.

$$
\begin{aligned}
&\hat{y} = h\left(x\right) \\
&H = \partial h / \partial x \\
&S = HPH^T + R \\
&K = PH^T S^{-1} \\
&dx = K\left(y - \hat{y}\right) \\
&x \leftarrow x + dx \\
&P \leftarrow \left(I - KH\right) P \left(I - KH\right)^T + KRK^T
\end{aligned} \tag{14}
$$

where, R is the measurement noise covariance of the whole window. H and R have the same meaning as in Section 4.2.1.

4.2.3 Unscented Kalman Filter: UKF

Here we refer to [18], [19]. We assume that the state vector, x at the initial time is μ and the corresponding covariance is $\sum$.

$$x = \begin{pmatrix} \mu & \mu + \gamma\sqrt{\sum} & \mu - \gamma\sqrt{\sum} \end{pmatrix} \tag{15}$$

γ is the tuning parameter. According to the control input and dynamic model, we predict the state vector and covariance:

$$\begin{aligned} x_i &\leftarrow x_i + \dot{x}_i dt \\ \mu &\leftarrow \sum_{i=0}^{2l} w_m x_i \\ \sum &\leftarrow \sum_{i=0}^{2l} (x_i - \mu) w_c (x_i - \mu)^T + Q dt \end{aligned} \tag{16}$$

Subscript i indicates column number. w_m and w_c are the corresponding weights. l is the state dimension. And Q is preset covariance of state noise. It has the same meaning as EKF part.

After receiving the measurement, state and covariance update are as follows:

$$\begin{aligned} x &= \begin{pmatrix} \mu & \mu + \gamma\sqrt{\sum} & \mu - \gamma\sqrt{\sum} \end{pmatrix} \\ \hat{y} &= h(x) \\ \mu_y &= \hat{y} w_m \\ S &= \sum_{i=0}^{2l} (\hat{y}_i - \mu_y) w_c (\hat{y}_i - \mu_y)^T + R \\ K &= \left(\sum_{i=0}^{2l} (x_i - \mu) w_c (\hat{y}_i - \mu_y)^T \right) S^{-1} \\ d\mu &= K(y - \mu_y) \\ \mu &\leftarrow \mu + d\mu \\ \sum &\leftarrow \sum - KSK^T \end{aligned} \tag{17}$$

Where, R is the preset covariance of measurement noise. It has the same meaning as EKF part.

4.2.4 Sliding Window Least Square: SWLS

The method of least squares is a standard approach in regression analysis to approximate the solution of over-determined systems by minimizing the sum of the squares of the residuals made in the results of each individual equation. The most important application is in data fitting. Applied to the online identification of dynamic systems, the algorithm should consider the input and the output of several step-times to avoid generating outliers due to the noise or spontaneous bad measurements of the sensors. Ideally, the least square method would use all the data collected during the system operation. However, in real-time applications this might not be applicable for two reasons. First, the amount of data accumulated could saturate the computational resources available in the robot. Second, if the system changes its dynamic variables online, the information collected in the past might not represent the robot's state in a specific step-time. For this reason, this time, we applied the Sliding Window Least Square method, in which the LS algorithm is applied with the data of the last n step-times. In this way, we create a sliding window of data that is used within the algorithm. Similar approach was previously presented in [1], [20].

The optimization variables in the sliding window are defined as follows:

$$\chi = \begin{pmatrix} x_0 & x_1 & \cdots & x_n \end{pmatrix} \tag{18}$$

Where, n is the size of the sliding window to be tuned offline according to the system. Between the two states of the sliding window, we will integrate the control input to establish their constraints. In order to avoid repeated integration during optimization iteration, we define some following pre-integration items, such as $\alpha_{b_{k+1}}^{b_k}$, $\beta_{b_{k+1}}^{b_k}$, and $\gamma_{b_{k+1}}^{b_k}$.
Where, k and $k+1$ represent the timestamp of adjacent states in the sliding window. b represents the body frame.

Next, we need to derive the error state dynamic model:

$$\begin{aligned} \delta z &= \begin{pmatrix} \delta\alpha & \delta\beta & \delta\gamma \end{pmatrix} \\ \delta z_{i+1}^{b_k} &= F\delta z_i^{b_k} + Gn \\ F &= \partial z_{i+1}^{b_k} \Big/ \partial z_i^{b_k} \\ G &= \partial z_{i+1}^{b_k} \Big/ \partial n \end{aligned} \tag{19}$$

Then we can propagate the covariance of pre-integration measurement:

$$P_{i+1}^{b_k} = FP_i^{b_k}F^T + GQG^T \tag{20}$$

The final optimization problem is:

$$\min_{\chi} \left\{ \|r_{prior}\|^2 + \sum \left\| r_P^k\left(x_k\right) \right\|_{W_P^k}^2 + \sum \left\| r_D^k \begin{pmatrix} x_k & x_{k+1} \end{pmatrix} \right\|_{W_D^k}^2 \right\} \tag{21}$$

Where, r_{prior} represents the prior constraint, $r_P^k\left(x_k\right)$ represents measurement residual, $r_D^k \begin{pmatrix} x_k & x_{k+1} \end{pmatrix}$ represents dynamic residual. W_P^k and W_D^k are weight matrix, which can be regarded as the inverse of covariance matrix.

The expression of $r_P^k\left(x_k\right)$ is straightforward:

$$r_P^k\left(x_k\right) = y - \hat{y} = y - h\left(x_k\right) \tag{22}$$

The expression of $r_D^k \begin{pmatrix} x_k & x_{k+1} \end{pmatrix}$ is as follows:

$$r_D^k \begin{pmatrix} x_k & x_{k+1} \end{pmatrix} = \begin{bmatrix} \alpha_{b_{k+1}}^{b_k} - \hat{\alpha}_{b_{k+1}}^{b_k} \\ \beta_{b_{k+1}}^{b_k} - \hat{\beta}_{b_{k+1}}^{b_k} \\ \gamma_{b_{k+1}}^{b_k} - \hat{\gamma}_{b_{k+1}}^{b_k} \end{bmatrix} \tag{23}$$

The least square optimization problem is solved by Ceres[1].

4.2.5 Data Driven Approach

Belonging to data-driven approaches, feed-forward neural networks have shown significant potential in many fields to approximate unknown nonlinear functions using samples and provide models for various dynamic systems that are difficult to handle using classical parametric techniques. Mathematically speaking, research on the approximation capabilities has been focused on two aspects, i.e., universal approximation on compact input sets and, approximation in a finite set of training samples.

In real applications, the neural networks are trained in a finite training set. For function approximation in such a set, it has been shown that a single-hidden layer feed-forward neural network with at most N hidden nodes and any nonlinear activation function can exactly learn N distinct observations [14].

Traditionally, the parameters of networks need to be tuned, and, thus, there is a dependency between different layers of parameters, i.e., weights and biases. For past decades, the gradient descent-based learning method has been used in various networks. It has been shown that the learning speed of feed-forward neural networks is generally slow, which causes a significant issue for practical application [21]. The reasons are

[1] http://ceres-solver.org/

i) the slow gradient-based learning algorithms, and

ii) the parameters are tuned iteratively.

Therefore, it might take several hours, several days, and even more time to train neural networks using traditional methods. Moreover, the gradient-based learning method may converge to local minima, and different steps may be required to resolve this issue. This further slows down the learning process.

Unlike these conventional approaches, ELM using single-hidden layer feed-forward neural networks chooses the network parameters randomly and then analytically determines the output weights. This stems from some simulation results on artificial and natural large applications in our work, i.e., it is not necessarily required to adjust the input weights and biases initially in applications [21]. In [14], it has been shown that a network of N hidden nodes with randomly selected input weights and biases can exactly learn N different samples if activation functions are infinitely differentiable. After the random selection of input weights and the hidden layer biases, the network can be considered as a linear system and the output weights can be analytically determined through a simple generalized inverse operation. Theoretically, ELM has superior generalization performance (slightest training error) with fast learning speed. This provides the ability of fast online prototyping in terms of online model identification.

4.2.5.1 Offline ELM To present the offline ELM structure, assume there are N arbitrary distinct samples (x_i, t_i), for $i = 1, ..., N$, where $x_i = [x_{i1}, \ldots, x_{in}]^T \in \mathbb{R}^n$ is the available inputs vector, $t_i = [t_{i1}, \ldots, t_{im}]^T \in \mathbb{R}^m$ is the target values vector. These samples are available offline for training the network. Also, the network has $\tilde{N}$ nodes with activation function of $g(.) : \mathbb{R} \to \mathbb{R}$ which is infinitely differentiable. The ELM, modelled as a single layer feed-forward network, is then constructed as

$$\sum_{i=1}^{\tilde{N}} \beta_i g(w_i.x_j + b_i) = o_j, \tag{24}$$

for $j = 1, ..., N$, where $w_i = [w_{i1}, \ldots, w_{in}]^T \in \mathbb{R}^n$ and $\beta_i = [\beta_{i1}, \ldots, \beta_{im}]^T \in \mathbb{R}^m$ are the weight vectors, connecting i^{th} hidden node and input nodes. b_i the bias of i^{th} hidden node. $w_i.x_j$ denotes the inner product of w_i and x_j. Finally, o_j is the output of the network. Theoretically, it is shown that there exist β_i, w_i and b_i such that $\sum_{i=1}^{\tilde{N}} \beta_i g_i(w_i.x_j + b_i) = t_j$, for $j = 1, ..., N$. These N equations can be encapsulated as

$$H\beta = T, \tag{25}$$

where,

$$\begin{aligned} H(w_1, \ldots, w_{\tilde{N}}, b_1, \ldots, b_{\tilde{N}}, x_1, \ldots, x_N) &= \begin{bmatrix} g(w_1.x_1 + b_1) & \cdots & (w_{\tilde{N}}.x_1 + b_{\tilde{N}}) \\ \vdots & \ddots & \vdots \\ g(w_1.x_N + b_1) & \cdots & (w_{\tilde{N}}.x_N + b_{\tilde{N}}) \end{bmatrix}_{N \times \tilde{N}}, \\ \beta &= \begin{bmatrix} \beta_1^T \\ \vdots \\ \beta_{\tilde{N}}^T \end{bmatrix}_{\tilde{N} \times m}, \\ T &= \begin{bmatrix} t_1^T \\ \vdots \\ t_N^T \end{bmatrix}_{N \times m}. \end{aligned} \tag{26}$$

As $g(.)$ is infinitely differentiable, it can be proven that required number of hidden nodes is $\tilde{N} \leq N$. then, if we choose w_i and b_i are selected randomly for any intervals of $\mathbb{R}^n$ and $\mathbb{R}$, respectively, according to any

continuous probability distribution, then with the probability of one, the hidden layer output matrix H of the network is invertible and $||H\beta - T|| = 0$. Also, given any small positive value ϵ, there exists $\tilde{N} \leq N$ such that for N arbitrary distinct samples (x_i, t_i), according to any continuous probability distribution, then with probability one, $||H_{N\times\tilde{N}}\beta_{\tilde{N}\times m} - T_{N\times m}|| < \epsilon$ [21]. Accordingly, the smallest norm least-squares solution of ELM network, given N arbitrary distinct samples (x_i, t_i) for randomly generated w_i and b_i, is obtained as,

$$\hat{\beta} = H^{\dagger}T, \tag{27}$$

where, $H^{\dagger} = (H^T H)^{-1}H^T$ is Moore–Penrose generalized inverse of matrix H. Therefore, the ELM algorithm with offline training is presented in Algorithm 1.

Algorithm 1 Offline ELM

Require: Given a training set $S = \{(x_i, t_i)|x_i \in \mathbb{R}^n, t_i \in \mathbb{R}^m, i = 1, ..., N\}$,

Step 1: Initialization

- Select activation function $g(.)$, and hidden node number $\tilde{N}$.
- Randomly assign w_i and b_i, for $i = 1, ..., \tilde{N}$.

Step 2: Training

- Calculate the hidden layer output matrix H, using (26).
- Calculate the output weight $\hat{\beta}$, using (27).

Step 3: Implementation

- Implement the obtained $\hat{\beta}$ and selected w_i and b_i, for $i = 1, ..., \tilde{N}$, online, using (25)

Since ELM adopts the square loss function in Algorithm 1, this means that it tends to force the margins of all the training samples exactly equaling one from the perspective of margin learning theory, which is unreasonable to some extent. Through solving a series of Regularized ELMs (RELMs). According to Bartlett's conclusion [22], the neural networks with the smaller norm of weights tend to suggest better generalization performance. To realize the weight decay, based on Tikhonov regularization theory [23], the RELM is obtained as following optimization problem [24].

$$\arg\min_{\beta} \frac{1}{2}||H_{N\times\tilde{N}}\beta_{\tilde{N}\times m} - T_{N\times m}||^2 + \frac{\lambda}{2}||\beta_{\tilde{N}\times m}||^2, \tag{28}$$

with scalar $\lambda > 0$ as the regularization parameter. The solution to the optimization problem (28) is obtained as

$$\hat{\beta} = (H^T H + \lambda I_{\tilde{N}})^{-1}H^T T. \tag{29}$$

Therefore, in Algorithm 1, Step 2, instead of using (27), one can use (29).

4.2.5.2 Online Sequential ELM As it is obvious in Algorithm 1, the whole training data needs to be available. However, in practice, this training data may be received gradually during the operation. Moreover, there might be some situations that the user wants to retrain the network online. Therefore, it is reasonable to modify this algorithm to be implementable online with the training process. As an alternative to Algorithm 1, an Online Sequential ELM (OSELM) can learn data one-by-one or chunk-by-chunk, i.e., a block of data, with fixed or varying chunk size, the output weights are analytically determined based on the sequentially arriving data. furthermore, OSELM discards the data for which the training has already been done. This reduces the chance of being over-fit to some data and, hence, improves the generalization.

Sequential learning algorithms have become popular for feed-forward networks nodes. Unlike other sequential learning algorithms which have many control parameters to be tuned, OSELM only requires the number of hidden nodes to be specified. Accordingly, OSELM is a versatile sequential learning algorithm in the following sense.

i) The training observations are sequentially (one-by-one or chunk-by-chunk with varying or fixed chunk length) presented to the learning algorithm.

ii) At any time, only the newly arrived single or chunk of observations (instead of the entire past data) are seen and learned.

iii) A single or a chunk of training observations is discarded as soon as the learning procedure for that particular (single or chunk of) observation(s) is completed.

iv) The learning algorithm has no prior knowledge as to how many training observations will be presented.

The batch ELM described previously assumes that all the training data (N samples) is available for training. However, in real applications, the training data may arrive chunk-by-chunk or one-by-one (a special case of chunk) and, hence, the batch ELM algorithm has to be modified for this case so as to make it online sequential.

Given a chunk of initial training set $S_0 = \{(x_i, t_i)\,|x_i \in \mathbb{R}^n, t_i \in \mathbb{R}^m, i = 1, ..., N_0\}$ and $N_0 \geq \tilde{N}$, if one considers using the batch ELM algorithm, one can obtain the trained weight as

$$\hat{\beta}_0 = K_0^{-1} H_0^T T_0, \tag{30}$$

where,

$$\begin{aligned} K_0 &= H_0^T H_0, \\ H_0 &= \begin{bmatrix} g\left(w_1.x_1 + b_1\right) & \cdots & \left(w_{\tilde{N}}.x_1 + b_{\tilde{N}}\right) \\ \vdots & \ddots & \vdots \\ g\left(w_1.x_{N_0} + b_1\right) & \cdots & \left(w_{\tilde{N}}.x_{N_0} + b_{\tilde{N}}\right) \end{bmatrix}_{N_0 \times \tilde{N}}, \\ T_0 &= \begin{bmatrix} t_1^T \\ \vdots \\ t_{N_0}^T \end{bmatrix}_{N_0 \times m}. \end{aligned} \tag{31}$$

Suppose now that we are given another chunk of data $S_1 = \{(x_i, t_i)\,|, i = N_0 + 1, ..., N_0 + N_1\}$ with N_1 represents new measurements or observations. If we use the traditional ELM algorithm, we discard the $\hat{\beta}_0$ and just find new weight $\hat{\beta}_1$, only using S_1. In contrast, in the OSELM, we obtain $\hat{\beta}_1$ using both $\hat{\beta}_0$ and S_1. This can be formulated as

$$\hat{\beta}_1 = \arg\min_{\beta} \left\| \begin{bmatrix} H_0 \\ H_1 \end{bmatrix} \beta - \begin{bmatrix} T_0 \\ T_1 \end{bmatrix} \right\|. \tag{32}$$

$$\begin{aligned} H_1 &= \begin{bmatrix} g\left(w_1.x_{N_0+1} + b_1\right) & \cdots & \left(w_{\tilde{N}}.x_{N_0+1} + b_{\tilde{N}}\right) \\ \vdots & \ddots & \vdots \\ g\left(w_1.x_{N_0+N_1} + b_1\right) & \cdots & \left(w_{\tilde{N}}.x_{N_0+N_1} + b_{\tilde{N}}\right) \end{bmatrix}_{N_1 \times \tilde{N}}, \\ T_1 &= \begin{bmatrix} t_{N_0+1}^T \\ \vdots \\ t_{N_0+N_1}^T \end{bmatrix}_{N_1 \times m}. \end{aligned}$$

The solution to this problem is

$$\hat{\beta}_1 = K_1^{-1} \begin{bmatrix} H_0 \\ H_1 \end{bmatrix}^T \begin{bmatrix} T_0 \\ T_1 \end{bmatrix}, \tag{33}$$

with $K_1 = \begin{bmatrix} H_0 \\ H_1 \end{bmatrix}^T \begin{bmatrix} H_0 \\ H_1 \end{bmatrix}$. It is easy to show that

$$\begin{aligned} K_1 &= K_0 + H_1^T H_1, \\ \begin{bmatrix} H_0 \\ H_1 \end{bmatrix}^T \begin{bmatrix} T_0 \\ T_1 \end{bmatrix} &= K_1 \hat{\beta}_0 - H_1^T H_1 \hat{\beta}_0 \; + H_1^T T_1. \end{aligned} \tag{34}$$

Accordingly,

$$\hat{\beta}_1 = \hat{\beta}_0 + K_1^{-1} H_1^T \left(T_1 - H_1 \hat{\beta}_0 \right). \tag{35}$$

Following similar procedure, for $(k+1)^{th}$ chunk of dataset, i.e. $S_{k+1} = \left\{ (x_i, t_i) \,|, i = \sum_{j=0}^{k} N_j + 1, ..., \sum_{j=0}^{k+1} N_j \right\}$, one can obtain that

$$\begin{aligned} K_{k+1} &= K_k + H_{k+1}^T H_{k+1}, \\ \hat{\beta}_{k+1} &= \hat{\beta}_k + K_{k+1}^{-1} H_{k+1}^T \left(T_{k+1} - H_{k+1} \hat{\beta}_k \right), \end{aligned} \tag{36}$$

where,

$$\begin{aligned} H_{k+1} &= \begin{bmatrix} g\left(w_1 . x_{\sum_{j=0}^{k} N_j + 1} + b_1 \right) & \cdots & \left(w_{\tilde{N}} . x_{\sum_{j=0}^{k} N_j + 1} + b_{\tilde{N}} \right) \\ \vdots & \ddots & \vdots \\ g\left(w_1 . x_{\sum_{j=0}^{k+1} N_j} + b_1 \right) & \cdots & \left(w_{\tilde{N}} . x_{\sum_{j=0}^{k+1} N_j} + b_{\tilde{N}} \right) \end{bmatrix}_{N_{k+1} \times \tilde{N}}, \\ T_{k+1} &= \begin{bmatrix} t^T_{\sum_{j=0}^{k} N_j + 1} \\ \vdots \\ t^T_{\sum_{j=0}^{k+1} N_j} \end{bmatrix}_{N_{k+1} \times m}. \end{aligned} \tag{37}$$

To avoid repetitive matrix inversion of K_{k+1}^{-1}, by using Woodbury matrix identity we obtain $K_{k+1}^{-1} = \left(K_k + H_{k+1}^T H_{k+1} \right)^{-1} = K_k^{-1} - K_k^{-1} H_{k+1}^T \left(I_{N_{k+1}} + H_{k+1} K_k^{-1} H_{k+1}^T \right)^{-1} H_{k+1} K_k^{-1}$. By setting $P_{k+1} = K_{k+1}^{-1}$,

$$\begin{aligned} P_{k+1} &= P_k - P_k H_{k+1}^T \left(I_{N_{k+1}} + H_{k+1} P_k H_{k+1}^T \right)^{-1} H_{k+1} P_k, \\ \hat{\beta}_{k+1} &= \hat{\beta}_k + P_{k+1} H_{k+1}^T \left(T_{k+1} - H_{k+1} \hat{\beta}_k \right). \end{aligned} \tag{38}$$

Now, the OSELM algorithm is obtained as follows.

Algorithm 2 OSELM

Require: dataset S_k, $k = 0, 1, \ldots, \bar{k}$, arrive sequentially. **Step 1: Initialization**

- Select activation function $g(.)$, and hidden node number $\tilde{N}$.
- Randomly assign w_i and b_i, for $i = 1, ..., \tilde{N}$.

Step 2: Training

- For S_0, compute the hidden layer output matrix H_0, using (31).
- Compute $\hat{\beta}_0 = P_0 H_0^T T_0$ with $P_0 = \left(H_0^T H_0 \right)^{-1}$.
- Use dataset $S_{k+1}, k = 0, 1, \ldots, \bar{k}$, and compute $\hat{\beta}_{k+1}$, using (38).

Step 3: Implementation

- Implement $\hat{\beta}_{\bar{k}+1}$, and selected w_i and b_i for $i = 0, 1, \ldots, \tilde{N}$, online, using (25).

5 System Identification Application: UAV simulation case

In order to compare the results of the presented model identification techniques, we will use a particular case of study that aims to cover the SESAME use cases in a generic way. In this case, the proposed system is a UAV that can be commanded in angular velocities and thrust independently (4 DoF). By default, this is the lower control loop accessible in the well-known and widely used DJI drones, and it is also a flight mode present in the standard open source autopilots, Arducopter and PX4. The mass of the system will not be constant during the flight (as it happens in the pesticide spraying operation). Additionally, initial errors have been introduced in the model to observe if the system is able to converge to the actual mass value. This system has been chosen due to its simplicity and its unique dependency on mass and inertia. The dynamic associated with the inertia can be omitted because we are taking the inputs prior to and after this dynamic.

First, the notation, the system's dynamic model, and the dataset generated are presented. Later, the capability to capture the changes and the result of each model identification technique are presented and discussed in Section 6.

5.1 Notation

Hereinafter, we will express the vector from A to B as r_{AB}. The coordinates of this vector in the coordinate system C is expressed as ${}_Cr_{AB}$. We use quaternions to represent the rotation of rigid bodies, which is a non-singular expression. For the detailed introduction and properties of quaternion, we refer to [25]. q_{BA} denote the attitude of a coordinate frame B with respect to frame A. The corresponding rotation matrix is $R\left(q_{BA}\right)$. The coordinate transformation is expressed as follows:

$$ {}_Br = R\left(q_{BA}\right) {}_Ar \tag{39} $$

According to quaternion algebra, We need to pay special attention to the addition of quaternion, $q+\delta$, and the subtraction between quaternion and quaternion $q_1 - q_2$, because they involve the operation between manifold and tangent space. Exponential mapping $\exp\left(\bullet\right)$ maps a vector in tangent space to quaternion. Logarithmic mapping does the opposite.

$$ \begin{aligned} q + \delta &:= q \otimes \exp\left(\tfrac{\delta}{2}\right) \\ q_1 - q_2 &:= 2\log\left(q_2^{-1} \otimes q_1\right) \end{aligned} \tag{40} $$

Here, δ is a vector in tangent space. $\otimes$ represents quaternion multiplication.

5.2 Nominal quadrotor dynamic model with variable mass

As was aforementioned, the dynamic system that we want to model is a UAV that can be commanded in angular rates and thrust (4 DoF). It is assumed that the estimator can provide us with the robot's full pose. There, let us define the world coordinate system as W, the geometric centre coordinate system as B, the centroid system as M, and the IMU coordinate system as I. Frame B, M and I are coincident. According to rigid body dynamics, we can get:

$$ \begin{aligned} {}_W\dot{r}_{WB} &= {}_Wv_B \\ {}_W\dot{v}_B &= \tfrac{1}{m} R\left(q_{WB}\right) T + g \\ \dot{q}_{WB} &= \tfrac{1}{2} q_{WB} \otimes \omega \end{aligned} \tag{41} $$

Where, T represents the thrust on the frame B and ω represents the angular velocity on the frame B. We don't know how the mass changes, so we assume:

$$ \dot{m} = 0 \tag{42} $$

Next, we define the following state vector x :

$$x = \begin{bmatrix} q \\ r \\ v \\ m \end{bmatrix} \tag{43}$$

Where q, r and v represents the attitude, the position and the linear velocities of the system respectively.

In order to simplify the notation, we delete the subscript in the dynamic equations, which will not affect the understanding. We can get the angular velocity ω directly from IMU. However, we cannot measure the real thrust, so we use the thrust command T^d instead of the measured thrust T, which means that we have the following assumptions:

$$T^d \approx T \tag{44}$$

Next, we define our control input u as:

$$u = \begin{bmatrix} T^d \\ \omega \end{bmatrix} \tag{45}$$

At the same time, it is assumed that we can obtain the following measurements from the state estimator:

$$y = \begin{bmatrix} q \\ r \\ v \end{bmatrix} \tag{46}$$

This is reasonable. For example, Visual-Inertial Odometry (VIO) system can output such high-frequency measurements. Finally, we get the following system equation and measurement equation.

$$\begin{aligned} \dot{x} = f(x,u) + n_x &= \begin{bmatrix} \frac{1}{2} q \otimes \omega \\ v \\ \frac{1}{m} R(q) T^d + g \\ 0 \end{bmatrix} + n_x \\ y = h(x) + n_y &= \begin{bmatrix} q \\ r \\ v \end{bmatrix} + n_y \end{aligned} \tag{47}$$

Where, n_x and n_y represent zero-mean Gaussian noise.

5.3 Dataset

The idea in this part of the deliverable was to generate a populated dataset to assess the performance of the different model identification techniques. Then, it was necessary to choose a control strategy to move the robot following a specific path or trajectories. To do so, we will follow the work previously presented in [26], [4]. The dataset generation framework is, therefore, as the one presented in the Figure 4. We need to set the robot dynamic model and reference trajectory with this architecture.

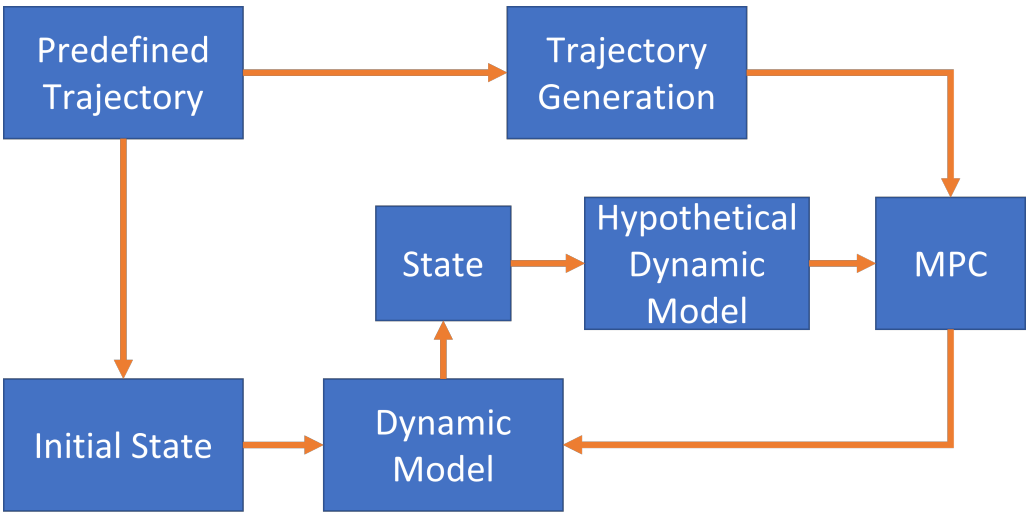


Figure 4: Dataset generation framework

The trajectory generation algorithm will further generate dynamically feasible trajectories according to the predefined trajectory. An MPC control approach generates the optimal control input according to the hypothetical dynamic model, measurement state and reference state. We use the actual dynamic model of the robot, the current state and control input to obtain the subsequent state of the robot.

We simulated five quadrotor flight trajectories, namely "random", "loop", "lemniscate", "zigzag", and "square", and five mass changes are simulated for each trajectory. To have a rich dataset, we have simulated each trajectory 50 times, making a total of 1250 simulations.

The reference trajectory and mass change curve we designed are shown in Figure 5.

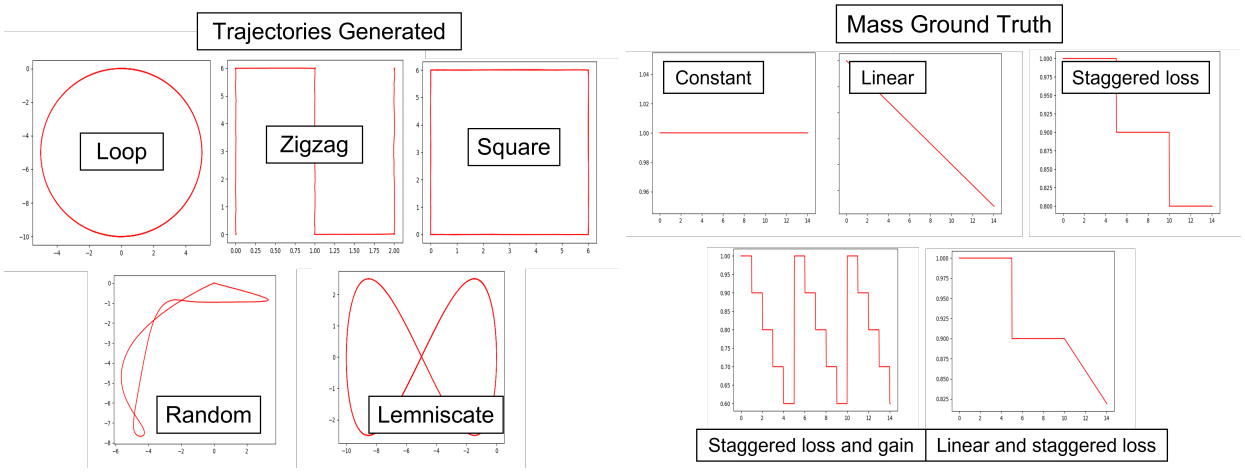


Figure 5: Left: Horizontal view (x-y) of the simulated trajectories; Right: Mass behaviour (y-axis) during the time (x-axis) simulated for each trajectory

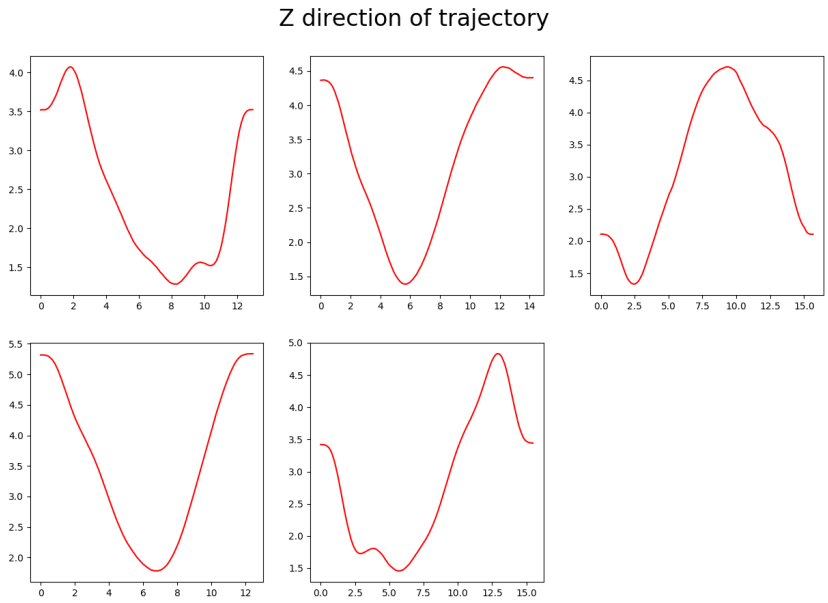


Figure 6: Z direction of the simulated random trajectories

6 Results

6.1 Implementation details

These algorithm are implemented with C++. All the experiments are conducted on a laptop computer with an Intel(R) Xeon(R) W-10855M CPU @ 2.80GHz, and 16 GB of RAM. Without any optimized acceleration.

6.2 Results of model-based methods

In order to compare the different techniques, We have calculated the mean value of the update time and the Root Mean Square Percentage Error (RMSPE) of the different algorithms for each dataset for 10 randomly selected experiments. RMSPE is defined as follows.

$$RMSPE = \sqrt{\frac{1}{n}\left(\sum_{i=1}^{n}\left(\frac{\hat{y}_i - y_i}{y_i}\right)^2\right)} \tag{48}$$

Where $\hat{y}_i$ is the estimated value and y_i is the groundtruth value.

In the tables found at the bottom of Figures 7-11, the first column under the algorithm name represents the update time in milliseconds, and the second column is RMSPE. Mass type 0-2 represent the mass change type from left to right in the first row of the curve graph. Mass types 3-4 represent the mass change type from left to right in the second row of the curve graph. Figure 7 to Figure 10 show the relevant results.

MSCKF particularization According to the formulation presented in Section 4.2.2 for MSCKF algorithm, it is necessary to define the following state before applying the algorithm:

$$\begin{aligned} x_I &= \begin{pmatrix} q & r & v & m \end{pmatrix} \\ x_{c_i} &= \begin{pmatrix} q_i & r_i & v_i \end{pmatrix} \end{aligned} \tag{49}$$

SWLS particularization For SWLS, the optimization variables in the sliding window are defined as follows:

$$x_k = \begin{pmatrix} q_k & r_k & v_k & m_k \end{pmatrix}, k \in [0, n] \tag{50}$$

Where, n is the size of the sliding window. We will integrate the control input between the two states of the sliding window to establish their constraints. In order to avoid repeated integration during optimization iteration, we define the following pre-integration items:

$$\begin{aligned} \alpha_{b_{k+1}}^{b_k} &= \int\int_{t_k}^{t_{k+1}} R_{b_\tau}^{b_k} T d\tau^2 \\ \beta_{b_{k+1}}^{b_k} &= \int_{t_k}^{t_{k+1}} R_{b_\tau}^{b_k} T d\tau \\ \gamma_{b_{k+1}}^{b_k} &= \int_{t_k}^{t_{k+1}} \frac{1}{2} q_{b_\tau}^{b_k} \otimes \omega d\tau \end{aligned} \tag{51}$$

Where, t_k and t_{k+1} represent the timestamp of adjacent states in the sliding window. b represents the body frame.

The propagation process of the pre-integration items are as follows:

$$\begin{aligned} \alpha_{i+1}^{b_k} &= \alpha_i^{b_k} + \beta_i^{b_k} dt + \frac{1}{2} R\left(\gamma_i^{b_k}\right) T dt^2 \\ \beta_{i+1}^{b_k} &= \beta_i^{b_k} + R\left(\gamma_i^{b_k}\right) T dt \\ \gamma_{i+1}^{b_k} &= \gamma_i^{b_k} \otimes \begin{pmatrix} 1 \\ \frac{1}{2}\omega dt \end{pmatrix} \end{aligned} \tag{52}$$

dt represents the time interval between two adjacent control input.

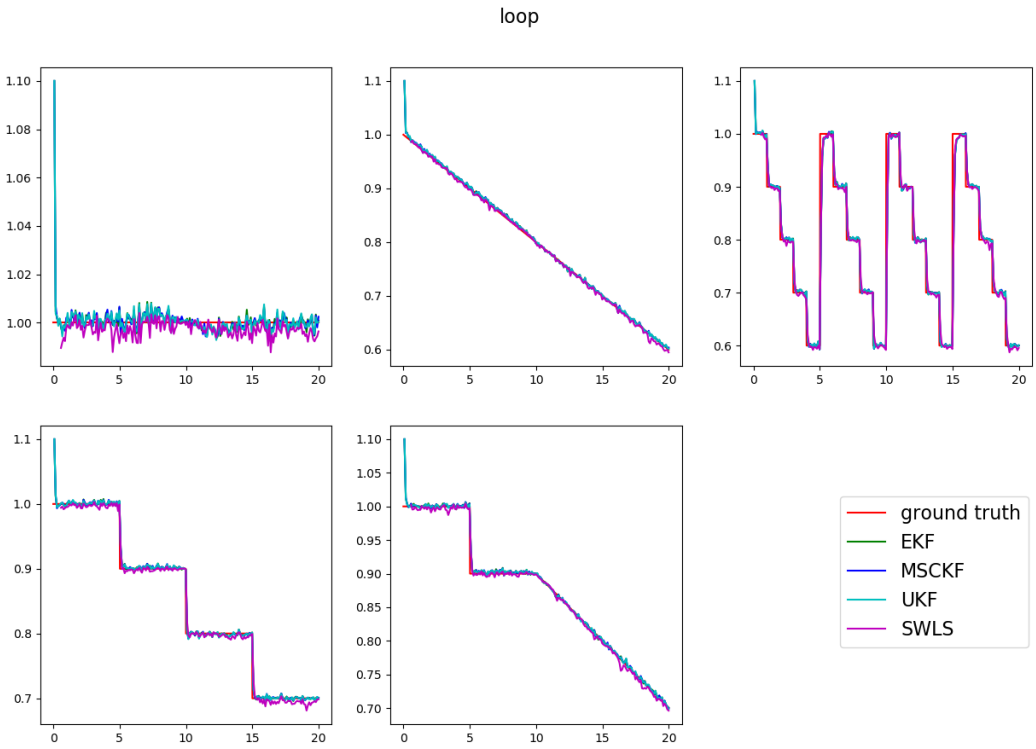
Next, we can derive the error state dynamic model by linearize pre-integration items:

$$\begin{aligned} \delta z &= \begin{pmatrix} \delta\alpha & \delta\beta & \delta\gamma & \delta m \end{pmatrix} \\ \delta z_{i+1}^{b_k} &= F \delta z_i^{b_k} + G n \\ F &= \partial z_{i+1}^{b_k} \Big/ \partial z_i^{b_k} \\ G &= \partial z_{i+1}^{b_k} \Big/ \partial n \end{aligned} \tag{53}$$

The expression of $r_D^k \begin{pmatrix} x_k & x_{k+1} \end{pmatrix}$ is as follows:

$$\begin{aligned} r_D^k \begin{pmatrix} x_k & x_{k+1} \end{pmatrix} &= \begin{bmatrix} \alpha_{b_{k+1}}^{b_k} - \hat{\alpha}_{b_{k+1}}^{b_k} \\ \beta_{b_{k+1}}^{b_k} - \hat{\beta}_{b_{k+1}}^{b_k} \\ \gamma_{b_{k+1}}^{b_k} - \hat{\gamma}_{b_{k+1}}^{b_k} \\ m_{k+1} - m_k \end{bmatrix} \\ \hat{\alpha}_{b_{k+1}}^{b_k} &= m_{k+1} R_w^{b_k} \left(p_{b_{k+1}}^w - p_{b_k}^w - v_{b_k}^w dt - \frac{1}{2} g dt^2 \right) \\ \hat{\beta}_{b_{k+1}}^{b_k} &= m_{k+1} R_w^{b_k} \left(v_{b_{k+1}}^w - v_{b_k}^w - g dt \right) \\ \hat{\gamma}_{b_{k+1}}^{b_k} &= q_w^{b_k} \otimes q_{b_{k+1}}^w \end{aligned} \tag{54}$$

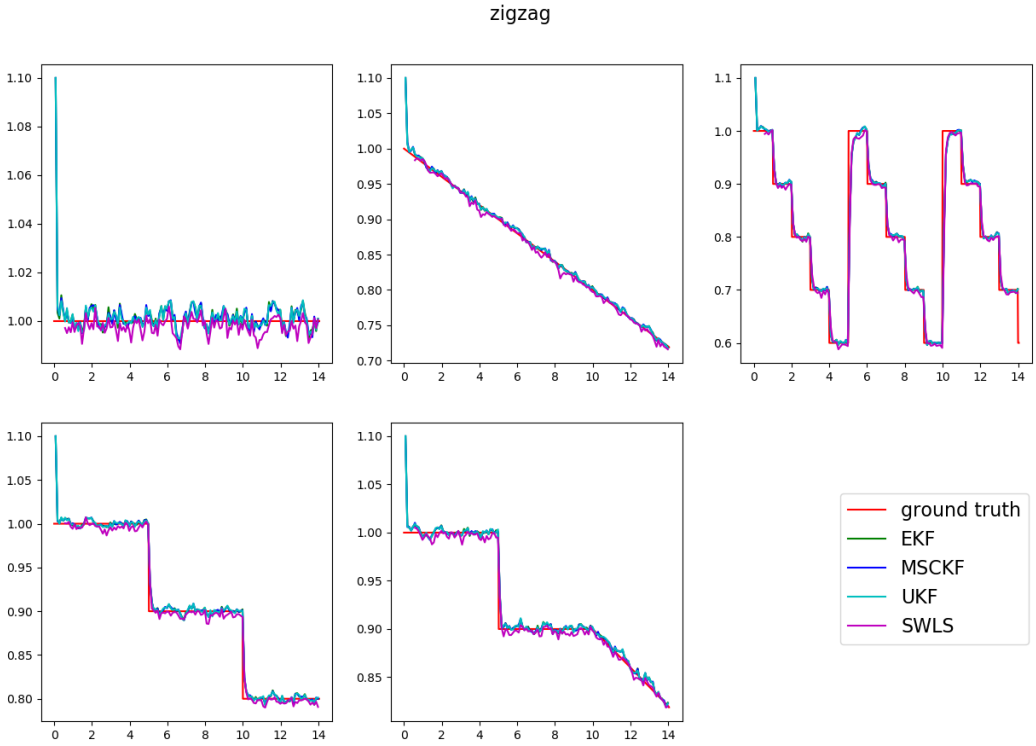
Loop trajectory Figure 7 shows the results obtained in the loop trajectory. These results show that all the algorithms performed well enough and could identify the system changes fast and accurately. According to the table, the faster algorithm is the EKF. This is an expected result because it is the lighter approach to compute. For this operation, the best performance according to the RMSPE is the SWLS in most cases. However, it can be observed how the EKF can overcome the SWLS method in the case in which the mass is decreasing and growing multiple times. This is because the SWLS has a more significant delay in capturing this behaviour.



Mass	EKF		MSCKF		UKF		SWLS	
0	0.005037	0.008745	0.156224	0.00876	0.007471	0.009106	0.597894	0.004954
1	0.004942	0.009057	0.156152	0.009064	0.00747	0.009921	0.610597	0.005879
2	0.00493	0.023998	0.156189	0.02399	0.007501	0.024243	0.616953	0.024268
3	0.00493	0.010843	0.156196	0.010849	0.007482	0.011378	0.60585	0.00937
4	0.004921	0.009312	0.156071	0.009316	0.007479	0.009972	0.622495	0.006196

Figure 7: Mass estimation (y-axis [kg]) vs time (x-axis [s]) for loop trajectory

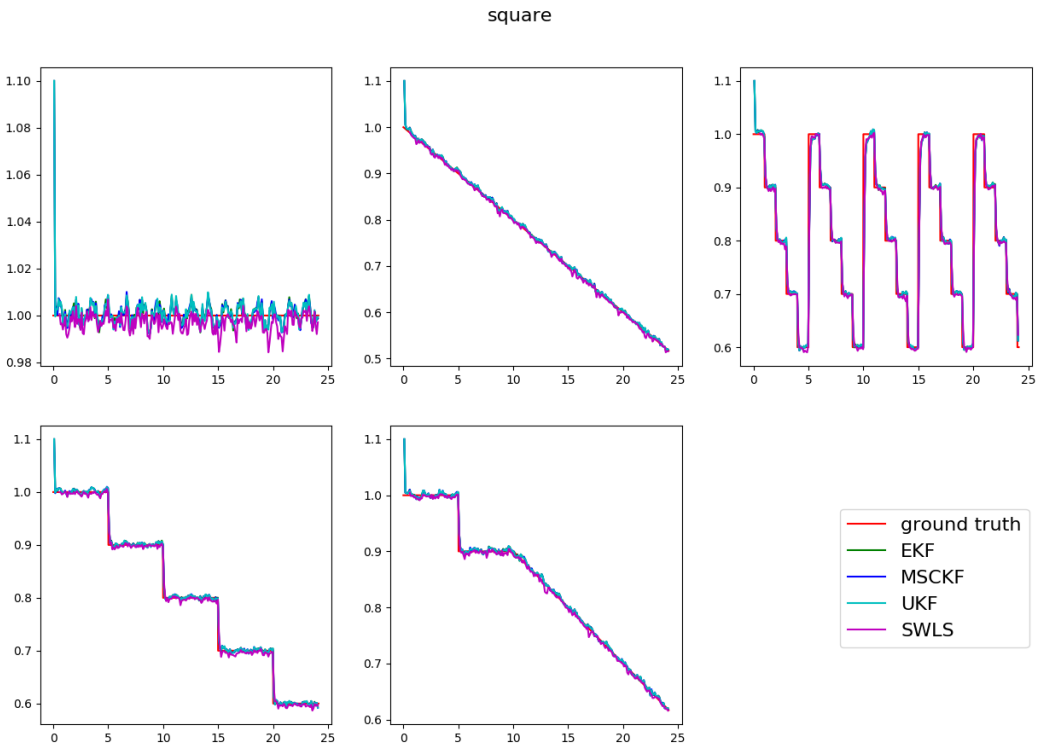
Zig-zag trajectory Figure 8 presents the performance of the different algorithms during the zigzag trajectory. The results are quite similar to the one obtained in the loop trajectories.Table results shows again that EKF is the lighter and faster approach while the SWLS is the more accurate one in most cases. However, SWLS is again unable to capture the behaviour of changing the mast abruptly several times and other approaches.



Mass	EKF		MSCKF		UKF		SWLS	
0	0.004917	0.01135	0.153067	0.011354	0.007444	0.011744	0.628381	0.005887
1	0.005011	0.010856	0.151846	0.010857	0.007439	0.011644	0.61556	0.005529
2	0.004925	0.026489	0.150888	0.026492	0.00745	0.026729	0.641007	0.027117
3	0.004918	0.012872	0.151925	0.012866	0.007408	0.013371	0.692178	0.009693
4	0.004913	0.011178	0.151301	0.011146	0.007417	0.011743	0.695861	0.006642

Figure 8: Mass estimation (y-axis [kg]) vs time (x-axis [s]) for zig-zag trajectory

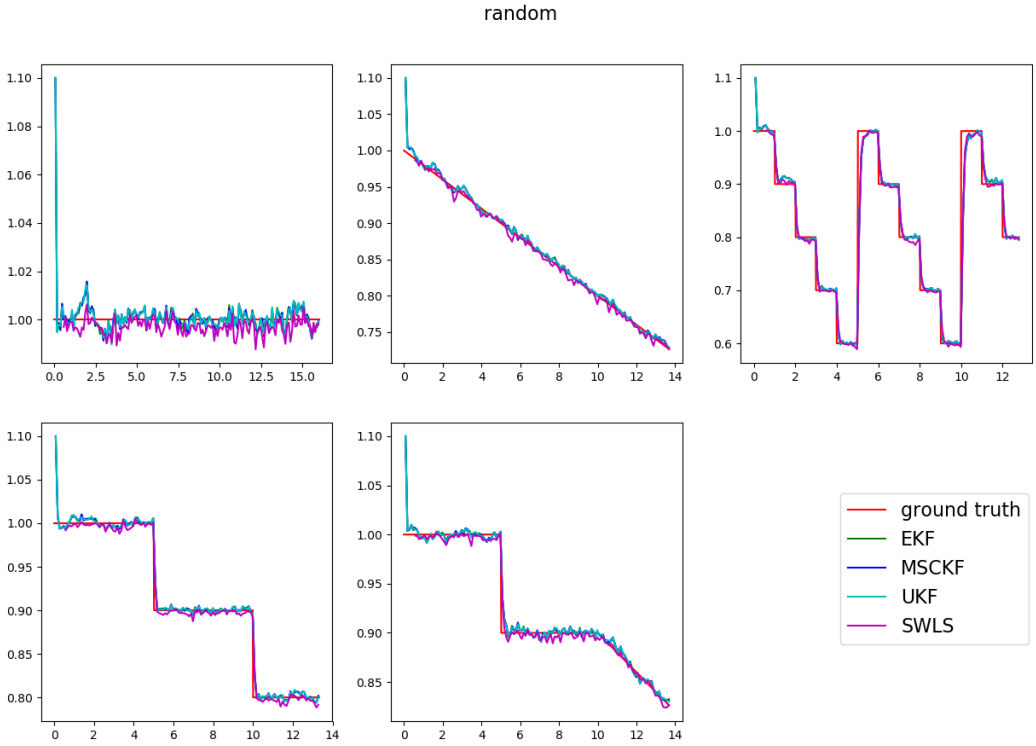
Square trajectory Figure 9 shows the results obtained during the square trajectory for the different mass behaviours. These results are really interesting and promising because most drone-based applications of SESAME will accomplish a square pattern. Again the EKF is the faster approach while the SWLS is the more accurate.



Mass	EKF		MSCKF		UKF		SWLS	
0	0.004924	0.008695	0.152749	0.008698	0.007467	0.009141	0.647751	0.005092
1	0.004936	0.0092	0.153848	0.009185	0.007444	0.01035	0.62049	0.006755
2	0.005032	0.026395	0.153689	0.026406	0.007446	0.026625	0.634881	0.027369
3	0.005329	0.011395	0.153959	0.011406	0.007466	0.012136	0.618204	0.011049
4	0.005206	0.009509	0.152988	0.009531	0.007458	0.010405	0.617943	0.006911

Figure 9: Mass estimation (y-axis [kg]) vs time (x-axis [s]) for square trajectory

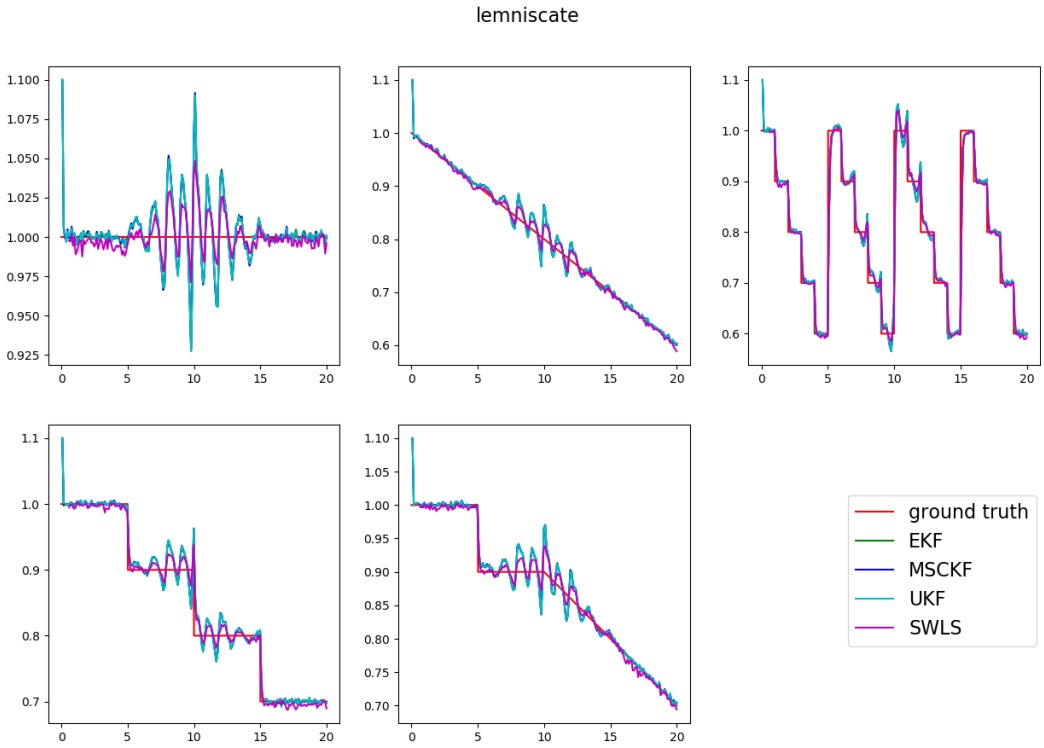
Random trajectory Figure 10 presents the performance of the algorithm while following a random trajectory. Although this part of the dataset was mainly created to enrich the training dataset of the data-driven approaches, it can be observed that the model-based algorithm can adequately capture the changes introduced in the dynamic of the system.



Mass	EKF		MSCKF		UKF		SWLS	
0	0.004514	0.009601	0.139542	0.009611	0.006776	0.00994	0.538735	0.00463
1	0.004977	0.011005	0.153236	0.011001	0.007384	0.011741	0.605798	0.005678
2	0.004951	0.027389	0.155007	0.027429	0.007484	0.027623	0.624608	0.027654
3	0.004974	0.013382	0.154583	0.013366	0.007457	0.013857	0.608136	0.009782
4	0.005004	0.011167	0.155007	0.011177	0.007479	0.0117	0.609635	0.007152

Figure 10: Mass estimation (y-axis [kg]) vs time (x-axis [s]) for random trajectory

Lemniscate trajectory Figure 11 presents the results obtained during the lemniscate trajectory. Although the results are quantitatively acceptable, the plots clearly show that the algorithms are not working as expected. In this case, the main issue is due to the trajectory itself.



Mass	EKF		MSCKF		UKF		SWLS	
0	0.004911	0.019865	0.156016	0.019973	0.007458	0.020285	0.595962	0.011106
1	0.004923	0.021039	0.155275	0.021154	0.007547	0.021703	0.611363	0.01219
2	0.004924	0.029075	0.155251	0.029123	0.007598	0.029423	0.611143	0.025179
3	0.004923	0.021913	0.155539	0.022019	0.007737	0.022417	0.608267	0.01439
4	0.004924	0.021279	0.154533	0.0214	0.007573	0.021859	0.653334	0.012348

Figure 11: Mass estimation (y-axis [kg]) vs time (x-axis [s]) for lemniscate trajectory

The lemniscate trajectory demands many efforts from the control perspective, and the control actions are very close to the limits of the robot. The trajectory is neither linear nor follows a constant rotational speed nor linear speed. The control inputs are changing frequently and abruptly. The robotic system cannot follow the references properly, and some of the required actions are not adequately followed by the controller. This situation makes the model-based observers unable to accurately estimate the mass of the system. Figure 12 compares the control actions of a lemniscate trajectory with the ones of a loop trajectory. It can be observed that the order of magnitude, the frequency and the smoothness of the input signals are very different in both cases.

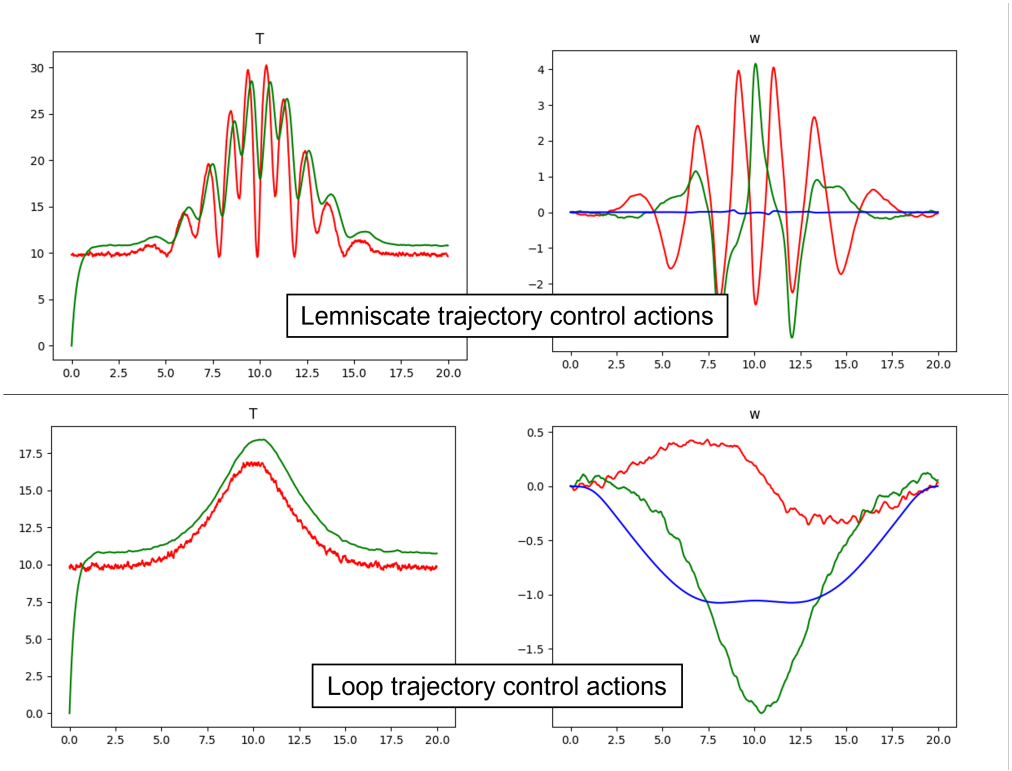


Figure 12: Results for lemniscate trajectory

6.3 Results of offline ELM and OSELM

In this section, the results of offline ELM and OSELM are presented. The activation function is selected as Sigmoid function, i.e., $g(x) = 1/(1 + e^{-x})$. For offline training, we have randomly selected 5 datasets from each trajectories and mass variation types. For different neurons number $\tilde{N}$ the results are presented. Also, we have analysed different numbers of neurons and regularization parameter values of λ in Table 1. Also, the effect of λ is analyzed in Table 2. Moreover, the different number of training datasets are considered in Table 3. Using these parameters, i.e., $\tilde{N} = 500$ and $\lambda = 0.00001$, the results are illustrated in Figures 13-15 for offline ELM and in Figures 16-17 for OSELM.

Evidently, both offline ELM and OSELM are able to estimate the mass variation accurately. This is obvious considering the total RMSE. More importantly, a higher estimation variance is obvious using offline ELM. This is because, in the offline ELM, the whole training dataset is used at once. Therefore, the training algorithm has to minimize the error over a wider range of data. However, in OSELM, the dataset is fed sequentially, which lets the training to learn the repetitive trends between the input data and the corresponding target values. It is worth noting that, since the training of the network relies on the fact that there must be a meaningful relation between the input and target datasets, offline ELM has poor performance for random trajectories, as illustrated in Figure 14. This is even worse for OSELM and the estimation error is significantly high and, therefore, the results are not presented here. As a result, the model-based approaches are recommended for the random trajectories.

Table 1: Offline ELM and RELM results for $\lambda = 0.0000001$ and 5 randomly selected datasets.

	Training Time		RMSE		Norm of weights	
$\tilde{N}$	RELM	ELM	RELM	ELM	RELM	ELM
50	3.0594 s	2.9783 s	5.6073%	5.5817%	6.4126×10^{3}	4.5661×10^{4}
100	2.9203 s	2.8903 s	3.6257%	3.4204%	2.3420×10^{4}	1.9392×10^{4}
300	3.1647 s	3.1154 s	2.3105%	2.0744%	2.6717×10^{4}	2.2691×10^{5}
500	3.4387 s	3.4709 s	2.2101%	1.8922%	2.6809×10^{4}	1.0978×10^{6}
1000	5.1501 s	5.2031 s	2.0948%	2.0035%	2.3237×10^{4}	8.9884×10^{7}

Table 2: RELM for various λ for $\tilde{N} = 500$ and 5 randomly selected datasets.

λ	Training Time	RMSE	Norm of weights
0.0000001	3.8071 s	2.1634 %	2.3441×10^{4}
0.00001	4.1647 s	2.5802 %	5.8101×10^{3}
0.001	4.1501 s	4.4006 %	1.0650×10^{3}
0.1	4.0520 s	6.8179 %	143.5273
0.3	3.9585 s	7.4042 %	100.8383
0.5	3.8273 s	7.8349 %	82.0590
0.8	4.0812 s	8.4250 %	64.8544
1.0	3.8565 s	8.5060 %	59.8437

Table 3: RELM for various number training datasets with $\tilde{N} = 500$ and $\lambda = 0.00001$.

X (% of total datasets)	Training Time	RMSE	Norm of weights
1 (2%)	0.6754 s	2.1162%	1.3895×10^{4}
3 (6%)	2.3554 s	2.1351%	2.1501×10^{4}
5 (10%)	3.5306 s	2.2240%	2.6017×10^{4}
7 (14%)	5.2201 s	2.1613%	2.7567×10^{4}
10 (20%)	7.6370 s	2.2374%	3.5137×10^{4}

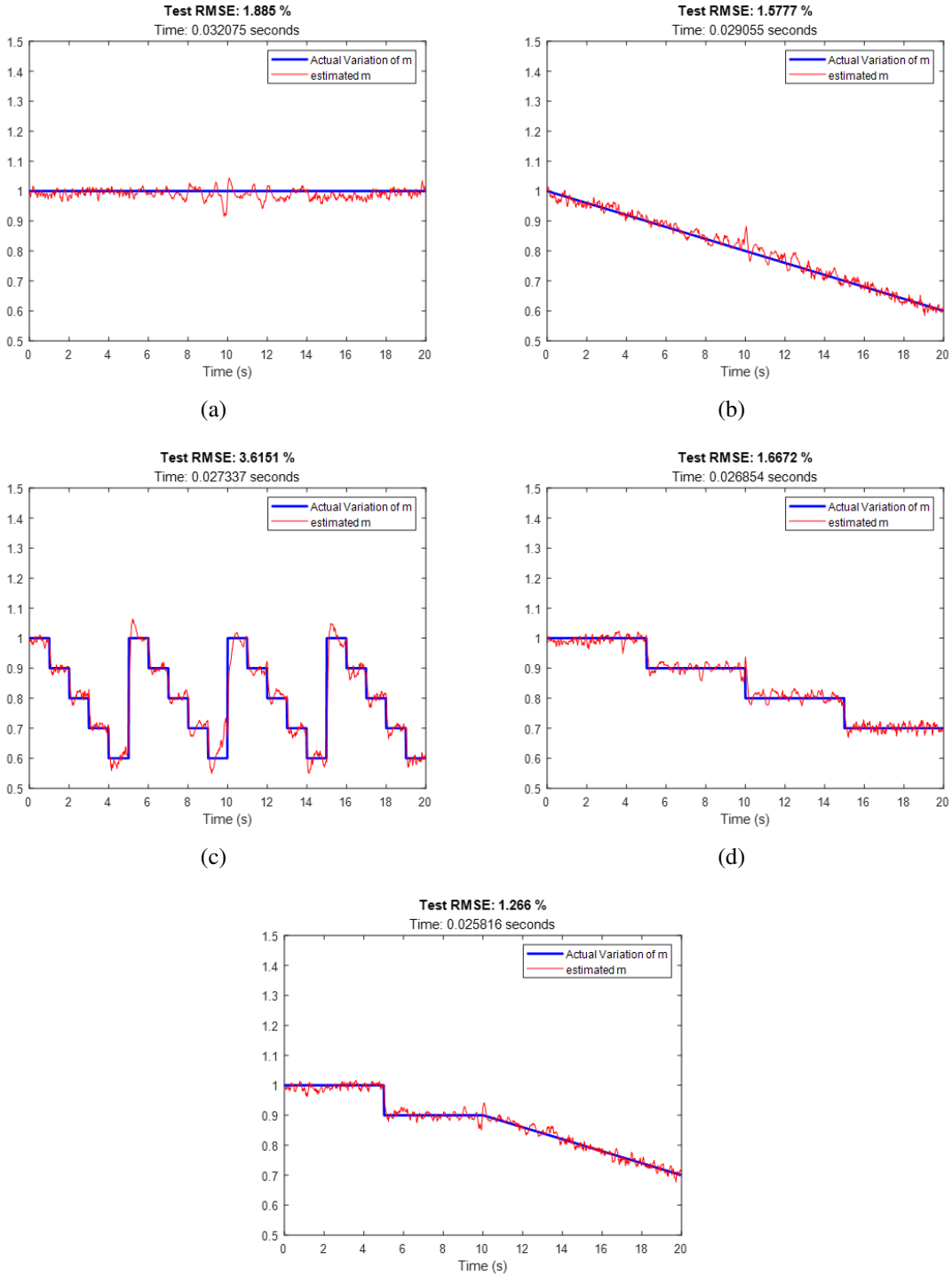


Figure 13: Offline ELM with lemniscate trajectory for (a) 1^{th}, (b) 2^{nd}, (c) 3^{rd}, (d) 4^{th}, and (e) 5^{th} type of mass variation. Vertical axis represents the mass variation in kilogram.

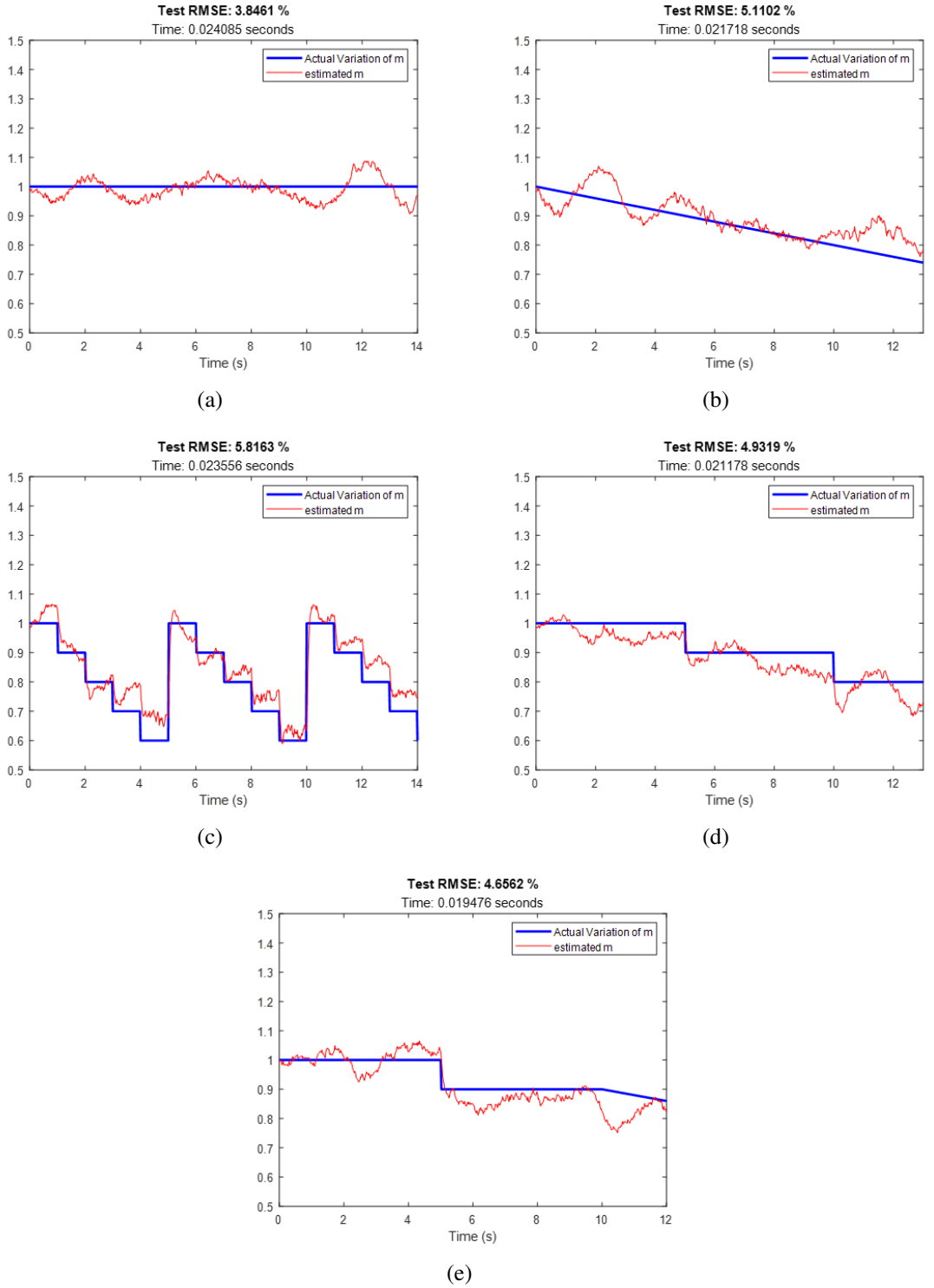


Figure 14: Offline ELM with random trajectory for (a) 1^{th}, (b) 2^{nd}, (c) 3^{rd}, (d) 4^{th}, and (e) 5^{th} type of mass variation. Vertical axis represents the mass variation in kilogram.

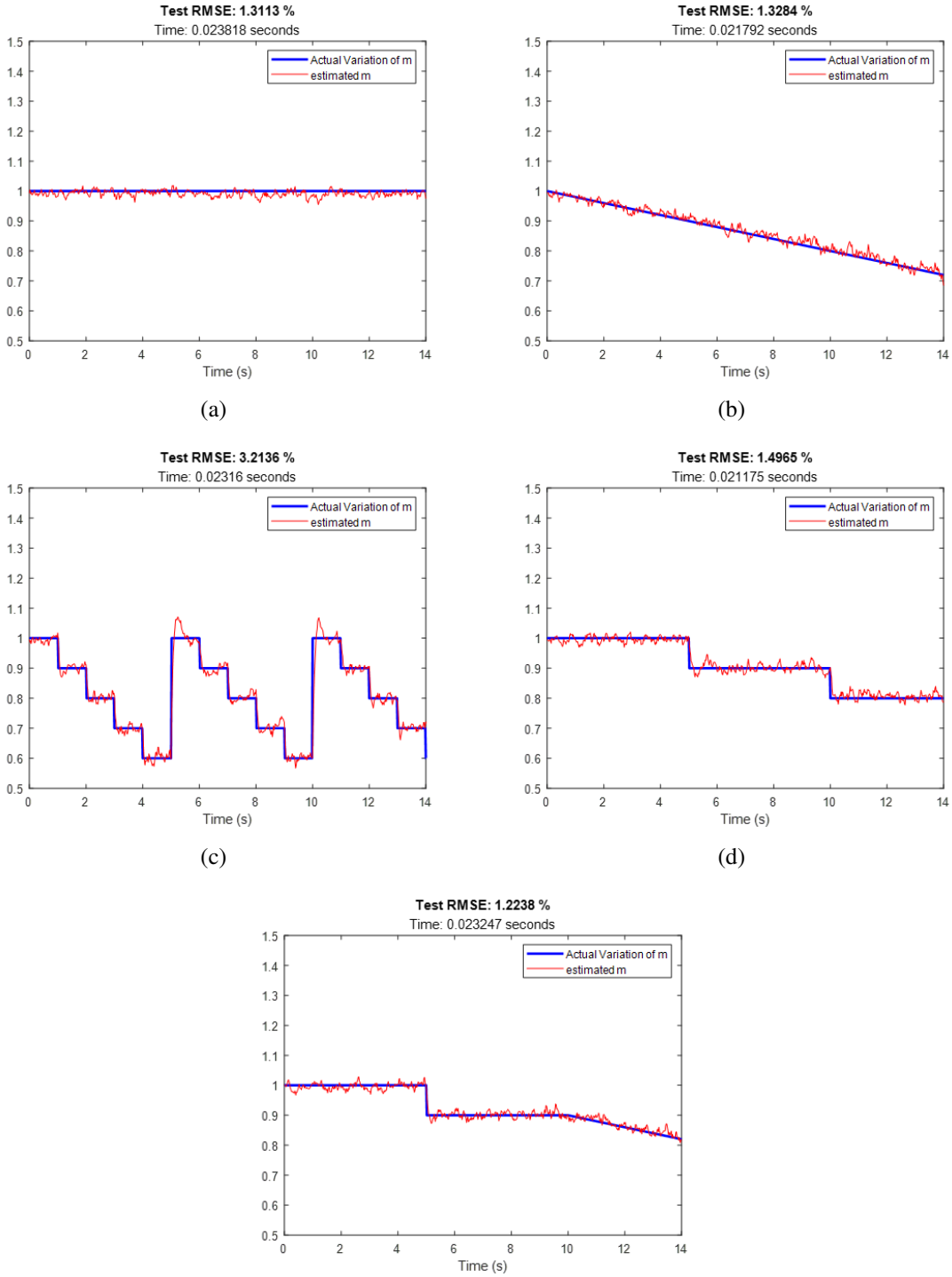
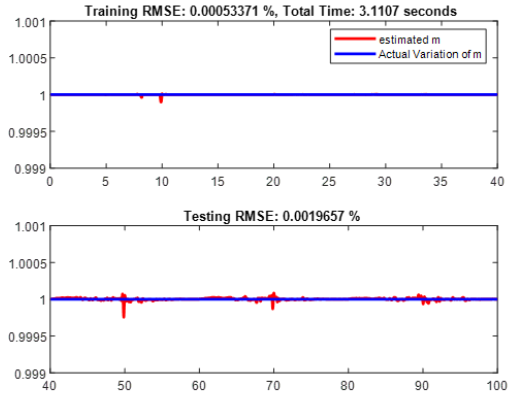
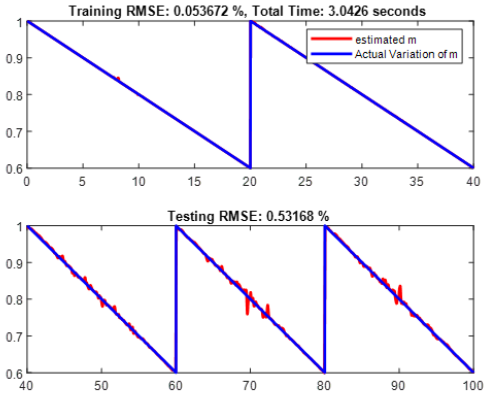


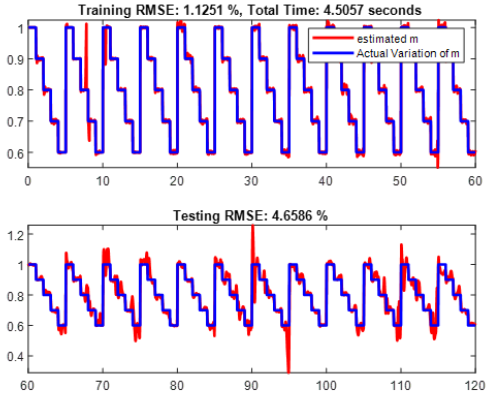
Figure 15: Offline ELM with zigzag trajectory for (a) 1^{th}, (b) 2^{nd}, (c) 3^{rd}, (d) 4^{th}, and (e) 5^{th} type of mass variation. Vertical axis represents the mass variation in kilogram.



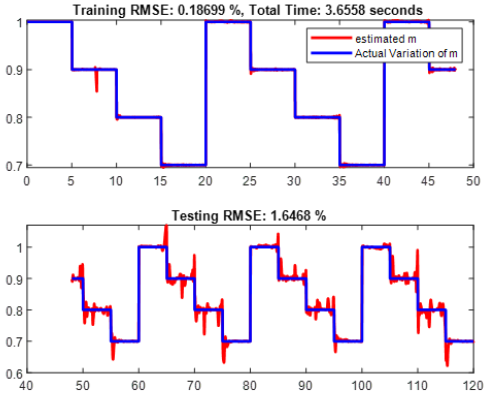
(a)



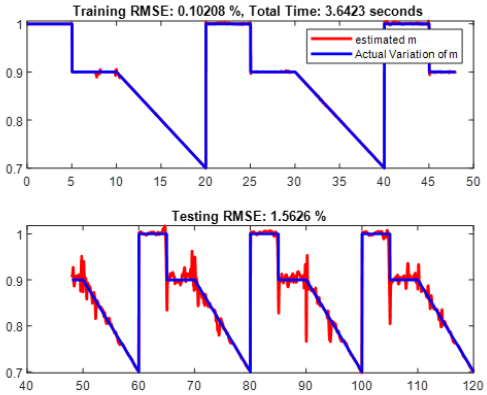
(b)



(c)

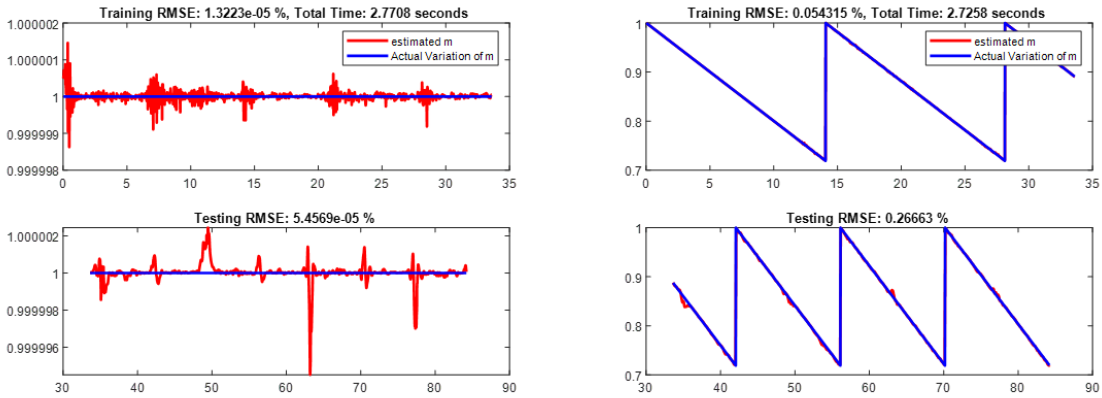


(d)



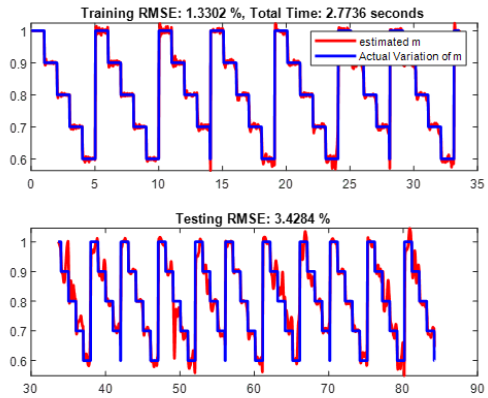
(e)

Figure 16: OSELM with lemniscate trajectory for (a) 1^{th}, (b) 2^{nd}, (c) 3^{rd}, (d) 4^{th}, and (e) 5^{th} type of mass variation. Vertical axis represents the mass variation in kilogram.

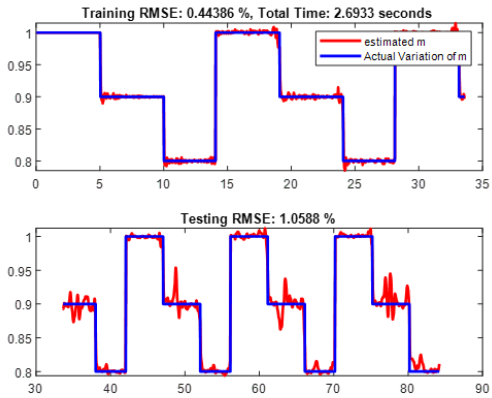


(a)

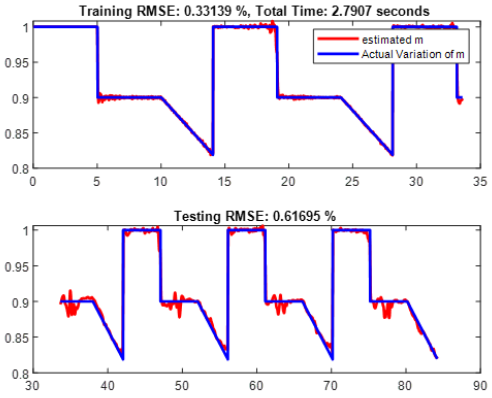
(b)



(c)



(d)



(e)

Figure 17: OSELM with zigzag trajectory for (a) 1^{th}, (b) 2^{nd}, (c) 3^{rd}, (d) 4^{th}, and (e) 5^{th} type of mass variation. Vertical axis represents the mass variation in kilogram.

7 Discussion

This deliverable has presented the work developed during the T2.2 of the WP2. This document aims to contribute to the SESAME knowledge database showing how different model identification techniques can be adapted to be executed in real-time and capture changes in the system dynamic.

As a final reflection and considering the presented results, we can conclude that selecting one model identification technique will depend on the robotic system and the specific operation. Quantitatively, the best approaches have been the EKF, SWLS, and OSELM. Depending on the real-time requirements, the available computational resources, the specific trajectories to be accomplished, and the possibility of having a previous data-set, the robotic user could decide using this document as a technical guideline.

The model identification techniques presented along this task will be integrated with the rest of the components developed in the WP2 to improve the safety and the security of a final multi-robot system. In this way, we aim to adapt the collaborative trajectory planning and the trajectory tracking algorithms with model information gathered and calculated in real-time. This will significantly improve the performance, robustness and reliability of the overall system.

8 Supplementary Study on Environmental Interference

8.1 Introduction

The autonomous navigation of UAV in complex and unknown environment is crucial to expand the application of UAV. And accurate system identification is the cornerstone of robust control and smooth planning of UAV. We have introduced relevant online system identification techniques in the previous sections. However, these techniques are only aimed at the model parameters of the UAV itself. Aerodynamic interference between UAV and environment is ignored. During the mission, the UAV may fly through strong winds or have physical interaction with the target (see Figure 18). Considering the external interference of UAV in the control and planning framework of UAV will be helpful to achieve more robust control and safer planning [27, 28]. This part will focus on the external force estimation of UAV.

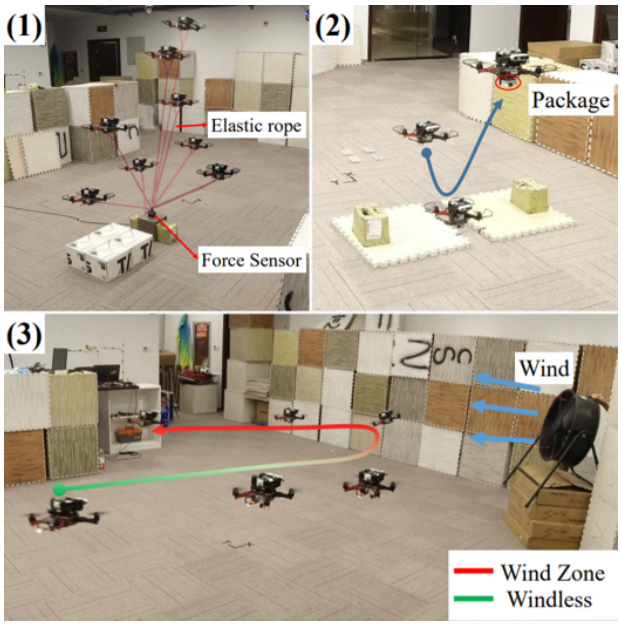


Figure 18: Environmental interference, taken from [1].

VIMO [29] is the first approach exploiting the external force within the framework of optimization based visual inertial odometry (VIO) [20]. The thrust measurements are preintegrated as relative motion constraints between key frames, refering to IMU preintegration [30]. However, the unknown external force is modeled as a zero-mean Gaussian. So the estimator will be highly affected and even leads to failure of the system by large or continuous external force. VID-Fusion [1] is an advanced version of VIMO. By exploiting the prior knowledge of external force derived from IMU and thrust measurements, VID-Fusion achieve more accurate external force estimation. However, the external force preintegration term between key frames is averaged to approximate the force variable in the sliding window. This approximation introduces estimation error.

To accurately and efficiently estimate the external force, we adopt a Multi-State Constraint Kalman Filter (MSCKF) [31] framework based on OpenVINS [32], a filter based VIO approach, which already shows superior computational efficiency compared with optimized based VIO approaches. Thrust and gyroscope measurements are utilized to propagate the state variable. Accelerometer measurements and camera measurements are used to update the state with respective frequency. The external force is updated with accelerometer frequency without average approximation error. In the real-world experiments, the accuracy superiority of proposed approach would be demonstrated.

Key contributions of this work are summarized as:

- We propose an online external force estimator based on MSCKF, tightly fusing visual inertial measurement and thrust measurement. To the best of our knowledge, we are the first to address this task using a tightly coupled filter framework. In addition, our estimator does not depend on GPS data, which means it can even be deployed in the exploration of extraterrestrial planets, such as Mars UAV [33, 34, 35].
- We evaluate the accuracy of the proposed algorithm with real-world datasets. Compared with the results of SOTA [1], the accuracy of external force estimation has significant improvement. RMSE is decreased over 50% for each sequence.

8.2 Problem Formulation

8.2.1 Notation

There, let us define the world coordinate system as W, the geometric centre coordinate system as B, the centroid system as M, and the IMU coordinate system as I. Frame B, M and I are coincident. In the following part, only I is used to refer to the body coordinate system of UAV.

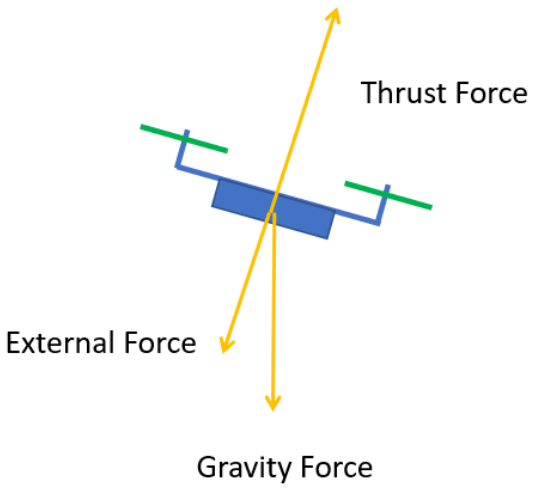


Figure 19: Forces of UAV.

We use reference ${}^{W}(\bullet)$ to represent a physical quantity in the coordinate system W. The position of the origin of the coordinate system I in the coordinate system W is expressed as ${}^{W}p_I$. The velocity of the origin point I in the coordinate system W is expressed as ${}^{W}v_I$. We use the unit quaternion to express the rotation of rigid body [36]. ${}^{I}_{W}q$ represents the attitude of the coordinate system I with respect to the coordinate system W, and its corresponding rotation matrix is ${}^{I}_{W}R$. $[\bullet]_\times$ is denoted as the skew symmetric matrix corresponding to a three-dimensional vector. The transpose of a matrix is $[\bullet]^T$.

The force analysis of a UAV is shown in the Figure 19. The resultant force other than gravity and thrust is named as external force. The mass normalized thrust and external force are denoted as T_m and F_{ext} respectively. These two forces are expressed in the body coordinate system.

8.2.2 State Vector

The MSCKF state vector includes the current robot state, N augmented historical pose clones and L augmented features:

$$\begin{aligned} x &= \begin{bmatrix} x_I^T & x_c^T & x_f^T \end{bmatrix}^T \\ x_I &= \begin{bmatrix} {}^{I}_{W}q^T & {}^{W}p_I^T & {}^{W}v_I^T & b_\omega^T & b_a^T & F_{ext}^T \end{bmatrix}^T \\ x_c &= \begin{bmatrix} x_{c_1}^T & \cdots & x_{c_N}^T \end{bmatrix}^T x_{c_i} = \begin{bmatrix} {}^{I_i}_{W}q^T & {}^{W}p_{I_i}^T \end{bmatrix}^T \\ x_f &= \begin{bmatrix} {}^{W}p_{f_1}^T & \cdots & {}^{W}p_{f_L}^T \end{bmatrix}^T \end{aligned} \tag{55}$$

Where x_I is the current robot state, including the robot pose, velocity and the bias of IMU. x_{c_i} is the augmented robot pose, which is obtained by cloning the first two physical quantities x_I at different camera times. N is known as the sliding window size, a fixed parameter. The pose clones in the sliding window are used for the triangulation of environmental feature points. ${}^{W}p_{f_j}$ is augmented feature, or SLAM feature [37, 38, 32].

8.2.3 Propagation

The nonlinear continuous-time process model of the system can be expressed as:

$$\begin{aligned} {}^{I}_{W}\dot{q} &= \frac{1}{2}\Omega\left(\omega_m - b_\omega\right){}^{I}_{W}q \\ {}^{W}\dot{p}_I &= {}^{W}v_I \\ {}^{W}\dot{v}_I &= {}^{I}_{W}R^T\left(T_m + F_{ext}\right) + g \\ \dot{b}_\omega &= n_{b_\omega} \\ \dot{b}_a &= n_{b_a} \\ \dot{F}_{ext} &= n_F \end{aligned} \tag{56}$$

Where,

$$\Omega\left(\omega\right) = \begin{bmatrix} -[\omega]_\times & \omega \\ -\omega^T & 0 \end{bmatrix} \tag{57}$$

g is the local gravity vector. $n_{[\bullet]}$ represents the zero mean Gaussian noise of $[\bullet]$. x_I is driven by angular velocity measurement ω_m and thrust measurement T_m.

8.2.4 Accelerometer Update

Accelerometer measurement a_m can observe external force. The measurement equation is as follows:

$$a_m = T_m + F_{ext} + b_a \tag{58}$$

Unlike VIMO and VID-Fusion, we update and optimize external force with accelerometer frequency.

8.2.5 Visual Measurement Update

We follow OpenVINS [32] for the technical details of visual update. Each time we receive a new image, we first clone the latest robot pose and augment it into the state vector to track the camera pose. After the state augmentation is completed, we check the sliding window size and marginalize the oldest clone state if it exceeds N. The selected feature points are used to update the poses over the sliding window.

8.3 Real-world Experiments

To the best of our knowledge, in the UAV community, VID-Dataset [2] is the first and only dataset that contains real-world visual-inertial measurement, thrust measurement, pose groundtruth and external force groundtruth. We chose sequence 17 and 18, because only these two sequences have the groundtruth of external force. The data collection platform is shown in the Figure 20. The external force of the UAV is dominated by the tension force of the elastic rope, so the tension force measured by the force sensor can be regarded as the external force of the UAV, namely F_{ext}.

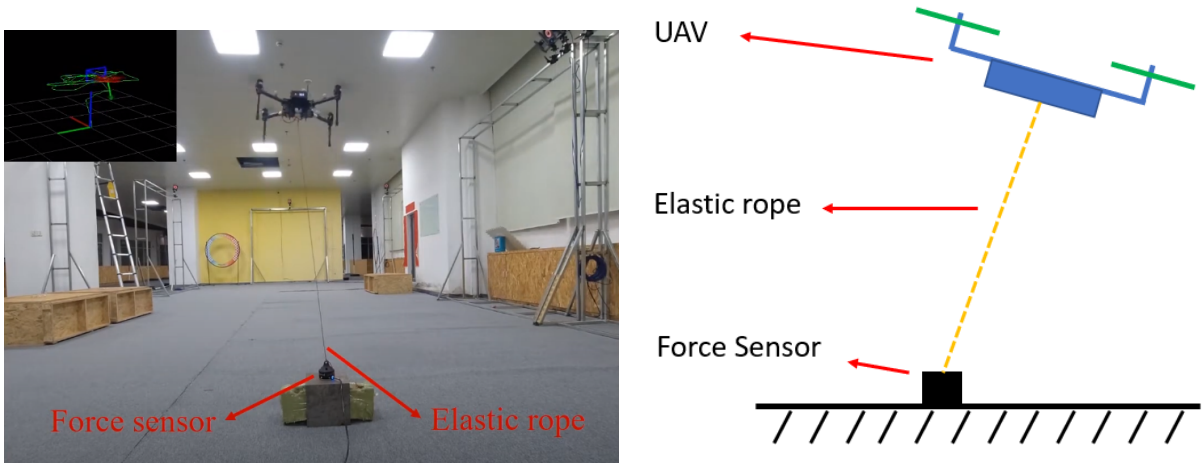


Figure 20: Left: Picture of experimental platform, taken from [2]; Right: Abstract diagram of experimental platform.

The estimation results of F_{ext} are shown in the Figure 21. The quantified RMSE results are shown in the Table 4. Compared with VID-Fusion, our method has significant accuracy improvement.

The trajectory estimation results are shown in the Figure 22. Our results are closer to the groundtruth. The quantified absolute trajectory error (ATE) results are shown in the Table 5. Our method achieves better pose estimation than VID-Fusion.

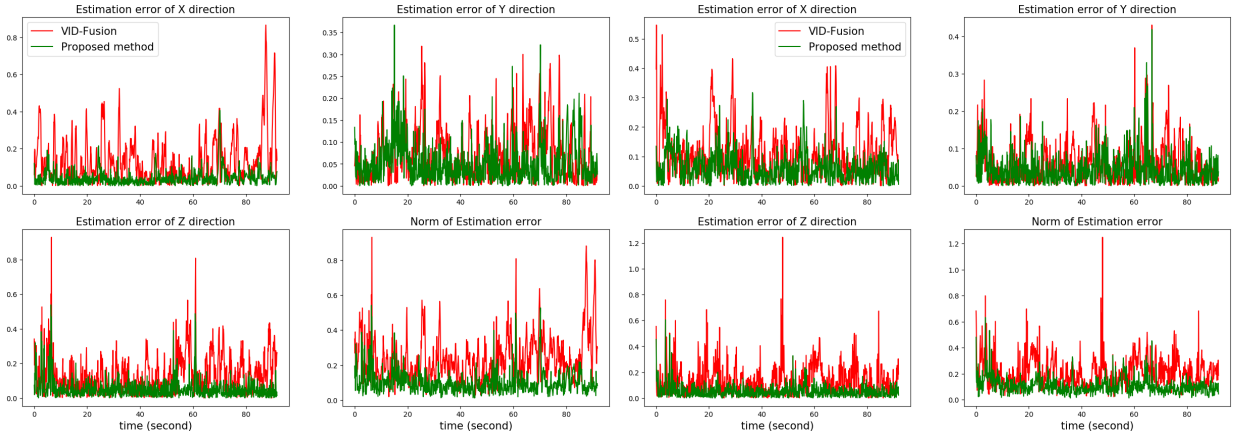


Figure 21: Left: External force estimation results for sequence 17; Right: External force estimation results for sequence 18.

Table 4: External force estimation RMSE (m/s^2) of different algorithms (VID-Fusion, Ours) was evaluated with different sequences of VID-Dataset.

Sequence	VID-Fusion	Ours	Decrease
17	0.206	**0.072**	65.05%
18	0.160	**0.074**	53.75%

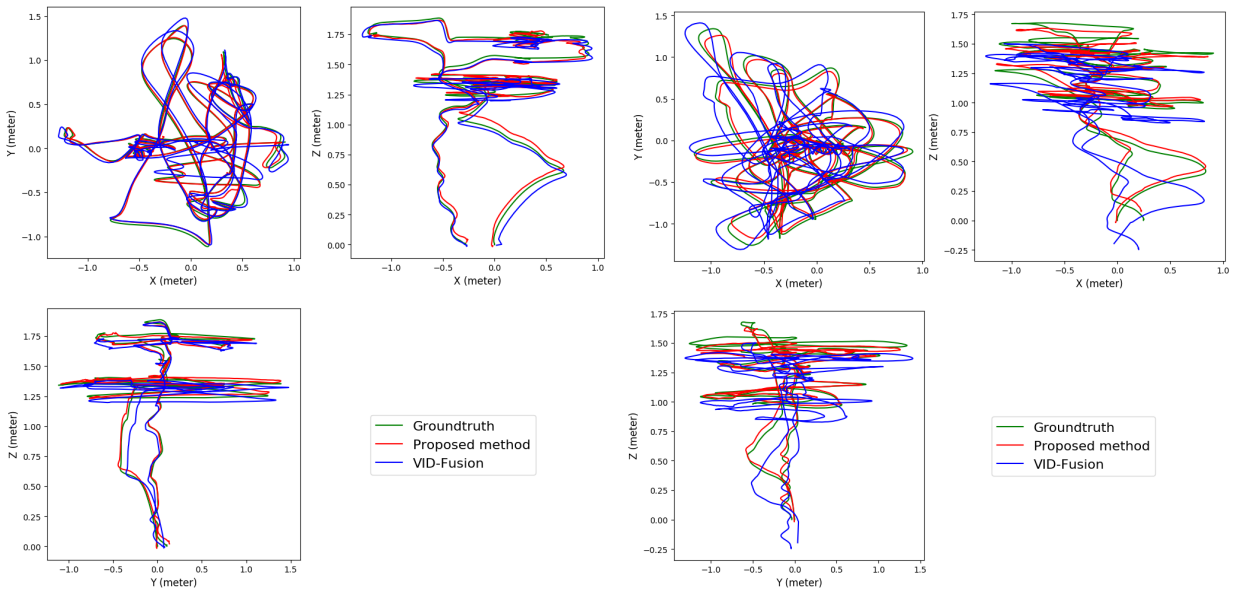


Figure 22: Left: Different views of the aligned trajectories for sequence 17; Right: Different views of the aligned trajectories for sequence 18.

Table 5: ATE (m) of different algorithms (VID-Fusion, Ours) was evaluated with different sequences of VID-Dataset.

Sequence	VID-Fusion	Ours	Decrease
17	0.0456	**0.0362**	20.61%
18	0.0788	**0.0499**	36.68%

8.4 Conclusion

In this work, a novel external force estimator for UAV is proposed to address the issue of unknown environmental interference. Compared with SOTA, the estimation accuracy is significantly improved. Accurate knowledge about external force would provide strong support for downstream applications, such as control and planning [27, 28].

References

[1] Z. Ding, T. Yang, K. Zhang, C. Xu, and F. Gao, "Vid-fusion: Robust visual-inertial-dynamics odometry for accurate external force estimation," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 14469–14475, IEEE, 2021.

[2] K. Zhang, T. Yang, Z. Ding, S. Yang, T. Ma, M. Li, C. Xu, and F. Gao, "The visual-inertial-dynamical multirotor dataset," in *2022 International Conference on Robotics and Automation (ICRA)*, pp. 7635–7641, IEEE, 2022.

[3] R. Teo, J. Jang, and C. Tomlin, "Model predictive quadrotor indoor position control," in *Proc 43rd IEEE Conf. Decision Control, Paradise Island, Bahamas*, vol. 4, pp. 4268–4273, 2004.

[4] D. Falanga, P. Foehn, P. Lu, and D. Scaramuzza, "PAMPC: Perception-aware model predictive control for quadrotors," in *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, 2018.

[5] A. Liniger, A. Domahidi, and M. Morari, "Optimization-based autonomous racing of 1: 43 scale rc cars," *Optimal Control Applications and Methods*, vol. 36, no. 5, pp. 628–647, 2015.

[6] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4906–4913, IEEE, 2012.

[7] J. Koenemann, A. Del Prete, Y. Tassa, E. Todorov, O. Stasse, M. Bennewitz, and N. Mansard, "Whole-body model-predictive control applied to the hrp-2 humanoid," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3346–3351, IEEE, 2015.

[8] F. Farshidian, E. Jelavic, A. Satapathy, M. Giftthaler, and J. Buchli, "Real-time motion planning of legged robots: A model predictive control approach," in *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pp. 577–584, IEEE, 2017.

[9] D. Mellinger, N. Michael, and V. Kumar, "Trajectory generation and control for precise aggressive maneuvers with quadrotors," *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 664–674, 2012.

[10] M. Castillo-Lopez, P. Ludivig, S. A. Sajadi-Alamdari, J. L. Sanchez-Lopez, M. A. Olivares-Mendez, and H. Voos, "A real-time approach for chance-constrained motion planning with dynamic obstacles," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3620–3625, 2020.

[11] J. L. Sanchez-Lopez, M. Castillo-Lopez, M. A. Olivares-Mendez, and H. Voos, "Trajectory tracking for aerial robots: an optimization-based planning and control approach," *Journal of Intelligent & Robotic Systems*, vol. 100, no. 2, pp. 531–574, 2020.

[12] S. X. Ding, *Model-based fault diagnosis techniques: design schemes, algorithms, and tools*. Springer Science & Business Media, 2008.

[13] S. Beghelli, R. P. Guidorzi, and U. Soverini, "The frisch scheme in dynamic system identification," *Automatica*, vol. 26, no. 1, pp. 171–176, 1990.

[14] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: theory and applications," *Neurocomputing*, vol. 70, no. 1-3, pp. 489–501, 2006.

[15] T. Takagi and M. Sugeno, "Fuzzy identification of systems and its applications to modeling and control," *IEEE transactions on systems, man, and cybernetics*, no. 1, pp. 116–132, 1985.

[16] J.-S. Jang, "Anfis: adaptive-network-based fuzzy inference system," *IEEE transactions on systems, man, and cybernetics*, vol. 23, no. 3, pp. 665–685, 1993.

[17] K. Sun, K. Mohta, B. Pfrommer, M. Watterson, S. Liu, Y. Mulgaonkar, C. J. Taylor, and V. Kumar, "Robust stereo visual inertial odometry for fast autonomous flight," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 965–972, 2018.

[18] V. Wüest, V. Kumar, and G. Loianno, "Online estimation of geometric and inertia parameters for multirotor aerial vehicles," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 1884–1890, IEEE, 2019.

[19] C. Hertzberg, R. Wagner, U. Frese, and L. Schröder, "Integrating generic sensor fusion algorithms with sound state representations through encapsulation of manifolds," *Information Fusion*, vol. 14, no. 1, pp. 57–77, 2013.

[20] T. Qin, P. Li, and S. Shen, "Vins-mono: A robust and versatile monocular visual-inertial state estimator," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1004–1020, 2018.

[21] N.-Y. Liang, G.-B. Huang, P. Saratchandran, and N. Sundararajan, "A fast and accurate online sequential learning algorithm for feedforward networks," *IEEE Transactions on neural networks*, vol. 17, no. 6, pp. 1411–1423, 2006.

[22] P. L. Bartlett, "The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network," *IEEE transactions on Information Theory*, vol. 44, no. 2, pp. 525–536, 1998.

[23] A. N. Tikhonov and V. Y. Arsenin, "Solutions of ill-posed problems. vh winston & sons," 1977.

[24] W. Deng, Q. Zheng, and L. Chen, "Regularized extreme learning machine," in *2009 IEEE symposium on computational intelligence and data mining*, pp. 389–395, IEEE, 2009.

[25] J. Sola, "Quaternion kinematics for the error-state kalman filter," *arXiv preprint arXiv:1711.02508*, 2017.

[26] G. Torrente, E. Kaufmann, P. Föhn, and D. Scaramuzza, "Data-driven mpc for quadrotors," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3769–3776, 2021.

[27] Y. Wu, Z. Ding, C. Xu, and F. Gao, "External forces resilient safe motion planning for quadrotor," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 8506–8513, 2021.

[28] Y. Wang, J. O'Keeffe, Q. Qian, and D. Boyle, "Kinojgm: A framework for efficient and accurate quadrotor trajectory generation and tracking in dynamic environments," in *2022 International Conference on Robotics and Automation (ICRA)*, pp. 11036–11043, IEEE, 2022.

[29] B. Nisar, P. Foehn, D. Falanga, and D. Scaramuzza, "Vimo: Simultaneous visual inertial model-based odometry and force estimation," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2785–2792, 2019.

[30] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza, "On-manifold preintegration for real-time visual–inertial odometry," *IEEE Transactions on Robotics*, vol. 33, no. 1, pp. 1–21, 2016.

[31] A. I. Mourikis and S. I. Roumeliotis, "A multi-state constraint kalman filter for vision-aided inertial navigation," in *Proceedings 2007 IEEE international conference on robotics and automation*, pp. 3565–3572, IEEE, 2007.

[32] P. Geneva, K. Eckenhoff, W. Lee, Y. Yang, and G. Huang, "Openvins: A research platform for visual-inertial estimation," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4666–4672, IEEE, 2020.

[33] J. Delaune, R. Brockers, D. S. Bayard, H. Dor, R. Hewitt, J. Sawoniewicz, G. Kubiak, T. Tzanetos, L. Matthies, and J. Balaram, "Extended navigation capabilities for a future mars science helicopter concept," in *2020 IEEE Aerospace Conference*, pp. 1–10, IEEE, 2020.

[34] J. Delaune, D. S. Bayard, and R. Brockers, "Range-visual-inertial odometry: Scale observability without excitation," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2421–2428, 2021.

[35] V. Polizzi, R. Hewitt, J. Hidalgo-Carrió, J. Delaune, and D. Scaramuzza, "Data-efficient collaborative decentralized thermal-inertial odometry," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 10681–10688, 2022.

[36] N. Trawny and S. I. Roumeliotis, "Indirect kalman filter for 3d attitude estimation," *University of Minnesota, Dept. of Comp. Sci. & Eng., Tech. Rep*, vol. 2, p. 2005, 2005.

[37] M. Li and A. I. Mourikis, "Optimization-based estimator design for vision-aided inertial navigation," in *Robotics: Science and Systems*, pp. 241–248, Berlin Germany, 2013.

[38] M. Li, *Visual-inertial odometry on resource-constrained systems*. University of California, Riverside, 2014.