Towards Online System Identification: Benchmark of Model Identification Techniques for Variable Dynamics UAV Applications

Junlin Song
SpaceR Research Group
SnT - University of Luxembourg
Luxembourg, Luxembourg
junlin.song@uni.lu

Pedro J. Sanchez-Cuevas
Advance Centre for Aerospace Technologies (CATEC)
Seville
psanchez@catec.aero

Miguel Olivares-Mendez
SpaceR Research Group
SnT - University of Luxembourg
Luxembourg, Luxembourg
miguel.olivaresmendez@uni.lu

***Abstract*—Providing self-modelling capabilities to robotic systems that could change their dynamics variables during their natural operation, like aerial manipulators, can significantly increase model-based algorithm's resilience and performance. Some samples of those techniques are model predictive control or model-based trajectory planning techniques. This paper aims to benchmark how classical model identification techniques perform when adapted to be used at run-time. This self-awareness capability will improve transparency giving the robot the capability to understand what it can do and how to perform optimally. To do so, this paper compares four different model identification techniques and how they perform online in terms of accuracy and computational cost. The online adaptation of the model identification methods has been developed generically to apply to any dynamic system. The methods are numerically evaluated simulating an Unmanned Aerial Vehicle (UAV) with time-varying mass. The system has been evaluated in 5 different manoeuvres with 5 different mass-behaviour. This creates 25 experiments for each of the four model identification algorithms. In total, this paper presents the result of 100 studied cases.**

***Index Terms*—Reliability of UAS, UAS modelling, System identification**

I. Introduction

Several State-Of-The-Art (SOTA) controllers, trajectory planners, and sensor fusion algorithms use kinematic and dynamic information of robots to optimize its behavior and maximize its performance. 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 [1], [2], autonomous racing [3], and legged locomotion [4], [5], [6]. These strategies enable control action optimization for a given cost function over time. MPC approaches are usually combined with a trajectory generation phase. In [7], 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 [8] where the authors aim to solve the control and the planning problems together using the model information. However, although model-based techniques provide optimal results when the used model is accurate, those techniques could become unsafe, unstable, and, for sure, not optimal if the used model is not properly formulated or inaccurate.

Recent applications of aerial robots require changing the mass or the topology of the robot in the middle of the operation could make mandatory to update the model in real time. For instance, a drone that sprays pesticide in a wine-yard [9] [10], a morphing robot that changes its morphology [11] [12] or even an aerial manipulator [13], 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 while feeding the control and planning approaches with model identification.

This paper analyses different model identification approaches to capture changes in dynamic variables at runtime. In order to generalize the different methodologies and be able to apply them to different robots in general, and aerial robots in particular, it is essential to remark that the presented techniques are presented as comprehensive and generic as possible to consider a variety of dynamic systems. One critical aspect is to identify the level of abstraction required for the specific use case. This will mainly depend on the robotic system and the inputs/outputs available. In this study, we will address the problem of identifying an aerial robot that follows the most widespread architecture in the market and the research community. However, those techniques could be adapted to identify other robotic systems.

The rest of the paper is structured as follows: Section II presents the problem and rationale behind the need for model identification and introduces the generic formulation of a dynamic system. Section III formulates different model identification techniques adapted to the UAV-based proposed use case along with section IV. Section V shows the results

This research was supported by the European Union's Horizon 2020 project SESAME (grant agreement No 101017258).

obtained with the different methods, and last, Section VI presents the conclusions and future works of this research.

II. Problem statement

A. Rationale

The classical architecture of model-based algorithms follow the architecture presented in Figure 1. In general terms, considering the model in the estimation, control and planning algorithms significantly improve the operation's optimality and reliability. However, the presence of inaccuracies in the model can compromise the entire operation, and, unfortunately, they are sometimes unavoidable.

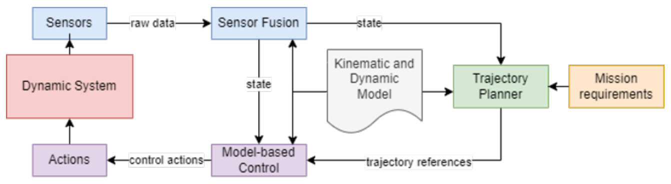


Fig. 1. Model-based approach scheme.

The system model is usually calculated offline after collecting information about the actual system during operation. Nevertheless, sometimes, the maneuvers accomplished during the identification process are not rich enough to capture the complete dynamics of the system and some behaviours could have been remained without modelling. With classical architecture, the errors produced during the identification process are present during the entire operation of the system. Moreover, it cannot capture changes in the model while operating.

In this paper, we propose to include the model identification as a runtime component that could be continuously estimating and updating the model of the system while working (see Figure 2). Therefore, we aim to evaluate the capability of adapting SOTA model identification techniques to be used online, capturing their performance and computational cost.

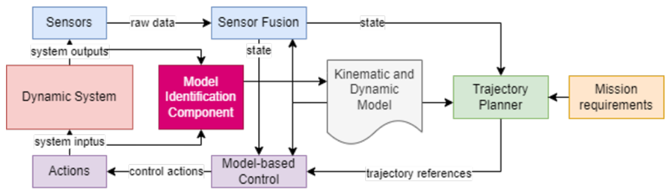


Fig. 2. Model-based approach scheme with the online model identification component.

B. Generic model formulation

A generic system is usually formulated with differential equations as:

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

The measured variables follow the equation as:

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

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.

III. Model identification techniques

An Extended Kalman Filter (EKF) [14] [15], a Multi-State Constraint Kalman Filter (MSCKF) [16], and an Unscented Kalman Filter (UKF) [17] 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 [18] has also been adapted to evaluate the performance of this purely mathematical fitting method to capture changes in the dynamic variables.

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 behaviour expected in their robots.

A. 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. We could include indirectly observable variables like the mass in the state vector 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 behaviour. The algorithm will compare the prediction with the actual input to recalculate the dynamic variables and their uncertainty in the next step-time. 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 $k+1$:

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

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.

B. Multi-State Constraint Kalman Filter: MSCKF

The idea of this part draws lessons from [19]. 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 III-A).

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 III-A.

C. Unscented Kalman Filter: UKF

Here we refer to [20], [21]. 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:

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

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.

D. 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 [22], [23].

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 $\gamma_{b_{k+1}}^{b_k}$ [23].

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:

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

The final optimization problem is:

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

Where, r_{prior} represents the prior constraint, $r_P^k(x_k)$ 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 $r_P^k(x_k)$ is straightforward:

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

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].

IV. Case study

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[2] project 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.

Following, the notation, the system's dynamic model, and how the dataset have been generated are presented.

A. 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 ${}_C r_{AB}$. 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 [24]. q_{BA} denote the attitude of a coordinate frame B with respect to frame A. The corresponding rotation matrix is $R(q_{BA})$. The coordinate transformation is expressed as follows:

$${}_B r = R(q_{BA})\, {}_A r \tag{24}$$

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

[2]https://www.sesame-project.org/

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

$$\begin{aligned} q+\delta &:= q \otimes \exp\left(\tfrac{\delta}{2}\right) \\ q_1 - q_2 &:= 2\log\left(q_2^{-1} \otimes q_1\right) \end{aligned} \tag{25}$$

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

B. 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:

$$\begin{aligned} {}_W\dot{r}_{WB} &= {}_W v_B \\ {}_W\dot{v}_B &= \frac{1}{m} R\left(q_{WB}\right) T + g \\ \dot{q}_{WB} &= \frac{1}{2} q_{WB} \otimes \omega \end{aligned} \tag{26}$$

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:

$$\dot{m} = 0 \tag{27}$$

Next, we define the following state vector x :

$$x = \begin{bmatrix} q \\ r \\ v \\ m \end{bmatrix} \tag{28}$$

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:

$$T^d \approx T \tag{29}$$

Next, we define our control input u as:

$$u = \begin{bmatrix} T^d \\ \omega \end{bmatrix} \tag{30}$$

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

$$y = \begin{bmatrix} q \\ r \\ v \end{bmatrix} \tag{31}$$

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.

$$\begin{aligned} \dot{x} &= f(x,u) + n_x = \begin{bmatrix} \frac{1}{2} q \otimes \omega \\ v \\ \frac{1}{m} R(q) T^d + g \\ 0 \end{bmatrix} + n_x \\ y &= h(x) + n_y = \begin{bmatrix} q \\ r \\ v \end{bmatrix} + n_y \end{aligned} \tag{32}$$

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

C. Dataset

In order to generate a populated dataset to assess the performance of the different model identification techniques, 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 [25], [2]. The dataset generation framework is, therefore, as the one presented in the Figure 3. We need to set the robot dynamic model and reference trajectory with this architecture.

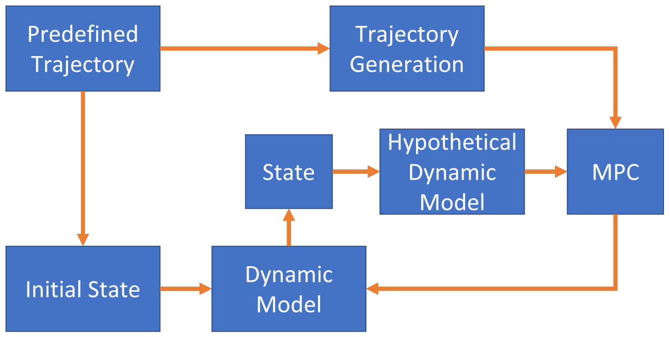


Fig. 3. Dataset generation framework

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 have perform 25 different simulations combining five different trajectories, namely "random", "loop", "lemniscate", "zigzag", and "square", with five mass behaviours. Moreover, to have a rich dataset and enabling future model identification with data-driven approaches, we have simulated each case trajectory 50 times, making a total of 1250 simulations.

The reference trajectory and mass change curve we designed are shown in Figure 4.

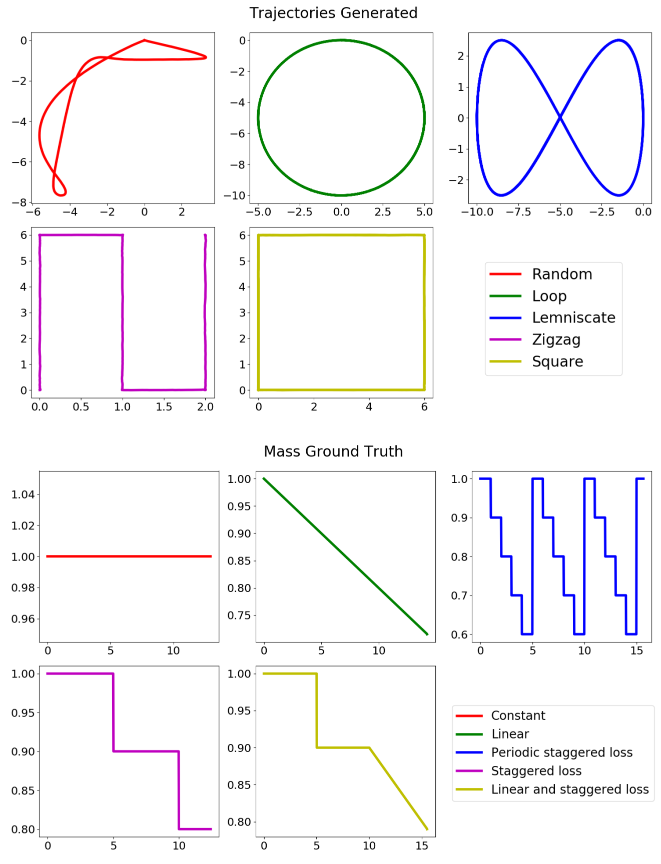


Fig. 4. Top: Horizontal view (x-y) of the simulated trajectories; Bottom: Mass behaviour (y-axis) during the time (x-axis) simulated for each trajectory

V. Results

A. Metrics

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.

$$RMSPE = \sqrt{\frac{1}{N}\left(\sum_{i=1}^{N}\left(\frac{\hat{y}_i - y_i}{y_i}\right)^2\right)} \tag{33}$$

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

In the tables found at the bottom of Figures 5-9, 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 5 to Figure 9 show the relevant results.

B. MSCKF particularization

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

$$\begin{aligned} x_I &= \begin{pmatrix} q & r & v & m \end{pmatrix} \\ x_{c_i} &= \begin{pmatrix} q_i & r_i & v_i \end{pmatrix} \end{aligned} \tag{34}$$

C. SWLS particularization

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

$$x_k = \begin{pmatrix} q_k & r_k & v_k & m_k \end{pmatrix}, k \in [0, N] \tag{35}$$

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:

$$\begin{aligned} \alpha^{b_k}_{b_{k+1}} &= \int\int_{t_k}^{t_{k+1}} R^{b_k}_{b_\tau} T d\tau^2 \\ \beta^{b_k}_{b_{k+1}} &= \int_{t_k}^{t_{k+1}} R^{b_k}_{b_\tau} T d\tau \\ \gamma^{b_k}_{b_{k+1}} &= \int_{t_k}^{t_{k+1}} \frac{1}{2} q^{b_k}_{b_\tau} \otimes \omega d\tau \end{aligned} \tag{36}$$

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:

$$\begin{aligned} \alpha^{b_k}_{i+1} &= \alpha^{b_k}_i + \beta^{b_k}_i dt + \frac{1}{2} R\left(\gamma^{b_k}_i\right) T dt^2 \\ \beta^{b_k}_{i+1} &= \beta^{b_k}_i + R\left(\gamma^{b_k}_i\right) T dt \\ \gamma^{b_k}_{i+1} &= \gamma^{b_k}_i \otimes \begin{pmatrix} 1 \\ \frac{1}{2}\omega dt \end{pmatrix} \end{aligned} \tag{37}$$

dt represents the time interval between two adjacent control input.

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

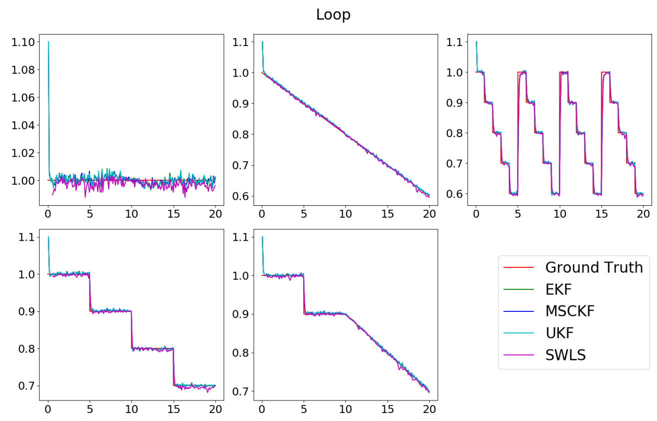
$$\begin{aligned} &\delta z = \begin{pmatrix} \delta\alpha & \delta\beta & \delta\gamma & \delta m \end{pmatrix} \\ &\delta z^{b_k}_{i+1} = F \delta z^{b_k}_i + G n \\ &F = \partial z^{b_k}_{i+1} \Big/ \partial z^{b_k}_i \\ &G = \partial z^{b_k}_{i+1} \Big/ \partial n \end{aligned} \tag{38}$$

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

$$\begin{aligned} r^k_D \begin{pmatrix} x_k & x_{k+1} \end{pmatrix} &= \begin{bmatrix} \alpha^{b_k}_{b_{k+1}} - \hat{\alpha}^{b_k}_{b_{k+1}} \\ \beta^{b_k}_{b_{k+1}} - \hat{\beta}^{b_k}_{b_{k+1}} \\ \gamma^{b_k}_{b_{k+1}} - \hat{\gamma}^{b_k}_{b_{k+1}} \\ m_{k+1} - m_k \end{bmatrix} \\ \hat{\alpha}^{b_k}_{b_{k+1}} &= m_{k+1} R^{b_k}_w \left(p^w_{b_{k+1}} - p^w_{b_k} - v^w_{b_k} dt - \frac{1}{2} g dt^2\right) \\ \hat{\beta}^{b_k}_{b_{k+1}} &= m_{k+1} R^{b_k}_w \left(v^w_{b_{k+1}} - v^w_{b_k} - g dt\right) \\ \hat{\gamma}^{b_k}_{b_{k+1}} &= q^{b_k}_w \otimes q^w_{b_{k+1}} \end{aligned} \tag{39}$$

D. Loop trajectory results

Figure 5 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	0.00504	0.00874	0.15622	0.00876	0.00747	0.00911	0.59789	0.00495
1	0.00494	0.00906	0.15615	0.00906	0.00747	0.00992	0.61060	0.00588
2	0.00493	0.02400	0.15619	0.02399	0.00750	0.02424	0.61695	0.02427
3	0.00493	0.01084	0.15620	0.01085	0.00748	0.01138	0.60585	0.00937
4	0.00492	0.00931	0.15607	0.00932	0.00748	0.00997	0.62250	0.00620

Fig. 5. Mass estimation (y-axis [kg]) vs time (x-axis [s]) for loop trajectory

E. Zig-zag trajectory results

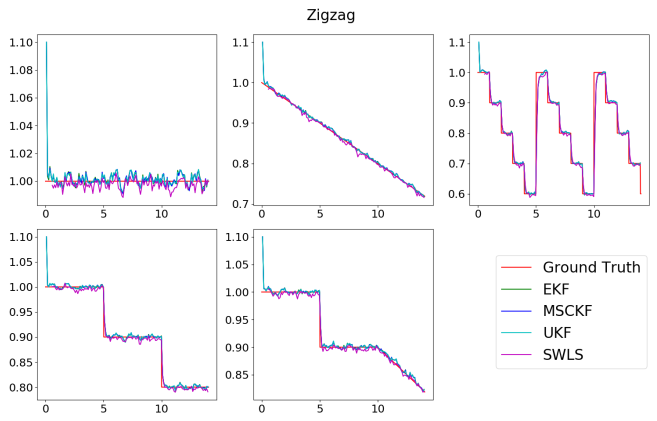
Figure 6 presents the performance of the different algorithms during the zigzag trajectory. The results are quite similar to the one obtained in the loop trajectories.Table results show again that EKF is the lighter and faster approach while the SWLS is the more accurate one in most cases. However, SWLS is again unable to capture the behaviour of changing the mass abruptly several times.

F. Square trajectory results

Figure 7 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.

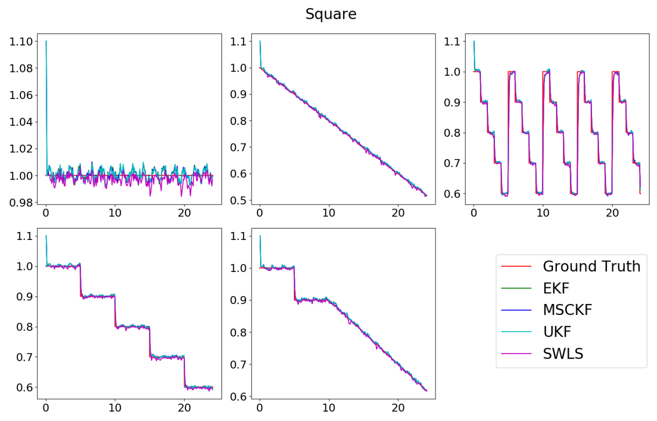
G. Random trajectory results

Figure 8 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	0.00492	0.01135	0.15307	0.01135	0.00744	0.01174	0.62838	0.00589
1	0.00501	0.01086	0.15185	0.01086	0.00744	0.01164	0.61556	0.00553
2	0.00492	0.02649	0.15089	0.02649	0.00745	0.02673	0.64101	0.02712
3	0.00492	0.01287	0.15193	0.01287	0.00741	0.01337	0.69218	0.00969
4	0.00491	0.01118	0.15130	0.01115	0.00742	0.01174	0.69586	0.00664

Fig. 6. Mass estimation (y-axis [kg]) vs time (x-axis [s]) for zig-zag trajectory



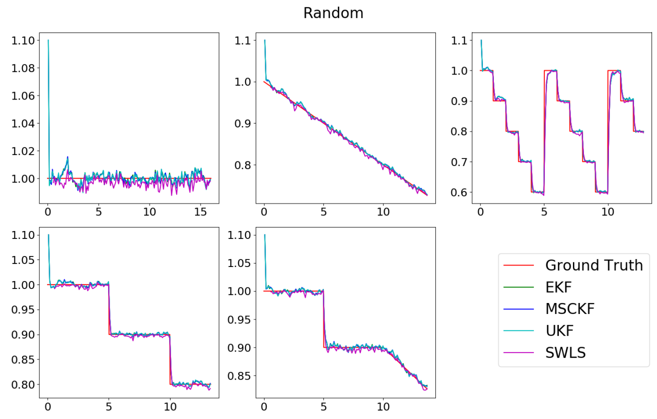
Mass	EKF		MSCKF		UKF		SWLS	
0	0.00492	0.00869	0.15275	0.00870	0.00747	0.00914	0.64775	0.00509
1	0.00494	0.00920	0.15385	0.00919	0.00744	0.01035	0.62049	0.00675
2	0.00503	0.02639	0.15369	0.02641	0.00745	0.02662	0.63488	0.02737
3	0.00533	0.01140	0.15396	0.01141	0.00747	0.01214	0.61820	0.01105
4	0.00521	0.00951	0.15299	0.00953	0.00746	0.01040	0.61794	0.00691

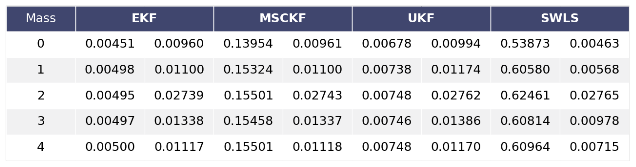
Fig. 7. Mass estimation (y-axis [kg]) vs time (x-axis [s]) for square trajectory

H. Lemniscate trajectory results

Figure 9 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.

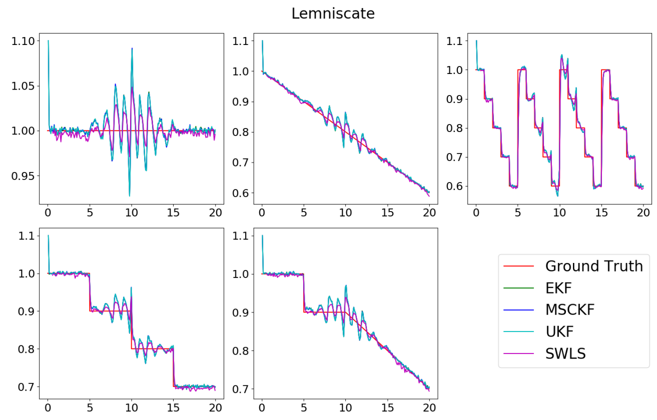
The lemniscate trajectory demands many efforts from the control perspective, and the control actions are very close to the limits of the robot. The trajectory is neither linear nor follows a constant rotational speed nor linear speed.

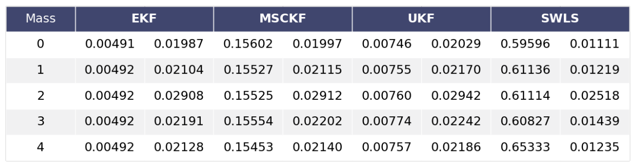




Mass	EKF		MSCKF		UKF		SWLS	
0	0.00451	0.00960	0.13954	0.00961	0.00678	0.00994	0.53873	0.00463
1	0.00498	0.01100	0.15324	0.01100	0.00738	0.01174	0.60580	0.00568
2	0.00495	0.02739	0.15501	0.02743	0.00748	0.02762	0.62461	0.02765
3	0.00497	0.01338	0.15458	0.01337	0.00746	0.01386	0.60814	0.00978
4	0.00500	0.01117	0.15501	0.01118	0.00748	0.01170	0.60964	0.00715

Fig. 8. Mass estimation (y-axis [kg]) vs time (x-axis [s]) for random trajectory





Mass	EKF		MSCKF		UKF		SWLS	
0	0.00491	0.01987	0.15602	0.01997	0.00746	0.02029	0.59596	0.01111
1	0.00492	0.02104	0.15527	0.02115	0.00755	0.02170	0.61136	0.01219
2	0.00492	0.02908	0.15525	0.02912	0.00760	0.02942	0.61114	0.02518
3	0.00492	0.02191	0.15554	0.02202	0.00774	0.02242	0.60827	0.01439
4	0.00492	0.02128	0.15453	0.02140	0.00757	0.02186	0.65333	0.01235

Fig. 9. Mass estimation (y-axis [kg]) vs time (x-axis [s]) for lemniscate trajectory

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 10 and Figure 11 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.

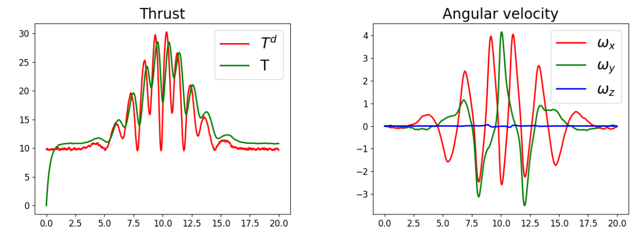


Fig. 10. Control efforts for lemniscate trajectory

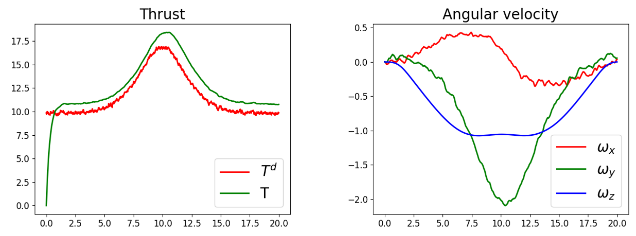


Fig. 11. Control efforts for loop trajectory

VI. Conclusions

This paper has shown 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 and SWLS. However, all of them have performed well enough.

Depending on the real-time requirements, the available computational resources, the specific trajectories to be accomplished, and previous model information, the robotic user could decide using this document as a technical guideline.

In future works, we will validate the feasibility of including the online model identification component in the overall robotic architecture (See Figure 2). We will compare how MPC performs with the classical and the proposed approach in different scenarios, including one with a system that changes its dynamic lively. Additionally, we will explore the application of data-driven techniques as a online model identification tool and compare it with the techniques proposed in this work.

Acknowledgment

This research was supported by the European Union's Horizon 2020 project SESAME (grant agreement No 101017258).

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] D. Yallappa, M. Veerangouda, D. Maski, V. Palled, and M. Bheemanna, “Development and evaluation of drone mounted sprayer for pesticide applications to crops,” in *2017 IEEE Global Humanitarian Technology Conference (GHTC)*, pp. 1–7, IEEE, 2017.

[10] S. Spoorthi, B. Shadaksharappa, S. Suraj, and V. Manasa, “Freyr drone: Pesticide/fertilizers spraying drone-an agricultural approach,” in *2017 2nd International Conference on Computing and Communications Technologies (ICCCT)*, pp. 252–255, IEEE, 2017.

[11] D. Falanga, K. Kleber, S. Mintchev, D. Floreano, and D. Scaramuzza, “The foldable drone: A morphing quadrotor that can squeeze and fly,” *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 209–216, 2018.

[12] A. Lopez-Lora, P. J. Sanchez-Cuevas, A. Suárez, A. Garofano-Soldado, A. Ollero, and G. Heredia, “Mhyro: Modular hybrid robot for contact inspection and maintenance in oil & gas plants,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1268–1275, IEEE, 2020.

[13] A. Ollero, M. Tognon, A. Suarez, D. Lee, and A. Franchi, “Past, present, and future of aerial robotic manipulators,” *IEEE Transactions on Robotics*, 2021.

[14] K. Fujii, “Extended kalman filter,” *Refernce Manual*, pp. 14–22, 2013.

[15] M. I. Ribeiro, “Kalman and extended kalman filters: Concept, derivation and properties,” *Institute for Systems and Robotics*, vol. 43, p. 46, 2004.

[16] A. I. Mourikis, S. I. Roumeliotis, *et al.*, “A multi-state constraint kalman filter for vision-aided inertial navigation.,” in *ICRA*, vol. 2, p. 6, 2007.

[17] K. Xiong, H. Zhang, and C. Chan, “Performance evaluation of ukf-based nonlinear filtering,” *Automatica*, vol. 42, no. 2, pp. 261–270, 2006.

[18] B.-Y. Choi and Z. Bien, “Sliding-windowed weighted recursive least-squares method for parameter estimation,” *Electronics Letters*, vol. 25, no. 20, pp. 1381–1382, 1989.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

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

[25] 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.