This work has been published in IEEE Robotics and Automation Letters, Vol 9, No. 3, page: 2869-2876, year: 2024.
DOI = https://doi.org/10.1109/LRA.2024.3358758

VDMNav: Software architecture for aerodynamically constrained navigation on small fixed-wing drones

Gabriel François Laupré, Aman Sharma*, and Jan Skaloud

***Abstract*— Navigation of drones is predominantly based on sensor fusion algorithms. Most of these algorithms make use of some form of Bayesian filtering with a majority employing an Extended Kalman Filter (EKF), wherein inertial measurements are fused with a Global Navigation Satellite System (GNSS), and other sensors, in a kinematic framework to yield a navigation solution (position, velocity, attitude, and time). However, the long-term accuracy of this solution is exacerbated during the absence of satellite positioning, especially for small drones with low-cost MEMS inertial sensors. On the other hand, a recently proposed vehicle dynamic model (VDM)-based navigation system has shown significant improvement in positioning accuracy during the absence of a satellite positioning solution, although in a mostly offline setting. In this article, we present the software architecture of its real-time implementation using Robot Operating System (ROS) that separates and interfaces its core from a particular hardware. The presented implementation asynchronously handles different sensor data in a modular fashion and allows i) adapting the underlying aerodynamic model, ii) including complementary sensors, and iii) reducing the dimensionality of the EKF state space at runtime without compromising the navigation performance. The real-time performance of the proposed software architecture is evaluated during long GNSS absences of up to eight minutes and compared to that of inertial coasting.**

- **Code: `https://gitlab.epfl.ch/laupre/vdm_c`**
- **Data: `https://zenodo.org/records/10337559`**

I. INTRODUCTION

Real-time control and navigation of drones are predominantly carried out by autopilots. Some examples of commercially available open-source autopilots are as follows: Emlid NAVIO/RPi3, 3D Robotics APM, Ardupilot, Pixhawk, Intel Aero Compute Board, and Paparazzi Lisa/M among others. We refer the readers to [1] to review other available options. These autopilots provide a platform for running a sensor fusion algorithm in real-time, wherein inertial measurement units (IMUs) are fused with GNSS (and other sensors such as magnetometer, Pitot tube, and barometer) in a state-space framework modeled using rigid body kinematics. The advantage of this framework is that it is independent of the choice of operating drone. However, its performance during a GNSS outage is mainly dictated by the quality of the IMU. For small drone with MEMS IMUs, position uncertainty after a minute of GNSS denial rapidly grows beyond practical use.

This project has received funding from the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No 754354..

The authors are with Environmental Sensing Observatory (formerly, Geodetic Engineering Laboratory), Laboratory of Cryospheric Sciences (CRYOS), École Polytechnique Fédérale de Lausanne (EPFL), Switzerland. *Corresponding author e-mail: `aman.sharma@epfl.ch`

Recently, [2], [3] replaced the kinematic modeling with an aerodynamic model of the drone, thereby engendering a new sensor fusion framework, whose effectiveness depends on the choice of the operating drone and the correct representation of the underlying aerodynamic model parameters. When properly accounted for, this significantly enhances navigation accuracy, especially during GNSS outages [2], [4], [5]. Hereafter, we briefly review the existing work and highlight the key areas that we address in the proposed work.

A. Related works: review and limitations

In the context of GNSS-denied navigation, existing works such as [6] and [7] have demonstrated positioning improvements on multirotor platforms and manned gliders respectively. However, the challenges posed by fixed-wing drones necessitate a more systematic approach, as evidenced in [8], showcasing considerable enhancements in relative navigation. Typically, relative navigation relies on an EKF for front-end estimation relative to the local environment, while a pose graph-based back-end is optimized with additional constraints from various sensors, such as camera/lidar, to yield global estimates. Nevertheless, the reliance on sensors such as a camera introduces challenges, especially in homogeneous terrains or foggy/poor-visibility scenarios. Besides, carrying lidar on small fixed-wing drones weighing less than 3 kg becomes impractical. Meanwhile, our work focuses on VDM-based systems, navigating without the dependency on additional sensors, especially in scenarios where environmental conditions or payload constraints could hinder traditional sensor-based systems.

1) Sensor-fusion libraries: Open source software development exploits the distributed intelligence of participants in internet communities and fosters concurrent design/testing of software modules [9]. Some of the existing software libraries that provide a framework to develop state estimators can be found in [10]–[12]. There has also been considerable work in developing sensor fusion methods using factor graphs [13], [14]. We take inspiration from [12] in terms of developing a software architecture for sensor fusion that allows the development of modular navigation filters and sensor integration strategies. However, unlike existing sensor-fusion libraries, we provide a modular framework to incorporate and exploit the aerodynamic model of the drone. We envisage that the proposed model-based navigation software architecture (so-called VDMNav) shall serve as the basis for future aerodynamically-guided real-time navigation solutions.

2) Offline VDM-based navigation system: The first loosely-coupled VDM-based navigation system for small

fixed-wing drones that showed a considerable gain in performance dates back to 2018 [2], yet in a strictly non-real-time setting. The same holds true for the first tightly-coupled implementation in 2020 [15]. There have been several other significant contributions in terms of identifying *a priori* values of aerodynamic model parameters [4] and refinement of these parameters using photogrammetry [16]. However, none of these aforementioned works were computing a navigation solution while being airborne. In accordance with [17], the correctness of a real-time computation relies not solely on the logical outcome of the computation but also on the time at which the result is produced. We, therefore, present a real-time software architecture that supports in-flight computation of a navigation solution.

3) Dimension-varying state-space: The existing offline implementation [2], [4], [16], [18] of the VDM-based navigation system is based on a 47-dimensional state space. However, while implementing the same, but in real-time, we show that similar results can be obtained with only 26 states after a short period of in-flight state estimation. During this period, we remove a subset of states, causing the state space to reduce from 47 to 26 in real-time. This leads to a dimension-varying system and as noted in [19], there are few proper tools to handle or even model such systems. In [19], the authors have rigorously introduced the concept of dimension-varying linear state space and showcased its employment for applications in control systems using simulations. Herein, we exploit the idea (not the mathematical contributions), but for a state-estimator governed by nonlinear dynamics and using a real-time environment.

B. Contributions

We have developed and experimentally tested an open-source real-time software, VDMc, that fuses aerodynamics with sensor measurements. VDMc allows modularity in i) adapting aerodynamic models for different fixed-wing drones and ii) integration of sensors. Furthermore, it allows the reduction of state-dimension at run-time for a conventional fixed-wing drone, thus lowering the computational burden.

II. VDM BASED NAVIGATION SYSTEM

We (briefly) review the VDM-based navigation system with the objective of motivating software development. The presented material is sufficient for general comprehension, however, for more details, a reader may refer to [2], [15], [18]. It is assumed that a reader is familiar with the fundamentals of Integrated navigation systems and EKF, which are discussed, for instance, in [20].

A. Notation

Let i, e, l, b denote inertial, Earth Centered Earth Fixed (ECEF), local-level, and body frames, respectively. Let $\mathbf{r}_e^l = [\phi, \lambda, h]^T$ be the drone's position in ellipsoidal coordinates (latitude, longitude, and height), $\mathbf{v}_e^l \in \mathbb{R}^3$ be its velocity in local level frame, $\mathbf{q}_b^l = [q_0, q_1, q_2, q_3]^T$ be the attitude representation using quaternions (body frame with respect to the local level frame), and $\boldsymbol{\omega}_{ib}^b \in \mathbb{R}^3$ be the angular velocity with respect to inertial frame, expressed in drone's own reference frame. In general, $\boldsymbol{\omega}_{ab}^c$ represents the angular velocity of frame b with respect to a, expressed in c. Also, $\mathbf{R}_a^b$ represent a rotation matrix from frame a to frame b; and $\boldsymbol{\Omega}_{ab}^c$ the skew-symmetric matrix of a vector $\boldsymbol{\omega}_{ab}^c$.

B. State dynamics

Most conventional navigation algorithms implement an EKF to fuse IMU and GNSS measurements. From a software point of view, EKF performs two major operations, namely, i) prediction and ii) update. For a VDM-based system, prediction solves the following system of non-linear differential equations by using Runge-Kutta method [21]:

$$\dot{\mathbf{r}}_e^l = \mathbf{D}^{-1}\mathbf{v}_e^l \tag{1}$$

$$\dot{\mathbf{v}}_e^l = \mathbf{R}_b^l\mathbf{f}^b + \mathbf{g}^l - (\boldsymbol{\Omega}_{el}^l + 2\boldsymbol{\Omega}_{ie}^l)\mathbf{v}_e^l \tag{2}$$

$$\dot{\mathbf{q}}_b^l = \frac{1}{2}\mathbf{q}_b^l \otimes \left[\boldsymbol{\omega}_{ib}^b - (\mathbf{R}_b^l)^T\left(\boldsymbol{\omega}_{ie}^l + \boldsymbol{\omega}_{el}^l\right)\right]_q \tag{3}$$

$$\dot{\boldsymbol{\omega}}_{ib}^b = \left(\mathbf{I}^b\right)^{-1}\left[\mathbf{M}^b - \boldsymbol{\Omega}_{ib}^b(\mathbf{I}^b\boldsymbol{\omega}_{ib}^b)\right], \tag{4}$$

$$\dot{a}_s = \frac{1}{\tau_s}\left(b_s u_s + c_s - a_s\right)\ \forall s \tag{5}$$

$$\dot{\mathbf{x}}_A = \mathbf{0} \tag{6}$$

Here, $\otimes$ denotes quaternion product and $[\cdots]_q$ denotes quaternion equivalent of $\boldsymbol{\omega}_{lb}^b$. $\mathbf{D}^{-1}$ denotes a transformation matrix from Cartesian to curvilinear coordinates, and $\mathbf{g}^l$ represents the local gravity field; both of these are a function of latitude and height. Most importantly, $\{\mathbf{f^b}, \mathbf{M}^b\}$, denote forces and moments and are obtained as a result of the aerodynamic model (given by Eq. (7) and Eq. (8)). IMU and GNSS measurements alongside other sensors (for instance a barometer) are considered external observations, which play a role in the update phase of the filter.

Additionally, $\mathbf{I}^b$ denotes the Inertia tensor, s denotes the set of actuators - $s \in \{$propeller, aileron, elevator, rudder$\}$, a_s denotes actuator state, u_s is the commanded actuator state whereas τ_s, b_s, c_s denote actuator parameters. $\mathbf{x}_A = \left[\mathbf{x}_p^T, \mathbf{x}_w^T, \mathbf{x}_e^T\right]^T$; $\mathbf{x}_p$ denotes aerodynamic model parameters - later defined in (16), $\mathbf{x}_w^l = [w_N,\ w_E,\ w_D]^T$ denotes wind velocity in local-level frame, $\mathbf{x}_e \in \mathbb{R}^6$ denotes IMU error states, here limited to biases.

C. Aerodynamic model of fixed-wing drones

Aerodynamic moments and forces are modeled by means of the following mathematical relations [22]:

$$\mathbf{M}^b = \begin{bmatrix} M_x^b & M_y^b & M_z^b \end{bmatrix}^T \tag{7}$$

$$\begin{aligned} \mathbf{f}^b &= \frac{1}{m_0}\left(\begin{bmatrix} F_T^b \\ 0 \\ 0 \end{bmatrix} + (\mathbf{R}_b^w)^T \begin{bmatrix} F_x^w \\ F_y^w \\ F_z^w \end{bmatrix}\right) \quad \text{with} \\ \mathbf{R}_b^w &= \begin{bmatrix} \cos\beta & \sin\beta & 0 \\ -\sin\beta & \cos\beta & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ -\sin\alpha & 0 & \cos\alpha \end{bmatrix}, \end{aligned} \tag{8}$$

The terms denote the following, F_T^b: thrust force, F_x^w: drag force, F_y^w: side force, F_z^w: lift force; M_x^b: roll moment, M_y^b:

pitch moment and M_z^b: yaw moment with α and β denoting the angle of attack and side-slip. Here,

$$F_T^b = \rho n^2 D^4 \left(C_{F_T1} + C_{F_T2} J + C_{F_T3} J^2\right) \quad (9)$$

$$F_x^w = \bar{q} S \left(C_{F_x1} + C_{F_x\alpha}\alpha + C_{F_x\alpha 2}\alpha^2 + C_{F_x\beta 2}\beta^2\right) \quad (10)$$

$$F_y^w = \bar{q} S \left(C_{F_y1}\beta\right) \quad (11)$$

$$F_z^w = \bar{q} S \left(C_{F_z1} + C_{F_z\alpha}\alpha\right) \quad (12)$$

$$M_x^b = \bar{q} S b \left(C_{M_xa}\delta_a + C_{M_x\beta}\beta + C_{M_x\tilde{\omega}_x}\tilde{\omega}_x + C_{M_x\tilde{\omega}_z}\tilde{\omega}_z\right) \quad (13)$$

$$M_y^b = \bar{q} S \bar{c} \left(C_{M_y1} + C_{M_ye}\delta_e + C_{M_y\tilde{\omega}_y}\tilde{\omega}_y + C_{M_y\alpha}\alpha\right) \quad (14)$$

$$M_z^b = \bar{q} S b \left(C_{M_z\delta_r}\delta_r + C_{M_z\tilde{\omega}_z}\tilde{\omega}_z + C_{M_z\beta}\beta\right), \quad (15)$$

Here $\mathbf{V}^b = \begin{bmatrix} V_x & V_y & V_z \end{bmatrix} = \mathbf{R}_l^b \left(\mathbf{v}_e^l - \mathbf{x}_w\right)$ denotes relative air velocity and $V = ||\mathbf{V}^b||$ is the airspeed, $\alpha = \arctan \frac{V_z}{V_x}$ and $\beta = \arcsin \frac{V_y}{V}$. ρ denotes air density and $\bar{q} = \frac{1}{2}\rho V^2$ is dynamic pressure. $b, S, \bar{c}, D$ are wing span, wing surface, mean aerodynamic chord, and propeller diameter, respectively. Note that $J = V/(D\pi n)$ with n denoting propeller speed. $\delta_a, \delta_e, \delta_r$ denote aileron, elevator, and rudder deflections, respectively. The non-dimensional angular velocity is given by $\tilde{\omega}_x = b\omega_x/(2V)$, $\tilde{\omega}_y = \bar{c}\omega_y/(2V)$, and $\tilde{\omega}_z = b\omega_z/(2V)$, where $\boldsymbol{\omega}_{lb}^b = [\omega_x \ \omega_y \ \omega_z]^T$. Altogether, the vector of aerodynamic model parameters is given as follows:

$$\mathbf{x}_p = \begin{bmatrix} \mathbf{C}_F^T & \mathbf{C}_M^T \end{bmatrix}^T \quad \text{where,} \quad (16)$$

$$\mathbf{C}_F = [C_{F_T1} \ C_{F_T2} \ C_{F_T3} \ C_{F_x1} \ C_{F_x\alpha} \ C_{F_x\alpha 2} \ C_{F_x\beta 2} \ C_{F_y1} \ C_{F_z1} \ C_{F_z\alpha}]^T$$

$$\mathbf{C}_M = [C_{M_xa} \ C_{M_x\beta} \ C_{M_x\tilde{\omega}_x} \ C_{M_z\tilde{\omega}_z} \ C_{M_y1} \ C_{M_ye} \ C_{M_y\tilde{\omega}_y} \ C_{M_x\alpha} \ C_{M_z\delta_r} \ C_{M_z\tilde{\omega}_z} \ C_{M_z\beta}]^T$$

D. Observation models

The update phase of EKF is based on the observation (or measurement) models of IMU and GNSS as provided below:

1) IMU:

$$\mathbf{Z}^{f_{imu}} = \mathbf{f}^b + \left(\dot{\boldsymbol{\Omega}}_{ib}^b + \boldsymbol{\Omega}_{ib}^b \boldsymbol{\Omega}_{ib}^b\right) \mathbf{r}_{bI}^b + \mathbf{b}_f + \boldsymbol{\eta}_f \quad (17)$$

$$\mathbf{Z}^{\omega_{imu}} = \boldsymbol{\omega}_{ib}^b + \mathbf{b}_\omega + \boldsymbol{\eta}_\omega \quad (18)$$

where $\mathbf{f}^b$ is defined by Eq. (8), $\dot{\boldsymbol{\Omega}}_{ib}^b$ is defined by Eq. (4), $\mathbf{r}_{bI}^b$ denotes lever arm between IMU and body frame, $\{\mathbf{b}_f, \mathbf{b}_\omega\}$ denote accelerometer and gyroscope bias respectively; $\boldsymbol{\eta}_f$ and $\boldsymbol{\eta}_\omega$ denote white noise.

2) GNSS:

$$\mathbf{Z}^{G_r} = \mathbf{r}_e^l + \mathbf{R}_l^e \mathbf{R}_b^l \mathbf{r}_{bG}^b + \boldsymbol{\eta}_r \quad (19)$$

$$\mathbf{Z}^{G_v} = \mathbf{v}_e^l + \left(\mathbf{R}_b^l \boldsymbol{\Omega}_{ib}^b - \boldsymbol{\Omega}_{ie}^l \mathbf{R}_b^l\right) \mathbf{r}_{bG}^b + \boldsymbol{\eta}_v \quad (20)$$

where $\mathbf{r}_{bG}^G$ denotes the lever arm between GNSS antenna and body frame; $\boldsymbol{\eta}_r$ and $\boldsymbol{\eta}_v$ denote white noise.

E. State space summary

The full state vector of the VDM-based navigation filter consists of 47 elements: position (3), velocity (3), attitude (4, but its linearized version consists of only 3 parameters), angular velocity (3), aerodynamic model parameters (21), actuator dynamics (4), wind (3), IMU biases (6).

Fig. 1. Left: Preparation for takeoff of TP2 (top), developed payload (bottom). Right: Ground control station

III. Hardware

We use an in-house developed conventional fixed-wing drone referred to as *TP2*- shown in Fig. 1. This platform has been used in previous research works [2], [4], [5], [16] but with other GNSS and IMU sensors for offline navigation. For this work, TP2 hosts a custom payload, an open-source autopilot (PX4), a communication interface, and different sensors for the purposes of data acquisition, control, and navigation. Fig. 2 illustrates a broad overview of the connections between various hardware components, with further elaboration provided later in this section.

A. Payload

The payload, shown in Fig. 1, comprises:

1) IMU, namely STIM 318.
2) Sentiboard [23] - a general board for time-tagging the IMU data in GPS time.
3) On-board computer [24] and other unused components.

B. Other Sensors

1) GNSS receiver: A multi-constellation, multi-frequency Topcon B125 GNSS receiver, and 3-frequency antenna, which gathers both code and phase measurements, and transmits its real-time position and velocity, the pulse-per-second (PPS), and other messages related to time to the Sentiboard (thus to the onboard computer) and autopilot.

2) Pitot tube: A hobby-plane compatible Pitot tube and pressure sensor [25] is mounted on the TP2. The data stream from this sensor is connected directly to the autopilot.

C. Autopilot

We use an open-source autopilot called PixHawk [26] (v.2) with a customized PX4 firmware. The firmware modification enables synchronization between the autopilot and the GNSS receiver, allowing accurate tagging of all autopilot data, particularly control commands (CC), with GPS timestamps.

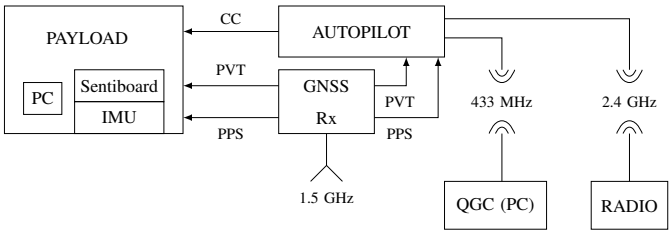

Fig. 2. General connection between hardware components, where CC: control commands, PPS: pulse per second, PVT: position velocity time, Rx: Receiver, QGC: QGroundControl

D. Communication

Two frequencies are utilized for communication with TP2:

1) 433 MHz: communication with a Ground Control Station (see Sec. III-E) for issuing commands to introduce a GNSS outage, and plotting the navigation solution.
2) 2.4 GHz: communication with the operator's remote radio to manually change the flight modes of the drone. This is used especially during take-off and landing.

E. Ground Control Station

The Ground Control Station (GCS), shown in Fig. 1, comprises a field computer and antennas to maintain duplex communication with the drone throughout the mission. The GCS PC uses QGroundControl [27] (QGC) as it provides a complete flight control and monitoring setup for the linked autopilot (`PX4`) via the MAVLink protocol [1]. QGC is used for i) mission planning, ii) sending additional commands to the drone to introduce a GNSS outage in real-time, and iii) monitoring the vehicle position by VDM-based and conventional INS/GNSS systems with respect to the reference.

IV. Architecture of Model-Based Navigation (VDMc)

This section introduces the proposed software architecture, the so-called VDMc, for fusing drone aerodynamics with sensor measurements. A schematic of VDMc is presented in Fig. 3, showcasing five major elements (classes) in blue, namely i) EKF, ii) VDM - TP2, iii) resource manager, iv) IMU model, and v) GNSS model. In most general terms, VDMc receives the sensor data (for example, IMU, GNSS position, and velocity) and flight control commands (CC) via ROS topics and yields a navigation solution as the output (noted as VDMc output in the figure). As VDMc uses aerodynamic forces/moments that are only experienced while being airborne, the EKF has to be initialized in the air [2] and this is done via a conventional sensor fusion algorithm denoted as Init (GiiNav) in the figure. The IMU, GNSS, and CC data are managed by the class resource manager, which places the asynchronously received data and associated GPS timestamp in a circular buffer. The buffered data is subsequently processed by a dedicated class that rests in the resource manager, namely the timer, which ensures

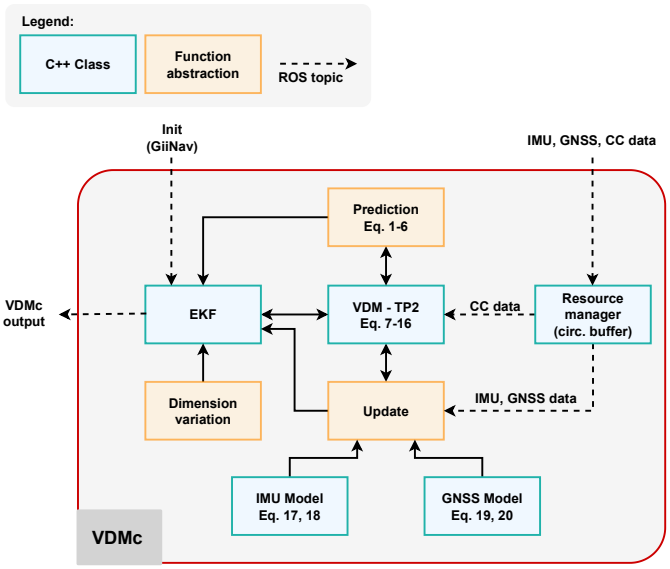

Fig. 3. VDMc node

the correct sequence of prediction and update steps by the EKF. More information on this asynchronous processing of sensor data is provided in Sec. IV-C. To continue, the implementation of EKF-prediction and -update equations [20] involve matrix operations, however, the computation of these matrices depends on the i) process model or state dynamics - Eq. (1)-(6), ii) observation models - Eq. (17)-(20) and iii) the Jacobians of the above models, which are altogether platform-specific. Thus, we have separated the implementation into i) platform-independent code, for instance, Eq. 1-6 in addition to other prediction and update operations, and ii) platform-dependent code encapsulated in the VDM-TP2 class that carries the structure of the model (Eq. 7-15). Furthermore, each sensor that leads to an EKF update is defined as a class, for instance, IMU model and GNSS model in the figure. Each such class consists of the code to compute the Jacobian and other entities as required by the update step.

A. Automated generation of linearized models

We generate C++ code for linearizing the state dynamics (Eq. (1) to (15)) and observation models (Eq. (17) to (20)), as required by the EKF in VDMc, via the following steps [28]:

1) Symbolic representation: The state dynamics involving 47 states and the observation models, described by 3 nonlinear equations for GNSS position and velocity respectively, and 6 nonlinear equations for the IMU are symbolically written using Mathematica[2].
2) Jacobian computation: Analytical partial derivatives (Jacobian) of the state dynamics and observation models are computed using Mathematica for linearization.
3) Code transformation: The computed models and their respective Jacobians are transformed into C++ code using Python and Mathematica.

[1] `https://mavlink.io/en/`

[2] `https://www.wolfram.com/mathematica/`

The C++ code is generated on a PC with an Intel i7 processor using the above steps; this code is then transferred to the onboard computer, where it is subsequently compiled.

B. Modular adaptation of aerodynamic models

The implementation uses an aerodynamic model of a fixed-wing drone (Eq. (9) to (15)) to derive state dynamics, observation models, and Jacobians. However, different drone platforms, like delta wings [29], require specific aerodynamic models due to different actuators, parameters, and functional structures. For instance, delta wings have two control surfaces as opposed to three on a conventional fixed-wing drone (see Appendix A). The above Mathematica workflow handles this by i) employing modular blocks for variable dimensionality of the actuators and model parameters ($\mathbf{x}_p$), and ii) a separate block for aerodynamics (Eq. (9)-(15)); with this and the platform-specific class (see Sec. IV), the developed architecture can be adapted to a different drone, as explained in [28].

C. Handling asynchronous sensor measurements

As sensor measurements are asynchronously received over ROS, the occurrence of delays or packet drops cannot be ignored and the real-time software needs to handle such situations efficiently. One naive solution is to ignore any data that is not received at the expected time or wait for the required data to perform predictions or updates, leading to idle time for the filter. This is undesirable in real-time applications. To address this concern, the filter states and covariance are saved at prediction and update times (i.e. when sensor data is expected), and later when the data arrives with a delay, a delayed filter update is performed using the stored states and covariance. This approach requires the onboard computer to perform filter computations faster than in real-time to catch up with the current time. This method is called back-propagation of state and covariance. It is implemented in the timer class (part of the resource manager - as discussed earlier in Sec. IV), thereby ensuring a continuous flow of the navigation solution even if sensor data are missing. The timer implementation acts as a heartbeat, defining the waiting time for expected data arrival and the maximum time for which the saved states and covariance are kept in memory, altogether specified in a configuration file.

D. Dimension varying sensor fusion - DVSF

The existing-offline implementation of VDM-based system [2], [4], [5], [16] for a conventional fixed-wing drone is based on 47 states (see Sec. II-E), of which 21 represent the aerodynamic model parameters. Although these parameters are coarsely constant, with the changes in payload placement, wearing of the airframe, changes in flight conditions, wind, and other external factors, their values can change [5]. Therefore, if the drone is sufficiently maneuvered for a reasonably long duration after the takeoff, these parameters need not be refined (re-estimated) further than a given time duration and thus can be removed from the state vector. To implement this, a parameter in the configuration file is set to

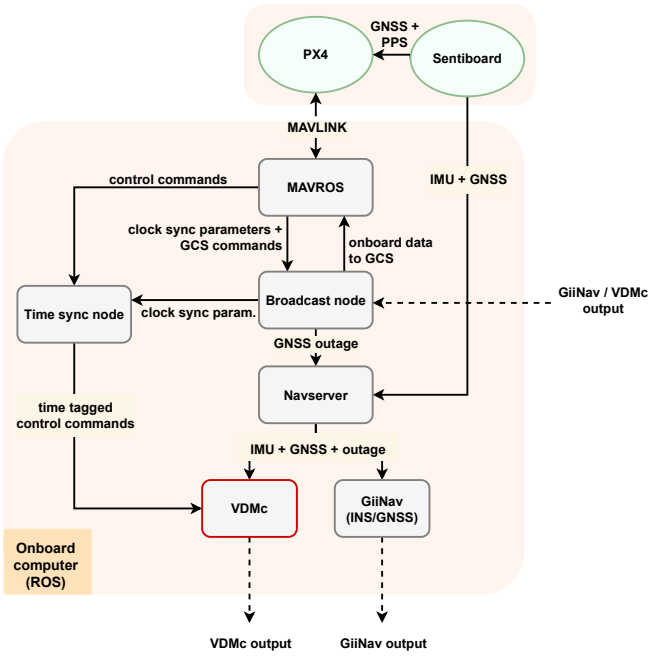

Fig. 4. Architecture of VDMNav highlighting the placement of VDMc

a duration-post-initialization, after which the state reduction is carried out during flight. From a software standpoint, invoking such a subroutine involves the following steps: 1) Blocking the critical section to pause EKF prediction and update operations. 2) Initializing the new state vector and covariance matrix with lower dimension using the old state vector and covariance matrix with higher dimension. 3) Initializing the saved states and covariance matrix buffer accordingly. 4) Notifying the a) sensor model and the b) VDM platform class about the state change. 5) Unblocking the critical section and returning to normal operation with a low-dimensional state and covariance.

E. Inclusion of complementary sensors

From a software perspective, sensor functionality is encapsulated through the resource manager (described earlier in Sec. IV) and a sensor model class. Currently, three subclasses exist for the TP2 platform: IMU, GNSS position, and GNSS velocity. To implement a new sensor, the following steps should be followed [28]: 1) Create a subclass in the resource manager to buffer the data along with its timestamp. 2) Generate C++ code for the observation model and its Jacobian (Sec. IV-A). 3) Integrate this code with the sensor model class of the new sensor. 4) Create a timer for the sensor to handle EKF update synchronization and enable saving states and covariances in case of missing or delayed measurements. 5) Link the class representing the sensor definitions and methods to the specific platform type.

F. ROS Nodes

Real-time navigation, in a holistic sense, is carried out by interfacing VDMc with other software modules or ROS nodes (see Fig. 4), details of which are presented here.

1) MAVROS: MAVROS [30] provides a communication interface between PX4 autopilot and general MAVLink communication protocol. MAVROS decodes incoming MAVLink data stream and publishes it on equivalent ROS topics.

2) Broadcast node: It is responsible for i) streaming the navigation solution from VDMc carried by the on-board computer to the GCS ii) reception of custom commands from the GCS to introduce GNSS outage in real-time and iii) reception of autopilot time and clock synchronization parameters and passing it to Time Sync node (see below).

3) Time Sync node: This node re-tags the autopilot data, such as control commands, in GPS time. It combines received topics with information from the broadcast node to associate a GPS timestamp with the control commands. Mathematically, this involves continuously computing a bias and scale factor between the GPS time and autopilot time.

4) NavServer: NavServer retrieves IMU and GNSS data from Sentiboard and publishes it on ROS topics. It also invokes the GPS message publisher to simulate a GPS outage upon receiving a custom command from the GCS.

5) GiiNav: Originally a software utilizing an Extended Kalman Filter, GiiNav [31], [32], has been adapted for ROS compatibility. It subscribes to ROS topics for GNSS and IMU data and publishes the navigation solution on another topic. GiiNav also plays a crucial role in initializing the VDM-based navigation system while TP2 is airborne.

6) VDMc: VDMc is the ROS node running the VDM-based navigation estimator (detailed in Sec. IV) wherein the different sensor measurements are fused with the platform aerodynamics to yield a navigation solution.

V. Results

A. Pseudo-real time navigation solution

For offline testing, VDMc and GiiNav are evaluated using recorded flight (ROSbag) data from *OFF_STIM13* in a pseudo-real-time environment. This recording is carried out by the on-board computer. To simulate a GNSS outage, the ROSbag API is utilized to remove GNSS-related ROS topics for a duration of two minutes. The accuracy of trajectory estimation during this outage is compared between the conventional INS/GNSS (GiiNav) solution (red) and VDMc with 47 states (cyan) in Fig. 5. The trajectory of the VDMc implementation closely follows the reference trajectory, while the free INS solution drifts rapidly. Next, a change in state space dimension (from 47 to 26) is triggered after 10 minutes of flight (setting in the configuration file). It should be noted that this 10-minute duration was empirically

TABLE I
VDMc performance summary

Flight	Duration [min]	Outage [min]	RMS error [m] INS	RMS error [m] VDMc	Max error [m] INS	Max error [m] VDMc	Transition time [s]
RTF_STIM14	28	5	680	117	2684	210	5.5
		5	551	280	2016	767	
RTF_STIM15	16	8	3057	448	12745	943	6.3
OFF_STIM13	23	2	189	68	117	621	1.0

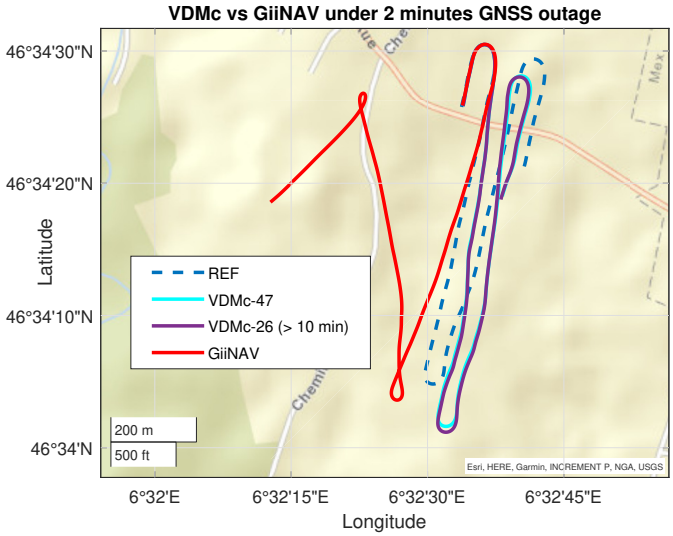

Fig. 5. Pseudo real-time implementation of VDMc for flight *OFF_STIM13*

found after testing on multiple outages for the conventional fixed-wing drone. The trajectory estimated by VDMc with the dimension-varying feature (in purple) closely resembles the trajectory obtained with the full 47 states. As the ROSbag replays the recorded flight data in the order they were received, this test validates the correct functioning of other ROS modules.

B. Real-time ***airborne*** navigation solution

The two real-time flights, referred to as *RTF_STIM14* and *RTF_STIM15*, are used to validate the presented VDMc implementation. The configuration of VDMc is set to introduce a state-space dimension variation from 47 to 26 after 10 minutes of flight. A real-time GNSS outage is introduced by the operator via the GCS. We then examine the positioning accuracy of VDMc and inertial coasting (GiiNav) in comparison to the autopilot (PX4) that uses uninterrupted GNSS reception, thus providing the reference. VDMc, GiiNav, and PX4 log their respective navigation solution to a file in addition to broadcasting them to the GCS where it is displayed. Here, we plot the logged solution by these modules without any additional processing for *RTF_STIM14* and *RTF_STIM15* flights in Fig. 6(a)(b)(c). *RTF_STIM14* is a 28-minute long flight, where a GNSS outage is introduced after 9 minutes of flight time for a duration of 5 minutes. The performance of VDMc (with automatic state reduction feature) during this outage is shown in Fig. 6(a). Subsequently, the GNSS position/velocity reception is restored. Thereafter, another outage of 5 minutes after 18 minutes of flight time is introduced. The performance of VDMc during this outage is shown in Fig. 6(b). A similar test is carried out for a manually controlled flight *RTF_STIM15*, lasting 16 minutes, with a longer outage of $\sim$ 8 minutes after 6 minutes of being airborne; the estimated trajectory is shown in Fig. 6(c). Additionally, the evolution of the horizontal positioning error for the three test scenarios is depicted in Fig. 6(d).

The duration of outages introduced in this study is consid-

erably longer than prior ones [2], [4], [15], [16]. A statistical summary, based on i) RMS of the absolute trajectory error (ATE) [33] and ii) maximum error in the horizontal direction for the three flights is presented in Tab. I; the tabulated results show significant mitigation of the positioning drift. Furthermore, a summary of the computational burden of the VDMc operations are tabulated in Tab. II. Also, it took 5 ms and 6 ms to switch from 47 to 26 states for *RTF_STIM14* and *RTF_STIM15* respectively. All the above time-values are less than 10 ms - nominal operating period of the EKF.

TABLE II
COMPUTATIONAL BURDEN OF VDMC

Operation	**47-states** [ms]	**26-states** [ms]
Prediction	2.5	0.8
Update - IMU	0.8	0.4
Update - GNSS pos.+vel.	1.5	0.6

VI. CONCLUSIONS AND FUTURE WORKS

We have presented a modular software architecture for fusing flight aerodynamics with sensor measurements via EKF in real-time. This architecture can i) asynchronously handle data from different sensors, ii) compute a navigation solution, iii) introduce a GNSS outage on a request from the GCS, and iv) change the dimensionality of state space from 47 to 26 states at run-time for the conventional fixed-wing drone. The proposed framework has been tested on the field under extended GNSS outages of up to 8 minutes to assess the real-time performance. The results show excellent mitigation of positioning drift of around an order of magnitude compared to inertial coasting. To show the framework's modularity, we have also presented its usage for a delta-wing drone. In the future, integration of VDM with visual-inertial odometry, and relative navigation frameworks could be carried out. On the other hand, a focus on finding a VDM for quadcopters would be of prime importance, followed by its incorporation into the VDMc framework. We envisage that the presented software architecture shall be a starting point for spreading the benefit of aerodynamically guided navigation strategies on small drones. Furthermore, the authors can assist readers in using this framework for their applications.

REFERENCES

[1] O. D. Dantsker and R. Mancuso, "Flight data acquisition platform development, integration, and operation on small-to medium-sized unmanned aircraft," in *AIAA Scitech 2019 Forum*, p. 1262, 2019.

[2] M. Khaghani and J. Skaloud, "Assessment of vdm-based autonomous navigation of a uav under operational conditions," *Robotics and Autonomous Systems*, vol. 106, pp. 152–164, 2018.

[3] M. Khaghani and J. Skaloud, "Autonomous and non-autonomous dynamic model based navigation system for unmanned vehicles," 2016. Patent US20160364990A1.

[4] A. Sharma, G. Laupre, and J. Skaloud, "Identifying aerodynamics of small fixed-wing drones using inertial measurements for model-based navigation," *Navigation: Journal of The Institute of Navigation*, vol. 70, no. 4, 2023.

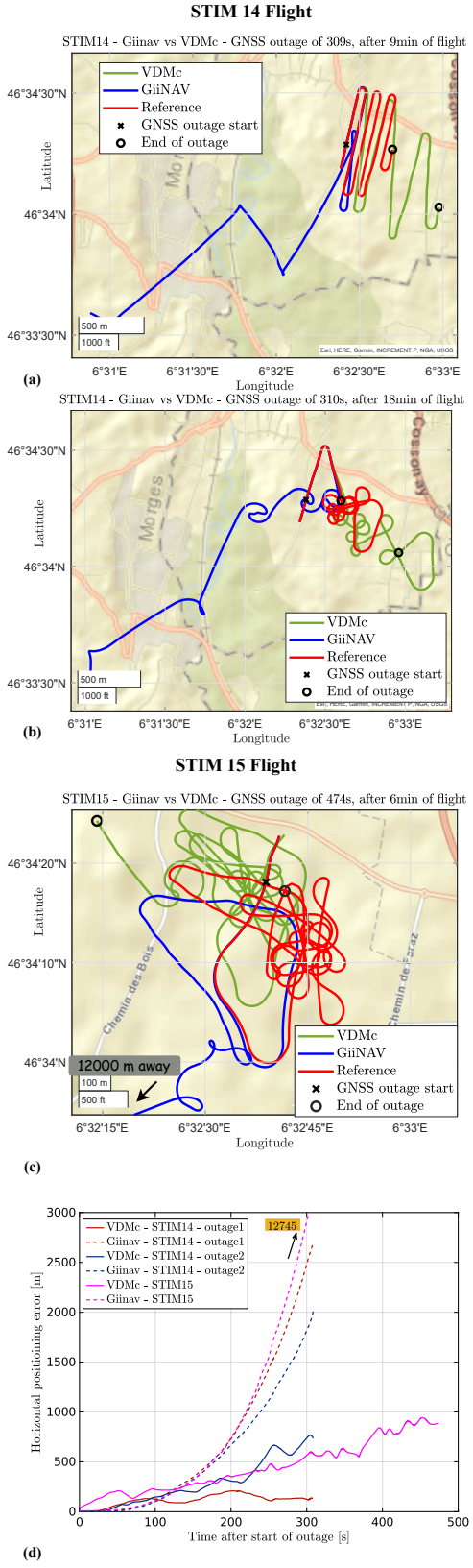

Fig. 6. Trajectory estimation for (a) RT_STIM14 during GNSS outage -1 (b) RTF_STIM14 during GNSS outage - 2 (c) RTF_STIM15 during a GNSS outage (d) Positioning error curve for the above outages

[5] G. Laupré, L. Pirlet, and J. Skaloud, "Reliable strategies for implementing model-based navigation on fixed-wing drones," *Royal Institute of Navigation*, 2023. In press.
[6] B. P. Lewis and R. W. Beard, "A framework for visual return-to-home capability in gps-denied environments," in *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 633–642, IEEE, 2016.
[7] S. Leutenegger and R. Y. Siegwart, "A low-cost and fail-safe inertial navigation system for airplanes," in *2012 IEEE International Conference on Robotics and Automation*, pp. 612–618, IEEE, 2012.
[8] G. Ellingson, K. Brink, and T. McLain, "Relative navigation of fixed-wing aircraft in gps-denied environments," *NAVIGATION: Journal of the Institute of Navigation*, vol. 67, no. 2, pp. 255–273, 2020.
[9] B. Kogut and A. Metiu, "Open-source software development and distributed innovation," *Oxford review of economic policy*, vol. 17, no. 2, pp. 248–264, 2001.
[10] S. Michael, *Bayes++ Bayesian Filtering*, 2020.
[11] R. R. Labbe, *Kalman and Bayesian Filters in Python.*, 2018.
[12] K. Kauffman, D. Marietta, J. Raquet, D. Carson, R. C. Leishman, A. Canciani, A. Schofield, and M. Caporellie, "Scorpion: A modular sensor fusion approach for complementary navigation sensors," in *2020 IEEE/ION Position, Location and Navigation Symposium (PLANS)*, pp. 156–167, IEEE, 2020.
[13] T. Pfeifer and Others, "librsf." `https://github.com/TUC-ProAut/libRSF`, 2022.
[14] F. Dellaert and G. Contributors, "borglab/gtsam." `https://github.com/borglab/gtsam`, 2022.
[15] H. A. Mwenegoha, T. Moore, J. Pinchin, and M. Jabbal, "A model-based tightly coupled architecture for low-cost unmanned aerial vehicles for real-time applications," *IEEE Access*, 2020.
[16] G. Laupré and J. Skaloud, "Calibration of fixed-wing uav aerodynamic coefficients with photogrammetry for vdm-based navigation," in *Proceedings of the 2021 International Technical Meeting of The Institute of Navigation*, pp. 775–786, 2021.
[17] J. A. Stankovic, "Misconceptions about real-time computing: A serious problem for next-generation systems," *Computer*, vol. 21, no. 10, pp. 10–19, 1988.
[18] M. Khaghani and J. Skaloud, "Autonomous vehicle dynamic model-based navigation for small uavs," *Navigation: Journal of The Institute of Navigation*, vol. 63, no. 3, pp. 345–358, 2016.
[19] D. Cheng, H. Qi, and Z. Liu, "Linear system on dimension-varying state space," in *2018 IEEE 14th International Conference on Control and Automation (ICCA)*, pp. 112–117, IEEE, 2018.
[20] P. D. Groves, *Principles of GNSS, Inertial, and Multisensor Integrated Navigation Systems*. Artech House, 2013.
[21] J. Butcher, "A history of runge-kutta methods," *Applied Numerical Mathematics*, vol. 20, no. 3, pp. 247–260, 1996.
[22] G. J. Ducard, "Nonlinear aircraft model," *Fault-tolerant Flight Control and Guidance Systems: Practical Methods for Small Unmanned Aerial Vehicles*, pp. 27–41, 2009.
[23] S. M. Albrektsen and T. A. Johansen, "User-configurable timing and navigation for uavs," *Sensors*, vol. 18, no. 8, p. 2468, 2018.
[24] A. E. B.V., "Up board series." `https://up-board.org/up/specifications/`, 2022. Accessed: 2023-08-25.
[25] J. Drones, "J drones." `https://wiki.jdrones.com/uav/asp4525plain`, 2022. Accessed: 2022-10-11.
[26] L. Meier, P. Tanskanen, L. Heng, G. H. Lee, F. Fraundorfer, and M. Pollefeys, "PIXHAWK: A Micro Aerial Vehicle Design for Autonomous Flight using Onboard Computer Vision," *Autonomous Robots*, vol. 33, no. 1-2, pp. 21–39, 2012.
[27] I. Dronecode Project, "Qground control." `http://http://qgroundcontrol.com/`, 2019. Accessed: 2023-08-22.
[28] G. Laupré, A. Sharma, and S. Gilgien, "Vdm_c." `https://gitlab.epfl.ch/laupre/vdm_c/`, 2020. Accessed: 20223-12-10.
[29] P. Longobardi and J. Skaloud, "On the scalability of experimentally determined aerodynamic model for model-based navigation on a delta-wing uav," in *2023 IEEE 10th International Workshop on Metrology for AeroSpace (MetroAeroSpace)*, pp. 19–24, 2023.
[30] V. Ermakov, "Mavlink extendable communication node for ros with proxy for ground control station." `http://wiki.ros.org/mavros`, 2022. Accessed: 20223-12-29.
[31] P. Tomé, *Integration of Inertial and Satellite Navigation Systems for Aircraft Attitude Determination*. PhD thesis, University of Oporto, Oporto, Portugal, 2002.
[32] J. Skaloud, P. Schaer, Y. Stebler, and P. Tomé, "Real-time registration of airborne laser data with sub-decimeter accuracy," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 65, no. 2, pp. 208–217, 2010.
[33] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of rgb-d slam systems," in *2012 IEEE/RSJ international conference on intelligent robots and systems*, pp. 573–580, IEEE, 2012.

APPENDIX

A. VDMc for Delta-wing drone

The modularity of VDMc [28] is shown by adapting it to a delta-wing drone, *ConcordeS* (Fig. 7), with two control surfaces, the elevons, as opposed to three on a conventional drone. A VDM of such a drone is characterized as:

$$F_x^w = \bar{q}S\left(C_{F_x0} + C_{F_x\alpha 2}\alpha^2\right) \tag{21}$$
$$F_y^w = \bar{q}S\left(C_{F_y0} + C_{F_y\beta}\beta\right) \tag{22}$$
$$F_z^w = \bar{q}S\left(C_{z0} + C_{F_z\alpha}\alpha\right) \tag{23}$$
$$M_x^b = \bar{q}Sb\left(C_{Mx0} + C_{Mx\delta_a}\delta_a\right) \tag{24}$$
$$M_y^b = \bar{q}S\bar{c}\left(C_{My0} + C_{Mx\delta_e}\delta_e\right) \tag{25}$$
$$M_z^b = \bar{q}Sb\left(C_{Mz\beta}\beta\right), \tag{26}$$

Here, $\delta_a = \frac{\delta_L - \delta_R}{2}, \delta_e = \frac{\delta_L + \delta_R}{2}$ with δ_L, δ_R denoting the

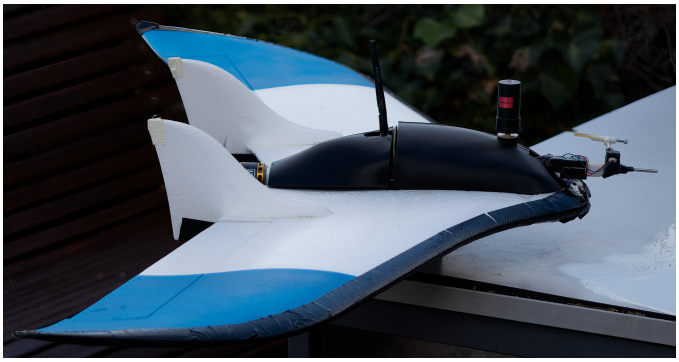

Fig. 7. Delta-wing drone

deflections of left and right control surfaces respectively. The software implementation has been done similarly to *TP-2*. Here, we contrast the performance of VDMc against INS for a 2-minute outage in Fig. 8; the result shows a similar performance as that of a conventional drone.

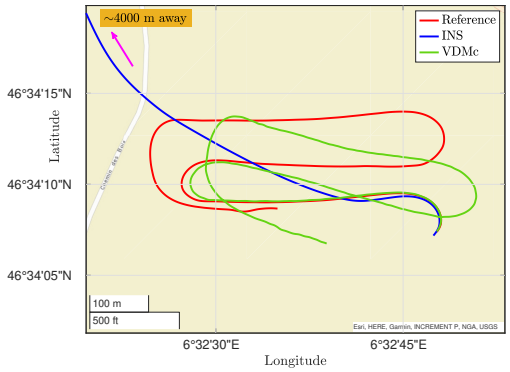

Fig. 8. VDMc vs INS for a delta-wing drone: 2-minute GNSS outage