Flow-Based Programming for Real-Time Multi-Sensor Data Fusion

Thomas Topp, Thomas Hobiger, *Institute of Navigation, University of Stuttgart, Germany*

BIOGRAPHY

Thomas Topp Thomas Topp, M.Sc., studied aerospace engineering at the University of Stuttgart, Germany and graduated with a master of science in 2019. Since 2020 he is working at the Institute of Navigation as a PhD candidate, in the course of which he created the sensor-fusion software INSTINCT and is its current lead developer. His field of expertise covers INS/GNSS sensor fusion, inertial navigation, filtering techniques and C++ programming. He is an active member of the open source community on GitHub.

Thomas Hobiger Prof. Hobiger is the head of the Institute of Navigation at the University of Stuttgart, Germany. His main research focuses on PNT and deals with autonomous flying, GNSS-R, software-defined radio, high-performance computing, propagation of radio waves, and time and frequency transfer. Prof. Hobiger is an IAG Fellow and was a recipient of the AGU Geodesy Section Award, in 2018. He is a recipient of the Tsuboi Award, the EPS Award 2010, and the EGU Outstanding Young Scientist Award 2011.

ABSTRACT

Flow-based programming (FBP) splits software functionality into modules which are triggered by data elements flowing from one module to the next. Thus, modules, which are not directly dependent on data from each other, can run independently and applications can be parallelized on a basic level what is a major improvement for the performance of multi-sensor data fusion algorithms. This paper discusses the flow-based software INSTINCT, which is a solid framework for implementing PNT algorithms and helps to reduce development times by making algorithms reusable. It provides an intuitive graphical user interface (GUI) that makes it usable in research and teaching. Inside the software, a variety of file formats and sensors for IMU und GNSS data processing are implemented. The realization of an INS/GNSS loosely-coupled Kalman filter flow is discussed and the results are compared to a flow representing a single point positioning (SPP) solution. Finally, a performance study of the algorithm on a 4th generation Intel CPU is presented, demonstrating that real-time capabilities can be ensured even on older systems.

I. INTRODUCTION

Designing a software for real-time multi-sensor data fusion is a challenging task. Not only does such a software have to provide correct and utmost accurate results, but it also has to handle the input from various sensors several hundred of times per second. Most often researchers dealing with the task of designing such a software have to start from scratch or refactor code that they have previously used in other projects. Especially researchers who just started their career and do not have access to even basic algorithms have to write everything from scratch. Thus, it is not surprising that even at the same facilities the same algorithms are written over and over again. In addition, it is hard to ensure real-time processing capabilities without being an expert in navigation and programming. Therefore, the lack of a strong software design background leads to very time-consuming implementation phases and takes a heavy toll on the time available for research.

In this paper, a new approach for PNT software design, based on flow-based programming (FBP), is presented that helps to overcome performance and reusability issues. In Section III, the software INSTINCT is introduced. It is explained how it implements the FBP paradigm and what features it has. The results and the performance from a multi-sensor data fusion algorithm are discussed in Section IV. Finally, a short conclusion and outlook are given in Section V.

II. FLOW-BASED PROGRAMMING

Flow-based programming (FBP), which was first discussed by Morrison (1971), was developed with the purpose to overcome limitations in "von Neumann" hardware (Silc et al., 1998) and with the goal of exploiting massive parallelism (Johnston et al., 2004). The main idea by Morrison was that data-processing tasks are separated into modules, which can communicate among each other with the help of data elements that are passed through queues. In the background, a scheduler is used to trigger the calculations in each module depending on the availability of data elements, service requests by modules and external events (Morrison, 1971). The flow of the data can be displayed as a directed graph and therefore is eponymous for the name of the approach "dataflow programming".

The smaller the workload that each module has to handle, the better FPB benefits from parallel computing capabilities of the hardware. Originally, as described by Silc et al. (1998), it was intended to have specialized dataflow hardware architecture. Consequently, dataflow programming languages were invented (Ackerman, 1982; Whiting and Pascoe, 1994). However, with the progress of multiprocessing and multithreading capabilities in the last decades, FBP can also be applied to "von Neumann" architecture-based computer systems. Besides performance benefits from parallelism, FBP also has other advantages. Because each module acts as a black box, which resembles the way object-oriented programming abstracts its functionality, one does not have to know every detail of its implementation, but only needs to understand its input and output data elements. This leads to higher and low-latent productivity of new programmers (Morrison, 1971). Furthermore, Morrison (1994) presented empirical evidence that FBP reduces development times and therefore reduces programming costs. These results are further confirmed by Lobunets and Krylovskiy (2014) who also claim that FBP increases code reuse and maintainability.

Conventional software usually follows a hard-coded processing chain and has little adaptability when sensors change, algorithms are exchanged, or the software is needed to work in both real-time and post-processing environments. FBP, however, allows more flexibility because modules with the same inputs and outputs, like a sensor or a data file reader, can be exchanged with each other. This provides the possibility to develop the program logic and algorithms in post-processing and then exchange only the input modules in order to deploy the program onto a test device with sensors.

More recent work from Paleyes et al. (2021) states that FBP can also exceed service-oriented architectures (SOA) (The Open Group, 2009) in terms of code complexity and code size. The initial effort to write an application, which can define and manipulate the dataflow graph, is bigger, but the data discovery and processing tasks become simpler. Moreover, there are frameworks available that might serve this purpose, such as Google Dataflow (Krishnan and Gonzalez, 2015), Kubeflow (Bisong, 2019), Apache NiFi (Apache, 2014), flowpipe (Schweizer, 2017) or Simulink (MathWorks, 1984). However, these existing frameworks introduce problems with licensing or are not suitable to be executed on low-cost hardware such as single-board computers, which are used on mobile PNT platforms.

III. INSTINCT

1. General Description, Goal and Purpose

Based on the idea of FBP and triggered by the demand to have access to in-house software for different projects, the Institute of Navigation (INS) at the University of Stuttgart, Germany has started to develop its own software suite named INSTINCT (INS Toolkit for Integrated Navigation Concepts and Training) in 2021. This software was developed to meet the rising need for flexible program structures and performance in research and development (R&D) environments. Its main purpose is to provide a solid base for position, navigation and timing (PNT) algorithms and to be used in the field of sensor fusion where data needs to be processed with high rates and performance is critical. INSTINCT is written in modern C++ and provides a GUI which can be used to configure and run the algorithms, but can also be disabled when the software should be executed on a headless computer. Other than existing GNSS analysis packages, like RTKLib (Takasu, 2009), MuSNAT (Pany et al., 2019) or MATLAB's Navigation (MathWorks, 2019) or Sensor Fusion and Tracking (MathWorks, 2018) Toolboxes, INSTINCT allows to create tailored software solutions without the need to write code, while maintaining full control over all details of every algorithm that is used. It can also run on numerous of architectures and operating systems, including Linux, Windows, macOS and Raspberry Pi. A detailed description on how to use the GUI and also a description of all implemented algorithms is available as a HTML documentation which will be released in fall 2022 when the software will be made available as an open source project on GitHub.

The GUI with a simple dataflow can be seen in Figure 1. In the example there are three modules, which are called "nodes" in INSTINCT. A trajectory and IMU measurements are simulated by the *ImuSimulator* node and passed to the *ImuIntegrator* node which numerically integrates the observations to obtain the inertial navigation solution. All data then is linked to a *Plot* node that plots the data inside the GUI. The dataflow highlights the convenience the software provides, in that everything from data generation, over processing with an algorithm, till evaluation of the results is done within the software. This eliminates the need to use external tools which speeds up the development process dramatically.

Each node has configuration options, which can be accessed in separate windows in the GUI. The configuration windows for the *ImuSimulator* node from Figure 1 can be seen in Figure 2 and for the *Plot* node in Figure 3. A more detailed explanation of the *ImuSimulator* node is given in Section III.3c. Settings can be changed and the algorithm can be run again without the need to restart the program. Afterwards, the settings can be saved to JSON (Ecma International, 2017) files and be reloaded at a later point. This facilitates sharing of dataflows with

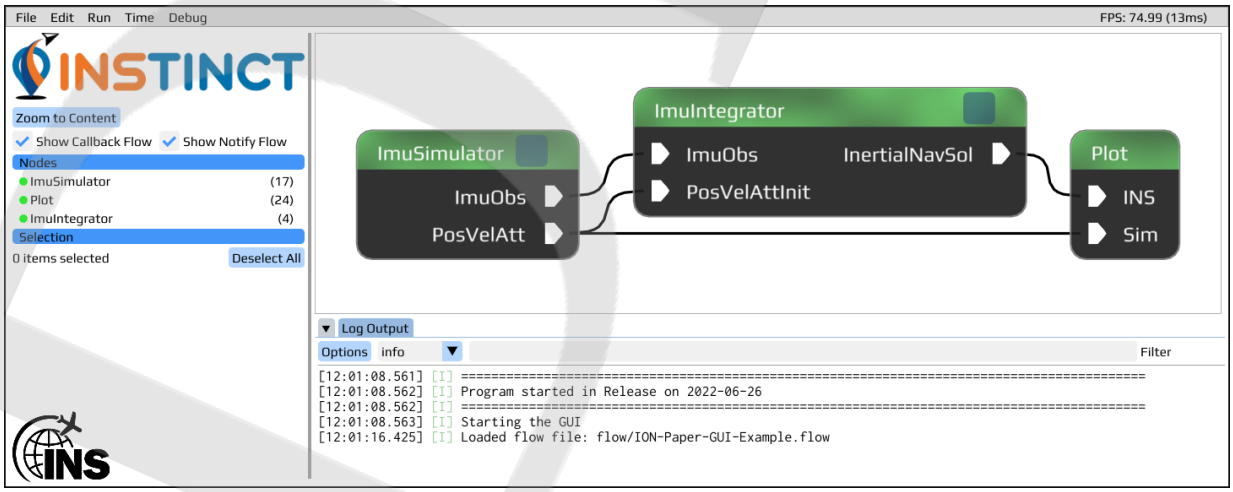


Figure 1: The GUI of INSTINCT with a simple flow. On the left, all nodes, which the user has assigned, independent if they are used or not, are displayed. The actual flow with the nodes, as well as the connections between them are shown in the large main window on the right. Once the flow is executed, status-, warning-, error- and debug-messages are displayed in the lower right window in real-time.

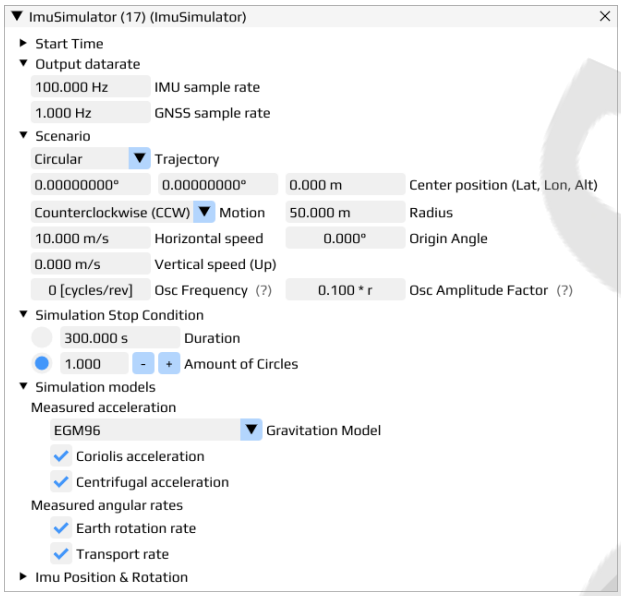


Figure 2: Configuration window of the *ImuSimulator* node. From the top to bottom, the "Start Time" can be chosen or set to computer time. The "Output datarate" specifies the time interval between simulated measurements. The "Scenario" specifies parameters to generate the trajectory. "Osc Frequency" can be used to modulate a harmonic oscillation onto the circular trajectory. Further the "Simulation Stop Condition" and the "Simulation models" can be changed or turned off. Last the "Imu Position & Rotation" can be used to specify the sensor mounting position and rotation in the simulated vehicle.

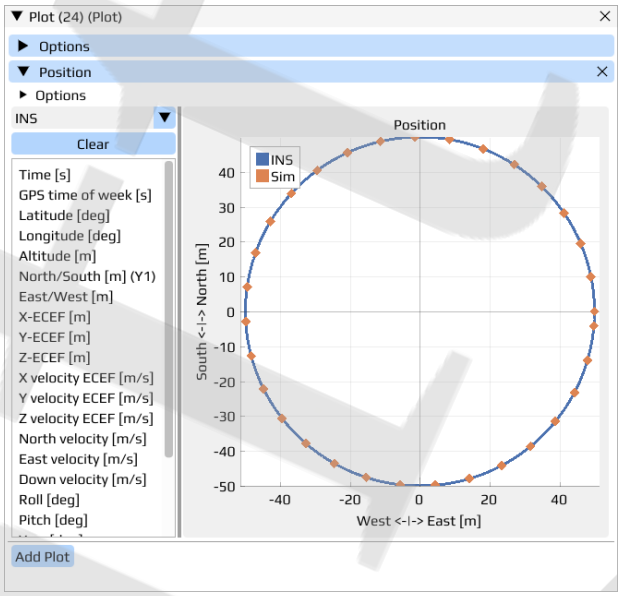


Figure 3: Configuration window of the *Plot* node. The list on the left shows all elements within the *InertialNavSol* data object. The drop-down above can be used to select the input pin of the node and this changes the content of the list. Items from the list can be dragged into the plot window on the right to show them. The plotting style of the data, as well as the legend text and axis settings are fully adaptable by right-click menus or in the option tabs above.

colleagues or project partners. A human readable format was chosen to enable script editing of the files to test different configurations programmatically.

Next to research, INSTINCT is used for teaching at the university in lectures and also for student theses. Students are provided with a reliable framework with an intuitive GUI where they can implement their research topics without having to deal with project management. All their code can be written inside a single node. This also makes it possible for less proficient programmers to work with the software and ensures that proven and tested code of other modules is neither touched nor modified.

2. Implementation of FBP

INSTINCT follows the flow-based programming paradigm but uses a different notation. Modules are called "nodes" but serve the same purpose on encapsuling functionality. A difference however is, that larger chunks of functionality are bundled in each node. For example, the single point positioning (SPP) algorithm that calculates positions and velocities from GNSS observations is represented by a single node despite consisting of multiple data processing steps. This obviously limits the possibility of parallelizing calculations but is a needed trade-off between simplicity of the dataflow and performance. Furthermore, on hardware architectures which are not designed for parallelism it does not make sense to split an application into thousands of threads as the overhead from passing the data and thread management becomes significant.

Like the modules in FBP, nodes can pass data elements to each other via so-called input and output "pins". However, in INSTINCT, nodes can have multiple in- and output pins, whereas in the original concept by Morrison (1971) only one of each was forseen. Data objects, which are supposed to flow between nodes, have to inherit from a common class that enables the required functionality. This makes it easy to define new custom node data classes tailored to the data that should be passed. Using specialized classes as node data does not create unnecessary overhead as it is done in other software implementing the FBP paradigm, e.g. Node-RED (OpenJS Foundation & Contributors, 2013), where JSON objects are used. It also enables the program to check if connections between nodes, so-called "links", are valid or if the data types differ and the connection should be refused. Therefore, even users who do not know the software cannot configure invalid dataflows. The base node data class also provides a way to set a timestamp for each object that enables the scheduler to trigger nodes in correct order. To avoid unnecessary copies while passing data, the data is passed as a reference to the data object, or more specifically as a shared C++ pointer (ISO/IEC, 2020). Shared pointers have the advantage, that they automatically keep track of how often they are referenced. When no reference exists anymore, they release their reserved memory preventing any possible memory leaks that could occur with normal pointers. This makes it possible for nodes to keep the data as long as it is needed without the risk that it is set invalid. Moreover, references are passed

Table 1: Supported file formats in INSTINCT. The column "Read" indicates if there is a node for reading the format from a file and the column "Write" if there is a node for logging data that is received by a sensor.

Format	Read	Write	Status
RINEX (IGS, 2018)	+	-	Version 3.xx observation and navigation file reading.
Positioning Solution File (Takasu, 2013)	+	-	For comparison to own algorithms.
ULog (PX4, 2021)	+	-	IMU data from Pixhawk logs.
UBX (ublox, 2021)	+	-	Decoding of raw GNSS data, logs whole data stream.
CSV	+	+	Used for most data loggers of sensors and calculation results.
Self defined binary formats	+	+	Drop-in replacement for CSV to log more efficiently.

as non-mutable variables because the same data can flow into multiple nodes. This could lead to data races and undefined behavior if nodes modify the objects.

Beside data flowing between nodes, the FBP paradigm was extended in INSTINCT through object pins that provide access to C++ objects living inside the nodes. This can be used as an interface to provide access to data that either does not change or changes very slowly with time and whose ownership should stay within the source node. An example for such data would be navigation GNSS data which is read from a file at the start of the execution or is slowly collected inside a node from a GNSS receiver. This can be seen in Figure 4. The object pins are highlighted in blue and orange depending on the type, while normal flow pins are white. Further explanation of the figure can be found in Section IV.1.

For some algorithms, like a closed-loop loosely-coupled error-state Kalman filter (described in Section III.3c), it is necessary to feed back results to nodes that are further up in the directed graph. This potentially causes data races as the nodes before could already receive new data while the later algorithms take more time to calculate. This problem, however, can be easily solved by defining trigger conditions for each pin of a node. In Figure 4, a dataflow with such a circular link is displayed. The *ImuIntegrator* node would have to wait until the *LooselyCoupledKF* node finishes its calculation, because the IMU data integration relies on the error state from the Kalman filter. Another problem still remains when the Kalman Filter takes more time than the interval between IMU measurements, which would lead the data queue of the *ImuIntegrator* node to fill up. This cannot be solved with FPB, as the *LooselyCoupledKF* node only runs after the *ImuIntegrator* node and therefore it is not possible to parallelize the calculation. However, as all other nodes are parallelized, the bottleneck is not further increased by them and is only restricted to these two nodes, which would also be the case if the algorithm is running in any other program.

3. Features

a) File Formats

As INSTINCT will be used in the field of PNT, a lot of standard file formats are implemented. A list of supported formats can be seen in Table 1. The table gives an overview of what formats exist and if there is a dedicated node available. In general file reader nodes read one message at a time and tell the scheduler the absolute time of the message. The scheduler can then sort all data messages and invoke the calculations in correct temporal order.

b) Sensors & Simulation Data Sources

In addition to reading files for post-processing, the software supports a variety of sensor protocols. Therefore, sensors can be connected to a computer in order to receive and process their data in real-time. Table 2 gives an overview of the implemented sensor types and their respective implementation status. The implementation is however limited to logging data for some sensors, as they were only used on some tests, but not in active development. When supporting sensors from different manufacturers, the problem occurs that each sensor has a different data format and also different information that is provided. To avoid writing algorithms that have to handle all different sensors, the sensor specific data is converted to a common data format. The nodes then provide output pins with the same data

Table 2: Supported sensors and data sources in INSTINCT. The column "Config" indicates if a sensor can be configured in the software or if an external tool is needed to configure the sensor in advance and only data logging is possible.

Type	Read	Config	Status
VectorNav (2022)	+	+	Complete configuration possible. Needs vnproglib of manufacturer.
Navio2 (Emlid, 2022)	+	+	Reading and data rate configuration of both onboard IMUs.
μblox (ublox, 2022)	+	-	Up to M9 series. Needs to be pre-configured in u-center.
KVH FOGs (KVH, 2022)	+	-	Tested with rates up to 400 Hz.
Skydel (Orolia, 2022)	+	-	Process simulated IMU data over an UDP network stream.
ImuSimulator	+	+	Simulation of trajectories and generation of IMU measurements. The GUI configuration can be seen in Figure 2. More details below.

types and, for example, an IMU sensor of one manufacturer can be exchanged for another one by simply rerouting the dataflow.

c) Algorithms

Within INSTINCT, a lot of algorithms concerning multi-sensor fusion already exist and have been tested and validated. Table 3 lists all data processing nodes currently implemented alongside their functionality. An example for the combination of several processing nodes can be seen in Figure 4. As mentioned earlier, a detailed description of all algorithms can be found in the HTML documentation. The implementation of tightly-coupled filter, real-time kinematic (RTK) and precise point positioning (PPP) modules is currently ongoing.

INS/GNSS Loosely-Coupled Kalman Filter. The loosely-coupled Kalman filter is implemented in both, the Earth centered Earth-fixed frame (ECEF/e-frame) and the local navigation frame (NED/n-frame). In this paper, only the local navigation frame equations will be discussed. The filter is based on an error-state implementation with 15 states, where $\boldsymbol{\delta x}$ includes the attitude error $\boldsymbol{\delta\psi}$, the velocity error $\boldsymbol{\delta v}^n$, the position error (latitude, longitude, altitude) $\boldsymbol{\delta p}_b$, accelerometer biases $\boldsymbol{\delta f}^b$ and gyroscope biases $\boldsymbol{\delta\omega}_{ib}^b$. The equations in the filter are mainly taken from Groves (2013) but were also compared to Titterton and Weston (2004), Noureldin et al. (2013) or Gleason and Gebre-Egziabher (2009), who define the same relation but with slightly different frames or sign conventions. The exact definition can be found in the HTML documentation and is beyond the scope of this paper.

In the configuration window of the *LooselyCoupledKF* node the user can choose if the random process should be modelled as a random walk or a first order Gauss-Markov process. All necessary parameters, like initial error covariance matrix, measurement noise covariance matrix, standard deviation of the noise and the dynamic biases, as well as the correlation length in case of the Gaus-Markov process can be set in the GUI. To calculate the process noise covariance matrix $\mathbf{Q}$, one can choose between a Taylor approximation (Groves, 2013) and the Van-Loan algorithm (Van Loan, 1978). The possibility to select different models and values is not only useful when tuning the filter, but also helps to depict the functionality in lectures and other teaching and training activities.

IMU Simulator. The *ImuSimulator* node is able to simulate different trajectories and calculate accelerometer and gyroscope measurements that a sensor would measure along the trajectory. Currently fixed position, linear, and circular trajectory generation is implemented. Furthermore, reading a trajectory from a CSV file is possible. All trajectories are internally represented mathematically as cubic B-spline functions, ensuring that velocity $\dot{\boldsymbol{x}}^e$ and acceleration $\ddot{\boldsymbol{x}}^e$ can be derived consistently at any given point in time (Baklanov, 2020). Thus, the simulated total acceleration, which an IMU would sense, expressed in navigation frame coordinates, is defined as

$$\overbrace{\boldsymbol{f}^n}^{\text{measured}} = \underbrace{\dot{\boldsymbol{v}}^n}_{\text{trajectory}} + \overbrace{(2\boldsymbol{\omega}_{ie}^n + \boldsymbol{\omega}_{en}^n) \times \boldsymbol{v}^n}^{\text{coriolis acceleration}} - \underbrace{\mathbf{g}^n}_{\text{gravitation}} + \mathbf{C}_e^n \cdot \underbrace{(\boldsymbol{\omega}_{ie}^e \times [\boldsymbol{\omega}_{ie}^e \times \boldsymbol{x}^e])}_{\text{centrifugal acceleration}}, \tag{1}$$

where $\boldsymbol{\omega}_{ie}$ is the Earth rotation rate given in n- or e-frame, $\boldsymbol{\omega}_{en}^n$ the transport rate given in the n-frame and $\mathbf{C}_e^n$ being the rotation matrix from the e- to n-frame. The velocity and kinematic acceleration are derived from

$$\dot{\boldsymbol{v}}^n = \frac{\partial}{\partial t}(\dot{\boldsymbol{x}}^n) = \frac{\partial}{\partial t}(\boldsymbol{C}_e^n \cdot \dot{\boldsymbol{x}}^e) = \dot{\boldsymbol{C}}_e^n \cdot \dot{\boldsymbol{x}}^e + \boldsymbol{C}_e^n \cdot \ddot{\boldsymbol{x}}^e \tag{2}$$

with

$$\dot{\boldsymbol{C}}_e^n = (\dot{\boldsymbol{C}}_n^e)^T = (\boldsymbol{C}_n^e \cdot \boldsymbol{\Omega}_{en}^n)^T, \tag{3}$$

where $\boldsymbol{\Omega}_{en}^n$ is the skew-symmetric matrix of the angular rate vector $\boldsymbol{\omega}_{en}^n$.

The angular rates measured by a gyroscope are the turn rate of the platform frame with respect to an inertial frame expressed in platform coordinates are

$$\begin{aligned}
\boldsymbol{\omega}_{ip}^p &= \boldsymbol{\omega}_{in}^p + \boldsymbol{\omega}_{np}^p \\
&= \mathbf{C}_n^p \cdot [\boldsymbol{\omega}_{in}^n + \boldsymbol{\omega}_{np}^n] \\
&= \mathbf{C}_n^p \cdot [(\boldsymbol{\omega}_{ie}^n + \boldsymbol{\omega}_{en}^n) + (\boldsymbol{\omega}_{nb}^n + \overbrace{\boldsymbol{\omega}_{bp}^n}^{=0})] \\
&= \mathbf{C}_n^p \cdot [\boldsymbol{\omega}_{ie}^n + \boldsymbol{\omega}_{en}^n + \boldsymbol{\omega}_{nb}^n] \\
&= \mathbf{C}_n^p \cdot [\boldsymbol{\omega}_{ie}^n + \boldsymbol{\omega}_{en}^n + \mathbf{C}_b^n \boldsymbol{\omega}_{nb}^b].
\end{aligned} \tag{4}$$

Table 3: Data processing nodes in INSTINCT.

Node	Description
ErrorModel	Adds a bias and/or Gaussian white noise to IMU observations and/or GNSS position and velocity measurements.
SinglePointPositioning	Calculates position and velocity from GNSS observations and navigation data with the least squares algorithm. So far GPS and GALILEO are implemented.
ImuIntegrator	Integrates IMU measurements in local navigation (NED) or Earth centered Earth-fixed (ECEF) frame coordinates with selectable numeric integration algorithms.
LooselyCoupledKF	Error state GNSS/INS loosely-coupled Kalman filter in NED and ECEF frame. More details below.

The turn rate of the body frame with respect to the local-navigation frame expressed in body coordinates $\boldsymbol{\omega}_{nb}^{b}$, given in equation 5, depends on the time derivative of the gimbal angles ϕ (roll), θ (pitch), ψ (yaw). Those values are also provided in cubic B-spline representation so that the time derivative of the Euler angles can be calculated at any epoch.

$$\boldsymbol{\omega}_{nb}^{b} = \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + \mathbf{C_3} \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + \mathbf{C_3 C_2} \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} \tag{5}$$

The matrix $\mathbf{C_3}$ is the rotation ϕ about the x-axis and $\mathbf{C_2}$ is the rotation θ about the y-axis (Titterton and Weston, 2004).

$$\mathbf{C_3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix}, \mathbf{C_2} = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \tag{6}$$

IV. RESULTS

1. Test Case

To evaluate INSTINCT with respect to correctness and performance, the flow displayed in Figure 4 has been used. For this test, a trajectory that was previously recorded on a flight with the Icaré II (IFB, 1996), a solar powered motor glider with 25 m wingspan, is used. The trajectory, which can be seen in Figure 5a, is fed into an Orolia Skydel GNSS simulator and subsequently recorded by a Septentrio PolaRx5TR with a frequency of 1 Hz. The atmosphere is simulated with the Saastamoinen (1973) troposphere and Klobuchar (1987) ionosphere model. By using GNSS observations from a real receiver that is connected to a GNSS simulator, it is possible to access the accuracy of the estimated trajectory and not only judge the quality of the results by their formal errors, i.e. the precision deduced from the Kalman filter covariance matrices.

In the flow on the left there are two boxes, *GNSS* and *IMU*. In the *GNSS* box, a single point positioning algorithm is executed based on RINEX observations and navigation files recorded by the Septentrio receiver. In the *IMU* box, the same trajectory, which has been used for the Orolia GNSS simulator, is read in from a CSV file where time, position and flight angles are defined. With this data the *ImuSimulator* node calculates its spline representation as discussed in Section III.3c. As the trajectory is provided before the simulation is launched, the calculation of the spline coefficients can already take place during the initialization of the software. When the flow is run, only the required derivatives have to be calculated. IMU measurements are simulated with a frequency of 100 Hz and fed to the *IMU ErrorModel* node where accelerometer and gyroscope biases and white noise are added. The values of the biases and noise can be taken from Table 4.

Table 4: IMU biases and standard deviation σ of the white noise added in the *IMU ErrorModel* node of Figure 4

Axis	Accelerometer		Gyroscope	
	bias	noise	bias	noise
	$[\mathrm{m\,s^{-2}}]$	$[\mathrm{m\,s^{-2}}]$	$[\mathrm{rad\,s^{-1}}]$	$[\mathrm{rad\,s^{-1}}]$
x	1×10^{-3}	3×10^{-4}	-1×10^{-3}	3×10^{-5}
y	2×10^{-3}	3×10^{-4}	-2×10^{-3}	3×10^{-5}
z	3×10^{-3}	3×10^{-4}	-3×10^{-3}	3×10^{-5}

In the *INS/GNSS Sensor Fusion* box of Figure 4, the *ImuIntegrator* node numerically integrates the IMU measurements and passes its inertial navigation solution to the *LooselyCoupledKF* node, where the error state from Section III.3c is calculated. The errors then flow back into the *ImuSimulator* node where the state and the IMU measurements are corrected.

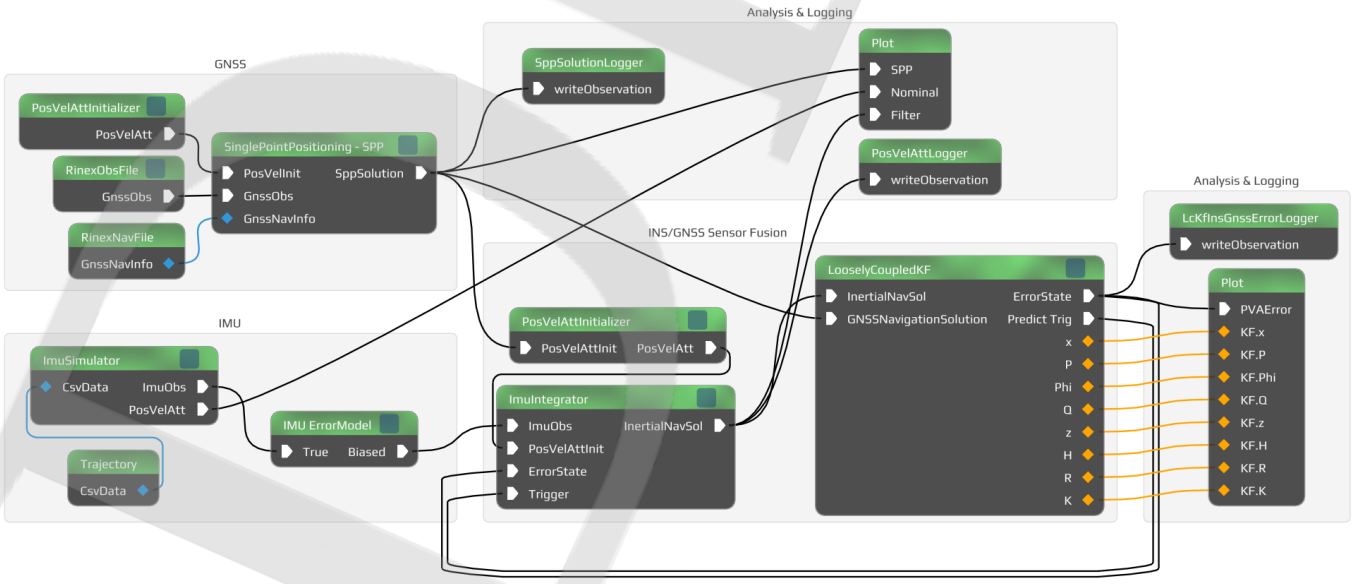


Figure 4: Loosely-coupled Kalman filter flow inside INSTINCT. A detailed description can be found in Section IV.1.

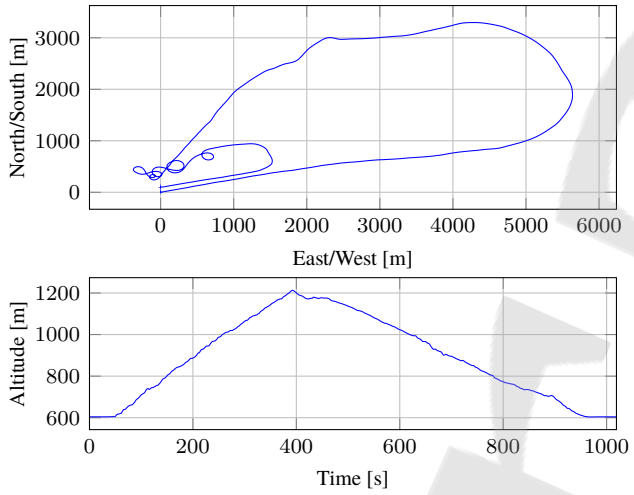
The last two boxes, named *Analysis & Logging*, contain data logging nodes to save the results into CSV files, as well as *Plot* nodes to directly display the results within INSTINCT as shown earlier in Figure 3. Something unusual can be observed when looking at the link between the *ImuIntegrator* and the *PosVelAttLogger* nodes. As the name suggests, the *PosVelAttLogger* node logs *PosVelAtt* data types and not *InertialNavSol* types. However, connecting them is still possible because both the GUI and passed data make use of C++ inheritance and *InertialNavSol* extends the *PosVelAtt* type. Therefore, it can be connected to all pins which are of parent types.

2. Single Point Positioning (SPP)

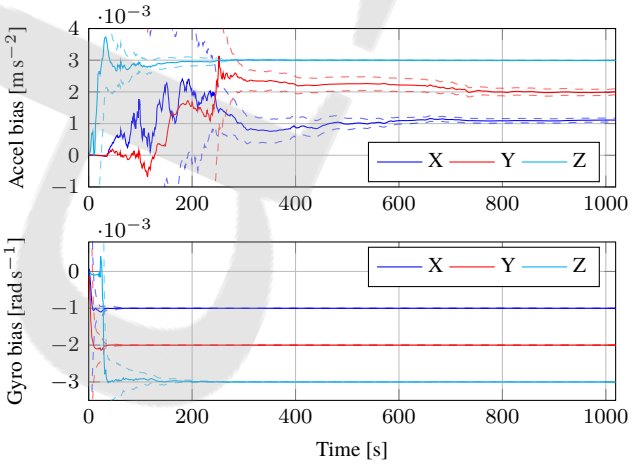
In order to cross-validate the SPP-only solution, two options exist. First, the estimated trajectory can be compared against the one that has been used to simulate the GNSS observations. This evaluation is depicted in Figure 5c and discussed later. In addition, an independent SPP solution was calculated with the software RTKLib (Takasu, 2009) and differences between the position solutions in north, east and up direction were computed. Results from this comparison are displayed in Figure 5d. Both algorithms use only GPS L1 and atmospheric effects are corrected with the same models as the ones that have been used by the GNSS simulator. As the difference is in the sub-millimeter range, which is far below the accuracy of the SPP algorithm, it is concluded that the results are consistent with ones provided by state-of-the art GNSS analysis software.

3. INS/GNSS Fusion (Loosely-Coupled Kalman Filter)

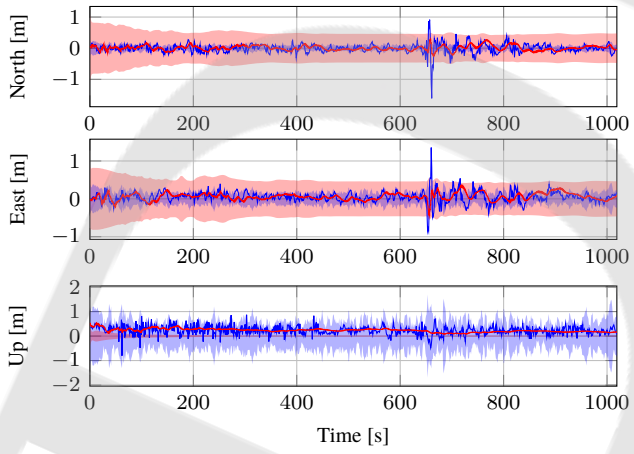
In this section, the solution from the Kalman fusion-filter is discussed in greater detail. Figure 5b depicts the estimated biases for the accelerometer and gyroscope. One can notice, that after



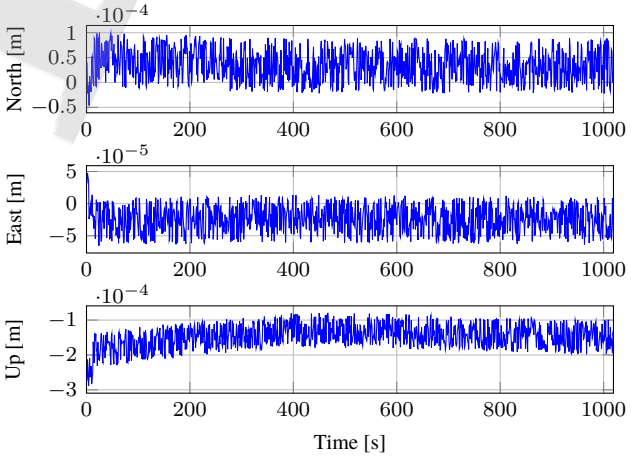
(a) Simulated trajectory as derived from a real flight.



(b) Accelerometer and gyroscope biases which have been estimated by the LCKF. Standard deviations σ are shown as dashed lines.



(c) Position errors with respect to the simulated trajectory. The SPP-only solution is shown in blue color and the IMU/GNSS solution is plotted in red. The shaded area represents the 3σ uncertainty bounds.



(d) Differences between the INSTINCT SPP position solution and a SPP solution calculated with RTKLib with the same settings.

Figure 5: Results from the loosely-coupled Kalman filter flow of Figure 4.

a short initialization time, the biases converge to the simulated values as listed in Table 4. The gyroscope x and y biases are the ones which are being estimated fastest, since they can be estimated even when the platform is not moving, i.e. in the beginning of the simulation, when the aircraft is still parked. The z axis gyro bias and the z axis accelerometer bias become estimable after the vehicle starts moving, which happens after approximately 30 seconds. However, as the vehicle is moving in a straight line upwards during take-off, x and y axis accelerometer biases still do not converge. This is due to the fact that these biases need a kinematic maneuver, such as a curve, to be correctly estimated. Such maneuver only happens 200 seconds after the start of the simulated flight where the estimation of the biases improves. As the curve is very wide and the rotational movement is small, the biases do not reach their steady-state values yet. It takes until around 650 seconds into the run for the biases to reach the simulated values. At this time, the aircraft performs some intense movements, which corresponds to the loops at the end of the trajectory.

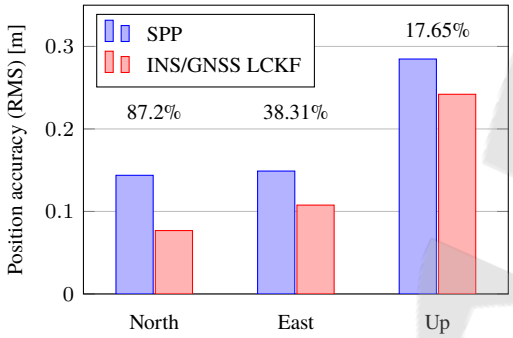


Figure 6: Accuracy of the SPP position solution compared to the INS/GNSS fused solution. Percentages are the improvement in accuracy of the LCKF solution to the SPP-only solution.

In Figure 5c, the differences between the solution from the Kalman filter and the reference trajectory are shown in red, the differences between the SPP solution and the reference in blue. The shaded colors are the 3σ bounds of the individual solutions. The SPP uncertainties are calculated with the reduced Chi-squared statistic of the least square estimation and the Kalman filter values are taken from the diagonal elements of the error covariance matrix. As for the SPP solution, differences between the estimates and the reference exceed the 3σ bounds quite often in the horizontal components. Only in the vertical component the error bounds are estimated more realistically. However, the SPP-only solution in this direction is also the least accurate, which is a well-known phenomenon for GNSS-based positioning solutions. In general, the SPP solution is more accurate than what is expected from this algorithm type. The reason for this is that the simulator uses the same atmospheric models as the SPP algorithm is compensating with. Therefore the simulation can be seen as error-free.

Figure 6 depicts the root mean squared (RMS) position accuracy of the SPP solution compared to the INS/GNSS fused solution. The Kalman filter improves the horizontal accuracy quite well, the vertical accuracy, however, is not improved significantly due to the small offset of the SPP solution that cannot be reduced much by the fusion filter.

4. Processing Performance

To demonstrate the real-time capabilities of INSTINCT the execution time of the dataflow from Figure 4 was measured. The results from this performance study are listed in Table 5. All tests were done with an Intel Core i7-4770K CPU @ 3.50GHz quad-core processor running Linux. There are 16 GB of 1600 MHz DDR3 RAM and a SDD with 540 MB/s sequential reading and up to 520 MB/s sequential writing available. The system is nine years old, thus outdated. Nevertheless, while processing 100 s of data for a typical setup with an IMU providing 200 Hz and a GNSS receiver providing 10 Hz data rates, the execution time was still below 1.4 s. Therefore it is unlikely to encounter real-time performance issues with the current algorithms on any modern platform. Nevertheless, utilizing multiple GNSS systems and frequencies can drastically deteriorate the performance. The deterioration is however locally contained to the SPP calculation node due to the parallelized execution of each node. This gives an advantage over traditional procedural software but does not remove the need for an efficient implementation of algorithms to guarantee real-time capabilities. The percentage values in the table display the improvement of multi-threaded execution of the dataflow

Table 5: Execution time in seconds of the dataflow from Figure 4 with different GNSS and IMU data rates. 100 seconds data processed, GNSS observations (GPS L1 only) with 10 satellites on average. Percentage is improvement compared to single-threaded execution of the dataflow.

	GNSS					
IMU	1 Hz	5 Hz	10 Hz	20 Hz	50 Hz	100 Hz
10 Hz	0.04 (120%)	0.06 (141%)	0.10 (122%)	0.18 (99%)	0.44 (90%)	0.87 (95%)
20 Hz	0.07 (135%)	0.09 (100%)	0.12 (117%)	0.18 (133%)	0.47 (93%)	0.95 (76%)
50 Hz	0.16 (100%)	0.18 (119%)	0.20 (113%)	0.29 (101%)	0.45 (96%)	0.87 (90%)
100 Hz	0.30 (111%)	0.32 (101%)	0.35 (106%)	0.41 (102%)	0.61 (96%)	0.90 (90%)
200 Hz	0.56 (111%)	0.58 (114%)	0.60 (116%)	0.72 (101%)	0.90 (97%)	1.22 (94%)
400 Hz	1.09 (114%)	1.10 (120%)	1.17 (107%)	1.29 (96%)	1.46 (96%)	1.82 (85%)
1000 Hz	2.58 (117%)	2.68 (115%)	2.80 (113%)	2.81 (111%)	3.16 (101%)	3.41 (104%)

compared to a single-threaded execution. This shows the potential increase in performance even though in the dataflow the *ImuIntegrator* and *LooselyCoupledKF* nodes are doing most of the work and so only 2 CPU cores are fully utilized.

V. CONCLUSIONS AND FUTURE WORK

FPB is a powerful tool to reduce execution time bottlenecks to the essential algorithms, as it removes any overhead from data preparation or logging. This can speed up software on multiprocessor systems tremendously. Nevertheless, it is not an almighty tool that speeds up the algorithms themselves. Optimization remains the responsibility of the developer.

In general, it can be stated that INSTINCT provides a robust foundation for PNT algorithms. Current limitations of the software are mostly on the side of the GNSS processing algorithms and the support of different signals and constellations. Additionally, the implementation of more algorithms, such as a tightly-coupled INS/GNSS sensor fusion, real-time kinematic (RTK), and precise point positioning (PPP) are ongoing.

The software will be publicly accessible in fall 2022. More details on INSTINCT and the roadmap concerning the release of the software under an open-source license can be found in Topp et al. (2023).

ACKNOWLEDGEMENTS

The authors wish to thank V. Trukhan, who gave major inspiration for the flow-based programming approach. They also wish to thank D. Becker and M. Maier, from the Institute of Navigation (INS) at the University of Stuttgart, Germany, who helped implementing some algorithms. Further acknowledgment goes to Orolia for letting the INS join the Orolia Academic Partnership Program (OAPP) and donating a free license to their GNSS simulation software Skydel, without which the results in this paper could not have been created. In addition, parts of this research were funded with the BMWK/DLR LuFo VI project CNS-Alpha (funding nr. 20V1904B).

REFERENCES

Ackerman, W. B. (1982). Data flow languages. *IEEE Computer*, 15(2):15–25.

Apache (2014). NiFi. `https://nifi.apache.org/`. Accessed: 2022-08-27.

Baklanov, F. (2020). *Simulation of ideal inertial sensor measurements*. Open Aided Navigation Project, 0.1 edition.

Bisong, E. (2019). Kubeflow and kubeflow pipelines. *In Building Machine Learning and Deep Learning Models on Google Cloud Platform. Springer*, pages 671–685.

Ecma International (2017). *Standard ECMA-404 - The JSON Data Interchange Syntax*. Ecma International, 2 edition.

Emlid (2022). Navio2 - Autopilot HAT for Raspberry Pi. `https://navio2.emlid.com`. Accessed: 2022-06-29.

Gleason, S. and Gebre-Egziabher, D. (2009). *GNSS Applications and Methods*. Artech House GNSS Library.

Groves, P. D. (2013). *Principles of GNSS, Inertial, and Multi-Sensor Integrated Navigation Systems*. Artech House GNSS Library, 2 edition.

IFB (1996). icaré. , Institute of Aircraft Design, University Stuttgart. `https://www.icare-solar.de/`. Accessed: 2022-06-29.

IGS (2018). The receiver independent exchange format - version 3.04. , International GNSS Service (IGS), RINEX Working Group and Radio Technical Commission for Maritime Services Special Committee 104 (RTCM-SC104).

ISO/IEC (2020). *ISO International Standard ISO/IEC 14882:2020(E) - Programming Language C++*. International Organization for Standardization (ISO), 6 edition.

Johnston, W. M., Hanna, J. R. P., and Millar, R. J. (2004). Advances in dataflow programming languages. *ACM Computing Surveys*, 36(1):1–34.

Klobuchar, J. A. (1987). Ionospheric time-delay algorithm for single-frequency gps users. *IEEE Transactions on Aerospace and Electronic Systems*, pages 325–331.

Krishnan, S. P. T. and Gonzalez, J. (2015). Google cloud dataflow. *In Building Your Next Big Thing with Google Cloud Platform. Springer*, pages 255–275.

KVH (2022). Fiber Optic Gyros (FOGs). `https://www.kvh.com/fog-and-inertial-systems/fiber-optic-gyros`. Accessed: 2022-06-29.

Lobunets, O. and Krylovskiy, A. (2014). Applying flow-based programming methodology to data-driven applications development for smart environments. *UBICOMM 2014 - 8th International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*, pages 216–220.

MathWorks (1984). Simulink. `https://mathworks.com/products/simulink.html`. Accessed: 2022-08-27.

MathWorks (2018). MATLAB Sensor Fusion and Tracking Toolbox. `https://www.mathworks.com/products/sensor-fusion-and-tracking.html`. Accessed: 2022-09-08.

MathWorks (2019). MATLAB Navigation Toolbox. `https://de.mathworks.com/products/navigation.html`. Accessed: 2022-09-08.

Morrison, J. P. (1971). Data responsive modular, interleaved task programming system. *IBM Technical Disclosure Bulletin*, 13(8):2425–2426.

Morrison, J. P. (1994). *Flow-based programming: a new approach to application development*. Van Nostrand Reinhold.

Noureldin, A., Karamat, T. B., and Georgy, J. (2013). *Fundamentals of Inertial Navigation, Satellite-based Positioning and their Integration*. Springer.

OpenJS Foundation & Contributors (2013). Node-RED. `https://nodered.org`. Accessed: 2022-06-19.

Orolia (2022). Skydel GNSS Simulation Software. `https://www.orolia.com/product/`

skydel-simulation-engine. Accessed: 2022-06-29.

Paleyes, A., Cabrera, C., and Lawrence, N. D. (2021). Exploring the potential of flow-based programming for machine learning deployment in comparison with service-oriented architectures. *CoRR*, arXiv:2108.04105.

Pany, T., Dötterböck, D., Gomez-Martinez, H., Hammed, M. S., Hörkner, F., Kraus, T., Maier, D., Sanchez-Morales, D., Schütz, A., Klima, P., and Ebert, D. (2019). The multisensor navigation analysis tool (musnat) - architecture, lidar, gpu/cpu gnss signal processing. *Proceedings of the 32nd International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2019)*.

PX4 (2021). PX4 User Guide. https://docs.px4.io/master/en/dev_log/ulog_file_format.html. Accessed: 2022-06-29.

Saastamoinen, J. (1973). Contributions to the theory of atmospheric refraction: Part ii. refraction corrections in satellite geodesy. *Bulletin géodésique*, 107:13–34.

Schweizer, P. (2017). flowpipe. https://github.com/PaulSchweizer/flowpipe. Accessed: 2022-08-27.

Silc, J., Robis, B., and Ungerer, T. (1998). Asynchrony in parallel computing: from dataflow to multithreading. *Scalable Computing: Practice and Experience*, page 34.

Takasu, T. (2009). Rtklib: Open source program package for rtk-gps. *FOSS4G 2009 Tokyo*.

Takasu, T. (2013). RTKLIB ver. 2.4.2 Manual. http://www.rtklib.com/prog/manual_2.4.2.pdf. Accessed: 2022-06-29.

The Open Group (2009). *SOA Source Book*. Van Haren Publishing.

Titterton, D. H. and Weston, J. L. (2004). *Strapdown Inertial Navigation Technology*. The Institution of Electrical Engineers, 2 edition.

Topp, T., Maier, M., Hobiger, T., and Becker, D. (2023). INSTINCT: INS Toolkit for Integrated Navigation Concepts and Training (in preparation). *GPS Solutions*.

ublox (2021). u-blox 8 / u-blox M8 Receiver description. https://content.u-blox.com/sites/default/files/products/documents/u-blox8-M8_ReceiverDescrProtSpec_UBX-13003221.pdf. Accessed: 2022-06-29.

ublox (2022). Positioning chips and modules. https://www.u-blox.com/en/positioning-chips-and-modules. Accessed: 2022-06-29.

Van Loan, C. (1978). Computing integrals involving the matrix exponential. *IEEE Transactions on Automatic Control*, 23(3):395–404.

VectorNav (2022). Product Overview. https://www.vectornav.com/products. Accessed: 2022-06-29.

Whiting, P. and Pascoe, R. (1994). A history of data-flow languages. *IEEE Annals of the History of Computing*, 16(4):38–59.