Implementation of SINDy for Real Time System Identification of UAVs

Stefan Bichlmaier
University of Victoria
Department of Electrical and Computer Engineering
Victoria BC, Canada

Using machine learning methods to understand and control dynamical systems is made possible by developments in data driven modelling and economic computational power. Applying these methods to real time systems allows them to be extremely robust, capable of adapting to changes in their operating environments. In this paper, the modelling method Sparse Identification of Non-linear Dynamics (SINDy) is implemented for use in unmanned aerial vehicles (UAVs) to aid in identifying their dynamics online. This lays the framework for further work in rapid airframe development and model identification adaptive controllers (MIAC).

I. Nomenclature

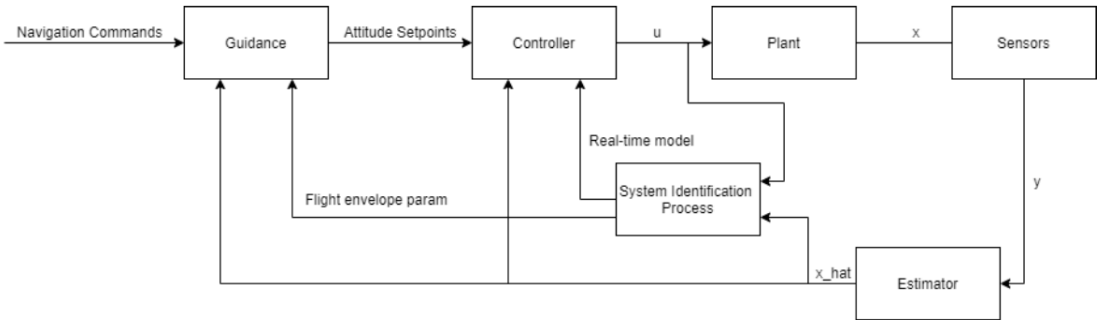
$CFAR$	=	UVic Center for Aerospace Research
$MIAC$	=	Model Identification Adaptive Control
MPC	=	Model Predictive Controller
SID	=	System Identification
$SINDy$	=	Sparse Identification of Nonlinear Dynamics

II. Introduction

Model Predictive Control (MPC) describes a set of control strategies wherein a controller maintains a dynamical model of the system it is controlling. It uses this model to predict the system's responses to inputs and may optimize them such that the error between the result and the set point is minimized [1]. There are numerous economic and technical challenges associated with this method. In order to create a model representing the system, ground-based analyses using computational fluid dynamics (CFD) simulations and wind tunnel testing are performed which are time consuming and iterative processes [2]. The model parameters obtained through this process are used to design the flight controller, which is further validated through simulations and flight testing. Inconsistencies are identified, and the process reiterates. In addition to development time, shortcomings of conventional MPCs are due to their static understanding of the airplane. If changes in the aerodynamics or flight controls occur without updating the model, the controller's predictions will not match closely to the actual outcomes.

An alternative which addresses these issues is MPC with online system identification, also termed Model Identification Adaptive Control (MIAC). In this scheme, the controller contains a system identification process which recomputes the system model. This way, much less preliminary work needs to be done to generate an accurate model. In practice, a model with less detail would be determined prior to flight and the online system identification could then refine it. This could decrease the time and resources required for developing new airframes since the controller could identify the system during flight. Another key benefit of the MIAC control scheme is its robustness to changes in the system. For example, damage to the airplane, ice build up, or control failure would cause an increased error between the airplane's actual and desired state. Online system identification could detect these changes and allow the controller to take them into account. Figure 1 shows the architecture of such a controller.

Fig. 1 MIAC Controller [2]



The system identification portion of this controller is a continuing area of research. Many groups have experimented with different linear and nonlinear methods with varying degrees of success [3–6]. This work aims to implement a system identification process based on a novel method called SINDy, with the goal of operating in real time.

A. SINDy

Sparse Identification of Nonlinear Dynamics, or SINDy is an innovative approach to system identification which implements identification of both linear and non-linear system dynamics [7]. This method is chosen because it produces tractable solutions to the nonlinear equations describing the aircraft's dynamics. It has promise of real time performance due to promoting parsimonious solutions. SINDy makes use of sparsity-promoting regression methods, a field which has seen recent development with the rise of machine learning and data science to generate parsimonious models [7]. In essence, it is based on the assumption of Occam's Razor, that the simplest solution (the one with fewest terms) is likely the correct one.

A dynamical system can be described as a system of differential equations, shown in equation 1, where x is a vector describing the states of the system through time.

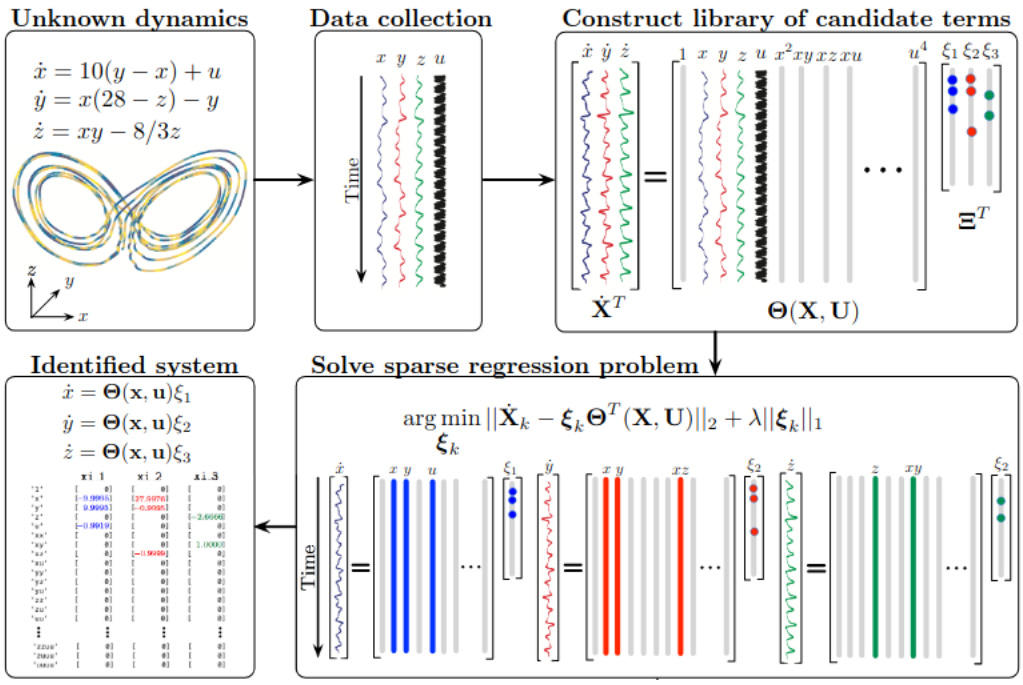
$$\dot{x} = f(x) \tag{1}$$

In matrix form, this relation can be written as shown in equation 2, where $\boldsymbol{X} = [\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_m}]$ represents the system states, $\boldsymbol{\Theta} = [\boldsymbol{1}, \boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_m}, \boldsymbol{x_1^2}, \boldsymbol{x_1 x_2}, \ldots, \boldsymbol{x_1 x_m}, \boldsymbol{x_2^2}, \boldsymbol{x_2 x_3}, \ldots, \boldsymbol{x_2 x_m}, \ldots, \boldsymbol{x_m^2}]$ represents unique combinations of said states called candidate functions, and $\boldsymbol{\Xi} = [\boldsymbol{\xi_1}, \boldsymbol{\xi_2}, \ldots, \boldsymbol{\xi_m}]$ are the coefficients weighting each candidate function.

$$\dot{\boldsymbol{X}} = \boldsymbol{\Theta}\boldsymbol{\Xi} \tag{2}$$

If the system states and their derivatives known, and the candidate functions are chosen correctly, a regression performed on the expression to solve for $\boldsymbol{\Xi}$ will describe the governing equations. An overview of this process is shown in figure 2.

Fig. 2 SINDy applied to a Lorenz Attractor [7]



To achieve parsimony, this process is repeated several times, where the solved coefficients below a threshold parameter are set to zero after each iteration. Thresholding the coefficients presents an important consideration when choosing the regression algorithm as it must be regularized in order to promote sparsity. For this reason, a conventional least squares approach is not appropriate since it produces solutions which are distributed among the solution vector, causing thresholding to discard useful weights. Instead, the Ridge Regression algorithm is used as it promotes sparsity [8].

Keywords	Inclusion Criteria	Exclusion Criteria
SINDy	Implementation-specific	Lack of implementation
Online System Identification	Online system identification	Offline system identification
SID	Aerospace/mechatronic systems	Lack of experimental results
MIAC	Embedded systems	
MPC		
Real-time		
UAV		
Adaptive Control		

Table 1 Literature review material collection

B. Literature Review

A literature review is performed to understand the current state of the art of online system identification in aircraft. The focus of the review is focused on implementations to help guide the application of SINDy to aircraft system identification. The review follows a methodology to collect resources in this area of interest. First, google scholar is used to obtain a wide breadth of candidates using a collection of keywords. This bulk group of papers is then refined using inclusion and exclusion criteria, identified by abstract skimming. The keywords and criteria are summarized in table 1. With this process, an initial collection of 52 published papers was collected, which was refined to 11 papers after abstract skimming. This was further reduced to 4 papers with more in-depth reading.

The system designed by the NASA L2F team [3] aims to identify nonlinear dynamics with little previous knowledge of the aircraft's dynamics. They use an architecture with RC and autopilot control isolated with an RC-Mux channel selector for manual control to be taken. Autopilot control and system identification are implemented on the same computational system, running on a Quad Core Intel Atom 1.9 GHz processor. The processor is on a COM Express type 6 board, configured with 8GB RAM and a 120GB SSD. The research flight control algorithm performed at a rate of 50 Hz and recorded over 1250 variables. These variables included general flight code parameters like flags, raw encoder and analog data, and a timestamp. The algorithms ran on the flight hardware with a processor overrun margin of roughly 20%.

A team at Georgia Tech used the Fourier Transform Regression method with a recursive butterworth filter for parameter identification. Results are satisfactory with parameter convergence in time and frequency domain. The system is based on an Altera Stratix FPGA with a NIOS soft core CPU used to implement the flight controller and system identification computations. Peripherals are added to provide sensor data collection, servo control, and telemetry. [4]

The Swiss Federal Institute of Technology presents a nonlinear model predictive controller using a pixhawk and companion computer [5]. A low level, standard cascaded PID system is executed on the pixhawk. It is well tuned first before performing any flight testing. The nonlinear model predictive controller computation occurs on the more

powerful companion computer, running on a 1.7 GHz Quad Core Cortex-A9 Processor and 2GB RAM. The companion computer has robot operating system (ROS) installed, making use of the wrappers provided by ROS for communication with the pixhawk over a serial connection. The system has a loop time with the statistics 9.96 ms mean, 0.250 ms standard deviation, and 10.5 ms max.

Texas A&M [6] implements a human in the loop model monitoring system to ensure aircraft safety. The operator analyzes the updated model and decides whether to accept it or not. Online system identification is performed in near real time locally on the aircraft using the Observer/Kalman linear system Identification Algorithm. System identification and excitation is computed onboard a beaglebone black development board, which has a Cortex-A8 1 GHz Processor with 512 MB RAM. Nominal flight control is performed with a Pixhawk2. The system includes a failsafe multiplexer between the pixhawk and RC receiver to recover from a hung process or unsafe maneuver originating from the flight computer. The resulting models are shown to be suitable for real time flight control design.

From the papers reviewed, it is clear that implementing experimental controllers or system identification is typically done on separated subsystems. A known flight controller is used for nominal flight and an experimental one is used for novel, untested processes.

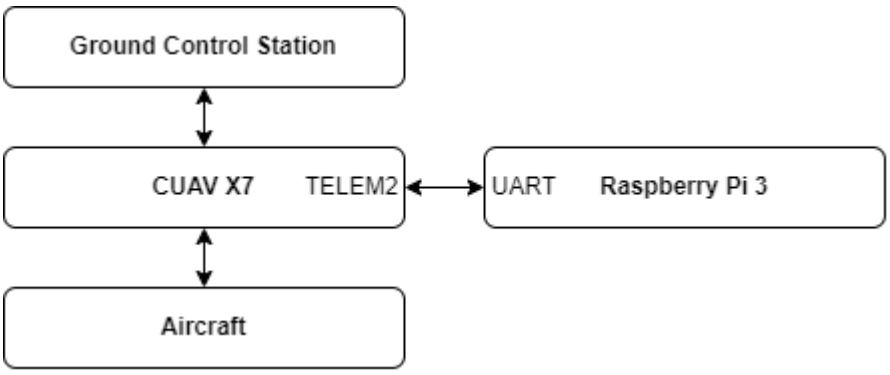
III. Methodology

A. Concept Development

Conclusions from the literature review and requirements presented by SINDy guide the design of a system architecture which can be safely used in a flight environment. The flight controller and system identification processes are separated onto two subsystems, allowing for failsafes to be implemented between the known, reliable flight controller and the experimental one. The PX4 autopilot firmware is chosen for its rich features, reliable flight controls, open source code base, and software in the loop simulation capabilities. It is also already integrated into many of the air frames available for testing at CFAR. The flight controller supports bidirectional communication and control with the lightweight Micro Air Vehicle (Mavlink) protocol. Using this protocol, aircraft state information such as vehicle speed, acceleration, and attitude can be sent to the companion computer for system identification. Mavlink libraries can be generated for several programming languages and platforms, making it a flexible communications protocol.

The companion computer offloads all computation for system identification from the autopilot. In future work, a MIAC may also be implemented on the companion computer, requiring a real time system identification process. For this reason, C++ is the language chosen. It has compiler support for many embedded platforms and has low overhead, enabling flexibility in the choice of companion computer. For this research, a Raspberry Pi 3 is used for ease of development and testing. It exposes a UART interface for communication with the autopilot using Mavlink. A high level system architecture is shown in figure 3.

Fig. 3 High level experimental system architecture



B. SINDy and STLSQ

Implementation of SINDy requires the adaptation of the Sequentially Thresholded Least Squares (STLSQ) algorithm to C++. This is the algorithm presented in the original paper which describes SINDy. As of the writing of this work, there are no existing implementations of the algorithm in C++. The algorithm as summarized by Kaiser and Brunton [7] is described in algorithm 1.

Algorithm 1 Sequentially Thresholded Least Squares

$\hat{\Xi}_0 \leftarrow (\Theta^T)\dot{X}$ ▷ Initial regression
while not converged **do**
  $k \leftarrow k + 1$ ▷ Increment iteration
  $I_{small} \leftarrow abs(\hat{\Xi} < \epsilon)$ ▷ Find indexes of coefficients smaller than ϵ
  $\hat{\Xi}_k(I_{small}) \leftarrow 0$ ▷ Set coefficients smaller than threshold to 0
  for all Variables **do**
    $I_{big} \leftarrow \tilde{}I_{small}$ ▷ Get indexes of coefficients which have not been thresholded
    $\hat{\Xi}_k(I_{big}) \leftarrow (\Theta^T)(I_{big})\dot{X}$ ▷ Regress on candidate functions without thresholded entries
  end for
end while

It is expected that a major factor in the speed of this algorithm is the performance of the regression algorithm, as the rest of the operations are computationally trivial. As the regression is a large series of matrix operations, it is important that the implementation is chosen carefully. Optimization of matrix operations has been a significant topic of research for decades, due to their prevalence in scientific computing, video games, and machine learning. Libraries developed for matrix operations have high maturity and are very optimized. For this reason, two existing C++ libraries, Armadillo and MLPack are used to handle data and operate on it instead of implementing a bespoke solution.

MLPack is a C++ library designed for machine learning applications [9]. It offers a selection of regression algorithms built on the Armadillo matrix library. Using MLPack provides an easy way to experiment with different regression algorithms.

Armadillo is an open source C++ library implementing high level language support for storage and manipulation of large matrices. Its backend implementation uses Basic Linear Algebra Subprograms (BLAS), which has support

for multiple platforms, such as Intel's parallelized MKL for multi-threaded CPUs, or NVidia's NVBLAS for GPU accelerated matrix computations [10]. By using Armadillo and libraries built upon it, future implementations have the flexibility of using higher throughput processing platforms to achieve real time performance.

C. SINDy and Aircraft System Identification

Applying SINDy to an aircraft is performed by formulating its governing equations in the same structure as equation 2. In this case, the state derivative matrix is $\dot{\boldsymbol{X}} = [\dot{\boldsymbol{u}}, \dot{\boldsymbol{v}}, \dot{\boldsymbol{w}}, \dot{\boldsymbol{p}}, \dot{\boldsymbol{q}}, \dot{\boldsymbol{r}}, \boldsymbol{p}, \boldsymbol{q}, \boldsymbol{r}, \dot{\alpha}, \dot{\boldsymbol{\beta}}]$, the candidate function matrix is $\boldsymbol{\Theta} = [\mathbf{1}, \boldsymbol{\phi}, \boldsymbol{\theta}, \boldsymbol{\psi}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}$,+ second order combinations], and the coefficient matrix is $\boldsymbol{\Xi} = [\boldsymbol{\xi_1}, \boldsymbol{\xi_2}, \ldots, \boldsymbol{\xi_{11}}]$, where each column corresponds to a state in $\dot{\boldsymbol{X}}$. These parameters are summarized in table 2.

Parameter	Definition
u	Body Velocity in X
v	Body Velocity in Y
w	Body Velocity in Z
ϕ	Pitch Angle
θ	Roll Angle
ψ	Yaw Angle
p	Pitch Rate
q	Roll Rate
r	Yaw Rate
α	Angle of Attack
β	Sideslip Angle

Table 2 Aircraft system parameters

A useful feature of the PX4 autopilot is that its state estimator provides the pitch angles and rates natively. However the body velocities, angle of attack, and sideslip angle have to be computed from other data sources. Linear velocities in the body frame are computed from linear inertial frame velocities, wind velocity estimates, and aircraft attitude in equation 3. The PX4 provides the parameters used in this method. The symbols W_x, W_y, W_z are the wind estimates in each dimension, and $\boldsymbol{R_z(\phi)}$, $\boldsymbol{R_y(\theta)}$, $\boldsymbol{R_x(\psi)}$ are rotation matrices representing the aircraft's attitude. At the time of writing, the states α and β have not yet been included in the system identification software.

$$\begin{bmatrix} u_{body} \\ v_{body} \\ w_{body} \end{bmatrix} = \begin{bmatrix} u_{intertial} - W_x \\ v_{body} - W_y \\ w_{body} - W_z \end{bmatrix} \boldsymbol{R_z(\phi) R_y(\theta) R_x(\psi)} \tag{3}$$

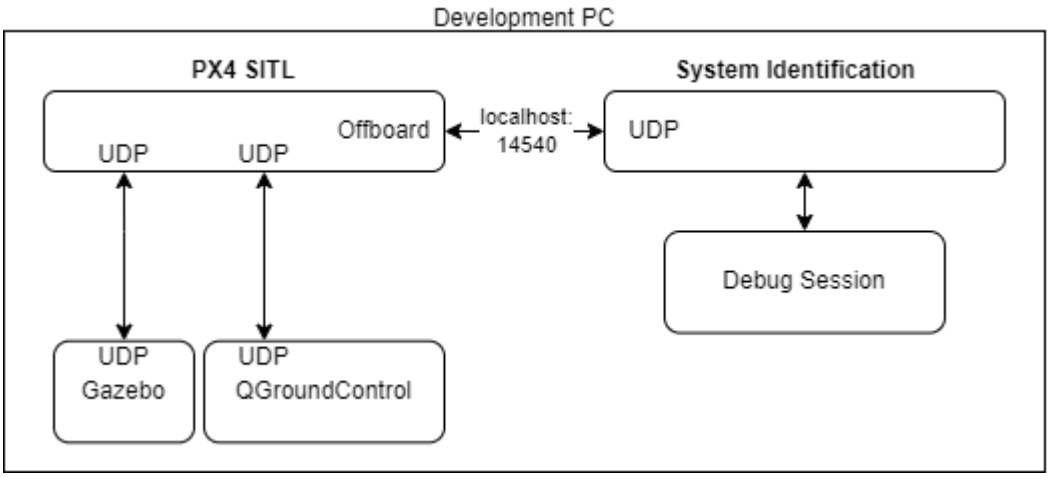
D. Configurations for Experimentation

Depending on the stage of development, the needs of the programmer in terms of software testing changes. In this project, two main testing paradigms are used, software in the loop and hardware in the loop simulation. The system configurations used in these two situations are discussed in the proceeding sections.

1. Software in the Loop

It is important to have a fast feedback loop during software development in order to catch errors in implementation and reasoning. A software in the loop (SITL) configuration is used to establish the initial tools required for communicating with the PX4. A specific SITL build of the PX4 firmware is used, which allows it to interface with a simulator, ground control software, and companion computer software all in a single development machine. This provides numerous benefits; the added complexity of working with embedded hardware in removed and the convenience of a fully featured IDE is easily accessible. Since the companion software and PX4 communicate using UDP in this configuration, running the software in debug mode presents no issues in collecting telemetry. A bulk of the software development occurs using this configuration. The SITL architecture is shown in figure 4.

Fig. 4 Software in the Loop architecture

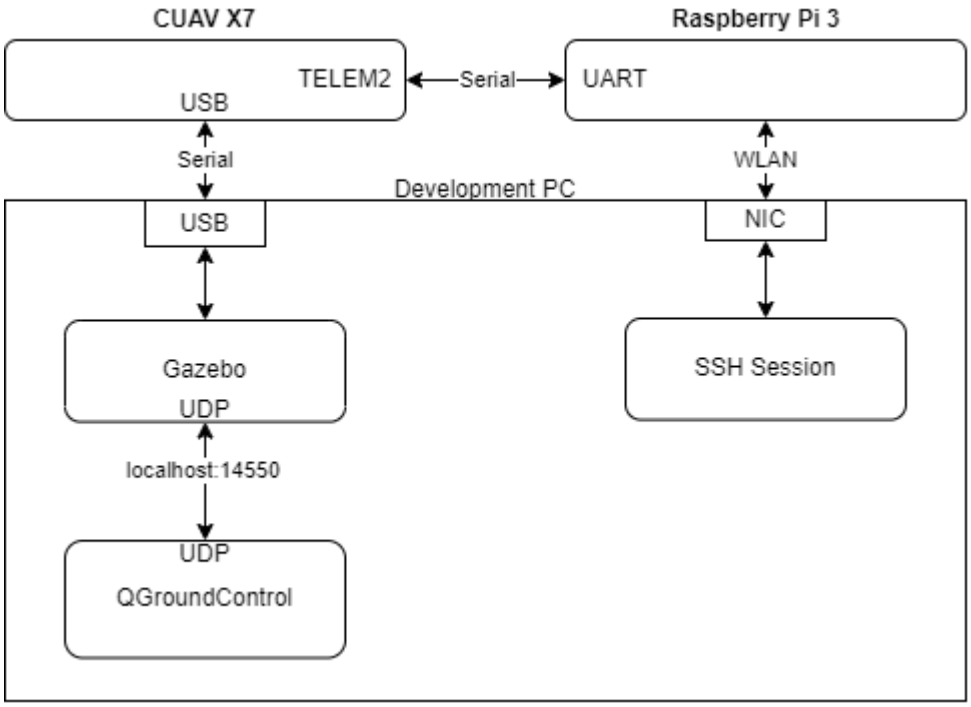


2. Hardware in the Loop Simulation

Before integrating the companion computer into an airframe for flight testing, a hardware in the loop (HITL) simulation is performed to validate the real hardware and communication interfaces to be used during flight. Invariably, the practicalities of embedded hardware introduce problems and it is worth verifying their operation before finding out in the field. This configuration also provides a convenient method of measuring the algorithm's real time performance on the bench. The HITL configuration is shown in 5.

Control of the companion computer during development is performed using a secure shell protocol (SSH) session

Fig. 5 Hardware in the Loop



from the development computer's terminal. Using this protocol allows for automation of the build and execution process from the development computer. In the HITL configuration, the companion computer software is running on an ARM based architecture rather than x86. Thus, a different compiler is required to generate the proper executable. Instead of building on the target system, a shell script is used to build the companion computer software with the appropriate compiler, login to the Raspberry Pi via SSH, transfer the executable, then run it with the desired command line arguments. This expedites the development process and minimizes the possibility of having files untracked by a version control system between the target and development computers. A wireless local area network provides a convenient connection for this process.

E. Software Architecture

Development of the system identification software is driven by the data requirements presented by SINDy and the communication interface of the PX4 autopilot. The following sections describe three main modules of the software.

1. Mavlink Interface

SINDy requires a set of time series state derivatives in order to estimate the aircraft's dynamics. Therefore, telemetry which directly or indirectly describes these states must be collected by the companion computer. A communication interface is adapted from example code provided by Mavlink developers. It sends and receives Mavlink messages which contain telemetry and commands[11]. Since it may communicate using a serial or UDP protocol depending on a SITL

or HITL configuration, the communication module is runtime configurable with the desired IP address, port number, and serial file descriptor. This module runs in its own thread to maintain consistent collection of telemetry.

2. Shared Data Buffer

Experimentation must occur to determine the algorithm's performance with different amounts of training data, so a data buffer is implemented to collect a configurable amount of telemetry. Once it is full, the entire buffer is consumed by the algorithm. Since the nominal telemetry rate is 50 Hz depending on the topic, there will be significant computational time available during the filling of the buffer. Thus, a multi-threaded, producer-consumer approach is taken to provide SINDy with computational power when the buffer is waiting for samples.

The producer-consumer problem is a classic multi-threaded software problem wherein a shared resource (the buffer) is filled by a producer (the autopilot) and emptied by a consumer (SINDy). To avoid undefined behaviour caused by race conditions between the two threads, a mutex and condition variables are used to control access to the buffer. The mutex acts as a lock on the buffer. When new data is added, the buffer cannot be emptied and vice versa. Condition variables allow the threads to communicate with each other; the producer thread notifies the consumer thread when the buffer is full, and the consumer thread notifies when it has been emptied. A subtle difference is implemented in this situation, because SINDy will only empty the buffer when it is completely full because it requires a specified amount of training data. This architecture is shown in figure 6.

3. System Identification

Once the buffer is full, the system identification thread is notified and dispatched. It conditions the buffered data to prepare it for coefficient computation with STLSQ. The first step in conditioning the data is resampling it to a common time base at a runtime configurable rate. This ensures that the state matrices and candidate functions are dimensioned properly because telemetry arrives at different rates and times. Once resampled, secondary computations are performed to obtain state information not natively provided by PX4. With all the state information collected, the candidate functions are calculated. At the time of writing, computation of 2nd order polynomials for all states are hardcoded. Finally, the state derivatives and candidate functions are passed into the STLSQ algorithm and the resulting coefficients are logged.

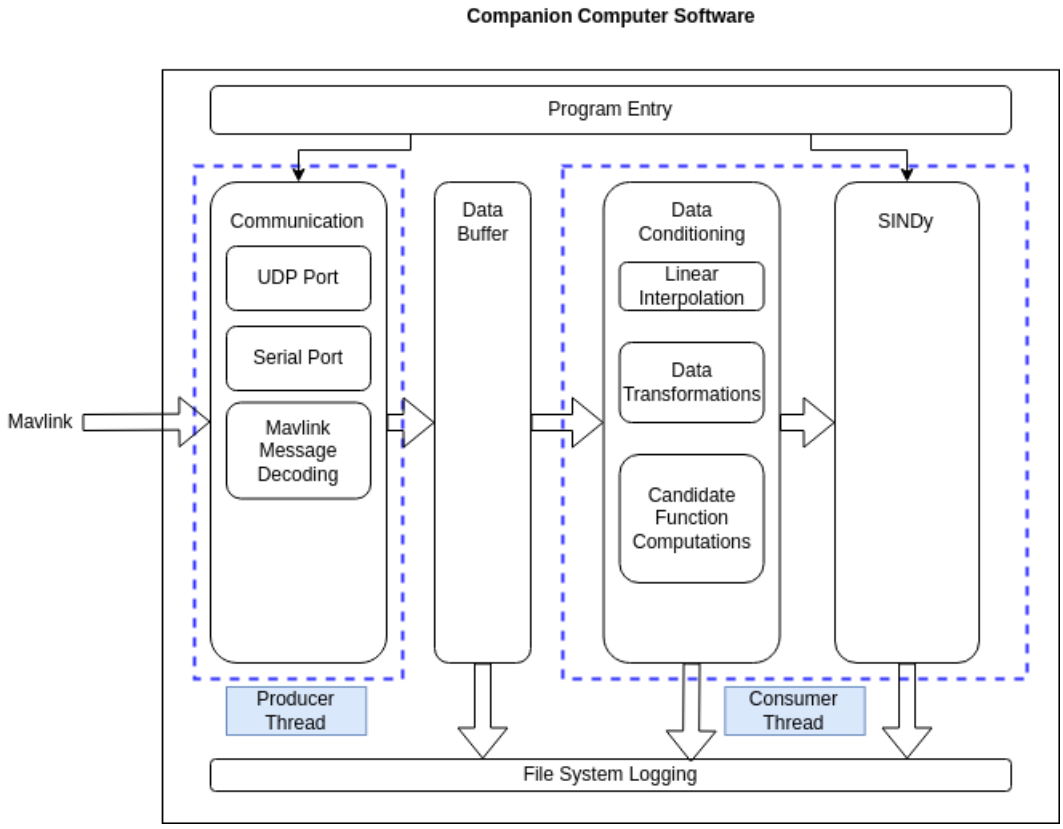
IV. Results

The timing results presented in this section are from execution on a desktop development computer with a Ryzen 5 1600 six-core processor and 16 GB RAM at 1600MHZ.

A. Validation

Validation of the SINDy module starts with data from a known system. A set of differential equations describing the lorenz attractor are used to verify the expected result. The equations describing it are shown in equation 4, where

Fig. 6 Companion Computer Software Architecture



$\sigma = 10, \rho = 28$, and $\beta = \frac{8}{3}$.

$$\begin{aligned} \dot{x} &= \sigma(y - x) \\ \dot{y} &= x(\rho - z) - y \\ \dot{z} &= xy - \beta z \end{aligned} \tag{4}$$

In order to apply SINDy to this problem, a series of 100,000 samples for each of $x, y, z, \dot{x}, \dot{y}, \dot{z}$ are generated with python code adapted from the PySINDy documentation [12]. The code listing corresponding to this is found in appendix VI.A. A set of 2nd order candidate functions $\boldsymbol{\Theta} = [\mathbf{1}, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}, \boldsymbol{x^2}, \boldsymbol{xy}, \boldsymbol{xz}, \boldsymbol{y^2}, \boldsymbol{yz}, \boldsymbol{z^2}]$ are computed from the generated states. Thus, the state derivatives have a total of 300,000 samples and the candidate functions have a total of 1,000,000 samples. Applying these candidate functions and state derivatives to the STLSQ algorithm yields an output shown in table 3. The coefficients correspond exactly with those in equation 4. The execution time of STLSQ is summarized in table 4 over 50 trials.

	x	y	z
1	0	0	0
x	-10.0000	28.0000	0
y	10.0000	-1.0000	0
z	0	0	-2.6667
x^2	0	0	0
xy	0	0	1.0000
xz	0	-1.0000	0
y^2	0	0	0
yz	0	0	0
z^2	0	0	0

Table 3 STLSQ Lorenz Attractor Solution

Mean	Standard Deviation
68.16 us	4.86285 us

Table 4 STLSQ Lorenz Attractor Execution Time

B. SITL

The execution time of STLSQ applied to simuated aircraft data is shown in figure 7. The aircraft data was obtained by running in a SITL configuration. The candidate functions for these trials are second order polynomials of the collected states, resulting in 45 candidate functions. An average over 10 trials is taken for each training time. A resampling rate of 200Hz is applied to the collected telemetry in order to align it to a common time base.

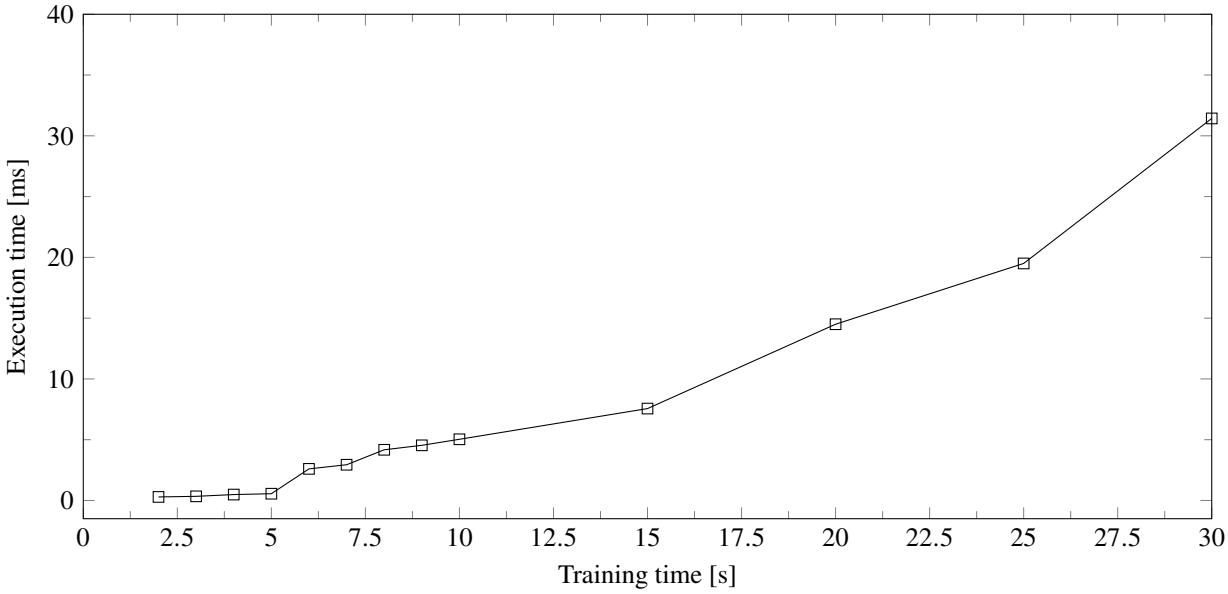


Fig. 7 Execution time of SINDy against training time

V. Conclusions

The STLSQ algorithm is implemented in C++ to perform SINDy on simulated aircraft data. Its implementation is verified using states and their derivatives generated from a 3D Lorenz attractor with correct results and a fast processing time of 68.16 μs. The software framework to interface STLSQ with a PX4 and the performance of the algorithm on simulated flight data is also demonstrated. From the preliminary execution times shown in figure 7 against training time, the application of the SINDy method to online aircraft system identification is feasible, especially with the availability of hardware accelerated artificial intelligence platforms specializing in accelerating matrix computations [13]. Further work must be performed to confirm the validity of the produced models to understand how much training data is required.

VI. Further Work

Due to time limitations, a full performance analysis of the algorithm on embedded hardware was not possible. It is recommended that the software produced in this work is ported to an embedded system to establish a baseline performance and determine its computational needs for the required execution time.

To fully apply SINDy to aircraft control as described in [7], the control outputs of the autopilot must be included in the candidate coefficient matrix. This can be implemented by using the actuator output status Mavlink messages, however care must be taken because the specific control outputs are highly dependent on the airframe used. In addition, computation of the angle of attack and sideslip angles should be implemented and included in the candidate functions.

A detailed analysis of the models produced by SINDy should be completed to further verify its validity in a real time environment. This may start with using a known physics model of an aircraft in Gazebo, applying system identification manouevers, and analyzing the resulting model. Once verified, the system should be integrated on an airframe and executed during a test flight.

Appendix

A. Lorenz Attractor Python Code

```
dt = 0.001
t_train = np.arange(0, 100, dt)
t_train_span = (t_train[0], t_train[-1])
x0_train = [-8, 8, 27]
x_train = solve_ivp(lorenz, t_train_span,
                    x0_train, t_eval=t_train, **integrator_keywords).y.T
x_dot_train_measured = np.array(
    [lorenz(0, x_train[i]) for i in range(t_train.size)])
```

B. Public Code Repository

https://github.com/schhtef/PX4-SID

References

[1] Camacho, E., and Bordons, C., *Model predictive control*, 2nd ed., 2007.

[2] Bazzocchi, S., "UNIVERSITY OF VICTORIA MECHANICAL ENGINEERING," , 2020.

[3] Riddick, S. E., Busan, R. C., Cox, D. E., and Laughter, S. A., "Learn to Fly Test Setup and Concept of Operations," ???? Includes the specifics on the computational platform used on the woodstock and E1 aircrafts. "The flight computer ran a Linux Ubuntu low latency operating system on an Intel Atom 1.9 GHz quadcore processor with 8 GB of RAM and a 120 GB solid state drive".

[4] Debusk, W. M., Chowdhary, G., and Johnson, E. N., "Real-Time System Identification of a Small Multi-Engine Aircraft," *AIAA*, 2012. Successfully uses the frequency domain Fourier Transform regression method to perform SID to obtain linear model. Implemented on fixed wing with FCS20 autopilot.

[5] Stastny, T. ., Dash, A. ., Siegwart, R., Stastny, T., and Dash, A., "Nonlinear MPC for Fixed-wing UAV Trajectory Tracking: Implementation and Flight Experiments ETH Library Nonlinear MPC for Fixed-wing UAV Trajectory Tracking: Implementation and Flight Experiments," 2017. https://doi.org/10.3929/ethz-a-010819607, URL https://doi.org/10.3929/ethz-a-010819607, "Onboard avionics 2 of 14 American Institute of Aeronautics and Astronautics including a 10-axis ADIS16448 Inertial Measurement Unit (IMU), u-Blox LEA-6H GPS receiver, and Sensirion SDP600 flow-based differential pressure sensor feed measurments to a Pixhawk Autopilot, an open source/open hardware project started at ETH Zurich.19 Pixhawk features a 168 MHz Cortex-M4F microcontroller with 192 kB RAM for online state estimation and low-level control"

"As processing power on the Pixhawk microcontroller is somewhat limited, an additional onboard ODROIDU3 computer with 1.7 GHz Quad-Core processor and 2 GB RAM, running Robotic Operating System (ROS)21 is integrated into the platform for

experimentation with more computationally taxing algorithms."

Limited to lateral direction control. Nonlinear model used in a NMPC system.

[6] Valasek, J., Lu, H.-H., Rogers, C. T., and Goecks, V. G., "Online Near Real Time System Identification on a Fixed-Wing Small Unmanned Air Vehicle Online Near Real-Time System Identification on a Fixed-Wing Small Unmanned Air Vehicle," ???? https://doi.org/10.2514/6.2018-0295, URL https://www.researchgate.net/publication/322311346.

[7] Kaiser, E., Kutz, J. N., and Brunton, S. L., "Sparse identification of nonlinear dynamics for model predictive control in the low-data limit," 2017. https://doi.org/10.1098/rspa.2018.0335, URL http://arxiv.org/abs/1711.05501http://dx.doi.org/10.1098/rspa.2018.0335.

[8] NCSS, "Ridge Regression," , ???? URL https://ncss-wpengine.netdna-ssl.com/wp-content/themes/ncss/pdf/Procedures/NCSS/Ridge_Regression.pdf.

[9] Curtin, R., Lozhnikov, M., Mentekidis, Y., Ghaisas, S., and Zhang, S., "mlpack3: a fast, flexible C++ machine learning library." *Journal of Open Source Software 3*, 2018, p. 726.

[10] Conrad, S., and Ryan, C., "Armadillo: a template-based C++ library for linear algebra," *Journal of Open Source Software*, Vol. 1, 2016, p. 26.

[11] Lorenz, M., and Trent, L., "C UART Interface Example," , 2013. URL https://github.com/mavlink/c_uart_interface_example.

[12] dynamicslab, "PySINDy Documentation," , ???? URL https://pysindy.readthedocs.io/en/latest/examples/3_original_paper.html#lorenz-system-nonlinear-ode.

[13] NVidia, "Advanced AI Embedded Systems," , 2022. URL https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/.