# Toward Nonlinear Flight Control for Fixed-Wing UAVs: System Architecture, Field Experiments, and Lessons Learned

Erlend M. Coates, Dirk Reinhardt, Kristoffer Gryte and Tor Arne Johansen

*Abstract*— **Inner-loop control algorithms in state-of-the-art autopilots for fixed-wing unmanned aerial vehicles (UAVs) are typically designed using linear control theory, to operate in relatively conservative flight envelopes. In the Autofly project, we seek to extend the flight envelopes of fixed-wing UAVs to allow for more aggressive maneuvering and operation in a wider range of weather conditions. Throughout the last few years, we have successfully flight tested several inner-loop attitude controllers for fixed-wing UAVs using advanced nonlinear control methods, including nonlinear model predictive control (NMPC), deep reinforcement learning (DRL), and geometric attitude control. To achieve this, we have developed a flexible embedded platform, capable of running computationally demanding low-level controllers that require direct actuator control. For safe operation and rapid development cycles, this platform can be deployed in tandem with well-tested standard autopilots. In this paper, we summarize the challenges and lessons learned, and document the system architecture of our experimental platform in a best-practice manner. This lowers the threshold for other researchers and engineers to employ new low-level control algorithms for fixed-wing UAVs. Case studies from outdoor field experiments are provided to demonstrate the efficacy of our research platform.**

## I. Introduction

### A. Background and motivation

Fixed-wing unmanned aerial vehicles (UAVs) can be an essential part of robotic networks deployed in ocean monitoring, search and rescue, crop monitoring in agriculture, and logistic infrastructure, to name a few. For guidance, navigation, and control (GNC) of these vehicles, several open-source hardware and software systems exist. For instance, the ArduPilot [1] and PX4 [2] software suites have been widely adopted by industry, research institutions, as well as hobbyists. Over the last decade, the open-source ecosystems have stimulated many drone startups and considerable growth in the unmanned systems industry.

The low-level control algorithms (i.e., control of the vehicle's attitude and speed) are typically based on linear control theory. The controllers are designed to operate close to some nominal operating conditions (trim conditions), and the flight performance is usually satisfactory for non-aggressive maneuvers under relatively calm wind conditions. However, for more aggressive flight trajectories, nonlinear aerodynamic effects become increasingly significant, causing performance deterioration and in the worst case, instability. A practical approach to this problem is to limit the range of flight conditions the UAV is operated in, referred to as the flight envelope.

The nonlinear nature of the underlying physics suggests that methods in nonlinear control theory should instead be used to design more advanced autopilots that are less conservative, more capable of agile maneuvering, and which allow for a wider flight envelope. Such functionality will increase the capabilities of fixed-wing UAVs and can lead to new innovative use-cases and products. Although promising results exist using nonlinear methods such as e.g. dynamic inversion [3], backstepping [4], [5], and adaptive control [6], these methods have arguably not been developed to the mature level required for implementation in UAV autopilots. Many nonlinear control algorithms have not been sufficiently tested outside nominal operating conditions for fixed-wing UAVs. Moreover, flight control methods developed for high-performance aircraft are not easily adopted since fixed-wing UAVs typically operate at low speeds with small stall margins, [7], and only limited resources are available for tuning in many cases.

Our research seeks to extend the flight envelope of fixed-wing UAVs to perform more aggressive maneuvers and operate in a wider range of weather conditions. In particular, we base our autopilot designs on nonlinear control theory and then demonstrate the performance of these methods in extensive field experiments. We argue that to leverage the full potential, the low-level, linear PID loops need to be replaced with nonlinear counterparts. To validate this hypothesis, the experimental platform requires direct access to the actuators.

Over the course of the last few years, we have successfully flight tested several advanced nonlinear algorithms for attitude control of fixed-wing UAVs: nonlinear model predictive control (NMPC), deep reinforcement learning (DRL), and geometric attitude control (GAC), each demanding different capabilities in the onboard avionics stack. In particular, NMPC is very computationally demanding, as a nonlinear optimization program needs to be solved at every controller update. DRL is more computationally efficient, but requires access to some framework for artificial neural networks, which is not provided by the standard autopilots. As a result of this work, we have established a common hardware

This research was funded by the Research Council of Norway through the Centres of Excellence funding scheme, grant number 223254 NTNU AMOS, and grant number 261791 Autofly.

Erlend M. Coates, Dirk Reinhardt, Kristoffer Gryte and Tor Arne Johansen are with the Centre for Autonomous Marine Operations and Systems, Department of Engineering Cybernetics, NTNU, Trondheim, Norway (e-mail: {erlend.coates, dirk.reinhardt, kristoffer.gryte, tor.arne.johansen}@ntnu.no)

platform, system architecture, and operating procedures for flight experiments of low-level nonlinear control algorithms for fixed-wing UAVs. In this paper, we summarize this work with a *focus on the challenges and lessons learned and document our experimental platform in a best-practice manner.*

## B. Related work

We classify algorithms as either *high-level* (guidance/outer-loop) controllers, or *low-level* (inner-loop) controllers. Whereas the high-level algorithms are typically implemented on a single-board computer, transmitting references to standard low-level control loops running on some commercial or open-source autopilot (e.g ArduPilot [1] or PX4 [2]), the low-level algorithms need direct access to the actuators. A comprehensive review of different control architectures is not in the scope of this paper. Although we mention some notable works that include experimental verification of high-level controllers, our focus is on low-level control and the computing platforms and system architectures used.

The low-level algorithms can be further classified concerning their requirements to the embedded computing platform as either: (a) *lightweight*, or (b) *computationally intensive*. Lightweight algorithms are easily integrated into open-source autopilots and can be run directly on hardware platforms such as the Pixhawk or CubePilot series of autopilots. For instance, in [8], the low-level control framework is based on explicit model predictive control (MPC) using linearized models and implemented directly on the resource-constrained onboard avionics. The unified guidance and low-level control architecture in [9] is implemented directly on a mRo PixRacer with a 180Mhz ARM Cortex M4 processor, by modifying the existing PX4 middleware.

Among the more computationally intensive algorithms, the NMPC scheme in [10] first poses a feasibility problem to generate dynamically feasible paths from an initial guess found via a Rapidly-Exploring Random Tree (RRT) algorithm combined with spline smoothing. Then, a time-varying Linear Quadratic Regulator (LQR) is used to follow the nominal trajectories calculated by the planner in a receding-horizon manner. Their experiments are conducted in a controlled lab environment with a motion capture system and desktop computer for nonlinear optimization. The desired actuator signals are transmitted to the vehicle through radio communication. The authors in [11] identify second-order models of the autopilot-controlled low-level dynamics which is used in their MPC guidance algorithm. The computing platform is an Intel UP board running a Robot Operating System (ROS) node. In [12] MPC controllers for trajectory-tracking of both fixed-wing and rotary-wing UAVs are presented, with a special focus on ROS integration. ROS is also used in [13], where an optimal path-following controller for windy conditions transmits roll and pitch angle references to low-level controllers running in PX4 on a Pixhawk.

Looking wider, into the realm of multirotor UAVs, [14] presents a controller for unified trajectory optimization and tracking with a hexacopter UAV. They explicitly aim at improving performance by giving the controller direct actuator access. The optimal controller is implemented on a stationary Intel Core i7 processor that sends the velocity commands of the rotors to the flight control unit. State estimation is done with an optical motion capture system. The work in [15] aims at showing that current optimal control solvers have become fast enough to handle the computational burden that is coming with highly dynamic robotics applications such as the low-level control of multirotors for which they provide experimental results.

DRL algorithms are computationally efficient during online execution, but often require more flexibility than is typically provided by standard flight controllers. Some specialized solutions exist, e.g. in [16], a low-level reinforcement learning (RL)-based controller for multirotors is validated experimentally using a PixRacer flight control board. In [17], a model-based RL algorithm for low-level control of a Quadrotor is validated using the open-source Crazyflie 2.0 quadrotor. pulse-width modulation (PWM) commands are calculated on a ROS server running on the ground and sent to the UAV using radio. The Neuroflight neural network-based flight control firmware is presented in [18], where experimental validation of a low-level RL controller for a quadcopter is carried out using a 216MHz ARM Cortex-M7 microcontroller.

## C. Contributions

There does not seem to be any documented standard solution or best practice for computationally demanding fixed-wing UAV control architectures that require low-level access to the actuators. Motivated by this, and based on our experience with extensive field testing of a wide range of nonlinear control algorithms, the main contributions of this paper are:

- An overview of the Autofly project, where we describe novel experimental results from flight testing of several advanced nonlinear control algorithms for attitude control of fixed-wing UAVs while *articulating the key challenges and lessons learned.*
- A set of *constructive guidelines* on how to deploy an experimental platform that is well-suited for the evaluation of control algorithms that require a lot of computational resources and direct access to the actuators. Our approach also allows for switching between the experimental algorithm and the standard autopilot, allowing safe experimental testing at an early stage.

In summary, this lowers the threshold for other researchers and engineers to employ new low-level control algorithms for fixed-wing UAVs. The individual components are off-the-shelf and thus readily available. This facilitates research with a minimum of resources without deviating from the main research focus.

TABLE I: Features of the compared controllers.

| | GAC | DRL | NMPC |
|---|---|---|---|
| Demand for comp. resources | Low | Low | High |
| Needs dynamic model | Low | Medium/High | High |
| Optimality wrt. cost/reward | Low | Medium | High |
| Constraint satisfaction | No | No | Yes |
| Interpretable performance | Yes | No | Yes |
| Open software available | Not needed | Yes | Yes |
| Available stability proofs | High | Low | Medium |

### D. Organization of the paper

The rest of this paper is structured as follows: In Section II we describe the control algorithms that have been successfully implemented and tested using our experimental platform. The system architecture is presented in Section III, and our test procedure, for both ground testing and flight experiments, is described in Section IV. In Section V, we summarize our results, and in Section VI we discuss the lessons learned before giving some concluding remarks in Section VII.

## II. Control algorithms: Overview

In this section, we describe the main control algorithms evaluated in the Autofly project, namely NMPC [19], DRL [20], GAC [21], as well as a proportional-integral-derivative (PID) benchmark implementation based on the ArduPilot [1] fixed-wing attitude controller. The capabilities needed to effectively run these algorithms online define the requirements for the system architecture presented in Section III. This section also provides the necessary background for the experimental results of Section V. An in-depth discussion of the experimental controllers is out of scope of this work and more details can be found in each of the respective references [19], [20], [21]. A comparison of the most significant features of the controllers is given in Tab. I.

A significant part of our controllers is, to some extent, relying on accurate models of the UAV. We also depend on dynamic models in our simulators, discussed in Section IV. Our dynamic models are based on previous and ongoing modeling efforts, in particular [22], [23], [24].

### A. PID Benchmark

As a benchmark, we use the widely adopted ArduPilot [1] open-source autopilot and compare our methods to the ArduPlane PID attitude controller for fixed-wing UAVs. To get a fair comparison, we implemented a version of this in the same system as our algorithms. This means that all algorithms run on the same hardware, in the same software environment, and with the same communication latencies. For reference, we have documented the equations in Appendix A of [20].

### B. Nonlinear Model Predictive Control (NMPC)

NMPC allows us to explicitly encode the flight envelope in the controller design as nonlinear constraints in an optimal control problem (OCP) that can be solved regularly to obtain an optimal control input trajectory. The OCP over a prediction horizon $T$ usually has the form

$$\begin{aligned}
\min_{\mathbf{x}(\cdot),\mathbf{u}(\cdot)} \int_0^T l(\mathbf{x}(\tau),\mathbf{u}(\tau),\mathbf{r}(\tau))d\tau + \frac{1}{2}\mathbf{s}^\top \mathbf{P}\mathbf{s} \qquad & t \in [0,T) \\
\text{s. t.} \quad \mathbf{x}(0) = \mathbf{x}_0 \qquad & \\
\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t),\mathbf{u}(t),\mathbf{d}(0) \qquad & t \in [0,T) \\
\mathbf{h}(\mathbf{x}(t),\mathbf{u}(t),\mathbf{s}) \geq \mathbf{0} \qquad & t \in [0,T),
\end{aligned}$$

where the cost consists of $l$, denoting a stabilizing stage cost that is often in quadratic form, and a penalty for the slack variables $\mathbf{s}$ that are included for constraint relaxation to guarantee the feasibility of the approximating quadratic program (QP). The constraints include the initial condition, denoted by $\mathbf{x}_0$, dynamic constraints in form of the continuous model, denoted by the vector Ordinary Differential Equation (ODE) defined by $\mathbf{f}$, and inequality constraints to reflect operational and actuator limits, denoted by $\mathbf{h}$. Optimal performance with respect to a defined cost function and prediction in addition to constraint satisfaction by use of the actuation system in an integrated multiple-input multiple-output (MIMO) fashion are traits that are hard to achieve with the existing autopilot software such as ArduPlane. The drawback however is its increased computational complexity and requirement for a dynamic model of the UAV, [25], which demands powerful embedded computing platforms and a considerable engineering effort in system identification [26]. The field of MPC is quite mature and conditions for performance and stability guarantees exist on a theoretical level but may be hard to formally verify for a particular system [27].

### C. Deep Reinforcement Learning (DRL)

In addition to NMPC, we looked at DRL, a set of data-driven methods for approximate optimal control, where the controller is implemented as an artificial neural network (ANN) [28]. These methods can operate on and optimize control performance for the full nonlinear dynamics in a model-free manner (at least in theory), their online operation is generally highly computationally efficient, and can exhibit (nearly) arbitrarily complex behavior. A downside of the DRL methods is the lack of interpretability of the deep neural network; stability, robustness, and constraint satisfaction properties are not guaranteed. In addition, DRL requires a lot of training data, a challenge that is further complicated for flight control applications by the high inherent risk associated with data collection using a suboptimal controller. A common approach is to instead train the controller in a simulator environment, which motivates the need for a good model. To deal with model mismatch when transferring the trained controller to the field, "Sim2Real" measures such as domain randomization [20] are typically applied.

### D. Geometric Attitude Control (GAC)

A third approach we considered is to apply nonlinear control methods based on Lyapunov stability theory, in particular GAC. Such methods require only a fraction of the computational resources required by NMPC, and stability and robustness guarantees can be given under some assumptions for the particular system under study, even with limited knowledge of system models. However, constraint handling is difficult to address and optimality is not addressed. Traditionally, the orientation/attitude of aircraft in 3D space is parametrized using Euler angles, which have singularities for certain orientations, which in itself contributes to a more limited flight envelope. In our work, we employ a global, nonsingular reduced attitude representation on the two-sphere. In addition to avoiding singularities, this enables more efficient, shortest path (geodesic) rotation maneuvers, [29].

## III. System Architecture

In this section, we describe our experimental platform, which has evolved over several years as a result of a wide range of research activities carried out at the NTNU UAV-lab. An alternative system architecture is described in [30], which provides a flexible architecture for system integration that is well-suited for research on high-level planning, guidance, and payload control, but less ideal for low-level control research. For our purpose, the goal was to extend the existing capabilities to satisfy the following requirements:

1) An embedded platform powerful enough to run low-level NMPC online on-board the vehicle at a sufficiently high update frequency.
2) This platform should also have direct access to the actuators.
3) The system should be flexible enough to run a wide range of advanced control algorithms.
4) To lower the threshold for early testing of highly experimental low-level control algorithms (and to lower the risk of crashing), we needed some way to safely transition between the well-tested (and trusted) standard autopilot and our experimental algorithms.
5) For continued safe operation, the added functionality should not interfere with the existing fail-safe systems.
6) A software-in-the-loop (SITL) simulation environment to test the airworthiness of the low-level algorithms before conducting flight experiments.
7) Support for an automated reference generator to gather repeated sample trajectories for system identification, as well as evaluation and comparison of different algorithms.

An overview of the hardware configuration and communication architecture of the UAV and ground station is depicted in Fig. 1. We proceed by describing each main element of the system architecture. The following discussion concerns fixed-wing UAVs, given that we focus our research on this type of UAV. Note, however, that it is straightforward to translate the hardware architecture and outlined test procedures to other UAV morphologies. The only requirement is access to the actuators through PWM signals and a fallback controller, which is usually available on off-the-shelf platforms.

### *A. UAV Platform*

Our platform is built around a Skywalker X8 airframe, depicted in Fig. 2. The X8 is a tailless aircraft with two elevon control surfaces, one on each wing, which can be moved differentially to produce a rolling acceleration, or collectively, to produce a pitch acceleration. The control signals consist of PWM signals to the two servo motors that actuate the elevons, and the throttle signal (also PWM) to a consumer-grade Electronic Speed Controller (ESC) that controls the motor and propeller in the back of the UAV.

The standard avionics flight stack is centered around a Cube Orange[1] that is running ArduPlane open-source autopilot, which is the fixed-wing build of the ArduPilot firmware [1]. The sensor suite consists of triple redundant inertial measurement units (IMUs) with magnetometers, pressure sensors for altitude and airspeed (pitot-static tube), and a global navigation satellite system (GNSS) receiver.

### *B. Payload Computer*

Alongside the Cube Orange, we use the Khadas Vim3[2] single-board computer (SBC) that includes four 2.2Ghz Cortex-A73 cores and two 1.8Ghz Cortex-A53 cores. We initially started using other SBCs, including the Odroid-XU4 and a Raspberry Pi 4. After a series of benchmarking tests, we settled with the Khadas Vim3, mainly driven by the computational requirements of the NMPC.

On the SBC we run the DUNE Uniform Navigation Environment. DUNE is part of the LSTS toolchain [31] developed at the Underwater Systems and Technology Laboratory (LSTS), University of Porto. DUNE allows us to (similarily to ROS) write different tasks that run independently from each other on separate threads or processes while exchanging data using a message bus mechanism.

The NMPC is implemented using acados [32], which we interfaced as a DUNE Task. The closed-loop runtime of the solver was benchmarked for each SBC based on simulations that reflect targeted maneuvers. Benchmarking results for the Khadas Vim3 are depicted in Fig. 3, which show the closed-loop runtime of the solver to find solutions to the OCP at each solver update for a Monte-Carlo study that includes a range of initial conditions and environmental disturbances. Approximately 96% of the simulations allow the solver to find a solution in less than 50 ms after two controller updates when warm-starting the solver based on the time-shifted previous solution. Therefore, we chose an update period of 50 ms in the experiments, which led to satisfactory performance. More details can be found in [25].

The DRL controller is implemented as a DUNE task in C++ with the ANN implemented in TensorFlow. Benchmarking tests show that the controller can run with an update

[1]https://cubepilot.org/
[2]https://www.khadas.com/vim3

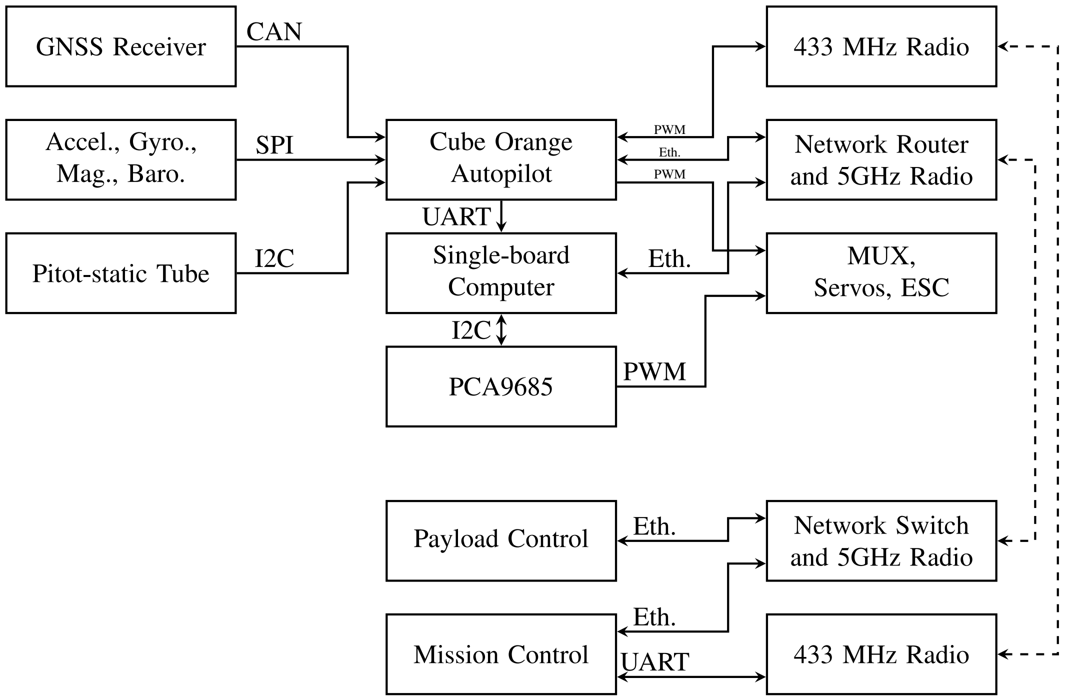


Fig. 1: Hardware configuration of the experimental platform.

Fig. 2: Skywalker X8 fixed-wing UAV.

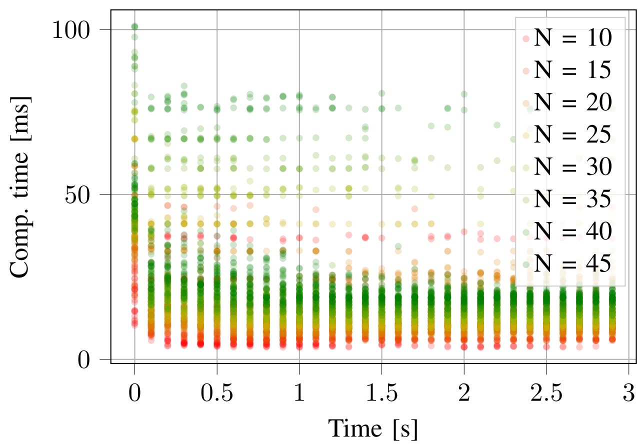


Fig. 3: NMPC benchmarking results for the employed SBC for different prediction horizons $N$ in a direct multiple-shooting scheme with 0.1 s shooting interval.

rate of several thousand hertz. However, this is orders of magnitude faster than needed since state estimates are delivered at 50 Hz (see next section). Naturally, the computational demands of the DRL controller is not the bottleneck when selecting our hardware, but rather that of the NMPC.

## *C. State Estimates*

State estimates, including estimates of the local wind velocity, are provided by ArduPilot's extended kalman filter (EKF), and is propagated to the SBC together with attitude references (either originating from the pilot's radio transmitter or ArduPlane's guidance system) and other auxiliary signals via a serial communication link using the MAVLink protocol. This provides our controllers with all the necessary data. The MAVLink communication was configured to provide data at the highest possible rate, which in this case is 50 Hz, corresponding to the loop rate of the ArduPlane scheduler. Communicating such large amounts of data at a high rate turned out to be a demanding task for the ArduPilot system, which in turn made us select the Cube Orange among several candidate autopilots. Cube Orange is (as of February 2022) the most powerful of the CubePilot series of autopilots, with a 400MHz ARM Cortex M7 processor. Benchmarking tests showed that less powerful autopilot hardware platforms such as the Pixhawk 1 and Pixhawk 2.1/Cube Black were not powerful enough to handle the high data throughput over the serial link.

## *D. Actuators*

Our controllers output desired throttle and control surface deflections that are converted to PWM duty cycle using static linear maps. For the elevons, these were identified based on lab experiments using a camera.

*Remark 1:* For future work, an interesting extension to a static linear mapping would be to identify a second-order model of the actuator dynamics. This can be done with a series of step responses that can be recorded in a motion capture lab [33]. More advanced servos that provide position feedback and control of the surface deflection angles can also be considered for model-based control.

Most of the SBCs require additional hardware for PWM output. For instance, the Odroid-XU4 has no hardware PWM ports, and the Raspberry PI 4 only has two (we need three). For the Khadas Vim3, we chose a solution based on a PCA9685 servo driver which is interfaced through Inter-Integrated circuit (I2C) communication.

### *E. Multiplexer Switch*

The PWM signals to the actuators can be set by both computing platforms. A PWM multiplexer (MUX) is used to switch between the controller that runs on the SBC and the ArduPilot controllers. A switch on the pilot's radio transmitter is mapped to the MUX switch using ArduPilot's RC pass-through functionality, allowing the pilot to choose the output source at any given time, including a manual recovery if loss of control should occur. To achieve an additional layer of safety, the manual mode always overrides the SBC output.

This architecture allows for a redundant PID controller to run on the autopilot that may overwrite the commands from the experimental controller whenever necessary to ensure a safe operation, for example, if instability occurs or when the required update rates of an optimization-based controller such as MPC can not be met by the employed solver. The switching mechanism enables us to safely engage the highly experimental low-level control code in flight, while takeoff and landing are performed by the pilot operating the standard ArduPlane autopilot.

When switching between different controllers, we reset integral terms to zero to avoid potential stability or performance issues. In ArduPlane this is done every time a switch happens between "MANUAL" mode and stabilized modes. For our custom controllers implemented in DUNE, we do this through dedicated parameters.

For an alternative control selection method, based on a performance monitoring scheme, together with a thorough discussion of MPC employed on alternative computing platforms such as field-programmable gate arrays (FPGAs), see [34].

### *F. Fail-Safe*

The ArduPilot system includes standard fail-safe functionality, such as an automatic return to launch (RTL) mode that is triggered if the pilot's radio transmitter signal is out of range or otherwise lost. Since this includes the MUX switch signal, we had to augment the fail-safe functionality. Otherwise, if the signal is lost when the SBC is in control, we would have no way to recover the aircraft should our algorithms fail. To solve this, the fail-safe configuration of our RC receiver (FrSky) is set to move the controls to the following preset values in the case of a lost control signal for some time:

- The mode switch is set to RTL.
- The MUX switch is set such that the Cube Orange's PWM output is sent to the servos.
- The other controls are set such that they correspond to centered sticks on the transmitter.

This way, our fail-safe system works similar to ArduPilot's. In addition, we are sure that ArduPilot will be in control should we lose the control signal. A potential downside of this is that the ArduPilot system will not be aware that the control signal is out of range since the receiver channels are just set to some preset values, instead of the usual "no signal".

### *G. Ground Station*

Communication with the ground station is handled through a redundant radio link using one 433MHz SiK Telemetry Radio and a 5GHz Ubiquiti Rocket M5, both providing MAVLink communication with the Cube autopilot. The 5GHz radio also enables us to communicate with the SBC on a local aera network (LAN) through an onboard network router (see [30] for details).

We use the ArduPilot-compatible ground control software Mission Planner running on a dedicated computer. Both radio communication links are used for redundancy, and multiplexing of the two radio signals is handled by MAVProxy.

Control of the DUNE controller tasks is done through Neptus, which is the command and control framework of the LSTS toolchain, communicating with DUNE using the Inter Module Communication (IMC) protocol. Neptus allows the operator to set configuration parameters, monitor telemetry data, and execute commands on the SBC.

### *H. Reference Generation for Automated Testing*

We can use pre-defined signals to overwrite references coming from the guidance controller to test our low-level motion controllers in a repeatable manner. This means that we can e.g. use ArduPlane to fly a square waypoint mission, where we run repeated custom maneuvers when on the long sides of the square. Step sequences and chirp signals with increasing frequency turned out to be a good way to test the closed-loop dynamics with different control algorithms that need to be compared.

We follow a similar approach to collect data for identifying dynamic models of a particular airframe. However, instead of manipulating reference signals for the low-level controller, the actuator signals of a particular actuator are overwritten by suitable step sequences or oscillating signals. A frequency analysis of the open-loop model dynamics or the linearized closed-loop system around trim states can be used to guide the parametrization of the test signals such that their power spectral density covers the natural frequencies of the system. The aim is to sufficiently excite the dynamics such that the collected aerodynamic data can be used for system identification.

## IV. Test procedures

This section describes our testing procedures. To assess the airworthiness of our algorithms, we use a three-stage ground-testing process before finally attempting field experiments: (a) initial verification of promising designs in our laptop simulators, (b) SITL simulation to verify the platform-specific implementation of the algorithms, and (c) system integration testing at the lab.

### A. Python Simulator

As an initial verification step, prototype implementations of promising designs are first tested in simulator environments implemented in Matlab or Python. Model mismatch can be introduced in a controlled environment to assess the algorithms' robustness to modeling errors. Also, initial tuning guidelines are established during this stage. Our Python-based DRL test bench is publicly available online[3]. This is the same simulator that is also used for training of the DRL algorithms, using our OpenAI Gym wrapper[4].

### B. Software-In-The-Loop (SITL) Simulations

The SITL simulator is based on a combination of a SITL configuration of our DUNE application, in combination with ArduPilot's SITL framework, using a JSBSim simulation model for the Skywalker X8 based on our previously mentioned models. The standard SITL framework is sufficient for systems where the SBC only sends commands to a low-level autopilot using the MAVLink interface, e.g. when testing high-level guidance controllers. However, since we need to simulate the case where the SBC has direct access to the actuators, we need to extend this functionality.

Our solution uses MAVLink's "RC override" functionality to emulate the behavior of our physical system. In DUNE, instead of sending actuator signals to the PWM driver, the controller output is transmitted to ArduPlane SITL using our MAVLink interface, using the RC override message. In the simulator, these values are interpreted as servo setpoints, *as if the UAV was under manual control*. Therefore, for this to work, ArduPlane needs to be in "MANUAL" mode.

To achieve automated testing of different maneuvers, we implemented a DUNE Task that essentially provides scripting capabilities of a succession of different maneuvers and system commands, including automated arming, takeoff, and loitering, mode switching, as well as switching between ArduPlane and our controllers.

[3]https://github.com/eivindeb/pyfly
[4]https://github.com/eivindeb/fixed-wing-gym

### C. Lab Testing

We conduct system integration tests on the physical hardware at the lab, checking all communication channels and verifying that critical systems work as expected. This includes the MUX switch, data logging, and telemetry. In particular, we check edge cases concerning arming/disarming of the propeller and confirm that the MUX switch does not interfere with the safety-critical features.

When preparing for field tests, we first communicate the expected behavior of our system to the pilot and demonstrate safety-critical features. An important tool we use when verifying and configuring our controller implementations is the surface deflection test ("ground test"), where we check that the control surfaces move in the correct directions in response to manually tilting the vehicle, or moving the transmitter sticks. Moreover, the magnitude of the deflection is an indicator of the controller response.

### D. Field Experiments

When performing field experiments, we typically use a team of three persons: (1) the pilot (first in command), operating the UAV in the manually controlled modes using an RC transmitter, (2) ground station operator (second in command) operating the automatically controlled modes and setting ArduPilot parameters through Mission Planner, and (3) one researcher controlling the payload computer through Neptus. This is typically the researcher that designed the experiment or implemented the algorithm that we test. During experiments, the team communicates using radio. Additional personnel, if any, is in charge of taking notes.

The flight testing procedure can be roughly broken down into the following steps:

1) After all pre-flights checks are passed, the pilot takes off manually and puts the UAV into loiter mode or a square pattern of waypoints.
2) With the experimental algorithm running in the background, we monitor its outputs while comparing them with the PWM values set by ArduPlane.
3) If everything looks good, we switch to our controller using the MUX switch mapped to a switch on the pilot's radio transmitter. When testing controllers with dynamic elements such as integral action or disturbance observers, the dynamic elements are engaged (or their states reset) when we perform the switch. This is to avoid any wind-up or other potential issues.
4) We then observe the behavior of the experimental controller and test it with increasingly challenging maneuvers, starting with straight and level flight. If the UAV performs any sudden maneuvers, or if substantial oscillations or instability occurs, the pilot takes back control over the UAV by using the MUX switch. The pilot's visual eye contact with the vehicle is aided by the other operators, constantly monitoring telemetry data, and warning the pilot if needed.

5) After some initial tuning, we initiate the automated maneuver sequences for tuning and repeatability of the collected evaluation data. This is especially useful when comparing the performance of two controllers.
6) When data collection is complete, we switch the actuator control back to ArduPlane using the MUX switch before landing.

## V. Experimental results

As a result of this work, we have been able to perform a series of successful outdoor flight experiments to evaluate the algorithms described in Section II. A detailed description of the specific experiments and the results obtained will appear in separate manuscripts. See [20] and [19] for DRL and NMPC results, respectively.

In this section, we demonstrate the efficacy of our experimental platform by presenting some of our results, with a special focus on the switching mechanism. In particular, we show initial results for GAC, based on [21], and look at one of our earliest attempts at closed-loop flight using DRL.

### A. Geometric Attitude Control (GAC)

See Fig. 4 (a) for an early attempt during the tuning process of GAC. Initially, after takeoff, the pilot uses ArduPlane's fly-by-wire A (FBWA) mode to control the UAV's roll and pitch angles using manual stick input. At 720 s (marked by the vertical line), the pilot switches the MUX to engage closed-loop operation of the GAC. During this initial test, integral action was disabled, explaining the increased offset visible after the switch. The switch between the PWM outputs from the Cube and those from the SBC is seamless, and any potential communication delays in the hardware architecture do not seem to have a significant effect on the controller.

Fig. 4 (b) shows the performance of the GAC controller after some further tuning. The steady-state offset has been removed and the test shows good performance through a series of repeated step responses in both roll and pitch channels. The parallel pipeline, including a solid backup system and in-air switching between the two, enabled a safe and comfortable tuning process. A more thorough evaluation of the GAC experiments is set to appear in a separate article.

### B. Deep Reinforcement Learning (DRL)

Fig. 5 shows one of the first attempts during flight testing of the DRL controller. Again, after takeoff and checking that all systems behave as expected, the pilot gave control over the servos to the DRL controller by flipping the MUX switch. The closed-loop pitch response was highly oscillatory (unstable), causing the pilot to switch to manual control of the UAV. After tweaking a few parameters to scale down the magnitude of the controller outputs (while the UAV was still flying), we were able to reduce the oscillations. Fig. 5 shows a marginally stable response where the pilot was comfortable leaving the experimental algorithm in control. After some design iterations with rapid test cycles, we were able to obtain results comparable to a well-tuned benchmark PID controller. See [20] for details.

These examples illustrate how our system lowers the threshold for high-risk tests of experimental low-level algorithms. The main takeaways are (a) the switch does not interfere when our algorithm works, and in case it doesn't, our system saves the day (although we did crash a few times due to human errors), and (b) the serial communication between the Cube and the SBC provide state estimates with a low enough latency to be satisfactory for our purposes.

## VI. Lessons learned

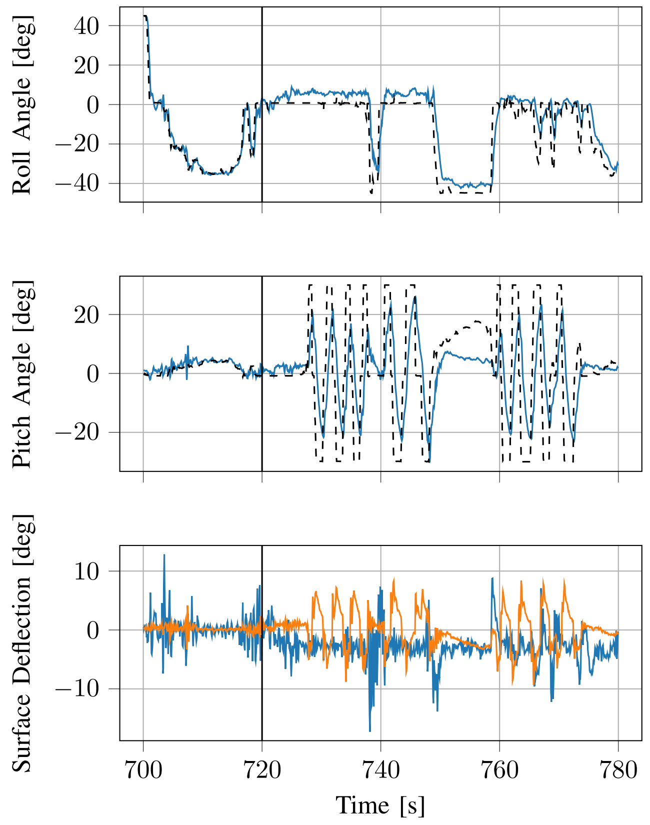
In this section, we discuss some lessons learned based on the experience gained in this project.

Previous work at our lab has focused on different aspects of state estimation for autonomous vehicles, including GNSS-aided inertial navigation [35] and estimation of airflow angles [36], [37]. However, in our work, the focus has been on control. To provide the SBC with the state estimates needed to run our control algorithms, we chose a pragmatic solution instead of spending time on implementing a custom solution: to utilize the already existing navigation solution provided by ArduPlane and send this to the SBC using the MAVLink protocol. Initially, we were worried that the latency of this link could cause problems with closed-loop operation of our algorithms, or that the achieved update frequency would be too low. However, this has not caused any issues to this day. Instead of spending too much time and resources on a perfect solution, we choose something simple and stick with it until something better is needed.

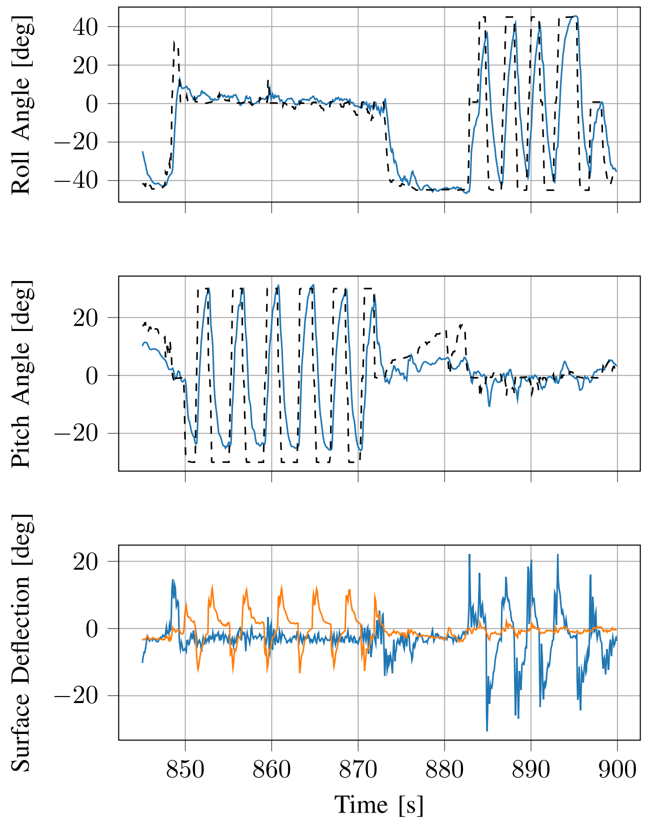
An unexpected problem that was particularly time-consuming during the preparations of the experiments was the electromagnetic interference between components of the flight stack. Being unaware of the fact that this is the source of error makes it hard to debug components of the system. For example on our test platform, an earlier controller board interfered with the GNSS antenna, ultimately causing the EKF to diverge in a non-deterministic way. It took a considerable amount of time until we discovered a correlation to the distance between the GNSS antenna and the controller board. Early integration tests with alternatives for each hardware component is our lesson learned in this case, assuming that an engineering team with expertise in electromagnetic interference is not part of the crew.

Before attempting the first flight experiments of our model-based designs, we lingered for too long because initial model validation efforts showed some discrepancies with flight data. Looking back, we would have rather performed more early flight tests before iteratively improving the model [26], [38]. The NMPC for instance, has proved to be robust to modeling errors. Do not underestimate the robustness of feedback control.

The pilot is an essential part of the crew and can be a good resource when doing experimental work. Supporting with

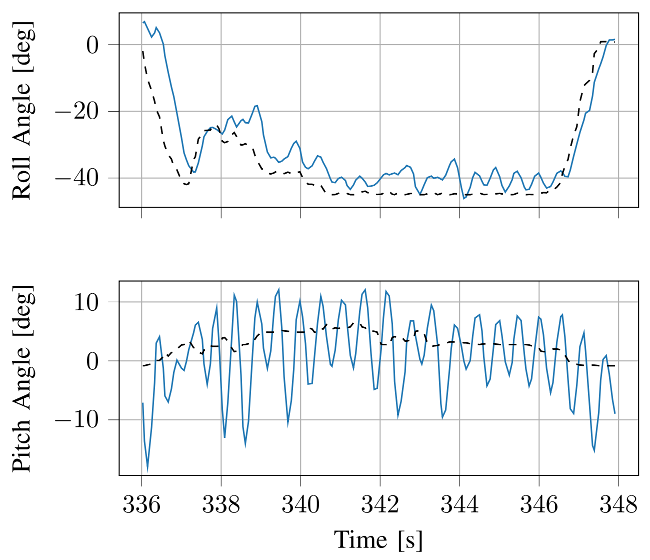


(a) Initial switch to GAC.

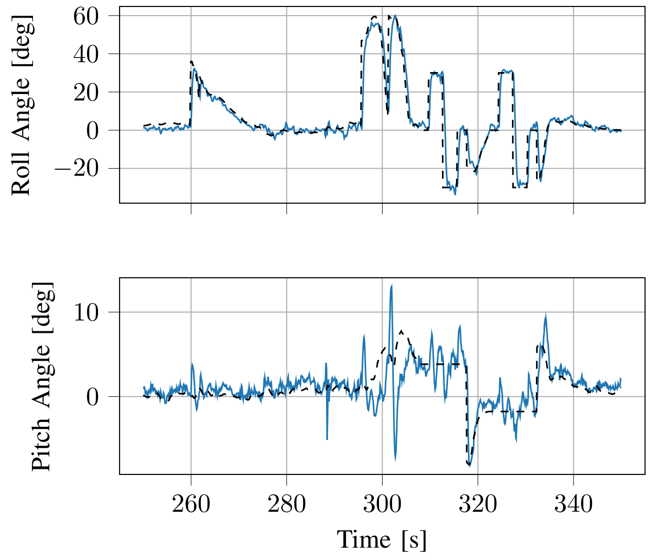


(b) Performance after a short tuning procedure.

Fig. 4: Experimental results with the GAC in control in the first trial (a) and after more tuning (b). The plots show the tracking response for roll and pitch references (dashed, black). The bottom subplots depict the virtual deflections of the aileron (blue) and the elevator (orange).



(a) Early DRL experiment.



(b) Performance after a few design iterations.

Fig. 5: Experimental results with DRL showing (a) oscillatory attitude response (blue) vs reference angles (dashed, black) in initial flight experiments, and (b) after a few design iterations the controller achieves good tracking performance. See [20] for details.

experience from the field and practical aspects of the UAV, it is a good idea to keep the pilot in close communication and discuss ideas early to get additional insights into the feasibility of the case study. Create an open environment where ideas can be freely discussed.

Having a working experimental platform is a solid foundation for rapid prototyping of practical control designs. However, it has taken some time to get there. Performing flight experiments with fixed-wing UAVs is an outdoor sport. Weather conditions, travel time to the airfield, and the size of our test crew are all elements that make this a substantial undertaking.

## VII. Conclusions

We provided a detailed description of a flight-stack architecture and experiment protocols to test advanced nonlinear control algorithms. The hardware architecture consists of off-the-shelf components that researchers can integrate into existing flight platforms with minimum effort. We demonstrate the practical use with examples of low-level motion control algorithms from our lab and conclude with lessons learned to help other researchers avoid pitfalls that we discovered while building our test infrastructure.

## Acknowledgment

Many people have contributed to the work presented in this paper, including Professor Thor I. Fossen, Martin Sollie, Andreas Wenz, and several skilled master's students. Special thanks goes to pilot Pål Kvaløy for his crucial role in the project, especially the field experiments, and Eivind Bøhn for his contributions to the presented results on deep reinforcement learning attitude control.

## References

[1] "ArduPilot open-source autopilot software," https://ardupilot.org/, accessed: 2021-10-09.

[2] L. Meier, D. Honegger, and M. Pollefeys, "Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 6235–6240.

[3] S. H. Lane and R. F. Stengel, "Flight control design using nonlinear inverse dynamics," in *1986 American Control Conference*, 1986, pp. 587–596.

[4] J. Farrell, M. Sharma, and M. Polycarpou, "Backstepping-based flight control with adaptive function approximation," *Journal of Guidance, Control, and Dynamics*, vol. 28, no. 6, pp. 1089–1102, 2005. [Online]. Available: https://doi.org/10.2514/1.13030

[5] L. Sonneveldt, Q. P. Chu, and J. A. Mulder, "Nonlinear flight control design using constrained adaptive backstepping," *Journal of Guidance, Control, and Dynamics*, vol. 30, no. 2, pp. 322–336, 2007. [Online]. Available: https://doi.org/10.2514/1.25834

[6] E. Lavretsky and S. Wise, *Robust and Adaptive Control: With Aerospace Applications*. London: Springer, 2013.

[7] A. Hovenburg, T. A. Johansen, and R. Storvold, "Mission performance trade-offs of battery-powered suas," in *Int. Conf. Unmanned Aircraft Systems, Miami*, 2017.

[8] P. Oettershagen, A. Melzer, S. Leutenegger, K. Alexis, and R. Siegwart, "Explicit model predictive control and l1-navigation strategies for fixed-wing uav path tracking," in *22nd Mediterranean Conference on Control and Automation*, June 2014, pp. 1159–1165.

[9] J.-M. Kai, A. Anglade, T. Hamel, and C. Samson, "Design and experimental validation of a new guidance and flight control system for scale-model airplanes," in *2018 IEEE Conference on Decision and Control (CDC)*, 2018, pp. 4270–4276.

[10] M. Basescu and J. Moore, "Direct nmpc for post-stall motion planning with fixed-wing uavs," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 9592–9598.

[11] T. Stastny and R. Siegwart, "Nonlinear Model Predictive Guidance for Fixed-wing UAVs Using Identified Control Augmented Dynamics," *2018 International Conference on Unmanned Aircraft Systems, ICUAS 2018*, pp. 432–442, 2018.

[12] M. Kamel, T. Stastny, K. Alexis, and R. Siegwart, "Model predictive control for trajectory tracking of unmanned aerial vehicles using robot operating system," in *Robot Operating System (ROS)*. Springer, 2017, pp. 3–39.

[13] J. Yang, C. Liu, M. Coombes, Y. Yan, and W.-H. Chen, "Optimal path following for small fixed-wing uavs under wind disturbances," *IEEE Transactions on Control Systems Technology*, vol. 29, no. 3, pp. 996–1008, May 2021.

[14] M. Neunert, C. De Crousaz, F. Furrer, M. Kamel, F. Farshidian, R. Siegwart, and J. Buchli, "Fast nonlinear Model Predictive Control for unified trajectory optimization and tracking," *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2016-June, pp. 1398–1404, 2016.

[15] A. Zanelli, "Nonlinear Model Predictive Control of a Humand-sized Quadrotor," *2018 European Control Conference (ECC)*, pp. 1542–1547, 2018. [Online]. Available: https://upcommons.upc.edu/handle/2117/98503

[16] C.-H. Pi, Y.-W. Dai, K.-C. Hu, and S. Cheng, "General purpose low-level reinforcement learning control for multi-axis rotor aerial vehicles," *Sensors*, vol. 21, no. 13, 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/13/4560

[17] N. O. Lambert, D. S. Drew, J. Yaconelli, S. Levine, R. Calandra, and K. S. J. Pister, "Low-level control of a quadrotor with deep model-based reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4224–4230, oct 2019.

[18] W. Koch, R. Mancuso, and A. Bestavros, "Neuroflight: Next Generation Flight Control Firmware," *arXiv:1901.06553 [cs]*, Sep. 2019, arXiv: 1901.06553. [Online]. Available: http://arxiv.org/abs/1901.06553

[19] D. Reinhardt, E. M. Coates, and T. A. Johansen, "Low-level Nonlinear Model Predictive Attitude and Speed Control of Fixed-Wing Unmanned Aerial Vehicles," *Control Engineering Practice*, submitted.

[20] E. Bøhn, E. M. Coates, D. Reinhardt, and T. A. Johansen, "Data-efficient deep reinforcement learning for attitude control of fixed-wing uavs: Field experiments," *arXiv preprint arXiv:2111.04153*, 2021.

[21] E. M. Coates and T. I. Fossen, "Geometric reduced-attitude control of fixed-wing uavs," *Applied Sciences*, vol. 11, no. 7, 2021. [Online]. Available: https://www.mdpi.com/2076-3417/11/7/3147

[22] K. Gryte, R. Hann, M. Alam, J. Roháč, T. A. Johansen, and T. I. Fossen, "Aerodynamic modeling of the skywalker x8 fixed-wing unmanned aerial vehicle," in *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2018, pp. 826–835.

[23] E. M. Coates, A. Wenz, K. Gryte, and T. A. Johansen, "Propulsion System Modeling for Small Fixed-Wing UAVs," in *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2019.

[24] K. Gryte, "Precision control of fixed-wing UAV and robust navigation in GNSS-denied environments," Ph.D. dissertation, Norwegian University of Science and Technology (NTNU), June 2020. [Online]. Available: https://hdl.handle.net/11250/2657113

[25] D. Reinhardt and T. A. Johansen, "Control of fixed-wing uav attitude and speed based on embedded nonlinear model predictive control," *IFAC-PapersOnLine*, vol. 54, no. 6, pp. 91–98, 2021, 7th IFAC Conference on Nonlinear Model Predictive Control NMPC 2021.

[26] D. Reinhardt, K. Gryte, and T. A. Johansen, "Modeling of the skywalker x8 fixed-wing uav: Flight tests and system identification," in *2022 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2022.

[27] L. Grüne and J. Pannek, *Nonlinear Model Predictive Control*. Cham: Springer International Publishing, 2017, pp. 45–69.

[28] E. Bøhn, E. M. Coates, S. Moe, and T. A. Johansen, "Deep reinforcement learning attitude control of fixed-wing uavs using proximal

policy optimization," in *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2019, pp. 523–533.

[29] E. M. Coates, D. Reinhardt, and T. I. Fossen, "Reduced-attitude control of fixed-wing unmanned aerial vehicles using geometric methods on the two-sphere," *IFAC World Congress 2020, Germany*, 2020.

[30] A. Zolich, T. A. Johansen, K. Cisek, and K. Klausen, "Unmanned aerial system architecture for maritime missions. design & hardware description," in *2015 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS)*, 2015, pp. 342–350.

[31] J. Pinto, P. S. Dias, R. Martins, J. Fortuna, E. Marques, and J. Sousa, "The lsts toolchain for networked vehicle systems," in *2013 MTS/IEEE OCEANS - Bergen*, 2013, pp. 1–9.

[32] R. Verschueren, G. Frison, D. Kouzoupis, N. van Duijkeren, A. Zanelli, B. Novoselnik, J. Frey, T. Albin, R. Quirynen, and M. Diehl, "acados: a modular open-source framework for fast embedded optimal control," *arXiv preprint*, 2019. [Online]. Available: https://arxiv.org/abs/1910.13753

[33] P. Ladosz, M. Coombes, J. Smith, and M. Hutchinson, *A Generic ROS Based System for Rapid Development and Testing of Algorithms for Autonomous Ground and Aerial Vehicles*. Cham: Springer International Publishing, 2019, pp. 113–153. [Online]. Available: https://doi.org/10.1007/978-3-319-91590-6_4

[34] T. A. Johansen, "Toward dependable embedded model predictive control," *IEEE Systems Journal*, vol. 11, no. 2, pp. 1208–1219, 2017.

[35] T. H. Bryne, J. M. Hansen, R. H. Rogne, N. Sokolova, T. I. Fossen, and T. A. Johansen, "Nonlinear observers for integrated insgnss navigation: Implementation aspects," *IEEE Control Systems Magazine*, vol. 37, no. 3, pp. 59–86, 2017.

[36] T. A. Johansen, A. Cristofaro, K. Sørensen, J. M. Hansen, and T. I. Fossen, "On estimation of wind velocity, angle-of-attack and sideslip angle of small uavs using standard sensors," in *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2015, pp. 510–519.

[37] A. Wenz and T. A. Johansen, "Moving horizon estimation of air data parameters for uavs," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 56, no. 3, pp. 2101–2121, 2020.

[38] D. Reinhardt, M. D. Pedersen, K. Gryte, and T. A. Johansen, "A symmetry calibration procedure to compensate for sensor-to-airframe misalignments in wind tunnel data," in *2022 Conference on Control Technologies and Applications (CCTA)*, 2022.