Agilicious: Open-Source and Open-Hardware Agile Quadrotor for Vision-Based Flight

PHILIPP FOEHN[†,*], ELIA KAUFMANN[†,*], ANGEL ROMERO, ROBERT PENICKA, SIHAO SUN, LEONARD BAUERSFELD, THOMAS LAENGLE, GIOVANNI CIOFFI, YUNLONG SONG, ANTONIO LOQUERCIO, AND DAVIDE SCARAMUZZA

All authors are with the Robotics and Perception Group, UZH, Zurich, Switzerland.
[†] *These authors contributed equally to this work.*
[*] *Corresponding authors: foehn@ifi.uzh.ch, ekaufmann@ifi.uzh.ch*

This is the accepted version of Science Robotics Vol. 7, Issue 67
DOI: 10.1126/scirobotics.abl6259 (2022)

Autonomous, agile quadrotor flight raises fundamental challenges for robotics research in terms of perception, planning, learning, and control. A versatile and standardized platform is needed to accelerate research and let practitioners focus on the core problems. To this end, we present Agilicious, a co-designed hardware and software framework tailored to autonomous, agile quadrotor flight. It is completely open-source and open-hardware and supports both model-based and neural-network–based controllers. Also, it provides high thrust-to-weight and torque-to-inertia ratios for agility, onboard vision sensors, GPU-accelerated compute hardware for real-time perception and neural-network inference, a real-time flight controller, and a versatile software stack. In contrast to existing frameworks, Agilicious offers a unique combination of flexible software stack and high-performance hardware. We compare Agilicious with prior works and demonstrate it on different agile tasks, using both model-based and neural-network–based controllers. Our demonstrators include trajectory tracking at up to 5 g and 70 km/h in a motion-capture system, and vision-based acrobatic flight and obstacle avoidance in both structured and unstructured environments using solely onboard perception. Finally, we demonstrate its use for hardware-in-the-loop simulation in virtual-reality environments. Thanks to its versatility, we believe that Agilicious supports the next generation of scientific and industrial quadrotor research.

CODE AND MULTIMEDIA MATERIAL

Code and data can be found at https://agilicious.dev. A video of the experiments can be found at https://youtu.be/fNYxPLyJ5YY.

1. INTRODUCTION

Quadrotors are extremely agile vehicles. Exploiting their agility in combination with full autonomy is crucial for time-critical missions, such as search and rescue, aerial delivery, and even flying cars. For this reason, over the past decade, research on autonomous, agile quadrotor flight has continually pushed platforms to higher levels of speed and agility [1–10].

To further advance the field, several competitions have been organized—such as the autonomous drone racing series at the recent IROS and NeurIPS conferences [3, 11–13] and the AlphaPilot challenge [6, 14]—with the goal to develop autonomous systems that will eventually outperform expert human pilots. Million-dollar projects, such as AgileFlight [15] and Fast Lightweight Autonomy (FLA) [4], have also been funded by the European Research Council and the United States government, respectively, to further push research. Agile flight comes with ever-increasing engineering challenges since performing faster maneuvers with an autonomous system requires more capable algorithms, specialized hardware, and proficiency in system integration. As a result, only a small number of research groups have undertaken the significant overhead of hardware and software engineering, and have developed the expertise and resources to design quadrotor platforms that fulfill the requirements on weight, sensing, and computational budget necessary for autonomous agile flight. This work aims to bridge this gap through an open-source agile flight platform, enabling everyone to work on agile autonomy with minimal engineering overhead.

The platforms and software stacks developed by research groups [2, 4, 16–21] vary strongly in their choice of hardware and software tools. This is expected, as optimizing a robot with respect to different tasks based on individual experience in a closed-source research environment leads to a fragmentation of the research community. For example, even though many research groups use the Robot Operating System middleware to accelerate development, publications are often difficult to reproduce or verify since they build on a plethora of previous implementations of the authoring research group. In the worst case, building on an imperfect or even faulty closed-source foundation can lead to wrong or non-reproducible conclusions, slowing down research progress. To break this vicious cycle and to democratize research on fast autonomous flight, the robotics community needs an open-source and open-hardware quadrotor platform that provides the versatility and performance needed for a wide range of agile flight tasks. Such an

open and agile platform does not yet exist, which is why we present Agilicious, an open-source and open-hardware agile quadrotor flight stack summarized in Figure 1.

To reach the goal of creating an agile, autonomous, and versatile quadrotor research platform, two main design requirements must be met by the quadrotor: it must carry the required compute hardware needed for autonomous operation, and it must be capable of agile flight.

To meet the first requirement on computing resources needed for true autonomy, a quadrotor should carry sufficient compute capability to concurrently run estimation, planning, and control algorithms onboard. With the emergence of learning-based methods, also efficient hardware acceleration for neural network inference is required. To enable agile flight, the platform must deliver an adequate thrust-to-weight ratio and torque-to-inertia ratio. While the thrust-to-weight ratio can often be enhanced using more powerful motors, which in turn require larger propellers and thus a larger size of the platform. However, the torque-to-inertia ratio typically decreases with higher weight and size, since the moment of inertia increases quadratic with the size, and linearly with the weight. As a result, it is desirable to design a lightweight and small platform [25, 26] to maximize agility (i.e. maximize both thrust-to-weight and torque-to-inertia). Therefore the platform should meet the best trade-off, since maximizing compute resources competes against maximizing the flight performance.

Apart from hardware design considerations, a quadrotor research platform needs to provide the software framework for flexible usage and reproducible research. This entails the abstraction of hardware interfaces and a general co-design of software and hardware necessary to exploit the platform's full potential. Such co-design must account for the capabilities and limitations of each system component, such as the complementary real-time capabilities of common operating systems and embedded systems, communication latencies and bandwidths, system dynamics bandwidth limitations, and efficient usage of hardware accelerators. In addition to optimally using hardware resources, the software should be built in a modular fashion to enable rapid prototyping through a simple exchange of components, both in simulation and real-world applications. This modularity enables researchers to test and experiment with their research code, without the requirement to develop an entire flight stack, accelerating time to development and facilitating reproducibility of results. Finally, the software stack should run on a broad set of computing boards, be efficient, easy to transfer and adapt by having minimal dependencies and provide known interfaces, such as the widely-used Robot Operating System (ROS).

The complex set of constraints and design objectives is difficult to meet. There exists a variety of previously published open-source research platforms, which, while well designed for low-agility tasks, could only satisfy a subset of the aforementioned hardware and software constraints. In the following section, we list and analyze prominent examples such as the FLA platform [4], the MRS quadrotor [20], the ASL-Flight [17], the MIT-Quad [27], the GRASP-Quad [2], or our previous work [18].

The FLA platform [4] relies on many sensors, including Lidars and laser-range finders in conjunction with a powerful onboard computer. While this platform can easily meet autonomous flight computation and sensing requirements, it does not allow to perform agile flight beyond 2.4g of thrust, limiting the flight envelope to near-hover flight. The MRS platform [20] provides an accompanying software stack and features a variety of sensors. Even though this hardware and software solution allows fully autonomous flight, the actuation renders the system not agile with a maximum thrust-to-weight of 2.5. The ASL-Flight [17] is built on the DJI Matrice 100 platform and features an Intel NUC as the main compute resource. Similarly to the MRS platform, the ASL-Flight has very limited agility due to its weight being on the edge of the platform's takeoff capability. The comparably smaller GRASP-Quad proposed in [2] operates with only onboard inertial measurement unit (IMU) and monocular camera while having a weight of only 250 g. Nevertheless, the Qualcomm Snapdragon board installed on this platform lacks computational power and also the actuation constrains the maximal accelerations below 1.5g. Motivated by drone racing, the MIT-Quad [27] reported accelerations of up to 2.1g while it was further equipped with NVIDIA Jetson TX2 in [28], however, it does not reach the agility of Agilicious and contains proprietary electronics. Finally, the quadrotor proposed in [18] is a research platform designed explicitly for agile flight. Although the quadrotor featured a high thrust-to-weight ratio of up to 4, its compute resources are very limited, prohibiting truly autonomous operation. All these platforms are optimized for either relatively heavy sensor setups or for agile flight in non-autonomous settings. While the former platforms lack the required actuation power to push the state of the art in autonomous agile flight, the latter have insufficient compute resources to achieve true autonomy.

Finally, several mentioned platforms rely on either Pixhawk-PX4 [29], the Parrot [30] or DJI [31] low-level controllers, which are mostly treated as blackboxes. This, together with the proprietary nature of the DJI systems, limits control over the low-level flight characteristics, which not only limits interpretability of results, but also negatively impacts agility. Full control over the complete pipeline is necessary to truly understand aerodynamic and high-frequency effects, model and control them, and exploit the platform to its full potential.

Apart from platforms mainly developed by research labs, several quadrotor designs are proposed by industry (Skydio [32], DJI [31], Parrot [30]) and open-source projects (PX4 [29], Paparazzi [21], Crazyflie [33]). While Skydio [32] and DJI [31] both develop platforms featuring a high level of autonomy, they do not support interfacing with custom code and therefore are of limited value for research and development purposes. Parrot [30] provides a set of quadrotor platforms tailored for inspection and surveillance tasks that are accompanied by limited software development kits that allow researchers to program custom flight missions. In contrast, PX4 [29] provides an entire ecosystem of open-source software and hardware as well as simulation. While these features are extremely valuable especially for low-speed flight, both cross-platform hardware and software are not suited to push the quadrotor to agile maneuvers. Similarly, Paparazzi [21] is an open-source project for drones, which supports various hardware platforms. However, the supported autopilots have very limited onboard compute capability, rendering them unsuited for agile autonomous flight. The Crazyflie [33] is an extremely lightweight quadrotor platform with a takeoff weight of only 27 g. The minimal hardware setup leaves no margin for additional sensing or computation, prohibiting any non-trivial navigation task.

To address the requirements of agile flight, the shortcomings of existing works, and to enable the research community to progress fast towards agile flight, we present an open-source and open-hardware quadrotor platform for agile flight at more than 5g acceleration while providing substantial onboard compute and versatile software. The hardware design leverages recent advances in motor, battery, and frame design initiated by the first-person-view (FPV) racing community. The design objectives resulted in creating a lightweight 750 g platform with maximal speed of 131 km/h. This high-performance drone hardware is combined with a powerful onboard computer that features an integrated GPU, enabling complex neural network architectures to run at high frequency while concurrently running optimization-based control algorithms at low latency onboard the drone. The most important features of the Agilicious framework are summarized and compared with relevant research and industrial platforms in Figure 2. A qualitative comparison of mutually contradicting onboard computational power and agility is presented in Figure 2.

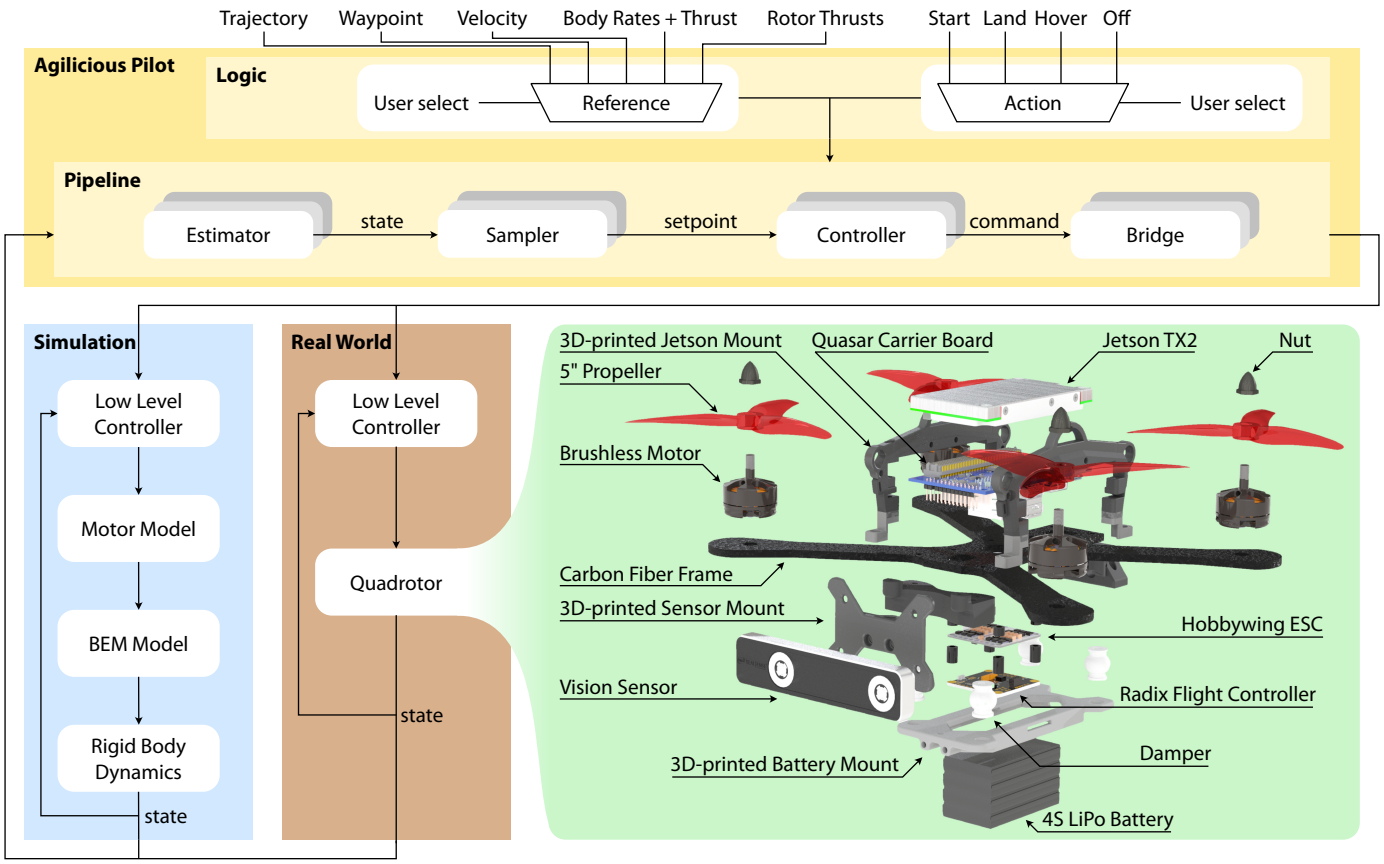


Fig. 1. The Agilicious software and hardware quadrotor platform are tailored for agile flight while featuring powerful onboard compute capabilities through an NVIDIA Jetson TX2. The versatile sensor mount allows for rapid prototyping with a wide set of monocular or stereo camera sensors. As a key feature, the software of Agilicious is built in a modular fashion, allowing rapid software prototyping in simulation and seamless transition to real-world experiments. The Agilicious Pilot encapsulates all logic required for agile flight, while exposing a rich set of interfaces to the user, from high-level pose commands to direct motor commands. The software stack can be used in conjunction with a custom modular simulator which supports highly accurate aerodynamics based on blade-element momentum theory [22], or with RotorS [23], hardware-in-the-loop, and rendering engines such as Flightmare [24]. Deployment on the physical platform only requires selecting a different bridge and a sensor-compatible estimator.

In co-design with the hardware, we complete the drone design with a modular and versatile software stack, called Agilicious. It provides a software architecture that allows to easily transfer algorithms from prototyping in simulation to real-world deployment in instrumented experiment setups, and even pure onboard-sensing applications in unknown and unstructured environments.

This modularity is key for fast development and high-quality research, since it allows to quickly substitute existing components with new prototypes and enables all software components to be used in standalone testing applications, experiments, benchmarks, or even completely different applications.

The hardware and software presented in this work have been developed, tested, and refined throughout a series of research publications [3, 9, 10, 34–38]. All these publications share the ambition to push autonomous drones to their physical limits. The experiments, performed in a diverse set of environments demonstrate the versatility of Agilicious by deploying different estimation, control, and planning strategies on a single physical platform. The flexibility to easily combine and replace both hard- and software components in the flight stack while operating on a standardized platform facilitates testing new algorithms and accelerates research on fast autonomous flight.

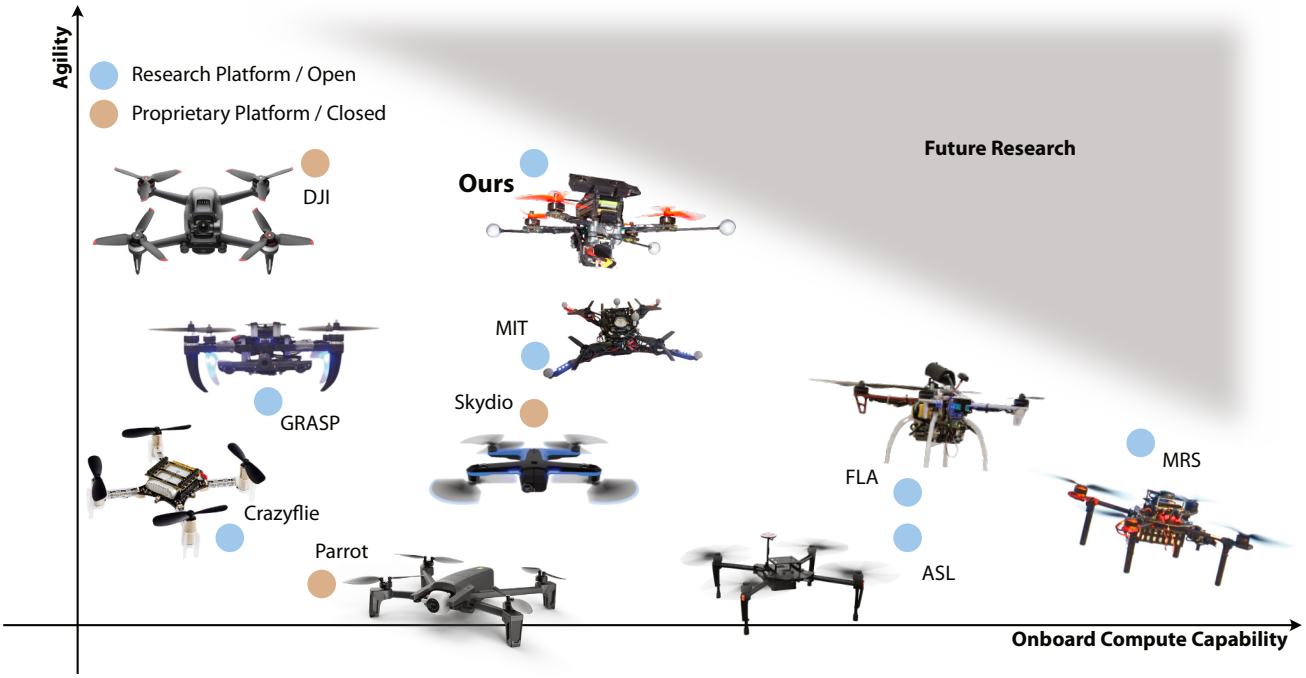
2. RESULTS

Our experiments, conducted in simulation and in the real world, demonstrate that Agilicious can be used to perform cutting-edge research in the fields of agile quadrotor control, quadrotor trajectory planning, and learning-based quadrotor research. We evaluate the capabilities of the Agilicious software and hardware stack in a large set of experiments that investigate trajectory tracking performance, latency of the pipeline, combinations of Agilicious with a set of commercially or openly available vision-based state estimators. Finally, we present two demonstrators of recent research projects that build on Agilicious.

A. Trajectory Tracking Performance

In this section, we demonstrate the tracking performance of our platform by flying an aggressive time-optimal trajectory in a drone racing scenario. Additionally, to benchmark our planning and control algorithms, we compete against a world-class drone racing pilot FPV pilot, reported in [10]. As illustrated in Figure 3, our drone racing track consists of seven gates that need to be traversed in a pre-defined order as fast as possible. The trajectory used for this evaluation reaches speeds of 60km/h and accelerations of 4 g.

Flying through gates at such high speed requires precise state estimates, which is still an open challenge using vision-based state es-



framework	open-source	simulation	onboard computer	low-level controller	CPU mark (higher is better)	GPU	maximum speed (0-100 km/h time)	thrust/weight
PX4 [29]	SW and HW	✓	✗	custom open source	-	✗	-	-
Paparazzi [21]	SW and HW	✓	✗	custom open source	-	✗	-	-
DJI [31]	-	✗	✗	proprietary	-	✗	140 km/h (2.0 s)	≈ 4.434
Skydio [32]	-	✗	✗	proprietary	-	✗	58 km/h	-
Parrot [30]	SW	✗	✗	proprietary	-	✗	55 km/h	-
Crazyflie [33]	SW and HW	✓	✗	custom open source	-	✗	-	≈ 2.26
FLA-Quad [4]	SW and HW	✗	✓	PX4	3,383	✗	-	≈ 2.38
GRASP-Quad [2]	-	✗	✓	custom	625	✗	-	≈ 1.80
MIT-Quad [28]	-	✗	✓	custom	1,343	✓	-	≈ 2.33
ASL-Flight [17]	SW and HW	✗	✓	DJI	3,383	✗	-	≈ 2.32
RPG-Quad [18]	SW and HW	✓	✓	Betaflight	633	✗	-	≈ 4.00
MRS UAV [20]	SW and HW	✓	✓	PX4	8,846	✗	-	≈ 2.50
Agilicious (Ours)	SW and HW	✓	✓	custom open source	1,343	✓	131 km/h (1.01 s)	≈ 5.00

Fig. 2. A comparison of different available consumer and research platforms with respect to available onboard compute capability and agility. The platforms are compared based on their openness to the community, support of simulation and onboard computation, used low-level controller, CPU power (reported according to publicly available benchmarks https://www.cpubenchmark.net and corresponding to the speed of solving a set of benchmark algorithms that represent a generic program), and the availability of onboard general-purpose GPU. The agility of the platforms is expressed in the terms of thrust-to-weight ratio; however, we also report the maximal velocity as an agility indicator due to limited information about the commercial platforms. The PX4 [29] and the Paparazzi [21] are rather low-level autopilot frameworks without high-level computation capability, but they can be integrated in other high-level frameworks [4, 20]. The open-source frameworks FLA [4], ASL[17], and MRS [20] have relatively large weight and low agility. The DJI [31], Skydio [32], and Parrot [30] are closed-source commercial products that are not intended for research purposes. The Crazyflie [33] does not allow for sufficient onboard compute or sensing, while the MIT [28] and GRASP [2] platforms are not available open-source. Finally, our proposed Agilicious framework provides agile flight performance, onboard GPU-accelerated compute capabilities, as well as open-source and open-hardware availability.

timators [39]. For this reason, we conduct these experiments in an instrumented flight volume of $30 \times 30 \times 8$ m (7200 m^3), equipped with 36 VICON cameras that provide precise pose measurements at 400 Hz. However, even when provided with precise state estimation, accurately tracking such aggressive trajectories poses considerable challenges with respect to the controller design, which usually requires several iterations of algorithm development and substantial tuning effort. The proposed Agilicious flight stack allows us to easily design, test, and deploy different control methods by first verifying them in simulation and then fine-tuning them in the real-world. The transition from simulation to real-world deployment requires no source-code changes or adaptions, which reduces the risk of crashing expensive hardware, and is one of the major features of Agilicious accelerating rapid-prototyping. Figure 3 includes a simulated flight that shows similar characteristics and error statistics compared to the real-world flights described next.

We evaluate three different system and control approaches including onboard computation with an off-the-shelf BetaFlight [40] flight controller, our custom open-source agiNuttx controller, and an offboard-control scenario. These three system configurations represent various

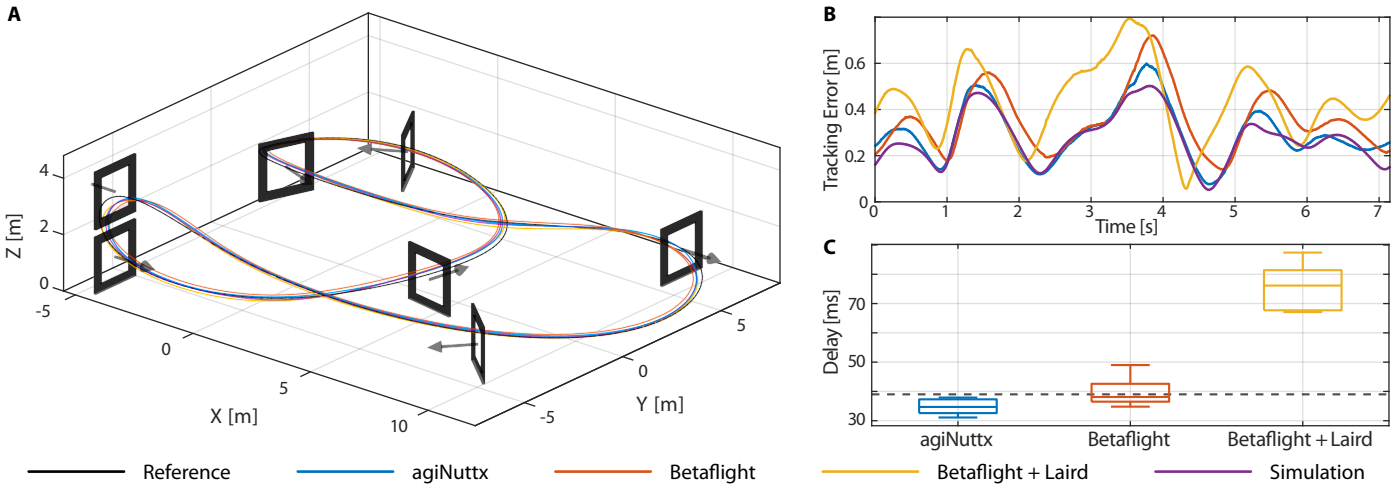


Fig. 3. An agile trajectory with speeds up to 60km/h and an acceleration of 4g, executed in an indoor instrumented flight volume. We compare multiple different drone configurations, including our own low-level flight controller software agiNuttx, an off-the-shelf BetaFlight controller, the BetaFlight controller together with offboard computing and remote control through a Laird wireless transmitter, and our included simulation. Figure A depicts an overview of the flown trajectory. Figure B shows the tracking errors along a single lap over all three configurations. Our provided agiNuttx achieves the best tracking performance, followed by BetaFlight combined with onboard computation. In contrast, offloading computation from the drone and controlling it remotely significantly impacts performance. This is due to the massively increased latency, depicted in Figure C, where, for reference, the motor time constant of 39.1ms is marked as a dashed line (- - -). Additionally, in Figure B it is visible that the simulation exhibits very similar error characteristics, thanks to our accurate aerodynamic modelling.

use cases of Agilicious, such as running state-of-the-art single-rotor control onboard the drone using our agiNuttx described in Sec. B, or simple remote control by executing Agilicious on a desktop computer and forwarding the commands to the drone. All configurations use the motion capture state estimate and our single-rotor model-predictive control (MPC) described in Section D.1 [41] as high-level controller. We use the single-rotor thrust formulation to correctly account for actuation limits, but use bodyrates and collective thrust as command modality.

The first configuration runs completely onboard the drone with an additional low-level controller in the form of incremental non-linear dynamic inversion (INDI) as described in Section D.1 [41]. It uses the MPC's output to compute refined single-rotor thrust commands using INDI, to reduce the sensitivity to model inaccuracies. These single-rotor commands are executed using our agiNuttx flight controller with closed-loop motor speed tracking.

The second configuration also runs onboard and directly forwards the bodyrate and collective thrust command from the MPC to a BetaFlight [40] controller. This represents the most simplistic system which does not require flashing the flight controller and is compatible with a wide range of readily available off-the-shelf hobbyist drone components. However, in this configuration, the user does not get any IMU or motor speed feedback, as those are not streamed by BetaFlight.

The third configuration is equal to the second configuration with the difference that the Agilicious flight stack runs offboard on a desktop or laptop computer. The bodyrate and collective thrust commands from the MPC are streamed to the drone using a serial wireless link implemented through LAIRD [42]. This configuration allows to run computationally demanding algorithms, such as GPU-accelerated neural networks, with minimal modifications. However, due to the additional wireless command transmission, there is a higher latency which can potentially degrade the control performance.

Finally, the Agilicious simulation is executed using the same setup as the first configuration. It uses accurate models for the quadrotor and motor dynamics, as well as a blade-element momentum theory aerodynamic model as described in [22].

Figure 3A,B depict the results of these trajectory tracking experiments. Our first proposed configuration (i.e. with onboard computation and the custom agiNuttx flight controller) achieves the best overall tracking performance with the lowest average positional root-mean-square error (RMSE) of just 0.322m at up to 60 km/h and 4 g. Next up is the second configuration with BetaFlight, still achieving less than 0.385m average positional RMSE. Finally, the third configuration with offboard control exhibits higher latency, leading to an increased average positional RMSE of 0.474m. As can be seen, our simulation closely matches the performance observed in real world in the first configuration. The simulated tracking results in slightly lower errors, 0.320m RMSE, since even the state-of-the-art aerodynamic models [22] fail to reproduce the highly non-linear and chaotic real aerodynamics. This simulation accuracy allows a seamless transition from simulation prototyping to real-world verification, and is one of the prominent advantages of Agilicious.

Additional experiments motivating the choice of MPC as outer-loop controller and its combination with INDI can be found in [41], details on the planning of the time-optimal reference trajectory are elaborated on in [10], and additional extensions to the provided MPC for fast flight are in [45] and for rotor-failure MPC in [46]. These related publications also showcase performance at even higher speeds of up to 70 km/h and accelerations reaching 5 g. The following section gives further insights into the latency of the three configurations tested herein, including on- and offboard control architecture, as well as BetaFlight and agiNuttx flight controllers.

B. Control Latency

All real systems with finite resources suffer from communication and computation delays, while dynamic systems and even filters can introduce additional latency and bandwidth limitations. Analyzing and minimizing these delays is fundamental for the performance in any control task, especially when tracking agile and fast trajectories in the presence of model mismatch, disturbances and actuator limitations. In this section we conduct a series of experiments that aim to analyse and determine the control latency, from command to actuation, of the

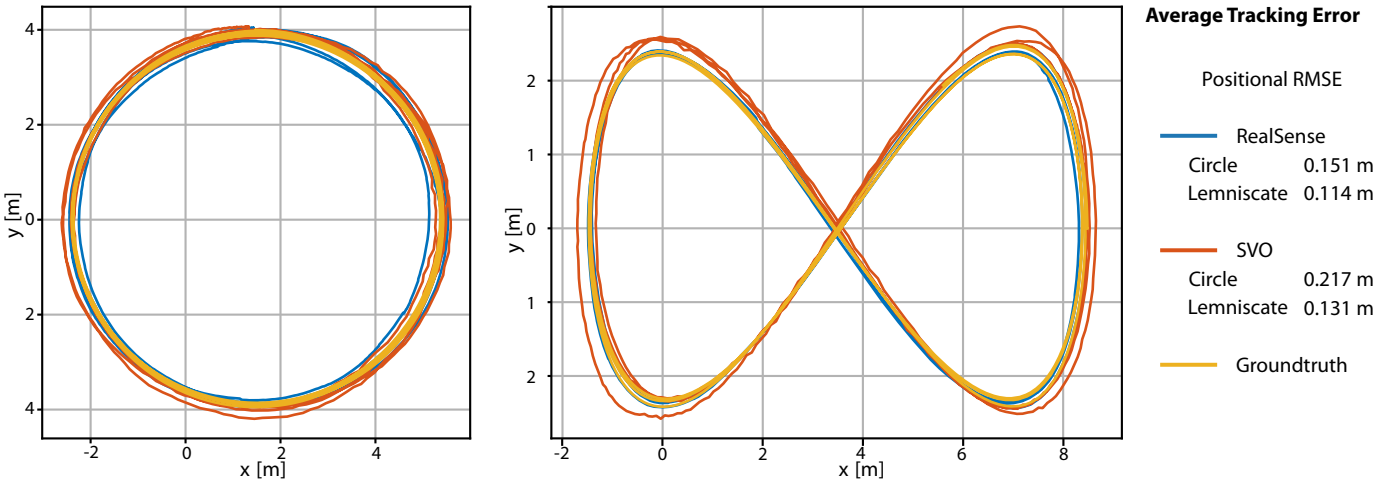


Fig. 4. A comparison of two different visual-inertial odometry solutions. The first solution consists of the Intel RealSense T265, an off-the-shelf sensor featuring a stereo camera, an IMU, and an integrated visual-inertial odometry (VIO) pipeline running on the integrated compute hardware. The second solution consists of a monocular camera, the IMU of the onboard flight controller, and SVO [43]. The estimates of both solutions are compared against motion-capture ground truth using [44] on a circle and a Lemniscate trajectory, flown using Agilicious in an indoor environment. Both systems show accurate tracking performance and could be used as cost-effective drop-in replacement for motion capture systems and enable deployment in the wild. While the Intel RealSense T265 is a convenient off-the-shelf option, using other cameras in combination with the onboard flight controller IMU and an open-source or custom visual-inertial odometry (VIO) pipeline enables tailored solutions and research-oriented data access.

proposed architecture for the three different choices of low level configurations: our agiNuttx, BetaFlight, and BetaFlight with offboard control.

For this experiment, the quadrotor has been mounted on a load cell (ATI Mini 40 SI-20-1 [47]) measuring the force and moment acting on the platform. To measure the latency, a collective thrust step command of 12N is sent to the corresponding low level controller, while measuring the exerted force on the drone. These force sensor measurements are time-synchronized with the collective thrust commands, and fitted through a first-order system representing to motor dynamics. The measured delays are reported in Figure 3C as the difference between the time at which the high level controller sends the collective thrust command, and the time at which the measured force effectively starts changing. The results show that both agiNuttx has the lowest latency at 35ms, with BetaFlight slightly slower at 40.15 ms. A large delay can be observed when using offboard control and sending the commands via Laird connection to the drone, in which case the latency rises to more than 75ms. The impact of these latencies is also reflected in the tracking error in Figure 3A,B. To put the measured delays into perspective, the motor's time constant of 39.1ms, which dictates the actuator bandwidth limitations, is indicated in Figure 3C. Finally, Section D.1 gives some insight into the latencies introduced when using Agilicious together with Flightmare [24] in a hardware-in-the-loop setup.

C. Visual-Inertial State Estimation

Deploying agile quadrotors outside of instrumented environments requires access to onboard state estimation. There exist many different approaches including GPS, lidar, and vision-based solutions. However, for size and weight constrained aerial vehicles, visual-inertial odometry has proven to be the go-to solution because of the sensors' complementary measurement modalities, low cost, mechanical simplicity, and reusability for other purposes, such as depth estimation for obstacle avoidance.

The Agilicious platform provides a versatile sensor mount that is compatible with different sensors and can be easily adapted to fit custom sensor setups. In this work, two different visual-inertial odometry (VIO) solutions are evaluated: (i) the proprietary, off-the-shelf Intel RealSense T265 and (ii) a simple camera together with the onboard flight controller IMU and an open-source VIO pipeline in the form of SVO Pro [43, 48] with its sliding-window backend. While the RealSense T265 performs all computation on-chip and directly provides a state estimate via USB3.0, the alternative VIO solution uses the Jetson TX2 to run the VIO software and allows researchers to interface and modify the state estimation software. Specifically, for sensor setup (ii), a single Sony IMX-287 camera at 30Hz with a 165° diagonal field-of-view is used, combined with the IMU measurements of the flight controller at 500Hz, calibrated using the Kalibr toolbox [49].

To verify their usability, a direct comparison of both VIO solutions with respect to ground-truth is provided. Performance is evaluated based on the estimation error [44] obtained on two trajectories flown with Agilicious. The flown trajectories consist of a circle trajectory with radius of 4m at a speed of 5m/s, and a Lemniscate trajectory with an amplitude of 5m at a speed of up to 7m/s.

Figure 4 shows the performance of both VIO solutions in an xy-overview of the trajectories together with their absolute tracking error (ATE) RMSE. Both approaches perform well on both trajectories, with the Intel RealSense achieving slightly better accuracy according to the ATE of 0.151m on the circle and 0.114m on the Lemniscate, compared to the monocular SVO approach with 0.217m and 0.131m, respectively. This is expected since the Intel RealSense uses a stereo camera plus IMU setup and is a fully integrated solution, while sensor setup (ii) aims at minimal cost by only adding a single camera and otherwise exploiting the existing flight-controller IMU and onboard compute resources.

However, at the timing of writing this manuscript, the Intel RealSense T265 is being discontinued. Other possible solutions include camera sensors such as SevenSense [50], MYNT EYE [51], or MatrixVision [52], and other stand-alone cameras, combined with software frameworks like ArtiSense [53], SlamCore [54], or open-source frameworks like VINSmono [55], OpenVINS [56], or SVO Pro [43, 48], evaluated in [57]. Furthermore, there are other fully integrated alternatives to the Intel RealSense [58], including the Roboception [59] and the ModalAI Voxl CAM [60].

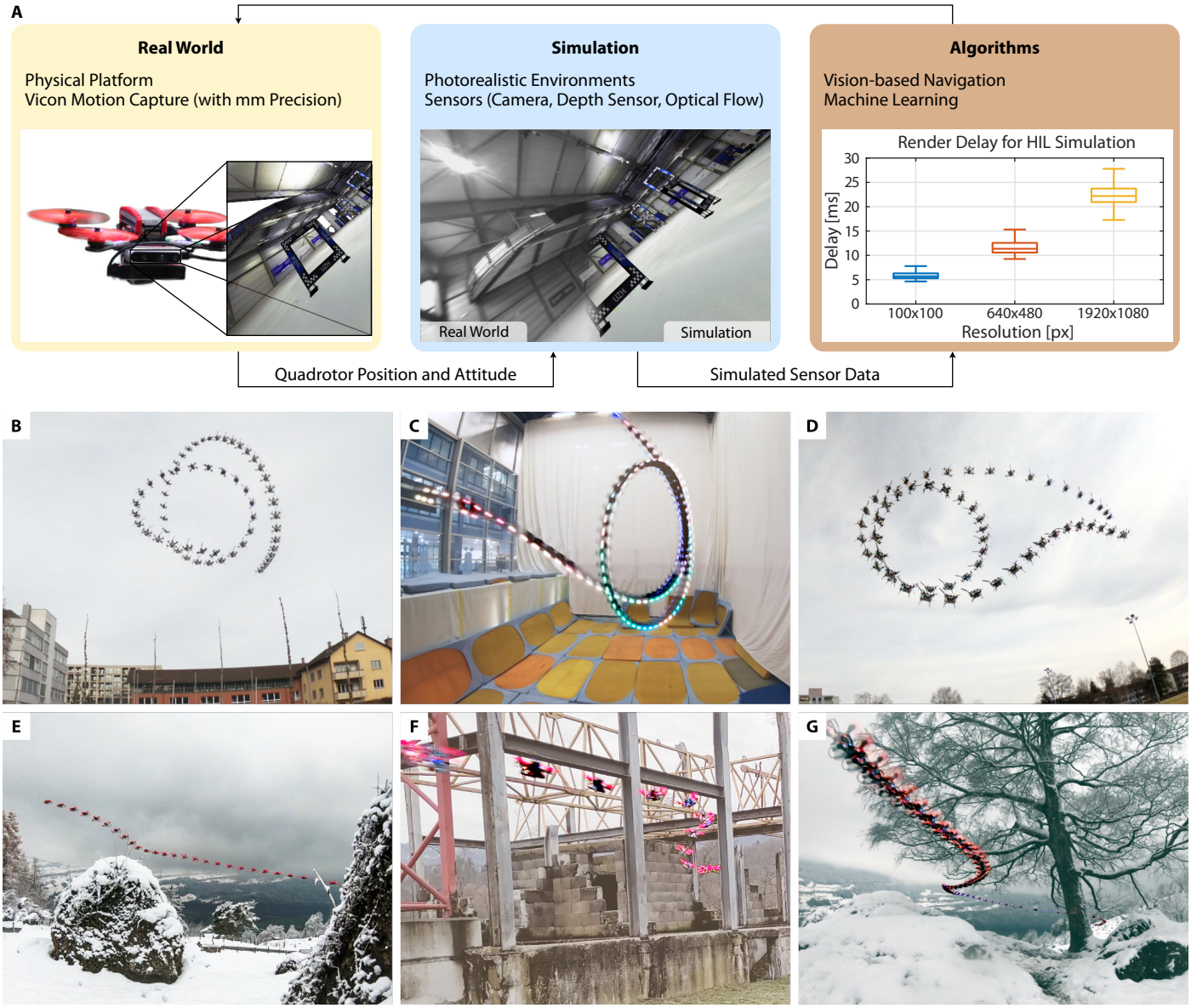


Fig. 5. A: The hardware-in-the-loop simulation of Agilicious consists of a real quadrotor flying in a motion capture system combined with photorealistic simulation of complex 3D environments. Multiple sensors can be simulated with minimal delays while virtually flying in various simulated scenes. Such hardware-in-the-loop simulation offers a modular framework for prototyping robust vision-based algorithms safely, efficiently, and inexpensively. B-G: The Agilicious platform is deployed in a diverse set of environments while only relying on onboard sensing and computation. B-D: The quadrotor performs a set of acrobatic maneuvers using a learned control policy. E-G: By leveraging zero-shot sim-to-real transfer, the quadrotor platform performs agile navigation through cluttered environments. Courtesy of Kaufmann et al. [9] and Loquercio et al. [35]

D. Demonstrators

The Agilicious software and hardware stack is intended as a flexible research platform. To illustrate its broad applicability, this section showcases a set of research projects that have been enabled through Agilicious. Specifically, we demonstrate the performance of our platform in two different experimental setups covering hardware-in-the-loop simulation and autonomous flight in the wild using only onboard sensing.

D.1. Hardware in the Loop Simulation

Developing vision-based perception and navigation algorithms for agile flight is not only slow and time-consuming, due to the large amount of data required to train and test perception algorithms in diverse settings, but it progressively becomes less safe and more expensive as more aggressive flights can lead to devastating crashes. This motivates the Agilicious framework to support hardware-in-the-loop simulation, which consists of flying a physical quadrotor in a motion-capture system while observing virtual photorealistic environments, as previously shown in [14]. The key advantage of hardware-in-the-loop simulation over classical synthetic experiments [23] is the usage of real-world dynamics and proprioceptive sensors, instead of idealized virtual devices, combined with the ability to simulate arbitrarily sparse or dense environments without the risk of crashing into real obstacles.

The simulation of complex 3D environments and realistic exteroceptive sensors is achieved using our high-fidelity quadrotor simulator [24] built on Unity [61]. The simulator can offer a rich and configurable sensor suite, including RGB cameras, depth sensors, optical flow, and

image segmentation, combined with variable sensor noise levels, motion blur, distortion and diverse environmental factors such as wind and rain. The simulator achieves this by introducing only minimal delays (see Figure 5A), ranging from 13ms for 640×480 VGA resolution to 22ms for 1920×1080 full HD images, when rendered on a NVIDIA RTX 2080 GPU.

Overall, the integration of our agile quadrotor platform and high-fidelity visual simulation provides an efficient framework for the rapid development of vision-based navigation systems in complex and unstructured environments.

D.2. Vision-based Agile Flight with Onboard Sensing and Compute

When a quadrotor can only rely on onboard vision and computation, perception needs to be effective, low-latency, and robust to disturbances. Violating this requirement may lead to crashes, especially during agile and fast maneuvers where latency has to be low and robustness to perception disturbances and noise must be high. However, vision systems either exhibit reduced accuracy or completely fail at high speeds due to motion blur, large pixel displacements, and quick illumination changes [62]. To overcome these challenges, vision-based navigation systems generally build upon two different paradigms. The first uses the traditional perception-planning-and-control pipeline, represented by standalone blocks which are executed in sequence and designed independently [2, 63–67]. Works in the second category substitute either parts or the complete perception-planning-and-control pipeline with learning-based methods [9, 68–76].

The Agilicious flight stack supports both paradigms and has been used to compare traditional and learning-based methods on agile navigation tasks in unstructured and structured environments (see Figure 5). Specifically, Agilicious facilitated quantitative analyses of approaches for autonomous acrobatic maneuvers [9](Fig. 5B-D) and high-speed obstacle avoidance in previously unknown environments [35](Fig. 5E-G). Both comparisons feature a rich set of approaches consisting of traditional planning and control [34, 64, 66] as well as learning-based methods [9, 35] with different input and output modalities. Thanks to its flexibility, Agilicious enables an objective comparison of these approaches on a unified control stack, without biasing results due to different low-level control strategies.

3. DISCUSSION

The presented Agilicious framework substantially advances the published state of the art in autonomous quadrotor platform research. It offers advanced computing capabilities combined with the most powerful open-source and open-hardware quadrotor platform created to date, opening the door for research on the next generation of autonomous robots. We see three main axes for future research based on our work.

First, we hypothesize that future flying robots will be smaller, lighter, cheaper, and consuming less power than what is possible today, increasing battery life, crash-resilience, as well as thrust-to-weight ratio and torque-to-inertia ratio [26]. This miniaturization is evident in state-of-the-art research towards direct hardware implementations of modern algorithms in the form of application-specific integrated circuit (ASIC)s, such as the Navion [77], the Movidius [78], or the PULP processor [79, 80]. These highly specialized in-silicon implementations are typically magnitudes smaller and more efficient than general compute units. Their success is rooted in the specific structure many algorithmic problems exhibit, such as the parallel nature of image data or the factor-graph representations used in estimation, planning, and control algorithms, like SLAM, model-predictive control, and neural network inference.

Second, the presented framework was mainly demonstrated with fixed-shape quadrotors. This is an advantage as the platform is easier to model and control, and less susceptible to hardware failures. Nevertheless, platforms with a dynamic morphology are by design more adaptable to the environment and potentially more power efficient [81–84]. For example, to increase flight time, a quadrotor might transform to a fixed-wing aircraft [85]. Due to its flexibility, Agilicious is the ideal tool for the future development of morphable and soft aerial systems.

Finally, vision-based agile flight is still in the early stages and has not yet reached the performance of professional human pilots. The main challenges lie in handling complex aerodynamics, e.g. transient torques or rotor inflow, low-latency perception and state estimation, and recovery from failures at high speeds. In the last few years, considerable progress has been made by leveraging data-driven algorithms [9, 22, 35, 86] and novel sensors as event-based cameras [36, 87], that provide a high dynamic range, low latency, and low battery consumption [88]. A major opportunity for future work is to complement the existing capabilities of Agilicious with novel compute devices such as the Intel Loihi [89–91] or SynSense Dynap [92] neuromorphic processing architecture, which are specifically designed to operate in an event-driven compute scheme. Due to the modular nature of Agilicious, individual software components can be replaced by these novel computing architectures, supporting rapid iteration and testing.

In summary, Agilicious offers a unique quadrotor testbed to accelerate current and future research on fast autonomous flight. Its versatility in both hardware and software allows deployment in a wide variety of tasks, ranging from exploration or search and rescue missions to acrobatic flight. Furthermore, the modularity of the hardware setup allows integrating novel sensors or even novel compute hardware, enabling to test such hardware on an autonomous agile vehicle. By open-sourcing Agilicious, we provide the research and industrial community access to a highly agile, versatile, and extendable quadrotor platform.

4. MATERIALS AND METHODS

Designing a versatile and agile quadrotor research platform requires to co-design its hardware and software, while carefully trading off competing design objectives such as onboard computing capacity and platform agility. In the following, the design choices that resulted in the flight hardware, compute hardware, and software design of Agilicious (see Figure 1) are explained in detail.

A. Compute Hardware

To exploit the full potential of highly unstable quadrotor dynamics, a high-frequency low-latency controller is needed. Both of these requirements are difficult to meet with general-purpose operating systems, which typically come without any real-time execution guarantees. Therefore, we deploy a low-level controller with limited compute capabilities but reliable real-time performance, which stabilizes high-bandwidth dynamics, such as the motor speeds or the vehicle's bodyrate. This allows complementing the system with a general-purpose high-level compute unit that can run Linux for versatile software deployment, with significantly relaxed real-time requirements.

High-Level Compute Board. The high-level of the system architecture provides all the necessary compute performance to run the full flight stack, including estimation, planning, optimization-based control, neural network inference, or other demanding experimental applications. Therefore, the main goal is to provide general-purpose computing power, while complying with the strict size and weight limits. We evaluate a multitude of different compute modules made from system-on-a-chip (SoC) solutions since they allow inherently small footprints. An overview is shown in Tab. 1. We exclude the evaluation of two popular contenders: (a) the Intel NUC platform, since it neither provides any

size and weight advantage over the Jetson Xavier AGX nor provides a general-purpose GPU; and (b) the Raspberry Pi compute modules since they do not offer any compute advantages over the Odroid and UpBoard, and no size and weight advantage over the NanoPi product family.

As we target general flight applications, fast prototyping, and experimentation, it is important to support a wide variety of software, which is why we chose a Linux-based system. TensorFlow [93] and PyTorch [94] are some of the most prominent frameworks with hardware-accelerated neural network inference. Both of them support accelerated inference on the Nvidia CUDA general-purpose GPU architecture, which renders nVidia products favorable, as other products have no or poorly-supported accelerators. Therefore, four valid options remain, listed in the second row of Tab. 1. While the Jetson Xavier AGX is beyond our size and weight goals, the Jetson Nano provides no advantage over the Xavier NX, rendering both the Jetson TX2 and Xavier NX viable solutions. Since these two CUDA-enabled compute modules require breakout boards to connect to peripherals, our first choice is the TX2 due to the better availability and diversity of such adapter boards and their smaller footprint. For the breakout board we recommend the ConnectTech Quasar [95], providing multiple USB ports, Ethernet, serial ports, and other interfaces for sensors and cameras.

Low-Level Flight Controller. The Low-Level Flight Controller provides real-time low-latency interfacing and control. A simple and widespread option is the open-source BetaFlight [40] software which runs on many commercially available flight controllers, such as the Radix[96]. However, BetaFlight is made for human-piloted drones and optimized for a good human flight feeling, but not for autonomous operation. Furthermore, even though it uses high-speed IMU readings for the control loop, it only provides very limited sensor readings at only 10 Hz. Therefore, Agilicious provides its own low-level flight controller implementation called "agiNuttx", reusing the same hardware as the BetaFlight controllers. This means that the wide variety of commercially available products can be bought and reflashed with agiNuttx to provide a low-level controller suited for autonomous agile flight.

In particular, we recommend using the BrainFPV Radix [96] controller, to deploy our agiNuttx software. The agiNuttx is based on the open-source NuttX [97] real-time operating system, optimized to run on embedded microcontrollers such as the STM32F4 used in many BetaFlight products. Our agiNuttx implementation interfaces with the motors' electronic speed controller (ESC) over the digital bi-directional DShot protocol, allowing not only to command the motors, but also receive individual rotor speed feedback. This feedback is provided to the high-level controller together with IMU, battery voltage, and flight mode information over a 1 MBaud serial bus at 500 Hz. The agiNuttx also provides closed-loop motor speed control, bodyrate control, and measurement time synchronization, allowing estimation and control algorithms to take full advantage of the available hardware.

B. Flight Hardware

To maximize the agility of the drone, it needs to be designed as lightweight and small as possible [25] while still being able to accommodate the Jetson TX2 compute unit. With this goal in mind, we provide a selection of cheap off-the-shelf drone components summarized in Tab. 1. The Armattan Chameleon 6 inch frame is used as a base since it is one of the smallest frames with ample space for the compute hardware. Being made out of carbon fiber, it is durable and lightweight. The other structural parts of the quadrotor are custom-designed plastic parts (PLA and TPU material) and produced using a 3D printer. Most components are made out of PLA which is stiffer and only parts that act as impact protectors or as predetermined breaking points are made out of TPU. For propulsion, a 5.1 inch three-bladed propeller is used in combination with a fast-spinning brushless DC motor rated at a high maximum power of 758 W. The chosen motor-propeller combination achieves a continuous static thrust of $4\times$ 9.5 N on the quadrotor and consumes about 400 W of power per motor. To match the high power demand of the motors, a lithium-polymer battery with 1800 mA h and a rating of 120C is used. Therefore, the total peak current of 110 A is well within the 216 A limit of the battery. The motors are powered by an electronic speed controller in the form of the Hobbywing XRotor ESC, due to its compact form factor, its high continuous current rating (60 A per motor), and support of the DShot protocol supporting motor speed feedback.

C. Sensors

To navigate arbitrary uninstrumented environments, drones need means to measure their absolute or relative location and state. Due to the size and weight constraints of aerial vehicles, and especially the direct impact of weight and inertia on the agility of the vehicle, visual-inertial odometry has proven to be the go-to solution for aerial navigation. The complementary sensing modality of cameras and IMUs, their low price and excellent availability, together with the depth-sensing capabilities of stereo camera configurations allow for a simple, compact, and complete perception setup.

Furthermore, our agiNuttxflight controller already provides high-rate filtered inertial measurements and can be combined with any off-the-shelf camera [50–52] and open-source [43, 48, 55, 56] or commercial [53, 54] software to implement a VIO pipeline. Additionally, there exist multiple fully integrated products providing out-of-the-box VIO solutions, such as the Intel RealSense [58], the Roboception rc_visard [59], and the ModalAI Voxl CAM [60].

D. The Agilicious Flight Stack Software

To exploit the full potential of our platform and enable fast prototyping, we provide the Agilicious flight stack as an open-source software package. The main development goals for Agilicious are aligned with our overall design goals: high versatility and modularity, low latency for agile flight, and transferability between simulation and multiple real-world platforms. These goals are met by splitting the software stack into two parts.

The core library, called "agilib", is built with minimal dependencies but provides all functionality needed for agile flight, implemented as individual modules (illustrated in Figure 1). It can be deployed on a large range of computing platforms, from lightweight low-power devices to parallel neural network training farms built on heterogeneous server architectures. This is enabled by avoiding dependencies on other software components that could introduce compatibility issues and rely only on the core C++-17 standard and the Eigen library for linear algebra. Additionally, agilib includes a standalone set of unit tests and benchmarks that can be run independently, with minimal dependencies, and in a self-contained manner.

To provide compatibility to existing systems and software, the second component is a ROS-wrapper, called "agiros", which enables networked communication, data logging, provides a simple GUI interface to command the vehicle and allows for integration with other software components. This abstraction between "agiros" and the core library "agilib" allows a more flexible deployment on systems or in environments where ROS is not available, not needed, or communication overhead must be avoided. On the other hand, the ROS-enabled Agilicious provides versatility and modularity due to a vast number of open-source ROS packages provided by the research community.

For flexible and fast development, "agilib" uses modular software components unified in an umbrella structure called "pipeline" and orchestrated by a control logic, called "pilot". The modules consist of an "estimator", "sampler", "controllers", and a "bridge", all working

without General-Purpose GPU

Product	Odroid XU4	Intel UpBoard	NanoPi Neo 3	NanoPi Neo air
CPU	8× 32-bit ARM 2.1 GHz	4× 64-bit Atom 1.92 GHz	4× 64-bit ARM 1.5 GHz	4× 32-bit ARM 1.2 GHz
RAM	2 GB LPDDR3	4 GB LPDDR3	2 GB LPDDR4	512 MB LPDDR3
GPU	Mali-T628	Intel HD400	Mali-450 MP4	Mali-400 MP2
FLOPS	∼120 GFLOPS	∼ 115 GFLOPS	∼40 GFLOPS	∼10 GFLOPS
Storage	up to 128 GB EMMC	up to 64 GB EMMC	only SD card	8 GB EMMC
Interfaces	USB, Ethernet, Serial, I2C, SPI, GPIO	USB, Ethernet, Serial, I2C, SPI, GPIO, 1 camera	USB, Ethernet, Serial, I2C, SPI, GPIO	USB, Ethernet, WIFI, Serial, I2C, SPI, GPIO, 1 camera
Size	83 × 58 × 19 mm	85 × 57 × 20 mm	40 × 40 × 23 mm	40 × 40 × 10 mm
Weight	59 g	79 g	36 g	24 g

with General-Purpose GPU

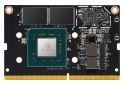
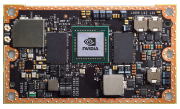
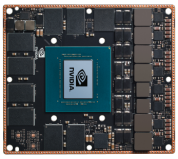
Product	 nVidia Jetson Nano	 nVidia Jetson TX2	nVidia Jetson Xavier NX	 nVidia Jetson AGX Xavier
CPU	4× 64-bit ARM 1.43 GHz	6× 64-bit ARM 2.0 GHz	6× 64-bit ARM 1.9 GHz	8× 64-bit ARM 2.26 GHz
RAM	4 GB LPDDR4	8 GB LPDDR4	8 GB LPDDR4	32 GB LPDDR4
GPU	128× Maxwell CUDA	256× Pascal CUDA	384× Volta CUDA	512× Volta CUDA
FLOPS	472 GFLOPS	1.33 TFLOPS	2.12 TFLOPS	11 TFLOPS
Storage	16 GB EMMC	32 GB EMMC	16 GB EMMC	32 GB EMMC
Interfaces	USB, Ethernet, Serial, I2C, SPI, GPIO, 4 cameras	USB, Ethernet, WIFI, Serial, I2C, SPI, GPIO, 6 cameras	USB, Ethernet, Serial, I2C, SPI, GPIO, 6 cameras	USB, Ethernet, Serial, I2C, SPI, GPIO, 6 cameras
Size	69.9 × 45 × 22 mm	87 × 50 × 34 mm	69.9 × 45 × 22 mm	100 × 87 × 58 mm
Weight	63 g	154 g	79 g	650 g

Table 1. Overview of compute hardware commonly used on autonomous flying vehicles. Due to the emerging trend of deploying learning-based methods onboard, hardware solutions are grouped according to the presence of a general-purpose GPU, enabling real-time inference.

Component	Product	Specification
Frame	Armattan Chameleon 6 inch	4 mm carbon fiber, 86 g
Motor	Xrotor 2306	23×6 mm stator, 2400 kV, 758 W, 4× 27.5 g
Propeller	Azure Power SFP 5148	5.1 inch length and 4.8 inch pitch, 4× 5 g
Battery	Tattoo R-Line 1800	4× 3.7 V, 1800 mA h, 199 g
Flight Controller	BrainFPV radix	BetaFlight or custom firmware, 6 g
Motor Controller	HobbyWing XRotor	DShot protocol, 4× 60 A, 15 g
Compute Unit	nVidia Jetson TX2	6× ARM 2.0 GHz, 256× CUDA cores, 8 GB, 154 g

Table 2. Overview of the components of the flight hardware design.

together to track a so-called "reference". These modules are executed in sequential order (illustrated in Figure 1) within a forward pass of the pipeline, corresponding to one control cycle. However, each module can spawn its individual threads to perform parallel tasks, e.g. asynchronous sensor data processing. Agilicious provides a collection of state-of-the-art implementations for each module inherited from base classes, allowing to create new implementations outside of the core library, and linking them into the pilot at runtime. Moreover, Agilicious is not only capable to control a drone when running onboard the vehicle, but can also run offboard on computationally more capable hardware and send commands to the drone over low-latency wireless serial interfaces.

Finally, the core library is completed by a physics simulator. While this might seem redundant due to the vast variety of simulation pipelines available [23, 24, 98], it allows to use high-fidelity models (e.g. BEM [22] for aerodynamics), evaluate software prototypes without having to interface with other frameworks, avoids dependencies, and enables even simulation-based continuous integration testing that can run on literally any platform. The pilot, software modules, and simulator are all described in the following sections.

D.1. Pilot

The pilot contains the main logic needed for flight operation, handling of the individual modules, and interfaces to manage references and task commands. In its core, it loads and configures the software modules according to YAML [99] parameter files, runs the control loop, and provides simplified user interfaces to manage flight tasks, such as position and velocity control or trajectory tracking. For all state descriptions, we use a right-handed coordinate system located in the center of gravity, with the ${}_B\boldsymbol{e}_z$ pointing in body-relative upward thrust direction, and ${}_B\boldsymbol{e}_x$ pointing along with the drone's forward direction. Motion is represented with respect to an inertial world frame with ${}_I\boldsymbol{e}_z$ pointing against the gravity direction, where translational derivatives (e.g. velocity) are expressed in the world frame and rotational derivatives (e.g. bodyrate) are expressed in the body frame.

Pipeline. The pipeline is a distinct configuration of the sequentially processed modules. These pipeline configurations can be switched at runtime by the pilot or the user, allowing to switch to backup con-

figurations in an emergency, or quickly alternate between different prototyping configurations.

Estimator. The first module in the pipeline is the estimator, which provides a time-stamped state estimate for the subsequent software modules in the control cycle. A state estimate $\boldsymbol{x} = [\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{v}, \boldsymbol{\omega}, \boldsymbol{a}, \boldsymbol{\tau}, \boldsymbol{j}, \boldsymbol{s}, \boldsymbol{b}_\omega, \boldsymbol{b}_a, \boldsymbol{f}_d, \boldsymbol{f}]$, represents position $\boldsymbol{p}$, orientation unit quaternion $\boldsymbol{q}$, velocity $\boldsymbol{v}$, bodyrate $\boldsymbol{\omega}$, linear $\boldsymbol{a}$ and angular $\boldsymbol{\tau}$ accelerations, jerk $\boldsymbol{j}$, snap $\boldsymbol{s}$, gyroscope and accelerometer bias $\boldsymbol{b}_\omega$ and $\boldsymbol{b}_a$, and desired and actual single rotor thrusts $\boldsymbol{f}_d$ and $\boldsymbol{f}$. Agilicious provides a feed-through estimator to include external estimates or ground-truth from a simulation, as well as two extended Kalman filters, one with IMU filtering, and one using the IMU as propagation model. These estimators can easily be replaced or extended to work with additional measurement sources, such as GPS or altimeters, other estimation systems, or even implement complex localization pipelines such as visual-inertial odometry.

Sampler. For trajectory tracking using a state estimate from the aforementioned estimator, the controller module needs to be provided with a subset of points of the trajectory that encode the desired progress along it, provided by the sampler. Agilicious implements two types of samplers: a time-based sampling scheme that computes progress along the trajectory based on the time since trajectory start, and a position-based sampling scheme that selects trajectory progress based on the current position of the platform, trading off temporally accurate tracking for higher robustness and lower positional tracking error.

Controller. To control the vehicle along the sampled reference setpoints, a multitude of controllers are available, which provide the closed-loop commands for the low-level controller. We provide a state-of-the-art MPC that uses the full non-linear model of the platform and which allows to track highly agile references using single-rotor thrust commands or bodyrate control.Additionally, we include a cascaded geometric controller based on the quadrotor's differential flatness [100]. The pipeline can cascade two controllers, which even allows combining the aforementioned MPC [41] or geometric approaches with an intermediate controller for which we provide an L1 adaptive controller [101] and an incremental nonlinear dynamic inversion controller [41].

Bridge. A bridge serves as an interface to hardware or software by sending control commands to a low-level controller or other means of communication sinks. Low-level commands can either be single rotor thrusts or bodyrates in combination with a collective thrust. Agilicious provides a set of bridges to communicate via commonly used protocols such as ROS, SBUS, and serial. While the ROS-bridge can be used to easily integrate Agilicious in an existing software stack that relies on ROS, the SBUS protocol is a widely used standard in the FPV community and therefore allows to interface Agilicious to off-the-shelf flight controllers such as BetaFlight [40]. For simple simulation, there is a specific bridge to interface with the popular RotorS [23] simulator, which is however less accurate than our own simulation described in Sec. D.2. As Agilicious is written in a general abstract way, it runs on onboard compute modules and offboard, for which case we provide a bridge to interface with the LAIRD[42] wireless serial interface. Finally, Agilicious also provides a bridge to communicate to the custom low-level controller described in Sec. A. This provides the advantage of gaining access to closed-loop single rotor speed control, high-frequency IMU, rotor speed, and voltage measurements at 500 Hz, all provided to the user through the bridge.

References. References are used in conjunction with a controller to encode the desired flight path of a quadrotor. In Agilicious, a reference is fed to the sampler, which generates a receding-horizon vector of setpoints that are then passed to the controller. The software stack implements a set of reference types, consisting of *Hover*, *Velocity*, *Polynomial*, and *Sampled*. While *Hover* references are uniquely defined by a reference position and a yaw angle, a *Velocity* reference specifies a desired linear velocity with a yaw rate. By exploiting the differential flatness of the quadrotor platform, *Polynomial* references describe the position and yaw of the quadrotor as polynomial functions of time. Sampled references provide the most general reference representations. Agilicious provides interfaces to generate, and receive such sampled references and also defines a message and file format to store references to a file. By defining such formats, a wide variety of trajectories can be generated, communicated, saved, and executed using Python or other languages. Finally, to simplify the integration and deployment of other control approaches, Agilicious also exposes a command feedthrough, that allows taking direct control over the applied low-level commands. For safety, even when command feedthrough is used, Agilicious provides readily available back-up control that can take over on user request or on timeout.

Guard. To further support users in fast prototyping, Agilicious provides a so-called guard. This guard uses the quadrotor's state-estimate or an alternative estimate (e.g. from motion capture when flying with VIO prototypes) together with a user-defined spatial bounding box to detect unexpected deviations from the planned flight path. Further detection metrics can be implemented by the user. Upon violation of e.g. the spatial bounding box, the guard can switch control to an alternative pipeline using a backup estimate and control configuration. This safety pipeline can e.g. use a motion capture system and a simple geometric controller, while the main pipeline runs a VIO estimator, an MPC, reinforcement learning control strategies, or other software prototypes. By providing this measure of backup, Agilicious significantly reduces the risk of crashes when testing novel algorithms, and allows to iterate over research prototypes faster.

D.2. Simulation

The Agilicious software stack includes a simulator that allows simulating quadrotor dynamics at various levels of fidelity to accelerate prototyping and testing. Specifically, Agilicious models motor dynamics and aerodynamics acting on the platform. To also incorporate the different, possibly off-the-shelf, low-level controllers that can be used on the quadrotors, the simulator can optionally simulate the behavior of low-level controllers. One simulator update, typically called at 1 kHz, includes a call to the simulated low-level controller, the motor model, the aerodynamics model, and the rigid body dynamics model in a sequential fashion. Each of these components is explained in the following.

Low-Level Controller & Motor Model. Simulated low-level controllers run at simulation frequency and convert collective thrust and bodyrate commands into individual motor speed commands. The usage of a simulated low-level controller is optional if the computed control commands are already in the form of individual rotor thrusts. In this case, the thrusts are mapped to motor speed commands and then directly fed to the simulated motor model. The motors are modeled as a first-order system with a time constant which can be identified on a thrust test stand.

Aerodynamics. The simulated aerodynamics model lift and drag produced by the rotors from the current ego-motion of the platform and the individual rotor speeds. Agilicious implements two rotor models: *Quadratic* and *BEM*. The *Quadratic* model implements a simple quadratic mapping from rotor rotational speed to produced thrust, as commonly done in quadrotor simulators [23, 24, 98]. While such a model does not account for effects imposed by the movement of a rotor through the air, it is highly efficient to compute. In contrast, the *BEM* model leverages Blade-Element-Momentum-Theory (BEM) to account for the effects of varying relative airspeed on the rotor thrust. To further

increase the fidelity of the simulation, a neural network predicting the residual forces and torques (e.g. unmodeled rotor to rotor interactions and turbulence) can be integrated into the aerodynamics model. For details regarding the *BEM* model and the neural network augmentation, we refer the reader to [22].

Rigid Body Dynamics. Provided with a model of the forces and torques acting on the platform predicted by the aerodynamics model, the system dynamics of the quadrotor are integrated using a 4th order Runge-Kutta scheme with a step size of 1 ms. Agilicious also implements different integrators such as explicit Euler or symplectic Euler.

Apart from providing its own state-of-the-art quadrotor simulator, Agilicious can also be interfaced with external simulators. Interfaces to the widely-used RotorS quadrotor simulator [23] and Flightmare [24], including the HIL simulator, are already provided in the software stack.

5. ACKNOWLEDGMENTS

Author Contribution. P.F. developed the Agilicious software concepts and architecture, contributed to the Agilicious implementation, helped with the experiments, and wrote the manuscript. E.K. contributed to the Agilicious implementation, designed the Agilicious hardware, helped with the experiments, and wrote the manuscript. A.R., R.P., S.S., and L.B. contributed to the Agilicious implementation, helped with the experiments, and wrote the manuscript. T.L. evaluated the hardware components, designed and built the Agilicious hardware, helped with the experiments, and wrote the manuscript. Y.S. contributed to the Agilicious implementation, helped with the experiments, and wrote the manuscript. A.L. helped with the experiments and wrote the manuscript. D.S. provided funding, contributed to the design and analysis of the experiments, and revised the manuscript.

Funding. This work was supported by the National Centre of Competence in Research (NCCR) Robotics through the Swiss National Science Foundation (SNSF) and the European Union's Horizon 2020 Research and Innovation Program under grant agreement No. 871479 (AERIAL-CORE) and the European Research Council (ERC) under grant agreement No. 864042 (AGILEFLIGHT).

Data and Materials. The main purpose of this paper is to share our data and materials. Therefore all materials, both software as well as hardware designs, are open-source accessible at https://agilicious.dev under the GPL v3.0 license.

REFERENCES

1. D. Mellinger, N. Michael, V. Kumar, Trajectory generation and control for precise aggressive maneuvers with quadrotors, *Int. J. Robot. Research* 664–674 (2012).
2. G. Loianno, C. Brunner, G. McGrath, V. Kumar, Estimation, control, and planning for aggressive flight with a small quadrotor with a single camera and imu, *IEEE Robotics and Automation Letters* 404–411 (2017).
3. E. Kaufmann, M. Gehrig, P. Foehn, R. Ranftl, A. Dosovitskiy, V. Koltun, D. Scaramuzza, Beauty and the beast: Optimal methods meet learning for drone racing, *2019 International Conference on Robotics and Automation (ICRA)*, 690–696 (IEEE, 2019).
4. K. Mohta, M. Watterson, Y. Mulgaonkar, S. Liu, C. Qu, A. Makineni, K. Saulnier, K. Sun, A. Zhu, J. Delmerico, K. Karydis, N. Atanasov, G. Loianno, D. Scaramuzza, K. Daniilidis, C. J. Taylor, V. Kumar, Fast, autonomous flight in gps-denied and cluttered environments, *J. Field Robot.* 101–120 (2018).
5. B. Zhou, J. Pan, F. Gao, S. Shen, RAPTOR: robust and perception-aware trajectory replanning for quadrotor fast flight, *IEEE Transactions on Robotics* (2021).
6. P. Foehn, D. Brescianini, E. Kaufmann, T. Cieslewski, M. Gehrig, M. Muglikar, D. Scaramuzza, Alphapilot: Autonomous drone racing, *Robotics: Science and Systems (RSS)* (2020).
7. S. Li, M. M. Ozo, C. D. Wagter, G. C. de Croon, Autonomous drone race: A computationally efficient vision-based navigation and control strategy, *Robotics and Autonomous Systems* **133** (2020).
8. H. Nguyen, M. Kamel, K. Alexis, R. Siegwart, Model predictive control for micro aerial vehicles: A survey, *2021 European Control Conference (ECC)*, 1556–1563 (IEEE, 2021).
9. E. Kaufmann, A. Loquercio, R. Ranftl, M. Müller, V. Koltun, D. Scaramuzza, Deep drone acrobatics, *RSS: Robotics, Science, and Systems* (2020).
10. P. Foehn, A. Romero, D. Scaramuzza, Time-optimal planning for quadrotor waypoint flight, *Science Robotics* **6** (2021).
11. H. Moon, J. Martinez-Carranza, T. Cieslewski, M. Faessler, D. Falanga, A. Simovic, D. Scaramuzza, S. Li, M. Ozo, C. De Wagter, others, Challenges and implemented technologies used in autonomous drone racing, *Intelligent Service Robotics* (2019).
12. J. A. Cocoma-Ortega, J. Martínez-Carranza, Towards high-speed localisation for autonomous drone racing, *Mexican International Conference on Artificial Intelligence* (Springer, 2019).
13. R. Madaan, N. Gyde, S. Vemprala, M. Brown, K. Nagami, T. Taubner, E. Cristofalo, D. Scaramuzza, M. Schwager, A. Kapoor, Airsim drone racing lab, *NeurIPS 2019 Competition and Demonstration Track* (PMLR, 2020).
14. W. Guerra, E. Tal, V. Murali, G. Ryou, S. Karaman, FlightGoggles: Photorealistic sensor simulation for perception-driven robotics using photogrammetry and virtual reality, *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, 2019).
15. CORDIS - European Comission, AgileFlight, https://cordis.europa.eu/project/id/864042. Accessed: 2021-7-30.
16. L. Meier, P. Tanskanen, L. Heng, G. H. Lee, F. Fraundorfer, M. Pollefeys, PIXHAWK: A micro aerial vehicle design for autonomous flight using onboard computer vision, *Autonom. Rob.* 21–39 (2012).
17. I. Sa, M. Kamel, M. Burri, M. Bloesch, R. Khanna, M. Popovic, J. Nieto, R. Siegwart, Build your own Visual-Inertial drone: A Cost-Effective and Open-Source autonomous drone, *IEEE Robot. Autom. Mag.* 89–103 (2018).
18. M. Faessler, A. Franchi, D. Scaramuzza, Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories, *IEEE Robot. Autom. Lett.* 620–626 (2018).
19. H. Oleynikova, C. Lanegger, Z. Taylor, M. Pantic, A. Millane, R. Siegwart, J. Nieto, An open-source system for vision-based micro-aerial vehicle mapping, planning, and flight in cluttered environments, *J. Field Robot.* 642–666 (2020).
20. T. Baca, M. Petrlik, M. Vrba, V. Spurny, R. Penicka, D. Hert, M. Saska, The MRS UAV system: Pushing the frontiers of reproducible research, real-world deployment, and education with autonomous unmanned aerial vehicles, *J. Intell. Rob. Syst.* p. 26 (2021).
21. G. Hattenberger, M. Bronz, M. Gorraz, Using the paparazzi uav system for scientific research, *International Micro Air Vehicle Conference and Competition*, 247–252 (2014).
22. L. Bauersfeld, E. Kaufmann, P. Foehn, S. Sun, D. Scaramuzza, Neurobem: Hybrid aerodynamic quadrotor model, *RSS: Robotics, Science, and Systems* (2021).
23. F. Furrer, M. Burri, M. Achtelik, R. Siegwart. Rotors—a modular gazebo mav simulator framework. *Robot Operating System (ROS)* (Springer, 2016), 595–625.
24. Y. Song, S. Naji, E. Kaufmann, A. Loquercio, D. Scaramuzza, Flightmare: A flexible quadrotor simulator, *Conference on Robot Learning* (2020).
25. V. Kumar, N. Michael, Opportunities and challenges with autonomous micro aerial vehicles, *The International Journal of Robotics Research* 1279–1291 (2012).
26. D. Floreano, R. J. Wood, Science, technology and the future of small autonomous drones, *Nature* 460–466 (2015).
27. E. Tal, S. Karaman, Accurate tracking of aggressive quadrotor trajectories using incremental nonlinear dynamic inversion and differential flatness, *IEEE Transactions on Control Systems Technology* 1203–1218 (2020).
28. A. Antonini, W. Guerra, V. Murali, T. Sayre-McCord, S. Karaman, The blackbird uav dataset, *The International Journal of Robotics Research* 1346–1364 (2020).

29. L. Meier, D. Honegger, M. Pollefeys, Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms, *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 6235–6240 (2015).
30. P. D. SAS, Parrot ANAFI ai, https://www.parrot.com/en/drones/anafi-ai. Accessed: 2021-7-20.
31. DJI, DJI digital FPV system, https://www.dji.com/fpv. Accessed: 2021-7-20.
32. Skydio, Skydio (2021).
33. W. Giernacki, M. Skwierczyński, W. Witwicki, P. Wroński, P. Kozierski, Crazyflie 2.0 quadrotor as a platform for research and education in robotics and control engineering, *International Conference on Methods and Models in Automation and Robotics (MMAR)*, 37–42 (2017).
34. D. Falanga, P. Foehn, P. Lu, D. Scaramuzza, Pampc: Perception-aware model predictive control for quadrotors, *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, 1–8 (IEEE, 2018).
35. A. Loquercio, E. Kaufmann, R. Ranftl, M. Müller, V. Koltun, D. Scaramuzza, Agile autonomy: Learning high-speed flight in the wild, *Science Robotics* (2021). Under review.
36. S. Sun, G. Cioffi, C. De Visser, D. Scaramuzza, Autonomous quadrotor flight despite rotor failure with onboard vision sensors: Frames vs. events, *IEEE Robotics and Automation Letters* 580–587 (2021).
37. B. Nisar, P. Foehn, D. Falanga, D. Scaramuzza, VIMO: Simultaneous visual inertial Model-Based odometry and force estimation, *IEEE Robot. Autom. Lett.* 2785–2792 (2019).
38. Y. Song, M. Steinweg, E. Kaufmann, D. Scaramuzza, Autonomous drone racing with deep reinforcement learning, *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)* (2021).
39. J. Delmerico, T. Cieslewski, H. Rebecq, M. Faessler, D. Scaramuzza, Are we ready for autonomous drone racing? the uzh-fpv drone racing dataset, *IEEE Int. Conf. Robot. Autom. (ICRA)* (2019).
40. The Betaflight Open Source Flight Controller Firmware Project, Betaflight, https://github.com/betaflight/betaflight. Accessed: 2021-7-20.
41. S. Sun, A. Romero, P. Foehn, E. Kaufmann, D. Scaramuzza, A comparative study of nonlinear mpc and differential-flatness-based control for quadrotor agile flight, *IEEE Transactions on Robotics* 1–17 (2022).
42. Laird connectivity, https://www.lairdconnect.com/. Accessed: 2021-7-20.
43. C. Forster, Z. Zhang, M. Gassner, M. Werlberger, D. Scaramuzza, SVO: Semidirect visual odometry for monocular and multicamera systems, *IEEE Trans. Robot.* 249–265 (2017).
44. Z. Zhang, D. Scaramuzza, A tutorial on quantitative trajectory evaluation for visual(-inertial) odometry, *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)* (2018).
45. A. Romero, S. Sun, P. Foehn, D. Scaramuzza, Model predictive contouring control for time-optimal quadrotor flight, *IEEE Transactions on Robotics* 1–17 (2022).
46. F. Nan, S. Sun, P. Foehn, D. Scaramuzza, Nonlinear mpc for quadrotor fault-tolerant control, *IEEE Robotics and Automation Letters* 5047–5054 (2022).
47. A. I. Automation, Ati mini40-si-20-1, https://www.ati-ia.com/products/ft/ft_models.aspx?id=Mini40. Accessed: 2021-11-30.
48. C. Forster, Z. Zhang, M. Gassner, M. Werlberger, D. Scaramuzza, Svo pro, http://rpg.ifi.uzh.ch/svo_pro.html. Accessed: 2021-11-30.
49. J. Rehder, J. Nikolic, T. Schneider, T. Hinzmann, R. Siegwart, Extending kalibr: Calibrating the extrinsics of multiple imus and of individual axes, *IEEE Int. Conf. Robot. Autom. (ICRA)* (2016).
50. SevenSense, Alphasense, https://www.sevensense.ai/product/alphasense-position. Accessed: 2021-11-30.
51. M. EYE, S210, https://www.mynteye.com/pages/s210. Accessed: 2021-11-30.
52. M. Vision, Bluefox, https://www.matrix-vision.com/en/products/areas/MA08. Accessed: 2021-11-30.
53. ArtiSense, Artislam, https://www.artisense.ai/vinspro-2020. Accessed: 2021-11-30.
54. SLAMcore, Spatial intelligence sdk, https://www.slamcore.com/spatial-intelligence-sdk. Accessed: 2021-11-30.
55. T. Qin, P. Li, S. Shen, Vins-mono: A robust and versatile monocular visual-inertial state estimator, *IEEE Trans. Robot.* (2018).
56. P. Geneva, K. Eckenhoff, W. Lee, Y. Yang, G. Huang, Openvins: A research platform for visual-inertial estimation, *IEEE Int. Conf. Robot. Autom. (ICRA)* (2020).
57. J. Delmerico, D. Scaramuzza, A benchmark comparison of monocular visual-inertial odometry algorithms for flying robots, *IEEE Int. Conf. Robot. Autom. (ICRA)*, 2502–2509 (2018).
58. Intel realsense d400 series product family, https://www.intel.com/content/dam/support/us/en/documents/emerging-technologies/intel-realsense-technology/Intel-RealSense-D400-Series-Datasheet.pdf (2019).
59. Roboception, rc_visard, https://roboception.com/en/rc_visard-en/.
60. ModalAI, Voxl cam core dev kit, https://www.modalai.com/pages/voxl-cam-perception-engine. Accessed: 2021-11-30.
61. A. Juliani, V.-P. Berges, E. Vckay, Y. Gao, H. Henry, M. Mattar, D. Lange, Unity: A general platform for intelligent agents, *arXiv e-prints* (2018).
62. C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, J. J. Leonard, Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age, *IEEE Trans. Robot.* (2016).
63. H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, J. I. Nieto, Voxblox: Incremental 3d euclidean signed distance fields for on-board MAV planning, *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, 1366–1373 (2017).
64. P. Florence, J. Carter, R. Tedrake. Integrated perception and control at high speed: Evaluating collision avoidance maneuvers without maps. *Algorithmic Foundations of Robotics XII* (Springer, 2020), 304–319.
65. D. Falanga, E. Mueggler, M. Faessler, D. Scaramuzza, Aggressive quadrotor flight through narrow gaps with onboard sensing and computing using active vision, *IEEE Int. Conf. Robot. Autom. (ICRA)*, 5774–5781 (IEEE, 2017).
66. B. Zhou, F. Gao, L. Wang, C. Liu, S. Shen, Robust and efficient quadrotor trajectory generation for fast autonomous flight, *IEEE Robotics and Automation Letters* 3529–3536 (2019).
67. D. Falanga, E. Mueggler, M. Faessler, D. Scaramuzza, Aggressive quadrotor flight through narrow gaps with onboard sensing and computing using active vision, *IEEE Int. Conf. Robot. Autom. (ICRA)*, 5774–5781 (2017).
68. W. Zeng, W. Luo, S. Suo, A. Sadat, B. Yang, S. Casas, R. Urtasun, End-to-end interpretable neural motion planner, *IEEE Int. Conf. Comput. Vis. Pattern Recog. (CVPR)*, 8660–8669 (2019).
69. W. Zeng, S. Wang, R. Liao, Y. Chen, B. Yang, R. Urtasun, Dsdnet: Deep structured self-driving network, *Eur. Conf. Comput. Vis. (ECCV)*, 156–172 (2020).
70. S. Bansal, V. Tolani, S. Gupta, J. Malik, C. J. Tomlin, Combining optimal control and learning for visual navigation in novel environments, *Conference on Robot Learning, CoRL 2019*, 420–429 (PMLR, 2019).
71. N. Homayounfar, W.-C. Ma, J. Liang, X. Wu, J. Fan, R. Urtasun, Dagmapper: Learning to map by discovering lane topology, *IEEE Int. Conf. Comput. Vis. Pattern Recog. (CVPR)*, 2911–2920 (2019).
72. Z. Zhang, D. Scaramuzza, Perception-aware receding horizon navigation for mavs, *IEEE Int. Conf. Robot. Autom. (ICRA)*, 2534–2541 (2018).
73. S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. Bagnell, M. Hebert, Learning monocular reactive UAV control in cluttered natural environments, *IEEE Int. Conf. Robot. Autom. (ICRA)*, 1765–1772 (2013).
74. F. Sadeghi, S. Levine, CAD2RL: real single-image flight without a single real image, *Robotics: Science and Systems RSS*, N. M. Amato, S. S. Srinivasa, N. Ayanian, S. Kuindersma, eds. (2017).
75. A. Loquercio, A. I. Maqueda, C. R. del-Blanco, D. Scaramuzza, Dronet: Learning to fly by driving, *IEEE Robotics Autom. Lett.* 1088–1095 (2018).
76. D. Gandhi, L. Pinto, A. Gupta, Learning to fly by crashing, *International Conference on Intelligent Robots and Systems, IROS*, 3948–3955 (2017).
77. A. Suleiman, Z. Zhang, L. Carlone, S. Karaman, V. Sze, Navion: A 2-mw fully integrated Real-Time Visual-Inertial odometry accelerator for autonomous navigation of nano drones, *IEEE J. Solid-State Circuits* 1106–1119 (2019).
78. Intel Corporation, Intel movidius myriad X vision process-

ing unit, https://www.intel.com/content/www/us/en/products/details/processors/movidius-vpu/movidius-myriad-x.html. Accessed: 2021-8-2.

79. D. Palossi, A. Loquercio, F. Conti, F. Conti, E. Flamand, E. Flamand, D. Scaramuzza, L. Benini, L. Benini, A 64mw dnn-based visual navigation engine for autonomous nano-drones, *IEEE Internet of Things Journal* 1–1 (2019).
80. D. Palossi, F. Conti, L. Benini, An open source and open hardware deep Learning-Powered visual navigation engine for autonomous Nano-UAVs, *2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, 604–611 (2019).
81. E. Ajanic, M. Feroskhan, S. Mintchev, F. Noca, D. Floreano, Bioinspired wing and tail morphing extends drone flight capabilities, *Science Robotics* **5** (2020).
82. E. Chang, L. Y. Matloff, A. K. Stowers, D. Lentink, Soft biohybrid morphing wings with feathers underactuated by wrist and finger motion, *Science Robotics* **5** (2020).
83. D. Falanga, K. Kleber, S. Mintchev, D. Floreano, D. Scaramuzza, The foldable drone: A morphing quadrotor that can squeeze and fly, *IEEE Robotics and Automation Letters* 209–216 (2019).
84. L. Bauersfeld, L. Spannagl, G. Ducard, C. Onder, Mpc flight control for a tilt-rotor vtol aircraft, *IEEE Transactions on Aerospace and Electronic Systems* 1–13 (2021).
85. R. D'Sa, D. Jenson, N. Papanikolopoulos, Suav:q - a hybrid approach to solar-powered flight, *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 3288–3294 (2016).
86. G. Torrente, E. Kaufmann, P. Föhn, D. Scaramuzza, Data-driven mpc for quadrotors, *IEEE Robotics and Automation Letters* 3769–3776 (2021).
87. D. Falanga, K. Kleber, D. Scaramuzza, Dynamic obstacle avoidance for quadrotors with event cameras, *Science Robotics* **5** (2020).
88. G. Gallego, T. Delbruck, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. Davison, J. Conradt, K. Daniilidis, D. Scaramuzza, Event-based vision: A survey, *IEEE Trans. Pattern Anal. Machine Intell.* (2020).
89. M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C.-K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y.-H. Weng, A. Wild, Y. Yang, H. Wang, Loihi: A neuromorphic manycore processor with On-Chip learning, *IEEE Micro* 82–99 (2018).
90. J. Dupeyroux, J. J. Hagenaars, F. Paredes-Vallés, G. C. de Croon, Neuromorphic control for optic-flow-based landing of mavs using the loihi processor, *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 96–102 (IEEE, 2021).
91. A. Vitale1, A. Renner, C. Nauer, D. Scaramuzza, Y. Sandamirskaya, Event-driven vision and control for uavs on a neuromorphic chip, *IEEE Int. Conf. Robot. Autom. (ICRA)* (2021).
92. S. Moradi, N. Qiao, F. Stefanini, G. Indiveri, A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs), *IEEE Trans. Biomed. Circuits Syst.* 106–122 (2018).
93. M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems (2015). Software available from tensorflow.org.
94. A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, R. Garnett, eds. (Curran Associates, Inc., 2019), 8024–8035.
95. Connect Tech Inc., Quasar carrier board, https://connecttech.com/product/quasar-carrier-nvidia-jetson-tx2/ (2019). Accessed: 2021-7-20.
96. M. Luessi, radix, https://www.brainfpv.com/product/radix-fc/ (2017). Accessed: 2021-7-20.
97. The Apache Software Foundation, NuttX, https://nuttx.apache.org/. Accessed: 2021-7-20.
98. S. Shah, D. Dey, C. Lovett, A. Kapoor, Airsim: High-fidelity visual and physical simulation for autonomous vehicles, *Field and service robotics*, 621–635 (Springer, 2018).
99. O. Ben-Kiki, C. Evans, I. D. Net, YAML ain't markup language (YAML™) version 1.2, https://yaml.org/spec/1.2/spec.pdf (2009). Accessed: 2021-7-20.
100. D. Mellinger, V. Kumar, Minimum snap trajectory generation and control for quadrotors, *IEEE Int. Conf. Robot. Autom. (ICRA)*, 2520–2525 (2011).
101. D. Hanover, E. Kaufmann, P. Foehn, D. Scaramuzza, Performance, precision, and payloads: Adaptive optimal control for quadrotors under uncertainty, *IEEE Robot. Autom. Lett.* (2021).