

Learning High-Speed Flight in the Wild

ANTONIO LOQUERCIO^{1,†,*}, ELIA KAUFMANN^{1,†}, RENÉ RANFTL², MATTHIAS MÜLLER², VLADLEN KOLTUN³, AND DAVIDE SCARAMUZZA¹

¹Robotics and Perception Group, UZH, Zurich, Switzerland

²Intelligent Systems Lab, Intel, Munich, Germany

³Intelligent Systems Lab, Intel, Santa Clara, CA, USA

[†]These authors contributed equally to this work

*Corresponding author: loquercio@ifi.uzh.ch

This is the accepted version of *Science Robotics* Vol. 6, Issue 59, abg5810 (2021)

DOI: 10.1126/scirobotics.abg5810

arXiv:2110.05113v1 [cs.RO] 11 Oct 2021

Quadruplets are agile. Unlike most other machines, they can traverse extremely complex environments at high speeds. To date, only expert human pilots have been able to fully exploit their capabilities. Autonomous operation with onboard sensing and computation has been limited to low speeds. State-of-the-art methods generally separate the navigation problem into subtasks: sensing, mapping, and planning. Although this approach has proven successful at low speeds, the separation it builds upon can be problematic for high-speed navigation in cluttered environments. The subtasks are executed sequentially, leading to increased processing latency and a compounding of errors through the pipeline. Here we propose an end-to-end approach that can autonomously fly quadruplets through complex natural and human-made environments at high speeds, with purely onboard sensing and computation. The key principle is to directly map noisy sensory observations to collision-free trajectories in a receding-horizon fashion. This direct mapping drastically reduces processing latency and increases robustness to noisy and incomplete perception. The sensorimotor mapping is performed by a convolutional network that is trained exclusively in simulation via privileged learning: imitating an expert with access to privileged information. By simulating realistic sensor noise, our approach achieves zero-shot transfer from simulation to challenging real-world environments that were never experienced during training: dense forests, snow-covered terrain, derailed trains, and collapsed buildings. Our work demonstrates that end-to-end policies trained in simulation enable high-speed autonomous flight through challenging environments, outperforming traditional obstacle avoidance pipelines.

CODE AND MULTIMEDIA MATERIAL

A video demonstrating our approach is available at <https://youtu.be/m89bNn6RFoQ>. We publicly release code and training datasets at https://github.com/uzh-rpg/agile_autonomy.

1. INTRODUCTION

Quadruplets are among the most agile and dynamic machines ever created¹ [1, 2]. Thanks to their agility, they can traverse complex environments, ranging from cluttered forests to urban canyons, and reach locations that are otherwise inaccessible to humans and machines alike. This ability has led to their application in fields such as search and rescue, logistics, security, infrastructure, entertainment, and agriculture [3]. In the majority of these existing applications, the quadruplet needs to be controlled by expert human pilots, who take years to train, and are thus an expensive and scarce resource. Infusing quadruplets with autonomy, that is the capability to safely operate in the world

without the need for human intervention, has the potential to massively enhance their usefulness and to revolutionize whole industries. However, the development of autonomous quadruplets that can navigate in complex environments with the agility and safety of expert human pilots or birds is a long-standing challenge that is still open.

The limiting factor for autonomous agile flight in arbitrary unknown environments is the coupling of fast and robust perception with effective planning. The perception system has to be robust to disturbances such as sensor noise, motion blur, and changing illumination conditions. In addition, an effective planner is necessary to find a path that is both dynamically feasible and collision-free while relying only on noisy and partial observations of the environment. These requirements, together with the limited computational resources that are available onboard, make it difficult to achieve reliable perception and planning at low latency and high speeds [4].

Various approaches to enable autonomous flight have been proposed in the literature. Some works tackle only perception and build high-quality maps from imperfect measurements [5–9], whereas others focus on planning without considering perception errors [10–13]. Numerous systems that combine online mapping with traditional planning

¹The quadruplet used for the experiments in this paper has a maximum acceleration of 4g. Formula 1 cars achieve accelerations of up to 1.45g, and the Eurofighter Typhoon reaches a longitudinal acceleration of up to 1.15g.

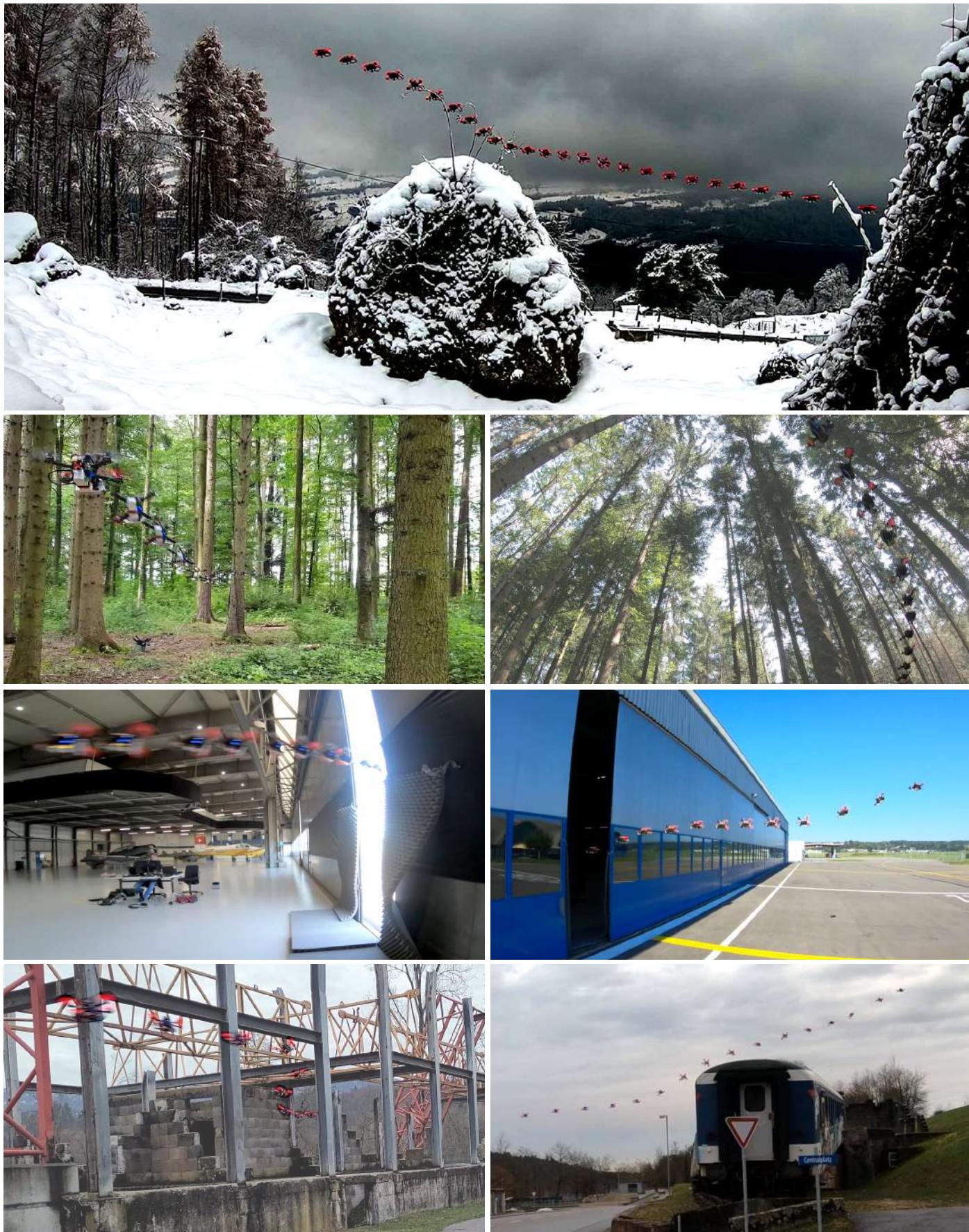


Fig. 1. Time-lapse illustrations of agile navigation at speeds between 5 and 10 m/s in a variety of environments. From top to bottom: A mountain trail, a forest, an airplane hangar, and a disaster zone. The middle rows show the same environments, but from different viewpoints. The last row shows navigation through a collapsed building and above a derailed train. Please watch [Movie 1](#) to get a better sense of the speed and agility of our approach.

algorithms have been proposed to achieve autonomous flight in previously unknown environments [14–22]. A taxonomy of prior works is presented in Figure S5 in the Supplementary Materials.

The division of the navigation task into the mapping and planning subtasks is attractive from an engineering perspective, because it enables parallel progress on each component and makes the overall system interpretable. However, it leads to pipelines that largely neglect interactions between the different stages and thus compound errors [22]. Their sequential nature also introduces additional latency, making high-speed and agile maneuvers difficult to impossible [4]. While these issues can be mitigated to some degree by careful hand-tuning and engineering, the divide-and-conquer principle that has been prevalent in research on autonomous flight in unknown environments for many years imposes fundamental limits on the speed and agility that a robotic system can achieve [23].

In contrast to these traditional pipelines, some recent works propose to learn end-to-end policies directly from data without explicit mapping and planning stages [24–27]. These policies are trained by imitating a human [24, 27], from experience that was collected in simulation [25], or directly in the real world [26]. Because the number of samples required to train general navigation policies is very high, existing approaches impose constraints on the quadrotor’s motion model, for example by constraining the platform to planar motion [24, 26, 27] and/or discrete actions [25], at the cost of reduced maneuverability and agility. More recent work has demonstrated that very agile control policies can be trained in simulation [28]. Policies produced by the last approach can successfully perform acrobatic maneuvers, but can only operate in unobstructed free space and are essentially blind to obstacles in the environment.

Here we present an approach to fly a quadrotor at high speeds in a variety of environments with complex obstacle geometry (Figure 1 and Movie 1) while having access to only onboard sensing and computation. By predicting navigation commands directly from sensor measurements, we decrease the latency between perception and action while simultaneously being robust to perception artifacts, such as motion blur, missing data, and sensor noise. To deal with sample complexity and not endanger the physical platform, we train the policy *exclusively* in simulation. We leverage abstraction of the input data to transfer the policy from simulation to reality [28, 29]. To this end, we utilize a stereo matching algorithm to provide depth images as input to the policy. We show that this representation is both rich enough to safely navigate through complex environments and abstract enough to bridge simulation and reality. Our choice of input representation guarantees a strong similarity of the noise models between simulated and real observations and gives our policy robustness against common perceptual artifacts in existing depth sensors.

We train the navigation policy via privileged learning [30] on demonstrations that are provided by a sampling-based expert. Our expert has privileged access to a representation of the environment in the form of a three-dimensional (3D) point cloud as well as perfect knowledge about the state of the quadrotor. Since simulation does not impose real-time constraints, the expert additionally has an unconstrained computational budget. While existing global planning algorithms [10, 12, 13] generally output a single trajectory, our expert uses Metropolis-Hastings (M-H) sampling to compute a *distribution* of collision-free trajectories. This captures the multi-modal nature of the navigation task where many equally valid solutions can exist (for example, going either left or right around an obstacle). We therefore use our planner to compute trajectories with a short time horizon to ensure that they are predictable from onboard sensors and that the sampler remains computationally tractable. We bias the sampler toward obstacle-free regions by conditioning it on trajectories from a classic global planning algorithm [13].

We also reflect the multi-modal nature of the problem in the design and training of the neural network policy. Our policy takes a noisy depth image and inertial measurements as sensory inputs and produces a set of short-term trajectories together with an estimate of individual trajectory costs. The trajectories are represented as high-order polynomials to ensure dynamical feasibility. We train the policy using a multi-hypothesis winner-takes-all loss that adaptively maps the predicted trajectories to the best trajectories that have been found by the sampling-based expert. At test time, we use the predicted trajectory costs to decide which trajectory to execute in a receding horizon. The policy network is designed to be extremely lightweight, which ensures that it can be executed onboard the quadrotor at the update rates required for high-speed flight.

The resulting policy can fly a physical quadrotor in natural and human-made environments at speeds that are unreachable by existing methods. We achieve this in a zero-shot generalization setting: we train on randomly generated obstacle courses composed of simple off-the-shelf objects, such as schematic trees and a small set of convex shapes such as cylinders and cubes. We then directly deploy the policy in the physical world without any adaptation or fine-tuning. Our platform experiences conditions at test time that were never seen during training. Examples include high dynamic range (when flying from indoor environments to outdoor environments), poorly textured surfaces (indoor environments and snow-covered terrain), thick vegetation in forests, and the irregular and complex layout of a disaster scenario (Figure 2). These results suggest that our methodology enables a multitude of applications that rely on agile autonomous drones with purely onboard sensing and computation.

2. RESULTS

Our experiments in simulation show that the proposed approach reduces the failure rate up to 10 times with respect to state-of-the-art methods. We confirm our results in a variety of real-world environments using a custom-built physical quadrotor; we deploy our policy trained in simulation without any further adaptations. In all experiments, the drone was provided with a reference trajectory, which is not collision-free (Figure 3-C, depicted in red), to encode the intended flight path. This reference can be provided by a user or a higher-level planning algorithm. The drone is tasked to follow that flight path and make adjustments as necessary to avoid obstacles. Recordings of the experiments can be found in Movie 1.

A. High-Speed Flight in the Wild

We tested our approach in diverse real-world environments, as illustrated in Figures 1 and 2. High-speed flight in these environments is very challenging because of their complex structure (*e.g.* thick vegetation, thin branches, or collapsed walls) and multiple options available to avoid obstacles. In addition, a high-level understanding of the environment is necessary, for example to pass through far-away openings (Figure 3-C) or nearby obstacles (Figure 2-M). Flying at high speeds also necessitates low-latency perception and robustness to sensor noise, which is worsened by challenging illumination conditions and low-texture surfaces (*e.g.* because of snow). At the same time, only limited computational resources are available onboard. Despite all of these challenges, our approach was able to successfully navigate in all environments that it was tested in. Note that our policy was trained in simulation and was never exposed to any of these environments or conditions at training time.

We measure performance according to success rate, *i.e.* the percentage of successful runs over the total number of runs, where we consider a run successful if the drone reaches the goal location within a radius of 5 m without crashing. We performed a total of 56 experiments at

(A) Thin trees



(B) Thick trees



(C) Vegetation



(D) Boulder



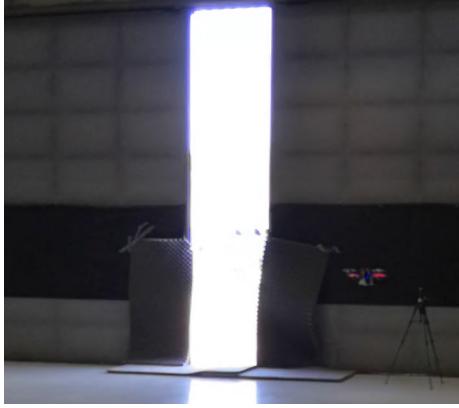
(E) Branches



(F) Rock formation



(H) Narrow gap



(I) Crane



(L) Building



(M) Collapsed building



(N) Train



(O) Ruins



Fig. 2. Testing Environments. Zero-shot generalization of our approach in complex natural (A to F) and human-made (H to O) environments. The encountered obstacles can often be avoided in multiple directions and have very different size and structure.

different speeds. Overall, our approach was always able to achieve the goal when not colliding into an obstacle. We report the cumulative and individual success rates at various speeds in Figure 3 (D and E). Our experimental platform performs state estimation and depth perception onboard using vision-based sensors. A detailed description of the platform is available in section S3. We group the environments that were used for experiments into two classes, *natural* and *human-made*, and highlight the distinct challenges that these types of environments present for agile autonomous flight.

Natural environments

We performed experiments in diverse natural environments: forests of different types and densities and steep snowy mountain terrains. Figures 2 (A and F) illustrate the heterogeneity of those environments. We performed experiments with two different reference trajectories: a 40 m-long straight line and a circle with a 6 m radius (Figure 3-A). Both trajectory types are not collision-free and would lead to a crash into obstacles if blindly executed. We flew the straight line at different average speeds in the range of 3 to 10 m s^{-1} . Flying at these speeds requires very precise and low-latency control of position and attitude to avoid bushes and pass through the openings between trees and branches (Figure 3-B). Traditional mapping and planning approaches generally fail to achieve high speeds in such conditions, as both the thick vegetation and the texture-less snow terrain often cause very noisy depth observations.

We conducted a total of 31 experiments in natural environments. At average speeds of 3 and 5 m s^{-1} our approach consistently completed the task without experiencing a single crash. For comparison, state-of-the-art methods with comparable actuation, sensing, and computation [18, 31] achieve in similar environments a maximum average speed of 2.29 m s^{-1} . To study the limit of the system, we set the platform's average speed to 7 m s^{-1} . Despite the very high-speed, the maneuver was successfully completed in 8 out of 10 experiments. The two failures happened when objects entered the field of view very late because of the high angular velocity of the platform. Given the good performance at 7 m s^{-1} , we push the average flight speed even higher to 10 m s^{-1} . At this speed, external disturbances, *e.g.* aerodynamics, battery power drops, and motion-blur start to play an important role and widen the simulation to reality gap. Nonetheless, we achieve a success rate of 60%, with failures mainly happening in the proximity of narrow openings less than a meter wide, where a single wrong action results in a crash.

Human-made environments

We also tested our approach in a set of human-made environments, illustrated in Figure 2 (G to O). In these environments, the drone faces a different set of challenges. It has to avoid obstacles with a variety of sizes and shapes (*e.g.* a train, a crane, a building, and ruins), slalom through concrete structures (c.f. Figure 2M), and exit a building through a single narrow opening (c.f. Figure 2H). The irregular and/or large structure of the encountered obstacles, the limited number of flyable openings, and the requirement to initiate the avoidance maneuver well in advance, offer a complementary set of challenges with respect to our natural testing environments.

As in the natural environments, we provide the drone with a straight reference trajectory with length of 40 m. The reference trajectory is in direct collision with obstacles and its blind execution would result in a crash. We performed a total of 19 experiments with flight speeds in the range of 3 to 7 m s^{-1} . Given the lower success rate experienced in the forest environment at 10 m s^{-1} , we did not test at higher speeds to avoid fatal crashes. As shown in Figure 3E, our approach is robust to zero-shot deployment in these environments, whose characteristics

were never observed at training time, and consistently completes the task without crashing.

We further compare our approach to a commercial drone². Specifically, the drones are required to exit a hangar by passing through a narrow gap of about 0.8 m in width (Figure 3-C). At the start of the experiment, the drones are placed at about 10 m in front of and about 5 m to the right of the gap. The task was represented by a straight reference trajectory passing through the wall (Figure 3C, in red). This experiment is challenging since it requires a high-level understanding of the environment to turn early enough toward the gap. The commercial drone, flying at a speed of about 2.7 m s^{-1} consistently failed to pass through the gap across three experiments. In two of the experiments it deemed the gap to be too small and stopped in front of it; in the third, it crashed into the wall. In contrast, the low latency and robustness to perception failures of our approach enabled the drone to successfully fly through the narrow gap every time. We performed a total of six experiments at flight speeds of 3 and 5 m s^{-1} and never experienced a crash.

B. Controlled experiments

We perform a set of controlled experiments in simulation to compare the performance of our approach with several baselines. We select two representative state-of-the-art approaches as baselines for navigation in unknown environments: the mapping and planning method of Zhou et al. [18] (*FastPlanner*) and the reactive planner of Florence et al. [32] (*Reactive*). The first approach [18] incrementally builds a map from a series of observations and plans a trajectory in this map to reach the goal while avoiding obstacles. In contrast, the second approach [32] does not build a map but uses instantaneous depth information to select the best trajectory from a set of pre-defined motion primitives based on a cost that encodes collision and progress toward the goal. The baselines receive the same input as our policy: the state of the platform, depth measurements from the stereo camera, and a goal in the form of a reference state that lies 5 s in the future. To provide a notion of the difficulty of the environments, we additionally show a naive baseline that blindly follows the reference trajectory to the goal without avoiding obstacles (*Blind*). As in the real-world experiments, we compare the different approaches according to their success rate, which measures how often the drone reaches the goal location within a radius of 5 m without crashing. We perform all experiments in the Flightmare simulator [33] using the RotorS [34] Gazebo plugin for accurate physics modeling and Unity as a rendering engine [35]. The experiments are conducted in four different environments: a forest, a narrow gap, a disaster scenario, and a city street. Those environments offer different challenges: from high-density but simple obstacle geometry in the forest, to complex obstacle geometries in the urban scenarios. Sample observations collected from these environments are available in Figure S4 and Movie S2. These environments are available out-of-the-box from the [Unity Asset store](#).

The results of these experiments are summarized in Figure 4. Throughout all environments, a similar pattern can be observed. At low speeds (3 m s^{-1}) all methods perform similarly. However, as the speed increases, the baselines' performances quickly drop: already at 5 m s^{-1} , no baseline is able to complete all runs without crashing. In contrast, our method can reliably fly at high-speeds through all environments, achieving an average success rate of 70% at 10 m s^{-1} . The drop in performance of the two baselines can be justified as follows. Although the reactive baseline has a low processing latency, yet it has a limited expressive power: the trajectory to be executed can only be selected from a relatively small set of primitives. This set is usually

²The commercial drone (a Skydio R1) was tasked to follow a person running through the narrow gap. The speed of this drone cannot be enforced nor controlled, and was therefore estimated in post-processing.

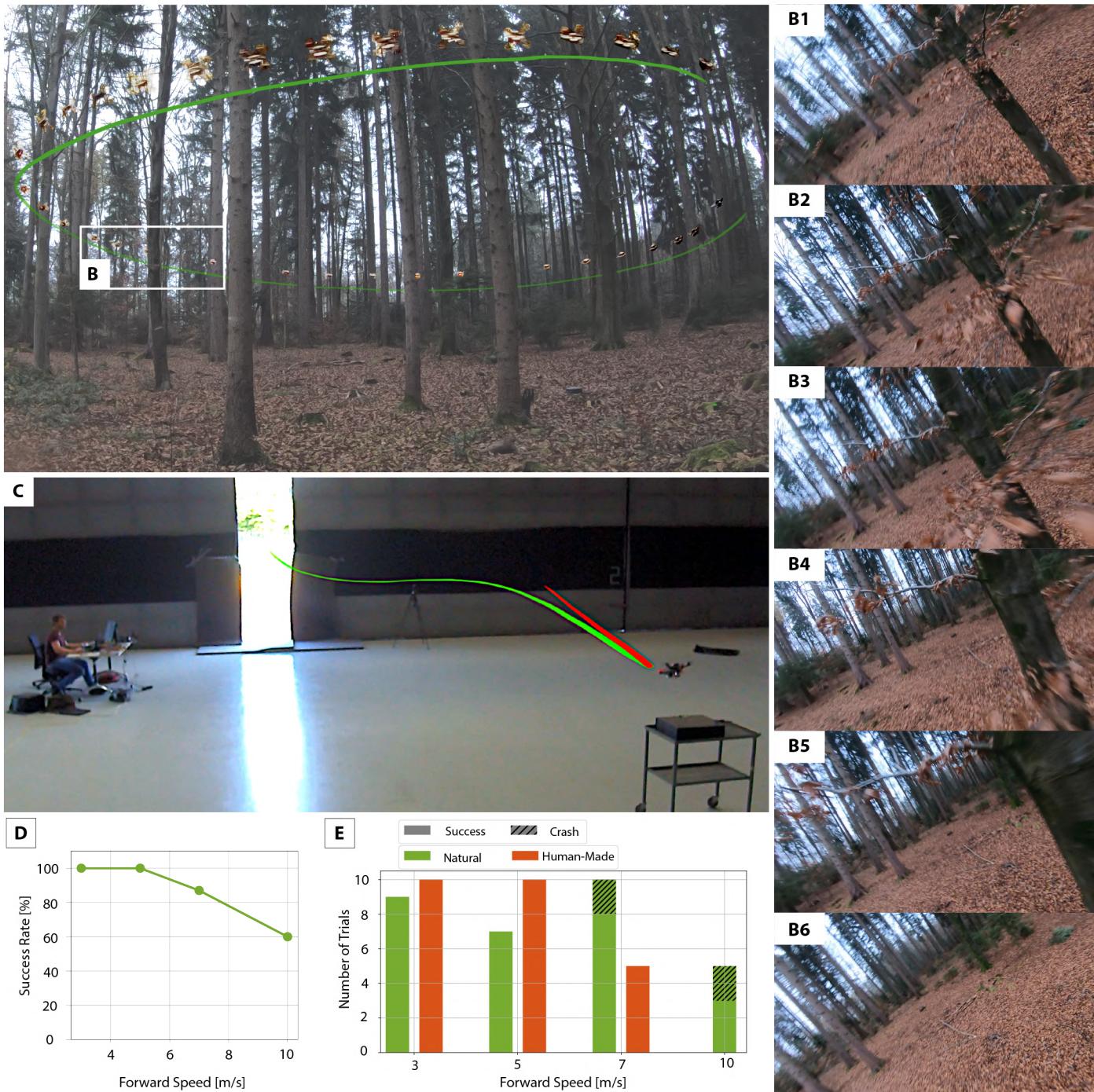


Fig. 3. Evaluation in indoor and outdoor environments. (A) A circular path in the forest at an average speed of 7 m s^{-1} . (B) Sequence of first-person views from the maneuver in A, observed during the avoidance of tree branches. The maneuver requires fine and fast adaptations in height (**B2, B5**) and attitude (**B3, B4**) to avoid the vegetation. After the obstacle is passed, the drone accelerates in the direction of travel (**B6**). (C) Comparison of reference trajectory passing through a wall (in red), to actual flight path (in green) in an airplane hangar. (D) Success rates for all experiments aggregated according to flight speed. (E) Number of trials per environment class at different speeds.

too small to account for all possible obstacle configurations and relative speeds between obstacles and drone, which are observed during high-speed flight. In addition, being the reactive baseline only conditioned on the current observation, it is strongly affected by noise in the observation. In contrast, the FastPlanner baseline can reject outliers in the depth map by leveraging multiple observations, which makes it more robust to sensing errors. However, this method generally results in higher processing latency: Multiple observations are required to add obstacles in the map, and, therefore, to plan trajectories to avoid them. This problem is worsened by high-speed motion, which generally results in little overlap between consecutive observations. By contrast, our approach has very low processing latency (cf. section C). In addition, the data-driven approach allows leveraging regularities in the data, which makes it more robust to sensor noise.

We thoroughly analyze those aspects in controlled studies on latency and sensor noise, which are shown in section C and section D, and additionally study the robustness of our method to noise in estimation and control in section S1. A video demonstrating the performance of our method in the simulation environments can be found in [Movie S2](#). We now describe the specific characteristics of all environments and their respective experimental results in detail.

Forest

We build a simulated forest [36] in a rectangular region $R(l, w)$ of width $w = 30\text{ m}$ and length $l = 60\text{ m}$, and fill it with trees that have a diameter of about 0.6 m . Trees are randomly placed according to a homogeneous Poisson point process P with intensity $\delta = 1/25 \text{ tree m}^{-2}$ [36]. As it is possible to see in Figure 4, the resulting forest is very dense. We provide the drone with a straight reference trajectory of 40 m length, and vary the average forward speed of the drone between 3 and 10 m s^{-1} . We repeat the experiments with 10 different random realizations of the forest, using the same random seed for all baselines. At a low speed of 3 m s^{-1} , all methods successfully complete every run. As speed increases, the success rates of the baselines quickly degrade. At 10 m s^{-1} , no baseline completes even a single run successfully. In contrast, our approach is very robust at higher speeds. It achieves 100% success rate up to 5 m s^{-1} . At the highest speed of 10 m s^{-1} , our approach still keeps a success rate of 60%. An additional study of performance with respect to tree density δ is available in section S2.

Narrow gap

We then mimic in simulation the real-world narrow gap experiment. We render a 40 m long wall with a single opening, 10 m in front of the drone. The gap is placed at a randomized lateral offset in the range of $[-5, 5]\text{ m}$ with respect to the starting location. The width of the opening is also randomized uniformly between 0.8 m and 1.0 m . All experiments are repeated 10 times with different opening sizes and lateral gap offsets for each speed. An illustration of the setup and the results of the evaluation are shown in Figure 4. The reactive and blind baselines consistently fail to solve this task, while FastPlanner has a success rate of up to 50% at 5 m s^{-1} , but still consistently fails at 7 m s^{-1} . We observe that the baselines adapt their trajectory only when being close to the wall, which is often too late to correct course toward the opening. Conversely, our approach always completes the task successfully at speeds of up to 5 m s^{-1} . Even at a speed of 7 m s^{-1} it only fails in 2 of 10 runs.

Disaster and urban scenario

We further test our method in a simulated disaster zone and an urban city street. These environments are challenging since they contain obstacles with complex geometry. For instance, the former contains, among others, collapsed buildings, crashed cars, and iron fences, while

the latter features street lamp poles, parked vehicles, and traffic signs. Some observations from these environments are shown in Figure S4. We start the drone from a random location in the map, and provide it with a straight line reference of 40 m in length. For each starting location, the reference is guaranteed to be in collision with at least one obstacle. We vary the average flight speed from 3 m s^{-1} to 10 m s^{-1} . The results of these experiments as well as the pointclouds from one of the starting locations are shown in Figure 4. Because of the complex obstacle geometries, even at a low speed of 3 m s^{-1} the mapping baseline does not achieve 100% success rate. The reactive baseline performs better overall, with a success rate of up to 90% at 5 m s^{-1} . However, its performance quickly degrades at higher speeds, with as much as 8 of 10 failures at 10 m s^{-1} . Similarly to the narrow-gap experiments, we observe that the baselines only adapt their motion when very close to obstacles, which is often too late to avoid collision in presence of large obstacles with complex shapes. In contrast to the baselines, our approach is much more robust at higher speeds, with a success rate of up to 100% at 7 m s^{-1} . Even at a speed of 10 m s^{-1} it only fails in 2 of 10 runs in the urban environment. The ability of our approach to combine long-term and short-term planning is crucial to achieving this performance, since the drone must steer early enough toward free space and at the same time perform small reactive corrections to avoid a collision. This ability, in addition to the low latency and robustness to sensor noise, gives our approach a substantial performance advantage with respect to the baselines in these challenging environments.

C. Computational cost

In this section, we compare the computational cost of our algorithm with the baselines. Table 1 shows the results of this evaluation. It highlights how each step of the methods contributes to the overall processing latency. All timings were recorded on a desktop computer with a 6-core i7-8700 central processing unit (CPU), which was also used to run the simulation experiments. To ensure a fair comparison, we report the timings when using only the CPU for all approaches. We also report the timings of our approach when performing neural network inference on a GeForce RTX 2080 graphics processing unit (GPU), because accelerators can be used with our approach without any extra effort. To paint a complete and realistic picture, we additionally evaluate the timing of our algorithm on the onboard computer of the quadrotor which is a Jetson TX2.

With a total computation time of 65.2 ms per frame, FastPlanner incurs the highest processing latency. It is important to note that the temporal filtering operations that are necessary to cope with sensing errors effectively make perception even slower. Two to three observations of an obstacle can be required to add it to the map, which increases the effective latency of the system. By foregoing the mapping stage altogether, the *Reactive* baseline significantly reduces computation time. This baseline is about three times faster than *FastPlanner*, with a total processing latency of 19.1 ms . However, the reduced processing latency comes at the cost of the trajectory complexity that can be represented, since the planner can only select primitives from a pre-defined library. In addition, the reactive baseline is sensitive to sensing errors, which can drastically affect performance at high speeds.

Our approach has significantly lower processing latency than both baselines; when network inference is performed on the GPU, our approach is 25.3 times faster than *FastPlanner* and 7.4 times faster than the *Reactive* baseline. When GPU inference is disabled, the network's latency increases by only 8 ms , and our approach is still much faster than both baselines. Moving from the desktop computer to the onboard embedded computing device, the network's forward pass requires 38.9 ms . Onboard, the total time to pass from the sensor reading to a plan is 41.6 ms , which corresponds to an update rate of

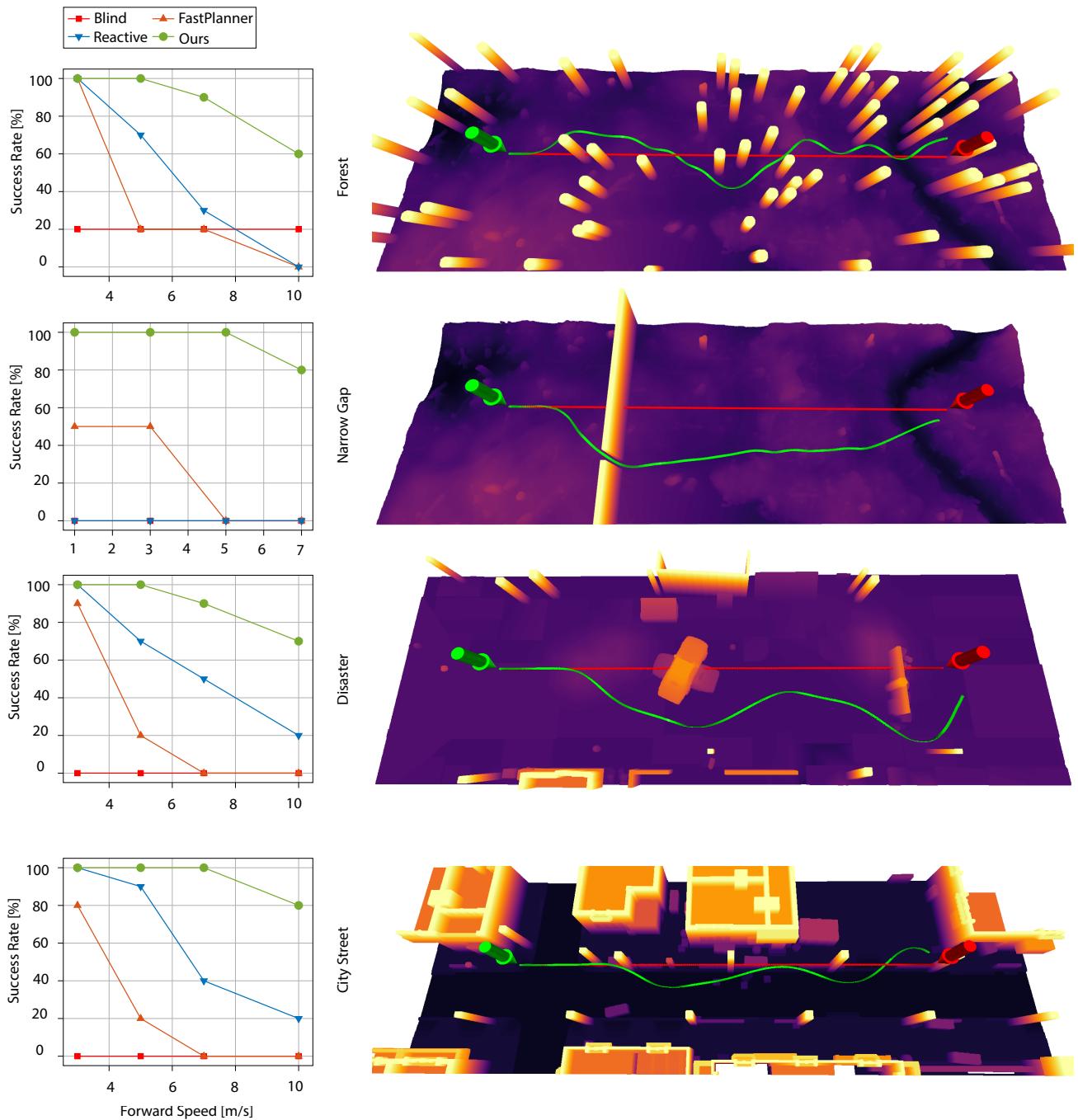


Fig. 4. Experiments in a variety of simulated environments. The left column reports success rates at various speeds. The right column shows a pointcloud of the environment together with paths taken by different policies from start (green arrow) to end (red arrow). The paths illustrate the blind policy (red) and the path taken by our approach (green). Our approach consistently outperforms the baselines in all environments and at all speeds. Sample observations from these environments are shown in Figure S4.

Method	Components	μ [ms]	σ [ms]	Perc. [%]	Total Proc. Latency [ms]
FastPlanner [18]	Pre-processing	14.6	2.3	22.3	65.2
	Mapping	49.2	8.7	75.5	
	Planning	1.4	1.6	2.2	
Reactive [32]	Pre-processing	13.8	1.3	72.3	19.1
	Planning	5.3	0.9	27.7	
Ours	Pre-processing	0.1	0.04	3.9	10.3 (2.6*)
	NN inference	10.1 (2.4*)	1.5 (0.8*)	93.0	
	Projection	0.08	0.01	3.1	
Ours (Onboard)	Pre-processing	0.2	0.1	0.4	41.6
	NN inference	38.9	4.5	93.6	
	Projection	2.5	1.5	6.0	

Table 1. Processing latency (μ) on a desktop computer equipped with a hexacore i7-8700 CPU and a GeForce RTX 2080 GPU. The standard deviation σ is computed over 1000 samples. For our approach, we report the computation time on the CPU and GPU (marked with *) on the desktop computer (*Ours*), as well as the computation time on the onboard computation unit Jetson TX2, (*Ours Onboard*). For the FastPlanner [18] and Reactive [32] baselines pre-processing represents the time to build a pointcloud from the depth image, while for our method pre-processing is the time to convert depth into an input tensor for the neural network. More details on the subtasks division are available in the Appendix in section S4. The proposed methodology is significantly faster than prior works.

about 24 Hz.

D. The effect of latency and sensor noise

We analyze the effect of sensor noise and planning latency in a controlled experiment. In this experiment, the quadrotor travels along a straight line at a constant forward speed and is required to laterally evade a single obstacle (a pole) while having only limited sensing range. This experimental setup was proposed in Falanga et al. [4] to understand the role of perception latency on the navigation ability of a robotic system subject to bounded inputs. Specifically, the authors derived an upper-bound for the forward speed at which a robot can fly and avoid a single obstacle as a function of perception latency and sensing range. They modeled the robot as a point-mass, which is a limited approximation for a quadrotor as it neglects the platform’s rotational dynamics. We thus extend their formulation to account for the latency introduced by the rotational motion that is necessary to avoid the obstacle. A detailed description of the formulation can be found in the Supplementary Materials in section S5.

We set up the experiment by placing a quadrotor with an initial forward velocity v at a distance of 6 m from a pole with diameter 1.5 m. The quadrotor is modeled as a sphere with radius 0.2 m. According to our formulation, we compute a theoretical maximum speed—*i.e.* the speed at which the task is no longer feasible—for each method. The maximum speed depends on the sensing range, *i.e.* how far can an obstacle be accurately perceived, the latency of the visual sensor, *i.e.* the inverse of the frame rate, and the processing latency, *i.e.* the time to convert an observation into motor commands. Note that the latter varies between methods (c.f. section C), and therefore the theoretical maximum speed differs for each approach. For the details of the calculation, we refer the reader to section S5 in the Supplementary Materials. We then perform the controlled experiment with varying forward speeds v in the range of 3 to 13 m s⁻¹. We perform 10 experiments for each speed with all approaches and report the success rate. We run the experiment in two settings: (i) with ground-truth depth

information, to isolate the effect of latency on performance, and (ii) with depth estimated by stereo matching [37] to analyze the effect of sensing errors to performance.

Ground-truth depth

Figure 5B1 illustrates the results of this experiment when perfect depth perception (Figure 5A1) is available. All approaches can complete the task perfectly up to 5 m s⁻¹. However, even in these ideal conditions, the performance of the baselines drops for speeds beyond 5 m s⁻¹. This drop in performance for *Reactive* can be attributed to the fact that the finite library of motion primitives does not contain maneuvers that are aggressive enough to complete this task. Similarly, the performance degrades for *FastPlanner* as a result of sub-optimal planning of actions. Although this baseline manages to map the obstacle in time, the planner frequently commands actions to stop the platform which leads to crashes when flying at high speeds. It is important to point out that the *FastPlanner* baseline was only demonstrated up to speeds of 3 m s⁻¹ in the original work [18], and thus was not designed to operate at high speeds. Our approach can successfully avoid the obstacle without a single failure up to 7 m s⁻¹. For higher speeds, performance gracefully degrades to 60% at 10 m s⁻¹. This decrease in performance can be attributed to the sensitivity to imperfect network predictions when flying at high speed, where a single wrong action can lead to a crash.

Estimated depth

While the previous experiments mainly focused on latency and the avoidance strategy, we now study the influence of imperfect sensory measurements on performance. We repeat the same experiment, but provide all methods with depth maps that have been computed from the stereo pairs (Figure 5A2). Figure 5B2 shows the results of this experiment. The baselines experience a significant drop in performance compared with when provided with perfect sensory readings. *FastPlanner* completely fails for speeds of 5 m s⁻¹ and beyond. This sharp drop in performance is because of the need for additional filtering of

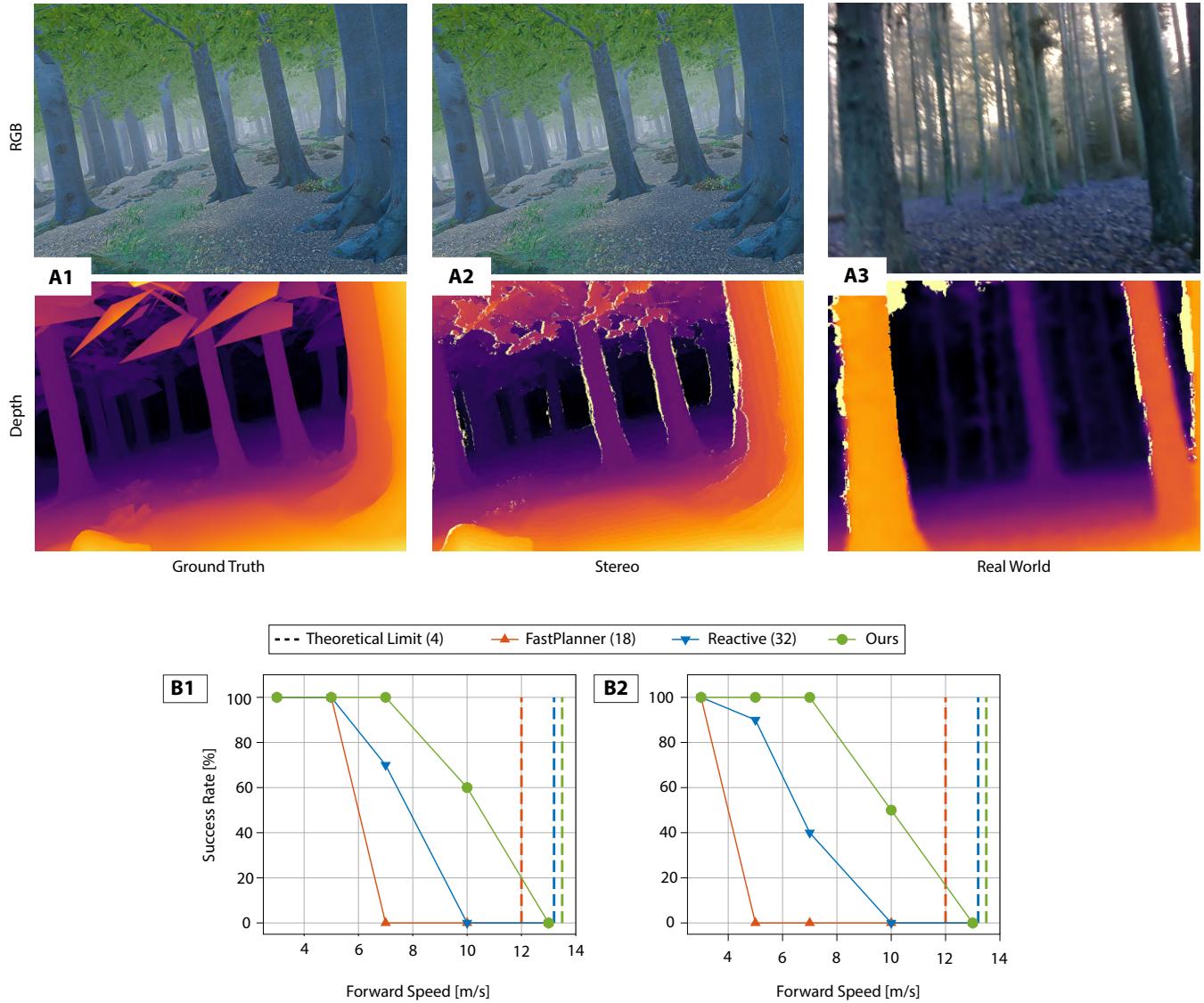


Fig. 5. The effect of sensor noise on performance. (A) When comparing RGB and depth images generated in simulation with images captured in the real world, we observe that the corresponding depth images are more similar than the RGB images. In addition, simulated depth estimated by stereo matching (A2) contains the typical failure cases of a depth sensor (A3), e.g. missing values and noise. (B) Results of the controlled experiment to study the effect of perception latency and noise on navigation ability. The experiment is performed on ground-truth depth (B1) and stereo depth (B2). Since each method has different processing latency (c.f. section C), the theoretical maximum speed differs between approaches (dashed lines). Our method can fly closer to the theoretical limit than the baselines and is only minimally affected by the noise of stereo depth estimates.

the noisy depth measurements that drastically increases the latency of the mapping stage. To reject outliers from the depth sensor, multiple observation (two/three) are required to completely map the obstacle. As a result, this baseline detects obstacles too late to be able to successfully evade them. Similarly, the performance of the *Reactive* baseline drops by 30% at 7 m s^{-1} . In contrast to the baselines, our approach is only marginally affected by the noisy depth readings, with only a 10% drop in performance at 10 m s^{-1} , but no change in performance at lower speeds. This is because our policy, trained on depth from stereo, learns to account for common issues in the data such as discretization artifacts and missing values.

3. DISCUSSION

Existing autonomous flight systems are highly engineered and modular. The navigation task is usually split into sensing, mapping, planning, and control. The separation into multiple modules simplifies the implementation of engineered systems, enables parallel development of each component, and makes the overall system more interpretable. However, modularity comes at a high cost: the communication between modules introduces latency, errors compound across modules, and interactions between modules are not modeled. In addition, it is an open question if certain subtasks, such as maintaining an explicit map of the environment, are even necessary for agile flight.

Our work replaces the traditional components of sensing, mapping, and planning with a single function that is represented by a neural network. This increases the system's robustness against sensor noise and reduces the processing latency. We demonstrate that our approach can reach speeds of up to 10 m s^{-1} in complex environments and reduces the failure rate at high speeds by up to 10 times when compared with the state of the art.

We achieve this by training a neural network to imitate an expert with privileged information in simulation. To cope with the complexity of the task and to enable seamless transfer from simulation to reality, we make several technical contributions. These include a sampling-based expert, a neural network architecture, and a training procedure, all of which take the task's multi-modality into account. We also use an abstract, but sufficiently rich input representation that considers real-world sensor noise. The combination of these innovations enables the training of robust navigation policies in simulation that can be directly transferred to diverse real-world environments without any fine-tuning on real data.

We see several opportunities for future work. Currently, the learned policy exhibits low success rates at average speeds of 10 m s^{-1} or higher in the real world. At these speeds even our expert policy, despite having perfect knowledge of the environment, often fails to find collision-free trajectories. This is mainly because of the fact that, at speeds of 10 m s^{-1} or higher, feasible solutions require temporal consistency over a long time horizon and strong variations of the instantaneous flying speed as a function of the obstacle density. This requirement makes feasible trajectories extremely sparse in parameter space, resulting in intractable sampling. Engineering a more complex expert to tackle this problem can be very challenging and might require specifically tailored heuristics to find approximate solutions. Therefore, we believe that this problem represents a big opportunity for model-free methods, which have the potential to ease the engineering requirements. The second reason for the performance drop at very high speeds is the mismatch between the simulated and physical drone in terms of dynamics and perception. The mismatch in dynamics is because of aerodynamics effects, motor delays, and dropping battery voltage. Therefore, we hypothesize that performance would benefit from increasing the fidelity of the simulated drone and making the policy robust to the unavoidable model mismatches. A third reason is

perception latency: Faster sensors can provide more information about the environment in a smaller amount of time and, therefore, can be used to provide more frequent updates, which improves the precision of the estimate. This could enable further reducing sensitivity to noise and promote a quicker understanding of the environment. Perception latency can possibly be reduced with event cameras [38], especially in the presence of dynamic obstacles [39].

Overall, our approach is a stepping stone toward the development of autonomous systems that can navigate at high speeds through previously unseen environments with only on-board sensing and computation. Combining our short-horizon controller for local obstacle avoidance with a long-term planner is a major opportunity for many robotics applications, including autonomous exploration, delivery, and cinematography.

4. MATERIALS AND METHODS

To perform agile flight through cluttered and previously-unseen environments, we train a sensorimotor policy to predict receding-horizon trajectories from on-board sensor measurements and a reference trajectory. We assume that the reference trajectory is provided by a higher-level planning algorithm or the user. The reference is not necessarily collision-free and only encodes the long-term goal of the platform. The agent is responsible to fly in the direction dictated by the reference while adapting its flight path to the environment.

An agent's observation \mathbf{o} consists of a depth image $\mathbf{d} \in \mathbb{R}^{640 \times 480}$, an estimate of the platform velocity $\mathbf{v} \in \mathbb{R}^3$ and attitude (expressed as a rotation matrix) $\mathbf{q} \in \mathbb{R}^9$, and a desired flight direction $\boldsymbol{\omega} \in \mathbb{R}^3$. The policy output is a set of motion hypotheses which are represented as receding-horizon trajectories with the corresponding estimated risk of collision. A model predictive controller then tracks the trajectory with the lowest collision probability and input cost. The controller is trained using privileged learning [30]. Specifically, the policy is trained on demonstrations provided by a sampling-based planner that has access to information that is not available to the sensorimotor student: the precise knowledge of the 3D structure of the environment and the exact platform state. Figure 6 shows an overview of our method.

We collect demonstrations and perform training entirely in simulation. This trivially facilitates access to perfect 3D and state data for the privileged expert, enables the synthesis of unlimited training samples for any desired trajectory, and does not put the physical platform in danger. We use the Flightmare simulator [33] with the RotorS [34] Gazebo plugin and Unity as a rendering engine [35]. Both the training and the testing environments are built by adding obstacles to the uneven ground of an out-of-the-box Unity environment.

To enable zero-shot transfer to real environments, special care has to be taken to minimize the domain shift of the (visual) input modalities from simulation to reality. To this end we use depth as an abstract input representation that shows only a negligible domain shift from simulation to the real world (cf. Figure 5A). Specifically, the sensorimotor agent is trained with depth images that have been computed using Semi-Global Matching (SGM) [37] from a simulated stereo camera pair. As off-the-shelf depth sensors, such as the Intel RealSense 435 camera used on our physical platform, compute depth from stereo images using similar principles [40], this strategy ensures that the characteristics of the input aligns between simulation and the real world. As a result, the trained sensorimotor agent can be directly deployed in the real world. The next section presents both the privileged expert and the sensorimotor agent in detail.

A. The privileged expert

Our privileged expert is a sampling-based motion planning algorithm. The expert has perfect knowledge of the platform state and the en-

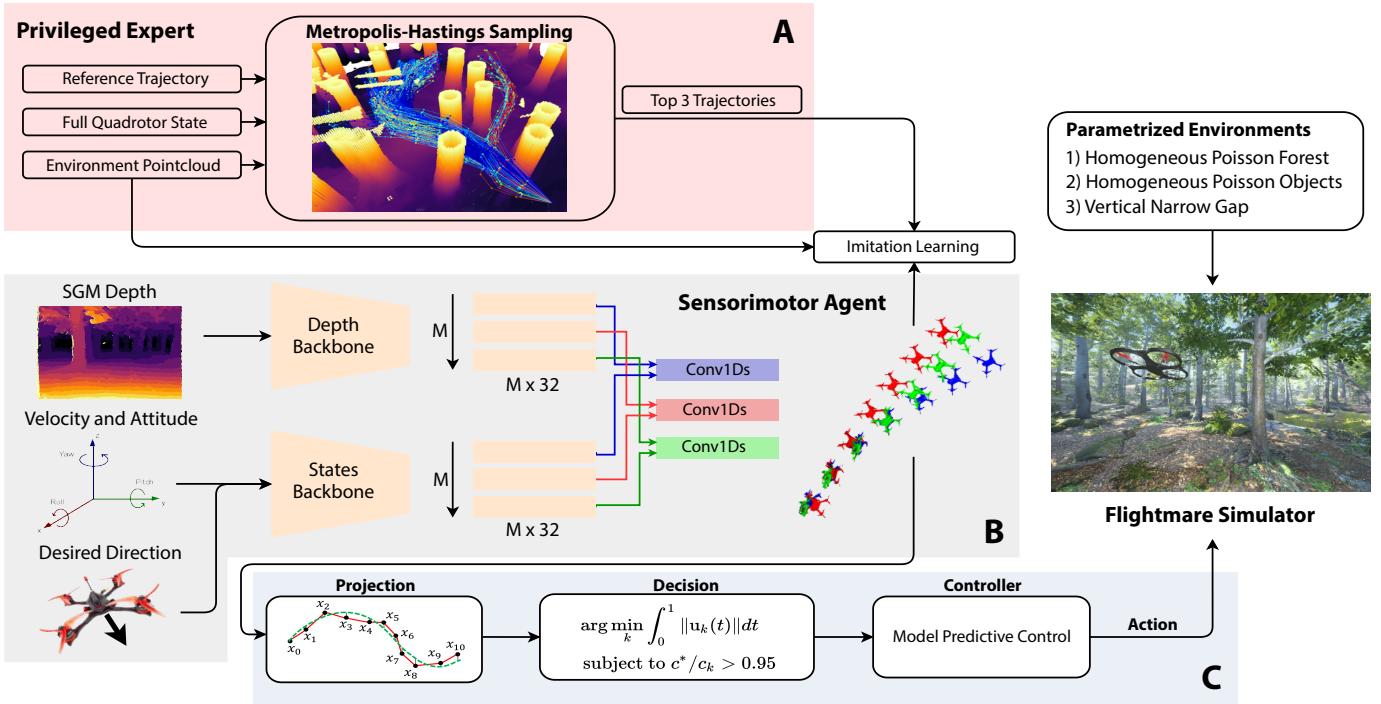


Fig. 6. Method overview. (A) Our offline planning algorithm computes a distribution of collision-free trajectories to follow a reference trajectory. The trajectories are computed with Metropolis-Hastings sampling and are conditioned on complete 3D knowledge of the environment, which is represented by a point cloud. (B) A sensorimotor agent is trained with imitation learning to predict the best three trajectories from the estimated depth, the drone’s velocity and attitude, and the desired direction that encodes the goal. (C) The predictions are projected on the space of polynomial trajectories and ranked according to their predicted collision cost c_k . The trajectory with the lowest predicted cost c_k is then tracked with a model predictive controller. If multiple trajectories have similar predicted cost (within a 5% range of the minimum $c^* = \min c_k$), the one with the smallest actuation cost is used.

vironment (complete 3D map), both of which are only available in simulation. The expert generates a set of collision-free trajectories τ representing the desired state of the quadrotor $x_{des} \in \mathbb{R}^{13}$ over the next second, starting from the current state of the drone, i.e. $\tau(0) = x$. To do so, it samples from a probability distribution P that encodes distance from obstacles and proximity to the reference trajectory. Specifically, the distribution of collision-free trajectories $P(\tau | \tau_{ref}, \mathcal{C})$ is conditioned on the reference trajectory τ_{ref} and the structure of the environment in the form of a point cloud $\mathcal{C} \in \mathbb{R}^{n \times 3}$. According to P , the probability of a trajectory τ is large if far from obstacles and close to the reference τ_{ref} . We define P as the following:

$$P(\tau | \tau_{ref}, \mathcal{C}) = \frac{1}{Z} \exp(-c(\tau, \tau_{ref}, \mathcal{C})) \quad (1)$$

where $Z = \int_{\tau} P(\tau | \tau_{ref}, \mathcal{C})$ is the normalization factor and $c(\tau, \tau_{ref}, \mathcal{C}) \in \mathbb{R}_+$ is a cost function indicating proximity to the reference and distance from obstacles. We define the trajectory cost function as

$$c(\tau, \tau_{ref}, \mathcal{C}) = \int_0^1 \lambda_c C_{collision}(\tau(t)) + \int_0^1 [\tau(t) - \tau_{ref}(t)]^\top Q [\tau(t) - \tau_{ref}(t)] dt \quad (2)$$

where $\lambda_c = 1000$, Q is a positive semidefinite state cost matrix, and $C_{collision}$ is a measure of the distance of the quadrotor to the points in \mathcal{C} . We model the quadrotor as a sphere of radius $r_q = 0.2$ m and define the collision cost as a truncated quadratic function of d_c , i.e. the distance

between the quadrotor and the closest point in the environment:

$$C_{collision}(\tau(t)) = \begin{cases} 0 & \text{if } d_c > 2r_q \\ -d_c^2/r_q^2 + 4 & \text{otherwise.} \end{cases} \quad (3)$$

The distribution P is complex because of the presence of arbitrary obstacles and frequently multi-modal in cluttered environments since obstacles can be avoided in multiple ways. Therefore, the analytical computation of P is generally intractable.

To approximate the density P , the expert uses random sampling. We generate samples with the M-H algorithm [41] as it provides asymptotic convergence guarantees to the true distribution. To estimate P , the M-H algorithm requires a target score function $s(\tau) \propto P(\tau | \tau_{ref}, \mathcal{C})$. We define $s(\tau) = \exp(-c(\tau, \tau_{ref}, \mathcal{C}))$, where $c(\cdot)$ is the cost of the trajectory τ . It is easy to show that this definition satisfies the conditions for the M-H algorithm to asymptotically estimate the target distribution P . Hence, the trajectories sampled with M-H will asymptotically cover all of the different modes of P . We point the interested reader to the Supplementary Materials (section S6), for an overview of the M-H algorithm and its convergence criteria.

To decrease the dimension of the sampling space, we use a compact yet expressive representation of the trajectories τ . We represent τ as a cubic B-spline $\tau_{bspline} \in \mathbb{R}^{3 \times 3}$ curve with three control points and a uniform knot vector, enabling interpolation with high computational efficiency [42]. Cubic B-splines are twice continuously differentiable and have a bounded derivative in a closed interval. Because of the differential flatness property of quadrotors [43], continuous and bounded acceleration directly translates to continuous attitude over the trajectory duration. This encourages dynamically feasible trajectories that

can be tracked by a model-predictive controller accurately [43, 44]. Therefore, instead of naively sampling the states of τ , we vary the shape of the trajectory by sampling the control points of the B-spline in a spherical coordinate system. In addition, for computational reasons, we discretize the trajectory at equally spaced time intervals of 0.1 s and evaluate the discrete version of Equation 2. Specifically, we sample a total of 50 thousands trajectories using a Gaussian with a variance of 2, 5, 10 that increases every 16 thousands samples as the proposal distribution. Despite this efficient representation, the privileged expert cannot run in real-time, given the large computational overhead introduced by sampling.

To bias the sampled trajectories toward obstacle-free regions, we replace the raw reference trajectory τ_{ref} in Equation 2 with a global collision-free trajectory τ_{gbl} from start to goal, that we compute using the approach of Liu et al. [13]. As illustrated in Figure S6, conditioning sampling on τ_{gbl} practically increases the horizon of the expert and generates more conservative trajectories. An animation explaining our expert is available in Movie 1. After removing all generated trajectories in collision with obstacles, we select the three best trajectories with lower costs. Those trajectories are used to train the student policy.

B. The student policy

In contrast to the privileged expert, the student policy produces collision-free trajectories in real time with access only to on-board sensor measurements. These measurements include a depth image estimated with SGM [37], the platform velocity and attitude, and the desired direction of flight. The latter is represented as a normalized vector heading toward the reference point 1 s in the future with respect to the closest reference state. We hypothesize that this information is sufficient for generating the highest probability samples of the distribution $P(\tau \mid \tau_{ref}, \mathcal{C})$ without actually having access to the point cloud \mathcal{C} . There are two main challenges to accomplishing this: (i) The environment is only partially observable from noisy sensor observations, and (ii) the distribution P is in general multi-modal. Multiple motion hypotheses with high probabilities can be available, but their average can have a very low probability.

We represent the policy as a neural network that is designed to be able to cope with these issues. The network consists of an architecture with two branches that produce a latent encoding of visual, inertial, and reference information, and outputs $M = 3$ trajectories and their respective collision cost. We use a pre-trained MobileNetV3 architecture [45] to efficiently extract features from the depth image. The features are then processed by a 1D convolution to generate M feature vectors of size 32. The current platform's velocity and attitude are then concatenated with the desired reference direction and processed by a four-layer perceptron with [64, 32, 32, 32] hidden nodes and LeakyReLU activations. We again use 1D convolutions to create a 32-dimensional feature vector for each mode. The visual and state features are then concatenated and processed independently for each mode by another four-layer perceptron with [64, 128, 128] hidden nodes and LeakyReLU activations. The latter predicts, for each mode, a trajectory τ and its collision cost. In summary, our architecture receives as input a depth image $d \in \mathbb{R}^{640 \times 480}$, the platform's velocity $v \in \mathbb{R}^3$, the drone's attitude expressed as a rotation matrix $q \in \mathbb{R}^9$, and reference direction $\omega \in \mathbb{R}^3$. From this input it predicts a set \mathcal{T}_n of trajectories and their relative collision cost, i.e. $\mathcal{T}_n = \{(\tau_n^k, c_k) \mid k \in [0, 1, \dots, M-1]\}$, where $c_k \in \mathbb{R}_+$. Differently from the privileged expert, the trajectory predicted by the network does not describe the full state evolution but only its position component, i.e. $\tau_n^k \in \mathbb{R}^{10 \times 3}$. Specifically, the network trajectory τ_n^k is described by:

$$\tau_n^k = [\mathbf{p}(t_i)]_{i=1}^{10}, \quad t_i = \frac{i}{10}, \quad (4)$$

where $\mathbf{p}(t_i) \in \mathbb{R}^3$ is the drone's position at time $t = t_i$ relative to its current state x . This representation is more general than the B-spline with three control points used by the sampling-based planner, and it was preferred to the latter representation to avoid the computational costs of interpolation at test time.

We train the neural network with supervised learning on the three trajectories with lowest cost found by the expert. To account for the multi-hypotheses prediction, we minimize the following *Relaxed Winner-Takes-All* (R-WTA) loss for each sample

$$\text{R-WTA}(\mathcal{T}_e, \mathcal{T}_n) = \sum_{i=0}^{|\mathcal{T}_e|} \sum_{k=0}^{|\mathcal{T}_n|} \alpha(\tau_{e,p}^i, \tau_n^k) \|\tau_{e,p}^i - \tau_n^k\|^2, \quad (5)$$

where \mathcal{T}_e and \mathcal{T}_n are the set of expert and network trajectories, $\tau_{e,p}$ denotes the position component of τ_e , and $\alpha(\cdot)$ is defined as

$$\alpha(\tau_{e,p}^i, \tau_n^k) = \begin{cases} 1 - \epsilon & \text{if } \|\tau_{e,p}^i - \tau_n^k\|^2 \leq \|\tau_{e,p}^j - \tau_n^k\|^2 \quad \forall j \neq i \\ \frac{\epsilon}{M-1} & \text{otherwise.} \end{cases} \quad (6)$$

A necessary condition for Equation 5 to be minimized is that the network predictions \mathcal{T}_n correspond to the centroids of a Voronoi tessellation of expert labels \mathcal{T}_e [46]. However, Equation 5 is not differentiable, since it performs a hard assignment between predictions and labels. Therefore, we relax the assignment with a small value ϵ (Equation 6) to optimize Equation 5 with gradient descent. Intuitively, an expert trajectory $\tau_e^i \in \mathcal{T}_e$ is associated with the closest network trajectory $\tau_n^k \in \mathcal{T}_n$ with a weight of $1 - \epsilon = 0.95$ and with $\epsilon/(M-1) = 0.025$ to all remaining hypotheses. This formulation, proposed by Rupprecht et al. [46], prevents mode collapse and was shown to outperform other popular approaches for multi-modal learning such as Mixture Density Networks [47]. In addition, the predicted collision cost c_k is trained with supervised learning on the ground-truth cost $C_{collision}(\tau_n^k)$ computed according to Equation 3. In summary, the final training loss for each sample is equal to:

$$\mathcal{L}((\mathcal{T}_e, \mathcal{T}_n)) = \lambda_1 \text{R-WTA}(\mathcal{T}_e, \mathcal{T}_n) + \lambda_2 \sum_{k=0}^{|\mathcal{T}_n|} \|c_k - C_{collision}(\tau_n^k)\|^2, \quad (7)$$

where $\lambda_1 = 10$ and $\lambda_2 = 0.1$ were empirically found to equalize the magnitudes of the two terms. This loss is averaged over a minibatch of 8 samples and minimized with the Adam optimizer [48] and a learning rate of 1×10^{-3} .

At test time we retrieve the full state information from the predicted trajectories by projecting them on the space of order-5 polynomials for each axis independently. This representation enforces continuity in position, velocity, and acceleration with respect to the current state, and facilitates dynamic feasibility because of differential flatness [43]. Considering for example the x axis, we define the polynomial projection $\mu_x(t) = \mathbf{a}_x^\top \cdot \mathbf{T}(t)$, where $\mathbf{a}_x^\top = [a_0, a_1, \dots, a_5]$ and $\mathbf{T}(t)^\top = [1, t, \dots, t^5]$. The projection term \mathbf{a}_x is found by solving the following optimization problem:

$$\begin{aligned} \underset{\mathbf{a}_x}{\text{minimize}} \quad & \sum_{i=1}^{10} \left(\tau_{n,x}^{k,i} - \mathbf{a}_x^\top \cdot \mathbf{T}\left(\frac{i}{10}\right) \right)^2 \\ \text{subject to} \quad & s_x(0) - \mathbf{a}_x^\top \cdot \mathbf{T}(0) = 0 \\ & \dot{s}_x(0) - \mathbf{a}_x^\top \cdot \dot{\mathbf{T}}(0) = 0 \\ & \ddot{s}_x(0) - \mathbf{a}_x^\top \cdot \ddot{\mathbf{T}}(0) = 0 \end{aligned} \quad (8)$$

where $\tau_{n,x}^{k,i}$ is the x component of the i^{th} element of τ_n^k and $s_x(0), \dot{s}_x(0), \ddot{s}_x(0)$ are the x position of the quadrotor and its derivatives obtained from the current state estimate. The latter corresponds

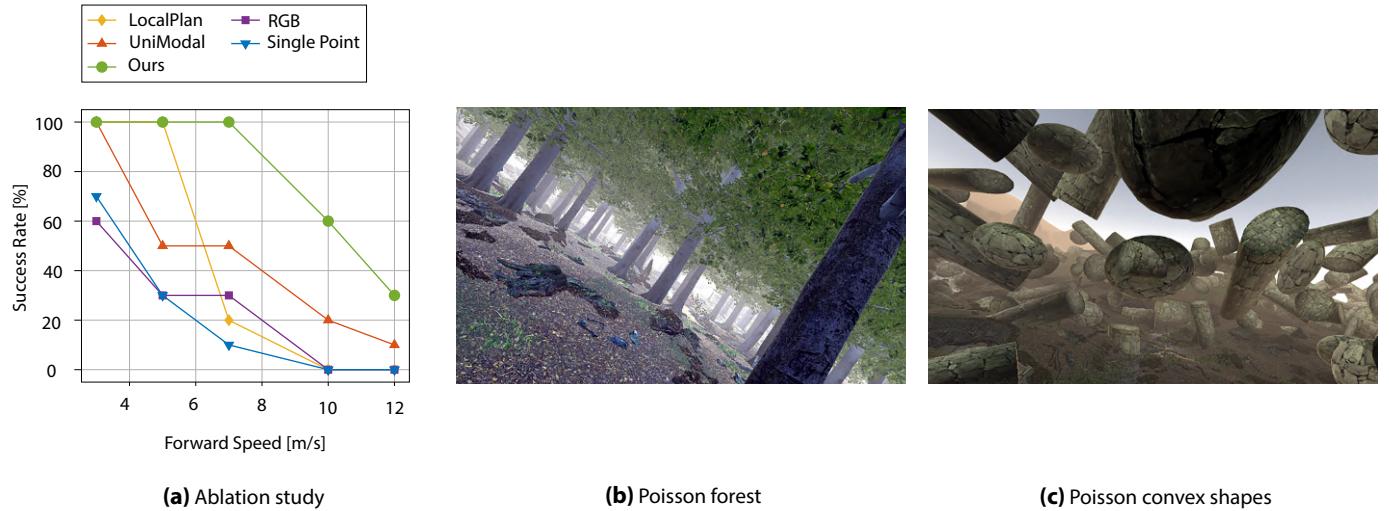


Fig. 7. (a) **Method validation.** With no initialization of the sampler on a global trajectory (*LocalPlan*), no multi-modal training (*UniModal*), or no training on SGM depth image (*RGB*), performance significantly drops. Predicting a single point in the future instead of a full trajectory (*SinglePoint*) has similar effects on performance. *Ours* consistently outperforms all the ablated versions of the system. (b-c) Training environments. We build training environments by spawning trees or convex shapes according to an homogeneous Poisson point process with varying intensities. Additional samples of the training and testing simulation environments are available in Figure S4.

to the ground-truth state when deployed in simulation and to the state estimate computed by the Intel RealSense T265 when deployed on the physical platform. To reduce abrupt changes in velocity during flight, which would lead to strong pitching motion, we additionally constrain the polynomial's average speed to a desired value v_{des} . To do so, we scale the time t of polynomial $\mu_x(t)$ by a factor $\beta = v_{des}/v_\mu^x$, i.e. $t' = \beta t$, where $v_\mu^x = \|\mu(1) - \mu(0)\|$.

Once all predicted trajectories are projected we select one for execution. To do so, we select trajectories with $c^*/c_k \geq 0.95$ ($c^* = \min c_k$) and compute their input costs according to Mellinger and Kumar [43]. The one with the lowest input costs is tracked by a model-predictive controller [44]. Intuitively, this choice enforces temporal continuity in the trajectories. For example, when dodging an obstacle to the right, we do not want to steer toward the left at the next iteration, unless strictly necessary because of the appearance of an obstacle.

C. Training environments

We build custom environments in the Flightmare simulator [33] to collect training data. All environments are built by spawning items on the uneven empty ground of an off-the-shelf Unity environment. We spawn items belonging to two categories: simulated trees, available off-the-shelf, and a set of convex shapes such as ellipsoids, cuboids, and cylinders. Sample observations from these environments are available in Figure 7. Additional training samples, together with observations collected in the simulated testing environments, are available in Figure S4. The dimensions of these shapes are randomized according to a continuous uniform random distribution with $x \in \mathcal{U}(0.5, 4)$, $y \in \mathcal{U}(0.5, 4)$, and $z \in \mathcal{U}(0.5, 8)$. Training environments are created by spawning either of the two categories of items according to a homogeneous Poisson point process with intensity δ .

We generate a total of 850 environments by uniform randomization of the following two quantities: item category, i.e., trees or shapes, and the intensity $\delta \in \mathcal{U}(4, 7)$, with $\delta \in \mathbb{N}_+$. For each environment, we compute a global collision-free trajectory τ_{gbl} from the starting location to a point 40 m in front of it. The trajectory τ_{gbl} is not observed by the student policy, but only by the expert. The student is

only provided with a straight, potentially not collision-free, trajectory from start to end to convey the goal.

To assure sufficient coverage of the state space, we use the dataset aggregation strategy (DAgger) [49]. This process consists of rolling out the student policy and labeling the visited states with the expert policy. To avoid divergence from τ_{gbl} and prevent crashes in the early stages of training, we track a trajectory predicted by the student only if the drone's distance from the closest point on τ_{gbl} is smaller than a threshold ξ , initialized to zero. Otherwise, we directly track τ_{gbl} with a model-predictive controller. Every 30 environments the student is re-trained on all available data and the threshold ξ set to $\xi' = \min(\xi + 0.25, 6)$. Aggregating data over all environments results in a dataset of about 90K samples. For the narrow-gap experiments, we finetune the student policy on 100 environments created by adding a 50 m long wall with a single vertical gap of random width $w_g \in \mathcal{U}(0.7, 1.2)$ meters in the center. In those environments, the drone starts at a 10 m distance from the wall with a randomized lateral offset $l \in \mathcal{U}(-5, 5)$ from the gap. We collect about 10K training samples from those environments.

We evaluate the trained policies in simulation on environments not seen during training but coming from the same distributions. The same policies are then used to control the physical platforms in real-world environments. When deployed in simulation, we estimate depth with SGM [37] from a simulated stereo pair and use the ground-truth state of the platform. Conversely, on the physical platform depth is estimated by an off-the-shelf Intel RealSense 435 and state estimation is performed by an Intel RealSense T265. More details on our experimental platform are available in section S3.

D. Method validation

Our approach is based on several design choices that we validate in an ablation study. We ablate the following components: (i) the use of global planning to initialize the sampling of the privileged expert, (ii) the use of depth as an intermediate representation for action, and (iii) multi-modal network prediction. The results in Figure 7 show that all components are important and that some choices have a larger impact

than others. Our study indicates that depth perception plays a fundamental rule for high-speed obstacle avoidance: When training on color images (*RGB*) performance drops significantly. This is inline with previous findings [28, 50] showing that intermediate image representations have a strong positive effect on performance when sufficiently informative for the task. Not accounting for multi-modal trajectory prediction (*UniModal*) is also detrimental for performance. This is because of the ambiguities inherent of the task of obstacle avoidance: the average of expert trajectories is often in collision with obstacles. Not initializing the sampling of expert trajectories on a global plan (*LocalPlan*) is not important for low-speed flight, but plays an important role for success at higher speeds. Biasing the motion away from regions that are densely populated by obstacles is particularly important for high-speed flight where little slack is available for reacting to unexpected obstacles. In addition, we compare our output representation, *i.e.* a trajectory 1 s in the future, to the less complex output representation in prior work [51], *i.e.* a single waypoint in the future (*SinglePoint*). The results indicate that the limited representation power of the latter representation causes performance to drop significantly, especially at high speeds.

5. ACKNOWLEDGMENTS

The authors thank T. Längle, M. Sutter, Y. Song, C. Pfeiffer, L. Bauersfeld, and N. Aepli for their contributions to the drone design, the simulation environment, and preparation of multimedia material. We also thank J. Lee for the template of this document. **Funding:** This work was supported by the Intel Network on Intelligent Systems, Armasuisse, the National Centre of Competence in Research (NCCR) Robotics through the Swiss National Science Foundation (SNSF) and by the European Research Council (ERC) under Grant Agreement 864042 (AGILEFLIGHT). **Author contributions:** A.L and E.K. formulated the main ideas, implemented the system, performed all experiments, and wrote the paper; R.R and M.M. contributed to the experimental design, data analysis, and paper writing; V.K. and D.S. provided funding and contributed to the design and analysis of experiments. **Competing interests:** The authors declare that they have no competing interests. **Data and materials availability:** All (other) data needed to evaluate the conclusions in the paper are present in the paper or the Supplementary Materials. Other material can be found at <http://rpg.ifl.uzh.ch/AgileAutonomy.html> and at [56].

SUPPLEMENTARY MATERIALS

- Section S1. Sensitivity to noise in estimation and control.
- Section S2. The impact of obstacle density on performance.
- Section S3. The platform's hardware.
- Section S4. Computational complexity.
- Section S5. Theoretical maximum speed while avoiding a pole.
- Section S6. Metropolis-Hastings sampling.
- Figure S1. Sensitivity to noise in estimation and control.
- Figure S2. Ablation of tree density.
- Figure S3. Illustration of the experimental platform.
- Figure S4. Illustration of the simulated environments.
- Figure S5. Qualitative illustration of the related work.
- Figure S6. Motivation of global planning for label generation.
- Movie S1.** Deployment at different speeds.
- Movie S2.** Deployment in simulation.

REFERENCES

1. J. Verbeke, J. D. Schutter, "Experimental maneuverability and agility quantification for rotary unmanned aerial vehicle," *Int. J. of Micro Air Veh.*, vol. 10, no. 1, pp. 3–11, 2018.
2. E. Ackerman, "Ai-powered drone learns extreme acrobatics" (IEEE Spectrum 2020).
3. J. Hilf, K. Umbach, "The commercial use of drones," *Comput. Law Rev. Int.*, vol. 16, no. 3, 2015.
4. D. Falanga, S. Kim, and D. Scaramuzza, "How fast is too fast? The role of perception latency in high-speed sense and avoid," *IEEE Robot. Autom. Lett.*, vol. 4, no. 2, pp. 1884–1891, 2019.
5. L. Heng, D. Honegger, G. H. Lee, L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys, "Autonomous visual mapping and exploration with a micro aerial vehicle," *J. Field Robotics*, vol. 31, no. 4, pp. 654–675, 2014.
6. S. S. G., C. Thibault, M. Trentini, and H. Li, "3d mapping for autonomous quadrotor aircraft," *Unmanned Syst.*, vol. 5, no. 3, pp. 181–196, 2017.
7. D. Scaramuzza, M. Achtelik, L. Doitsidis, F. Fraundorfer, E. B. Kosmatopoulos, A. Martinelli, M. W. Achtelik, M. Chli, S. A. Chatzichristofis, L. Kneip, D. Gurdan, L. Heng, G. H. Lee, S. Lynen, M. Pollefeys, A. Renzaglia, R. Siegwart, J. C. Stumpf, P. Tanskanen, C. Troiani, S. Weiss, and L. Meier, "Vision-controlled micro flying robots: From system design to autonomous navigation and mapping in gps-denied environments," *IEEE Robotics Autom. Mag.*, vol. 21, no. 3, pp. 26–40, 2014.
8. M. Faessler, F. Fontana, C. Forster, E. Mueggler, M. Pizzoli, and D. Scaramuzza, "Autonomous, vision-based flight and live dense 3d mapping with a quadrotor micro aerial vehicle," *J. Field Robotics*, vol. 33, no. 4, pp. 431–450, 2016.
9. M. Blösch, S. Weiss, D. Scaramuzza, and R. Siegwart, "Vision based MAV navigation in unknown and unstructured environments," in *IEEE Int. Conf. Robot. Autom. (ICRA)*, 2010, pp. 21–28.
10. A. Bry and N. Roy, "Rapidly-exploring random belief trees for motion planning under uncertainty," in *IEEE Int. Conf. Robot. Autom. (ICRA)*, 2011.
11. C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *Proc. Int. Symp. Robot. Research (ISRR)*, 2013.
12. R. Allen and M. Pavone, "A real-time framework for kinodynamic planning with application to quadrotor obstacle avoidance," in *AIAA Guidance, Navigation, and Control Conference*, 2016, p. 1374.
13. S. Liu, K. Mohta, N. Atanasov, and V. Kumar, "Search-based motion planning for aggressive flight in se (3)," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2439–2446, 2018.
14. H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. I. Nieto, "Voxblox: Incremental 3d euclidean signed distance fields for on-board MAV planning," in *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, 2017, pp. 1366–1373.
15. T. Ozaslan, G. Loianno, J. Keller, C. J. Taylor, V. Kumar, J. M. Wozenckraft, and T. Hood, "Autonomous navigation and mapping for inspection of penstocks and tunnels with mavs," *IEEE Robot. Autom. Lett.*, vol. 2, no. 3, pp. 1740–1747, 2017.
16. 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, and V. Kumar, "Fast, autonomous flight in gps-denied and cluttered environments," *J. Field Robot.*, vol. 35, no. 1, pp. 101–120, 2018.
17. A. J. Barry, P. R. Florence, and R. Tedrake, "High-speed autonomous obstacle avoidance with pushbroom stereo," *J. Field Robot.*, vol. 35, no. 1, pp. 52–68, 2018.
18. B. Zhou, F. Gao, L. Wang, C. Liu, and S. Shen, "Robust and efficient quadrotor trajectory generation for fast autonomous flight," *IEEE Robot. Autom. Lett.*, vol. 4, no. 4, pp. 3529–3536, 2019.
19. M. Ryll, J. Ware, J. Carter, and N. Roy, "Efficient trajectory planning for high speed flight in unknown environments," in *2019 Int. Conf. Robot. Autom. (ICRA)*. IEEE, 2019, pp. 732–738.
20. J. Tordesillas, B. T. Lopez, and J. P. How, "FASTER: fast and safe trajectory planner for flights in unknown environments," in *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, 2019, pp. 1934–1940.
21. T. Cieslewski, E. Kaufmann, and D. Scaramuzza, "Rapid exploration with multi-rotors: A frontier selection method for high speed flight," in

- IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, 2017, pp. 2135–2142.
22. Z. Zhang and D. Scaramuzza, “Perception-aware receding horizon navigation for mavs,” in *IEEE Int. Conf. Robot. Autom. (ICRA)*, 2018, pp. 2534–2541.
 23. G. Loianno and D. Scaramuzza, “Special issue on future challenges and opportunities in vision-based drone navigation,” *J. Field Robot.*, vol. 37, no. 4, pp. 495–496, 2020.
 24. S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. Bagnell, and M. Hebert, “Learning monocular reactive UAV control in cluttered natural environments,” in *IEEE Int. Conf. Robot. Autom. (ICRA)*, 2013, pp. 1765–1772.
 25. F. Sadeghi and S. Levine, “CAD2RL: real single-image flight without a single real image,” in *Robotics: Science and Systems RSS*, N. M. Amato, S. S. Srinivasa, N. Ayanian, and S. Kuindersma, Eds., 2017.
 26. D. Gandhi, L. Pinto, and A. Gupta, “Learning to fly by crashing,” in *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, 2017, pp. 3948–3955.
 27. A. Loquercio, A. I. Maqueda, C. R. del-Blanco, and D. Scaramuzza, “Dronet: Learning to fly by driving,” *IEEE Robotics Autom. Lett.*, vol. 3, no. 2, pp. 1088–1095, 2018.
 28. E. Kaufmann, A. Loquercio, R. Ranftl, M. Müller, V. Koltun, and D. Scaramuzza, “Deep drone acrobatics,” *RSS: Robotics, Science, and Systems*, 2020.
 29. M. Müller, A. Dosovitskiy, B. Ghanem, and V. Koltun, “Driving policy transfer via modularity and abstraction,” in *Conference on Robot Learning*, 2018, pp. 1–15.
 30. D. Chen, B. Zhou, V. Koltun, and P. Krähenbühl, “Learning by cheating,” in *Conference on Robot Learning*, 2019, pp. 66–75.
 31. B. Zhou, J. Pan, F. Gao, and S. Shen, “RAPTOR: robust and perception-aware trajectory replanning for quadrotor fast flight,” *CoRR*, vol. abs/2007.03465, 2020.
 32. P. Florence, J. Carter, and R. Tedrake, “Integrated perception and control at high speed: Evaluating collision avoidance maneuvers without maps,” in *Algorithmic Foundations of Robotics XII*. Springer, 2020, pp. 304–319.
 33. Y. Song, S. Naji, E. Kaufmann, A. Loquercio, and D. Scaramuzza, “Flightmare: A flexible quadrotor simulator,” in *Conference on Robot Learning*, 2020.
 34. F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, “Rotors—a modular gazebo mav simulator framework,” in *Robot Operating System (ROS)*. Springer, 2016, pp. 595–625.
 35. A. Juliani, V.-P. Berges, E. Vckay, Y. Gao, H. Henry, M. Mattar, and D. Lange, “Unity: A general platform for intelligent agents,” *arXiv e-prints*, 2018.
 36. S. Karaman and E. Frazzoli, “High-speed flight in an ergodic forest,” in *IEEE Int. Conf. Robot. Autom. (ICRA)*. IEEE, 2012, pp. 2899–2906.
 37. H. Hirschmuller, “Stereo processing by semiglobal matching and mutual information,” *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 30, no. 2, pp. 328–341, 2007.
 38. G. Gallego, T. Delbrück, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. Davison, J. Conradt, K. Daniilidis, and D. Scaramuzza, “Event-based vision: A survey,” *IEEE Trans. Pattern Anal. Machine Intell.*, 2020.
 39. D. Falanga, K. Kleber, and D. Scaramuzza, “Dynamic obstacle avoidance for quadrotors with event cameras,” *Science Robotics*, vol. 5, no. 40, 2020.
 40. L. Keselman, J. Iselin Woodfill, A. Grunnet-Jepsen, and A. Bhowmik, “Intel realsense stereoscopic depth cameras,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2017.
 41. W. K. Hastings, “Monte carlo sampling methods using markov chains and their applications,” *Biometrika*, vol. 57, no. 1, pp. 97–109, 1970.
 42. J. Gallier, *Curves and Surfaces in Geometric Modeling: Theory and Algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999.
 43. D. Mellinger and V. Kumar, “Minimum snap trajectory generation and control for quadrotors,” in *IEEE Int. Conf. Robot. Autom. (ICRA)*, 2011, pp. 2520–2525.
 44. D. Falanga, P. Foehn, P. Lu, and D. Scaramuzza, “Pampc: Perception-aware model predictive control for quadrotors,” in *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2018, pp. 1–8.
 45. A. Howard, R. Pang, H. Adam, Q. V. Le, M. Sandler, B. Chen, W. Wang, L. Chen, M. Tan, G. Chu, V. Vasudevan, and Y. Zhu, “Searching for mobilenetv3,” in *Int. Conf. Comp. Vis., ICCV*, 2019, pp. 1314–1324.
 46. C. Rupprecht, I. Laina, R. S. DiPietro, and M. Baust, “Learning in an uncertain world: Representing ambiguity through multiple hypotheses,” in *Int. Conf. Comp. Vis., ICCV*, 2017, pp. 3611–3620.
 47. C. M. Bishop, *Mixture density networks*. Aston University, 1994.
 48. D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *Int. Conf. Learn. Repr. (ICLR)*, 2015.
 49. S. Ross, G. Gordon, and D. Bagnell, “A reduction of imitation learning and structured prediction to no-regret online learning,” in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 627–635.
 50. B. Zhou, P. Krähenbühl, and V. Koltun, “Does computer vision matter for action?” *Sci. Robotics*, vol. 4, no. 30, 2019.
 51. A. Loquercio, E. Kaufmann, R. Ranftl, A. Dosovitskiy, V. Koltun, D. Scaramuzza, “Deep drone racing: From simulation to reality with domain randomization,” *IEEE Trans. Rob.*, vol. 36, pp. 1–14, 2019.
 52. E. Tomppo, “Models and methods for analysing spatial patterns of trees,” PhD dissertation, Finnish Forest Research Institute, 1986.
 53. D. Hernandez-Juarez, A. Chacón, A. Espinosa, D. Vázquez, J. C. Moure, and A. M. López, “Embedded real-time stereo estimation via semi-global matching on the GPU,” in *Int. Conf. Comp. Sci. ICCS*, 2016.
 54. C. Andrieu, N. de Freitas, A. Doucet, and M. I. Jordan, “An introduction to MCMC for machine learning,” *Mach. Learn.*, vol. 50, no. 1-2, pp. 5–43, 2003.
 55. G. O. Roberts and A. F. Smith, “Simple conditions for the convergence of the gibbs sampler and metropolis-hastings algorithms,” *Stochastic processes and their applications*, vol. 49, no. 2, pp. 207–216, 1994.
 56. A. Loquercio, E. Kaufmann, R. Ranftl, M. Mueller, V. Koltun, D. Scaramuzza, “Code and Dataset for the paper Speed Flight in the Wild” (Science Robotics,2021), *Zenodo*, DOI: 10.5281/zenodo.5517791, (2021).

SUPPLEMENTARY MATERIALS

S1. SENSITIVITY TO ESTIMATION AND CONTROL NOISE

We compare the sensitivity of our proposed approach to noise in state estimation as well as the applied control action against the *Reactive* and *FastPlanner* baselines. To this end, we test in the high-density simulated forest as explained in section B and perturb the simulated state as well as the applied control command with additive disturbances. Both types of disturbances are approximated as Gaussian noise with mean and variance identified from fast flights in an instrumented motion capture volume. This instrumented tracking volume allows us to accurately identify noise and drift in state estimation and the applied control action. Table S1 shows the identified noise parameters for the estimated state. The parameters are identified component-wise. Note that attitude is denoted in terms of the residual quaternion that rotates the estimated attitude onto the ground-truth attitude. The noise in control action is represented as a multiplicative factor m on the commanded collective thrust, which mimics a drop in motors efficiency because of aerodynamics effects or shortages in battery power. This factor m is selected from a uniform distribution $m \in \mathcal{U}(0.9, 1)$ for each rollout.

As laid out in section B, we test the performance for a set of increasing speeds, while the tree density is kept constant at the highest density of $\delta_3 = \frac{1}{25}$ treem $^{-2}$. For each average forward speed, we perform 10 rollouts and compute the respective success rate. We repeat the set of experiments five times with different random seeds and report the mean and standard deviation over sets of rollouts.

Figure S1 shows the results of this experiment, with the line indicating the mean success rate and the shaded area marking one standard deviation. At a low speed of 3 m s^{-1} , the performance of our approach is not affected by neither the noise in estimation nor control. In comparison, the *Reactive* baseline shows significant drop in performance when exposed to noise in action and even more in case of noise in state estimation. The *FastPlanner* is less sensitive to noise and only exhibits a small drop in performance at 3 m s^{-1} . For higher speeds, the performance of our approach with noise is between 5% and 10% worse than in the noise-free case, with noise in estimation resulting in a larger drop in success rate. The same pattern can be observed for the *FastPlanner* and the *Reactive* baselines, which both show a larger performance decrease when exposed to noise in estimation. In summary, neither noise in estimation nor noise in control did significantly affect performance of our approach, while the performance of the *Reactive* baseline is strongly reduced. The *FastPlanner* baseline shows larger robustness against noise, but performs inferior in general.

S2. PERFORMANCE ANALYSIS IN FUNCTION OF OBSTACLE DENSITY

We perform a set of experiments in a simulated forest to evaluate the performance of our approach with respect to obstacle density. Similarly to previous experiments, we use as baselines the mapping and planning method of Zhou et al. [18] (*FastPlanner*) and the reactive planner of Florence et al. [32] (*Reactive*). We compare the different approaches according to their success rate, which measures how often the drone reaches the goal location within a radius of 5 meters without crashing.

We build a simulated forest [36] in a rectangular region $R(l, w)$ of width w and length l , and fill it with trees that have a diameter of about 0.6 m. Trees are randomly placed according to a homogeneous Poisson point process P with intensity δ treem $^{-2}$ [36]. Note that Tomppo et al. [52] found that around 30% of the forests in Finland could be considered as a realization of a spatial Poisson point process. We control the task difficulty by changing the tree density δ . We set

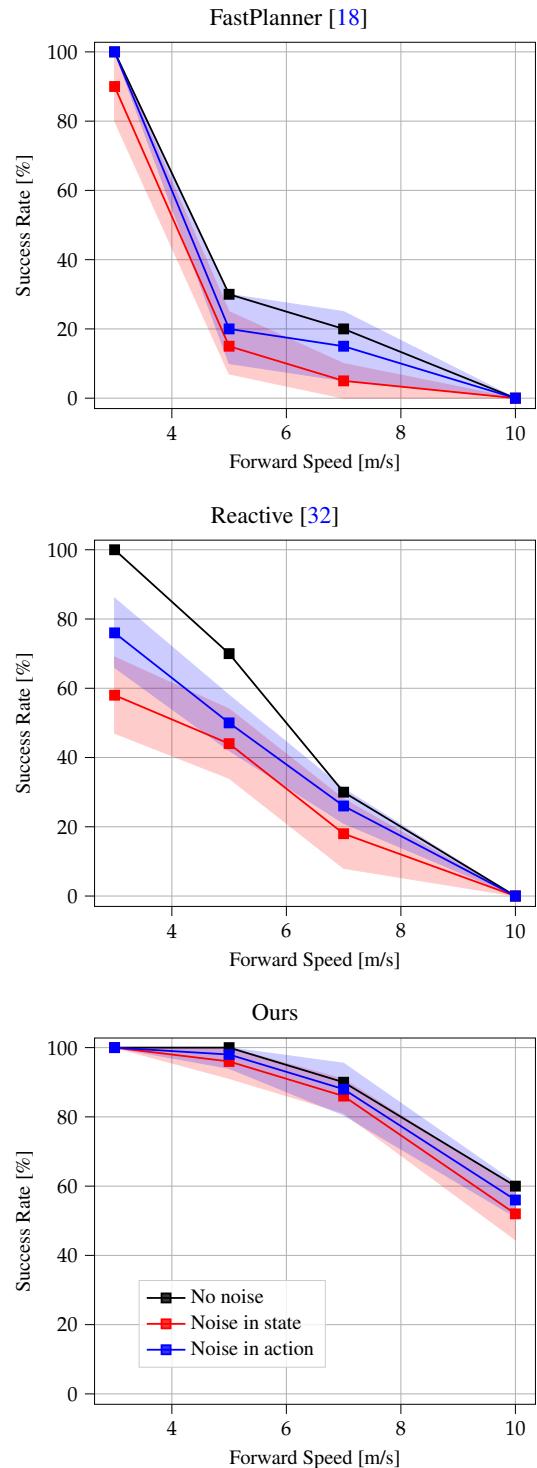


Fig. S1. Sensitivity analysis of noise in state estimation and control.

$w = 30 \text{ m}$ and $l = 60 \text{ m}$ and start the drone at position $s = (-\frac{l}{2}, -\frac{w}{2})$ (the origin of the coordinate system is at the center of $R(l, w)$). We provide the drone with a straight reference trajectory of 40 m length. We test on three different tree densities with increasing difficulty: $\delta_1 = \frac{1}{49}$ (low), $\delta_2 = \frac{1}{36}$ (medium), and $\delta_3 = \frac{1}{25}$ treem $^{-2}$. We vary the average forward speed of the drone between 3 m s^{-1} and 12 m s^{-1} . We repeat the experiments with 10 different random realizations of the forest for each difficulty, using the same random

	v_x	v_y	v_z	ω_x	ω_y	ω_z	q_w	q_x	q_y	q_z
μ	0.009	-0.198	-0.570	-0.009	0.012	-0.004	0.997	0.002	0.022	0.003
σ	0.496	0.210	1.243	0.302	0.587	0.031	0.000	0.003	0.001	0.001

Table S1. State estimation noise for velocity, angular velocity and attitude (quaternion). The noise parameters are obtained from real-world flights in a vicon-based system.

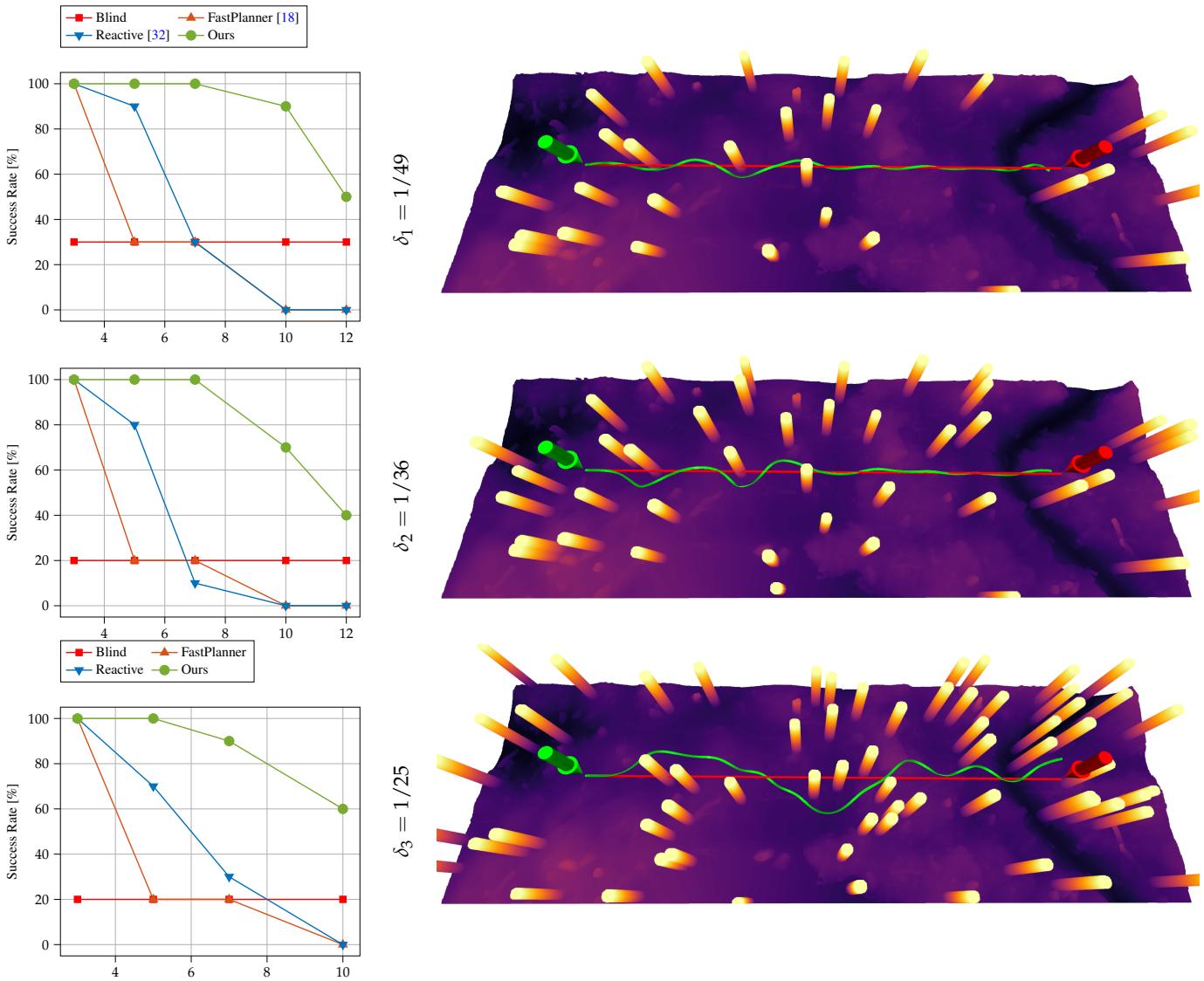


Fig. S2. Experiments in a simulated forest. Experiments are ordered for increasing difficulty, which is controlled by the tree density δ . The left column reports success rates at various speeds. The right column shows one of the random realizations of the environment together with paths taken by different policies from start (green arrow) to end (red arrow). The paths illustrate the blind policy (red) and the path taken by our approach (green). Our approach consistently outperforms the baselines in all environments and at all speeds.

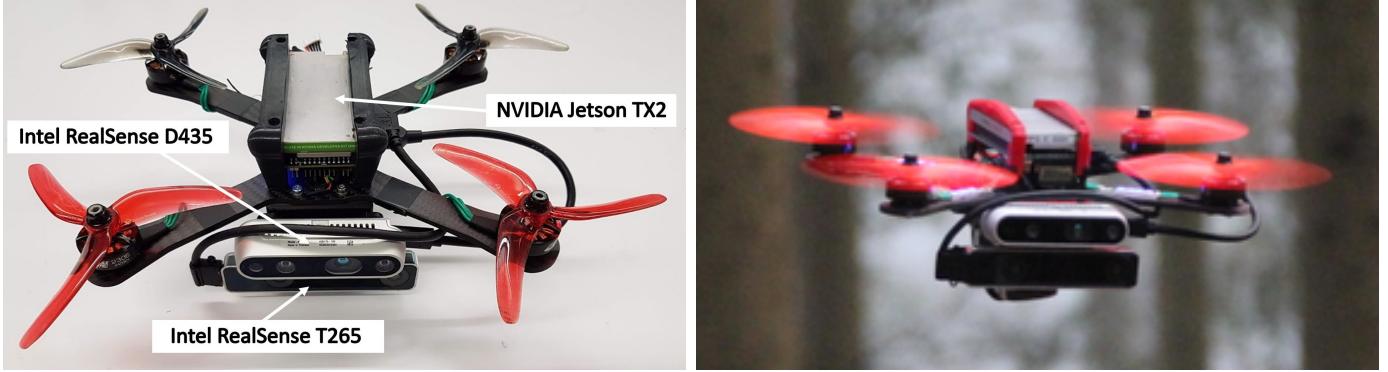


Fig. S3. Illustration of our experimental platform. The main computational unit is an NVIDIA Jetson TX2, whose GPU is used for neural network inference and CPU for the control stack. Sensing is performed by an Intel Realsense T265 for state estimation and an Intel Realsense D435 for depth estimation.

seed for all methods.

The first three rows in Figure S2 show the results of this experiment, together with one example environment for each difficulty. At a low speed of 3 m s^{-1} , all methods successfully complete every run even for high difficulties. As speed increases, the success rates of the baselines quickly degrade. At 10 m s^{-1} , no baseline completes even a single run successfully, irrespective of task difficulty. In contrast, our approach is significantly more robust at higher speeds. It achieves 100% success rate up to 5 m s^{-1} . For speeds of 10 m s^{-1} , our approach has a success rate of 90% in the low difficulty task and 60% in the high difficulty task. Moreover, we show that our approach can go as fast as 12 m s^{-1} with a success rate of up to 50%. Our method achieves this performance by decreasing the latency between sensors and actions and increasing the robustness to sensor noise, developed using regularities in the data. We show additional controlled studies on latency and sensor noise in section C and section D.

S3. EXPERIMENTAL PLATFORM

To validate our approach with real-world experiments, we designed a lightweight, but powerful, quadrotor platform. The main frame is an Armattan Chameleon 6", equipped with Hobbywing XRotor 2306 motors and 5", three-bladed propellers. The platform has a total weight of 890 grams and can produce a maximum thrust of about 40 N, which results in a thrust-to-weight ratio of 4.4. The weight and power of this platform is comparable to the ones used by professional pilots in drone racing competitions.

The platform's main computational unit is an NVIDIA Jetson TX2 accompanied by a ConnectTech Quasar carrier board. The GPU of the Jetson TX2 is used to run neural network inference and its CPU for the rest of our control framework. The output of this framework is a low-level control command including a collective thrust and angular rates to be achieved for flying. The desired commands are sent to a commercial flight controller running BetaFlight, which produces single-rotor commands that are fed to the 4-in-1 motor controller.

Our quadrotor is equipped with two off-the-shelf sensing units: an Intel RealSense T265 and an Intel RealSense D435i. Both have a stereo camera setup and an integrated IMU. The RealSense T265 runs a visual-inertial odometry pipeline to output the state estimation of the platform at 200 Hz, which we directly use without any additional processing. Our second sensing unit, the RealSense D435i, outputs a hardware-accelerated depth estimation pipeline on its stereo setup. The latter pipeline provides to our framework a dense depth in VGA resolution ($640 \times 480 \text{ px}$) at 30 Hz, which we use without further

processing. The horizontal field of view of this observation is about 90° , which appeared to be sufficient for the task of obstacle avoidance. For experiments at speeds of 7 m s^{-1} and above, we tilt the depth sensor by 30° to assure that the camera would look forward during flight. This is indeed a typical camera setup for high-speed flight in drone racing competitions. To support the transfer from simulation to reality, we build a simulated stereo setup with the same characteristics of the RealSense D435, on which we run a [GPU implementation of SGM \[53\]](#) to estimate dense depth in simulation.

Our software stack is developed in both C++ and Python, and will be made publicly available upon acceptance. Specifically, we implement the trajectory prediction framework with Tensorflow in a Python node and the rest of our trajectory projection and tracking software in separate C++ nodes. The communication between different nodes is implemented with ROS. Specifically, the Tensorflow node predicts trajectories at 24.7 Hz on the physical platform, and the MPC generates commands in the form of collective thrust and body rates at 100 Hz to track those trajectories. The low-level controller, responsible for tracking desired body rates and collective thrust predicted by the MPC, runs at 2 kHz. The platform only receives a start and stop command from the base computer, and is therefore completely autonomous during flight.

S4. COMPUTATIONAL COMPLEXITY

The baseline with the largest latency is FastPlanner. This baseline has three components: sensing, mapping and planning. Sensing includes transforming a depth image to a pointcloud after filtering. Mapping includes ray casting and Euclidean Signed Distance Field (ESDF) computation. Finally, planning includes finding the best trajectory to reach the goal while avoiding obstacles. The total time to perform all these operations is 65.2 ms. However, it is important to note that, while the depth filtering and the probabilistic ray casting process are necessary to remove sensing errors, those operations make the inclusions of obstacles in the local map slower. Practically, 2-3 observations are required to add an obstacle to the local map, therefore largely increasing the overall latency of the system.

Removing the mapping stage altogether, the *Reactive* baseline experiences significant gains in computation time. For this baseline, the sensing latency is the time between receiving a depth observation and generating a point cloud after filtering and outlier rejection. The planning latency consists of building a KD-Tree from the filtered point cloud, and selecting the best trajectory out of the available motion primitives according to a cost based on collision probability and proximity

to the goal. This baseline is about three times faster than *FastPlanner*, with a total latency of 19.1 ms. However, the reduced latency comes at the cost of a lower trajectory-representation power, since the planner can only select a primitive from a pre-defined motion library. In addition, given the lack of temporal filtering, the reactive baseline is very sensitive to sensing errors, which can drastically affect performance at high speeds.

Our approach has significantly lower latency than both baselines: when network inference is performed on the GPU, our approach is 25.3 times faster than *FastPlanner* and 7.4 times faster than the *Reactive* baseline. When GPU inference is disabled, the network's latency increases by only 8 ms, and our approach is still significantly faster than both baselines. For our approach, we break down computation into three operations: (i) sensing, which includes the time for recording an image and convert it to an input tensor, (ii) neural network inference, and (iii) projection, which is the time to project the prediction of the neural network into the space of dynamically feasible trajectories for the quadrotor. Moving from the desktop computer to the onboard embedded computing device, the network's forward pass requires 38.9 ms. Onboard, the total time to pass from sensor to action is then 41.6 ms, which is sufficient to update actions at up to 24.3 Hz.

5. ROTATIONAL DYNAMICS

Extending [4] to the quadrotor platform, we approximate the maximum speed v_{\max} that still allows successful avoidance of a vertical cylindrical obstacle. The avoidance maneuver is described as a sequence of two motion primitives consisting of pure rolling and pure acceleration. Both primitives are executed with maximum motor inputs. Concretely, we treat the time required to reorient the quadrotor as additional latency in the system, which can be computed by

$$t_{\text{rot}} = \sqrt{\frac{2\phi J}{T_{\max}}}, \quad (9)$$

with J being the moment of inertia, T_{\max} the maximum torque the quadrotor can produce, and ϕ the desired roll angle. As the roll angle becomes a decision variable itself in this setting, we identify the best roll angle by maximizing the speed that still allows successful avoidance. Assuming that the quadrotor oriented at a roll angle ϕ accelerates with full thrust, its lateral position p_{lat} can be described by

$$p_{\text{lat}}(t) = \frac{1}{2} \sin \phi \cdot c_{\max} \cdot t^2, \quad (10)$$

where c_{\max} denotes the maximum mass-normalized thrust of the platform. Solving this equation for t and setting $p_{\text{lat}} = r_{\text{obs}}$ allows to formulate a maximum linear speed that still allows successful avoidance when considering the sensing range s , the combined radius of the obstacle and the drone r_{obs} , the sensing latency t_s , the processing latency t_p , *i.e.* the time to convert an observation into motor commands, and the latency introduced to reorient the platform t_{rot} :

$$v_{\max} = \frac{s}{t_s + t_p + t_{\text{rot}} + \sqrt{\frac{2r_{\text{obs}}}{\sin \phi \cdot c_{\max}}}}. \quad (11)$$

Inserting 9 into 11 we can identify ϕ that maximizes v_{\max} . In our study case, we avoid a pole with diameter 1.5 m and model the drone as a sphere with diameter 0.4 m. Therefore, the combined radius of the obstacle and drone is $r_{\text{obs}} = 0.95$ m. In addition, we set $c_{\max} = 35.3 \text{ m s}^{-2}$, $J = 0.007 \text{ kg m}^{-2}$. Similarly to Falanga et al. [4], we define the sensing latency as the worst-case time between the quadrotor being closer than the sensing range s to the obstacle and an image containing the obstacle being rendered and provided to the navigation

s	T_{\max}	J	c_{\max}	t_s	r_{obs}
6 m	1.02 N m	0.007 kg m ⁻²	35.3 m s ⁻²	66 ms	0.95 m

Table S2. Fixed parameters for the calculation of the theoretical maximum speed of our simulated sensor and quadrotor. Note that, differently from the physical quadrotor, the simulated drone is in the *plus* configuration.

	t_p	t_{rot}	ϕ	v_{\max}
FastPlanner [18]	65.2 ms	125.2 ms	65.5°	12.0 m s ⁻¹
Reactive [18]	19.1 ms	125.2 ms	65.5°	13.2 m s ⁻¹
Ours	10.3 ms	125.2 ms	65.5°	13.5 m s ⁻¹

Table S3. We compute the theoretical maximum speed by optimizing Equation 11 according to the variable ϕ using grid-search. The processing latency t_p represents the time to convert an image to a motor command (see section C), and the rotation latency t_{rot} is the time for the quadrotor to rotate at the angle ϕ . Interestingly, the methods' processing latencies do not affect the rotation angle and latency.

algorithms. This time corresponds to the inverse of the frame rate. Therefore, $t_s = 66$ ms, since images are rendered at 15 Hz. Also similarly to Falanga et al. [4], we select as sensing range s for our stereo camera the range at which the depth uncertainty because of triangulation is below 20% of the actual depth. Using this definition and the parameters of our simulated stereo camera (VGA resolution, baseline of 0.1 m, and focal length of 4 mm), we compute a sensing range of $s = 6$ m. Table S2 summarizes all the variables of the problem.

Using these values, we derive the theoretical maximum speed for each method by considering their processing latency t_p (c.f. section C) and optimizing Equation 11 according to the free variable ϕ . Table S3 presents the results of the optimization. Note that this approximation does not account for the fact that the quadrotor platform already performs some lateral acceleration during the rotation phase.

6. METROPOLIS-HASTINGS SAMPLING

In statistics, the Metropolis-Hastings (M-H) algorithm [41] is used to sample a distribution $P(w)$ which can't be directly accessed. To generate the samples, the M-H algorithm requires a score function $d(w)$ proportional to $P(w)$. Requiring $d(w) \propto P(w)$ waives the need to find the normalization factor $Z = \int_w d(w)$ such that $P(w) = \frac{1}{Z}d(w)$, which can't be easily calculated for high-dimensional spaces. Metropolis-Hastings is a Markov Chain Monte Carlo sampling method [54]. Therefore, it generates samples by constructing a Markov chain that has the desired distribution as its equilibrium distribution. In this Markov chain, the next sample w_{t+1} comes from a distribution $t(w_{t+1}|w_t)$, referred to as transition model, which only depends on the current sample w_t . The transition model $t(w_{t+1}|w_t)$ is generally a pre-defined parametric distribution, e.g. a Gaussian. The next sample w_{t+1} is then accepted and used for the next iteration, or it is rejected, discarded, and the current sample w_t is re-used. Specifically, the sample is accepted with probability equal to

$$\alpha = \min \left(1, \frac{d(w_{t+1})}{d(w_t)} \right) = \min \left(1, \frac{P(w_{t+1})}{P(w_t)} \right). \quad (12)$$

Therefore, M-H always accepts a sample with a higher score than its predecessor. However, the move to a sample with a smaller score will sometimes be rejected, and the higher the drop in score $\frac{1}{\alpha}$, the smaller the probability of acceptance. Therefore, many samples come from the high-density regions of $P(w)$, while relatively few from the low-density regions. Roberts et. al [55] have shown that under the mild condition that

$$\alpha > 0 \quad \forall w_t, w_{t+1} \in \mathcal{W}, \quad (13)$$

$\hat{P}(w)$ will asymptotically converge to the target distribution $P(w)$. According to Eq. 13, the probability of accepting a sample with lower score than its predecessor is always different from zero, which implies that the method will not ultimately get stuck into a local extremum. Intuitively, this is why the empirical sample distribution $\hat{P}(w)$ approximates the target distribution $P(w)$. In contrast to other Monte Carlo statistical methods, e.g. importance sampling, the MH algorithm tend to suffer less from the curse of dimensionality and are therefore preferred for sampling in high-dimensional spaces [55].

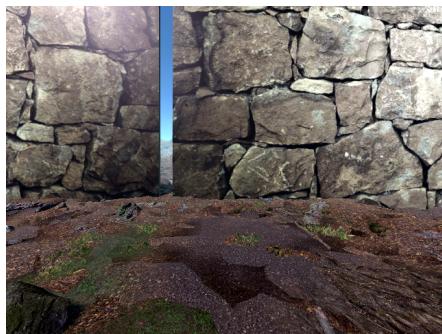
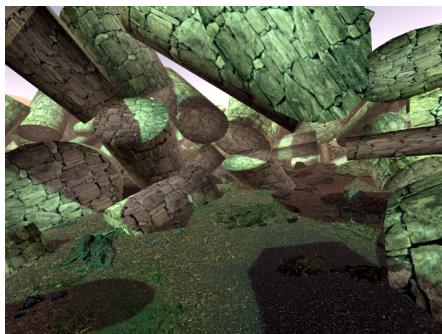
A Forest**B Objects****D Disaster****E City Street**

Fig. S4. Simulation Environments. The selected environments are challenging since they contain obstacles with complex geometry or narrow passages(C). Despite training only in simulated forests (A) and artificial environments with obstacles of convex shapes (B), our approach generalizes zero-shot to complex disaster zones (D) and urban environments (E). An illustration of the simulation environments can also be found in [Movie S2](#).

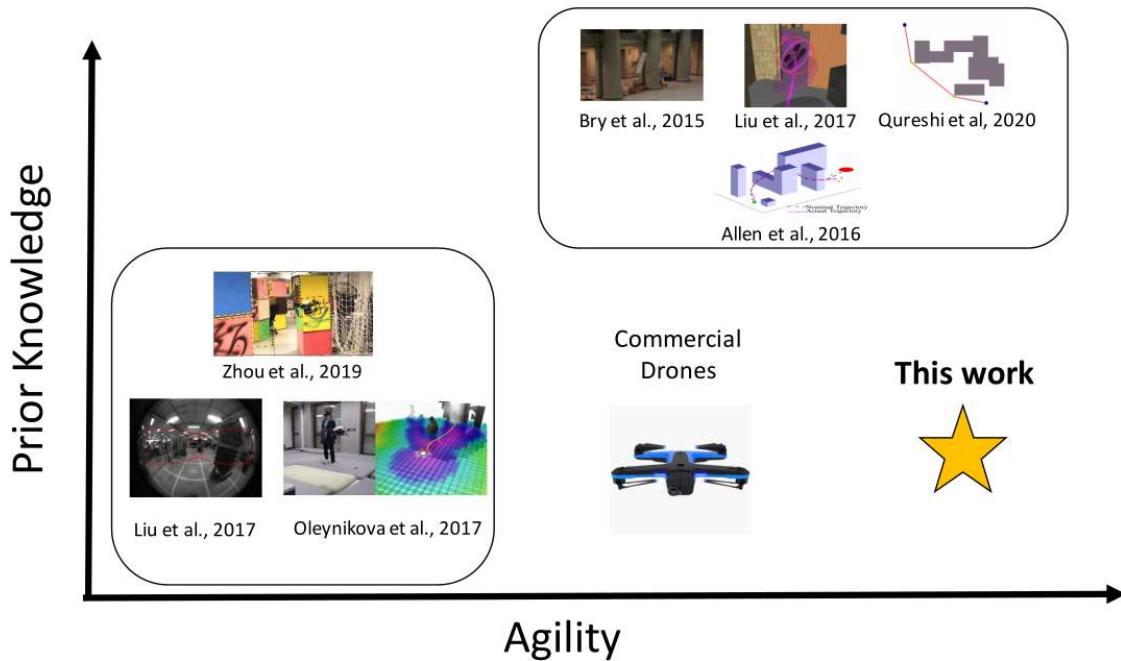


Fig. S5. Taxonomy of existing approaches for drone navigation in challenging and cluttered environments. Approaches are ordered with respect to the required prior knowledge about the environment and the maximum agility they achieve.

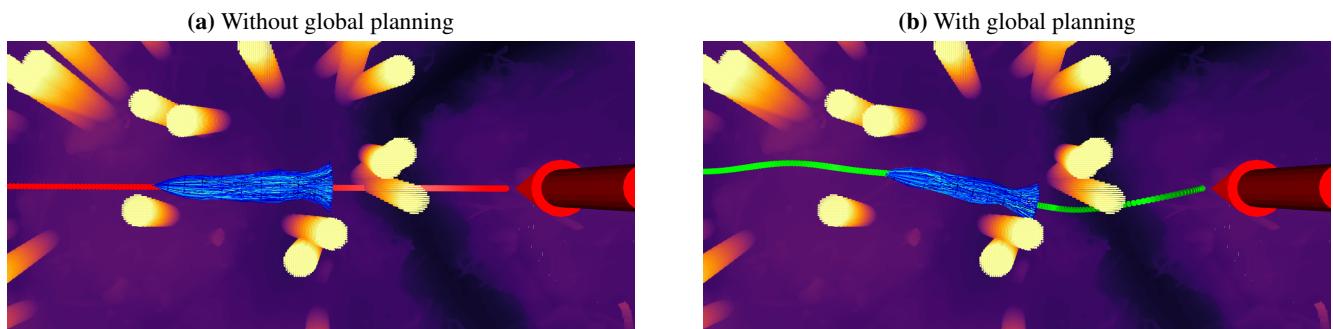


Fig. S6. Influence of global planning on the sampled trajectories. Sampling around the raw reference trajectory (in red) strictly limits the expert's sight to the immediate horizon (a). Conversely, sampling around a global collision-free trajectory (in green) results in a bias toward obstacle-free regions even beyond the immediate horizon (b). Best viewed in color.