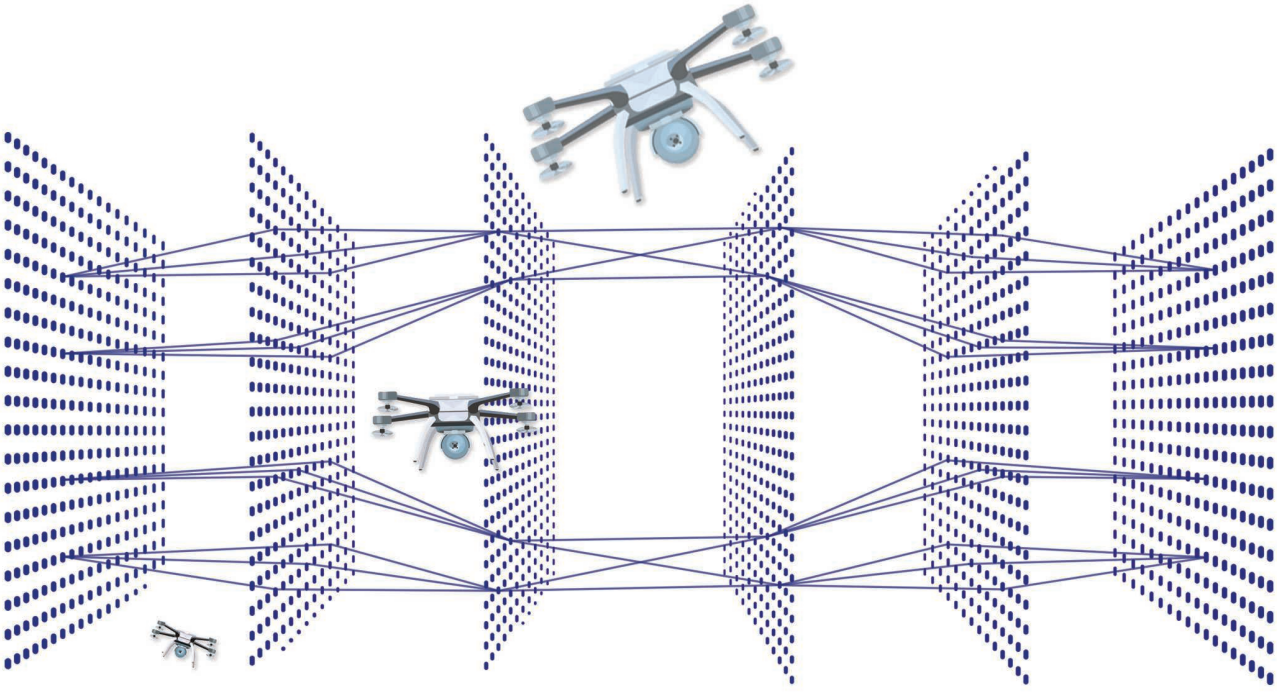
By Esen Yel, Taylor J. Carpenter, Carmelo Di Franco, Radoslav Ivanov, Yiannis Kantaros, Insup Lee, James Weimer, and Nicola Bezzo

Autonomous systems operating in uncertain environments under the effects of disturbances and noises can reach unsafe states even while using fine-tuned controllers and precise sensors and actuators. To provide safety guarantees on such systems during motion planning operations, reachability analysis (RA) has been demonstrated to be a powerful tool. RA, however, suffers from computational complexity, especially when dealing with intricate systems characterized by high-order dynamics, making it hard to deploy for runtime monitoring.

To deal with this issue, in this article, a neural network (NN)-based framework is proposed to perform fast online monitoring for safety, and an approach for the verification of NNs is presented. Training is performed offline using precise RA tools, while the trained NN is harnessed online as a fast safety checker for motion planning. In this way, at runtime, a planned trajectory can be quickly predicted to be safe or unsafe. When unsafe, a replanning procedure is triggered until a safe trajectory is obtained. The results of the trained network are tested for verification using our

Assured Runtime Monitoring and Planning

Toward Verification of Neural Networks for Safe Autonomous Operations



Digital Object Identifier 10.1109/MRA.2020.2981114

Date of current version: 14 April 2020

1070-9932/20©2020IEEE

recent tool Verisig, in which the NN is transformed into a hybrid system to provide guarantees before deployment. In the case of unverified NNs, the outputs of the verification are used to retrain the network until verification is achieved. Two illustrative case studies on a quadrotor unmanned aerial vehicle (UAV) (a pickup/drop-off operation and navigation in a cluttered environment) are presented to validate the proposed framework in simulations and experiments.

Safety Predictions

As autonomous vehicles find their way into our society, it becomes critical to guarantee safety against unpredictable uncertainties and disturbances during their operations at runtime. In fact, while model-driven motion planning and control techniques can be robust against noises and disturbances, they cannot prevent deviations from the desired behavior during system operation, potentially leading to unsafe states (e.g., crashing into an obstacle in the environment due to excessive wind disturbance). To assure safety during autonomous operations, it is necessary to take the effect of noises and disturbances into account during planning. Traditional RA tools, such as Hamilton–Jacobi reachability [1], and hybrid system RA techniques [2], [3] have proved to be very effective in providing safety guarantees by leveraging knowledge about the model of the system. However, their computational complexity makes them difficult to use at runtime, which is the subject of this article.

On the other hand, contrary to traditional RA tools, recent developments in machine learning have considerable potential to enable fast RA thanks to their efficiency and accuracy. Unfortunately, since the performance of such learning enabled components (LECs) depends significantly on the properties of the training data, it is challenging to provide guarantees in safety-critical operations. To deal with these challenges, this article presents a framework to verify LECs for fast, safe monitoring and planning of autonomous operations.

An NN trained to predict whether an operation will be safe or unsafe is verified if its output decision (safe or unsafe) always concurs with the results obtained by running the operation under the worst case conditions in which it was trained. Since it is unlikely that an NN with this strong property exists, in this article, we define one that is conservatively verified; in short, it is verified if, at the least, its safe decision output always concurs with the result obtained by running the actual operation. In other words, if the NN output is safe, the system can never reach an unsafe state; however, the NN is allowed to output unsafe when the system will be safe, since this is a conservative decision. This definition also holds in the extreme case that an NN outputs only unsafe decisions, in which a vehicle will be always safe, although it will not be able to move anywhere.

In the proposed scheme, an NN is trained to recognize safe and unsafe trajectories for an autonomous vehicle in an obstacle-populated environment. Specifically, training is performed, creating a library of trajectories and computing their reachable sets to make safety decisions; hence, the computational burden is limited to the offline stage. A trajectory is labeled safe if its reachable sets do not overlap with any obstacle in the environment; otherwise, it is marked unsafe.

Verification of the obtained NN is achieved using our recent technique Verisig [4], in which the NN is transformed into a hybrid system and the verification problem is cast as a hybrid system reachability problem. If the NN is not verified, meaning that it is not safe to be used, the output of the verification is employed as feedback to retrain the NN more conservatively until it is verified. Once the NN is verified, it is used at runtime as a safety checker for the planned trajectories from an untrained initial state under the effect of unknown online disturbances. If unsafe, a new trajectory is planned and tested until a safety-guaranteed course is obtained, if possible.

Contributions of the article include the following: with the proposed framework, we 1) develop a fast safety checking and replanning approach for autonomous vehicles' operations in cluttered environments under unknown runtime disturbances and 2) leverage a novel NN verification tool, Verisig, to verify and retrain the used NN as a fast safety monitor, before its deployment.

As a result, with this framework, it is possible to eliminate the need for computationally expensive RA tools for planning safe trajectories at runtime, and our verification method, Verisig, is capable of assessing the validity of the proposed NN. To better illustrate our proposed framework, two case studies on a quadrotor UAV are presented with simulations and experiments under the presence of unknown disturbances during runtime: 1) a pickup/drop-off mission and 2) safe navigation in a cluttered environment.

Verified Safe Motion Planning

Our verified safe motion planning framework consists of offline and online stages, as depicted in Figure 1. During the offline phase, a library of trajectories is generated. We parametrize the trajectory generation based on the initial state of the UAV, the desired goal position, the location of the obstacles on the way to the goal site, and the distance that must be maintained from these obstacles. These trajectory parameters are labeled safe or unsafe using RA, and an NN is trained with this set of parameters. To be able to use the NN in autonomous operations, it should be guaranteed that the trained network never outputs safe when the trajectory is actually unsafe. To provide this guarantee, we verify the trained NN using our recent tool Verisig, as outlined in the "Verification" section. In case the NN is not verified, the Verisig output is utilized to retrain the NN more conservatively (i.e., it outputs more unsafe decisions) until it is verified. Once the NN is verified, it is used to make decisions about the safety of a new set of trajectory parameters at runtime. If the NN decides that the trajectory is safe, the UAV

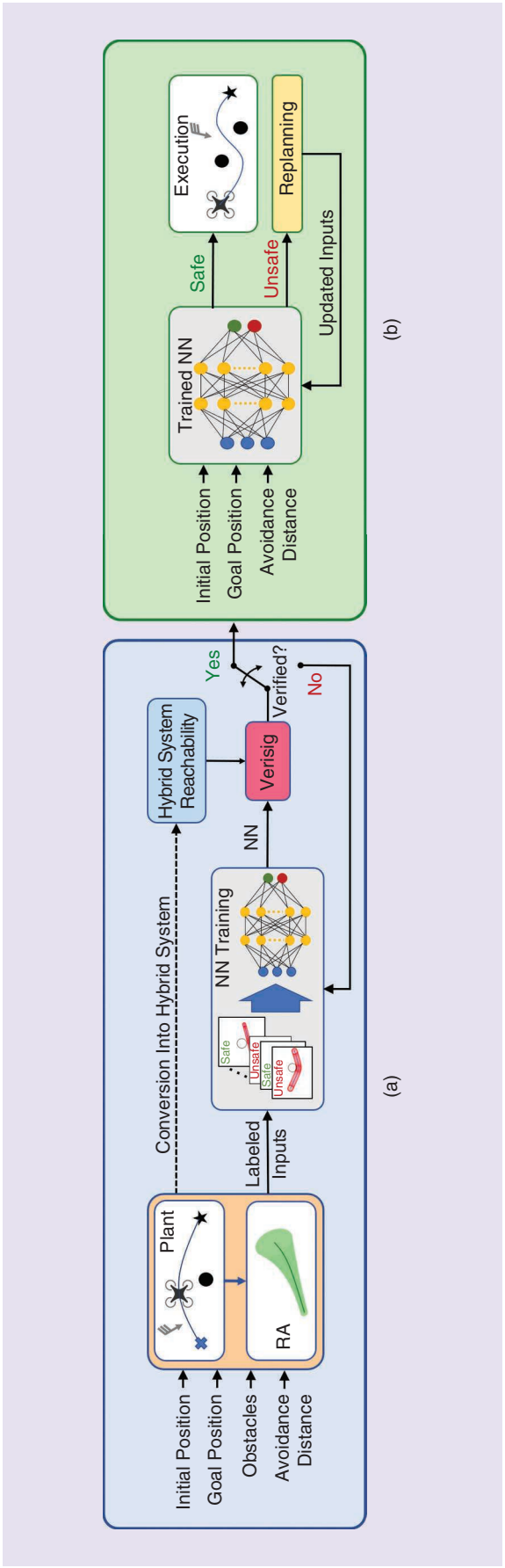


Figure 1. The architecture of the proposed framework for verifying NNs for runtime monitoring and planning autonomous operations. (a) During the offline stage, an NN is trained and verified, followed by its deployment at runtime for monitoring and replanning purposes. (b) The online stage.

executes the course. If the decision is unsafe, the trajectory is replanned by changing the parameters until a safe choice is obtained.

Reachability Analysis

As mentioned previously, RA is a very powerful method for computing the sets that a system could reach starting from an initial set. A hybrid system RA tool, Flow* [2], uses Taylor models to compute flowpipe overapproximations of the dynamics, while dReach [3] encodes the reachability problem as first-order formulas across real numbers and solves the problem using δ-decision procedures. Hamilton–Jacobi RA is also a widely used approach to provide guarantees for the safety of safety-critical systems' optimal system trajectories [5]. It performs well in terms of the generality of system dynamics, flexibility in the representation of sets, and control policy computation; however, it suffers from computational scalability [6]. A significant effort has been made to overcome the scalability problem for high-dimensional systems, such as decomposing the system dynamics [7] and using NNs to approximate the reachable sets [6], [8]. Other types of reachable sets, such as robust control invariant tubes [9], have also been proposed for safety-guaranteed UAV planning. All these traditional RA tools are very effective in providing safety assurances, although their computational complexity makes them difficult to use in making runtime safety decisions.

In the literature, there have been some efforts to make RA more usable for runtime applications. For example, in [10], the authors precomputed a library of trajectories and funnels (analogous to reachable sets) offline and combined these trajectories online to navigate in a priori unknown environments under disturbance effects. However, with this approach, the system is restricted to a discrete set of motion primitives. In [11], using Hamilton–Jacobi reachability, a lookup table is computed offline to find the bounds on the planned trajectory that are used to augment the obstacles to guarantee collision-free behavior under bounded disturbances in unknown environments. Similarly, in [12], forward reachable sets are computed offline for parameterized trajectories, and at runtime, safe trajectory parameters are picked to avoid the sensed obstacles in unknown environments with model uncertainties. Different from these works, our framework solves this problem of safe navigation in known and unknown environments under the presence of external disturbances by using verified NNs to perform safety decisions at runtime, leaving the RA computation offline. Our framework is also general and modular, meaning that any type of control and planning method can be considered. It is also independent from the choice of reachability analysis tool. Specifically, during the offline stage, reachable sets are generated for a given trajectory, and if they do not intersect with obstacles, the corresponding trajectory parameters are labeled as safe:

$$\begin{cases} s_\tau = 1 & \text{if } \boldsymbol{R}(\boldsymbol{p}_\tau, t) \cap \boldsymbol{p}_{o,j} = \varnothing \\ & \forall t \in [0, T], \forall j \in \{1, \ldots, N_o\}, \\ s_\tau = 0 & \text{otherwise} \end{cases} \tag{1}$$

where $\boldsymbol{p}_\tau$ is the desired positions along the trajectory τ, $\boldsymbol{R}(\boldsymbol{p}_\tau, t)$ is the corresponding reachable set, s_τ is the safety label, $\boldsymbol{p}_{o,j}$ is the jth obstacle position, and N_o is the number of obstacles in the environment. An NN is trained using these safety-labeled parameters to make decisions for trajectories with different conditions.

NN Training for Safety Decisions

Following the diagram in Figure 1, after collecting a rich library of labeled trajectory parameters, we train an NN to provide safety decisions without having to run computationally expensive RA operations at runtime. The inputs to the NN are the trajectory parameters, which are the initial and final positions of the system, and the obstacle-avoidance distance. The output of the NN is a binary safe/unsafe decision. Since these conclusions are used by a safety-critical system, it is required that the NN never outputs a decision of safe if the trajectory is unsafe, as it can lead to dangerous outcomes (e.g., colliding with an obstacle). Therefore, we are interested in training an NN with zero false positives (FPs). The drawback of reducing the number of FPs is that the number of false negatives (FNs) (safe trajectories marked unsafe) may increase. Even a conservatively trained NN can output a safe decision for a set of untrained, unsafe trajectory parameters. Therefore, the absence of such FPs needs to be verified prior to its deployment.

Verification

In this article, we use Verisig to verify that the trained NN does not output safe if the robot plans an unsafe trajectory. As described in our prior work [4], Verisig was developed to verify safety properties of closed-loop systems with NN components. Verisig focuses specifically on sigmoid-based NNs and works by transforming the NN into an equivalent hybrid system. The NN's hybrid system is then composed with the plant, resulting in a new hybrid system that describes the entire closed-loop system, as depicted in Figure 1. This enables us to cast the verification problem as a hybrid system reachability problem, which is solved by an optimized hybrid system verification tool, such as Flow* [2]. Specifically, Verisig transforms the NN S into a hybrid system H_S by noting that the sigmoid derivative can be expressed in terms of the sigmoid itself, i.e.,

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)),$$

where $\sigma(x) = 1/(1 + \exp(-x))$ is the sigmoid. With this observation in mind, we introduce a proxy function $\sigma_p(t, x) = \sigma(xt)$ such that $\sigma_p(1, x) = \sigma(x)$ and

$$\dot{\sigma}_p(x) = x\sigma_p(t, x)(1 - \sigma_p(t, x))$$

using the chain rule. In other words, σ_p can be considered as a state in a dynamical system that starts at $\sigma_p(0, x) = 0.5$ and is equal to $\sigma(x)$ at time 1. Thus, each neuron in the NN can be mapped to a state in a hybrid system, and each layer can be mapped to a mode. Transitions between modes occur when $t = 1$. Note that this time is local to the NN; global time does not progress during the NN's execution.

Although Verisig was originally applied to closed-loop systems with NN controllers, it applies to the setup considered in this article, as well. In particular, in the verification problem, the NN's hybrid system does not interact with the plant directly; rather, for a given set of initial conditions, we compute the reachable set for the NN's output and check whether it is possible for the robot to crash (when started from that initial set) while the NN outputs safe.

The composed hybrid system $H = H_q \| H_S$ is shown in Figure 2, where H_q is the hybrid system describing the robot's

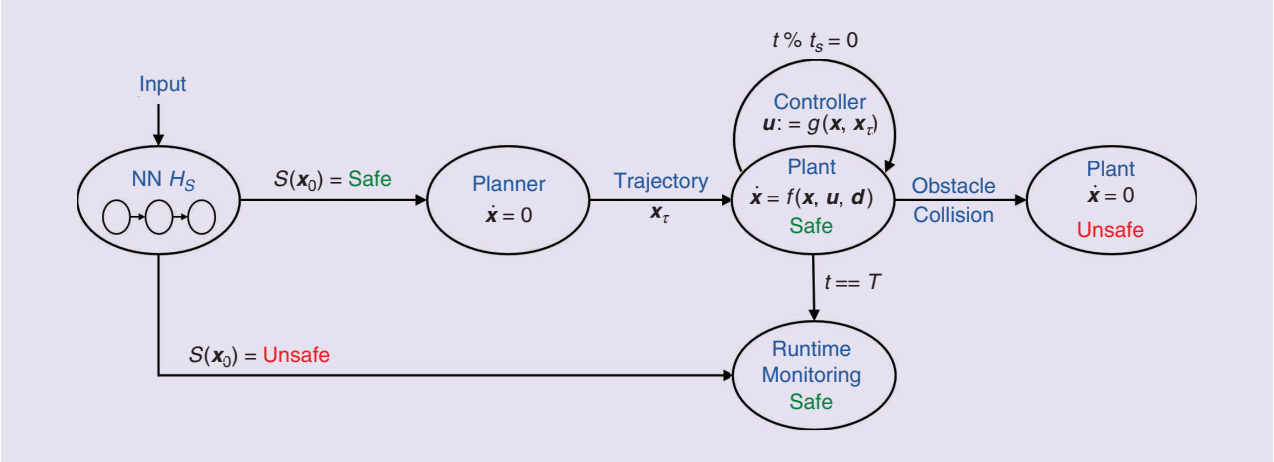


Figure 2. The composed hybrid system considered for verifying NNs for runtime monitoring. The plant dynamics evolve as a function of the state, input, and disturbance $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{d})$. A controller is designed to follow the desired trajectory $\boldsymbol{x}_\tau$ by generating a control input with sampling time t_s, $\boldsymbol{u} = g(\boldsymbol{x}, \boldsymbol{x}_\tau)$. The goal is to verify that the plant (Unsafe) mode is never reached during the duration T of the mission, i.e., that the NN never outputs safe when the plant is unsafe.

dynamics. H contains the union of modes and states of H_q and H_S. H starts in the initial mode of H_S and adds a transition from the last mode of H_S to the initial mode of H_q, at which point the H_q execution begins. The goal is to verify that H does not enter the plant (Unsafe) mode when $S(\boldsymbol{x}_0) = \text{Safe}$.

Given a hybrid system description of the closed-loop system, one could use a tool such as Flow* to verify the system's safety. Note that, since hybrid system verification is undecidable in general, the typical approach used in these tools is to overapproximate the reachable sets. If the overapproximation does not contain any unsafe states, the system is safe. If the overapproximation contains safe and unsafe states, the outcome is unknown, since the unsafe states could be spurious; i.e., they do not exist in the true reachable set but only in the overapproximated one. Finally, if all states are unsafe, the system is unsafe. Various shapes have been explored to overapproximate the reachable sets, including polytopes, ellipsoids, and hyperrectangles. Flow* uses a Taylor model approximation, which is a Taylor series approximation with worst case error bounds. Taylor models scale well when used with interval analysis and are shown to have a low approximation error for a large class of nonlinear systems [2].

NN Retraining

At the end of the verification, Verisig specifies the regions in which 1) the NN is safe to be used (i.e., the plant is not unsafe when the NN output is safe), 2) the NN is not safe to be used (i.e., the plant is unsafe when the NN output is safe), and 3) the system is not able to make a decision due to the approximation errors introduced in the hybrid system RA during verification (i.e., the NN output is "unknown"). In case there are regions in which the NN is not safe to be used or Verisig cannot decide, the NN needs to be retrained. The output of Verisig can be leveraged to retrain the NN in several ways. One is to collect more data around those regions where Verisig is not able to make a decision, followed by NN retraining. An increased density of data around previously untrained regions may help with the verification.

However, how much data needs to be collected in those regions is not known a priori and is hard to predict, so the process could require multiple iterations of data collection and retraining. In addition, collecting new data to improve the training set may not always be possible. Instead, we propose adding points from the unsafe/unknown regions obtained from the Verisig output to the existing training set, marking them with unsafe labels, and finally retraining the NN. By retraining the NN with more unsafe points, a more conservative version is obtained in which unsafe regions are inflated, helping with the verification process. This retraining process is repeated until the NN is verified.

Case Studies

As a proof of concept, our verified safe monitoring and planning approach is applied to two case studies of quadrotor motion planning: 1) a pickup/drop-off mission and 2) a navigation operation in a cluttered environment. Both case studies use similar NNs to predict whether the vehicle trajectory will be safe or unsafe (and thus require offline training and verification). At runtime, the verified NN is used for different purposes. The pickup/drop-off task requires the UAV to go from one side of a static environment to the other, resembling operations that could happen inside a warehouse or factory. The UAV makes decisions about the safety of the planned trajectory based on the NN results and replans by adjusting the obstacle-avoidance distance until it finds a longer but safe course to its goal position. In the latter case study, the UAV is tasked with navigating a previously unknown cluttered area. Training is executed in a smaller environment with only one obstacle, acting as a primitive scenario that can appear and be composed multiple times at runtime. Training in a primitive environment enables the NN and verification to be generalized to different settings with the same type of obstacles located in previously unknown positions. Replanning here is executed by querying different waypoints along the path to the goal until the NN outputs a safe decision. In both case studies, we use the same vehicle, controller, planner, and disturbances, whose models are briefly summarized in the following section.

System Models

Quadrotor UAV and Controller Model

A quadrotor can be modeled using the following simplified sixth-order state vector $\boldsymbol{x} = [x \;\; y \;\; z \;\; v_x \;\; v_y \;\; v_z]^\top$, where x, y, and z are the world frame positions and v_x, v_y, and v_z are the world frame velocities. The quadrotor dynamics can be defined as $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{d})$, where $\boldsymbol{u} = [F \;\; \phi \;\; \theta]^\top$ is the input vector with thrust, roll, and pitch commands and $\boldsymbol{d} = [d_x \;\; d_y \;\; d_z]^\top$ is the external disturbance vector. The dynamics can be described as

$$[\dot{x} \;\; \dot{y} \;\; \dot{z}]^\top = [v_x \;\; v_y \;\; v_z]^\top$$

$$\begin{bmatrix} \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \end{bmatrix} = \begin{bmatrix} g\theta \\ -g\phi \\ \frac{F}{m} - g \end{bmatrix} + k_d \begin{bmatrix} d_x - v_x \\ d_y - v_y \\ d_z - v_z \end{bmatrix},$$

where m is the mass of the quadrotor, g is gravity, and k_d is the drag coefficient. It should be noted that, in this article, a simplified quadrotor UAV model is used to alleviate the verification problem. Validating a high-fidelity model [13] is left for future work. To generate the necessary roll, pitch, and thrust inputs to follow the desired trajectory, a cascaded set of PID controllers is used [14].

Disturbance Model

The external disturbance considered in this article is bounded in magnitude: $\|\boldsymbol{d}\| \leq D_{\max}$, $\forall \boldsymbol{d} \in \mathcal{D}$, where $D_{\max}$ is the upper bound to the disturbance magnitude and $\mathcal{D}$ is the set of all possible disturbances. Here, we assume that the online disturbance is unknown but constant through time. This is a reasonable assumption, as wind disturbance generally follows a Brownian motion and does not change erratically during short periods of time [15], [16].

Trajectory Planning

Obstacle-avoidance trajectories are computed using a simple geometric approach. Specifically, if an obstruction is present along the way to the goal position, a waypoint is added to the path at a specified avoidance distance r_a away from the obstacle center. A course is finally generated to visit all waypoints on the path by using a minimum jerk trajectory generation [17]. It should be noted that we use this path-planning method due to its simplicity in implementation in simulations and experiments; however, the overall proposed framework is independent from the choice of path-planning approach.

Pickup/Drop-Off Task

The first case study that we present in this article is a pickup/drop-off task, an operation that is commonly used in factory applications where a vehicle moves back and forth between a warehouse and a workstation. The environment has a designated pickup area (warehouse) and drop-off position (workstation), with obstacles at known locations in between. The vehicle is tasked to move from a point inside the pickup area to the drop-off location. Once it reaches the drop-off site, it can move back to a new point in the pickup area. To complete its mission safely, the UAV needs to decide whether the planned trajectory, parametrized by the initial position $\boldsymbol{p}_0$, final goal $\boldsymbol{p}_g$, and avoidance distance r_a, is safe and if not, replanning is required. In this case, replanning is executed by adapting r_a.

To train an NN to make safety decisions in this scenario, two sets of trajectories with different avoidance distances are generated and labeled using RA: one set links a rich set of initial positions in the pickup area to the drop-off position, and the other connects the drop-off position to a rich set of final positions in the pickup area. The NN queries the initial and final positions and the avoidance distances; if they are unsafe, it checks a larger avoidance distance until it outputs a safe decision.

Simulation-Based Reachability

In this case study, we use a simulation-based RA. During the offline stage, we run each training trajectory under the worst-case scenario which, in our example, is the largest possible disturbance attainable in the environment. Under this condition, for a given trajectory $\boldsymbol{p}_\tau$, the maximum deviation d_m is calculated as follows:

$$d_m = \max_{\boldsymbol{d}\in\mathcal{D}} \max_{t\in[0,T]} \min_{\xi\in[0,T]} \|\boldsymbol{p}_{\boldsymbol{d}}(t) - \boldsymbol{p}_\tau(\xi)\|, \tag{2}$$

where $\boldsymbol{p}_{\boldsymbol{d}}$ is the position of the vehicle under disturbance $\boldsymbol{d}$. Here, d_m is used as an upper bound for the actual deviation from the trajectory, and it is conservative since it is the maximum deviation measured through the entire trajectory. The position-reachable sets are then generated as follows:

$$\boldsymbol{R}(\boldsymbol{p}_\tau, t) = \{\boldsymbol{p}(t) : \|\boldsymbol{p}(t) - \boldsymbol{p}_\tau(t)\| \leq d_m\}. \tag{3}$$

After generating the reachable sets, the trajectory is labeled safe or unsafe according to (1). In Figure 3, we show the reachable sets of two sample trajectories.

Offline Training

The environment has a designated rectangular pickup area that is limited between [0.0, 1.3] m in the x-axis and [−1.0, 1.0] m in the y-axis. The drop-off point is located at $\boldsymbol{p}_g = [4.0, 0.0]$ m. There are two obstacles between the pickup area and drop-off location, positioned at $\boldsymbol{p}_{o1} = [2.0, 0.1]$ m and $\boldsymbol{p}_{o2} = [3.0, -0.1]$ m. For training, 294 points are uniformly distributed in the pickup area and used as the initial and final positions. For trajectory generation, seven different avoid distances are considered: $r_a \in \{0.3,\ 0.35, 0.4, 0.45, 0.5, 0.6, 0.7\}$ m.

Two NNs were trained: one for the drop-off operation and the other for the pickup task. To implement the NNs, we chose Keras (https://keras.io), a deep-learning library capable of running on top of Tensorflow (https://tensorflow.org) through a set of application programming interfaces written in Python. For all layers (input, hidden, and output), we use a sigmoid activation function. The NN is composed of three input nodes (the x-y initial position and avoidance distance pair), one hidden layer of 40 nodes, and one output that determines whether the label is safe or unsafe. We trained two different NNs, one

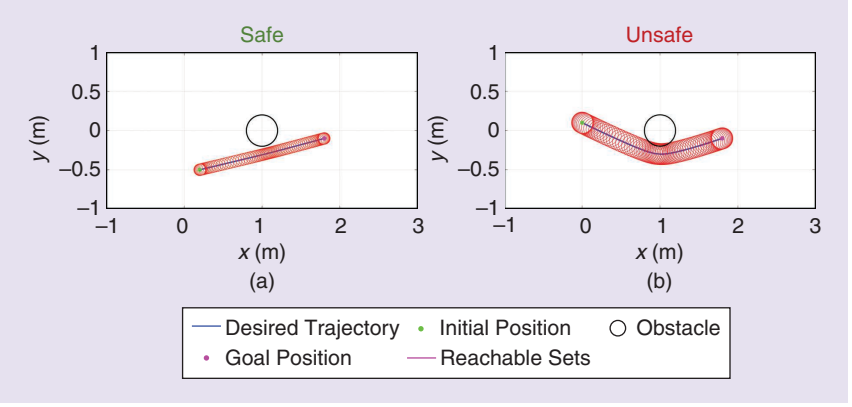


Figure 3. The reachable sets for two sample trajectories. (a) A safe trajectory. (b) An unsafe trajectory.

for each subtask (drop-off and pickup). The training results showed 0% FPs and roughly 5.2% FNs for the first NN and 0% FPs and approximately 1.2% FNs for the second one.

In Figure 4, the initial positions in the training set for the drop-off and pickup operations are presented with their respective labels and the NN results inside the proposed workspace. Each subfigure denotes a different avoidance

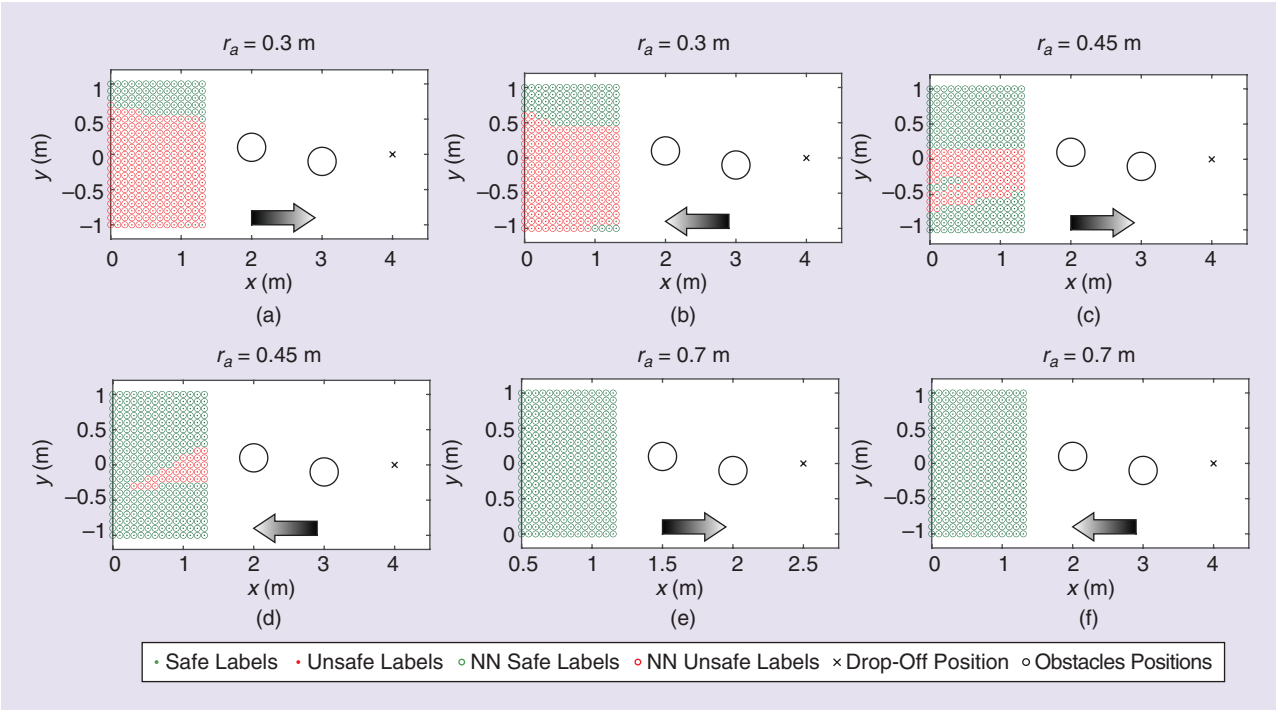


Figure 4. Safety maps for the initial and final positions in training sets with different avoidance distances. The (a) drop-off task and (b) pickup mission for $r_a = 0.3$ m. The (c) drop-off task and (d) pickup mission for $r_a = 0.45$ m. The (e) drop-off task and (f) pickup mission for $r_a = 0.7$ m.

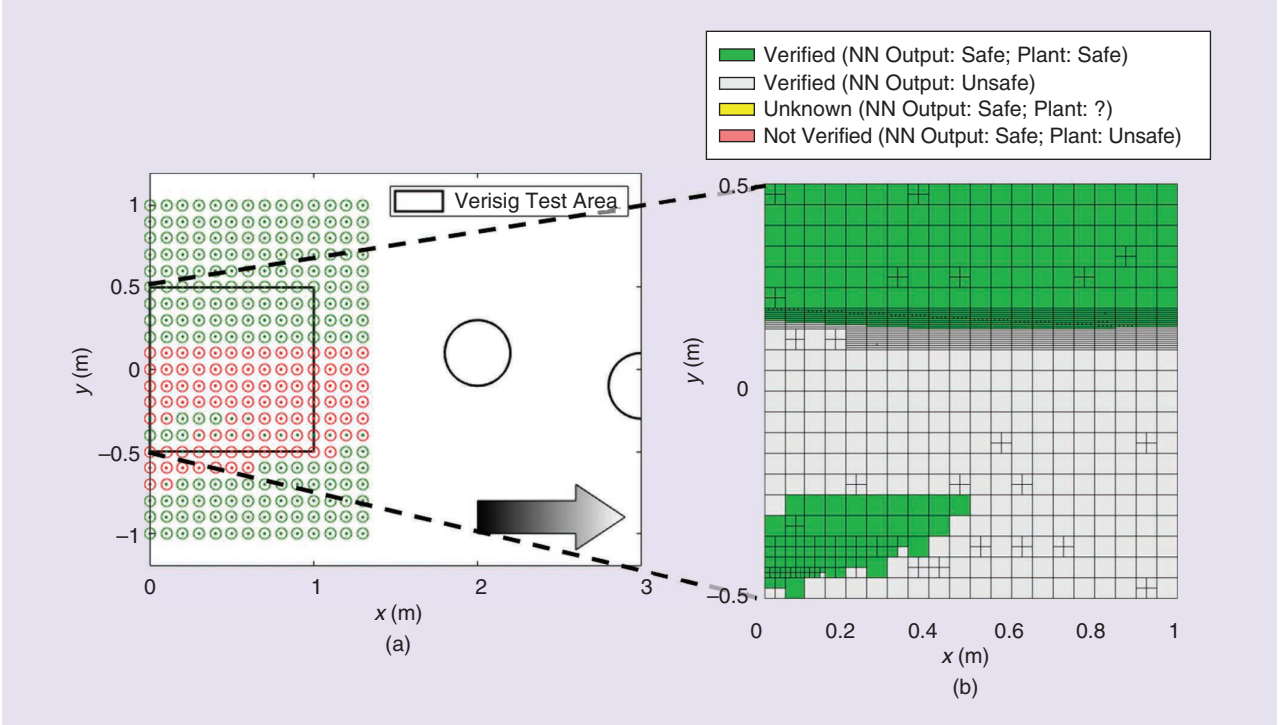


Figure 5. (a) The pickup task with avoid distance $r_a = 0.45$ m in Figure 4(c). (b) The initial set was divided into small subsets and verified. No unsafe sets were obtained.

distance marked on top of the figure. Due to space constraints, we show examples of data for only three avoidance distance values: $r_a = 0.3$ m, $r_a = 0.45$ m, and $r_a = 0.7$ m. Arrows inside the workspace indicate the direction of motion of the vehicle. Inside each subfigure, the green (red) dots represent initial positions from which the trajectories to the goal are labeled safe (unsafe) using reachability analysis. The green (red) circles around the dots denote the decisions of the NN on the same training points. As can be noticed and as expected, when the avoidance distance increases, the number of safe initial positions also increases in both missions because the distance between the desired trajectories and the obstacles becomes larger. Therefore, increasing the avoidance distance improves safety; however, the routes become longer, which generally is not desirable due to energy concerns.

Verification Results

Verification was performed with the methods in the "Verification" section. Here, we present the results for the second drop-off task shown in Figure 4, namely, the case in which the UAV starts in the set $x_0 \in [0, 1]$, $y_0 \in [-0.5, 0.5]$ and aims to reach the goal at [4, 0] m, with $r_a = 0.45$ m and disturbances d_x, $d_y \in \{-0.1, 0.1\}$ m/s.

The verification results are presented in Figure 5. We divided the initial set into smaller subsets and verified each one separately to keep the approximation error in Flow* small enough. The size of these subsets [i.e., the 5-cm boxes in Figure 5(b)] was chosen after some preliminary testing; these subsets were large enough to verify the majority of the initial set with a small approximation error. Some subsets were further refined when an instance resulted in an error that was too large. Refinements were necessary at the NN's decision boundary as well as for sets that triggered multiple if-cases in the planner (e.g., when planning around multiple obstacles). The verification was performed using Amazon

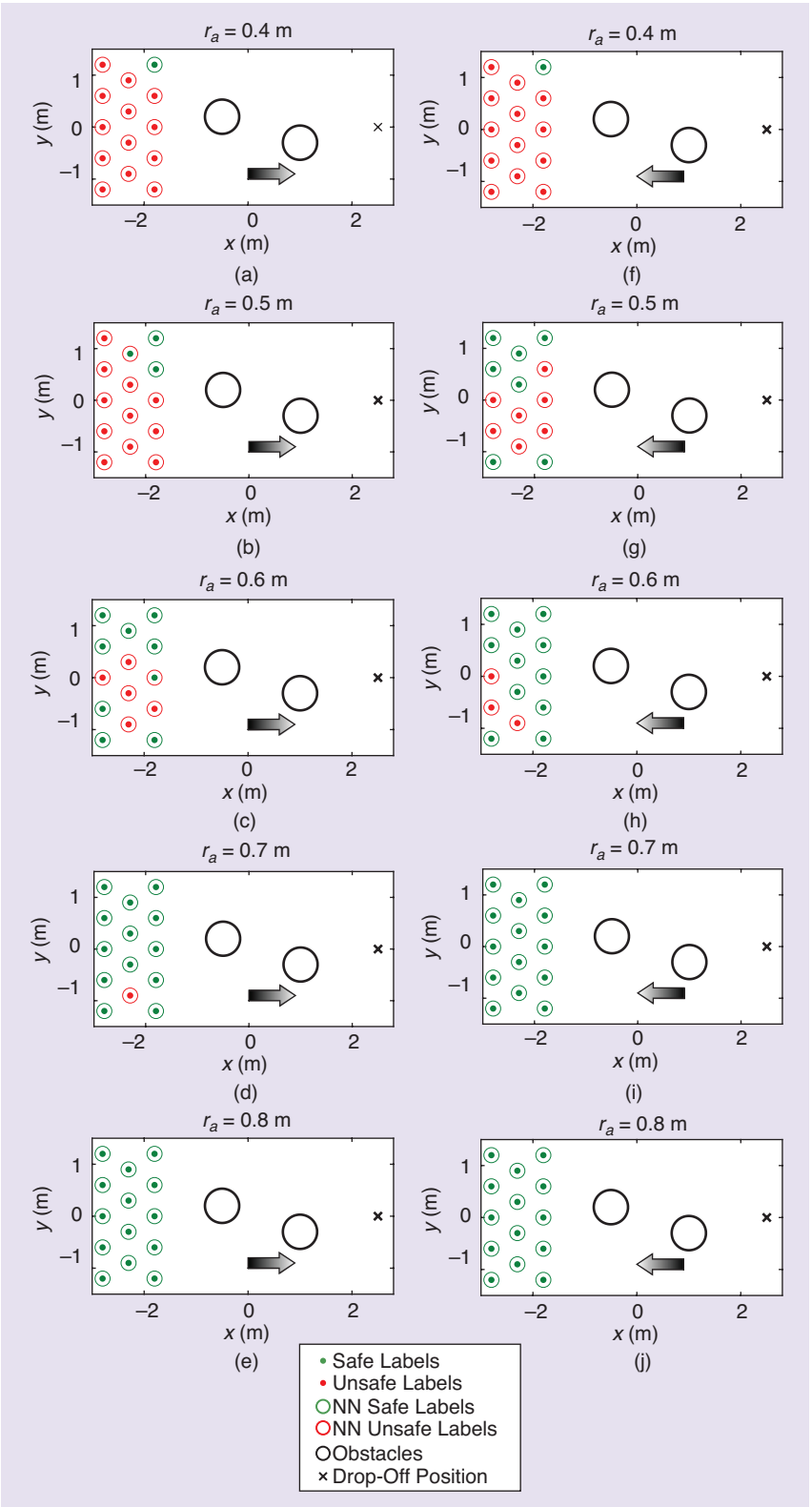


Figure 6. The safe and unsafe training and NN results for the pickup and drop-off tasks in the area where experiments were performed. An FP (red dot inside a green circle) corresponds to an unsafe label (red dot) with a safe NN decision (green circle), while an FN (green dot inside a red circle) is a safe label (green dot) marked as unsafe (red circle). The number of FPs is zero for both NNs, and there are two and zero FNs for the first and second NN, respectively. The drop-off tasks where (a) $r_a = 0.4$ m, (b) $r_a = 0.5$ m, (c) $r_a = 0.6$ m, (d) $r_a = 0.7$ m, and (e) $r_a = 0.8$ m. The pickup tasks where (f) $r_a = 0.4$ m, (g) $r_a = 0.5$ m, (h) $r_a = 0.6$ m, (i) $r_a = 0.7$ m, and (j) $r_a = 0.8$ m.

Web Services (https://aws.amazon.com). Each subset took roughly an hour to verify, although some took longer due to branching introduced by if-cases in the planner.

Figure 5 closely matches the corresponding graph in Figure 4(c). We confirmed that the NN was, indeed, conservative so that no unsafe events occurred when its output was safe. The same procedure can be used for all other cases. Note that we verified the safety for only the first NN as a proof of concept; the procedure for the second NN would be exactly the same.

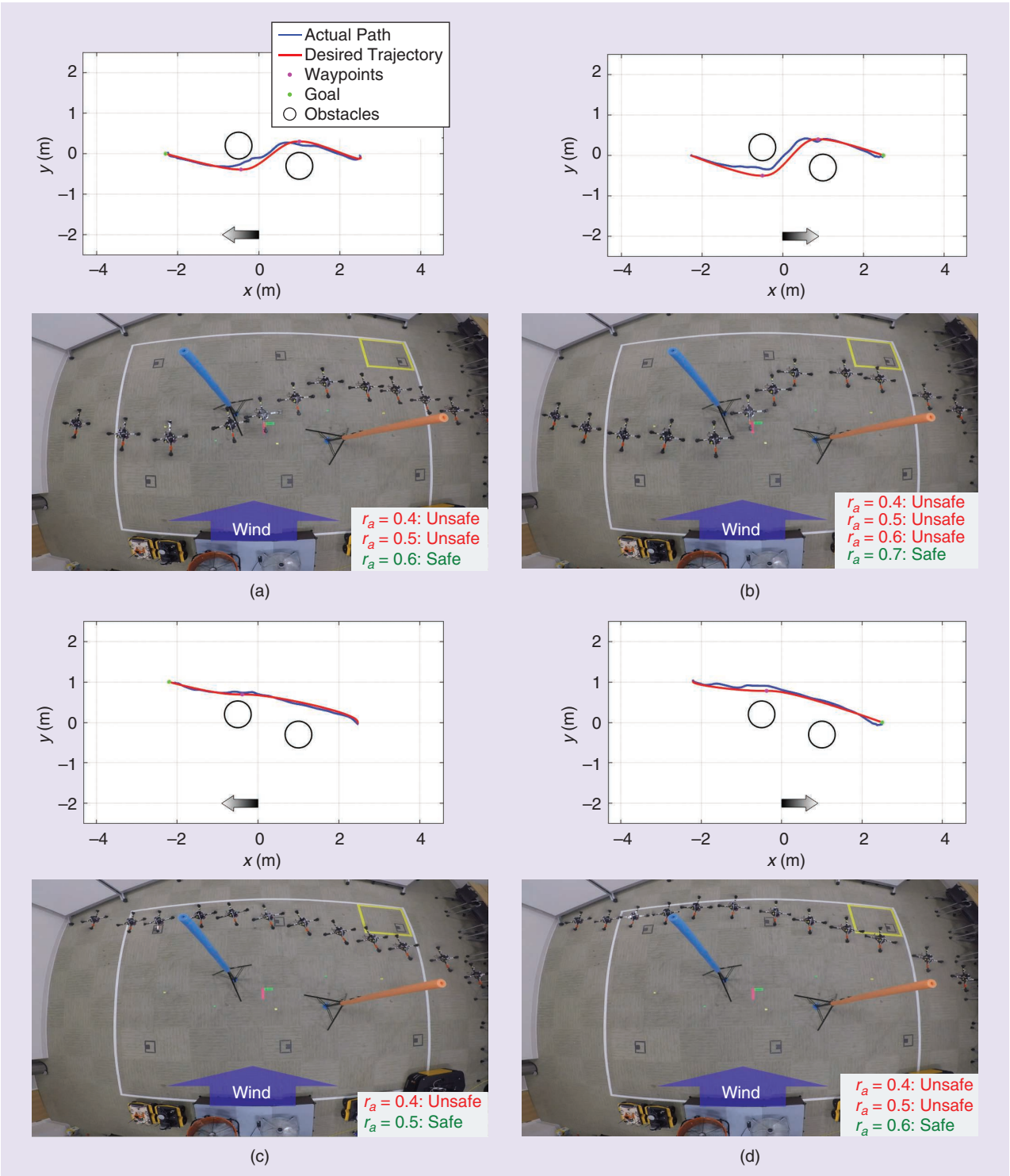


Figure 7. The experimental results in which the trained NN was used to make safety decisions and replan accordingly. The desired versus the actual trajectories are presented in the top row of subfigures, with the relative snapshots from the experiments in the bottom row. (a) The round 1 pickup task. (b) The round 1 drop-off task. (c) The round 2 pickup task. (d) The round 2 drop-off task.

Experimental Results

To test our framework, we designed a similar pickup/drop-off scenario (see Figure 6) in which the quadrotor was tasked to visit different points in the pickup area and return to the drop-off location every round while avoiding two obstacles (Figure 7). Real flights were performed with an AscTec Hummingbird quadrotor controlled through the Robot Operating System. A Vicon motion capture system was used to track the position of the quadrotor and provide ground truth position information. Two industrial fans blew wind in the middle of the area, creating a disturbance toward the obstacles. To generate safe and unsafe labels for the real scenario, 14 positions equidistant from one another in the pickup area were considered. For every avoidance distance $r_a \in \{0.4,\ 0.5,\ 0.6,\ 0.7,\ 0.8\}$ m, trajectories were generated from each starting position to the drop-off goal, and vice-versa, and safety decisions about these routes were made using a similar approach explained in the "Reachability Analysis" section. A trajectory was labeled unsafe if the maximum deviation from the desired course became larger than the distance between the obstacle and the desired route at any point along the path. We trained two NNs, one for each subtask. In Figure 6, the safety decisions are shown when the training set is provided as input to the NN.

During testing, we exploited the trained NN by executing multiple passes back and forth between different points in the pickup area through an unknown disturbance, generated in the middle of the area, that could push the quadrotor toward the obstacles. Throughout the experiment, the vehicle was tasked to navigate using the smallest avoid distance $r_a = 0.4$ m, if possible. As expected, the NN generated unsafe outputs for some of the points. Consequently, r_a was increased by 0.1-m increments until a safe decision was returned by the NN before sending the vehicle to the goal. The computation time for the NN to make a safety decision was on the order of milliseconds and constant for all trajectories, making it suitable for online replanning operations. Conversely, the simulation-based reachability used during training is not suitable at runtime, as its computation time increases linearly with the duration of the trajectory. Figure 7 shows the sequence of snapshots, the decisions of the NN, and the comparison between the desired and actual trajectories (upper rows) for two rounds of the operation.

Finally, we ran an experiment in which we generated trajectories with the minimum avoid distance $r_a = 0.4$ m, using the wind disturbance from the previous experiment and disregarding the decision from the NN. During the first round, which was predicted to be unsafe with $r_a = 0.4$ m, the quadrotor collided with an obstacle (Figure 8), confirming the unsafe decision predicted by the NN.

Navigation in Cluttered Environments

The other case study that we demonstrate in this article is a navigation operation in a cluttered environment, such as a heavily forested area. A UAV was tasked to reach a goal position in an area in which obstacles were scattered in a priori unknown locations discoverable at runtime as the system moved toward the final objective. To plan safety-guaranteed trajectories, we considered a smaller environment with only one obstacle in the center and trained and verified an NN to predict the safety of the routes within this smaller region. The trained and verified primitive space could then be fitted and composed multiple times at runtime to assess safety in larger regions with more obstacles.

Throughout a mission, every time the UAV encounters an obstacle along its path, it selects an intermediate goal point around the obstruction and queries the trained NN about the safety of the trajectory to the selected target. If unsafe, a new intermediate goal is queried until the output of the NN is safe. This procedure is repeated multiple times for each obstacle along the path until the vehicle reaches the final target position.

Offline Training

To train the NN for this operation, we used a small, box-shaped workspace with one obstacle in the center, as shown in Figure 9. We picked 44 initial and 44 final positions uniformly distributed across the start (to the left of the obstacle) and goal (to the right of the obstacle) regions. The safety of the trajectories generated from those 1,936 start–goal position pairs

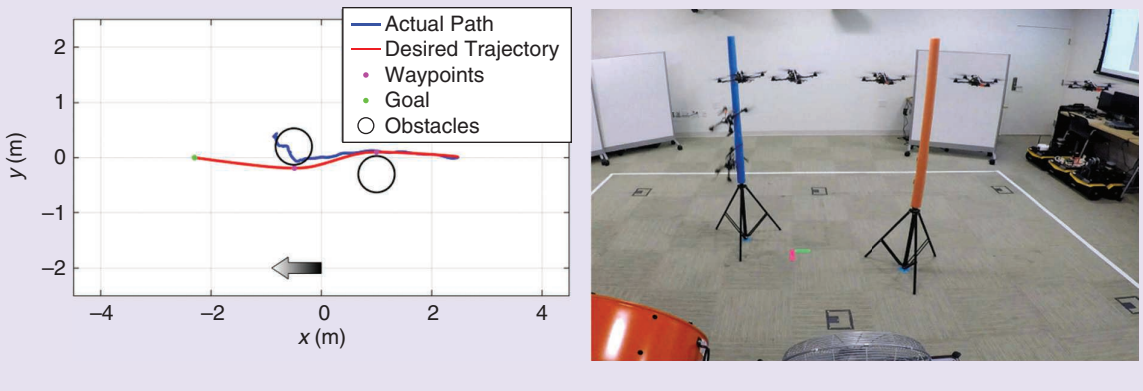


Figure 8. The trajectory followed by the quadrotor for the first pickup task in Figure 7(a), with $r_a = 0.4$ m and marked unsafe, resulting in a crash.

was decided offline using Flow* RA [2]. On average, it took roughly 5 min for Flow* to make a safety decision for one start–goal position pair under bounded disturbance conditions $d_x, d_y \in [-0.4, 0.4]$ m/s, reinforcing the fact that RA is expensive to perform at runtime.

Using this set of initial–final position pairs, an NN was trained to predict the safety of an untrained pair of initial–final goals. The NN was composed of four input nodes (the *x*-*y* initial position and *x*-*y* goal position pair), one hidden layer of 40 nodes, and one output, which determined whether the label was safe or unsafe. The NN performed with 0% FPs and roughly 0.2% FNs. In Figure 9, we present examples of labels and NN decisions for trajectories starting from three different initial positions to all of the final goals in the training set. Similar to the previous case, a green (red) dot represents a final position in which the trajectory was labeled safe (unsafe) from a given starting point, and a green (red) circle represents the NN decision for the same point.

NN Verification

Similar to the previous case study, since the results of an NN could be erroneous, we use Verisig to verify the safety predictions obtained by the trained NN. As a proof of concept, to demonstrate the procedures explained in the "Verification" and "NN Retraining" sections, the results of the NN from a single initial position to all goal positions in the primitive environment are verified. However, the same NN verification and retraining procedure can be performed for all possible initial and final regions. The training data for this case are presented in Figure 9(a), while Figure 10(a) displays the results of the verification. The gray shaded regions in Figure 10(a) represent areas where the NN outputs unsafe and where Verisig concurs without performing the whole verification, since the NN output is already unsafe and thus, in the worst-case scenario, conservative. Green regions represent areas where the NN outputs safe and where Verisig verifies that the plant is safe, too. In the yellow regions, the NN outputs safe, but Verisig cannot decide whether the plant is safe or not.

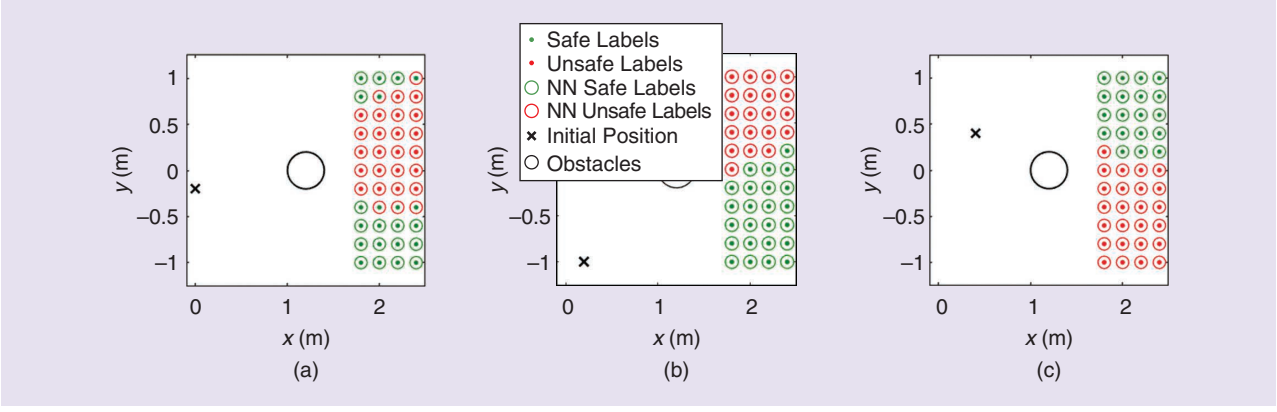


Figure 9. The safe and unsafe trained final goals and NN decisions from various initial positions. (a) Initial position [0.0, −0.2] m. (b) Initial position [0.2, −1.0] m. (c) Initial position [0.4, 0.4] m.

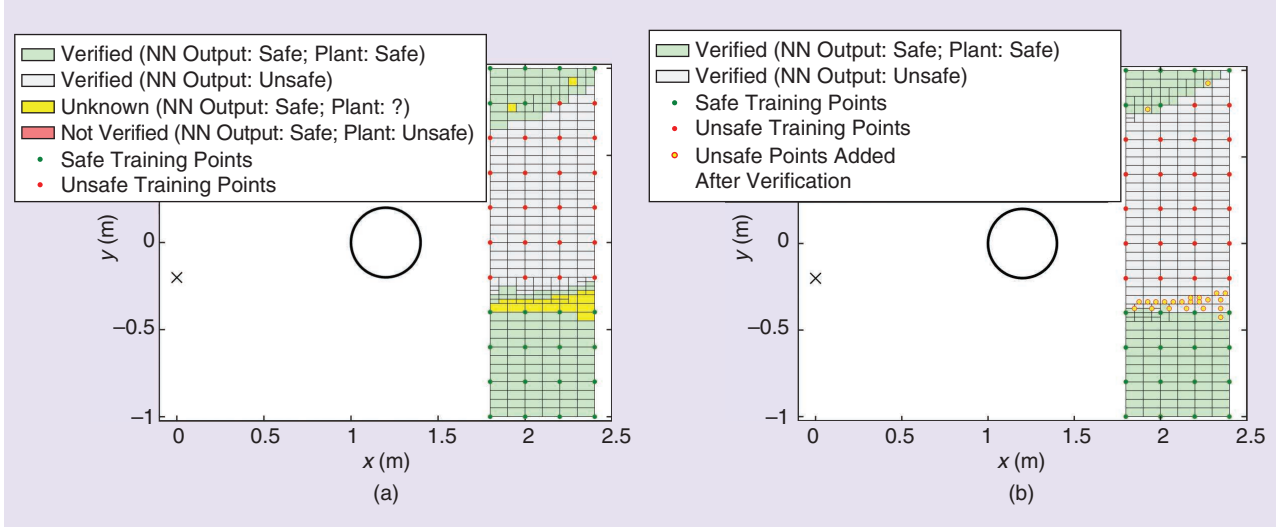


Figure 10. The NN verification results. (a) The verification of the NN trained with the original data set. (b) The verification of the retrained NN with the conservative data set.

Since the NN is not completely verified due to the undecided regions, points from these regions are added to the existing training set, and the NN is retrained as explained in the "NN Retraining" section. These points are shown by yellow dots in Figure 10(b). After retraining with the addition of these points, the NN performed with 0% FPs and roughly 1.9% FNs, which was expected since the NN was trained to be more conservative. Figure 10(b) gives the verification results with the retrained NN, and, as can be noted, the entire goal region was verified by Verisig.

Simulation Results

In this simulation, the trained NN is used to make decisions about the safety of a trajectory in the cluttered environment presented in Figure 11. First, the primitive box space used for training is superimposed around each obstacle (the square areas inside Figure 11) in such a way that there is only one obstacle in each primitive space; otherwise, the results of the NN may not be reliable due to the difference from the training conditions.

Once the mission is started, the quadrotor picks the closest point to the final goal inside the target region of the first primitive as an intermediate location. The NN makes a decision about the safety of this intermediate goal from the current position of the UAV. If the decision is deemed safe, the UAV moves to this intermediate position. If the NN decision is unsafe, it searches for a safe goal location in the target region of the current primitive. This search is performed by randomly querying points in the target area of the primitive, starting from a closer proximity of the initially selected goal and radially enlarging the search area if no safe goals are obtained immediately. Note that, to deploy such an approach, the NN must contain at least one safe point in the goal area of the primitive. This process continues until the UAV reaches its final destination.

Figure 11(a) shows the trajectory followed by the quadrotor in this environment. The queried intermediate goal positions found by the NN to be unsafe are shown by red dots in the goal regions in the primitives areas, while the safe intermediate goals traveled to by the UAV are shown by cyan dots. Wind disturbance is present throughout the entire mission, blowing in the northeast direction, as shown by the orange arrow inside the figures. The UAV is able to complete the mission without any collision. In Figure 11(b), we repeat the same case without using the NN decisions; here, the UAV moves to the intermediate goal positions even if the NN decision is unsafe. As expected, there are instances where the UAV crashes or gets very close to the obstacles. These results confirm that NNs can be used to monitor safety properties of motion planning operations using the composition of smaller verified regions into larger, more complex, and untrained environments.

Experimental Results

The same case study was also performed in experiments following a similar setup as the one presented in the previous example. NN training was done on a smaller primitive environment with one obstacle by performing 100 flights with our aerial testbed under a wind disturbance blowing in the $+y$ direction. An initial–final position pair was labeled unsafe if the reachable sets generated using the approach explained in the "Simulation-Based Reachability" section collided with

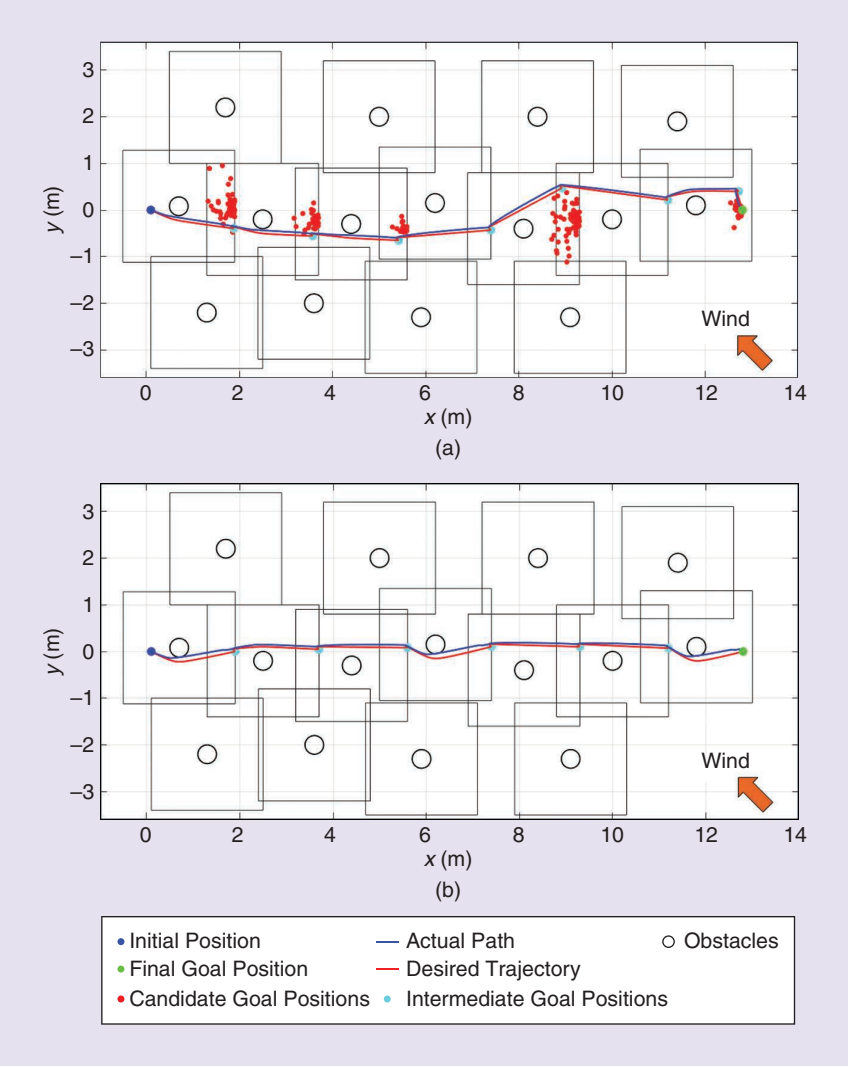


Figure 11. The navigation simulation in a cluttered environment. (a) The safe replanning using NN decisions. (b) The planning without using NN decisions.

the obstacle. Using these safe-/unsafe-labeled initial–final position pairs, a conservative NN was trained to make safety decisions about untrained initial–final position pairs at runtime. Figure 12 shows the safe and unsafe initial–final position pairs and corresponding NN decisions, using the same color coding as the previous cases.

The safe navigation approach was validated in an environment with three obstacles and two fans blowing air in the $+y$ direction, as seen in Figure 13. Obstacles in Figure 13 are represented as circles having a radius equal to the actual obstacle size plus the size of the UAV. In Figure 13(a), the intermediate goal positions queried by the NN are shown using the same color code as the simulation results. The UAV queried 20 points to avoid the first obstruction until it found a safe intermediate goal and repeated this operation until it reached its final destination. Using this NN-based framework, it takes a few milliseconds to search for a safe target. The experimental results of the proposed approach are compared with the ones in which the NN is not utilized to make safety decisions about intermediate goal positions. As expected, in Figure 13(b), the UAV fails to complete its mission safely, as it crashes and gets very close to the other obstacles. These experiments were executed without the obstructions, which are overlaid in the figure for reference.

Conclusions

The recent interest in LECs and their rapid introduction in our society, in particular in autonomous systems technologies, reveal considerable potential and benefits, especially in terms of computation overhead and decision-making applications. However, new challenges have emerged due to the lack of models and the complete reliance on data that do not provide the assurance and guarantees necessary for their deployment in safety-critical operations.

The framework for verification discussed in this article moves toward this assurance-driven design of LECs. However, many challenges remain that need to be addressed to fully integrate these technologies in safety-critical operations. In this section, we provide an overview of these challenges and offer some possible solutions and directions for future research on assured runtime monitoring.

Discussion

A first challenge typical of LECs centers around how to select the appropriate training set. The accuracy of any machine learning technique depends largely on the type, amount, and heterogeneity of the data used during the training phase. A poorly trained NN results in poor performance, leading to unsafe or overconservative behavior of autonomous systems. Similarly, overfitting can introduce overhead and poor prediction. To deal with these issues, it is possible to perform sensitivity analysis [18] and nonconformity analysis [19] on the system prior to training to better interpret and select data. It is also possible to leverage knowledge about the system dynamics to perform verification before the deployment of the LEC, as suggested in this article.

Verification provides safety guarantees for the outputs of such LECs; however, it does not provide any robustness guarantees against changes between training and testing conditions. Currently, state-of-the-art verification techniques, beyond machine learning applications [20], require knowledge about system model and bounded conditions. However, no guarantees can be provided if, for example, the disturbance acting on a system is above or below the bounds for which training was performed. This challenge becomes even more evident when dealing with real systems. In fact, typically (including in our setup in this article), verification considers a specific model that may and will change from the actual model of a real system. Even precise system-identification techniques are not able to

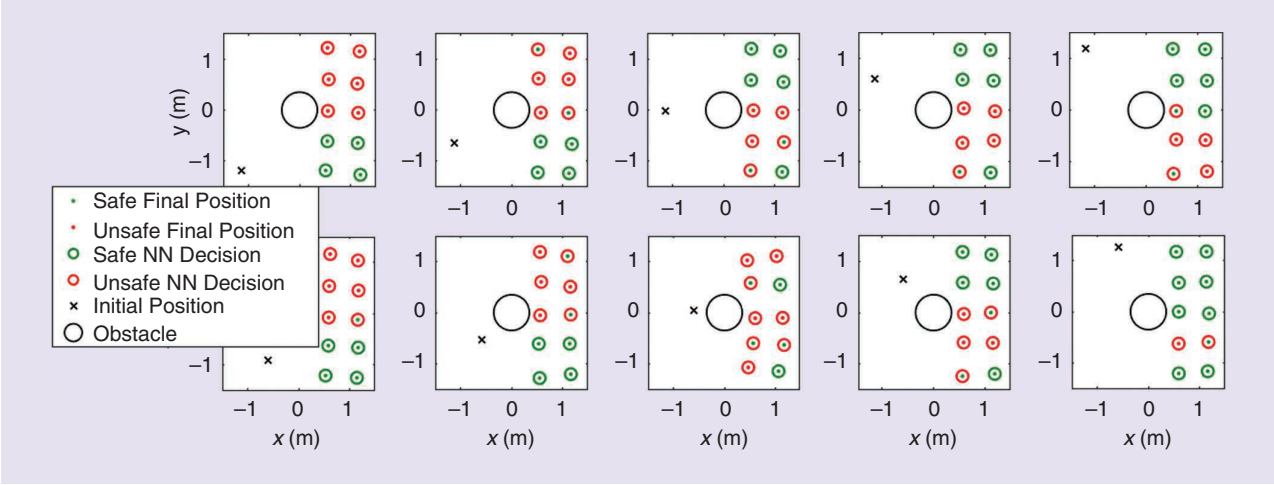


Figure 12. The safe and unsafe final positions and NN results from different initial locations in the experimental setting. The NN here is composed of four input nodes (an x-y initial position and x-y final position pair), one hidden layer of 40 nodes, and one output for the safety decision. The NN performed with 0% FPs and roughly 16% FNs.

provide fully accurate models and often rely on specific operating conditions.

One way to provide verification for systems with model uncertainties is to consider a conservative abstracted model and verify that the real system is equivalent to or safer than this abstraction. Building on the intuition in [21], if the system performance and behavior are verified to be closer to the desired behavior than the abstraction, verifying the abstraction also verifies the real system. Note that, since these conservative abstractions capture the worst-case behavior of the system, failing to verify the abstraction may lead to the belief that the actual system is not safe, which may not necessarily be true but safe.

Training and verification operations require heavy computation time; although this is a minor problem since these operations occur offline, it is still a concern, especially when dealing with high-order dynamical models and large unknowns in the system. Several services, such as Amazon Web Services (used in this article), Microsoft Azure (https://azure.microsoft.com), and Google Cloud Platform (https://cloud.google.com), are available and projected to become faster and more accessible in the future.

Lastly, we note that, similar to the approach presented in the second case study in this article, although verification is done for a static model, the approach that we presented could be generalized to other settings where verifying a subset of the state space can be sufficient for use compositionally to check safety properties about larger spaces.

Future Work

The framework proposed in this article enables fast and assured predictive and proactive monitoring of autonomous systems operations in cluttered and uncertain environments at runtime. NNs are leveraged to make decisions about the safety of planned trajectories at runtime and perform replanning accordingly. RA is used during the training phase and for verification purposes, bypassing its usage at runtime. In this way, most of the computation burden is limited to offline operations, leaving the fast decision-making and replanning tasks for the online application.

The applicability of the proposed framework was demonstrated in two case studies, and the designed NNs were verified using our recent Verisig tool to produce decisions that never lead to unsafe states. Safety was the main concern in this article. Possible future directions include adding energy constraints while designing safe operations, developing robust NNs to deal with changes in environments, and, as discussed in the previous section, diving deeper into the problem of verifying real systems.

Acknowledgment

This material is based upon work supported by the Air Force Research Laboratory (AFRL) and the Defense Advanced Research Projects Agency (DARPA) under the Assured Autonomy program, Contract FA8750-18-C-0090. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not

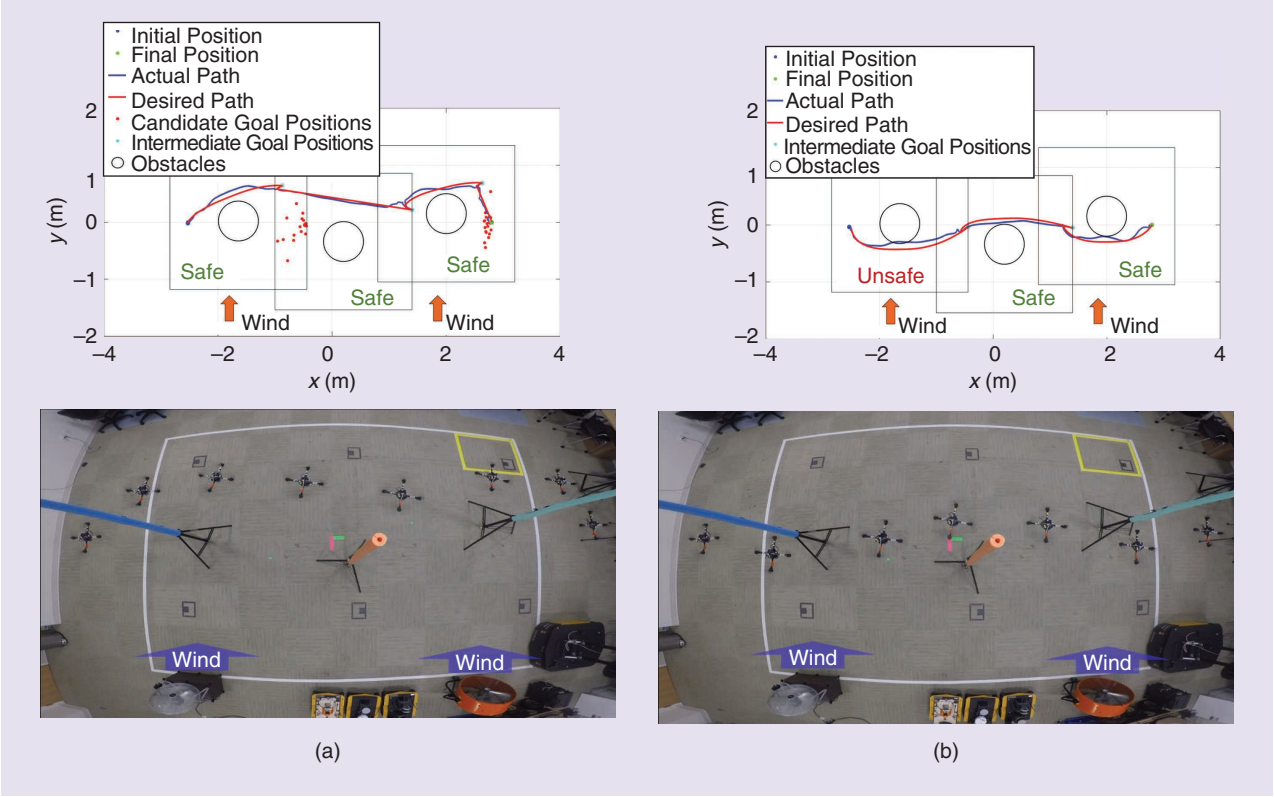


Figure 13. (a) The experimental results of the navigation of the quadrotor using the trained NN to make safety decisions and replan. (b) The unsafe navigation in which NN decisions were disregarded. The lower row shows the experiments relative to the upper row.

necessarily reflect the views of AFRL, DARPA, the Department of Defense, or the U.S. Government.

References

[1] S. Bansal, M. Chen, S. Herbert, and C. J. Tomlin, "Hamilton-Jacobi reachability: A brief overview and recent advances," in *Proc. IEEE 56th Annu. Conf. Decision and Control (CDC)*, Dec. 2017, pp. 2242–2253. doi: 10.1109/CDC.2017.8263977.

[2] X. Chen, E. Ábrahám, and S. Sankaranarayanan, "Flow*: An analyzer for non-linear hybrid systems," in *Computer Aided Verification*, N. Sharygina and H. Veith, Eds. Berlin: Springer-Verlag, 2013, pp. 258–263.

[3] S. Kong, S. Gao, W. Chen, and E. Clarke, "dReach: δ-Reachability analysis for hybrid systems," in *Tools and Algorithms for the Construction and Analysis of Systems*, C. Baier and C. Tinelli, Eds. Berlin: Springer-Verlag, 2015, pp. 200–205.

[4] R. Ivanov, J. Weimer, R. Alur, G. J. Pappas, and I. Lee, "Verisig: Verifying safety properties of hybrid systems with neural network controllers," *Proc. 22nd Int. Conf. Hybrid Systems: Computation and Control*, 2019, pp. 169–178. doi: 10.1145/3302504.3311806.

[5] J. F. Fisac, M. Chen, C. J. Tomlin, and S. S. Sastry, "Reach-avoid problems with time-varying dynamics, targets and constraints," in *Proc. ACM 18th Int. Conf. Hybrid Systems: Computation and Control*, 2015, pp. 11–20. doi: 10.1145/2728606.2728612.

[6] V. R. Royo, D. Fridovich-Keil, S. L. Herbert, and C. J. Tomlin, Classification-based approximate reachability with guarantees applied to safe trajectory tracking. 2018. [Online]. Available: http://arxiv.org/abs/1803.03237

[7] M. Chen, S. Herbert, and C. J. Tomlin, "Exact and efficient Hamilton-Jacobi guaranteed safety analysis via system decomposition," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, May 2017, pp. 87–92. doi: 10.1109/ICRA.2017.7989015.

[8] B. Djeridane and J. Lygeros, "Neural approximation of PDE solutions: An application to reachability computations," in *Proc. 45th IEEE Conf. Decision and Control*, Dec. 2006, pp. 3034–3039. doi: 10.1109/CDC.2006.377184.

[9] S. Singh, A. Majumdar, J. J. Slotine, and M. Pavone, "Robust online motion planning via contraction theory and convex optimization," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, May 2017, pp. 5883–5890. doi: 10.1109/ICRA.2017.7989693.

[10] A. Majumdar and R. Tedrake, "Funnel libraries for real-time robust feedback motion planning," *Int. J. Robot. Res.*, vol. 36, no. 8, pp. 947–982, 2017. doi: 10.1177/0278364917712421.

[11] S. L. Herbert, M. Chen, S. Han, S. Bansal, J. F. Fisac, and C. J. Tomlin, "Fastrack: A modular framework for fast and guaranteed safe motion planning," in *Proc. 2017 IEEE 56th Annu. Conf. Decision and Control (CDC)*, Dec, 2017, pp. 1517–1522.

[12] S. Kousik, S. Vaskov, F. Bu, M. Johnson-Roberson, and R. Vasudevan, Bridging the gap between safety and real-time performance in receding-horizon trajectory design for mobile robots. 2018. [Online]. Available: arXiv:1809.06746.

[13] E. Yel, T. X. Lin, and N. Bezzo, "Self-triggered adaptive planning and scheduling of UAV operations," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, Brisbane, Australia, May 21–25, 2018, pp. 7518–7524. doi: 10.1109/ICRA.2018.8463205.

[14] N. Michael, D. Mellinger, Q. Lindsey, and V. Kumar, "The GRASP multiple micro-UAV testbed," *IEEE Robot. Autom. Mag.*, vol. 17, no. 3, pp. 56–65, Sept. 2010. doi: 10.1109/MRA.2010.937855.

[15] E. van Doorn, B. Dhruva, K. R. Sreenivasan, and V. Cassella, "Statistics of wind direction and its increments," *Phys. Fluids*, vol. 12, no. 6, pp. 1529–1534, 2000. doi: 10.1063/1.870401.

[16] R. T. Palomaki, N. T. Rose, M. van den Bossche, T. J. Sherman, and S. F. De Wekker, "Wind estimation in the lower atmosphere using multirotor aircraft," *J. Atmos. Ocean. Technol.*, vol. 34, no. 5, pp. 1183–1191, 2017. doi: 10.1175/JTECH-D-16-0177.1.

[17] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *Proc. IEEE Int. Conf. Robotics and Automation*, May 2011, pp. 2520–2525. doi: 10.1109/ICRA.2011.5980409.

[18] G. Montavon, W. Samek, and K.-R. Müller, "Methods for interpreting and understanding deep neural networks," *Digit. Signal Process.*, vol. 73, pp. 1–15, Feb. 2018. doi: 10.1016/j.dsp.2017.10.011.

[19] H. Papadopoulos, V. Vovk, and A. Gammerman, "Regression conformal prediction with nearest neighbours," *J. Artif. Intell. Res.*, vol. 40, pp. 815–840, Jan. 2011.

[20] W. Xiang et al., Verification for machine learning, autonomy, and neural networks survey. 2018. [Online]. Available: arXiv:1810.01989

[21] A. Girard and G. J. Pappas, "Approximate bisimulation relations for constrained linear systems," *Automatica*, vol. 43, no. 8, pp. 1307–1317, 2007. doi: 10.1016/j.automatica.2007.01.019.

Esen Yel, Department of Engineering Systems and Environment, University of Virginia, Charlottesville. Email: esenyel@virginia.edu.

Taylor J. Carpenter, Department of Computer and Information Science, University of Pennsylvania, Philadelphia. Email: carptj@seas.upenn.edu.

Carmelo Di Franco, Departments of Engineering Systems and Environment and Electrical and Computer Engineering, University of Virginia, Charlottesville. Email: cd8gm@virginia.edu.

Radoslav Ivanov, Department of Computer and Information Science, University of Pennsylvania, Philadelphia. Email: rivanov@seas.upenn.edu.

Yiannis Kantaros, Department of Computer and Information Science, University of Pennsylvania, Philadelphia. Email: kantaros@seas.upenn.edu.

Insup Lee, Department of Computer and Information Science, University of Pennsylvania, Philadelphia. Email: lee@seas.upenn.edu.

James Weimer, Department of Computer and Information Science, University of Pennsylvania, Philadelphia. Email: weimerj@seas.upenn.edu.

Nicola Bezzo, Departments of Engineering Systems and Environment and Electrical and Computer Engineering, University of Virginia, Charlottesville. Email: nbezzo@virginia.edu.