Autonomous Emergency Landing for Multicopters using Deep Reinforcement Learning

Luca Bartolomei, Yves Kompis, Lucas Teixeira and Margarita Chli
Vision For Robotics Lab, ETH Zürich, Switzerland

Abstract— **This work presents a pipeline for autonomous emergency landing for multicopters, such as rotary wing Unmanned Aerial Vehicles (UAVs), using deep Reinforcement Learning (RL). Mechanical malfunctions, strong winds, sudden battery life drops (e.g. due to cold weather), failure in localization or GPS jamming are not uncommon and all constitute emergency situations that require a UAV to abort its mission early and land as quickly as possible in the immediate vicinity. To this end, it is crucial for a UAV that is deployed in real missions to be able to detect a safe landing spot efficiently and proceed to land autonomously, avoiding damage to both its integrity and the surroundings. Driven by the advances in semantic segmentation and depth completion using machine learning, the proposed architecture uses deep RL to infer actions from semantic and depth information, flying the robot towards secure areas, while respecting safety constraints. Thanks to our robust training strategy and the choice of these mid-level representations as input to the RL agent, we show that our policy can directly transfer to the real world, without the need for any additional fine-tuning. In a series of challenging experiments both in simulation and with a real platform, we demonstrate that our planner guides a rotorcraft UAV to a safe landing spot up to 1.5 times faster and with double success rate than the state of the art (including a commercially available solution), paving the way towards realistically deployable UAVs.**

I. Introduction

Multicopters are highly agile and versatile Unmanned Aerial Vehicles (UAVs), which can be utilized in a variety of applications, such as search-and-rescue missions, inspection and 3D reconstruction, surveying and goods delivery. However, several dangerous events can occur during deployment. Adverse atmospheric conditions, such as cold weather or strong wind gusts, can unexpectedly shorten the battery life, whereas mechanical malfunctions and localization failures can potentially lead to crashes. Commercial drones, since they are designed to fly over populated areas, remain capable of landing in such critical situations, even with reduced battery voltage or partially working rotors. It is, therefore, fundamental to enable UAVs to detect safe landing spots nearby and autonomously land there in case of an emergency. The problem of detecting a safe landing area assumes greater importance in the case of navigation in urban settings, as damage needs to be avoided not only to the body of the drone and the landscape, but also to moving cars, people, etc. Moreover, given the limited payload capabilities of multi-rotor drones, such UAVs are generally equipped with a minimal sensor set-up composed of cameras and range sensors, rendering the emergency landing problem even more challenging, as a suitable landing location has to be identified by relying on limited sensory information. Recent solutions proposed in the literature either require manual intervention at the landing area by placing fiducial markers to identify a safe spot on the ground [1], [2], or require dense online 3D mapping of the environment [3], [4]. While the former methods are not suitable for emergency landing in unstructured environments, the latter ones, despite being more flexible, require high-quality sensory inputs. This assumption limits the applicability of these approaches, as their sensing pipeline cannot cope with the high level of noise during outdoor real-world flights, leading to failures. Moreover, these methods reason only about the geometry of the navigation area dismissing more general and relevant context, for example, the fact that some areas can be identified as clear spaces, but are unsuitable for landing, such as rooftops and roads. In order to overcome this issue, pipelines use high-level semantic information, which has been receiving growing attention in recent years [5], as semantic segmentation has been proven to be extremely useful for path-planning applications [6]. However, these approaches have been deployed in simulation only, raising questions about their applicability in real-world missions.

This work was supported by the Swiss National Science Foundation (SNSF, Agreement no. PP00P2183720), NCCR Robotics, the Amazon Research Awards and the HILTI group.
Video – https://youtu.be/p8tpLL7Q0GE
Code – https://github.com/VIS4ROB-lab/multirotors_landing

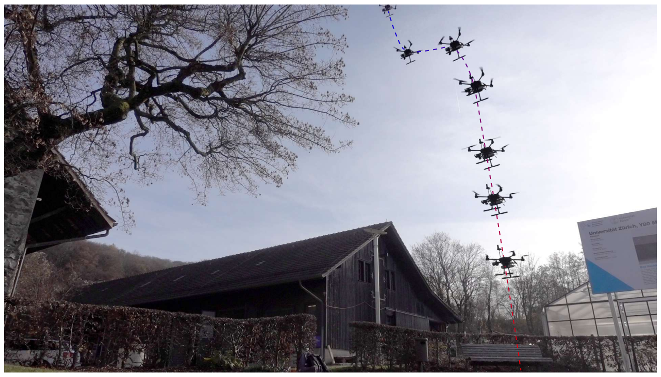


Fig. 1: Safe emergency landing performed during a real-world experiment. Using semantic and depth information, the multi-rotor UAV finds a safe landing area and autonomously descends towards it, while avoiding collisions with the environment.

Motivated by these challenges, we propose a deep Reinforcement Learning-based (RL) emergency landing pipeline that detects safe landing areas from both depth and semantic information, by relying on a minimal sensor set-up composed of a monocular RGB camera. Thanks to recent findings in depth estimation and semantic segmentation using deep learning, a single camera snapshot is sufficient to recover the necessary information to land a UAV safely. The proposed RL agent, taking as input depth and semantic images, outputs high-level commands to find a suitable landing spot and navigates the UAV towards it as fast as possible, while avoiding collisions (Fig. 1). We demonstrate that we can train the proposed policy only in simulation and directly deploy it in real-world experiments, where other state-of-the-art planners fail. This work represents a fundamental step towards more secure and reliable autonomous drones, able to detect safe areas and land in case of an emergency respecting safety constraints.

In brief, the contributions of this work are the following:

- the design of a semantic-aware policy architecture that can be trained in simulation only and then deployed on a real UAV,
- an extensive evaluation of the proposed system in both simulation and real-world experiments, and
- the source code of the proposed system.

II. Related Works

Despite the increasing importance given to safety, emergency landing of drones still remains an open problem, especially in unknown environments [7], [8]. Detecting a safe landing area and approaching it are both challenging tasks, since factors such as obstacles, uncertainties in state estimation, and the platform's dynamics must be accounted for. From a control theory perspective, this problem is cast as an optimization, where the optimal motor commands are found with respect to a final objective, e.g. minimizing fuel usage [9]. A plethora of approaches have been proposed, ranging from optimization-based nonlinear control strategies [10] to more complex solutions, built upon optimal policy search using deep reinforcement learning [9]. These solutions are demonstrated to be extremely effective in controlling a range of different platforms, such as quadrotors [10] and spacecrafts [9], even under noisy state estimation by using learning-based strategies to model the robot's dynamics [11]. However, as the focus is solely on the control problem, they assume that a landing target is given and do not deal with obstacle avoidance. In the literature, as well as in commercially available solutions such as Google Wing and Amazon Prime Air, a common approach is to manually mark suitable landing spots using highly-distinctive fiducial markers. These are also employed to enable small quadcopters to land on top of other moving platforms, such as ground robots [2], or maritime vehicles [12]. In these cases, the presence of the fiducial marker is fundamental, as its simple tracking facilitates the task greatly. In this direction, the recent work by Polvara *et al.* [1] proposes a reinforcement learning-based system, using domain randomization to enable a quadrotor to land on top of a fiducial marker under perceptual aliasing caused by different background textures, including floor tiles, grass, terrain and concrete. While they demonstrate it is possible to reach a human-like level of performance in real-world experiments by relying on RGB images only, their approach cannot be utilized for emergency landing in unknown environments, since it relies on the presence of a marker and does not deal with obstacles and occlusions. To overcome this limitation, more sophisticated vision-based pipelines that do not rely on fiducial markers have been proposed, using either geometrical [3], [4], [8] or deep learning approaches [7], [13]. These solutions are more flexible and allow robots to land safely in unstructured and unknown environments [8]. To identify a suitable spot without maps known *a priori*, they employ RGB, depth and semantic images, and output the desired landing location [7], [14]. They are able to actively avoid collisions against obstacles by first generating a map of the area online, and then planning in it [8]. The work by Foster *et al.* [4] explicitly creates an elevation map using depth completion to identify a suitable landing spot, and then generates a path to that position in a separate path-planning module. Similarly, Mittal *et al.* [3] identify a safe landing area by processing dense depth images from a stereo camera and extracting terrain information, such as steepness and flatness, while also considering depth accuracy and the estimated energy consumption to reach a candidate location. Once a final position is identified, a safe path to the landing area is computed. While these approaches are validated in real-world experiments, they assume that the dense depth input, either from depth completion or stereo matching, is stable and accurate. However, this hypothesis might not be valid when flying at higher altitudes, as depth uncertainty increases with the squared value of depth [15], and stereo matching can provide accurate depth information only in a short range. Another line of research utilizes Neural Networks (NNs) for binary classification of the terrain into either safe or hazardous areas [7], while performing landing site detection and tracking of people [5]. The training is performed on either satellite images or synthetic datasets [16], with very few works tested in real-world experiments [14]. Furthermore, this category of approaches performs binary classification of the landing area, not exploiting the complete semantic knowledge extracted from sensor readings. Using the full semantic mask can be beneficial, as it could allow higher-level reasoning on the structure of the environment to identify safer landing areas faster.

Inspired by the shortcomings of these approaches and the need for emergency landing strategies for realistically deployable UAVs, in this work we propose a deep RL approach that employs semantically labeled images and depth maps to perform emergency landing of UAVs in urban scenarios. This is performed by leveraging the most recent results in deep learning for depth completion [17] and semantic segmentation [18]. Using these mid-level representations as inputs can alleviate the simulation-to-reality gap, allowing for faster and more stable training of deep RL agents [6], [19]. We demonstrate that our policy, trained exclusively in simulation, can be directly deployed onboard a UAV in real-world missions. To run onboard drones with limited computing capabilities, the proposed pipeline is designed in a modular fashion, that allows to outsource depth estimation and semantic segmentation to cloud-based resources, if needed. We compare our approach against both a state-of-the-art landing pipeline [3] and a commercially available

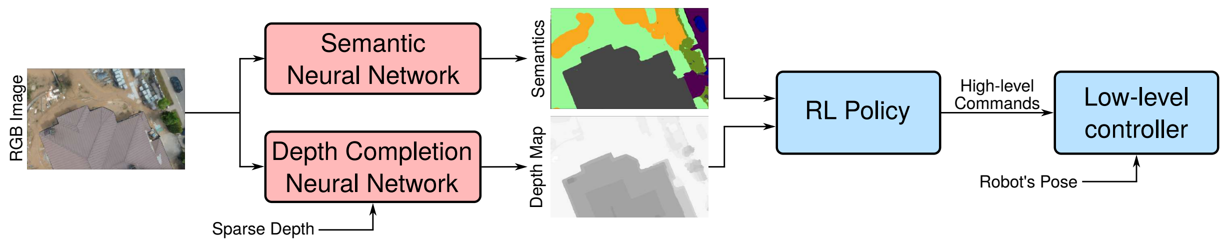


Fig. 2: Schematic overview of the pipeline. The input to the system is a monocular RGB image, which is used by a semantic segmentation NN [18] to generate a semantic mask of the environment. A depth completion network [17] outputs a dense depth map of the surroundings using the RGB image and sparse depth information. The semantic mask and the depth map are used by the RL agent to generate high-level commands, that are successively fed to the robot's low-level controller.

solution, demonstrating that our approach reaches higher success rates, as well as faster landings.

III. Methodology

Our main objective is to identify a safe landing area and perform the descending maneuvers safely and as fast as possible to reduce the flight time. We formulate this as a deep reinforcement learning problem, where an RL agent is trained to identify suitable areas by means of depth maps and semantically labeled images. While depth maps carry information about the shape of the environment (e.g. obstacles, free space), semantics expose the agent to the spatial relationships between different classes (e.g. cars and roads; houses and grass) [6], speeding up the search for valid landing spots. Moreover, using these mid-level representations as input to a deep RL agent is proven to yield better generalization of the policy [19]. Learning directly from raw camera data to command actions requires an implicit semantic segmentation and depth estimation step, which would take an extremely long training time in an RL fashion. Our architecture allows to decouple the problems of depth estimation and semantic labeling from path planning and control. This decoupling is essential for portability and deployability, as by simply providing access to cloud-based computing resources, we can enable any flying platform to employ our learnt policy. Given the recent findings in semantic segmentation [18] and depth completion [17] using deep learning, this work assumes that semantically labeled images and depth maps are available. These inputs are fed to the agent, which outputs high-level commands that are then processed by a low-level controller. The RL agent is trained in photorealistic simulations only, initially by using ground-truth semantics and depth, and then fine-tuned by replacing these high-quality inputs with predictions from NNs. This procedure fills the simulation-to-reality gap and leads to policy generalization, allowing us to test the agent in real-world experiments without additional fine tuning.

A. System Overview

An overview of the proposed pipeline is shown in Fig. 2, where we assume a stream of RGB images as input. Thanks to the level of maturity of state-of-the-art semantic segmentation and depth completion using deep learning, a monocular camera snapshot and sparse depth seeds are sufficient to extrapolate the semantic masks and the depth maps. The RL agent is then tasked to find a suitable landing spot and descend towards it by generating high-level commands that are sent to the robot's low-level controller. While depth and stereo cameras can provide information in the range of a few meters, depth completion methods can estimate depth with a longer range, albeit at the expense of higher noise levels. Nevertheless, long range information is beneficial in drone landing, enabling a platform to be proactive with respect to collisions. The modular design of the proposed pipeline decouples the RL problem from semantic segmentation and depth estimation, allowing our system to run onboard a small UAV.

B. Deep RL Policy

The RL agent maps semantically labeled images and depth information to actions employing an Actor-Critic model with the architecture shown in Fig. 3. Here an action consists of a high-level command (lateral movement, halt motion, descending motion) that can be selected from a set of 10 possible actions. In the following, we describe the policy architecture and the reward function used to train the agent in more detail.

1) Policy Architecture: The Actor and the Critic networks share the first part, composed of a 3-layer Convolutional Neural Network (CNN). The final part of the Critic consists of two Fully Connected (FC) layers composed of 32 units, while the action is output by the Actor from three FC layers with 64 units each. Policy optimization is then performed at fixed-step intervals employing the on-policy algorithm PPO [20]. Moreover, in order to reduce the hyperspace dimension, we convert the color semantic mask into a single channel and downsample both the resulting image and the depth map to 128×128 pixels. Semantics and depth are mid-level visual representations that are more generic than raw color images, and they are demonstrated to allow faster training and improved policy performances [19].

2) Reward Function: The training of the policy is performed based on the data and the rewards collected in each episode. At each timestep, the total reward is composed of different terms:

$$R_{TOT} := \omega_t R_t + \omega_S R_S + \omega_C R_C + \omega_A R_A + R_T, \quad (1)$$

where R_t is the step reward, R_S is associated to the semantic classes in view, R_C to collisions between the robot and the environment, R_A to the selected action, and R_T is the terminal reward.

The step reward is assigned at constant intervals until the

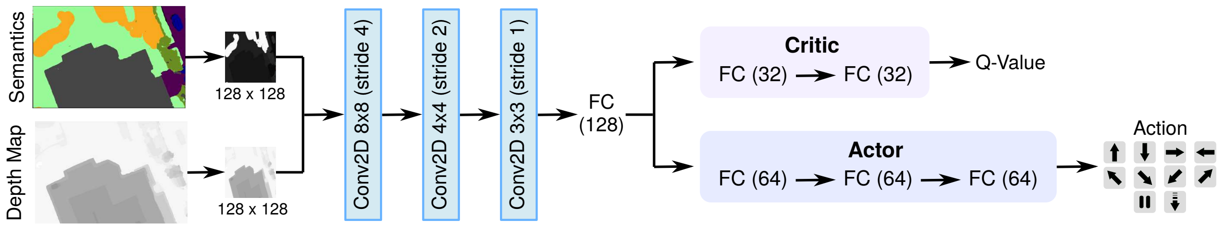


Fig. 3: Schematic overview of the policy architecture. First, the semantic image, converted into a single channel, and the depth map are downsampled. The resulting images go through the Actor-Critic network, composed of three CNN layers and a linear Fully Connected (FC) layer. The Critic then outputs the Q-Value from two FC layers, while the Actor feeds the high-level command to the controller. Policy optimization is performed with the PPO algorithm [20].

episode ends:

$$R_t := \begin{cases} 0 & \text{on episode termination} \\ -0.2 & \text{otherwise} \end{cases} . \quad (2)$$

The negative contribution to the total reward encourages the policy to land as fast as possible.

The reward associated to semantics, R_S, incites the agent to fly on top of the classes that are considered safe for landing, such as terrain, while penalizing flights on dangerous areas, including roads and houses. Given the semantic mask I_S and the set of possible semantic classes $\mathcal{S}$, this reward is calculated as

$$R_S := \sum_{s \in \mathcal{S}} \gamma_s \frac{count(I_S^s)}{count(I_S)} , \quad (3)$$

where $count(I_S^s)$ the number of pixels in I_S associated to class s, $\gamma_s \in \{-1, 1\}$ is the weight associated to s, and $count(I_S)$ the total number of pixels in I_S. A negative weight γ_s represents a penalty associated to flying over class $s \in \mathcal{S}$ and leads to more exploratory actions, while a positive value encourages the agent to start landing on top of s.

While the objective of this reward is to identify a suitable landing spot, the collision reward R_C deals with the problem of obstacle avoidance. In this case, we model the UAV as a sphere with radius r and this reward pushes the agent away from obstacles:

$$R_C := \begin{cases} 0 & \text{if } d_{min} \geq d^* \\ -\frac{1}{d_{min}-r} & \text{if } r < d_{min} < d^* \end{cases} , \quad (4)$$

where d_{min} is the distance to the closest obstacle and d^* is a threshold to check if the UAV is too close to an object, with $d^* > r$. When the robot is closer than d^* to an obstacle, the policy is rewarded negatively and, in case of a collision ($d_{min} \leq r$), the episode ends and the agent receives a negative terminal reward R_T.

The reward R_A is associated to the action selected by the policy. As our pipeline is built for emergency landing and our objective is to reach the ground quickly, we reward the descending movements, while slightly penalizing lateral moves and halting the motion:

$$R_A := \begin{cases} -0.05 & \text{if lateral movement} \\ -0.5 & \text{if halt motion} \\ 1 & \text{if descending movement} \end{cases} . \quad (5)$$

At the end of each episode, we assign the terminal reward R_T. The episode terminates when the robot either lands safely, collides against an obstacle or when the maximum allowed time to land is reached. In case of a successful

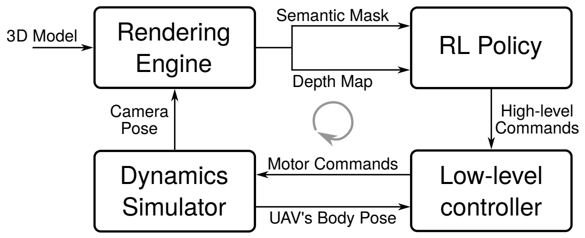


Fig. 4: Schematic overview of the simulation used for training the deep RL policy. Given a 3D model and a camera pose, a Vulkan-based rendering engine generates semantic and depth images that are fed to the agent. The policy outputs high-level commands that are transformed into motor commands by a low-level controller, and the UAV position is updated according to the dynamic model for quadrotors introduced in [21]. At test time, the ground-truth semantic and depth images are substituted by the output of the semantic segmentation and depth-completion NNs from [18] and [17], respectively.

run, for example when landing on a safe area, such as flat terrain or grass, we assign a positive terminal reward, while in case of collisions, exhausted time budget or landing on a hazardous area, the policy receives a penalty as follows:

$$R_T := \begin{cases} 50 & \text{if landing on safe areas} \\ -20 & \text{if landing on dangerous areas} \\ -20 & \text{if in collision} \\ -50 & \text{if maximum time reached} \end{cases} . \quad (6)$$

C. Training Environment

Fig. 4 shows the pipeline used to train the deep RL policy. We assume that the robot is equipped with a downward-looking camera, and we render photorealistic RGB images, semantic masks and depth maps using a custom pipeline based on the Vulkan library[1]. The Vulkan API allows to create a fast and lightweight rendering scheme that, given a camera pose and a 3D model with associated textures, generates semantics and depth information that is then fed to the RL policy. The communication between the renderer and the policy is established with TCP sockets, in order to mimic the pipeline used in real-world experiments, where semantic segmentation and depth completion are outsourced to web computing resources. The high-level actions from the policy are then fed to the low-level controller that, given the current robot's pose, generates motor commands. By adopting the dynamic UAV model proposed in the Flightmare framework [21] and assuming a fixed integration step, the quadrotor flies to the target location. This cycle repeats until the end of the episode.

Notice that, in order to have a stable training process, ground-truth semantics and depth images are fed to the agent, since the outputs from the semantic segmentation

[1]`https://www.vulkan.org`

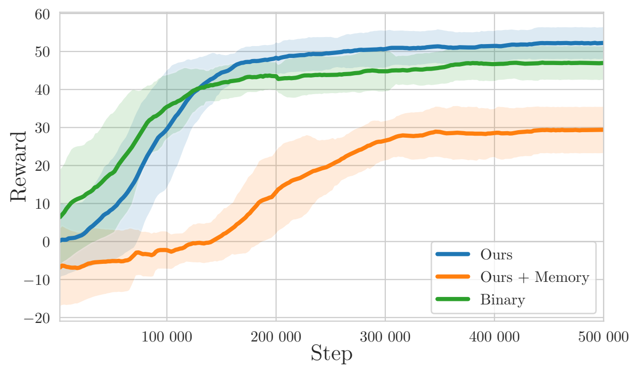


Fig. 5: Reward curve over the training steps. The proposed policy is depicted in blue. In the initial phase of training, the sharp increase in the reward demonstrates that the RL agent quickly learns the actions to land on safe areas. In green, we show the training performance of the policy using binary images as input, and in orange our policy with an additional memory module after the fully connected layer.

and depth-completion NNs are subjected to a high level of noise. The training performance of a policy when exposed to stochastic inputs can degrade drastically, and it is a well-known problem in RL. Only after the initial training phase, we fine-tune our policy by exposing the agent to the complete pipeline. In Sections IV-A and IV-B, the pipeline used for fine-tuning and testing is described in more detail.

IV. Experiments

A. Policy Training

We collect a series of static, photorealistic 3D models generated from photogrammetry[2], where the policy is trained (Fig. 6). The UAV is equipped with a downward-looking camera, and at each episode the robot is placed at a random initial position and it is tasked to land safely on the ground. All the scenes are characterized by the same semantic classes (pavement/road, terrain, water, trees, buildings and cars). We use common classes used by several state-of-the-art semantic-segmentation algorithms, but our system can handle any set of classes. Each semantic class is either considered safe or unsafe to land on. During training, only terrain is deemed safe, as landing on any other class might potentially be dangerous. To have a faster and more stable training, the process is parallelized by utilizing four drones at a time. Each agent outputs actions at fixed time intervals (or *steps*), communicates them to the robot's controller and collects the reward as feedback. The 3D model of the environment changes at constant time intervals, allowing the policy to experience different scenes and avoiding overfitting. In the first episode of the training process, the policy is initialized randomly. The training continues until the maximum number of steps across all episodes is reached. To compute the total reward as in (1), the weights are set to $\omega_t = 1$, $\omega_S = 1$, $\omega_C = 0.5$ and $\omega_A = 1$, while d^* and r are set to $3\,\mathrm{m}$ and $0.5\,\mathrm{m}$, respectively. Fig. 5 reports the training performance of the agent over $500\,000$ steps (blue curve), with a linearly decaying learning rate. The policy is initially trained with ground-truth inputs and shows a converging reward curve. The agent is successively fine-tuned by exposing it to noisy inputs coming from semantic segmentation [18] and depth-completion [17] NNs, where the sparse depth seeds are obtained by sampling ground-truth depth images. This is a necessary step to deal with the simulation-to-reality gap, which is of extreme importance in robotic applications. While mid-level representations can help to mitigate this problem, most of the works in the literature feed only ground-truth semantic masks and depth maps corrupted by mild noise. In reality, the outputs from deep learning-based methods can introduce severe noise and inconsistencies in the pipeline, especially when NNs are exposed to inputs that differ from the training data. These inconsistencies can be potentially harmful for the policy, as RL is notoriously sensitive to noise on the inputs and on the reward. By fine-tuning the policy for $100\,000$ additional steps with a fixed learning rate of 10^{-4}, the policy is adjusted to improve its performance when exposed to noisy data.

B. Tests in Previously Unseen Environments

A series of experiments in challenging scenes are conducted to show the effectiveness of the proposed method. We select 7 static models of real-life places not experienced at training time, all created from photogrammetry (Fig. 7). The test models are chosen to create hard challenges for the pipeline (e.g. number of obstacles, small valid landing areas) and present the same semantic classes experienced at training time.

1) Test Set-Up: We select 15 initial positions in the scenes and we report the number of successful landings, as well as the time to land. We consider a run to be successful if the drone lands without collisions on terrain and if the landing does not exceed a maximum allowed time, set to $30\,\mathrm{s}$. Timings provide insights about the efficiency of the planning solution, as in emergency situations the objective is to reach the ground as fast as possible, avoiding large detours to find a suitable landing spot. The starting height of the UAV is set between $20\,\mathrm{m}$ and $30\,\mathrm{m}$. At test time, we fix the weights of the NN modelling the policy and we replace the ground-truth inputs with the outputs from the semantic network [18] and the depth-completion method [17]. The depth-completion network receives sparse depth seeds by sampling ground-truth depth maps available from the simulator.

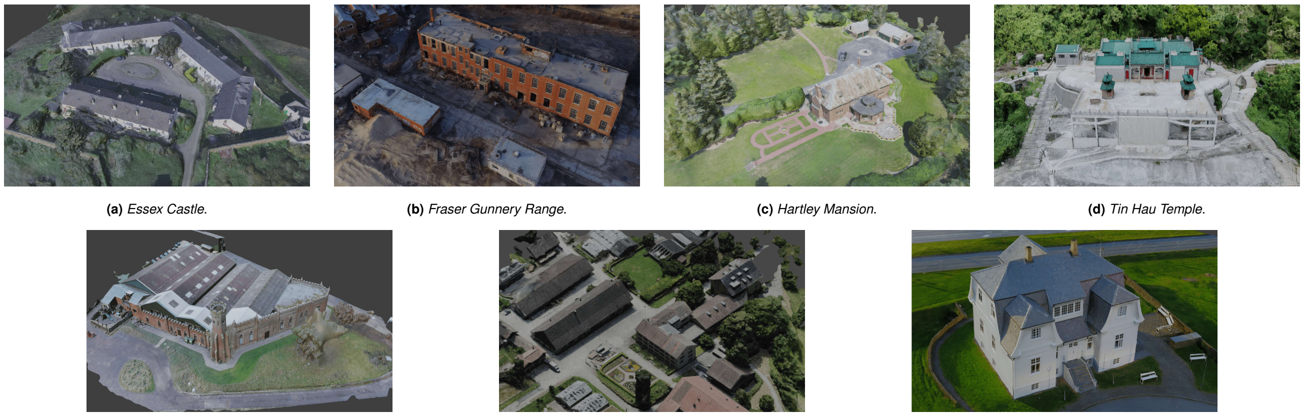
We compare our approach against a naive baseline strategy, the planner proposed by Mittal *et al.* [3] and the commercially available solution by PX4[3]. While the naive solution lands the UAV by guiding it straight towards the ground, [3] detects a safe landing spot processing the depth images to find a flat area with the best characteristics in terms of inclination, estimated depth uncertainty and expected energy consumption to reach that position. Then, RRT* is used to find a suitable path from the current UAV location to the target checking for collisions in a 3D map.

Similarly, the commercial solution by PX4 detects suitable landing spots using depth information. To decide whether a location is a good candidate landing spot, normal vectors in

[2]`https://sketchfab.com`

[3]`https://github.com/PX4/PX4-Avoidance`

Fig. 6: Views of some of the 3D models experienced by the RL agent at training time. The models, extracted from urban scenes only, are characterized by the same semantic classes (buildings, roads and pavements, terrain, cars and vegetation).



(a) *Essex Castle.* **(b)** *Fraser Gunnery Range.* **(c)** *Hartley Mansion.* **(d)** *Tin Hau Temple.* **(e)** *Warehouse.* **(f)** *Irchel.* **(g)** *Hofdi House.*

Fig. 7: Views of the 3D models where the deep RL policy is tested. These models are not experienced by the agent at training time.

the area are estimated and the terrain gets classified as either safe or dangerous. This planner then triggers the UAV's descent until an obstacle is sensed in its path. If the distance to the obstacle is smaller than a threshold, the UAV flies in an exploratory pattern until a suitable landing spot is found, and then descends towards it. This cycle repeats until the ground is reached.

Both path planners by [3] and PX4 assume accurate depth measurements from an onboard stereo camera with a small, fixed baseline. However, the decreasing stereo disparity with increasing flight altitude renders this assumption unrealistic in practice. Moreover, following the trend in Robotic Vision literature, as in this work, replacing stereo depth estimation with deep learning-based depth completion as a workaround, can also result in highly noisy depth estimates, limiting the applicability of these planners. Note that, while neither of these planners consider semantic information in their original forms, for fairness of comparison with our pipeline, we extend them to use semantic masks alongside depth maps, performing an additional binary classification of the landing spots on top of their original pipelines, by considering the class 'terrain' as the only safe area.

2) Results and Discussion: The success rates of the test runs, as well as the average time to land, are reported in Table I. The latter is computed by considering only the successful runs. Notice that in the first column we show the performance of the 'Baseline', which refers to a naive planner guiding the UAV to land immediately descending straight to the ground. This is the fastest approach, but exhibits the lowest success rates as it does not consider the UAV's surroundings. The proposed policy exhibits the highest success rates and the best timings across almost all cases, with the agent respecting both the safety and the timing constraints, while landing the UAV on valid

Experiment	Baseline	Ours	Mittal *et al.* [3]	PX4
Essex Castle				
Success Rate [%]	13.3	**73.3**	53.3	53.3
Average Time [s]	7.9 ± 2.1	**9.7 ± 2.3**	12.2 ± 3.9	13.8 ± 4.4
Fraser Gunnery Range				
Success Rate [%]	6.7	**86.7**	40.0	20.0
Average Time [s]	10.8 ± 0.0	**16.4 ± 5.9**	17.0 ± 4.1	19.5 ± 2.5
Hartley Mansion				
Success Rate [%]	6.7	**80.0**	60.0	53.3
Average Time [s]	11.5 ± 0.0	**10.5 ± 2.1**	15.9 ± 2.7	18.5 ± 1.9
Tin Hau Temple				
Success Rate [%]	6.7	**86.7**	26.7	13.3
Average Time [s]	10.0 ± 0.0	**10.6 ± 0.7**	17.8 ± 7.1	15.9 ± 0.9
Warehouse				
Success Rate [%]	20.0	**93.3**	26.7	86.7
Average Time [s]	11.2 ± 1.0	14.3 ± 3.1	**14.2 ± 2.4**	19.9 ± 1.8
Irchel				
Success Rate [%]	0.0	**93.3**	73.3	40.0
Average Time [s]	N.A.	**12.9 ± 1.3**	18.2 ± 3.9	22.4 ± 2.5
Hofdi House				
Success Rate [%]	0.0	**100.0**	60.0	80.0
Average Time [s]	N.A.	**8.1 ± 2.3**	13.8 ± 4.2	17.1 ± 5.3

TABLE I: Results of the experiments in the 7 simulated scenes. We report the success rate over 15 runs, as well as the average time to land. The averages of the timings are computed considering only the successful landings. The best performance in each scene is shown in bold, without considering the results of the baseline planner.

locations and avoiding collisions. Thanks to the use of full semantic masks, the RL agent learns to leverage the spatial relationships between the semantic classes during training, and thus the policy identifies safe landing spots quickly and effectively, avoiding extensive exploration of the surroundings. On the contrary, the effectiveness of the other planners to land the UAV safely and without crashes seems to be limited due to the noisy input, often guiding the UAV to fly over large detours. In particular, terrain information, such

Experiment	Ours	Ours + Memory	Binary
Essex Castle	**73.3**	26.7	60.0
Fraser Gunnery Range	**86.7**	6.7	33.3
Hartley Mansion	**80.0**	0.0	46.7
Tin Hau Temple	**86.7**	6.7	46.7
Warehouse	**93.3**	13.3	80.0
Irchel	**93.3**	0.0	66.7
Hofdi House	**100.0**	0.0	66.7

TABLE II: Success rates (in %) on the test models for the ablation study comparing our architecture with a more complex policy with a memory module (*Ours + Memory*) and against a policy using depth maps and binary semantic masks as inputs (*Binary*). The best results are shown in bold.

as its flatness or inclination that are extracted from the depth maps generated using depth completion are noisy, rendering the identification of valid landing spots in these methods difficult. In fact, PX4 collides against obstacles, while [3] generates long, intertwined paths.

C. Ablation Study

To further analyze our pipeline, we run ablation studies on the policy architecture and type of input to the agent. We compare our approach against two variants; the *Ours + Memory* policy with an additional Long Short-Term Memory (LSTM) component with 256 cells before the Actor-Critic module (similarly to [6]), and a *Binary* policy with the same architecture as in Fig. 3, but trained with depth images and binary semantic masks, where the only valid landing area is 'terrain'. The training performances of all variants are shown in Fig. 5, and the success rates in the test models are reported in Table II. Extending the policy with the memory module exhibits the worst performances at both training and testing time, with no successful runs in 3 out of 7 test models. While LSTMs could be beneficial in dynamic scenes [6], they require more time to train, leading to lower rewards. Instead, the *Binary* policy converges faster, but to a lower average reward than our policy. This is reflected in the performances during testing, since our method achieves the highest success rates, demonstrating the benefit of using the full semantic masks over binary inputs.

D. Real-world Experiments

We test the proposed pipeline in a series of real-world experiments, where semantic segmentation and the depth-completion NN are outsourced to cloud-based computing resources (Amazon AWS). However, notice that the modular design of the pipeline allows to replace the cloud with a GPU carried onboard the UAV. We directly deploy the policy trained exclusively in simulation without any additional fine-tuning, demonstrating that the use of depth and semantic images as input to the RL agent can fill the simulation-to-reality gap.

1) System Set-up: The UAV used in the experiments is a *Holybro X500*, equipped with an Intel NUC with an Intel Core i5-1145G7 CPU and a 4G modem stick to communicate with the cloud resources. We use the *Pixhawk* autopilot[4] as a low-level controller, while state estimation is performed by fusing RTK GPS estimates with inertial measurements. RGB images are captured with a down-looking global-shutter camera and are successively sent to the cloud via TCP connections to extract depth and semantic information. Alongside color images, the depth-completion NN receives sparse depth seeds in the form of 3D landmarks estimated by ORB-SLAM3 [22] running onboard the drone. Once semantic and depth information is received from the cloud, it gets fed into the RL agent, which communicates high-level commands to the UAV's controller. The experiments are carried out in an urban area. The UAV starts at an altitude of $25\,\mathrm{m}$ off the ground over a street, and it is tasked to land on grass, avoiding collisions with the nearby buildings and vegetation. We consider a run to be successful if the drone lands without interventions from the safety pilot. In Table III we report the latency from sending RGB images to the cloud and receiving depth and semantic information back, as well as the bandwidth consumption for communication. Once the policy receives these inputs, it takes an additional $0.40 \pm 0.07\,\mathrm{ms}$ to output the high-level control commands.

[4]https://pixhawk.org

Image Type	Latency [s]	Bandwidth [MB/s]
Depth	3.03 ± 1.21	3.18 ± 0.13
Semantics	1.34 ± 0.47	0.27 ± 0.03

TABLE III: Latency and bandwidth consumption statistics for the communication between the cloud and the UAV's onboard computer in a real-world experiment.

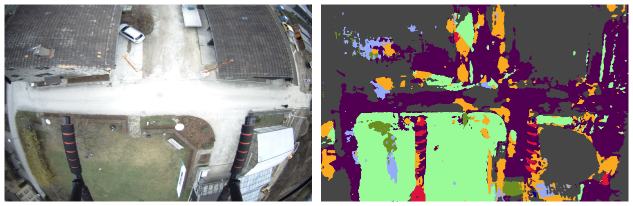
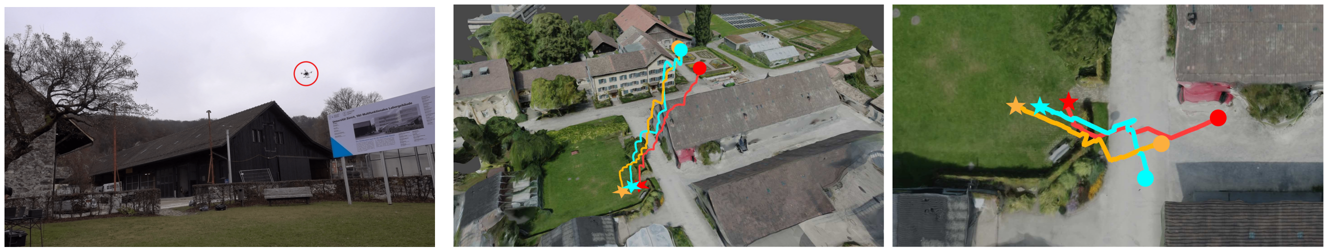

(a) Visual Image. **(b)** Semantic Mask.

Fig. 8: One example image and its corresponding semantic mask during a real-world experiment captured by the downward-looking camera. The light green areas correspond to grass and terrain, and are the only valid landing areas.

2) Results and Discussion: Our policy, even if it was trained only in simulation, is the only approach able to land the UAV safely. In contrast, the planners by [3] and PX4 cannot cope with the noisy inputs from real-world data (Fig. 8). In particular, due to high noise in the depth, the former fails to find a suitable landing position, guiding the UAV to hover over the initial position. Even when a suitable landing spot is found, the path generated is extremely convoluted because of the presence of artifacts in the 3D map used for obstacle avoidance. Similarly, PX4 starts the landing procedure, but, as soon as an obstacle is sensed, it flies the UAV in dangerous detours that force the safety pilot to take over to avoid crashes into the nearby buildings. As in simulation, also in the real experiments, the power of semantics to identify safe landing spots as advocated by the pipeline, is emphasized. Moreover, our policy does not build a full 3D reconstruction of the environment, but instead reacts to potential collisions in a receding horizon fashion, allowing the robot to safely land on grass (Fig. 9).

During testing, we experience some failure cases also with our policy. In one run, semantic misclassification causes the



(a) External Camera Frame. (b) 3D View. (c) Top View.

Fig. 9: View of the environment and examples of trajectories flown by the UAV during successful real-world experiments when guided by our policy. In (a) the drone is highlighted with a red circle. The paths are shown in the 3D reconstruction of the real place where experiments are carried out. The initial position is shown as a colored blob, while the landing spot as a star.

UAV to fly towards bushes, wrongly labeled as grass, while in another case, the UAV almost crashes into a lamppost as this is not present as an obstacle in the depth map. While the object is correctly identified in the semantic image, ORB-SLAM3 does not detect any 3D landmarks on the lamppost because of its featureless visual appearance. Consequently, no sparse seeds are generated and the obstacle is missing in the depth image. Then the policy tries to land unaware of the presence of the obstacle. These experiments show that our pipeline can deal with noisy inputs, but it is still prone to failures in case of systematic errors, especially in semantics.

V. Conclusion and Future Work

In this work, we present a pipeline based on deep RL for autonomous landing of multicopter UAVs in case of an emergency. Our policy maps semantic and depth information onto actions, flying the UAV towards safe areas. We evaluate our system in a series of challenging experiments, both in simulation and reality, demonstrating that we can reach higher success rates and faster landings compared to the state of the art, including a commercially available solution. Moreover, we show that the use of multi-class semantic segmentation is beneficial compared to binary approaches, as it allows the UAV to acquire more context, learning the spatial relationships between the different classes of the scene. In our experiments, using the full semantic knowledge leads to success rates on average 150% higher than cases when binary segmentation is employed. More importantly, our policy transfers directly from simulation to reality, and we demonstrate it can safely land a drone in real-world experiments even when exposed to inputs corrupted by noise.

As future work, we propose to extend the pipeline to consider the uncertainty in semantic classification and in depth prediction to increase robustness against noise, and to further develop the pipeline to work in dynamic scenes.

References

[1] R. Polvara, M. Patacchiola, M. Hanheide, and G. Neumann, "Sim-to-Real Quadrotor Landing via Sequential Deep Q-Networks and Domain Randomization," *Robotics*, 2020.

[2] A. Ramos, C. Sampedro, H. Bavle, I. G. Moreno, and P. Campoy, "A Deep Reinforcement Learning Technique for Vision-Based Autonomous Multirotor Landing on a Moving Platform," in *International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[3] M. Mittal, R. Mohan, W. Burgard, and A. Valada, "Vision-based autonomous uav navigation and landing for urban search and rescue," *International Symposium on Robotics Research (ISRR)*, 2019.

[4] C. Forster, M. Faessler, F. Fontana, M. Werlberger, and D. Scaramuzza, "Continuous on-board monocular-vision-based elevation mapping applied to autonomous landing of micro aerial vehicles," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[5] C. Symeonidis, E. Kakaletsis, I. Mademlis, N. Nikolaidis, A. Tefas, and I. Pitas, "Vision-based UAV Safe Landing exploiting Lightweight Deep Neural Networks," in *International Conference on Image and Graphics Processing (ICIGP)*, 2021.

[6] L. Bartolomei, P. Teixeira, and M. Chli, "Semantic-aware Active Perception for UAVs using Deep Reinforcement Learning," in *International Conference on Intelligent Robots and Systems (IROS)*, 2021.

[7] X. Guo, S. Denman, C. Fookes, and S. Sridharan, "A robust UAV landing site detection system using mid-level discriminative patches," in *International Conference on Pattern Recognition (ICPR)*, 2016.

[8] T. Yang, P. Li, H. Zhang, J. Li, and Z. Li, "Monocular Vision SLAM-Based UAV Autonomous Landing in Emergencies and Unknown Environments," *Electronics*, 2018.

[9] M. A. Pereira, C. A. Duarte, I. Exarchos, and E. A. Theodorou, "Deep $\mathcal{L}^1$ Stochastic Optimal Control Policies for Planetary Soft-landing," in *CoRR*, 2021.

[10] H. Voos and H. Bou-Ammar, "Nonlinear tracking and landing controller for quadrotor aerial robots," in *2010 IEEE International Conference on Control Applications*, 2010.

[11] G. Shi, X. Shi, M. O'Connell, R. Yu, K. Azizzadenesheli, A. Anandkumar, Y. Yue, and S.-J. Chung, "Neural Lander: Stable Drone Landing Control Using Learned Dynamics," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019.

[12] R. Polvara, S. Sharma, J. Wan, A. Manning, and R. Sutton, "Autonomous Vehicular Landings on the Deck of an Unmanned Surface Vehicle using Deep Reinforcement Learning," *Robotica*, 2019.

[13] V. Vasilopoulos, G. Pavlakos, S. L. Bowman, J. D. Caporale, K. Daniilidis, G. J. Pappas, and D. E. Koditschek, "Reactive Semantic Planning in Unexplored Semantic Environments Using Deep Perceptual Feedback," *IEEE Robotics and Automation Letters*, 2020.

[14] P. Fraczek, A. Mora, and T. Kryjak, "Embedded Vision System for Automated Drone Landing Site Detection," in *Computer Vision and Graphics*, 2018.

[15] C. V. Nguyen, S. Izadi, and D. Lovell, "Modeling Kinect Sensor Noise for Improved 3D Reconstruction and Tracking," in *Second International Conference on 3D Imaging, Modeling, Processing, Visualization Transmission*, 2012.

[16] L. O. R. Pérez, R. M. Silva, and J. M. Carranza, "Real-Time Landing Zone Detection for UAVs using Single Aerial Images," in *International Micro Vehicle Competition and Conference (IMAV)*, 2018.

[17] L. Teixeira, M. R. Oswald, M. Pollefeys, and M. Chli, "Aerial single-view depth completion with image-guided uncertainty estimation," *IEEE Robotics and Automation Letters*, 2020.

[18] S. Wang, F. Maffra, R. Mascaro, L. Pinto Teixeira, and M. Chli, "Viewpoint-Tolerant Semantic Segmentation for Aerial Logistics," in *German Conference on Pattern Recognition (GCPR)*, 2021.

[19] B. Chen, A. Sax, G. Lewis, I. Armeni, S. Savarese, A. Zamir, J. Malik, and L. Pinto, "Robust policies via mid-level visual representations: An experimental study in manipulation and navigation," *Conference on Robot Learning (CoRL)*, 2020.

[20] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *CoRR*, 2017.

[21] Y. Song, S. Naji, E. Kaufmann, A. Loquercio, and D. Scaramuzza, "Flightmare: A Flexible Quadrotor Simulator," in *Conference on Robot Learning (CoRL)*, 2020.

[22] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. M. Montiel, and J. D. Tardós, "ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual–Inertial, and Multimap SLAM," *IEEE Transactions on Robotics*, 2021.