Chapter 1

Parking Space Detection using a Convolutional Neural Network and a Reinforcement Learning-based Unmanned Aerial Vehicle in a Virtual Environment

Akhil Giddaluri, Alex Jiang, Nikhil Giddaluri, Thomas Li, Audrey Liang, Dalei Wu and Yu Liang

1.1. Introduction

Unmanned aerial vehicles (UAVs), commonly referred to as drones, have undergone significant improvements over the past few years, becoming increasingly prominent in a wide range of scenarios [1]. Specifically, one of their biggest improvements has been in remote sensing due to their ability to capture high-quality images while being cost-effective [2]. Drones are equipped with multiple high-resolution cameras and sensors, allowing them to easily capture images and detect surrounding objects [3]. One scenario that could benefit from using drones is parking lots, which are often very hectic and crowded, making it difficult and time-consuming to find a vacant parking spot. Drones can be chosen over other methods of parking spot identification, such as CCTV cameras, due to their mobile nature. They are cost-effective, as only a small number of drones are needed to cover a large parking lot. Additionally, if there are obstacles such as cars or trees blocking the view of the camera, a drone can move to find an unobstructed angle while an immobile camera cannot. The use of UAVs are also much more temporary than CCTV cameras; for events that happen a few times per year, the use of drones for parking lot surveillance is much better than

Akhil Giddaluri
McCallie School, Chattanooga, TN 37404, USA

CCTV cameras, since the cameras will not be used apart from the times the event happens. Currently, there are not many real-world methods with the purpose of locating vacant parking spaces. One concern that must be addressed for real-world implementation is battery life. Most drones last around 1–2 h per charge; however, this can be combatted by utilizing multiple drones cyclically, which can sequentially extend the operational time [1].

Reinforcement learning (RL) is a category of machine learning (ML) where an agent gathers inputs and receives feedback through interactions with its environment, outputting the best action based on its policy given the information it has received to maximize accumulated rewards over a period of time [4]. RL has been applied to control UAVs [5]. Drones have been applied to parking lots in several cases, including to detect illegal parking and vacant spaces [6].

In this work, we study the combination of a reinforcement learning algorithm called a Deep-Q-Network (DQN) and a convolutional neural network (CNN) for parking space detection. The DQN was used to automate the drone's movement and manipulate it to avoid colliding with objects while sensing the parking spots. A CNN was utilized to identify if the parking spot was vacant or filled using images of parking spaces. CNNs are frequently used in image classification problems [7].

In this investigation, we chose to study smart drone-based parking space detection using DQN and CNN in a virtual environment, rather than deploying a real drone in a physical setting, due to the latter's higher cost, greater time requirements, and associated safety concerns [8]. Challenges regarding physical parking lots include obstacles such as trees or lamp posts that can damage both the UAV and surrounding property, alongside the need for several consenting automobiles to be parked in logical positions so that the CNN can accurately analyze the parking lot. It is because of these risks associated with using a physical parking lot that led to the use of an inexpensive virtual simulation to train the UAV. Also, taking into account the nature of the reinforcement learning model implemented, it is expected that the drone performs poorly at the beginning of its training. It is taking random actions in order to learn a policy that will yield a great reward; therefore, if this training were to be implemented physically, it would be very expensive, as a high number of collisions would be expected early on. However, real drone-based agent validation/implementation could be a future direction of this investigation.

Camera visibility may become an issue at night. However, due to the Federal Aviation Administration's requirement for drones to be equipped with lights during times between sunset and sunrise, the agent will have sufficient light to capture images from the environment, even when considering the time of day. Thus, drones manufactured for these parking lots will have lights attached to abide by these regulations.

The significance of this work is twofold: in a specific sense, parking in a crowded parking lot can be very time-consuming, and drones coupled with machine learning may assist that process and make it much more efficient, while, in a general sense, this study shows the capabilities of UAVs enhanced by DQN and CNN and their potential applications.

1.2. System Overview & Methodology

In our study, AirSim [9], an open-source, cross-platform drone simulator, was used to simulate the UAV. As stated above, virtual drone training is more feasible, less dangerous, and more cost-effective than real-life drone training, and transitioning this system to real-life is a future direction of this work.

To address this issue, alternative systems have been developed that do not rely on the use of drones. For example, Boda et al., proposed the utilization of ultrasonic and magnetic sensors for parking space detection in parking lots. However, this implementation turned out to be expensive, especially for large-scale parking lots, as each parking space requires a sensor [10].

Methods derived by other researchers to expedite the parking process are generally similar to those presented in this paper, as CNNs, GANs, and YOLOs have all been tested in various papers in this field. For example, Peng et al. explored the use of a DNN to detect parking space occupancy with a 97% average accuracy [6]. Using a unique method, Hsieh et al., utilized LPNs to analyze the CARPK dataset and had metrics that seemed to outperform conventional YOLOs and fast R-CNNs [11]. The aforementioned group also weighed the final prediction with a spatial layout score that changed based on the density of cars surrounding the position in question.

AirSim has been utilized significantly in virtual drone applications. AirSim has been leveraged for real-time autonomous path planning [12]. It has also been used alongside a CNN-based YOLO network and DQN to precisely determine the existence of automobiles and people within a virtual environment [13]. Similarly, multiple studies have utilized CNNs and UAVs to analyze the status of parking spaces, averaging 90 % and 92 % accuracy, respectively [14, 15]. Murani et al. utilizes a CNN to predict if the parking space is empty or occupied [14].

The virtual drone can be seen in Fig. 1.1 below, alongside different car models, tree models, and the parking spaces. To control the drone's movement, Python APIs provided in Microsoft's AirSim documentation were used.

The virtual environment used in this study was first created in Unreal Engine [16] version 4.27.2 and mimics a simple parking lot of 16 spaces, both vacant and occupied. As shown in Fig. 1.1 above. UE version 4.27.2 was used due to its compatibility with AirSim. Models of cars were also imported and placed in these spaces to represent a basic parking lot that the CNN-based drone was tasked to analyze. As for the creation of the parking lot itself, the environment is a simple base plate colored with an asphalt texture with white lines representing the borders of the parking spaces. In the vacant spaces, empty actors were added to guide the drone to capture the image, which would then be input to the CNN. In occupied spaces, the virtual car models' coordinates were used. Some miscellaneous objects (street lamps, trees, *etc*.) were also added to serve as obstacles that the RL-based agent has to navigate around; when combined together, these elements formed a rudimentary parking lot that served as one method to test the viability of the CNN in tandem with the DQN.

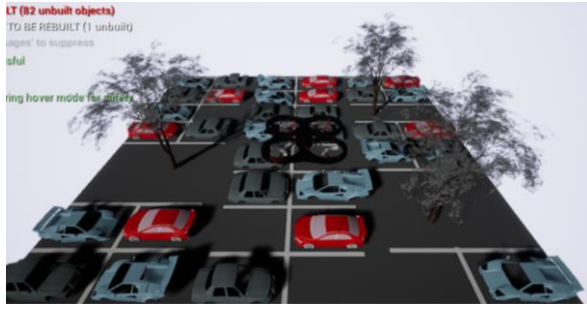


Fig. 1.1. Virtual parking lot in Unreal Engine.

The virtual AirSim drone was the agent, while the Unreal Engine environment (virtual parking lot) served as the environment. The Bellman Equation was used to calculate the return (the accumulated reward) from any state, where $V(s)$ is the expected return at the current state s, $r(s,a)$ is the expected reward for taking action a at state s, γ is the discount factor – a constant that determines the value of future rewards – and $V(s')$ is the expected reward of the next state s' [17].

$$V(s) = max_a\,(r(s,a)) + \gamma V(s') \tag{1.1}$$

The immediate reward component $r(s,a)$ quantifies the direct benefit received when taking action a from state s. This reward can be positive (representing gains), negative (representing penalties), or zero (representing neutral outcomes). The reward structure fundamentally shapes the agent's learning and decision-making process.

The discount factor γ typically ranges between 0 and 1, serving multiple critical functions. When γ approaches 0, the agent becomes myopic, focusing primarily on immediate rewards while largely ignoring future consequences. Conversely, when γ approaches 1, the agent places nearly equal weight on future rewards as immediate ones, promoting far-sighted behavior. The discount factor also ensures mathematical convergence in infinite-horizon problems and reflects practical considerations, such as uncertainty about the future or the time value of rewards.

The state $s \in S$ represents the complete observable configuration of the drone and its environment at time step t. Formally, the state can be defined as: $s = \{p, \theta, v, \omega, E, O\}$ where $p = (x, y, z) \in \mathbb{R}^3$ represents the drone's 3D position coordinates in the world frame, $\theta = (roll, pitch, yaw) \in [0, 2\pi)^3$ represents the drone's orientation angles, $v = (v_x, v_y, v_u) \in \mathbb{R}^3$ represents the drone's linear velocity vector, $\omega = (\omega_x, \omega_y, \omega_u) \in \mathbb{R}^3$ represents the drone's angular velocity vector, E represents the environmental perception data (e.g., camera images, depth sensor readings, LiDAR point clouds), and O represents the set of detected obstacles and their spatial relationships to the drone. The state space S encompasses all possible combinations of these variables.

The action $a \in A$ represents a discrete or continuous control command issued to the drone at time step t. The action space can be formally divided into three categories:

- Movement actions: $a = (\Delta x, \Delta y, \Delta z) \in \mathbb{R}^3$representing translational displacements or velocity commands;
- Rotational actions: $a = (\Delta roll, \Delta pitch, \Delta yaw) \in \mathbb{R}^3$ representing angular displacements or angular velocity commands;
- Composite actions: $a = (\Delta x, \Delta y, \Delta z, \Delta yaw) \in \mathbb{R}^4$ for simultaneous translation and rotation.

Discrete actions include: *a ∈ {forward, backward, left, right, up, down, rotate_left, rotate_right, hover}* for simplified control schemes. The action space *A* is constrained by the drone's physical limitations (maximum velocities, acceleration limits) and safety boundaries defined within the virtual parking lot environment.

R(s,a) can be further broken down to match this paper's suggested reward function. *R(s, a)* gives the reward value per every state-action pair, where w_i is the customizable weight of the reward category *i*. *s* represents a step taken, *c* represents a collision, *ap* represents an altitude penalty, *sv* represents a spot visitation, *cp* represents a correct prediction, and *ip* represents an incorrect prediction.

$$R(s,a) = w_s \cdot s(s,a) + w_c \cdot c(s,a) + w_{ap} \cdot ap(s,a) + \\ + w_{sv} \cdot sv(s,a) + w_{cp} \cdot cp(s,a) + w_{ip} \cdot ip(s,a) \quad (1.2)$$

The DQN algorithm (Algorithm 1) is established below.

Algorithm 1: Drone DQN Algorithm.

1: Initialize Experience Replay buffer D to capacity N = 50000
2: Initialize Q-network with HybridDQNPolicy and parameters θ
3: Initialize target network with parameters $\theta^- = \theta$
4: Initialize drone environment with AirSim client and parking spots S
5: Initialize exploration parameters: ε_start, ε_final = 0.05, exploration_fraction = 0.4
6: **for** episode = 1 to M **do**
7: Reset drone to starting position (-2, 0, -2)
8: Reset episode tracking: spots_photographed = ∅, visit_counts = ∅
9: Get initial observation s^1 = {position, image}
10: for t = 1 to max_steps = 600 do
11: With probability ε select random action $a^t \in$ {0,1,2,3,4,5,6}

12: otherwise, select $a^t = \text{argmax } Q(s^t, a; \theta)$
13: Execute action a^t via drone movement (dx, dy, dz)
14: Capture image if near parking spot (distance < 2.0)
15: Observe reward r^t and next state s^{t+1}
16: if collision or altitude violation or out of bounds then
17: terminal = True
18: else if all spots photographed then
19: terminal = True (success)
20: else
21: terminal = (t ≥ max_steps)
22: Store transition $(s^t, a^t, r^t, s^{t+1}, \text{terminal})$ in D
23: if |D| > learning_starts = 1000 and t mod 4 = 0 then
24: Sample random batch of transitions $(s_j, a_j, r_j, s_j^{+1}, \text{terminal}_j)$ from D
25: for each transition j in batch do
26: if terminal_j then $y_j = r_j$
27: else $y_j = r_j + \gamma \max Q(s_j^{+1}, a; \theta^-)$
28: Perform gradient descent on $(y_j - Q(s_j, a_j; \theta))^2$
29: Update parameters: $\theta \leftarrow \theta + \alpha\nabla\theta$
30: if t mod 1000 = 0 then
31: Update target network: $\theta^- \leftarrow \theta$
32: if terminal then break
33: Update exploration: $\varepsilon \leftarrow \max(\varepsilon_final, \varepsilon - decay_rate)$
34: end for

1.3. Performance Evaluation

This section describes the overall configurations of both machine learning models as well as the results from model training and evaluation.

1.3.1. Configuration of DQN

Fig. 1.2 shows the individual components of our DQN-enabled system.

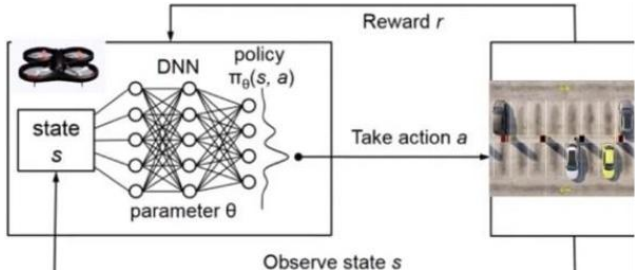


Fig. 1.2. Deep Q-Network training loop with drone as the agent and parking lot as environment.

As stated previously, Fig. 1.2 shows the training loop of the DQN utilized in this investigation: the agent interacts with the environment, performs an action according to its policy, and receives the new state of the environment and reward. This training loop occurs every timestep.

The state, St, in our virtual environment is informed by the observation captured by the drone's camera as well as the (x, y, z) coordinates of the drone. Both these components make up the state. However, an image is not captured every timestep, since the images that should be input to the CNN are only parking space images. In timesteps where images of parking spaces are not taken, a zero tensor is given alongside the position vector as the state. When images of parking spaces are taken, both image data and position data are provided as the state.

There are several possible actions that the virtual drone can take in a timestep. It can move along the *x, y,* and *z* directions. It can hover (zero net movement). The quadrotor can also capture images.

The example DQN algorithm listed in AirSim's documentation was used as a template. From there, the base DQN was expanded on, with one of the primary changes being made to the policy. Since the DQN is guided by a CNN, one of the main changes that was made was to change the DQN's policy to a CNN policy. However, to incorporate the positional vector of the virtual drone, a hybrid policy was used. Additionally, because the drone does not need to capture images when the images do not depict a parking space, the virtual agent was programmed to only capture images when near parking spaces.

The reward function, *r*, is customizable. Our main goals that we wanted to implement in our reward function were to heavily penalize collisions

and going out of the specified *xyz* bounds, reward correct predictions by the CNN, and minimally reward general survival of the drone (without collisions). The reward function is optimizable; per timestep, the specific values are noted in Table 1.1.

Table 1.1. Original reward function.

Category	Condition	Reward Value per Episode	Description
Base Reward	Always active	+1.0	Living reward for each step
Collision	Collision detected	−100.0	Terminates episode if collision occurs
Altitude penalty	Z < min_altitude or Z > max_altitude	−20.0	Flying too high or too low
Spot visitation	Visiting a spot	+20.0	Incentivizes the drone to visit spots
Image classification	Correct prediction	+20.0	Rewards correct predictions
	Incorrect prediction	−2.0	Penalizes incorrect predictions

It is important to note that the reinforcement learning episode would terminate if a collision occurred or if the virtual drone (agent) went out of the specified bounds.

Crafting an effective reward function is one of the most important aspects of RL. Handcrafting a reward function with the correct task specifications is a popular challenge in reinforcement learning [14]. In fact, a phenomenon called reward hacking may occur, where the virtual drone learns to exploit the hand-crafted reward function to increase the long-term return of the agent while not performing according to the engineer's goals. Thus, much time was spent tweaking and testing the reward function to achieve desirable numerical performance (evaluated by the average reward/return) and applicable performance (evaluated by the progression of the number of collisions, number of out-of-bounds instances, and number of correct predictions).

One of the issues that was seen during initial training was that the drone kept revisiting the same spots. This was coupled with a low exploration factor, meaning the drone was exploiting the policy it had learned.

Because, in the real world, it is not acceptable for the drone to only capture/classify images of the same few spots, the reward function was tweaked to penalize excessive revisitation (Table 1.2).

Table 1.2. Modified reward function.

Category	Condition	Reward Value per Episode	Description
Base Reward	Always active	+1.0	Living reward for each step
Collision	Collision detected	−100.0	Terminates the episode if a collision occurs
Altitude penalty	Z < min_altitude or Z > max_altitude	−20.0	Flying too high or too low
Spot visitation	First time visiting a spot	+8.0	Incentivizes the drone to visit new spots
	Second visit to spot	+3.0	
	Third visit to spot	+1.0	
	Fourth+ visit to spot	−5.0	Penalizes the drone for over-visiting the same spot in the same episode
Image classification	Correct prediction	+20.0	Rewards correct predictions
	Incorrect prediction	−2.0	Penalizes incorrect predictions

Results following the modification of the reward function are outlined in Section 3.3.

1.3.2. Configuration of CNN

The convolutional neural network (CNN) was designed using Google Colab. In this investigation, transfer learning was utilized. Transfer learning refers to the process of leveraging knowledge gained from performing one task and applying it to a different task. In this investigation, the CNN used by the DQN was trained on an external dataset before being incorporated into the DQN system. Images of empty and occupied parking spaces sourced from GitHub were fed into the CNN to train the network. The training data had 96 images of empty parking spaces (negative class) and 285 images of occupied parking spaces (positive class). 80 % of these images made up the training set,

while 20 % of these images made up the validation set. After each training epoch, the validation set was used to monitor overfitting. Overfitting occurs when a model memorizes the training data instead of generalizing – learning general patterns – in the data [18]. By using a validation set, if the model is truly memorizing the training data instead of generalizing, its performance on the training data will be visibly higher than its performance on the validation data. Ensuring that the model is not overfit is crucial for real-world applications: if the model is overfitted, it cannot be effectively applied to the real world. The dataset used for final model testing contained 38 images of empty parking spaces and 126 images of occupied parking spaces.

A drawback of the dataset is its size; most deep learning models need more than a few hundred samples for model training. However, we combatted this through data augmentation. Data augmentation is one of many regularizations – referring to attempts at improving the model's ability to generalize – techniques: it works by varying the existing data by applying transformation operations to it, thereby modifying the samples. Transformations that were applied to the data include a horizontal/vertical shift of up to 10 % of the image's original width/height, randomly zooming in by up to 20 %, randomly brightening/dimming the image by up to 20 %, and randomly horizontally flipping the image. By applying these transformations to the data, the model is exposed to variations of the data, and the size of the dataset is synthetically increased. It is important to distinguish that data augmentation (and, by extension, the aforementioned transformations) was only applied to the training set, not the validation or the testing set. The purpose of the validation/testing sets is to observe how the proposed model would perform on unseen, real-world data. If data augmentation is applied to the validation or testing set, the data would be artificially modified and would not resemble real-world data, meaning the model's performance on these data would not represent real-world performance.

During model training, hyperparameters such as the learning rate, optimizer, batch size, number of layers, types of layers, *etc*. were optimized to improve performance.

For final model testing, the learning rate used was 0.001, the optimizer used was Adam, and the batch size used was 32. The final architecture for the CNN is made up of multiple parts. First, the model had an input layer designed for two-dimensional inputs consisting of matrices with

dimensions 62 × 30, corresponding to the height and width of the images (in pixels), respectively. These dimensions matched the average dimensions of all images in the training set. A pair of convolutional and max pooling layers follows the input layer. A dropout layer succeeds these layers; a dropout layer combats overfitting by "dropping out" an optimizable rate of neurons (and their connections) in the preceding layer. It is a regularization technique meant to prevent overfitting [19]. Finally, there is a hidden layer with 128 neurons and the output layer with 1 neuron. This layer's activation function is the sigmoid activation function. The layers are visible in Fig. 1.3.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 62, 30, 32)	896
max_pooling2d (MaxPooling2D)	(None, 31, 15, 32)	0
conv2d_1 (Conv2D)	(None, 29, 13, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 14, 6, 64)	0
flatten (Flatten)	(None, 5376)	0
dense (Dense)	(None, 128)	688,256
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 1)	129

Fig. 1.3. Figure of Model Architecture (Layers, Layer Outputs, Layer Types, and Parameters).

The sigmoid activation function works very well with binary classification, since its range is from 0 to 1, non-inclusive. It is important to note that the value that the sigmoid function outputs is the model's predicted probability that the parking spot is occupied [20]. For example, if the model outputs 0.86, it is predicting that for the given image, there is an 86 % chance that the parking spot is filled. To simplify this, we established that any value outputted by the model that is greater than 0.5 indicates that the image depicts an occupied parking space, while any value less than or equal to 0.5 indicates that the image depicts an empty parking spot.

1.3.3. Results

This section describes the training/evaluation performance for the DQN model and CNN model.

1.3.3.1. DQN Model Training Performance

During training, approximately 150 episodes were completed using the DQN algorithm. Initial training with the original reward function (Table 1.1) revealed a critical limitation: the drone exhibited spatial bias, repeatedly visiting only 3–4 parking spaces while ignoring others, resulting in a baseline mean reward of 109. In Fig. 1.4, this is evident; although the reward function is always positive, which would imply good performance, the figure is misleading. The drone was simply visiting the same spots, which was an oversight in the original reward function, producing a very positive reward. A spot counter was programmed into the algorithm to take this into account.

To address this spatial bias, the reward function was modified (Table 1.2) to include progressive penalties for repeatedly visiting the same spots, thereby incentivizing more exploration. This modification produced dramatic performance improvements, which can be visualized in Fig. 1.5. The mean reward progressed from 109 to 394.5 (109 → 142 → 216 → 230 → 255 → 291 → 394.51), marking a 262 % increase, while the maximum reward in a single episode was 1288.37. This increase is also made more trustworthy by the fact that the drone's clustering behavior was successfully eliminated (verified by our spot counter).

The final trained model achieved a prediction accuracy of 71.49 % while visiting an average of 4 out of 16 parking spots per episode. Safety performance was excellent with only 2 collisions across the final 54 training episodes. The significant performance improvement demonstrates the critical importance of reward shaping in addressing spatial exploration challenges in reinforcement learning applications.

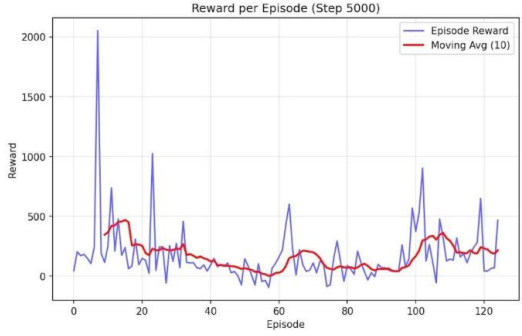


Fig. 1.4. Training metrics using original reward function (with spatial bias).

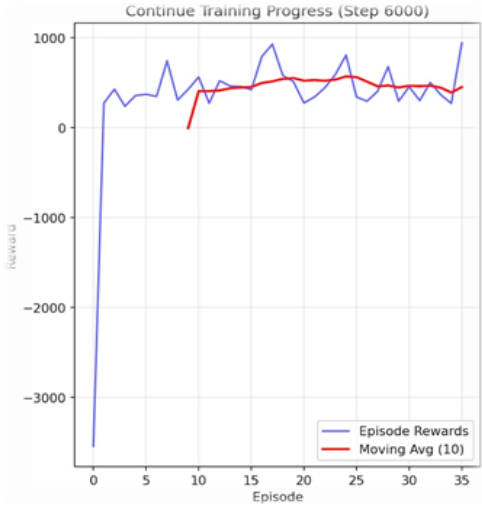


Fig. 1.5. Training metrics after implementing modified reward function to eliminate spatial bias.

1.3.3.2. DQN Model Evaluation Performance

Following training completion, a comprehensive model evaluation was conducted over 100 episodes using deterministic policy evaluation to assess real-world deployment readiness. This means that the model used the policy it learned during training to determine its next action 100 % of the time: there were no random actions. The evaluation results demonstrated exceptional model reliability and consistency, which can be seen in Table 1.3.

Table 1.3. DQN Evaluation Performance Metrics.

Metric	Mean ± Standard Deviation	Range
Prediction Accuracy	70.3 % ± 2.5 %	68.2–80.0 %
Mean Reward (per Episode)	460.42 ± 177.71	185.96–1100.52
Episode Duration	22.9 ± 11.1 steps	8–59 steps

Furthermore, there were no collisions. The evaluation results confirm that the trained model maintains high performance consistency in deployment conditions, with prediction accuracy remaining within a narrow 2.5 % standard deviation band. The perfect safety record

demonstrates the model's suitability for real-world autonomous drone operations. These metrics represent a significant advancement over the baseline performance, validating the effectiveness of the modified reward function in producing a deployable parking detection system.

1.3.3.3. Convolutional Neural Network Performance

For the CNN image classification, the primary metric used to evaluate performance was recall, although others, such as accuracy, precision, F1 score, and AUC score, were reported as well. In machine learning, recall is defined as the number of correct positive predictions made, divided by the sum of the correct positive predictions and false negatives (instances that belonged to the positive class but were predicted as negative by the model) [21]. By penalizing false negatives (where the spot is occupied but the model predicts that the spot is empty), the probability of a driver being misled and going to an occupied spot that they were informed was empty is decreased. Although accuracy is a very common metric, it was not used primarily because of a class imbalance. In crowded lots, there will be more occupied spots (positive class) than negative spots. Therefore, by prioritizing false negatives and using recall (a metric more specialized and capable of handling class imbalance), the model's performance can be transparently conveyed through a metric that has been customized to the dataset's conditions [22].

Still, metrics like accuracy (the total number of correct predictions divided by the total number of predictions) and precision (the total number of correct positive predictions divided by the total amount of positive predictions made) were also considered for performance evaluation [21]. In model testing, the model achieved 98 % recall, 92 % precision, and 93 % accuracy.

1.4. Conclusion and Discussion

The utilization of a UAV based on a DQN and CNN will help drivers navigate through a congested parking lot and locate vacant spaces. In DQN training, the mean reward was 394.5 with a maximum single-episode reward of 1288.37. The model maintained excellent safety performance with only 2 collisions across the final 54 training episodes.

The comprehensive evaluation over 100 episodes using deterministic policy evaluation confirmed the model's deployment readiness. The trained DQN achieved 70.3 % prediction accuracy during evaluation. The model demonstrated exceptional reliability with a mean episodic reward of 460.42 ± 177.71 and episode durations averaging 22.9 ± 11.1 steps. Most importantly, the evaluation achieved a perfect safety record with zero collisions across all 100 episodes, validating the model's suitability for real-world autonomous drone operations.

Since the CNN is accurately able to identify vacant and occupied spots based on images input into the network by the drone (98 % recall), parking will be made more efficient and less time-consuming for the driver, who can then park in the vacant spot identified by the convolutional neural network. A limitation of this work's real-life applicability is that, currently, there is no way for the driver to receive the CNN's predictions: a solution to this limitation (and a future step in this research) is a potential mobile app that facilitates a driver's interaction with the CNN's predictions regarding the vacancy of a parking spot. Ideally, it would be installed on any user's phone to guide them through the parking lot to a vacant parking space. This way, the driver will be able to see in real-time which parking spots are vacant and occupied.

This ability is especially useful when the lot in question is large in size or when it is crowded (*i.e.*, at public events), as it significantly reduces the time it takes a driver to find a parking spot and thereby makes the overall process more efficient. The ever-changing status of the parking lot's vacant and taken spaces also makes the UAV's actions in response to the always-changing state a viable answer to this issue, since the UAV can perform in real-time. The DQN's ability to learn optimal exploration patterns while maintaining high prediction accuracy and perfect safety performance demonstrates that this integrated system represents a significant advancement in autonomous parking detection technology, ready for real-world deployment.

References

[1]. S. A. H. Mohsan, N. Q. H. Othman, Y. Li, M. H. Alsharif, et al., Unmanned aerial vehicles (UAVs): Practical aspects, applications, open challenges, security issues, and future trends, *Intelligent Service Robotics*, Vol. 16, 2023, pp. 109-137.

[2]. J. Fan, M. Saadeghvaziri, Applications of drone in infrastructure: Challenges and opportunities, *International Journal of Mechanical and Mechatronics Engineering*, Vol. 13, Issue 10, 2019, pp. 649-655.
[3]. A. A. Laghari, A. K. Jumani, R. A. Laghari, H. Nawaz, Unmanned aerial vehicles: A review, *Cognitive Robotics*, Vol. 3, 2023, pp. 8-22.
[4]. A. K. Shakya, G. Pillai, S. Chakrabarty, Reinforcement learning algorithms: A brief survey, *Expert Systems with Applications*, Vol. 231, 2023, 120495.
[5]. A. T. Azar, A. Koubaa, N. A. Mohamed, H. A. Ibrahim, et al., Drone deep reinforcement learning: A review, *Electronics*, Vol. 10, Issue 9, 2021, 999.
[6]. C. Peng, Drone-based vacant parking space detection, in *Proceedings of the 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA'18)*, 2018, pp. 618-622.
[7]. L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, et al., Review of deep learning: Concepts, CNN architectures, challenges, applications, and future directions, *Journal of Big Data*, Vol. 8, 2021, 53.
[8]. J. H. Chan, K. Liu, Y. Chen, A. S. M. S. Sagar, et al., Reinforcement learning-based drone simulators: Survey, practice, and challenge, *Artificial Intelligence Review*, Vol. 57, 2024, 281.
[9]. Microsoft, AirSim project repository, https://microsoft.github.io/AirSim/
[10]. V. K. Boda, A. Nasipuri, I. Howitt, Design considerations for a wireless sensor network for locating parking spaces, in *Proceedings of the IEEE SoutheastCon*, 2007, pp. 698-703.
[11]. M. Hsieh, Y. Lin, W. Hsu, Drone-based object counting by spatially regularized regional proposal network, in *Proceedings of the IEEE International Conference on Computer Vision (ICCV'17)*, 2017, pp. 4145-4153.
[12]. T. Ko, J. Park, S. Choi, J. Shim, Autonomous flight of UAV in complex multi-obstacle environment using data-driven and vision-based deep reinforcement learning and AirSim, in *Proceedings of the AIAA Aviation Forum and ASCEND*, 2025, 3686.
[13]. J. Park, K. Farkhodov, S. Lee, K. Kwon, Deep reinforcement learning-based DQN agent algorithm for visual object tracking in a virtual environmental simulation, *Applied Sciences*, Vol. 12, Issue 7, 2022, 3220.
[14]. S. Murani, G. B. Tirado, S. K. Semwal, Automated car parking space detection using deep learning, in *Proceedings of the 29th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG'21)*, 2021, pp. 33-40.
[15]. G. B. Tirado, S. K. Semwal, Autonomous parking spot detection system for mobile phones using drones and deep learning, in *Proceedings of the International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG'21)*, 2021, pp. 115-124.
[16]. Epic Games, Unreal Engine, https://www.unrealengine.com/
[17]. M. Lin, S. Shi, Y. Guo, B. Chalaki, et al., Navigating noisy feedback: Enhancing reinforcement learning with error-prone language models, in

Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP'24), 2024, pp. 10986-11003.
[18]. X. Ying, An overview of overfitting and its solutions, *Journal of Physics: Conference Series*, Vol. 1168, Issue 2, 2019, 022022.
[19]. X. Zhang, P. Srinivasan, S. Mahadevan, Sequential deep learning from NTSB reports for aviation safety prognosis, *Safety Science*, Vol. 142, 2021, 105390.
[20]. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, et al., Dropout: A simple way to prevent neural networks from overfitting, *Journal of Machine Learning Research*, Vol. 15, Issue 1, 2014, pp. 1929-1958.
[21]. P. Fränti, R. Mariescu-Istodor, Soft precision and recall, *Pattern Recognition Letters*, Vol. 167, 2023, pp. 115-121.
[22]. O. Rainio, J. Teuho, R. Klén, Evaluation metrics and statistical tests for machine learning, *Scientific Reports*, Vol. 14, 2024, 6086.
[23]. D. Oladepo, Park-Detection dataset repository, https://github.com/Oladepo/Park-Detection