Simulation-learned drone path planner for efficient real-world object localization in agricultural fields

Rick van Essen
Agricultural Biosystems Engineering
Wageningen University and Research
Wageningen, The Netherlands
0000-0002-5177-4271

Gert Kootstra
Agricultural Biosystems Engineering
Wageningen University and Research
Wageningen, The Netherlands
0000-0002-2579-4324

***Abstract*—Drones are promising for precision agriculture, however, they are limited by their battery capacity. Efficient path planners are therefore required. This paper presents the application of a drone path planner trained in an abstract simulation on real-world data for efficient object search in large fields. By using the agent in combination with the drone's flight controller and a detection network, it is possible to evaluate its performance on real-world data. The simulation-trained path planner yielded a 72% shorter flight path compared to a full coverage planner, however, at the cost of a 25% lower recall. Overall, we conclude that for applications where it is not crucial to find all objects, the learned-based path planner is suitable and efficient.**

I. Introduction

Drones are a promising tool in precision agriculture to gather information about crops [1]. Data gathered by drones can, for example, be used for early disease detection [2], weed detection [3] and soil quality assessment [4]. A limitation of using drones is their limited battery capacity [1], [5], which limits the area that can be covered in a single flight. Traditional coverage path planners map the whole field with an equal spatial distribution, however, this is not needed for all applications. For example, some weed species and plant diseases are distributed in spatially distinct patterns instead of randomly [6], [7]. Using adaptive path planners that are reactive to the environment can make data collection more efficient and less time-consuming [8], by exploiting these spatial patterns.

Adaptive path planners make decisions during flight to adapt the flight path when new information comes available [8]. These decisions can be made rule-based or learning based. The advantages of learning based planning methods over rule-based is their ability to naturally account for uncertainties and changes in an environment [8]. One possible manner to train a learning-based adaptive path planner is Reinforcement Learning (RL). RL can be used to learn an agent to make sequential decisions in an environment by optimizing an action-value function that converts the state of an environment into an action that gives the highest long-term reward [9].

This research is part of the research program Synergia, funding was obtained from the Dutch Research Council (NWO grant 17626), IMEC-One Planet and other private parties. The authors have declared that no competing interest exist in the writing of this publication.

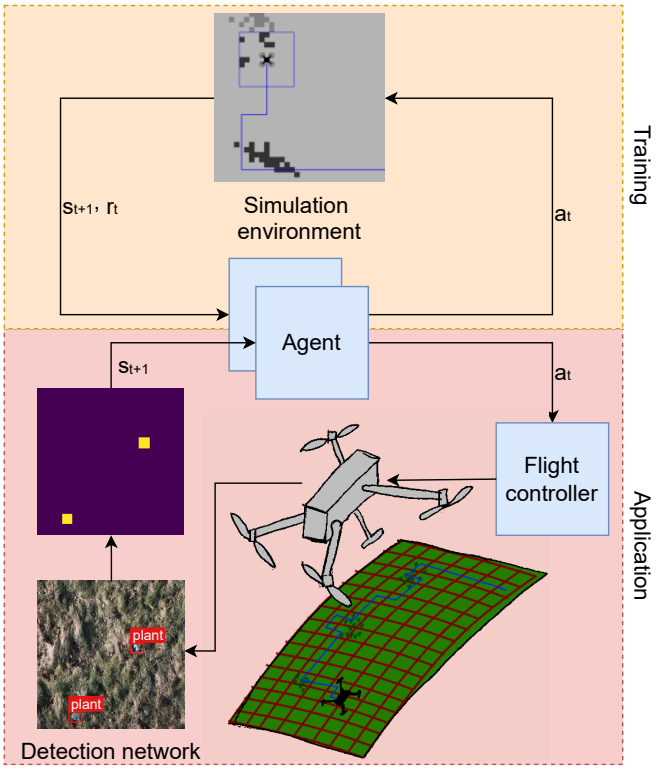


Fig. 1. Overview of the learning-based adaptive path planner which learns a policy in simulation that can be applied in the real world. The agent is trained in an abstract simulation using a Deep Q-Network to select the best action a_t given state s_t that maximizes reward r_t in the long term. During application, the drone's flight controller executes the action a_t and the output from the detection network is converted to state s_{t+1}.

RL has gained more attention for path planning for drones over the last years [10]–[12]. Most RL path planners assume full prior knowledge about the locations the drone has to visit or the obstacles to avoid, however, these prior knowledge is often uncertain or even absent. [11] showed a path planner that uses RL to deviate from its flight path when detecting an obstacle. [12] introduced an RL-based agent for path planning to localize objects of interest, such was weeds, in a field. The agent used uncertain prior knowledge about the locations of the objects in combination with the output of a detection network. In a simulated environment, they showed that the learned path planner outperformed a coverage flight path by

localizing 80% of the objects in less than half of the flight time.

Training an RL agent using real-world data is infeasible, since the agent needs to explore a gigantic state-action space [13]. Fig. 1 shows an alternative approach by training the agent using the abstract simulation environment presented in [12]. For a real-world application, the action selected by the agent is executed using the drone's low-level flight controller, and a detection network is used to create the required state for the agent.

A realistic simulator is needed to transfer a simulation-learned RL policy to the real world. Different methods exist decrease the gap between simulation and reality. Domain randomization is most widely used in RL, where instead of precisely parameterizing the world to match the real-world distribution of data, enough variability is included in the simulator to generalize to real-world data [13]. In this paper, we apply domain randomization by training the network on a huge variety of different object distributions.

The objective of this paper is to evaluate the simulation-learned path planner proposed in [12] on real-world data. To do so, we increased the simulator field size, generate prior knowledge from real-world data and investigate the performance differences of the RL agent between simulation and the real-world data.

II. Materials and methods

In this chapter, we describe the training simulation environment, the training of the RL agent, it's employment on real-world data, the datasets used for evaluation, and the experiments.

A. Simulation environment

The path planner is trained on an abstract field simulator based on [12]. The field has a size of $M \times M \in \mathbb{N}^2$ grid cells. In this field, the number of objects, n_{obj} is drawn from a normal distribution $\mathcal{N}_{\text{obj}}(\mu, \sigma)$ and rounded to the nearest integer. These objects are distributed following $k \sim \mathcal{N}_k(\mu, \sigma)$ multivariate Gaussian distributions. Each distribution i has a random mean coordinate $\boldsymbol{\mu}_i$ and covariance $\Sigma_i \in \{\Sigma_1, \Sigma_2\}$.

A drone with a field-of-view of $N \times N \in \mathbb{N}^2$ grid-cells is flying over the field. The drone starts at a random position at the border of the field. The observation is a binary map with all objects that fall within the field-of-view of the drone. To simulate typical detection errors, each map value has the probability of $p_{\text{dt,fp}}$ to be a false positive and each detection has the probability of $p_{\text{dt,fn}}$ to be a false negative, where false negatives are removed from the observation map. Positional errors are simulated by shifting some objects in the observation map a few pixels. A binary prior knowledge map is generated from the ground truth object locations. Each map value has the probability of $p_{\text{pk,fp}}$ to be a false positive and each object location has the probability of $p_{\text{pk,fn}}$ to be a false negative, where false negatives are removed from the prior knowledge map. Additionally, positional errors are simulated similarly to the observation map. Detection errors are simulated independently for each detection, errors in the prior knowledge are only sampled once for each simulation.

TABLE I
Parameters for the simulation environment.

Symbol	Value	Explanation
M	96	Field size
$\mathcal{N}_{\text{obj}}(\mu, \sigma)$	$\mathcal{N}(100, 30)$	Number of objects in the field
$\mathcal{N}_k(\mu, \sigma)$	$\mathcal{N}(5, 2)$	Number of clusters in the field
Σ_1	$\left[\begin{smallmatrix} 5 & 8 \\ 8 & 15 \end{smallmatrix}\right]$	Covariance distribution 1
Σ_1	$\left[\begin{smallmatrix} 15 & 0 \\ 0 & 5 \end{smallmatrix}\right]$	Covariance distribution 2
N	11	Field-of-view size of the drone
$p_{\text{dt,fp}}$	0.05	Probability false positive detection
$p_{\text{dt,fn}}$	0.0001	Probability false negative detection
$p_{\text{pk,fp}}$	0.0005	Probability false positive prior knowledge
$p_{\text{pk,fn}}$	0.25	Probability false negative prior knowledge
b_{init}	200	Initial battery level
b_{step}	0.2	Battery usage per step
r_{dt}	1.0	Detection reward
r_{step}	-0.125	Step reward
r_{nfz}	-1.0	Hit no-fly-zone reward
r_{crash}	-150.0	Crash reward
r_{nocov}	-0.5	Not exploring new grid-cells reward

Table I shows the parameters of the simulation environment. In comparison to [12], the field size is quadrupled and more object clusters are present in the field.

B. Markov Decision Process

An RL environment is modeled as Markov Decision Process (MDP) [9], containing a state-space, action-space and reward:

1) State-space: Inspired by [14], we model the state-space as a drone-centered global and local map, together with the drone's battery level. The global map gives low-resolution information of the whole field, whereas the local map contains high-resolution information about the drone's field-of-view. The global and local map share three layers: a field-border map indicating the borders of the field, a map containing the locations of the already found objects, and, in addition to [12], a coverage map indicating the grid-cells that are already seen by the drone (to encourage the agent to not revisit previously seen parts of the field). Additionally, the global map contains a layer with the prior knowledge map, whereas the local map contains a layer for the observation map. The global map is padded to a size of $(2M-1) \times (2M-1)$ to make it drone-centered. The local map has a size of $N \times N$. The battery level is calculated by $b = b_{\text{init}} - n \cdot b_{\text{step}}$ where n is the number of flight steps, b_{init} the initial battery level and b_{step} the battery usage for each step. An example of the global and local map is shown in Fig. 2 as input for the network.

2) Action-space: The action-space of the agent contains the following actions: fly north, fly south, fly west, fly east and land. Each action moves the drone one grid-cell in the specified direction, and the land action terminates the search.

3) Reward: A positive reward, r_{dt}, is given for each correctly discovered object, a negative reward, r_{step}, for each flight step, a negative reward, r_{nfz}, for each time the drone tries to fly outside the field and a large negative reward, r_{crash}, when the drone runs out of battery before landing. In addition to [12], a negative reward, r_{nocov}, is given when the agent did not observe any unseen grid-cell to encourage the agent to not revisit already covered parts of the field.

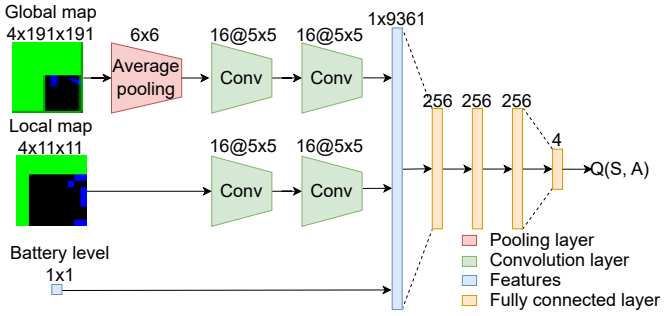


Fig. 2. DQN architecture, showing the pooling, convolutional and fully connected layers with the input size, the number of kernels and their size and the size of the flatten layer. Modified from [12].

TABLE II
TRAINING PARAMETERS FOR THE DEEP Q-LEARNING METHOD.

Symbol	Value	Explanation
n_{buffer}	50000	Experience replay buffer size
n_{batch}	128	Training batch size
γ	0.99	Discount factor
α	$3 \cdot 10^{-5}$	Learning rate
n_{steps}	10^7	Training steps
n_{eval}	120	Number of evaluation simulations

Table I shows the parameter values for the MDP. Compared to [12], the initial battery level, b_{init}, was increased and the negative reward for each flight step, r_{step}, was reduced to account for the larger field size.

C. RL agent

Fig. 2 shows the network architecture for the RL agent that is used to map the state to action values. To limit the number of trainable parameters, the global map is down-sampled using average pooling with a kernel size of 6x6. The network consist of two parallel convolutional layers that extract independent features from the local and global map. The output from both convolutional layers is flattened and concatenated with the battery level of the drone. The resulting feature vector is converted to action values using three fully connected layers.

The agent was trained using Deep Q-Learning (DQN) [15], with a replay buffer size of n_{buffer}, a batch size of n_{batch}, a discount factor of γ and a learning rate α. Training started when 50% of the buffer was filled, and the network was trained for n_{steps} steps. The best weights were selected based on the highest average reward obtained over n_{eval} evaluation simulations. Table II shows the parameters used to train the RL-agent. For more details about the training procedure, we refer to [12].

D. Real-world application

To apply the simulation-trained path planner in the real world, real-world observations need to be mapped to the state-space and the action-space needs to be mapped to real-world coordinates. This requires (1) a conversion between real-world Universal Transverse Mercator (UTM) coordinates and the grid-cell coordinate system of the simulation, (2) a detection network for object detection, (3) a conversion of detected objects to the observation map, and (4) the generation of real-world prior knowledge:

1) Coordinate conversion: To apply the learned adaptive path planner in the real-world, the field needed to be divided into a grid of $M \times M$ grid-cells. The grid coordinates, $[x^{\text{grid}}, y^{\text{grid}}]^T$, of real-world UTM coordinates, $[x^{\text{utm}}, y^{\text{utm}}]^T$ (representing Easting and Northing), are calculated by:

$$\begin{bmatrix} x^{\text{local}} \\ y^{\text{local}} \end{bmatrix} = \begin{bmatrix} \cos(\psi) & -\sin(\psi) \\ \sin(\psi) & \cos(\psi) \end{bmatrix} \cdot \begin{bmatrix} -\dfrac{y^{\text{utm}} - y^{\text{utm}}_{\text{center}}}{s_{\text{grid}}} \\ \dfrac{x^{\text{utm}} - x^{\text{utm}}_{\text{center}}}{s_{\text{grid}}} \end{bmatrix} \tag{1}$$

$$\begin{bmatrix} x^{\text{grid}} \\ y^{\text{grid}} \end{bmatrix} = \begin{bmatrix} x^{\text{local}} \\ y^{\text{local}} \end{bmatrix} + \frac{M-1}{2} \tag{2}$$

where ψ is the counter-clockwise rotation of the grid with the North, $[x^{\text{utm}}_{\text{center}}, y^{\text{utm}}_{\text{center}}]^T$ the grid center and s_{grid} the size of a single grid-cell in meters. The size of a single grid-cell depends on the required resolution and area to be covered by the grid. The grid coordinates are rounded to the nearest integer.

2) Detection work: At every flight step, the drone takes an image from top-down perspective. This image is resized to dimensions $[I_h, I_w]^T$, and YOLOv8-nano [16] is used to detect objects. All detections with a detection confidence lower than 0.5 are discarded. For this work, $Iw = 2048$ and $I_h = 2048$. Sec. II-F describes the training procedure of the detection network.

3) Detections to observation map: The observation map has a size of $N \times N$ grid-cells, corresponding with the field-of-view of the drone, and is independent from the drone's location. The observation map coordinates, $[x^{\text{obs}}_i, y^{\text{obs}}_i]^T$, of object i is calculated by:

$$\begin{bmatrix} x^{\text{obs}}_i \\ y^{\text{obs}}_i \end{bmatrix} = \begin{bmatrix} \dfrac{N}{I_h} \cdot y^{\text{img}}_i \\ \dfrac{N}{I_w} \cdot x^{\text{img}}_i \end{bmatrix} \tag{3}$$

where $[x^{\text{img}}_i, y^{\text{img}}_i]^T$ are the image pixel coordinates of the center of object i and N is the size of the field-of-view in grid-cells. The observation map coordinates are rounded to the nearest integer. Multiple detections can fall within a single element of the map.

4) Generation of prior knowledge: There are multiple ways to generate prior knowledge about the object positions in the field. For this work, we generate a prior knowledge map of size $M \times M$ by a single high-altitude full-coverage flight without overlap. The field-of-view for the high-altitude coverage flight is set to $N_{\text{pk}} \times N_{\text{pk}} \in \mathbb{N}^2$ grid-cells corresponding with a real-world size of $(N_{\text{pk}} \cdot s_{\text{grid}}) \times (N_{\text{pk}} \cdot s_{\text{grid}})$. At every waypoint, an image is taken from top-down perspective, and objects are detected using the detection network. The image coordinates of all detections with a confidence larger than 0.05 are converted to grid coordinates and added to the prior knowledge map. For this work, $N_{\text{pk}} = 48$, corresponding to a total of four images for the whole field.



Fig. 3. Example of the artificial plants in the grass field.

E. Real-world datasets

To allow repeatable experiments on the same data, the learned path planner is evaluated on four offline, real-world, datasets. Each dataset consist of a high-resolution orthomosaic and is geo-referenced in the UTM coordinate system. The ortomosaic contains a grass field with artificial plants to simulate a weed detection application (see Fig. 3). The plants were distributed in clusters in the field and their ground-truth location was measured using a Topcon HiPer SR RTK-GNNS receiver. Images from the field were taken using a DJI M300 drone with a Zenmuse P1 RGB camera at 12m altitude with 70% overlap to the side and 80% to the front. The orthomosaics were made using Agisoft Metashape [17] with the high-quality settings. The dataset is published in [18]. Note that the field shape and area slightly differs between the datasets due to availability of some parts of the field.

Camera images were created by rotating the orthomosaic to ψ. Then, for each flight action, the field-of-view of the drone, $N \cdot s_{\text{grid}} \times N \cdot s_{\text{grid}}$, was cropped from the orthomosaic around the drone's location.

F. Training of the detection network

The YOLOv8 detection network was trained for 250 epochs on a dataset of real-world drone images from the grass field with artificial plants taken from 12, 24 and 32m altitude. The training dataset is identical to that in [18], but with both classes merged into a single 'plant' class. In total 1618 images with 2000 plant annotations were available. The precision and recall on the validation set were 0.94 and 0.95 respectively.

G. Performance metrics

The learned path planner was evaluated using the precision, recall and the flight-path length. A plant was counted as true positive when it falls within the same grid cell as the ground-truth position. Grid cells containing a detected plant without a corresponding ground-truth plant location were counted as false positives, while grid cells with a ground-truth plant location without a detected plant were counted as false negatives. As baseline, we compared the path planner to a traditional full coverage path planner calculated by Fields2Cover [19] using the size of the field-of-view, N, as row width without overlap.

H. Experiments

To show the generalization of the learned path planner to real-world data, we evaluated the path planner on four levels of realism. *Level 1* corresponds to the original training simulation environment as defined in Sec. II-A. *Level 2* uses the same environment but replaces the randomly generated object positions with the real-world object positions from the four datasets. *Level 3* uses the simulation environment with the real-world object positions, but additionally replaces the prior knowledge map with the prior knowledge generated from the real-world data. *Level 4* corresponds to the real-world application as defined in Sec. II-D. Level 1 is evaluated using 1000 random generated fields. Levels 2 - 4 are evaluated using the four real-world datasets. To increase the number of repetitions, each orthomosaic is evaluated four times using a different grid rotation ψ (which creates a completely different state-space for the RL agent), leading to a total of 16 repetitions. We compared the precision and recall and flight path length for both the learned path planner and the baseline full coverage planner.

III. Results

Fig. 4 shows the effect of the flight path length on the recall for the learned path planner and the baseline coverage planner for the four realism levels. For the learned path planner, the higher the realism, the lower the recall, except for level 4. Up to around 500 flight steps, the learned path planner was faster in discovering the objects than the coverage planner. The drop in recall between the full simulation and using real object distributions (level 1 and 2) was smaller then the drop in performance when introducing real-world prior knowledge (level 2 and 3). This could be explained by the higher accuracy of the simulated prior knowledge compared to the real-world prior knowledge. The precision and recall for the simulated prior knowledge was 0.56 ± 0.10 and 0.42 ± 0.06 respectively, whereas real-world prior knowledge reached a precision and recall of 0.24 ± 0.06 and 0.16 ± 0.04. Since the full coverage planner did not use any prior knowledge, the performance with real-world object distribution with and without real-world prior knowledge (level 3 and 4) was equal. The full coverage planner did not find all objects on the real-world dataset (level 4) due to detection errors.

Table III shows the recall for the learned path planner and the baseline coverage planner at 200, 400, 600 and 800 flight steps across the different realism levels. Up until 400 flight steps, the learned path planner had a significant higher recall than the coverage planner, showing that the learned path planner found more objects at that timestep. Already at 200 flight steps, the learned path planner found more than 64% of the objects on average, compared to only 21% for the coverage planner. After 800 flight steps, the coverage planner had found more objects than the learned path planner, except when using

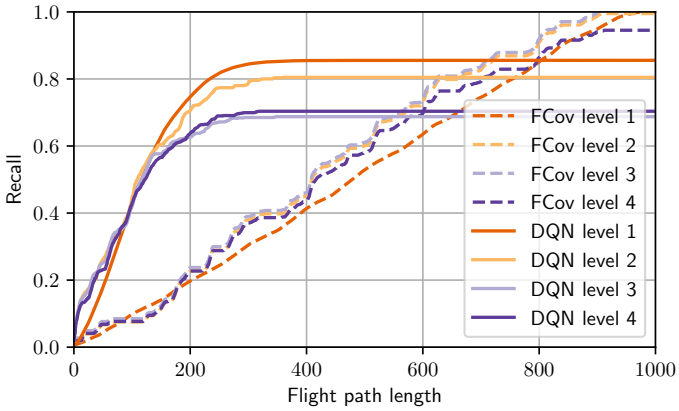


Fig. 4. Effect of flight path length on the recall for the learned path planner (DQN) and the baseline full coverage planner (FCov) with the different realism levels. Note that the lines for 'FCov level 2' and 'FCov level 3' are equal.

fully simulated fields (level 1). The recall of the simulations with the real-world object distribution and real-world prior knowledge (level 3) had a lower recall than the real-world evaluation (level 4), showing that the errors of the real detector were lower than the simulated errors with the parameters from Table I.

Table IV shows the precision, recall and flight path length at the end of the flight for the learned path planner and full coverage path planner at the different realism levels. The learned path planner had higher precision than the coverage planner in simulation, because the coverage planner's longer flight path led to more simulated false positives. The precision on the real-world data was equal for the learned path planner and the coverage planner. On real-world data, the learned path planner yielded a 72% shorter flight path compared to the full coverage planner, however, at the cost of a 25% lower recall.

Fig. 5 show some flights for the learned path planner on the four real-world datasets. Most clusters of objects were found by the agent, however, within each cluster, not all objects were found. Mostly the outer objects within each cluster were not detected. Possibly, discovering new clusters was deemed more rewarding by the agent. Although these flight paths quickly discover most objects, the flight paths were still not optimal and terminating the search was often done too late.

IV. Discussion

On the real-world datasets, the learned path planner showed a 72% shorter flight path compared to a coverage planner, at the cost of a 25% lower recall. [18] showed a 37% shorter flight path compared to a coverage planner for a rule-based adaptive path planner on the same dataset, at the cost of a 2% lower F1-score performance. In comparison to the rule-based planner, the learned-based planner is much faster, however, it has a lower detection performance. When it is important to find most objects in a short time-window, using the learning based path planner has clear advantages. However, when it is important to find all objects without time limitation, a full coverage planner or a rule-based planner is more suitable. For a weed detection application, it may be fine to miss some weeds. By doing multiple flight throughout the season, weeds missed in one flight may be detected in a later flight.

Around one-third of the lower recall between the simulation environment and the real world datasets can be attributed to the real-world object distribution. One possible explanation for the lower performance is that the real-world datasets consist of a rectangular field, which, when transferred to the square simulation environment, resulted in the clusters not being distributed across the entire field. In the simulation environment on which the RL agent was trained, the clusters were distributed over the entire area. Increasing the domain randomization of the object distribution in the training simulation could possibly help the agent to learn to deal with object clustered that were not spread throughout the whole field.

Two-third of the lower recall on the real-world datasets compared to the simulation environment was due to the use of real-world prior knowledge. This could be due to the lower precision and recall of the prior knowledge on the real-world data compared to the simulated prior knowledge. Additionally, because the prior knowledge was generated from high-altitude images, small localization errors could result in detections being placed in the wrong grid-cell. Results from [12] indicates that the RL-agent could deal with high levels of uncertainty in the prior knowledge. Therefore, a possible solution is to increase the uncertainty of the prior knowledge in the training simulation. Another option is to reduce the resolution of the prior knowledge map, which decreases the misalignment in grid position between the prior-knowledge map and the real field.

The flight steps needed to gather prior knowledge using the high-altitude coverage flight were not included in the flight distance. Including the high-altitude coverage flight would increase the flight path length by around 250 flight steps. Even by including these flight steps, the learned path planner was quicker than the baseline full coverage planner. However, since the waypoints for the coverage flight can be calculated on forehand, and the distance between the waypoints is large, the drone's flight speed may be larger for the coverage flight than for the flight path executed by the RL agent. Additionally, when using the learned-based path planner for frequent field monitoring, detected objects from previous flights may be used as prior knowledge, eliminating the need for a coverage flight to collect prior knowledge.

V. Conclusion

This paper showcases the application of a simulation-trained adaptive path planner on real-world data. The learned path planner showed a 72% shorter flight path compared to a full coverage planner, however, at the cost of a 25% lower recall. For applications where it is important to find all objects in the field, a coverage planner is more suitable. However, when the goal of the application is to find most objects in a short time window, the learned path planner is useful.

TABLE III

RECALL VALUES AT DIFFERENT FLIGHT STEPS FOR THE LEARNED PATH PLANNER (DQN) AND THE BASELINE FULL COVERAGE PLANNER (FCOV). THE VALUES SHOW THE MEAN ± THE STANDARD DEVIATION. VALUES INDICATED WITH A '*' HAVE A SIGNIFICANT ($\alpha = 0.001$) HIGHER RECALL THAN THE BASELINE FULL COVERAGE FLIGHT PATH.

Method	200 steps	400 steps	600 steps	800 steps
FCov level 1	0.20 ± 0.19	0.41 ± 0.25	0.63 ± 0.25	0.85 ± 0.17
FCov level 2	0.23 ± 0.14	0.45 ± 0.18	0.73 ± 0.21	0.91 ± 0.09
FCov level 3	0.24 ± 0.14	0.46 ± 0.18	0.74 ± 0.21	0.92 ± 0.09
FCov level 4	0.23 ± 0.13	0.43 ± 0.18	0.69 ± 0.19	0.86 ± 0.08
DQN level 1	0.74 ± 0.16*	0.85 ± 0.10*	0.86 ± 0.10*	0.86 ± 0.10
DQN level 2	0.70 ± 0.09*	0.80 ± 0.08*	0.80 ± 0.08	0.80 ± 0.08
DQN level 3	0.63 ± 0.10*	0.69 ± 0.09*	0.69 ± 0.09	0.69 ± 0.09
DQN level 4	0.64 ± 0.07*	0.70 ± 0.10*	0.70 ± 0.10	0.70 ± 0.10

TABLE IV

PRECISION, RECALL AND FLIGHT PATH LENGTH FOR THE LEARNED PATH PLANNER (DQN) AND THE FULL COVERAGE PLANNER (FCOV) AT DIFFERENT REALISM LEVELS. VALUES INDICATE THE MEAN ± THE STANDARD DEVIATION.

Realism level	Precision		Recall		Flight path length	
	DQN	FCov	DQN	FCov	DQN	FCov
1	0.82 ± 0.07	0.77 ± 0.06	0.86 ± 0.10	1.00 ± 0.00	242 ± 74	962 ± 14
2	0.78 ± 0.04	0.70 ± 0.03	0.80 ± 0.08	1.00 ± 0.00	267 ± 61	956 ± 8
3	0.78 ± 0.03	0.72 ± 0.04	0.69 ± 0.09	1.00 ± 0.00	248 ± 63	956 ± 8
4	0.87 ± 0.03	0.87 ± 0.02	0.70 ± 0.10	0.95 ± 0.03	269 ± 52	956 ± 8

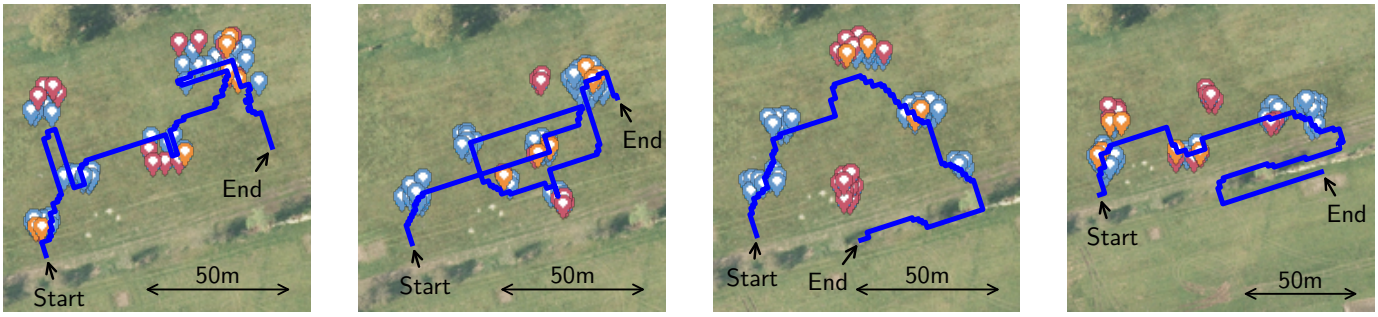


Fig. 5. Flight paths for the learned path planner on the four real-world datasets. Blue, orange and red markers indicate true positive, false positive and false negative markers respectively. Start and end location of the flight path is indicated with an arrow.

REFERENCES

[1] R. Guebsi, S. Mami, and K. Chokmani, "Drones in Precision Agriculture: A Comprehensive Review of Applications, Technologies, and Challenges," *Drones*, vol. 8, no. 11, p. 686, 2024.

[2] A. Bouguettaya, H. Zarzour, A. Kechida, and A. M. Taberkit, "A survey on deep learning-based identification of plant and crop diseases from UAV-based aerial images," *Cluster Computing*, vol. 26, no. 2, pp. 1297–1317, 2023.

[3] M. Gašparović, M. Zrinjski, Đ. Barković, and D. Radočaj, "An automatic method for weed mapping in oat fields based on UAV imagery," *Computers and Electronics in Agriculture*, vol. 173, p. 105385, 2020.

[4] R. Näsi, H. Mikkola, E. Honkavaara, N. Koivumäki, R. A. Oliveira, P. Peltonen-Sainio, N.-S. Keijälä, M. Änäkkälä, L. Arkkola, and L. Alakukku, "Can Basic Soil Quality Indicators and Topography Explain the Spatial Variability in Agricultural Fields Observed from Drone Orthomosaics?" *Agronomy*, vol. 13, no. 3, p. 669, 2023.

[5] A. Rejeb, A. Abdollahi, K. Rejeb, and H. Treiblmaier, "Drones in agriculture: A review and bibliometric analysis," *Computers and Electronics in Agriculture*, vol. 198, p. 107017, 2022.

[6] S. Garibay, W. Richner, P. Stamp, T. Nakamoto, J. Yamagishi, C. Abivardi, and P. Edwards, "Extent and Implications of Weed Spatial Variability in Arable Crop Fields," *Plant Production Science*, vol. 4, no. 4, pp. 259–269, 2001.

[7] C. L. Campbell and J. P. Noe, "The Spatial Analysis of Soilborne Pathogens and Root Diseases," *Annual Review of Phytopathology*, vol. 23, no. 1, pp. 129–148, 1985.

[8] M. Popović, J. Ott, J. Rückin, and M. J. Kochenderfer, "Learning-based methods for adaptive informative path planning," *Robotics and Autonomous Systems*, vol. 179, p. 104727, 2024.

[9] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*. Cambridge, Massachusetts, USA: MIT press, 2018.

[10] A. T. Azar, A. Koubaa, N. Ali Mohamed, H. A. Ibrahim, Z. F. Ibrahim, M. Kazim, A. Ammar, B. Benjdira, A. M. Khamis, I. A. Hameed, and G. Casalino, "Drone Deep Reinforcement Learning: A Review," *Electronics*, vol. 10, no. 9, p. 999, 2021.

[11] G. T. Tu and J. G. Juang, "UAV Path Planning and Obstacle Avoidance Based on Reinforcement Learning in 3D Environments," *Actuators 2023, Vol. 12, Page 57*, vol. 12, no. 2, p. 57, 2023, publisher: Multidisciplinary Digital Publishing Institute.

[12] R. van Essen, E. van Henten, and G. Kootstra, "Learning UAV-based path planning for efficient localization of objects using prior knowledge," 2024, arXiv:2412.11717 [cs].

[13] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: a Survey," in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, Dec. 2020, pp. 737–744.

[14] M. Theile, H. Bayerlein, R. Nai, D. Gesbert, and M. Caccamo, "UAV Path Planning using Global and Local Map Information with Deep Reinforcement Learning," in *2021 20th International Conference on Advanced Robotics (ICAR)*. Ljubljana, Slovenia: IEEE, 2021, pp. 539–546.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through

deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[16] G. Jocher, J. Qiu, and A. Chaurasia, "Ultralytics YOLO," 2023, version: 8.3.49. [Online]. Available: https://github.com/ultralytics/ultralytics

[17] Agisoft, "Metashape Professional," St. Petersburg, Russia, 2023, version: 2.0.3. [Online]. Available: https://www.agisoft.com

[18] R. van Essen, E. van Henten, L. Kooistra, and G. Kootstra, "Adaptive path planning for efficient object search by UAVs in agricultural fields," 2025, arXiv:2504.02473 [cs].

[19] G. Mier, J. Valente, and S. de Bruin, "Fields2Cover: An open-source coverage path planning library for unmanned agricultural vehicles," *IEEE Robotics and Automation Letters*, 2023.