

Encoder-Decoder Reinforcement Learning for Active Search and Coverage in Point Clouds

Matthias ROSYNSKI¹ and Lucian BUȘONIU^{1,2,*}

¹Technical University of Cluj-Napoca, Memorandumului 28, 400114 Cluj-Napoca, Romania

²Romanian Academy, Calea Victoriei 125, Sector 1, 010071 Bucharest, Romania

Email: `rosynski.matthias@aut.utcluj.ro`, `lucian.busoniu@aut.utcluj.ro`*

* Corresponding author

Abstract. Consider an *active target search and coverage* problem in which a mobile 3D sensor aims both to quickly find relevant targets in its environment and to quickly scan these targets so as to cover them. We propose a deep reinforcement learning method to find a sequence of sensor poses that solves this problem. The method uses a deep hierarchical encoder-decoder architecture that heavily modifies the point cloud transformer network. To reduce computation, a new way is proposed to select representative points from the cloud, based on cosine similarities between feature vectors. In simulated experiments, the novel encoder-decoder architecture significantly improves performance compared both to a greedy baseline, and to a network structure closer to the point cloud transformer. The architecture also improves classification and segmentation performance in supervised learning on point clouds. We successfully train much deeper networks than those usually employed in reinforcement learning, and we believe the ideas used to achieve this can be adapted to deep reinforcement learning in general.

Key-words: Active sensing; artificial intelligence; deep reinforcement learning; point cloud.

1. Introduction

The search for targets in the environment of a robot is an important objective in active robotic sensing. Examples include search and rescue, where the targets are victims [1], as well as looking for mines [2], litter, or weeds [3]. We consider a scenario in which a mobile robot equipped with a 3D sensor aims not only to find relevant targets as quickly as possible, but also to scan these targets so as to fully cover them. Coverage is needed to obtain maximal information about the targets for later tasks, such as classification [4]. We call this problem *target search and coverage*, and propose an end-to-end deep reinforcement learning (RL) approach to solve it. One

such problem involves an underwater vehicle equipped with a sonar, such as in the SeaClear2.0 project (<https://seaclear2.eu/>), where the goal is to locate and identify submerged litter objects. The type of object will often not be apparent from a single sonar scan, and different images must instead be taken from various viewpoints. Similarly, in air, LiDAR sensors are often used to search for and identify objects [5].

An RL agent uses experience obtained by interacting with the environment to optimize cumulative, long-term rewards [6]. Here, the main input of the agent is a 3D point cloud representation of the scene, which is partitioned in two subsets of points: targets and background, using another network that we call a discriminator [7].¹ More reward is received for finding target points, while still rewarding background points less in order to encourage exploration. The discriminator is trained from ground-truth data collected by the agent along random-action trajectories that are therefore representative for the task. To evaluate our methods, a simulator is developed where cylinders or pyramids must be found by a mobile Kinect sensor in a multi-room environment.

Inspired by the point cloud transformer (PCT) [8] and a few other methods, we devise a novel deep neural network architecture for RL to address target search and coverage. The network combines three main components: encoder, decoder, and tail. The encoder uses deep hierarchical feature learning [9], selecting at each hierarchical stage a subset of representative points. This selection is done either with farthest point sampling [9] or with a newly developed, computationally cheaper alternative: cosine similarity sampling. The second, decoder component takes the synthetic information provided by the representative points selected by each encoder stage, and merges it back across stages, in a novel transformer structure for point clouds. The final component is an RL tail that combines distributional dueling [10, 11] with spectral normalization [12].

Previous studies have reported that large reinforcement learning (RL) networks are prone to instabilities and often perform worse than smaller networks [13–17]. In contrast, our proposed large encoder-decoder network outperforms a smaller network, which is still larger than those typically employed in RL. To our best knowledge, the only bigger network so far is the adaptive agent network from [18], which has 500 million parameters. In that method, a smaller “teacher” network helps the bigger “student” network kick-start the learning. In contrast, our method directly learns the 163 million parameters of a single large network.

While creating the novel network structure for RL, we also make several improvements to *supervised* learning on point clouds. Firstly, when combined with PCT, cosine similarity sampling yields better results than farthest point sampling on the ModelNet40 classification dataset [19], at significantly smaller computational costs. Secondly, the encoder-decoder structure works better than PCT on the ShapeNet Parts dataset for segmentation [20].

Returning to the active search and coverage problem, we compare RL using the best network structure found to a greedy method that at each step selects a next pose for the sensor maximizing the immediate reward. For this purpose, the next observation must be known in advance, so this baseline is stronger than would be implementable in practice. We nevertheless manage to improve upon it using RL, in terms of both mean performance and variance.

Compared to [21], where we initially proposed active search and coverage, both the methodology and the experiments are new. *Methodologically*, the encoder-decoder architecture is novel, with the key innovation of a hierarchical decoder structure; in [21] we used a network much closer to PCT. Furthermore, cosine similarity sampling is new, and we use the discriminator

¹Even though the discriminator segments the point cloud, we prefer to reserve the term “segmentation” for other concepts, such as a network similar to point cloud transformers for segmentation, which we employ for RL.

neural network to automatically separate targets from background, whereas [21] required this separation to be performed a priori, by an oracle. The supervised-learning results are also novel. *Experimentally*, we consider a larger and more intricate environment consisting of three rooms, as opposed to a single room in [21].

2. Related Work

While we are the first to address active target search and coverage, point-cloud RL exists for coverage-only problems. In [22], RL is used to find out a sequence of viewpoints that may each transport the sensor anywhere in the environment, whereas in our method the sensor moves locally at each step, in order to realistically reflect the dynamics of a robot. Moreover, in [22] a CAD model is built from the point cloud, whereas here the point cloud is directly used as an RL input. In [23], RL explores the environment using both a different network structure from ours (convolutional) and a different representation that preprocesses the point cloud into e.g. an occupancy grid. In [24], a point cloud is also used as part of the RL state, but the problem solved is different from ours: navigation. Despite modifying ShellNet [25] to preserve geometric data, the approach of [24] still relies on max-pooling, resulting in the loss of some spatial information. The method of [4] finds the pose of a known target in a point cloud that is supplied fully from the start, whereas here we incrementally construct the cloud so as to cover the targets. Reference [26] segments an already given point cloud using RL and a PointNet-based network.

Regarding representative point sampling, the critical points layer [27] uses max-pooling to select one point for each feature, providing an alternative to farthest point sampling. Points that are frequently selected are sampled because they are seen as the most important. This method avoids the use of K -nearest neighbors, so it is computationally cheaper than farthest point sampling. However, it has significantly worse performance, whereas our cosine similarity sampling improves performance while also reducing computational cost. Differently from our goals, the method of [28] uses learning to get dense, uniform point clouds.

Overall, we show that – differently from recent techniques in 2D active target sensing using RL [29] or informative path planning [3] – it is possible to learn not only how to search for targets in a 3D environment, but also how to cover them, directly from 3D shape information.

3. Preliminaries

Consider a point cloud $P_t = \{p_i \in \mathbb{R}^3 \mid i = 1, \dots, n_t\}$ at discrete time t that represents a scene containing targets of interest at initially unknown locations. The point cloud is incrementally constructed by a mobile sensor, which has pose (position and orientation) q_t at time t . From point cloud P_t and newly measured points z_t , an updated point cloud P_{t+1} is obtained as explained in Section 3.1. Each cloud P_t comprises points T_t belonging to targets and background points B_t . A neural-network discriminator approximately separates targets from background.

The objective is to find a sensor path (sequence of sensor poses) $\mathbf{q} = \{q_0, q_1, \dots\}$ so that the targets are both found and covered with points in as few steps as possible. This objective will be encoded in an RL reward function as explained in Section 3.2.

3.1. Simulator

To investigate RL for active target search and coverage, we developed a simulator using the Robot Operating System [30] and Gazebo [31]. A mobile Kinect sensor is modeled that stays horizontally oriented and moves in a 2D plane, so that its pose $q_t \in \mathcal{R}^2$ consists of a 2D position and a scalar orientation (yaw). The sensor moves only in discrete cells, so we will use these

cells as a unit of measurement. The sensor receives point returns from surfaces that are between 0.2 and 5 cells away, and has a resolution of 640×380 points. Each 3D point cloud z_t newly measured by the sensor is concatenated with the previous point cloud P_t , after which a voxel filter [32] eliminates duplicates and decreases the number of points, so as to reduce computational and training time requirements. The new point cloud P_{t+1} is thereby obtained.

The experiments are run in the three-room problem illustrated in Fig. 3. The environment has a size of 14×14 cells, but due to walls, the agent can only move in an area of 12×12 cells. The sensor elevation is 1 cell, the walls are 2 cells high, and the targets sought are pyramids of size 2 cells on all axes, see Fig. 3. A voxel filter with a distance of 0.24 cells is applied to reduce the number of points while still placing an adequate number of points on the targets.

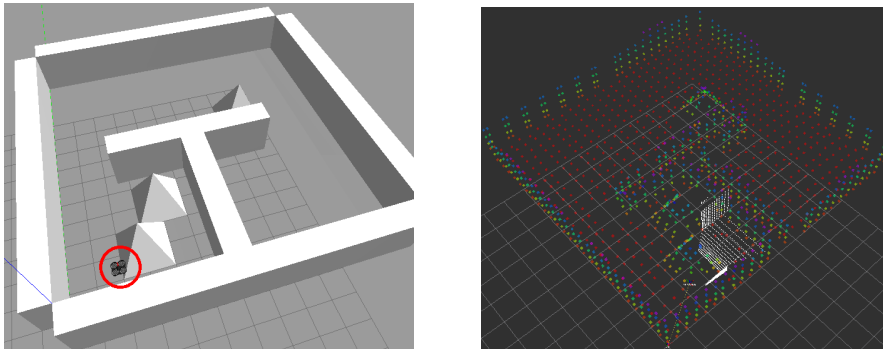


Fig. 1. Left: The environment with three rooms and three objects (pyramids), where the sensor is represented by the drone circled in red near the lower-left pyramid. Right: A point cloud of the same environment at a late step in one of the experiments. The new incoming point cloud measurement z_t is shown in white. The colored points represent the point cloud P_t prior to concatenation with z_t , where the color of each point indicates its height, colors representing lower points.

3.2. RL problem

The problem is modeled as a Markov decision process [33] $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ where \mathcal{S} is the state space, \mathcal{A} the action space, \mathcal{T} gives the transition dynamics, and \mathcal{R} is the reward function. At time step t , the state is $s_t \in \mathcal{S}$ and the agent performs an action $a_t \in \mathcal{A}$. As a result, the state stochastically changes to $s_{t+1} \sim \mathcal{T}(\cdot | s_t, a_t)$ and the agent receives a reward $r_{t+1} = R(s_t, a_t, s_{t+1})$. The objective is to maximize the expected value of the discounted return $G_t = \sum_{j=1}^{\infty} \gamma^j r_{t+j}$, where $\gamma \in (0, 1)$ is a discount factor.

Policy $\pi(a|s)$ indicates for each state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$ the probability of executing a in s . The Q-value $Q_\pi(s, a)$ is the expected return G_t when starting from s , performing a , and then following π : $Q_\pi(s, a) := \mathbb{E}_{\pi, \mathcal{T}} [G_t | s_t = s, a_t = a]$. The optimal Q-value function is $Q^*(s, a) = \max_{\pi} Q_\pi(s, a)$, and the RL objective is to learn from interaction with the environment an optimal policy π^* that achieves Q^* .

States and observations: The state s_t includes the true scene with its background and targets, the filtered point cloud P_t gathered so far, and the agent's pose q_t . Note that the true scene is of course unknown in advance, and q_t and P_t are in fact a partial *observation* [34] of the state. However, following standard practice in deep RL, we will use algorithms that are designed for state signals and simply plug in the observations instead of the states.

Actions and transitions: The actions a_t are performed with respect to the agent’s local reference frame. At each time step, the agent can either move one cell forward, backward, left, or right; or rotate 90° clockwise or counterclockwise. When the agent attempts to move into a wall or target, an illegal, terminal transition occurs.

Rewards: The objective is to find and cover the targets T as quickly as possible, so in general rewards will focus on the amount of target points found. Nevertheless, to help the agent explore more effectively, we also reward it – to a lesser extent – for finding new background points:

$$r_{t+1} = \max\{0, 1.25(|\hat{T}_{t+1}| - |\hat{T}_t|) + 0.25(|\hat{B}_{t+1}| - |\hat{B}_t|)\} \quad (1)$$

where \hat{T} , \hat{B} denote the target and background points as approximately determined by the discriminator network. Sometimes, due to the approximate nature of the discriminator, the number of target points decreases from one step to the next, leading to a negative reward which could make the agent needlessly avoid the associated states. To prevent this, rewards are kept positive via the max operation.

4. Algorithm and Network Structure

The experiments use the Rainbow [35] deep RL approach, together with collision-free exploration. Collisions cause the trajectory to end with a bad return, discouraging the agent from revisiting positions from which collisions are possible. In order to prevent this, terminal transitions that would result in collisions are prohibited. To still allow the agent to learn from such transitions, the collision-generating action is nevertheless stored in the experience replay buffer, together with its associated reward and a termination flag. A different action is then taken to continue the trajectory.

We use a number of networks for deep RL, for the discriminator, and more generally for supervised learning. These networks are all constructed using elements shown in Fig. 2, to which we will refer many times throughout the paper. We explain here how network components and layer types are indicated visually, leaving algorithmic details for the following subsections. Green-shaded blocks show the three main components of the network: encoder, decoder, and tail, the latter with two types: for supervised learning or for RL. Turquoise blocks are detailed, zoomed-in views of an encoder stage (with two options: farthest point sampling and cosine similarity), and the purple block details a decoder stage. Blue-shaded, dotted-edge blocks show elements that form the core of the PCT classification or segmentation networks [8]. The notations E_b , D_b , P_b , F_b , Y_b respectively mean encoder, decoder, points, point features, and decoder outputs at stage b ; the number of stages is B . The two E_1 stages are never used together. In the encoder, point paths are blue, feature paths are black, and indices of selected points are red. In some blocks, signals can follow two paths depending on the network variant, as explained later; in that case, the default path is dashed and the alternative path is dotted.

Moving on to layers types, linear layers are pink and labeled by “NNNf” with NNN the number of features. Max-pooling layers are yellow, and purple layers perform concatenation. Brown layers are reshaping the input, where M is the batch size, u is the feature size, v is the number of points, and K is the number of nearest neighbors. Fully connected layers are dark blue, labeled “FC NNN” with NNN the number of outputs. Dropout layers are light blue, labeled by the dropout probability. In the encoder stage zoom-ins, blue layers perform representative point sampling, and the orange layer selects matrix rows with the smallest sum. The bright purple “Get clouds” layer extracts the representative points and their features based on the indices. The light purple layer “K-NN” finds K neighbors for each point, followed by centering in the light blue

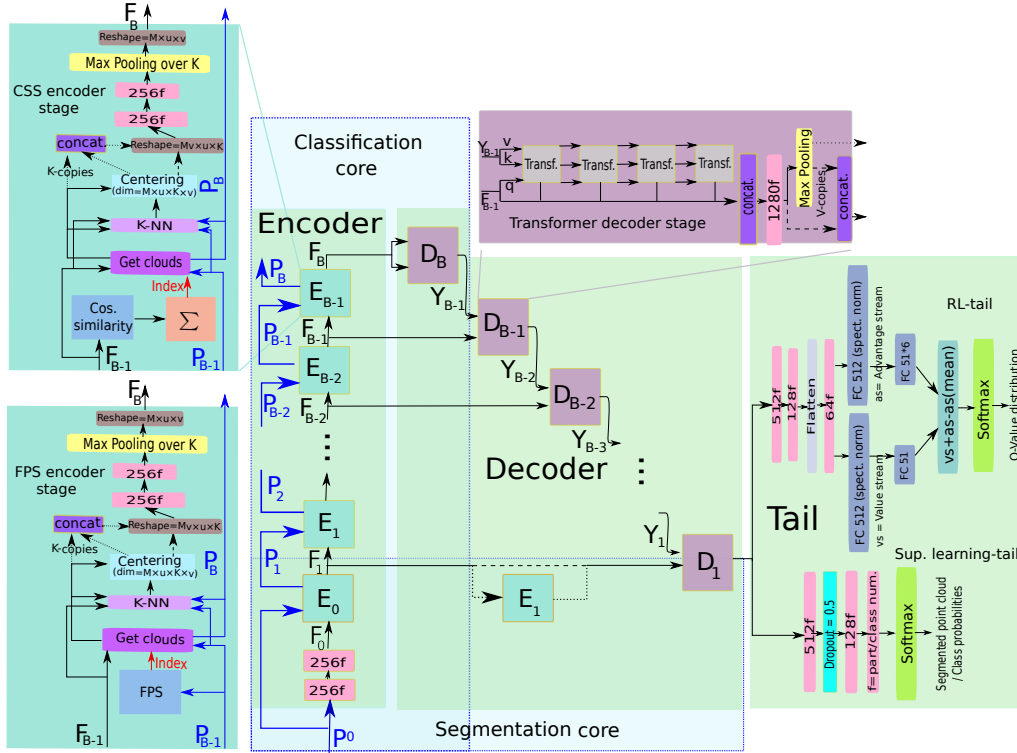


Fig. 2. Network architecture template. The figure is meant to be used together with the explanations throughout Section 4, and cannot be understood in isolation. Notations, layer types, colors, and the meanings of the network elements are all explained in the text (including e.g. the two versions instances of encoder stage E_1).

layer. In the decoder, light gray layers are transformers, each with query, key, and value inputs “q”, “k”, “v”. In the tail, green indicates softmax layers. The supervised learning tail outputs either class probabilities or the segmented point cloud, depending on whether the network is used for classification or segmentation. The blue layer in the RL tail computes Q-values using dueling, and then a softmax layer provides distributional Q-values.

As input, the network receives the point cloud, which is translated and rotated into the agent’s coordinate system. Thus, from the agent’s point of view its actions rotate and move the point cloud, so the sensor pose no longer needs to be included explicitly.

To make it easier to keep track of the various architectures we use in the sequel, we establish a standard naming convention of the form Core-Repres+Tail. Core is the core of the network, and can be one of Segm(entation), Classif(ication), or Enc(coder)Dec(oder) consisting of the full encoder and decoder blocks. Repres is the representative point sampling algorithm, one of FPS (farthest point sampling) or CSS (cosine similarity sampling). Note that Segm does not use representative point sampling, so Repres will be skipped for this core type. Tail is the network tail, one of RL, Segm(entation), or Classif(ication), respectively for the RL tail, or the supervised learning tail configured for segmentation or classification. Thus, for example, the final network that we use for RL is EncDec-CSS+Segm: its core consists of the complete multistage encoder and decoder, CSS is used in the encoder for representative point sampling, and the RL tail is

appended; all this along the default signal paths (dashed).

Next, we delve into the components of Fig. 2, starting with the encoder in Section 4.1 and the decoder in Section 4.2. The PCT classification and segmentation cores are discussed in Section 4.3. The tails for RL and supervised learning are discussed in Sections 4.4 and 4.5, respectively.

4.1. Encoder

The encoder embeds the points using deep hierarchical feature learning, introduced by PointNet++ [9]. At each hierarchical encoder stage $b = 0, \dots, B - 1$, starting from the points $P_b \in \mathbb{R}^{3 \times u_b}$ and their features $F_b \in \mathbb{R}^{v_b \times u_b}$, a sampling algorithm selects a smaller number $v_{b+1} \leq v_b$ of representative points via their indices $\rho_b \subseteq \{1, \dots, v_b\}$, $|\rho_b| = v_{b+1}$.² Notations v_b and u_b are respectively the number of points and the feature length in stage b . The smaller, representative point cloud $P_{b+1} = \{p_i | i \in \rho_b\}$ is passed to the next stage.

To compute the next-stage features, each representative point $i \in \rho_b$ is associated with the set of indices of its K -nearest neighbors $N_b(i)$, in the “K-NN” layers in the turquoise encoder stages of Fig. 2. Denote by $F_{b,i}$ the i th row in feature matrix F_b . In the “Centering” layers of Fig. 2, the features $F_{b,i}$ of each point i are subtracted from the features $F_{b,j}$ of each neighbor $j \in N_b(i)$, obtaining centered features. All of these centered features are put together in a tensor $\tilde{F}_b \in \mathbb{R}^{v_{b+1} \times K \times u_b}$, which is then passed through two linear layers and a max-pooling layer over K , finally leading to the features $F_{b+1} \in \mathbb{R}^{v_{b+1} \times u_{b+1}}$ in the next stage. Note that to accurately estimate the value in RL, all points should remain represented in the centered features \tilde{F}_b of each stage, for which it is necessary that $Kv_{b+1} \geq v_b$.

Stage 0 is special in that representative point sampling is skipped, so that all points are selected; $P_1 = P_0$, $\rho_0 = \{1, \dots, v_0\}$, and $v_1 = v_0$. With this observation, the generic stage description above holds. The feature input F_0 of stage 0 is found by preprocessing the initial point cloud $P_0 \in \mathbb{R}^{v_0 \times 3}$ by two linear layers, so that $F_0 \in \mathbb{R}^{v_0 \times u_0}$, with $v_0 = n$ a constant upper bound on the size of the point cloud given the filtering settings; smaller point clouds are increased to this size by appending zero points.

The iterative application of all the stages $b = 0, \dots, B - 1$ leads to the overall hierarchical structure. Recall that Fig. 2 shows the hierarchical encoder in the green block labeled “Encoder”. Note that the total number of stages is B , and the output of the final stage is denoted F_B . As will be explained in Section 5, in both supervised and reinforcement learning experiments we achieved good performance by using a ratio of 0.25 between the number of points in two consecutive stages, $\frac{v_{b+1}}{v_b}$, which in practice provides upper bounds on (i) the size of the network and (ii) the number of stages B , since we expect stages with fewer than about 10 points to not be useful.

Next, two variants for the representative point sampling algorithm are discussed: farthest point sampling (FPS) [9], illustrated in the turquoise block labeled “FPS encoder stage” in Fig. 2, as well as a new technique: cosine similarity sampling (CSS), illustrated as “CSS encoder stage”. The objective in both techniques is to find ρ_b at any stage $b \geq 1$.

FPS creates the representative set ρ_b by repeatedly adding to it a point that is farthest from the previously selected points. Set ρ_b is initialized to $\{i\}$, where i is uniformly randomly selected from $\{1, \dots, v_b\}$. The distance between any point p and any set of points with indices ρ is defined as $d(p, \rho) = \min_{i \in \rho} \|p - p_i\|$, where $\|\cdot\|$ denotes the Euclidean norm. Then, ρ_b is iteratively updated using $\rho_b \leftarrow \rho_b \cup \{\operatorname{argmax}_{j \in \{1, \dots, v_b\} \setminus \rho_b} d(p_j, \rho_b)\}$ until the desired number v_{b+1} of representative points is reached. Ties in the argmax are broken to obtain a single point.

²In this section, by an abuse of notation we reuse the index of the point cloud P to mean stage (P_b) instead of time (P_t), with the understanding that the network works with the point cloud at the current time step at the input.

Note we keep updating the same set ρ_b , which is why we use an assignment instead of an equal sign.

CSS aims to reduce the computational cost of FPS. The cosine similarity [36] between every pair of feature vectors i and j in F_b is computed by taking the dot product of the vectors and dividing it by the product of their lengths: $c_{i,j} = \frac{F_{b,i} \cdot F_{b,j}^T}{\|F_{b,i}\| \cdot \|F_{b,j}\|}$. Then, points i are ranked by their cumulative similarity with all other points j : $C_i = \sum_{j=1}^{v_b} c_{i,j}$. Denote by C' the result of sorting vector C in ascending order. The representative set of indices ρ_b consists then of the first v_{b+1} indices i in C' . Intuitively, the points with the smallest cumulative similarities are the ones with the “most unique” features. The computational cost of CSS is smaller than that of FPS mainly because it is quadratic in v_b , compared to cubical in v_b like FPS.

Note that feature length remains constant across encoder stages in the default configuration (dashed paths in the FPS and CSS blocks of Fig. 2). This is a significant advantage compared to PCT [8], since at each stage that method performs feature concatenation (dotted paths in FPS and CSS), leading to doubling of the feature length.

4.2. Decoder

In the classical PCT classifier [8], the encoder is followed by a single-stage decoder consisting of a stack of four transformer layers applied to the output F_B of the last encoding stage $B - 1$, where the outputs of these transformer layers are concatenated. In our network, we also concatenate the transformers with F_B itself, and pass the result through a max pooling layer, two linear layers, and a dropout layer.

In addition to this classical single-stage decoder, inspired by the hierarchical encoder structure, we exploit the hierarchy of point clouds by adding earlier stages to the decoder as well. This connects the higher-level features in later stages with the finer-grained point clouds in earlier stages. To this end, at each stage $b = B - 1, \dots, 1$ in reverse order, we use another stack of four transformers similar to the one at stage B described above, but altering the first transformer in the stack as follows.³ The output Y_b of decoder stage $b + 1$ (which was applied to the point cloud of smaller size v_{b+1}), is fed into this first transformer in the stack at decoder stage b via key and value, whereas the query is computed based on the point features F_b at the input of encoder stage b . By making these choices for the key, query, and value, the dimensions are compatible with the earlier stage, and the reverse hierarchical computation can continue.

Fig. 2 details a decoder stage in the purple block on the right, emphasizing the query, key, and value signals entering the first transformer in the stack. This structure applies to both stage B and earlier stages, with the difference being which signal is fed into the query. The overall hierarchical decoder is the green “Decoder” block. Note that the alternate encoder stage E_1 is skipped in this default configuration; instead, the query input of D_1 is F_1 (dashed path).

4.3. PCT classification and segmentation cores

In our supervised learning experiments, we use simpler network components that are similar to PCT for classification and segmentation. We call these components “classification core” and “segmentation core”. Complete classification or segmentation networks can be obtained by appending a supervised learning tail to these cores. The two core types are illustrated in dotted-contour blue blocks in Fig. 2.

³Note that encoder stages are numbered $0, \dots, B - 1$ since their inputs contain F_0, \dots, F_{B-1} and P_0, \dots, P_{B-1} , whereas decoder stages are numbered $B, \dots, 1$ since their inputs are F_B, \dots, F_1 . The number of stages is B in both the encoder and decoder.

The *classification core* consists of the hierarchical encoder and only the last decoder stage. As mentioned in Section 4.1, the classification encoder concatenates the features of each set of nearest neighbors $\rho_b(i)$ with the features of their respective representative point i (dotted paths in the FPS and CSS blocks of Fig. 2). Nevertheless, to prevent iterative doubling of the feature size in the encoder, differently from standard PCT, in our architecture the feature vector is halved back to its initial length of 256 with the next linear layer. Moreover, unlike the default configuration, representative point selection is performed also in encoder stage E_0 . Finally, in the decoder stage, the last concatenation layer is skipped, so the signal follows the alternative, dotted path.

The *segmentation core* uses E_0 and the alternate, dotted-path encoder stage E_1 (to the right of E_0 in Fig. 2). Feature vectors are halved like in the classification core. Both E_0 and E_1 select the full set of points $P_0 = P_1 = P_2$, using feature concatenation. The output F_2 of E_1 is used for the query, key, and value in the decoder, symbolized in Fig. 2 by cutting through the decoder stage D_1 with the segmentation core outline, so that Y_1 is left outside of the block.

Finally, note that our architecture provides point features of length 256 as an output of the initial two linear layers in Fig. 2, whereas the length was 64 in the original PCT.

4.4. RL tail and normalization

The tail can be configured in several ways, in order to obtain the RL network or supervised-learning networks for classification or segmentation. This section discusses the RL tail, depicted at the top of the green “Tail” block in Fig. 2.

The RL tail is a dueling network [11] with a distributional output [10], both of which are components of Rainbow [35]. The distributional part, instead of directly representing expected values of the returns, represents the distribution of returns over a discrete number of 51 bins defined over a finite interval from V_{min} to V_{max} [10]. The dueling network structure divides the network into an advantage stream $A(s, a)$ and a value stream $V(s)$, and based on these streams it represents Q-values as $Q(s, a) = V(s) + \left(A(s, a) - \frac{1}{|A|} \sum_{a'} A(s, a') \right)$ [11].

We now turn to discussing normalization of the RL network. In deep learning in general, batch normalization has been shown to smooth the optimization landscape, which makes training easier, and to stabilize gradient estimation, which allows higher values of the learning rate to be used [37]. However, in deep RL, batch normalization is usually not applied, because the distribution of the data changes due to the changing policy over time, and so the mean features of Q-values are not very meaningful [38].

To address this issue, consider that the first layers of the network do not directly learn Q-values, but extract lower-level features of the data. These features are not changing fast, unlike later layers that must track the Q-values directly. Therefore, in our method batch normalization is applied on each linear layer in the encoder and decoder. In the tail, where the Q-values are learned, we apply instead spectral normalization [12] on the first fully connected value and advantage stream layers.

4.5. Supervised learning tail. Discriminator and its training

To obtain supervised learning networks, we append a supervised learning tail, depicted at the bottom of the green “Tail” block in Fig. 2. This tail is similar to that used in PCT [8], and consists of a first linear layer, a dropout layer, and two other linear layers that reduce the number of features to either the number of classes (in classification) or to the number of part types that each point may have (in segmentation). Then, for classification, the tail outputs a distribution over the classes for the entire point cloud, while for segmentation, it outputs a segmented point

cloud, in which each point is associated with a distribution over part types. Since we are dealing with supervised learning, batch normalization is applied as usual.

One important segmentation network is the discriminator, whose objective is to determine whether a point belongs to the background or to targets, on the basis of which the RL reward (1) is then computed. There are only two part types: background and targets. To train the network, rather than presenting arbitrary point clouds, the agent was run with a random policy and the resulting point clouds were collected and labeled with the ground-truth classes for each point. This procedure leads to training data that is representative for the RL regime.

Note that standard segmentation experiments will also be run, using the supervised learning tail configured for segmentation, in which we aim to improve on the PCT baseline. The name “segmentation” is used for this general setting, whereas “discriminator” is reserved for the specific network used to partition the point cloud into background and targets.

5. Experiments

We start with a first batch of experiments that search for a good network architecture in the setting of supervised learning, and compare with PCT in this setting. Due to space limits, we report these results in supplementary material, at [39].

For our main, RL experiments, we take the best-performing core from the supervised learning experiments: the 3-stage encoder-decoder with CSS, and append the RL tail to it, obtaining a network labeled EncDec-CSS+RL. Choosing 25% of the points for each subsequent stage yields the best performance, so we retain the entire set of $v_0 = 1620$ points in the first stage, then sample $v_1 = 400$ representative points in the second stage, and finally sample $v_2 = 100$ points in the third stage, using $K = 32$ neighbors (continuing with a fourth stage of 25 points would no longer provide meaningful information).

A discriminator network with structure EncDec-CSS+Segm partitions the point cloud into targets and background. To train it, we collected a ground-truth dataset of 400 episodes with 100 steps each. A second dataset of the same size is used for validation. The discriminator is 99.9% accurate on the validation data, so the prediction is wrong on average about 1.62 points.

The RL settings are as follows. The agents are trained for 10000 episodes of 100 steps each, using an ε -greedy exploration strategy [6]. Linear decay is used for ε , beginning at 1 (full exploration) and ending at 0 (exploitation only) after 3000 episodes. Training begins after 200 episodes. At each step, the network is trained with a batch size of 16 samples. For the distributional representation, the minimum value $V_{\min} = 0$ and the maximum value $V_{\max} = 530$; this maximum was experimentally established using the largest return achieved by the agent in preliminary experiments (515) plus a safety margin of 15. For the loss, the L2 norm is utilized and the gradient is clipped to a maximum value of 1.5, to prevent the otherwise large gradients caused by the deep network and estimation bias [11]. The Adam optimizer was applied with a learning rate of 10^{-6} and an ϵ parameter of 10^{-4} (note this ϵ is a different variable from the exploration probability ε). Other hyper-parameters are taken from [35].

We start by comparing our main network EncDec-CSS+RL to a Segm+RL baseline that performed better than networks with a classification core in preliminary RL experiments. Then, EncDec-CSS+RL is compared to a greedy baseline that selects at each step a next sensor pose maximizing the immediate reward.

EncDec-CSS+RL versus Segm+RL. Fig. 3 left shows that EncDec-CSS+RL exhibits faster learning and better final performance than Segm+RL. Note that despite the fact that EncDec-CSS+RL uses many more transformers than Segm+RL, the two networks are close in complexity,

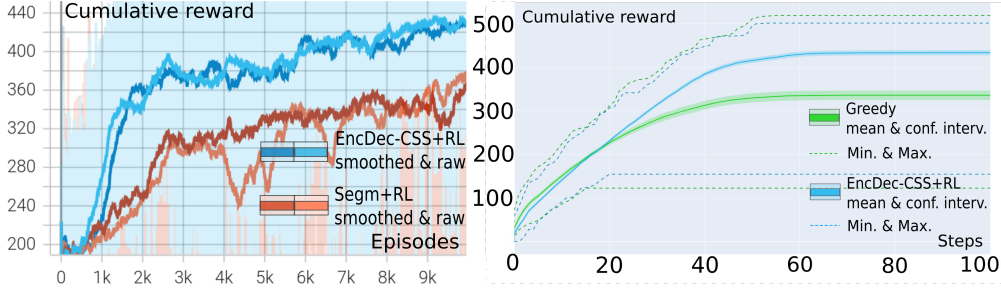


Fig. 3. Left: Learning curves with EncDec-CSS+RL (blue) versus Segm+RL (red). In both experiments, the darker and lighter shades of either color are used to differentiate the two seeds, and competing experiments of the same shade use the same seed. To make the graphs more readable, instead of plotting the raw cumulative reward C_τ at episode τ , we plot a smoothed version $\bar{C}_{\tau+1} = 0.99\bar{C}_\tau + (1 - 0.99)C_\tau$. Right: Performance with the final policy learned by EncDec-CSS+RL (blue) versus the greedy baseline (green), across 100 trials. The continuous line is the mean performance, the shaded region is the 95% confidence on the mean, and the dashed lines are the best and worst performances at each step.

Table 1. Complexity of the two architectures used in the final RL experiments. An NVIDIA RTX A40 GPU card was used.

Metric	Segm+RL	EncDec-CSS+RL
Params	144.2 M	161.4 M
GPU Memory	19240MiB	18399MiB
Network size	552.6MiB	617.5MiB

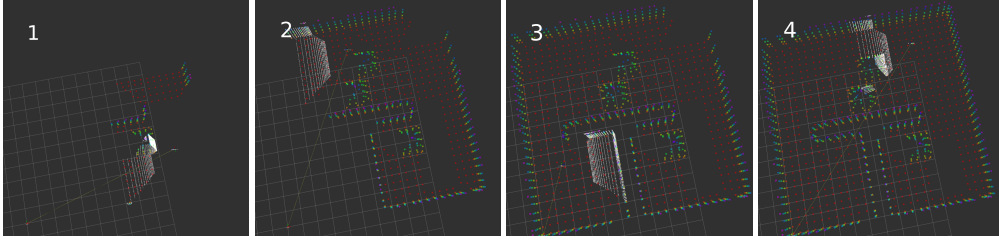


Fig. 4. A trajectory of the fully trained EncDec-CSS+RL agent. Top row from the left: 1) The first pyramid is found and its three sides are scanned. 2) The agent scans most of the scene, inspecting the wall to explore, and then finds two more pyramids and scans each one from three sides. 3) The agent enters and scans the third room. 4) The agent returns and goes to scan the back of the second pyramid, the corner and the remaining portion of the wall.

see Table 1. The reason for this is that the segmentation core seeks for K nearest neighbors of all points in the cloud in both stages E_0 and E_1 , without representative point sampling; see again Fig. 2 and Section 4.3.

Greedy versus RL. In our final experiment, we pit the best-performing EncDec-CSS+RL agent against the greedy baseline. Fig. 3 right shows that the RL agent outperforms the greedy strategy. Fig. 4 illustrates the behavior of the final policy learned by RL, which manages to solve the active search and coverage problem.

It is interesting to examine which points are selected by FPS and CSS, see Fig. 5. FPS selects points uniformly, which is not appropriate in this case since it wastes points on representing the

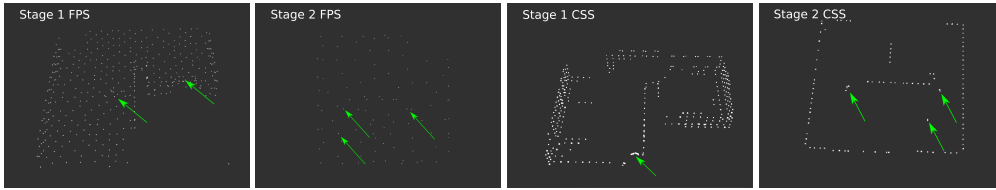


Fig. 5. Representative points selected by FPS (the two left images) and CSS (the two right images), with green arrows highlighting points on the target objects. In stage 1, 400 points are chosen after exploring roughly half of the environment. In stage 2, 100 points are chosen after exploring nearly the entire environment.

floor, and makes the targets difficult to distinguish even in stage 1; the entire environment loses structure in stage 2. In contrast, in stage 1 CSS keeps largely the edges of the environment plus some points on the pyramid it has discovered so far, whereas in stage 2, where the number of points is smaller, it keeps a 2D outline of the walls and the tops of the pyramids.

6. Conclusions and Future Work

We proposed a hierarchical encoder-decoder architecture for deep reinforcement learning on point clouds, and applied it to an active target search and coverage problem. The novel architecture worked better than a greedy baseline and than structures closer to the point cloud transformer, and in addition to RL it also led to improvements in supervised learning.

A first direction for future work is to apply the proposed method to real sensors mounted on a mobile robot, addressing the challenges involved in transferring results from simulation to the real world: increasing sample efficiency, handling large point clouds, noise, and task variability. A second, more fundamental direction is to adapt to RL in general the ways we discovered to train very deep RL networks in active search and coverage: batch normalization on the earlier, feature detection layers, and clipping the gradient more strongly. It is also interesting to apply CSS sampling to other point-cloud problems.

Acknowledgements. This work was been financially supported from SeaClear2.0, a project that received funding from the European Climate, Infrastructure and Environment Executive Agency (CINEA) under grant agreement No 101093822.

References

- [1] L. LIN and M. A. GOODRICH, *UAV intelligent path planning for wilderness search and rescue*, Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, St. Louis, MO, USA, 2009, pp. 709–714.
- [2] C. CAI, J. CHEN, Q. YAN, F. LIU and R. ZHOU, *A prior information-based coverage path planner for underwater search and rescue using Autonomous Underwater Vehicle (AUV) with side-scan sonar*, IET Radar, Sonar & Navigation **16**(7), 2022, pp. 1225–1239.
- [3] M. POPOVIC , T. VIDAL-CALLEJA, G. HITZ, J. J. CHUNG, I. SA, R. SIEGWART and J. NIETO, *An informative path planning framework for UAV-based terrain monitoring*, Autonomous Robots **44**(6–7), 2020, pp. 889–911.
- [4] O. KRISHNA, G. IRIE, X. WU, T. KAWANISHI and K. KASHINO, *Adaptive spotting: Deep reinforcement object search in 3D point clouds*, Proceedings of the Asian Conference on Computer Vision, online, 2020, pp. 481–497.
- [5] K. CHEN, R. OLDJA, N. SMOLYANSKIY, S. BIRCHFIELD, A. POPOV, D. WEHR, I. EDEN and J. PEHSERL, *MVLidarNet: Real-time multi-class scene understanding for autonomous driving us-*

- ing multiple views, Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, online, 2020, pp. 2288–2294.
- [6] R. S. SUTTON and A. G. BARTO, *Reinforcement Learning: An Introduction* (2nd edition), MIT Press, Cambridge, MA, 2018.
 - [7] L. TANG, Y. ZHAN, Z. CHEN, B. YU and D. TAO, *Contrastive boundary learning for point cloud segmentation*, Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, New Orleans, LA, USA, 2022, pp. 8489–8499.
 - [8] M.-H. GUO, J.-X. CAI, Z.-N. LIU, T.-J. MU, R. R. MARTIN and S.-M. HU, *PCT: Point cloud transformer*, Computational Visual Media 7(2), 2021, pp. 187–199.
 - [9] C. R. QI, L. YI, H. SU and L. J. GUIBAS, *Pointnet++: Deep hierarchical feature learning on point sets in a metric space*, Proceedings of the 30th International Conference on Neural Information Processing Systems, Long Beach, CA, USA, 2017, pp. 5099–5108.
 - [10] M. G. BELLEMARE, W. DABNEY and R. MUNOS, *A distributional perspective on reinforcement learning*, Proceedings of the International Conference on Machine Learning, Sydney, Australia, 2017, pp. 449–458.
 - [11] Z. WANG, T. SCHAUL, M. HESSEL, H. VAN HASSELT, M. LANCTOT and N. DE FREITAS, *Dueling network architectures for deep reinforcement learning*, Proceedings of the International Conference on Machine Learning, New York, NY, USA, 2016, pp. 1995–2003.
 - [12] F. GOGIANU, T. BERARIU, M. C. ROSCA, C. CLOPATH, L. BUSONIU and R. PASCANU, *Spectral normalisation for deep reinforcement learning: An optimisation perspective*, Proceedings of the International Conference on Machine Learning, online, 2021, pp. 3734–3744.
 - [13] K. OTA, D. K. JHA and A. KANEZAKI, *Training larger networks for deep reinforcement learning*, arXiv:2102.07920, 2021.
 - [14] P. HENDERSON, R. ISLAM, P. BACHMAN, J. PINEAU, D. PRECUP and D. MEGER, *Deep reinforcement learning that matters*, Proceedings of the AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2018, pp. 3412–3419.
 - [15] H. VAN HASSELT, Y. DORON, F. STRUB, M. HESSEL, N. SONNERAT and J. MODAYIL, *Deep reinforcement learning and the deadly triad*, arXiv:1812.02648, 2018.
 - [16] J. ACHIAM, E. KNIGHT and P. ABBEEL, *Towards characterizing divergence in deep Q-learning*, arXiv:1903.08894, 2019.
 - [17] S. SINHA, H. BHARADHWAJ, A. SRINIVAS and A. GARG, *D2RL: Deep Dense architectures in Reinforcement Learning*, arXiv:2010.09163, 2020.
 - [18] J. BAUER, K. BAUMLI, S. BAVEJA, F. BEHBAHANI, A. BHOOPCHAND, N. BRADLEY-SCHMIEG, M. CHANG, N. CLAY, A. COLLISTER et al., *Human-timescale adaptation in an open-ended task space*, arXiv:2301.07608, 2023.
 - [19] Z. WU, S. SONG, A. KHOSLA, F. YU, L. ZHANG, X. TANG and J. XIAO, *3D ShapeNets: A deep representation for volumetric shapes*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 2015, pp. 1912–1920.
 - [20] L. YI, V. G. KIM, D. CEYLAN, I.-C. SHEN, M. YAN, H. SU, C. LU, Q. HUANG, A. SHEFFER and L. GUIBAS, *A scalable active framework for region annotation in 3D shape collections*, ACM Transactions on Graphics 35(6), 2016, pp. 1–12.
 - [21] M. ROSYNSKI, A. POP and L. BUSONIU, *Active search and coverage using point-cloud reinforcement learning*, Proceedings of the 27th International Conference on System Theory, Control and Computing, Timisoara, Romania, 2023, pp. 289–296.
 - [22] C. LANDGRAF, B. MEESE, M. PABST, G. MARTIUS and M. F. HUBER, *A reinforcement learning approach to view planning for automated inspection tasks*, Sensors 21(6), 2021, paper 2030.

- [23] H. LI, Q. ZHANG and D. ZHAO, *Deep reinforcement learning-based automatic exploration for navigation in unknown environment*, IEEE Transactions on Neural Networks and Learning Systems **31**(6), 2019, pp. 2064–2076.
- [24] K. LOBOS-TSUNEKAWA and T. HARADA, *Point cloud based reinforcement learning for sim-to-real and partial observability in visual navigation*, Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, online, 2020, pp. 5871–5878.
- [25] Z. ZHANG, B.-S. HUA and S.-K. YEUNG, *Shellnet: Efficient point cloud convolutional neural networks using concentric shells statistics*, Proceedings of the IEEE/CVF International Conference on Computer Vision, Seoul, South Korea, 2019, pp. 1607–1616.
- [26] M. TIATOR, C. GEIGER and P. GRIMM, *Point cloud segmentation with deep reinforcement learning*, Proceedings of the European Conference on Artificial Intelligence, online, 2020, pp. 2768–2775.
- [27] E. NEZHADARYA, E. TAGHAVI, R. RAZANI, B. LIU and J. LUO, *Adaptive hierarchical down-sampling for point cloud classification*, Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, online, 2020, pp. 13645–13654.
- [28] D. DING, C. QIU, F. LIU and Z. PAN, *Point cloud upsampling via perturbation learning*, IEEE Transactions on Circuits and Systems for Video Technology **31**(12), 2021, pp. 4661–4672.
- [29] H. GOEL, L. J. LIPSCHITZ, S. AGARWAL, S. MANJANNA and V. KUMAR, *Reinforcement learning for agile active target sensing with a UAV*, arXiv:2212.08214, 2022.
- [30] M. QUIGLEY, K. CONLEY, B. GERKEY, J. FAUST, T. FOOTE, J. LEIBS, R. WHEELER, A. Y. NG et al., *ROS: An open-source Robot Operating System*, Proceedings of the ICRA Workshop on Open Source Software, Kobe, Japan, 2009, paper 5.
- [31] N. KOENIG and A. HOWARD, *Design and use paradigms for Gazebo, an open-source multi-robot simulator*, Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Sendai, Japan, 2004, pp. 2149–2154.
- [32] Q.-Y. ZHOU, J. PARK and V. KOLTUN, *Open3D: A modern library for 3D data processing*, arXiv:1801.09847, 2018.
- [33] M. L. PUTERMAN, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, John Wiley & Sons, Hoboken, NJ, 2014.
- [34] M. T. J. SPAAN, *Partially observable Markov decision processes*, in Reinforcement Learning: State-of-the-Art, M. WIERING and M. VAN OTTERLO, Eds., Springer, Berlin, Heidelberg, pp. 387–414, 2012.
- [35] M. HESSEL, J. MODAYIL, H. VAN HASSELT, T. SCHAUL, G. OSTROVSKI, W. DABNEY, D. HORGAN, B. PIOT, M. AZAR and D. SILVER, *Rainbow: Combining improvements in deep reinforcement learning*, Proceedings of the AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2018, pp. 3433–3440.
- [36] T. MIKOLOV, K. CHEN, G. CORRADO and J. DEAN, *Efficient estimation of word representations in vector space*, arXiv:1301.3781, 2013.
- [37] S. SANTURKAR, D. TSIPRAS, A. ILYAS and A. MADRY, *How does batch normalization help optimization?*, Proceedings of the 32nd International Conference on Neural Information Processing Systems, Montreal, Canada, 2018, pp. 8315–8326.
- [38] A. BHATT, M. ARGUS, A. AMIRANASHVILI and T. BROX, *Crossnorm: Normalization for off-policy TD reinforcement learning*, Proceedings of the International Conference on Machine Learning, Long Beach, CA, USA, 2019, pp. 653–663.
- [39] M. ROSYNSKI and L. BUSONI, *Supplementary Material for Encoder-Decoder Reinforcement Learning for Active Search and Coverage in Point Clouds*, *Romanian Journal of Information Science and Technology*, 2026. [Online]. http://busoni.net/files/papers/encdec_suppl.pdf.