Neural Path Planning With Multi-Scale Feature Fusion Networks

Xiang Jin, Wei Lan, and Xin Chang

School of Naval Architecture and Ocean Engineering, Dalian Maritime University, Dalian 116026, China

Corresponding author: Xin Chang (xin.chang@dlmu.edu.cn)

This work was supported in part by the National Key Research and Development Program of China under Grant 2016YFC0301500.

ABSTRACT Path planning is critical for planetary rovers that perform observation and exploration missions in unknown and dangerous environment. And due to the communication delay, it is difficult for the planet rover to receive instructions from Earth in time to guide its own movement. In this work, we present a novel neural network-based algorithm to solve the global path planning problem for planetary rovers. Inspired by feature pyramid networks used for object detection, we construct a deep neural network model, termed the Pyramid Path Planning Network (P3N), which has a well-designed backbone that efficiently learns a global feature representation of the environment, and a feature pyramid branch that adaptively fuses multi-scale features from different levels to generate the local feature representation with rich semantic information. The P3N learns environmental dynamics from terrain images of planetary surface taken by satellites, without using additional elevation information to construct an explicit environmental model in advance, and can perform path planning policy after end-to-end training. We evaluate the effectiveness of the proposed method on synthetic grid maps and a realistic data set constructed from the lunar terrain images. Experimental results demonstrate that our P3N has higher prediction accuracy and faster computation speed compared to the baseline methods, and generalize better in large-scale environments.

INDEX TERMS Deep learning, feature pyramid network, global path planning, multi-scale feature fusion, planetary rover.

I. INTRODUCTION

Path planning is a very important technology for mobile robots, and can be divided into global path planning and local path planning [1], [2]. Global path planning refers to finding an optimal collision-free path from the start to the end state under the condition that the environmental information is known or predictable. Local path planning means that when the environmental information is known, the mobile robot obtains more information by actively exploring the environment and tries to find a feasible path to reach the target through repeated attempts.

As a special class of mobile robots, planet rovers usually perform observation and exploration missions on other planets in the solar system, such as the Moon or Mars [3]. These planets are so far away from Earth that communication delays make it difficult for ground commanders to monitor and control the rovers’ movements. Although the rover is usually equipped with cameras and other sensors to help it perceive its surroundings, when the rover needs to explore a target outside its field of view, given the complex terrain on the planet and limited movement capability of rovers, blind exploration behavior may put it in danger and also substantially increase energy consumption.

With recent technological advances, we have been able to easily obtain high-resolution terrain images of other planets taken by satellites. For the path planning problem of the rover, an effective solution is to pre-plan a globally optimal path from the rover’s location to the target area based on the terrain data obtained from satellite images [4], [5], and the rover then follow this path and fine-tune the path based on the local environmental information obtained during the journey. Although this path may not be optimal due to the resolution of satellite images, it is necessary to ensure the safety of rover and the success of exploration mission.
Traditional path planning methods first need to integrate various environmental information and build up an environmental model that can be used by planning algorithms, such as configuration space (C-space), visibility graph, grid map, and Voronoi diagram, etc., and then use these models to find the optimal path [6]. We can broadly classify these planning algorithms into the following categories. The first is search-based algorithms [7], such as Dijkstra and A*, whose advantage is that if optimal paths exist, then through exploring the whole environment step by step, these algorithms are guaranteed to find optimal solutions. The second class of sampling-based algorithms [8], such as Probabilistic Road Map (PRM) and Rapidly-exploring Random Tree (RRT) algorithms, find feasible paths by randomly exploring the environment space, which are more efficient than search-based algorithms when facing high-dimensional and large-scale environments, but only guarantee that the solution found is asymptotically optimal. The third is heuristic algorithms [2], such as genetic algorithm (GA) and particle swarm optimization (PSO) algorithm, which are more efficient when facing partially known or unknown environments. They generate a set of local optimal solutions at each iteration, and then iteratively improve them according to different fitness functions and optimization policies.

With the rapid development of deep learning techniques in recent years, researchers have started to focus more on learning-based planning algorithms [9]. Learning-based algorithms build suitable neural network architectures by taking as input raw data about environmental information, such as data from satellites and radars, or data collected by sensors carried by robots, without relying on environmental models, and then train these networks by supervised learning or reinforcement learning to output feasible paths that meet specified requirements. MPNet [10] takes 3D point cloud data of the environment space as input and implements collision-free path planning with motion constraints considered. MPNet can also be combined with traditional planning algorithms to speed up the training of neural networks and improve the quality of planned paths. In addition, many novel deep learning techniques have been applied to solve motion planning problems, for example, OracleNet uses RNN [11], TDPP-NET adopts Imitation learning method [12], Pathgan even applies Generative Adversarial Neworks (GANs) [13]. However, these methods are usually trained and predicted in a certain environment instance, and it is difficult to effectively transfer the learned policy to other similar environments, lacking the ability to solve a class of problems through planning computations.

Reinforcement learning (RL) is a trial-and-error approach to train a policy that allows an agent to make decisions that maximize future cumulative rewards, rather than just focusing on the reward that can be obtained in the present, which in essence requires an algorithm with some ability to plan [14]. If we describe the path planning task as a Markov decision process (MDP), we can use RL algorithms to solve the path planning problem [15]. Tamar et al. [16] approximated the value iteration (VI) algorithm as a convolution architecture embedded in a neural network to obtain a value iteration network (VIN) with “planning capability”, which is trained to effectively generalize to new environments similar to the training set and outperforms standard convolutional neural network (CNN) architectures on navigation and path planning tasks. Although the VIN does not explicitly model the environment, it combines the advantages of both model-free and model-based RL algorithms, enabling both explicit planning computation and end-to-end training and inference, while not requiring a reward function to be set. However, limited by the convolutional architecture it uses, the VIN can only accept structurally regular data (e.g., images) as input, and the task-related MDP must be fixed and known. Several recent research works have further extended this value iteration-based planning algorithm, such as GVIN [17] and XLVIN [18] which apply VIN to graph-structured data using graph operators instead of the convolutions. The VPN [19] defines a maximum propagation algorithm and also approximates it as a convolution and max-pooling operations, achieving better results in dynamic environments compared to VIN. The UVIN [20] introduces a clustering algorithm and successfully extends the VIN to MDP-variable environments.

While conventional CNNs, such as ResNet [21], progressively compress the spatial resolution of feature maps through multi-layer convolution operations to extract global features for planning tasks, the VIN achieves higher accuracy and better generalization capability by extracting local features using an explicit planning module derived from the value iteration algorithm. It is worth noting that when we talk about local features here, we mean that the global information is aggregated to every position in the environmental space by the convolution operation. The Gated path planning network (GPPN) [22] proved that the explicit value iteration process is not necessary and can be replaced by an implicit LSTM unit, and what really matters is actually the extraction of local feature representation of environments. The Dual-branch convolutional neural network (DB-CNN) [4] built on the architecture of ResNet by extracting local features with a parallel convolution branch and fusing the global and local features for planning, achieving a better performance beyond. In general, the DB-CNN gives a more generic approach to path planning tasks by using CNNs, which has higher accuracy while significantly improving computational efficiency. However, since the space resolution of feature maps in the local feature extraction branch remains the same, it is difficult for the DB-CNN to adopt a deeper network architecture to balance efficiency and accuracy, which actually limits the expressiveness of the model, yet a deeper network will further increase the model parameters and easily cause over-fitting.

In this work, we continue to follow the idea of DB-CNN by constructing a neural network to extract both global and local features of the input, and then use the fused features for path planning. Inspired by the feature pyramid network (FPN) [23] commonly used in the field of object detection, we connect the two branches of DB-CNN and use the output of the global
feature branch as the input of the local feature branch, which is equivalent to doubling the depth of the local feature extractor with almost no increase in parameters, which is conducive to obtaining a better local feature representation. We name this novel neural network architecture as the Pyramid Path Planning Network, or P3N for short. In addition, unlike the DB-CNN which only utilizes the output of the last layer of local feature branch, the proposed P3N simultaneously extracts multi-scale features from each stage of the local feature branch, and then adaptively fuses these features into a better representation by a learnable weighting operation. We conducted extensive experiments on both grid maps and satellite terrain image datasets. Results show that our P3N has faster computation speed and higher prediction accuracy compared to the VIN and DB-CNN, and generalizes better on large-scale environments thanks to the deeper local feature extraction network.

The main contributions of this work are summarized as follows:

- We design a network architecture based on the feature pyramid network that can better extract global and local features from the input.
- By introducing the novel architecture, we propose the Pyramid Path Planning Network which can adaptively fuse multi-scale features and effectively learn to plan from natural images.
- Experimental results on grid-world maps and terrain images show that the P3N significantly outperforms the baseline methods with lower computational cost and has better generalization on large-scale domains.

The paper is organized as follows. Section 2 provides some preliminaries of this work. Section 3 describes the proposed P3N method for global path planning. Experimental results and discussion are presented in section 4. And the conclusion is given in Section 5.

II. PRELIMINARIES
A. VALUE ITERATION NETWORK
1) VALUE ITERATION ALGORITHM
The MDP corresponds to a tuple \([S, A, P, R]\), where \(S\) is the set of all possible states of the agent, \(A\) is the set of all legal actions of state \(s\), \(P\) is the probability distribution of transferring from the current state \(s\) to the next state \(s'\), and \(R \subseteq \mathbb{R}\) is the set of reward received from the environment during the state transition. In RL, the goal of agent is to maximize the cumulative reward obtained from the current time \(t\)

\[
G(t) = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k} \gamma^k R_{t+k+1},
\]

where \(\gamma \in [0, 1]\) is the discount factor to balance the importance of the rewards received at the current and later moments.

In order to obtain the maximum cumulative reward, the agent needs to learn a policy \(\pi\) by repeated attempts.

The policy is a mapping from state \(s\) to the probability \(\pi(a | s)\) of all possible actions, i.e., if the agent executes the policy \(\pi\) at moment \(t\), then \(\pi(a | s)\) is the probability that \(A_t = a\) when \(S_t = s\). The state-value function of the policy \(\pi\) can be written as

\[
V^\pi(s) = E_{\pi}[G(t)|S_t = s] = E_{\pi} [\sum_{k} \gamma^k R_{t+k+1}] | S_t = s,\]

which denotes the expected return obtained by executing the policy \(\pi\) from state \(s\) at all times. Similarly, the action-value function for the policy \(\pi\) can be written as

\[
Q^\pi(s, a) = E_{\pi} \left[ \sum_{k} \gamma^k R_{t+k+1} \right] | S_t = s, A_t = a,
\]

which denotes the expected return obtained after choosing action \(a\) from state \(s\), given that the policy \(\pi\) is executed all the time.

For the optimal policy \(\pi^*\), there is \(V^{\pi^*}(s) = V^*(s)\), where \(V^*(s)\) is the optimal value function, and

\[
V^*(s) = \max_{a \in A} Q^*(s, a)
\]

\[
= \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma V^*(s')],
\]

which is the optimal Bellman equation for the policy \(\pi^*\). For a finite MDP problem, there exists a unique optimal solution to (4) independent of the policy. For any policy \(\pi\), \(V^\pi(s)\) will converge to \(V^*(s)\) with probability 1 by iteratively solving (4), which is known as the value iteration algorithm.

2) APPROXIMATE VALUE ITERATION MODULE
We consider the path planning problem on a two-dimensional grid map and denote by \(M\) the MDP of the domain with respect to the planning policy \(\pi\). We also assume that there exists an unknown \(\bar{M}\) where the optimal policy \(\pi^*\) contains some useful information about \(\pi^*\) in \(M\). If it is possible to solve \(\bar{M}\) and use the solution of \(\bar{M}\) as part of \(\pi\), then \(\pi\) can automatically learn and use \(\pi^*\) to solve \(M\). To establish a connection between \(\bar{M}\) and \(M\), let \(\bar{r} = f_{\bar{r}}(\phi(s))\), and \(\bar{p} = f_p(\phi(s))\), where \(\phi(s)\) denotes the observation of state \(s\).

In VIN, the approximate value iteration module can be written as

\[
\bar{v}^k = \max_{a \in A} \bar{q}^k, \bar{q}^k = h \left( \bar{r}, \bar{v}^{k-1} \right),
\]

where \(k \in [1, K]\) is the number of iterations, and the selection of \(K\) is dependent on the map size. According to (4), we have

\[
h (\bar{r}, \bar{v}) = \bar{r} + \gamma \sum_{\bar{s}, \bar{a}} \bar{p} (\bar{s}, \bar{a}) \bar{v}.
\]

In the 2D grid-world environment, the state transition is a one-step movement from one cell to the surrounding 8 cells, and the transition probability \(p\) has the property of local connectivity. So (6) can be approximated as a convolution operation, with

\[
\bar{q}^k_{a, i, j} = \sum_{(i', j')} W^{[r, v]}_{a, i, j \rightarrow (i', j')} \left[ \bar{r}_{i', j'}, \bar{v}^{k-1}_{i', j'} \right].
\]
\[
\bar{v}_{i,j}^k = \max_a \bar{q}_{a,i,j}^k
\]

where the tuple \((i', j') \in \mathcal{N}(i, j)\) is the neighbor of the agent’s position \((i, j)\), and \(W\) the parameters of the convolution kernel. And this is the approximate value iteration module in VIN.

### B. DB CNN

Before DB-CNN proposed, most deep CNNs for path planning tasks only had a backbone network for extracting environmental features and estimating value functions. However, due to the local connectivity structure of convolution operation, more convolution layers have to be stacked to obtain better feature representation, which both increase the computational cost and make the models difficult to train. The DB-CNN built two parallel branches to extract both global and local features, which allowed the network to be shallower and more computationally efficient and with better model performance.

The architecture of DB-CNN consists of three parts, which are pre-processing stage, branch one for global feature representation and branch two for local feature representation. The pre-processing stage usually contains multiple convolution layers, which serve to filter the noise in the input and decide whether to compress the feature maps according to the task properties. If needed, pooling layers can be added after the convolutions to compress the spatial resolution of feature maps.

The branch one includes a convolution layer, several residual modules and two fully connected layers. Both the convolution layer and residual modules are followed by max-pooling layers to progressively reduce the spatial dimension of feature maps. The residual modules include two convolution layers with shared parameters, a skip connection, and two ReLU activation functions [24], which are located after the first convolutional layer and before the final output, respectively. The stacked residual modules can increase the depth of the model and improve the training accuracy, while the shared parameters can not only reduce the computational effort, but also prevent the model from over-fitting. The output of last residual module is expanded into a one-dimensional form to feed the fully-connected layers, and after two nonlinear transformations, the global features of the environment are obtained.

The branch two consists of two convolution layers and several residual modules. By following the VIN, the branch two keeps the spatial resolution of feature maps constant throughout, so the computational cost is higher compared to the branch one. After trading off the model accuracy and computational efficiency, the DB-CNN chooses a channel dimension of 20, which preserves more input information without dramatically affecting the computational speed.

Finally, the DB-CNN concatenates global and local features in the channel dimension, and then feeds them into one or more fully-connected layers, outputting Q values for each executable action in the current state.

In general, the DB-CNN replaces the computationally tedious value iteration module in the VIN with a local feature extractor, and learns global features with another regular CNN branch, giving a more universal framework for solving path planning problems, which has a higher prediction accuracy while significantly reducing computational cost. On the one hand, the DB-CNN, compared to the VIN, uses a relatively shallow network architecture, which actually limits its performance in large-scale environments. On the other hand, if we blindly increase the depth of two feature extraction networks, it is easy to cause over-fitting of the model, and thus we are caught in a dilemma.

In the next section, we will give the design of a novel path planning network that can effectively increase the model depth without significantly increasing the computational cost, and improve the model performance at the meantime.

### C. FEATURE PYRAMID NETWORK

With the rapid development of deep learning techniques, researchers have designed various deep neural network architectures for image recognition, detection, and segmentation. In particular, the object detection task means to detect and localize multiple objects at different scales from a single image, thus requiring learning to obtain a multi-scale feature representation of the image.

FIGURE 1 shows several schemes for learning from the multi-scale representation of images for object detection.
Among them, the first scheme down-samples the original image to construct the image pyramid at different scales, and then detects objects individually on each of the image scales independently. These image pyramids are scale-independent, and contains rich semantic information, thus providing better performance with high computational cost. The second is inspired by the deep neural networks applied to image recognition, using the well-designed CNNs to automatically learn features at different scales from the input image and then make predictions with the small-scale features for faster detection. Although the high-level feature maps are semantically strong, they do not contain the accurate location information of small objects, which harms the final prediction accuracy. The third reuses the inherent, pyramidal feature hierarchy computed by CNNs and extracts multi-scale features of different spatial resolutions with marginal computational cost, but introduces large semantic gaps caused by different layers. The high-resolution feature maps may harm the representation capacity for object detection due to lack of adequate semantic information.

The FPN, shown in FIGURE 2, makes full use of the pyramidal hierarchy structure of CNN by constructing a new top-down pathway in addition to the bottom-up backbone, and fully integrates the high-level semantic features with low resolution and the low-level positional features with high resolution through the lateral connections between the two pathways. The FPN has the property of full convolution, which can receive arbitrary size images as input and then output the corresponding size feature maps. Moreover, the construction of the feature pyramid pathway is independent of the architecture of the backbone, so they can be designed separately.

The bottom-up backbone, usually derived from modern CNNs for image recognition, consists of stacked multi-stage convolutional modules that gradually reduce the spatial dimensions of the feature maps by pooling or convolution operations with stride size 2. The top-down pathway is built step-by-step via the fusion operation shown in FIGURE 2 (b). The feature maps with resolution \( m \times m \times c_1 \) from higher levels are up-sampled to \( 2m \times 2m \times c_1 \), then the feature maps with resolution \( 2m \times 2m \times c_2 \) from the backbone are convolved to adjust the channel dimension to \( c_1 \), and finally the two features are first summed and then convolved (in order to eliminate the confounding effect of up-sampling) to generate feature maps that contain both rich semantic and location information. The FPN takes full advantage of the multi-stage pyramidal structure of the backbone to gradually integrates semantic information into high-resolution feature maps from top to bottom, which is conducive to the detection of small objects.

FIGURE 2. (a) The architecture of feature pyramid network (FPN). (b) The up-sample module in the top-down pathway of FPN.

### III. METHODS

#### A. THE IMPROVED DB-CNN

The VIN approximates the value iteration algorithm in RL with a convolution operator, and performs explicit planning computation to implement an end-to-end path planning algorithm using high-dimensional images as input. Most of the improved methods built upon VIN retain the explicit planning module, which differ in their implementation but are generally based on iterative computation. We refer to this class of methods as VI-based planning algorithms. Experiments in [16] show that although CNNs also have good single-step prediction accuracy in path planning tasks, they lack long-term planning capability, performing poorly on tasks that require multi-step decision-making.

The DB-CNN builds two parallel network branches. The branch one, consistent with the conventional CNN, learns the low-dimensional global feature representation of the high-dimensional input by compressing the feature map dimensions step by step. The branch two, following the VIN, extracts the local features related to the current position from the environmental information. Without the explicit iterative computation, the branch two of DB-CNN is still the conventional CNN architecture, except that the spatial dimension of feature maps is always kept the same in the forward computation process to facilitate the extraction of local features by the attention mechanism. Although the quality of local features obtained by this way may not be as good as that of VIN, after aggregating with the global features of branch one, the DB-CNN achieves a better performance on the path planning task and has a lower computational effort. We refer to this class of methods as CNN-based planning algorithms.

In general, the VI-based methods accomplishes path planning with only local features through explicit planning
computation, while the CNN-based methods, relying on the conventional CNN architecture, achieve better performance with a simpler network architecture. However, the CNN-based methods are more prone to over-fitting and perform much worse on the test set. Although DB-CNN uses parameter sharing in the convolution module, it still has much more parameters than the VIN, making it more difficult to train.

In addition, when facing large-scale environments, VI-based methods have full-convolution property and can receive any size input without modifying the model architecture, only the number of iterations of the VI module needs to be adjusted accordingly. However, this also leads to a rapid growth in the computational effort as the input size increases, and the time required to train a model from scratch that can perform path planning on large-size maps is almost unacceptable. The CNN-based methods, when receiving large-size maps as input, requires stacking more convolution layers to obtain a larger effective receptive field. Although this is much smaller in terms of the increase of computation, it is still costly to achieve planning on large-size maps because the feature map resolution of the local feature extractor always remains the same.

In this section, we follow the idea of DB-CNN by building a network to extract both global and local features, and then use the fused features for path planning. Since the branch one of DB-CNN has already obtained a global representation about the input information, we can actually continue to use this global feature as the input of branch two, which is equivalent to directly doubling the network depth with almost no increase in computation, which is beneficial to obtain a better representation of local features.

FIGURE 3 shows a simple improvement to the DB-CNN, where we keep the structure of branch one the same and take the output of the last convolution module as the input of branch 2, which can significantly increase the network depth of branch two to obtain better local feature representation with marginal extra cost. The improved model no longer contains two parallel branches, but is similar to the FPN.

**B. PYRAMID PATH PLANNING NETWORK**

We can see that the branch one of the improved DB-CNN is similar to the backbone part of the FPN, while the branch two can be regarded as the feature pyramids. And with the lateral connections established between the two branches, we can obtain a Pyramid-based Path Planning Network, abbreviated as the P3N.

FIGURE 4 illustrates the overall architecture of P3N, where the bottom-up pathway is used to extract the global features, and the top-down pyramid pathway fuses features across a large of scales to construct a multi-scale feature representation. Unlike the DB-CNN which only gets local features at the high-resolution level, the P3N extracts features at each stage, and then fuses them to obtain the semantically-strong local features.

We assume that features from different stages should have different contributions to the final local feature, so we consider assigning different weights to these features and learning them by end-to-end training. We denote the weights by \( w_i \) and the feature vector corresponding to the \( i \)th stage of the feature pyramid branch by \( I_{p_i} \), then the final local features can be expressed as

\[
I_{\text{local}} = \sum w_i \cdot I_{p_i},
\]

where \( w_i \), as a learnable scalar parameter, can take any value, which may lead to training instability.
Therefore we further constrain the sum of all weight parameters to be 1, with

$$I_{\text{local}} = \sum_j \frac{e^{w_j}}{\sum_j e^{w_j}} \cdot I_{p_i}, \quad (9)$$

where we apply the softmax operation to all weight parameters and constrain their values to range from 0 to 1. However, the softmax function will bring a large computational cost and may affect the running speed of the model. Tan et al. [25] proposed a fast normalization method that can replace the softmax operation, so we can rewrite (9) as

$$I_{\text{local}} = \sum_j \frac{|w_j|}{\epsilon + \sum_j |w_j|} \cdot I_{p_i}, \quad (10)$$

where $\epsilon$ is a small positive number to avoid the numerical instability associated with a denominator value close to 0. The weight parameter is also constrained to be between 0 and 1, but this method is much more efficient than the softmax.

Furthermore, to obtain a better global feature representation, we reconstruct the backbone of P3N on the basis of DB-CNN, drawing on a variety of modern CNN design paradigms. FIGURE 5 illustrates the backbone of P3N used to take in the 128 × 128 resolution images, which consists of three parts. The first part is the pre-processing stage, which includes a 7 × 7 convolution layer with a stride size of 1 to keep the resolution of input images constant. Since object detection is usually performed on a relatively lower-resolution feature maps, the FPN reduces the feature map resolution in the pre-processing stage to decrease the computational effort in the subsequent steps. Whereas in the path planning task, we need to keep at least one full-resolution feature map, with the aim of being able to accurately compute the Q value of agent at any position on the map.

The second part is the core of the backbone, which involves four stages, and at the end of each stage the feature map resolution is reduced by a 2 × 2 convolution with a stride of 2. We built the Convblock with reference to MobileNetV3 [26], composed of an Inverted Residual and Linear Bottleneck module. The first 1 × 1 convolution layer expands the number of channels of the input features. The 5 × 5 depthwise separable convolution (DWConv) [27] layer has a larger receptive field compared to the 3 × 3 conventional convolution, with a better balance between computation and performance. And the last 1 × 1 convolution operation is used to reduce the dimension of output features. It is worth noting that we only apply the pre-BN and ReLU to the 5 × 5 convolution, because ConvNeXt [28] points out that fewer normalization layers and activation functions are beneficial sometimes. We refer to the design of most multi-stage networks and set the ratio of layers in each stage to 1:1:3:1.

The last part is the post-processing stage, where we first transforms the 2D feature maps into a 1D feature vectors through an adaptive average-pooling operation, and then introduce more nonlinearities through two fully connected layers while adjusting the dimension of output global features.

IV. EXPERIMENTS AND DISCUSSION

In this section, we empirically evaluate the proposed P3N architecture when used as a policy representation for global path planning task. We compare the performance differences between the P3N and two baseline methods (VIN and DB-CNN) on two datasets, grid maps and terrain images. For a fairer comparison, our evaluation metrics are consistent with those in [16], including prediction loss, path planning success rate, and trajectory difference. The prediction loss refers to the error rate of the algorithm’s single-step prediction. The planning success rate is the probability that the methods can find a collision-free path given the specified start and target states. The trajectory difference means the length difference between the successfully planned trajectory and the optimal.

We also discuss the details of model implementation and the impact of training strategies on model performance. All models and experimental code are implemented based on the PyTorch [29] framework and will be open sourced when the work accepted.

A. PATH PLANNING IN GRID-WORLD DOMAIN

Our first experimental scenario is a synthetic grid-map domain as shown in FIGURE 6, where the start and target state, as well as the positions of obstacles, are generated randomly. Each obstacle occupies one grid, and the proportion of grids with obstacles to all grids is fixed in order to facilitate control of the planning difficulty. In the framework of RL, the agent can only move one step at a time to the surrounding eight grids, and the goal is to find a collision-free shortest trajectory from the start position to the goal.
1) PERFORMANCE IN SMALL-SCALE DOMAIN

Let’s start with the grid maps of $28 \times 28$ size. The training set contains 10,000 randomly generated maps, where the proportion of obstacles to the entire environmental space is always kept at 50%. For each map instance, we first randomly specify the start and target positions, then generate the expert trajectory using the A* algorithm, and finally discretize it into a series of single-step state-action pairs as training samples. All models receive the environmental map and the agent’s position as input and output the Q value of the optional actions in the current state. We ask the agent to choose the action with the highest Q value and move to the next position until it reaches the target.

We train each model for 30 epochs using the RMSprop optimizer, with an initial learning rate set to $4e-3$ and a mini-batch size of 256, and reduce the learning rate by $10 \times$ in the last 6th and 2nd epochs, respectively [30]. At the end of each training epoch, we immediately test the prediction loss of all models on a test set containing another 5000 maps. At the end of all 30 training epochs, we evaluate the planning success rate and trajectory difference of all models on the test set. We train each model five times with different random seeds and then average the results. For the VIN, we set the number of iterations of the planning module to $1.5 \times$ the map size, i.e., for a $28 \times 28$ grid map, the number of iterations is set to 42.

FIGURE 7 shows the variation of training error and test error for all models during the training process. The performance of VIN on the training and test sets is much worse than the other two methods, and the training is smoother. The proposed P3N, although slightly under-performing DB-CNN in the early stage, gradually outperforms the latter as the training proceeds.

TABLE 1 shows more experimental results. Our P3N outperforms the baseline methods in all evaluation metrics, beats the VIN by 7% in planning success rate to 99.04%, and the training time of the model is only half of that of VIN. The P3N achieves 1.36$\times$ faster computation speed than the DB-CNN, although it is only slightly ahead of the latter in three metrics. Considering the small size of the domain in this experiment, the shallow network is sufficient to extract useful environmental features, and thus our method does not achieve a significant advantage.

2) GENERALIZATION IN LARGE-SCALE DOMAIN

We further increase the size of grid maps to test the performance of these methods. When the map size grows, the planning module of VIN needs to perform more iterations to ensure that the reward signal can be efficiently propagated across the whole planning space, which can significantly increase the computational cost. And more unconstrained iterations can also cause training instability, resulting in dramatic degradation of the model performance. Thanks to the fact that the size of feature maps in the VI-based models is always consistent with the input data, Jin et al. [31] proposed a two-stage training strategy that can effectively improve the performance of VI-based methods on large-scale domains while substantially reducing the computational effort. In this work, however, we aim to explore the performance differences between models due to different architectures, not to examine the impact of various tricks on model performance. Thus, we only use some of the most basic training strategies in the next experiments.

We continue to construct two datasets of 10,000 grid maps each, with map sizes of $64 \times 64$ and $128 \times 128$. Since training of VIN is unstable and sensitive to the choice of training strategies and hyper-parameters. The performance of the two CNN-based methods is relatively similar, and the training is smoother. The proposed P3N, although slightly under-performing DB-CNN in the early stage, gradually outperforms the latter as the training proceeds.
increasing the map size results in a larger memory usage, we adjust the mini-batch size to 64 and 32, respectively, and also reduce the learning rate accordingly, and increase the number of training epochs to 60 and 120 to allow models to be fully trained.

TABLE 2 shows the performance of all models on larger-scale domains. Our P3N still obtains the best performance and further widens the performance gap with the other methods, with a planning success rate of almost 1.8× the VIN on the 64×64 maps and 2.7× on the 128×128 maps. As the domain size increases, the DB-CNN gradually exposes the shortcoming of not having enough depth of local feature extractor. The planning success rate on the 64×64 maps is 8% lower than that of P3N, and the gap is further expanded to 15% on the 128×128 maps. In conclusion, although path planning on grid maps is a relatively simple task, our proposed FPN-based path planning method is still more competitive than the baseline. The P3N has higher planning success rate and lower trajectory difference on large-size grid maps, while the computational speed is rather faster due to the novel architectural design.

B. ROVERS NAVIGATION

The benefits of NN-based algorithms, compared to traditional path planning algorithms, lie in the ability to identify useful environmental information from natural images and then execute the planning policy end-to-end. So we further verify whether the proposed P3N can still perform better than the baseline methods on terrain images.

We construct the second experimental scenario with the orthomosaic (overhead terrain image) created from images provided by the Lunar Reconnaissance Orbiter Camera (LROC) Narrow Angle Camera (NAC) with a resolution of 0.5m per pixel, the corresponding digital elevation model (DEM) generated from four LROC NAC stereo image pairs, and the grid map where regions with an elevation angle of 20 degrees or more as obstacles.

We want planet rovers to be able to start from any position, actively avoid those obstacles, and then safely reach the designated target area. We would like to emphasize that the terrain image itself does not contain any elevation information, and its corresponding digital elevation model (DEM), as shown in FIGURE 8, are generated from the LROC NAC stereo images.

We crop the orthomosaic randomly into small-size images that do not overlap each other to construct the test scenarios. For comparison, we continue to construct three datasets containing 10,000 terrain images with resolutions of 32×32, 64×64, and 128×128, respectively. we determine the slope at each location in the environment by calculating the gradient of adjacent pixels using the DEMs corresponding to the terrain images, and then mark the locations with slope greater than 20 degrees as obstacles, thus converting the terrain images into grid maps. We specify a random set of start and goal positions on the grid map, and then use the A* algorithm to generate the optimal path. It is worth mentioning that the elevation data is only used to help generate demonstration trajectories, and all models can only accept terrain images as input and then infer decision-friendly environmental information from them by end-to-end learning.

We randomly select 6/7 samples from each data set for training and the rest for test. The model training strategies and the selection of other hyper-parameters remain the same as in the previous section. FIGURE 9 shows a representative set of test examples of what the trajectories predicted by our method look like. We can qualitatively conclude that solving such a path planning problem should be very difficult due to the differences between terrain images and DEMs.

TABLE 3 shows the performance of all models on the lunar terrain images. The VIN performs better on small-size terrain images than on grid maps, with a planning success rate close to 6% higher. This is because on the one hand, the VIN has enough network depth to extract valid information as obstacles. We want planet rovers to be able to start from any position, actively avoid those obstacles, and then safely reach the designated target area. We would like to emphasize that the terrain image itself does not contain any elevation information, and its corresponding digital elevation model (DEM), as shown in FIGURE 8, are generated from the LROC NAC stereo images.
from small-size images, and on the other hand, obstacles in the lunar environment take up a much smaller proportion of the entire planning space compared to grid maps, as can be seen in the rightmost column of FIGURE 9. Therefore, even if the single-step prediction given by the VIN is not optimal, the probability that the agent can finally reach the target is still high, and of course the trajectory difference will be relatively larger. The performance of DB-CNN decreases significantly on large-size terrain images, and its planning success rate in the 128 × 128 lunar domains has dropped by more than 8% compared with that on grid maps, which again illustrates it is difficult for the DB-CNN to learn effective environmental representation on large-scale domains due to the lack of enough network depth in its local feature extraction branch.

In contrast, the P3N has a well-designed backbone that can better learn the global representation of environmental information. The FPN-based local feature extractor can effectively fuse semantic and location information from different levels, and then generate context-rich local representation by means of adaptive feature aggregation. Thanks to the powerful representation network, the P3N can easily infer the elevation information of the environment from terrain images and accurately distinguish obstacles and non-obstacles, thus achieving similar performance on terrain images as on grid maps. Also, the P3N outperforms the baseline methods in all evaluation metrics, with the lead being greater as the image size increases.

V. CONCLUSION

In this work, we propose a effective neural network-based computational framework to solve the global path planning problem for planet rovers. We design a novel neural network architecture based on feature pyramid networks, named the Pyramidal Path Planning network (P3N), which take the terrain image of the planet surface and the position coordinates of the rover as input, and output a safe and energy-efficient path to the specified target area through implicit planning computation. Our P3N has a well-designed backbone that efficiently learns the global feature representation of the environment, and a feature pyramid branch that adaptively fuses multi-scale features from different levels to generate a strong local feature representation. While previous studies generally used two independent network branches to extract global and local features separately, we use the multi-scale global features learned by the backbone as the input to the local feature extractor, and obtain a fine-grained representation of the environment containing rich semantic and location information.

We compare our P3N with two baseline methods, including the VIN and DB-CNN, on path planning tasks of grid maps and a data set generated from lunar terrain images. The experimental results show that the P3N achieves the best performance in all evaluation metrics, and the computation speed is 86% and 36% faster than the two baseline methods on 28 × 28 grid maps, respectively. And our method has better generalization performance on the large-scale environment, with a path planning success rate of 81.8% when training from scratch on the 128 × 128 lunar domain, outperforming the VIN by 52% and the DB-CNN by 23.6% with less computational cost.

ACKNOWLEDGMENT

Xiang Jin would like to thank J. Zhang for his passionate help in data set preparation.

REFERENCES

[1] P. Raja and S. Pugazhenthi, “Optimal path planning of mobile robots: A review,” Int. J. Phys. Sci., vol. 7, no. 9, pp. 1314–1320, Feb. 2012.
[2] M. N. A. Wahab, S. Nefti-Meziani, and A. Atyabi, “A comparative review on mobile robot path planning: Classical or meta-heuristic methods?” Ann. Rev. Control, vol. 50, pp. 233–252, Sep. 2020.
[3] M. Saitoh, M. Osuki, S. Wakabayashi, T. Hoshino, and T. Hashimoto, “The right path: Comprehensive path planning for lunar exploration rovers,” IEEE Robot. Autom. Mag., vol. 22, no. 1, pp. 22–33, Mar. 2015.
[4] J. Zhang, Y. Xia, and G. Shen, “A novel learning-based global path planning algorithm for planetary rover,” Neurocomputing, vol. 361, pp. 69–76, Oct. 2019.
[5] M. Pfueger, A. Agha, and G. S. Sukhatme, “Rover-IRL: Inverse reinforcement learning with soft value iteration networks for planetary rover path planning,” IEEE Robot. Autom. Lett., vol. 4, no. 2, pp. 1387–1394, Apr. 2019.
[6] T. Lozano-Perez, “Spatial planning: A configuration space approach,” in Autonomous Robot Vehicles. New York, NY, USA: Springer, 1990, pp. 259–271.
[7] R. Yonetani, T. Tanai, M. Barekatain, M. Nishimura, and A. Kanazaki, “Path planning using neural A∗ search,” in Proc. ICML, 2021, pp. 12029–12039.
[8] A.-I. Toma, H.-Y. Hsueh, H. A. Jaafar, R. Murai, P. H. J. Kelly, and S. Saeedi, “PathBench: A benchmarking platform for classical and learned path planning algorithms,” in Proc. 18th Conf. Robots Vis. (CVR), May 2021, pp. 79–86.
[9] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. Cambridge, MA, USA: MIT Press, 2016.
[10] A. H. Qureshi, Y. Miao, A. Simeonov, and M. C. Yip, “Motion planning networks: Bridging the gap between learning-based and classical motion planners,” IEEE Trans. Robot., vol. 37, no. 1, pp. 48–66, Aug. 2021.
[11] M. J. Bency, A. H. Qureshi, and M. C. Yip, “Neural path planning: Fixed time, near-optimal path generation via Oracle imitation,” in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS), Macau, China, Nov. 2019, pp. 3965–3972.
[12] K. Wu, M. A. Esfahani, S. Yuan, and H. Wang, “TDPP-Net: Achieving three-dimensional path planning via a deep neural network architecture,” Neurocomputing, vol. 357, pp. 151–162, Sep. 2019.
[13] M. Assens, X. G.-I. Nieto, K. McGuinness, and N. E. O’Connor, “PathGAN: Visual scanpath prediction with generative adversarial networks,” in Proc. ECCV, Munich, Germany, 2018, pp. 1–18.
[14] S. Richard and B. Andrew, Reinforcement Learning: An Introduction. Cambridge, MA, USA: MIT Press, 2018.
[15] Y. F. Chen, M. Liu, M. Everett, and J. P. How, “Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning,” in Proc. IEEE Int. Conf. Robot. Autom. (ICRA), Singapore, May 2017, pp. 285–292.
[16] A. Tamar, Y. Wu, G. Thomas, S. Levine, and P. Abbeel, “Value iteration networks,” in Proc. Adv. Neural Inf. Process. Syst., Barcelona, Spain, vol. 29, 2016, pp. 2154–2162.
[17] S. Niu, S. Chen, H. Guo, C. Targonski, M. C. Smith, and J. Kovaevi, “Generalized value iteration networks: Life beyond layers,” in Proc. AAAI, New Orleans, LA, USA, 2018, pp. 6246-6253.
[18] A. Deac, P. Velickovic, O. Milinkovic, P.-L. Bacon, J. Tang, and M. Nikolic, “XLVIN: Executed latent value iteration nets,” 2020, arXiv:2010.13146.

[19] N. Nardelli, P. Kohli, G. Synnaeve, P. Torr, Z. Lin, and N. Usunier, “Value propagation networks,” in Proc. ICLR, New Orleans, LA, USA, 2019, pp. 1–13.

[20] L. Zhang, X. Li, S. Chen, H. Zang, J. Huang, and M. Wang, “Universal value iteration networks: When spatially-invariant is not universal,” in Proc. AAAI, New York, NY, USA, 2020, pp. 6778–6785.

[21] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Las Vegas, NA, USA, Jun. 2016, pp. 770–778.

[22] L. Lee, E. Parisotto, D. Chaplot, E. Xing, and R. Salakhutdinov, “Gated path planning networks,” in Proc. ICML, Stockholm, Sweden, 2018, pp. 4597–4608.

[23] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, “Feature pyramid networks for object detection,” in Proc. CVPR, Honolulu, HI, USA, Jul. 2017, pp. 933–944.

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” Commun. ACM, vol. 60, no. 6, pp. 84–90, May 2017, doi: 10.1145/3065386.

[25] M. Tan and Q. V. Le, “EfficientNet: Rethinking model scaling for convolutional neural networks,” in Proc. 36th Int. Conf. Mach. Learn., Long Beach, CA, USA, 2019, pp. 6105–6114.

[26] A. Howard, M. Sandler, B. Chen, W. Wang, L.-C. Chen, M. Tan, G. Chu, V. Vasudevan, Y. Zhu, R. Pang, H. Adam, and Q. Le, “Searching for MobileNetV3,” in Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV), Seoul, South Korea, Oct. 2019, pp. 1314–1324.

[27] F. Chollet, “Xception: Deep learning with depthwise separable convolutions,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Honolulu, HI, USA, Jul. 2017, pp. 1800–1807.

[28] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, “A ConvNet for the 2020s,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), New Orleans, LA, USA, Jun. 2022, pp. 11976–11986.

[29] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, and A. Desmaison, “PyTorch: An imperative style, high-performance deep learning library,” in Proc. Adv. NeuIPS, Vancouver, BC, Canada, vol. 32, 2019, pp. 8026–8037.

[30] K. He, R. Girshick, and P. Dollár, “Rethinking ImageNet pre-training,” in Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV), Oct. 2019, pp. 4918–4927.

[31] X. Jin, W. Lan, T. Wang, and P. Yu, “Value iteration networks with double estimator for planetary rover path planning,” Sensors, vol. 21, no. 24, p. 8418, Dec. 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/24/8418

Xiang Jin received the B.S. degree in naval architecture and ocean engineering, in 2015, and the M.S. degree in design and construction of naval architecture and ocean structure from Dalian Maritime University, Dalian, China, in 2018, where he is currently pursuing the Ph.D. degree in marine electrical engineering.

His research interests include the motion planning and formation control of the autonomous underwater vehicles.

Wei Lan received the B.S. degree in marine engineering, in 2014, and the M.S. degree in naval architecture and ocean engineering from Dalian Maritime University, Dalian, China, in 2016, where he is currently pursuing the Ph.D. degree in marine engineering.

His research interests include the research of small underwater vehicle, and formation motion and control composed of underwater vehicles.

Xin Chang received the B.S. degree in ship engineering and the M.S. and Ph.D. degrees in design and construction of naval architecture and ocean structure from Harbin Engineering University, Harbin, China, in 2000, 2003, and 2005, respectively, and the Ph.D. degree from the Harbin Institute of Technology, in 2009.

His research interests include the research of ship propulsion performance and energy saving, ship overall performance evaluation technology, and ship life cycle health management.