Integrating Symmetry into Differentiable Planning

Anonymous Author(s)
Affiliation
Address
email

Abstract

We study how group symmetry helps improve data efficiency and generalization for end-to-end differentiable planning algorithms, specifically on 2D robotic path planning problems: navigation and manipulation. We first formalize the idea from Value Iteration Networks (VINs) on using convolutional networks for path planning, because it avoids explicitly constructing equivalence classes and enable end-to-end planning. We then show that value iteration can always be represented as some convolutional form for (2D) path planning, and name the resulting paradigm Symmetric Planner (SymPlan). In implementation, we use steerable convolution networks to incorporate symmetry. Our algorithms on navigation and manipulation, with given or learned maps, improve training efficiency and generalization performance by large margins over non-equivariant counterparts, VIN and GPPN.

1 Introduction

Model-based planning usually struggles in complex problems, and planning in more structured and abstract space is a major solution [1, 2, 3, 4]. Symmetry is ubiquitous in learning and decision-making problems and can effectively reduce search space for planning. However, existing planning algorithms using symmetry assumes perfect dynamics knowledge, needs to explicitly build equivalence classes, or does not consider problem structure [5, 4, 6, 7, 8]. For example, if we use A* on path planning, we cannot specify visually obvious rotation symmetry in Figure 1, and need to detect in manually from the provided dynamics model. This would be even more challenging to detect in differentiable planning.

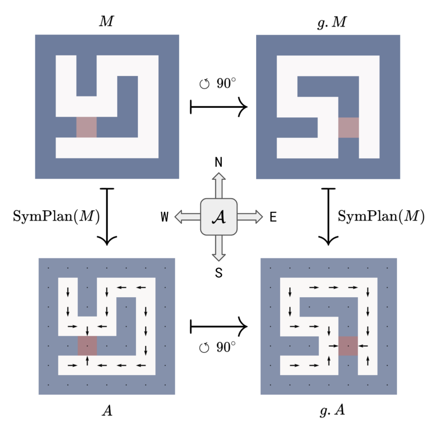

Figure 1: The *path planning problem* has symmetry, so we study how to *exploit* its symmetry in (differentiable) *planning*. Red dots are goal. The optimal actions $A = \texttt{SymPlan}(M)$ (bottom row) for the maps M (top row) are guaranteed to be equivariant $\texttt{SymPlan}(g.M) = g.\texttt{SymPlan}(M)$ under ⟲ rotations for (2D) path planning. For example, the action in the NW corner of A is the same as the action in the SW corner of $g.A$, after also rotating the arrow ⟲ 90°.

Nevertheless, symmetry in model-free deep reinforcement learning (RL) has been studied recently [9, 10]. However, it can only effectively handle pixel-level "element-wise" symmetry, such as flipping or rotating state and action together. Despite of this, a critical benefit of model-free RL agents that enables great asymptotic performance is its end-to-end differentiability. This motivates us to combine the spirit of both: *is it possible to design an end-to-end differentiable planning algorithm that makes use of symmetry in environments?*

In this work, we propose to (1) avoid explicitly building equivalence classes for symmetric states while (2) realize planning in an end-to-end differentiable manner. We are motivated by work in the equivariant network and geometric deep learning community [11, 12, 13, 14, 15, 16], which treat an RGB image as a mapping $\mathbb{Z}^2 \to \mathbb{R}^3$ and apply equivariant convolutions between feature maps. It satisfies our desiderata: equivariant networks on images do not need to explicitly consider "symmetric pixels" while guarantee symmetry properties. Based on the intuition, we propose a framework, *Symmetric Planning* (SymPlan), to understand a straightforward but general problem, path planning, as operating like images, called steerable feature fields [14, 16]. We focus on 2D grid and prove that value iteration (VI) for 2D path planning is equivariant under the *isometries* of $\mathbb{Z}^2$: translations, rotations, and reflections, and further show that VI here is a special form of steerable convolution network [14]. This provides us a foundation to equip Value Iteration Network (VIN, [17]) with steerable convolution. We implement the equivariant steerable version of VIN, named SymVIN, and use a variant, GPPN [18], to build SymGPPN. Both SymPlan methods achieve great improvement on training efficiency and generalization performance to unseen random maps, which showcases the advantage of exploiting symmetry from environments for planning.

Our contributions are as follows:

- Understand the inherent symmetry in path planning problems (on 2D grids), formulate value iteration in as steerable convolution network, and connect both to incorporate symmetry into VI.
- Based on the formulation, implement equivariant steerable version of VIN and GPPN.
- Show significant improvement in training and generalization on 2D navigation and manipulation.

2 Related work

Planning with symmetries (Symmetric Planning). Symmetries widely exist in various domains, and have been exploited in classic planning algorithms as well as model checking [5, 4, 6, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30]. Zinkevich and Balch [7] show the invariance of value function for an MDP with symmetry. Narayanamurthy and Ravindran [8] prove that finding exact symmetry in MDPs is graph isomorphism complete. However, they are based on classic planning algorithms, such as A*, and have a fundamental issue with exploitation of symmetries: they explicitly construct equivalence classes of symmetric states, which explicitly represents states and introduces symmetry breaking. Therefore, they are intractable (NP-hard) in maintaining symmetries in trajectory rollout and forward search (for large state space and symmetry group) and incompatible with differentiable pipelines for representation learning, hindering it from wider applications in RL and robotics.

State abstraction for detecting symmetries. Coarsest state abstraction aggregates all symmetric states into equivalence classes, studied in MDP homomorphisms and bisimulation [3, 31, 2]. However, they usually require *perfect* MDP dynamics knowledge and do not scale up well, because of the complexity in maintaining abstraction mappings (homomorphisms) and abstracted MDPs. van der Pol et al. [32] integrate symmetry into model-free RL based on MDP homomorphisms [3], which avoids the challenges in handling symmetry in forward search. Park et al. [33] learn equivariant transition models, but do not consider planning. Additionally, the formulation in commonly defined symmetric MDPs [3, 9, 6, 7] is different from our symmetry formulation for path planning, since they study "element-wise" symmetry for every state-action pairs and require reward to be symmetric. Our reward is not symmetric and we mainly study symmetry of the underlying domain (2D grid), as further discussed in Section F.2.

Symmetries and equivariance in deep learning. Equivariant neural networks are used to incorporate symmetry in supervised learning for different domains (e.g. grid and sphere), symmetry groups (e.g. translations and rotations), and group representation on feature spaces [34]. Cohen and Welling [15] introduce G-CNNs, followed by Steerable CNNs [14] which generalizes from scalar feature fields to vector fields with induced representations. Kondor and Trivedi [13], Cohen et al. [12] study theory on the relation between equivariant maps and convolutions. Weiler and Cesa [16] propose to solve kernel constraints under arbitrary representations for $E(2)$ and its subgroups by decomposing into irreducible representations, named $E(2)$-CNN.

Differentiable planning. Our pipeline is based on learning to plan in a neural network in a differentiable manner. Value iteration network (VIN) [35] is a representative work that performs value iteration using convolution on lattice grids, and has been further extended [36, 18, 37, 38]. Other than using convolution network, works on integrating learning and planning into differentiable networks include [39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49]. In the theoretical side, Grimm et al. [50, 51] propose to understand the differentiable planning algorithms from value equivalence perspective, while Gehring et al. [52] study its gradient dynamics.

3 Preliminaries

Markov decision processes. We model the path planning problems as Markov decision processes (MDP) [1]. An MDP is a 5-tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$, with state space $\mathcal{S}$, action space $\mathcal{A}$, transition probability function $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}_+$, reward function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and discount factor $\gamma \in [0, 1]$. Value functions $V : \mathcal{S} \to \mathbb{R}$ and $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ represent expected future returns [1].

Symmetry groups and equivarance. A symmetry *group* is defined as a set G together with a binary composition map satisfying the axioms of associativity, identity, and inverse. A (left) *group action* of G on a set $\mathcal{X}$ is defined as the mapping $(g, x) \mapsto g.x$ which is compatible with composition. Given a function $f : \mathcal{X} \to \mathcal{Y}$ and G acting on $\mathcal{X}$ and $\mathcal{Y}$, then f is *G-equivariant* if it commutes with group actions: $g.f(x) = f(g.x), \forall g \in G, \forall x \in \mathcal{X}$. In the special case the action on $\mathcal{Y}$ is trivial $g.y = y$, then $f(x) = f(g.x)$ holds, and we say f is *G-invariant*.

Group representations. We mainly use two groups: dihedral group D_4 and cyclic group C_4. The cyclic group of 4 elements is $C_4 = \langle r \mid r^4 = 1 \rangle$, a symmetry group of rotating a square. The dihedral group $D_4 = \langle r, s \mid r^4 = s^2 = (sr)^2 = 1 \rangle$ includes both rotations r and reflections s, and has size $|D_4| = 8$. A group representation defines how a group action transforms a vector space $G \times S \to S$. These groups have three types of representations of our interest: *trivial*, *regular*, and *quotient* representations, see [16]. The *trivial representation* ρ_{triv} maps each $g \in G$ to 1 and hence fixes all $s \in S$. The *regular representation* ρ_{reg} of C_4 group sends each $g \in C_4$ to a 4×4 permutation matrix that cyclically permutes a 4-element vector, such as a one-hot 4-direction action. The regular representation of D_4 maps each element to an 8×8 permutation matrix which does not act on 4-direction actions, which requires the *quotient representations* (quotienting out sr^2 reflection part) and forming a 4×4 permutation matrix. It is worth mentioning the *standard representation* of the cyclic groups, which are 2×2 rotation matrices, only used for visualization (Figure 2 middle).

Steerable feature fields and Steerable CNNs. The concept of *feature fields* is used in (equivariant) CNNs [11, 12, 13, 14, 15, 16]. The pixels of an 2D RGB image $x : \mathbb{Z}^2 \to \mathbb{R}^3$ on a domain $\Omega = \mathbb{Z}^2$ is a feature field. In steerable CNNs for 2D grid, features are formed as *steerable feature fields* $f : \mathbb{Z}^2 \to \mathbb{R}^C$ that associate a C-dimensional feature vector $f(x) \in \mathbb{R}^C$ to each element on a base space, such as $\mathbb{Z}^2$. Defined like this, we know how to transform a steerable feature field and also the feature field after applying CNN on it, using some group [14]. The type of CNNs that operates on steerable feature fields is called Steerable CNN [14], which is equivariant to groups including *translations* as subgroup $(\mathbb{Z}^2, +)$, extending [15]. It needs to satisfy a *kernel steerability* constraint, where the $\mathbb{R}^2$ and $\mathbb{Z}^2$ cases are considered in [16]. We consider the 2D grid as our domain $\Omega = \mathcal{S} = \mathbb{Z}^2$ and use $G = p4m$ group as the running example. The group $p4m = (\mathbb{Z}^2, +) \rtimes D_4$ (wallpaper group) is semi-direct product of discrete translation group $\mathbb{Z}^2$ and dihedral group D_4, see [15, 14]. We visualize the *transformation law* of $p4m$ on a feature field on $\Omega = \mathbb{Z}^2$ in **Figure 2 (Middle)**, usually referred as *induced representation* [14, 16]. Additional details in Section G.

Planning as convolution. The core component behind dynamic programming (DP) based algorithms in planning or reinforcement learning is *Bellman (optimality) equation* [53, 1]: $V(s) = \max_a R(s, a) + \gamma \sum_{s'} P(s'|s, a)V(s')$. Value Iteration (VI) iteratively applies Bellman operator and converges to fixed points [1, 53]. The key component of our interest is $\sum_{s'} P(s'|s, a)V(s')$ that aggregates values $V(s')$ from adjacent states by expectation using transition probabilities, here referred as **expected value operation**. Tamar et al. [17] propose Value Iteration Network (VIN) that uses convolution (networks) for planning, as an instance of differentiable planning, by recursively applying planar convolutions and max-pooling over feature spaces on 2D grid $\mathbb{Z}^2$.

4 Symmetric Planning Framework

This section formulates the notion of Symmetric Planning (SymPlan). We expand the understanding of path planning in neural networks by planning as convolution on steerable feature fields (*steerable planning*). We use that to build *steerable value iteration* and show it is equivariant.

4.1 Steerable Planning: planning on steerable feature fields

We start the discussion based on Value Iteration Networks (VINs, [17]) and use a running example of planning on the 2D grid $\mathbb{Z}^2$. We aim to understand (1) how VIN-style networks embed planning

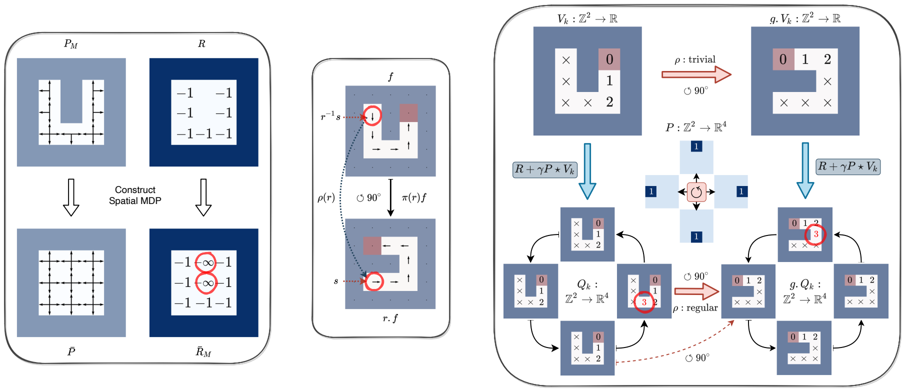

Figure 2: **(Left)** Construction of spatial MDPs from path planning problems, enabling G-invariant transition. **(Middle)** The group acts on a feature field (MDP actions). We need to find the element in the original field by $f(r^{-1}x)$, and also rotate the arrow by $\rho(r)$, where $r \in D_4$. We represent one-hot actions as arrows (vector field, using ρ_{std}) for visualization. **(Right)** Equivariance of $V \mapsto Q$ in Bellman operator on feature fields, under $\circlearrowleft 90^\circ \in C_4$ rotation, which visually explains Theorem 4.1. The example simulates VI for one step (see red circles; minus signs omitted) with true transition P using $\circlearrowleft$ N-W-S-E actions. The Q-value field are for 4 actions and can be viewed as either $\mathbb{Z}^2 \to \mathbb{R}^4$ ([14, 16]) or $\mathbb{Z}^2 \rtimes C_4 \to \mathbb{R}$ (on $p4$ group, [15]). See Appendix H for more details.

and how its idea generalizes, (2) how is symmetry structure defined in path planning and how could it be injected into such planning networks.

Constructing G-invariant transition: spatial MDP. Intuitively, the embedded MDP in a VIN is different from the original path planning problem, since (planar) convolutions are translation equivariant but there are different obstacles in different regions.

We found the key insight in VINs is that it implicitly uses an MDP that has translation equivariance. The core idea behind the construction is that it converts *obstacles* (encoded in transition probability P, by *blocking*) into "*traps*" (encoded in reward $\bar{R}$, by $-\infty$ *reward*). This allows to use planar convolutions with translation equivariance, and also enables use to further use steerable convolutions.

The demonstration of the idea is shown in **Figure 2 (Left)**. We call it *spatial MDP*, with different transition and reward function $\bar{\mathcal{M}} = \langle \mathcal{S}, \mathcal{A}, \bar{P}, \bar{R}_m, \gamma \rangle$, which converts the "complexity" in the transition function P in $\mathcal{M}$ to the reward function $\bar{R}_m$ in $\bar{\mathcal{M}}$. The state and action space are kept the same: state $\mathcal{S} = \mathbb{Z}^2$ and action $\mathcal{A} \subset \mathbb{Z}^2$ to move Δs in four directions in a 2D grid. We provide the detailed construction of the spatial MDP in Appendix H.1.

Steerable features fields. We generalize the idea from VIN, by viewing functions (in RL and planning) as *steerable feature fields*, motivated by [11, 12, 14]. This is analogous to pixels on images $\Omega \to [255]^3$, and would allow us to apply convolution on it. The state value function is expressed as a field $V : \mathcal{S} \to \mathbb{R}$, while the Q-value function needs a field with $|\mathcal{A}|$ channels: $Q : \mathcal{S} \to \mathbb{R}^{|\mathcal{A}|}$. Similarly, a policy field[1] has probability logits of selecting $|\mathcal{A}|$ actions. For the transition probability $P(s'|s,a)$, we can use action to index it as $P^a(s'|s)$, similarly for reward $R^a(s)$. The next section will show that we can convert the transition function to field and even convolutional filter. Additional details are in Appendix H.

4.2 Symmetric Planning: symmetry by equivariance

The seemingly slight change in the construction of spatial MDPs brings important symmetry structure. The general idea in exploiting symmetry in path planning is to use *equivariance* to avoid explicitly constructing equivalence classes of symmetric states. To this end, we construct value iteration over steerable feature fields, and show it is *equivariant* for path planning.

In VIN, the convolution is over 2D grid $\mathbb{Z}^2$, which is symmetric under D_4 (rotations and reflections). However, we also know that VIN is already equivariant under translations. To consider all symmetries, as in [14, 16], we understand the group $p4m = G = B \rtimes H$ as constructed by a *base space*

[1] We avoid the symbol π for policy since it is used for induced representation in [14, 16].

$B = G/H = (\mathbb{Z}^2, +)$ and a *fiber* group $H = D_4$, which is a *stabilizer subgroup* that fixes the origin $\mathbf{0} \in \mathbb{Z}^2$. We could then formally study such symmetry in the spatial MDP, since we construct it to ensure that the transition probability function in $\bar{\mathcal{M}}$ is G-invariant. Specifically, we can uniquely decompose any $g \in \mathbb{Z}^2 \rtimes D_4$ as $t \in \mathbb{Z}^2$ and $r \in D_4$ (and translations act "trivially" on action), so

$$\bar{P}(s' \mid s, a) = \bar{P}(g.s' \mid g.s, g.a) \equiv \bar{P}\left((tr).s' \mid (tr).s, r.a\right), \quad \forall g = tr \in \mathbb{Z}^2 \rtimes D_4, \forall s, a, s'. \quad (1)$$

Expected value operator as steerable convolution. The equivariance property can be shown step-by-step: (1) *expected value operation*, (2) *Bellman operator*, and (3) full *value iteration*. First, we use G-invariance to prove that the expected value operator $\sum_{s'} P(s'|s,a)V(s')$ is equivariant.

Theorem 4.1. *If transition is G-invariant, the expected value operator E over $\mathbb{Z}^2$ is G-equivariant.*

The proof is in Appendix I.1 and visual understanding is in Figure 2 middle. However, this provides intuition but is inadequate since we do not know: (1) how to implement it with CNNs, (2) how to use multiple feature channels like VINs, since it shows for scalar-valued transition probability and value function (corresponding to trivial representation). To this end, we next prove that we can implement value iteration using steerable convolution with general steerable kernels.

Theorem 4.2. *If transition is G-invariant, there exists a (one-argument, isotropic) matrix-valued steerable kernel $P^a(s - s')$ (for every action), such that the expected value operator can be written as a steerable convolution and is G-equivariant:*

$$E^a[V] = P^a \star V, \quad [g.[P^a \star V]](s) = [P^{g.a} \star [g.V]](s), \quad \forall s \in \mathbb{Z}^2, \forall g \in \mathbb{Z}^2 \rtimes D_4. \quad (2)$$

The full derivation is provided in Appendix I. We write the transition probability as $P^a(s, s')$, and we show it only depends on *state difference* $P^a(s - s')$ (or *one-argument* kernel [12]) using G-invariance, which is the key step to show it is some *convolution*. Note that we use one kernel P^a for each action (four directions), and when the group acts on E, it also acts on the action $P^{g.a}$ (and state, so technically acting on $\mathcal{S} \times \mathcal{A}$). Additionally, if the steerable kernel also satisfies the D_4-*steerability constraint* [16, 54], the steerable convolution is *equivariant* under $p4m = \mathbb{Z}^2 \rtimes D_4$. We can then extend VINs from $\mathbb{Z}^2$ translation equivariance to $p4m$-equivariance (translations, rotations, reflections). The derivation follows the existing work on steerable CNNs [15, 14, 16, 12], while this is our goal: to justify the close connection between path planning and steerable convolutions.

Steerable Bellman operator and value iteration. We can now represent all operations in Bellman (optimality) operator on steerable feature fields over $\mathbb{Z}^2$ (or *steerable Bellman operator*) as follows:

$$V_{k+1}(s) = \max_a R^a(s) + \gamma \times \left[P^a \star V_k\right](s), \quad (3)$$

where $V, R^a, \bar{P}^a$ are steerable feature fields over $\mathbb{Z}^2$. As for the operations, $\max_a$ is (max) pooling (over group channel), $+, \times$ are point-wise operations, and $\star$ is convolution. As the second step, the main idea is to prove every operation in Bellman (optimality) operator on steerable fields is equivariant, including the nonlinear $\max_a$ operator and $+, \times$. Then, iteratively applying Bellman operator forms value iteration and is also equivariant, as shown below and proved in Appendix I.4.

Proposition 4.3. *For a spatial MDP with G-invariant transition, the optimal value function can be found through G-steerable value iteration.*

Remark. Our framework generalizes the idea behind VINs and enables us to understand its applicability and restrictions. More importantly, this allows us to integrate symmetry but avoid explicitly building equivalence classes and enables planning with symmetry in end-to-end fashion. We emphasize that the *symmetry in spatial MDPs* is different from *symmetric MDPs* [7, 3, 9], since our reward function is *not* G-invariant (if not conditioning on reward). Although we focus on $\mathbb{Z}^2$, we can generalize to path planning on higher-dimensional or even continuous Euclidean spaces (like $\mathbb{R}^3$ space [54] or spatial graphs in $\mathbb{R}^3$ [55]), and use *equivariant operations* on *steerable feature fields* (such as steerable convolutions, pooling, and point-wise non-linearities) from steerable CNNs. We refer the readers to Appendix H and to [15, 14, 56, 16] for more details.

5 Symmetric Planning in Practice

In this section, we discuss how to achieve Symmetric Planning on 2D grids with $\mathrm{E}(2)$-steerable CNNs [16]. We focus on implementing symmetric version of value iteration, SymVIN, and generalize the methodology to make a symmetric version of a popular follow-up of VIN, GPPN [18].

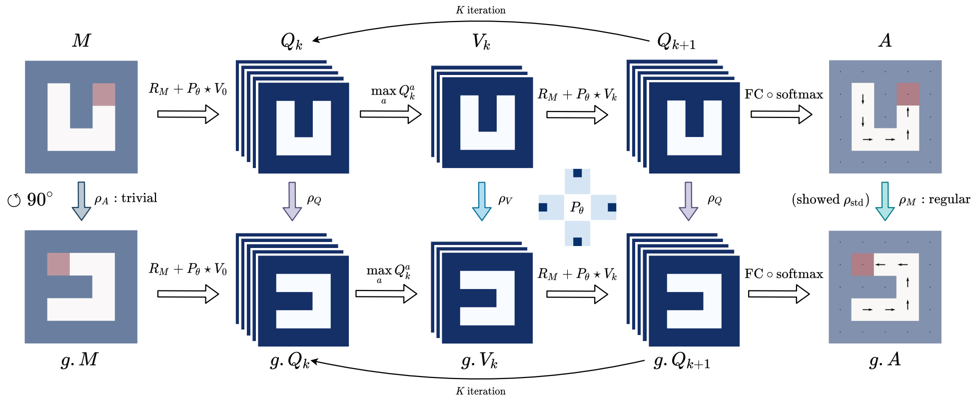

Figure 3: Commutative diagram for the full pipeline of SymVIN on steerable feature fields over $\mathbb{Z}^2$ (every grid). If rotating the input map M by $\pi_M(g)$ of any g, the output action $A = \texttt{SymVIN}(M)$ is guaranteed to be transformed by $\pi_A(g)$, i.e. the entire steerable SymVIN is equivariant under induced representations π_M and π_A: $\texttt{SymVIN}(\pi_M(g)M) = \pi_A(g)\texttt{SymVIN}(M)$. We use stacked feature fields to emphasize that SymVIN supports direct-sum of representations beyond scalar-valued.

Steerable value iteration. We have showed that, value iteration for path planning problems on $\mathbb{Z}^2$ consists of equivariant maps between steerable feature fields. It can be implemented as an equivariant steerable CNN, with recursively applying two alternating (equivariant) layers:

$$Q_k^a(s) = R_m^a(s) + \gamma \times \left[P_\theta^a \star V_k\right](s), \quad V_{k+1}(s) = \max_a Q_k^a(s), \quad s \in \mathbb{Z}^2, \tag{4}$$

where $k \in [K]$ indexes iteration, V_k, Q_k^a, R_m^a are steerable feature fields over $\mathbb{Z}^2$ output by equivariant layers, P_θ^a is a learned kernel in neural network, and $+, \times$ are element-wise operations.

Pipeline. We follow the pipeline in VIN [17]. The commutative diagram for the full pipeline is shown in Figure 3. The path planning task is given by a $m \times m$ spatial binary obstacle occupancy map and one-hot goal map, represented as a feature field $M : \mathbb{Z}^2 \to \{0,1\}^2$. For the iterative process $Q_k^a \mapsto V_k \mapsto Q_{k+1}^a$, the reward field R_M is predicted from map M (by a 1×1 convolution layer) and the value field V_0 is initialized as zeros. The network output is (logits of) planned actions for all locations[2], represented as $A : \mathbb{Z}^2 \to \mathbb{R}^{|\mathcal{A}|}$, predicted from the final Q-value field Q_K (by another 1×1 convolution layer). The number of iterations K and the convolutional kernel size F of P_θ^a are set based on map size M, and the spatial dimension $m \times m$ is kept consistent.

Building Symmetric Value Iteration Networks. Given the pipeline of VIN fully on steerable feature fields, we are ready to build equivariant version with $\mathrm{E}(2)$-steerable CNNs [16]. The idea is to replace every `Conv2d` with a steerable convolution layer between steerable feature fields, and associate the fields with proper fiber representations $\rho(h)$.

VINs use ordinary CNNs and can choose the size of intermediate feature maps. The design choices in steerable CNNs is the feature fields and fiber representations (or *type*) for every layer [14, 16]. The main difference[3] in steerable CNNs is that we also need to tell the network how to *transform* every *feature field*, by specifying *fiber representations*, as shown in Figure 3.

Specification of input map and output action. We first specify *fiber representations* for the input and output field of the network: map M and action A. For input **occupancy map and goal** M : $\mathbb{Z}^2 \to \{0,1\}^2$, it does not D_4 to act on the 2 channels, so we use two copies of trivial representations $\rho_M = \rho_{\text{triv}} \oplus \rho_{\text{triv}}$. For **action**, the final action output $A : \mathbb{Z}^2 \to \mathbb{R}^{|\mathcal{A}|}$ is for logits of four actions $\mathcal{A} = \texttt{(north, west, south, east)}$ for every location. If we use $H = C_4$, it naturally acts on the four actions (ordered $\circlearrowleft$) by *cyclically* $\circlearrowleft$ *permuting* the $\mathbb{R}^4$ channels. However, since the D_4 group has 8 elements, we need a *quotient representation*, see [16] and Appendix J.

Specification of intermediate fields: value and reward. Then, for the intermediate feature fields: Q-values Q_k, state value V_k, and reward R_m, we are free to choose fiber representations, as well as the width (number of copies). For example, if we want 2 copies of regular representation of D_4, the feature field has $2 \times 8 = 16$ channels and the stacked representation is 16×16 (by direct-sum).

[2]Technically, it also includes values or actions for obstacles, since the network needs to learn to approximate the reward $R_M(s, \Delta s) = -\infty$ with enough small reward and avoid obstacles.

For the Q**-value field** $Q_k^a(s)$, we use representation ρ_Q and its size as C_Q. We need at least $C_A \geq |\mathcal{A}|$ channels for all actions of $Q(s,a)$ as in VIN and GPPN, then stacked together and denoted as $Q_k \triangleq \bigoplus_a Q_k^a$ with dimension $Q_k : \mathbb{Z}^2 \to \mathbb{R}^{C_Q * C_A}$. Therefore, the representation is direct-sum $\bigoplus \rho_Q$ for C_A copies. The **reward** is implemented similarly as $R_M \triangleq \bigoplus_a R_M^a$ and must have same dimension and representation to add element-wisely. For **state value** field, we denote the choose as fiber representation as ρ_V and its size C_V. It has size $V_k : \mathbb{Z}^2 \to \mathbb{R}^{C_V}$ Thus, the steerable kernel is *matrix-valued* with dimension $P_\theta : \mathbb{Z}^2 \to \mathbb{R}^{(C_Q * C_A) \times C_V}$. In practice, we found using *regular representations* for all three works the best. It can be viewed as "augmented" state and is related to group convolution, detailed in Appendix J.

Other operations. We now visit the remained (equivariant) operations. (1) The $\max$ **operation in** $Q_k \mapsto V_{k+1}$**.** While we have showed the $\max$ operation in $V_{k+1}(s) = \max_a Q_k^a(s)$ is equivariant in Theorem 4.3, we need to apply max(-pooling) for all actions along the "representation channel" from stacked representations $C_A * C_Q$ to one C_Q. More details are in Appendix J.5. (2) The **final output layer** $Q_K \mapsto A$**.** After the final iteration, the Q-value field Q_k is fed into the policy layer with 1×1 convolution to convert the action logit field $\mathbb{Z}^2 \to \mathbb{R}^{|\mathcal{A}|}$.

Extended method: Symmetric GPPN. Gated path planning network (GPPN [18]) proposes to use LSTM to alleviate the issue of unstable gradient in VINs. Although it does not strictly follow value iteration, it still follows the spirit of steerable planning. Thus, we first obtained a fully convolutional variant of GPPN from [Redacted for anonymous review], called ConvGPPN. It replaces the MLPs in the original LSTM cell with convolutional layers, and then replaces convolutions with equivariant steerable convolutions, resulting in a fully equivariant SymGPPN. See Appendix J.3 for details.

Extended tasks: planning on learned maps with mapper networks. We consider two planning tasks on 2D grids: 2D navigation and 2-DOF manipulation. To demonstrate the ability of handling symmetry in differentiable planning, we consider more complicated state space input: visual navigation and workspace manipulation, and discuss how to use mapper networks to convert the state input and use end-to-end learned maps, as in [18, 37]. See Appendix J.2 for details.

6 Experiments

We experiment VIN, GPPN and our SymPlan methods on four path planning tasks, including using *given* or *learned* maps. The additional experiments and ablation studies are in Appendix E.

Environments and datasets. We demonstrate the idea in two major robotics tasks: *navigation* and *manipulation*. We focus on the 2D regular grid setting for path planning, as adopted in prior work [17, 18, 37]. For each task, we consider using either *given* (2D navigation and 2-DOF configuration-space manipulation) or *learned* maps (visual navigation and 2-DOF workspace manipulation). In the latter case, the planner needs to jointly learn a mapper that converts egocentric panoramic images (visual navigation) or workspace states (workspace manipulation) into plannable loss, as in [18, 37]. In both cases, we randomly generate training, validation and test data of $10K/2K/2K$ maps for all map sizes, to demonstrate data efficiency and generalization ability of symmetric planning. Note that the test maps are unlikely to be symmetric to the training maps by any transformation from the symmetry groups G. For all environments, the planning domain is the 2D regular grid $\mathcal{S} = \Omega = \mathbb{Z}^2$, and the action space is to move in 4 ↺ directions[4]: $\mathcal{A} =$ `(north, west, south, east)`.

Methods: planner networks. We compare five planner methods, where two are our SymPlan version of their non-equivariant counterparts. Our equivariant implementation is based on *Value Iteration Networks* (**VIN**, [17]) and *Gated Path Planning Networks* (**GPPN**, [18]). We implement the equivariant version of VIN, named **SymVIN**. For GPPN, we first obtained a *fully convolutional* version, named **ConvGPPN** [Redacted for anonymous review], and furthermore **SymGPPN** with steerable CNNs. All methods use (equivariant) convolutions with *circular padding* in planning in configuration spaces for the manipulation tasks, except GPPN that is not fully convolutional. Chaplot et al. [37] propose SPT based on Transformers, while integrating symmetry to Transformers is beyond steerable convolutions, thus we do not consider it but still adopt some useful setup.

[4]Note that the MDP action space $\mathcal{A}$ needs to be *compatible* with the group action $G \times \mathcal{A} \to \mathcal{A}$. Since the E2CNN package [16] uses *counterclockwise* rotations ↺ as generators for rotation groups C_n, the action space needs to be *counterclockwise* ↺.

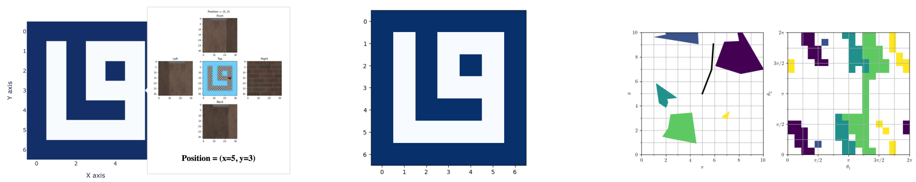

Figure 4: **(Left)** A visual navigation environment rendered from a randomly generated 7×7 maze **(Middle)**, where the hover is the visualization of four views at position $(5, 3)$. **(Right)** A 2-joint manipulation task in workspace (topdown) and configuration space (2 DOFs) in 18×18 resolution.

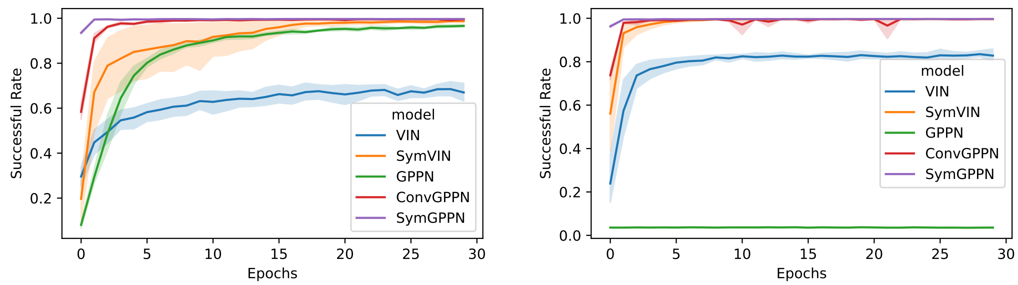

Figure 5: Training curves on **(Left)** 2D navigation with 10K of 15×15 maps and on **(Right)** 2DoFs manipulation with 10K of 18×18 maps in configuration space. Faded areas indicate standard error.

Training and evaluation. We report success rate and training curves over 3 seeds. The training process (on given maps) follows [17, 18], where we train 30 epochs with batch size 32, and use kernel size $F = 3$ by default. The gradient clip threshold is set to 5. The default batch size is 32, while we need to reduce for some GPPN variants, since LSTM consumes much more memory.

6.1 Planning on given maps

Environmental setup. In the **2D navigation** task, the map and goal are randomly generated, where the map size is $\{15, 28, 50\}$. In **2-DOF manipulation** in configuration space, we adopt the setting in [37] and train networks to take as input of configuration space, represented by two joints. We randomly generate 0 to 5 obstacles in the manipulator workspace. Then the 2 degree-of-freedom (DOF) configuration space is constructed from workspace and discretized into 2D grid with sizes $\{18, 36\}$, corresponding to bins of $20°$ and $10°$, respectively. All methods are trained using the same network size, where for equivariant versions, we use *regular* representations for all layers, which has size $|D_4| = 8$. We keep the same parameters for all methods, so all equivariant convolution layers with *regular* representations will have higher embedding sizes. Due to memory constraint, we use $K = 30$ iterations for 2D maze navigation, and $K = 27$ for manipulation. We use kernel sizes $F = \{3, 5, 5\}$ for $m = \{15, 28, 50\}$ navigation, and $F = \{3, 5\}$ for $m = \{18, 36\}$ manipulation.

Results. We show the averaged test results for both 2D navigation and C-space manipulation tasks on generalizing to unseen maps (Table 1) and the training curves for all methods (Figure 5). For VIN series, our SymVIN is much better than the vanilla VIN in terms of generalization and training performance in both environments, which learns much faster and achieves almost perfect asymptotic performance. As for GPPN, we found the fully convolutional variant ConvGPPN actually works better than the original one in [18], especially in learning speed. However, SymVIN does fluctuate in some runs, which seems to come from initialization and label, further studied in Appendix. SymGPPN further boosts ConvGPPN and outperforms all other methods. One exception is GPPN learns poorly in C-space manipulation. For GPPN, the added circular padding in the convolution encoder leads to gradient vanishing problem.

Additionally, we found using regular representations (for D_4 or C_4) for state value $V : \mathbb{Z}^2 \to \mathbb{R}^{C_V}$ (and for Q-value) works better than trivial representations. This is counterintuitive since we expect the V value to be scalar $\mathbb{Z}^2 \to \mathbb{R}$. One reason is that switching between regular (for Q) and trivial (for V) representation introduces unnecessary bottleneck. Depending on the choice of representations, we implement different max-pooling, with details in Appendix J.5. We also empirically found using FC only in the final layer $Q_K \mapsto A$ helps stabilize the training a bit. The ablation study on this and more are in Appendix E.

Table 1: Averaged test success rate (%) for using 10K/2K/2K dataset for all four types of tasks.

Method (10K Data)	Navigation 15×15	28×28	50×50	Visual	Manipulation 18×18	36×36	Workspace
VIN	66.97	67.57	57.92	50.83	77.82	84.32	80.44
SymVIN	98.99	98.14	86.20	95.50	99.98	99.36	**91.10**
GPPN	96.36	95.77	91.84	93.13	2.62	1.68	3.67
ConvGPPN	99.75	99.09	97.21	98.55	99.98	99.95	89.88
SymGPPN	**99.98**	**99.86**	**99.49**	**99.78**	**100.00**	**99.99**	90.50

Remark. Two symmetric planners are both significantly better than their counterparts. Notably, we did not include any symmetric maps to the test data that symmetric planners would perform much better. There are several potential sources of advantages: (1) SymPlan allows parameter sharing across positions and maps and implicitly enables planning in a reduced space: every (s, a, s') seamlessly generalizes to $(g.s, g.a, g.s')$ for any $g \in G$, (2) thus it uses training data more efficiently, (3) it reduces the space of hypothesis class and facilitate generalization to unseen maps.

6.2 Planning on learned maps: simultaneously planning and mapping

Environmental setup. For **visual navigation**, we randomly generate maps using the same strategy as before, and then render four egocentric panoramic views for each location from produced 3D environments with *Gym-MiniWorld* [57], since it allows to generate 3D mazes with any layout. For $m \times m$ maps, all egocentric views for a map is represented by $m \times m \times 4$ RGB images. For **workspace manipulation**, we randomly generate 0 to 5 obstacles in workspace as before. We use a mapper network to convert the 96×96 workspace (image of obstacles) to the $m \times m$ 2 degree-of-freedom (DOF) configuration space (2D occupancy grid). In both environments, the setup is similar to Section 6.1, while we only use $m = 15$ maps but longer 100 epochs for visual navigation and $m = 18$ maps still with 30 epochs for workspace manipulation.

Methods: mapper networks and setup. For **visual navigation**, we follow the mapper network setup in [18]. A mapper network converts every image into a 256-dimensional embedding $m \times m \times 4 \times 256$ and then predicts map layout $m \times m \times 1$. For **workspace manipulation**, we use U-net [58] with residual-connection [59] as a mapper. See Section E for details.

Results. The results are also shown in Table 1, denoted as Visual (navigation, 15×15) and Workspace (manipulation, 18×18). In visual navigation, the trends are similar to 2D case: two symmetric planners both train much faster. Besides vanilla VIN, all approaches finally converge to near-optimal successful rate (around 95%), while the validation and test results show large gaps. SymGPPN has almost no generalization gap, while VIN does not generalize well to new 3D visual navigation environments. Our SymVIN improves test successful rate from less than 50% to 90% and is comparable with GPPN. Since the input is raw images and a mapper is used to learn end-to-end, it potentially causes one major source of generalization gap for some approaches. In workspace manipulation, the results are also analogous to C-space, while ours advantages over baselines are smaller. In our inspection, we found the mapper network is the bottleneck, since the mapping for obstacles from workspace to C-space is nontrivial to learn.

Remark. The SymPlan models demonstrate end-to-end planning and learning ability, potentially enabling further applications to other tasks as a differentiable component for planning. The additional results and ablation studies are provided in Appendix E.

7 Discussion

In this work, we study the symmetry in 2D path planning problem, and build a framework using the theory of steerable CNNs to prove that value iteration in path planning is actually a form of steerable CNN (on 2D grids). Although we focus on $\mathbb{Z}^2$, we can generalize to path planning on higher-dimensional or even continuous Euclidean spaces [54, 55], and use *equivariant operations* on *steerable feature fields* (such as steerable convolutions, pooling, and point-wise non-linearities) from steerable CNNs. We practically show that the SymPlan algorithms exactly motivated by the theory provide great improvement. We hope the framework along with the design of practical algorithms can provide a new pathway to exploiting symmetry structure in differentiable planning.

References

[1] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: an introduction*. Adaptive computation and machine learning series. The MIT Press, Cambridge, Massachusetts, second edition edition, 2018. ISBN 978-0-262-03924-6.

[2] Lihong Li, Thomas J. Walsh, and M. Littman. Towards a Unified Theory of State Abstraction for MDPs. In *AI&M*, 2006.

[3] Balaraman Ravindran and Andrew G Barto. *An algebraic approach to abstraction in reinforcement learning*. PhD thesis, University of Massachusetts at Amherst, 2004.

[4] Maria Fox and Derek Long. Extending the exploitation of symmetries in planning. In *In Proceedings of AIPS'02*, pages 83–91, 2002.

[5] Maria Fox and Derek Long. The Detection and Exploitation of Symmetry in Planning Problems. In *In IJCAI*, pages 956–961. Morgan Kaufmann, 1999.

[6] Nir Pochter, Aviv Zohar, and Jeffrey S. Rosenschein. Exploiting Problem Symmetries in State-Based Planners. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*, August 2011. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI11/paper/view/3732.

[7] Martin Zinkevich and Tucker Balch. Symmetry in Markov decision processes and its implications for single agent and multi agent learning. In *In Proceedings of the 18th International Conference on Machine Learning*, pages 632–640. Morgan Kaufmann, 2001.

[8] Shravan Matthur Narayanamurthy and Balaraman Ravindran. On the hardness of finding symmetries in Markov decision processes. In *Proceedings of the 25th international conference on Machine learning - ICML '08*, pages 688–695, Helsinki, Finland, 2008. ACM Press. ISBN 978-1-60558-205-4. doi: 10/bkswc2. URL http://portal.acm.org/citation.cfm?doid=1390156.1390243.

[9] Elise van der Pol, Daniel E. Worrall, Herke van Hoof, Frans A. Oliehoek, and Max Welling. MDP Homomorphic Networks: Group Symmetries in Reinforcement Learning. *arXiv:2006.16908 [cs, stat]*, June 2020. URL http://arxiv.org/abs/2006.16908. arXiv: 2006.16908.

[10] Dian Wang, Robin Walters, and Robert Platt. $\mathrm{SO}(2)$-Equivariant Reinforcement Learning. September 2021. URL https://openreview.net/forum?id=7F9cOhdvfk_.

[11] Michael M Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

[12] Taco Cohen, Mario Geiger, and Maurice Weiler. A General Theory of Equivariant CNNs on Homogeneous Spaces. *arXiv:1811.02017 [cs, stat]*, January 2020. URL http://arxiv.org/abs/1811.02017. arXiv: 1811.02017.

[13] Risi Kondor and Shubhendu Trivedi. On the Generalization of Equivariance and Convolution in Neural Networks to the Action of Compact Groups. *arXiv:1802.03690 [cs, stat]*, November 2018. URL http://arxiv.org/abs/1802.03690. arXiv: 1802.03690.

[14] Taco S. Cohen and Max Welling. Steerable CNNs. November 2016. URL https://openreview.net/forum?id=rJQKYt5ll.

[15] Taco S. Cohen and Max Welling. Group Equivariant Convolutional Networks. *arXiv:1602.07576 [cs, stat]*, June 2016. URL http://arxiv.org/abs/1602.07576. arXiv: 1602.07576.

[16] Maurice Weiler and Gabriele Cesa. General $E(2)$-Equivariant Steerable CNNs. *arXiv:1911.08251 [cs, eess]*, April 2021. URL http://arxiv.org/abs/1911.08251. arXiv: 1911.08251.

[17] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value Iteration Networks. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, pages 4949–4953, Melbourne, Australia, August 2017. International Joint Conferences on Artificial Intelligence Organization. ISBN 978-0-9992411-0-3. doi: 10/ggjfst. URL https://www.ijcai.org/proceedings/2017/700.

[18] Lisa Lee, Emilio Parisotto, Devendra Singh Chaplot, Eric Xing, and Ruslan Salakhutdinov. Gated Path Planning Networks. *arXiv:1806.06408 [cs, stat]*, June 2018. URL http://arxiv.org/abs/1806.06408. arXiv: 1806.06408.

[19] Carmel Domshlak, Michael Katz, and Alexander Shleyfman. Enhanced Symmetry Breaking in Cost-Optimal Planning as Forward Search. page 5.

[20] Alexander Shleyfman. Symmetry Breaking: Satisficing Planning and Landmark Heuristics. page 5.

[21] Alexander Shleyfman, Michael Katz, Malte Helmert, Silvan Sievers, and Martin Wehrle. Heuristics and Symmetries in Classical Planning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 29(1), March 2015. ISSN 2374-3468. URL https://ojs.aaai.org/index.php/AAAI/article/view/9649. Number: 1.

[22] Silvan Sievers, Martin Wehrle, Malte Helmert, Alexander Shleyfman, and Michael Katz. Factored Symmetries for Merge-and-Shrink Abstractions. page 8.

[23] Martin Wehrle, Malte Helmert, Alexander Shleyfman, and Michael Katz. Integrating Partial Order Reduction and Symmetry Elimination for Cost-Optimal Classical Planning. page 7.

[24] Mohammad Abdulaziz, Michael Norrish, and Charles Gretton. Exploiting Symmetries by Planning for a Descriptive Quotient. page 8.

[25] Silvan Sievers, Martin Wehrle, Malte Helmert, and Michael Katz. An Empirical Case Study on Symmetry Handling in Cost-Optimal Planning as Heuristic Search. In Steffen Hölldobler, Rafael Peñaloza, and Sebastian Rudolph, editors, *KI 2015: Advances in Artificial Intelligence*, volume 9324, pages 166–180. Springer International Publishing, Cham, 2015. ISBN 978-3-319-24488-4 978-3-319-24489-1. doi: 10.1007/978-3-319-24489-1_13. URL http://link.springer.com/10.1007/978-3-319-24489-1_13. Series Title: Lecture Notes in Computer Science.

[26] Silvan Sievers. Structural Symmetries of the Lifted Representation of Classical Planning Tasks. page 8.

[27] Dominik Winterer, Martin Wehrle, and Michael Katz. Structural Symmetries for Fully Observable Nondeterministic Planning. page 7.

[28] Gabriele Röger, Silvan Sievers, and Michael Katz. Symmetry-Based Task Reduction for Relaxed Reachability Analysis. In *Twenty-Eighth International Conference on Automated Planning and Scheduling*, June 2018. URL https://aaai.org/ocs/index.php/ICAPS/ICAPS18/paper/view/17772.

[29] Silvan Sievers, Gabriele Röger, Martin Wehrle, and Michael Katz. Theoretical Foundations for Structural Symmetries of Lifted PDDL Tasks. *Proceedings of the International Conference on Automated Planning and Scheduling*, 29:446–454, 2019. ISSN 2334-0843. URL https://ojs.aaai.org/index.php/ICAPS/article/view/3509.

[30] Daniel Fišer, Álvaro Torralba, and Alexander Shleyfman. Operator Mutexes and Symmetries for Simplifying Planning Tasks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):7586–7593, July 2019. ISSN 2374-3468. doi: 10.1609/aaai.v33i01.33017586. URL https://ojs.aaai.org/index.php/AAAI/article/view/4751. Number: 01.

[31] N. Ferns, P. Panangaden, and Doina Precup. Metrics for Finite Markov Decision Processes. In *AAAI*, 2004.

[32] Elise van der Pol, Daniel Worrall, Herke van Hoof, Frans Oliehoek, and Max Welling. Mdp homomorphic networks: Group symmetries in reinforcement learning. *Advances in Neural Information Processing Systems*, 33, 2020.

[33] Jung Yeon Park, Ondrej Biza, Linfeng Zhao, Jan Willem van de Meent, and Robin Walters. Learning Symmetric Embeddings for Equivariant World Models. *arXiv:2204.11371 [cs]*, April 2022. URL http://arxiv.org/abs/2204.11371. arXiv: 2204.11371.

[34] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges. *arXiv:2104.13478 [cs, stat]*, April 2021. URL http://arxiv.org/abs/2104.13478. arXiv: 2104.13478.

[35] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. *arXiv preprint arXiv:1602.02867*, 2016.

[36] Sufeng Niu, Siheng Chen, Hanyu Guo, Colin Targonski, Melissa C. Smith, and Jelena Kovačević. Generalized Value Iteration Networks: Life Beyond Lattices. *arXiv:1706.02416 [cs]*, October 2017. URL http://arxiv.org/abs/1706.02416. arXiv: 1706.02416.

[37] Devendra Singh Chaplot, Deepak Pathak, and Jitendra Malik. Differentiable Spatial Planning using Transformers. *arXiv:2112.01010 [cs]*, December 2021. URL http://arxiv.org/abs/2112.01010. arXiv: 2112.01010.

[38] Andreea Deac, Petar Veličković, Ognjen Milinković, Pierre-Luc Bacon, Jian Tang, and Mladen Nikolić. Neural Algorithmic Reasoners are Implicit Planners. October 2021. URL https://arxiv.org/abs/2110.05442v1.

[39] Junhyuk Oh, Satinder Singh, and Honglak Lee. Value Prediction Network. *arXiv:1707.03497 [cs]*, November 2017. URL http://arxiv.org/abs/1707.03497. arXiv: 1707.03497.

[40] Peter Karkus, David Hsu, and Wee Sun Lee. QMDP-Net: Deep Learning for Planning under Partial Observability. *arXiv:1703.06692 [cs, stat]*, November 2017. URL http://arxiv.org/abs/1703.06692. arXiv: 1703.06692.

[41] Théophane Weber, Sébastien Racanière, David P. Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adria Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, Razvan Pascanu, Peter Battaglia, Demis Hassabis, David Silver, and Daan Wierstra. Imagination-Augmented Agents for Deep Reinforcement Learning. *arXiv:1707.06203 [cs, stat]*, February 2018. URL http://arxiv.org/abs/1707.06203. arXiv: 1707.06203.

[42] Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Universal Planning Networks. *arXiv:1804.00645 [cs, stat]*, April 2018. URL http://arxiv.org/abs/1804.00645. arXiv: 1804.00645.

[43] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model. *arXiv:1911.08265 [cs, stat]*, November 2019. URL http://arxiv.org/abs/1911.08265. arXiv: 1911.08265.

[44] Brandon Amos, Ivan Dario Jimenez Rodriguez, Jacob Sacks, Byron Boots, and J. Zico Kolter. Differentiable MPC for End-to-end Planning and Control. *arXiv:1810.13400 [cs, math, stat]*, October 2019. URL http://arxiv.org/abs/1810.13400. arXiv: 1810.13400.

[45] Tingwu Wang and Jimmy Ba. Exploring Model-based Planning with Policy Networks. June 2019. URL https://arxiv.org/abs/1906.08649v1.

[46] Arthur Guez, Mehdi Mirza, Karol Gregor, Rishabh Kabra, Sébastien Racanière, Théophane Weber, David Raposo, Adam Santoro, Laurent Orseau, Tom Eccles, Greg Wayne, David Silver, and Timothy Lillicrap. An investigation of model-free planning. *arXiv:1901.03559 [cs, stat]*, May 2019. URL http://arxiv.org/abs/1901.03559. arXiv: 1901.03559.

[47] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to Control: Learning Behaviors by Latent Imagination. *arXiv:1912.01603 [cs]*, March 2020. URL http://arxiv.org/abs/1912.01603. arXiv: 1912.01603.

[48] Vitchyr Pong, Shixiang Gu, Murtaza Dalal, and Sergey Levine. Temporal Difference Models: Model-Free Deep RL for Model-Based Control. *arXiv:1802.09081 [cs]*, February 2018. URL http://arxiv.org/abs/1802.09081. arXiv: 1802.09081.

[49] Ignasi Clavera, Violet Fu, and Pieter Abbeel. Model-Augmented Actor-Critic: Backpropagating through Paths. *arXiv:2005.08068 [cs, stat]*, May 2020. URL http://arxiv.org/abs/2005.08068. arXiv: 2005.08068.

[50] Christopher Grimm, André Barreto, Satinder Singh, and David Silver. The Value Equivalence Principle for Model-Based Reinforcement Learning. *arXiv:2011.03506 [cs]*, November 2020. URL http://arxiv.org/abs/2011.03506. arXiv: 2011.03506.

[51] Christopher Grimm, André Barreto, Gregory Farquhar, David Silver, and Satinder Singh. Proper Value Equivalence. *arXiv:2106.10316 [cs]*, December 2021. URL http://arxiv.org/abs/2106.10316. arXiv: 2106.10316.

[52] Clement Gehring, Kenji Kawaguchi, Jiaoyang Huang, and Leslie Pack Kaelbling. Understanding End-to-End Model-Based Reinforcement Learning Methods as Implicit Parameterization. May 2021. URL https://openreview.net/forum?id=xj2sE--Q90e.

[53] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, May 2006. ISBN 978-1-139-45517-6.

[54] Maurice Weiler, M. Geiger, M. Welling, Wouter Boomsma, and Taco Cohen. 3D Steerable CNNs: Learning Rotationally Equivariant Features in Volumetric Data. In *NeurIPS*, 2018.

[55] Johannes Brandstetter, Rob Hesselink, Elise van der Pol, Erik J. Bekkers, and Max Welling. Geometric and Physical Quantities Improve E(3) Equivariant Message Passing. *arXiv:2110.02905 [cs, stat]*, December 2021. URL http://arxiv.org/abs/2110.02905. arXiv: 2110.02905.

[56] T. S. Cohen. Equivariant convolutional networks. 2021. URL https://dare.uva.nl/search?identifier=0f7014ae-ee94-430e-a5d8-37d03d8d10e6.

[57] Maxime Chevalier-Boisvert. Miniworld: Minimalistic 3d environment for rl & robotics research. https://github.com/maximecb/gym-miniworld, 2018.

[58] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015. URL http://arxiv.org/abs/1505.04597.

[59] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [No]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] Briefly in Section 4, and in full in the supplementary material.

(b) Did you include complete proofs of all theoretical results? [Yes] See supplementary material.

3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] The code and data will be cleaned and released prior to final publication.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Briefly in Section 6, and in full in the supplementary material.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Briefly in Section 6, and in full in the supplementary material.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See supplementary material.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [No]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Outline

The appendix is organized as follows. Blue text highlights new content in revision for rebuttal.

1. A temporary section for new figures and results for rebuttal.
2. A preliminary version of a section on Symmetric Planning with less prerequisites on equivariant networks.
3. Additional experimental setup and empirical results. (This section is moved here previously at the end of appendix.)
4. Discussion on the considered symmetry, as well as limitations and extensions.
5. Additional technical background and concepts on steerable CNNs and group CNNs, useful for understanding how to apply our setup to other problems and setup.
6. More details and interpretation on the steerable planning framework.
7. Full derivation and proofs.
8. Other practice implementation details.

B A temporary section for new figures and results

This temporary section is a collection of all new figures and results for the rebuttal purpose. All the content will be merged into the corresponding sections in the future version.

B.1 Updated environment figures

To emphasize the tasks, we update the figures for the environments in our experiments, along with demonstration of the learned model.

We show the figures for **Configuration-space and Workspace manipulation** in Figure 6, and the figures for **2D and Visual Navigation** in Figure 7.

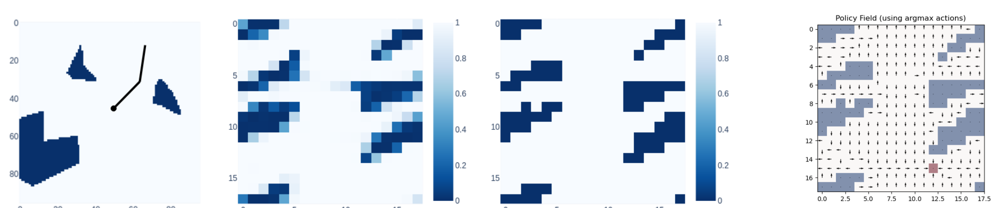

Figure 6: A set of visualization for a 2-joint manipulation task. The obstacles are randomly generated. **(1)** The 2-joint manipulation task shown in top-down workspace with 96×96 resolution. This is used as the input to the **Workspace Manipulation** task. **(2)** The predicted configuration space in resolution 18×18 from a mapper module, which is jointly optimized with a planner network. **(3)** The ground truth configuration space from a handcraft algorithm in resolution 18×18. This is used as input to the **Configuration-space (C-space) Manipulation** task and as target in the auxiliary loss for the Workspace Manipulation task (as done in SPT [37]). **(4)** The predicted policy (overlaid with C-space obstacle for visualization) from an end-to-end trained SymVIN model that uses a mapper to take the top-down workspace image and plans on a learned map. The red block is the goal position.

B.2 Results on generalization to larger maps

To better demonstrate the empirical difference, we conduct new experiment on generalization to larger maps. We hope this can alleviate some concern on (1) scalability and (2) performance gap between SymGPPN and ConvGPPN.

We experiment all methods on map size 15×15 through 99×99, averaging over 3 seeds (3 model checkpoints) for each method and 1000 maps for each size. Note that all models are trained on 15×15 with $K = 30$.

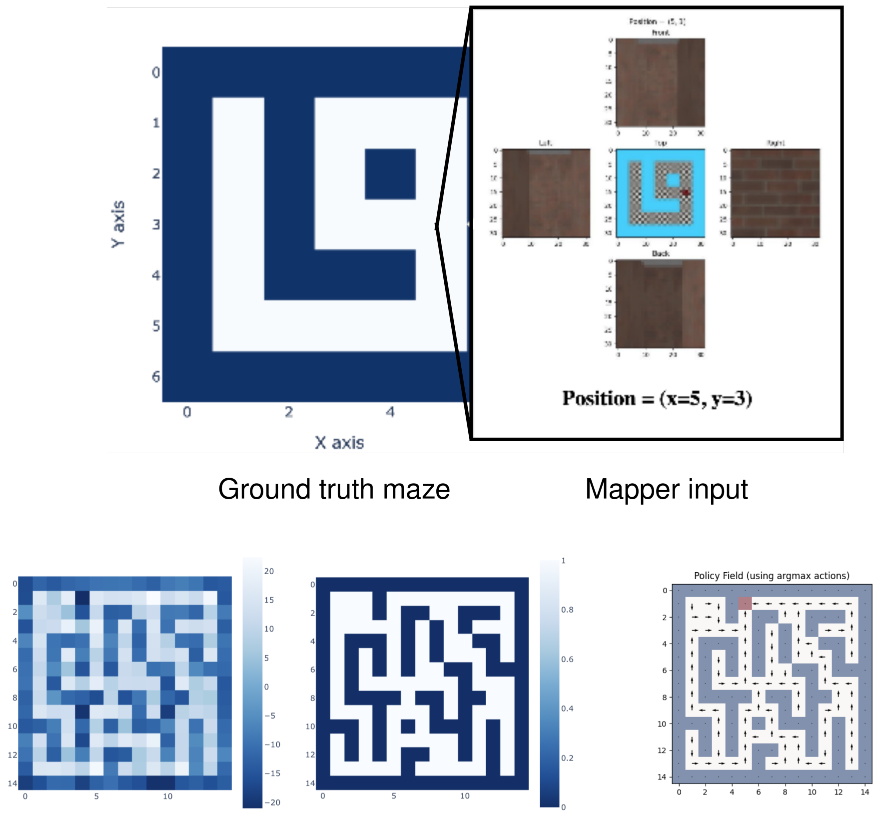

Figure 7: A set of visualization for 2D navigation and visual navigation. The maze is randomly generated. **(1, top)** The 3D visual navigation environment generated by an illustrative 7×7 map, where we highlight the panoramic view at a position $(5, 3)$ with four RGB images (resolution $32 \times 32 \times 3$). The entire observation tensor for this 7×7 example visual navigation environment is $7 \times 7 \times 4 \times 32 \times 32 \times 3$. This is used as the input to the **Visual Navigation** task. **(2)** Another predicted map in resolution 15×15 from a mapper module, which is jointly optimized with a planner network. We show the visualization a different map used in actual training. **(3)** The ground truth map in resolution 15×15. This is also used as input to the **2D Navigation** task and as target in the auxiliary loss for the Visual Navigation task (as done in GPPN). **(4)** The predicted policy from an end-to-end trained SymVIN model that uses a mapper to take the observation images (formed as a tensor) and plans on a learned map. The red block is the goal position.

Between 15×15 and 49×49 we use all odd-size maps, and between 51×51 and 99×99 we use interval of 4 ($51 \times 51 \to 55 \times 55$ ...). We only use odd size maps because for the even size maps the maze generation algorithm would cause non-symmetric pattern (missing right and bottom boundary).

Note that we disable backward pass during evaluation. However, we observe that GPPN variants do use much more memory if backward pass is enabled because (1) they rely on the costly computation of LSTM, and (2) the number of parameters is also significantly larger. The training and inference time used by GPPN variants is also significantly longer. We omit the consideration of resource and time issue and focus on the final generalization results.

We focus on comparing SymPlan methods with non-equivariant baselines, by grouping them based on VIN or GPPN. The results are shown in Figure 8.

Fixed K. For fixed K setup in Figure 8 (left), we keep number of iterations to be $K = 30$ and kernel size $F = 3$ for all methods.

For SymVIN, it far surpasses VIN for all sizes and preserves the gap throughout the evaluation. Additionally, SymVIN has slightly higher variance across three random seeds (three separately trained models).

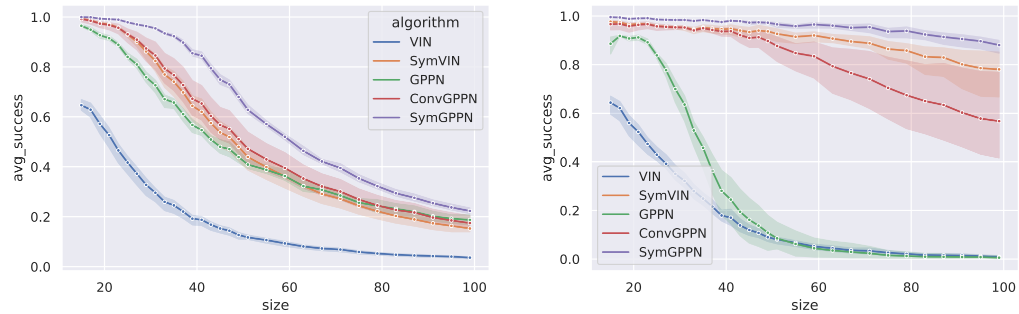

Figure 8: Results for generalization on larger maps for all methods. **(Left)** Fixed $K = 30$ iterations. **(Left)** Variable K iterations, where $K = \sqrt{2} \cdot M$ and M is the generalization map size (x-axis).

Among GPPN and its variants, SymGPPN significantly outperforms both GPPN and ConvGPPN. Interestingly, ConvGPPN has sharper drop with map size than both SymGPPN and ConvGPPN and thus has increasingly larger gap with SymGPPN and finally even got surpassed by GPPN. Across random seeds, the three trained models of ConvGPPN give unexpectedly high variance compared to GPPN and SymGPPN.

Variable K**.** We also experiment all methods in the same setting but with variable number of iterations $K = \sqrt{2} \cdot M$, where M is the generalization map size (x-axis). The trend is very different from fixed K setup.

SymVIN generalizes extremely well compared to VIN, although the variance is greater. GPPN seems to diverge for larger variable K since it is even worse than fixed $K = 30$ in all map sizes. ConvGPPN somehow helps convergence, while it fluctuates for different seeds, and SymGPPN is even better and more stable. Surprisingly, SymVIN is even better than ConvGPPN, although injecting symmetry (into SymVIN) does not explicitly deal with convergence.

C A Guide to Symmetric Planning

To address the common concern on the accessibility issue for technical section, as a step to solve this, we write a section on explaining the SymVIN method with PyTorch-style pseudocode, since it directly corresponds to what we propose in Section 4 and 5. We try to relate (1) existing concepts with VIN, (2) what we propose in Section 4 and 5 for SymVIN, and (3) actual PyTorch implementation of VIN and SymVIN aligned line-by-line based on semantic correspondence.

We will consider to have another short section on intuitively explaining our Symmetric Planning framework and practical considerations in the next few days.

C.1 PyTorch-style pseudocode

We provide the key Python code snippets to demonstrate how easy it is to implement SymVIN, our symmetric version of VIN [17].

In the current Section 5 (SymPlan practice), we heavily use the concepts from Steerable CNNs. Thanks to the equivariant network community and the `e2cnn` package, the actual implementation is compact and closely corresponds to their non-equivariant counterpart, VIN, line-by-line. Thus, the ultimate goal here is to illustrate that, whatever concepts we have in regular CNNs (e.g., have whatever channels we want), we can can use steerable CNNs that incorporate desired extra symmetry (of D_4 rotation+reflection or C_4 rotation).

We highlight the implementation of the value iteration procedure in VIN and SymVIN:

$$V := \max_a R^a + \gamma \times P^a * V. \tag{5}$$

Note that we use actual code snippets to avoid hiding any details.

```python
import torch










# Define regular 2D convolution
q_conv = torch.nn.Conv2d(
    in_channels=1,
    out_channels=2 * q_size,
    kernel_size=F, stride=1, bias=False
)
```

Listing 1: Define ‘expected value‘ convolution layer for VIN.

```python
1 import torch
2 import e2cnn
3
4 # Define the symmetry group to be D4
5 gspace = e2cnn.gspaces.FlipRot2dOnR2(N=4)
6 # Define feature (fiber) representations
7 field_type_q_in = e2cnn.nn.FieldType(
8     gspace=gspace,
9     representations=2 * q_size * [gspace.
       regular_repr]
10 )
11 # Define steerable convolution
12 q_r2conv = e2cnn.nn.R2Conv(
13     in_type=field_type_q_in,
14     out_type=field_type_q_out,
15     kernel_size=F, stride=1, bias=False
16 )
```

Listing 2: Define ‘expected value‘ (steerable) convolution layer for SymVIN.

Defining (steerable) convolution layer. First, we show the definition of the key convolution layer for a key operation in VIN and SymVIN: expected value operator, in Listing 1 and 2.

As proved in Theorem 4.2, the expected value operator can be executed by a steerable convolution layer for (2D) path planning. This serves as the theoretical foundation on how we should use a steerable layer here.

For the left side, a regular 2D convolution is defined for VIN. The right side defines a steerable convolution layer, using the library `e2cnn` from [16]. It provides high-level abstraction for building equivariant 2D steerable convolution networks. As a user, we only need to specify how the feature fields transform (as shown in Figure 3), and it will solve the G-steerability constraints, process what needs to be trained for equivariant layers, etc. We use name `q_r2conv` to highlight the difference.

Value iteration procedure. Second, we compare the for loop for value iteration updates in VIN and SymVIN, where the former one has regular 2D convolution `Conv2D` (Listing 3), and the latter one uses steerable convolution [16] (Listing 4).

The lines are aligned based on semantic correspondence. The `e2cnn` layers, including steerable convolution layers, operate on its `GeometricTensor` data structure, which is to wrap a PyTorch tensor. We denote them with `_geo` suffix. It only additionally needs to specify how this tensor (feature field) transforms under a group (e.g., D_4), i.e. the user needs to specify a group representation for it.

`tensor_directsum` is used to concatenate two `GeometricTensor`’s (feature fields) and compute their associated representations (by direct-sum).

Thus, the `e2cnn` steerable convolution layer on the right side `q_r2conv` can be used as a regular PyTorch layer, while the input and output are `GeometricTensor`.

We also define the $\max$ operation as a customized max-pooling layer, named `q_max_pool`. The implementation is similar to the left side of VIN and needs to additionally guarantee equivariance, and the detail is omitted.

Note that for readability, we assume we use regular representations for the Q-value field Q and the state-value field V. They are empirically found to work the best. This corresponds to the definition in `field_type_q_in` in line 9 in the SymVIN definition listing and the comments in line 16-17 in the steerable VI procedure listing for SymVIN.

Other components are omitted.

D Simplified Version: Symmetric Planning

This is a new preliminary version during the rebuttal period that aims to introduce symmetric planning in a more intuitive way, with minimum prerequisites of equivariant networks. This section is intended to be an alternative and more intuitive version to Section 4 (SymPlan framework) and Sec-

```python
# Input: maze and goal map, #iterations K




x = torch.cat([maze_map, goal_map], dim=1)

r = r_conv(x)

# Init value function V
v = torch.zeros(r.size())


for _ in range(K):
    # Concat and convolve V with P
    rv = torch.cat([r, v], dim=1)
    q = q_conv(rv)

    # Max over action channel
    # > Q: batch_size x q_size x W x H
    # > V: batch_size x 1 x W x H
    q = q.view(-1, q_size, W, H)
    v, _ = torch.max(q, dim=1)
    v = v.view(-1, W, H)

# Output: 'q' (to produce policy map)
```

Listing 3: The central value iteration procedure for VIN. Some variable names are adjusted accordingly for readability. W and H are width and height for 2D map.

```python
# Input: maze and goal map, #iterations K

from e2cnn.nn import GeometricTensor
from e2cnn.nn import tensor_directsum

x = torch.cat([maze_map, goal_map], dim=1)
x_geo = GeometricTensor(x, type=field_type_x)
r_geo = r_r2conv(x_geo)

# Init V and wrap V in e2cnn 'geometric tensor'
v_raw = torch.zeros(r_geo.size())
v_geo = GeometricTensor(v_raw, field_type_v)

for _ in range(K):
    # Concat (direct-sum) and convolve V with P
    rv_geo = tensor_directsum([r_geo, v_geo])
    q_geo = q_r2conv(rv_geo)

    # Max over group channel
    # > Q: batch_size x (|G| * q_size) x W x H
    # > V: batch_size x (|G| * 1) x W x H
    v_geo = q_max_pool(q_geo)



# Output: 'q_geo' (to produce policy map)
```

Listing 4: The equivariant steerable value iteration procedure for SymVIN. Lines are aligned by semantic correspondence. Definition of other field types are similar and thus omitted.

tion 5 (SymPlan in practice). We would appreciate feedback and consider to make further revision to this section and the organization of the entire paper.

D.1 Overview

In this work, we aim to exploit the inherent symmetry in a broadly existed problem: path planning. Intuitively, since a rotated or reflected 2D map are still another instance of 2D map, such as in Figure 1, their policies and optimal paths are related. This unveils an inherent symmetry property of the path planning problem on the 2D grid that we could exploit.

In our work, we provide a rigorous algorithmic framework that can *provably* make use of symmetry in an *efficient* manner. In this section, we will first introduce the algorithm we are based on: Value Iteration Networks (VINs) [17], and use it as foundation to build our algorithm: Symmetric VIN. Finally, we provide intuition to the theoretical guarantees on how we make use of symmetry.

D.2 Value Iteration Network: Background and Interpretation

Value Iteration Network (VIN) [17] is an example of a differentiable planning algorithm. It empirically found that, for 2D path planning, value iteration can be implemented by a deep convolution network.

Background: VIN. Value iteration is an instance of a dynamic programming (DP) method to solve Markov decision processes (MDPs). It iteratively applies the Bellman (optimality) operator until convergence, which is based on the following Bellman (optimality) equation:

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a)V(s'), \quad V(s) = \max_a Q(s,a) \tag{6}$$

Tamar et al. [17] used a convolution network to parameterize value iteration. It jointly learns in a latent MDP on 2D grid, which has the latent reward function $\bar{R} : \mathbb{Z}^2 \to \mathbb{R}^{|\mathcal{A}|}$ and value function $\bar{V} : \mathbb{Z}^2 \to \mathbb{R}$, and applies value iteration on that MDP:

$$\bar{Q}^{(k)}_{\bar{a},i',j'} = \bar{R}_{\bar{a},i,j} + \sum_{i,j} W^V_{\bar{a},i,j} \bar{V}^{(k-1)}_{i'-i,j'-j} = \bar{R}_{\bar{a},i,j} + \texttt{Conv2D}(\bar{V}; W^V), \quad \bar{V}^{(k)}_{i,j} = \max_{\bar{a}} \bar{Q}^{(k)}_{\bar{a},i',j'} \tag{7}$$

Later, we generalize the idea of VIN that (1) represents reward and value functions as fields on 2D grid, and (2) realizes value iteration by operations on the fields. Our final goal is to use VIN

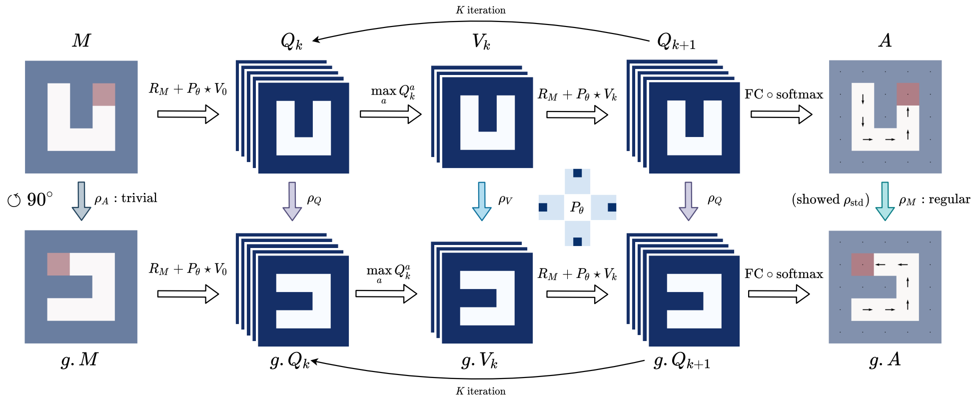

Figure 9: (This is a copy of Figure 3.) The commutative diagram of Symmetric Value Iteration Network (SymVIN). Every *row* is a full computation graph of VIN. Every *column* is to rotate field by ↻ 90°. The key message: if we *rotate* the map (from M to $g.M$), to guarantee the final policy function to also be *equivalently rotated* (from A to $g.A$), we shall guarantee every *transformation* (e.g., $Q_k \mapsto V_k$ and $V_k \mapsto Q_{k+1}$) in value iteration to also be *equivariant* ($g.f(x) = f(g.x)$, for every *pair of columns*).

to demonstrate a principled method for incorporating symmetry in differentiable planning. After reviewing the basics of VIN, we will next summarize the reasoning of choosing VIN.

D.3 Symmetric Value Iteration Network: A Practical Symmetric Planning Algorithm

Why do we choose VIN? There are two reasons behind the choice of VIN.

1. The expected value operator in value iteration $\sum_{s'} P(s'|s,a)V(s')$ is *linear* in value function. As we show in Theorem 4.1, it is also *equivariant* for (2D) path planning, this means that it is a *linear equivariant operator*. According to Cohen et al. [12], any linear equivariant operator (on spaces such as 2D grid) has one-to-one correspondence to a (group equivariant) convolution operator.
2. Value iteration, or Bellman (optimality) operator, is fully convolutional, i.e. only relies on operating on functions ("fields") over $\mathbb{Z}^2$, such as value function, reward function, and transition functions: $V_{k+1}(s) = \max_a R^a(s) + \gamma \times [P^a \star V_k](s)$. This enables us to inject symmetry by enforcing equivariance within convolution. For example, for a 2D map in Figure 1, the 4 corner states are symmetric under any one of the eight transformations in D_4, and we can enforce those 4 states to have the same value if we rotate or flip the map (D_4-equivariance). This avoids the need to find if a new state is symmetric to any existing state, which is shown to be NP-hard [8].

In summary, VIN satisfies both desiderata: (1) it uses convolution as the backbone, and (2) it operates on fields. Furthermore, we find VIN is empirically and conceptually the *simplest* differentiable planning algorithm that satisfies them, which leads to our decision.

How to inject symmetry? VIN uses a regular 2D convolutional network (Equation 7), which has *translation equivariance* [15, 13]. More concretely, a VIN will output the same value function for the same map patches that only differ by 2D translation. We omit how to characterize translation equivariance here, since it requires a different mechanism to handle and does not *decrease* the search space nor *reduce* a path planning MDP to an easier problem.

Beyond translation, we are more interested in *rotation* and *reflection* symmetries. Intuitively, as in Figure 1, if we find the optimal solution to a map, it automatically **generalizes** the solution to all 8 transformed maps (4 rotations times 2 reflections, including identity transformation). This can be characterized by *equivariance* of a planning algorithm `Plan`, such as value iteration `VI`, visualized in Figure 9: $g.\texttt{Plan}(M) = \texttt{Plan}(g.M)$, where M is a maze map, and g is the symmetry group D_4 under which 2D grid is invariant.

More importantly, symmetry also helps **training** of differentiable planning algorithms. Intuitively, symmetry in path planning poses additional constraint to its search space: if the goal is in the north, go up; if in the east, go right. In other words, the knowledge can be shared between symmetric cases, or the path planning is effectively reduced by symmetry to a smaller one. This property can also be depicted by equivariance of Bellman operators $\mathcal{T}$, or a step of value iteration: $g.\mathcal{T}[V_0] = \mathcal{T}[g.V_0]$. If we use $\texttt{VI}(M)$ to denote applying Bellman operators on arbitrary initialization until convergence $\mathcal{T}^\infty[V_0]$, value iteration is also equivariant, as demonstrated in Figure 9:

$$g.\texttt{VI}(M) \equiv g.\mathcal{T}^\infty[V_0] = \mathcal{T}^\infty[g.V_0] \equiv \texttt{VI}(g.M). \tag{8}$$

Thus, we inject equivariance into value iteration w.r.t. *rotation* and *reflection*, in addition to *translation*, through **steerable convolution (network)** from Cohen and Welling [14], which exactly matches our criteria. Cohen et al. [12] prove that steerable convolution is the most general linear equivariant map under some conditions, which value iteration satisfies. Weiler and Cesa [16] build $E(2)$-Steerable CNNs for 2D space, and we use their package `e2cnn` in our implementation. In practice, to inject symmetry into VIN, we simply need to replace the translation-equivariant `Conv2D` with `SteerableConv`:

$$\bar{Q}^{(k)}_{\bar{a},i',j'} = \bar{R}_{\bar{a},i,j} + \texttt{SteerableConv}(\bar{V}; W^V), \quad \bar{V}^{(k)}_{i,j} = \max_{\bar{a}} \bar{Q}^{(k)}_{\bar{a},i',j'}. \tag{9}$$

We formally justify our design in Section D.4 below and provide more technical details in Section 4.

D.4 Theoretical Justification: Why does it work?

In the Section D.3, we show how to exploit symmetry in path planning by equivariance from convolution via intuition. The goal of this new section is to (1) connect the theoretical justification with the algorithmic design, and (2) provide intuition for the justification. Even through we focus on a specific task, we hope that the underlying guidelines on integrating symmetry into planning are useful for broader planning algorithms and problems as well.

This version is for the purposes of rebuttal and preview, so we may refer some details to the original Section 4. We will consider further revision depending on how to form the method section.

Overview. There are numerous types of symmetry in various planning tasks. We study symmetry in **path planning** as an example, because it is a straightforward planning problem, and its solutions have been intensively studied in robotics and artificial intelligence [53, 1]. However, even for this problem, the symmetry has *not* been *effectively* exploited in its planning algorithms, such as Dijkstra's algorithm, A*, or RRT, because of NP-hard orbit finding [8]. Additionally, we focus on **value iteration** because it is both widely use and connects closely with convolution [14].

Theory: symmetry in planning. If we want to exploit symmetry in a task to improve planning, there are two major steps: (1) characterize the symmetry in the task, and (2) incorporate corresponding symmetry into the planning algorithm. The theoretical results in Section 4.2 mainly characterize the symmetry and direct us to a feasible planning algorithm.

The **symmetry in tasks** or MDPs can be specified by the equivariance property of the transition and reward function, studied in Ravindran and Barto [3], van der Pol et al. [32]:

$$\bar{P}(s' \mid s, a) = \bar{P}(g.s' \mid g.s, g.a), \quad \forall g \in G, \forall s, a, s' \tag{10}$$

$$\bar{R}_M(s, a) = \bar{R}_{g.M}(g.s, g.a), \quad \forall g \in G, \forall s, a \tag{11}$$

Note that how the group G acts on states and actions is decided by the space $\mathcal{S}$ or $\mathcal{A}$, which has been discussed in Equation 1 in Section 4.2. We emphasize that the equivariance property of the reward function is different from prior work [3, 32]: in our case, the reward function encodes obstacles as well, and thus depends on map input M. Intuitively, using Figure 1 as an example, if a position s is rotated $g.s$, to find how the correct original reward R, the input map M must also be rotated $g.M$. More details in Section 4.2 and Section H.

As for exploiting the **symmetry in planning algorithms**, we focus on value iteration and the VIN algorithm. We first prove in Theorem 4.1 that value iteration for path planning respects the *equivariance* property. This confirms that value iteration is a feasible method to incorporate symmetry. The next result in Theorem 4.2 further proves that value iteration is a general form of convolution (*steerable convolution*), motivating the use of steerable CNNs by Cohen and Welling [14] to replace regular CNNs in VIN.

x

Retrospect. We study how to inject symmetry into VIN for (2D) path planning, and expect the task-specific technical details are useful for two types of readers. *(i) Using VIN.* If one uses VIN for differentiable planning, the resulting algorithms SymVIN or SymGPPN can be a plug-in alternative, as a part in a larger end-to-end system. *(ii) Studying path planning.* The proposed framework characterizes the symmetry in (2D) path planning, so it is possible to apply the underlying ideas to other domains. For example, it is possible to extend to higher-dimensional continuous Euclidean spaces.

This concludes the section. We appreciate any feedback for this new simplified section on implementation and theory of symmetric planning. We will keep improving it and better integrate with the current "detailed" version in the future iterations.

E Experiments: Details and Additional Results (moved)

E.1 Details: Setup

Action space. Note that the MDP action space $\mathcal{A}$ needs to be *compatible* with the group action $G \times \mathcal{A} \to \mathcal{A}$. Since the E2CNN package [16] uses *counterclockwise* rotations as generators for rotation groups C_n, the action space needs to be *counterclockwise*.

Mapper training: manipulation. During training, we pre-train the mapper and the planner separately for 15 epochs. Where the mapper takes manipulator workspace and outputs configuration space. The mapper is trained to minimize the binary cross entropy between output and ground truth configurations space. The planner is trained in the same way as described in Section 6.1. After pre-training, we switch the input to the planner from ground truth configuration space to the one from the mapper. During testing, we follow the pipeline in [37] that the mapper-planner only have access to the manipulator workspace.

E.2 Details: Environments.

Manipulation. For planning in configuration space, the configuration space of the 2 DoFs manipulator has no constraints in the $\{0, \pi\}$ boundaries, i.e., no joint limits. To reflect this nature of the configuration space in manipulation tasks, we use circular padding before convolution operation. The circular padding is applied to convolution layers in VIN, SymVIN, ConvGPPN, and SymGPPN. Moreover, in GPPN, there is a convolution encoder before the LSTM layer. We add the circular padding in the convolution layers in GPPN as well.

In **2-DOF manipulation** in configuration space, we adopt the setting in [37] and train networks to take as input of configuration space, represented by two joints. We randomly generate 0 to 5 obstacles in the manipulator workspace. Then the 2 degree-of-freedom (DOF) configuration space is constructed from workspace and discretized into 2D grid with sizes $\{18, 36\}$, corresponding to bins of $20°$ and $10°$, respectively.

We allow each joint to rotate over 2π, so the configuration space of 2-DOF manipulation forms a torus $\mathbb{T}^2$. Thus, the both boundaries need to be connected when generating action demonstrations, and (equivariant) convolutions need to be circular (with padding mode) to wrap around for all methods. We allow each joint to rotate over 2π, so the both boundaries in configuration space need to be connected when generating action demonstrations, and (equivariant) convolutions need to be circular (with padding mode) to wrap around for all methods.

E.3 Details: Model Architecture

We try to mimic the setup in VIN and GPPN [18].

For non-SymPlan related parameters, we use learning rate of 10^{-3}, batch size of 32 if possible (GPPN variants need smaller), RMSprop optimizer.

For SymPlan parameters, we use 150 hidden channels (or 150 *trivial* representations for SymPlan methods) to process the input map. We use 100 hidden channels for Q-value for VIN (or 100 *regular* representations for SymVIN), and use 40 hidden channels for Q-value for GPPN and ConvGPPN (or 40 *regular* representations for SymGPPN on 15×15, and 20 for larger maps because of memory constraint).

E.4 Visualization of learned models

We visualize a trained VIN and a SymVIN, evaluated on a 15×15 map and its rotated version. For non-symmetric VIN in Figure 10, the learned policy is obviously not equivariant under rotation.

We also visualize SymVIN on larger map sizes: 28×28 and 50×50, to demonstrate its performance and equivariance.

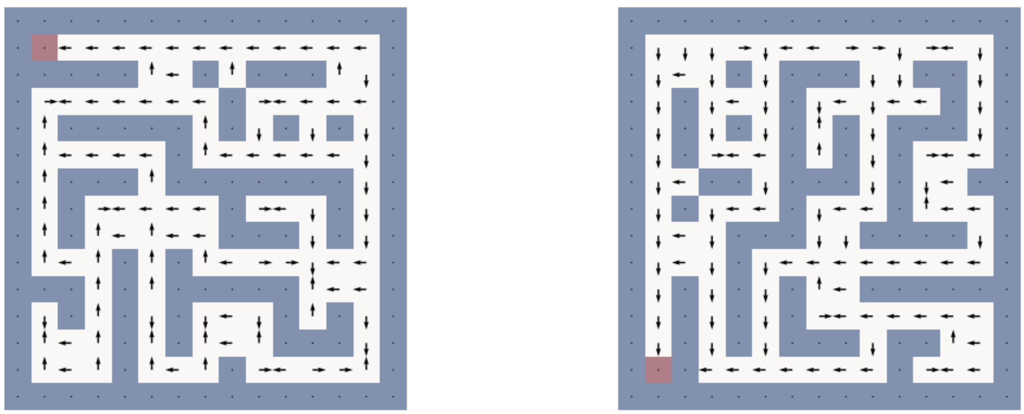

Figure 10: A trained VIN evaluated on a 15×15 map and its rotated version. It is obvious that the learned policy is not equivariant under rotation.

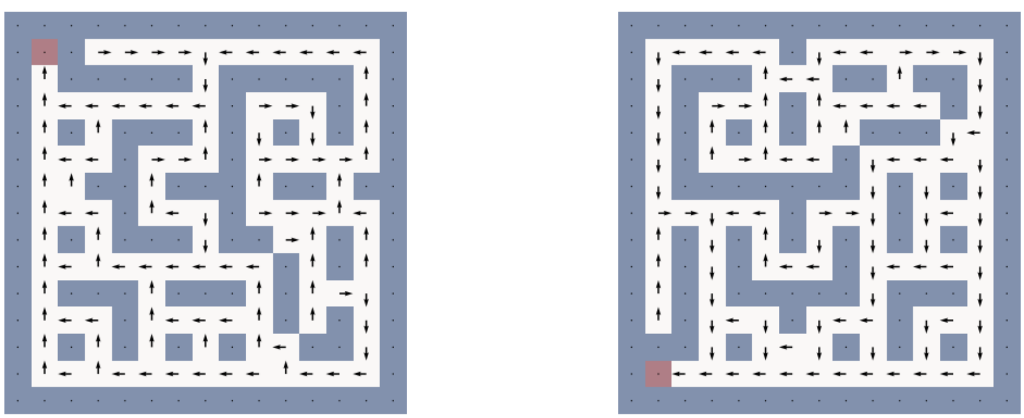

Figure 11: A trained SymVIN evaluated on a 15×15 map and its rotated version.

E.5 Further Analysis

Additional training curves. We also provide other training curves that we only show test numbers in the main text.

Training efficiency with less data. Since the supervision is still dense, we experiment on training with even smaller dataset to experiment in more extreme setup. We experiment how symmetry may affect the training efficiency of Symmetric Planners by further reducing the size of training dataset. We compare on two environments: 2D navigation and visual navigation, with training/validation/test size of 1K/200/200, for all methods.

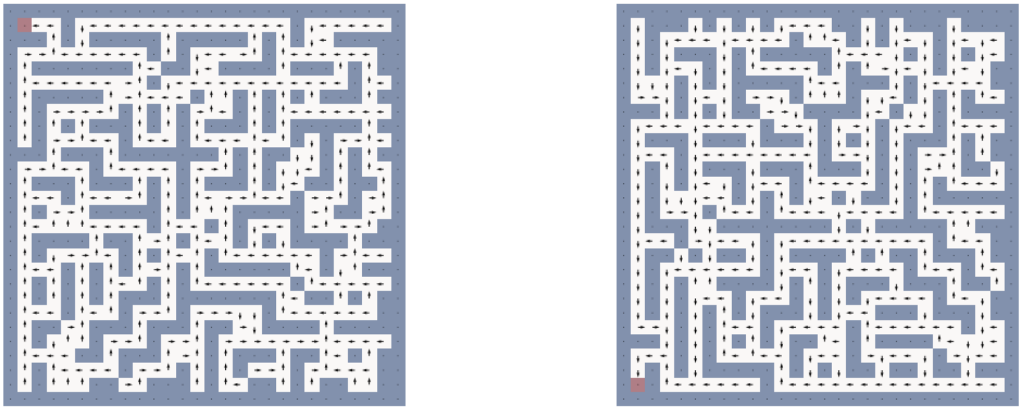

Figure 12: A fully trained SymVIN evaluated on a 28×28 map and its rotated version.

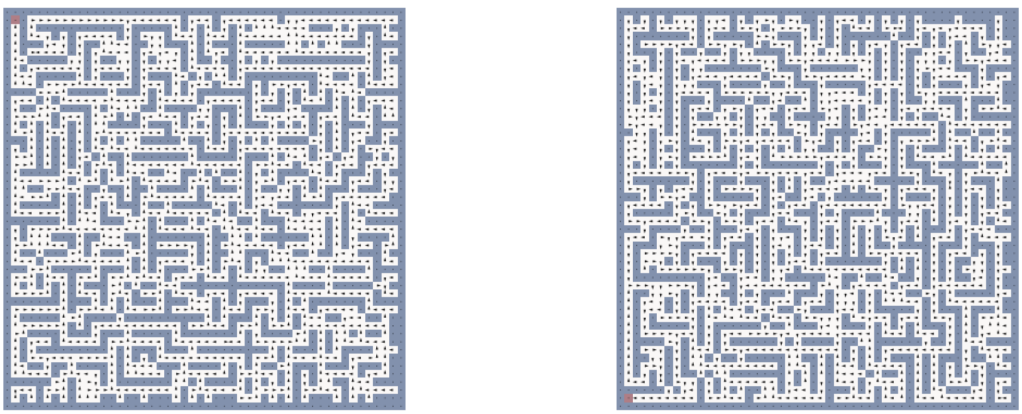

Figure 13: A fully trained SymVIN evaluated on a 50×50 map and its rotated version.

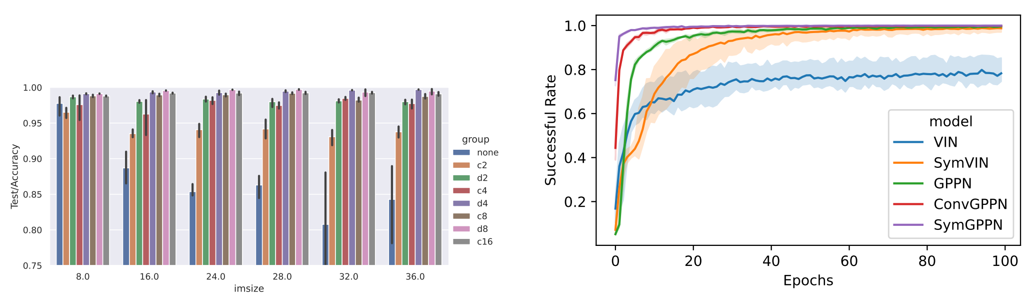

Figure 14: **(Left)** Accuracy evaluated on unseen test maps. The x-axis is the width of the map, and the y-axis is the accuracy, reported on every map size and every size and every chose symmetry group G. **(Right)** Visual navigation 15×15 with 10K data.

Choose of symmetry groups for navigation. One important benefit of partially equivariant network is that, we do not need to design the group representation of MDP action space $\rho_{\mathcal{A}}(g)$ for different group or action space. Thus, we experiment several G-equivariant variants with different group equivariance: (discrete rotation group) C_2, C_4, C_8, C_{16}, and (dihedral group) D_2, D_4, D_8, all based on $E(2)$-steerable CNN [16]. For all intermediate layers, we use regular representations $\rho_{\text{reg}}(g)$ of each group, followed by a final policy layer with non-equivariant 1×1 convolution.

The results are reported in the Figure 14 (left). We only compare VIN (denoted as "none" symmetry) against our $\mathrm{E}(2)$-VIN (other symmetry group option) on 2D navigation with 15×15 maps.

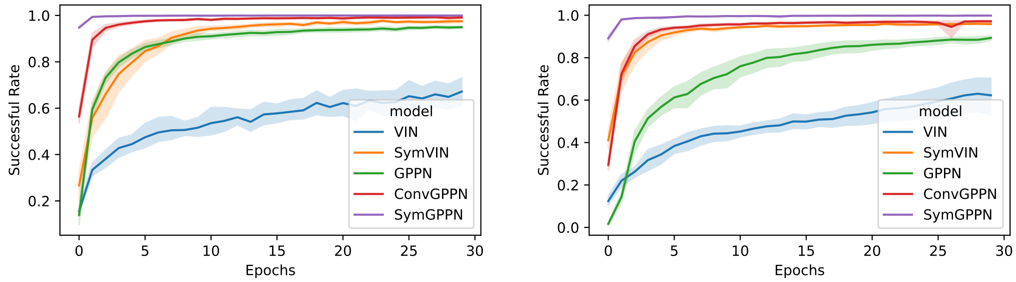

Figure 15: Training curves for **(Left)** 28×28 and **(Right)** 50×50.

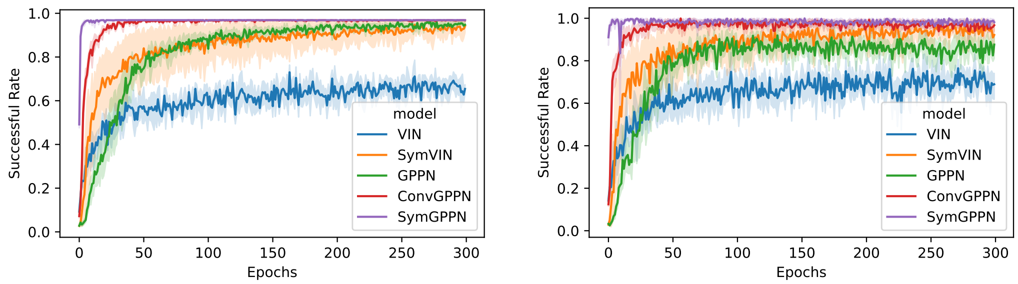

Figure 16: Training curves for 15×15 2D navigation 1K data **(Left)** training and **(Right)** validation successful rate.

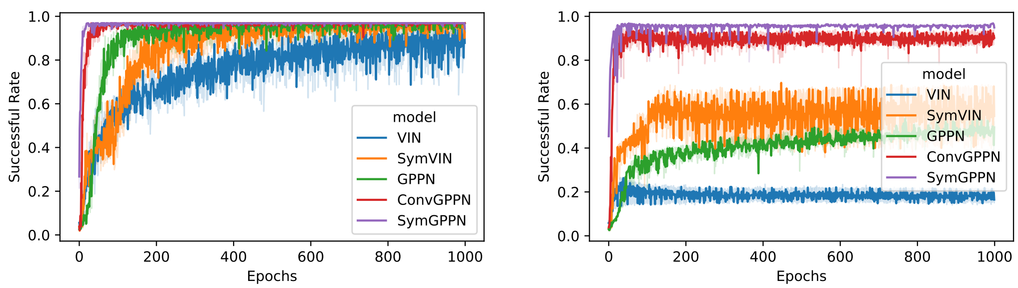

Figure 17: Training curves for 15×15 visual navigation 1K data **(Left)** training and **(Right)** validation successful rate.

Table 2: Fiber representations

(Fiber representation)	SymVIN
Default	98.45
Hidden: trivial to regular	99.07
State-value ρ_V: regular to trivial	63.08
Q-value ρ_Q: regular to trivial	21.30
ρ_Q and ρ_V: both trivial	2.814

In general, the planners equipped with any G group equivariance outperform the vanilla non-equivariant VIN, and D_4-equivariant steerable CNN performs the best on most map sizes. Additionally, since the environment has actions in 8 directions (4 diagonals), C_8 or D_8 groups seem to take advantage of that and have slightly higher accuracy on some map sizes, while C_{16} is over-constrained compared to the true symmetry $G = D_4$ and be detrimental to performance. The non-equivariant VIN also experiences higher variance on large maps.

Choosing fiber representations. As we use steerable convolutions [16] to build symmetric planners, we are free to choose the representations for feature fields, where intermediate equivariant convolutional layers will be equivariant between them $f(\rho_{\text{in}}(g)x) = \rho_{\text{out}}(g)f(x)$. We found representations for some feature fields are critical to the performance: mainly $V : \mathcal{S} \to \mathbb{R}$ and $Q : \mathcal{S} \to \mathbb{R}^{|A|}$.

We use the best setting as default, and ablate every option. As shown in Table 2, changing ρ_V or ρ_Q to trivial representation would result in much worse results.

Fully vs. Partially equivariance for symmetric planners. One seemingly minor but critical design choice in our SymPlan networks is the choice of the final policy layer, which maps Q-values $\mathcal{S} \to \mathbb{R}^{|\mathcal{A}|}$ to policy logits $\mathcal{S} \to \mathbb{R}^{|\mathcal{A}|}$. Fully equivariant is expected to perform better, but it has some points worth to mention. (1) We experience unstable training at the beginning, where the loss can go up to 10^6 in the first epoch, while we did not observe it in non-equivariant or partially equivariant counterparts. However, this only slightly affects training.

In summary, we found even though fully equivariant version can perform slightly better in the best tuned setting, on average setting, partially equivariant version is more robust and the gap is much larger, as shown in the follow table, which an example of averaging over three choices of representations introduced in the last paragraph. On average partially equivariant version is much better. In our experiments, partially equivariant version also is easier to tune.

Table 3: Fully vs. Partially equivariance

(Equivariance)	SymVIN
Partially equivariant averaged over all representations	91.04
Fully equivariant averaged over all representations	42.61

F Additional Discussion

F.1 Limitations and Extensions

Assumption on known domain structure. As in VIN, although the framework of steerable planning can potentially handle different domains, one important hidden assumption is that the underlying domain Ω (state space), is known. In other words, we fix the structure of learned transition kernels $p(s' \mid s, a)$ and estimate coefficients of it. One potential method is to use Transformers that learn attention weights to all states in $\mathcal{S}$, which has been partially explored in SPT [37]. Additionally, it is also possible to treat unknown MDPs as learned transition graphs, as explored in XLVIN [38]. We leave the consideration of symmetry in unknown underlying domains for future work.

The curse of dimensionality. The paradigm of steerable planning still requires full expansion in computing value iteration (opposite to *sampling-based*), since we realize the symmetric planner using group equivariant convolutions (essentially summation or integral). Convolutions on high-dimensional space could suffer from the curse of dimensionality for higher dimensional domains, and are vastly under-explored. This is a primary reason why we need sampling-based planning algorithms. If the domain (state-action transition graph) is sparsely connected, value iteration can still scale up to higher dimensions. It is also unclear either when steerable planning would fail, or how sampling-based algorithms could be integrated with the symmetric planning paradigm.

F.2 The considered symmetry in spatial MDPs

We need to differentiate between two types of symmetry in MDPs. Let's take spatial graph as illustrative example to understand the potential symmetry from a higher level, which means that the nodes $\mathcal{V}$ in the graph have spatial coordinates $\mathbb{Z}^n$ or $\mathbb{R}^n$. Our 2D path planning is a special case of spatial graph, where the actions can only move to adjacent spatial nodes.

Let the graph denoted as $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$. $\mathcal{E}$ is the set of edges connecting two states with an action. One type of symmetry is the symmetry of the graph itself. For the grid case, it means that after D_4 rotation or reflection, the map is unchanged.

Another type of symmetry comes from the isometries of the space. For a spatial graph, we can rotate it freely in a space, while the relative positions are unchanged. For our grid case, it is shown in the Figure 1 that rotating a map resulting in the rotated policy. However, the map or policy itself can never be equal under any transformation in D_4.

In other words, the first type is symmetry within a MDP (rely on the property of the MDP itself $\mathcal{M}$, or $\mathrm{Aut}(\mathcal{M})$), and the second type is symmetry between MDPs (only rely on the property of the underlying spatial space $\mathbb{Z}^2$, or $\mathrm{Aut}(\mathbb{Z}^2)$).

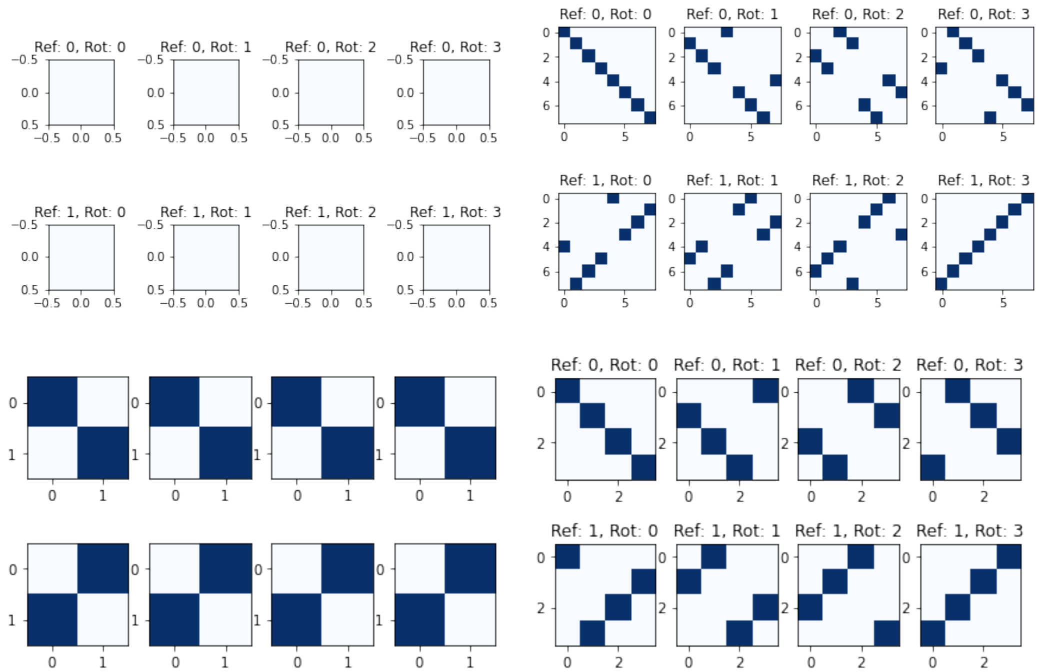

Figure 18: Visualization of the permutation representations of D_4 group for every element $g \in D_4$ (4 rotations each row and 2 reflections each column). They are (1) the trivial representation, (2) the regular representation, (3) the quotient representation (quotienting out *rotations*), (4) the quotient representation (quotienting out *reflections*).

Nevertheless, we could input map M and somehow treat symmetric states between MDPs as one state. See the proofs section for more details.

G Additional Background and Concepts

G.1 Group representations: visual understanding

A group representation is a (linear) group action that defines how a group acts on some space. Cohen and Welling [15, 14], Weiler and Cesa [16] provide more formal introduction to them in the context of equivariant neural networks. We provide visual understanding and refer the readers to them for comprehensive account.

To visually understand how the group D_4 acts on some vector space, we visualize the trivial, regular, and quotient (quotienting out reflections sr^2) representations, which are *permutation matrices*. If we apply such a representation $\rho(g)(g \in D_4)$ to a vector, the elements get *cyclically permuted*. See Figure 18.

The quotient representation that quotients out reflections and has dimension 4×4 is what we need to use on the 4-direction action space.

G.2 Geometric Deep Learning

We review another set of important concepts that motivate our formulation of steerable planning: geometric deep learning and the theories on connecting equivariance and convolution [11, 12, 13]. Bronstein et al. [11] use x for feature fields while Cohen and Welling [14], Cohen et al. [12], Weiler and Cesa [16] use f.

Convolutional feature fields. The signals are taken from set $\mathcal{C} = \mathbb{R}^D$ on some structured *domain* Ω, and all mappings from the domain to signals forms the space of $\mathcal{C}$-valued signals $\mathcal{X}(\Omega, \mathcal{C}) = \{f : \Omega \to \mathcal{C}\}$, or $\mathcal{X}(\Omega)$ for abbreviation. For instance, for RGB images, the domain is the 2D $n \times n$

grid $\Omega = \mathbb{Z}_n \times \mathbb{Z}_n$, and every pixel can take RGB values $\mathcal{C} = \mathbb{R}^3$ at each point in the domain $u \in \Omega$, represented by a mapping $x : \mathbb{Z}_n \times \mathbb{Z}_n \to \mathbb{R}^3$. A function on images thus operates on $3n^2$-dimensional inputs.

It is argued that the underlying geometric structure of domains Ω plays key role in alleviating the curse of dimensionality, such as convolution networks in computer vision, and this framework is named *Geometric Deep Learning*. We refer the readers to Geometric Deep Learning [11] for more details, and to more rigorous theories on the relation between equivariant maps and convolutions in [12] (vector fields through induced representations) and [13] (scalar fields through trivial representations).

Group convolution. Convolutions are shift-equivariant operations, and vice versa. This is the special case for $\Omega = \mathbb{R}$, which can be generalized to any group G (that we can integrate or sum over). The *group convolution* for signals on Ω is then defined[5] as

$$(f \star \psi)(g) = \langle f, \rho(g)\psi \rangle = \int_\Omega f(u)\psi(g^{-1}u)\mathrm{d}u, \tag{12}$$

where $\psi(u)$ is shifted copies of a filter, usually locally supported on a subset of Ω and padded outside. Note that although x takes $u \in \Omega$, the feature map $(x \star \psi)$ takes as input the elements $g \in G$ instead of points on the domain $u \in \Omega$. All following group convolution layers take G: $\mathcal{X}(G) \to \mathcal{X}(G)$. In the grid case, the domain Ω is *homogeneous* space of the group G, i.e. the group G acts transitively: for any two points $u, v \in \Omega$ there exists a symmetry $g \in G$ to reach $u = gv$.

Analogous to classic shift-equivariant convolutions, the generalized group convolution is G-equivariant [12]. It is observed that $\langle x, \rho(g)\theta \rangle = \langle \rho(g^{-1})x, \theta \rangle$, and from the defining property of group representations $\rho(h^{-1})\rho(g) = \rho(h^{-1}g)$, the G-equivariance of group convolution follows [11]:

$$(\rho(h)x \star \theta)(g) = \langle \rho(h)x, \rho(g)\theta \rangle = \left\langle x, \rho(h^{-1}g)\theta \right\rangle = \rho(h)(x \star \theta)(g) \tag{13}$$

Steerable convolution kernels. Steerable convolutions extend group convolutions to more general setup and decouple the computation cost with the group size [14, 56]. For example, $E(2)$-steerable CNNs [16] apply it for $E(2)$ group, which is semi-direct product of translations $\mathbb{R}^2$ and a fiber group H, where H is a group of transformations that fixes the origin and is $O(2)$ or its subgroups. The representation on the signals/fields is induced from a representation of the fiber group H. Use $\mathbb{R}^2$ as example, a steerable kernel only needs to be H-equivariant by satisfying the following constraint [16]:

$$\psi(hx) = \rho_{\text{out}}(h)\psi(x)\rho_{\text{in}}(h^{-1}) \quad \forall h \in H, x \in \mathbb{R}^2. \tag{14}$$

G.3 Steerable CNNs

We still use the running example on $\mathbb{Z}^2$ and group $p4m = \mathbb{Z}^2 \rtimes D_4$.

Induced representations. We follow [14, 12] to use π for *induced* representations. We still use feature fields over $\mathbb{Z}^2$ as example.

As shown in **Figure 2 middle**, to transform a feature field $f : \mathbb{Z}^2 \to \mathbb{R}^C$ on base $\mathbb{Z}^2$ with group $p4m = \mathbb{Z}^2 \rtimes D_4$, we need the *induced representation* [14, 12]. The induced representation in this case is denoted as $\pi(g) \triangleq \mathrm{ind}_{D_4}^{\mathbb{Z}^2 \rtimes D_4} \rho(g)$ (for all g), which means how the group action of D_4 transforms a feature field on $\mathbb{Z}^2 \rtimes D_4$.

It acts on the feature field with two parts: (1) on the base space $\mathbb{Z}^2$ and (2) on the fibers (feature channels $\mathbb{R}^C$) by fiber group $H = D_4$ [14, 16]. More specifically, applying a translation $t \in \mathbb{Z}^2$ and a transformation $r \in D_4$ to some field f, we get $\pi(tr)f$ [14, 16]:

$$f(x) \mapsto [\pi(tr)f](x) \triangleq \rho(r) \cdot \left[f\left((tr)^{-1}x\right)\right]. \tag{15}$$

[5] The definition of group convolution needs to assume that (1) signals $\mathcal{X}(\Omega)$ are in a Hilbert space (to define an inner product $\langle x, \theta \rangle = \int_\Omega x(u)\theta(u)\mathrm{d}u$) and (2) the group G is locally compact (so a Haar measure exists and "shift" of filter can be defined).

$\rho(r)$ is the fiber representation that transforms the fibers $\mathbb{R}^C$, and $(tr)^{-1}x$ finds the element before group action (or equivalently transforming the base space $\mathbb{Z}^2$). Thus, π only depends on the fiber representation ρ but not the latter part, thus named *induced representation* by ρ.

Steerable convolution vs. group convolution. The steerable convolution on $\mathbb{Z}^2$ The understanding of this point helps to understand how a group acts on various feature fields and the design of state space for path planning problems. We use the discrete group $p4 = \mathbb{Z}^2 \rtimes C_4$ as example, which consists of $\mathbb{Z}^2$ translations and $90°$ rotations. The only difference with $p4m$ is $p4$ does not have reflections.

The group convolution with filter ψ and signal x on grid (or $\mathbf{p} \in \mathbb{Z}^2$), which outputs signals (a function) on group $p4$

$$[\psi \star x](\mathbf{t}, r) := \sum_{\mathbf{p} \in \mathbb{Z}^2} \psi((\mathbf{t}, r)^{-1} \mathbf{p})\, x(\mathbf{p}). \tag{16}$$

A group G has a natural action on the functions over its elements; if $x : G \to \mathbb{R}$ and $g \in G$, the function $g.x$ is defined as $[g.x](h) := x(g^{-1} \cdot h)$.

For example: The group action of a rotation $r \in C_4$ on the space of functions over $p4$ is

$$[r.y](\mathbf{p}, s) := y(r^{-1}(\mathbf{p}, s)) = y(r^{-1}\mathbf{p}, r^{-1}s), \tag{17}$$

where $r^{-1}\mathbf{p}$ spatially rotates the pixels, $r^{-1}s$ cyclically permutes the 4 channels.

The G-space (functions over $p4$) with a natural action of $p4$ on it:

$$[(\mathbf{t}, r).y](\mathbf{p}, s) := y((\mathbf{t}, r)^{-1} \cdot (\mathbf{p}, s)) = y(r^{-1}(\mathbf{p} - \mathbf{t}), r^{-1}s) \tag{18}$$

The group convolution in discrete case is defined as

$$[\psi \star x](g) := \sum_{h \in H} \psi(g^{-1} \cdot h)\, x(h). \tag{19}$$

The group convolution with filter ψ and signal x on $p4$ group is given by:

$$[\psi \star x](\mathbf{t}, r) := \sum_{s \in C_4} \sum_{\mathbf{p} \in \mathbb{Z}^2} \psi((\mathbf{t}, r)^{-1}(\mathbf{p}, s))\, x(\mathbf{p}, s). \tag{20}$$

Using the fact

$$\psi((\mathbf{t}, r)^{-1}(\mathbf{p}, s)) = \psi(r^{-1}(\mathbf{p} - \mathbf{t}, s)) = [r.\psi](\mathbf{p} - \mathbf{t}, s), \tag{21}$$

the convolution can be equivalently written into

$$[\psi \star x](\mathbf{t}, r) := \sum_{s \in C_4} \left(\sum_{\mathbf{p} \in \mathbb{Z}^2} [r.\psi](\mathbf{p} - \mathbf{t}, s)\, x(\mathbf{p}, s) \right). \tag{22}$$

So $\left(\sum_{\mathbf{p} \in \mathbb{Z}^2} [r.\psi](\mathbf{p} - \mathbf{t}, s)\, x(\mathbf{p}, s)\right)$ can be implemented in usual shift-equivariant convolution CONV2D.

The inner sum $\sum_{\mathbf{p} \in \mathbb{Z}^2}$ is equivalently for the sum in steerable convolution, and the outer sum $\sum_{s \in C_4}$ implement rotation-equivariant convolution that satisfies H-steerability kernel constraint. Here, the outer sum is essentially using the *regular* fiber representation of C_4.

In other words, group convolution on $p4 = \mathbb{Z}^2 \rtimes C_4$ group is equivalent to steerable convolution on base space $\mathbb{Z}^2$ with the fiber group of C_4 with regular representation.

Stack of feature fields. Analogous to ordinary CNNs, a feature space in steerable CNNs can consist of multiple feature fields $f_i : \mathbb{Z}^2 \to \mathbb{R}^{c_i}$. The feature fields are stacked $f = \bigoplus_i f_i$ together by concatenating the individual feature fields f_i (along the fiber channel), which transforms under the directly sum $\rho = \bigoplus_i \rho_i$ of individual (fiber) representations. Every layer will be equivariant between input and output field $f_{\text{in}}, f_{\text{out}}$ under induced representations $\pi_{\text{in}}, \pi_{\text{out}}$. For a steerable convolution between more than one-dimensional feature fields, the kernel is matrix-valued [12, 16].

[5] Technically, we still need to solve the linear equivariance constraint in Eq. 35 to enable weight-sharing for equivariance, while Weiler and Cesa [16] have implemented it for 2D case.

H Symmetric Planning Framework: Additional Details

H.1 Path planning in neural networks

We provide the detailed construction of doing path planning in neural networks in the Section 4. This further explains the visualization in Figure 2 left.

We use the running example of planning on the 2D grid $\mathbb{Z}^2$. We aim to understand (1) how VIN-style networks embed planning and how its idea generalizes, (2) how is symmetry structure defined in path planning and how could it be injected into such planning networks. Recall that we aim to understand (1) how VIN-style networks embed planning and how its idea generalizes, (2) how is symmetry structure defined in path planning and how could it be injected into such planning networks.

Path planning as MDPs. To answer the above two questions, we first need to understand how a VIN embeds a path planning problem into a convolutional network as some embedded MDP. Intuitively, the embedded MDP in a VIN is different from the original path planning problem, since (planar) convolutions are translation equivariant but there are different obstacles in different regions.

For path planning on the 2D grid $\mathcal{S} = \mathbb{Z}^2$, the objective is to avoid some obstacle region $\mathcal{C}_{\text{obs}} \subset \mathbb{Z}^2$ and navigate to the goal region $\mathcal{C}_{\text{goal}}$ through free space $\mathcal{C} \backslash \mathcal{C}_{\text{obs}}$. An action $a = \Delta s \in \mathcal{A}$ is to move from the current state s to a next *free* state $s' = s + \Delta s$, where for now we limit it to be in four directions: $\mathcal{A} =$. Assuming deterministic transition, the agent moves to s' with probability 1 if $s + \Delta s \in \mathcal{C} \backslash \mathcal{C}_{\text{obs}}$. If it hits an obstacle, it stays at s if $s + \Delta s \in \mathcal{C}_{\text{obs}}$: $P\left(s + \Delta s \mid s, \Delta s\right) = 0$ and $P\left(s \mid s, \Delta s\right) = 1$. Every move has a constant negative reward $R(s, a) = -1$ to encourage shortest path. We call this *ground* path planning MDP, a 5-tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$.

Constructing embedded MDPs. However, such transition function is not translation-invariant, i.e. at different position, the transition probabilities are not related by any symmetry: $P(s'|s, a) \neq P(g.s'|g.s, g.a)$. Instead, we could always construct a "symmetric" MDP that has equivalent optimal value and policy for path planning problems, which is implicitly realized in VINs. The idea is to move the information of obstacles from transition function to reward function: when we hit some action $s + \Delta s \in \mathcal{C}_{\text{obs}}$, we instead allow transition $\bar{P}\left(s + \Delta s \mid s, \Delta s\right) = 1$ (with all other s' as 0 probability) while set a "trap" with negative infinity reward $\bar{R}_m\left(s, \Delta s\right) = -\infty$. The reward function needs the information from the occupancy map M, indicating obstacles $\mathcal{C}_{\text{obs}}$ and free space. For the free region, the reward is still a constant $\bar{R}_M\left(s, \Delta s\right) = -1$, indicating the cost of movement.

We call it the *embedded* MDP, with different transition and reward function $\bar{\mathcal{M}} = \langle \mathcal{S}, \mathcal{A}, \bar{P}, \bar{R}_M, \gamma \rangle$, which converts the “complexity” in the transition function P in $\mathcal{M}$ to the reward function $\bar{R}_m$ in $\bar{\mathcal{M}}$. Here, map M shall also be treated as an “input”, thus later we will derive how the group acts on the map $g.M$. It has the same optimal policy and value as the ground MDP $\mathcal{M}$, since the optimal policies in both MDPs will avoid obstacles in $\mathcal{M}$ or trap cells in $\bar{\mathcal{M}}$. It could be easily verified by simulating value iteration backward in time from the goal position.

The transition probability $\bar{P}$ of the embedded MDP $\bar{\mathcal{M}}$ is for an “empty” maze and thus translation-invariant. Note that the reward function $\bar{R}$ is not not necessarily invariant. This construction is not limited to 2D grid and generalizes to continuous state space or even higher dimensional space, such as $\mathbb{R}^6$ configuration space for 6-DOF manipulation.

Note, all of this is what we use to conceptually understand how a VIN is possible to learn. The reward cannot be negative infinity, but the network will learn it to be smaller than all desired Q-values.

H.2 Understanding steerable planning

How do we deal with potential symmetry in path planning? how do we characterize it? We try to understand symmetric planning (steerable planning after integrating symmetry with equivariance) and how it is difference classic planning algorithms, such as A*, for planning under *symmetry*.

[5] We avoid the symbol π for policy since it is used for induced representation in [14, 16].

Steerable planning. Recall that we generalize the idea of VIN by considering it as a planning network that composes of mappings between steerable feature fields.

The critical point is that, convolutions directly operate on local patches of pixels and never directly touch coordinates of pixels. In analogy, this avoids a critical drawback in other *explicit* planning algorithms: in sampling-based planning, a trajectory $(s_1, a_1, s_2, a_2, \ldots)$ is sampled and inevitable represented by states $\Omega = \mathcal{S}$. However, to find another symmetric state $g.s$, we potentially need to compare it against all known states $\mathcal{S}' \subset \mathcal{S}$ with all symmetries $g \in G$. On high level, an implicit planner can avoid such symmetry breaking and is more easily compatible with symmetry by using equivariant constraints.

We can use MDP homomorphism to understand this [3, 32].

MDP homomorphisms. An *MDP homomorphism* $h : \mathcal{M} \to \overline{\mathcal{M}}$ is a mapping from one MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$ to another $\overline{\mathcal{M}} = \langle \overline{\mathcal{S}}, \overline{\mathcal{A}}, \overline{P}, \overline{R}, \gamma \rangle$ [3, 32]. h consists of a tuple of surjective maps $h = \langle \phi, \{\alpha_s \mid s \in \mathcal{S}\} \rangle$, where $\phi : \mathcal{S} \to \overline{\mathcal{S}}$ is the state mapping and $\alpha_s : \mathcal{A} \to \overline{\mathcal{A}}$ is the *state-dependent* action mapping. The mappings are constructed to satisfy the following conditions:

$$\begin{aligned} \overline{R}\left(\phi(s), \alpha_s(a)\right) &\triangleq R(s, a) \ , \\ \overline{P}\left(\phi\left(s'\right) \mid \phi(s), \alpha_s(a)\right) &\triangleq \sum_{s'' \in \phi^{-1}(\phi(s'))} P\left(s'' \mid s, a\right) \ , \end{aligned} \tag{23}$$

for all $s, s' \in \mathcal{S}$ and for all $a \in \mathcal{A}$.

We call the *reduced* MDP $\overline{\mathcal{M}}$ the *homomorphic image* of $\mathcal{M}$ under h. If $h = \langle \phi, \{\alpha_s \mid s \in \mathcal{S}\} \rangle$ has *bijective* maps ϕ and $\{\alpha_s\}$, we call h an *MDP isomorphism*. Given MDP homomorphism h, (s, a) and (s', a') are said to be h-equivariant if $\sigma(s) = \sigma\left(s'\right)$ and $\alpha_s(a) = \alpha_{s'}\left(a'\right)$.

Symmetry-induced MDP homomorphisms. Given group G, an MDP homomorphism h is said to be *group structured* if any state-action pair (s, a) and its transformed counterpart $g.(s, a)$ are mapped to the same abstract state-action pair: $(\phi(s), \alpha_s(a)) = (\phi(g.s), \alpha_{g.s}(g.a))$, for all $s \in \mathcal{S}, a \in \mathcal{A}, g \in G$. For convenience, we denote $g.(s, a)$ as $(g.s, g.a)$, where $g.a$ implicitly[6] depends on state s. Applied to the transition and reward functions, the transition function P is G-invariant if P satisfies $P(g.s'|g.s, g.a) = P(s'|s, a)$, and reward function R is G-invariant if $R(g.s, g.a) = R(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}, g \in G$.

However, this only fits the type of symmetry in [9, 10]. And also, they cannot handle invariance to translation $\mathbb{Z}^2$. In our case, we need to augment the reward function with map M input:

$$R_{g.M}(g.s, g.a) = R_M(s, a), \tag{24}$$

for all $s \in \mathcal{S}, a \in \mathcal{A}, g \in G = p4m$.

This means that, at least for rotations and reflections D_4, the MDPs constructed from transformed maps $\{g.M\}$ are MDP *isomorphic* to each other.

I Symmetric Planning Framework: Proofs

We show the derivation and proofs for all theoretical results in this section.

We follow the notation in [12] to use $\star$ for (one-argument) convolution and $\cdot$ for (two-argument) multiplication:

$$E^a[V](s) = [P^a \cdot V](s) \equiv \sum_{s'} P^a\left(s' \mid s\right) \cdot V(s') \tag{25}$$

I.1 Proof: equivariance of scalar-valued expected value operation

We present the Theorem 4.1 here and its formal definition.

[6] The group operation acting on action space $\mathcal{A}$ *depends on state*, since G actually acts on the *product space* $\mathcal{S} \times \mathcal{A}$: $(g, (s, a)) \mapsto g.(s, a)$, while we denote it as $(g.s, g.a)$ for consistency with $h = \langle \phi, \{\alpha_s \mid s \in \mathcal{S}\} \rangle$. As a bibliographical note, in van der Pol et al. [32], the group acting on state and action space is denoted as state transformation $L_g : \mathcal{S} \to \mathcal{S}$ and *state-dependent* action transformation $K_g^s : \mathcal{A} \to \mathcal{A}$.

Theorem I.1. *If transition is G-invariant, the expected value operator E over $\mathbb{Z}^2$ is G-equivariant:*

$$[g.E^a[V]]\,(s) = [E^{g.a}[g.V]]\,(s), \quad \textit{for all } g = tr \in \mathbb{Z}^2 \rtimes D_4.$$

Proof. E is the expected value operator. We also write the transition probability as

Recall the G-invariance condition of transition probability, the group element g acts on s, a, s':

$$\bar{P}(s' \mid s, a) = \bar{P}(g.s' \mid g.s, g.a) \equiv \bar{P}\left((tr).s' \mid (tr).s, r.a\right), \quad \forall g = tr \in \mathbb{Z}^2 \rtimes D_4, \forall s, a, s', \tag{26}$$

where we can uniquely decompose any $g \in \mathbb{Z}^2 \rtimes D_4$ as $t \in \mathbb{Z}^2$ and $r \in D_4$ [14]. Note that, since the action is the difference between states $a = \Delta s = s' - s$, the translation part t acts trivially on it, so $g.a = (tr).a = r.a$ for all $r \in D_4$.

We transform the feature field and show its equivariance:

$$\begin{aligned}
[g.E^a[V]](s) &\equiv [g.[P^a \cdot V]](s) && (27)\\
&\equiv \sum_{s'} \rho_{\text{triv}}(r) P^a\left(s' \mid (tr)^{-1}.s\right) \cdot V(s') && (28)\\
&= \sum_{s'} \rho_{\text{triv}}(r) P^{r.a}\left((tr).s' \mid s\right) \cdot V(s') && (29)\\
&= \sum_{\tilde{s}'} \rho_{\text{triv}}(r) P^{r.a}\left(\tilde{s}' \mid s\right) \cdot V\left((tr)^{-1}\tilde{s}'\right) && (30)\\
&= \sum_{\tilde{s}'} P^{r.a}\left(\tilde{s}' \mid s\right) \cdot \rho_{\text{triv}}(r) V\left((tr)^{-1}\tilde{s}'\right) && (31)\\
&\equiv [P^{r.a} \cdot [g.V]](s) && (32)\\
&\equiv [E^{r.a}[g.V]](s). && (33)
\end{aligned}$$

We use the trivial representation $\rho_{\text{triv}}(g) = \mathrm{Id}_{1\times 1} = 1$ to emphasize that (1) the group element g acts on *feature fields* P^a and V, and (2) both feature fields P^a and V are scalar-valued and correspond to the one-dimensional trivial representation of $r \in D_4$.

In the third line, we use the G-invariance of transition probability.

The fourth line uses substitution $\tilde{s}' \triangleq (tr).s'$, for all $s' \in \mathbb{Z}^2$ and $tr \in \mathbb{Z}^2 \rtimes D_4$. This is an one-to-one mapping and the summation does does not change.

$\square$

I.2 Proof: *expected value operator* as steerable convolution

In this section, we derive how to cast expected value operator as steerable convolution. The equivariance proof is in the next section.

In Theorem 4.1, we show equivariance of value iteration in 2D path planning, while it is only for the case that feature fields P^a and V are scalar-valued and correspond to one-dimensional trivial representation of $r \in D_4$.

Here, we provide the derivation for Theorem 4.2 show that steerable CNNs [14] can achieve value iteration since we could construct the G-invariant transition probability as a steerable convolutional kernel. This generalizes Theorem 4.1 from scalar-valued kernel (for transition probability) with trivial representation to matrix-valued kernel with any combination of representations, enabling using stack (direct-sum) of feature fields and representations.

We state Theorem 4.2 here for completeness:

Theorem I.2. *If transition is G-invariant, there exists a (one-argument, isotropic) matrix-valued steerable kernel $P^a(s - s')$ (for every action), such that the expected value operator can be written as a steerable convolution and is G-equivariant:*

$$E^a[V] = P^a \star V, \quad [g.[P^a \star V]](s) = [P^{g.a} \star [g.V]](s), \quad \forall s \in \mathbb{Z}^2, \forall g \in \mathbb{Z}^2 \rtimes D_4. \tag{34}$$

Steerable kernels. In our earlier definition, ψ^a and f_{in} are transition probability and value function, which are both real-valued $\psi^a : \mathbb{Z}^2 \to \mathbb{R}, f_{\text{in}} : \mathbb{Z}^2 \to \mathbb{R}$. However, this is a *special case* which corresponds to use one-dimensional *trivial representation* of the fiber group D_4. In the general case in steerable CNNs [14, 16], we can choose the feature fields $\psi^a : \mathbb{Z}^2 \to \mathbb{R}^{C_{\text{out}} \times C_{\text{in}}}$ and $f_{\text{in}} : \mathbb{Z}^2 \to \mathbb{R}^{C_{\text{in}}}$ and their fiber representations, which we will introduce the group representations of D_4 and how to choose in practice in the next section.

Weiler et al. [54] show that *convolutions* with *steerable kernels* $\psi^a : \mathbb{Z}^2 \to \mathbb{R}^{C_{\text{out}} \times C_{\text{in}}}$ is the most general *equivariant linear map* between steerable feature space, transforming under ρ_{in} and ρ_{out}. In analogy to the continuous version[7] in [16], the convolution is equivariant *iff* the kernel satisfies a H-steerability kernel constraint:

$$\psi^a(hs) = \rho_{\text{out}}(h)\psi^a(s)\rho_{\text{in}}(h^{-1}) \quad h \in H = D_4, s \in \mathbb{Z}^2. \tag{35}$$

Expected value operation as steerable convolution. The foremost step is to show that the expected value operation is a form of convolution and is also G-equivariant. By definition, if we want to write a (linear) operator as a form of convolution, we need one-argument kernel. Cohen et al. [12] show that every linear equivariant operator is some convolution and provide more details. For our case, this is formally shown as follows.

Proposition I.3. *If the transition probability is G-invariant, it can be expressed as an (one-argument) kernel $P^a(s'|s) = P^a(s' - s)$ that only depends on the difference $s' - s$.*

Proof. The form of our proof is similar to [12], while its direction is different from us. We construct a MDP such that the transition probability kernel is G-invariant, while Cohen et al. [12] assume the linear operator $\psi \cdot f$ is linear *equivariant* operator on a homogeneous space, and then derive that the kernel is G-invariant and expressible as one-argument kernel. Additionally, our kernel $\psi^a(s, s')$ and $\psi^a(s - s')$ both live on the base space $B = \mathbb{Z}^2$ but not on the group $G = \mathbb{Z}^2 \rtimes D_4$.

We show that the transition probability only depends on the difference $\Delta s = s' - s$, so we can define the two-argument kernel $P^a(s'|s)$ on $\mathcal{S} \times \mathcal{S}$ by an one-argument kernel $P^a(s' - s)$ (for every action a) on $\mathcal{S} = \mathbb{Z}^2$, without loss of generality:

$$\begin{aligned}
P^a(s' - s) &\equiv P^a(\mathbf{0}, s' - s) && (36)\\
&= P^{g.a}(g.\mathbf{0}, g.(s' - s)) && (37)\\
&= P^{r.a}((rs).\mathbf{0}, (rs).(s' - s)) && (38)\\
&= P^{r.a}(r.s, r.(s' - s + s)) && (39)\\
&= P^{r.a}(r.s, r.s') && (40)\\
&= P^a(s, s'), && (41)
\end{aligned}$$

where the second step uses G-invariance with $g = sr$, understood as the composition of a translation $s \in \mathbb{Z}^2$ and a transformation in $r \in D_4$.

$\square$

Additionally, we can also derive that, for the one-argument kernel, if we rotate state difference $r.(s' - s)$, the probability is the same for rotated action $r.a$.

$$P^a(s' - s) = P^{r.a}(r.(s' - s)), \text{for all } r \in D_4, s, s' \in \mathbb{Z}^2 \tag{42}$$

The *expected value operator* with two-argument kernel can be then written as

$$E[V](s) \equiv [P^a \cdot V](s) = \sum_{s'} P^a(s'|s)V(s') = \sum_{s'} P^a(s' - s)V(s') \equiv [P^a \star V](s). \tag{43}$$

Note that we do not differentiate between cross-correlation $(s' - s)$ and convolution $(s - s')$.

[7] Weiler and Cesa [16] use letter G to denote the stabilizer subgroup $H \leq \mathrm{O}(2)$ of $\mathrm{E}(2)$.

I.3 Proof: equivariance of *expected future value*

Our derivation follows the existing work on group convolution and steerable convolution networks [15, 14, 16, 12]. However, the goal of providing the proof is not just for completeness, but instead to emphasize the close connection between how we formulate our planning problem and the literature of steerable CNNs, which explains and justifies our formulation.

Additionally, there are several subtle differences worth to mention. (1) Throughout the paper, we do not discuss kernels or fields that live on a group G to make it more approachable. Nevertheless, group convolutions are a special case of steerable convolutions with fiber representation ρ as regular representation. (2) We use $\mathbb{Z}^2$ as running example. Some prior work uses $\mathbb{R}^2$ or $\mathbb{Z}^2$, but they are merely just differ in integral and summation. (3) The definition of convolution and cross-correlation might be defined and used interchangeably in the literature of (equivariant) CNNs.

Notation. To keep notation clear and consistent with the literature [14, 12, 16], we denote the transition probability $\bar{P}(s'|s,a) \triangleq \psi^a(s,s') \in \mathbb{R}$ (one kernel for an action) and value function as $V(s') \triangleq f_{\text{in}}(s') \in \mathbb{R}$, and the resulting expected value as $f^a_{\text{out}}(s) = \sum_{s'} \psi^a(s,s') f_{\text{in}}(s')$ (given a specific action a).

Transformation laws: induced representation. For some group acting on the base space $\mathbb{Z}^2$, the signals $f : \mathbb{Z}^2 \to \mathbb{R}^c$ are transformed like [14]:

$$[\pi(g)f](x) = f(g^{-1}x) \tag{44}$$

Apply a translation t and a transformation $r \in D_4$ to f, we get $\pi(tr)f$. The transformation law on the input space f_{in} is [14, 16]:

$$f(x) \mapsto [\pi(tr)f](x) \triangleq \rho(r) \cdot \left[f\left((tr)^{-1}x\right)\right] \tag{45}$$

The transformation law of the output space after applying π_{in} on input f_{in} is given by [14]:

$$[\psi \star f](x) \mapsto [\psi \star [\pi(tr)f]](x) \triangleq \rho(r) \cdot \left[[\psi \star f]\left((tr)^{-1}x\right)\right]. \tag{46}$$

In our case, the output space is $f^a_{\text{out}} : \mathbb{Z}^2 \to \mathbb{R}^{C_{\text{out}}}$ and the input space is $f_{\text{in}} : \mathbb{Z}^2 \to \mathbb{R}^{C_{\text{in}}}$. Intuitively, if we rotate a vector field (fibers represent arrows) by the induced representation $\pi(tr)$ of f, we also need to rotate the direction of arrows by $\rho(r), r \in D_4$.

Equivariance. Now we prove the steerable convolution is equivariant:

$$[\psi^a \star [\pi_{\text{in}}(g) f_{\text{in}}]](s) = [\pi_{\text{out}}(g) f^a_{\text{out}}](s) \quad \forall s \in \mathcal{S}, \forall g \in G. \tag{47}$$

The induced representation of input field f_{in} is induced by the fiber representation ρ_{in}, expressed by $\pi_{\text{in}} \triangleq \operatorname{ind}_H^G \rho_{\text{in}} = \operatorname{ind}_{D_4}^{\mathbb{Z}^2 \rtimes D_4} \rho_{\text{in}}$, where ρ_{in} is the fiber representation of group $H = D_4$. The induced representation of output field π_{out} is analogously from ρ_{out}.

Weiler and Cesa [16] proved equivariance of steerable convolutions for $\mathbb{R}^2$ case, while we include the proof under our setup for completeness. The definition in [16] uses a form of *cross-correlation* and we use *convolution*, while it is usually referred to interchangeably in the literature and is equivalent. Cohen and Welling [14], Weiler et al. [54], Weiler and Cesa [16], Cohen et al. [12], Cohen [56] provide more details and we refer the readers to them for more comprehensive account.

The convolution on discrete grids $\mathbb{Z}^2$ with input field f_{in} transformed by the induced representation π_{in} gives:

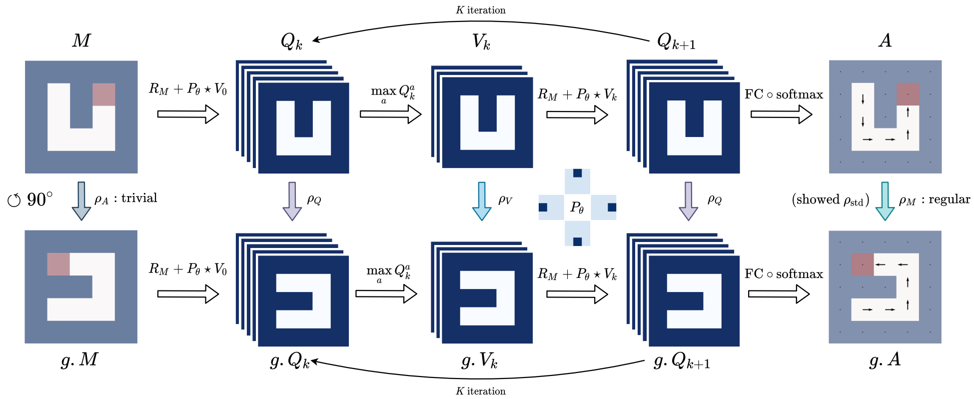

Figure 19: *We attach a copy of the commutative diagram of SymVIN to show the equivariance of steerable value iteration.* Commutative diagram for the full pipeline of SymVIN on steerable feature fields over $\mathbb{Z}^2$ (every grid). If rotating the input map M by $\pi_M(g)$ of any g, the output action $A = \texttt{SymVIN}(M)$ is guaranteed to be transformed by $\pi_A(g)$, i.e. the entire steerable SymVIN is equivariant under induced representations π_M and π_A: $\texttt{SymVIN}(\pi_M(g)M) = \pi_A(g)\texttt{SymVIN}(M)$. We use stacked feature fields to emphasize that SymVIN supports direct-sum of representations beyond scalar-valued.

$$
\begin{aligned}
[\psi^a \star [\pi_{\text{in}}(rt) f_{\text{in}}]](s) &= \sum_{s' \in \mathbb{Z}^2} \psi^a(s - s')[\pi_{\text{in}}(rt) f_{\text{in}}](s') \\
&= \sum_{s' \in \mathbb{Z}^2} \psi^a(s - s') \rho_{\text{in}}(r) f_{\text{in}}(r^{-1}(s' - t)) \\
&= \sum_{s' \in \mathbb{Z}^2} \rho_{\text{out}}(r) \psi^a(r^{-1}(s - s')) \rho_{\text{in}}(r)^{-1} \rho_{\text{in}}(r) f_{\text{in}}(r^{-1}(s' - t)) \\
&= \rho_{\text{out}}(r) \sum_{s' \in \mathbb{Z}^2} \psi^a(r^{-1}(s - s')) f_{\text{in}}(r^{-1}(s' - t)) \\
&= \rho_{\text{out}}(r) \sum_{\tilde{s} \in \mathbb{Z}^2} \psi^a(r^{-1}(s - t) - \tilde{s}) f_{\text{in}}(\tilde{s}) \\
&= \rho_{\text{out}}(r) f_{\text{out}}(r^{-1}(s - t)) \\
&= \left[\pi_{\text{out}}(rt) f_{\text{out}}^a\right](s),
\end{aligned}
\tag{48}
$$

where $s' \in \mathcal{S} = \mathbb{Z}^2$, and thus satisfies the equivariance condition:

$$
\left[\psi^a \star [\pi_{\text{in}}(rt) f_{\text{in}}]\right](s) = \left[\pi_{\text{out}}(rt) f_{\text{out}}^a\right](s), \forall s \in \mathbb{Z}^2, \forall rt \in \mathbb{Z}^2 \rtimes D_4. \tag{49}
$$

1. Definition of $\star$
2. Transformation law of the induced representation π_{in} [14, 16]
3. Kernel steerability $\psi^a(s) = \rho_{\text{out}}(h)\psi^a(h^{-1}s)\rho_{\text{in}}(h^{-1})$ [16]
4. Move and cancel
5. Substitutes $\tilde{s} = r^{-1}(s' - t)$, $r^{-1}s' = r^{-1}t + \tilde{s}$, so $r^{-1}(s - s') = r^{-1}(s - t) - \tilde{s}$. Since $r \in D_4$ and $s - s' \in \mathbb{Z}^2$, the result is still in $p4m$, it is one-to-one correspondence $p4m \times \mathbb{Z}^2 \to \mathbb{Z}^2$, and the summation does not change. Weiler and Cesa [16] analogously considers the continuous case, where D_4 is orthogonal transformations so the Jacobian is always 1.
6. Definition of $\star$
7. Transform law of the induced representation π_{out}

I.4 Proof: equivariance of steerable value iteration

As the third and final step, we would like to show that the full steerable value iteration pipeline is equivariant under $G = \mathbb{Z}^2 \rtimes D_4$. We need to show that every operation in the steerable value iteration is equivariant.

The key is to prove that $\max_a$ is an equivariant non-linearity over feature fields, which follows Section D.2 in [16].

Step 1: $V \mapsto Q$**.** Here, we prove the equivariance of $Q_k^a(s) = \bar{R}_M^a(s) + \gamma \times \left[\bar{P}_\theta^a \star V_k\right](s)$. First, let the group acts on both sides:

$$
\begin{aligned}
& Q_k^a(s) = \bar{R}_M^a(s) + \gamma \times \left[\bar{P}_\theta^a \star V_k\right](s) && (50)\\
\iff\; & [\pi_{\text{out}}(g)Q_k^a](s) = [\pi_{\text{out}}(g)\bar{R}_M^a](s) + \gamma \times \left[\pi_{\text{out}}(g)\left[\bar{P}_\theta^a \star V_k\right]\right](s) && (51)\\
\iff\; & [\pi_{\text{out}}(g)Q_k^a](s) = [\pi_{\text{out}}(g)\bar{R}_M^a](s) + \gamma \times \left[\bar{P}_\theta^a \star [\pi_{\text{in}}(g)V_k]\right](s) && (52)\\
\iff\; & Q_k^{g.a}(g^{-1}s) = \bar{R}_{g.M}^{g.a}(g^{-1}s) + \gamma \times \left[\bar{P}_\theta^{g.a} \star V_k\right](g^{-1}s) && (53)\\
\iff\; & Q_k^{\tilde{a}}(\tilde{s}) = \bar{R}_{\pi_{\text{M}}(g)M}^{\tilde{a}}(\tilde{s}) + \gamma \times \left[\bar{P}_\theta^{\tilde{a}} \star V_k\right](\tilde{s}) && (54)
\end{aligned}
$$

The the last step we substitute $\tilde{s} = g^{-1}s$ and $\tilde{a} = g.a$.

$M : \mathbb{Z}^2 \to \{0,1\}^2$ is the concatenation of maze occupancy map and goal map, which also lives on $\mathbb{Z}^2$. We use two copies of trivial representations as fiber representation ρ_{M}, and denote the induced representation of the field M as π_{M}.

Then, we prove the equivariance: if we transform the occupancy map (and goal map), the value iteration should have both input V and output Q transformed. Since this is an iterative process, the only input to the value iteration is actually the occupancy map $M : \mathbb{Z}^2 \to \{0,1\}^2$.

Before that, we observe that the reward also has G-invariance when we have map as input:

$$
\bar{R}_M^a(s) = \bar{R}_{g.M}^{g.a}(g.s). \tag{55}
$$

Additionally, since the reward $\bar{R}_M^a(s)$ means the reward at given position in map M **after executing action** a, when we transform the map, we also need to transform the action: $\bar{R}_{g.M}^{g.a}(s)$.

Since it is iterative process, let the Q-map being transformed by g:

$$
\begin{aligned}
[g.Q_k^a](s) &= Q_k^a(g^{-1}s) && (56)\\
&= \bar{R}_M^a(g^{-1}s) + \gamma \times \left[\bar{P}_\theta^a \star V_k\right](g^{-1}s) && (57)\\
&= \bar{R}_{g.M}^{g.a}(s) + \gamma \times \left[\bar{P}_\theta^a \star V_k\right](g^{-1}s) && (58)\\
&= \bar{R}_{g.M}^{g.a}(s) + \gamma \times \left[\bar{P}_\theta^{g.a} \star [g.V_k]\right](s) && (59)
\end{aligned}
$$

The second last step uses the G-invariance condition $\bar{R}_M^a(s) = \bar{R}_{g.M}^{g.a}(g.s)$. The last step uses the equivariance of steerable convolution.

It should be understood as: (1) transforming map $g.M$ and action $g.a$, is always equal to (2) transforming values $[g.Q_k^a]$ and $[g.V_k]$. This proves the equivariance visually shown in Figure 19.

Step 2: $Q \mapsto V$**.** The second step is to show for $V_{k+1}(s) = \max_a Q_k^a(s)$.

Intuitively, we sum over every channel of each representation. For example, if we have N copies of the regular representation with size $|D_4| = 8$, we transform the tensor $(N \times 8) \times m \times m$ to $(1 \times 8) \times m \times m$ along the N channel. Thus, how we use the 8×8 regular representation to transform the $N \times 8$ channels still holds for 1×8, which implies equivariance. The $m \times m$ spatial map channels form the base space $\mathbb{Z}^2$ and are transformed as usual (spatially rotated).

Weiler and Cesa [16] provide detailed illustration and proofs for equivariance of different types of non-linearities.

Step 3: multiple iterations. Since each layer is equivariant (under induced representations), Cohen and Welling [15], Kondor and Trivedi [13], Cohen et al. [12] show that stacking multiple equivariant layers is also equivariant. Thus, we know iteratively applying step 1 and 2 (*equivariant steerable Bellman operator*) is also *equivariant* (*steerable value iteration*).

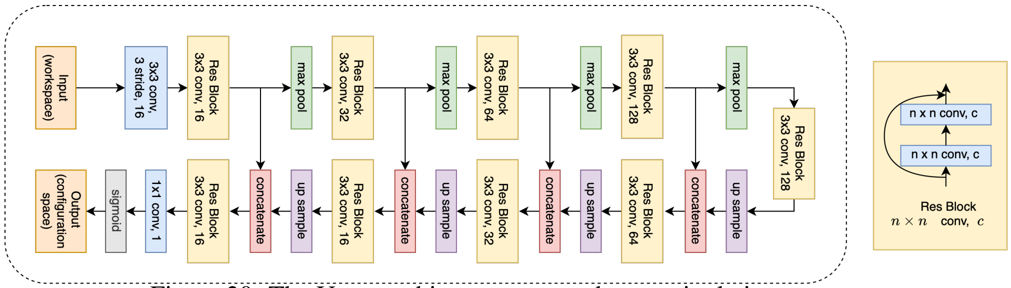

Figure 20: The U-net architecture we used as manipulation mapper.

J Practice and Implementation Details

J.1 Note

We provide additional practical and implementation details, and leave results in the next section.

J.2 Building Mapper Networks

For visual navigation. For navigation, we follow the setting in GPPN [18]. The input is $m \times m$ panoramic egocentric RGB images in 4 directions of resolution $32 \times 32 \times 3$, which forms a tensor of $m \times m \times 4 \times 32 \times 32 \times 3$. A mapper network converts every image into a 256-dimensional embedding and results in a tensor in shape $m \times m \times 4 \times 256$ and then predicts map layout $m \times m \times 1$.

For the first image encoding part, we use a CNN with first layer of 32 filters of size 8×8 and stride of 4×4, and second layer with 64 filters of size 4×4 and stride of 2×2, with a final linear layer of size 256.

The second obstacle prediction part, the first layer has 64 filters and the second layer has 1 filter, all with filter size 3×3 and stride 1×1.

For workspace manipulation. For **workspace manipulation**, we use U-net [58] with residual-connection [59] as a mapper, see Figure.20. The input is 96×96 top-down occupancy grid of the workspace with obstacles, and the target is to output 18×18 configuration space as the maps for planning.

During training, we pre-train the mapper and the planner separately for 15 epochs. Where the mapper takes manipulator workspace and outputs configuration space. The mapper is trained to minimize the binary cross entropy between output and ground truth configurations space. The planner is trained in the same way as described in Section 6.1. After pre-training, we switch the input to the planner from ground truth configuration space to the one from the mapper. During testing, we follow the pipeline in [37] that the mapper-planner only have access to the manipulator workspace.

J.3 SymGPPN

ConvGPPN [Redacted for anonymous review] is inspired by VIN and GPPN. To avoid the training issues in VIN, GPPN proposes to use LSTM to alleviate them. In particular, it does not use max pooling in the VIN. Instead, it uses a CNN and LSTM to mimic the value iteration process. ConvGPPN, on the other hand, integrates CNN into LSTM, resulting in a single component convLSTM for value iteration. We found that ConvGPPN performs better than GPPN in most cases. Based on ConvGPPN, SymGPPN replaces each convolutional layer with steerable convolutional layer.

J.4 Understand group conv and "augmented state"

We derive the relationship between group convolution and steerable convolution in Section G.3.

The augmented state $\mathbb{Z}^2 \rtimes D_4 \to \mathbb{R}$ can be similarly treated on the group $p4m = \mathbb{Z}^2 \rtimes D_4$. It is equivalent to using regular representation on the base space $\mathbb{Z}^2$ as $\mathbb{Z}^2 \to \mathbb{R}^8$.

J.5 Implementation of $\max$ operation

Here, we consider how to implement the $\max$ operation in $V_{k+1}(s) = \max_a Q_k^a(s)$. The $\max$ is taken over every state, so the computation mainly depends on our choice of fiber representation.

For example, if we use *trivial representations* for both input and output, the input would be $Q_k : \mathbb{Z}^2 \to \mathbb{R}^{1*C_A}$ and the output is state-value $V_k : \mathbb{Z}^2 \to \mathbb{R}$. This recovers the default value iteration since we take $\max$ over $\mathbb{R}^{C_A}$ vector.

In steerable CNNs, we can use stack of fiber representations. We can choose from regular-regular, trivial-trivial, and regular-trivial (trivial-regular is not considered).

We already covered *trivial* representations for both input and output, they would be $Q_k : \mathbb{Z}^2 \to \mathbb{R}^{C_Q*C_A}$ and $V_k : \mathbb{Z}^2 \to \mathbb{R}^{C_V}$ with $C_Q = C_V = 1$, since every channel would need a trivial representation.

If we use *regular* representation for Q and *trivial* for V, they are $Q_k : \mathbb{Z}^2 \to \mathbb{R}^{C_Q*C_A}$ and $V_k : \mathbb{Z}^2 \to \mathbb{R}^{C_V}$ with $C_Q = |D_4| = 8$ and $C_V = 1$. It degenerates that we just take $\max$ over all $C_Q * C_A$ channels.

For both using regular representations, we need to make sure they use the same fiber group (such as D_4 or C_4), so $C_Q = C_V$. If using D_4, we have $Q_k : \mathbb{Z}^2 \to \mathbb{R}^{8*C_A}$ and $V_k : \mathbb{Z}^2 \to \mathbb{R}^8$, and we take max over every C_A channels (for every location) and have 8 channels left, which are used as $\mathbb{Z}^2 \to \mathbb{R}^8$.

Empirically, we found using regular representations for both works the best overall.