Exploiting Symmetry in Dynamics for Model-Based Reinforcement Learning with Asymmetric Rewards

Yasin Sonmez, Neelay Junnarkar, *Student Member, IEEE*, and Murat Arcak, *Fellow, IEEE*

***Abstract*— Recent work in reinforcement learning has leveraged symmetries in the model to improve sample efficiency in training a policy. A commonly used simplifying assumption is that the dynamics and reward both exhibit the same symmetry; however, in many real-world environments, the dynamical model exhibits symmetry independent of the reward model. In this paper, we assume only the dynamics exhibit symmetry, extending the scope of problems in reinforcement learning and learning in control theory to which symmetry techniques can be applied. We use Cartan's moving frame method to introduce a technique for learning dynamics that, by construction, exhibit specified symmetries. Numerical experiments demonstrate that the proposed method learns a more accurate dynamical model.**

***Index Terms*— Symmetry, neural networks, reinforcement learning.**

I. Introduction

A Major research area in reinforcement learning (RL) is improving sample efficiency, the amount of interaction with the environment needed to learn a good policy. While model-free methods such as [1]–[3] have shown the ability to achieve high rewards in many complex environments, they require many environment interactions. Model-based reinforcement learning (MBRL) methods, on the other hand, have shown promise in learning policies quickly relative to the number of environment interactions. This improvement in sample efficiency confirms that models contain a great deal of information useful for improving policies [4].

One idea gaining recent attention in RL methods is to integrate symmetries into model-based information. Using knowledge of symmetries, one can extrapolate observed behavior to other scenarios. Recent work has leveraged symmetries via equivariant networks [5], [6], which enforce certain transformation relationships on the inputs and outputs, and via data augmentation [7], where artificial data is created based on observed data and known transformations for training. However, a limitation of equivariant networks is that they must be designed for the particular group of symmetries to which they are equivariant. Unlike the above methods, which assume that the symmetries are known apriori, [7] learns symmetries from data and augments the training data using these symmetries.

Methods that leverage symmetry have also been explored for control design and verification. Reference [8] uses symmetry to deduce the structure of the optimal controllers for nonlinear systems. Reference [9] uses symmetry to improve the convergence of observers. Reference [10] explores the use of symmetry for reducing the dimension of variables involved in dynamic programming, accelerating the computation of control policies. Reference [11] uses symmetry reduction to accelerate reachability computations. References [12] and [13] use symmetry in accelerating safety verification.

A common assumption when using symmetry in RL and optimal control is that both the dynamics and reward (or cost) function exhibit the same symmetries, which results in the optimal policy (or control law) likewise being symmetric [10], [14]. The symmetric policy structure is exploited for Markov Decision Processes (MDPs) and RL in [5], [6], [14], [15], and for dynamic programming in [10]. In many scenarios, however, symmetries in the dynamical model are not necessarily applicable to the reward model. For example, multiple tasks might be executed in the same environment, where the same dynamical symmetries exist, while the reward model changes from task to task. Exploiting symmetries only in the dynamical model widens the scope of RL problems to which symmetry can be applied.

Motivated by commonly encountered translation and rotation symmetries in dynamics, we consider a method of enforcing continuous symmetries through transformation into a reduced coordinate space. For example, the dynamics of the cart-pole system, commonly used for control systems pedagogy, are invariant to the position of the cart. This fact can be exploited when learning the system dynamics by simply removing the position data from the training dataset. It is, however, less clear how to account for symmetries that involve more complex transformations.

In this paper, we present a method to learn dynamical models that are, by construction, invariant under specified continuous symmetries. This enables encoding apriori known symmetry structure when learning a dynamical model. To address a large class of symmetries, we use Cartan's moving frame method [16], which simplifies models by adapting moving reference frames to the structure of the space.

This work was in supported in part by the National Science Foundation award CNS-2111688 and Air Force Office of Scientific Research grant FA9550-21-1-0288.

Yasin Sonmez, Neelay Junnarkar, and Murat Arcak are with the department of Electrical Engineering and Computer Sciences at the University of California, Berkeley. (emails: {yasin_sonmez, neelay.junnarkar, arcak} @berkeley.edu)

This paper is organized as follows: in Section II, we provide background information on symmetries, and then use Cartan's method of moving frames to relate a symmetrical dynamical model to a function operating on an input space of reduced dimension. In Section III, we use the result of Section II to learn a dynamical model that, by construction, satisfies specified symmetries. In Section IV we apply symmetry reduction to learn dynamical models for two examples.

II. Symmetries in Model Dynamics

Consider the deterministic discrete-time system

$$x_{k+1} = F(x_k, u_k) \tag{1}$$

where $x_k \in \mathcal{X} = \mathbb{R}^n$ is the state, $u_k \in \mathcal{U} = \mathbb{R}^{n_u}$ is the control input, and $k \in \mathbb{N}$ denotes the time step. *Symmetry* refers to a structure of the dynamical system such that transformations of the input (x_k, u_k) to F result in corresponding, known, transformations to the output x_{k+1}. We formalize this notion with *Lie transformation groups*, an introduction to which can be found in [17] and [18].

Definition 1 (Lie Transformation Group): Let G be a Lie group acting on a smooth manifold Σ. Denote the group action by $w(g, \sigma) \mapsto g \cdot \sigma$. The group G is a Lie transformation group if $w : G \times \Sigma \to \Sigma$ is smooth.

Important properties of the group action are:

- $e \cdot \sigma = \sigma$ for all σ where e is the identity of group G.
- $g_1 \cdot (g_2 \cdot \sigma) = (g_1 * g_2) \cdot \sigma$ for all $g_1, g_2 \in G, \sigma \in \Sigma$, where $g_1 * g_2$ denotes the group operation, which we will denote by $g_1 g_2$ henceforth.

We represent symmetries of (1) with an r-dimensional Lie group G which is a Lie transformation group of both $\mathcal{X}$ and $\mathcal{U}$. Assume $r \leq n$. For notation, we consider the maps $\phi_g : \mathcal{X} \to \mathcal{X}$ defined by $x \mapsto g \cdot x$ and $\psi_g : \mathcal{U} \to \mathcal{U}$ defined by $u \mapsto g \cdot u$. Since the group actions are smooth, both of these maps are smooth for all $g \in G$. Further, they have smooth inverses $\phi_g^{-1} = \phi_{g^{-1}}$ and $\psi_g^{-1} = \psi_{g^{-1}}$.

We next define dynamics invariant to symmetry:

Definition 2 (Invariant Dynamics): The system (1) is G-invariant if $F(\phi_g(x), \psi_g(u)) = \phi_g(F(x, u))$ for all $g \in G$, $x \in \mathcal{X}$, $u \in \mathcal{U}$.

The premise of our method is to transform (x, u) to its canonical form, corresponding to an element of $\{(\phi_g(x), \psi_g(u)) \mid g \in G\}$. When F in (1) is G-invariant, we parameterize it by a function that operates on these canonical forms, which reside in a lower-dimensional space.

A. Cartan's Moving Frame

We use Cartan's moving frames [16] to construct a map from elements of the state space to their canonical forms. What follows is a brief recap of Cartan's method based on [9], [11], [19]. Note that, in general, this method constructs a map which exists only locally, due to the use of the implicit function theorem. On many practical examples, such as those in this work and [11], the map exists over all $\mathcal{X}$ except a lower-dimensional submanifold. Thus, in our presentation, we will focus on transformations within a single smooth chart. Refer to [19] for a thorough treatment of the subject.

Consider the Lie group G and its action on $\mathcal{X}$ as before. Assume $\mathcal{X}$ can be split into $\mathcal{X}^a$ and $\mathcal{X}^b$ with r and $n - r$ dimensions respectively such that, for some $x_0 \in \mathcal{X}$, the map from $g \in G$ to the projection of $g \cdot x_0$ onto $\mathcal{X}^a$ is invertible. Pick $c \in \mathcal{X}^a$ in the range of this map. For all $g \in G$, let $\phi_g^a : \mathcal{X} \to \mathcal{X}^a$ and $\phi_g^b : \mathcal{X} \to \mathcal{X}^b$ denote the compositions of ϕ_g and the projections onto $\mathcal{X}^a$ and $\mathcal{X}^b$ respectively. Assume that, for all $x \in \mathcal{X}$, there exists a unique $g \in G$ such that $\phi_g^a(x) = c$. The set $\mathcal{C} = \{x | \phi_e^a(x) = c\}$ is an $n-r$ dimensional submanifold of $\mathcal{X}$ called a *cross-section*. We can then define the unique map $\gamma : \mathcal{X} \to G$, such that $\phi_{\gamma(x)}(x) \in \mathcal{C}$. This γ is called a *moving frame*. Using γ, we define the map $\rho : \mathcal{X} \to \mathcal{X}^b$ from elements in $\mathcal{X}$ to their canonical forms:

$$\rho(x) \triangleq \phi^b_{\gamma(x)}(x). \tag{2}$$

This ρ is invariant to the action of G on the state, as shown in the following lemma adapted from [9].

Lemma 1: For all $g \in G$ and $x \in \mathcal{X}$, $\rho(\phi_g(x)) = \rho(x)$.

Proof: Let $g \in G$. Then, $\rho(\phi_g(x)) = \phi^b_{\gamma(\phi_g(x))}(\phi_g(x)) = \phi^b_{\gamma(\phi_g(x))g}(x)$. From the definition of γ, we have that both $\phi^a_{\gamma(x)}(x)$ and $\phi^a_{\gamma(\phi_g(x))}(\phi_g(x))$ equal c. Note that $\phi^a_{\gamma(\phi_g(x))}(\phi_g(x)) = \phi^a_{\gamma(\phi_g(x))g}(x)$. Since $\gamma(x)$ is the unique group element h such that $\phi^a_h(x) = c$, we have that $\gamma(\phi_g(x))g = \gamma(x)$. Plugging this into the expression for $\rho(\phi_g(x))$ gives $\rho(\phi_g(x)) = \phi^b_{\gamma(x)}(x) = \rho(x)$. ∎

Example 1: In this example, we demonstrate solving for the moving frame for a system with both translation and rotation symmetries. Consider a car with state $x = (y, z, v_y, v_z, h_y, h_z)$ where y and z correspond to positions in a plane, v_y and v_z correspond to velocities along the y and z directions, and h_y and h_z correspond to the cosine and sine of the heading angle (and thus $h_y^2 + h_z^2 = 1$). Since we assume the dynamics of the car are both translation- and rotation-invariant, we consider the special Euclidean group $G = \mathrm{SE}(2)$, and parameterize it by a translation $(\tilde{y}, \tilde{z})$ and a rotation $\tilde{\theta}$. Then, with rotation matrix $R_{\tilde{\theta}} = \left[\begin{smallmatrix} \cos\tilde{\theta} & -\sin\tilde{\theta} \\ \sin\tilde{\theta} & \cos\tilde{\theta} \end{smallmatrix}\right]$, we define the action of G on the state space by

$$\phi_{(\tilde{y},\tilde{z},\tilde{\theta})}(x) = \begin{bmatrix} R_{\tilde{\theta}} & 0 & 0 \\ 0 & R_{\tilde{\theta}} & 0 \\ 0 & 0 & R_{\tilde{\theta}} \end{bmatrix} x + \begin{bmatrix} \begin{bmatrix} \tilde{y} \\ \tilde{z} \end{bmatrix} \\ 0 \\ 0 \end{bmatrix}. \tag{3}$$

We now derive γ and ρ. Let $\mathcal{X}^a$ be the projection of $\mathcal{X}$ onto the (y, z, h_y, h_z) components. Pick $c = (0, 0, 1, 0)$, representing a position at the origin with a heading angle of 0. There exists a unique $(\tilde{y}, \tilde{z}, \tilde{\theta})$ to transform each $x \in \mathcal{X}$ into the cross-section defined by $\mathcal{C}$. We solve the following system of equations to find the moving frame:

$$\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \phi^a_{\gamma(x)}(x) = \begin{bmatrix} R_{\tilde{\theta}} & 0 \\ 0 & R_{\tilde{\theta}} \end{bmatrix} \begin{bmatrix} y \\ z \\ h_y \\ h_z \end{bmatrix} + \begin{bmatrix} \tilde{y} \\ \tilde{z} \\ 0 \\ 0 \end{bmatrix}.$$

This gives the solution

$$\gamma(x) = \begin{bmatrix} \tilde{y} \\ \tilde{z} \\ \tilde{\theta} \end{bmatrix} = \begin{bmatrix} -yh_y - zh_z \\ yh_z - zh_y \\ \arctan2(-h_z, h_y) \end{bmatrix}. \tag{4}$$

We can then compute the group inverse of $\gamma(x)$ as:

$$\gamma(x)^{-1} = \begin{bmatrix} y \\ z \\ \arctan2(h_z, h_y) \end{bmatrix}. \tag{5}$$

Substituting $\gamma(x)$ into the definition of ρ from (2), we get:

$$\rho(x) = \begin{bmatrix} h_y v_y + h_z v_z \\ -h_z v_y + h_y v_z \end{bmatrix}. \tag{6}$$

B. Dimension Reduction

Using the map ρ from elements in $\mathcal{X}$ to their canonical forms, we show that the dynamics in (1) are equivalent to a function operating on a lower-dimensional input space.

Theorem 1: $F : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$ in system (1) is G-invariant if and only if there exists $\bar{F} : \mathcal{X}^b \times \mathcal{U} \to \mathcal{X}$ such that

$$\phi_{\gamma(x)}(F(x,u)) = \bar{F}(\rho(x), \psi_{\gamma(x)}(u)) \tag{7}$$

for all $x \in \mathcal{X}$ and $u \in \mathcal{U}$.

Proof: Assume that the dynamics F are G-invariant. Note that ρ restricted to $\mathcal{C}$ is invertible. Define $\bar{F}$ by $\bar{F}(\bar{x}, u) = F(\rho|_{\mathcal{C}}^{-1}(\bar{x}), u)$. Letting $x = \rho|_{\mathcal{C}}^{-1}(\bar{x})$, this can equivalently be written as $\bar{F}(\rho(x), u) = F(x, u)$ when $x \in \mathcal{C}$. Since F is G-invariant, $\phi_{\gamma(x)}(F(x,u)) = F(\phi_{\gamma(x)}(x), \psi_{\gamma(x)}(u))$. Note that $\phi_{\gamma(x)}(x) \in \mathcal{C}$. This can be rewritten in terms of $\bar{F}$ as $\bar{F}(\rho(\phi_{\gamma(x)}(x)), \psi_{\gamma(x)}(u))$. By Lemma 1, this equals $\bar{F}(\rho(x), \psi_{\gamma(x)}(u))$, as desired.

To prove the reverse direction, assume $\bar{F} : \mathcal{X}^b \times \mathcal{U} \to \mathcal{X}$ exists such that (7) holds for all $x \in \mathcal{X}$, $u \in \mathcal{U}$. Let $g \in G$. Then,

$$\begin{aligned} &F(\phi_g(x), \psi_g(u)) \\ &\quad = \phi^{-1}_{\gamma(\phi_g(x))}(\bar{F}(\rho(\phi_g(x)), \psi_{\gamma(\phi_g(x))}(\psi_g(u)))) \\ &\quad = \phi^{-1}_{\gamma(\phi_g(x))}(\bar{F}(\rho(x), \psi_{\gamma(\phi_g(x))g}(u))). \end{aligned}$$

From the proof of Lemma 1, we know $\gamma(\phi_g(x))g = \gamma(x)$. Therefore, the above can be simplified to

$$\begin{aligned} F(\phi_g(x), \psi_g(u)) &= \phi^{-1}_{\gamma(x)g^{-1}}(\bar{F}(\rho(x), \psi_{\gamma(x)}(u))) \\ &= \phi_g(\phi_{\gamma(x)^{-1}}(\phi_{\gamma(x)}(F(x,u)))) \\ &= \phi_g(F(x,u)). \end{aligned}$$

Thus, F is G-invariant. ∎

III. Learning Model Dynamics

A critical part of the theory in Section II is that the only knowledge of system dynamics required is that of the state space, the control input space, and the system symmetries. We leverage this to learn a dynamical model that is invariant to a given transformation group using Theorem 1. Unlike a standard formulation, which would learn the model $F : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$, we learn a function $\bar{F} : \mathcal{X}^b \times \mathcal{U} \to \mathcal{X}$ and construct F from:

$$F(x,u) \triangleq \phi^{-1}_{\gamma(x)}(\bar{F}(\rho(x), \psi_{\gamma(x)}(u))). \tag{8}$$

Note that, in this section, we use (x, u, x') to denote a state, an action applied to the state, and the next state.

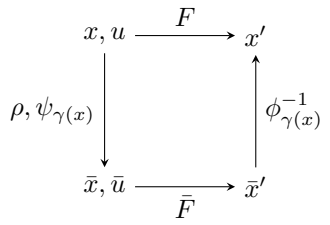

Fig. 1. Relationship between F and $\bar{F}$.

The relationship between F and $\bar{F}$ is depicted in Figure 1. From Theorem 1, F is, by construction, G-invariant. A side-benefit is that training is done with a lower dimensional input space ($\mathcal{X}^b \times \mathcal{U}$ instead of $\mathcal{X} \times \mathcal{U}$). To learn this function $\bar{F}$, given a dataset $\mathcal{D}$ consisting of tuples (x, u, x'), we train $\bar{F}$ to map $(\rho(x), \psi_{\gamma(x)}(u))$ to $\phi_{\gamma(x)}(x')$.

Note that, in practice, it is typical to learn a function which maps (x, u) to $x' - x$ instead of x' directly. To do this, we learn $\Delta\bar{F}$ to map $(\rho(x), \psi_{\gamma(x)}(u))$ to $\phi_{\gamma(x)}(x') - \phi_{\gamma(x)}(x)$. Then, we construct F by

$$F(x,u) = \phi^{-1}_{\gamma(x)}(\Delta\bar{F}(\rho(x), \psi_{\gamma(x)}(u)) + \phi_{\gamma(x)}(x)).$$

This construction for F is G-invariant since it is equivalent to using $\bar{F}(\bar{x}, \bar{u}) \triangleq \Delta\bar{F}(\bar{x}, \bar{u}) + \rho|_{\mathcal{C}}^{-1}(\bar{x})$, which satisfies the condition in Theorem 1. Finally, $\Delta F(x,u)$ can be defined as $\Delta F(x,u) \triangleq F(x,u) - x$ to provide the same $x' - x$ interface expected by many libraries. This construction for ΔF in terms of $\Delta\bar{F}$ expands as follows.

$$\Delta F(x,u) \triangleq \phi^{-1}_{\gamma(x)}(\Delta\bar{F}(\rho(x), \psi_{\gamma(x)}(u)) + \phi_{\gamma(x)}(x)) - x \tag{9}$$

One special case where this simplifies is when ϕ_g is a homomorphism with respect to $+$ for all $g \in G$. Then, (9) simplifies to

$$\Delta F(x,u) = \phi^{-1}_{\gamma(x)}(\Delta\bar{F}(\rho(x), \psi_{\gamma(x)}(u))).$$

An example where this occurs is when ϕ_g is linear.

IV. Experiments

In this section, we demonstrate the performance of this method on two numerical examples involving learning the dynamics of a system. The two environments that we used were "Parking" from "Highway-env" [20] and "Reacher" from OpenAI Gym [21]. These exhibit rotational and/or translational symmetry, as depicted in Figure 2. In "Parking" there are two controlled vehicles that we need to park to corresponding parking spots, and in "Reacher" we control a robot arm with 2 joints to reach to a target position. To be able to compare our method fairly to a baseline we generated a static dataset of transitions (x, u, x'), and we learn a dynamical model from this dataset. Consequently, the comparison between the two methodologies was centered on their respective capabilities in dynamics learning rather than policy generation. This approach ensures a fair comparison, as it relies on the same offline dataset for both methods, avoiding discrepancies that may arise from online training and dataset variations.

The dataset is constructed by training an agent using Soft Actor-Critic [2], a model-free reinforcement learning method.

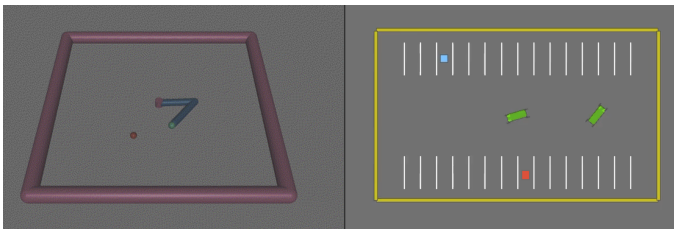

Fig. 2. Experimental environments included: (1) “Reacher” on the left, featuring two rotating controlled joints that exhibit rotation symmetry with the objective of reaching a target point, and (2) “Parking” on the right, involving two controlled vehicles maneuvering to park in designated spots without collision, demonstrating both rotational and translational symmetry in dynamics.

We use D3rlpy [22] to train dynamical models from the pre-collected dataset. We implement the symmetry method discussed above using (8), where $\bar{F}$ is the neural network whose parameters are trained. The observation error (on the test dataset) of this dynamical model is compared against the observation error of learning F directly. We further compare these methods (with and without symmetry) across a variety of parameter sizes. The rollouts from the resulting policy, the datasets used in this work and the code to reproduce the results are available in a GitHub repository[1].

A. Two Cars Parking Scenario

We use the parking environment from [20] with two cars with separate goal positions. The aim is to park the two cars at their goal spots by controlling them separately and not allowing them to crash into obstacles or to each other. The dynamics of each car are translation- and rotation-invariant, but the reward function (relating to the location and orientation of the parking spot) is not. Therefore, methods that assume both the dynamics and the reward function satisfy the same symmetries are not applicable.

The state of this parking environment is 24-dimensional, with the first 6 states corresponding to the first car, the second 6 states corresponding to the second car, the third 6 states corresponding to the first car’s goal, and the final 6 states corresponding to the second car’s goal. The state incorporates the two car models and two goal models. The goals follow the model $g_{k+1} = g_k$. We include the goal states in the state representation because it aligns with the conventional RL framework where all states are utilized directly in the training process without explicit distinction. We apply the method presented in Sections II and III to construct transformation groups for each of the four systems. The joint system then satisfies the transformation group formed as the product of the individual transformation groups.

Each car has state $x = (y, z, v_y, v_z, h_y, h_z)$, corresponding to the y position, z position, y component of velocity, z component of velocity, cosine of heading angle, and sine of heading angle. We use the group $G = \mathrm{SE}(2)$ to represent the symmetries. See Example 1 for the action of G on the state space and the solutions for γ and ρ. The action of G on the control input space is the identity: $\psi_g(u) = u$ for all $g \in G$.

[1] https://github.com/YasinSonmez/symmetry-cs285

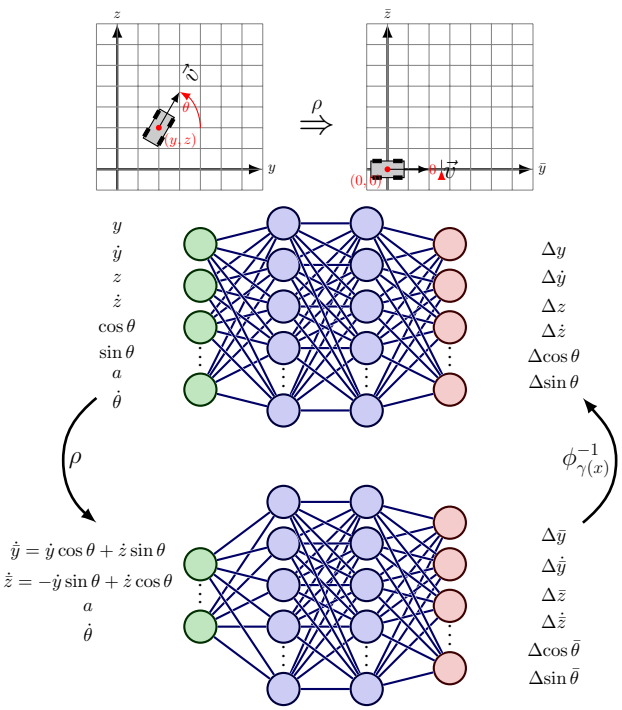

Fig. 3. Illustration of translational and rotational invariance in car dynamics. By applying Cartan’s moving frame method, coordinates are transformed to position the car at the origin with a neutral orientation. The function $\boldsymbol{\rho}$ reduces the system’s state to a lower-dimensional space, without losing essential dynamics information. The dynamics are learned in these modified coordinates via a smaller neural network (bottom) compared to the usual NN (middle). The NN’s output is then reconverted to original coordinates using $\boldsymbol{\phi^{-1}_{\gamma(x)}}$. As usual, the neural network training is simplified by outputting the difference between the next and current states, represented as $\boldsymbol{\Delta F(x,u)}$ and $\boldsymbol{\Delta \bar{F}(x,u)}$ in Section III. This is an illustration of a single car. In the experiments, additional symmetries are observed due to the presence of two cars; however, the second car has been omitted here for clarity.

The goal dynamical model satisfies the trivial transformation group $(\mathbb{R}^6, +)$ which acts on the goal by $\phi_\delta(g) = g + \delta$. There is no input to act on. The a component of ϕ has 6 components, leaving the b component with 0.

Using the above transformation groups applied to each car and goal, the transformation group applied to the joint state of the environment gives a ρ which transforms the input state from a 24-dimensional vector to a 4-dimensional one. Thus, we learn a neural network with significantly reduced input size. Note that the computation of these functions only depends on the symmetry and not on the dynamics. The dynamical model need not be known apriori and can be learned using the data; see Figure 3.

The three neural network configurations used in the experiments are: 1 hidden layer with size 128, 2 hidden layers with size 128, and 3 hidden layers with size 128. When using symmetry, the input size is 8 (reduced coordinate size for each car of 2 and control input size of 2 for each car), and the output size is 24. In the method not using symmetry, the input size is 28 (state space size of 24 and control input size of 2 for each of the two cars), and the output size is 24.

The results of observation error vs. number of parameter updates are shown in Figure 4. At the lower number of

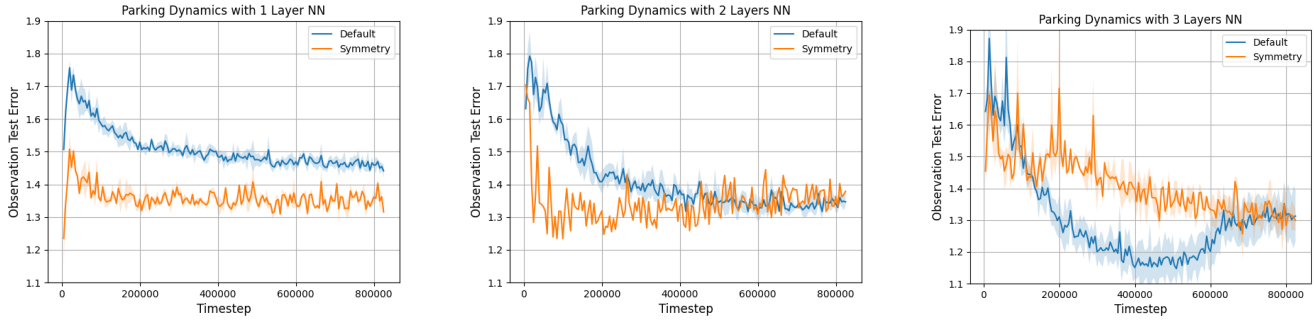

Fig. 4. Comparison of learning the dynamics with and without using symmetry in parking scenario with different NN architectures. The y-axis (observation error) is the error on the test dataset. Mean and standard deviation reported over 4 runs.

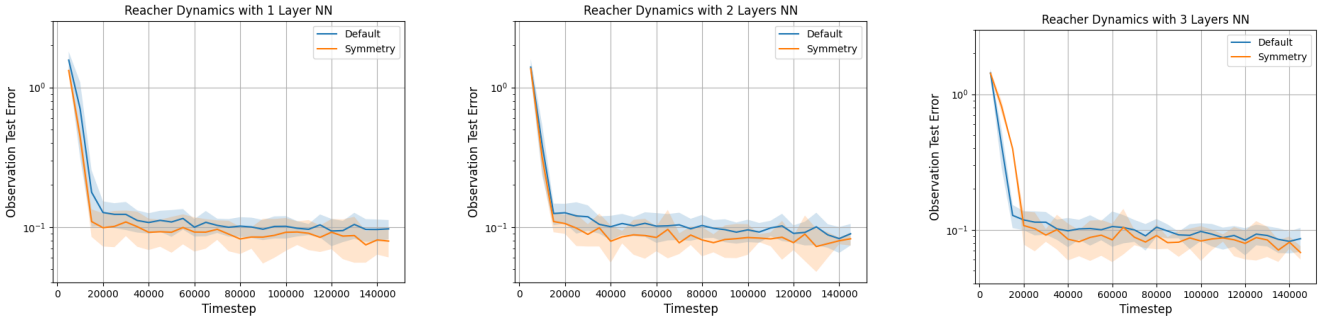

Fig. 5. Comparison of learning the dynamics with and without using symmetry in reacher environment with different NN architectures. The y-axis (observation error) is the error on the test dataset. Mean and standard deviation reported over 4 runs.

parameters, the dynamical model training with symmetry outperforms the default method without symmetry by achieving a lower observation error. When there are two hidden layers, the method with symmetry learns faster than the default method without symmetry, although eventually the method without symmetry catches up in performance. When there are three hidden layers, the method without symmetry exceeds the performance of the method with symmetry. This suggests that the relative benefit of leveraging symmetry compared to the benefit from the number of parameters diminishes as the number of neural network parameters increases. One possible cause for this is that our method of leveraging symmetry removes redundant information. This redundant information is a transformed version of the other available information, and might be more immediately useful to the network. As a result, the neural network does not have to learn these transformations on its own, making the redundant information beneficial when a sufficiently high number of parameters is available.

B. Reacher

In this experiment, we consider the standard reacher environment in OpenAI Gym. The dynamics exhibit rotational symmetry with the first joint angle, but the dynamics of the target do not since the target is fixed. We use symmetry reduction to learn model dynamics which are rotation-invariant.

The state of the reacher is 11-dimensional, with x_1 and x_2 representing the cosines of the first and second joint angles, x_3 and x_4 representing the sines of the joint angles, x_5 and x_6 representing the x- and y-coordinates of the target, x_7 and x_8 representing the angular velocities of the first and second arm, x_9 and x_{10} representing the distance between the reacher fingertip and the target along the x- and y-axes, and x_{11} equaling 0, representing the z coordinate of the fingertip.

We now describe the transformation group applied to learn a rotation-invariant dynamical model. For x_5, x_6, and x_{11}, we apply the same symmetry used in Section IV-A to describe constants. Let G be parameterized by $g = (\theta', \delta_1, \delta_2, \delta_3) \in \mathbb{R}^4$, where θ' represents the angle by which the reacher is rotated, and δ represents translations of the target position and x_{11}. We define the action of G on the state space as:

$$\phi_g(x) = \begin{bmatrix} \cos(\theta')x_1 - \sin(\theta')x_3 \\ x_2 \\ \sin(\theta')x_1 + \cos(\theta')x_3 \\ x_4 \\ \cos(\theta')(x_5 + \delta_1) - \sin(\theta')(x_6 + \delta_2) \\ \sin(\theta')(x_5 + \delta_1) + \cos(\theta')(x_6 + \delta_2) \\ x_7 \\ x_8 \\ \cos(\theta')(x_9 - \delta_1) - \sin(\theta')(x_{10} - \delta_2) \\ \sin(\theta')(x_9 - \delta_1) + \cos(\theta')(x_{10} - \delta_2) \\ x_{11} + \delta_3 \end{bmatrix}. \quad (10)$$

The action of G on the control input is the identity: $\psi_g(u) = u$ for all $g \in G$. We define $\mathcal{C}$ by the states x for which $x_1 = 1$, $x_3 = 0$, $x_5 = 0$, $x_6 = 0$, and $x_{11} = 0$. The solution for the

moving frame $\gamma(x)$ is

$$\gamma(x) = \begin{bmatrix} \arctan2(-x_3, x_1) \\ -x_5 \\ -x_6 \\ -x_{11} \end{bmatrix}$$

and the solution for the group inverse of $\gamma(x)$ is

$$\gamma(x)^{-1} = \begin{bmatrix} \arctan2(x_3, x_1) \\ x_1 x_5 + x_3 x_6 \\ -x_3 x_5 + x_1 x_6 \\ x_{11} \end{bmatrix}.$$

We now plug γ into (2) to solve for ρ:

$$\rho(x) = \begin{bmatrix} x_2 \\ x_4 \\ x_7 \\ x_8 \\ x_1(x_9 + x_5) + x_3(x_{10} + x_6) \\ -x_3(x_9 + x_5) + x_1(x_{10} + x_6) \end{bmatrix}. \tag{11}$$

Using this symmetry group, the input size of the neural network is 8 (reduced coordinate size 6 and control input size of 2), and the output size is 11. When not using symmetry, the input size is 13 (state space size of 11 and control input size of 2), and the output size is 11. The three neural network configurations that we used in our experiments are: 1 hidden layer with size 64, 2 hidden layers with size 64, and 3 hidden layers with size 64. We used 64 neuron size compared to 128 in "Parking" because of the simplicity of this problem.

The results of observation error vs. number of parameter updates are shown in Figure 5. In all cases, the dynamical model training with symmetry achieves slightly better performance than the default method without symmetry by achieving a lower observation error at the end of the training. This is most evident when the neural network has the least number of parameters, in the 1-layer case.

V. Conclusion

This study highlights the benefits of exploiting symmetry when learning dynamical models. By focusing on the symmetry inherent in dynamics, independent of the reward function, we broaden the range of problems where symmetry can be effectively utilized. We investigated symmetries such as rotation and translation symmetries through Cartan's method of moving frames and our proposed method proves to be effective in leveraging these symmetries to learn more accurate models, particularly at low numbers of model parameters. The experiments confirm the potential of our approach to enhance learning efficiency. Future work involves training policies using the proposed symmetry-enhanced method and evaluating their performance. Additionally, a theoretical framework that includes discrete symmetries can be developed.

VI. Acknowledgements

We thank Sergey Levine for pointing to relevant references and for discussions on preliminary results, and Hussein Sibai for discussions on symmetry reduction prior to this project.

References

[1] T. P. Lillicrap, J. J. Hunt, A. Pritzel, *et al.*, *Continuous control with deep reinforcement learning*, 2019. arXiv: 1509.02971 [cs.LG].

[2] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proceedings of the 35th International Conference on Machine Learning*, PMLR, Jul. 3, 2018, pp. 1861–1870.

[3] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal policy optimization algorithms*, 2017. arXiv: 1707.06347 [cs.LG].

[4] T. M. Moerland, J. Broekens, A. Plaat, and C. M. Jonker, "Model-based reinforcement learning: A survey," *Foundations and Trends® in Machine Learning*, vol. 16, no. 1, pp. 1–118, 2023.

[5] E. van der Pol, D. Worrall, H. van Hoof, F. Oliehoek, and M. Welling, "MDP homomorphic networks: Group symmetries in reinforcement learning," in *Advances in Neural Information Processing Systems*, vol. 33, Curran Associates, Inc., 2020, pp. 4199–4210.

[6] D. Wang, R. Walters, and R. Platt, "SO(2)-equivariant reinforcement learning," in *International Conference on Learning Representations*, 2022.

[7] M. Weissenbacher, S. Sinha, A. Garg, and K. Yoshinobu, "Koopman Q-learning: Offline reinforcement learning via symmetries of dynamics," in *Proceedings of the 39th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 162, PMLR, Jul. 2022, pp. 23 645–23 667.

[8] J. Grizzle and S. Marcus, "Optimal control of systems possessing symmetries," *IEEE Transactions on Automatic Control*, vol. 29, no. 11, pp. 1037–1040, Nov. 1984, ISSN: 1558-2523.

[9] S. Bonnabel, P. Martin, and P. Rouchon, "Symmetry-preserving observers," *IEEE Transactions on Automatic Control*, vol. 53, no. 11, pp. 2514–2526, 2008.

[10] J. Maidens, A. Barrau, S. Bonnabel, and M. Arcak, "Symmetry reduction for dynamic programming," *Automatica*, vol. 97, pp. 367–375, Nov. 1, 2018, ISSN: 0005-1098.

[11] J. Maidens and M. Arcak, "Exploiting symmetry for discrete-time reachability computations," *IEEE Control Systems Letters*, vol. 2, no. 2, pp. 213–217, Apr. 2018, ISSN: 2475-1456.

[12] H. Sibai, N. Mokhlesi, C. Fan, and S. Mitra, "Multi-agent safety verification using symmetry transformations," in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. Lecture Notes in Computer Science, Springer International Publishing, 2020, pp. 173–190.

[13] H. Sibai, N. Mokhlesi, and S. Mitra, "Using symmetry transformations in equivariant dynamical systems for their safety verification," in *Automated Technology for Verification and Analysis*, ser. Lecture Notes in Computer Science, Springer International Publishing, 2019, pp. 98–114.

[14] B. Ravindran and A. G. Barto, "Model minimization in hierarchical reinforcement learning," in *Abstraction, Reformulation, and Approximation*, ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2002, pp. 196–211.

[15] M. Zinkevich and T. R. Balch, "Symmetry in Markov decision processes and its implications for single agent and multiagent learning," in *Proceedings of the Eighteenth International Conference on Machine Learning*, ser. ICML '01, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., Jun. 28, 2001, p. 632.

[16] E. Cartan, *La théorie des groupes finis et continus et la géométrie différentielle: traitées par la méthode du repère mobile* (Cahiers scientifiques). Gauthier-Villars, 1937.

[17] F. Bullo and A. D. Lewis, *Geometric Control of Mechanical Systems* (Texts in Applied Mathematics). Springer, NY, Nov. 4, 2004.

[18] J. M. Lee, *Introduction to Smooth Manifolds* (Graduate Texts in Mathematics), 2nd ed. Springer New York, NY, Aug. 26, 2012.

[19] P. J. Olver, *Classical Invariant Theory* (London Mathematical Society Student Texts). Cambridge: Cambridge University Press, 1999.

[20] E. Leurent, *An environment for autonomous driving decision-making*, https://github.com/eleurent/highway-env, 2018.

[21] G. Brockman, V. Cheung, L. Pettersson, *et al.*, *OpenAI Gym*, 2016. arXiv: 1606.01540 [cs.LG].

[22] T. Seno and M. Imai, "D3rlpy: An offline deep reinforcement learning library," *Journal of Machine Learning Research*, vol. 23, no. 315, pp. 1–20, 2022.