Supplementary Materials for "PI-SONet: A Physics-Informed Symplectic Operator Network for Real-Time Optimal Control of Multi-Agent Systems"*

1 Implementation Details

1.1 Block diagonal structure of K

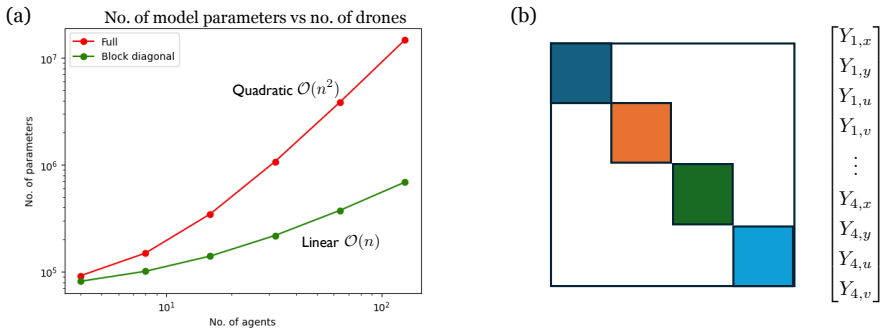


Fig. 1: **Block diagonal structure of K.** (a) Scaling of the number of SONet parameters with the number of drones for full vs. block-diagonal parameterizations of K. (b) Schematic illustration of the block-diagonal structure we impose for K, where each diagonal block corresponds to an individual drone and cross-drone blocks are set to zero. By imposing a relatively sparse structure for K, we balance expressivity of SONet with computational scalability. In our numerical experiments, we do not observe any noticeable degradation in accuracy as a result of this sparsity.

The matrix K has dimension $Nd_x \times Nd_x$, where N denotes the number of drones and d_x is the dimension of the state space (with $d_x = 4$ in our planar experiments and $d_x = 6$ in our 3D experiments). As the number of drones increases, learning a full K

*Distribution Statement A. Approved for Public Release; Distribution is unlimited. PR 26-0070.

matrix quickly becomes impractical since the output layer of the network that predicts K scales quadratically with n, resulting in prohibitively larger numbers of parameters.

To address this issue, we impose a block-diagonal structure on the matrix K. We assume that only the blocks corresponding to individual drones are nonzero, while blocks associated with cross-drone terms are set to zero. This parameterization of K matrix reduces the number of model parameters and improves scalability.

In addition, the the network prediction K shares weights and biases across all pre-final layers. The final layer is implemented using multiple heads (as shown in Figure 5(d)), where each head outputs the block of K corresponding to a single drone. This design further improves parameter efficiency while maintaining sufficient representational capacity. Empirically, we observe that enforcing a block-diagonal structure on K does not degrade the performance of the operator network in our experiments.

No. of drones	Full	Block diagonal
4	91,848	81,480
8	149,448	101,064
16	347,592	140,232
32	1,075,656	218,568
64	3,858,888	375,240

Table 1: **Number of model parameters for full vs. block-diagonal parameterizations of K as a function of the number of drones.** Full, dense structure quickly leads to infeasibly large numbers of network parameters, whereas the block diagonal structure significantly mitigates this effect.

Figure 1 illustrates how the number of parameters in SONet scales with the number of drones for both the full and block-diagonal parameterizations of K; corresponding values are also reported in Table 1. When a full K matrix is used, the number of model parameters scales quadratically with the number of drones. In contrast, the block-diagonal structure leads to linear scaling with respect to n, resulting in a substantially more parameter-efficient model.

1.2 Latent solver

1.2.1 LQR solver with rotation

The standard LQR formulation provides an efficient latent solver for linear dynamics but implicitly favors straight-line trajectories. In multi-agent settings with many drones operating in a confined domain, standard LQR induces a degenerate topology; trajectories tend to intersect near the center of the domain, creating a high-density

bottleneck. As a result, the downstream operator must significantly deform these intersecting trajectories to satisfy collision constraints, leading to optimization difficulties and poor scalability.

In the space-time domain $[0, T] \times \mathbb{R}^2$, the joint trajectories of the agents form a braid. Feasible, collision-free solutions correspond to non-intersecting strands, i.e., trajectories lying in a non-trivial homotopy class of the braid space [1]. For symmetric initial and terminal configurations, these feasible solutions typically exhibit a rotational structure, whereas the standard LQR solution lies in a degenerate class with strand intersections.

To address this mismatch while retaining the efficiency of the standard LQR solver, we introduce a rotational component into the latent dynamics. Specifically, we augment the system matrix with a skew-symmetric term:

$$A_{\text{rot}} = \begin{pmatrix} 0 & I \\ 0 & \Omega \end{pmatrix}, \quad \Omega = \begin{pmatrix} 0 & -C_B \\ C_B & 0 \end{pmatrix}. \tag{1}$$

This modification induces curvature in the latent trajectories, encouraging coordinated rotational motion, thereby avoiding the central bottleneck. This skew-symmetric structure ensures that this component does not introduce artificial dissipation, preserving the conservative nature of the dynamics.

From a geometric perspective, this rotational bias steers the latent trajectories toward a non-intersecting braid class, providing a more suitable initialization for the downstream operator. Empirically, this behavior leads to significantly improved robustness and scalability compared to standard LQR, particularly in high-density regimes (see Section 3.1).

1.2.2 Eikonal reference solver

To find a PMP trajectory in a highly non-convex environment one has to initialize the trajectory properly so that the loss converges to a good basin of attraction. The reference trajectory solves a stochastic particle system via Euler-Maruyama on the SDE

$$d\boldsymbol{X}_t = f(\boldsymbol{X}_t, t)\, dt + \sigma\, dW_t,$$

where $\boldsymbol{X}_t \in \mathbb{R}^{2N}$ stacks the positions of N agents. Drift is given by $f = G \cdot \boldsymbol{v}(\boldsymbol{X})$, where $G \approx I$ is a perturbed-identity Riemannian metric and $\boldsymbol{v} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_N)$ $(\boldsymbol{v}_i \in \mathbb{R}^2, \forall i)$ is a sum of vector fields acting on each agent i given by

$$\boldsymbol{v}_i(X) = \boldsymbol{v}_{\text{rep},i}(\boldsymbol{X}) + \boldsymbol{v}_{\text{wall},i}(\boldsymbol{X}) + \boldsymbol{v}_{\text{nav},i}(\boldsymbol{X}),$$

where inter-agent repulsion (gradient of a log-barrier) is given by

$$\boldsymbol{v}_{\text{rep},i}(\boldsymbol{X}) = -\sum_{j \neq i} \frac{\boldsymbol{x}_i - \boldsymbol{x}_j}{c_1 \left(\|\boldsymbol{x}_i - \boldsymbol{x}_j\| - 2r_{\text{agent}} \right)^2},$$

wall repulsion (gradient of an inverse-square barrier from the closest segment of the wall $\pi_k(\boldsymbol{x}_i)$) is given by

$$\boldsymbol{v}_{\text{wall},i}(\boldsymbol{X}) = \sum_k \frac{\boldsymbol{x}_i - \pi_k(\boldsymbol{x}_i)}{c_2 \left(\|\boldsymbol{x}_i - \pi_k(\boldsymbol{x}_i)\| - r_{\text{agent}} - r_{\text{obs}} \right)^3},$$

and the navigation vector field is given by

$$\boldsymbol{v}_{\text{nav},i}(\boldsymbol{X}) = -\frac{\nabla u(\boldsymbol{x}_i)}{\|\nabla u(\boldsymbol{x}_i)\|},$$

which is evaluated by bilinear interpolation of the precomputed gradient field of the Eikonal navigation $u : \mathbb{R}^2 \to \mathbb{R}$. We take u to be the solution to the Eikonal equation

$$\|\nabla u(\boldsymbol{x})\| = \frac{1}{s(\boldsymbol{x})}, \quad u(\boldsymbol{x}^*) = 0,$$

where $s(\boldsymbol{x}) \approx 0$ inside obstacles and $s(\boldsymbol{x}) = 1$ in free space. Note that $\boldsymbol{v}_{\text{nav},i}$ is not the gradient of any closed-form potential. Hence, we compute the gradient as follows. Five trials are run with different random seeds; the best run is selected by minimizing the detour ratio:

$$\mathcal{C} = \max_i \frac{\int_0^T \|\dot{\boldsymbol{x}}_i(t)\| \, dt}{\|\boldsymbol{x}_i(0) - \boldsymbol{x}_i^*\|}.$$

Next, we rescale the resulting trajectory in time to match the final time of the original optimal control problem. The rescaled reference trajectory is then used to pretrain the SONet by training the loss function $\mathcal{L}(\theta) = \|\boldsymbol{x}_\theta - \boldsymbol{x}_{ref}\|^2$, where $\boldsymbol{x}_\theta$ represents the transformed trajectory. We note that in our experiments we used this only for the *Maze* case.

1.3 Perturbed initial conditions

1.3.1 Crossing a square ('free'): scalability to large swarms

To evaluate the scalability of PI-SONet to high-dimensional state spaces, we consider a multi-agent room-crossing task in the absence of obstacles. The agents must move from perturbed initial positions to antipodal target locations while avoiding inter-agent collisions. The complexity of this problem arises purely from pairwise interactions, which scale quadratically with the number of drones. We consider swarm sizes ranging from $N = 4$ to $N = 64$ agents.

The SONet architecture consists of 3 up layers and 3 down layers across all cases, with a network width of 8 for both K and b. Latent trajectories are generated using an LQR solver with rotation.

To ensure feasibility as the swarm density increases, we adjust the physical parameters across different regimes. For $N \in \{4, 8, 16, 32\}$, the drone radius is fixed at 0.020 with a perturbation radius of 0.05. For denser swarms ($N = 56$ and $N = 64$), the drone radius is reduced to 0.016, and the perturbation radius is adjusted to 0.04 and 0.025, respectively.

Training strategies differ between low- and high-dimensional regimes. For smaller swarms ($N \in \{4, 8, 16\}$), we use constant penalty parameters $\epsilon = 10^{-4}$ and $\lambda = 10^{-4}$ and train using 150 Adam steps followed by 100 L-BFGS steps. For larger swarms ($N \in \{32, 56, 64\}$), we employ an annealing strategy, initializing ϵ and λ at 0.1 and decaying them by a factor of 0.6 every 20 epochs. In these cases, the model is trained using 100 Adam epochs followed by 100 L-BFGS steps. Additionally, weight decay (L^2 regularization) is applied across all experiments with values ranging from 10^{-6} to 5×10^{-5}.

Figure 2 visualizes representative trajectories for two test samples for each swarm size. PI-SONet demonstrates strong scalability, achieving 100% collision-free success rates on the training set across all swarm sizes. Generalization to unseen test perturbations remains robust: 99% for 4 agents, 95% for 8 agents, 83% for 16 agents, 97% for 32 agents, 99% for 56 agents, and 100% for 64 agents.

In addition to accuracy, PI-SONet exhibits consistently low inference latency across all swarm sizes. On an NVIDIA H100 GPU, the average batched forward-pass time remains on the order of 10^{-2} seconds with measured runtimes of 0.018s ($N = 4$), 0.015s ($N = 8$), 0.019s ($N = 16$), 0.021s ($N = 32$), 0.019s ($N = 56$), and 0.019s ($N = 64$). All inference times (in this and all other test scenarios) are computed on a single batch consisting of all test samples. These results indicate that the method maintains efficient inferences even as the number of agents increases, thereby enabling real-time trajectory generation in practice.

No. of drones	Drone radius	Perturbation radius	Annealing	Weight decay (λ)	C_B	C_Q
4	0.020	0.05	No	Yes ($1e-6$)	$\pi/20$	1.0
8	0.020	0.05	No	Yes ($1e-6$)	$\pi/20$	0.1
16	0.020	0.05	No	Yes ($5e-5$)	$\pi/20$	0.05
32	0.016	0.05	Yes	Yes ($1e-6$)	$\pi/10$	0.001
56	0.016	0.04	Yes	Yes ($1e-6$)	$\pi/10$	0.001
64	0.016	0.025	Yes	Yes ($1e-6$)	$\pi/10$	0.001

Table 2: **Hyperparameters and physical settings for obstacle-free scalability experiments.** The table details the configuration for swarms of varying sizes N navigating an obstacle-free environment. As the agent density increases ($N \geq 32$), the drone radii and initial perturbation radii are reduced to maintain geometric feasibility. Additionally, training stability is enforced through annealing, weight decay (λ), and adjusted coefficients for the rotation prior (C_B) and quadratic velocity (C_Q) cost terms in the latent LQR solver.

These results suggest that the combination of the symplectic prior and the annealing strategy effectively structures the optimization landscape, such that PI-SONet avoids poor local minima and maintains high performance even in high-dimensional, dense-interaction settings.

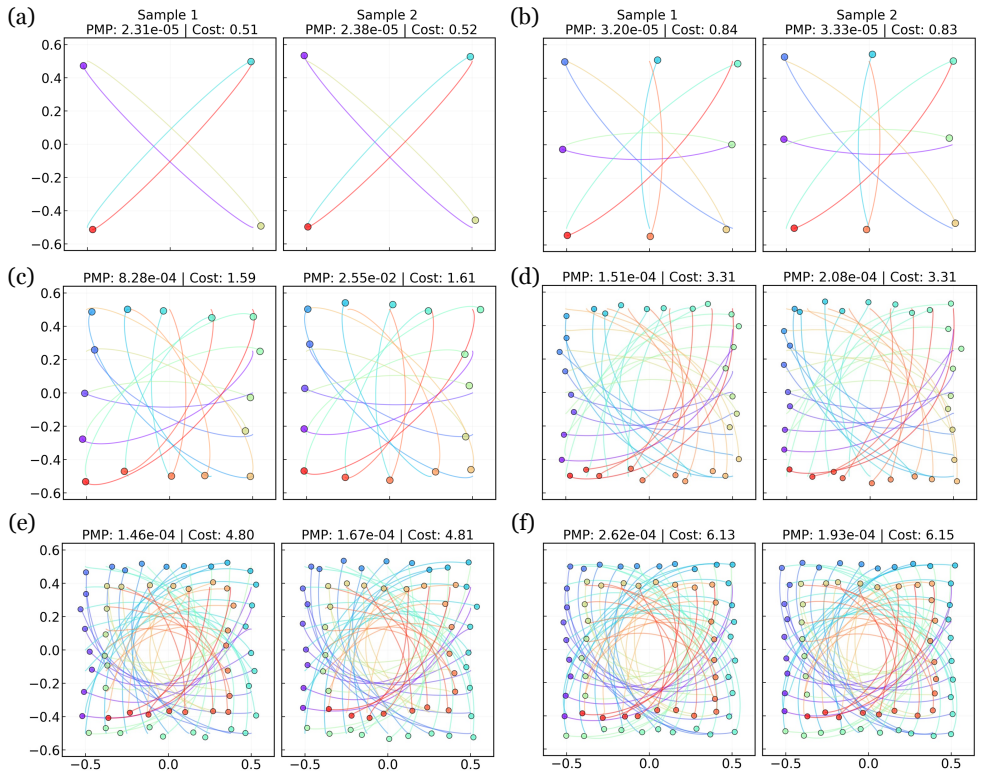


Fig. 2: **Scalability in an obstacle-free room.** Trajectories for swarms of varying sizes ($N \in \{4, \ldots, 64\}$) navigating an obstacle-free room. For each swarm size, we show two samples of perturbed initial positions from the test dataset. PI-SONet exhibits high optimality, high feasibility, and real-time inferences across all swarm sizes, demonstrating efficient scalability for high-dimensional trajectory planning problems.

# Drones	Pass (Train)	Pass (Test)	Avg PMP (Train)	Avg PMP (Test)	Avg L (Train)	Avg L (Test)
4	100/100	99/100	9.02×10^{-5}	1.71×10^{-4}	0.511	0.512
8	100/100	95/100	7.93×10^{-5}	1.21×10^{-3}	0.826	0.828
16	100/100	83/100	1.14×10^{-3}	7.70×10^{-3}	1.614	1.612
32	100/100	97/100	1.33×10^{-4}	7.39×10^{-4}	3.313	3.315
56	100/100	99/100	1.38×10^{-4}	2.50×10^{-4}	4.807	4.809
64	100/100	100/100	1.91×10^{-4}	2.12×10^{-4}	6.123	6.122

Table 3: **Performance of PI-SONet for the 'free' case with perturbed initial position.** PI-SONet exihibits high optimality and feasibility across all swarm sizes tested.

1.3.2 Crossing a square with a circular obstacle ('obstacle'): Scalability to Large Swarms

We extend the scalability analysis to include a static environmental constraint. In this setting, a swarm of size $N \in \{4, \ldots, 64\}$ performs an antipodal position swap while avoiding a fixed circular obstacle of radius $r_{\text{obs}} = 0.15$ located at the center of the domain. The objective remains to reach target states from perturbed initial positions without inter-agent or obstacle collisions.

The presence of the obstacle introduces a nontrivial topological constraint by blocking direct straight-line paths between antipodal pairs. As a result, agents must learn coordinated trajectories that circumnavigate the obstacle. We employ LQR with rotation and pretrained TSympOCNet as the TRS solver to guide trajectory generation.

The SONet architecture consists of 3 up layers and 3 down layers across all cases. The network width for K and b is set to 4 for $N \in \{4, 8, 16\}$, 16 for $N = 32$, and 8 for $N \in \{56, 64\}$.

Training strategies vary with swarm size. For $N \in \{4, 8, 16\}$, we use constant penalty parameters $\epsilon = \lambda = 10^{-4}$ and train using 150 Adam steps followed by 200 L-BFGS steps. For $N = 32$, we increase the penalty parameters to $\epsilon = \lambda = 10^{-3}$ and train using 150 Adam steps followed by 50 L-BFGS steps. In the aforementioned low-dimensional cases ($N \leq 32$), no annealing is applied.

For larger swarms ($N \in \{56, 64\}$), we employ annealing to stabilize optimization in the high-dimensional regime. The penalty parameters are initialized at $\epsilon = \lambda = 0.1$ and decayed by a factor of 0.6 every 20 steps. The model is trained using 100 Adam steps followed by 50 L-BFGS steps for $N = 56$ and 100 Adam steps followed by 30 L-BFGS steps for $N = 64$.

The inclusion of the obstacle yields consistently strong performance. PI-SONet achieves a 100% collision-free success rate on the training set across all swarm sizes. Generalization to unseen test perturbations remains robust, with success rates of 100% (4 agents), 94% (8 agents), 99% (16 agents), 97% (32 agents), 99% (56 agents), and 98% (64 agents).

In addition to accuracy, PI-SONet maintains consistently low inference latencies across all swarm sizes in the presence of obstacles. On an NVIDIA H100 GPU, the average forward-pass time per batch remains on the order of 10^{-2} seconds with measured runtimes of 0.017s ($N = 4$), 0.017s ($N = 8$), 0.019s ($N = 16$), 0.018s ($N = 32$), 0.019s ($N = 56$), and 0.020s ($N = 64$). These results demonstrate that the method retains efficient inferences even when faced with additional geometric constraints, which shows PI-SONet's potential for real-time trajectory generation in obstacle-rich environments.

Interestingly, performance in the obstructed setting is comparable to, and in some cases slightly better than, the obstacle-free case. This behavior can be attributed to the geometric role of the obstacle; in the absence of obstacles, antipodal trajectories tend to converge toward the origin, creating a high-density bottleneck. The central obstacle eliminates this degeneracy by forcing agents to circumnavigate the center, inducing a coordinated circular flow that naturally distributes agents in space and reduces collision complexity despite the reduced free volume.

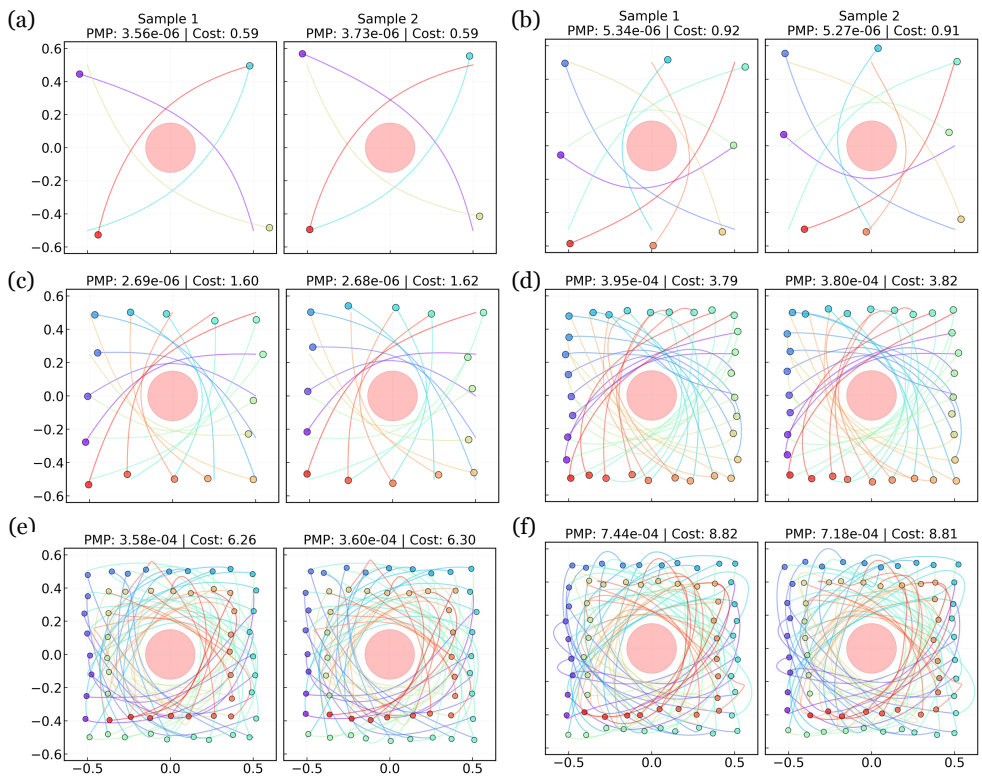


Fig. 3: **Scalability with a central obstacle.** Trajectories for swarms of sizes ($N \in \{4, \ldots, 64\}$) navigating around a fixed circular obstacle ($r = 0.15$). The obstacle prevents agents from crowding the center of the domain, inducing a coordinated circular flow that simplifies inter-agent avoidance. As such, PI-SONet's performance in this scenario is in some cases better than that in the obstacle-free scenario. For each case, we show two test samples with perturbed initial positions.

Figure 3 illustrates representative trajectories for this circular obstacle scenario, while Tables 4 and 5 summarize the corresponding experimental setup details and quantitative performance results.

1.3.3 Crossing a square with a maze: handling nonconvex constraints

We evaluate PI-SONet on a challenging nonconvex optimal control problem – multi-agent navigation through a maze. Unlike the convex obstacle setting, the maze introduces complex nonconvex and nonsmooth topological constraints that require agents to traverse narrow corridors and avoid dead-ends while maintaining collision-free trajectories.

We consider swarms of $N \in \{4, 8\}$ agents navigating from perturbed initial positions to a fixed target configuration. For both swarm sizes, perturbations are sampled

No. of drones	Drone radius	Perturbation radius	TRS Solver	Annealing	Weight decay (λ)
4	0.020	0.10	LQR	No	No
8	0.020	0.10	LQR + Pretrained TSympOCNet	No	No
16	0.020	0.05	LQR + Pretrained TSympOCNet	No	No
32	0.020	0.025	LQR + Pretrained TSympOCNet	No	Yes ($1e-5$)
56	0.016	0.025	LQR + Pretrained TSympOCNet	Yes	Yes ($1e-6$)
64	0.016	0.025	LQR + Pretrained TSympOCNet	Yes	Yes ($1e-6$)

Table 4: **Hyperparameters and physical settings for scalability experiments in the presence of a circular obstacle.** The table details the configuration for swarms of varying sizes N navigating around a static circular obstacle. As the agent density increases ($N \geq 32$), the drone radius and initial perturbation radius are reduced to maintain geometric feasibility. Additionally, training stability is enforced through annealing, weight decay (λ).

# Drones	Pass (Train)	Pass (Test)	Avg PMP (Train)	Avg PMP (Test)	Avg L (Train)	Avg L (Test)
4	100/100	100/100	3.15×10^{-6}	3.42×10^{-6}	0.579	0.580
8	100/100	94/100	4.82×10^{-6}	1.29×10^{-1}	0.886	0.890
16	100/100	99/100	2.47×10^{-6}	6.18×10^{-3}	1.625	1.623
32	100/100	97/100	3.91×10^{-4}	7.17×10^{-4}	3.806	3.807
56	100/100	99/100	3.60×10^{-4}	3.83×10^{-4}	6.275	6.276
64	100/100	98/100	7.15×10^{-4}	7.60×10^{-4}	8.770	8.762

Table 5: **Performance of PI-SONet for the 'obstacle' case with perturbed initial position.** PI-SONet maintains high feasibility and optimality across all tested swarm sizes.

uniformly from a ball of radius 0.05 around a nominal initial configuration. We generate 100 training and 100 testing samples.

Due to the highly nonconvex environment geometry, LQR-based latent trajectories alone are insufficient to capture the correct solution topology. To address this issue, we construct reference trajectories using an Eikonal solver for the unperturbed initial configuration. We observe that the resulting trajectories do indeed encode the correct homotopy class (i.e., feasible paths through the maze).

We first pretrain the SONet to regress these reference trajectories (positions only, ignoring velocity and co-state) using 500 Adam steps. This stage provides a good initialization that ensures that the predicted trajectories lie in the correct topological class, although they may be suboptimal and may still contain collisions.

Starting from this initialization, we further train the model using the PMP loss to enforce optimality and collision avoidance. In both cases, we do not employ annealing. Instead, we use fixed penalty parameters: $\epsilon = \lambda = 10^{-4}$ for 4 drones and $\epsilon = \lambda = 10^{-3}$ for 8 drones.

The SONet architecture in this case consists of 3 up layers and 3 down layers. The network width for K and b is set to 4 for 4 drones and 8 for 8 drones. For the 4-agent case, the model is trained using 200 Adam steps followed by 200 L-BFGS steps. For the 8-agent case, we use 300 Adam steps followed by 100 L-BFGS steps.

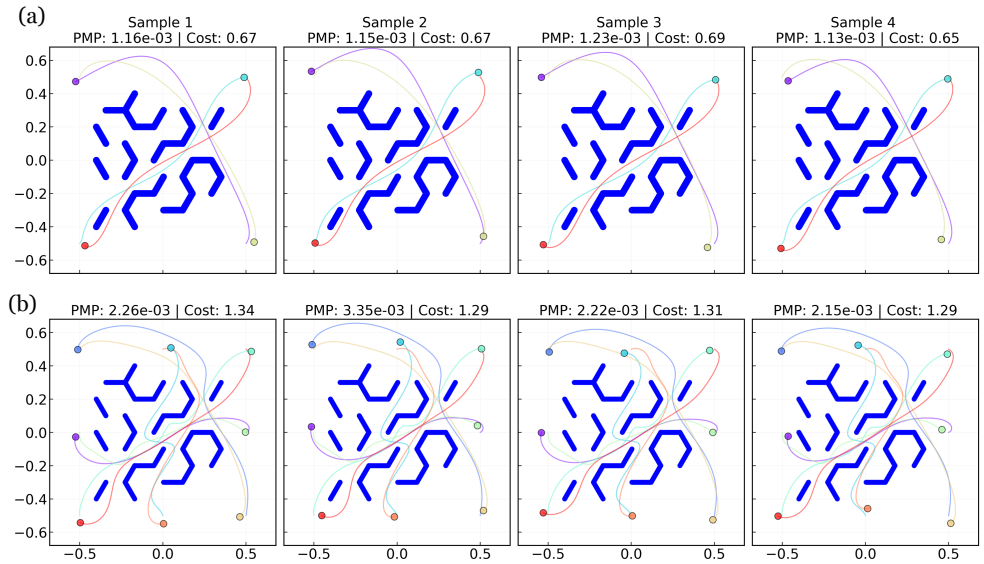


Fig. 4: **Multi-agent navigation in a non-convex maze. (a)** Sample trajectories for 4 agents under perturbed initial conditions. The agents successfully navigate the blue obstacles (maze walls) to reach the target configuration. **(b)** Results for 8-agent swarm under perturbed initial conditions. Despite the high density and narrow corridors, PI-SONet, aided by adaptive activation and annealing, generates collision-free trajectories for the majority of test cases.

Table 6 summarizes the qualitative performance in this maze setting. We observe that PI-SONet achieves strong performance for both swarm sizes with high collision-free success rates on both the training and test sets. For 4 agents, the model achieves 100% success on the training set and 97% on the test set. For 8 agents, the model achieves 100% training success and 95% test success.

Inference remains efficient even in this non-convex setting. On an NVIDIA H100 GPU, the average batched forward-pass time is 0.022s for $N = 4$ and 0.020s for $N = 8$. These results indicate that the added topological complexity of the maze does not significantly impact runtime, allowing fast trajectory generation even in environments with narrow corridors and constrained feasible regions.

Moreover, the results in this setting highlight the importance of incorporating topology-aware latent trajectories in environments with complex geometries. In particular, the Eikonal-based pretraining effectively guides the model toward feasible homotopy classes, while the PMP-based refinement enforces optimality and collision avoidance. Despite the increased difficulty due to narrow corridors and dense agent interactions, PI-SONet successfully learns a robust solution operator for this highly constrained setting.

# Drones	Pass (Train)	Pass (Test)	Avg PMP (Train)	Avg PMP (Test)	Avg L (Train)	Avg L (Test)
4	100/100	97/100	1.17×10^{-3}	1.04×10^{-2}	0.667	0.668
8	100/100	95/50	2.23×10^{-3}	2.60×10^{-3}	1.295	1.301

Table 6: **Performance of PI-SONet for swarms navigating a 2D maze with perturbed initial positions.** Despite the complex environment geometry, PI-SONet exhibits high feasibility and optimality, demonstrating the power of topology-aware latent right-space solvers and physics-informed training losses.

1.4 Perturbed obstacle geometries

We evaluate the ability of PI-SONet to approximate the solution operator across varying geometric constraints. Specifically, we consider a multi-agent optimal control problem in which $N \in \{4, 8, 16\}$ drones navigate around a central circular obstacle of variable radius r.

To isolate the effect of the obstacle geometry, the initial and terminal positions of the agents are fixed across all samples, while the obstacle radius is sampled as $r \sim \mathcal{U}[0.05, 0.25]$. For each setting, we generate 50 training and 50 testing samples, where each sample corresponds to a different obstacle size. The latent trajectories are generated using LQR with rotation with parameters $C_B = -\pi/20$ and $C_Q = 1$.

All models use 3 up layers and 3 down layers. The network width for K and b is set to 8 for the 4- and 8-drone cases and 16 for the 16-drone case. Each drone has a fixed radius $C_r = 0.02$. No explicit regularization is used during training. Models are trained for 100 epochs using Adam, followed by 200 epochs using L-BFGS. We employ an annealing strategy for the penalty parameters ϵ and λ, initialized at 0.1 and decayed by a factor of 0.6 every 20 epochs.

# Drones	Pass (Train)	Pass (Test)	Avg PMP (Train)	Avg PMP (Test)	Avg L (Train)	Avg L (Test)
4	50/50	50/50	1.24×10^{-4}	2.30×10^{-4}	0.556	0.559
8	50/50	50/50	4.53×10^{-4}	5.40×10^{-4}	0.838	0.846
16	50/50	48/50	4.65×10^{-4}	5.38×10^{-4}	1.549	1.565

Table 7: **Performance of PI-SONet for swarms navigating around a circular obstacle of perturbed radius.** PI-SONet achieves high optimality and feasibility, indicating its ability to capture abstract problem features, such as obstacle geometry.

Table 7 summarizes the performance across different agent counts. PI-SONet achieves perfect or near-perfect feasibility on both training and test sets for 4 and 8 drones with only a slight drop for 16 drones (48/50). The PMP residuals remain low across all cases, indicating accurate satisfaction of optimality conditions. As expected, the average cost increases with the number of agents due to higher interaction complexity and tighter spatial constraints.

Inference remains fast across varying obstacle geometries. On an NVIDIA H100 GPU, the average batched forward-pass time is 0.019s for $N = 4$, 0.019s for $N = 8$,

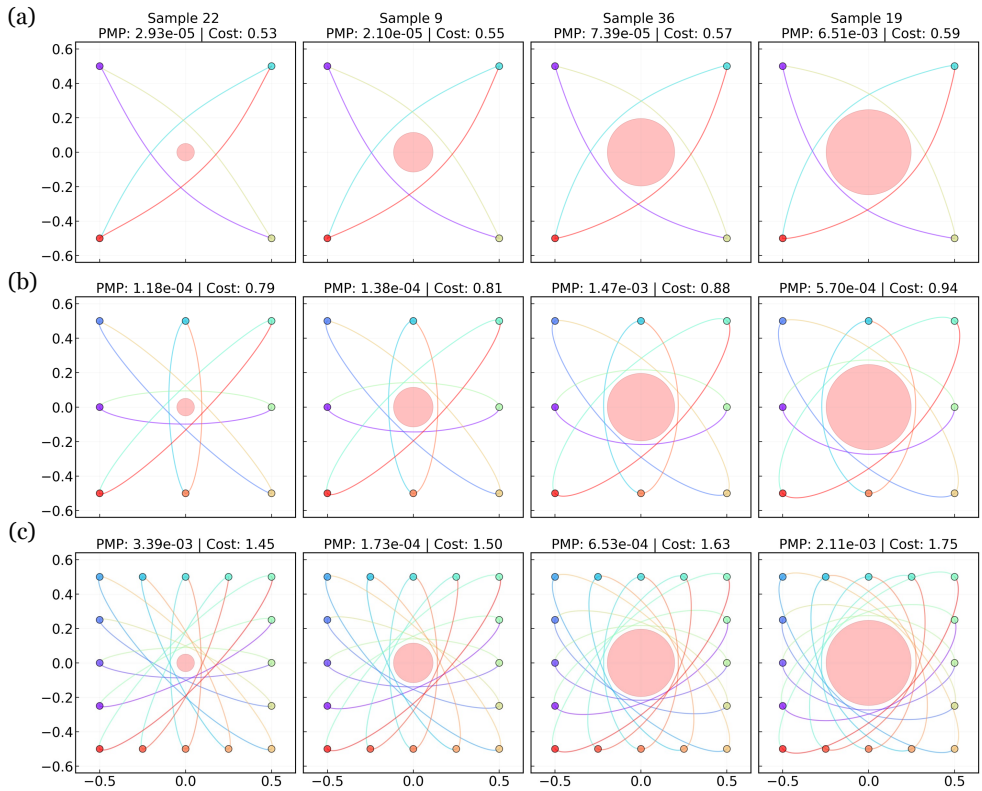


Fig. 5: **PI-SONet under varying obstacle geometries.** We consider drones navigating around a central obstacle with radius $r \in [0.05, 0.25]$. Each column corresponds to a different obstacle radius (increasing from left to right), while rows (a)–(c) denote increasing number of drones, 4, 8 and 16. The initial positions are fixed across all cases to isolate the effect of geometry. PI-SONet successfully adapts to changing obstacle sizes, maintaining feasible, collision-free trajectories across most configurations, with increasing cost as the obstacle becomes larger and constrains the feasible space.

and 0.019s for $N = 16$. These results indicate that changes in obstacle size do not introduce any noticeable computational overhead, allowing rapid trajectory generation even as the feasible region undergoes significant geometric variation.

Figure 5 illustrates representative trajectories under varying obstacle sizes. PI-SONet adapts effectively to changing geometries, producing collision-free trajectories across most configurations. As the obstacle radius increases, the feasible region shrinks, leading to more curved trajectories and higher costs, while still preserving feasibility in the majority of cases.

1.5 Perturbed drone radius: heterogenous setting

To test the ability of PI-SONet to handle settings with heterogeneous capabilities, we consider scenarios where each drone in the swarm has a different radius.

1.5.1 Two-dimensional room crossing

We consider a two-dimensional room-crossing task with heterogeneous drone radii. The dynamics for each drone is now given by

$$\dot{\boldsymbol{x}}_i = \boldsymbol{v}_i, \quad \dot{\boldsymbol{v}}_i = \boldsymbol{u}_i - k_i \boldsymbol{v}_i \|\boldsymbol{v}_i\|_2, \tag{2}$$

where the resistance coefficient k_i depends on the drone radius r_i. Using the drag relation $F_d \propto \rho C_d A v^2$ and noting that in the 2D setting the projected area scales as $A \propto r_i$ (top-down footprint) while the mass scales as $m_i \propto r_i^2$, we obtain

$$m_i k_i \propto A \;\; \Rightarrow \;\; r_i^2 k_i \propto r_i \;\; \Rightarrow \;\; k_i \propto \frac{1}{r_i}.$$

Intuitively, larger drones have more inertia (mass grows as r_i^2) relative to the drag they experience (which grows only linearly with r_i), leading to a smaller effective resistance coefficient. Conversely, smaller drones experience stronger resistance coefficient, resulting in more velocity-aligned controls.

To evaluate generalization across heterogeneous capabilities, we sample drone radii as $r_i \sim \mathcal{U}[0.01, 0.10]$ for 4 drones and $r_i \sim \mathcal{U}[0.01, 0.05]$ for 8 and 16 drones. For each setting, we generate 50 training and 50 testing samples, where each sample corresponds to a different combination of radii. The obstacle radius is fixed at $C_o = 0.15$ in all cases.

We use LQR with rotation as the latent trajectory solver with parameters $C_B = \pi/20$ and $C_Q = 1.0$. The SONet architecture consists of 3 up layers and 3 down layers with a network width of 64 for both K and b. No explicit regularization is used. The resistance coefficient is implemented as $k_i = 1/(50 r_i)$.

Models are trained using a combination of Adam and L-BFGS. For 4 drones, we use 100 Adam epochs followed by 100 L-BFGS epochs, for 8 drones, 100 Adam and 200 L-BFGS epochs, and for 16 drones, 100 Adam and 250 L-BFGS epochs. We employ an annealing strategy for the penalty parameters (ϵ, l). For 4 and 8 drones, both are initialized at 0.1 and decayed by a factor of 0.6 every 20 epochs. For 16 drones, they are initialized at 0.01 and decayed by a factor of 0.8 every 20 epochs.

Table 8 summarizes the qualitative performance in this scenario across different agent counts. PI-SONet achieves near-perfect feasibility for 4 and 8 drones with only slight degradation for 16 drones due to the corresponding increased agent-interaction complexity. The PMP residuals remain low across all cases, indicating accurate satisfaction of optimality conditions.

Inference remains efficient under heterogeneous agent dynamics. On an NVIDIA H100 GPU, the average batched forward-pass time is 0.016s for $N = 4$, 0.018s for $N = 8$, and 0.017s for $N = 16$. These results indicate that variations in agent-specific parameters (e.g., here, radius-dependent resistance coefficients) do not significantly

# Drones	Pass (Train)	Pass (Test)	Avg PMP (Train)	Avg PMP (Test)	Avg L (Train)	Avg L (Test)
4	50/50	49/50	1.15×10^{-5}	1.71×10^{-5}	0.591	0.591
8	50/50	50/50	1.06×10^{-5}	2.85×10^{-5}	0.865	0.865
16	50/50	48/50	1.18×10^{-5}	3.49×10^{-2}	1.640	1.640

Table 8: **Performance of PI-SONet for swarms of heterogeneous perturbed drone radii navigating around a circular obstacle in 2D.** PI-SONet demonstrates high feasibility and high optimality, indicating its ability to generalize to heterogeneous agent capabilities.

impact runtime, indicating that a pre-trained PI-SONet could be trivially transferred in real-time between swarms of slightly different capabilities.

Figure 6 shows representative trajectories at a fixed time. PI-SONet adapts effectively to heterogeneous radii, producing collision-free trajectories in most cases. As the radius decreases (larger k_i), the control becomes increasingly aligned with the velocity, reflecting stronger damping effects.

1.5.2 Three-dimensional swarm

We consider a three-dimensional swarm navigation task with $N = 100$ drones in the presence of two rectangular obstacles to demonstrate that PI-SONet can be applied to very high-dimensional cases that move beyond planar physical spaces. In this scenario, the agents must move from their initial positions to their target locations while avoiding collisions with both rectangular obstacles in the center of the domain and other drones. The obstacles are defined as axis-aligned boxes:

$$\text{Obstacle 1:} \quad -2 \le x \le 2,\ -0.5 \le y \le 0.5,\ 0 \le z \le 7, \tag{3}$$

$$\text{Obstacle 2:} \quad 2 \le x \le 4,\ -1 \le y \le 1,\ 0 \le z \le 4. \tag{4}$$

The dynamics are linear:

$$\dot{\boldsymbol{x}}_i = \boldsymbol{v}_i, \quad \dot{\boldsymbol{v}}_i = \boldsymbol{u}_i, \tag{5}$$

and the objective is to minimize the control effort $\int \frac{1}{2}\|\boldsymbol{u}_i\|^2\, dt$.

To evaluate robustness to heterogeneous agent sizes, the drone radii are sampled as $r_i \sim \mathcal{U}[0.1, 0.2]$. We generate 50 training and 50 testing samples, where each sample corresponds to a different realization of drone radii.

The SONet architecture consists of 3 up layers and 3 down layers, with a hidden dimension of 32 for the networks corresponding to K and b. We use LQR-generated latent trajectories. To improve training stability, we first train a SONet model for the homogeneous-radius setting and use its weights to initialize the model for the heterogeneous case. This warm-start significantly stabilizes optimization in the high-dimensional, dense-interaction regime.

The model is trained using 100 Adam epochs followed by 50 L-BFGS epochs. We employ annealing for the penalty parameters with ϵ initialized at 5×10^{-2} and l at 5×10^{-3}, both decayed by a factor of 0.8 every 100 epochs.

PI-SONet successfully scales to large swarms with heterogeneous agent sizes, generating collision-free trajectories while maintaining low control costs. The results

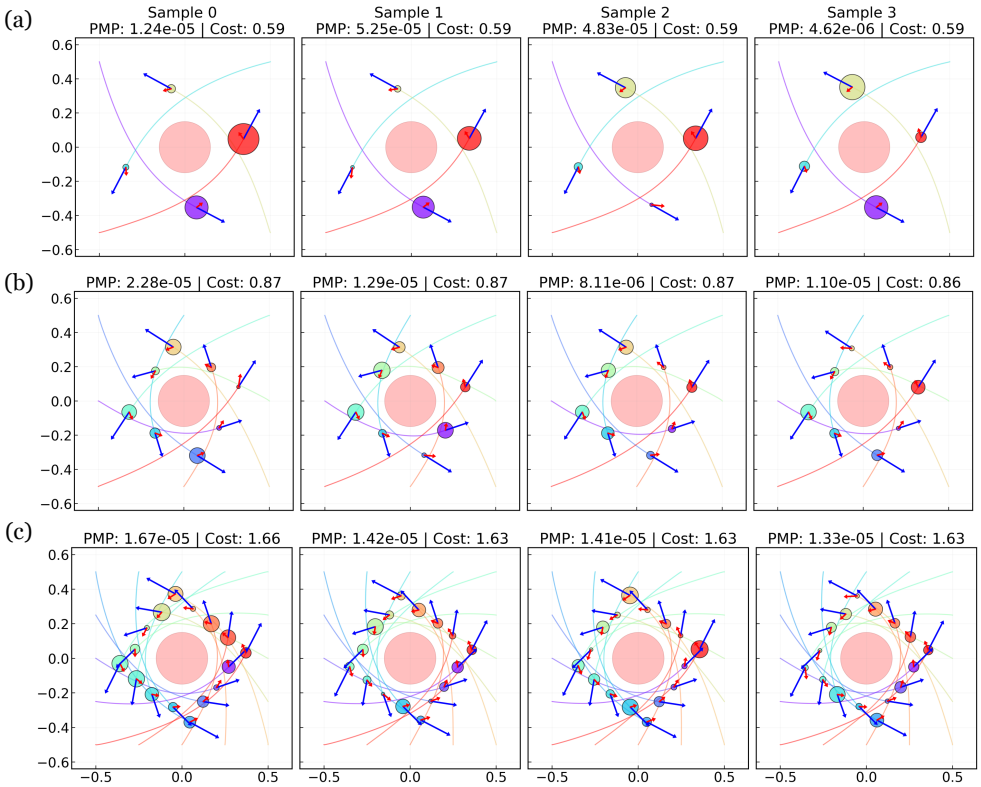


Fig. 6: **PI-SONet for swarms with heterogeneous drone radii.** Snapshots at fixed time $t = 6.7$ for (a) 4, (b) 8, and (c) 16 drones. Each column corresponds to a different random realization of radii $r_i \sim \mathcal{U}[0.01, 0.10]$ for 4 drones and $r_i \sim \mathcal{U}[0.01, 0.05]$ for 8 and 16 drones. The obstacle is shown in red, while colored curves denote agent trajectories. PI-SONet adapts to heterogeneous agent sizes, producing collision-free trajectories in most cases. As the radius decreases (larger k_i), the control becomes more aligned with the velocity, reflecting the expected physical response to larger resistance coefficients.

demonstrate that our approach mitigates the curse of dimensionality and that the resulting operator can generalize across both high-dimensional state spaces and varying physical agent properties.

1.5.3 Urban city environment

We next consider a more realistic two-dimensional urban navigation scenario, motivated by drone delivery in city environments. In this setting, a swarm of drones must navigate from prescribed initial positions to target locations while avoiding both inter-agent collisions and collisions with buildings. The environment consists of multiple rectangular obstacles, which represent buildings, together with open regions that

represent parks or other flyable spaces. Unlike the room-crossing examples with a single obstacle, this setting introduces a more complex obstacle geometry and multiple feasible homotopy classes for the trajectories.

To model heterogeneous drone capabilities, we assign each drone a different radius. This heterogeneity represents drones carrying packages of different sizes, where larger packages increase the effective physical footprint of the drone. For each problem instance, the drone radii are sampled independently as $r_i \sim \mathcal{U}[0.005, 0.025]$. The dynamics of each drone are given by

$$\dot{\boldsymbol{x}}_i = \boldsymbol{v}_i, \qquad \dot{\boldsymbol{v}}_i = \boldsymbol{u}_i - k_i \boldsymbol{v}_i ||\boldsymbol{v}_i||_2, \tag{6}$$

where the resistance coefficient is radius-dependent and defined as $k_i = \frac{1}{200 r_i}$. Thus, smaller drones experience a larger effective resistance coefficient, while larger drones have relatively lower resistance due to their increased inertia. This choice introduces agent-specific dynamics in addition to heterogeneous collision constraints.

For both the 4-drone and 8-drone cases, we use LQR-generated trajectories as the latent trajectories. These trajectories provide a structured initialization for the symplectic operator map, while the PI-SONet training enforces the optimality conditions and collision-avoidance constraints in the urban environment. For each case, we generate 50 training and 50 testing samples, where each sample corresponds to a different realization of the drone radii. The SONet architecture consists of 3 alternating up and down layers, with hidden dimension 32 for the networks corresponding to K and b. The rectangular building obstacles are treated as impenetrable regions, whereas the park regions remain available for navigation. This allows the learned trajectories to exploit the geometry of the environment while satisfying both drone-building and drone-drone collision constraints.

For the 4-drone case, we train the model using 200 Adam steps with learning rate 10^{-3}, followed by 200 L-BFGS steps. We use an annealing strategy for the penalty parameters, with both ϵ and l initialized at 0.01 and decayed by a factor of 0.7 every 50 steps. For the 8-drone case, we use the same annealing strategy but increase the number of Adam steps to 300, followed by 100 L-BFGS steps. This additional Adam optimization improves stability in the higher-dimensional setting with more inter-agent interactions and obstacle-avoidance constraints.

The urban city environment provides a challenging benchmark for evaluating PI-SONet under realistic geometric constraints and heterogeneous agent properties. In this setting, the model must simultaneously account for radius-dependent collision margins, radius-dependent resistance coefficients, and nonconvex obstacle-induced trajectory choices. The resulting trajectories demonstrate the ability of PI-SONet to generate feasible controls for multi-drone navigation in structured environments that resemble practical delivery scenarios.

2 Additional benchmarking details

2.1 State-of-the-art baseline methods

For benchmarking, we compare the performance of PI-SONet against the following representative state-of-the-art baseline methods:

1. *Multiple shooting method*, which discretizes the optimal control problem in time and solves it as a series of initial-value problems; multiple shooting generally has relatively fast computational times but is highly sensitive to initialization and may struggle with long time horizons and high dimensions.
2. *Pseudospectral optimal control*, which employes collocation/quadrature to provide high-accuracy solutions but suffers from the curse of dimensionality and generally cannot provide real-time computations.
3. *TSympOCNet*, which is a sympletic PINN shown to be able scale to very high dimensions/swarm sizes but only learns single-instance solutions and hence does not provide a solution that generalizes to ranges of problem parameters.

For multiple shooting, we use the Python implementation provided by CasADi [2, 3]. We initialize multiple shooting with a linear interpolation between the initial and terminal states, where we add Gaussian noise (mean 0 and standard deviation 0.01) to the velocity to encourage it not to remain at 0 for the whole trajectory. The time to compute the initial guess is not included in the timing results. We set the max number of iterations to be 200 but otherwise use the default CasADi parameters.

For pseudospectral optimal control, we use the MATLAB implementation provided by GPOPS-II [4, 5]. We initialize the method with just the initial and terminal states. We set the max number of IPOPT iterations to be 500, the mesh tolerance to be 1e-3, and an initial mesh consisting of 10 sub-meshes with 6 nodes each.

For TSympOCNet [6], we use the same JAX-enabled Python implementation that was used in the original paper. We initialize the method using the solution to an SDE system that represents the swarm as a particle system. The time to compute this initialization is not included in the timing results. Each run of TSympOCNet represents the results of using a different random seed for the optimizer in training.

A comparison of features between all methods considered is summarized in Table 9. We also list our key innovations over TSympOCNet in Table 10.

For both multiple shooting and pseudospectral method, we compute the safety constraint metric using a linear or polynomial interpolation of the resulting trajectory, respectively, to check for collisions on grid points that might otherwise be missed by their corresponding discretizations. For all methods and all test scenarios, we consider a trajectory as containing a collision if its corresponding maximum safety constraint violation metric is greater than 1e-6 to allow for a small amount of numerical roundoff error.

2.2 Some qualitative results for the *Maze* scenario

Discretization-based methods, such as multiple shooting and pseudospectral method, generally rely on having a good initial guess in order to converge to a feasible,

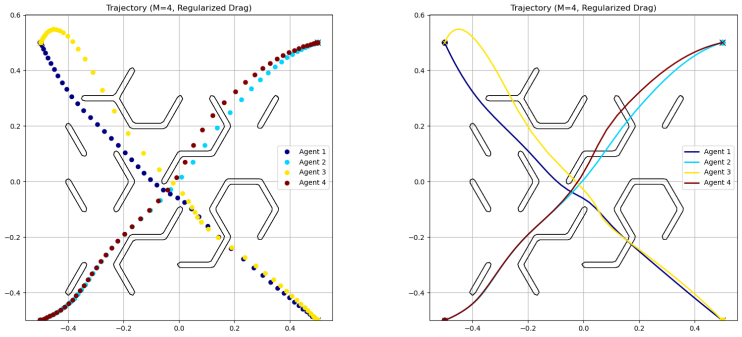
	Multiple Shooting	Pseudospectral Method	TSympOCNet	PI-SONet
Structure-preserving	✗	✗	✓	✓
Continuous constraint enforcement	✗	✗	✓	✓
Real-time performance	✗	✗	✗	✓
Generalizabilty across problem instances	✗	✗	✗	✓

Table 9: **Feature comparison between PI-SONet and representative baseline methods.** Like TSympOCNet, PI-SONet preserves symplecticity to maintain optimality and continuously enforces constraints to maintain feasibility. Unlike all baseline methods considered, PI-SONet's generalizability to new problem instances circumvents the need for complete rerunning/retraining, thereby enabling sub-second level inference times on new problem settings.

TSympOCNet	**PI-SONet (ours)**
✓ Parameterized symplectic map	✓ Parameterized *family* of symplectic maps
✓Families of TRS solvers: LQR, LQR + TL-SympNet	✓New families of TRS solvers: LQR, LQR with rotation, LQR + pretrained TSympOCNet, Eikonal
X Single-instance solver	✓ Generalizable to new family of problem scenarios
X Retrain for new problem scenarios	✓ Fast test-time finetuning without re-training
X Full K matrices	✓ Efficient block diagonal structure for K matrix

Table 10: **Comparison between TSympOCNet and PI-SONet.** While TSympOCNet learns a parameterized symplectic map for a fixed problem instance, PI-SONet extends this framework to a parameterized *family* of symplectic maps, enabling generalization across classes of problem scenarios. PI-SONet performs real-time inference without retraining and fast test-time fine tuning when needed, incorporates richer families of latent TRS solvers, and leverages an efficient block-diagonal structure for the learned operators for improved scalability.

(sub)optimal solution. However, problem constraints (e.g., collision or obstacle avoidance constraints) are only enforced at discrete grid points. Hence, when not initialized properly, these methods may report generated trajectories as successful when, in actuality, they contain visually obvious collisions. An example of this behavior is shown in Figure 7, in which multiple shooting generates trajectories with no collisions at the discretization points, but any reasonable interpolation of this discrete trajectory results in obvious and severe crashes with the maze.



(a) Discretized Trajectory (b) Interpolated Trajectory

Fig. 7: **Sample trajectories using discretization-based baseline methods for a 4-agent swarm navigating through a nonconvex maze.** Without a good initialization, discretization-based methods, such as multiple shooting and pseudospectral method, often lead to infeasible trajectories as problem constraints are only enforced at discrete points. This behavior exemplifies the issue of initialization sensitivity that many existing baseline methods experience and which becomes more pronounced in nonconvex cases, such as the one shown here. In such cases, using a finer grid may not be enough to recover convergence to a feasible solution (and in fact, it was not able to in the case shown here). In contrast, as an operator-based approach, PI-SONet continuously enforces constraints for all times and states and generally does not suffer from this behavior.

3 Ablation studies

3.1 Choice of latent solver

We investigate the effect of the latent solver used to generate the reference trajectories. In all cases, the downstream PI-SONet architecture and training procedure are kept fixed, and only the latent trajectories are varied. We consider three choices: (i) standard LQR, (ii) LQR with an additional rotation term in the latent dynamics, and (iii) LQR composed with a pretrained TSympOCNet. The pretrained TSympOCNet is trained only on the unperturbed initial condition and is then reused to generate latent trajectories for perturbed cases. All experiments in this section are conducted in the *Free* setting, i.e., without obstacles.

The quantitative results are summarized in Table 11 with corresponding trends shown in Figure 8. For small systems ($n = 4, 8$), all three approaches achieve comparable performance in terms of cost and feasibility. However, as the number of drones increases, clear differences emerge. The standard LQR solver fails to train stably beyond small system sizes, indicating that the latent trajectories do not provide a suitable initialization for the downstream optimization.

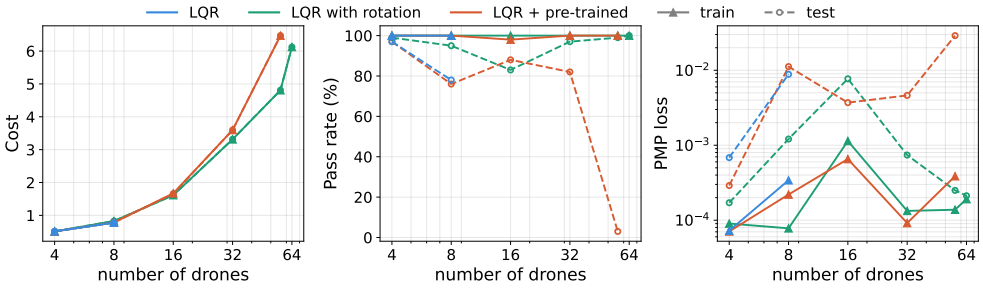


Fig. 8: **Effect of different latent solvers (in the 'free' case).** We compare LQR, LQR with rotation, and LQR composed with a pretrained TSympOCNet across increasing swarm sizes in the absence of obstacles. (*Left*) Cost increases with the number of drones for all methods with LQR with rotation consistently achieving lower costs than LQR + pretrained TSympOCNet at larger scales. (*Middle*) Pass rate (%) on train (solid) and test (dashed) datasets. While all methods perform similarly for small systems, LQR with rotation maintains high success rates as the number of drones increases, whereas LQR + pretrained TSympOCNet exhibits a sharp degradation at larger scales. (*Right*) PMP residuals (log scale). LQR with rotation generally achieves lower test residuals compared to the other methods, indicating better adherence to optimality conditions.

In contrast, incorporating rotation into the latent dynamics significantly improves scalability. LQR with rotation maintains near-perfect pass rates across all tested system sizes with consistently low PMP residuals and competitive objective costs. These results suggest that even a simple modification to the latent dynamics can better capture the structure of the solution manifold, leading to improved robustness as the system dimension grows.

The pretrained TSympOCNet composition performs competitively at small scales, but its performance deteriorates as the number of drones increases. In particular, we observe a drop in pass rate and a sharp increase in PMP residuals for larger systems (e.g., $n = 56$), indicating poor generalization. This behavior is expected, as the pretrained model is trained only on the unperturbed configuration and is not adapted to variations in initial conditions or system size.

Overall, these results highlight the importance of the choice of latent solver in enabling scalable learning of optimal control solutions. Even in this geometrically simple free-space setting, incorporating rotation in the latent dynamics significantly improves robustness and scalability, while naive reuse of a pretrained model fails to generalize to larger systems.

3.2 Effect of enforcing symplectic structure

To assess the importance of enforcing symplectic structure, we replace the SONet architecture with a standard multilayer perceptron (MLP) that directly maps latent trajectories to physical states. Specifically, the MLP takes as inputs the concatenated latent state $(\boldsymbol{y}(t), \boldsymbol{q}(t))$, obstacle radius r, and time t and outputs $(\boldsymbol{x}'(t), \boldsymbol{p}'(t))$. The

		Train			Test		
Metric	n	LQR	LQR w/ rotation	LQR + pretrained	LQR	LQR w/ rotation	LQR + pretrained
cost	4	0.512	0.511	0.511	0.513	0.512	0.511
	8	0.776	0.826	0.782	0.778	0.828	0.783
	16	—	1.614	1.670	—	1.612	1.660
	32	—	3.313	3.597	—	3.315	3.596
	56	—	4.807	6.470	—	4.809	6.470
	64	—	6.123	—	—	6.122	—
pass (%)	4	100	100	100	97	99	97
	8	100	100	100	78	95	76
	16	—	100	98	—	83	88
	32	—	100	100	—	97	82
	56	—	100	100	—	99	3
	64	—	100	—	—	100	—
PMP	4	7.24e−5	9.02e−5	7.08e−5	6.84e−4	1.71e−4	2.90e−4
	8	3.41e−4	7.79e−5	2.20e−4	8.84e−3	1.21e−3	1.12e−2
	16	—	1.14e−3	6.54e−4	—	7.70e−3	3.70e−3
	32	—	1.33e−4	9.16e−5	—	7.39e−4	4.62e−3
	56	—	1.38e−4	3.86e−4	—	2.50e−4	2.90e−2
	64	—	1.91e−4	—	—	2.12e−4	—

Table 11: **Quantitative comparison of latent solvers in free space under increasing system size.** We report cost, pass rate (%), and PMP residuals for LQR, LQR with rotation, and LQR composed with a pretrained TSympOCNet, evaluated on both training and test sets without obstacles. LQR fails to train stably beyond small system sizes. Incorporating rotation into the latent dynamics significantly improves both feasibility and optimality, maintaining high pass rates and low PMP residuals across all scales. In contrast, while the pretrained composition performs competitively at small scales, it exhibits degraded generalization and stability as the number of drones increases, reflected in reduced pass rates and higher PMP residuals.

network consists of fully-connected layers with SiLU activations and a hidden dimension of $d = 256$, chosen such that the total number of parameters is comparable to that of PI-SONet across all problem sizes.

To ensure a fair comparison, we enforce boundary conditions in the same manner as SONet by constructing the output as

$$\boldsymbol{x}_{\text{out}}(t) = \boldsymbol{y}(t) + t(1-t)\,\boldsymbol{x}'(t), \quad \boldsymbol{p}_{\text{out}}(t) = \boldsymbol{p}'(t),$$

so that the initial and terminal constraints are satisfied by construction. The training procedure, loss formulation, and optimization schedule (Adam followed by L-BFGS) are kept identical to the SONet setting. For the 16-drone case, a small ℓ_2 regularization ($\lambda = 10^{-6}$) was required to stabilize training.

Table 12 summarizes the quantitative comparison. While both models achieve similar success rates in terms of feasibility, the MLP consistently exhibits higher PMP residuals and suboptimal costs with the gap becoming more pronounced as the number of drones increases. Notably, these differences arise despite comparable parameter counts, indicating that the performance gain of PI-SONet is not simply due to model capacity.

# Drones	Model	# Params	Train Time	PMP (Train)	Cost (Train)	Test Time	PMP (Test)	Cost (Test)	Pass
4	MLP	82,976	25.45	2.31×10^{-4}	0.601	0.0034	1.12×10^{-3}	0.601	49/50
	SONet	76,608	106.42	1.24×10^{-4}	0.556	0.0189	7.24×10^{-5}	0.559	50/50
8	MLP	99,392	26.28	6.19×10^{-4}	0.967	0.0053	2.22×10^{-3}	0.968	49/50
	SONet	90,624	85.35	4.53×10^{-4}	0.838	0.0192	3.41×10^{-4}	0.846	50/50
16	MLP	132,224	34.26	2.10×10^{-4}	1.753	0.0046	7.28×10^{-4}	1.754	49/50
	SONet	134,208	107.74	4.65×10^{-4}	1.549	0.0191	2.30×10^{-4}	1.565	48/50

Table 12: **Ablation study on enforcing symplectic structure**. We compare a standard MLP against SONet (symplectic map). SONet achieves consistently lower cost and PMP residuals with comparable parameter counts, highlighting the importance of preserving symplecticity.

The qualitative behavior is illustrated in Figure 9. Unlike PI-SONet, which adapts its trajectories to the obstacle geometry, the MLP produces trajectories that are largely insensitive to changes in the obstacle radius. As the obstacle size increases, the predicted paths remain qualitatively similar, resulting in degraded optimality and increased PMP residuals. Although the trajectories are mostly collision-free, they fail to adjust in a geometry-aware manner, leading to suboptimal paths.

These results highlight that enforcing symplectic structure plays an important role in capturing the underlying Hamiltonian dynamics and geometry-dependence. In its absence, even an analogously expressive MLP struggles to learn the correct dependence on problem parameters, resulting in solutions that satisfy constraints but deviate from optimality.

References

[1] Mavrogiannis, C.I., Knepper, R.A.: Decentralized multi-agent navigation planning with braids. In: Algorithmic Foundations of Robotics XII: Proceedings of the Twelfth Workshop on the Algorithmic Foundations of Robotics, pp. 880–895 (2020). Springer

[2] Andersson, J., Gillis, J.: CasADi. Version 3.7.2. https://web.casadi.org/. https://web.casadi.org/

[3] Andersson, J.A., Gillis, J., Horn, G., Rawlings, J.B., Diehl, M.: CasADi: a software framework for nonlinear optimization and optimal control. Mathematical Programming Computation **11**(1), 1–36 (2019)

[4] Patterson, M.A., Rao, A.V.: GPOPS-II: Next-Generation Optimal Control Software. Version 2.3. https://www.gpops2.com/. https://www.gpops2.com/

[5] Patterson, M.A., Rao, A.V.: GPOPS-II: A MATLAB software for solving multiple-phase optimal control problems using hp-adaptive Gaussian quadrature collocation methods and sparse nonlinear programming. ACM Transactions on Mathematical Software (TOMS) **41**(1), 1–37 (2014)

[6] Zhang, Z., Wang, C., Liu, S., Darbon, J., Karniadakis, G.E.: A time-dependent

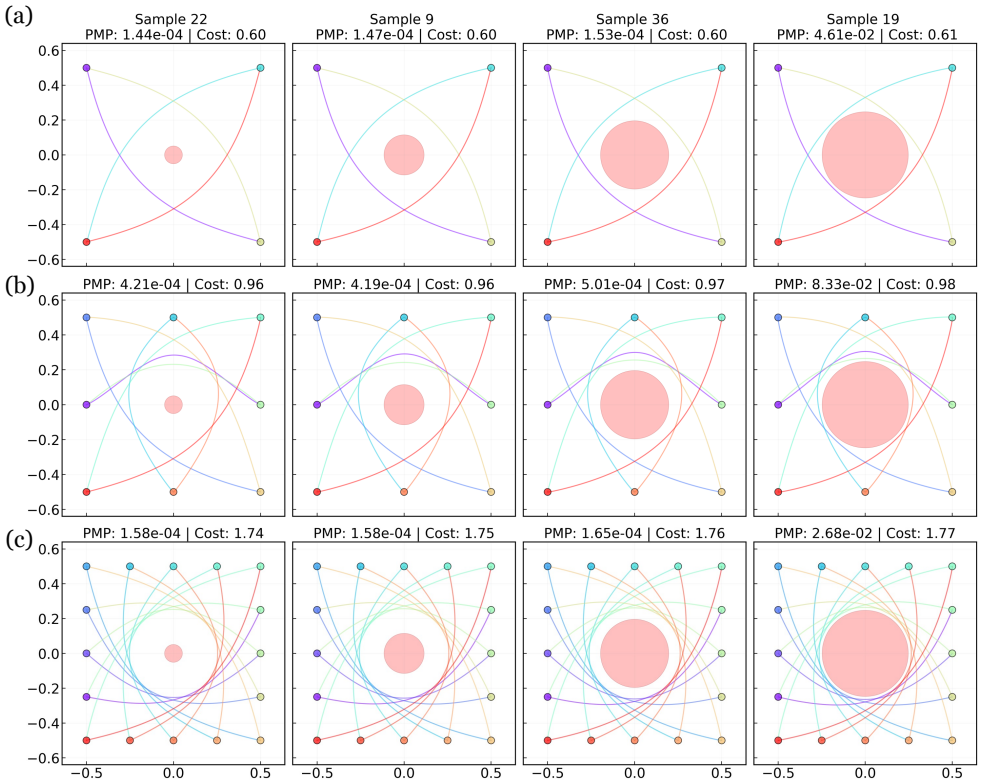


Fig. 9: **Ablation study: MLP trajectories without symplectic structure under varying obstacle geometries.** We consider drones navigating around a central obstacle with radius $r \in [0.05, 0.25]$. Each column corresponds to a different obstacle radius (increasing from left to right), while rows (a)–(c) denote increasing numbers of drones (4, 8 and 16). The initial positions are fixed across all cases to isolate the effect of the obstacle geometry. Unlike PI-SONet, the MLP fails to adapt its trajectories to the changing obstacle size; the predicted paths remain qualitatively similar across columns, leading to increased PMP residuals and costs. These results highlight the importance of enforcing symplectic structure for capturing geometry-dependent optimal solutions.

symplectic network for nonconvex path planning problems with linear and nonlinear dynamics. SIAM Journal on Scientific Computing **47**(4), 769–794 (2025)