Flow-Opt: Scalable Centralized Multi-Robot Trajectory Optimization with Flow Matching and Differentiable Optimization

Simon Idoko, Arun Kumar Singh

***Abstract*—Centralized trajectory optimization in the joint space of multiple robots provides access to a larger feasible space, yielding smoother trajectories, especially in tight environments. Unfortunately, it is computationally intractable beyond small swarm sizes. We propose Flow-Opt, a learning-based framework for fast, high-quality approximation of centralized multi-robot trajectory optimization. We reduce the problem to first learning a generative model to sample candidate trajectories and then using a learned Safety Filter (SF) to ensure fast inference-time constraint satisfaction. We propose a flow-matching model based on a Diffusion Transformer (DiT) augmented with permutation-invariant encoders as the generative model, and develop a custom differentiable SF solver equipped with a neural initialization network trained in a self-supervised manner. Experiments show that Flow-Opt generates trajectories for tens of robots in cluttered environments in tens of milliseconds—up to 30× faster than existing centralized optimizers and 160× faster than diffusion-based baselines—while maintaining comparable trajectory quality and enabling parallel solving of multiple problem instances.**

I. Introduction

Generating coordinated, collision-free trajectories is a fundamental requirement for deploying multi-robot systems in applications such as warehouse automation, drone cinematography [1], and large-scale environmental mapping. Furthermore, diverse feasible multi-robot motions can be used to construct data-driven simulations to train navigation policies [2], [3].

Current approaches are primarily divided into two paradigms. Distributed methods [4], [5] allow each robot to plan independently using communication or predictions, achieving computational speed but only considering inter-robot interactions implicitly, which limits the feasible solution space. Centralized methods [6], [7] plan in the joint space of all robots, dramatically improving trajectory quality in terms of smoothness and arc-length. However, the number of inter-robot collision constraints grows quadratically with swarm size, leading to poor scalability.

Recent works have applied diffusion models to multi-robot planning [8], [9], training generative priors on expert trajectories and refining samples at inference time. However, the denoising process is slow, and embedding constraint-satisfaction within each denoising step further compounds the cost.

Our Approach. Flow-Opt follows the paradigm of combining generative modeling with inference-time refinement but differs in two key ways. First, we build our pipeline around flow matching rather than diffusion. Flow policies are based on ordinary differential equations (ODEs) rather than stochastic differential equations, resulting in simpler training and faster inference. We leverage a DiT backbone [10] augmented with permutation-invariant start-goal and obstacle encoders. To the best of our knowledge, this is the first application of flow matching to multi-robot trajectory planning.

Second, rather than embedding refinement within the denoising process, we apply it only to the final flow output through a Safety Filter (SF). We develop a custom differentiable SF solver and equip it with a neural initialization network that provides context-specific warm-starts. The initialization network is trained in a self-supervised manner by leveraging the end-to-end differentiability of the SF solver.

Benefits. Our approach achieves up to an order of magnitude speedup over [7] while generating trajectories of comparable quality. Compared to [8], our method produces smoother trajectories while reducing computation time by a factor of 160. Moreover, each component can be batched, enabling tens of independent problem instances to be solved in parallel within a fraction of a second.

All authors are with the University of Tartu. Emails: simon.idoko@ut.ee, aks1812@gmail.com.

II. Problem Formulation

Consider n robots modeled as double integrators in an n_d-dimensional workspace ($n_d \in \{2, 3\}$), which is expressive enough for quadrotors [11] and holonomic mobile robots. Let $\mathbf{p}_{i|k}$ denote the position of robot i at time step k over a planning horizon of $K+1$ steps. The centralized trajectory optimization is:

$$\min_{\mathbf{p}_{i|0:K}} \quad \frac{1}{2}\sum_{i=1}^{n}\sum_{k=0}^{K}\|\ddot{\mathbf{p}}_{i|k}\|_2^2 \tag{1a}$$

$$\text{s.t.} \quad \|\mathbf{M}_r^{-1}(\mathbf{p}_{i|k}-\mathbf{p}_{j|k})\|_2^2 \geq 1,\ \forall k,\ i \neq j \tag{1b}$$

$$\|\mathbf{M}_o^{-1}(\mathbf{p}_{i|k}-\mathbf{p}_{o,m|k})\|_2^2 \geq 1,\ \forall k,\ i,\ m \tag{1c}$$

$$\mathbf{p}_{\min} \leq \mathbf{p}_{i|k} \leq \mathbf{p}_{\max},\ \forall i, k \tag{1d}$$

$$\mathbf{p}_{i|0} = \mathbf{b}_{i,0},\ \mathbf{p}_{i|K} = \mathbf{b}_{i,K},\ \forall i \tag{1e}$$

The cost (1a) minimizes total squared acceleration. Constraints (1b)–(1c) enforce inter-robot and obstacle collision avoidance, where each robot is modeled as a spheroid with dimensions $(\frac{a}{2}, \frac{a}{2}, \frac{b}{2})$ encoded in diagonal matrix $\mathbf{M}_r$, and $\mathbf{M}_o$ captures obstacle dimensions inflated by robot size. Constraint (1d) enforces workspace bounds and (1e) fixes boundary conditions.

Polynomial Parametrization. We parametrize the trajectory of robot i as $\mathbf{p}_{i|0:K} = \overline{\mathbf{W}}\boldsymbol{\xi}_i$, where $\overline{\mathbf{W}}$ is a block-diagonal matrix of polynomial basis functions and $\boldsymbol{\xi}_i = [\boldsymbol{\xi}_{i,x}^\top, \boldsymbol{\xi}_{i,y}^\top, \boldsymbol{\xi}_{i,z}^\top]^\top$ are the coefficients. Velocities and accelerations follow analogously through derivative matrices $\dot{\overline{\mathbf{W}}}$,

$\ddot{\overline{\mathbf{W}}}$. Rolling all robot coefficients into a single vector $\boldsymbol{\xi}$, the optimization becomes:

$$\begin{aligned} \min_{\boldsymbol{\xi}} \ & \frac{1}{2}\boldsymbol{\xi}^\top \mathbf{Q}\boldsymbol{\xi} + \mathbf{q}^\top \boldsymbol{\xi} \\ \text{s.t.} \ & \mathbf{A}\boldsymbol{\xi} = \mathbf{b}, \ \ \mathbf{G}\boldsymbol{\xi} \leq \mathbf{h}, \ \ \mathbf{g}(\boldsymbol{\xi}) \leq \mathbf{0} \end{aligned} \tag{2}$$

where $\mathbf{Q}, \mathbf{A}, \mathbf{G}, \mathbf{q}, \mathbf{b}, \mathbf{h}$ are constants derived from the polynomial basis. The affine equality and inequality constraints stem from boundary conditions (1e) and workspace bounds (1d), respectively. The function $\mathbf{g}$ contains the quadratic collision constraints (1b)–(1c) expressed in terms of polynomial coefficients.

III. Method

Flow-Opt has two core components: a conditional flow-matching policy for trajectory sampling, and a Safety Filter (SF) for constraint projection with learned warm-start. The pipeline first draws diverse trajectory candidates from the flow policy, selects the most promising based on constraint residual, and refines them through the SF.

A. Conditional Flow Matching for Trajectories

We learn a generative model over trajectory coefficients $\boldsymbol{\xi}$ conditioned on context $\mathbf{c} = [\mathbf{c}_{sg}, \mathbf{c}_{ob}]$ (start-goal pairs and obstacle information). A neural network $v_\theta(\boldsymbol{\xi}_t, t, \mathbf{c})$ is trained to approximate a vector field transporting samples from a Gaussian prior q_0 to the data distribution $q(\boldsymbol{\xi}|\mathbf{c})$. Using a linear interpolation path $\boldsymbol{\xi}_t = (1-t)\boldsymbol{\xi}_0 + t\boldsymbol{\xi}_1$ between a prior sample $\boldsymbol{\xi}_0 \sim q_0$ and a data sample $\boldsymbol{\xi}_1 \sim q(\boldsymbol{\xi}|\mathbf{c})$, the Conditional Flow Matching objective is:

$$\mathcal{L}_{\text{CFM}}(\boldsymbol{\theta}) = \mathbb{E}_{t,\boldsymbol{\xi}_0,(\boldsymbol{\xi}_1,\mathbf{c})}\left[\left\|v_\theta(\boldsymbol{\xi}_t, t, \mathbf{c}) - (\boldsymbol{\xi}_1 - \boldsymbol{\xi}_0)\right\|^2\right]. \tag{3}$$

At inference, trajectories are generated by integrating the learned ODE from t=0 to t=1:

$$\boldsymbol{\xi} = \boldsymbol{\xi}_0 + \int_0^1 v_\theta(\boldsymbol{\xi}_t, t, \mathbf{c}_{\text{new}})\, dt. \tag{4}$$

Different initial samples $\boldsymbol{\xi}_0$ yield diverse trajectories, computed in a batched fashion.

Architecture. The backbone is a DiT [10]. Noisy trajectory coefficients $\boldsymbol{\xi}_t \in \mathbb{R}^{n_\xi \times n \times n_d}$ are patchified via a CNN into tokens $\boldsymbol{\xi}'_t \in \mathbb{R}^{S \times D}$, where the CNN kernel $(1, n_d)$ produces one token per robot, aggregating state dimensions without mixing across robots. Context is incorporated through a CNN-based start-goal encoder ($\mathbf{c}'_{sg} \in \mathbb{R}^{S\times D}$) and a permutation-invariant PointNet-based [12] obstacle encoder ($\mathbf{c}'_{ob} \in \mathbb{R}^{\omega \times D}$). The DiT processes trajectory tokens conditioned on start-goal features via self-attention, and on obstacle features via cross-attention.

A key strength of flow matching is its ability to learn multi-modal distributions. In our context, this manifests as diverse collision avoidance behaviors—variation in both speed profiles and path choices among robots—which is particularly valuable for generating rich simulation data for training navigation policies [3].

B. Safety Filter with Learned Initialization

The flow policy is trained on demonstrations and is unaware of trajectory-level constraints, so predicted trajectories may not be feasible. We project them onto the feasible set via:

$$\begin{aligned} \min_{\overline{\boldsymbol{\xi}}} \ & \frac{1}{2}\|\overline{\boldsymbol{\xi}} - \boldsymbol{\xi}\|_2^2 \\ \text{s.t.} \ & \mathbf{A}\overline{\boldsymbol{\xi}} = \mathbf{b}, \ \ \mathbf{G}\overline{\boldsymbol{\xi}} \leq \mathbf{h}, \ \ \mathbf{g}(\overline{\boldsymbol{\xi}}) \leq \mathbf{0} \end{aligned} \tag{5}$$

Fixed-Point Formulation. Following [7], we reformulate the quadratic constraints (1b)–(1c) in spherical form and apply Augmented Lagrangian relaxation with Alternating Minimization (AM). Each AM sub-step admits a closed-form solution, yielding a differentiable fixed-point iteration:

$${}^{l+1}\overline{\boldsymbol{\xi}},\ {}^{l+1}\boldsymbol{\lambda} = \mathcal{T}({}^{l}\overline{\boldsymbol{\xi}},\ {}^{l}\boldsymbol{\lambda}), \tag{6}$$

where $\boldsymbol{\lambda}$ denotes the Lagrange multipliers and l is the iteration index. Two properties are worth highlighting. First, all computations within $\mathcal{T}$ involve only element-wise operations and matrix-matrix products, enabling efficient GPU batching. The key matrix factorization in the trajectory update is independent of the flow input $\boldsymbol{\xi}$ and can be pre-stored, reducing the batched solve across B instances to a single matrix-matrix product. Second, since every step has a closed-form, $\mathcal{T}$ can be unrolled into a fully differentiable computational graph.

Learned Initialization. A transformer-based initialization network predicts context-specific warm-start values $({}^{0}\overline{\boldsymbol{\xi}}, {}^{0}\boldsymbol{\lambda})$ conditioned on the flow output $\boldsymbol{\xi}$ and context $\mathbf{c}$. The network uses CNN-based encoders architecturally identical to those in the flow model, and produces both initial trajectory coefficients and Lagrange multipliers. It is trained by minimizing the fixed-point residual over L unrolled iterations:

$$\min_{\phi} \underbrace{\sum_{l=0}^{L-1} \left\| \begin{bmatrix} {}^{l+1}\overline{\boldsymbol{\xi}} \\ {}^{l+1}\boldsymbol{\lambda} \end{bmatrix} - \mathcal{T}({}^{l}\overline{\boldsymbol{\xi}}, {}^{l}\boldsymbol{\lambda}) \right\|_2^2}_{\mathcal{L}_{FP}} + \|{}^{L}\overline{\boldsymbol{\xi}} - \boldsymbol{\xi}\|_2^2. \tag{7}$$

The first term accelerates convergence by reducing the residual at each iteration [13]. The second ensures minimal displacement from the flow prediction. Gradients flow through the unrolled $\mathcal{T}$ chain, making the initialization solver-aware—the network learns not just near-optimal trajectories, but initializations specifically suited to how the downstream SF solver operates. This training is fully self-supervised: no ground-truth fixed-point solutions are required.

This design insight is crucial: directly initializing the SF with the flow policy output (bypassing the initialization network) produces significantly slower residual decay. The flow policy is trained to imitate expert trajectories and is agnostic to the optimization landscape of the solver. The initialization network bridges this gap by learning to transform the flow output into a starting point that is effective for the SF's specific optimization process.

IV. Experiments

A. Setup

The pipeline is implemented in JAX [14] with Equinox [15]. All benchmarks run on an RTX 5090 with an Intel i9 and

32 GB RAM. The flow policy is trained on over 20,000 multi-robot trajectories generated by a batched extension of [7]. At test time, new start and goal configurations are sampled from the same distribution. We evaluate using three metrics: success rate (SR), computation time, and trajectory quality (smoothness cost $\sum_k \|\ddot{\mathbf{p}}_{i,k}\|^2$ and arc-length).

B. Comparison with Optimization Baseline [7]

Table I compares Flow-Opt against the centralized optimizer [7] (given an upper bound of 10K iterations). Our approach achieves consistent speedups across all scenarios: $4\times$ for 8 robots, up to $11\times$ for 32 robots in 2D, and $33\times$ for 64 robots in 3D. The speedup is most pronounced in 2D, where reduced maneuvering space makes the problem harder for the baseline. Both methods achieve near-100% success rates, but our approach consistently produces feasible solutions faster. The slight increase in smoothness cost for larger scenarios reflects the trade-off for massive computation savings.

The convergence advantage is quantified by the number of iterations to reach low primal residuals. For the 2D/16-robot case, our approach requires a mean of 82 iterations to reach a primal residual of 0.01, compared to 2606 for [7]—a $30\times$ reduction.

TABLE I
PERFORMANCE COMPARISON WITH BASELINE [7].

Dim	Robots	Method	Time (s)	SR (%)	Smooth.	Arc L.
2D	8	Baseline	0.218	100	0.265	1.297
		Ours	**0.052**	100	0.214	1.049
	16	Baseline	0.593	99.9	0.278	1.362
		Ours	**0.065**	**100**	0.240	1.175
	32	Baseline	3.407	99.6	0.203	0.993
		Ours	**0.297**	**100**	0.241	1.181
3D	16	Baseline	0.148	100	0.631	3.093
		Ours	**0.069**	100	0.627	3.073
	32	Baseline	0.999	99.8	0.638	3.124
		Ours	**0.151**	**100**	0.700	3.431
	64	Baseline	33.70	99.7	0.714	3.500
		Ours	**1.02**	**99.9**	0.807	3.952

C. Comparison with Diffusion-Based Planner [8]

Table II compares against the diffusion-based multi-robot planner MMD [8]. Both methods achieve comparable trajectory quality and 100% success rate across 1000 random problem instances. However, computation times differ strikingly: our approach is $104\times$ faster for 8 robots and $163\times$ faster for 32 robots. The gap widens because MMD's conflict-based search and slow diffusion inference scale poorly. Additionally, since our flow policy is trained on joint multi-robot trajectories (unlike MMD's single-robot prior), it captures global interactions and produces trajectories with better inter-robot clearance when maneuvering space is available—approximately 16% improvement in average pairwise distances for 8-robot scenarios.

D. Comparison with Batch-Sequential Planner [16]

We also benchmark against [16], which partitions robots into groups, performs joint optimization within each group, and then plans sequentially across groups. Table III shows results in 3D. Our approach produces shorter trajectories (33% and 29% reduction for 16 and 32 robots), achieves 100% success rate versus 90% and 63% for [16], and is nearly an order of magnitude faster. The sequential approach restricts cooperative inter-group interactions—a limitation our centralized method avoids.

TABLE II
COMPARISON WITH MMD [8] IN 2D SCENARIOS.

Method	Robots	Time (s)	SR (%)	Smooth.	Arc Len.
MMD	8	5.37	100	0.206	1.016
	16	18.0	100	0.239	1.178
	32	48.18	100	0.249	1.224
Ours	8	**0.052**	100	0.214	1.049
	16	**0.065**	100	0.240	1.175
	32	**0.297**	100	0.241	1.181

TABLE III
COMPARISON WITH [16] IN 3D.

Robots	[16]			Ours		
	Time	SR (%)	Arc L.	Time	SR (%)	Arc L.
16	0.54	89.9	4.067	**0.069**	**100**	**3.073**
32	1.72	63.1	4.441	**0.151**	**100**	**3.431**

E. Scalability and Parallel Problem Solving

A distinguishing feature of our framework is its ability to solve multiple independent problem instances simultaneously via GPU parallelism. We validate this in Fig. 1. Fig. 1(a) shows that the flow policy can generate hundreds of candidate trajectories in tens of milliseconds, even for large systems. Fig. 1(b) demonstrates that SF computation time scales nearly linearly with batch size, a direct benefit of our GPU-vectorized solver. For example, generating 10 candidates for 50 problems (500 total) takes under 60 ms; refining the top 5 per problem (250 total) takes under 300 ms. Thus, 50 distinct planning problems are solved in well under a second.

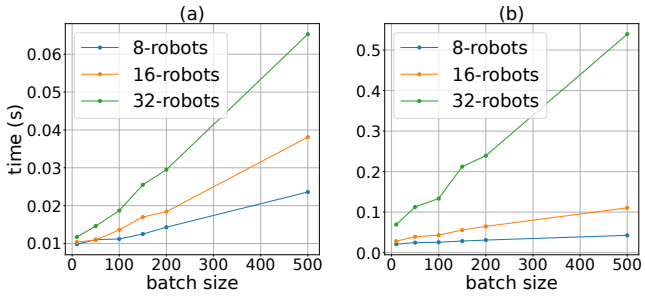

Fig. 1. (a) Computation time to sample a batch of trajectories from the flow model. (b) Computation time for 500 SF iterations to refine a batch of trajectories. Both scale nearly linearly with batch size due to GPU parallelization.

F. Robustness to Out-of-Distribution Perturbations

The initialization network is trained with a fixed number of obstacles (eight in our experiments). To test generaliza-

tion, we introduced an additional obstacle not present during training. As shown in Fig. 2(a)–(b), the SF solver remains robust, successfully computing smooth and feasible trajectories. Fig. 2(c)–(d) further confirms this: the primal and fixed-point residual decay remains close to the unperturbed setting, demonstrating resilience to out-of-distribution environmental changes.

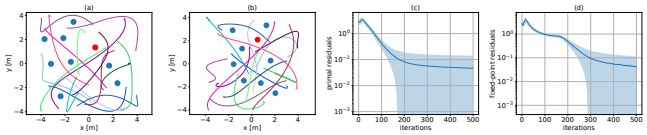

Fig. 2. Robustness to OOD perturbations. The SF initialization network is trained with 8 obstacles (cyan), but at test time an additional obstacle is introduced. (a)–(b): The SF produces smooth, feasible trajectories despite the perturbation. (c)–(d): Primal and fixed-point residual decay remains close to the unperturbed setting.

V. Conclusion

We presented Flow-Opt, a framework combining flow matching and differentiable optimization for scalable centralized multi-robot trajectory planning. Our approach generates coordinated trajectories for tens of robots in milliseconds, achieving order-of-magnitude speedups over both optimization and diffusion baselines while maintaining comparable trajectory quality and high success rates. The self-supervised learned initialization for the SF solver is key to bridging the gap between generative modeling and constraint satisfaction. Future work includes end-to-end joint training of the flow policy and SF, and extension to non-holonomic robots for applications in autonomous driving.

References

[1] T. Nägeli, L. Meier, A. Domahidi, J. Alonso-Mora, and O. Hilliges, "Real-time planning for automated multi-view drone cinematography," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, pp. 1–10, 2017.

[2] J. Ren, W. Xiang, Y. Xiao, R. Yang, D. Manocha, and X. Jin, "Heter-sim: Heterogeneous multi-agent systems simulation by interactive data-driven optimization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 3, pp. 1953–1966, 2021.

[3] A. Prorok, J. Blumenkamp, Q. Li, R. Kortvelesy, Z. Liu, and E. Stump, "The holy grail of multi-robot planning: Learning to generate online-scalable solutions from offline-optimal experts," in *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, 2022, pp. 1804–1808.

[4] V. K. Adajania, S. Zhou, A. K. Singh, and A. P. Schoellig, "Amswarm: An alternating minimization approach for safe motion planning of quadrotor swarms in cluttered environments," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 1421–1427.

[5] C. E. Luis, M. Vukosavljev, and A. P. Schoellig, "Online trajectory generation with distributed model predictive control for multi-robot motion planning," *IEEE Robotics and Automation Letters*, vol. 5, pp. 604–611, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:202558898

[6] F. Augugliaro, A. P. Schoellig, and R. D'Andrea, "Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 1917–1922.

[7] F. Rastgar, H. Masnavi, J. Shrestha, K. Kruusamäe, A. Aabloo, and A. K. Singh, "Gpu accelerated convex approximations for fast multi-agent trajectory optimization," *IEEE Robotics and Automation Letters*, vol. 6, pp. 3303–3310, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:226281419

[8] Y. Shaoul, I. Mishani, S. Vats, J. Li, and M. Likhachev, "Multi-robot motion planning with diffusion models," *arXiv preprint arXiv:2410.03072*, 2024.

[9] J. Liang, J. K. Christopher, S. Koenig, and F. Fioretto, "Simultaneous multi-robot motion planning with projected diffusion models," *arXiv preprint arXiv:2502.03607*, 2025.

[10] W. Peebles and S. Xie, "Scalable diffusion models with transformers," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2023, pp. 4195–4205.

[11] A. Singletary, K. Klingebiel, J. Bourne, A. Browning, P. Tokumaru, and A. Ames, "Comparative analysis of control barrier functions and artificial potential fields for obstacle avoidance," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 8129–8136.

[12] C. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 77–85, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:5115938

[13] R. Sambharya, G. Hall, B. Amos, and B. Stellato, "Learning to warm-start fixed-point optimization algorithms," *Journal of Machine Learning Research*, vol. 25, no. 166, pp. 1–46, 2024.

[14] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, "JAX: composable transformations of Python+NumPy programs," 2018. [Online]. Available: http://github.com/google/jax

[15] P. Kidger and C. Garcia, "Equinox: neural networks in JAX via callable PyTrees and filtered transformations," *Differentiable Programming workshop at Neural Information Processing Systems 2021*, 2021.

[16] J. Park, J. Kim, I. Jang, and H. J. Kim, "Efficient multi-agent trajectory planning with feasibility guarantee using relative bernstein polynomial," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 434–440.