GLAS: Global-to-Local Safe Autonomy Synthesis for Multi-Robot Motion Planning with End-to-End Learning

Benjamin Rivière, Wolfgang Hönig, Yisong Yue, and Soon-Jo Chung

***Abstract*— We present GLAS: Global-to-Local Autonomy Synthesis, a provably-safe, automated distributed policy generation for multi-robot motion planning. Our approach combines the advantage of centralized planning of avoiding local minima with the advantage of decentralized controllers of scalability and distributed computation. In particular, our synthesized policies only require relative state information of nearby neighbors and obstacles, and compute a provably-safe action. Our approach has three major components: i) we generate demonstration trajectories using a global planner and extract local observations from them, ii) we use deep imitation learning to learn a decentralized policy that can run efficiently online, and iii) we introduce a novel differentiable safety module to ensure collision-free operation, thereby allowing for end-to-end policy training. Our numerical experiments demonstrate that our policies have a 20 % higher success rate than optimal reciprocal collision avoidance, ORCA, across a wide range of robot and obstacle densities. We demonstrate our method on an aerial swarm, executing the policy on low-end microcontrollers in real-time.**

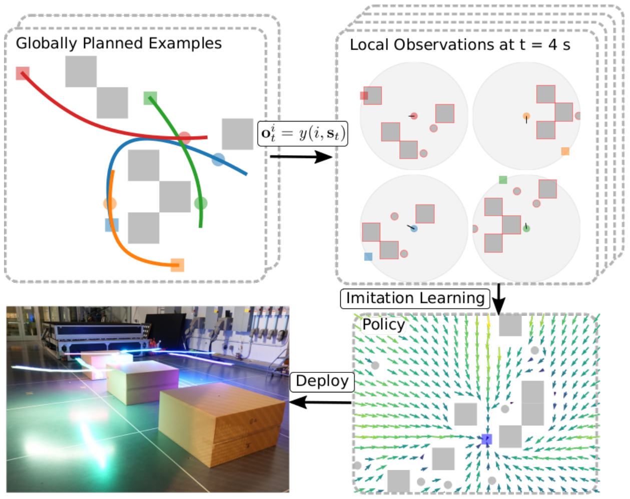
Globally Planned Examples
$\mathbf{o}_t^i = y(i, \mathbf{s}_t)$
Local Observations at t = 4 s
Imitation Learning
Policy
Deploy

Fig. 1. We learn distributed policies using trajectories from a global planner. We mask non-local information with an observation model, and perform imitation learning on an observation-action dataset. In the policy, the grey boxes are static obstacles, the grey circles are other robots, and the blue square is the robot's goal position.

I. Introduction

Teams of robots that are capable of navigating in dynamic and occluded environments are important for applications in next generation factories, urban search and rescue, and formation flying in cluttered environments or in space. Current centralized approaches can plan such motions with completeness guarantees, but require full state information not available to robots on-board, and are too computationally expensive to run in real-time. Distributed approaches instead use local decoupled optimization, but often cause robots to get trapped in local minima in cluttered environments. Our approach, GLAS, bridges this gap by using a global planner offline to learn a decentralized policy that can run efficiently online. We can thus automatically synthesize an efficient policy that avoids getting trapped in many cases. Unlike other learning-based methods for motion planning, GLAS operates in continuous state space with a time-varying number of neighbors and generates provably safe, dynamically-coupled policies. We demonstrate in simulation that our policy achieves significantly higher success rates compared to ORCA, a state-of-the-art decentralized approach for single integrator dynamics. We also extend our approach to double integrator dynamics, and demonstrate that our synthesized policies work well on a team of quadrotors with low-end microcontrollers.

The overview of GLAS is shown in Fig. 1. First, we generate trajectories for random multi-robot motion planning instances using a global planner. Second, we apply a local observation model to generate a dataset of observation-action pairs. Third, we perform deep imitation learning to generate a local policy that imitates the expert (the global policy) to avoid local minima. The vector field of Fig. 1 shows that the policy successfully avoids local minima traps between obstacles and gridlock between robots.

We guarantee the safety of our policy through a convex combination of the learned desired action and our safety module. Instead of using existing optimization-based methods, we derive an analytic form for our safety module. The advantage of this safety module is that it is fully differentiable, which enables us to train our policy in an end-to-end fashion, resulting in policies that use less control effort.

Our main contributions are: i) to our knowledge, this is the first approach that automatically synthesizes a local policy from global demonstrations in a continuous state/action domain while guaranteeing safety and ii) derivation of a novel differentiable safety module compatible with end-to-end learning for dynamically-coupled motion planning.

The authors are with California Institute of Technology, USA. {briviere, whoenig, yyue, sjchung}@caltech.edu.

This paper is a version of the work accepted to IEEE RA-L 2020, DOI: 10.1109/LRA.2020.2994035, with reduced Related Work, Theoretical, and Simulation sections. This work was supported by the Raytheon Company and Caltech/NASA Jet Propulsion Laboratory. Video: https://youtu.be/z9LjSfLfG6c. Code: https://github.com/bpriviere/glas.

We show that our policies outperform the state-of-the-art distributed multi-robot motion planners for single integrator dynamics in terms of success rate in a wide range of robot/obstacle density cases. We also implement GLAS for more physically-realistic double integrator dynamics in experimentation with an aerial swarm, demonstrating real-time computation on low-end microcontrollers.

A. Related Work

Multi-robot motion planning is an active area of research because it is a non-convex optimization problem with high state and action dimensionality. We compare the present work with state-of-the-art methods: (a) collision avoidance controllers, (b) optimal motion-planners, and (c) deep-learning methods.

Collision Avoidance: Traditional controller-level approaches include Optimal Reciprocal Collision Avoidance (ORCA) [1], Buffered Voronoi Cells [2], Artificial Potential Functions [3–5], and Control Barrier Functions [6]. These methods are susceptible to trapping robots in local minima. We address this problem explicitly by imitating a complete global planner with local information.

Motion Planners: Motion planners are a higher-level approach that explicitly solves the optimal control problem over a time horizon. Solving the optimal control problem is non-convex, so most recent works with local guarantees use approximate methods like Sequential Convex Programming to quickly reach a solution [7, 8]. Motion planners are distinguished as either global and centralized [8] or local and decentralized [7, 9], depending on whether they find solutions in joint space or computed by each robot.

Deep Learning Methods: Recently, there have been new learning-based approaches for multi-robot path planning [10–14]. These works use deep Convolutional Neural Networks (CNN) in a discrete state/action domain. Such discretization prevents coupling to higher-order robot dynamics whereas our solution permits tight coupling to the system dynamics by operating in a continuous state/action domain using a novel network architecture based on Deep Sets [15].

II. Problem Formulation

Notation: We denote vectors with a boldface lowercase, functions with an italics lowercase letter, scalar parameters with plain lowercase, subspaces/sets with calligraphic uppercase, and matrices with plain uppercase. We denote a robot index with a superscript i or j, and a state or action without the robot index denotes a joint space variable of stacked robot states. A double superscript index denotes a relative vector, for example $\mathbf{s}^{ij} = \mathbf{s}^j - \mathbf{s}^i$. We use a continuous time domain, and we suppress the explicit time dependency for states, actions, and dependent functions for notation simplicity.

Problem Statement: Let $\mathcal{V}$ denote the set of n_i robots, $\mathcal{G} = \{\mathbf{g}^1, \dots, \mathbf{g}^{n_i}\}$ denote their respective goal states, $\mathcal{S}_0 = \{\mathbf{s}_0^1, \dots, \mathbf{s}_0^{n_i}\}$ denote their respective start states, and Ω denote the set of m static obstacles. At time t, each robot i makes a local observation, $\mathbf{o}^i$, uses it to formulate an action, $\mathbf{u}^i$, and updates its state, $\mathbf{s}^i$, according to the dynamical model. Our goal is to find a controller, $u : \mathcal{O} \to \mathcal{U}$ that synthesizes actions from local observations through:

$$\mathbf{o}^i = y(i, \mathbf{s}), \quad \mathbf{u}^i = u(\mathbf{o}^i), \quad \forall i, t \tag{1}$$

to approximate the solution to the optimal control problem:

$$\begin{aligned} \mathbf{u}^* &= \text{argmin}_{\{\mathbf{u}^i \mid \forall i, t\}} c(\mathbf{s}, \mathbf{u}) && \text{s.t.} \\ \dot{\mathbf{s}}^i &= f(\mathbf{s}^i, \mathbf{u}^i) && \forall i, t \\ \mathbf{s}^i(0) &= \mathbf{s}_0^i,\ \mathbf{s}^i(t_f) = \mathbf{g}^i,\ \mathbf{s} \in \mathcal{X} && \forall i \\ \|\mathbf{u}^i\|_2 &\le u_{\max} && \forall i, t \end{aligned} \tag{2}$$

where $\mathcal{O}$, $\mathcal{U}$, and $\mathcal{S}$ are the observation, action, and state space, y is the local observation model, c is some cost function, f is the dynamical model, $\mathcal{X} \subset \mathcal{S}$ is the safe set capturing safety of all robots, t_f is the time of the simulation, and $u_{\max}$ is the maximum control bound.

Dynamical Model: We consider both single and double integrator systems. The single integrator state and action are position and velocity vectors in $\mathbb{R}^{n_q}$, respectively. The double integrator state is a stacked position and velocity vector in $\mathbb{R}^{2n_q}$, and the action is a vector of accelerations in $\mathbb{R}^{n_q}$. Here, n_q is the dimension of the workspace. The dynamics of the i^{th} robot for single and double integrator systems are:

$$\dot{\mathbf{s}}^i = \dot{\mathbf{p}}^i = \mathbf{u}^i \quad \text{and} \quad \dot{\mathbf{s}}^i = \begin{bmatrix} \dot{\mathbf{p}}^i \\ \dot{\mathbf{v}}^i \end{bmatrix} = \begin{bmatrix} \mathbf{v}^i \\ \mathbf{u}^i \end{bmatrix}, \tag{3}$$

respectively, where $\mathbf{p}^i$ and $\mathbf{v}^i$ denote position and velocity.

Observation Model: We are primarily focused on studying the transition from global to local, which is defined via an observation model, $y : \mathcal{V} \times \mathcal{S} \to \mathcal{O}$. An observation is:

$$\mathbf{o}^i = \left[\mathbf{e}^{ii}, \{\mathbf{s}^{ij}\}_{j \in \mathcal{N}_{\mathcal{V}}^i}, \{\mathbf{s}^{ij}\}_{j \in \mathcal{N}_{\Omega}^i}\right], \tag{4}$$

where $\mathbf{e}^{ii} = \mathbf{g}^i - \mathbf{s}^i$ and $\mathcal{N}_{\mathcal{V}}^i$, $\mathcal{N}_{\Omega}^i$ denote the neighboring set of robots and obstacles, respectively. These sets are defined by the observation radius, r_{sense}, e.g.,

$$\mathcal{N}_{\mathcal{V}}^i = \{j \in \mathcal{V} \mid \|\mathbf{p}^{ij}\|_2 \le r_{\text{sense}}\}. \tag{5}$$

We encode two different neighbor sets because we input robots and obstacles through respective sub-networks of our neural network architecture in order to generate heterogeneous behavior in reaction to different neighbor types. We denote the union of the neighboring sets as $\mathcal{N}^i$.

Performance Metrics: To evaluate performance, we have two criteria as specified by the optimal control problem. We define our metrics over the set of successful robots, $\mathcal{I}$, that reach their goal and have no collisions:

$$\mathcal{I} = \{i \in \mathcal{V} \mid \mathbf{s}^i(t_f) = \mathbf{g}^i \text{ and } \|\mathbf{p}_t^{ij}\| > r_{\text{safe}},\ \forall j, t\}. \tag{6}$$

Our first metric of success, r_s, is the number of successful robots, and our second metric, r_p, is the cost of deploying a successful robot trajectory. For example, if the cost function $c(\mathbf{s}, \mathbf{u})$ is the total control effort, the performance metrics are:

$$r_s = |\mathcal{I}|, \quad \text{and} \quad r_p = \sum_{i \in \mathcal{I}} \int_0^{t_f} \|\mathbf{u}^i\|_2 dt. \tag{7}$$

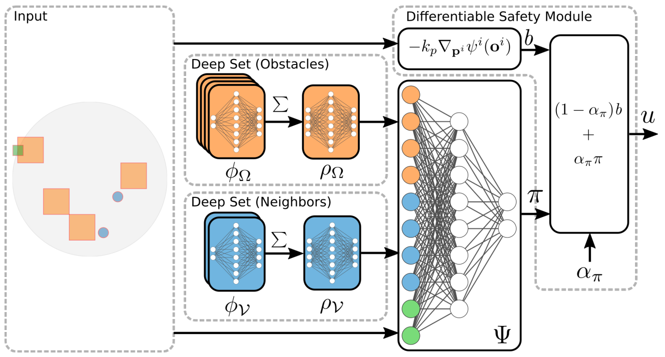

Fig. 2. Neural network architecture consisting of 5 feed-forward components. Each relative location of an obstacle is evaluated in ϕ_Ω; the sum of all ϕ_Ω outputs is the input of ρ_Ω (deep set for obstacles). Another deep set ($\phi_\mathcal{V}$ and $\rho_\mathcal{V}$) is used for neighboring robot positions. The desired control π is computed by Ψ, which takes the output of ρ_Ω, $\rho_\mathcal{V}$, and the relative goal location as input. The actual number of hidden layers is larger than shown. The formula for the single integrator b function is shown.

III. Algorithm Description and Analysis: GLAS

In this section, we derive the method of **GLAS**, **Gl**obal-to-**L**ocal **A**utonomy **S**ynthesis, to find policy u:

$$u(\mathbf{o}^i) = \alpha_\pi(\mathbf{o}^i)\pi(\mathbf{o}^i) + (1-\alpha_\pi(\mathbf{o}^i))b(\mathbf{o}^i), \tag{8}$$

where $\pi : \mathcal{O} \to \mathcal{U}$ is a learned function, $b : \mathcal{O} \to \mathcal{U}$ is a safety control module to ensure safety, and $\alpha_\pi : \mathcal{O} \to [0,1]$ is an adaptive gain function. We discuss each of the components of the controller in this section. The overview of our controller architecture is shown in Fig. 2.

A. Neural Policy Synthesis via Deep Imitation Learning

We describe how to synthesize the neural policy π that imitates the behavior of an expert, where the expert refers to a global optimal planner. Explicitly, we take batches of observation-action pairs from an expert demonstration dataset and we train a neural network by minimizing the loss on output action given an observation. To use this method, we need to generate an observation-action pair dataset from expert demonstration and design a deep learning architecture compatible with dynamic sensing network topologies.

1) Generating Demonstration Data: Our dataset is generated using expert demonstrations from an existing centralized planner [8]. This planner is resolution-complete and avoids local minima; it is computationally efficient so we can generate large expert demonstration datasets; and it uses an optimization framework that can minimize control effort, so the policy imitates a solution with high performance according to the previously defined metrics. Specifically, we create our dataset by generating maps with (i) fixed-size static obstacles with random uniformly sampled grid positions and (ii) start/goal positions for a variable number of robots, and then by computing trajectories using the centralized planner. For each timestep and robot, we retrieve the local observation, $\mathbf{o}^i$ by masking the non-local information with the observation model y, and retrieving the action, $\mathbf{u}^i$ through the appropriate derivative of the robot i trajectory, see Fig. 1. We repeat this process n_{case} times for each robot/obstacle case. Our dataset, $\mathcal{D}$, is:

$$\mathcal{D} = \{(\mathbf{o}^i, \mathbf{u}^i)_k \mid \forall i \in \mathcal{V}, \forall k \in \{1 \dots n_{\text{case}}\}, \forall t\}. \tag{9}$$

2) Model Architecture with Deep Sets: The number of visible neighboring robots and obstacles can vary dramatically during each operation, which causes the dimensionality of the observation vector to be time-varying. Leveraging the permutation invariance of the observation, we model variable number of robots and obstacles with the Deep Set architecture [15, 16]. Theorem 7 from [15] establishes this property:

Theorem 1: Let $f : [0,1]^l \to \mathbb{R}$ be a permutation invariant continuous function iff it has the representation:

$$f(x_1, \dots, x_l) = \rho\left(\sum_{m=1}^{l} \phi(x_m)\right), \tag{10}$$

for some continuous outer and inner function $\rho : \mathbb{R}^{l+1} \to \mathbb{R}$ and $\phi : \mathbb{R} \to \mathbb{R}^{l+1}$, respectively. The inner function ϕ is independent of the function f.

Intuitively, the ϕ function acts as a contribution from each element in the set, and the ρ function acts to combine the contributions of each element. In effect, the policy can learn the contribution of the neighboring set of robots and obstacles with the following network structure:

$$\begin{aligned}\pi(\mathbf{o}^i)_\text{n} &= \Psi([\rho_\Omega(\sum_{j \in \mathcal{N}_\Omega^i} \phi_\Omega(\mathbf{s}^{ij})); \rho_\mathcal{V}(\sum_{j \in \mathcal{N}_V^i} \phi_\mathcal{V}(\mathbf{s}^{ij}))]), \\ \pi(\mathbf{o}^i) &= \pi(\mathbf{o}^i)_\text{n} \ \min\{\pi_\text{max}/\|\pi(\mathbf{o}^i)_\text{n}\|_2, 1\},\end{aligned} \tag{11}$$

where the semicolon denotes a stacked vector and $\Psi, \rho_\Omega, \phi_\Omega, \rho_\mathcal{V}, \phi_\mathcal{V}$ are feed-forward networks of the form:

$$FF(\mathbf{x}) = W^l \sigma(\dots W^1 \sigma(\mathbf{x})), \tag{12}$$

where FF is a feed-forward network on input $\mathbf{x}$, W^l is the weight matrix of the l^{th} layer, and σ is the activation function. We define the parameters for each of the 5 networks in Sec. IV. We also scale the output of the π network to always be less than π_max, to maintain consistency with our baselines.

3) End-to-End Training: We train the neural policy π with knowledge of the safety module b, to synthesize a controller u with symbiotic components. We train through the output of u, not π, even though b has no tunable parameters. In effect, the parameters of π are updated such that the policy π smoothly interacts with b while imitating the global planner. With respect to a solution that trains through the output of π, end-to-end learning generates solutions with lower control effort, measured through the r_p metric (7), see Fig. 3.

4) Additional Methods in Training: We apply additional preprocessing methods to the observation to improve our training process performance and to regularize the data. We denote the difference between the original observation and the preprocessed data with an apostrophe; e.g., the input to the neural network is an observation vector denoted by $\mathbf{o}^{i\prime}$.

We scale the relative goal vector observation as follows:

$$\mathbf{e}^{ii\prime} = \alpha_g \mathbf{e}^{ii}, \text{ where } \alpha_g = \min\{r_\text{sense}/\|\mathbf{e}^{ii}\|, 1\}. \tag{13}$$

This regularizes cases when the goal is beyond the sensing radius. In such cases, the robot needs to avoid any robots/obstacles and continue toward the goal. However, the magnitude of $\mathbf{e}^{ii}$ outside the sensing region is not important.

We cap the maximum cardinality of the neighbor and obstacle sets with $\overline{\mathcal{N}_{\mathcal{V}}}$ and $\overline{\mathcal{N}_{\Omega}}$, e.g.,

$$\mathcal{N}_{\mathcal{V}}^{i\,\prime} = \{j \in \mathcal{N}_{\mathcal{V}}^{i} \mid \overline{\mathcal{N}_{\mathcal{V}}}\text{-closest robots w.r.t. } \|\mathbf{p}^{ij}\|\}. \quad (14)$$

This enables batching of observation vectors into fixed-dimension tensors for fast training, and upper bounds the evaluation time of π to guarantee real-time performance on hardware in large swarm experiments. We include an example observation encoding in Fig. 1.

B. System Safety

We adopt the formulation of safe sets used in control barrier functions and define the global safe set $\mathcal{X}$ as the super-level set of a global safety function $g : \mathcal{S} \rightarrow \mathbb{R}$. We define this global safety as the minimum of local safety functions, $h : \mathcal{O} \rightarrow \mathbb{R}$ that specify pairwise collision avoidance between all objects in the environment:

$$\mathcal{X} = \{\mathbf{s} \in \mathcal{S} \mid g(\mathbf{s}) > 0\}, \quad (15)$$

$$g(\mathbf{s}) = \min_{i,j} h(\overline{\mathbf{p}}^{ij}), \quad h(\overline{\mathbf{p}}^{ij}) = \frac{\|\overline{\mathbf{p}}^{ij}\| - r_{\text{safe}}}{r_{\text{sense}} - r_{\text{safe}}}, \quad (16)$$

where $\overline{\mathbf{p}}^{ij}$ denotes the vector between the closest point on object j to center of object i. This allows us to consistently define r_{safe} as the radius of the robot, where $r_{\text{safe}} < r_{\text{sense}}$. Intuitively, if a collision occurs between robots i, j, then $h(\overline{\mathbf{p}}^{ij}) < 0$ and $g(\mathbf{s}) < 0$, implying that the system is not safe. In order to synthesize local controls with guaranteed global safety, we need to show non-local safety functions cannot violate global safety. Consider a pair of robots outside of the neighborhood, $\|\overline{\mathbf{p}}^{ij}\| > r_{\text{sense}}$. Clearly, $h(\overline{\mathbf{p}}^{ij}) > 1$, implying this interaction is always safe.

C. Controller Synthesis

We use these safety functions to construct a global potential function, $\psi : \mathcal{S} \rightarrow \mathbb{R}$ that becomes unbounded when any safety is violated, which resembles logarithmic barrier functions used in the interior point method in optimization [17]. Similarly, we can construct a local function, $\psi^i : \mathcal{O} \rightarrow \mathbb{R}$:

$$\psi(\mathbf{s}) = -\log \prod_{i} \prod_{j \in \mathcal{N}^i} h(\overline{\mathbf{p}}^{ij}), \quad (17)$$

$$\psi^i(\mathbf{o}^i) = -\log \prod_{j \in \mathcal{N}^i} h(\overline{\mathbf{p}}^{ij}). \quad (18)$$

We use the local potential ψ^i to synthesize the safety control module b. We first state some assumptions.

Assumption 1: Initially, the distance between all objects is at least $r_{\text{safe}} + \Delta_r$, where Δ_r is a user-specified parameter.

Assumption 2: We assume robot i's geometry not to exceed a ball of radius r_{safe} centered at $\mathbf{p}^i$.

Theorem 2: For the single integrator dynamics (3), the safety defined by (15) is guaranteed under the control law (8) with the following $b(\mathbf{o}^i)$ and $\alpha_\pi(\mathbf{o}^i)$ for a scalar gains $k_p > 0$ and $k_c > 0$:

$$b(\mathbf{o}^i) = -k_p \nabla_{\mathbf{p}^i} \psi^i(\mathbf{o}^i) \quad (19)$$

$$\alpha_\pi(\mathbf{o}^i) = \begin{cases} \frac{(k_p - k_c)\|\nabla_{\mathbf{p}^i}\psi^i(\mathbf{o}^i)\|^2}{k_p\|\nabla_{\mathbf{p}^i}\psi^i(\mathbf{o}^i)\|^2 + |\langle \nabla_{\mathbf{p}^i}\psi^i(\mathbf{o}^i), \pi(\mathbf{o}^i)\rangle|} & \Delta_h(\mathbf{o}^i) < 0 \\ 1 & \text{else} \end{cases} \quad (20)$$

with $\Delta_h(\mathbf{o}^i) = \min_{j \in \mathcal{N}^i} h(\overline{\mathbf{p}}^{ij}) - \Delta_r$, and $\nabla_{\mathbf{p}^i}\psi^i$ in the proof of the full paper.

Proof: In this version, the proof is omitted for paper length considerations. ■

Theorem 3: For the double integrator dynamics given in (3), the safety defined by (15) is guaranteed under control law (8) for the barrier-controller and the gain defined as:

$$b = -k_v(\mathbf{v}^i + k_p \nabla_{\mathbf{p}^i}\psi^i) - k_p \frac{d}{dt}\nabla_{\mathbf{p}^i}\psi^i - k_p \nabla_{\mathbf{p}^i}\psi^i,$$

$$\alpha_\pi = \begin{cases} \frac{a_1 - k_c(k_p\psi^i + \frac{1}{2}\|\mathbf{v} - \mathbf{k}\|^2)}{a_1 + |a_2|} & \Delta_h(\mathbf{o}^i) < 0 \\ 1 & \text{else} \end{cases}, \quad (21)$$

$$a_1 = k_v\|\mathbf{v}^i + k_p\nabla_{\mathbf{p}^i}\psi^i\|^2 + k_p^2\|\nabla_{\mathbf{p}^i}\psi^i\|^2,$$

$$a_2 = \langle \mathbf{v}^i, k_p\nabla_{\mathbf{p}^i}\psi^i\rangle + \langle \mathbf{v}^i + k_p\nabla_{\mathbf{p}^i}\psi^i, \pi + k_p\frac{d}{dt}\nabla_{\mathbf{p}^i}\psi^i\rangle,$$

where $k_p > 0$, $k_c > 0$, and $k_v > 0$ are scalar gains, Δ_h is defined as in Theorem 2, $\frac{d}{dt}\nabla_{\mathbf{p}^i}\psi^i$ is defined in the proof and the dependency on the observation is suppressed for legibility.

Proof: In this version, the proof is omitted for paper length considerations. ■

IV. Experiments

We now present results of simulation comparing GLAS and its variants with state-of-the-art baselines as well as experimental results on physical quadrotors. Our supplemental video includes additional simulations and experiments.

A. Learning Implementation and Hyperparameters

For data generation, we use an existing implementation of a centralized global trajectory planner [8] and generate $\approx 2{\times}10^5$ (200 k) demonstrations in random $8\,\text{m} \times 8\,\text{m}$ environments with 10 or 20 % obstacles randomly placed in a grid pattern and 4, 8, or 16 robots (e.g., see Fig. 3 for 10 % obstacles and 8 robots). We sample trajectories every $0.5\,\text{s}$ and generate $|\mathcal{D}| = 40{\times}10^6$ (40 M) data points in total, evenly distributed over the 6 different environment kinds. We use different datasets for single and double integrator dynamics with different desired smoothness in the global planner.

We implement our learning framework in Python using PyTorch [18]. The ϕ_Ω and $\phi_\mathcal{V}$ networks have an input layer with 2 neurons, one hidden layer with 64 neurons, and an output layer with 16 neurons. The ρ_Ω and $\rho_\mathcal{V}$ networks have 16 neurons in their input and output layers and one hidden layer with 64 neurons. The Ψ network has an input layer with 34 neurons, one hidden layer with 64 neurons, and outputs π using two neurons. All networks use

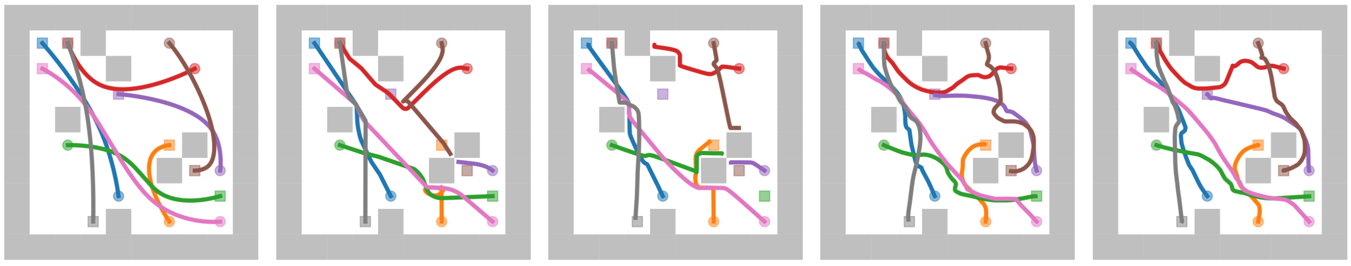
(a) Global (b) ORCA (c) GLAS Barrier (d) GLAS Two-stage (e) GLAS End-to-end

Fig. 3. Example trajectories for baselines (a-c) and our proposed method (d,e), where the goal is to move robots from their starting position (circles) to the goal position (squares). Our methods achieve the highest success rate. The GLAS end-to-end policy generates trajectories that use less control effort.

a fully connected feedforward structure with ReLU activation functions. We use an initial learning rate of 0.001 with the PyTorch optimizer ReduceLROnPlateau function, a batch size of 32 k, and train for 200 epochs. During manual, iterative hyperparameter tuning, we found that the hidden layers should at least use 32 neurons. For efficient training of the Deep Set architecture, we create batches where the number of neighbors $|\mathcal{N}_{\mathcal{V}}|$ and number of obstacles $|\mathcal{N}_{\Omega}|$ are the same and limit the observation to a maximum of 6 neighbors and 6 obstacles.

B. *GLAS Variants*

We study the effect of each component of the system architecture by comparing variants of our controller: *end-to-end*, *two-stage*, and *barrier*. End-to-end and two-stage are synthesized through (8), but differ in how π is trained. For end-to-end we calculate the loss on $u(\mathbf{o}^i)$, while for two-stage we calculate the loss on $\pi(\mathbf{o}^i)$. Comparing these two methods isolates the effect of the end-to-end training. The barrier variant is a linear feedback to goal controller with our safety module. Essentially, barrier is synthesized with (8), where the π heuristic is replaced with a linear goal term: $K\mathbf{e}^{ii}$, where, for single integrator systems, $K = k_p I$, and for double integrator systems, $K = [k_p I, k_v I]$, with scalar gains k_p and k_v. Studying the performance of the barrier variant isolates the effect of the global-to-local heuristic training.

C. *Single Integrator Dynamics*

We compare our method with ORCA, a state-of-the-art decentralized approach for single integrator dynamics. Unlike GLAS, ORCA requires relative velocities with respect to neighbors in addition to relative positions. All methods compute a velocity action with guaranteed safety.

We show example trajectories for the global planner, ORCA, and GLAS variants in Fig. 3. In Fig. 3(b)/3(c), the purple and brown robots are getting stuck in local minima caused by obstacle traps. In Fig. 3(d)/3(e), our learned policies are able to avoid those local minima. The end-to-end approach produces smoother trajectories that use less control effort, e.g., red and brown robot trajectories in Fig. 3(e).

1) Evaluation of Metrics: We deploy the baseline and variants over 100 validation cases with 2, 4, 8, 16, and 32 robots and 10 % and 20 % obstacle density (10 validation instances for each case) and empirically evaluate the metrics

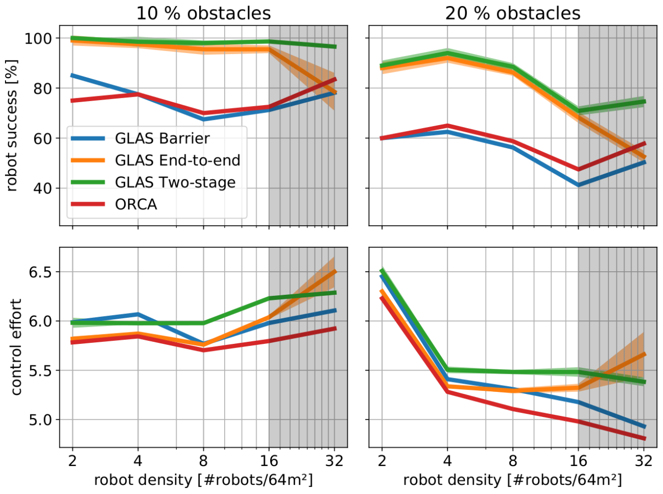

Fig. 4. Success rate and control effort with varying numbers of robots in a $8\,\mathrm{m} \times 8\,\mathrm{m}$ space for single integrator systems. Shaded area around the lines denotes standard deviation over 5 repetitions. The shaded gray box highlights validation outside the training domain.

defined in (7). Our training data only contains different examples with up to 16 robots. We train 5 instances of end-to-end and two-stage models, to quantify the effect of random-weight initialization in the neural networks on performance, see Fig. 4.

In the top row, we consider the success metric r_s. In a wide range of robot/obstacle cases (2–16 robots/$64\,\mathrm{m}^2$), our global-to-local methods outperform ORCA by 20 %, solving almost all instances. Our barrier variant has a similar success rate as ORCA, demonstrating that the neural heuristic π is crucial for our high success rates. The two-stage approach generalizes better to higher-density cases beyond those in the training data. We observe the inverse trend in the double integrator case and analyzing this effect is an interesting future direction.

In the bottom row, we measure control effort r_p. Our end-to-end approach uses less control effort than the two-stage approach. ORCA has the lowest control effort, because the analytical solution to single integrator optimal control is a bang-bang controller, similar in nature to ORCA's implementation.

2) Effect of Radius of Sensing on Performance: We quantify the transition from local-to-global by evaluating the performance of models trained with various sensing radii and dataset size. We evaluate performance on a validation set of 4, 8, and 16 robot cases with 10 % and 20 % obstacle

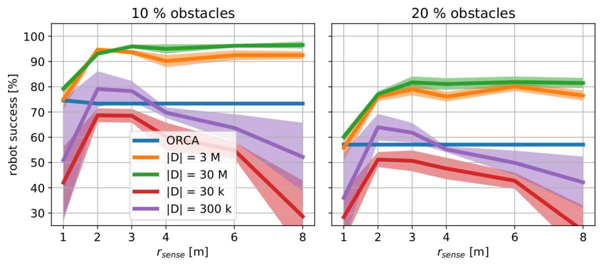

Fig. 5. Effect of sensing radius and amount of training data on robot success rate. The validation has 4, 8, and 16 robot cases with 10 instances each. Training and validation were repeated 5 times; the shaded area denotes the standard deviation.

densities. First, we found that there exists an optimal sensing radius for a given amount of data, which increases with larger datasets. For example, in the $20\,\%$ obstacle case, the optimal sensing radius for $|\mathcal{D}| = 300\,\mathrm{k}$ is around $2\,\mathrm{m}$ and the optimal radius for $|\mathcal{D}| = 30\,\mathrm{M}$ is $8\,\mathrm{m}$. Second, we found that between models of various dataset sizes the performance gap at small sensing radii is smaller compared to the performance gap at large sensing radii. This result suggests that little data is needed to use local information well, and large amounts of data is needed to learn from global data.

D. Experimental Validation with Aerial Swarms

We implement the policy evaluation (π, α_π, b) in C to enable real-time execution on-board of Crazyflie 2.0 quadrotors using double integrator dynamics (see Fig. 1). The quadrotors use a small STM32 microcontroller with $192\,\mathrm{kB}$ SRAM running at $168\,\mathrm{MHz}$. Our policy evaluation takes $3.4\,\mathrm{ms}$ for 1 neighbor and $5.0\,\mathrm{ms}$ for 3 neighbors, making it computationally efficient enough to execute our policy in real-time at $40\,\mathrm{Hz}$. On-board, we evaluate the policy, forward-propagate double integrator dynamics, and track the resulting position and velocity setpoint using a nonlinear controller. The experimental validation demonstrates that our policy generalizes to novel environments where the obstacles are arranged in continuous space, as opposed to on a grid.

We use a double integrator GLAS end-to-end policy in three different scenarios with up to 3 obstacles and 12 quadrotors. We fly in a motion capture space, where each robot is equipped with a single marker, using the Crazyswarm [19] for tracking and scripting. Our demonstration shows that our policy works well on robots and that it can also handle cases that are considered difficult in decentralized multi-robot motion planning, such as swapping positions with a narrow corridor.

V. Conclusion

In this work, we present GLAS, a novel approach for multi-robot motion planning that combines the advantages of existing centralized and distributed approaches. Unlike traditional distributed methods, GLAS avoids local minima in many cases. Unlike existing centralized methods, GLAS only requires local relative state observations, which can be measured on board or communicated locally. We propose an end-to-end training approach using a novel differentiable safety method compatible with general dynamical models, resulting in a dynamically-coupled motion planner with guaranteed collision-free operation and empirically-validated low control effort solutions. In future work, we will explore adaptive data-set aggregation methods [20] for more efficient expert querying. We will also compare the computational effort and performance of the Deep Set representation with deep CNN methods as well as Graph Neural Network methods [21].

References

[1] J. van den Berg, S. J. Guy, M. C. Lin, and D. Manocha, "Reciprocal n-body collision avoidance", in *Int. Symp. on Robot. Res.*, ser. Springer Tracts in Advanced Robotics, vol. 70, Springer, 2009, pp. 3–19.

[2] S. Bandyopadhyay, S.-J. Chung, and F. Y. Hadaegh, "Probabilistic and distributed control of a large-scale swarm of autonomous agents", *IEEE Trans. Robot.*, vol. 33, no. 5, pp. 1103–1123, 2017.

[3] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots", in *Autonomous Robot Vehicles*, Springer, 1990, pp. 396–404.

[4] E. Rimon and D. E. Koditschek, "Exact robot navigation using artificial potential functions", *IEEE J. Robot. Autom.*, vol. 8, no. 5, pp. 501–518, 1992.

[5] H. G. Tanner and A. Kumar, "Formation stabilization of multiple agents using decentralized navigation functions", in *Robotics: Science & Systems*, 2005, pp. 49–56.

[6] L. Wang, A. D. Ames, and M. Egerstedt, "Safety barrier certificates for collisions-free multirobot systems", *IEEE Trans. Robot.*, vol. 33, no. 3, pp. 661–674, 2017.

[7] D. Morgan, G. P. Subramanian, S.-J. Chung, and F. Y. Hadaegh, "Swarm assignment and trajectory optimization using variable-swarm, distributed auction assignment and sequential convex programming", *I. J. Robotics Res.*, vol. 35, no. 10, pp. 1261–1285, 2016.

[8] W. Hönig, J. A. Preiss, T. K. S. Kumar, G. S. Sukhatme, and N. Ayanian, "Trajectory planning for quadrotor swarms", *IEEE Trans. Robot.*, vol. 34, no. 4, pp. 856–869, 2018.

[9] C. E. Luis and A. P. Schoellig, "Trajectory generation for multiagent point-to-point transitions via distributed model predictive control", *IEEE Trans. Robot. Autom. Lett.*, vol. 4, no. 2, pp. 375–382, 2019.

[10] G. Sartoretti, J. Kerr, Y. Shi, *et al.*, "PRIMAL: pathfinding via reinforcement and imitation multi-agent learning", *IEEE Trans. Robot. Autom. Lett.*, vol. 4, no. 3, pp. 2378–2385, 2019.

[11] Q. Li, F. Gama, A. Ribeiro, and A. Prorok, "Graph neural networks for decentralized multi-robot path planning", *CoRR*, vol. abs/1912.06095, 2019.

[12] A. Khan, V. Kumar, and A. Ribeiro, "Graph policy gradients for large scale unlabeled motion planning with constraints", *CoRR*, vol. abs/1909.10704, 2019.

[13] A. Khan, C. Zhang, S. Li, *et al.*, "Learning safe unlabeled multi-robot planning with motion constraints", in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2019, pp. 7558–7565.

[14] D. Raju, S. Bharadwaj, and U. Topcu, "Decentralized runtime synthesis of shields for multi-agent systems", *CoRR*, vol. abs/1910.10380, 2019.

[15] M. Zaheer, S. Kottur, S. Ravanbakhsh, *et al.*, "Deep sets", in *Advances in Neural Information Processing Systems 30*, 2017, pp. 3391–3401.

[16] G. Shi, W. Hönig, Y. Yue, and S.-J. Chung, "Neural-swarm: Decentralized close-proximity multirotor control using learned interactions", in *Proc. IEEE Int. Conf. Robot. Autom.*, 2020.

[17] S. P. Boyd and L. Vandenberghe, "Convex Optimization". Cambridge University Press, 2014, ISBN: 978-0-521-83378-3.

[18] A. Paszke, S. Gross, F. Massa, *et al.*, "Pytorch: An imperative style, high-performance deep learning library", in *Advances in Neural Information Processing Systems 32*, 2019, pp. 8024–8035.

[19] J. A. Preiss*, W. Hönig*, G. S. Sukhatme, and N. Ayanian, "Crazyswarm: A large nano-quadcopter swarm", in *Proc. IEEE Int. Conf. Robot. Autom.*, 2017, pp. 3299–3304.

[20] S. Ross, G. J. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning", in *Proc. Int. Conf. Artificial Intelligence and Statistics*, vol. 15, 2011, pp. 627–635.

[21] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model", *IEEE Trans. Neural Netw.*, vol. 20, no. 1, pp. 61–80, 2008.