Centralized and Decentralized Application of Neural Networks Learning Optimized Solutions of Distributed Agents

Joshua A. Shaffer[a] and Huan Xu[b]

[a]Department of Aerospace Engineering, University of Maryland, College Park, MD, USA
[b]Department of Aerospace Engineering, University of Maryland, College Park, MD, USA

ABSTRACT

This paper explores a methodology for training recurrent neural networks in replicating path planning solutions from optimization problems of multi-agent systems. Training data is generated by solving a centralized nonlinear programming problem, from which both centralized (representing all agents) and decentralized (representing individual agents) recurrent neural networks are trained with reinforcement learning to produce an agent's state path through fixed time-step execution. Path-tracking controllers are formulated for each agent to follow the path generated by the network. The control signal from such a controller should mimic that of the optimized solution. Results for a 10 agent, 2D dynamics problem with synchronized arrival and collision avoidance constraints showcase the ability of this approach to achieve the desired controller execution and resulting state path. Through these results, this work showcases the ability of recurrent neural networks to learn and generalize centralized and synchronous multi-agent optimization solutions, the end of which is a much more computationally fast multi-agent path planner that trends the slower-to-compute optimization solutions.

Keywords: Multi-agent, centralized neural network, decentralized neural network, reinforcement learning

1. INTRODUCTION

Path planning for multi-agent systems encompasses a broad range of methodologies, from which the resulting applications require varying degrees of accuracy and computational speed for the solutions solved. The introduction of kinematic/dynamic constraints alongside global optimality conditions increases the difficulty in producing quick and feasible solutions.[1] As a result, faster path planning methodologies tend to ignore or greatly reduce kinematic/dynamic constraints in favor of finding solutions to satisfy a varied environment.[2] Oftentimes when kinematics/dynamics and optimality conditions must be considered, they are abstracted to simpler models alongside the use of heuristics, from which controllers must enforce in real-time.[3] Application performance of such methods is greatly dependent upon how well the abstracted system applies to the full dynamic model and controller utilized.

In cases where the total solution must not only consider environmental constraints but also full dynamic/kinematic constraints and optimality conditions, optimization focused algorithms involve much higher computation times, which impacts their real-time performance.[3] With respect to multi-agent systems, this requires greatly increased state spaces, which only exacerbates the issue.[1,3,4] To circumvent this problem, many multi-agent planners operate in a decentralized manner, often over receding horizons.[5–7] Unfortunately, the ability of decentralized planners to optimize a global cost and ensure synchronous behaviors is greatly reduced as compared to a global planner.

Because of the aforementioned issues related to both kinematic/dynamic and non-kinematic/dynamic focused path planning with optimality conditions (especially with respect to multi-agent systems), an encompassing solution that can incorporate the benefits of centralized path planning alongside the speed of a decentralized planner is of great interest. Decreasing the computation time associated with an algorithm that satisfies optimality conditions alongside kinematic/dynamic constraints typically requires some approach of quickly providing strong initial guesses for generalized methods,[8] or re-planning from adequate initial guesses.[9]

Machine learning with recurrent neural networks (RNN) provides a platform from which unknown time-dependent processes can be form-fitted through training data in which the correct input and output sets are known. The use of

Further author information: (Send correspondence to H.X.)
J.A.S.: E-mail: jshaffe9@terpmail.umd.edu
H.X.: E-mail: mumu@umd.edu, Telephone: +33 (0)1 98 76 54 32

these (and neural networks in general) to learn and execute sequential actions to minimize or maximize a reward is often referred to deep reinforcement learning (RL). RL with neural networks provides two primary benefits with respect to path planning problems, speed of computation and generalization of the solution space to new environments. With respect to multi-agent systems, neural networks have the potential to tackle problems related to either scalability (controlling large numbers of agents at once) or learning decentralized protocols (generalizing group rules among multiple agents to localized protocols).[10]

For this paper, we present a methodology utilizing an RNN to learn the solution space of a robust, computationally slow centralized path planner that considers kinematic/dynamic constraints and optimality conditions across all agents. From here, two forms of RNNs are created to generate the state paths of the agents. The first form is a centralized network that generates the state paths of all agents together. The second form is decentralized where individual RNNs are formulated to generate the paths for individual agents. We present performance comparisons between the centralized and decentralized RNNs in how well they can generate state paths that follow the optimally controlled state path for a given scenario. Additionally we assess how well path tracking controllers can follow these path outputs and recreate the desired control signals. These assessments illustrate the feasibility of RNNs and path-tracking controllers to recreate the originally slow-to-compute, centralized optimization solutions for multi-agent systems, where the RNNs provide far faster computation times as compared to the original solvers.

The rest of the paper follows as such: Section 2 discusses related work to the presented solution, Sec. 3 introduces the formal problem definition, Sec. 4 formulates the entire methodology and approach, Sec. 5 provides the sample scenario tested alongside provided constraints, Sec. 6 examines the results of our implementation, and Sec. 7 concludes the paper and presents future work.

2. RELATED WORK

The use of machine learning directly in path planning is an expansive research topic, in part due to the opposing nature between the need for constraint satisfaction in generated paths and the difficulty in formally verifying machine learning outputs.[11] Despite such hurdles, various authors have investigated the benefits of machine learning in aiding path planning, albeit from varying perspectives.

Ref. 12 examined the specific application of using a 2-input, 2-output neural network to provide controls to an inter-planetary rover navigating a known terrain, tested in simulation. Specifically, the 2 inputs represented x and y coordinates, with the outputs representing control signals to the wheels. In this case, training was performed on a static environment, and the rover had to navigate from any point to a static final destination. This work showcases the ability for a simpler network to accurately abstract the dynamics and controls associated with using position feedback to drive a robot to a final destination, a concept also successfully explored in Ref. 13. Additionally, Ref. 14 achieved such an approach formulated in a local frame about the robot for easy integration with actual sensor data, achieving practical application.

Ref. 15 utilized pulse-coupled neural networks for determining the shortest path in an unknown environment, similarly in Ref. 16. Again, the state space was discretized, and dynamics were enforced at lower levels. Ref. 15 also performed physical tests, though, like the previously-mentioned Ref. 14, and moved towards validating the practical application of a neural network based approach.

With respect to multi-agent systems, Ref. 17 explored how well convolutional neural networks can create velocity controls to match that of a multi-drone flocking algorithm. This decentralized approach utilized visual data as input to a convolutional neural network which then provided output velocity commands trained to match those of a common relative-position-based flocking algorithm. The flocking algorithm's velocity commands and resulting state paths were not formulated with respect to optimization of a metric over the entire state and control paths of the agents. Ref. 18 explored how mean-embeddings of a decentralized multi-agent environment could improve the ability of a neural network to provide the correct control outputs of individual agents in a swarm. Similarly to Ref. 17, the result was a network that could reproduce the desired commands of a known multi-agent control algorithm (one not formulated to optimize entire state and control paths with respect to a desired metric) under condensed information of relative agents (i.e. through the use of mean embeddings).

The authors in Ref. 8 and Ref. 19 present methodologies most similar to the one presented in this paper. In Ref. 8, regression learning is utilized to select previous planner solutions for a robotic manipulator as initial guesses for the optimization planner, resulting in speed-ups of up to an order in magnitude. In Ref. 19, an RNN is utilized to learn from

shortest path solutions between two points provided randomized obstacles. Additionally, an environment encoder network is utilized to reduce the state space size of the environment and robustly represent such constraints. The result is an RNN that takes in any obstacle configuration and produces the shortest path between a current location and the desired final location. The computational advantages were up to an order of magnitude or more when compared to some of the fastest conventional planners, and scaled well with larger state spaces.

Each of the aforementioned approaches represent varying ways of utilizing networks in path planning, with the foremost advantage of decreased computation costs in execution. With respect to our work, Ref. 12, Ref. 13, and Ref. 14 showcased the ability of a network to abstract dynamics of specific scenarios, Ref. 8 and Ref. 19 displayed the ability of a network to generate physical paths with respect to a specific optimization parameter for highly varied environments, and Ref. 17 and Ref. 18 showcased how well neural networks could recreate decentralized control protocols of multi-agent systems. This paper attempts to integrate an RNN's abilities of learning dynamics and generating optimized paths in order to achieve both aspects.

In the following sections, we present relevant problem formulations and methods of solving each for the three primary components of this methodology: generating training data composed of optimized solutions in a varied environment through nonlinear programming (NLP), creation and training of an RNN on said data, and executing controls over generated paths from the trained RNN. These components are represented in Fig. 1. The result is a multi-agent path planning RNN and individual agent controller that operate far faster than the robust, slower optimizer and yet still provide satisfactory solutions.

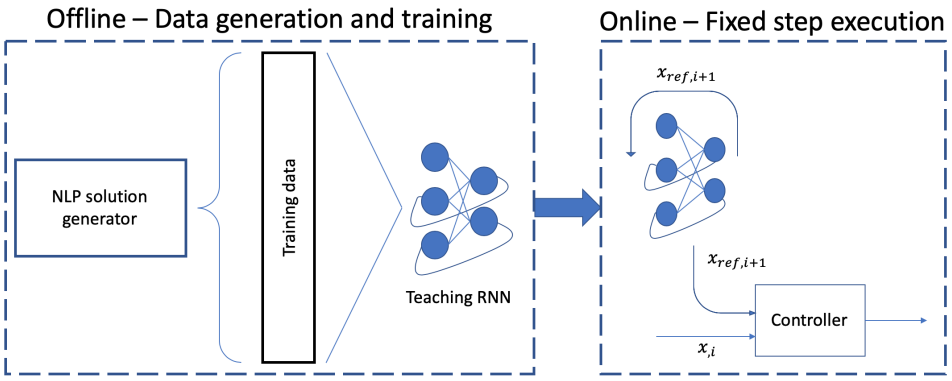

Figure 1: Overview of the proposed methodology. Offline data generation of optimized solutions through nonlinear programming (NLP) is used to train RNNs in a supervised/reinforced learning scheme. From such, online execution makes use of the state paths generated by the RNN to provide reference trajectories for a path-tracking controller, of which the generated controls trend that of the optimized solution.

3. PROBLEM FORMULATION

3.1 Path Planning

Our generalized path planning problem is formulated as the optimal control problem presented in the following way for a system of agents:

$$\underset{\mathbf{u}(t),t_f}{\text{minimize}} \quad \int_0^{t_f} \left(\sum_{i=1}^{N_a} \mathscr{W}(t,\mathbf{x}_i(t),\mathbf{u}_i(t)) \right) dt + \sum_{i=1}^{N_a} \mathscr{L}(\mathbf{x}_i(0),t_f,\mathbf{x}_i(t_f)), \tag{1a}$$

subject to

$\forall i \in \{1,...,N_a\}$

$$\dot{\mathbf{x}}_i = \mathbf{f}(\mathbf{x}_i(t),\mathbf{u}_i(t)), \tag{1b}$$

$$\mathbf{x}_i(0) = \mathbf{x}_{i,0}, \tag{1c}$$

$$\mathbf{x}_i(t_f) = \mathbf{x}_{i,f}, \tag{1d}$$

$$\mathbf{C}_i(\mathbf{x}(t),\mathbf{u}(t),\mathbf{P}_i) \leq \mathbf{0}. \tag{1e}$$

Here, $\mathbf{x}_i(t) \in \mathbb{R}^N$ is the system state of an agent, $\mathbf{u}_i(t) \in \mathbb{R}^M$ is the control signal of said agent, N_a is the number of agents, and t is time. Eq. (1a) is an optimization metric across all agents (consisting of both an integrated scalar function $\mathscr{W}$ and non-integrated scalar function $\mathscr{L}$) for the provided kinematic/dynamic systems of each agent defined by Eq. (1b), in which $\mathbf{f}(\mathbf{x}_i(t),\mathbf{u}_i(t)) \in \mathbb{R}^N$. Eq. (1c) and (1d) represent desired initial and final conditions on the state of each agent, respectively, and Eq. (1e) (with $\mathbf{C}_i(\mathbf{x}(t),\mathbf{u}(t),\mathbf{P}_i) \in \mathbb{R}^Q$) contains all desired nonlinear constraints for agent i utilizing the entire swarm vectors $\mathbf{x}(t)$ and $\mathbf{u}(t)$. The vector $\mathbf{P}_i \in \mathbb{R}^O$ represents all possible static variables in the constraints for the agent i. This formulation enables a global optimization across all agents utilizing individual environments for each agent. A global control solution to the above optimization formulation and its corresponding state path is represented as $\mathbf{G}(t) \in \mathbb{R}^{N_a(N+M)}$, with individual agent solutions of $\mathbf{G}_i(t) \in \mathbb{R}^{N+M}$.

3.2 Network Learning

Provided a domain for the initial and final conditions of each agent, $\mathbf{x}_{i,0,min} \leq \mathbf{x}_{i,0} \leq \mathbf{x}_{i,0,max}$ and $\mathbf{x}_{i,f,min} \leq \mathbf{x}_{i,f} \leq \mathbf{x}_{i,f,max}$, and a constraint domain for each agent of $\mathbf{P}_{i,min} \leq \mathbf{P}_i \leq \mathbf{P}_{i,max}$, individual optimized solutions $\mathbf{G}(t)$ for the entire swarm exist as outputs to the optimization solution when using these variables. Provided sets X_0 and X_f and P_{set} of sample points (each composed of samples for each agent) from these domains, a set of solutions G_{set} exists, composed of the solutions to the optimization problem using these variables.

Given G_{set}, an RNN must be formed and trained upon the provided data, operating under fixed time-step t_k. The RNN is represented under two forms, centralized and decentralized. The centralized form is represented generally as,

$$\mathbf{x}(t_k+1) = \Phi(\mathbf{x}(t_k),t_k,\mathbf{x}_i,\mathbf{x}_f,\mathbf{P}), \tag{2}$$

where $\mathbf{x}(t_k)$ is the vector composed of all individual agents $\mathbf{x}_i(t_k)$, $\mathbf{P}$ is the vector composed of all individual agent environments $\mathbf{P}_i$, and t_k represents sampled time. At a given time step and for all solutions $\mathbf{G}(t) \in G_{set}$, the RNN output must be trained to minimize a performance function composed of the term,

$$\Phi(\mathbf{x}(t_k),t_k,\mathbf{x}_0,\mathbf{x}_f,\mathbf{P}) - \mathbf{G}_x(t_k+1), \tag{3}$$

where $\mathbf{G}_x(t)$ is the state component of all agents of a given solution vector $\mathbf{G}(t)$.

The decentralized form for each agent is represented generally as,

$$\mathbf{x}_i(t_k+1) = \Phi_i(\mathbf{x}_i(t_k),t_k,\mathbf{x}_0,\mathbf{x}_f,\mathbf{P}_i), \tag{4}$$

where $\mathbf{x}_i(t_k)$ is the individual agent's state vector, and $\mathbf{P}_i$ is the individual agent's environment. At a given time step and for all solutions $\mathbf{G}_i(t) \in \mathbf{G}(t) \in G_{set}$, each agent RNN must be trained to minimize a performance function composed of the term,

$$\Phi_i(\mathbf{x}_i(t_k),t_k,\mathbf{x}_0,\mathbf{x}_f,\mathbf{P}_i) - \mathbf{G}_{i,x}(t_k+1), \tag{5}$$

where $\mathbf{G}_{i,x}$ is the state component of an agent for the given agent solution vector $\mathbf{G}_i$.

3.3 Path Execution

Provided an agent's state path $\sigma_i(t_k) : \mathbb{T} \longrightarrow \mathbb{R}^N$ generated by closed loop execution of the RNN under set values of $\mathbf{x}_{i,0}$, $\mathbf{x}_{f,0}$, and $\mathbf{P}_i$ with $\|\mathbf{x}_{i,0} - \sigma_i(0)\| \leq \delta$, where δ is an arbitrarily small number, a controller $\mathbf{u}_{i,e}(t, \mathbf{x}_i(t), \sigma_i(t_k))$ must be formulated such that the error norm $\|\mathbf{x}_i(t) - \sigma_i(t_k)\|$ is minimized while all constraints $\mathbf{C}_i(\mathbf{x}(t), \mathbf{u}(t), \mathbf{P}_i) \leq \mathbf{0}$ are satisfied. Additionally, the control signal $\mathbf{u}_{i,e}$ should mimic that of the true control signal $\mathbf{G}_{i,u}(t)$, minimizing the error between the optimized control signal of an agent and the executed control signal.

4. METHODOLOGY

Each individual problem discussed in the previous section is solved and integrated with the other solutions into the entire methodology. This overall scheme is shown in Fig. 1, and presented in further detail here.

4.1 Pseudospectral Method and Nonlinear Programming

Individual solutions of the optimization problem presented in Eq. (1a) - (1e) are formulated utilizing pseudospectral methods and solved as nonlinear programming problems (NLP). The benefit of such an approach is the flexibility in representing a broad range of problems and kinematic/dynamic systems. For our work, we utilize the pseudospectral method presented in Ref. 20 and Ref. 21. In this approach, Chebyshev polynomials of the form

$$C_N = \cos\left(N_t \cos^{-1}(\tau)\right), \tag{6}$$

where $N_t + 1$ represents the number of collocation nodes and $\tau \in \{-1, 1\}$, serve as representations of the state over the optimization horizon.

At discrete $N_t + 1$ nodes of the polynomial, formulated from the chosen values of τ as

$$\tau_k = -\cos\left(\frac{\pi k}{N_t}\right) \quad k = \{0, 1, ..., N_t\}, \tag{7}$$

the polynomial's time derivative is constrained to equal that of the state's dynamics. These discrete state points, alongside an equal number of control points, constitute the free parameters in an NLP problem (formulation provided in the Appendix). In this discrete scheme, NLP is well suited for finding an optimized solution under the aforementioned problem formulation.

4.2 Recurrent Neural Networks and Reinforcement-based Learning

RNNs, specifically of the Jordan network form, are feedforward neural networks in which the output vector of the network serves as part of the input vector. This structure serves well to predicting state paths since a state at t_k is part of the input used in providing the next state output at t_{k+1}. Layers of these networks function similarly to the usual structure present in feedforward multilayer perceptron (MLP) networks. Specifically, an input vector to a layer is multiplied by a matrix of weights and then added to a bias term, from which an activation function $\sigma_h(\cdot)$ is applied to the result, producing an output vector. This output vector serves as the input to the next layer in the network, if one exists.

As a result, training of such networks can utilize typical feedforward schemes without worrying about backpropagation through time. This means that training of both the centralized and decentralized networks is performed per sampled point of each solution $\mathbf{G}(t)$ found through the NLP solver by minimizing the standard mean square error performance function,

$$MSE = \frac{1}{N_t} \sum_{k=0}^{k=N_t} \left\| \Phi(\mathbf{x}(t_k), \mathbf{x}_i, \mathbf{x}_f, \mathbf{P}) - \mathbf{G}_x(t_k + 1) \right\|^2. \tag{8}$$

For the decentralized RNNs, each network is trained utilizing the individual agent's state information (i.e. $(\Phi_i - G_{i,x})$ is used for the *MSE* calculations)

Training on just the provided optimized solutions yields poor results when the networks are recursively executed to produce $\sigma_i(t)$, where the output states are fed back in as inputs to generate a state path in time under a given environment. Improving this closed-loop execution requires a form of learning that iteratively assesses the network's output states per closed-loop execution and trains the network based on these paths. This learning scheme is outlined in Algorithm 1.

Algorithm 1 Whole-path Reinforcement Learning

procedure TRAIN_RNN(G_{set}) ▷ Train centralized RNN or decentralized RNNs from set of optimized solutions
  Train RNN over $(X_{in}, Y_{out}) \subseteq (G_{set}(t_k), G_{set}(t_k+1))$ for all k ▷ Utilizing any common training scheme
  for 1 to $Training\_Iterations$ **do**
    for $i \leftarrow 1 : len_{horizon} : N_{t_f}$ **do**
      Generate $\sigma(t_k)$ from $k=0$ to $k=i$ for all x_i, x_f, and P sets
      Create training data $(X_{horizon}, Y_{horizon}) = (\sigma(t_k), G_{set}(t_k+1))$ from $k=0$ to $k=i$
      Append $(X_{horizon}, Y_{horizon})$ to (X_{in}, Y_{out}) to create (X_{mod}, Y_{mod})
      Train RNN over (X_{mod}, Y_{mod})

This method of training we developed for this problem exploits a specific balance between supervised learning and reinforcement learning. Reinforcement learning, in general, assesses the results of a network's outputs with respect to a transition relation, calculating a reward value of such. The network is then retrained to increase such rewards in an iterative manner. Unlike the reinforcement learning method explored in Ref. 18, our approach compares network output actions (i.e. next continuous state) to those of already optimized training data. As a result, this combines supervised training over ideal data (i.e. the optimized sequential state paths) with that of reinforced learning (i.e. training the network to produce entire state paths that minimize the distance to ideal paths), enabling a closed-loop execution that follows a path optimized over the entire execution duration.

4.3 Path-tracking Controller

Design of a controller to track the generated path $\sigma(t)$ of the RNN is a problem-specific task tied to the kinematics/dynamics of the prescribed system. The ability to track an arbitrary path provided a system and control definition is dependent upon the controllability of the system and realizability of a reference track.[22] Fortunately, the generated path, assuming minimal errors produced by the RNN, is already derived from a dynamic/kinematic formulation, with considerations to controllability enforced in the optimization. Under such, a control signal must exist that can track the system path accurately.

For this paper, we observe systems in which feedback control loops for each individual agent are more than adequate for following the produced RNN state history. For a simple mechanical system (as explored in the *Implementation* section), the velocity feedback portion of the control signal constitutes the error in desired velocity of the state with that of the RNN path, and the position feedback portion constitutes the error between the current state position and the desired position of the RNN path. This control signal is formulated as,

$$\mathbf{u}_{f,f}(x(t), \sigma(t_k)) = -K_p(\mathbf{x}_p(t) - \sigma_p(t_k)) - K_v(\mathbf{x}_v(t) - \sigma_v(t_k)), \tag{9}$$

where t is continuous time, t_k is sampled time per the RNN time interval, $\mathbf{x}_p$ is the position vector of the state, σ_p is the position vector of the RNN output path, $\mathbf{x}_v$ is the velocity vector of the state, σ_v is the velocity vector of the RNN output path, K_p is the position gain matrix, and K_v is the velocity gain matrix.

5. IMPLEMENTATION

While this methodology is general enough to be applicable to far more complex problems, a simple 2D synchronized, multi-agent, point-to-point problem with collision avoidance and dynamics is explored to investigate the feasibility of the proposed methodology and performance difference between the two RNNs. The model utilized per agent is,

$$\dot{\mathbf{x}}_i = \mathbf{v}_i \tag{10}$$

$$\dot{\mathbf{v}}_i = \mathbf{u}_i m, \tag{11}$$

where $m = 1$. The state boundary is $(-6,-8) \leq (x,y) \leq (6,8)$ and $(-2.5,-2.5) \leq (v_x, v_y) \leq (2.5, 2.5)$, with control constraints of $(-10,-10) \leq (u_x, u_y) \leq (10,10)$. The environment consists of collision avoidance between all agents, with $\mathbf{C}_i$ formulated as,

$$0.3 - \left\|\mathbf{x}_{p,i} - \mathbf{x}_{p,j}\right\| \quad \forall j \in \{i, ..., N_a\}, \tag{12}$$

where x_p represents the position of an agents. The desired final time is fixed at 20 seconds. Initial state positions (with zero velocity) are sampled randomly within the state domain, while final state positions (with zero velocity) are set at equal intervals along the x axis, fixed for each agent. The optimization function to minimize is,

$$\int_0^{t_f=20} \sum_{i=1}^{N_a} \|\mathbf{u}_i\| \, dt. \tag{13}$$

Utilizing 10 agents ($N_a = 10$), approximately 5,000 solutions were generated for training and validation (2,800 used in training, 700 used in validation assessment during training, and 1,500 used in validation post-training), each consisting of randomized initial conditions for each agent. The solver SNOPT[23] was utilized for solving the optimization problem formulated as an NLP problem in each configuration. The centralized RNN was constructed with 5 hidden layers (sizes 300, 200, 150, 100, and 80) utilizing the hyperbolic tangent activation function and an output layer utilizing a linear activation. The decentralized RNNs were constructed with 4 hidden layers (sizes 150, 110, 70, 20, and 4) utilizing the hyperbolic tangent activation function and output layers utilizing the linear activations, too. Keras[24] with the TensorFlow[25] backend were utilized for network construction and training through the Nesterov Adam optimization scheme[26] (learning rate of 0.002 and schedule delay of 0.01). Controller execution over the closed-loop paths generated by the resulting RNNs utilized gains $K_p = K_v = 25$ for all agents.

6. RESULTS

The optimization problem discussed above is designed to assess the performance of both the centralized and decentralized RNN forms in recreating synchronized optimal paths with collision avoidance constraints. In the centralized form, all agent states are known to all other agents states within the network. In the decentralized form, only initial global information (the initial positions of all agents) is provided to each RNN. The performance of both RNNs is assessed on their ability to recreate the optimal state paths over training and validation solutions. Furthermore, the agent controllers are executed over each path to assess the performance in following the desired path and recreating the optimized control signal, tied directly to the optimization metric defined for the problem.

The root mean square error (RMSE) of an RNN's state output ($\sigma(t_k)$) in closed-loop (CL) form against the training and validation data is used in assessing the overall performance of both networks. RMSE values provide an average on the difference between the network outputs and the optimal outputs. The closer an RNN's RMSE value is to zero, the more accurate its ability is to track the desired optimal solution. Table 1 provides a comprehensive overview of both the centralized and decentralized RNN RMSE performances on the training and validation sets. Note that the RMSE value of the decentralized RNN is calculated across all agents.

Table 1: RMSE path values of centralized and decentralized RNNs in closed-loop (CL) execution over both training and validation data sets. RMSE values are provided in the units used for the property stated. The closer an RNN's RMSE value is to zero, the more accurate its ability is to track the desired optimal solution. Comparatively, the decentralized RNN outperformed that of the centralized RNN.

	Centralized RNN CL output	Decentralized RNN CL output
Training position (m) RMSE	0.520	0.310
Training velocity (m/s) RMSE	0.384	0.278
Validation position (m) RMSE	0.694	0.557
Validation velocity (m/s) RMSE	0.537	0.491

Fig. 2 and Fig. 3 provide training and validation examples (respectively) of the CL RNN outputs as compared to the optimal paths. When examining both the RMSE values and example plots, it becomes obvious that the decentralized RNN outperforms that of the centralized RNN. This is observable in Fig. 2 and Fig. 3 in not only the greater drift present in the CL paths of the centralized controller but also the unaligned nodes of the produced path (representing evenly space points in time).

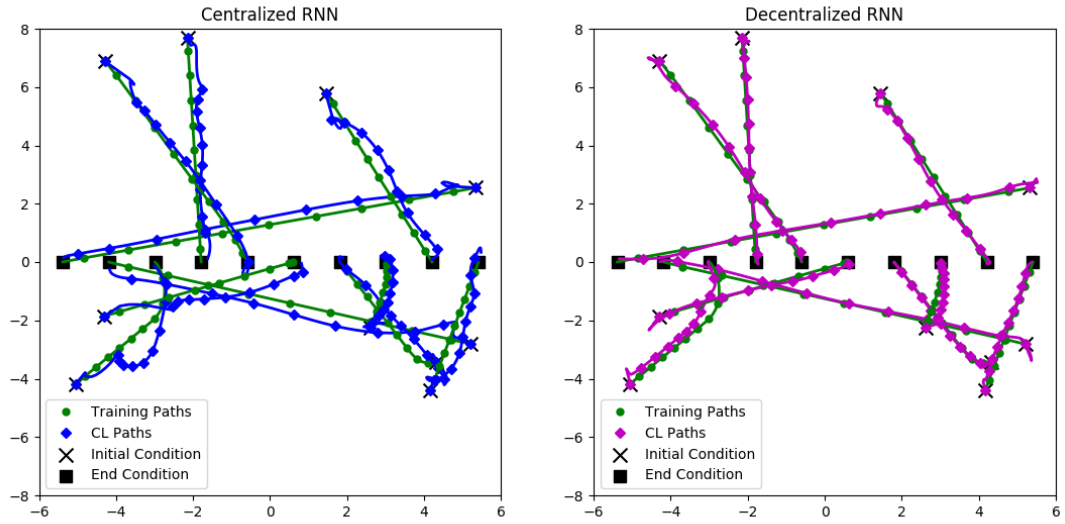

Figure 2: Training example of the centralized and decentralized closed-loop (CL) state outputs compared against the optimized path. Nodes are added at equal intervals (2 secs) to represent fixed-interval points in time. Diamond nodes represent CL paths and circle nodes represent the optimized path. X's represent the initial conditions of all agents, and squares represent the end conditions. Generally, the decentralized RNN produces more accurate paths than the centralized version, while both consistently end at the desired final condition.

Interestingly, though, while the decentralized RNN may outperform that of the centralized version in the overall path accuracy, both sets were consistent in ending at the desired final locations within the execution time of 20 seconds. On the flip side, while the whole-path reinforcement learning scheme does well in aligning the state paths overall, it did tend to produce initial path sequences that diverged before aligning back with the optimized path. This notion along with the networks' ability to consistently hit the desired final location indicate that the network is able to generalize the optimization problem through time and needs further training improvements to increase the CL path accuracy.

The validation sets for both networks showcases the ability of them to generalize over new agent configurations. When comparing the decentralized RMSE values to the centralized RMSE values on the validation set, though, we observe a less obvious improvement. This indicates that both RNN setups may be generalizing the solution space in a more similar manner, one that may become more apparent with a larger data set.

Controller executions were performed over all closed-loop RNN paths for comparison against the optimal state and control signals. Table 2 displays the RMSE executed control (CTRL) values of the resulting state paths, control signals, and evaluation of the optimization metric.

Fig. 4 and Fig. 6 provide training and validation examples of the resulting state paths from the controller execution. Fig. 5 and Fig. 7 provide the controller signals association with the training and validation examples of Fig. 4 and Fig. 6. The paths followed by all agents in the controller executed form tend to follow that of the closed-loop paths produced by the RNNs themselves, resulting in the same comparisons between the centralized and decentralized RNNs. Greater RMSE values are present in the CTRL cases as compared to the CL cases. This appears influenced by the greater initial lag introduced by the greater initial errors present within the CL paths. The path-tracking controller of an agent spends more time around the initial condition before being yanked along the path, resulting in a greater general error between the CTRL path and the optimized path.

Unfortunately, this translates to less than favorable control signals observed in Fig. 5 and Fig. 7 and the produced RMSE values in Table 2. While the control signals trend the same path, the resulting integration of the optimization metric (i.e. the control norm) results in more excessive mean errors when compared to the optimal solutions. Observable in both graphs is the common control spike present around the 0.1 second mark. This is tied to the higher error in the initial

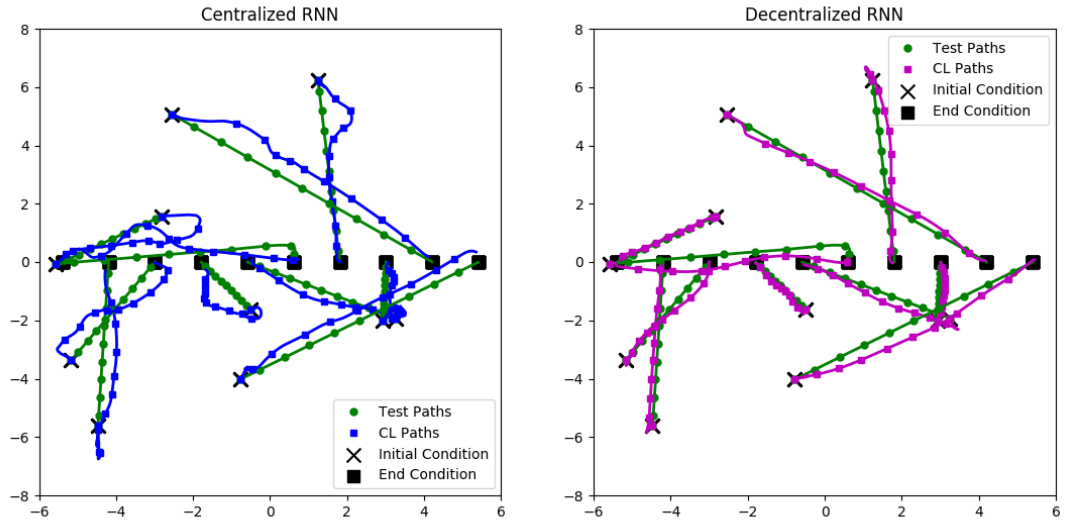

Figure 3: Validation example of the centralized and decentralized closed-loop (CL) state outputs compared against the optimized path. Nodes are added at equal intervals (2 secs) to represent fixed-interval points in time. Diamond nodes represent CL paths and circle nodes represent the optimized path. X's represent the initial conditions of all agents, and squares represent the end conditions. Generally, the decentralized RNN produces more accurate paths than the centralized version, while both consistently end at the desired final condition.

Table 2: RMSE path and control values of executed control (CTRL) over the centralized and decentralized RNNs for both training and validation data sets. RMSE values are provided in the units used for the property stated. The closer an RNN's RMSE value is to zero, the more accurate its ability is to track the desired optimal solution. Comparatively, the decentralized RNN outperformed that of the centralized RNN.

	Centralized RNN CTRL output	Decentralized RNN CTRL output
Training position (m) RMSE	0.777	0.637
Training velocity (m/s) RMSE	0.577	0.554
Training control (N) RMSE	0.862	0.819
Training integrated control (Ave. $\lvert\%$ Error$\rvert$)	61.3	61.4
Validation position (m) RMSE	0.912	0.760
Validation velocity (m/s) RMSE	0.587	0.567
Validation control (N) RMSE	0.883	0.850
Validation integrated control (Ave. $\lvert\%$ Error$\rvert$)	62.3	62.4

closed-loop state output of the RNNs observed in the closed-loop graphs. With better training, this initial error can be reduced, producing smoother closed-loop state paths and better resulting control trends.

7. CONCLUSIONS AND FUTURE WORK

This paper presented a methodology for formulating path planning problems as nonlinear programming problems, from which solutions are used as training data in a recurrent neural network. The closed-loop path generated by the neural network under a given environment is used for a path-tracking controller, the aim of which is to trend the original optimized control signal. The formulation is presented with respect to multi-agent systems, and a simple 10 agent synchronized point-to-point problem with collision avoidance is explored. The results showcase the performance differences between

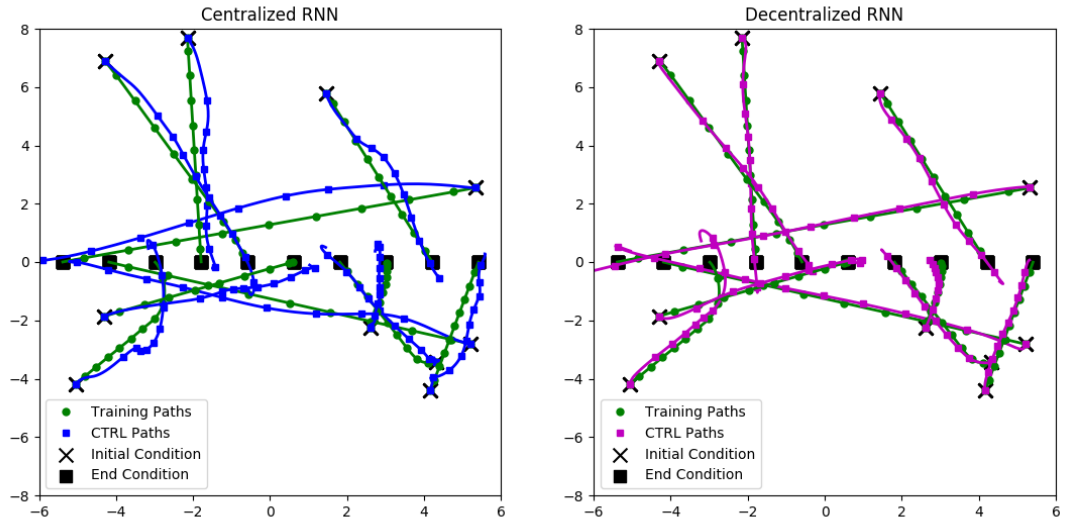

Figure 4: Training example of the centralized and decentralized executed control (CTRL) state outputs compared against the optimized path. Nodes are added at equal intervals (2 secs) to represent fixed-interval points in time. Diamond nodes represent CTRL paths and circle nodes represent the optimized path. X's represent the initial conditions of all agents, and squares represent the end conditions. Generally, the decentralized RNN produces more accurate paths than the centralized version, while both consistently end at the desired final condition.

the favored decentralized RNN and the centralized RNN (of which the decentralized version only made use of initial global information). The results indicate that improved training can increase the accuracy of the generated CL paths, enough that path tracking control will produce the desired control signal trends.

Future work should explore the results in the context of a much larger training data set to observe if the decentralized approach is still favorable, or if validation trends show that both approaches will perform similarly. Other applications and agent dynamics should also be explored, including greater numbers of agents to assess the scalability of both the centralized and decentralized approaches. Results shown in Ref. 10 showcase that at least decentralized neural networks can learn simpler swarm protocols with more complex dynamics for up to 100 agents. The framework of this paper could be extended to examine optimized paths derived for an individual agent with constraints defined for neighboring agents. This would enable easier obtainment of larger sets of optimization data for training, and the result would be a much more scalable (and solely decentralized) RNN path planner for each agent.

REFERENCES

[1] Kamil, F. and W Zulkifli N, K., "A review on motion planning and obstacle avoidance approaches in dynamic environments," Advances in Robotics & Automation **04** (01 2015).

[2] Valero-Gomez, A., Gomez, J. V., Garrido, S., and Moreno, L., "The path to efficiency: Fast marching method for safer, more efficient mobile robot trajectories," IEEE Robotics Automation Magazine **20**, 111–120 (Dec 2013).

[3] Youakim, D. and Ridao, P., "Motion planning survey for autonomous mobile manipulators underwater manipulator case study," Robotics and Autonomous Systems **107**, 20 – 44 (2018).

[4] Mohanan, M. and Salgoankar, A., "A survey of robotic motion planning in dynamic environments," Robotics and Autonomous Systems **100**, 171 – 185 (2018).

[5] Wang, K.-H. C. and Botea, A., "Mapp: a scalable multi-agent path planning algorithm with tractability and completeness guarantees," J. Artif. Intell. Res. **42**, 55–90 (2011).

[6] Bhattacharya, S., Likhachev, M., and Kumar, V., "Multi-agent path planning with multiple tasks and distance constraints," in [2010 IEEE International Conference on Robotics and Automation], 953–959 (May 2010).

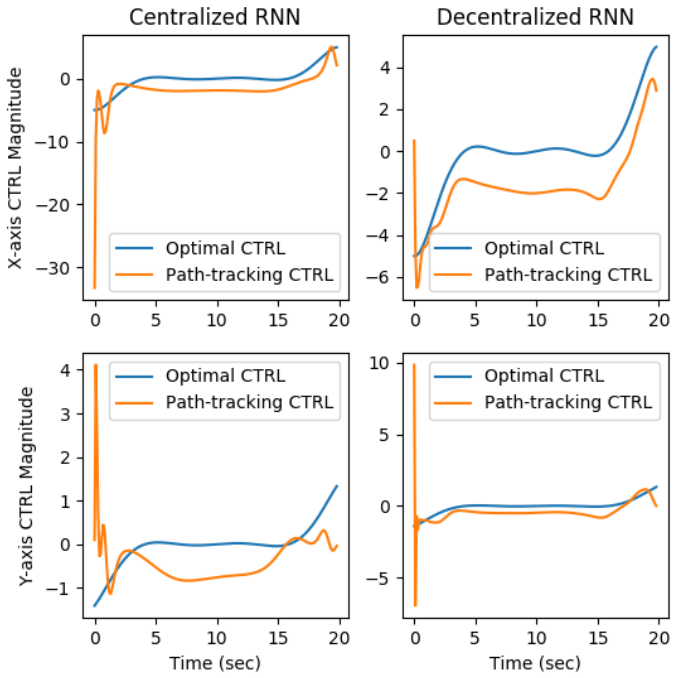

Figure 5: Control signal outputs for agent 1 when following state paths produced by the centralized and decentralized RNNs on the training example. In the ideal performance, the path-tracking control trends that of the optimal control. Initial errors in the CL paths produce the initial spikes in the control signals.

[7] Morgan, D., Chung, S.-J., and Hadaegh, F., "Model predictive control of swarms of spacecraft using sequential convex programming," Journal of Guidance, Control, and Dynamics **37**, 1–16 (04 2014).

[8] Jetchev, N. and Toussaint, M., "Fast motion planning from experience: trajectory prediction for speeding up movement generation," Autonomous Robots **34**, 111–127 (Jan 2013).

[9] mei Zhang, H. and long Li, M., "Rapid path planning algorithm for mobile robot in dynamic environment," Advances in Mechanical Engineering **9**(12), 1687814017747400 (2017).

[10] Gupta, J. K., Egorov, M., and Kochenderfer, M., "Cooperative multi-agent control using deep reinforcement learning," in [Autonomous Agents and Multiagent Systems], Sukthankar, G. and Rodriguez-Aguilar, J. A., eds., 66–83, Springer International Publishing, Cham (2017).

[11] Leofante, F., Narodytska, N., Pulina, L., and Tacchella, A., "Automated verification of neural networks: Advances, challenges and perspectives," CoRR **abs/1805.09938** (2018).

[12] Bassil, Y., "Neural network model for path-planning of robotic rover systems," CoRR **abs/1204.0183** (2012).

[13] Kumar Singh, M. and Parhi, D., "Intelligent neuro-controller for navigation of mobile robot," Proceedings of the International Conference on Advances in Computing, Communication and Control, ICAC3'09 (01 2009).

[14] Al-Sagban, M. and Dhaouadi, R., "Neural-based navigation of a differential-drive mobile robot," in [2012 12th International Conference on Control Automation Robotics Vision (ICARCV)], 353–358 (Dec 2012).

[15] Chen, Y. and Chiu, W., "Optimal robot path planning system by using a neural network-based approach," in [2015 International Automatic Control Conference (CACS)], 85–90 (Nov 2015).

[16] Roth, U., Walker, M., Hilmann, A., and Klar, H., "Dynamic path planning with spiking neural networks," in [Biological and Artificial Computation: From Neuroscience to Technology], Mira, J., Moreno-Díaz, R., and Cabestany, J., eds., 1355–1363, Springer Berlin Heidelberg, Berlin, Heidelberg (1997).

[17] Schilling, F., Lecoeur, J., Schiano, F., and Floreano, D., "Learning vision-based cohesive flight in drone swarms," CoRR **abs/1809.00543** (2018).

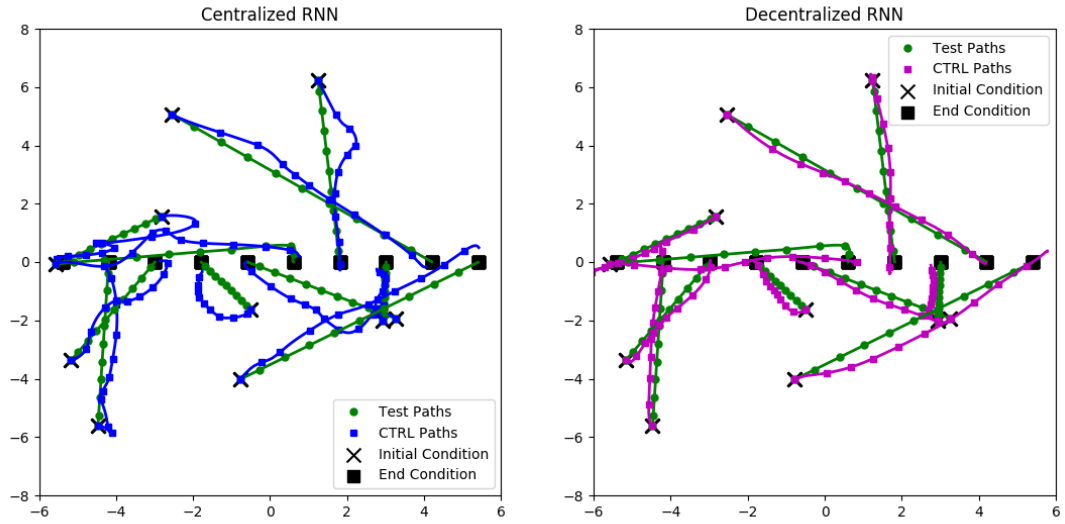

Figure 6: Validation example of the centralized and decentralized executed control (CTRL) state outputs compared against the optimized path. Nodes are added at equal intervals (2 secs) to represent fixed-interval points in time. Diamond nodes represent CTRL paths and circle nodes represent the optimized path. X's represent the initial conditions of all agents, and squares represent the end conditions. Generally, the decentralized RNN produces more accurate paths than the centralized version, while both consistently end at the desired final condition.

[18] Hüttenrauch, M., Sosic, A., and Neumann, G., "Deep reinforcement learning for swarm systems," CoRR **abs/1807.06613** (2018).

[19] Qureshi, A. H., Bency, M. J., and Yip, M. C., "Motion planning networks," CoRR **abs/1806.05767** (2018).

[20] Herber, D. R., "Basic implementation of multiple-interval pseudospectral methods to solve optimal control problems," tech. rep., UIUC-ESDL-2015-01 (June 2015).

[21] Fahroo, F. and Ross, I. M., "Direct trajectory optimization by a chebyshev pseudospectral method," Journal of Guidance, Control, and Dynamics **25**, 160–166 (Jan 2002).

[22] Löber, J., "Optimal trajectory tracking," arXiv e-prints , arXiv:1601.03249 (Dec 2015).

[23] Gill, P. E., Murray, W., and Saunders, M. A., "Snopt: An sqp algorithm for large-scale constrained optimization," SIAM Rev. **47**, 99–131 (Jan. 2005).

[24] Chollet, F. et al., "Keras." https://github.com/fchollet/keras (2015).

[25] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X., "TensorFlow: Large-scale machine learning on heterogeneous systems," (2015). Software available from tensorflow.org.

[26] Dozat, T., "Incorporating nesterov momentum into adam," (2015).

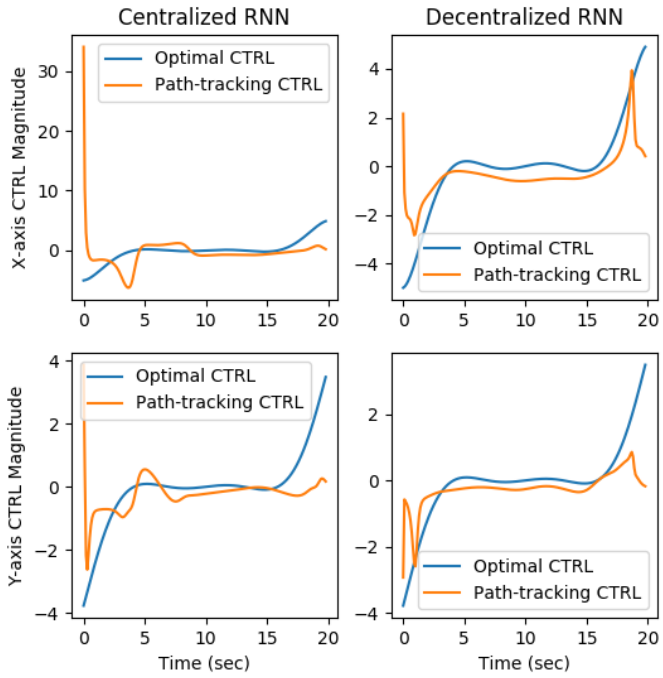

Figure 7: Control signal outputs for agent 1 when following state paths produced by the centralized and decentralized RNNs on the validation example. In the ideal performance, the path-tracking control trends that of the optimal control. Initial errors in the CL paths produce the initial spikes in the control signals.

APPENDIX A. DISCRETE FORMULATION FOR NONLINEAR PROGRAMMING

Under the discretization of the state path as a Chebyshev polynomial and presented in,[20] the optimization problem is reformulated as the following:

$$\underset{U_d, X_d, t_{N_t}}{\text{minimize}} \quad \sum_{k=0}^{N_t} \left(w_k \mathscr{W}(\tau_k, \mathbf{x}(\tau_k), \mathbf{u}(\tau_k))\right) + \mathscr{L}(\mathbf{x}(0), t_{N_t}, \mathbf{x}(\tau_{N_t})), \tag{14a}$$

$$\text{subject to} \quad DX_d - F = 0, \tag{14b}$$

$$\mathbf{x}(0) = \mathbf{x}_0, \tag{14c}$$

$$\mathbf{x}(\tau_{N_t}) = \mathbf{x}_f, \tag{14d}$$

$$C(X_d, U_d, \mathbf{P}) \leq 0. \tag{14e}$$

In the above equations, $N_t + 1$ is the number of discrete nodes. $X_d \in \mathbb{R}^{(N_t+1)\times N}$, $U_d \in \mathbb{R}^{(N_t+1)\times M}$, $F \in \mathbb{R}^{(N_t+1)\times N}$ and $C \in \mathbb{R}^{(N_t+1)\times Q}$ are matrix representations of the discrete nodes of the Chebyshev polynomial and the corresponding state

derivative and constraint evaluations at each, explicitly written as:

$$X_d = \begin{bmatrix} x_1(0) & x_2(0) & \dots & x_N(0) \\ x_1(\tau_1) & x_2(\tau_1) & \dots & x_N(\tau_1) \\ \vdots & \vdots & \ddots & \vdots \\ x_1(\tau_{N_t}) & x_2(\tau_{N_t}) & \dots & x_N(\tau_{N_t}) \end{bmatrix}, \tag{15a}$$

$$U_d = \begin{bmatrix} u_1(0) & u_2(0) & \dots & u_M(0) \\ u_1(\tau_1) & u_2(\tau_1) & \dots & u_M(\tau_1) \\ \vdots & \vdots & \ddots & \vdots \\ u_1(\tau_{N_t}) & u_2(\tau_{N_t}) & \dots & u_M(\tau_{N_t}) \end{bmatrix}, \tag{15b}$$

$$F = \frac{t_{N_t}}{2} \begin{bmatrix} f_1(\mathbf{x}(0),\mathbf{u}(0)) & \dots & f_N(\mathbf{x}(0),\mathbf{u}(0)) \\ f_1(\mathbf{x}(\tau_1),\mathbf{u}(\tau_1)) & \dots & f_N(\mathbf{x}(\tau_1),\mathbf{u}(\tau_1)) \\ \vdots & \ddots & \vdots \\ f_1(\mathbf{x}(\tau_{N_t}),\mathbf{u}(\tau_{N_t})) & \dots & f_N(\mathbf{x}(\tau_{N_t}),\mathbf{u}(\tau_{N_t})) \end{bmatrix}, \tag{15c}$$

$$C = \begin{bmatrix} C_1(\mathbf{x}(0),\mathbf{u}(0),\mathbf{P}) & \dots & C_Q(\mathbf{x}(0),\mathbf{u}(0),\mathbf{P}) \\ C_1(\mathbf{x}(\tau_1),\mathbf{u}(\tau_1),\mathbf{P}) & \dots & C_Q(\mathbf{x}(\tau_1),\mathbf{u}(\tau_1),\mathbf{P}) \\ \vdots & \ddots & \vdots \\ C_1(\mathbf{x}(\tau_{N_t}),\mathbf{u}(\tau_{N_t}),\mathbf{P}) & \dots & C_Q(\mathbf{x}(\tau_{N_t}),\mathbf{u}(\tau_{N_t}),\mathbf{P}) \end{bmatrix}, \tag{15d}$$

where the scaling term $t_{N_t}/2$ in the state derivative is introduced due to state's transformation onto the time domain expressed in Eq. (7). The matrix $D \in \mathbb{R}^{(N_t+1)\times(N_t+1)}$ is the differentiation matrix of the Chebyshev polynomial, explicitly written as,

$$D_{i,k} = \begin{cases} \frac{a_k(-1)^{k+i}}{a_i(\tau_k-\tau_i)} & \text{if } k \neq i \\ -\frac{\tau_k}{2(1-\tau_k^2)} & \text{if } 1 \leq k = i \leq N_t - 1 \\ \frac{2N_t^2+1}{6} & \text{if } k = i = 0 \\ -\frac{2N_t^2+1}{6} & \text{if } k = i = N_t - 1, \end{cases} \tag{16}$$

where $a_{k,i} = 2$ if $k,i = \{0,N_t\}$ and $a_{k,i} = 1$ otherwise. The quadrature weights w_k are used in approximating the integral of a function evaluation on a Chebyshev polynomial, formulated as,

$$w_k = \frac{c_k}{N_t}\left(1 - \sum_{j=1}^{N_t/2} \frac{b_j}{4j^2-1}\cos(2j\tau_k)\right), \tag{17}$$

where $b_j = 2$ if $j = N_t$ and $b_j = 1$ otherwise. The variable $c_k = 1$ if $k = \{0,N_t\}$ and $c_k = 2$ otherwise. Under the presented formulation, the discrete values in U_d, X_d and t_{N_t} make up the free parameters for an NLP program, alongside the provided constraints and optimization function.