2019 IEEE 58th Conference on Decision and Control (CDC)
Palais des Congrès et des Expositions Nice Acropolis
Nice, France, December 11-13, 2019

Expanding Kinodynamic Optimization Solutions with Recurrent Neural Networks and Path-tracking Control

Joshua Shaffer, and Huan Xu, Member, IEEE

***Abstract*— This paper explores a methodology for training recurrent neural networks in replicating path planning solutions from optimization problems. Training data is generated from a kinodynamic rapidly-exploring random tree, from which a recurrent neural network is trained upon to produce the state path through fixed time-step execution. Path-tracking controllers are formulated to follow the path generated by the network alongside the use of local potential functions to mitigate minor constraint violations. The control signal from such a controller should mimic that of the optimized solution. Preliminary results for a 2D dynamics problem with obstacle constraints showcase the ability of this approach to achieve the desired controller execution and resulting state path. We also show that better network training and greater amounts of data lead to an increase in the overall performance.**

I. Introduction

Path planning for general autonomous vehicles encompasses a broad range of methodologies, from which resulting online applications require varying degrees of accuracy and computational speed for the solutions solved. Kinodynamic problems include continuous kinematic and/or dynamic state constraints plus environment-dependent kinematic constraints, such as obstacles. The introduction of these constraint types with optimality conditions typically increases the difficulty in producing quick and feasible solutions [1]. As a result, the fastest path planning methodologies tend to ignore kinematic/dynamic state constraints in favor of finding solutions to satisfy a varied environment [2]. Otherwise, when kinematic/dynamic constraints must be considered, they often are abstracted to simpler, possibly discrete, models, 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 and continuous kinematic/dynamic model. In cases where the solution must utilize complete kinodynamic constraints and optimality conditions in its formulation, trajectory optimization algorithms can find a solution at the cost of much higher computation times [3]. This greatly impacts their online applicability. Furthermore, increases in state space size only exacerbate the issue for all approaches [4], [1], [3]. Because of these issues related to kinodynamic path planning with optimality conditions, an encompassing solution that incorporates complete and continuous kinodynamic constraints and computes as fast as methods that reduce these constraints to simpler or non-existent models is of great interest.

J. Shaffer and H. Xu are with the Department of Aerospace Engineering, University of Maryland, College Park, MD 20740, USA. e-mail: jshaffe9@terpmail.umd.edu, mumu@umd.edu.

Supervised machine learning, using models such as 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. In relation to the controls community, RNNs often find uses as controllers of complex systems, e.g. in highly nonlinear systems [5], [6]. In some cases, the use of learning in controllers is well defined in order to obtain greater understanding of their effects on system stability and convergence [7]. Typically, though, the uses of machine learning in the controls community have been focused towards generating controls directly for state tracking or representing a plant model, and not for providing a path planning solution from which the control solution is built upon. In relation to the machine learning community, control problems explored are often formulated to avoid some of the larger issues that affect training of neural networks, such as large, continuous state and environment spaces with varied and strict constraints [8]. The introduction of these attributes to a problem scenario can often affect training much more severely than the inclusion of nonlinear dynamics and therefore present a significant hurdle for the machine learning community.

For this paper, we present a methodology utilizing an RNN to learn the solution space of a robust, computationally slow path planner that considers kinodynamic constraints and optimality conditions. From here, the RNN provides state paths that follow the desired solutions for new environments, and controllers are formulated to force the state to follow such while satisfying local constraints. The desired result is a control signal and state path that closely resembles the desired solution from the slow-speed path planner while capable of real-time execution at a far greater speed. To assess this methodology, we explore a demonstration example that challenges the merits of this approach. This is a problem scenario in which learning the solution space is difficult for the RNN (e.g. high number of environmental variables per path planning scenario alongside strict constraints [8]) and in which the computation speed of the slow path planning method approaches that of the RNN execution (e.g. the generation method can nearly be used online given simplified assumptions). Evaluating our results from this problem scenario provides a better assessment on the feasibility of this approach with respect to its weakest components.

The rest of the paper follows as such: *Section II* discusses related work to the presented solution, *Section III* introduces the formal problem definition, *Section IV* formulates the entire methodology and approach, *Section V* provides the sample scenario tested alongside provided constraints, *Section VI* examines the results of our implementation, and *Section VII* concludes the paper and presents future work.

II. Related Work

The use of neural networks (NN) 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 NN outputs [9]. With respect to training, works in this area tend to focus on two forms of NN training, supervised learning and reinforcement learning. Supervised learning trains on data sets in which the correct input/output relationships are known while reinforcement learning trains to improve the results of sequential outputs provided by an NN that receives inputs from a simulated environment. For this paper, we do not address reinforcement learning and instead choose to focus on the implementation of models trained using supervised learning. Previous related works have explored various approaches for implementing their supervised learning models in the path planning problems.

[10] implements their NN models to provide initial guesses for a separate online planner. In [10], regression learning (a form of supervised 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. Here, the primary aim is to utilize machine learning in order to speed up the planning procedure for a complex kinodynamic problem. Unfortunately, the optimization planner is still liable to computation hold ups despite strong initial guesses, hampering consistent online performance.

[11], [12], and [13] implement their NN models to provide direct control inputs to a system in order to perform path planning. Each of these examples utilize the models to act as end-to-end controllers, i.e. the models receive the direct environmental input and provide an output control that drives the system to the next time step. These examples demonstrate the ability of a NN to abstract a complex system and provide controls that fulfill common kinodynamic constraints of final configurations and obstacle avoidance, all while executing online. Inversely, the examples either: explored a small set of environments [11]; made use of higher level planners to provide nearby waypoints [12]; or restricted the input to a locally observed space in relation to the path planning domain [13]. Each of these demonstrate the use of a NN in addressing a specific piece of a general path planning formulation and not the entire, general formulation.

[14] and [15] implement their RNN models to provide the entire state path under the whole path planning environment (e.g. desired initial and final configurations and obstacle configurations for the entire domain). Both of these approaches demonstrate the ability of an RNN to generate an entire state path for avoiding obstacles before online use, computing faster than common path planning algorithms. With respect to shortcomings, [14] only generates the shortest euclidean distance while avoiding obstacles, and [15] provides limited

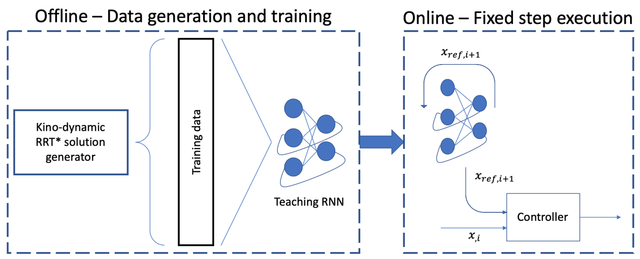
Offline – Data generation and training
Online – Fixed step execution
Kino-dynamic RRT* solution generator
Training data
Teaching RNN
$x_{ref,i+1}$
$x_{ref,i+1}$
Controller
$x_{,i}$

Fig. 1: Methodology overview

demonstrations and did not test the ability of the RNN models to operate in a large set of new obstacle configurations.

Each of the aforementioned approaches demonstrate varying ways of implementing networks in path planning, with the foremost advantage of decreased computation costs in execution. Our work aims to address some of the mentioned shortcomings of previous approaches. Unlike [11], [12], and [13], our model is designed to generate state paths before online execution, on top of which we execute a control solution. This is to separate critical control policies from the unverified results of the NN (e.g. building obstacle constraint satisfaction into the implemented controller design.) Unlike [15] and [11], we assess our trained NN on a large set of unseen environments to better address its viability in general use. Unlike [14], our RNN is trained on path planning problems that make use of dynamic constraints. And unlike [10], our model generates the path planning solution directly, resulting in an essentially fixed computation cost.

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, 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 path planning RNN and controller that operate far faster than the robust, slower optimizer and provide satisfactory solutions.

III. Problem Formulation

A. Path Planning

Our generalized path planning problem is formulated as the optimal control problem presented in the following way:

$$\underset{\mathbf{u}(t),t_f}{\text{minimize}} \quad \int_0^{t_f} \mathscr{W}(t,\mathbf{x}(t),\mathbf{u}(t))dt + \mathscr{L}(\mathbf{x}(0),t_f,\mathbf{x}(t_f)), \tag{1a}$$

$$\text{subject to} \quad \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}(t),\mathbf{u}(t)), \tag{1b}$$

$$\mathbf{x}(0) = \mathbf{x}_0, \tag{1c}$$

$$\mathbf{x}(t_f) = \mathbf{x}_f, \tag{1d}$$

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

Here, $\mathbf{x}(t) \in \mathbb{R}^N$ is the system state, $\mathbf{u}(t) \in \mathbb{R}^M$ is the control signal, and t is time. Eq. (1a) is an optimization metric (consisting of both an integrated scalar function $\mathscr{W}$ and non-integrated scalar function $\mathscr{L}$) for the provided kinodynamic system defined by Eq. (1b), in which $\mathbf{f}(\mathbf{x}(t),\mathbf{u}(t)) \in \mathbb{R}^N$. Eq. (1c) and (1d) represent desired initial and final conditions on the state, respectively, and Eq. (1e) (with $\mathbf{C}(\mathbf{x}(t),\mathbf{u}(t),\mathbf{P}) \in \mathbb{R}^Q$) contains all desired nonlinear constraints on the system states and control signals. The vector $\mathbf{P} \in \mathbb{R}^O$ represents all possible static variables in the constraints. A control solution to the above optimization formulation and its corresponding state path is represented as $\mathbf{G}(t) \in \mathbb{R}^{N+M}$.

B. Network Learning

Provided a domain for the initial and final conditions, $\mathbf{x}_{i,min} \le \mathbf{x}_i \le \mathbf{x}_{i,max}$ and $\mathbf{x}_{f,min} \le \mathbf{x}_f \le \mathbf{x}_{f,max}$, and a constraint domain of $\mathbf{P}_{min} \le \mathbf{P} \le \mathbf{P}_{max}$, individual optimized solutions $\mathbf{G}(t)$ exist as outputs to the optimization solution when using these variables. Provided sets X_i, X_f, and P_{set} of sample points 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 generally as the form,

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

where 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,

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

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

C. Path Execution

Provided a state path $\sigma(t) : \mathbb{T} \longrightarrow \mathbb{R}^N$ generated by closed loop execution of the RNN under set values of $\mathbf{x}_i$, $\mathbf{x}_f$, and $\mathbf{P}$ with $\|\mathbf{x}_i - \sigma(0)\| \le \delta$, where δ is an arbitrarily small number, a controller $\mathbf{u}_e(t,\mathbf{x}(t),\sigma(t_k))$ must be formulated such that the error norm $\|\mathbf{x}(t) - \sigma(t_k)\|$ is minimized while all constraints $\mathbf{C}(\mathbf{x}(t),\mathbf{u}(t),\mathbf{P}) \le \mathbf{0}$ are satisfied. Additionally, the control signal $\mathbf{u}_e$ should mimic that of the true control signal $\mathbf{G}_u(t)$, minimizing the error between the optimized control signal and the executed control signal.

IV. 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 visualized in Fig. 1, and presented in further detail here.

A. Optimal Rapidly-Exploring Random Tree and Nonlinear Programming

We utilize optimal rapidly-exploring random trees (RRT*) in solving individual solutions of the optimization problem presented in Eq. (1a) - (1e) due to its ability to provide global solutions with regards to optimality conditions (a major advantage discussed in [3] and [16]). Specifically, we expand upon the formulation of the kinodynamic RRT* found in [17] through the use of pseudospectral methods.

With respect to the RRT* algorithm, the kinodynamic formulation is built upon the typical RRT* algorithm (as presented in [16]), with a few primary changes. Foremost, the metric for branch creation and pruning is calculated from the optimization formulation in Eq. (1a). Additionally, branches themselves are calculated by solving the optimization problem consisting of Eq. (1a) - (1d), where the initial and final state conditions are assigned from the branches and sampled points. Each candidate branch is checked and discarded if any constraints from Eq. (1e) are violated along the branch path. Due to the probabilistic completeness property of RRT*, as originally proved in [16] and reasoned upon in [17], enough samples will result in the best branch satisfying the desired constraints, dynamic constraints, and optimization metric, providing the global solution to the total optimization problem.

In the original kinodynamic RRT* algorithm presented in [17], the individual branches were formulated as B-splines and optimized as a nonlinear programming problem (NLP) according to the optimization problem. For our work, we utilize the pseudospectral method presented in [18] that makes use of Chebyshev polynomials. This approach allows for a broader class of resulting solutions.

To note, we employ the RRT* and NLP combined approach for two primary reasons. First, NLP solutions from poor initial guesses can take a much greater time (orders of magnitude difference) than those from strong initial guesses. As a result, the incremental approach of an RRT* in finding an initial guess then smoothing said guess (i.e., solve the optimization scheme including Eq. (1e) using NLP and the first RRT* solution as an initial guess) yielded faster results than attempting to solve the optimization problem using NLP and a random initial guess. Second, and more importantly, the RRT* approach enables a broader range of constraint definitions, such as generic logic constraints, due to the binary approach towards deeming if a branch is viable or not. To utilize such constraints, though, the smoothing procedures cannot occur (since all constraints must be formed as inequalities), and the RRT* must operate on a much larger number of sample points. This is a consideration for future work.

B. Recurrent neural network

RNNs, specifically of the Jordan network form, are feedforward neural networks in which the output vector of the net 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 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.

For the purposes of this paper, we represent the RNN function $\Phi\left(\mathbf{x}(t_k), \mathbf{x}_i, \mathbf{x}_f, \mathbf{P}\right)$ as the form,

$$\mathbf{x}(t_k+1) = W_o \sigma_h (W_h \mathbf{x}(t_k) + W_i \mathbf{I} + \mathbf{B}_h) + \mathbf{B}_o, \tag{4}$$

where $W_o \in \mathbb{R}^{N \times HL}$, $W_h \in \mathbb{R}^{HL \times N}$, and $W_i \in \mathbb{R}^{HL \times (N+N+P)}$ are weighting matrices, $\mathbf{B}_o \in \mathbb{R}^{HL}$ and $\mathbf{B}_h \in \mathbb{R}^{N}$ are bias vectors, and $\mathbf{I} \in \mathbb{R}^{N+N+P}$ is the input vector composed of the optimization problem's chosen $\mathbf{x}_i$, $\mathbf{x}_f$, and $\mathbf{P}$ values. The value HL represents the size of the hidden layer. The function,

$$\sigma_h(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \tag{5}$$

σ_h is the chosen activation function of the neurons (standard hyperbolic tangent sigmoid operator), operating over each element of the input vector. This structure is a 2-layer network, in which the output layer utilizes a linear activation function (no operations act on the output vector).

Training the network per sampled points of each solution $\mathbf{G}(t)$ found through the RRT* is performed by minimizing the standard mean square error performance function,

$$MSE = \frac{1}{N_t} \sum_{k=0}^{k=N_t} \left\| \Phi(\mathbf{x}(t_k), \mathbf{x}_i, \mathbf{x}_f, \mathbf{P}) - \mathbf{G}_x(t_k+1) \right\|^2. \tag{6}$$

The actual mechanics of training any type of network is a field that extends far beyond conventional back-propagation techniques, and various methods exist to increase overall performance on training data. For the purposes of this paper, we utilize standard software packages in MATLAB that utilize the Levenberg-Marquardt learning scheme.

C. Path Feedback and Potential Controller

Design of a controller to track the generated path $\sigma(t)$ of the RNN is a problem-specific task tied to the 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 [19]. Fortunately, the generated path, assuming minimal errors produced by the RNN, is already derived from a system using kinodynamic constraints, 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 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,

$$\mathbf{u}_{f,f}(x(t), \sigma(t_k)) = -K_p(\mathbf{x}_p(t) - \sigma_p(t_k)) - K_v(\mathbf{x}_v(t) - \sigma_v(t_k)), \tag{7}$$

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.

While the above controller design can maintain path tracking, no guarantees are provided with respect to constraint satisfaction if the RNN output path fails such. In order to combat this issue for real-time execution, localized potential functions are used about the current position state, derived from [20].

Provided local bounds $\mathbf{x}_{p,min,local}$ and $\mathbf{x}_{p,max,local}$ on the system position state at any given time, potential functions of the form,

$$U(\mathbf{x}_p(t), \mathbf{x}_{p,s}) = \begin{cases} \frac{-c}{N_s \left\| \mathbf{x}_p(t) - \mathbf{x}_{p,s} \right\|} & \text{Eq. (1e)} \nleq 0 \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

are placed at uniform sample points $\mathbf{x}_{p,s}$ of resolution $\mathbf{r} < \left(\mathbf{x}_{p,max,local} - \mathbf{x}_{p,min,local}\right)$ about $\mathbf{x}_p(t)$. In Eq. (8), N_s is a factor to mitigate the scaling issue when using multiple sample points, and c is a gain used for the controller. The condition of Eq. (1e) $\nleq 0$ utilizes $\mathbf{x}_{p,s}$ instead of $\mathbf{x}_p$. The derivative of the repulsive potential functions results in the combined forces shown as,

$$\mathbf{F} = \sum_{s}^{N_s} \frac{-c\left(\mathbf{x}_p(t) - \mathbf{x}_{p,s}\right)}{N_s \left\| \mathbf{x}_p(t) - \mathbf{x}_{p,s} \right\|^2}. \tag{9}$$

The purpose of these potential functions is to provide local constraint satisfaction, not global satisfaction. As a result, egregious errors in the RNN path are not mitigated by the use of these functions. They simply serve as a means of maintaining constraint violations on position in real-time and in a manner that could be employed locally on-board a robotic platform.

As a result of the feedback controller and potential function forces, the executed controller results in the form,

$$\mathbf{u}_e = \mathbf{F}(t_k) + \mathbf{u}_{f,f}(\mathbf{x}(t), \sigma(t_k)), \tag{10}$$

where the term $\mathbf{F}(t_k)$ is calculated by Eq. (9) at each sampled time t_k

V. Implementation

While this methodology is applicable to problems with far more complex dynamic constraints, a 2D, point-to-point problem with obstacle avoidance and linear dynamic constraints is explored to investigate the feasibility of the proposed methodology. The strict constraint of obstacle avoidance and the desired optimization of the state path and control over the entire execution horizon constitute challenging aspects of this problem. In addition to these constraint *types* creating a difficult problem for an RNN to learn, the *number* of obstacle constraints, which contribute large domains and vector sizes to the environmental input variables $\mathbf{x}_i$, $\mathbf{x}_f$, and $\mathbf{P}$ (from Eq. (2)), also present a significant hurdle in the ability of the RNN to generalize learned solutions to entirely new scenarios. Each of these aspects make this apparently simple problem deceptively challenging for an RNN to learn.

The dynamic model utilized for this problem is,

$$\dot{\mathbf{x}} = \mathbf{v} \tag{11}$$

$$\dot{\mathbf{v}} = \mathbf{u}_e m, \tag{12}$$

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 7 rectangular obstacles randomly placed within the state domain, each formulated as,

$$(-6,-8,0.5,0.5) \leq \mathbf{P}_i = (x_p,y_p,h_p,w_p) \leq (6,8,5,5), \tag{13}$$

where x_p and y_p are position coordinates of the rectangle's center and h_p and w_p are heights and widths, respectively. Initial and final state positions (with zero velocity) are sampled randomly within the state domain and outside of obstacles. The optimization function to minimize for a path is,

$$\int_0^{t_f} \|\mathbf{u}_e\| \, dt - t_f. \tag{14}$$

Approximately 5,000 solutions were generated for training, consisting of 10 final positions per 10 initial positions per 50 random object sets. The solver SNOPT [21] was utilized for solving each branch of the kinodynamic RRT* and performing the smoothing procedure for each solution. MATLAB's neural network toolbox was utilized for training the RNN (using 20 neurons in the hidden layer, i.e. $HL = 20$) over the generated data set, sampling at 0.1 seconds (i.e. $t_1 = 0.1, t_2 = 0.2...$). Additionally, the RNN closed-loop execution and state response of the aforementioned controller with gains $c = 25$, $K_p = 10$, and $K_v = 15$ were tested in MATLAB. Results were also generated using just the obstacle avoidance and goal-attracted potential controller from [20] for comparison against the training data solutions and the RNN plus controller solutions. This provides a straightforward, baseline comparison of the methodology solution with a simpler approach. Last, timing comparisons were made between a sped-up version of the solution generation method (RRT* without any NLP optimization) and RNN executions over the longest time frame. This was performed to assess how much faster the RNN approach can execute compared to the original solution generation method.

VI. Results

Three types of comparisons from the methodology's processes are useful in assessing how well the final controller outcome can perform against the optimized training data. First, we examine how well the RNN's state output $\Phi(\mathbf{x}(t_k), \mathbf{x}_i, \mathbf{x}_f, \mathbf{P})$ matches that of the state values $\mathbf{G}_x(t_k+1)$ from all training data points. This approach showcases how well the RNN can mimic desired outputs strictly on the training data, and is referred to as the open-loop (OL) performance of the path generation. Second, we examine the RNN's ability to generate accurate paths when utilizing only initial condition states from each training solution set, $\mathbf{G}_x(1)$. Specifically, for each solution $\mathbf{G}_x$, the initial state is fed into the RNN, from which the RNN's output state $\mathbf{x}(t_k+1)$ is recursively fed back into the RNN's input to generate a path. This approach and its comparison to the solution sets showcases how well the RNN can generate the desired state path provided only the environmental inputs and initial condition, the form in which use of the RNN for path planning would actually be implemented. We label this comparison as the closed-loop (CL) performance of the RNN. Last, we assess the controller's performance in following the state path as a reference. Specifically, provided a CL path generated by the RNN, how well does the controller (starting with the same initial state) maintain path-tracking through the entire path. The resulting state path from the controller is again compared to the solution sets, alongside a comparison of the control signals and resulting optimization function. This approach is referred to as the controlled (CTRL) scenario and represents the desired final outcome of this methodology, since this is what would execute in real-time.

Due to the amount of data explored, we selected a few solution sets for discussion in order to showcase the performance of the CL and CTRL outcomes. Plots of the position paths and control histories are provided for the three samples in Fig. 2 and Fig. 3 respectively, showcasing ideal results, adequate results, and poor results. In the x-y plots, we also included the results of utilizing just the obstacle-avoidance potential controller from [20]. This was to provide comparison against a straightforward solution method, one that executes online like the RNN approach but was not formulated with respect to path optimization.

In the ideal performance case, the CL state output path follows closely that of the training path over the execution duration of the training path. Furthermore, the resulting CTRL control signal trends that of the optimized path as long as the RNN output matches the training data well. The potential controller result nearly matches that of the training path, though, showcasing how the simpler solution can better serve in trivial cases. In the adequate case, similar trends follow, except the CL output path traces the other side of the obstacle before reaching the final point, resulting in larger deviations from the desired control signal in the CTRL case. Again, the potential controller better tracks the training path, but modifications had to be made to the potential radius in order to avoid adding the effect of the near-miss obstacle. This highlights that while again the simple solution could provide a better result, its formulation required modification to handle this obstacle placement. Last, in the poor case, the CL output path does not closely match that of the desired path, but the CTRL path compensates for the RNN's constraint violations while still ending in the vicinity of the desired final location. This satisfaction of the state constraints results in excessive CTRL control spikes, unfortunately, for scenarios in which the RNN greatly violates state constraints. The final outcome of the potential controller resulted in unintended path oscillation due to the obstacle placement. This highlights that further modifications to the potential function design would need to be made in order to better track the desired training path, a common issue found with potential controllers executing in cluttered environments [20].

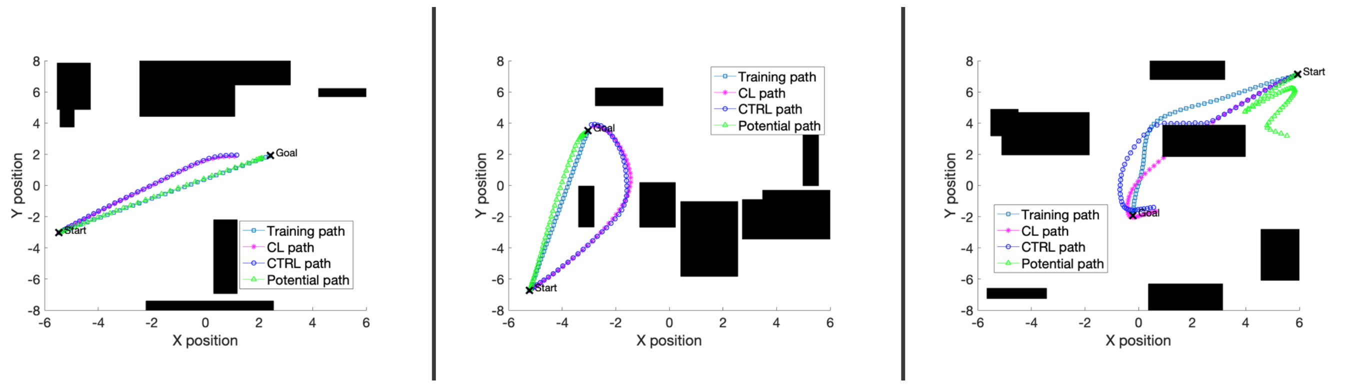

Fig. 2: Ideal (left), adequate (middle), and poor (right) performance performance x-y plots, including solution sets, CL outputs, CTRL outputs, and potential controller comparisons.

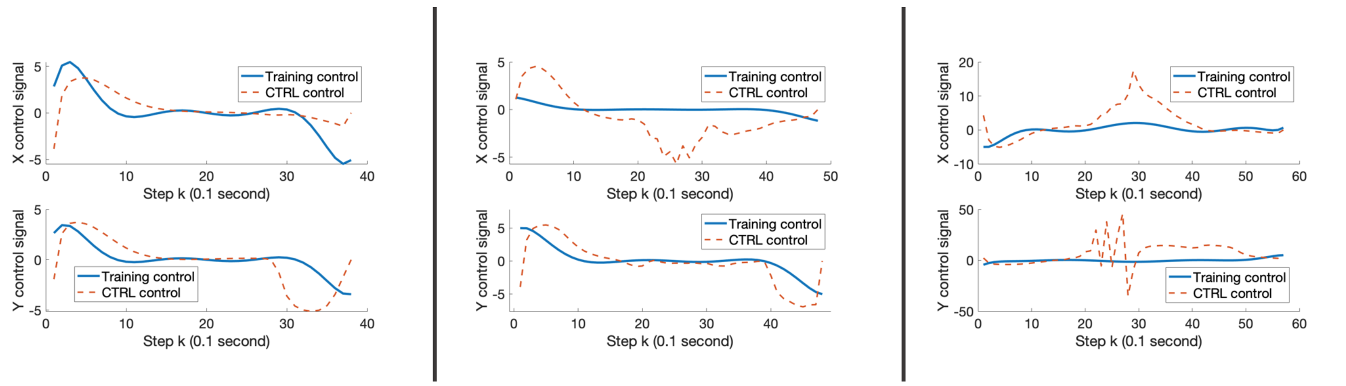

Fig. 3: Ideal (left), adequate (middle), and poor (right) performance control plots, including solution sets and CTRL outputs.

TABLE I: Open-loop (OL), closed-loop (CL), and controller executed (CTRL) RMSE values for 3 training scenarios

Comparison vector	WF	WS	PF
OL Position (m)	3.52e-5	7.84e-5	0.065
OL Velocity (m/s)	0.012	0.011	0.050
CL Position (m)	2.89	5.18	13.07
CL Velocity (m/s)	0.79	2.90	4.70
CTRL Position (m)	6.43	8.54	8.94
CTRL Velocity (m/s)	2.92	3.35	3.60
CTRL Control Signal (N)	17.9	66.29	39.08
CTRL Optimization Function (N-m)	54.2	291.29	137.48

Beyond visual representations, a useful metric to examine across the entire data set is the root mean square error (RMSE) of the state for each comparison approach. This provides a rough understanding of how well each method performs in relation to the training data. Due to the limited data set, though, we provide further comparisons of the OL, CL, and CTRL approaches between three different training scenarios in order to highlight how much training improvements can positively impact the desired results. The three training scenarios consist of a "well-trained" network over the full data set (WF), a "well-trained" network over a subset (1000 solutions) of the data set (WS), and a "poorly-trained" network over the full data set (PF).

From the RMSE values, the training performance in OL outperforms that of CL, which outperforms that of CTRL. More plainly, the open-loop performance of the RNN impacts how well the closed-loop execution performs, which directly affects the controller's ability to match the desired training data results. As observed in the differences between the WF and PF results, greater training performance on the same amount of data resulted in lower RMSE values across all types of comparisons. Additionally, greater amounts of training data also results in lower RMSE values, observed in the differences between the WF and WS results.

It is clear from these results that the RNN closed-loop's ability to match that of the solution set is critical to the success of this methodology and reproduction of desired control signals during controller execution over the path. Fortunately, in the cases of state constraint violations in the path generated by the RNN (observed in the poor case of Fig 2), the executed controller's localized potential functions can advert such violations and still maintain the ability to track the RNN output path towards the final state, a desired result of this methodology. Otherwise, in the ideal performance results of Fig. 2 and Fig. 3, when the RNN closed-loop performs well, the state path of the controller and control signal match the desired solution set from the RRT*, the primary desired result of this methodology.

With regards to computation speed, the execution of the controller is limited solely by the speed of the RNN in generating the execution path, or specifically the proceeding state to follow at each time step. In the case of this example, this is comprised of 2 matrix multiplications, 2 vector additions, and the application of the activation function in the hidden layer. These processes are orders of magnitude faster than the RRT*'s ability to generate a solution in an arbitrary environment. To demonstrate this, executions of the RNN over a 200 time step horizon were performed along with executions of a sped-up version of the RRT* (utilizing no NLP optimization but resulting in suboptimal solutions) sampling 1000 nodes. Both of these tests were performed 100 times to find the average execution times on a 2.5 GHz Macbook Pro. The average execution time of the RRT* was approximately 1.7 seconds, and the average execution time of the RNN was approximately 0.055 seconds. This illustrates that in this scenario, even the suboptimal execution of the RRT* required just over an order of magnitude more time to execute than the RNN over the entire time horizon of the problem.

VII. Conclusions and Future Work

This paper formulates and explores a methodology for training an RNN over optimized solutions generated by a kinodynamic RRT* in order to create state paths in real-time that a path-tracking controller can follow, producing near-optimized solutions for new environments. A small data set for a 2D problem, formulated to challenge the merits of this approach, were used to explore promising preliminary results. We furthermore show that with greater training and larger amounts of training data, better performance in the executed controller (the primary goal of this methodology) can be achieved for real-time implementation, which would execute much faster than the generation method for this straightforward example.

Future work involves multiple avenues. First, larger data sets and greater exploration of training schemes are desired to improve the results for this problem. The inclusion of strict obstacle constraints and large variations in the static environments creates a difficult problem for machine learning to solve, and these results highlight the need for more robust training if an RNN is to generate strong solutions across all possible environments. Second, various environment encodings for the neural network, discussed in [22] and used in [14], can help improve training performance across the board and more robustly represent the environment in training the RNN and for generating the paths. Last, additional complex scenarios should be explored to better assess an RNN's ability to recreate optimized solutions for different dynamics, constraints, optimization metrics, and state space sizes. Additionally, this opens further research into the types of path-tracking controllers that will better recreate the desired control signals from the optimized solutions.

References

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

[2] A. Valero-Gomez, J. V. Gomez, S. Garrido, and L. Moreno, "The path to efficiency: Fast marching method for safer, more efficient mobile robot trajectories," IEEE Robotics Automation Magazine, vol. 20, no. 4, pp. 111–120, Dec 2013.

[3] D. Youakim and P. Ridao, "Motion planning survey for autonomous mobile manipulators underwater manipulator case study," Robotics and Autonomous Systems, vol. 107, pp. 20 – 44, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0921889018300368

[4] M. Mohanan and A. Salgoankar, "A survey of robotic motion planning in dynamic environments," Robotics and Autonomous Systems, vol. 100, pp. 171 – 185, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0921889017300313

[5] M. B. Nasr and M. Chtourou, "Neural network control of nonlinear dynamic systems using hybrid algorithm," Applied Soft Computing, vol. 24, pp. 423 – 431, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1568494614003640

[6] S. L. Brunton and B. R. Noack, "Closed-loop turbulence control: Progress and challenges," Applied Mechanics Reviews, vol. 67, no. 5, p. 050801, Aug. 2015.

[7] H. Singh and N. Sukavanam, "Simulation and stability analysis of neural network based control scheme for switched linear systems," ISA Transactions, vol. 51, no. 1, pp. 105 – 110, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0019057811001005

[8] G. Dulac-Arnold, D. J. Mankowitz, and T. Hester, "Challenges of real-world reinforcement learning," CoRR, vol. abs/1904.12901, 2019. [Online]. Available: http://arxiv.org/abs/1904.12901

[9] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," in Computer Aided Verification, R. Majumdar and V. Kunčak, Eds. Cham: Springer International Publishing, 2017, pp. 3–29.

[10] N. Jetchev and M. Toussaint, "Fast motion planning from experience: Trajectory prediction for speeding up movement generation," Autonomous Robots, vol. 34, no. 1, pp. 111–127, Jan 2013. [Online]. Available: https://doi.org/10.1007/s10514-012-9315-y

[11] Y. Fu, D. Jha, Z. Zhang, Z. Yuan, and A. Ray, "Neural network-based learning from demonstration of an autonomous ground robot," Machines, vol. 7, p. 24, 04 2019.

[12] W. Gao, D. Hsu, W. S. Lee, S. Shen, and K. Subramanian, "Intention-net: Integrating planning and deep learning for goal-directed autonomous navigation," CoRR, vol. abs/1710.05627, 2017. [Online]. Available: http://arxiv.org/abs/1710.05627

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

[14] A. H. Qureshi, A. Simeonov, M. J. Bency, and M. C. Yip, "Motion planning networks," 2019 International Conference on Robotics and Automation (ICRA), pp. 2118–2124, 2018.

[15] M. J. Bency, A. H. Qureshi, and M. C. Yip, "Neural path planning: Fixed time, near-optimal path generation via oracle imitation," CoRR, vol. abs/1904.11102, 2019. [Online]. Available: http://arxiv.org/abs/1904.11102

[16] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," The International Journal of Robotics Research, vol. 30, no. 7, pp. 846–894, 2011. [Online]. Available: https://doi.org/10.1177/0278364911406761

[17] S. Stoneman and R. Lampariello, "Embedding nonlinear optimization in RRT* for optimal kinodynamic planning," in 53rd IEEE Conference on Decision and Control, Dec 2014, pp. 3737–3744.

[18] D. R. Herber, "Basic implementation of multiple-interval pseudospectral methods to solve optimal control problems," UIUC-ESDL-2015-01, Tech. Rep., June 2015.

[19] J. Löber, "Optimal trajectory tracking," arXiv e-prints, p. arXiv:1601.03249, Dec 2015.

[20] G. Fedele, L. D'Alfonso, F. Chiaravalloti, and G. D'Aquila, "Obstacles avoidance based on switching potential functions," Journal of Intelligent & Robotic Systems, vol. 90, no. 3, pp. 387–405, Jun 2018. [Online]. Available: https://doi.org/10.1007/s10846-017-0687-2

[21] P. E. Gill, W. Murray, and M. A. Saunders, "SNOPT: An SQP algorithm for large-scale constrained optimization," SIAM Rev., vol. 47, no. 1, pp. 99–131, Jan. 2005. [Online]. Available: http://dx.doi.org/10.1137/S0036144504446096

[22] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio, "Contractive auto-encoders: Explicit invariance during feature extraction," in ICML, 2011.