

Near-Optimal Finite-Time Feedback Controller Synthesis Using Supervised and Unsupervised Learning

Dongliang Zheng* and Panagiotis Tsiotras[†]
Georgia Institute of Technology, Atlanta, GA, 30332-0150

This paper proposes near-optimal controller synthesis for finite-time optimal control problems using ideas borrowed from the machine learning community. Based on a finite number of pre-computed optimal trajectories, a model of the optimal controller is built using both supervised (regression) and unsupervised (classification) learning. Neural networks are trained to approximate the mapping from state to control, resulting in a state-feedback controller. The resulting neural network controller (NNC) is used for online control prediction and generating near-optimal trajectories to steer the system from any initial state to the goal state. Two extensions are introduced compared with previous related works in the literature. First, the use of clustering divides the offline optimal trajectories into different groups and then builds suitable regression models for each group. Second, the use of time labels as an additional input to the regression model highlights the improvement enabled by the use of time stamped data for finite-time optimal controller synthesis. Multiple simulations, including a 3D quadrotor with 12-dimensional state-space, are provided to evaluate the performance of the proposed neural network controller. The proposed method generates near-optimal trajectories for nontrivial nonlinear systems and is suitable for real-time implementation.

I. Introduction

Computing optimal control actions for nonlinear systems is a computationally demanding task [1, 2]. It typically takes a long time to generate a solution, thus hindering real-time implementation. Often, convergence is not even guaranteed without expert human intervention. The typical problem in optimal control is the generation of a control sequence that steers a dynamical system from an initial state to a goal state while, at the same time, satisfy state and control constraints and also minimize a given objective function. This problem formulation results in a two-point boundary value problem (TPBVP) [3]. Solving TPBVPs for real-time embedded applications, especially in the presence of state and control constraints, is extremely challenging.

Alternative methods for trajectory generation, especially in the robotics community, use motion planners to solve high-dimensional problems with many obstacles. Traditionally, motion planning deals with the generation of feasible paths or trajectories, but recent efforts have focused on the generation of optimal paths. Sampling-based motion planners (SBMP) such as RRT* are able to find optimal paths in very high-dimensional search spaces. However they still need to solve locally the so-called “steering” problem that essentially amounts to a solution of a TPBVP [4–6]. Whenever the initial state or the goal state changes, a new TPBVP needs to be solved. Therefore, it is desirable to have a controller that: a) can steer the system from any state that belongs to a given set to the goal state; b) is a state-feedback controller, thus helping with exogenous disturbances; and c) has low computation load, thus facilitating real-time implementation, but it is still close to the optimal controller in terms of minimizing the cost.

This paper proposes a near-optimal controller synthesis for the finite-time optimal control problem using techniques borrowed from machine learning [7]. Specifically, we use a neural network to build a model of the optimal controller. The neural network is trained using a dataset that contains the state and control input pairs from a pre-computed set of optimal trajectories. The neural network predicts the optimal control action when a new set of boundary conditions is encountered using some form of regression and classification. Regression algorithms use statistical interpolation to generate a function that captures the relation between the states of the system and the controls that steer the system to the goal. Given a new state, the regression model predicts the corresponding control input. The pre-computed optimal trajectories are the extremal trajectories of the problem [8, 9]. The resulting feedback control strategy steers the system from any initial state that belongs to the convex hull of the initial states of the extremal trajectories to the goal state.

*Graduate Student, School of Aerospace Engineering.

[†]David and Andrew Lewis Chair Professor, School of Aerospace Engineering and Institute for Robotics and Intelligent Machines; AIAA Fellow.

Compared to similar approaches that use regression techniques for optimal trajectory generation, the proposed approach introduces an additional clustering step, which is essential in many optimal control problems with discontinuous controls and/or value function. These problems are not amenable to standard regression methods that are based on statistical interpolation – something that seems to have escaped the attention of the machine learning community [8–12]. An additional point that also seems to have been overlooked in these prior works is the importance of incorporating time information for the generation of solutions to *finite-time* optimal control problems. These problems have non-stationary feedback policies and thus are not accurately described by purely state-feedback (stationary) policies.

The paper is organized as follows. First, some prior related work is discussed in Section II. In Section III two motivating examples are introduced in order to elucidate some of the problems with current regression-based optimal control synthesis methods. In Section IV the proposed approach is presented. The training data is generated by repeatedly solving the optimal control problems offline. The details of the neural network controller are given in Section IV.B. Simulation results are given in Section V and, finally, some conclusions are drawn in Section VI.

II. Related Work

Optimal control is a powerful framework to generate feasible and optimal trajectories for systems with complex dynamics subject to constraints. The field of numerical optimal control has experienced enormous advances in recent years and reliable numerical algorithms have been proposed to generate optimal trajectories for a variety of engineering problems [13]. Despite these advances, current numerical optimal control solutions mainly focus on generating open-loop optimal controllers since optimal feedback control design requires the solution of formidable partial differential equations. Furthermore, current trajectory optimizers do not allow real-time computation of optimal trajectories. Differential dynamic programming (DDP) and sequential model predictive control (MPC) [14] have been used to solve optimal control problems in practice, where successive linearization of the plant is used to generate a sequence of linear optimization problems. However, these methods are sensitive to the initial guess, they require to solve the problem iteratively, and are computational demanding for general nonlinear systems.

To bypass the online solution of a nonlinear optimal control problem researchers have turned to alternative methods that generate near-optimal solutions that are computationally more efficient. Reference [15] tries to learn the optimal trajectories for a quadrotor for online control. A neural network is trained using a library of pre-computed optimal trajectories. The neural network builds a mapping from initial conditions to trajectories. This method predicts the whole trajectory using only the initial state and an additional one-step optimization is needed to generate the final trajectory. Reference [16] also uses a neural network to predict the whole trajectory and the discontinuity of the optimal trajectories is considered using a mixture of experts method. In our approach, instead of predicting the whole trajectory, we learn a state-feedback controller. To deal with the discontinuity of the optimal trajectories, we divide the trajectories into different groups and train a neural network for each group. Indirect optimal control and supervised learning are used to learn the solution of the TPBVP in [17]. In that approach, neural networks are trained to predict the initial values of the co-states for the new problems. Based on the necessary conditions of optimality, the optimal trajectory can be computed by predicting the initial values of the costates. The learned component is integrated into a kinodynamic sampling-based motion planner. However, the quality of the resulting trajectories is not analyzed. A similar approach is introduced in [18] for both planetary entry and powered descent guidance problem of a 2-D point mass vehicle. Our method is more related to [8–12, 19–21] where supervised learning is used to build a model of the state feedback controller. Reference [8] introduced a near-optimal feedback strategy for pursuit-evasion games, and [9] proposed a near-optimal feedback controller for aggressive vehicle maneuvers. Both of these works use several pre-computed extremal trajectories to build a regression model of the optimal controller. Gaussian Process Regression (GPR) is used to build regression models in these references. Deep Neural Networks are used to learn the optimal state-feedback controller in [10, 19, 20]. In order to solve the free final time TPBVP, hundreds of thousands of optimal trajectories are pre-computed for this purpose. Similar approaches were introduced in [11] and [12] for the time-optimal planetary landing problem, and in [21] for the trajectory generation problem of a 2-D hypersonic vehicle.

Using pre-computed optimal trajectories for controller synthesis is closely related to what is known in the robotics community as learning-from-demonstration (LfD) and imitation learning. Learning-from-demonstration has been used for robot control [22] and learning driving styles [23]. Imitation learning has been successfully implemented for robot path planning [24] and learning optimal control [19]. In our case, the training data consisting of optimal state-action pairs provide the demonstrations for the optimal controller.

The main contributions in this paper are: First, we add a clustering procedure to divide the offline optimal trajectories

into different groups and build regression models for each group. We show that if the optimal trajectories associated with an initial set contains trajectories from different groups, it is not sufficient to build a single regression model for this initial state set. Thus, a preliminary clustering step is needed for nontrivial nonlinear systems. Second, we use time information as an additional state of the training data to increase the accuracy of the neural network controller. Time information implies non-stationary feedback policies, which is the typical case for finite-time optimal control problems. The proposed method is evaluated using several examples. Simulation results show that the proposed methodology generates near-optimal control inputs in all cases. The computation time of the neural network controller is two orders of magnitude time faster than the use of numerical solvers that solve the associated TPBVP from scratch.

III. Preliminaries

In this section, two examples are introduced to motivate the approach developed in this paper. The first example deals with the finite-time Linear Quadratic Regulator (LQR) problem and aims to show that finite-time optimal control problems lead to non-stationary (time-varying) feedback policies. Hence, regression methods that assume time-invariant feedback control policies are necessarily suboptimal. The second example deals with the nonlinear Reeds-Shepp car and aims to show that regression of optimal trajectories without some prior classification may lead to unsatisfactory results. This is owing to the fact that for this problem, the optimal trajectories can vary significantly even for small changes of the boundary conditions.

A. Motivating Example I: Finite-Time LQR

In order to motivate the proposed developments, let us consider the following finite-time classical LQ problem. The analysis can be found in many textbooks, but it is repeated here to emphasize the non-optimality of stationary control law for finite time problems and the connection between open-loop and closed-loop optimal controller formulation. Consider the time-invariant (LTI) system

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}, \quad (1)$$

where $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{u} \in \mathbb{R}^m$. We wish to find the control law $\mathbf{u}(\cdot)$ that will transfer the system from the initial state $\mathbf{x}_o$ to the final state $\mathbf{x}_f$, while minimizing the LQ cost

$$J(\mathbf{u}(\cdot)) = \frac{1}{2} \int_0^{t_f} [\mathbf{x}^\top(t)\mathbf{Q}\mathbf{x}(t) + \mathbf{u}^\top(t)\mathbf{R}\mathbf{u}(t)] dt, \quad (2)$$

subject to the linear, time-invariant state dynamics (1). It is well known [25] that the *open-loop* optimal control for this problem is given by

$$\mathbf{u}(t) = -\mathbf{R}^{-1}\mathbf{B}^\top \left[\mathbf{P}e^{\tilde{\mathbf{A}}t}(\mathbf{x}_o - \tilde{\mathbf{W}}(-t)\mathbf{z}(0)) + e^{\tilde{\mathbf{A}}^\top(t_f-t)}\tilde{\mathbf{W}}^{-1}(t_f)(e^{\tilde{\mathbf{A}}t_f}\mathbf{x}_o - \mathbf{x}_f) \right]. \quad (3)$$

where $\tilde{\mathbf{A}} = \mathbf{A} - \mathbf{B}\mathbf{R}^{-1}\mathbf{B}^\top\mathbf{P}$, where

$$\mathbf{z}(0) = e^{\tilde{\mathbf{A}}^\top t_f}\tilde{\mathbf{W}}^{-1}(t_f)(e^{\tilde{\mathbf{A}}t_f}\mathbf{x}_o - \mathbf{x}_f), \quad \tilde{\mathbf{W}}(t) = \int_0^t e^{\tilde{\mathbf{A}}s}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^\top e^{\tilde{\mathbf{A}}^\top s} ds, \quad (4)$$

and where $\mathbf{P} \in \mathbb{R}^{n \times n}$ satisfies the algebraic Riccati equation

$$\mathbf{A}^\top\mathbf{P} + \mathbf{P}\mathbf{A} + \mathbf{Q} - \mathbf{P}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^\top\mathbf{P} = 0. \quad (5)$$

To obtain a feedback formulation of the previous open-loop control law, define $\tilde{\mathbf{Z}}(t) = \tilde{\mathbf{W}}(t - t_f)$ and let $\mathbf{x}_o = \mathbf{x}$ to obtain

$$\mathbf{u}(t, \mathbf{x}) = -\mathbf{R}^{-1}\mathbf{B}^\top \left([\mathbf{P} - \tilde{\mathbf{Z}}^{-1}(t)]\mathbf{x} - [\tilde{\mathbf{Z}}^{-1}(t)\tilde{\mathbf{Z}}(t_f + t)e^{-\tilde{\mathbf{A}}^\top t}\mathbf{Z}^{-1}(0)e^{-\tilde{\mathbf{A}}t_f} - e^{-\tilde{\mathbf{A}}^\top t}\tilde{\mathbf{Z}}(0)^{-1}e^{-\tilde{\mathbf{A}}t_f}]\mathbf{x}_f \right). \quad (6)$$

In the special case $\mathbf{x}_f = 0$, the feedback control (6) becomes

$$\mathbf{u}(t, \mathbf{x}) = -\mathbf{R}^{-1}\mathbf{B}^\top(\mathbf{P} - \tilde{\mathbf{W}}^{-1}(t - t_f))\mathbf{x}. \quad (7)$$

It is evident from (6) or (7) that for finite-time optimal control problems, the optimal feedback control is non-stationary, and therefore is an explicit function of *both* the current state *and* the current time. As a result, if one is provided with state/control pairs of optimal trajectories computed off-line (say, from a number of demonstrations), these data are not sufficient to reconstruct the optimal control (and the corresponding optimal trajectory). Instead, the corresponding time stamps when these data were collected are also imperative as an additional piece of information. It follows that any regression method that does not take into consideration this non-stationarity of the data and, instead, tries to reconstruct the optimal trajectories from demonstrations that only consider state/control pairs will give inaccurate results. In Section V we demonstrate, via numerical examples, that this is indeed the case.

B. Motivating Example II: Reeds-Shepp Car

Consider the shortest path problem of the Reeds-Shepp car, a nonholonomic vehicle that can move both forward and backward [26]. The kinematic model of the Reeds-Shepp vehicle is given by

$$\dot{x} = v \cos \beta, \quad \dot{y} = v \sin \beta, \quad \dot{\beta} = u, \quad (8)$$

where x, y are the position of the vehicle and β is the heading angle. The velocity is assumed to be constant and equal to $v = a$ or $v = -a$, where a is a constant scalar. The angular velocity u is bounded, thus constraining the maximum curvature for traceable paths. The car only can turn with less than or equal to the maximum curvature.

The objective is to find the shortest path for the vehicle to reach a goal position. This problem has been treated extensively in [26, 27] where it has been shown that the optimal trajectories for this problem consist of maximum curvature and/or straight line segments. An optimal trajectory is shown in Fig. 1, while Fig. 2 shows additional optimal trajectories. Note in Fig. 2 the distinct shape of the optimal trajectories despite the fact that the initial conditions of the trajectories are the same and the final conditions are close. As will be shown in Section V, directly using all data for regression will result in poor performance. Optimal trajectories with different initial and final conditions that are close can be very different from each other. This motivates the idea that the optimal trajectories should be clustered into different groups prior to regression to improve performance. In this work we first use clustering to divide the offline trajectories into different groups. Within each group we train a neural network to build a model of the corresponding optimal controller.

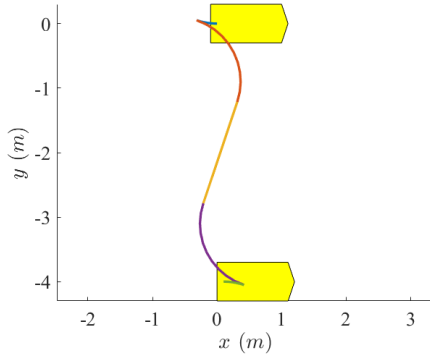


Fig. 1 Time optimal trajectories of a Reeds-Shepp car. Initial state $(x, y, \theta) = (0, 0, 0)$, goal state $(x, y, \theta) = (0.1, -4, 0)$. The optimal trajectory is composed by 5 segments shown by different colors.

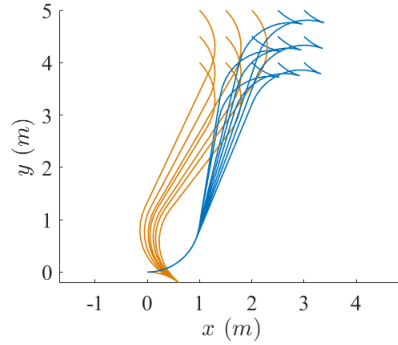


Fig. 2 Optimal trajectories of a Reeds-Shepp car. All trajectories have the same initial state $(x, y, \theta) = (0, 0, 0)$ and final heading angle $\theta = 3\pi/4$, but have different final positions. Note the discontinuity of the optimal trajectories. Although the final positions are close to each other, the optimal trajectories are quite different and can be clustered into two groups.

IV. Outline of Proposed Approach

The proposed method includes two phases: an offline phase and an online phase. Training data generation and Neural Network (NN) training are done offline, which takes of most of the computation. Then, the trained NN is used for online control prediction. Below we provide the details for both these phases.

A. Training Data Generation

Consider the optimal control problem

$$\begin{aligned} \mathbf{u}^* &= \arg \min_{\mathbf{u}} J(\mathbf{x}, \mathbf{u}, t), \\ \text{s.t. } \dot{\mathbf{x}}(t) &= f(\mathbf{x}(t), \mathbf{u}(t), t), \\ \mathbf{x}(t_0) &= \mathbf{x}_0, \quad \mathbf{x}(t_f) = \mathbf{x}_f, \\ \mathbf{x}(t) &\in \mathbf{X}, \quad \mathbf{u}(t) \in \mathbf{U}, \quad t \in [t_0, t_f], \end{aligned} \quad (9)$$

where $\mathbf{X}$ is the obstacle-free state set, $\mathbf{U}$ is the admissible control, and the cost function is given by

$$J(\mathbf{x}, \mathbf{u}, t) = C(\mathbf{x}(t_f), t_f) + \int_{t_0}^{t_f} L(\mathbf{x}(\tau), \mathbf{u}(\tau), \tau) d\tau, \quad (10)$$

where C is the terminal cost function, and L is the running cost along the trajectory.

In this paper, we use GPOPS-II [13] to solve the optimal control problem given in (9) and (10). We generate optimal trajectories that have the same goal state but different initial states. We consider both free final time problems and fixed final time problems where all offline trajectories have the same final time. Each trajectory can be represented by a sequence of control inputs and a sequence of states indexed by time $\{t^i, \mathbf{x}^i, \mathbf{u}^i\}_{i=1}^n$. The control input and state pairs from the optimal trajectories are the training data of the neural networks.

B. Neural Network Controller

Figure 3 shows the pipeline of the proposed near-optimal feedback controller. There are four main steps.

1) Training data generation. As introduced in Section IV.A, the pre-computed training data are the state and control pairs from the optimal trajectories. The optimal trajectories have the same goal state but different initial states. The final time of the trajectories can be either left free or it can be fixed.

2) Clustering of the optimal trajectories computed in step 1. This clustering step divides the training dataset into several distinct groups.

3) Regression Model. For each group of the training data, we train a neural network so that different groups correspond to different initial state sets, and each neural network is responsible for an initial state set.

4) Online prediction. Given a new initial state, the trained NN is used to generate the optimal trajectory to reach the goal. First, we find out in which set the new initial state belongs to. Then we use the corresponding NN to predict the controls. The NNC is a state-feedback controller. Given the current state of the system, the NNC predicts a control action. After applying the control action for a time step, the system arrives at a new state. Then, the Neural Network controller (NNC) predicts a new control action for the new state. By repeating the process, we obtain the corresponding trajectory.

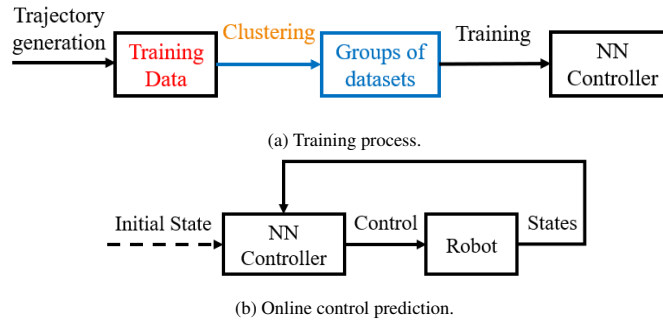


Fig. 3 Pipeline of the proposed optimal feedback generation method.

1. Clustering of Trajectories

Before training the neural network, we cluster the optimal trajectories into several similarity groups. The motivation for clustering stems from the fact that for nontrivial nonlinear systems the optimal trajectories starting from close initial

conditions can be very different from each other (see Fig. 2). Without considering this fact, and without distinguishing between the different groups, the regression-based control may perform poorly.

There are well-developed algorithms for clustering [28–30]. The k-means clustering algorithm [31] is chosen for clustering in this work, and the remaining steps deal with how to choose the data for clustering and choose the number of clusters to divide the data. In our case, the data are the trajectories. Every trajectory is represented by a sequence of data points along the trajectory. Each point of the trajectory is a vector consisting of the state and control at the corresponding time instant. For each trajectory, we concatenate all the points into a long vector. Then the clustering algorithm is applied on those vectors. The k-mean clustering algorithm divides these vectors based on the Euclidean distance metric, and the trajectories are clustered into different groups. The number of clusters/groups for a given dataset is given a prior.

The clustering step divides a large initial state set into several small initial state sets based on the similarity between the trajectories. Without distinguishing trajectories from different groups and naïvely using all the data to build a regression model, the regression-based control may perform poorly. Another benefit that comes with clustering is the reduced computation, both online and offline. If a dataset with N data points is divided equally into s groups, the regression algorithm only need to operate on $N/s \times N/s$ matrices s times, instead of operating on $N \times N$ matrices. This matrix operation difference will make a big difference on large datasets.

2. Time Labels as an Additional Inputs

For the finite-time optimal control problem, the control input may be a function of both the state and time. It is evident from (6) or (7) that for finite-time optimal control problems, the optimal feedback control is non-stationary, and therefore is an explicit function of *both* the current state *and* the current time. In this paper we wish to capture the non-stationarity of *finite-time* optimal control problems when constructing feedback control laws via off-line optimal trajectories.

The trajectory of the state is given by $\{\mathbf{x}^i\}_{i=1}^n$, where n is the number of data points of the trajectory, $\mathbf{x}^i$ is the state at the i^{th} time step and $\mathbf{x}^n$ is the endpoint/goal of the trajectory. Denote t^i the time-to-reach value at $\mathbf{x}^i$, which is the time required to go from $\mathbf{x}^i$ to $\mathbf{x}^n$. For every data point along a trajectory, we compute the corresponding time-to-reach value. The original state $\mathbf{x}^i$ is augmented with its time label t^i . We train the neural network using the augmented states.

For the free final time TPBVP we also train a time-to-reach neural network to predict the time duration to simulate the system. Given the goal state $\mathbf{x}^n$ and the current state $\mathbf{x}^i$ of the system, the time-to-reach NN predicted the time needed to go from $\mathbf{x}^i$ to $\mathbf{x}^n$. Thus the training data of the time-to-reach NN has the form $\{(\mathbf{x}^i, \mathbf{x}^n, t^i)\}$ and the neural network is a mapping from $(\mathbf{x}^i, \mathbf{x}^n)$ to t^i .

V. Numerical Simulations

In this section, we implement the proposed method on five systems: a double integrator which represents a simple linear system, a Reeds-Shepp car, a two-link robot arm which is highly nonlinear, a reusable launch vehicle, and a 3D quadrotor which has a 12-dimensional state space. After the training data is generated and rearranged as discussed in Section IV.B, the data is used for training the neural network controller. In all examples, the neural networks are trained using Matlab's Machine Learning Toolbox. We use the Bayesian Regularization backpropagation algorithm [32] for training in all examples.

A. Double Integrator

We consider the minimum-energy control of a double integrator with state $\mathbf{x}(t) = [x_1(t) \ x_2(t)]^\top$. The final state is fixed at $\mathbf{x}_f = [x_1(t_f) \ x_2(t_f)]^\top = [0 \ 0]^\top$ and t_f is the final time. This problem has an analytic solution given by

$$u(t) = \left(\frac{12x_1(0)}{t_f^3} + \frac{6x_2(0)}{t_f^2} \right) t - \left(\frac{6x_1(0)}{t_f^2} + \frac{4x_2(0)}{t_f} \right), \quad (11)$$

$$u(t, \mathbf{x}) = \frac{6x_1(t)}{(t_f - t)^2} - \frac{4x_2(t)}{t_f - t}, \quad (12)$$

where (11) is the open-loop controller, and (12) is the closed-loop state feedback controller.

The offline trajectories are shown as blue lines in Fig. 4. Each trajectory has 100 data points. The neural network has one hidden layer with 10 hidden units. The control results are also shown in Fig. 4 in orange color. Time label

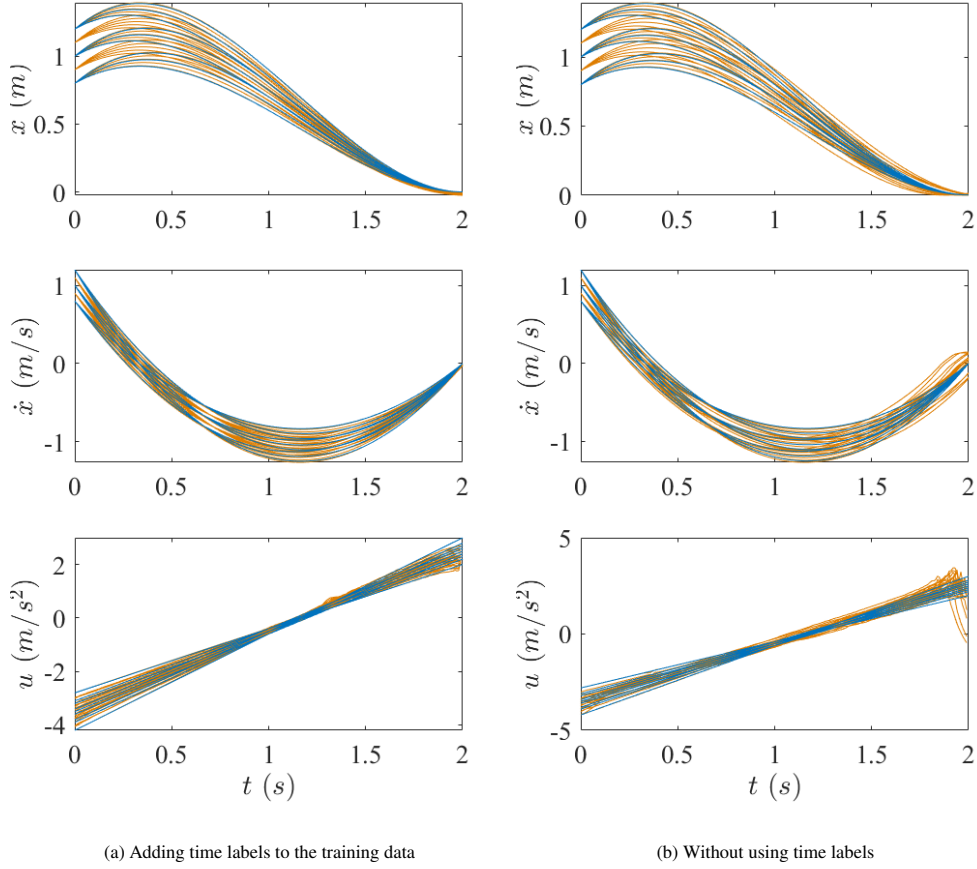


Fig. 4 Control result for a double integrator. The blue lines are offline trajectories used for neural network training. Orange lines are the trajectories generated by the NNC for new initial states.

information is used in Fig. 4(a) while the results without using time label information is given in Fig. 4(b). It is evident that the results shown in Fig. 4(a) are more close to the optimal solutions compared with Fig. 4(b) especially when close to the terminal time.

B. Reeds-Shepp Car

The kinematic model of the Reeds-Shepp car is given in Section III.B. An optimal trajectory is shown in Fig. 1. This trajectory is composed of 5 segments. Define the mode of a segment by $\text{mod} = [\text{mod}_1, \text{mod}_2]^T$, where $\text{mod}_1 \in \{1, -1\}$ and $\text{mod}_2 \in \{1, 0, -1\}$. The label $\text{mod}_1 = 1$ implies that the vehicle moves forward, $\text{mod}_1 = -1$ implies that it moves backward, $\text{mod}_2 = -1$ implies that it turns left with maximum curvature, and $\text{mod}_2 = 0$ implies that the vehicle is travelling straight. The optimal trajectory in Fig. 1, for example, is fully specified by the modes of its five segments

$$\mathbf{M} = \begin{bmatrix} -1 & 1 & 1 & 1 & -1 \\ -1 & -1 & 0 & 1 & 1 \end{bmatrix},$$

where each column of $\mathbf{M}$ corresponds to the mode of a segment, and the lengths of the five segments are given by

$$\mathbf{l} = [0.3231 \quad 1.5708 \quad 1.6577 \quad 1.5708 \quad 0.3231].$$

From these $\mathbf{M}$ and $\mathbf{l}$, we know for instance, that the mode of the first segment is $\text{mod} = [-1, -1]^T$ and its length is 0.3231 m, and similarly for the other segments.

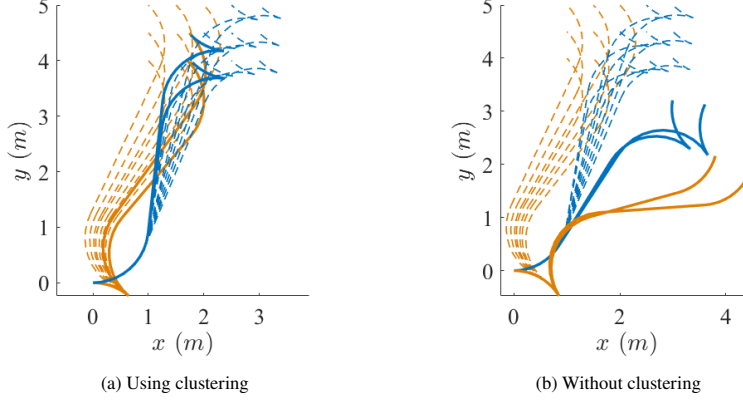


Fig. 5 Control result for a Reeds-Shepp car. Training trajectories belong to two modes shown in dash blue and orange lines respectively. (a) Our method trains two neural network for each mode. The solid blue lines and orange lines are the predicted trajectories from the two NNC respectively. (b) Train one neural network using all offline trajectories. Solid blue lines and orange lines are the predicted trajectories using the predicted segment lengths and the two modes. The predicted lengths are far from the optimal solution for both modes.

In this case, the mode of the optimal trajectories is a natural index to use to separate the trajectories. The mode of a trajectory is given by $\mathbf{M}$. The trajectories used for training are shown in Fig. 2. The goal is to generate optimal trajectories that steer the vehicle to different goal states from the same initial state $(x, y, \theta) = (0, 0, 0)$. The trajectories in Fig. 2 belong to two modes (groups). Our method builds a neural network for each mode (group). The neural network provides a mapping from the final state of the trajectory to the lengths of each segment of that trajectory. Given a new final state, the NNC predicts the corresponding lengths $\mathbf{l}$. Simulation results are shown in Fig. 5(a). For comparison, the results without clustering are shown in Fig. 5(b). It is clear from Fig. 5(b) that the results are far from optimal without the clustering step.

C. Two-link Robot Arm

A two-link robot arm is a highly nonlinear dynamic system. The equations of motion can be found in [33, Ch. 4]. The two-link robot arm has four states: two angles and two angular velocities for the two arms. The offline optimal trajectories are shown in Fig. 6. All trajectories have the same final condition but different initial conditions. In this case, we vary the initial angles while the initial angular velocities are set to zero. By gradually varying the initial condition, we obtain very different trajectories even if their initial conditions are close.

The proposed method first clusters the trajectories into different groups as described in Section IV.B. For the two-link robot arm example, each data point is a six-dimensional vector. For each trajectory, we concatenate all the points into a long vector. Each trajectory has 100 data points, thus the trajectories are represented by vectors of dimension 6×100 . Clustering is performed on those vectors. Figure 7 shows an example of the clustering result.

A neural network is trained for each group. Figure 8 shows the result of the proposed method. The neural network has one hidden layer with 16 hidden units. For each group of offline trajectories, one neural network is trained. Each neural network is responsible for an initial state set, which is a subset of the original problem. The result for one group is shown in Fig. 8. The results for other groups are similar.

For comparison, the results without clustering are given in Fig. 9. The neural network has one hidden layer with 30 hidden units. In Fig. 9, some of the worst trajectories are observed for initial states that are between groups. This result is expectable since the neural network may average the control commands from two diverse groups. Comparing to Fig. 9, the proposed method (Fig. 8) achieves better results.

D. Reusable Launch Vehicle

The Reusable Launch Vehicle Entry problem is taken from [13], [34]. The vehicle has six state variables $(r, \theta, \phi, v, \gamma, \psi)$ corresponding to altitude, longitude, latitude, velocity, flight path angle, and azimuth, respectively, and two control

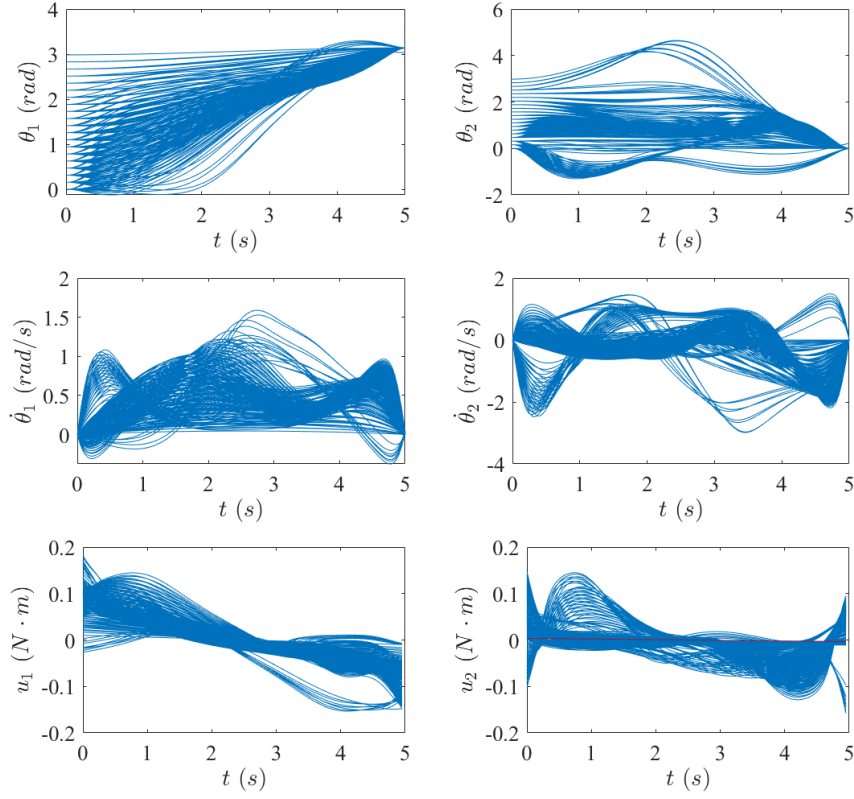


Fig. 6 Optimal trajectories for a two-link robot arm with different initial states, fixed final state, and fixed final time. The robot has four degrees of freedom (two joint angles and two angular velocities). The two torques at the two joints are the control commands.

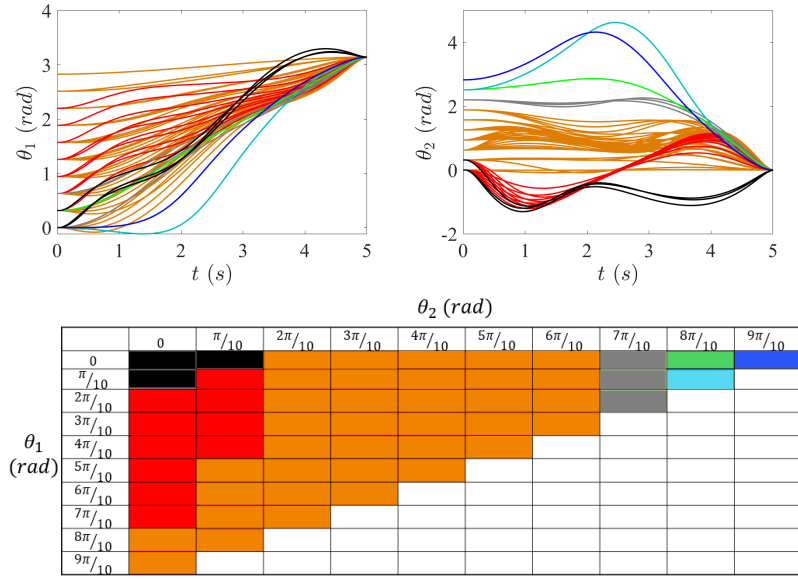


Fig. 7 An example of the clustering result for the two-link robot arm example. 55 trajectories are clustered into 7 groups. Different colors represent different groups.

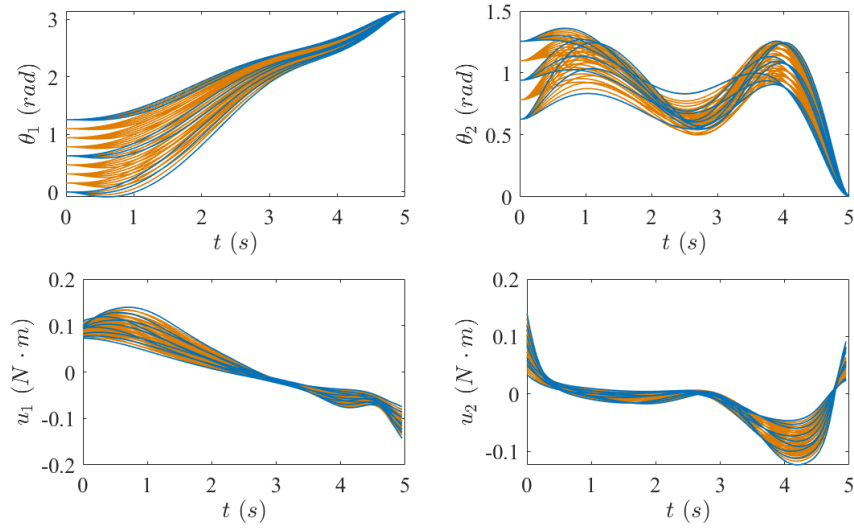


Fig. 8 Results for the two-link robot arm example using time labels and clustering. Trajectories are clustered into different groups. One neural network is trained for each group, and each neural network is responsible for an initial state set. Blue lines are the pre-computed trajectories used for training. Orange lines are the trajectories generated by the controller for new initial states.

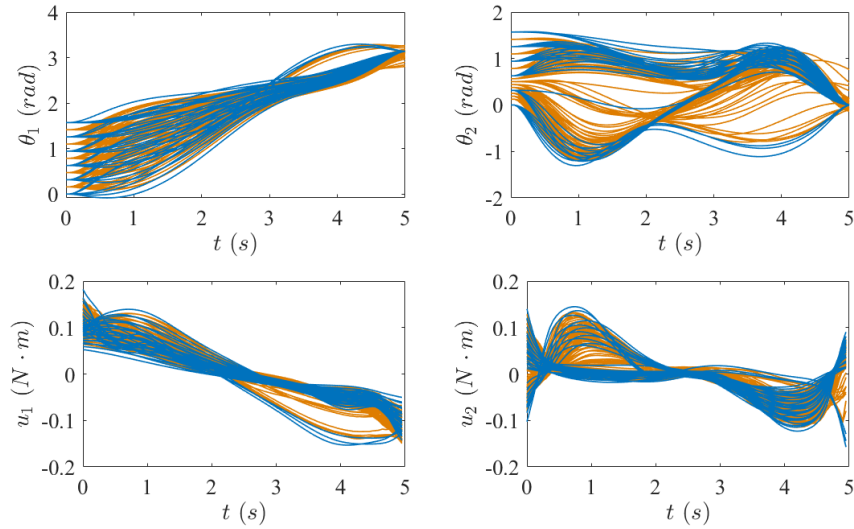


Fig. 9 Results for the two-link robot arm example using time labels but without clustering. Blue lines are the pre-computed trajectories used for training. Orange lines are the trajectories generated by the controller for new initial states. Training data from different groups is used to train one neural network, which results in less optimal trajectories.

inputs (α, σ) corresponding to the angle of attack and bank angle respectively. The dynamics is given by

$$\begin{aligned}\dot{r} &= v \sin \gamma, \\ \dot{\theta} &= \frac{v \cos \gamma \sin \psi}{r \cos \phi}, \\ \dot{\phi} &= \frac{v \cos \gamma \cos \psi}{r}, \\ \dot{v} &= -\frac{D}{m} - g \sin \gamma, \\ \dot{\gamma} &= \frac{L \cos \sigma}{mv} - \left(\frac{g}{v} - \frac{v}{r}\right) \cos \gamma, \\ \dot{\psi} &= \frac{L \sin \sigma}{mv \cos \gamma} + \frac{v \cos \gamma \sin \psi \tan \phi}{r},\end{aligned}$$

where D and L are aerodynamic forces, m is the mass of the vehicle, g is the gravitational acceleration. L is a function of α .

Table 1 Initial states of the offline trajectories of the Reusable Launch Vehicle.

r km	θ deg	ϕ deg
78.248, 80.248	0, 5	0, 5
v km/s	γ deg	ψ deg
7.80288, 7.90288	-1, -2	85, 90

Table 2 Final states of the offline trajectories of the Reusable Launch Vehicle.

r km	θ deg	ϕ deg
24.384	free	free
v km/s	γ deg	ψ deg
0.762	-5	free

The goal is to maximize the crossrange during the atmospheric entry of the vehicle, $\phi(t_f)$. The values of the initial states of the offline trajectories are shown in Table 1. Each state can be either of the two values. Thus, by combining the different initial values of the six states, there are 2^6 different initial conditions. Similar to [13], the final states are shown in Table 2. The final time is fixed to 2,000 sec. A total of 64 optimal trajectories were generated. The neural network has a single hidden layer with 30 hidden units. The results for the new initial states, which have not appeared in the training data, are shown in Fig. 10. In Fig. 10, 96 new initial states are tested, and all trajectories obtained from the controller end up around the final goal state specified in Table 2.

E. 3D Quadrotor

The last example is a 3D quadrotor with a 12 dimensional state sapce. Consider a quadrotor with mass m_q and body frame inertia J . Its dynamics is given by

$$\begin{aligned}\dot{p} &= v, \\ m_q \dot{v} &= m_q g e_3 - R(\phi, \theta, \psi) f e_3, \\ \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} &= \begin{bmatrix} 1 & \sin \phi \tan \theta & \cos \phi \tan \theta \\ 0 & \cos \phi & -\sin \phi \\ 0 & \sin \phi / \cos \theta & \cos \phi / \cos \theta \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix}, \\ J \dot{\omega} &= -\omega \times J \omega + \tau,\end{aligned}\tag{13}$$

where $p = [x, y, z]^T$ is the position vector, $v = [v_x, v_y, v_z]^T$ is the linear velocity, $\omega = [\omega_1, \omega_2, \omega_3]^T$ is the angular in the body fixed frame, R is the rotation matrix between the quadrotor body frame and the inertial frame, $e_3 = [0, 0, 1]^T$. The attitude of the quadrotor is given by the Euler angles ϕ , θ , and ψ denoting, respectively, the roll, pitch, and yaw angle. The state space is $\mathbf{x} = [p^T, \phi, \theta, \psi, v^T, \omega^T]^T \in \mathbb{R}^{12}$, the control space is $\mathbf{u} = [f, \tau^T]^T \in \mathbb{R}^4$.

The parameters of the quadrotor are $m = 2$ kg and $J = \text{diag}\{0.0625, 0.0625, 0.125\}$ kg m²/rad². For this example, 5,794 trajectories with different initial conditions were generated. The initial conditions of the 12 states are uniformly sampled from a set, which is specified in Table 3. The final conditions of the states are set to be zero. Some of the offline trajectories are given in Fig. 11.

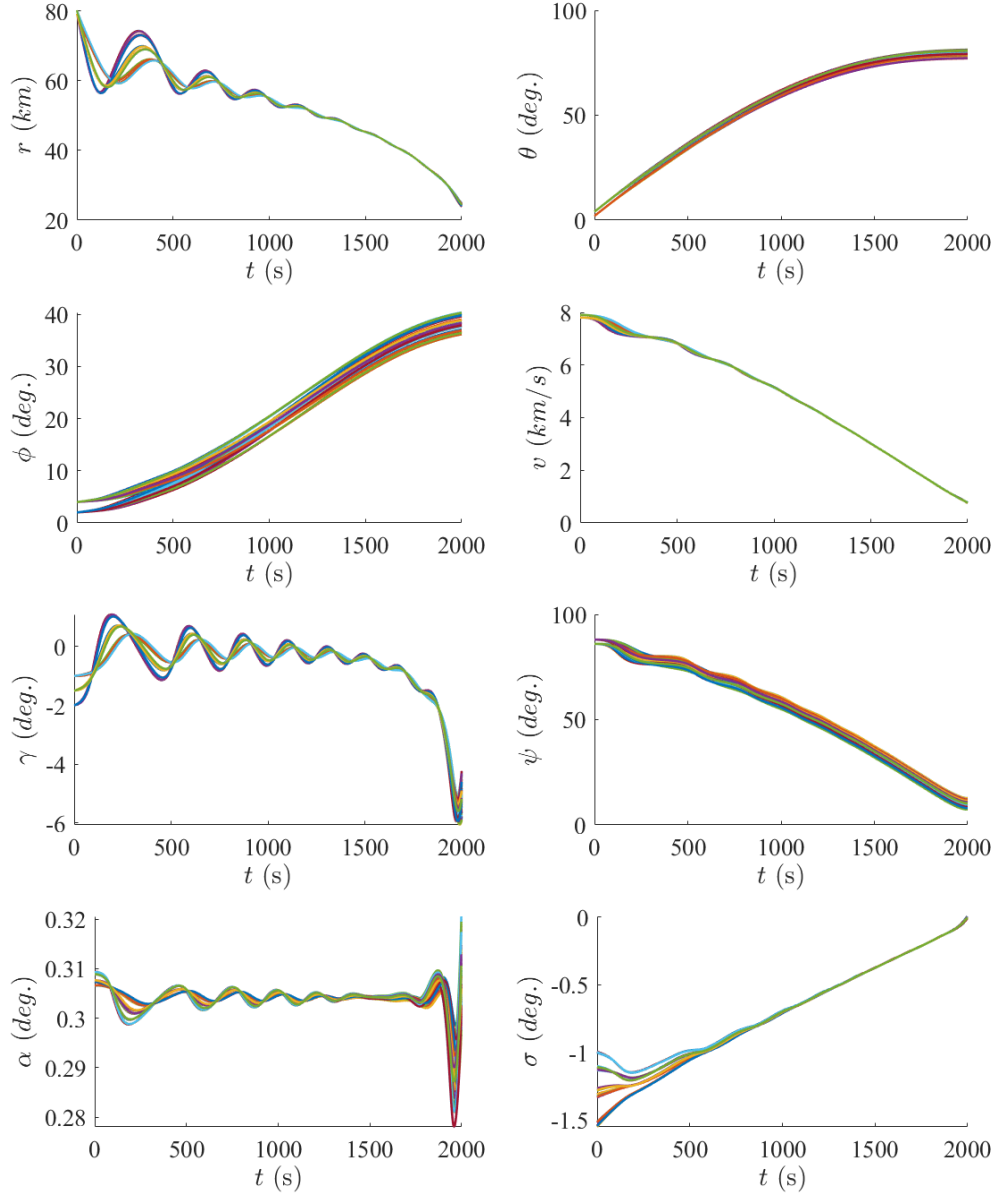


Fig. 10 Control results of Reusable Launch Vehicle using the neural network controller starting from different initial conditions. The two figures at the bottom are the control trajectories of angle of attack and bank angle respectively.

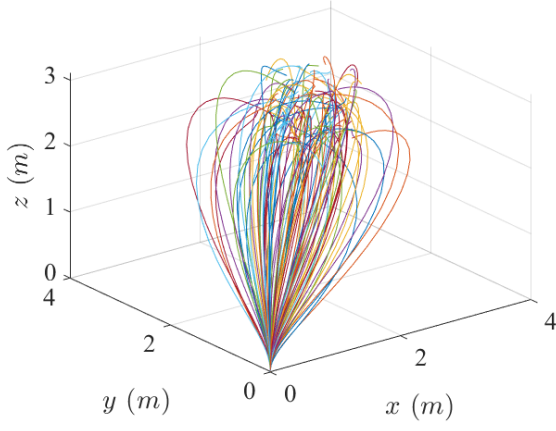


Fig. 11 Offline trajectories correspond to the 3D position of the 3D quadrotor.

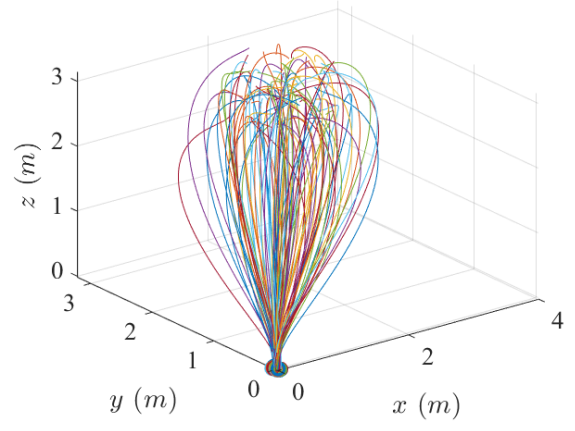


Fig. 12 Trajectories generated by the neural network controller. This figure shows the 3D position of the quadrotor.

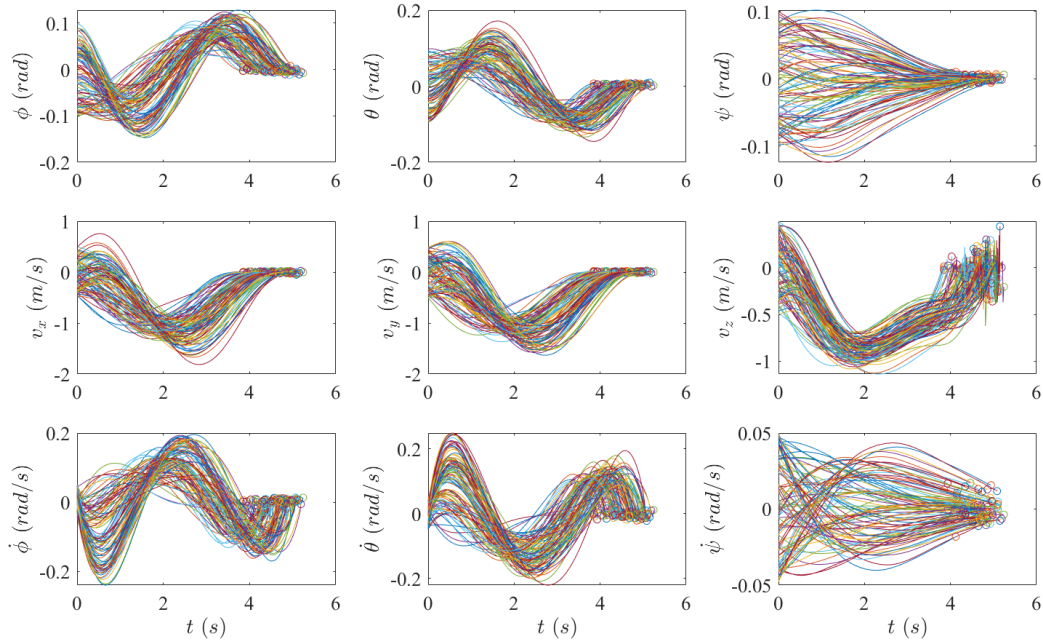


Fig. 13 Trajectories generated by the neural network controller. This figure shows the Euler angles, linear velocities, and angular velocities.

Table 3 Boundary conditions for the 3D quadrotor. Each dimension of the initial state is uniformly sampled from the range specified in this table.

x m	y m	z m
[2, 3]	[2, 3]	[2, 3]
ϕ rad	θ rad	ψ rad
[-0.1, 0.1]	[-0.1, 0.1]	[-0.1, 0.1]
v_x m/s	v_y m/s	v_z m/s
[-0.5, 0.5]	[-0.5, 0.5]	[-0.5, 0.5]
Ω_1 rad/s	Ω_2 rad/s	Ω_3 rad/s
[-0.05, 0.05]	[-0.05, 0.05]	[-0.05, 0.05]

Table 4 MSE of the 12 states at the endpoints of 500 trajectories.

x m	y m	z m
9.4498×10^{-5}	2.2681×10^{-4}	4.3930×10^{-5}
ϕ rad	θ rad	ψ rad
1.7736×10^{-5}	1.7236×10^{-5}	1.6242×10^{-5}
v_x m/s	v_y m/s	v_z m/s
1.9240×10^{-4}	1.3249×10^{-4}	0.0322
Ω_1 rad/s	Ω_2 rad/s	Ω_3 rad/s
8.6194×10^{-5}	1.4074×10^{-4}	3.2316×10^{-5}

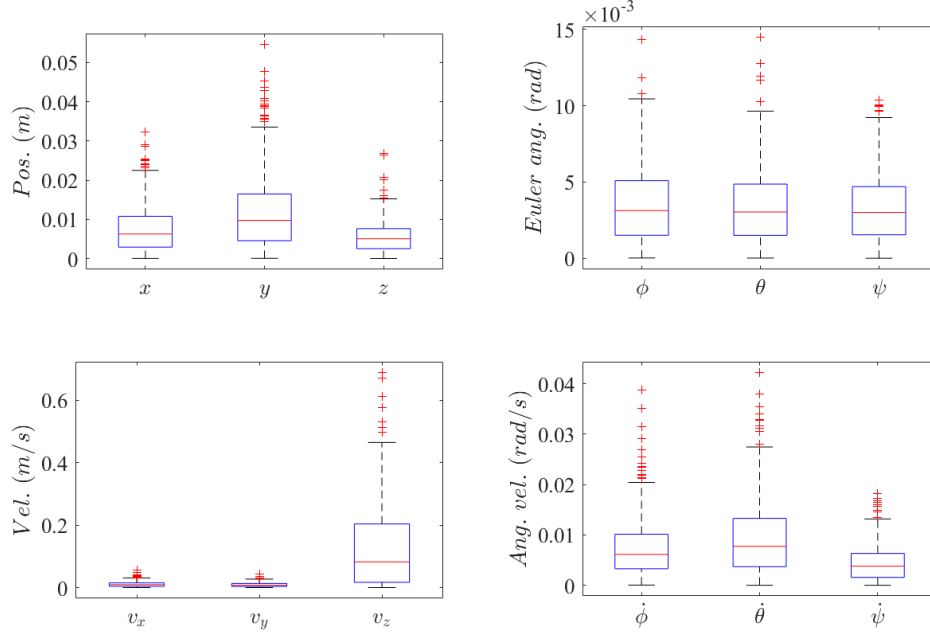


Fig. 14 Boxplots of the endpoints errors of the 3D quadrotor example. 500 trajectories are generated using the NNC. The absolute errors of the 12 states at the endpoints of the trajectories are shown. Using the neural network controller, all 12 states of the quadrotor reach their desired values with small errors.

The NNC has 1 hidden layer with 70 units. The *time-to-reach* neural network has 2 hidden layers, the units in the hidden layers are 20 and 10, respectively. After training the neural network, we randomly sampled new initial conditions to test the NNC. Some example trajectories obtained using the NNC are shown in Figures 12 and 13.

We tested the controller using 500 new randomly sampled initial conditions. Since we are interested in steering the system to a specific goal state, one index of the accuracy of the NNC is the closeness between the endpoints of the newly generated trajectories and their goal states. We computed the error between the endpoint of each trajectory and its goal. The errors of the endpoints of those trajectories are shown in Fig. 14. The MSE of all 12 states are given in Table 4.

One major advantage of the NNC is its real-time implementation potential. We compared the time required to solve a TPBVP using the NNC and GPOPS-II. For 500 repeated trials the time required to solve the TPBVP is shown in Fig. 15. The simulations are performed on a laptop computer with an Intel Core i5-8250U 1.6GHz CPU. When using the NNC, the average time for solving a TPBVP (and obtain the trajectory) is 0.0176 sec, while the average time for GPOPS-II is 4.1811 sec. The proposed approach is more than 200 times faster than GPOPS-II. With most of the computation performed offline, the learning-based approach is computationally efficient, and is suitable for real-time implementation for nonlinear systems.

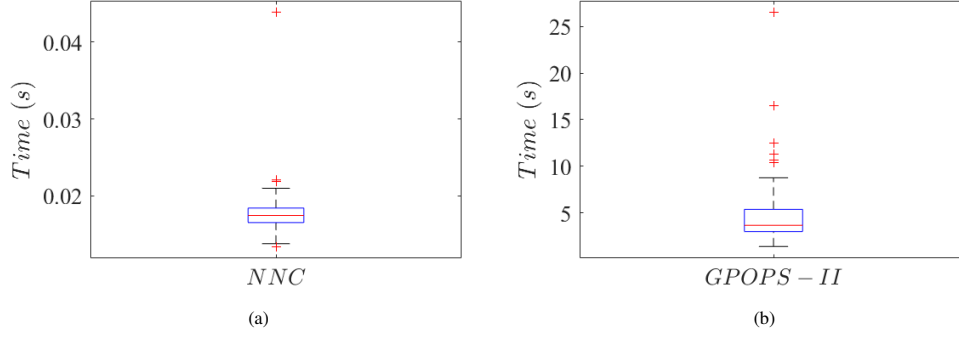


Fig. 15 Comparison between the NNC and GPOPS-II of the required time to solve a TPBVP for the 3D quadrotor. 500 TPBVPs are solved using the NNC and GPOPS-II, respectively. The two boxplots of the required time of the two methods are shown in (a) and (b), respectively. The average time to solve the TPBVP using the NNC is 0.0176 sec, while the average time to solve the TPBVP using GPOPS-II is 4.1811 sec. The online computation of the NNC is more than 200 times faster than GPOPS-II.

VI. Conclusions

A feedback controller for the finite-time optimal control problem is built using machine learning techniques. We use a Neural Network (NN) to learn the optimal feedback control law. Based on the observation that the optimal feedback control law for the finite-time control problem is non-stationary and also may be discontinuous, we use the time label as an additional state of the dataset and introduce a clustering step. Clustering divides the offline trajectories into groups, and a NN is trained for each group. Using time information in the dataset considers the case where the optimal control law is an explicit function of the time. We have demonstrated that the NN can be trained to learn the nonlinear relationship between the augmented state and the optimal control underlying the nonlinear finite-time optimal control problem. Our method generates near-optimal trajectories that steer a nonlinear system from any initial state inside a set to a goal state. The training data generation and NN training are done offline thus the online computation is minimized. The algorithm is tested on several systems. The proposed method is suitable for real-time optimal control and trajectory generation for large classes of nonlinear systems.

References

- [1] Betts, J. T., “Survey of numerical methods for trajectory optimization,” *Journal of Guidance, Control, and Dynamics*, Vol. 21, No. 2, 1998, pp. 193–207.
- [2] Geisert, M., and Mansard, N., “Trajectory generation for quadrotor based systems using numerical optimal control,” *IEEE International Conference on Robotics and Automation*, Stockholm, Sweden, May 2016, pp. 2958–2964.
- [3] Ardakani, M. M. G., Olofsson, B., Robertsson, A., and Johansson, R., “Model Predictive Control for Real-Time Point-to-Point Trajectory Generation,” *IEEE Transactions on Automation Science and Engineering*, Vol. 26, No. 2, 2019, pp. 972–983.
- [4] Karaman, S., and Frazzoli, E., “Sampling-based algorithms for optimal motion planning,” *The International Journal of Robotics Research*, Vol. 30, No. 7, 2011, pp. 846–894.
- [5] Perez, A., Platt, R., Konidaris, G., Kaelbling, L., and Lozano-Perez, T., “LQR-RRT*: Optimal sampling-based motion planning with automatically derived extension heuristics,” *IEEE International Conference on Robotics and Automation*, Saint Paul, MN, May 2012, pp. 2537–2542.
- [6] Chiang, H. T. L., Hsu, J., Fiser, M., Tapia, L., and Faust, A., “RL-RRT: Kinodynamic motion planning via learning reachability estimators from RL policies,” *IEEE Robotics and Automation Letters*, Vol. 4, No. 4, 2019, pp. 4298–4305.
- [7] Bishop, C. M., *Pattern Recognition and Machine Learning*, Springer, 2006.
- [8] Ghosh, P., and Conway, B., “Near-optimal feedback strategies for optimal control and pursuit-evasion games: a spatial statistical approach,” *AIAA/AAS Astrodynamics Specialist Conference*, Minneapolis, MN, August 2012.

- [9] Tsiotras, P., and Diaz, R. S., “Real-time near-optimal feedback control of aggressive vehicle maneuvers,” *Optimization and Optimal Control in Automotive Systems*, edited by H. Waschl, I. Kolmanovsky, M. Steinbuch, and L. del Re, Springer, 2014, Chap. 2, pp. 109–129.
- [10] Sanchez-Sanchez, C., and Izzo, D., “Real-time optimal control via Deep Neural Networks: study on landing problems,” *Journal of Guidance, Control, and Dynamics*, Vol. 41, No. 5, 2018, pp. 1122–1135.
- [11] Furfaro, R., Bloise, I., Orlandelli, M., Di Lizia, P., Topputo, F., and Linares, R., “A Recurrent Deep Architecture for Quasi-Optimal Feedback Guidance in Planetary Landing,” *IAA/AAS SciTech Forum on Space Flight Mechanics and Space Structures and Materials*, Moscow, Russian Federation, 2018, pp. 1–24.
- [12] Cheng, L., Wang, Z., Song, Y., and Jiang, F., “Real-Time Optimal Control for Irregular Asteroid Landings Using Deep Neural Networks,” *Acta Astronautica*, Vol. 170, 2020, pp. 66–79.
- [13] Patterson, M. A., and Rao, A. V., “GPOPS-II: A MATLAB software for solving multiple-phase optimal control problems using hp-adaptive Gaussian quadrature collocation methods and sparse nonlinear programming,” *ACM Transactions on Mathematical Software*, Vol. 41, No. 1, 2014, pp. 1–41.
- [14] Del Re, L., Allgöwer, F., Glielmo, L., Guardiola, C., and Kolmanovsky, I., *Automotive Model Predictive Control: Models, Methods and Applications*, Springer, 2010.
- [15] Tang, G., Sun, W., and Hauser, K., “Learning Trajectories for Real-Time Optimal Control of Quadrotors,” *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Madrid, Spain, 2018, pp. 3620–3625.
- [16] Tang, G., and Hauser, K., “Discontinuity-sensitive optimal control learning by mixture of experts,” *IEEE International Conference on Robotics and Automation*, Montreal, Canada, May 2019, pp. 7892–7898.
- [17] Wolfslag, W. J., Bharatheesha, M., Moerland, T. M., and Wisse, M., “RRT-CoLearn: towards kinodynamic planning without numerical trajectory optimization,” *IEEE Robotics and Automation Letters*, Vol. 3, No. 3, 2018, pp. 1655–1662.
- [18] You, S., Wan, C., Dai, R., Lu, P., and Rea, J. R., “Learning-based Optimal Control for Planetary Entry, Powered Descent and Landing Guidance,” *AIAA SciTech Forum*, Orlando, FL, Jan 2020, pp. 849–862.
- [19] Tailor, D., and Izzo, D., “Learning the optimal state-feedback via supervised imitation learning,” *Astrodynamics*, Vol. 3, No. 4, 2019, pp. 361–374.
- [20] Sprague, C. I., Izzo, D., and Ögren, P., “Learning a Family of Optimal State Feedback Controllers,” *arXiv preprint arXiv:1902.10139*, 2019.
- [21] Shi, Y., and Wang, Z., “A Deep Learning-Based Approach to Real-Time Trajectory Optimization for Hypersonic Vehicles,” *AIAA SciTech Forum*, Orlando, FL, Jan 2020, pp. 23–41.
- [22] Schaal, S., and Atkeson, C. G., “Learning control in robotics,” *IEEE Robotics and Automation Magazine*, Vol. 17, No. 2, 2010, pp. 20–29.
- [23] Kuderer, M., Gulati, S., and Burgard, W., “Learning driving styles for autonomous vehicles from demonstration,” *IEEE International Conference on Robotics and Automation*, Seattle, WA, May 2015, pp. 2641–2646.
- [24] Bency, M. J., Qureshi, A. H., and Yip, M. C., “Neural Path Planning: Fixed Time, Near-Optimal Path Generation via Oracle Imitation,” *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Macau, China, 2019, pp. 3965–3972.
- [25] Bryson, A. E., and Ho, Y. C., *Applied Optimal Control: Optimization, Estimation and Control*, CRC Press, 1975.
- [26] Reeds, J. A., and Shepp, R. A., “Optimal paths for a car that goes both forward and backwards,” *Pacific J. Mathematics*, Vol. 145, No. 2, 1990, pp. 367–393.
- [27] Anisi, D. A., Hamberg, J., and Hu, X., “Nearly time-optimal paths for a ground vehicle,” *Journal of Control Theory and Applications*, Vol. 1, No. 1, 2003, pp. 2–8.
- [28] Johnson, S. C., “Hierarchical clustering schemes,” *Psychometrika*, Vol. 32, No. 3, 1967, pp. 241–254.
- [29] Hartigan, J. A., and Wong, M. A., “Algorithm AS 136: A k-means clustering algorithm,” *Journal of the Royal Statistical Society. Series C*, Vol. 28, No. 1, 1979, pp. 100–108.

- [30] Elhamifar, E., and Vidal, R., “Sparse subspace clustering: Algorithm, theory, and applications,” *IEEE transactions on pattern analysis and machine intelligence*, Vol. 35, No. 11, 2013, pp. 2765–2781.
- [31] Kanungo, T., Mount, D. M., Netanyahu, N. S., Piatko, C. D., Silverman, R., and Wu, A. Y., “An efficient k-means clustering algorithm: Analysis and implementation,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 24, No. 7, 2002, pp. 881–892.
- [32] MacKay, D. J., “Bayesian interpolation,” *Neural computation*, Vol. 4, No. 3, 1992, pp. 415–447.
- [33] Murray, R. M., *A Mathematical Introduction to Robotic Manipulation*, CRC press, 2017.
- [34] Betts, J. T., *Practical Methods for Optimal Control and Estimation using Nonlinear Programming*, SIAM, 2010.