A Data-driven Indirect Method for Nonlinear Optimal Control

Gao Tang[1] and Kris Hauser[2]

Abstract— **Nonlinear optimal control problems are challenging to solve due to the prevalence of local minima that prevent convergence and/or optimality. This paper describes nearest-neighbors optimal control (NNOC), a data-driven framework for nonlinear optimal control using indirect methods. It determines initial guesses for new problems with the help of precomputed solutions to similar problems, retrieved using k-nearest neighbors. A sensitivity analysis technique is introduced to linearly approximate the variation of solutions between new and precomputed problems based on their variation of parameters. Experiments show that NNOC can obtain the global optimal solution orders of magnitude faster than standard random restart methods, and sensitivity analysis can further reduce the solving time almost by half. Examples are shown on two optimal control problems in vehicle control.**

I. INTRODUCTION

Nonlinear optimal control problems are observed in many engineering applications. Despite decades of research, they are still difficult to solve globally with high confidence. Methods for solving optimal control problems mainly fall into two categories: direct methods and indirect methods [3]. Direct methods, such as transcription and collocation, convert the optimal control problem into an optimization problem through parameterization. Indirect methods derive the necessary conditions for optimality using costate variables, and convert the optimal control problem into a two-point boundary value problem (TPBVP) [5]. Indirect methods can be more efficient than direct methods. However, they are difficult to apply successfully because the TPBVP has a narrow convergence domain [9].

The main difficulty of indirect methods is the costate variables lack physical meanings so good starting values are difficult to provide. For problems with strong nonlinearity, the convergence domain is so narrow that a large number of initial guesses have to be tried to obtain convergence. Hence, they must be augmented with another approach like random restarts to have any chance of obtaining a global optimum. Such a shortcoming makes these methods impractical for real-time applications.

In recent years, the approach of using precomputed data to initialize optimization processes has received some attention, with some success in trajectory optimization [8] and global nonlinear optimization [7]. The general idea is that a database of solutions can be precomputed among a parameterized set of optimization problems. For a novel problem, if an example exists in the database whose parameters are sufficiently close to its parameters, then solution to the existing problem should be near the solution to the new one. To our knowledge, this approach has not yet been applied to indirect solution of optimal control problems.

We present NNOC, a data-driven implicit optimal control method that attempts to solve TPBVPs using the k nearest problems in the database to determine initial costate guesses. We also present a new technique that uses sensitivity analysis to approximate how solutions vary with respect to problem parameters, which gives more accurate method for guessing initial costates. In our experiments, we study two vehicle control problems of 4D and 5D, respectively. Experiments study how the technique performs compared to other methods, and demonstrate extremely high success rates as the size of the database grows. The use of our proposed sensitivity analysis technique also reduces solution times by approximately 50%.

II. RELATED WORK

For nonlinear optimal control problems solved by indirect methods, the optimal control might not be continuous, i.e. bang-bang control. The system dynamical and corresponding Euler-Lagrange are usually difficult to integrate analytically so numerical integration has to be applied. Another problem is the strong nonlinearity of the problem so the convergence domain is quite small. As a result, the optimal control for quite simple dynamical system is difficult to solve so its application is limited.

In [9], [2] a homotopic approach is used so we can start from an easier problem and gradually approximate the original problem. In [9] a distribution of initial costate variables is proposed which more or less solve the problem that we do not know the bounds of costate variables. Other techniques include grid search of costate variables [16].

The idea of learning from experience has attracted researchers' interest for decades in robotics field. Machine learning techniques have been used in trajectory optimization to predict the outcome of a starting point [6] or to predict good initial trajectories [14] for local optimization. The grasps of novel objects are generated using the experience of grasps [4]. In [10] precomputed examples of optimal trajectory are generalized to enable the optimizer to work in real-time, as applied to a robot ball-catching problem.

Our technique is related to explicit model predictive control (MPC) [1], a technique for linearly constrained linear systems with quadratic costs, that analytically computes the piecewise-linear optimal MPC control function over all initial

*This work was partially supported by NSF grant IIS-#1218534

[1]G. Tang is with department of Mechanical Engineering and Material Science, Duke University, Durham, NC 27708, USA gao.tang@duke.edu

[2]K. Hauser is with the Departments of Electrical and Computer Engineering and of Mechanical Engineering and Materials Science, Duke University, Durham, NC, 27708 USA kris.hauser@duke.edu

states. Our approach takes a similar approach to nonlinear optimal control using data.

III. DATA-DRIVEN FRAMEWORK

A. Indirect Methods for Optimal Control

Indirect methods essentially convert an optimal control problem to a system of nonlinear equations. Suppose a nonlinear optimal control problem is given as

$$\text{Minimize} \quad J = \varphi(t_0, \boldsymbol{x}_0, t_f, \boldsymbol{x}_f) + \int_{t_0}^{t_f} L(t, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p}) \mathrm{d}t \tag{1}$$

$$\text{s.t.} \quad \dot{\boldsymbol{x}} = \boldsymbol{f}(t, \boldsymbol{x}) \tag{2}$$

$$\boldsymbol{c}(\boldsymbol{u}) \leq \boldsymbol{0} \tag{3}$$

$$\boldsymbol{h}(t_0, \boldsymbol{x}_0, t_f, \boldsymbol{x}_f) = \boldsymbol{0} \tag{4}$$

where $\boldsymbol{x} \in \mathbb{R}^n$ is the state variable, $\boldsymbol{u} \in \mathbb{R}^m$ is the control, $\boldsymbol{p} \in \mathbb{R}^k$ is the problem parameter, and $t \in [t_0, t_f]$ is time. The path constraint $\boldsymbol{c}$ collects inequality constraints on control. $\boldsymbol{h}$ is a collection of equality constraint on state variables at initial and final time. For simplicity we will ignore other types of path constraints. The indirect method introduces corresponding costate variables $\boldsymbol{\lambda} \in \mathbb{R}^n$ and the Hamiltonian [5]

$$H = L(t, \boldsymbol{x}, \boldsymbol{u}) + \boldsymbol{\lambda}^{\mathrm{T}} \dot{\boldsymbol{x}} = L(t, \boldsymbol{x}, \boldsymbol{u}) + \boldsymbol{\lambda}^{\mathrm{T}} f(t, \boldsymbol{x}, \boldsymbol{u}) \tag{5}$$

and derive the Euler-Lagrange equations

$$\begin{cases} \dot{\boldsymbol{x}} = \dfrac{\partial H}{\partial \boldsymbol{\lambda}} \\ \dot{\boldsymbol{\lambda}} = -\dfrac{\partial H}{\partial \boldsymbol{x}} \end{cases} \tag{6}$$

and the optimal control

$$\boldsymbol{u}^\star = \underset{\boldsymbol{c}(\boldsymbol{u}) \leq \boldsymbol{0}}{\arg\min} H \tag{7}$$

It is from this latter condition that the optimal control $\boldsymbol{u}^\star$ as a function of $\boldsymbol{x}$ and $\boldsymbol{\lambda}$ can (usually) be determined. If the control $\boldsymbol{u}$ has no constraint, we uses $\partial H / \partial \boldsymbol{u} = \boldsymbol{0}$ to calculate $\boldsymbol{u}^\star$. For example, suppose

$$L(t, \boldsymbol{x}, \boldsymbol{u}) = \boldsymbol{x}^{\mathrm{T}} \boldsymbol{Q} \boldsymbol{x} + \boldsymbol{u}^{\mathrm{T}} \boldsymbol{R} \boldsymbol{u}$$

be a quadratic cost and

$$f(t, \boldsymbol{x}, \boldsymbol{u}) = f_0(t, \boldsymbol{x}) + \sum_{i=1}^{m} f_i(t, \boldsymbol{x}) u_i$$

be the dynamics function. Then

$$\frac{\partial H}{\partial \boldsymbol{u}} = 2\boldsymbol{R}\boldsymbol{u} + \begin{bmatrix} \boldsymbol{\lambda}^{\mathrm{T}} f_1(t, \boldsymbol{x}) \\ \vdots \\ \boldsymbol{\lambda}^{\mathrm{T}} f_m(t, \boldsymbol{x}) \end{bmatrix} = \boldsymbol{0} \tag{8}$$

and hence $\boldsymbol{u}$ can be determined by multiplying the second summand by $-\boldsymbol{R}^{-1}/2$.

Then, to solve for the optimal trajectory given two-point boundary conditions including an initial state,

$$\boldsymbol{x}(t_0) = \boldsymbol{x}_0 \tag{9}$$

and final state

$$\boldsymbol{x}(t_f) = \boldsymbol{x}_f, \tag{10}$$

a shooting method is used to determine the unknowns so that the boundary condition at t_f in Eq. (10) is satisfied. For the fixed-time problem where t_0 and t_f are fixed, the unknowns $\boldsymbol{s}$ are the initial costate variables $\boldsymbol{\lambda}(t_0)$.

In this paper we also consider problems with free t_f. In those problems the final time t_f is also a unknown, so $\boldsymbol{s} \equiv (\boldsymbol{\lambda}, t_f)$ and another boundary condition at t_f will be imposed. Specifically, the following statistic condition should be imposed:

$$H(t_f) = 0. \tag{11}$$

The TPBVP solver guesses all the unknowns $\boldsymbol{s}$ and integrates Eqs. (6) simultaneously using the optimal control from time t_0 to t_f. The unknowns are updated until the boundary conditions (i.e. Eq. (10) for fixed t_f) at t_f are satisfied up to a given tolerance. We implement the TPBVP shooting method by using the nonlinear least-squares software Minpack [12]. Note that these least-squares solvers, typically based on Gauss-Newton or Levenberg-Marquardt methods, are only local optimizations and may indeed fail to find a solution that successfully meets the final state. An initial guess close to the solution is required otherwise convergence cannot be guaranteed. In practice random restart method is usually employed to find the global optimal solution. Thus the efficiency and reliability of indirect methods are limited.

B. Parametric Optimal Control with Varying Initial Conditions

NNOC addresses the span of optimal control problems ranging over all initial conditions, but with a fixed final state $\boldsymbol{x}_f = \boldsymbol{0}$. A complete, globally optimal method for solving the optimal control problem can be viewed as a mapping $\boldsymbol{g}$ from $\boldsymbol{x}_0$ to the optimal solution of the unknowns:

$$\boldsymbol{g} : \boldsymbol{x}_0 \to \boldsymbol{s}^\star. \tag{12}$$

The goal of NNOC is to approximate this map. Assuming $\boldsymbol{x}_0$ is defined in set $\boldsymbol{X}$, the first step to build the database is sampling from $\boldsymbol{X}$ and calculating the corresponding global optimal solution. We can thus form a database of parameter-solution pairs where $\boldsymbol{x}_0$ is the parameter and $\boldsymbol{s}_0^\star$ is the solution. Specifically, the database $D = \{(\boldsymbol{x}_0^{(i)}, \boldsymbol{s}_0^{\star(i)}) | i = 1, \ldots, N\}$ where we denote $(\boldsymbol{x}_0^{(i)}, \boldsymbol{s}_0^{\star(i)})$ as an optimal control pair. Since the computation of the database is offline, we employ heuristic global optimization techniques such as random restarts. It should be noted that for a general nonlinear optimal control problem there is no guarantee that a global optimal solution can be found. The best local optimal solution is considered as the global optimal solution so a sufficiently large number of restarts is used. However, it is possible that for certain initial states no solution exist or we fail to find a feasible solution. In that case we simply mark that no solution exists.

C. Extending the Database along Trajectories

We observe that each successful trajectory solve provides not only the optimal costate at the initial time t_0, but also all times thereafter. Hence, we can populate the database more quickly by generating optimal problem/solution pairs $(\boldsymbol{x}(t), \boldsymbol{s}(t))$ along the trajectory. So, after calculating an optimal trajectory $\boldsymbol{x}(t)$, costate trajectory $\boldsymbol{\lambda}(t)$, and optionally the final time t_f, we evenly sample M states and costates along the trajectory and add their examples to the database. Specifically, we divide the time range $[t_0, t_f]$ at intermediate values $t_i, i = 1, \ldots, M$, and add all of $(\boldsymbol{x}(t_i), \boldsymbol{s}(t_i))$ to the database. In the case where $\boldsymbol{s}$ contains final time we set the final time for point i to be $t_0 + t_f - t_i$.

D. Query of Database

For a novel problem, NNOC performs a k-nearest neighbor query of the database, and the costates for each of the k nearest neighbors are used as seeds for the TPBVP solver. The feasible solution with minimum cost is kept. The NN query is performed using the approximate nearest-neighbors library FLANN [13] [1]. Several parameters affect performance, including:

1) Database size N. If N is too small, the distance between new parameters and its nearest neighbors might be too large, so the solutions might not lead to convergence. Larger N is needed for problems that are highly nonlinear with small convergence domain. Nearest-neighbor lookup time is fairly insensitive to N due to the use of approximate methods.
2) Distribution of the examples in the database should approximate the query distribution. A naïve method is to sample each state component uniformly at random in a range, but if knowledge is available about which states are encountered in practice, it should be employed.
3) Number of neighbors k determines how many precomputed solutions are used as tentative values for new problems. A larger k contributes to the robustness by combating the effects of local optimal solutions. However, larger values increase running time.

We also present the option to employ sensitivity analysis when determining an initial guess to a novel problem. Rather than using precomputed solutions directly as the initial guess for new problems, this method builds a first-order approximation of $\boldsymbol{g}$ to determine a better guess. It is described below.

E. Sensitivity Analysis

Assuming the mapping from parameters to solutions is continuous and differentiable, sensitivity analysis can be used to build a first-order approximation to their variations. We can thus obtain a better initial guess than directly using the solutions of precomputed problems.

NNOC queries the database to find neighbors of the new state. But instead of using the solutions of the neighbors directly as the initial guess, we can obtain a better guess using the first-order approximation of the relation between the variation of parameters and the variation of solutions.

Let us first explain the method for the fixed-time case. Denote $\boldsymbol{\lambda}_0 \equiv \boldsymbol{\lambda}(t_0)$. We take the variation of (10) and obtain

$$\frac{\partial \boldsymbol{x}(t_f)}{\partial \boldsymbol{x}_0} \delta \boldsymbol{x}_0 + \frac{\partial \boldsymbol{x}(t_f)}{\partial \boldsymbol{\lambda}_0} \delta \boldsymbol{\lambda}_0 = \boldsymbol{0} \tag{13}$$

where $\frac{\partial \boldsymbol{x}(t_f)}{\partial \boldsymbol{x}_0}$ and $\frac{\partial \boldsymbol{x}(t_f)}{\partial \boldsymbol{\lambda}_0}$ are easily obtained by integrating the variational equation of the system dynamics. We refer to [11] for further details.

Using (13) we can obtain a linear relationship between the change of initial state and the change of initial costate variables

$$\delta \boldsymbol{\lambda}_0 = \frac{\partial \boldsymbol{\lambda}_0}{\partial \boldsymbol{x}_0} \delta \boldsymbol{x}_0 = -\frac{\partial \boldsymbol{x}(t_f)}{\partial \boldsymbol{\lambda}_0}^{-1} \frac{\partial \boldsymbol{x}(t_f)}{\partial \boldsymbol{x}_0} \delta \boldsymbol{x}_0. \tag{14}$$

It should be noted that the matrix $\frac{\partial \boldsymbol{\lambda}_0}{\partial \boldsymbol{x}_0}$ can be computed offline since it is determined by $\boldsymbol{x}_0$ and $\boldsymbol{\lambda}_0$ and can be computed when the database is being built.

Then, when a query problem $\boldsymbol{x}_0$ is matched to an example $(\boldsymbol{x}, \boldsymbol{\lambda})$ in the database (noting $\boldsymbol{\lambda} = \boldsymbol{s}$ in this example), we seed the solver with the initial costate

$$\boldsymbol{\lambda} + \frac{\partial \boldsymbol{\lambda}}{\partial \boldsymbol{x}} (\boldsymbol{x}_0 - \boldsymbol{x}). \tag{15}$$

For free-time problems, we must compute the sensitivity of both $\boldsymbol{\lambda}_0$ and t_f with respect to $\boldsymbol{x}_0$. To do so, compute the variation of (10) and (11), obtaining

$$\begin{cases} \frac{\partial \boldsymbol{x}(t_f)}{\partial \boldsymbol{x}_0} \delta \boldsymbol{x}_0 + \frac{\partial \boldsymbol{x}(t_f)}{\partial \boldsymbol{\lambda}_0} \delta \boldsymbol{\lambda}_0 + \frac{\partial \boldsymbol{x}(t_f)}{\partial t_f} \delta t_f = \boldsymbol{0} \\ \frac{\partial H(t_f)}{\partial \boldsymbol{x}_0} \delta \boldsymbol{x}_0 + \frac{\partial H(t_f)}{\partial \boldsymbol{\lambda}_0} \delta \boldsymbol{\lambda}_0 + \frac{\partial H(t_f)}{\partial t_f} \delta t_f = 0 \end{cases} \tag{16}$$

Here, $\frac{\partial \boldsymbol{x}(t_f)}{\partial \boldsymbol{x}_0}$ and $\frac{\partial \boldsymbol{x}(t_f)}{\partial \boldsymbol{\lambda}_0}$ are obtained as before; $\frac{\partial \boldsymbol{x}(t_f)}{\partial t_f} = \dot{\boldsymbol{x}}(t_f)$ is obtained by substituting the optimal control into the dynamics function; $\frac{\partial H(t_f)}{\partial t_f} = 0$ since H does not depend on time [5]; $\frac{\partial H(t_f)}{\partial \boldsymbol{x}_0} = \frac{\partial H(t_0)}{\partial \boldsymbol{x}_0} = -\dot{\boldsymbol{\lambda}}(t_0)$; and $\frac{\partial H(t_f)}{\partial \boldsymbol{\lambda}_0} = \frac{\partial H(t_0)}{\partial \boldsymbol{\lambda}_0} = \dot{\boldsymbol{x}}(t_0)$. Then we can calculate $\delta \boldsymbol{\lambda}_0$ and δt_f by solving a system of $n+1$ linear equations.

$$\begin{bmatrix} \delta \boldsymbol{\lambda}_0 \\ \delta t_f \end{bmatrix} = -\begin{bmatrix} \frac{\partial \boldsymbol{x}(t_f)}{\partial \boldsymbol{\lambda}_0} & \frac{\partial \boldsymbol{x}(t_f)}{\partial t_f} \\ \frac{\partial H(t_f)}{\partial \boldsymbol{\lambda}_0} & \frac{\partial H(t_f)}{\partial t_f} \end{bmatrix}^{-1} \begin{bmatrix} \frac{\partial \boldsymbol{x}(t_f)}{\partial \boldsymbol{x}_0} \\ \frac{\partial H(t_f)}{\partial \boldsymbol{x}_0} \end{bmatrix} \delta \boldsymbol{x}_0. \tag{17}$$

As with the fixed-time case, the sensitivity matrix can be done offline and stored with the example in the database.

IV. RESULT

We test NNOC in two problems. Planar Car is a minimum effort problem on a second-order Dubins car. Quadcopter is a minimum-time problem on a dynamic quadcopter model.

[1] http://www.cs.ubc.ca/research/flann/ (accessed 9/2/2017)

In each case we seed the database using randomly sampled initial states, solved using a random restart method. The first example is a basic demonstration of the data-driven technique. Test sets of 1,000 problems are randomly generated from the same distributions as we generate the training set. The database was extended in the Quadcopter case with $M = 100$ trajectory samples. This example demonstrates we can use the technique of extending database along trajectories to enlarge our database. In order to use the database, the states of the quadcopter when it is moving towards the target are also added to the test set. We study the effects of parameters such as the stopping criteria for random restart, database size, number of neighbors queried, and whether sensitivity analysis is used. The Planar Car experiments are run on a Macbook Pro with a 2.90 GHz CPU while the Quadcopter experiments are run on a desktop with a 3.60 GHz CPU.

A. Planar Car

We consider a simplified planar car with the following dynamics [19]:

$$\dot{x} = v\sin\theta, \dot{y} = v\cos\theta, \dot{\theta} = u_\theta v, \dot{v} = u_v \tag{18}$$

where the state $\boldsymbol{x} = [x, y, \theta, v]$ includes the planar coordinates, orientation, and velocity of the vehicle. The control $\boldsymbol{u} = [u_\theta, u_v]$ includes the control variables which change the steering angle and velocity, respectively.

1) Optimal Control Formulation: The performance index is given by the quadratic control effort

$$J = \int_{t_0}^{t_f} \boldsymbol{u}^{\mathrm{T}} \boldsymbol{R} \boldsymbol{u} \mathrm{d}t \tag{19}$$

where $\boldsymbol{R}$ is a diagonal matrix with entries $r_1 = 0.2$ and $r_2 = 0.1$, as chosen in accordance with [19].

We arbitrarily choose fixed initial and final times, i.e. $t_0 = 0, t_f = 2.5$. Initial states are sampled from the range $x \in [-3,0], y \in [-3,3], \theta \in [-\pi,\pi], v = 0$. It should be noted that due to the symmetry of the problem we do not have to consider positive x. The target state is the origin, so that final orientation and velocity is 0.

The costate variables $\boldsymbol{\lambda} = [\lambda_x, \lambda_y, \lambda_\theta, \lambda_v]$ are governed by the Hamiltonian [5]

$$\begin{aligned} H &= \boldsymbol{\lambda}^{\mathrm{T}} \dot{\boldsymbol{x}} + \boldsymbol{u}^T \boldsymbol{R} \boldsymbol{u} \\ &= \lambda_x v\sin\theta + \lambda_y v\cos\theta + \lambda_\theta v u_\theta + \lambda_v u_v + r_1 u_\theta^2 + r_2 u_v^2 \end{aligned} \tag{20}$$

and we derive the Euler-Lagrange equations

$$\begin{cases} \dot{\lambda}_x = -\dfrac{\partial H}{\partial x} = 0 \\ \dot{\lambda}_y = -\dfrac{\partial H}{\partial y} = 0 \\ \dot{\lambda}_\theta = -\dfrac{\partial H}{\partial \theta} = -\lambda_x v\cos\theta + \lambda_y v\sin\theta \\ \dot{\lambda}_v = -\dfrac{\partial H}{\partial v} = -\lambda_x \sin\theta - \lambda_y \cos\theta - \lambda_\theta u_\theta \end{cases} \tag{21}$$

Then we readily derive the optimal control which minimizes H as

$$u_\theta^* = -\frac{v\lambda_\theta}{2r_1}, u_v^* = -\frac{\lambda_v}{2r_2} \tag{22}$$

2) Estimation of magnitude of costate variables: The difficulty for providing tentative $\boldsymbol{\lambda}(t_0)$ is partially caused by the its unknown bounds. Admittedly, with the help of the normalization of initial costate variables [9] we can sample them on the surface of a high-dimension ball. Here we use a non-rigorous formula to help provide bounds for initial costate variables. Experiments show that it helps to increase success rate of random restart technique. The formula we use for estimation of the magnitude of λ_v is

$$\bar{s} = 2\sqrt{x^2 + y^2}, \bar{v} = \frac{\bar{s}}{t}, \bar{a} = \frac{4\bar{v}}{t}, \bar{\lambda}_v = 2\bar{a} r_2 \tag{23}$$

where $\bar{(\cdot)}$ denotes the magnitude. We first estimate the length of the trajectory, then average velocity and average acceleration assuming a constant acceleration and deceleration. Then the formulation of the optimal control is used to get the magnitude of λ_v. Using a similar way, we obtain the magnitude of λ_θ as

$$\bar{\lambda}_\theta = \frac{16|\theta_0| r_1}{t^2 \bar{v}^2} \tag{24}$$

It should be noted that an additional term $\bar{v}^2$ is multiplied in the denominator because the additional multiplication of v in Eqs. (22) and (18).

3) Simulation result: We evaluate four methods that differ in how initial guesses are provided and how the iteration is terminated.

1) RR: Random Restart of k initial guesses, where $k = 5, 10, 50, 100$
2) RL: Random restart after k Locally optimal solutions are solved, where $k = 1, 3, 5, 10$
3) NNOC: start with the solutions of the k Nearest Neighbors where $k = 1, 5, 10$
4) NNOC+SA: start with the solutions of the k Nearest Neighbors where $k = 1, 5, 10$ with Sensitivity Analysis

It should be noted that in RL a maximum restart number of 100 is set in order to avoid an infinite loop. For random restart techniques, the normalization of initial costates is used [9].

To build the database, we randomly generate 20,000 initial states and calculate the corresponding costates. Then 5 databases of size 1,250, 2,500, 5,000, 10,000, 20,000 are constructed, respectively. In Fig. 1 we plot the optimal trajectories of 30 examples where the arrow shows the direction the car is heading. We record the running time, number of solutions, and the best performance index of each method. The results for RR and RL are listed in Tab. I and Tab. II, respectively. Success Rate denotes the percentage of obtaining at least one local optimal solution; Avg. Conv. denotes the average number of local optima; Global Rate denotes the fraction of times a method obtains a globally optimal solution; Avg. Time is the average computation time for solving the TPBVP using shooting methods. To calculate Global Rate, for each test example we compute the minimum cost solution found over all methods tested. This value is then considered the globally minimum cost. A local optimum returned by each method is considered global if its cost

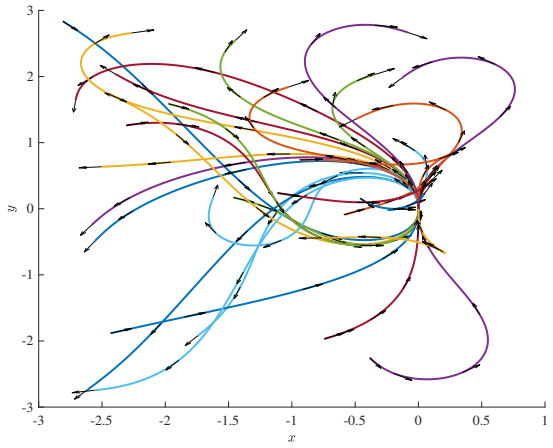

Fig. 1. Sample of optimal trajectories for Planar Car example.

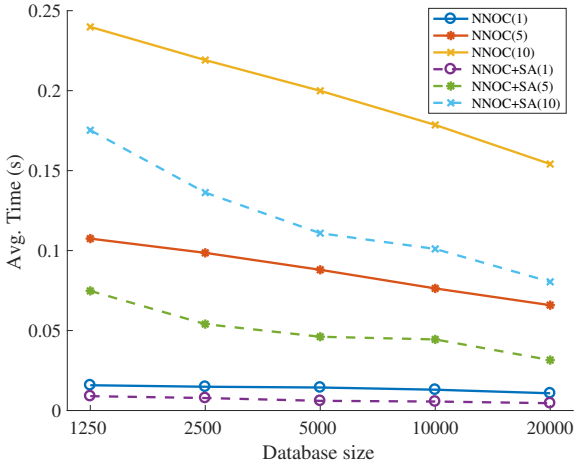

Fig. 2. Average computation time of NNOC for Planar Car example.

is close to the minimum cost (the difference is less than 1×10^{-6}). For this problem, we find that for a desired Global Optimal Rate of 95%, about 100 random initial guesses must be tried, and RR takes about 7 s.

TABLE I
RESULTS OF RR FOR PLANAR CAR EXAMPLE

k	Success Rate	Avg. Conv.	Global Rate	Avg. Time (s)
5	0.673	1.145	0.494	0.378
10	0.862	2.385	0.684	0.769
50	0.997	11.824	0.939	3.831
100	1	23.697	0.966	7.649

TABLE II
RESULTS OF RL FOR PLANAR CAR EXAMPLE

k	Success Rate	Avg. Conv.	Global Rate	Avg. Time (s)
1	1	5.679	0.610	0.562
3	0.992	15.516	0.854	1.285
5	0.982	25.070	0.910	1.982
10	0.932	45.002	0.917	3.276

Fig. 2 compares running times to NNOC, and Fig. 3 shows the Global Optimal Rate, for differing numbers of database size N, neighbor count k, and whether sensitivity analysis is enabled. It can be observed that the average computation time decreases with increasing N. The influence of N on average computation time is more significant for larger k. This is reasonable since when k is large, the distance of some neighbors might be quite large and increasing N can help avoid initial guesses that takes a long time to converge. Increasing k increases global optimal rate significantly, but at the cost of increasing computation time. Sensitivity analysis significantly reduces the average computation time and increases global optimal rate. For a target global optimal rate

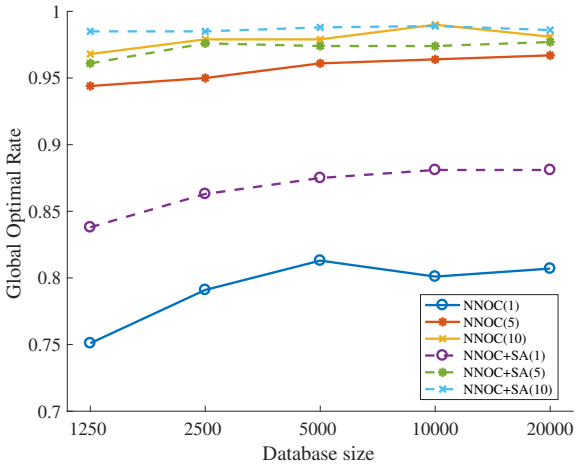

Fig. 3. Global Optimal Rate of NNOC for Planar Car example.

of 95%, $k=5$ and $N=20{,}000$ leads to average computation time of 0.05 s which is nearly two orders of magnitude faster than RR.

B. Planar Quadcopter

The Quadcopter example defines the following dynamics [15]:

$$\ddot{x}=\frac{F_T}{m}\sin\theta,\ \ddot{z}=\frac{F_T}{m}\cos\theta-g,\ \dot{\theta}=\omega \tag{25}$$

where g is the gravitational acceleration; m the mass of the quadcopter; F_T the total thrust force; and ω the pitch rate. The state $\boldsymbol{x}=[x,z,\dot{x},\dot{z},\theta]$ include the x,z coordinates, the velocity, and the pitch angle. The control is defined as $\boldsymbol{u}=[u,\omega]$ where $u=F_T/m$. Both controls are subject to saturation:

$$\underline{u}\le u\le\overline{u},\ |\omega|\le\overline{\omega} \tag{26}$$

where $\underline{u}=1,\overline{u}=20,\overline{\omega}=5$. The initial state of the quadcopter is randomly sampled such that $x\in[-10,0],z\in[-10,10],\dot{x}=\dot{z}=\theta=0$. Due to symmetry of the problem, we do not have to sample the cases with positive x.

1) Optimal Control Formulation: The objective in this problem is to move to and hover at the origin in minimum time. The time-optimal performance index is

$$J = \int_0^{t_f} 1 \mathrm{d}t \tag{27}$$

where t_f is a free variable. However, the resulting optimal control is bang-bang control which is numerically challenging to solve [15]. Hence, we use an alternate formulation that adds regularization parameters to the performance index so the resulting optimal control is continuous [18]. We introduce a logarithmic barrier function to the performance index, which is a widely-used method in the field of aerospace engineering. [2], [17], as follows,

$$\begin{aligned} J &= \int_0^{t_f} L \mathrm{d}t \\ &= \int_0^{t_f} 1 - \varepsilon_1 \ln[\hat{u}(1-\hat{u})] - \varepsilon_2 \ln[\hat{\omega}(1-\hat{\omega})] \mathrm{d}t. \end{aligned} \tag{28}$$

where $\hat{u} \in [0,1]$ and $\hat{\omega} \in [0,1]$ are nondimensionalized controls such that $u \equiv \underline{u} + (\overline{u} - \underline{u})\hat{u}$ and $\omega \equiv \underline{\omega} + (\overline{\omega} - \underline{\omega})\hat{\omega}$. It can be shown that with this modification, the resulting optimal control is continuous and differentiable. The parameters ε_1 and ε_2 control the magnitude of the logarithmic barrier. Smaller values lead to a better approximation to the bang-bang control, but they also enlarge numerical sensitivity and thus reduce the convergence domain. In this work we initially choose the values $\varepsilon_1 = \varepsilon_2 = 1$ which are relatively large compared with [17], but also have a larger convergence domain as the optimal control problem becomes less ill-conditioned.

With costate variables $\boldsymbol{\lambda} = [\lambda_x, \lambda_z, \lambda_{\dot{x}}, \lambda_{\dot{z}}, \lambda_\theta]$, we write the Hamiltonian as

$$H = \lambda_x \dot{x} + \lambda_z \dot{z} + \lambda_{\dot{x}} u \sin\theta + \lambda_{\dot{z}}(u\cos\theta - g) + \lambda_\theta \omega + L \tag{29}$$

and derive the Euler-Lagrange equations

$$\begin{cases} \dot{\lambda}_x = -\dfrac{\partial H}{\partial x} = 0, \dot{\lambda}_y = -\dfrac{\partial H}{\partial x} = 0 \\ \dot{\lambda}_{\dot{x}} = -\dfrac{\partial H}{\partial \dot{x}} = -\lambda_x, \dot{\lambda}_{\dot{z}} = -\dfrac{\partial H}{\partial \dot{z}} = -\lambda_z \\ \dot{\lambda}_\theta = -\dfrac{\partial H}{\partial \theta} = \lambda_{\dot{z}} u \sin\theta - \lambda_{\dot{x}} u \cos\theta \end{cases} \tag{30}$$

Then the non-dimensionalized optimal control that minimizes H is

$$\begin{cases} u^* = \dfrac{2\varepsilon_1}{2\varepsilon_1 + \rho_1 + \sqrt{4\varepsilon_1^2 + \rho_1^2}} \\ \omega^* = \dfrac{2\varepsilon_2}{2\varepsilon_2 + \rho_2 + \sqrt{4\varepsilon_2^2 + \rho_2^2}} \end{cases} \tag{31}$$

where ρ_1 and ρ_2 are called switching functions and are defined as

$$\rho_1 = (\overline{u} - \underline{u})(\lambda_{\dot{x}} \sin\theta + \lambda_{\dot{z}} \cos\theta), \rho_2 = (\overline{\omega} - \underline{\omega})\lambda_\theta. \tag{32}$$

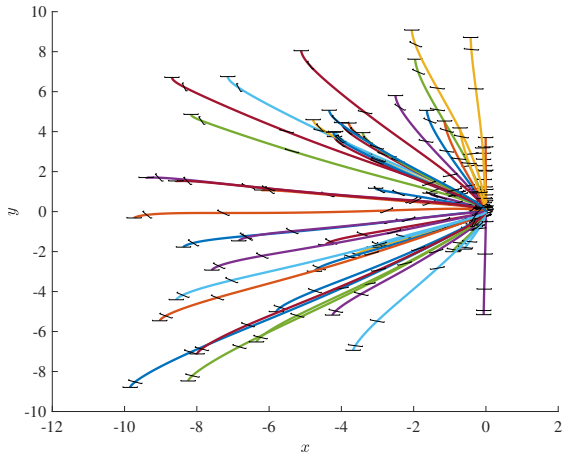

Fig. 4. Sample optimal trajectories for the Quadcopter example.

2) Simulation Result: In this example, a different approach is applied to compare RR and data-driven technique. We use the same method as we generate the training set to get 500 samples for the test set. We use the same way of extending the database along trajectories to generate testing examples. Along each trajectory we evenly sample $l = 10$ states. The following results is a summary of the simulation results on the sampled initial states.

To build the database, we randomly generate 2,000 initial states and calculate the corresponding costates. Then the technique of sampling optimal trajectories to extend the database is applied. 5 databases of size 12,500, 25,000, 50,000, 100,000, 200,000 are constructed, respectively. In Fig. 4 we plot the optimal trajectories of 50 examples. The results for RR and RL are listed in Tab. III- IV. We note that for this problem getting local optima is not a serious problem and the Global Optimal Rate approximately equals the success rate of returning one result. Random restart until 1 local optimal solution is found is a good choice to solve this problem. On average this takes about 0.2 s.

TABLE III
RESULTS OF RR FOR PLANAR QUADCOPTER EXAMPLE

k	Success Rate	Avg. Conv.	Global Rate	Avg. Time (s)
5	0.732	2.030	0.732	0.244
10	0.799	4.063	0.899	0.490
50	0.853	20.137	0.853	2.448
100	0.861	40.285	0.861	4.901

Fig. 5 shows solving time for NNOC. Increasing k leads to longer solving time since more instances of TPBVP must be solved. But increasing the database size and using sensitivity analysis decreases computation time. However, when $k = 1$, the decrease in computation time is quite small. Fig. 6 shows the Global Optimal Rate, showing that NNOC almost always computes the global optimum. When $k = 1$ and sensitivity

TABLE IV

RESULTS OF RL FOR PLANAR QUADCOPTER EXAMPLE

k	Success Rate	Avg. Conv.	Global Rate	Avg. Time (s)
1	0.861	4.111	0.861	0.191
3	0.848	10.181	0.861	0.542
5	0.832	15.150	0.861	0.850
10	0.789	25.376	0.861	1.494

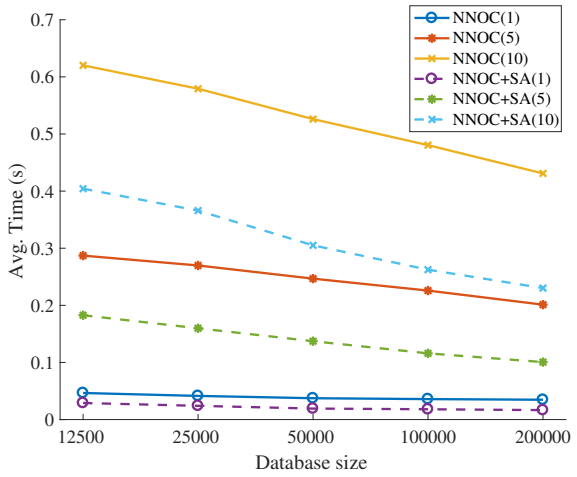

Fig. 5. Average computation time of NNOC for Quadcopter example

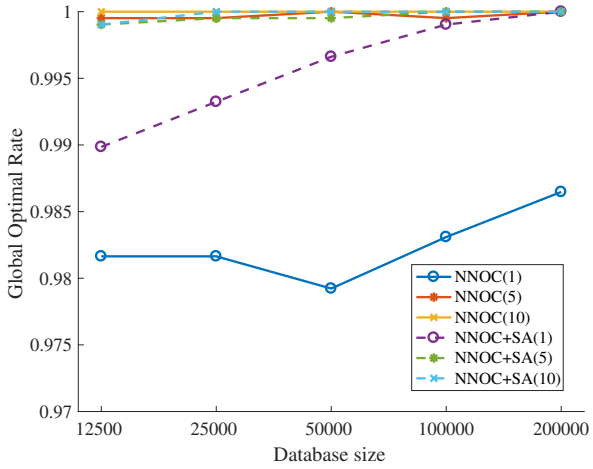

Fig. 6. Global Optimal Rate of NNOC for Quadcopter example

TABLE V

RESULTS FOR MPC WITH NNOC ON THE QUADCOPTER PROBLEM

	Standard DB	Robust DB
NNOC+SA solve failures	1.6%	**0.65%**
Convergence failures	3.2%	**1.4%**

analysis, the global optimum is found about 99% of the time and average computation time is less than 4 ms.

3) Model Predictive Control Simulation: A potential application of NNOC is to implement real-time nonlinear MPC, which we illustrate using the simulated quadcopter problem under disturbances (e.g., wind gusts).

We simulate model uncertainty by adding small perturbation to the system dynamics. The simulated Eq.(25) becomes

$$\ddot{x} = \frac{F_T}{m}\sin\theta + a_x, \ddot{z} = \frac{F_T}{m}\cos\theta - g + a_z, \dot{\theta} = \omega + \alpha\Delta t \quad (33)$$

where a_x and a_z are the unmodelled acceleration along x and z axes; α is the unmodelled angular acceleration; Δt is the instantaneous time from the last control step such that $\alpha\Delta t$ is the unmodelled change of angular velocity. At every control step ($\Delta t = 0.1$), a_x, a_z, α are sampled from normal distributions with standard deviations of 1.

Our implementation of the NNOC-MPC controller receives the current state as input at 10Hz and uses NNOC+SA with $k = 5$ nearest neighbors to solve the optimal control problem. In some cases, convergence cannot be obtained using NNOC. In this case, we try solving using the costate trajectory from the last step as the initial guess, integrating Eq.(30) forward by Δt. If convergence cannot be reached again, then we consider this a failure of the controller. The controller repeats until 1) the quadcopter is close to the origin, as measured by $\|x\| < 2$ or 2) the quadcopter is too far from the origin, as measured by $\|x\| > 20$.

Our first NNOC database simply uses states along the optimal trajectories from states at rest, as before. However, this database does not achieve adequate coverage in areas of the state space that the system might reach under disturbances. We consider generating a *robust database* by simulating the NNOC-MPC controller from random initial states using a larger perturbation and collecting the state and solutions. If the novel state is not solved successfully using the above technique, a brute-force RR(100) is used. Each database has 500,000 parameter-solution pairs. Symmetry about the x axis is exploited by sampling states only with $x \leq 0$, and mirroring any states with $x > 0$. Specifically, if $(x, z, \dot{x}, \dot{z}, \theta)$ and $(\lambda_x, \lambda_z, \lambda_{\dot{x}}, \lambda_{\dot{z}}, \lambda_\theta)$ form a state-solution pair, then $(-x, -z, \dot{x}, \dot{z}, -\theta)$ and $(-\lambda_x, \lambda_z, -\lambda_{\dot{x}}, \lambda_{\dot{z}}, -\lambda_\theta)$ also form a pair.

We perform 500 simulations with random initial states as shown in Fig. 7. We record failed NNOC solves (which can be corrected during execution using a backup solver), as well as overall failures of convergence. Table V gives a numerical comparison. Using the standard database, the controller fails to achieve solutions via NNOC in 1.6% of intermediate states. This also leads to overall failure of convergence in 16 / 500 initial states. Using the robust database, NNOC+SA failed to obtain solutions in 0.65% of intermediate optimal control problems. This leads to a failure of convergence in 7 / 500 initial states. These results demonstrate that NNOC is able to control model uncertainty in a large fraction of trajectories, and that performance is improved by ensuring the database has adequate coverage of the states encountered in practice.

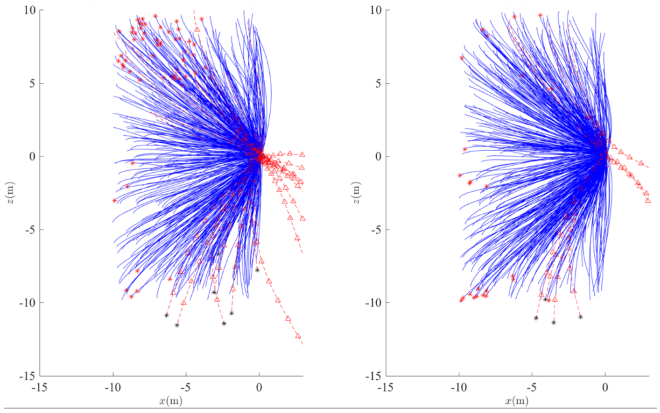

Fig. 7. Comparison of NNOC-MPC simulation on the quadcopter for two databases. Left: the standard database is constructed by sampling along unperturbed optimal trajectories. Right: the database is generated by running MPC under perturbation. Blue trajectories are successful, and red are unsuccessful. Red stars and triangles denote states that are not solved by NNOC-MPC. Black stars mark the final states of failed trajectories (some are not visible in the depicted range).

V. CONCLUSIONS

In this paper a data-driven technique is proposed to help solve nonlinear optimal control problems. NNOC addresses the major difficulty faced in indirect optimal control — providing tentative values for the unknowns — by retrieving the solutions to problems that have already been solved using brute force methods. The effects of several crucial parameters such as the database size, number of neighbors, and whether to use sensitivity analysis are investigated. Compared with brute-force random restart technique, this method can obtain the global optimal solution an order of magnitude faster and has the potential for real-time application in nonlinear MPC.

In future work we intend to enhance the suitability of NNOC for real time control of physical systems. Although current results are promising, robustness could be improved in a number of ways. One approach so might use NNOC to calculate a reference trajectory for a trajectory-tracking controller. Or, we could explicitly optimize robust trajectories for use in the database. Future work should also address scalability to higher-dimensional systems as well as state and parameter uncertainty.

ACKNOWLEDGMENT

This work was partially supported by NSF grant # IIS-1218534.

REFERENCES

[1] A. Bemporad, M. Morari, V. Dua, and E. Pistikopoulos, "The explicit solution of model predictive control via multiparametric quadratic programming," in *Proc. American Control Conf.*, vol. 1–6, 2000, pp. 872 – 876.

[2] R. Bertrand and R. Epenoy, "New smoothing techniques for solving bang–bang optimal control problems: numerical results and statistical interpretation," *Optimal Control Applications and Methods*, vol. 23, no. 4, pp. 171–197, 2002.

[3] J. T. Betts, "Survey of numerical methods for trajectory optimization," *Journal of guidance, control, and dynamics*, vol. 21, no. 2, pp. 193–207, 1998.

[4] J. Bohg, A. Morales, T. Asfour, and D. Kragic, "Data-driven grasp synthesis: a survey," *IEEE Transactions on Robotics*, vol. 30, no. 2, pp. 289–309, 2014.

[5] A. E. Bryson, *Applied optimal control: optimization, estimation and control*. CRC Press, 1975.

[6] A. Cassioli, D. Di Lorenzo, M. Locatelli, F. Schoen, and M. Sciandrone, "Machine learning for global optimization," *Computational Optimization and Applications*, vol. 51, no. 1, pp. 279–303, 2012.

[7] K. Hauser, "Learning the problem-optimum map: Analysis and application to global optimization in robotics," *IEEE Trans. Robotics*, vol. 33, no. 1, pp. 141–152, Feb. 2017.

[8] N. Jetchev and M. Toussaint, "Fast motion planning from experience: trajectory prediction for speeding up movement generation," *Autonomous Robots*, vol. 34, no. 1-2, pp. 111–127, Jan. 2013.

[9] F. Jiang, H. Baoyin, and J. Li, "Practical techniques for low-thrust trajectory optimization with homotopic approach," *J. Guid. Control Dynam.*, vol. 35, no. 1, pp. 245–258, 2012.

[10] R. Lampariello, D. Nguyen-Tuong, C. Castellini, G. Hirzinger, and J. Peters, "Trajectory planning for optimal robot catching in real-time," in *2011 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 3719–3726.

[11] H. Maurer and D. Augustin, "Sensitivity Analysis and Real-Time Control of Parametric Optimal Control Problems Using Boundary Value Methods," in *Online Optimization of Large Scale Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 17–55.

[12] J. J. Moré, B. S. Garbow, and K. E. Hillstrom, "User guide for minpack-1," CM-P00068642, Tech. Rep., 1980.

[13] M. Muja and D. G. Lowe, "Scalable nearest neighbor algorithms for high dimensional data," in *Pattern Analysis and Machine Intelligence*, vol. 36, 2014.

[14] J. Pan, Z. Chen, and P. Abbeel, "Predicting initialization effectiveness for trajectory optimization," in *IEEE Intl. Conf. Robotics and Automation*, 2014.

[15] R. Ritz, M. Hehn, S. Lupashin, and R. D'Andrea, "Quadrocopter performance benchmarking using optimal control," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE, 2011, pp. 5179–5186.

[16] R. P. Russell, "Primer vector theory applied to global low-thrust trade studies," *Journal of Guidance, Control, and Dynamics*, vol. 30, no. 2, pp. 460–472, 2007.

[17] G. Tang and F. Jiang, "Capture of near-Earth objects with low-thrust propulsion and invariant manifolds," *Astrophysics and Space Science*, vol. 361, no. 1, p. 10, 2015.

[18] T. Tomić, M. Maier, and S. Haddadin, "Learning quadrotor maneuvers from optimal control and generalizing in real-time," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 1747–1754.

[19] Z. Xie, C. K. Liu, and K. K. Hauser, "Differential dynamic programming with nonlinear constraints," in *IEEE Int'l. Conf. Robotics and Automation*, Sept. 2016, pp. 1–8.