Astrodynamics Vol. 3, No. 4, 345–359, 2019 https://doi.org/10.1007/s42064-019-0051-3

A data-driven indirect method for nonlinear optimal control

Gao Tang[1] (✉), **Kris Hauser**[2]

1. *Department of Mechanical Engineering and Material Science, Duke University, Durham, NC 27705, USA*
2. *Department of Electrical and Computer Engineering, Duke University, Durham, NC 27705, USA*

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 optimal control problems in vehicle control and agile satellite reorientation demonstrating that global optima can be determined with more than 99% reliability within time at the order of 10–100 milliseconds.

KEYWORDS

data-driven approach
indirect method
optimal control
sensitivity analysis

Research Article

Received: 13 January 2019
Accepted: 19 February 2019
© Tsinghua University Press 2019

1 Introduction

Nonlinear optimal control problems are observed in many engineering applications, but 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 and indirect methods [1]. Direct methods, such as transcription and collocation, convert the optimal control problem into an optimization problem through a parameterization of the state trajectory. 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) [2]. Indirect methods can be more efficient than direct methods. However, they are difficult to apply successfully because the TPBVP has a narrow convergence domain [3].

The main difficulty of indirect methods is that the costate variables lack physical meaning, 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 [4] and global nonlinear optimization [5]. The general idea is that a database of solutions can be precomputed among a parameterized set of optimization problems. If the parameters specifying a novel problem are sufficiently close to the parameters of an example in the database, then the existing solution should be near the solution of the new problem. Using the prior solution as a seed improves both the solution time and the success rate of optimization. Although this approach pays a price in precomputation of thousands or millions of optimization problems, the speed gains may be worthwhile for systems with minimal computing resources, or on-line control applications that require repeated solution of similar problems, like model predictive control (MPC). Moreover, high-performance computing resources can be used to make

✉ gao.tang@duke.edu

the precomputation phase relatively practical. To our knowledge, this approach has not yet been applied to indirect solution of optimal control problems.

This paper presents Nearest Neighbor Optimal Control (NNOC), a data-driven implicit optimal control method that uses a precomputed database of solutions and performs k nearest neighbors lookups to determine initial costate guesses for solving TPBVPs. In addition to the basic approach, we develop a *sensitivity analysis* (SA) technique to approximate how solutions vary with respect to problem parameters. Using this technique to guess initial costates is more accurate than the basic approach.

We evaluate this work in simulation on two vehicle control problems with 4D and 5D state spaces, as well as a satellite reorientation problem with a 6D state space. Experiments study how the technique performs compared to random restart optimization on computation time and likelihood of successfully determining global near-optima. NNOC exhibits success rates close to 100% and is 1–2 orders of magnitude faster than random restarts. As the size of the database grows, NNOC demonstrates increasingly high success rates and lower computation time. The use of our proposed SA technique also reduces solution times by approximately 50% beyond plain NNOC.

2 Related work

For nonlinear optimal control problems solved by indirect methods, the difficulty comes from non-continuous control, i.e., bang-bang control [2], inaccurate numerical integration, and strong sensitivity of the TPBVP. As a result, shooting methods have narrow convergence domain and require an initial guess of costates sufficiently close to optimum in order to converge. Moreover, since TPBVP is built on necessary condition of optimality, shooting methods might converge into a local optimum.

In Refs. [3,6] a homotopic approach is used to address these challenges, in which the solver starts from an easier problem and gradually improves the approximation of the original problem. In Ref. [3] a distribution of initial costates is proposed which roughly overcomes the problem of not knowing the bounds of costate variables. Other techniques have been proposed to guess costates, including grid search of costate variables [7], initialization by direct methods [8] and initialization from linearization [9]. Each of these methods are proposed for specific problems and require researchers' understanding of the problem being studied so they are not easily applied to other problems.

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 [10] or to predict good initial trajectories [11] for local optimization. The grasps of novel objects are generated using the experience of grasps [12]. In Ref. [13] precomputed examples of optimal trajectory are generalized to enable the optimizer to work in real-time, as applied to a robot ball-catching problem. With the advance of deep learning, both deep reinforcement learning [14] and supervised learning [15] have been used to learn a policy for robot control. Other works use supervised learning to directly learn the optimal trajectories [16,17].

Our technique is related to explicit model predictive control (MPC) [18], a technique for linearly constrained linear systems with quadratic costs, that analytically computes the piecewise-linear optimal MPC control function over all initial states. Our approach takes a similar approach to nonlinear optimal control using data.

NNOC is similar to a numerical continuation approach where the solution of a similar problem is used to solve a novel problem [15]. In that work, a random walk approach is used to generate optimal trajectories to train deep neural network. However, in Ref. [15] the issue of local optima is not considered and the data distribution is not uniform. In our approach, multiple neighbors are used to solve novel problems so the chance of obtaining the global optimum is higher.

Recently, a great deal of attention has been devoted to the idea of applying machine learning, especially deep learning, to optimal control problems [15, 16, 19–22]. These learning approaches use a function approximator, like a neural network, to build a policy function, an estimate of the optimal trajectory, or a surrogate model for the value function. NNOC is related, but uses a nonparametric machine learning method to approximate initial costates for each state. We note that for indirect methods a trajectory is fully determined by the state and costate at initial time. As a result, approximating initial costate is equivalent to approximating the optimal trajectory. From now

on, we use trajectory and costate approximation interchangeably. In our experiments, we observe that applying a small amount of trajectory optimization greatly improves learned estimates, and furthermore we are able to better handle discontinuities in the function using multiple initializations from $k > 1$ nearest-neighbors.

This work is an extended version of a previous conference publication [23]. In this version, we extend the discussion of related work, perform extended experiments to analyze the performance of our technique, and add a new satellite reorientation example.

3 Data-driven framework

3.1 Indirect methods for optimal control

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

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

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

$$\boldsymbol{c}(\boldsymbol{u}(t)) \leqslant \boldsymbol{0} \text{ for all } t \tag{3}$$

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

where $\boldsymbol{x}(t) \in \mathbb{R}^n$ is the state variable, $\boldsymbol{u}(t) \in \mathbb{R}^m$ is the control, 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. In this paper, we consider problems where $\boldsymbol{c}$ constrains the bounds on control and $\boldsymbol{h}$ constraints the exact values of initial and final states. For simplicity we will ignore other types of path constraints. The indirect method introduces corresponding costate variables $\boldsymbol{\lambda}(t) \in \mathbb{R}^n$ and the Hamiltonian [2]

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

and derives 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

$$\boldsymbol{u}^\star(t) = \underset{\boldsymbol{u} \,|\, \boldsymbol{c}(\boldsymbol{u}) \leqslant \boldsymbol{0}}{\arg\min} \; H(t, \boldsymbol{x}(t), \boldsymbol{u}) \tag{7}$$

If the control $\boldsymbol{u}$ has no constraint, we use $\partial H/\partial \boldsymbol{u} = \boldsymbol{0}$ 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 $-\boldsymbol{R}^{-1}/2$. In cases where control $\boldsymbol{u}$ is indeed constrained, the optimum $\boldsymbol{u}^*$ is found subject to those constraints.

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 we let $\boldsymbol{s} \equiv (\boldsymbol{\lambda}, t_f)$ and impose another boundary condition at t_f:

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

The TPBVP solver guesses all the unknowns $\boldsymbol{s}$ and integrates Eqs. (6) simultaneously from time t_0 to t_f using the optimal control derived from Eq. (7). 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 a TPBVP shooting method using the nonlinear least-squares software Minpack [24]. In general, the convergence of a nonlinear equation solver depends on several parameter choices, and whether a relaxation technique is used, which

usually increases convergence time but also success rate. Experiments in this paper set all of the parameters of Minpack to their defaults. We observe when the initial guess is quite close to an optimum, as it is in NNOC, the performance of the solver is relatively insensitive to the parameter settings.

Note that these least-squares solvers, typically based on Gauss–Newton or Levenberg–Marquardt methods, are only local optimizers, they may indeed fail to find a solution that successfully meets all the boundary conditions. The basin of attraction of the optimum is often quite small and cannot be predicted easily from the problem specification. In practice, a random restart method is usually employed to increase the likelihood of finding an initial guess in the basin of attraction of a solution. Multiple solutions can be generated and the minimum cost kept, which increases the likelihood of finding a globally optimal solution. This is not particularly reliable without using a large number of guesses, which makes the approach too slow for real-time application.

3.2 Problem-solution mapping

NNOC addresses the scenario where a family of optimal control problems has to be solved and each problem is parameterized by a vector $\boldsymbol{p} \in \mathbb{R}^p$. For example, in model predictive control we may set $\boldsymbol{p} = \boldsymbol{x}_0$ to be the initial state. In problems where a target or cost function may be chosen by an operator, $\boldsymbol{p}$ can contain a parameterization of these choices.

A complete, globally optimal method for solving a parametric optimal control problem can be viewed as a mapping $\boldsymbol{g}$ from $\boldsymbol{p}$ to the optimal solution of the unknowns, denoted as $\boldsymbol{s}^*$:

$$\boldsymbol{g} : \boldsymbol{p} \to \boldsymbol{s}^\star \tag{12}$$

Of course, in nearly all practical problems, the solution $\boldsymbol{g}$ cannot be obtained analytically.

NNOC approximates this map using finite instances of problems that have been solved to global optima in a precomputation step. Assuming $\boldsymbol{p}$ 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 N parameter-solution pairs where $\boldsymbol{p}$ is the parameter and $\boldsymbol{s}^\star$ is the solution. Specifically, the database $D = \{(\boldsymbol{p}^{(i)}, \boldsymbol{s}^{\star(i)}) | i = 1, \ldots, N\}$ where we denote $(\boldsymbol{p}^{(i)}, \boldsymbol{s}^{\star(i)})$ 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.

3.3 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. For problems that obey the stationary property, such as our MPC-like formulations below, 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 the final time is a free variable, we set the final time for point i to be $t_0 + t_f - t_i$.

3.4 Function approximator

During a query, NNOC approximates $\boldsymbol{g} : \boldsymbol{p} \to \boldsymbol{s}^\star$ based on database D to generate initial guesses. This problem has been widely studied in statistical learning community and several models have been proposed as function approximator. Typical function approximators including linear regression, neural network, and Gaussian process assume continuity of the function and tend to perform worse in the presence of function discontinuity. However, for nonlinear optimal control problems the problem-solution mapping can be discontinuous due to the existence of multiple local optima. As a result, NNOC applies a k-Nearest Neighbor (k-NN) optimization model that determines multiple candidates for optimization, without performing undesirable interpolation between candidates.

The standard k-NN function approximator approximates an arbitrary function using a piecewise constant function:

$$\boldsymbol{g}(\boldsymbol{p}) \approx \frac{1}{k} \sum_{i \in N_k(\boldsymbol{p})} \boldsymbol{s}_i^\star \tag{13}$$

where $N_k(\boldsymbol{p})$ are the indices of the k nearest neighbors in D. But rather than averaging the neighbors as in standard function approximation, NNOC uses *all k nearest neighbors* as initial guesses for optimization. By doing so, we obtain multiple local minima $\boldsymbol{z}_i \leftarrow Opt(\boldsymbol{s}_i^\star, \boldsymbol{p})$ where $Opt(\boldsymbol{s}, \boldsymbol{p})$ uses the costate $\boldsymbol{s}$ as an initial seed for the TPBVP problem defined by $\boldsymbol{p}$. We then pick the best as follows:

$$\boldsymbol{g}(\boldsymbol{p}) \approx \arg \min_{\boldsymbol{z} \in \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_k\}} Cost(\boldsymbol{z}) \tag{14}$$

and returns the objective function value. We adopt the convention that $Cost(\boldsymbol{z}_i) = \infty$ if no feasible solution is found for the ith optimization. To make the nearest neighbors lookup fast, our implementation performs the NN query using the approximate nearest-neighbors library FLANN ①.

By returning multiple candidate function values, k-NN increases the probability of obtaining a global optimum. Other function approximators that only return one function value run a higher risk of obtaining a local optimum, and moreover can obtain poorer results by interpolating across discontinuities.

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

3.5 Sensitivity analysis

Assuming the mapping from parameters to solutions is continuous and differentiable, SA 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. Specifically, for a neighbor i of problem $\boldsymbol{p}$, we derive a better guess $\boldsymbol{s}$ via a first-order approximation

$$\boldsymbol{s} = \boldsymbol{s}_i^\star + S_i(\boldsymbol{p} - \boldsymbol{p}_i) \tag{15}$$

① http://www.cs.ubc.ca/research/flann/,retrieved Dec/25/2018.

in which the sensitivity matrix S_i links the variation of parameters and the variation of solutions. We derive the form of this matrix below.

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

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

where $\dfrac{\partial \boldsymbol{x}(t_f)}{\partial \boldsymbol{x}_0}$ and $\dfrac{\partial \boldsymbol{x}(t_f)}{\partial \boldsymbol{\lambda}_0}$ are easily obtained by integrating the variational equation of the system dynamics. We refer to Ref. [25] for further details.

Using Eq. (16) we can obtain a linear relationship between the change of initial state and the change of initial costate variables

$$\delta \boldsymbol{\lambda}_0 = \frac{\partial \boldsymbol{\lambda}_0}{\partial \boldsymbol{x}_0} \delta \boldsymbol{x}_0 = -\frac{\partial \boldsymbol{x}(t_f)}{\partial \boldsymbol{\lambda}_0}^{-1} \frac{\partial \boldsymbol{x}(t_f)}{\partial \boldsymbol{x}_0} \delta \boldsymbol{x}_0 \tag{17}$$

It should be noted that the matrix $\dfrac{\partial \boldsymbol{\lambda}_0}{\partial \boldsymbol{x}_0}$ 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 $\boldsymbol{p} = \boldsymbol{x}$ is matched to an example $\boldsymbol{p}_i = \boldsymbol{x}_0$, $\boldsymbol{s}_i^\star = \boldsymbol{\lambda}_0$ in the database, we seed the solver with the initial costate

$$\boldsymbol{\lambda} = \boldsymbol{\lambda}_0 + \frac{\partial \boldsymbol{\lambda}_0}{\partial \boldsymbol{x}_0} (\boldsymbol{x} - \boldsymbol{x}_0) \tag{18}$$

In other words, the sensitivity matrix is $S_i = \dfrac{\partial \boldsymbol{\lambda}_0}{\partial \boldsymbol{x}_0}(\boldsymbol{p}_i, \boldsymbol{s}_i^\star)$.

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 Eqs. (10) and (11), obtaining

$$\begin{cases} \dfrac{\partial \boldsymbol{x}(t_f)}{\partial \boldsymbol{x}_0} \delta \boldsymbol{x}_0 + \dfrac{\partial \boldsymbol{x}(t_f)}{\partial \boldsymbol{\lambda}_0} \delta \boldsymbol{\lambda}_0 + \dfrac{\partial \boldsymbol{x}(t_f)}{\partial t_f} \delta t_f = \boldsymbol{0} \\ \dfrac{\partial H(t_f)}{\partial \boldsymbol{x}_0} \delta \boldsymbol{x}_0 + \dfrac{\partial H(t_f)}{\partial \boldsymbol{\lambda}_0} \delta \boldsymbol{\lambda}_0 + \dfrac{\partial H(t_f)}{\partial t_f} \delta t_f = 0 \end{cases} \tag{19}$$

Here, $\dfrac{\partial \boldsymbol{x}(t_f)}{\partial \boldsymbol{x}_0}$ and $\dfrac{\partial \boldsymbol{x}(t_f)}{\partial \boldsymbol{\lambda}_0}$ are obtained as before; $\dfrac{\partial \boldsymbol{x}(t_f)}{\partial t_f} = \dot{\boldsymbol{x}}(t_f)$ is obtained by substituting the optimal control into the dynamics function; $\dfrac{\partial H(t_f)}{\partial t_f} = 0$ since H does not depend on time [2]; $\dfrac{\partial H(t_f)}{\partial \boldsymbol{x}_0} = \dfrac{\partial H(t_0)}{\partial \boldsymbol{x}_0} = -\dot{\boldsymbol{\lambda}}(t_0)$; and $\dfrac{\partial H(t_f)}{\partial \boldsymbol{\lambda}_0} = \dfrac{\partial H(t_0)}{\partial \boldsymbol{\lambda}_0} = \dot{\boldsymbol{x}}(t_0)$. Then we can calculate $\delta \boldsymbol{\lambda}_0$ and δt_f by solving a system of $n+1$ linear

equations:

$$\begin{bmatrix} \delta\boldsymbol{\lambda}_0 \\ \delta t_f \end{bmatrix} = -\begin{bmatrix} \dfrac{\partial \boldsymbol{x}(t_f)}{\partial \boldsymbol{\lambda}_0} & \dfrac{\partial \boldsymbol{x}(t_f)}{\partial t_f} \\ \dfrac{\partial H(t_f)}{\partial \boldsymbol{\lambda}_0} & \dfrac{\partial H(t_f)}{\partial t_f} \end{bmatrix}^{-1} \begin{bmatrix} \dfrac{\partial \boldsymbol{x}(t_f)}{\partial \boldsymbol{x}_0} \\ \dfrac{\partial H(t_f)}{\partial \boldsymbol{x}_0} \end{bmatrix} \delta \boldsymbol{x}_0 \tag{20}$$

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

3.6 Summary

The overall NNOC procedure is listed below.

Precomputation phase

1) Determine a parameterization $\boldsymbol{p}$ of the set of problems that will be encountered during the query phase.

2) Determine a representative distribution over problem parameters.

3) Sample N/M problem parameters, and use a global optimization such as random-restarts to solve them.

4) Apply the method of Section 3.3 to extend the database by a factor of M examples.

5) Store the database D and precompute data structures for nearest-neighbor lookup.

Query phase

1) For a new problem $\boldsymbol{p}$, determine the k nearest neighbor examples $(\boldsymbol{p}_i, \boldsymbol{s}_i^\star)$.

2) Optionally (NNOC+SA), for each of k examples, use sensitivity analysis (Section 3.5) to determine a seed $\boldsymbol{s}_i$.

3) For each of k examples, run a local optimization $\boldsymbol{z}_i = Opt(\boldsymbol{s}_i, \boldsymbol{p})$. Return the one with minimum cost.

Several hyperparameters affect performance, including:

(1) Database size N. If N is too small, the distance between new parameters and the 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 naive 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.

4 Numerical examples

We test NNOC in three problems:

- Planar Car: a minimum effort problem on a second-order Dubins car.
- Quadcopter: a minimum-time problem on a dynamic quadcopter model.
- Satellite: an agile satellite rest-to-rest reorientation problem.

In each case we seed the database using randomly sampled initial states, solved using a random restart method. The test set is randomly generated from the same distribution as the training set. We study the effects of parameters such as the database size, stopping criteria for random restart, database size, number of neighbors queried, and whether SA is used.

4.1 Planar Car

We consider a simplified planar car with the following dynamics [26]:

$$\dot{x} = v\sin\theta,\ \dot{y} = v\cos\theta,\ \dot{\theta} = u_\theta v,\ \dot{v} = u_v \tag{21}$$

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

4.1.1 Optimal control formulation

The performance index is given by the quadratic control effort

$$J = \int_{t_0}^{t_f} \boldsymbol{u}^{\mathrm{T}} \boldsymbol{R} \boldsymbol{u} \mathrm{d}t \tag{22}$$

where $\boldsymbol{R}$ is a diagonal matrix with entries $r_1 = 0.2$ and $r_2 = 0.1$, as chosen in accordance with Ref. [26].

We arbitrarily choose fixed initial and final time, 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 [2]

$$\begin{aligned} H &= \boldsymbol{\lambda}^{\mathrm{T}}\dot{\boldsymbol{x}} + \boldsymbol{u}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{u} \\ &= \lambda_x v\sin\theta + \lambda_y v\cos\theta + \lambda_\theta v u_\theta + \lambda_v u_v + r_1 u_\theta^2 + r_2 u_v^2 \end{aligned} \tag{23}$$

and the Euler–Lagrange equations

$$\begin{cases} \dot{\lambda}_x = -\dfrac{\partial H}{\partial x} = 0 \\ \dot{\lambda}_y = -\dfrac{\partial H}{\partial y} = 0 \\ \dot{\lambda}_\theta = -\dfrac{\partial H}{\partial \theta} = -\lambda_x v\cos\theta + \lambda_y v\sin\theta \\ \dot{\lambda}_v = -\dfrac{\partial H}{\partial v} = -\lambda_x\sin\theta - \lambda_y\cos\theta - \lambda_\theta u_\theta \end{cases} \tag{24}$$

Then we readily derive the optimal control which minimizes H as

$$u_\theta^* = -\frac{v\lambda_\theta}{2r_1},\ u_v^* = -\frac{\lambda_v}{2r_2} \tag{25}$$

4.1.2 Estimation of costate magnitudes

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 [3] 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

$$\begin{cases} \bar{s} = 2\sqrt{x^2+y^2} \\ \bar{v} = \dfrac{\bar{s}}{t} \\ \bar{a} = \dfrac{4\bar{v}}{t} \\ \bar{\lambda}_v = 2\bar{a}r_2 \end{cases} \tag{26}$$

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

$$\bar{\lambda}_\theta = \frac{16|\theta_0|r_1}{t^2\bar{v}^2}. \tag{27}$$

It should be noted that an additional term $\bar{v}^2$ is multiplied in the denominator because the additional multiplication of v in Eqs. (25) and (21).

4.1.3 Simulation result

We evaluate four methods that differ in how initial guesses are provided and how the iteration is terminated.

(1) RR: Random Restart using k restarts, where $k = 5, 10, 50, 100$.

(2) RL: Random restart, terminating after k Locally optimal solutions are solved, where $k = 1, 3, 5, 10$.

(3) NNOC: our method with k neighbors and without sensitivity analysis, where $k = 1, 5, 10$.

(4) NNOC+SA: our method with k neighbors and with sensitivity analysis, where $k = 1, 5, 10$.

In RL, the maximum number of restarts is limited to 100 in order to avoid an infinite loop. For random restart techniques, the normalization of initial costates is used [3].

To build the database, we randomly generate $N =$ 20,000 initial states and calculate the corresponding costates. No database extension is performed ($M = 1$). Then 5 databases having size 1,250, 2,500, 5,000, 10,000, 20,000 are constructed. In the database construction phase, it is important to make sure the solutions are indeed global optimal. We adopt a random restart technique with 100 initial guesses. As experiments will later show in Fig. 4, such a choice does achieve about 95% global optimal rate. To further reduce the likelihood of the database containing local optima, for each example we use NNOC to adapt the solutions from its neighbors, and keep the optimized solution with lowest cost. This also suggests an incremental construction technique, whereby random restarts is used to solve a subset of examples and then NNOC is used to propagate to other examples. Building the database in this manner takes a few hours on a single core of a CPU.

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. In the presented results, the *Global Optimum Rate* denotes the fraction of times a method obtains a globally optimal solution; and *Avg. Time* is the average computation time for solving the TPBVP using shooting methods. To calculate global optimum 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

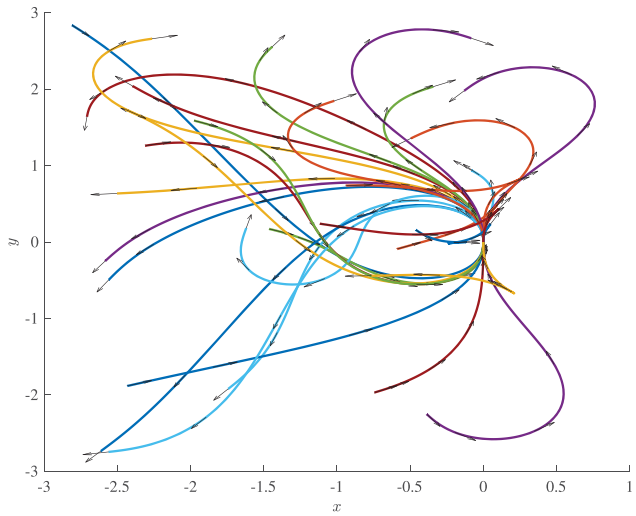


Fig. 1 Sample of optimal trajectories for the Planar Car example. The arrow shows the initial angle.

if its cost is close to the minimum cost (the difference is less than 1×10^{-3}).

The results of average computation time and global optimum rate are shown in Figs. 2 and 3 for NNOC with different choice of hyperparameters including database size N, neighbor count k, and whether SA is used.

The average computation time decreases with increasing N since larger N indicates closer distance from novel problem to its nearest neighbors. 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. Larger k causes more trial of initial guess and increases average computation

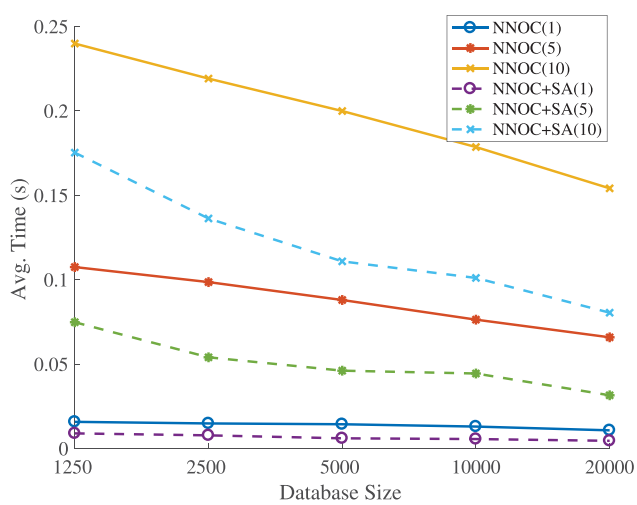


Fig. 2 Average computation time for the Planar Car example.

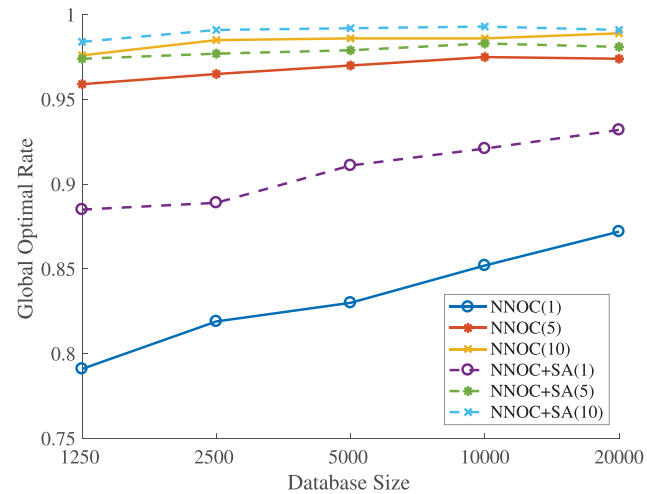


Fig. 3 global optimum rate of for Planar Car example.

time. SA significantly reduces the computation time since the linear approximation leads to a guess closer to the optimum.

For global optimum rate, similar trends are observed. With larger N and k the global optimum rate for NNOC related methods also increases. Furthermore, SA also helps to improve it.

To compare between random restart technique and NNOC, the trade-off between computation time and solution optimality is shown in Fig. 4. We note that in this figure, the results for NNOC+SA use the largest database size. For random restart techniques, the

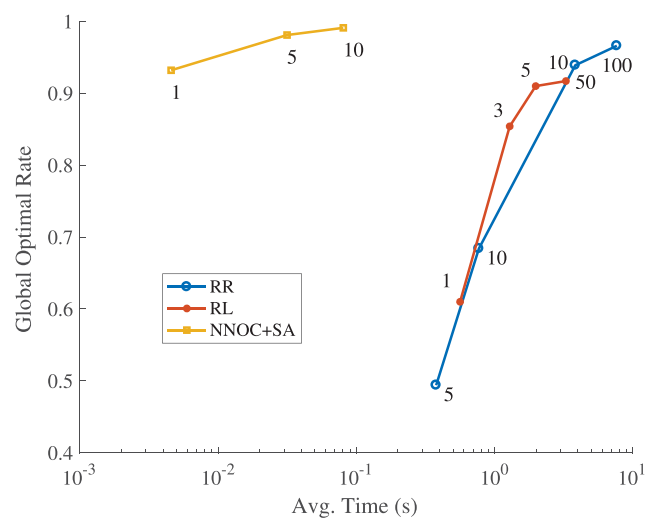


Fig. 4 Trade-off between average computation time and global optimum rate for between random restart and NNOC+SA with different choice of initial guesses. The upper left corner is ideal. To achieve the same level of global optimum rate, NNOC+SA is much faster than random restart.

computation time increases as the stopping criteria increases. With increasing stopping criteria for random restart methods, the global optimum rate increases, too. We note that for this problem with many local optima, a random restart technique has pretty low global optimum rate unless many initial guesses are tried, while NNOC with only 1 trial outperforms random restart with more than 10 initial guesses. To achieve the same global optimum rate with NNOC+SA with 1 guess, random restart techniques cost two orders of magnitude more time.

4.2 Planar Quadcopter

The Quadcopter example defines the following dynamics [27]:

$$\ddot{x} = \frac{F_T}{m}\sin\theta,\ \ddot{z} = \frac{F_T}{m}\cos\theta - g,\ \dot{\theta} = \omega \tag{28}$$

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 $\boldsymbol{x} = [x, z, \dot{x}, \dot{z}, \theta]$ includes 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:

$$\underline{u} \leqslant u \leqslant \overline{u},\ |\omega| \leqslant \overline{\omega} \tag{29}$$

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.

4.2.1 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

$$J = \int_0^{t_f} 1\mathrm{d}t \tag{30}$$

where t_f is a free variable. However, the resulting optimal control is bang-bang control which is numerically challenging to solve [27]. Hence, we use an alternate formulation that adds regularization parameters to the performance index so the resulting optimal control is continuous [28]. We introduce a logarithmic barrier function to the performance index, which is a widely-used method in the field of aerospace engineering [6, 29], as follows:

$$\begin{aligned} J &= \int_0^{t_f} L\,\mathrm{d}t \\ &= \int_0^{t_f} 1 - \epsilon_1 \ln[\hat{u}(1-\hat{u})] - \epsilon_2 \ln[\hat{\omega}(1-\hat{\omega})]\,\mathrm{d}t \end{aligned} \tag{31}$$

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 $\epsilon_1 = \epsilon_2 = 1$ which are relatively large compared with Ref. [29], 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

$$H = \lambda_x\dot{x} + \lambda_z\dot{z} + \lambda_{\dot{x}}u\sin\theta + \lambda_{\dot{z}}(u\cos\theta - g) + \lambda_\theta\omega + L \tag{32}$$

and derive the Euler–Lagrange equations:

$$\begin{cases} \dot{\lambda}_x = -\dfrac{\partial H}{\partial x} = 0,\ \dot{\lambda}_y = -\dfrac{\partial H}{\partial x} = 0 \\ \dot{\lambda}_{\dot{x}} = -\dfrac{\partial H}{\partial \dot{x}} = -\lambda_x,\ \dot{\lambda}_{\dot{z}} = -\dfrac{\partial H}{\partial \dot{z}} = -\lambda_z \\ \dot{\lambda}_\theta = -\dfrac{\partial H}{\partial \theta} = \lambda_{\dot{z}}u\sin\theta - \lambda_{\dot{x}}u\cos\theta \end{cases} \tag{33}$$

Then the non-dimensionalized optimal control that minimizes H is

$$\begin{cases} u^* = \dfrac{2\epsilon_1}{2\epsilon_1 + \rho_1 + \sqrt{4\epsilon_1^2 + \rho_1^2}} \\ \omega^* = \dfrac{2\epsilon_2}{2\epsilon_2 + \rho_2 + \sqrt{4\epsilon_2^2 + \rho_2^2}} \end{cases} \tag{34}$$

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

$$\rho_1 = (\overline{u} - \underline{u})(\lambda_{\dot{x}}\sin\theta + \lambda_{\dot{z}}\cos\theta),\ \rho_2 = (\overline{\omega} - \underline{\omega})\lambda_\theta \tag{35}$$

4.2.2 Simulation result

To build the database, we randomly generate 2,000 initial states and calculate the corresponding costates. In Fig. 5 we plot the optimal trajectories of 50 examples. Then the database extension technique of Section 3.3 is applied using $M = 100$ to get 200,000 state-costate

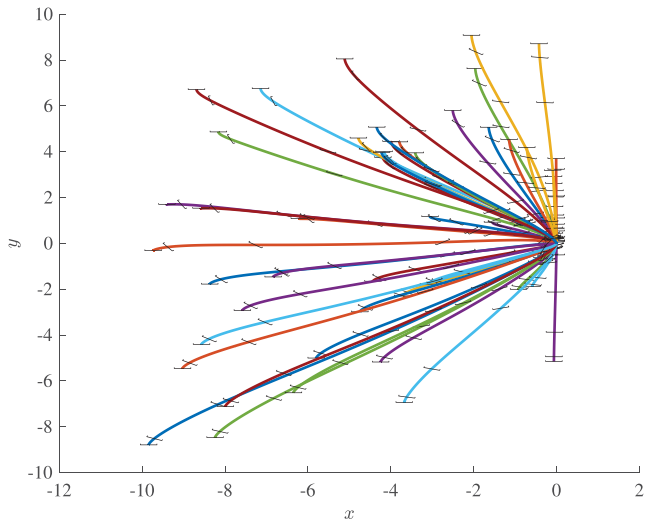


Fig. 5 Sample optimal trajectories for the Quadcopter example. The short line shows orientation of the quadcopter.

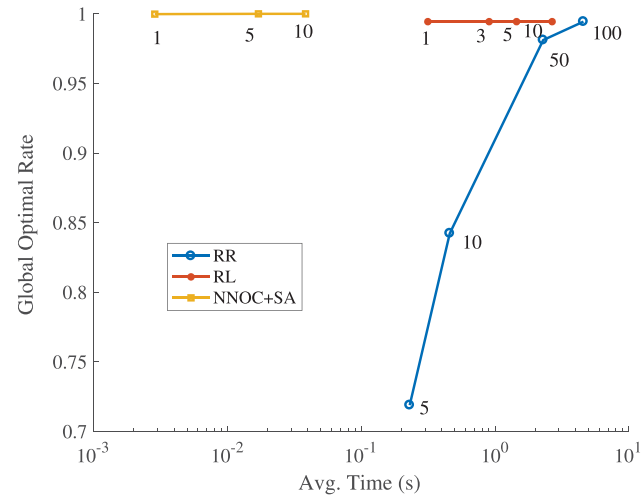


Fig. 7 Trade-off between average computation time and global optimum rate for the Quadcopter problem. A detailed description is in Fig. 4.

pairs. Databases of size 25,000, 50,000, 100,000, and 200,000 are constructed. As for the test set, 500 samples of initial states are randomly generated. The database extension technique is also used to extend the test set by sampling 10 states along each trajectory. As a result, the test set has 5,000 samples.

Figure 6 shows solving time for NNOC for varying parameters. The trends agree with the planar car problem. Figure 7 shows the trade-off between computation time and global optimum rate. A comparison between random restart and NNOC is presented. Observe that for this problem, the difficulty is convergence to a solution, not to a local optimal solution. The low global optimum rate for RR is they fail to find a solution within the given number of trials, not the sub-optimality of the found solutions. In practice, with 100 random initial guesses, the success rate is 99.4%. Since existence of multiple local optima is not an issue, the global optimum rate of RL is determined by if convergence can be achieved within the maximum trials numbers, which is chosen to be the same with maximum number of initial guesses for RR. As the result, RL has the same global optimum rate with whatever stopping criteria. For problems without multiple local optima, RL is surly a better choice than RR since returning the first found solution turns out to be sufficient. However, in general this property is unknown in advance. Besides, for problems with multiple local optima, such as the planar car problem, RL tends to achieve lower global optimum rate, as Fig. 7 shows, unless the same amount of initial guesses are used. However, random restart techniques still cost much higher computation time than NNOC+SA which is less than 4 ms with $k = 1$ and achieves 99.9% global optimum rate.

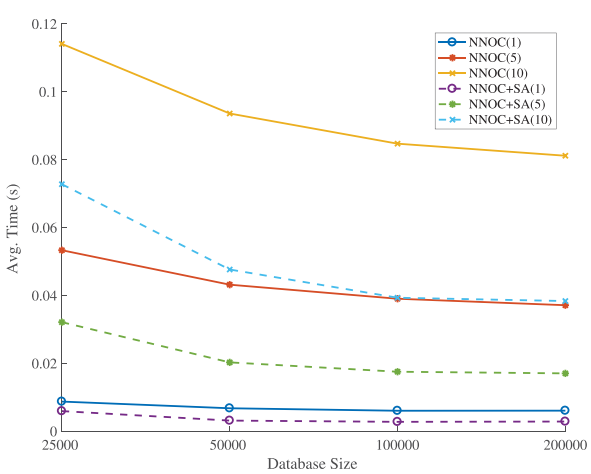


Fig. 6 Average computation time of NNOC for the Quadcopter problem.

4.3 Agile satellite reorientation

A satellite is modeled as a rigid body under torque control and without perturbation. The system dynamics are

$$\boldsymbol{J}\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times \boldsymbol{J}\boldsymbol{\omega} = M\boldsymbol{u} \tag{36}$$

where $\boldsymbol{\omega} = [\omega_1, \omega_2, \omega_3]^{\mathrm{T}}$ is the the vector of angular velocities; "×" is the cross product operator; $\boldsymbol{J} = \mathrm{diag}([J_1, J_2, J_3])$ is the inertia matrix (along the principal axes so it is diagonal); M is the maximum

torque magnitude; and $\boldsymbol{u} = [u_1, u_2, u_3]^{\mathrm{T}}$ denotes the vector of the torque directions. It should be noted that here the subscript 1, 2, 3 denotes the component along x, y, z axis, respectively. We assume that the maximum torque magnitudes along all axes are the same so the constraint

$$-\mathbf{1} \leqslant \boldsymbol{u} \leqslant \mathbf{1} \tag{37}$$

where the inequality denotes element-wise inequality. It should be noted that Eq. (36) can be expanded into matrix-vector form by expressing the cross product as the multiplication of a skew-symmetric matrix and a vector, i.e.,

$$\dot{\boldsymbol{\omega}} = -\boldsymbol{J}^{-1}[\boldsymbol{\omega}]_{\times}\boldsymbol{J}\boldsymbol{\omega} + \boldsymbol{J}^{-1}M\boldsymbol{u} \tag{38}$$

where

$$[\boldsymbol{\omega}]_{\times} \stackrel{\text{def}}{=} \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \tag{39}$$

In order to describe the attitude of the satellite, a variety of attitude coordinates have been developed each of which has its advantages and weakness [30]. We use the modified Rodrigues parameters (MRPs) defined as

$$\boldsymbol{\sigma} = \boldsymbol{e}\tan(\Phi/4) \tag{40}$$

where $\boldsymbol{e}$ and Φ are the eigenaxis and rotation angle. Although MRP is a minimal parameterization, it has singularity at $\Phi = 2\pi$. However, in this paper we only consider rotation within one revolution and by choosing the eigenaxis direction properly we constrain the rotation angle to be within $[0, \pi]$.

From Ref. [30] the differential equation of MRPs is derived as

$$\dot{\boldsymbol{\sigma}} = \frac{1}{4}\boldsymbol{B}\boldsymbol{\omega} \tag{41}$$

where the matrix $\boldsymbol{B}$ is defined as

$$\boldsymbol{B} = \begin{bmatrix} 1-\sigma^2+2\sigma_1^2 & 2(\sigma_1\sigma_2-\sigma_3) & 2(\sigma_1\sigma_3+\sigma_2) \\ 2(\sigma_1\sigma_2+\sigma_3) & 1-\sigma^2+2\sigma_2^2 & 2(\sigma_2\sigma_3-2\sigma_1) \\ 2(\sigma_1\sigma_3-\sigma_2) & 2(\sigma_2\sigma_3+\sigma_1) & 1-\sigma^2+2\sigma_3^2 \end{bmatrix} \tag{42}$$

where $\sigma_i (i = 1, 2, 3)$ denotes the three components of MRP and $\sigma^2 \equiv \sigma_1^2 + \sigma_2^2 + \sigma_3^2$.

The overall 6D state consists of $(\boldsymbol{\sigma}, \boldsymbol{\omega})$ and the system dynamics consist of Eqs. (41) and (38).

4.3.1 Optimal control formulation

We want to solve the time-optimal problem and use homotopic approach to bypass the issue of control discontinuity. The cost function is defined as

$$J = \int_{t_0}^{t_f} 1 - \sum_{i=1}^{3} \varepsilon \ln[(u_i - 1)(1 - u_i)]\mathrm{d}t \tag{43}$$

where ε controls the magnitude of the perturbation. By introducing costate variables $\boldsymbol{\lambda}_\sigma$ and $\boldsymbol{\lambda}_\omega$, the Hamiltonian is defined as

$$H = 1 - \sum_{i=1}^{3} \varepsilon \ln[(u_i - 1)(1 - u_i)] + \boldsymbol{\lambda}_\sigma^{\mathrm{T}}\dot{\boldsymbol{\sigma}} + \boldsymbol{\lambda}_\omega^{\mathrm{T}}\dot{\boldsymbol{\omega}} \tag{44}$$

and the terms that depends on $u_i (i = 1, 2, 3)$ are

$$-\varepsilon \ln[(u_i - 1)(1 - u_i)] + \frac{\lambda_{\omega i} M}{J_i} u_i \tag{45}$$

and the optimal control can be calculated by solving the u_i such that the derivative of Eq. (45) with respect to u_i is zero, i.e.,

$$u_i = \frac{-\rho_i}{\varepsilon + \sqrt{\varepsilon^2 + \rho_i^2}} \tag{46}$$

where

$$\rho_i = \frac{\lambda_{\omega i} M}{J_i} \tag{47}$$

The Euler–Lagrange equations are derived by Eq. (6) and found in Refs. [31, 32].

For the rest-to-rest time-optimal problem, the initial and final orientation of the satellite, denoted as $\boldsymbol{\sigma}_0$ and $\boldsymbol{\sigma}_f$, respectively, are already known. Specifically, any reorientation problem between two orientations is essentially equivalent to the reorientation from $\boldsymbol{\sigma}_0 = \mathbf{0}$ to another $\boldsymbol{\sigma}'_f$. From now on we assume $\boldsymbol{\sigma}_f$ has already been converted for simplicity. The TPBVP is built and $\boldsymbol{s} \equiv [\boldsymbol{\lambda}_\sigma(t_0), \boldsymbol{\lambda}_\omega(t_0), t_f]$ is solved to satisfy boundary conditions:

$$\boldsymbol{\sigma}(t_f) = \boldsymbol{\sigma}_f, \quad \boldsymbol{\omega}(t_f) = \mathbf{0} \tag{48}$$

and

$$H(t_f) = 0 \tag{49}$$

4.3.2 Simulation result

For simplicity, a symmetric satellite with $J = \mathrm{diag}([1, 1, 1])$ with maximum torque magnitude $M = 1$ is considered. The parameters are the target orientations which are sampled and the corresponding solutions are computed. In order to generate a uniform distribution in $SO(3)$, the method in Ref. [33] is used and the corresponding program ① is used. 10,000 samples

① http://msl.cs.uiuc.edu/~yershova/software/so3sampling/so3sampling.htm, retrieved Oct/16/2018.

of orientations are generated. For each sample, the time-optimal control problem with homotopic approach ($\varepsilon = 0.01$) is solved using a random restart technique with a maximum iteration number of 100. The normalization of initial costate variables [3] is used and the initial guesses of costate variables are generated randomly. Numerical results show that a corresponding solution is found for every sample. In order to see how the database size N affects the computation time for novel problems, the first 1,250, 2,500, 5,000 and all the 10,000 samples are used to build databases, respectively. We show in Fig. 8 three trajectories of reorientation. It should be noted that these results are obtained with a homotopic approach, so the optimal reorientation time is a little lower than the ones obtained here.

Figure 9 compares the running time of NNOC for different database size N, neighbor count k, and whether SA is enabled. The trends agree with previous two benchmark problems except that SA does not decrease average computation time when dataset size N is small. As N increases, the improvement by SA also increases. The reason is that SA is based on linearization and is only locally valid. As a result, such linearization causes larger error as difference in $\boldsymbol{p}$ increases, caused by increasing k. Again, for this problem the global optimum rate is not a serious issue and the comparison is thus omitted.

The comparison between random restart and NNOC is shown in Fig. 10. Compared to NNOC+SA with 1 initial guess, random restarts take one to two orders of magnitude more time to obtain the same global optimum rate.

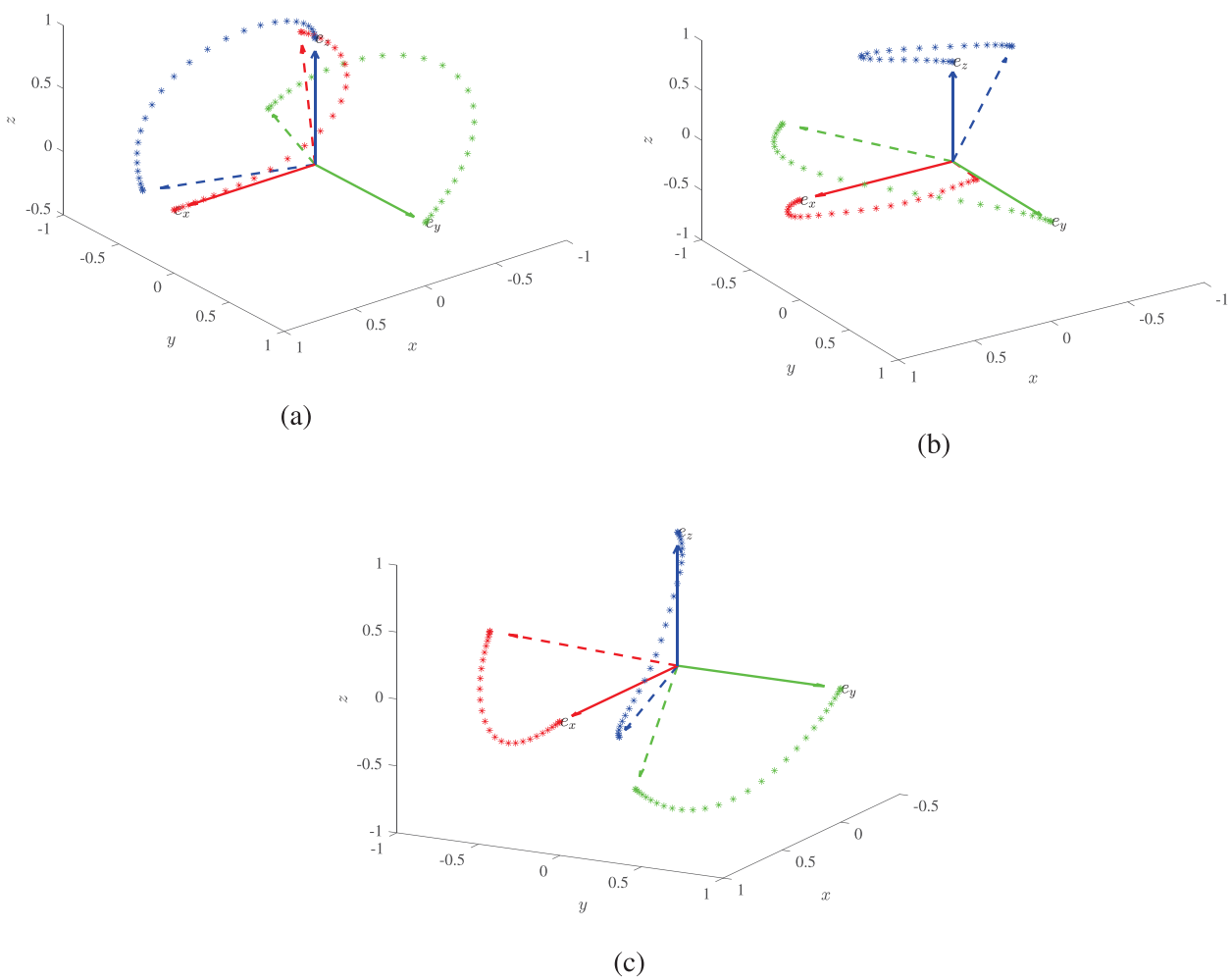


Fig. 8 Three example trajectories of reorientation. The red, green, blue axis denotes the x, y, z axis, respectively. The solid and dashed lines show the initial and final axes. The stars show the trace of the tails of the unit vectors.

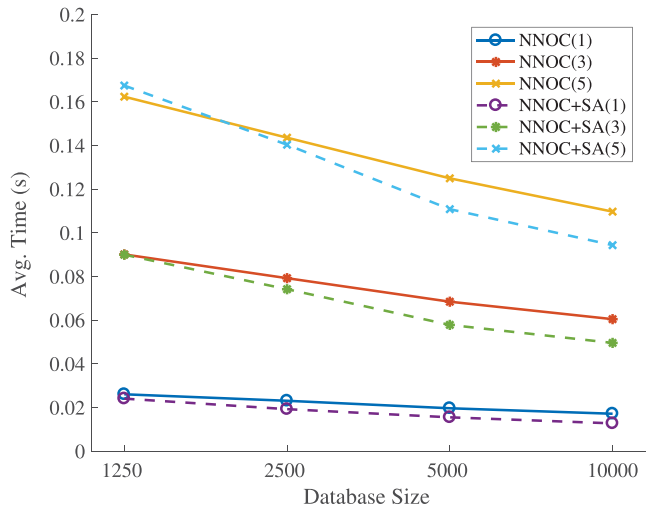


Fig. 9 Average computation time of NNOC for random satellite reorientations.

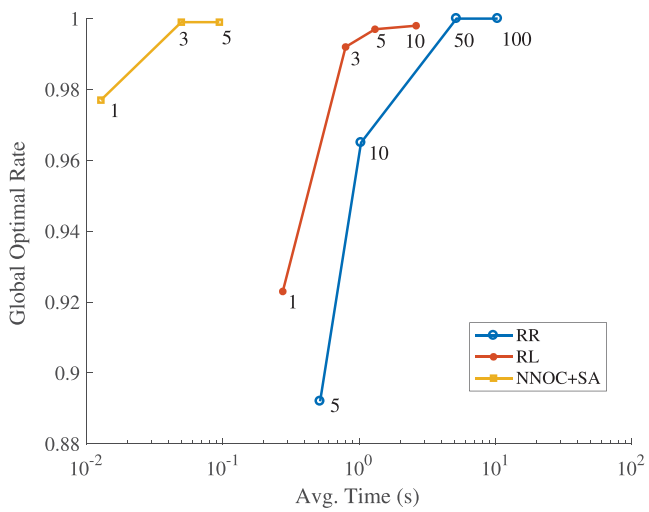


Fig. 10 Trade-off between computation time and global optimum rate for random satellite reorientations.

5 Conclusions

In this paper a data-driven technique is proposed to help solve nonlinear optimal control problems quickly and reliably. 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. In some applications it may be challenging to define the problem parameterization, such as obstacles observed through sensor data, and one approach may be to compute a feature mapping from sensor information directly. Future work should also address scalability to higher-dimensional systems as well as state and model uncertainty.

Acknowledgements

This work was partially supported by NSF (Grant No. IIS-1816540).

References

[1] Betts, J. T. Survey of numerical methods for trajectory optimization. *Journal of Guidance, Control, and Dynamics*, **1998**, 21(2): 193–207.

[2] Bryson Jr, A. E., Ho Y. C. *Applied Optimal Control: Optimization, Estimation and Control*. CRC Press, **1975**.

[3] Jiang, F. H., Baoyin, H. X., Li, J. F. Practical techniques for low-thrust trajectory optimization with homotopic approach. *Journal of Guidance, Control, and Dynamics*, **2012**, 35(1): 245–258.

[4] Jetchev, N., Toussaint, M. Fast motion planning from experience: trajectory prediction for speeding up movement generation. *Autonomous Robots*, **2013**, 34(1–2): 111–127.

[5] Hauser, K. Learning the problem-optimum map: analysis and application to global optimization in robotics. *IEEE Transactions on Robotics*, **2017**, 33(1): 141–152.

[6] Bertrand, R., Epenoy, R. New smoothing techniques for solving bang–bang optimal control problems—Numerical results and statistical interpretation. *Optimal Control Applications and Methods*, **2002**, 23(4): 171–197.

[7] Russell, R. P. Primer vector theory applied to global low-thrust trade studies. *Journal of Guidance, Control, and Dynamics*, **2007**, 30(2): 460–472.

[8] Tang, G., Jiang, F. H., Li, J. F. Fuel-optimal low-thrust trajectory optimization using indirect method and successive convex programming. *IEEE Transactions on Aerospace and Electronic Systems*, **2018**, 54(4): 2053–2066.

[9] Jiang, F. H., Tang, G., Li, J. F. Improving low-thrust trajectory optimization by adjoint estimation with shape-based path. *Journal of Guidance, Control, and Dynamics*, **2017**, 40(12): 3282–3289.

[10] Cassioli, A., Di Lorenzo, D., Locatelli, M., Schoen, F., Sciandrone, M. Machine learning for global optimization. *Computational Optimization and Applications*, **2012**, 51(1): 279–303.

[11] Pan, J., Chen, Z., Abbeel, P. Predicting initialization effectiveness for trajectory optimization. In: Proceedings of 2014 IEEE International Conference on Robotics and Automation, **2014**, 5183–5190.

[12] Bohg, J., Morales, A., Asfour, T., Kragic, D. Data-driven grasp synthesis—A survey. *IEEE Transactions on Robotics*, **2014**, 30(2): 289–309.

[13] Lampariello, R., Nguyen-Tuong, D., Castellini, C., Hirzinger, G., Peters, J. Trajectory planning for optimal robot catching in real-time. In: Proceedings of 2011 IEEE International Conference on Robotics and Automation, **2011**, 3719–3726.

[14] Sutton, R. S., Barto, A. G. *Reinforcement Learning: An Introduction*. MIT Press, **1998**.

[15] Sánchez-Sánchez, C., Izzo, D. Real-time optimal control via Deep Neural Networks: study on landing problems. *Journal of Guidance, Control, and Dynamics*, **2018**, 41(5): 1122–1135.

[16] Tang, G., Sun, W. D., Hauser, K. Learning trajectories for real-time optimal control of quadrotors. In: Proceedings of 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, **2018**, 3620–3625.

[17] Tang, G., Hauser, K. Discontinuity-sensitive optimal control learning by mixture of experts. *arXiv preprint* arXiv:1803.02493, **2018**.

[18] Bemporad, A., Morari, M., Dua, V., Pistikopoulos, E. N. The explicit solution of model predictive control via multiparametric quadratic programming. In: Proceedings of 2000 American Control Conference, **2000**, 872–876.

[19] Furfaro, R., Bloise, I., Orlandelli, M., Di Lizia, P., Topputo, F., Linares, R. Deep learning for autonomous lunar landing. In: Proceedings of 2018 AAS/AIAA Astrodynamics Specialist Conference, **2018**, 1–22.

[20] Ampatzis, C., Izzo, D. Machine learning techniques for approximation of objective functions in trajectory optimisation. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI) 2009, Workshop on Artificial Intelligence in Space, **2009**, 1–6.

[21] Mereta, A., Izzo, D., Wittig, A. Machine learning of optimal low-thrust transfers between near-earth objects. In: Proceedings of the 12th International Conference on Hybrid Artificial Intelligence Systems, **2017**, 543–553.

[22] Izzo, D., Sprague, C., Tailor, D. Machine learning and evolutionary techniques in interplanetary trajectory design. *arXiv preprint* arXiv:1802.00180, **2018**.

[23] Tang, G., Hauser, K. A data-driven indirect method for nonlinear optimal control. In: Proceedings of 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, **2017**, 4854–4861.

[24] Moré, J. J., Garbow, B. S., Hillstrom, K. E. User guide for MINPACK-1. Argonne National Laboratory Report ANL-80-74, Argonne National Laboratory, **1980**.

[25] Maurer, H., Augustin, D. Sensitivity analysis and real-time control of parametric optimal control problems using boundary value methods. *Online Optimization of Large Scale Systems*, **2001**, 17–55.

[26] Xie, Z. M., Liu, C. K., Hauser, K. K. Differential dynamic programming with nonlinear constraints. In: Proceedings of 2017 IEEE International Conference on Robotics and Automation, **2017**, 695–702.

[27] Ritz, R., Hehn, M., Lupashin, S., D'Andrea, R. Quadrocopter performance benchmarking using optimal control. In: Proceedings of 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, **2011**, 5179–5186.

[28] Tomić T., Maier, M., Haddadin, S. Learning quadrotor maneuvers from optimal control and generalizing in real-time. In: Proceedings of 2014 IEEE International Conference on Robotics and Automation, **2014**, 1747–1754.

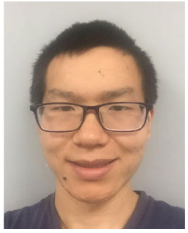
[29] Tang, G., Jiang, F. H. Capture of near-Earth objects with low-thrust propulsion and invariant manifolds. *Astrophysics and Space Science*, **2016**, 361(1): 10.

[30] Schaub, H., Junkins, J. L. *Analytical Mechanics of Space Systems*, 2nd edn. AIAA Education Series, **2009**.

[31] Li, J., Xi, X. N. Time-optimal reorientation of the rigid spacecraft using a pseudospectral method integrated homotopic approach. *Optimal Control Applications and Methods*, **2015**, 36(6): 889–918.

[32] Bai, X. L., Junkins, J. L. New results for time-optimal three-axis reorientation of a rigid spacecraft. *Journal of Guidance, Control, and Dynamics*, **2009**, 32(4): 1071–1076.

[33] Yershova, A., Jain, S., Lavalle, S. M., Mitchell, J. C. Generating uniform incremental grids on $SO(3)$ using the Hopf fibration. *The International Journal of Robotics Research*, **2010**, 29(7): 801–812.



Gao Tang is a Ph.D. student in the Department of Mechanical Engineering and Materials Science at Duke University. He received his B.S. and M.S. degrees in the Department of Aerospace Engineering at Tsinghua University. He is interested in trajectory optimization with an ambitious goal to achieve them in real time. E-mail: gao.tang@duke.edu.

Kris Hauser is an associate professor at Duke University with joint appointments at the Departments of Electrical & Computer Engineering and Mechanical Engineering and Materials Science. He received his Ph.D. degree in computer science from Stanford University in 2008, B.S. degrees in computer science and mathematics from UC Berkeley in 2003, and was a postdoc at UC Berkeley. He joined the faculty of Indiana University from 2009 to 2014, where he started the Intelligent Motion Lab, and began his current position at Duke in 2014. He is a recipient of a Stanford Graduate Fellowship, Siebel Scholar Fellowship, Best Paper Award at IEEE Humanoids 2015, and an NSF CAREER Award.