RRT-CoLearn: Towards Kinodynamic Planning Without Numerical Trajectory Optimization

Wouter J. Wolfslag, Mukunda Bharatheesha, Thomas M. Moerland, and Martijn Wisse

***Abstract*—Sampling-based kinodynamic planners, such as rapidly-exploring random trees (RRTs), pose two fundamental challenges: computing a reliable (pseudo-)metric for the distance between two random nodes, and computing a steering input to connect the nodes. The core of these challenges is a two point boundary value problem, known to be NP-hard. Recently, the distance metric has been approximated using supervised learning, reducing computation time drastically. Previous work on such learning RRTs use direct optimal control to generate the data for supervised learning. This letter proposes to use indirect optimal control instead, because it provides two benefits: it reduces the computational effort to generate the data, and it provides a low-dimensional parametrization of the action space. The latter allows us to learn both the distance metric and the steering input. This eliminates the need for a local planner in learning RRTs. Experimental results on a pendulum swing up show tenfold speed-up in both the offline data generation and the online planning time.**

***Index Terms*—Motion and path planning, optimization and optimal control, learning and adaptive systems.**

I. Introduction

FOR motion planning of robotic manipulators, kinodynamic planning and sampling-based planning are getting increasingly popular. Kinodynamic planning, i.e., planning in state space rather than configuration space, improves robustness, speed and energy efficiency of robots [1]–[3]. Combining configuration-space planning with post-processing in state-space is often successful [4], but cannot solve all challenging dynamical problems. Sampling based planning has been shown to be the most viable way to handle high dimensional spaces and obstacles [5], [6]. Therefore, this letter will consider how to apply Rapidly-exploring Random Trees (RRTs) [6], the most popular sampling-based single-query planning algorithm, directly to state space planning for deterministic, fully modeled systems.

Manuscript received September 8, 2017; accepted January 14, 2018. Date of publication February 2, 2018; date of current version February 27, 2018. This letter was recommended for publication by Associate Editor J.-M. Lien and Editor N. Amato upon evaluation of the reviewers' comments. This work is part of the research program STW, which is (partly) funded by the Netherlands Organization for Scientific Research. This work is also part of the European Commission's Seventh Framework Programme (FP7/2007-2013) under Grant Agreement 609206. *(Corresponding author: Wouter J. Wolfslag.)*

W. J. Wolfslag, M. Bharatheesha, and M. Wisse are with the Faculty of Mechanical, Maritime and Materials Engineering (3ME), Delft University of Technology, Delft 2628 CD, The Netherlands (e-mail: w.j.wolfslag@tudelft.nl; m.bharatheesha@tudelft.nl; m.wisse@tudelft.nl).

T. M. Moerland is with the Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, Delft 2628 CD, The Netherlands (e-mail: t.m.moerland@tudelft.nl).

Digital Object Identifier 10.1109/LRA.2018.2801470

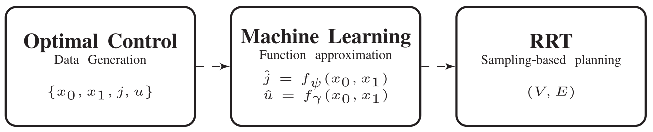

Fig. 1. Schematic of the Learning-RRT architecture. First, optimal control generates a dataset that specifies the local steering cost j and control input u for a given start state x_0 and end state x_1. Subsequently, machine learning predicts the steering cost and input given a start and goal node. These first two (computationally intensive) phases happen *offline*. Subsequently, the RRT handles *online* planning. The function approximators provide the RRT with quick cost and input prediction without online optimization.

RRT builds a tree graph structure with the states of the system as nodes and the trajectories of the system between two states as edges. The algorithm selects a node to expand from based on a *distance* to a randomly sampled node in the state space, i.e., it selects the *nearest* node in the current tree. That node is then expanded by means of a local planner that aims to reach the randomly-sampled node. These steps happen online, so computation time is crucial. Unfortunately, in state-space, both local planning and distance computation are computationally expensive [6].

In literature, the main approach to reduce the computational burden is to approximate the true distance function by a heuristic [7]–[10]. Two frequently used heuristics are the Euclidean distance and the optimal steering cost for a linear approximation to the system. The convergence of RRT variants using the Euclidean distance heuristic, and random steering inputs, was extensively analyzed in [11]. For promising results for the linearizing heuristic see [12]–[14]. However, these heuristics only minimally utilize the dynamical properties of the system, and therefore typically require more nodes to solve a given problem using RRT.

Recent literature proposes a promising different approach, which we call Learning-RRT [15]–[17]. Learning-RRT involves an offline machine learning phase that learns the distance and steering function in RRT (Fig. 1). An optimal control algorithm provides a database of optimal trajectories, which is the input for a supervised learning algorithm. This algorithm learns to approximate the functions the RRT requires. Note that trajectories found by RRTs are in general not optimal, even if the segments in the database are. Using optimal segments in the database provides a meaningful steering-cost metric and similarly shaped trajectories for connecting similar points in state space, which helps the learning algorithm. Supervised learning provides two benefits: 1) generalization over state-space and

2) fast online predictions. Thereby, the computations for trajectory optimization does not have to be repeated for new situations, and is shifted offline.

This letter proposes a method that helps to overcome the two remaining challenges of Learning-RRT:

1) **Local planning**: In literature on learning-RRT [15], [16], only the distance function is approximated by machine learning. Supervised learning of the steering function is hard due to the large number of parameters typically required to describe optimal input signals. Therefore, previous Learning-RRTs resort to computationally expensive optimizations for their steering function [15].
2) **Dataset generation**: The dataset for the supervised learning algorithm consists of many optimal trajectories. As optimizing a single trajectory already is a significant computational burden, generating a full dataset is very computationally demanding.

The main contribution of this letter is the use of indirect optimal control to generate the dataset. This allows us to solve both the problems mentioned above, thereby decreasing the computational burden by up to two orders of magnitude, both in the offline and the online phase of the learning RRT planning. However, the dataset generated by this method contains a bias, which is problematic for the learning algorithm. It turns out we can efficiently remove this bias through a simple dataset cleaning algorithm.

The focus of this letter is the introduced method and its implementation details. Our experiments provide proof of concept by evaluating our method on a pendulum swing-up, as was done in [15]. While simple, this task allows us to demonstrate and compare the validity of our method. To our knowledge, we are the first to demonstrate learning of the control input in a kinodynamic sampling-based planner.

The structure of this letter is as follows. Section II describes the Learning-RRT algorithm. This algorithm is applicable to any data generation method, and intended to structure all Learning-RRT components. The subsequent sections combine to introduce RRT-CoLearn, which is a Learning-RRT. The main contribution comes in Section III, which tackles the problems of *local planning* and *dataset generation* via indirect optimal control. The class of systems for which this approach is valid can be found in that section as well. Section IV discusses how to remove the dataset bias introduced by optimal control. In Section V, we discuss how the learned steering-cost metric affects the convergence of the RRT algorithm. Section VI presents experimental validation of our algorithm. Finally, Sections VI and VII contain discussion and conclusions.

II. Learning-Based RRT

Learning-based RRTs leverage (supervised) learning to speed up the computationally expensive modules of a kinodynamic RRT. The Learning-RRT algorithm is presented in Algorithm 1. Here we present the standard, forward search RRT version of the algorithm as used in the experiments. Enhancements are briefly discussed in Section VII.

The first step in the algorithm is to create a dataset of optimal trajectories through the state-space of the system ($\mathcal{X}$). This step disregards obstacles, as they are handled later on. The dataset $D = \{b^i\}_{i=1}^N$, has entries $b^i = \{x_0^i, x_1^i, j^i, u^i\}$ that consists of an initial state $x_0 \in \mathcal{X}$, a final state $x_1 \in \mathcal{X}$, a distance metric/local steering cost $j \in \mathbb{R}^+$, and a set of parameters $u \in \mathcal{U}$, that describe the optimal input leading the system from state x_0 to state x_1. Note that $\mathcal{U}$ can take many forms, depending on the discretization used. For example, it can be the coefficients of a polynomial or the values at the switch times of a piecewise-linear function.

Algorithm 1: Learning RRT ((V, E), N).

```
$\hat{D} \leftarrow$ generate_data($N$)   // Section III
$D \leftarrow$ clean_data($\hat{D}$)   // Section IV
$\hat{J} \leftarrow$ fit_cost($D$)
$\hat{U} \leftarrow$ fit_input($D$)
$\hat{V} \leftarrow$ fit_valid($D$)   // Section V
$(X, E) \leftarrow (x_{\text{initial}}, \emptyset)$
solutionfound $\leftarrow$ False
while NOT(solutionfound) do
    $x_{\text{target}} \leftarrow$ sample()
    if $ANY(\hat{V}(x, x_{\text{target}}))\forall x \in X$ then
        $x_{\text{nearest}} \leftarrow \arg\min_{x\in X} \hat{J}(x, x_{\text{target}})$
        $(c, x, u) \leftarrow$ simulate($x_{\text{nearest}}, \hat{U}(x_{\text{nearest}}, x_{\text{target}})$)
        if not collision($x_{\text{nearest}}, x$) then
            $X \leftarrow X \cup \{x\}$
            $E \leftarrow E \cup \{(x_{\text{nearest}},\ x,\ c)\}$
return $X, E$
```

The optimal trajectories are generally found using a numerical algorithm searching for local optima, which poses a challenge for the supervised learning algorithm. If two solutions are nearby in x_0 and x_1, but hail from a different local optimum, a deterministic supervised learning algorithm (for example trained on mean-squared error) will predict the average over the two solutions. This not only makes the cost prediction inaccurate, but most importantly it ruins the steering input prediction: the average of the two steering inputs in the data could lead to a completely different state than the target state. This local-optimum bias requires us to create the dataset D in two stages. The first stage creates a dataset $\hat{D}$ of size N which contains local-optimum-bias, and is indicated in Algorithm 1 by the function `generate_data`. The second stage `clean_data` removes the local-optimum-bias to create the desired dataset D.

The second step is to use a supervised learning algorithm on D to approximate the two functions that define the optimal control solution: 1) the function $\hat{J} : (\mathcal{X},\ \mathcal{X}) \to \mathbb{R}^+$, which maps from an initial and a final state to a steering cost, 2) the function $\hat{U} : (\mathcal{X},\ \mathcal{X}) \to \mathcal{U}$, mapping the initial and final state to the required input parameters.

For both function approximators we implement k-nearest neighbours [18], a standard non-parametric function approximator with robust performance in smaller state-spaces. For a test point x, we identify the k nearest neighbors in our dataset D. The predicted value (e.g. for cost j) for the test point is the average value of these neighbors. We cover extensions to other learning techniques in Section VII.

The next step, `fit_valid`, addresses an inherent limitation of supervised learning and is treated in Section V.

The fourth stage of the Learning-RRT algorithm, the online RRT stage, handles long-term planning and obstacles. First, it samples a point and tests for a valid connection from any node in the tree to the sampled point. If that exists, it expands the node that is nearest to the sampled point according to the function $\hat{J}$. The expansion will use the learned steering function $\hat{U}$, which likely makes a small error. The new node is therefore not exactly at the sampled state. The trajectory to the new state is checked for collisions. If collision free, the new node is added to the tree. The algorithm iterates these steps until it connects to the desired region in state-space. As standard in RRTs, the sampling of new nodes is biased towards the goal by intermittently replacing the sampled state with a desired end-state.

III. Data Generation

The dataset for the function approximator is generated from a set of optimal trajectories. The most common approach to find these trajectories are the so-called *direct* optimal control approaches [19], [20]. In these approaches the state equations and cost function are approximated by a discretized system, which is then numerically optimized.

An alternative to direct optimal control is the much older *indirect* approach [21], [22], which first optimizes and then discretizes. For many applications, direct approaches replaced the indirect approach due to better numerical stability at long planning distances. However, it turns out indirect optimal control is ideally suited for the RRT scenario. First, the numerical instability poses no problem for the short segments that are required for RRT. Furthermore, indirect optimal control brings two important benefits. First, it parametrizes the control input in a low dimensional space, which allowing it to be learned. Second, it removes the need for optimization in the sampling process, which speeds up data generation. Both benefits are further explained at the end of this section, after the indirect optimal control method is described. At that point, we will also explain the remaining downside of the indirect optimal control approach: a more biased dataset.

Indirect Optimal Control

Here the indirect optimal control procedure is used to derive the optimal equations of motion for a pendulum swing-up. The procedure can be applied to other systems, with small differences. For more details and proofs see [23].

Indirect optimal control finds the functions $x(t)$ and $u(t)$ from time $t \in \mathbb{R}$ to state $x \in \mathbb{R}^n$ and input $u \in \mathbb{R}^m$, that minimizes a cost function of the following form:

$$J(x(t), u(t)) = \int_0^{t_f} C(x(t), u(t))\mathrm{d}t \quad (1)$$

$$\text{Subject to:} \quad \dot{x}(t) = f(x(t), u(t)) \quad \forall t \in (0, t_f),$$
$$x(0) = x_{\text{initial}}, \quad x(t_f) = x_{\text{final}} \quad (2)$$

where x_{initial} and x_{final} are fixed initial and goal states, and the final time t_f is optimized along with $x(t)$ and $u(t)$. We will often drop the explicit dependency on the time t.

For the pendulum we get $f(x, u) = (\omega, \sin(\theta) + u)$, with $x = (\theta, \omega)$, θ and ω the (angular) position and velocity respectively, and u the torque. The cost integrand $C = 1 + u^2/2$, which takes into account both the time it takes to reach the goal-state as the control effort to do so.

The first step in the indirect optimal control approach is to define the Hamiltonian $\mathcal{H}$, the sum of the integrand C and the inner product of the state derivatives with a vector of Lagrange multipliers, also called the costate. With $(\lambda_\theta, \lambda_\omega)$ as costate, we obtain:

$$\mathcal{H}(x, \lambda, u) = 1 + 1/2\, u^2 + \lambda_\theta \omega + \lambda_\omega(\sin(\theta) + u) \quad (3)$$

The second step is finding an optimal input u^*, by minimizing the Hamiltonian with respect to the input:

$$u^* = \arg\min_u \mathcal{H} = -\lambda_\omega \quad (4)$$

The third step takes partial derivatives of the optimal Hamiltonian, which is created by replacing the input with the optimal input: $\mathcal{H}^*(x, \lambda) = 1 + \lambda_\theta \omega + \lambda_\omega \sin(\theta) - 1/2\lambda_\omega^2$. Note that this equation only depends on the state and costate. The partial derivatives form a system of ordinary differential equations (ODEs) specifying the evolution of the optimal state and costate over time:

$$\dot{\theta} = \frac{\partial \mathcal{H}^*}{\partial \lambda_\theta} = \omega \qquad \dot{\omega} = \frac{\partial \mathcal{H}^*}{\partial \lambda_\omega} = \sin(\theta) - \lambda_\omega \quad (5)$$

$$-\dot{\lambda}_\theta = \frac{\partial \mathcal{H}^*}{\partial \theta} = \lambda_\omega \cos(\theta) \qquad -\dot{\lambda}_\omega = \frac{\partial \mathcal{H}^*}{\partial \omega} = \lambda_\theta \quad (6)$$

The last step is to use (5) and (6) to find the optimal trajectory towards a desired state. For a given costate, the above equations are (numerically) integrated, which results in a locally optimal state trajectory. This trajectory depends on the choice of initial costate and the time duration of the integration. The initial costate and final time are tuned to find a locally optimal state trajectory that reaches the desired state. This tuning requires a numerical method that minimizes the difference between final state and desired state. Later we show that this tuning can be avoided when generating the database.

Optimal control problems solved for Learning-RRTs typically have a free final time and a cost integrand C that does not explicitly depend on time. On this type of problem, the constraint that sets the final time can be rewritten as a constraint on the initial costate [23]:

$$\mathcal{H}^*(x(0), \lambda(0)) = 0 \quad (7)$$

The main reason why *indirect* optimal control is largely replaced by direct methods, such as multiple shooting, is that the resulting differential equations are unstable, and therefore difficult to numerically solve reliably. However, because a learning-RRT database only requires short trajectories, this instability is not as important.

Benefits of Using Indirect Optimal Control

There are two major advantages to using indirect optimal control for Learning-RRTs. First, the input directly follows from the costates; in principle, there is a map $U(x_{\text{initial}}, x_{\text{final}}) \rightarrow (\lambda_\theta(0), \lambda_\omega(0), t_f)$, from initial and final state to a set of only

Algorithm 2: generate_data$(\mathrm{N}, \mathrm{T}_{\mathrm{final}}, \mathrm{r}_{\mathrm{bound}})$.

```
Optimal_ODEs ← Eqs. 1–6
$\hat{D}$ ← empty()
for $n = 1 : N$ do
    $x_{\mathrm{initial}}$ ← random_State()
    $\lambda_{\mathrm{initial}}$ ← random_Costate() s.t. Eq. 7
    $T_{\mathrm{final}}$ ← random_Time()
    $x_{\mathrm{final}}, J$ ← integrate(Optimal_ODEs,$x_{\mathrm{initial}}$ ,
      $\lambda_{\mathrm{initial}}, T_{\mathrm{final}}$)
    append($\hat{D}$,{$x_{\mathrm{initial}}, x_{\mathrm{final}}, J, \lambda_{\mathrm{initial}}$})
return $\hat{D}$
```

three parameters that describe the input function. This set is small, and will only grow linearly with the size of the state space (and is independent of the number of inputs). In contrast, direct optimal control approaches require to parametrize inputs as functions over time, which results in much larger spaces of parameters. The reduction in the number of parameters means the input function can be learned efficiently, thereby solving the problem of *local planning* in RRTs as identified in the introduction.

To see the second major advantage, look at how the whole dataset is created. In current learning RRTs [15], [16], the dataset is generated by sampling from the $(x_{\mathrm{initial}}, x_{\mathrm{final}})$-space, and then, find an optimal trajectory and cost for each sample. Note that every combination of initial state, initial costate and final time produces an optimal trajectory for a certain final state. So, if we sample from the allowed initial states, initial costates and final times, we effectively sample over all initial state - final state combinations. Thus, we can eliminate the need for numerical optimization by sampling the costate, meaning the data are generated much faster.

The data generation procedure is outlined in Algorithm 2. The functions *random_State*, *random_Costate*, and *random_Time* sample random states, costates and final times depending on the problem domain and constraint (7). The function *integrate* numerically integrates the optimal control equations until the final time (T_{final}) or until the distance from the initial state reaches a reachability bound (r_{bound}). To optimize the efficiency of the data generation we add the intermediate integration results to the dataset, causing dependencies between the data-points. We observed that the learning algorithm performed well despite these dependencies.

In this section, we outlined the indirect optimal control approach to *data generation* in learning RRTs. Because the optimal control algorithm incorporates *costates*, we name the overall algorithm RRT-CoLearn. Its two major advantages are: learning optimal steering inputs (online speed) and generating data without optimizing (offline speed).

IV. Dataset Cleaning

The dataset generated by Algorithm 2 originates from a search for local optima, and can therefore include a bias that interferes with learning performance. The problem is illustrated for with an artificial dataset in Fig. 2 (top), where in the middle input

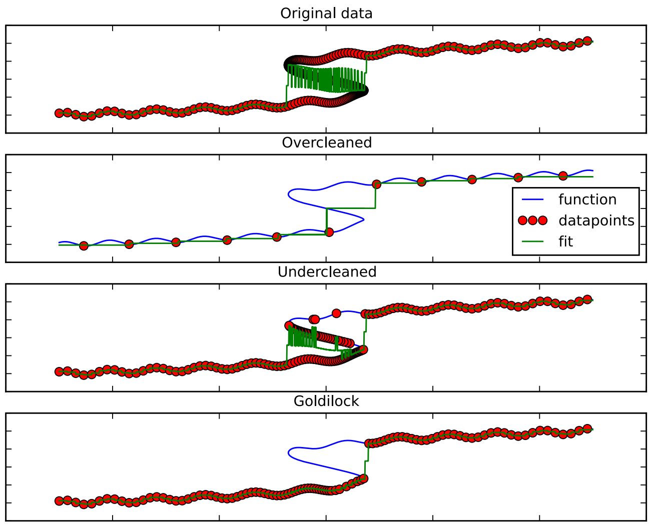

Fig. 2. The data-bias problem and the effect of the d parameter in the data cleaning algorithm. The top figure shows an imaginary dataset, which has a problem with bias in the middle of its domain. The fitted function is a poor approximation of the least cost part of the datapoints. In the second figure d was chosen too large: the bias is gone, but there is not enough resolution left to accurately fit the function. In the third figure, d is too small: not all bias is removed. The bottom figure shows a proper choice for d: the bias is removed, and enough resolution remains.

region we have the global optimum at the bottom, but there are local optima above it. A standard function approximator (with squared loss) for a given point input (independent variable) predicts the conditional expectation of the dependent variable, shown in green. Note that the predicted function deteriorates in the middle segment, where the average is predicted instead of the optimal cost.

This is problematic for predicting the cost function and especially harmful for predicting the control parameters. Averaging over two locally optimal control inputs by no means guarantees that we end up anywhere close to the target. As an intuitive example, consider the Dubins vehicle, which can be seen as a model of a non-holonomic car. The vehicle can reach a goal behind it by either steering left or right, but will fail to reach the target when taking the average (straight ahead). To counteract this issue, we need a dataset cleaning algorithm, i.e., a procedure that somehow eliminates conflicting (non-optimal) datapoints. In literature, there are resampling methods for dataset imbalance, most noteworthy class label imbalance in classification tasks [24]. However, our dataset is not imbalanced, but rather contains a systematic bias. It turns out we can leverage that structure to come up with a simple resampling/cleaning algorithm.

For each point in input space, we are interested in retaining the lower bound of the cost of the generated datapoints. First note that we prefer to remove points in high-density regions, as in low-density regions there is little to throw away, and we may only hope that our data are accurate. We implicitly remove from high density regions by first uniformly sampling a point from our dataset. We then search for its nearest neighbor in the dataset based on Euclidean distance. If this neighbor is within a distance d from our sampled point, we remove the node of the two with

Algorithm 3: clean_data($\hat{D}, d, k_{\max}$).

```
$D \leftarrow \hat{D}$
$k \leftarrow 0$
while $k < k_{\max}$ do
    $p_{\text{sample}} \leftarrow$ selectRandom($D$)
    $p_{\text{neigh}} \leftarrow$ nearestNeighbor($p_{\text{sample}}, D$)
    if distance($p_{\text{sample}}, p_{\text{neigh}}$) $< d$ then
        $k \leftarrow 0$
        $p_{\text{high}} \leftarrow \arg\min_{p \in \{p_{\text{sample}}, p_{\text{neigh}}\}}$ Cost($p$)
        remove($D, p_{\text{high}}$)
    else
        $k \leftarrow k + 1$
return D
```

the highest cost, frequently removing a biased data-point while retaining a good one. Otherwise, we retain both points, which will happen in low density regions. This process is repeated until no points are removed for k_m consecutive steps, after which we return the cleaned dataset. The procedure is outlined in Algorithm 3.

The main parameter in this algorithm, d, can be interpreted as a neighborhood size. Fig. 2 shows it has an optimal setting that depends on the dataset and which has to be found empirically. To evaluate it d, we fix it, run the cleaning, fit the function predictors, and then sample new datapoints to assess the error in cost and co-state parameters on the predictions. This evaluation is not ideal, but the ideal metric (RRT performance) is too expensive to compute.

V. Notes on Completeness

The machine learning approximations in the proposed algorithm intuitively might interfere with its completeness properties. In this letter we do not prove probabilistic completeness of learning RRTs or RRT-CoLearn. However, we made a number of small additions that improve the algorithm and make it more amenable to a future proof of convergence. This section discusses these additions, and the reasoning behind them. A future proof of completeness will likely follow the lines of earlier work such as [6], [11], [25].

For the first addition, note that our inputs are generated based on a prediction of the costate from the function approximator. If the input is constructed directly based on this prediction, the rest of the inputs will have no chance of being selected. If the prediction is not perfectly accurate, this could cause the algorithm to get stuck. Therefore, the input resulting from the prediction is generated based on a probability distribution that associates a high probability of choosing a value close to the prediction, while giving some probability to all possible inputs. In our experiments, the control parameters are samples from truncated normals, with the bounds for each parameter specified by its sampled domain. The means are the value predicted by the learned model. The standard deviation σ of the (non-truncated)-normal is a parameter of the algorithm.

To make our algorithm more similar to the algorithm discussed in [6], which might make it easier to adjust the proof found there to our algorithm, we discretized the action space. This was done by rounding the input to two decimals. As this is a relatively fine discretization, we expect it had little effect on the experiments.

Experiments showed that inaccurate prediction makes selecting the right input difficult when the problem requires the RRT to very precisely reach a small region in state space, as happens when near the goal region. Therefore, when the target in an RRT step involves the goal region, we increase the standard deviation of the input distribution. This empirically improved performance.

Errors made by the distance function approximation could also cause the algorithm to get stuck. If the distance towards a certain node is always underestimated, other nodes might no longer be expanded. Alternatively, if the distance to a node is overestimated, that node might no longer be expanded.

The more dominant method to prevent approximation errors in the distance function to hinder the algorithm looks at a specific type of error. Learning algorithms are intended to generalize using interpolation, so may make large errors when extrapolating. This is especially true for Learning RRTs, for which this problem has not been identified in literature yet. In RRTs we *uniformly* sample state-space, while we have confined our dataset to only contain short motion segments. Therefore, sampled combinations (x_0, x_1) are often outside the dataset, where function approximation may make large errors. Notably, the approximated distance metric might greatly underestimate the steering cost from a certain node, causing that node to be incorrectly chosen for expansion.

This issue can be solved by a binary classifier that decides whether a query would yield a valid prediction, i.e., whether the dataset D covers the queried point. The implement classifier $\hat{V} : (\mathcal{X}, \mathcal{X}) \rightarrow$ v, with $v \in [\texttt{true}, \texttt{false}]$, simply computes the summed distance to the nearest neighbors of the queried point to the points in the dataset, and rejects the query point if this sum becomes too large. An alternative approach, explored in [10], relies on reachability: the notion that the dataset contains only short segments, meaning the final states should be reachable within a short period of time.

The second method to limit the effect of approximation errors is by directly excluding overly small or large values for the distance. In our experiments, it was implemented by clipping the distance value to $10^{\pm 5}$ times the Euclidean distance, and had little effect on the computation.

VI. Experiments and Results

To test our approach, we perform experiments on the pendulum described in Section III. The task is to move the pendulum from its stable equilibrium $(\theta, \omega) = (-\pi, 0)$ to its unstable equilibrium $(0, 0)$.

This experiment does not require obstacle handling; we do not explicitly focus on collision checking or obstacles in our work, as our focus is on the RRT itself and we assume the collision checking is handled by an external algorithm.

Data were generated and cleaned for separate epochs, with 300 runs of the RRT algorithm per epoch. The data for each epoch consist of 40000 simulations, which ended when the local

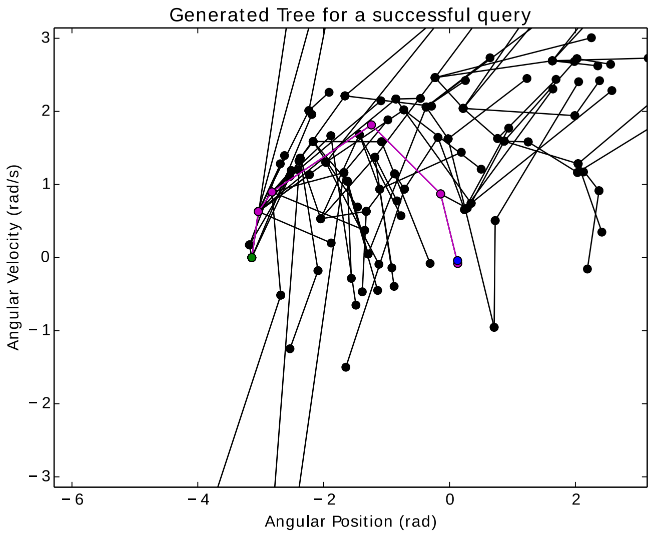

Fig. 3. The state-space coordinates of the tree-nodes of a successful run of the algorithm. The edges are shown as straight lines, with those connecting the initial point to the goal being highlighted.

cost exceeds 2 or when the norm of the state difference with the initial state (r_{bound}) exceeds 1.5. Integration was done by the 4th order Runge-Kutta algorithm with a time step of 0.01 s. The initial position was uniformly sampled from $(-3\pi/2, \pi/2)$rad, the initial velocity from $(-\pi, \pi)$rad s^{-1}, and the initial costate sampled as described below. The data cleaning resolution d equals 0.05. The data cleaning stopping parameter $k_{\max}$ is set to 5000. The nearest neighbor fitting algorithm during the RRT takes $m = 3$ nearest neighbors. Finally, the standard deviation of the sampling distribution $\sigma = \pi/4$ normally, and $\pi/2$ when the query involves the goal state. The experiments ran on a MacBook with Intel(R) Core(TM) i5-3210M 2.50 GHz CPU and 8GB of RAM, running Ubuntu Linux 14.04 and code in Python and Julia.

To avoid projecting on the costate constraint (7), which is computationally expensive for larger systems, the constraint is solved explicitly by uniformly sampling the parameter $\phi \in (-\pi/2, 3\pi/2)$ that sets the initial costate as follows: $\lambda_\theta = \tan(\phi)$ and $\lambda_\omega = -\sin(\theta) + \text{sign}(\cos(\phi))\sqrt{\sin(\theta)^2 + 2 + \tan(\phi)\omega}$. If λ_ω has an imaginary part, the simulation is disregarded.

Fig. 3 shows a typical run of the algorithm. Three important points can be seen from this figure. First, the algorithm neatly expands through state-space. While it favors expanding along the circular paths that correspond to low input trajectories, it expands nodes in all feasible directions. This indicates that the distance metric works properly, and contrasts with trees that are grown without appropriate distance metric, which tend to have many nodes clustered. Secondly, the figure highlights the approximation errors that still exists; the edges that lead to the border of the figure all had target nodes inside the figure. As these nodes are outside the figure, the target was not reached exactly. Finally, the algorithm has tried to reach the goal-state a number of times from the same node. This shows the effect of the increased randomness; because the attempts are spread out sufficiently, the goal is eventually reached.

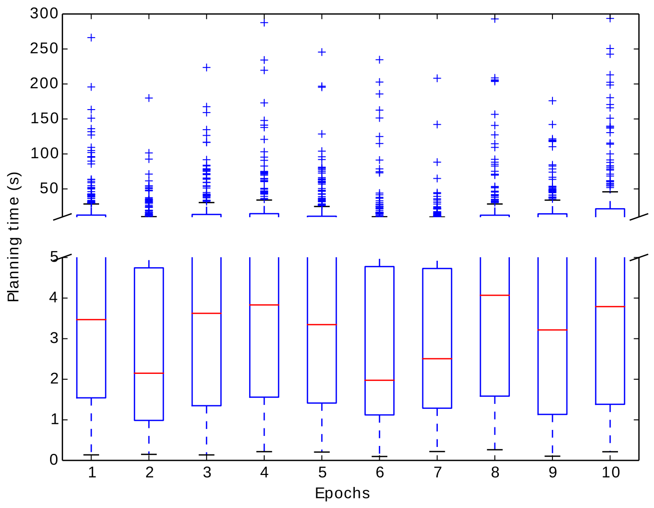

Fig. 4. Boxplot of planning times for ten epochs.

Fig. 4 shows the computation times for 10 epochs. The planning time has the same variation in each epoch, indicating that data generation and cleaning are performed robustly. The median time to reach the target over all samples was 2.43 s, over 10 times faster than [15] on the same hardware. Data generation and cleaning took $\sim$25 min per epoch, an order of magnitude faster than [15].[1]

The performance of the learned steering function is assessed by the mean squared error between the target state and the final state attained by using the predicted costate. The median of this error over all epochs is 0.11.

The quality of the combined machine learning is assessed using the number of nodes needed to reach the target. For the results in Fig. 4, the median over all runs is 84 nodes. About 30% smaller (better) than in [15], this result is best interpreted as a roughly equal performance, as the problem here does not require the pendulum to swing back and forth to reach its goal. Equal performance indicates the learned steering function approximates the online optimization well.

Fig. 5 shows the effect of the reachability bound on the number of nodes needed to reach the goal position. This investigates how the length of the simulation in the dataset influences the performances of the algorithm. When the simulations are too short, the online algorithm is forced to use many segments, hindering convergence. When simulations are too long, the coverage of all possible trajectories becomes sparse, causing poorer learning performance.

Fig. 6 shows the number of nodes needed to reach the goal position with various parts of the algorithm removed or replaced. This allows us to identify the important aspects of the algorithm, and to compare the algorithm to the state-of-the-art. The last four algorithms have the validity check turned off and perform worst. Note that [15] does not use such a check, but instead implements an online trajectory optimization to avoid the poor performance without validity check shown here. The figure also shows that the difference between learned and Euclidean distance is small

[1]The cited paper does not report the offline computation time. However, due to overlapping authors we know offline computation took nearly a week.

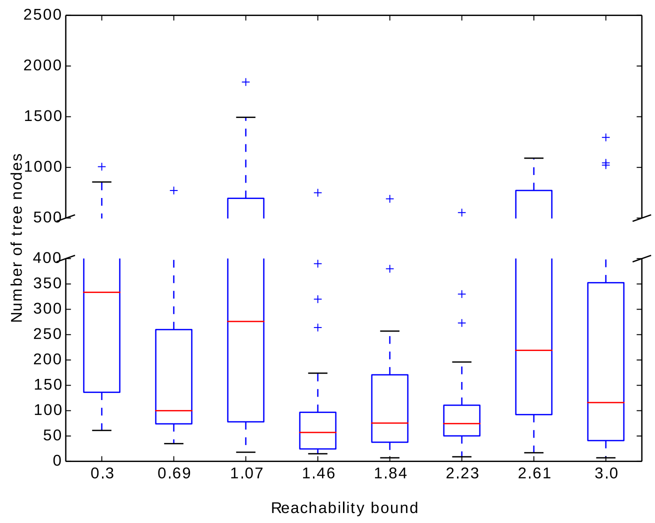

Fig. 5. The number of nodes required to find a solution using different settings of the reachability bound.

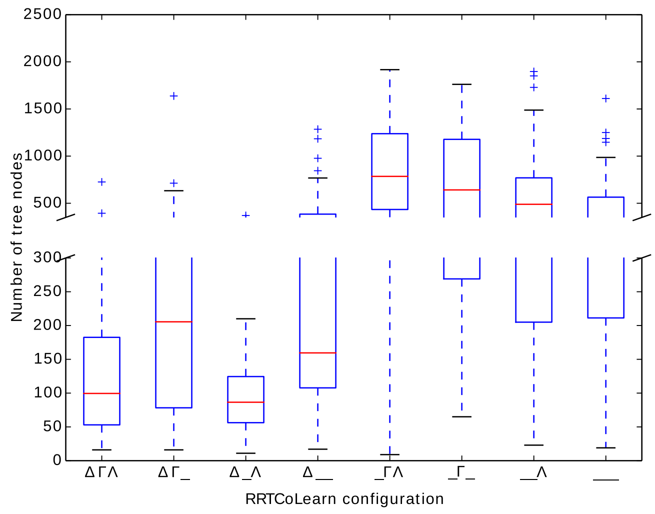

Fig. 6. The number of tree nodes required to find a solution for different configuration settings for RRT-CoLearn algorithm. The horizontal axis is labeled with three symbols, which can be turned on or off. In the first position, the Δ signifies the validity check being turned on. Similarly, the Γ switches between learned and Euclidean distance, and the Λ between learned and random costate (action) selection.

when using a validity check, confirming the intuition from [10]. The algorithm in that paper is very close to the Δ__-algorithm in Fig. 6. Finally, we see that learning the actions makes the algorithm perform better than using random costate selection.

VII. Discussion

The algorithm introduced in this letter allows learning of not only the distance metric, but also the steering input. As proof of concept, we tested our algorithm on a basic pendulum swing up problem. It reduces the time spend both in the offline learning and in the online computation by a factor of more than 10, a large step towards sampling based kinodynamic planning in a practical setting.

The main direction for future research is to support higher degree of freedom systems. Such systems will require more sample efficient data generation and learning. To improve the efficiency of data generation, new simulations might be selected based on already obtained data, and the partially learned cost and steering functions, similar to what has been done for reinforcement learning [26]. (Deep) generative models could provide more efficient learning. These models allow sampling from complex, high-dimensional probability distributions, meaning we could retain all data points without averaging over solutions [27], [28].

A secondary direction for future research is to support input bounds. We have taken a preliminary step in this direction by implementing RRT-CoLearn on a time optimal pendulum swing up with the torque bound set to 0.5, and all other constants as in (3). The resulting optimal equations of motion are invariant to the norm of the costate vector. Therefore the constraint of (7) is satisfied by setting the norm of the costate to 1. For 50 runs of RRT-CoLearn, the solution was always found within 2000 nodes, using a median planning time of 12.32 s, and median number of 195 nodes. These results compare favorably with those reported by the state-of-the-art in kinodynamic planning [4], [10], [15].

Unfortunately, the input constraints can cause an exact overlap between trajectories starting from different costates, at least for a finite time. Such overlapping trajectories are difficult for the learning and cleaning algorithms we used. Therefore, a larger dataset was required, severely slowing down the RRT. Extending the machine learning aspects of RRT-CoLearn, such that they can cope with overlapping trajectories is an important theoretical and practical issue.

The third direction for future research is the combination of the RRT-CoLearn algorithm with other (non-learning) enhancements of the basic RRT algorithm, such as [29], [30]. RRT-Connect [29] would be the most prominent addition. It grows two trees: one forwards from the initial state, and one backwards from a goal state. A new model based on backwards integrated data must be learned for creating the backwards-searching tree. As the system of differential equations (5), (6) is unstable in both forward and backward direction [23], the learning challenges remain similar.

Finally, note that RRTs, and therefore learning RRTs, are suited to finding novel motions for deterministic systems for which an accurate model is available. When planning for non-deterministic or uncertain systems, funnel based approaches are more appropriate than trajectories [31]. If the system has to perform similar motions repeatedly, multi query methods such as probabilistic roadmaps [32] or motion libraries can be more effective [33]. For the last problem, learning has already been used to find similarity between obstacle locations between planning instances [34]. It is interesting to investigate if using indirect optimal control similarly to this letter could be beneficial for those settings.

VIII. Conclusion

This letter first described a general Learning RRT algorithm. Table I shows the benefits and challenges of its components:

TABLE I
BENEFITS AND CHALLENGES OF COMPONENTS OF LEARNING-RRTS

	Optimal control	Machine Learning	RRT
+	Distance computation Optimal local planner	Generalization Fast online prediction	High dimensional Obstacle avoidance
−	Costly computation$^{\mathrm{III}}$ Local optima→bias$^{\mathrm{IV}}$	Needs large dataset$^{\mathrm{III}}$ Needs unbiased data$^{\mathrm{IV}}$ Bad extrapolation$^{\mathrm{V}}$	Needs distance metric Needs local planner$^{\mathrm{III}}$

Optimal Control, Machine Learning and RRT. The superscripts in the table refer to the sections in which the challenges are addressed.

RRT-CoLearn, an instance of a learning-RRT, was proposed. RRT-CoLearn generates data faster and allows learning of the steering function, both by using indirect optimal control. It also uses a newly proposed data-cleaning algorithm for more accurate function approximation.

RRT-CoLearn was successfully used on a pendulum swing up both with and without input constraints. The main results are on the system without input constraints and show that RRT-CoLearn is 10 times faster than a state-of-the-art learning RRT [15].

REFERENCES

[1] S. H. Collins, A. Ruina, R. Tedrake, and M. Wisse, "Efficient bipedal robots based on passive-dynamic walkers," *Science*, vol. 307, pp. 1082–1085, Feb. 2005.

[2] W. J. Wolfslag, M. Plooij, R. Babuska, and M. Wisse, "Learning robustly stable open-loop motions for robotic manipulation," *Robot. Auton. Syst.*, vol. 66, pp. 27–34, Apr. 2015.

[3] M. Plooij, H. Vallery, and M. Wisse, "Reducing the energy consumption of robots using the bi-directional clutched parallel elastic actuator," *IEEE Trans. Robot.*, vol. 32, no. 6, pp. 1512–1523, Dec. 2016.

[4] Q.-C. Pham, S. Caron, P. Lertkultanon, and Y. Nakamura, "Admissible velocity propagation: Beyond quasi-static path planning for high-dimensional robots," *Int. J. Robot. Res.*, vol. 36, no. 1, pp. 44–67, 2016.

[5] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," *Int. J. Robot. Res.*, vol. 21, no. 3, pp. 233–255, 2002.

[6] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," *Int. J. Robot. Res.*, vol. 20, pp. 378–400, 2001.

[7] E. Glassman and R. Tedrake, "A quadratic regulator-based heuristic for rapidly exploring state space," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2010, pp. 5021–5028.

[8] A. Perez, R. Platt Jr, G. Konidaris, L. Kaelbling, and T. Lozano-Perez, "LQR-RRT*: Optimal sampling-based motion planning with automatically derived extension heuristics," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2012, pp. 2537–2542.

[9] S. Karaman and E. Frazzoli, "Sampling-Based optimal motion planning for non-holonomic dynamical systems," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2013, pp. 5041–5047.

[10] A. Shkolink, M. Walter, and R. Tedrake, "Reachability-guided sampling for planning under differential constraints," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2009, pp. 2859–2865.

[11] Y. Li, Z. Littlefield, and K. Bekris, "Asymptotically optimal sampling-based kinodynamic planning," *Int. J. Robot. Res.*, vol. 35, no. 5, pp. 528–564, 2016.

[12] G. Goretkin, A. Perez, R. Platt Jr., and G. Konidaris, "Optimal sampling-based planning for linear-quadratic kinodynamic systems," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2013, pp. 2429–2436.

[13] D. J. Webb and J. van den Berg, "Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2013, pp. 5054–5061.

[14] E. Schmerling, L. Janson, and M. Pavone, "Optimal sampling-based motion planning under differential constraints: The drift case with linear affine dynamics," in *Proc. IEEE 54th Annu. Conf. Decis. Control*, 2015, pp. 2574–2581.

[15] M. Bharatheesha, W. Caarls, W. Wolfslag, and M. Wisse, "Distance metric approximation for state-space RRTs using supervised learning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2014, pp. 252–257.

[16] L. Palmieri and K. O. Arras, "Distance metric learning for RRT-based motion planning with constant-time inference," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2015, pp. 637–643.

[17] R. Allen and M. Pavone, "A real-time framework for kinodynamic planning with application to quadrotor obstacle avoidance," in *Proc. AIAA Guid., Navigat., Control Conf.*, San Diego, CA, USA, 2016, pp. 5021–5028.

[18] J. Friedman, T. Hastie, and R. Tibshirani, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. (Springer Series in Statistics), vol. 1. Berlin, Germany: Springer, 2001.

[19] M. Posa, C. Cantu, and R. Tedrake, "A direct method for trajectory optimization of rigid bodies through contact," *Int. J. Robot. Res.*, vol. 33, no. 1, pp. 69–81, 2014.

[20] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2012, pp. 4906–4913.

[21] A. V. Rao, "A survey of numerical methods for optimal control," *Adv. Astronaut. Sci.*, vol. 135, no. 1, pp. 497–528, 2009.

[22] L. S. Pontryagin, *Mathematical Theory of Optimal Processes*. Boca Raton, FL, USA: CRC Press, 1987.

[23] D. S. Naidu, *Optimal Control Systems*. Boca Raton, FL, USA: CRC press, 2002.

[24] M. Galar, A. Fernandez, E. Barrenechea, H. Bustince, and F. Herrera, "A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 42, no. 4, pp. 463–484, Jul. 2012.

[25] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot. Res.*, vol. 30, pp. 846–894, 2011.

[26] S. Levine and V. Koltun, "Guided policy search," in *Proc. 30th Int. Conf. Mach. Learn.*, 2013, pp. 1–9.

[27] I. Goodfellow *et al.*, "Generative Adversarial Nets," *Adv. Neural Informa. Process. Syst. 27*, pp. 2672–2680, 2014. [Online]. Available: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

[28] T. M. Moerland, J. Broekens, and C. M. Jonker, "Learning multimodal transition dynamics for model-based reinforcement learning," arXiv:1705.00470, 2017.

[29] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, vol. 2, 2000, pp. 995–1001.

[30] R. A. Knepper, S. S. Srinivasa, and M. T. Mason, "Toward a deeper understanding of motion alternatives via an equivalence relation on local paths," *Int. J. Robot. Res.*, vol. 31, no. 2, pp. 167–186, 2012.

[31] R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts, "LQR-trees: Feedback motion planning via sums-of-squares verification," *Int. J. Robot. Res.*, vol. 29, no. 8, pp. 1038–1052, 2010.

[32] R. Geraerts and M. H. Overmars, "A comparative study of probabilistic roadmap planners," in *Algorithmic Foundations of Robotics V*. New York, NY, USA: Springer, 2004, pp. 43–57.

[33] D. Berenson, P. Abbeel, and K. Goldberg, "A robot path planning framework that learns from experience," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2012.

[34] A. Tallavajhula, S. Choudhury, S. Scherer, and A. Kelly, "List prediction applied to motion planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2016, pp. 3671–3678.