Learning indirect optimal control for dynamic motion planning with RRT

W.J. Wolfslag, M. Bharatheesha, T.M. Moerland and M. Wisse
Delft University of Technology
e-mail: w.j.wolfslag@tudelft.nl

***Abstract*—Sampling based motion planners are well suited to plan motions around obstacles for high dimensional systems, but require extensive online computation to plan in state-space. A novel learning approach, based on indirect optimal control, shifts the computation cost offline. We show results on a pendulum swing up problem, and plan to evaluate on higher dimensional systems.**

I. Introduction

Many tasks, including walking, are best planned while explicitly considering the equations of motion. Unfortunately, such state-space planning is computationally complex, particularly when there are obstacles to avoid. Sampling based planning algorithms, such as Rapidly-exploring Random Trees (RRT), have proven to be the most viable way to handle high dimensional spaces and obstacles [3, 4] when planning in configuration space. We research the use of RRT for planning in state-space.

RRT works by splitting the planning into smaller segments, see Figure 1. It builds a tree structure with the states of the system as nodes, and the trajectories between them as edges. By sampling a random point in state-space, and *connecting/steering* it to the *nearest* node in the tree, the tree is grown. This process continues until the desired state is reached. The challenge of using this approach in state-space is that both connecting nodes and determining which node is nearest are computationally expensive tasks [4].

A promising approach to avoid performing these tasks online is called Learning-RRT[1, 5]. In Learning-RRT, an optimal control algorithm provides a database of optimal trajectories. These trajectories are the input for a supervised learning algorithm, which in turn provides approximations to the *steering* and *distance* functions the RRT requires. As these approximations are quick to compute, the online performance of the planning algorithm is increased.

Until now, learning-RRT algorithms have only learned the distance function. The steering function typically has a much higher dimension, and is therefore more difficult to approximate. Online steering therefore remains a computational burden [1]. Furthermore, in order to learn the distance function, a dataset of many optimal trajectories needs to be found. As all these trajectories are found numerically, the increase in online performance comes at a large cost in offline-computation. Here we propose to use a specific optimal control scheme, known as indirect optimal control to generate the dataset. This approach, which results in the RRTcoLearn algorithm, addresses both issues mentioned above.

II. Method

The first step in a learning-RRT is to create a dataset of optimal trajectories. Specifically, each datum captures an initial state, a final state, a cost-to-go and a set of parameters that describe the input used for the trajectory. The most common approach to find these trajectories are the so-called *direct* optimal control approaches. In direct optimal control the state equations and cost function are approximated by a discretized system. A numerical optimization scheme then optimizes this discretized approximation. Depending on the type of discretization, various schemes have been devised, e.g. [6, 8, 10, 11].

An alternative to these direct approaches is the much older *indirect* approach [9, 7], which first optimizes using calculus of variations and then discretizes. For many applications, direct approaches replaced the indirect approach, due to better numerical properties: indirect approach is numerically unstable at long planning distances. However, the numerical instability is no problem for the short segments in RRT.

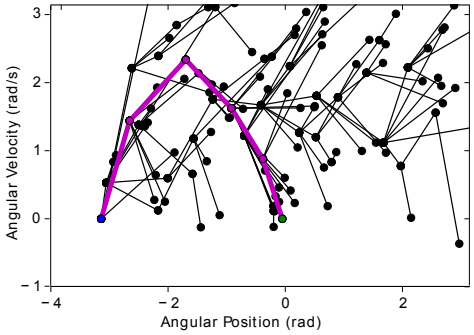

Fig. 1: State-space coordinates of the nodes in the tree of a succesful run. The path that leads from the initial point to the goal is highlighted.

Indirect optimal control includes the equations of motion in the cost function by using Lagrange multipliers, which are called the costate. By setting the variation of the augmented cost function to zero, we obtain a set of differential equations for the evolution of the state and the costate. Typically, the indirect optimal control approach proceeds as follows. For a given costate, these differential equations are (numerically) integrated, which result in a locally optimal state trajectory. Note that the final state of this trajectory depends on the choice of initial costate and the final time of the integration. By tuning the initial costate and the final time, we find a locally optimal trajectory that reaches the desired state. The first advantage of the indirect optimal control approach is apparent here, because the costate effectively parametrizes the input using the same number of parameters as there are states. This is much smaller than the number of parameters in most direct optimal control schemes.

In typical use, the costate and time are tuned by a numerical optimization method, which minimizes the difference between final state and desired state. However, for our purposes, such numerical optimization is not required. Note that all feasible final states are reached (optimally) by some combination of initial state, initial costate and final time . Therefore, if we sample from the allowed initial states, initial costates and final times, we effectively sample over all initial state-final state combinations. While previous approaches had to sample across state-space and then numerically optimize the steering input, we can now move the sampling to the costate. This is the second advantage of using the indirect optimal control approach, as it eliminates the need for numerical optimization, and therefore the data can be generated much faster.

The second step of a learning-RRT is to train a supervised learning algorithm on the dataset in order to approximate the connection and distance functions. We use the k-nearest neighbours [2] algorithm in our experiments, because it is a robust approach for low dimensional problems. More sophisticated algorithms potentially improve the results on higher dimensional problems, and are currently being implemented.

The third, online, step of the Learning RRT algorithm is the same as a normal RRT-algorithm, which consists of the following steps. First, sample a point in statespace. Then, select the node that is nearest to the sampled point according to the approximated distance function. Finally, use the approximated connection function to expand the selected node. The algorithm iterates these steps until it connects to the desired region in state-space.

Challenges of learning RRT

The use of machine learning to approximate distances and trajectories in RRT leads to three issues, that were left undiscussed in previous literature on the topic. When not adressed, these issues cause the RRTcoLearn algorithm to fail.

First, as the dataset is generated by an algorithm that finds local optima, it might find multiple ways to find a connection between two states, each associated with a local optimum. This is only a minor problem for the cost-to-go function, but harmful for predicting the control parameters. A naïve learning algorithm presented with two solutions to the same problem will, roughly speaking, average over the solutions. Averaging over two control inputs that reach the target often results in a motion that ends up nowhere near the target. To resolve this issue we use a procedure that aims to eliminate the non-globally-optimal datapoints. Our procedure is based on a simple heuristic: when two datapoints are close together, remove the one with the highest cost-to-go. By repeatedly sampling datapoints, and applying this heuristic, the dataset is cleaned up.

Second, a machine learning algorithm can only hope to approximate well near data it has already seen. In a learning RRT, the dataset consists of short motions, whereas the RRT-algorithm might ask questions about long motions. Therefore our machine learning step has an additional phase, which provides a classifier that tells whether predictions are valid or not. For an initial and final state sampled online, this classifier tells whether there are enough nearby samples in the dataset for the output of the learned estimators to make sense. If this is not the case, the sample is marked as invalid, and the predictions are not used in the rest of the algorithm.

Third, if the RRT has to precisely reach a certain state, even small approximation errors by the learning algorithm can stop the algorithm from finding a solution. This problem can occur due to stringent constraints, for example from obstacles, but most prominently it occures near the goal-state. The RRT's state sampler is often set to occasionally sample the goal-state. When the nearest node is expanded, and there is an approximation error, the goal region is not reached. In a bad case, the node added to the tree is further away from the goal-state than its parent node. In that case, renewed sampling of the goal-state will keep giving the same result. This means the algorithm gets stuck near the goal state, which we empirically observed in our experiments. To avoid repeating the same malicious trajectory, we added some noise to the predicted input. This resolved the goal-state problem in our experiments.

III. Experiments

To provide a proof of concept of our algorithm, we tested it on a pendulum swing up. A torque controlled pendulum has to move from its stable equilibrium, angle $\theta = -\pi$, to its unstable equilibrium, $\theta = 0$. The data generation and cleaning algorithms were run 10 times, to create 10 epochs, with 1000 runs of the RRT algorithm per epoch.

The results of our experiment are shown in Figures 1 and 2. The former shows that the tree of a typical run expands neatly through state-space, with average number of nodes needed to find the target is 93. These results are similar to [1], and indicate the distance metric is approximated well. An issue is also visible: there are nodes outside the figure, and there is a large density of nodes around the goal-state. This suggests errors in approximating the steering input. The latter figure shows the computation times, as found during each epoch seperately. The average time to reach the target over all samples was 2.4 seconds. The offline phases, simulation and data cleaning, took approximately 25 minutes per epoch. Both offline and online phases are an order of magnitude shorter than the algorithm from [1]. Furthermore, the variation between epochs is small, indicating the the algorithms performs robustly.

IV. Outlook

We proposed a novel algorithm, RRTcoLearn, that combines Learning RRT with indirect optimal control. This combination reduces the number of parameters that describe the steering input, making it feasible to learn this input. Furthermore, the indirect optimal control approach allows the dataset to be generated without numerical optimization, which is much faster. The RRT coLearn algorithm was tested on a pendulum swing up, where it performed 10 times faster than the state-of-the-art learning based kinodynamic RRT.

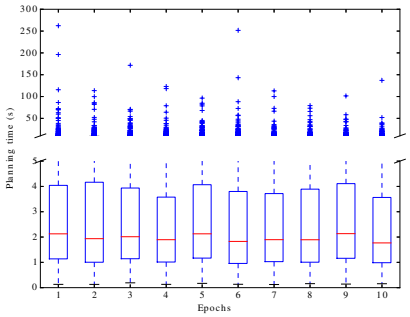

Fig. 2: Planning times, separated by epoch.

Currently, we are working to extend our results to multiple degree of freedom systems. As part of that effort, we incorporate more sophisticated machine learning during the learning phase of our algorithm. These algorithms should learn from fewer samples while handling the bias in the dataset. The second goal we are working towards is the inclusion of input bounds. The indirect optimal control approach allows such bounds, but the resulting dataset of optimal trajectories comes with an additional challenge: the trajectories overlap for long periods, making them hard to distinguish. Again, we look for a solution by incorporating/proposing machine learning techniques that can cope with the relevant issues.

Acknowledgements

This work is part of the research program STW, which is (partly) funded by the Netherlands Organization for Scientific Research (NWO). The work leading to these results has also received funding from the European Communitys Seventh Framework Programme (FP7/2007-2013) under grant agreement No. 609206.

References

[1] M. Bharatheesha, W. Caarls, W. Wolfslag, and M. Wisse. Distance metric approximation for state-space RRTs using supervised learning. In *"2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2014)"*, pages 252–257, 2014.

[2] J. Friedman, T. Hastie, and R. Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics Springer, Berlin, 2001.

[3] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock. Randomized Kinodynamic Motion Planning with Moving Obstacles. *The International Journal of Robotics Research*, 2002.

[4] S. M. LaValle and J. J. Kuffner-Jr. Randomized Kinodynamic Planning. *The International Journal of Robotics Research*, 20:378–400, 2001.

[5] L. Palmieri and K. O. Arras. Distance Metric Learning for RRT-based Motion Planning for Wheeled Mobile Robots. In *Proceedings of Machine Learning in Planning and Control of Robot Motion Workshop, IROS 2014*, pages 637–643, 2014.

[6] M. A. Patterson and A. V. Rao. GPOPS-II: A MATLAB software for solving multiple-phase optimal control problems using hp-adaptive Gaussian quadrature collocation methods and sparse nonlinear programming. *ACM Transactions on Mathematical Software (TOMS)*, 41(1):1, 2014.

[7] L. S. Pontryagin. *Mathematical theory of optimal processes*. CRC Press, 1987.

[8] M. Posa, C. Cantu, and R. Tedrake. A direct method for trajectory optimization of rigid bodies through contact. *The International Journal of Robotics Research*, 33(1):69–81, 2014.

[9] A. V. Rao. A survey of numerical methods for optimal control. *Advances in the Astronautical Sciences*, 135(1):497–528, 2009.

[10] Y. Tassa, T. Erez, and E. Todorov. Synthesis and Stabilization of Complex Behaviors through Online Trajectory Optimization. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4906–4913, 2012.

[11] Y. Tassa, N. Mansard, and E. Todorov. Control Limited Differential Dynamic Programming. In *IEEE International Conference on Robotics and Automation*, pages 1168–1175, 2014.