Deep sequential models for sampling-based planning

Yen-Ling Kuo, Andrei Barbu, and Boris Katz

***Abstract*— We demonstrate how a sequence model and a sampling-based planner can influence each other to produce efficient plans and how such a model can automatically learn to take advantage of observations of the environment. Sampling-based planners such as RRT generally know nothing of their environments even if they have traversed similar spaces many times. A sequence model, such as an LSTM, guides the search for good paths. The resulting model, called DeRRT*, observes the state of the planner and the local environment to bias the next move and next planner state. The neural-network-based models avoid manual feature engineering by co-training a convolutional network which processes map features and observations from sensors. We incorporate this sequence model in a manner that combines its likelihood with the existing bias for searching large unexplored Voronoi regions. This leads to more efficient trajectories with fewer rejected samples even in difficult domains such as when escaping bug traps. The techniques presented here are general and can be adapted to a range of planners.**

This extended abstract is a significantly shortened version of the accepted IROS 2018 paper with the same name omitting technical details, extensions, and results.

I. INTRODUCTION

When you navigate an environment containing new agents, obstacles, and goals, you can rely on previous experiences to guide your actions. Having seen similar agents before allows you to predict the motions of the ones you encounter in the future. Having seen obstacles, whether static or dynamic, allows you to efficiently navigate around them. Your expectations about the future of the plan are conditioned on your previous experiences, current plan, and local observations to help you navigate. This is the process we are modeling here.

Existing sampling-based planners have difficulty taking advantage of such information. Most planners, like RRT* [1], sample uniformly and take no heed of the environment. RRT*, Rapidly-exploring Random Tree, is part of a family of algorithms [2, 3, 4, 5] that explore a configuration space by sampling moves while avoiding invalid states. Dynamic environments, in particular, pose many challenges. They combine uncertain sensing of the position of obstacles and agents with uncertainty about the future path of those obstacles and the actions being performed by other agents. To improve planning in these domains, we adopt a set of techniques from computer vision. We bias the growth of the RRT* search tree [6, 7] given prior experience and a sensed environment. Hidden Markov Models [8], and stacked LSTMs [9] are powerful activity recognizers [10, 11, 12] but so far they have seen little use in improving robotic planning. We demonstrate how to adapt such sequence models to robotic planning using a general approach that can employ either graphical models or neural networks.

This work was supported by the Center for Brains, Minds and Machines, NSF STC award 1231216, the Toyota Research Institute, CBMM-Siemens Graduate Fellowship, and the MIT-IBM Brain-Inspired Multimedia Comprehension project.

Computer Science and Artificial Intelligence Laboratory, MIT
{ylkuo,abarbu,boris}@mit.edu

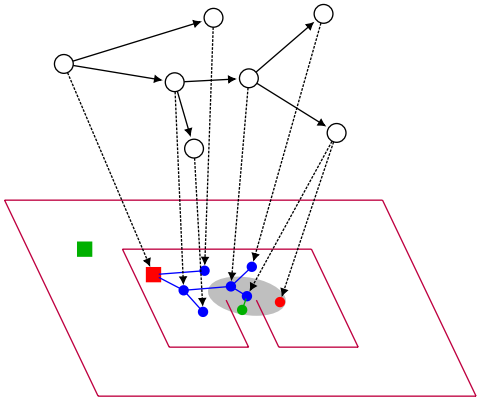


Fig. 1. A DeRRT*-based planner starts at the red square and tries to reach the green square while escaping from a bug trap. The search tree, shown as blue circles, is mirrored by a sequence model, an HMM or LSTM. When expanding the tree, a free-space sample is drawn, steered toward, and the resulting node, shown as a red circle, is used to find the closest node in the tree; as in RRT. The sequence model, with state corresponding to that closest node, observes this free-space sample, the path leading to this node, along with local visual or map features, shown in gray, and predicts a modified direction, shown in green, which is then connected to the search tree. A new state for the sequence model is also predicted and connected. This process incorporates the bias to explore free space of RRT-based planners with a co-evolving sequence model and observations of the environment.

A sequence model co-evolves alongside a sampling-based planner as shown in Fig. 1. At each planning step, both the planner and the sequence model are stepped forward while the next sample from the planner is conditioned on the sequence model. That sequence model can observe local features of the environment as well as the current plan to provide good samples. Moreover, we can avoid feature engineering and learn the relevant features of the environment by co-training a convolutional network (CNN) with the LSTMs. We refer to this algorithm as DeRRT*, for deep RRT*, although the techniques presented here can be adapted to other sampling-based planners.

II. PLANNING WITH SEQUENCE MODELS

DeRRT* combines a sequence model with a sampling-based planner, RRT*. RRT-based algorithms create a tree which explores a configuration space. Given an initial state, RRT* samples locations uniformly and then attempts to connect them to that original node. The tree reaches outward to cover the configuration space with a bias for large unexplored Voronoi cells to find the goal state. See algorithm 1 for a prototypical RRT.

Algorithm 1 A prototypical RRT algorithm.

$V \leftarrow \{x_{\text{init}}\}; E \leftarrow \emptyset$
for $1 \dots n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}()$
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$
  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$
  if $\texttt{ObstacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $V \leftarrow V \cup \{x_{\text{new}}\}$
    $E \leftarrow E \cup \{(x_{\text{nearest}}, x_{\text{new}})\}$
return $G = (V, E)$

At each iteration of the RRT algorithm, we simultaneously extend the tree and a sequence model. Just as RRT trees have a branching structure, the sequence model will have that same branching structure. This conditions future states of the sequence model on past states for that particular hypothesized plan. Fig. 1 shows an example of this process. As a node is expanded, the precise position in the configuration space of a new candidate node is sampled from the sequence model. To implement this, we modify the steering function to move in a direction given by the sequence model while conditioning it on the current state, the desired free space direction, and observations of the local environment around the current state.

While several extensions to RRT consider changing the sampling function, here we instead change the steering function. This distinction is important and we take this approach for several reasons.

1) It preserves the most desirable property of RRT, its bias for large Voronoi regions. Exploring novel regions helps in difficult domains where simply attempting to directly reach the goal is unlikely to succeed.
2) There is no need to change the sampling function. If the sequence model for the steering function has high confidence, the random sampled direction in free space is irrelevant. Intuitively, the sequence model controls how much exploration vs. exploitation is occurring based on its confidence in the next direction.
3) We would like to take advantage of local observations to help guide the algorithm. To do this, we allow the sequence model to learn the nature of the features from local observations. Steering moves are small, making local decisions about the direction of motion, while free space-sampling controls the overall direction of motion. Local features are far more informative for small moves than for deciding what the overall direction across an entire map or maze might be.

We keep the overall structure of RRT*[1] unchanged modifying two of its component functions, `Steer` and `Rewire`. We modify the `Steer` to guide the planner toward a direction informed by the sequence model instead of just minimizing the distance to the uniformly sampled node x_{rand}. For performance reasons, this necessitates an update to `Rewire` to cache the state of the sequence model when a node changes its parent.

When steering, one starts from node x_{nearest} and heads in the direction of the sampled point x_{rand}. The end node of a single step of the steering function, x_{new}, lies within a distance r of x_{nearest}, within a sphere $\mathcal{B}_{x_{\text{nearest}}, r}$. In the original RRT*, x_{new} is chosen to minimize the distance to x_{rand}. We replace this function with `SteerWithModel`, as shown in algorithm 2.

`SteerWithModel` proceeds as follows. First, we find μ, the optimal point according to the original RRT* algorithm. Next, we sample a point within steering distance r of x_{nearest} conditioned on the sequence model, λ, along with any observations from available sensors, `Obs`. When the sequence model allows for efficient conditioning of the samples based on this sphere and sensor data, we can directly sample from the posterior. Most models do not allow for this. We instead sample a fixed number of points, k, compute the likelihood of each, and sample proportionally to those likelihoods.

Algorithm 2 $\texttt{SteerWithModel}(x_{\text{nearest}}, x_{\text{rand}})$

$\mu \leftarrow \underset{z \in \mathcal{B}_{x_{\text{nearest}}, r}}{\operatorname{argmin}} \|z - x_{\text{rand}}\|$
$P \leftarrow \emptyset$
for $1 \dots k$ **do**
  $x_{\text{next}} \leftarrow \texttt{SampleUniform}(x_{\text{nearest}}, \mu, r)$
  $p_{\text{next}} \leftarrow P(x_{\text{next}}, \mu, \texttt{Obs} | \lambda, x_{\text{init}}, \dots, x_{\text{nearest}})$
  $S \leftarrow S \cup \{(x_{\text{next}}, p_{\text{next}})\}$
$(x_{\text{new}}, p_{\text{new}}) \leftarrow \texttt{Sample}(S)$
return x_{new}

Intuitively, when the sequence doesn't provide any information about the configuration space, it can learn to simply provide high likelihood when the hypothesized direction x_{next} is close to μ. This reverts the steering function to the one from the original RRT*. At the other extreme, the sequence model may choose to disregard the free space samples if the future path is clear.

Using recurrent networks to approximate complex probability distributions is not new. For example, Le et al. [13] show that probabilistic programs can be compiled into neural networks that take observations as input and learn to perform inference. We consider classes of recurrent models such as RNNs [14], LSTMs [9], and GRUs [15] to approximate the likelihood of the sampled points.

At each time step, models observe the difference between the hypothesized direction, x_{next}, and the optimal direction according to the original RRT*. The recurrent networks take a local observation around the current point, the current position, and the current optimal direction to compute a likelihood for each direction in the steering function. Local observations and map features are embedded into a fixed-dimensional input vector. Convolutional layers can be co-trained with the recurrent model to take input images of the map or any other perceptual information. This eliminates the need for feature engineering and provides robustness to perceptual uncertainty. In addition, at each time step, the previous state is propagated and both a new state and a new direction are produced.

The recurrent models can in principle directly score a future state. In practice, we found that having an explicit mixture model that combines the optimal direction per the original RRT* with the direction preferred by the LSTM results in models which are easier to train. At each time step,

we use the recurrent network, a step of which is evaluated by the function η, to produce a mean and covariance matrix for a normal distribution from which new directions can be sampled. The likelihood of a steering move is computed as a mixture of

$$q(x_t|\eta(x_{t-1}, \mathtt{s}_{t-1}, \mathtt{Obs}, \phi)) \quad (1)$$

and the likelihood of following the RRT* direction, μ, is computed as a normal distribution $N(\mu, \sigma)$ where q is normal proposal distribution, η is the recurrent model (a function returning a mean direction and a covariance matrix), $\mathtt{s}$ is the state vector of the model, $\mathtt{Obs}$ is an embedding of the observation vector, and ϕ are the parameters of the recurrent model. For efficiency, we store the current state of the recurrent network at each node in the search tree and incrementally compute the likelihood of a path.

At training time, the network is supplied with a series of traces of successful plans. At each time step during training, stochastic gradient descent is used to maximize the likelihood shown in equation 1. In essence, we have samples from an n-dimensional normal distribution, where n is the size of the configuration space, along with the network which produced the mean and covariance matrix from which these samples were drawn at each time step. We then train this network to maximize the likelihood of the observed sequences.

III. EXPERIMENT RESULTS

The sequence models described above were implemented in PyTorch and integrated with the Open Motion Planning Library [16], using the provided Python bindings. We tested DeRRT* in the bug trap environment.

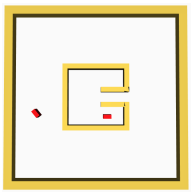
Bug trap requires that a 2D robot escape from an inner chamber through a narrow passage and then reach a goal in a large free space; see Fig. 2(a). This is made particularly hard by the shape of the exit which includes two dead ends. Most samples in the free space will lead to steering into these areas.

To quickly escape, one has to recognize not just the presence of a gap but the particular features of the central channel. In this experiment, we cotrain the convolutional layers of sequence models to learn to recognize the presence of relevant map features in order to reach a goal. This eliminates the need for feature engineering or any other annotation aside from a series of prior plans.

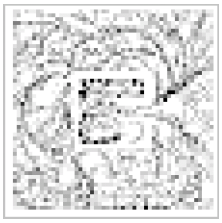
We randomly rotated and translated the trap and randomly sampled the starting position inside the trap and the goal configuration outside the trap to ensure that the training and test data are disjoint. Training samples were provided by running RRT* for 10000 planning steps on each problem instance. In total, we collected 1000 training sequences.

We take as an observation 21×21 local patch centered at the current node from a 110×110-sized map. We trained a GRU with a two-layer convolutional network, each layer containing a convolution followed by max polling. The convolutions used 3×3 filters with 32 and 64 output channels respectively. Max pooling used a 2×2 window with step size 2.

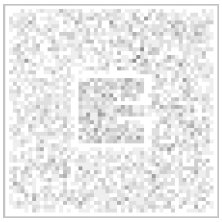
Fig. 3 shows the solution length as a function of the number of samples drawn. Already by 4000 samples the sequence-model guided planner is performing as well as RRT* with twice the number of samples. The colored regions show 95% confidence intervals. When considering more complex scenarios, such as an articulated robot with a complex mesh, this can have an even more significant impact as the more expensive collision checking can become a dominant concern in the runtime of sampling-based planners. Figs. 2(b) and 2(c) show heat maps where intensity is proportional to the density of nodes.



(a) Bug trap



(b) DeRRT*



(c) RRT*

Fig. 2. (a) The default bug trap example with its start and end position. Note the difficult central passage and dead-ends on either side of it. (b) A heat map of the DeRRT*/GRU search tree. It learns to exit the trap and focuses on sweeping in large arcs to locate the goal. (c) A heat map of the RRT* search tree. It spends more time in the trap and less time on finding the goal in the free space. Pure black represents allocating 0.2% of the samples inside the cell with linear interpolation to pure white.

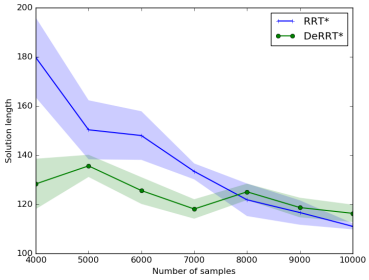


Fig. 3. Solution length as a function of the number of samples. DeRRT* is both more efficient and more stable.

IV. CONCLUSIONS

We have introduced DeRRT*, a sampling-based planner extending RRT* with a neural network in order to learn to plan more efficiently. This opens the door to using models that are successful in other areas, for example, compositional models in vision or sequence-to-sequence models in natural language. In the future, we expect that planners which understand more about their environments, perhaps by incorporating existing CNNs trained on large vision corpora, will navigate more efficiently. Similarly, sequence models which can capture existing knowledge about an environment and reason about the consequences of actions may be better suited to carrying out complex tasks.

REFERENCES

[1] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[2] M. Elbanhawi and M. Simic, "Sampling-based robot motion planning: A review," *IEEE Access*, 2014.

[3] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Batch informed trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs," in *ICRA*, 2015.

[4] F. Burget, M. Bennewitz, and W. Burgard, "BI^2 RRT*: An efficient sampling-based path planning framework for task-constrained mobile manipulation," in *IROS*, 2016.

[5] O. Adiyatov and H. A. Varol, "A novel RRT*-based algorithm for motion planning in dynamic environments," in *International Conference on Mechatronics and Automation*, 2017.

[6] C. Urmson and R. Simmons, "Approaches for heuristically biasing RRT growth," in *IROS*, 2003.

[7] S. R. Lindemann and S. M. LaValle, "Incrementally reducing dispersion by increasing voronoi bias in RRTs," in *ICRA*, 2004.

[8] L. E. Baum and T. Petrie, "Statistical inference for probabilistic functions of finite state markov chains," *The annals of mathematical statistics*, vol. 37, no. 6, pp. 1554–1563, 1966.

[9] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[10] N. Siddharth, A. Barbu, and J. Mark Siskind, "Seeing what you're told: Sentence-guided activity recognition in video," in *CVPR*, 2014.

[11] H. Yu, N. Siddharth, A. Barbu, and J. M. Siskind, "A compositional framework for grounding language inference, generation, and acquisition in video," *Journal of Artificial Intelligence Research*, vol. 52, pp. 601–713, 2015.

[12] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," in *CVPR*, 2015.

[13] T. A. Le, A. G. Baydin, and F. Wood, "Inference compilation and universal probabilistic programming," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, 2017.

[14] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the national academy of sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[15] K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder–decoder approaches," *Syntax, Semantics and Structure in Statistical Translation*, p. 103, 2014.

[16] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012.