State Supervised Steering Function for Sampling-based Kinodynamic Planning

Pranav Atreya
University of Texas at Austin
Austin, TX, United States
pranavatreya@utexas.edu

Joydeep Biswas
University of Texas at Austin
Austin, TX, United States
joydeepb@cs.utexas.edu

ABSTRACT

Sampling-based motion planners such as RRT* and BIT*, when applied to kinodynamic motion planning, rely on steering functions to generate time-optimal solutions connecting sampled states. Implementing exact steering functions requires either analytical solutions to the time-optimal control problem, or nonlinear programming (NLP) solvers to solve the boundary value problem given the system's kinodynamic equations. Unfortunately, analytical solutions are unavailable for many real-world domains, and NLP solvers are prohibitively computationally expensive, hence fast and optimal kinodynamic motion planning remains an open problem. We provide a solution to this problem by introducing State Supervised Steering Function (S3F), a novel approach to learn time-optimal steering functions. S3F is able to produce near-optimal solutions to the steering function orders of magnitude faster than its NLP counterpart. Experiments conducted on three challenging robot domains show that RRT* using S3F significantly outperforms state-of-the-art planning approaches on both solution cost and runtime. We further provide a proof of probabilistic completeness of RRT* modified to use S3F.

KEYWORDS

Kinodynamic Motion Planning; Learning Steering Functions; Sampling-based Planning

ACM Reference Format:
Pranav Atreya and Joydeep Biswas. 2022. State Supervised Steering Function for Sampling-based Kinodynamic Planning. In *Proc. of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2022), Online, May 9–13, 2022*, IFAAMAS, 9 pages.

1 INTRODUCTION AND RELATED WORK

This work tackles the kinodynamic motion planning (KDMP) problem, which is the problem of computing a kinodynamically feasible motion plan that takes a robot from an initial configuration to a goal region. We begin by formally defining the KDMP problem and then survey the various approaches to solving it.

Let X_C be the configuration space of the robot. The state space X is defined as the Cartesian product of X_C with X_D, the set of dynamics variables needed to fully describe the dynamics of the robot at any given instance in time. X_D typically consists of time derivatives of elements of X_C. Let U be the control space of the robot. The kinodynamic constraints are described by the differential equation $\dot{x}(t) = f(x(t), u(t))$, where $x(t) \in X$ and $u(t) \in U$. The KDMP problem differs from the purely kinematic motion planning (KMP) problem in that the KMP problem operates only on the configuration space X_C. Let $X_{\text{obs}} \in X$ be the set of obstacle-colliding states and let $X_{\text{free}} = X \backslash X_{\text{obs}}$ be the set of valid states. Let $x_{\text{init}} \in X_{\text{free}}$ be the initial state of the robot and let $X_{\text{goal}} \subset X_{\text{free}}$ be the goal region. The objective of the KDMP problem is to find a collision free path that takes the robot from x_{init} to X_{goal} while obeying the kinodynamic constraints. The solution to the KDMP problem is a mapping $c(t) : [0, t_f] \rightarrow U$ from time to control inputs such that applying $c(t)$ starting from the state x_{init} traces out a path $\xi(t) : [0, t_f] \rightarrow X_{\text{free}}$ such that $\xi(t_f) \in X_{\text{goal}}$. A motion plan is considered optimal if it minimizes some cost function $C(t_f, c, \xi)$. The time-optimal solution minimizes the total time t_f.

We review the state of the art approaches to solving the KDMP problem, including search-based planning, sampling-based planning, and learning-based solutions.

Search-based planning typically involves constructing a state lattice $G = (V, E)$ where $V \subset X_{\text{free}}$ and the edges E are pre-defined kinodynamically feasible motion primitives [23]. This lattice can then be searched using any graph search algorithm to obtain a solution. Increasing the resolution of the lattice increases the chances that a solution can be found, but comes with an exponential increase in computational cost. Finding a set of motion primitives that work well can also be difficult. Search-based planning algorithms are resolution optimal, in that they can find solutions that are optimal with respect to the discretization used.

Sampling-based planning makes use of a continually improving discretization of the state space through random sampling. One of the most effective sampling-based planning algorithms is the Rapidly Exploring Random Tree (RRT) [17] algorithm. The RRT algorithm works by incrementally sampling the state space and extending the nearest vertex in the tree towards that sample. Because this extension can be made by a random propagation of controls, the RRT algorithm can be applied to kinodynamic systems.

RRTs have also been integrated with machine learning approaches to solve the KDMP problem. One such work employs the k-nearest-neighbors algorithm within the RRT framework to approximate the cost-to-go function and expand vertices in the tree [29]. It however suffers from lack of optimality of computed trajectories and is only demonstrated to work for simple environments. Reinforcement Learning RRT (RL-RRT) [6] trains an RL agent to do local planning and uses an RRT to guide exploration. The resulting motion plan is suboptimal and since the RL local planner is trained on particular obstacle configurations, may not generalize well to new obstacle environments. Probabilistic Roadmap RL (PRM-RL) [8] also uses RL for local planning but maps sensor observations directly to actions and does not attempt to produce optimal trajectories.

Proc. of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2022), P. Faliszewski, V. Mascardi, C. Pelachaud, M.E. Taylor (eds.), May 9–13, 2022, Online.

RRT and the aforementioned RRT based algorithms do not produce optimal solutions. An alternative algorithm that produces optimal solutions while maintaining the computational efficiency of RRT is the RRT* algorithm [14]. The RRT* algorithm makes use of a rewiring step to ensure that the path from the root to any vertex in the tree is optimal with respect to the connections in the tree. Because of this, the RRT* algorithm is asymptotically optimal. Many variants of the RRT* algorithm exist that have proven to work well in practice. Informed RRT* [9] improves on RRT* by ensuring that after an initial solution has been found, only states that have the potential to improve the solution are considered as candidate vertices. The BIT* algorithm [10] integrates graph-based and sampling-based planning techniques to more efficiently find and improve on solutions to the planning problem.

One caveat of optimal sampling-based algorithms including RRT* and BIT* is that they all require an optimal steering function to connect states. For any two states $x_a, x_b \in X$, a steering function $S(x_a, x_b)$ produces a trajectory $T : [0, t_f] \rightarrow U$, a mapping from time to control inputs. Integrating T from x_a according to the equation of motion f produces a path $\Gamma : [0, t_f] \rightarrow X$, a mapping from time to states. An optimal steering function $S^*(x_a, x_b)$ produces a trajectory $T^* : [0, t_f] \rightarrow U$ and a path $\Gamma^* : [0, t_f] \rightarrow X$ that in addition to satisfying the aforementioned constraints, satisfies $\Gamma^*(t_f) = x_b$ and minimizes some cost function, most commonly time. There exist algorithms like Stable-Sparse RRT (SST) [19] and Asymptotically Optimal RRT (AO-RRT) [12] that do not require a steering function, but in practice they tend to take a significant amount of time to find good quality solutions. Analytical solutions to the steering function exist for some robots, such as those with linear dynamics [28], and so do iterative solutions for specific systems such as omnidirectional robots with bounded acceleration [2], but for most systems computing the optimal steering function requires a call to a computationally expensive nonlinear programming (NLP) solver. There are ways to decrease the computational overhead of NLP solvers to make planning tractable [30], but the NLP solver still remains a significant bottleneck. Previous work has explored whether the steering function can be learned [32]. The learning setup used however was unable to connect arbitrary start and goal states, a necessity if the steering function is to be used in an optimal sampling-based planning algorithm.

Reinforcement learning has also been applied to the KDMP problem. One approach to KDMP for linear systems uses continuous-time Q-learning [16] to deal with dynamics whose differential equations of motion are inaccurate or unreliable. Some have also proposed formulating the KDMP problem entirely as a Markov Decision Process (MDP), where the solution KDMP policy is learned by RL [5].

Learning optimal control policies is a research area that has also been recently explored. Past works [11] [27] [25] [26] have attempted to train a neural network to learn to produce optimal controls. All of these works however keep the goal state fixed, and so a new policy would need to be learned for every goal state.

Optimization-based planning methods rely on numerical optimization to find a solution to the goal that minimizes some cost objective. Example works that fall under this category include GuSTO [4], CHOMP [24], and STOMP [13]. While such optimization-based methods are effective at finding solutions given good initialization, they find difficulty in handling cases where initial solutions are unknown, or when the optimization objective function has local minima (often due to obstacles).

Integrated planning and learning approaches have recieved significant attention lately. Search on the Replay Buffer (SoRB) [7] demonstrates how the success rate of goal-conditioned RL on long horizon tasks can be improved by adding a planning component. SoRB however is unable to provide theoretical guarantees on completeness and faces difficulty when run on unseen environments. One approach [1] uses precomputation and machine learning to enable real-time kinodynamic planning for quadrotors. It is able to avoid solving two-point boundary value problems directly on quadrotor dynamics by using minimum snap polynomial splines, a technique that only works for a limited class of systems. Model-Predictive Motion Planning Networks (MPC-MPNet) [18] proposes the integration of multiple neural components along with Model Predictive Control to solve the kinodynamic motion planning problem. The algorithm is compared with SST and is shown to have faster planning times. It however is unable to produce lower cost paths than SST and drops in performance on unseen environments.

While many approaches exist for kinodynamic planning, none so far are able to find low cost solutions in a computationally efficient manner. Approaches either sacrifice low solution cost or performance in pursuit of the other. We propose with this work that both are attainable. In contrast to many learning approaches, our work is also agnostic to obstacle configurations, and so generalizes well to new environments.

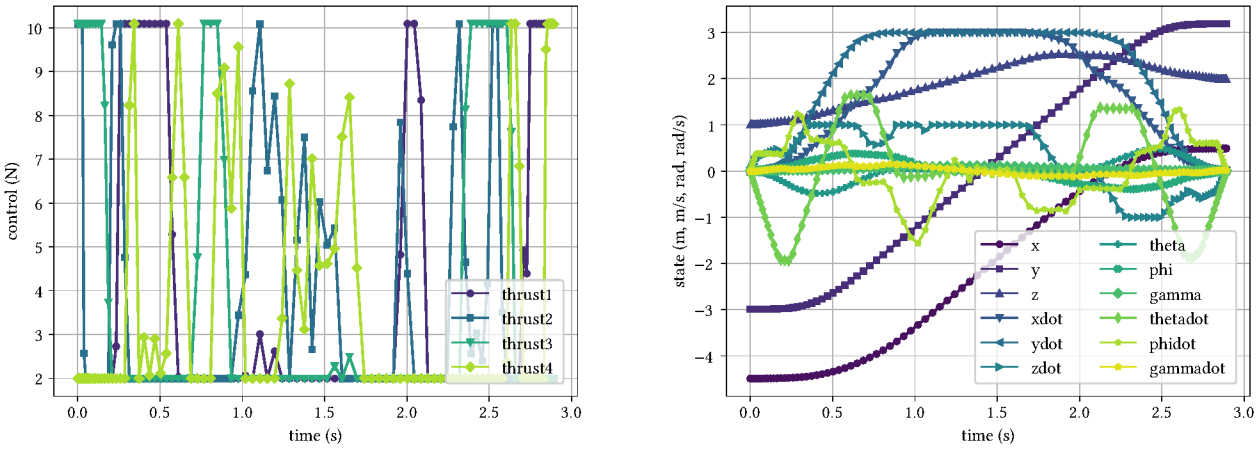
In summary, in this paper we contribute: 1) State Supervised Steering Function (S3F), a learning-based technique to efficiently compute the steering function required by optimal sampling-based planners; 2) S3F-RRT*, a probabilistically complete RRT* algorithm that uses S3F as its steering function; and 3) Empirical results for three kinodynamically-complex robots that demonstrate that S3F-RRT* outperforms state-of-the-art kinodynamic planners.

2 KINODYNAMIC PLANNING WITH STATE SUPERVISED STEERING FUNCTION

Recall from earlier that given two arbitrary states $x_a, x_b \in X$ the optimal steering function $S^*(x_a, x_b)$ produces a trajectory T^* that optimally connects these two states. We are interested in learning a function $\tilde{S}$ that approximates S^* such that $\tilde{S}(x_a, x_b)$ produces a near-optimal trajectory $\tilde{T} \simeq T^*$. The control trajectory $\tilde{T}$ can be integrated to obtain a path $\tilde{\Gamma}$.

2.1 Steering Function Formulation

Rather than learning $\tilde{S}$ that produces $\tilde{T}$ directly, we simplify the learning problem by constructing $\tilde{T}$ in an iterative manner. This can be done by learning a policy $\pi : X \times X \rightarrow U$ where π takes as input the current state of the robot x_t and the goal state x_b and produces as output a constant-time control input u to be executed for a fixed period of time τ, resulting in a new state x_{t+1}. Iteratively calling π for a fixed number of iterations n results in the generation of a piecewise constant control function that we denote $T_{\max}$. Integrating $T_{\max}$ from the start state x_a yields the state function $\Gamma_{\max}$.



(a) Control function **(b) State function**

Figure 1: Quadrotor optimal control and state functions

$\tilde{T}$ can be obtained from $T_{\max}$ by discarding from $T_{\max}$ all controls past the time when the robot has reached the goal. To be able to do this, $n\tau$, the duration of $T_{\max}$, needs to be greater than the time it takes to connect any two states in X optimally. The next step is to determine when $T_{\max}$ actually reaches the goal. The naive approach is to simply select the time at which $\Gamma_{\max}$ is closest to x_b where closeness is defined using Euclidean distance. The problem with this approach is that our trajectories not only need to reach the goal but also be optimal with respect to the time to goal. Let's say for one particular trajectory the robot reaches a distance d_1 from the goal at time t_1 and a distance d_2 from the goal at time t_2. If d_2 is the closest distance, then we are guaranteed to pick t_2 as our ending time, even if d_2 is marginally less than d_1. However it may be possible that t_2 is significantly greater than t_1, and so just to reach a little closer to the goal we're sacrificing significant time optimality. This type of analysis motivates the solution to this problem. Since there are in essence two objectives that we are optimizing over when selecting the end time – distance to goal and time to reach goal – we should construct a reward function that fairly takes into account both. The following reward function $R(t)$ does exactly this:

$$R(t) = \alpha \frac{||x_a - x_b|| - ||\Gamma_{\max}(t) - x_b||}{||x_a - x_b||} - t + R_b(\Gamma_{\max}(t), x_b)$$
$$R_b(\Gamma_{\max}(t), x_b) = \begin{cases} \beta, \text{ if } ||\Gamma_{\max}(t) - x_b|| \leq \mu \\ 0, \text{ otherwise} \end{cases} \quad (1)$$

The first term is a normalized difference of potential functions, and is maximized when the candidate terminal state is situated at the goal. The use of such potential functions was first introduced as a policy invariant mechanism for reward shaping [21]. The second term, $-t$, takes into account the second objective: minimizing the time to the goal. Finally the third term provides an additional incentive if the candidate terminal state is very close ($\leq \mu$ distance away) to the goal. The hyperparameters α, β, and μ are positive constants which can be tuned to adjust the relative weights of the three terms.

For all time points which this reward function is calculated, the end time will be the time with the greatest reward. $\tilde{T}$ can then be obtained by discarding all control inputs in $T_{\max}$ after the end time.

2.2 Learning the Policy

The previous section showed how the steering function $\tilde{S}$ can be constructed from a learned policy π. We next present how π is learned.

We employ a supervised learning approach to learn π. Since the end goal is to learn the optimal steering function, our dataset consists of solutions to the optimal steering function for a large number of start and goal states. This dataset, generated by an NLP solver, consists of a series of trajectories each described by a tuple (T^*, Γ^*, t_f). Here $T^* : [0, t_f] \rightarrow U$ and $\Gamma^* : [0, t_f] \rightarrow X$ are the control and state functions introduced earlier. We used the PSOPT [3] optimal control library to generate the trajectories. The start and goal states for each trajectory in the dataset are sampled uniformly at random from the full state space to ensure good state space coverage.

To learn π using this dataset we employ the fact that π is used to generate control and state functions $\tilde{T}$ and $\tilde{\Gamma}$. The arguably simplest approach is to have $\tilde{T}$ imitate T^* for each start and goal pair in the dataset. The discrepancy in the fact that $\tilde{T}$ is piecewise constant whereas T^* is continuous can be accounted for by simply averaging controls in T^* at each length τ time interval. π would then be directly supervised by the averaged constant controls in T^*. Although straightforward, this approach fails to learn a well-performing policy. The primary reason for this is that the learning problem involves the approximation of a highly discontinuous function. π is tasked with learning the optimal control function which for many kinodynamic systems is a bang-bang control function. Figure 1a shows an example of this for the quadrotor robot – such discontinuous control functions are hard to represent and learn directly, even by supervised learning.

The solution to this problem is to not use the optimal control function T^* to supervise the learning, but to instead use the optimal

S3F-RRT*()

$V \leftarrow \{x_{\text{init}}\}, E \leftarrow \emptyset$
for $i = 1..n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree()}$
  $x_{\text{parent}} \leftarrow \emptyset, x_{\text{ext}} \leftarrow \emptyset, c_{\text{min}} \leftarrow \infty$
  $X_{\text{near}} \leftarrow \texttt{NearTo}(G = (V, E), x_{\text{rand}})$
  for each $x \in X_{\text{near}}$ **do**
    $T \leftarrow \texttt{Steer}(x, x_{\text{rand}})$
    $x_{\text{new}} \leftarrow \texttt{EndState}(x, T)$
    $c_{\text{traj}} \leftarrow \texttt{SteeringCost}(T)$
    $b \leftarrow \texttt{Dist}(x_{\text{new}}, x_{\text{rand}}) < r_{\text{error}} \wedge \texttt{ObstacleFree}(x, T)$
    if $\texttt{Cost}(x) + c_{\text{traj}} < c_{\text{min}} \wedge b$ **then**
      $x_{\text{parent}} \leftarrow x$
      $x_{\text{ext}} \leftarrow x_{\text{new}}$
      $c_{\text{min}} \leftarrow \texttt{Cost}(x) + c_{\text{traj}}$
    end if
  end for
  if $c_{\text{min}} \neq \infty$ **then**
    $V \leftarrow V \cup \{x_{\text{ext}}\}$
    $E \leftarrow E \cup \{(x_{\text{parent}}, x_{\text{ext}})\}$
  end if
  $\texttt{Rewire}(V, E, x_{\text{ext}})$
end for
return $G = (V, E)$

Rewire(V, E, x_{ext})

$X_{\text{near}} \leftarrow \texttt{NearFrom}(G = (V, E), x_{\text{ext}})$
for each $x \in X_{\text{near}}$ **do**
  $T \leftarrow \texttt{Steer}(x_{\text{ext}}, x)$
  $x_{\text{new}} \leftarrow \texttt{EndState}(x_{\text{ext}}, T)$
  $c_{\text{traj}} \leftarrow \texttt{SteeringCost}(T)$
  $b \leftarrow \texttt{Dist}(x_{\text{new}}, x) < r_{\text{error}} \wedge \texttt{ObstacleFree}(x_{\text{ext}}, T)$
  if $\texttt{Cost}(x_{\text{ext}}) + c_{\text{traj}} < \texttt{Cost}(x) \wedge b$ **then**
    $V \leftarrow V \backslash \{x\} \cup \{x_{\text{new}}\}$
    $E \leftarrow E \backslash \{(\texttt{Parent}(x), x)\} \cup \{(x_{\text{ext}}, x_{\text{new}})\}$
    $\texttt{PropagateRewiring}(x, x_{\text{new}})$
  end if
end for

PropagateRewiring(x, x_{new})

for each $x_{\text{child}} \in \texttt{Children}(x)$ **do**
  $T \leftarrow \texttt{Trajectories}(x, x_{\text{child}})$
  if $\texttt{ObstacleFree}(x_{\text{new}}, T)$ **then**
    $x_{\text{next}} \leftarrow \texttt{EndState}(x_{\text{new}}, T)$
    $V \leftarrow V \backslash \{x_{\text{child}}\} \cup \{x_{\text{next}}\}$
    $E \leftarrow E \backslash \{(x, x_{\text{child}})\} \cup \{(x_{\text{new}}, x_{\text{next}})\}$
    $\texttt{PropagateRewiring}(x_{\text{child}}, x_{\text{next}})$
  else
    $\texttt{DeleteSubtree}(x_{\text{child}})$
  end if
end for

Figure 2: S3F-RRT* Algorithm

state function Γ^*. We term this approach State Supervised Steering Function (S3F). Due to the differential equation f that defines the kinodynamic constraints, state functions are guaranteed to be differentiable (and thus continuous), making learning the optimal state function a feasible problem. Figure 1b shows an example of such a state function for the quadrotor robot – note that despite the associated control function (Figure 1a) being discontinuous, the state function is smooth and continuous. The goal now is to have $\tilde{\Gamma}$ imitate Γ^* for each trajectory in the dataset. This can be done by ensuring that for various time points t in the range $[0, t_f]$, $\tilde{\Gamma}(t) = \Gamma^*(t)$. Recall that $\tilde{\Gamma}$ is only obtained by integrating $\tilde{T}$. This can be accounted for with the following procedure: sample a series of time points $(t_0 ... t_k)$ in the range $[0, t_f - \tau]$. For each time point t, assume that the robot is currently at $\Gamma^*(t)$. If $\tilde{\Gamma}$ is to imitate Γ^*, the robot should be at $\Gamma^*(t + \tau)$ at time $t + \tau$. The actual location of the robot at this time under the current policy π can be calculated by evaluating $F(\Gamma^*(t), \pi(\Gamma^*(t), x_{t_f}))$ where x_{t_f} is the goal state of the trajectory and $F : X \times U \rightarrow X$ is an integration function that given a current state and a constant control, integrates the differential equation of motion f to compute the state τ units of time later. To get $\tilde{\Gamma}$ to imitate Γ^* we can thus optimize the following learning objective:

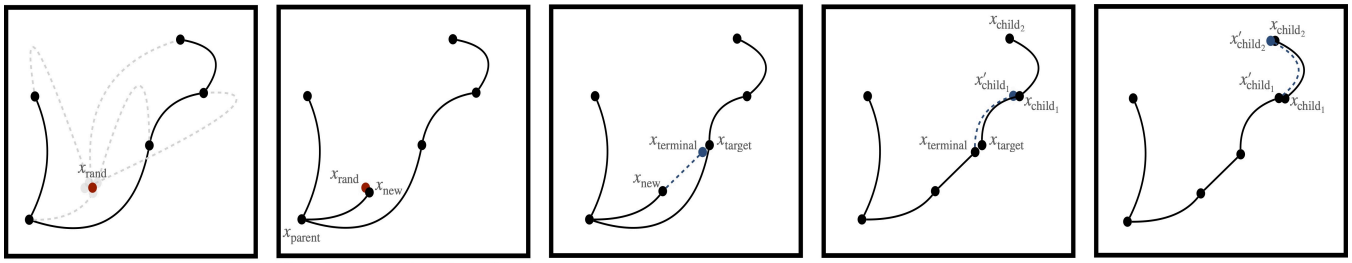
$$\arg\min_{\theta} \sum_{\Gamma^* \in D} \sum_{t \in (t_0, \ldots, t_k)} [F(\Gamma^*(t), \pi(\Gamma^*(t), x_{t_f})) - \Gamma^*(t + \tau)]^2 \quad (2)$$

where θ is the parameter set of π and D is the dataset of optimal trajectories. The key takeaway from this learning procedure is that we are learning π indirectly. π is a component of a state function that we are training to be optimal, and by learning this state function we are indirectly learning the control function π.

2.3 Sampling-based Planning With Learned Steering Functions: S3F-RRT*

We present S3F-RRT*, a sampling-based planning algorithm that uses the learned steering function to solve the optimal kinodynamic motion planning problem. S3F-RRT* uses S3F as the steering function, and employs a modified rewiring procedure to overcome any potential local inaccuracies in S3F's trajectories.

Figure 2 presents the algorithmic formulation of S3F-RRT*. Figure 3 shows a visualization of what goes on in each S3F-RRT* iteration. Each iteration begins by sampling a random collision-free state x_{rand}. The `NearTo` function is then called to obtain the set of all vertices in the current RRT* tree that are near x_{rand}. A state is considered to be near x_{rand} if the time of the optimal trajectory from that state to x_{rand} is below some threshold. Each state in X_{near} is then evaluated as a possible parent to x_{rand}. $\texttt{Steer}(x, x_{\text{rand}})$ invokes S3F to compute a control function T that connects x to x_{rand}. To determine x_{new}, where the trajectory actually ends, $\texttt{EndState}(x, T)$ integrates T from x. $\texttt{SteeringCost}(T)$ returns the cost of the trajectory T, which for a time-optimal planning problem is simply the duration of T. $\texttt{Cost}(x)$ returns the cost of going from the start state to x in the current RRT* tree. The `Dist` function returns the Euclidean distance between two states and is used to ensure that the terminal state of the trajectory is close enough to the target state. $\texttt{ObstacleFree}(x, T)$ integrates the control function T beginning at x to obtain a state function that maps time to states. `ObstacleFree`



Sample a random state x_{rand} shown in red. Attempt connections from all nearby states currently in the tree to x_{rand} using S3F. The start and goal states of the planning problem are not shown.

Pick the connection that allows reaching x_{rand} with minimal cost and is obstacle free. The state actually reached x_{new} need not equal x_{rand}, but it needs to be close.

For each nearby state in the tree (x_{target} is one example), see if it can be reached faster by going through x_{new}. Like before, the connections are generated by S3F. The path to x_{target} will be replaced by the faster one.

If $x_{terminal} \neq x_{target}$, x_{target} 's children need to be shifted. The trajectory $(x_{target}, x_{child_1})$ was generated in a previous iteration by S3F. We can re-execute this trajectory beginning at $x_{terminal}$.

Recursively propagate the rewiring to children deeper in the subtree. Similar to the previous step, the connection $(x_{child_1}, x_{child_2})$ that was generated earlier by S3F will be re-executed beginning from x'_{child_1}.

Figure 3: Illustration of the steps that take place in one iteration of S3F-RRT*

then ensures that every state in this state function does not collide with obstacles.

After the best parent has been found and the state has been added to the tree, the rewiring procedure is invoked. Here, the set X_{near} is constructed by calling $\texttt{NearFrom}(G = (V, E), x_{\text{ext}})$. The difference between `NearFrom` and `NearTo` is that $\texttt{NearFrom}(G = (V, E), x_{\text{ext}})$ considers connections from x_{ext} to other states as opposed to from other states. $\texttt{Parent}(x)$ returns the parent of x in the current RRT* tree.

The rewiring procedure internally calls `PropagateRewiring`. $\texttt{Children}(x)$ returns the set of all children states to x in the current RRT* tree. $\texttt{Trajectories}(x, x_{\text{child}})$ returns the control function that was computed earlier by S3F to connect x and x_{child}.

One of the key differences between this algorithm and the original RRT* algorithm is the absence in this algorithmic formulation of finding the nearest state. In the original RRT* algorithm, after a state is randomly sampled, the nearest state in the tree is selected as a source of expansion. A new state is obtained by extending the nearest state towards the randomly sampled state up to a distance η, and the resultant state is used as the target for the subsequent steering function evaluations. We entirely eliminate this component of the algorithm for simplicity, a modification that was first proposed in Kinodynamic RRT* [28]. This modification is known to not hurt theoretical asymptotic optimality of the RRT* algorithm. The main other difference in this algorithm is a series of modifications that deal with the fact that the learned steering function will reach within an error radius of the goal state. Notable among these is the existence of the `PropagateRewiring` procedure.

2.4 Correctness of S3F-RRT*

There are two criteria for correctness: solutions returned by S3F-RRT* must satisfy the kinodynamic constraints and must avoid obstacles. Any operation on the S3F-RRT* tree (such as rewiring) can be reformulated as a sequence of state addition and state deletion operations. State deletion by default cannot violate correctness. State addition also satisfies correctness because (1) a state is only added to the tree if the path from the parent to the state is collision free and (2) the path from the parent to the state is generated by integrating the differential equation of motion, implying that the path to the state satisfies kinodynamic constraints. Thus S3F-RRT* is correct.

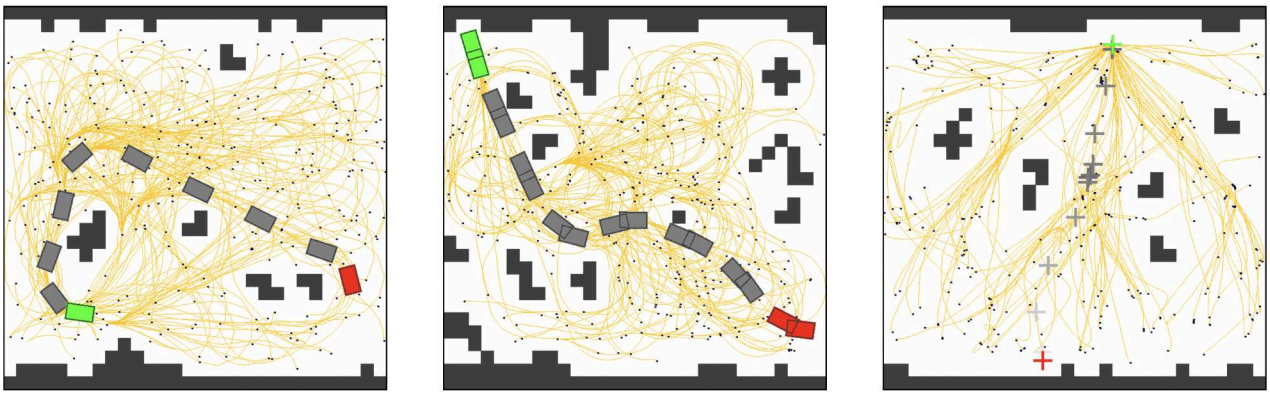
2.5 Probabilistic Completeness Proof of S3F-RRT*

Here we present a summary of the proof of probabilistic completeness (PC) of the S3F-RRT* algorithm. S3F-RRT* is a modification of the original RRT* algorithm [14] designed to make use of a learned steering function. The proof largely follows the structure of the proof of probabilistic completeness of geometric RRT [15], though significant modifications have been made to take into account the presence of kinodynamic constraints and the use of a learned steering function. The full proof can be found in the supplementary materials.

Let $c^*(x_a, x_b)$ denote the cost of the optimal trajectory from x_a to x_b, or equivalently the kinodynamic distance from x_a to x_b. We assume that c^* obeys the triangle inequality, that is, $c^*(x_a, x_b) \leq c^*(x_a, x) + c^*(x, x_b)$ for all $x \in X$. Let $\tilde{S}$ be a learned steering function. We assume that with nonzero probability p, $\tilde{S}(x_a, x_b)$ yields a state function $\tilde{\Gamma}$ that satisfies $c^*(\tilde{\Gamma}(t), x_b) \leq c^*(x_a, x_b)$ for all $t \in [0, t_f]$. This assumption in essence states that every state along the path produced by $\tilde{S}$ is kinodynamically closer to the goal state than the start state is. For a steering function trained to be optimal, this is a reasonable assumption.

We will use $B_r(x)$ to denote the subset of the state space X defined by $\{x' | c^*(x', x) \leq r\}$. For simplicity, we assume that there exist $\delta_{\text{goal}} > 0, x_{\text{goal}} \in X_{\text{goal}}$ such that $B_{\delta_{\text{goal}}}(x_{\text{goal}}) \subseteq X_{\text{goal}}$. We denote this simplified goal region $B_{\delta_{\text{goal}}}(x_{\text{goal}})$ as X^*_{goal}. The goal of the motion planning problem is to find a kinodynamically feasible path $\pi : [0, t_\pi] \rightarrow X_{\text{free}}$ such that $\pi(0) = x_{\text{init}}$ and $\pi(t_\pi) \in X^*_{\text{goal}}$. The clearance of π is the maximal δ_{clear} such that $B_{\delta_{\text{clear}}}(\pi(t)) \in X_{\text{free}}$ for all $t \in [0, t_\pi]$.

We assume for this proof that there exists a valid trajectory $\pi : [0, t_\pi] \rightarrow X_{\text{free}}$ with clearance $\delta_{\text{clear}} > 0$. Without loss of generality, assume that $\pi(t_\pi) = x_{\text{goal}}$, i.e., the trajectory terminates at the center of the goal region. Let L be the total cost of π, and let $v = min(\delta_{\text{clear}}, \delta_{\text{goal}})$. Let $m = \frac{3L}{v}$. Define a sequence of $m + 1$ points $x_0 = x_{\text{init}}, ..., x_m = x_{\text{goal}}$ along π such that the cost of traversal from one point to the next is $\frac{v}{3}$. Therefore, $c^*(x_i, x_{i+1}) \leq \frac{v}{3}$ for every $0 \leq i < m$. We will now prove that as the number of iterations increases, the S3F-RRT* algorithm will generate a path passing through the vicinity of these $m + 1$ points with probability asymptotically approaching one.



(a) Dubin's Car with Acceleration (b) Tractor Trailer (c) Quadrotor

Figure 4: Sample planning trees after running S3F-RRT* on the three robot domains. The best solution found from the (green) start state to the (red) goal state is shown explicitly. A large spacing between consecutive gray states indicates a high velocity. In the planning trees, the dots are the vertices of the tree and the orange connections are the edges. In (c), dark gray states are of low elevation and light gray states are of high elevation.

Lemma 2.1. *Suppose that S3F-RRT* has reached $B_{\frac{v}{3}}(x_i)$, that is, its tree contains a vertex x_i' such that $x_i' \in B_{\frac{v}{3}}(x_i)$. If $x_{\text{rand}} \in B_{\frac{v}{3}}(x_{i+1})$ and $c^*(x_i, x_{\text{rand}}) \leq \frac{v}{3}$ (equivalently $x_i \in B_{\frac{v}{3}}(x_{\text{rand}})$), then the path from the nearest neighbor x_{near} to x_{rand} lies entirely in X_{free} with probability p.*

Proof. See supplementary materials. □

Theorem 2.2. *The probability that S3F-RRT* fails to reach X^*_{goal} from x_{init} after k iterations is at most ae^{-bk}, for some constants $a, b \in \mathbb{R}_{>0}$.*

Proof. See supplementary materials for full proof of Theorem 2.2. Here we present an overview. Assume that $B_{\frac{v}{3}}(x_i)$ already contains an S3F-RRT* vertex. Let r_i be the probability that in the next iteration a S3F-RRT* vertex will be added to $B_{\frac{v}{3}}(x_{i+1})$. The proof in essence relies on the fact that with Lemma 2.1 in place, it can be shown that the probability r_i is nonzero and is independent of the number of S3F-RRT* iterations k. In order for the S3F-RRT* algorithm to reach X^*_{goal} from x_{init}, a S3F-RRT* vertex must be added to $B_{\frac{v}{3}}(x_{i+1})$ m times for $0 \leq i < m$. If we let r be the minimum of the transition probabilities $\{r_i | \forall i (0 \leq i < m)\}$, reaching the goal can be described as k Bernoulli trials with success probability r, where the goal is reached after m successful outcomes. With this formulation it can be shown that the probability the goal is not reached decays to zero exponentially with k, and thus S3F-RRT* is probabilistically complete. □

3 EXPERIMENTAL RESULTS

We compared S3F to the current state of the art on three challenging problem spaces: Dubin's car with acceleration, tractor trailer, and quadrotor robots. For each problem space, we solve a series of minimum-time motion planning problems using S3F-RRT*, RRT* using NLP for steering, RRT, and SST. The BARN dataset [22] was used to obtain realistic, obstacle dense maps to run the comparisons on. Figure 4 depicts sample solutions and their planning trees found by S3F-RRT* on the three problem spaces.

3.1 Robot Kinodynamics

The three robot models used in this paper are the Dubin's car with acceleration, tractor trailer, and quadrotor robots. Here we introduce these robot domains in more detail along with their equations of motion.

Dubin's Car with Acceleration: $X = [x, y, \theta, v], U = [a, k]$

$$\begin{aligned} \dot{x} &= v\cos(\theta) & \dot{y} &= v\sin(\theta) \\ \dot{\theta} &= vk & \dot{v} &= a \end{aligned} \tag{3}$$

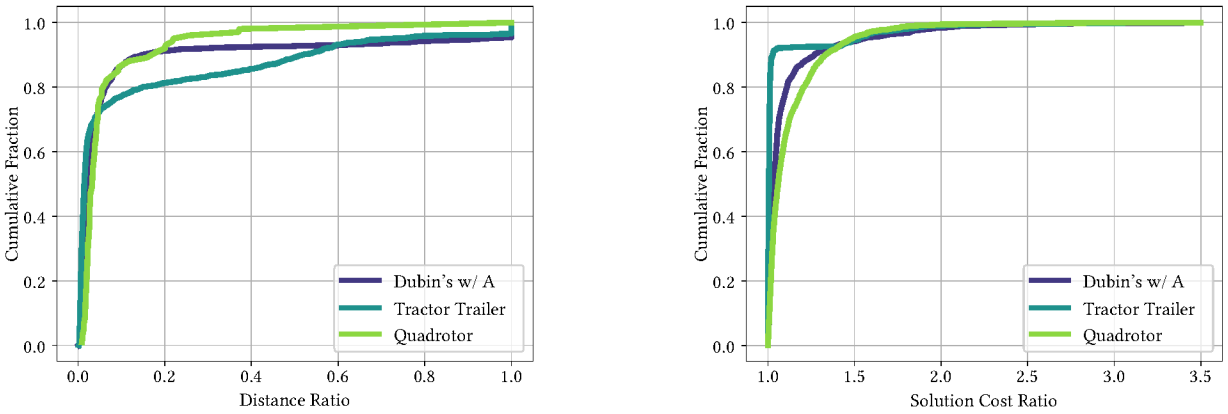
The Dubin's car with acceleration is a curvature constrained robot car. x, y, θ, and v are the x-position, y-position, orientation, and velocity of the car, and a and k are the acceleration and curvature control inputs. The motion of the car is subject to the curvature constraint $|k| \leq |\frac{1}{r_{\min}}|$ where $r_{\min}$ is the minimum radius of turning.

Tractor Trailer: $X = [x, y, \theta, v, \alpha], U = [a, \phi]$

$$\begin{aligned} \dot{x} &= v\cos(\theta) & \dot{v} &= a \\ \dot{y} &= v\sin(\theta) & \dot{\alpha} &= (\frac{v}{D})\sin(\theta - \alpha) \\ \dot{\theta} &= (\frac{v}{L})\tan(\phi) \end{aligned} \tag{4}$$

The tractor trailer robot consists of a four wheeled robot car pulling a two wheeled trailer. The robot car in isolation has the same dynamics as the Dubin's car with acceleration. x, y, θ, v, and α are the x-position, y-position, orientation, of the car, velocity of the car, and orientation of the trailer, respectively. The control inputs are a and ϕ which represent the acceleration and heading. L is the distance between the front and rear axles of the robot car, and D is the length of the rod connecting the trailer with the car.

Quadrotor: $X = [x, y, z, \dot{x}, \dot{y}, \dot{z}, \theta, \phi, \gamma, \dot{\theta}, \dot{\phi}, \dot{\gamma}], U = [\tau_1, \tau_2, \tau_3, \tau_4]$



(a) Distance to goal CDFs

(b) Cost ratio CDFs

Figure 5: CDFs of (a) the distance remaining to the goal for the three robot domains and (b) the ratios of costs of S3F's solutions over NLP's solutions for the three robot domains. Plots depict 1500 data points.

$$
\begin{aligned}
\ddot{x} &= \frac{1}{w}(\cos\theta\sin\phi\cos\gamma + \sin\theta\sin\gamma)(\tau_1+\tau_2+\tau_3+\tau_4)\\
\ddot{y} &= \frac{1}{w}(\cos\theta\sin\phi\sin\gamma - \sin\theta\cos\gamma)(\tau_1+\tau_2+\tau_3+\tau_4)\\
\ddot{z} &= \frac{1}{w}(\cos\theta\cos\phi)(\tau_1+\tau_2+\tau_3+\tau_4)\\
\ddot{\theta} &= \frac{L(\tau_1-\tau_3) - 2wL^2\dot{\phi}\dot{\gamma}}{2wr^2/5 + 2wL^2}\\
\ddot{\phi} &= \frac{L(\tau_2-\tau_4) + 2wL^2\dot{\theta}\dot{\gamma}}{2wr^2/5 + 2wL^2}\\
\ddot{\gamma} &= \frac{b(\tau_1-\tau_2+\tau_3-\tau_4)}{2wr^2/5 + 4wL^2}
\end{aligned}
\tag{5}
$$

The quadrotor is a lightweight, agile robot heavily used in research and industrial applications. x, y, and z represent the Cartesian coordinates of the quadrotor. θ, ϕ, and γ represent the pitch, roll, and yaw, respectively. w is the weight of the quadrotor, L is the length of an arm, r is the radius of the sphere representing the center blob of the quadrotor, g is the gravitational acceleration, and b is a constant. τ_1 through τ_4 represent the thrusts generated by each of the four motors and are the control inputs for the quadrotor.

3.2 S3F Evaluation

We evaluate the learned steering function for each of the three problem spaces on its ability to consistently reach the goal and on the time optimality of its solutions.

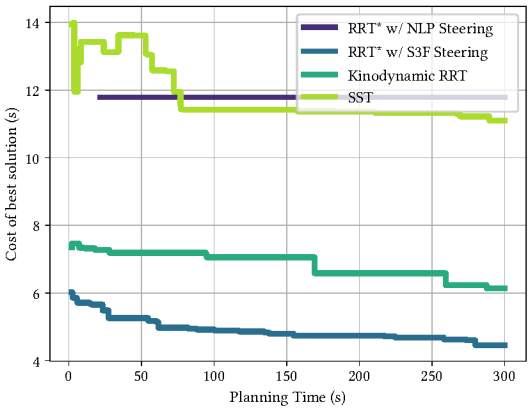
We measured the former by computing for 1500 steering function queries how much of the initial distance between the start and goal states was not traversed in the produced trajectory. Mathematically this is expressed by $\frac{d_f}{d_s}$ where d_s is the distance from the start to the goal and d_f is the distance from the end state of the trajectory produced by S3F to the goal. A value of 0 indicates that the goal is reached exactly. Figure 5a depicts the cumulative distribution function (CDF) plot of 1500 evaluations of this expression. To list a few numbers, we see that for the Dubin's car with acceleration problem space, 85% of the trajectories are within 10% of d_s to the goal; for the tractor trailer problem space, 75% of the trajectories are within 10% of d_s to the goal; and for the quadrotor problem space, 85% of the trajectories are within 10% of d_s to the goal. These results indicate that on average, S3F is able to reach very close to the desired goal.

Measuring the quality of the solutions produced by S3F in terms of time optimality can easily be done by comparing S3F's trajectory costs with the optimal costs as determined by the NLP solver. Figure 5b shows the CDF plots of the ratios of the cost of solutions of trajectories produced by S3F with the cost of solutions of trajectories produced by the NLP solver. An ideal value of the ratio is close to 1. Results are depicted for 1500 trajectories. We can see that for all three problem spaces the trajectories are very close to optimal. Specifically, for the Dubin's car with acceleration problem space, 90% of S3F's trajectories have costs that are less than 1.25 times as suboptimal as the optimal cost; for the tractor trailer problem space, 90% of S3F's trajectories have costs that are less than 1.25 times as suboptimal as the optimal cost; and for the quadrotor problem space, 80% of S3F's trajectories have costs that are less than 1.25 times as suboptimal as the optimal cost.

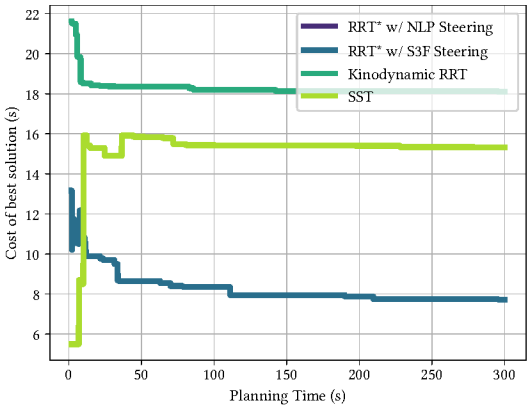
3.3 Planning Comparisons

Here we compare planning using the S3F-RRT* algorithm against RRT* with NLP steering, RRT, and SST. By comparing against SST, we can omit a comparison against AO-RRT since previous work [20][31] has shown that empirically SST outperforms AO-RRT. Comparisons are done on all three problem spaces. Starting and ending points for each planning query are sampled randomly across five different maps.

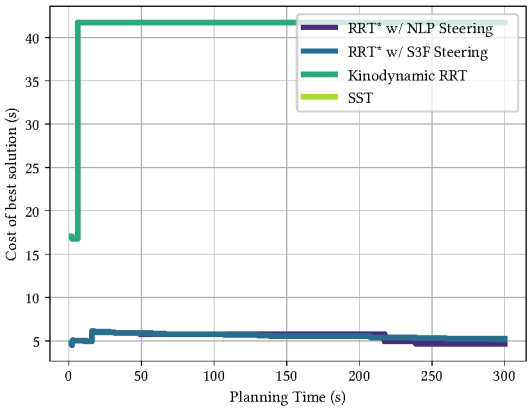
Figures 6a, 6b, and 6c plot the average cost of best solution found by each of the algorithms against wall-clock time for the different robot domains. Results of 25 planning problems are depicted in each plot. In many cases, it takes the algorithms quite a long time to find their first solution. This causes the graphs to not be monotonically



(a) Dubin's Car with Acceleration average solution cost vs runtime



(b) Tractor Trailer average solution cost vs runtime



(c) Quadrotor average solution cost vs runtime

Figure 6: Comparison of the planning results of the S3F-RRT*, NLP-RRT*, RRT and SST planning algorithms on the three robot domains. Planning time is plotted against the cost of the best solution found thus far, averaged across 25 planning trials.

	Dubin's Car		Tractor Trailer		Quadrotor	
	f (%)	t (s)	f (%)	t (s)	f (%)	t (s)
RRT	4	0.251	0	0.083	10	0.789
S3F-RRT*	20	0.480	28	1.910	0	16.386
NLP-RRT*	92	21.307	100	–	70	168.656
SST	12	10.013	48	11.075	100	–

Figure 7: Failure rate (f) and time to first solution (t) of different planners

decreasing, since the cost of best solution before a solution is found cannot be plotted. We observe in the graphs that S3F-RRT* is able to find solutions very quickly, and is able to find better solutions than the baseline algorithms irrespective of the amount of computation time given. One of the key reasons why this occurs is that due to the speed of evaluation of the learned steering function, many more RRT* iterations can be completed in a unit time as opposed to NLP-RRT*, enabling the more rapid exploration of the state space by the sampling-based planning algorithm. Furthermore, because S3F does a good job at approximating the optimal steering function, waypoints in the final planned path are connected in a near-optimal fashion. This is something that the baseline algorithms like SST and RRT are unable to do, because in these algorithms waypoints are connected by randomly sampled trajectories, resulting in significant suboptimality.

Figure 7 depicts the rate of failure and average time to first solution of the different algorithms. The time to first solution differs from the cost of best solution in Figure 6 in that the former only considers how long it takes to find the first feasible solution. We can see that across the different problem spaces, S3F-RRT* has lower rates of failure than SST and NLP-RRT*. Figure 6c seems to show that S3F-RRT* and NLP-RRT* have similar performance on the quadrotor domain, but the data in the table shows that S3F-RRT* has a much lower rate of failure and finds its first solution far more quickly, demonstrating that S3F-RRT* indeed has better performance. S3F-RRT* on average is able to find its first solution almost as quickly as RRT. It takes on average an order of magnitude more time for SST and NLP-RRT* to find their first solutions.

4 CONCLUSION

We introduced State Supervised Steering Function, a learning based approximation of the optimal steering function for complex kinodynamic systems. We demonstrate that the learned steering function can be used in sampling-based planners to achieve superior planning results. This superiority is assessed on metrics of time to find solution and quality of solution for three challenging robot domains. Finally, we present a proof of probabilistic completeness of RRT* using S3F, demonstrating its theoretical soundness.

ACKNOWLEDGMENTS

This work has taken place in the Autonomous Mobile Robotics Laboratory (AMRL) at UT Austin. AMRL research is supported in part by NSF (CAREER-2046955, IIS-1954778, SHF-2006404), ARO (W911NF-19-2-0333,W911NF-21-20217), DARPA (HR001120C0031), Amazon, JP Morgan, and Northrop Grumman Mission Systems. The views and conclusions contained in this document are those of the authors alone.

REFERENCES

[1] Ross Allen and Marco Pavone. 2016. A real-time framework for kinodynamic planning with application to quadrotor obstacle avoidance. In *AIAA Guidance, Navigation, and Control Conference*. 1374.

[2] David Balaban, Alexander Fischer, and Joydeep Biswas. 2018. A Real-Time Solver For Time-Optimal Control Of Omnidirectional Robots with Bounded Acceleration. 8027–8032. https://doi.org/10.1109/IROS.2018.8594306

[3] Victor M Becerra. 2010. Solving complex optimal control problems at no cost with PSOPT. In *2010 IEEE International Symposium on Computer-Aided Control System Design*. IEEE, 1391–1396.

[4] Riccardo Bonalli, Abhishek Cauligi, Andrew Bylard, and Marco Pavone. 2019. GuSTO: Guaranteed sequential trajectory optimization via sequential convex programming. In *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 6741–6747.

[5] Leonid Butyrev, Thorsten Edelhäußer, and Christopher Mutschler. 2019. Deep reinforcement learning for motion planning of mobile robots. *arXiv preprint arXiv:1912.09260* (2019).

[6] Hao-Tien Lewis Chiang, Jasmine Hsu, Marek Fiser, Lydia Tapia, and Aleksandra Faust. 2019. RL-RRT: Kinodynamic motion planning via learning reachability estimators from RL policies. *IEEE Robotics and Automation Letters* 4, 4 (2019), 4298–4305.

[7] Benjamin Eysenbach, Ruslan Salakhutdinov, and Sergey Levine. 2019. Search on the replay buffer: Bridging planning and reinforcement learning. *arXiv preprint arXiv:1906.05253* (2019).

[8] Aleksandra Faust, Kenneth Oslund, Oscar Ramirez, Anthony Francis, Lydia Tapia, Marek Fiser, and James Davidson. 2018. PRM-RL: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 5113–5120.

[9] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. 2014. Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2997–3004.

[10] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. 2015. Batch informed trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. In *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE, 3067–3074.

[11] Pradipto Ghosh and Bruce Conway. 2012. Near-optimal feedback strategies for optimal control and pursuit-evasion games: a spatial statistical approach. In *AIAA/AAS astrodynamics specialist conference*. 4590.

[12] Kris Hauser and Yilun Zhou. 2016. Asymptotically optimal planning by feasible kinodynamic planning in a state–cost space. *IEEE Transactions on Robotics* 32, 6 (2016), 1431–1443.

[13] Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal. 2011. STOMP: Stochastic trajectory optimization for motion planning. In *2011 IEEE international conference on robotics and automation*. IEEE, 4569–4574.

[14] Sertac Karaman and Emilio Frazzoli. 2011. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research* 30, 7 (2011), 846–894.

[15] Michal Kleinbort, Kiril Solovey, Zakary Littlefield, Kostas E Bekris, and Dan Halperin. 2018. Probabilistic completeness of RRT for geometric and kinodynamic planning with forward propagation. *IEEE Robotics and Automation Letters* 4, 2 (2018), x–xvi.

[16] George P Kontoudis and Kyriakos G Vamvoudakis. 2019. Kinodynamic motion planning with continuous-time Q-learning: An online, model-free, and safe navigation framework. *IEEE transactions on neural networks and learning systems* 30, 12 (2019), 3803–3817.

[17] Steven M LaValle and James J Kuffner Jr. 2001. Randomized kinodynamic planning. *The international journal of robotics research* 20, 5 (2001), 378–400.

[18] Linjun Li, Yinglong Miao, Ahmed H Qureshi, and Michael C Yip. 2021. MPC-MPNet: Model-Predictive Motion Planning Networks for Fast, Near-Optimal Planning under Kinodynamic Constraints. *IEEE Robotics and Automation Letters* 6, 3 (2021), 4496–4503.

[19] Yanbo Li, Zakary Littlefield, and Kostas E Bekris. 2015. Sparse methods for efficient asymptotically optimal kinodynamic planning. In *Algorithmic foundations of robotics XI*. Springer, 263–282.

[20] Zakary Littlefield and Kostas E Bekris. 2018. Efficient and asymptotically optimal kinodynamic motion planning via dominance-informed regions. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 1–9.

[21] Andrew Y Ng, Daishi Harada, and Stuart Russell. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, Vol. 99. 278–287.

[22] Daniel Perille, Abigail Truong, Xuesu Xiao, and Peter Stone. 2020. Benchmarking metric ground navigation. In *2020 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. IEEE, 116–121.

[23] Mihail Pivtoraiko and Alonzo Kelly. 2011. Kinodynamic motion planning with state lattice motion primitives. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2172–2179.

[24] Nathan Ratliff, Matt Zucker, J Andrew Bagnell, and Siddhartha Srinivasa. 2009. CHOMP: Gradient optimization techniques for efficient motion planning. In *2009 IEEE International Conference on Robotics and Automation*. IEEE, 489–494.

[25] Carlos Sánchez-Sánchez and Dario Izzo. 2018. Real-time optimal control via deep neural networks: study on landing problems. *Journal of Guidance, Control, and Dynamics* 41, 5 (2018), 1122–1135.

[26] Dharmesh Tailor and Dario Izzo. 2019. Learning the optimal state-feedback via supervised imitation learning. *Astrodynamics* 3, 4 (2019), 361–374.

[27] Panagiotis Tsiotras and Ricardo Sanz Diaz. 2014. Real-time near-optimal feedback control of aggressive vehicle maneuvers. In *Optimization and optimal control in automotive systems*. Springer, 109–129.

[28] Dustin J Webb and Jur Van Den Berg. 2013. Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics. In *2013 IEEE International Conference on Robotics and Automation*. IEEE, 5054–5061.

[29] Wouter J Wolfslag, Mukunda Bharatheesha, Thomas M Moerland, and Martijn Wisse. 2018. RRT-CoLearn: towards kinodynamic planning without numerical trajectory optimization. *IEEE Robotics and Automation Letters* 3, 3 (2018), 1655–1662.

[30] Christopher Xie, Jur van den Berg, Sachin Patil, and Pieter Abbeel. 2015. Toward asymptotically optimal motion planning for kinodynamic systems using a two-point boundary value problem solver. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 4187–4194.

[31] Mandy Xie and Frank Dellaert. 2020. Batch and incremental kinodynamic motion planning using dynamic factor graphs. *arXiv preprint arXiv:2005.12514* (2020).

[32] Dongliang Zheng and Panagiotis Tsiotras. 2021. Sampling-based Kinodynamic Motion Planning Using a Neural Network Controller. In *AIAA Scitech 2021 Forum*. 1754.

State Supervised Steering Function for Sampling-based Kinodynamic Planning

Pranav Atreya
University of Texas at Austin
Austin, TX, United States
pranavatreya@utexas.edu

Joydeep Biswas
University of Texas at Austin
Austin, TX, United States
joydeepb@cs.utexas.edu

ACM Reference Format:
Pranav Atreya and Joydeep Biswas. 2022. State Supervised Steering Function for Sampling-based Kinodynamic Planning. In *Proc. of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2022), Online, May 9–13, 2022*, IFAAMAS, 2 pages.

1 SUPPLEMENTARY MATERIALS

1.1 Robot State/Control Space Bounds

The following are the bounds of the state and control variables for the Dubin's Car with Acceleration robot domain:

$$x : [-5, 5]m \qquad y : [-5, 5]m$$
$$\theta : [0, 2\pi]rad \qquad v : [-3, 3]\frac{m}{s}$$
$$k : [-1, 1]m^{-1} \qquad a : [-1, 1]\frac{m}{s^2}$$

The following are the bounds of the state and control variables for the Tractor Trailer robot domain:

$$x : [-5, 5]m \qquad y : [-5, 5]m$$
$$\theta : [0, 2\pi]rad \qquad v : [-1, 1]\frac{m}{s}$$
$$\alpha : [0, 2\pi]rad \qquad L : 0.25m$$
$$D : 0.5m \qquad a : [-1, 1]\frac{m}{s^2}$$
$$\phi : [\tan^{-1}(-L), \tan^{-1}(L)]rad$$

The following are the bounds of the state and control variables for the Quadrotor robot domain:

$$x : [-5, 5]m \qquad y : [-5, 5]m$$
$$z : [0, 5]m \qquad \dot{x} : [-3, 3]\frac{m}{s}$$
$$\dot{y} : [-3, 3]\frac{m}{s} \qquad \dot{z} : [-1, 1]\frac{m}{s}$$
$$\theta : [-\frac{\pi}{2}, \frac{\pi}{2}]rad \qquad \phi : [-\frac{\pi}{2}, \frac{\pi}{2}]rad$$
$$\gamma : [-\pi, \pi]rad \qquad \dot{\theta} : [-\pi, \pi]\frac{rad}{s}$$
$$\dot{\phi} : [-\pi, \pi]\frac{rad}{s} \qquad \dot{\gamma} : [-\frac{\pi}{2}, \frac{\pi}{2}]\frac{rad}{s}$$
$$w : 1.2kg \qquad L : 0.3m$$
$$r : 0.1m \qquad b : 0.0245$$
$$\tau_1 : [1.994, 10.095]N \qquad \tau_2 : [1.994, 10.095]N$$
$$\tau_3 : [1.994, 10.095]N \qquad \tau_4 : [1.994, 10.095]N$$

Proc. of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2022), P. Faliszewski, V. Mascardi, C. Pelachaud, M.E. Taylor (eds.), May 9–13, 2022, Online.

1.2 Implementation Details

All of the experiments were run on a Parallels Desktop virtual machine running Ubuntu ARM64 on a 2020 M1 Macbook Air. The virtual machine was equipped with 4 processing cores and 4 GB RAM.

For the planning experiments, the S3F-RRT*, NLP-RRT*, and RRT algorithms were implemented in C++ by the authors. The Open Motion Planning Library (OMPL) [4] was used for the implementation of the SST algorithm. For training dataset generation and in NLP-RRT*, the PSOPT [1] optimal control library was used as the NLP solver.

The policy π in S3F was represented as a feedforward neural network. A two hidden layer 256 neuron network with tanh activations was used for both the Dubin's car with acceleration and tractor trailer problem spaces. A three hidden layer 256 neuron network with the same activations was used for the quadrotor problem space.

1.3 Probabilistic Completeness Proof

Here we present a proof of probabilistic completeness (PC) of the S3F-RRT* algorithm. S3F-RRT* is a modification of the original RRT* algorithm [2] designed to make use of a learned steering function. The proof largely follows the structure of the proof of probabilistic completeness of geometric RRT [3], though significant modifications have been made to take into account the presence of kinodynamic constraints and the use of a learned steering function.

Let $c^*(x_a, x_b)$ denote the cost of the optimal trajectory from x_a to x_b, or equivalently the kinodynamic distance from x_a to x_b. We assume that c^* obeys the triangle inequality, that is, $c^*(x_a, x_b) \leq c^*(x_a, x) + c^*(x, x_b)$ for all $x \in X$. Let $\tilde{S}$ be a learned steering function. We assume that with nonzero probability p, $\tilde{S}(x_a, x_b)$ yields a state function $\tilde{\Gamma}$ that satisfies $c^*(\tilde{\Gamma}(t), x_b) \leq c^*(x_a, x_b)$ for all $t \in [0, t_f]$. This assumption in essence states that every state along the path produced by $\tilde{S}$ is kinodynamically closer to the goal state than the start state is. For a steering function trained to be optimal, this is a reasonable assumption.

We will use $B_r(x)$ to denote the subset of the state space X defined by $\{x' | c^*(x', x) \leq r\}$. For simplicity, we assume that there exist $\delta_{\text{goal}} > 0, x_{\text{goal}} \in X_{\text{goal}}$ such that $B_{\delta_{\text{goal}}}(x_{\text{goal}}) \subseteq X_{\text{goal}}$. We denote this simplified goal region $B_{\delta_{\text{goal}}}(x_{\text{goal}})$ as X^*_{goal}. The goal of the motion planning problem is to find a kinodynamically feasible path $\pi : [0, t_\pi] \to X_{\text{free}}$ such that $\pi(0) = x_{\text{init}}$ and $\pi(t_\pi) \in X^*_{\text{goal}}$. The clearance of π is the maximal δ_{clear} such that $B_{\delta_{\text{clear}}}(\pi(t)) \in X_{\text{free}}$ for all $t \in [0, t_\pi]$.

We assume for this proof that there exists a valid trajectory $\pi : [0, t_\pi] \to X_{\text{free}}$ with clearance $\delta_{\text{clear}} > 0$. Without loss of

generality, assume that $\pi(t_\pi) = x_{\text{goal}}$, i.e., the trajectory terminates at the center of the goal region. Let L be the total cost of π, and let $v = min(\delta_{\text{clear}}, \delta_{\text{goal}})$. Let $m = \frac{3L}{v}$. Define a sequence of $m+1$ points $x_0 = x_{\text{init}}, ..., x_m = x_{\text{goal}}$ along π such that the cost of traversal from one point to the next is $\frac{v}{3}$. Therefore, $c^*(x_i, x_{i+1}) \leq \frac{v}{3}$ for every $0 \leq i < m$. We will now prove that as the number of iterations increases, the S3F-RRT* algorithm will generate a path passing through the vicinity of these $m+1$ points with probability asymptotically approaching one.

Lemma 1.1. *Suppose that S3F-RRT* has reached $B_{\frac{v}{3}}(x_i)$, that is, its tree contains a vertex x_i' such that $x_i' \in B_{\frac{v}{3}}(x_i)$. If $x_{\text{rand}} \in B_{\frac{v}{3}}(x_{i+1})$ and $c^*(x_i, x_{\text{rand}}) \leq \frac{v}{3}$ (equivalently $x_i \in B_{\frac{v}{3}}(x_{\text{rand}})$), then the path from the nearest neighbor x_{near} to x_{rand} lies entirely in X_{free} with probability p.*

Proof. Because x_{near} is the nearest neighbor, it is true that $c^*(x_{\text{near}}, x_{\text{rand}}) \leq c^*(x_i', x_{\text{rand}})$. Invoking the triangle inequality,

$$\begin{aligned} c^*(x_{\text{near}}, x_{i+1}) &\leq c^*(x_{\text{near}}, x_{\text{rand}}) + c^*(x_{\text{rand}}, x_{i+1}) \\ &\leq c^*(x_i', x_{\text{rand}}) + c^*(x_{\text{rand}}, x_{i+1}) \\ &\leq c^*(x_i', x_i) + c^*(x_i, x_{\text{rand}}) + c^*(x_{\text{rand}}, x_{i+1}) \\ &\leq 3\frac{v}{3} = v \end{aligned}$$

Thus $x_{\text{near}} \in B_v(x_{i+1})$, meaning $x_{\text{near}} \in X_{\text{free}}$. Assume that $c^*(\tilde{\Gamma}(t), x_b) \leq c^*(x_a, x_b)$. The probability that this occurs is p. Since each state along $\tilde{\Gamma}$ is closer or as close to x_{rand} as x_{near}, the same logic that was applied above to x_{near} can be applied to each respective state. Thus, with probability p, the path from x_{near} to x_{rand} will lie entirely in x_{free}. □

Theorem 1.2. *The probability that S3F-RRT* fails to reach X^*_{goal} from x_{init} after k iterations is at most ae^{-bk}, for some constants $a, b \in \mathbb{R}_{>0}$.*

Proof. Assume that $B_{\frac{v}{3}}(x_i)$ already contains an S3F-RRT* vertex. Let r_i be the probability that in the next iteration a S3F-RRT* vertex will be added to $B_{\frac{v}{3}}(x_{i+1})$. Recall that due to lemma 1.1, $x_{\text{rand}} \in B_{\frac{v}{3}}(x_{i+1})$ and $c^*(x_i, x_{\text{rand}}) \leq \frac{v}{3}$ implies that the path from x_{near} to x_{rand} will lie entirely in X_{free} with probability p. In the S3F-RRT* algorithm, after x_{rand} is sampled, all states in X_{near} are considered as possible parent states. By the definition of X_{near}, x_{near} is a part of this candidate set. Thus, it is guaranteed that x_{new} will be added as a S3F-RRT* vertex with probability greater than or equal to p. Assume that the probability that both $x_{\text{rand}} \in B_{\frac{v}{3}}(x_{i+1})$ and $c^*(x_i, x_{\text{rand}}) \leq \frac{v}{3}$ is $\gamma_i > 0$. It is safe to assume that this probability is nonzero because any state along the path produced by $S^*(x_i, x_{i+1})$ satisfies these constraints, and so does any state along the portion of π from x_i to x_{i+1}. Finally, let the conditional probability that $x_{\text{new}} \in B_{\frac{v}{3}}(x_{i+1})$ given that $x_{\text{rand}} \in B_{\frac{v}{3}}(x_{i+1})$ and $c^*(x_i, x_{\text{rand}}) \leq \frac{v}{3}$ be $\kappa_i > 0$. It is again safe to assume that this probability is nonzero because $\tilde{\Gamma}$ closely approximates Γ^*, meaning x_{new} will be close to x_{rand}. Taking into account these probabilities, we have $r_i = p\gamma_i\kappa_i$. Note that this expression is independent of k.

Let r be the minimum of the probabilities $\{r_i | \forall i (0 \leq i < m)\}$. In order for the S3F-RRT* algorithm to reach X^*_{goal} from x_{init}, a S3F-RRT* vertex must be added to $B_{\frac{v}{3}}(x_{i+1})$ m times for $0 \leq i < m$. This stochastic process can be defined as a Markov chain. Alternatively, this process can be described as k Bernoulli trials with success probability r. The planning problem can be solved after m successful outcomes. Note that the success probability r is an underestimate of the true success probability for each trial, and that it is possible that the process ends after less than m successful outcomes. Defining the problem in such a manner allows us to obtain an upper bound on the probability of failure.

Next, we bound the probabilty of faiure, that is, the probability that the process does not reach state m after k steps. Let X_k denote the number of successes in k trials, then

$$\begin{aligned} Pr[X_k < m] &= \sum_{i=0}^{m-1} \binom{k}{i} r^i (1-r)^{k-i} \\ &\leq \sum_{i=0}^{m-1} \binom{k}{m-1} r^i (1-r)^{k-i} \\ &\leq \binom{k}{m-1} \sum_{i=0}^{m-1} (1-r)^k \\ &\leq \binom{k}{m-1} \sum_{i=0}^{m-1} (e^{-\tau})^k \\ &= \binom{k}{m-1} m e^{-rk} \\ &= \frac{\prod_{i=k-m}^{k} i}{(k-1)!} m e^{-rk} \\ &\leq \frac{m}{(m-1)!} k^m e^{-rk} \end{aligned}$$

where the second statement is justified since $m << k$, the third statement uses the fact that $r < \frac{1}{2}$, and the fourth statement relies on $(1-r) \leq e^{-\tau}$. As r, m are fixed and independent of k, the expression $\frac{1}{(m-1)!} k^m m e^{-rk}$ decays to zero exponentially with k. Therefore, S3F-RRT* is probabilistically complete. □

REFERENCES

[1] Victor M Becerra. 2010. Solving complex optimal control problems at no cost with PSOPT. In *2010 IEEE International Symposium on Computer-Aided Control System Design*. IEEE, 1391–1396.

[2] Sertac Karaman and Emilio Frazzoli. 2011. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research* 30, 7 (2011), 846–894.

[3] Michal Kleinbort, Kiril Solovey, Zakary Littlefield, Kostas E Bekris, and Dan Halperin. 2018. Probabilistic completeness of RRT for geometric and kinodynamic planning with forward propagation. *IEEE Robotics and Automation Letters* 4, 2 (2018), x–xvi.

[4] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. 2012. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine* 19, 4 (December 2012), 72–82. https://doi.org/10.1109/MRA.2012.2205651 https://ompl.kavrakilab.org.