JOURNAL OF AEROSPACE INFORMATION SYSTEMS
Vol. 19, No. 7, July 2022

Utilizing Reinforcement Learning to Continuously Improve a Primitive-Based Motion Planner

Zachary C. Goddard* and Kenneth Wardlaw†
Georgia Institute of Technology, Atlanta, Georgia 30332
Kyle Williams‡ and Julie Parish§
Sandia National Laboratories, Albuquerque, New Mexico 87123
and
Anirban Mazumdar¶
Georgia Institute of Technology, Atlanta, Georgia 30332

https://doi.org/10.2514/1.I011044

This paper describes how the performance of motion primitive-based planning algorithms can be improved using reinforcement learning. Specifically, we describe and evaluate a framework that autonomously improves the performance of a primitive-based motion planner. The improvement process consists of three phases: exploration, extraction, and reward updates. This process can be iterated continuously to provide successive improvement. The exploration step generates new trajectories, and the extraction step identifies new primitives from these trajectories. These primitives are then used to update rewards for continued exploration. This framework required novel shaping rewards, development of a primitive extraction algorithm, and modification of the Hybrid A* algorithm. The framework is tested on a navigation task using a nonlinear F-16 model. The framework autonomously added 91 motion primitives to the primitive library and reduced average path cost by 21.6 s, or 35.75% of the original cost. The learned primitives are applied to an obstacle field navigation task, which was not used in training, and reduced path cost by 16.3 s, or 24.1%. Additionally, two heuristics for the modified Hybrid A* algorithm are designed to improve effective branching factor.

Nomenclature

A	=	bearing to desired position
b^*	=	effective branching factor
C_q	=	maneuver start set of trim q
D_p	=	maneuver end set
d	=	search depth
d_1	=	distance to goal
d_2	=	distance to trim state
g	=	vehicle transform
$\mathcal{H}$	=	symmetry group
h_{lat}	=	lateral time to go heuristic
h_z	=	altitude time to go heuristic
h_ψ	=	heading time to go heuristic
J	=	cost to go
k_{dist}	=	distance reward coefficient
k_{obs}	=	obstacle penalty coefficient
k_{trim}	=	distance to trim state reward coefficient
N	=	nodes expanded
O	=	obstacle set
(p_b, q_b, r_b)	=	body frame angular velocity
Q_M	=	set of maneuvers
Q_T	=	set of trim primitives
R_{dist}	=	distance reward
R_{obs}	=	obstacle penalty
R_{trim}	=	distance to trim state reward
(u_b, v_b, w_b)	=	body frame linear velocity
$\mathcal{X}$	=	state space
$\boldsymbol{X}$	=	vehicle state
(x, y, z)	=	position in rectangular coordinates
Γ_p	=	cost of maneuver p
γ_q	=	incremental cost for trim state q
$\hat{\xi}_q$	=	twist of a trim q
τ	=	coasting time
(ϕ, θ, ψ)	=	attitude

Presented as Paper 2021-1752 at the AIAA SciTech Forum 2021, Virtual Event, January 11–21, 2021; received 2 August 2021; revision received 21 January 2022; accepted for publication 22 January 2022; published online 7 March 2022.

*Graduate Student, Woodruff School of Mechanical Engineering. Student Member AIAA.
†Undergraduate Student, Woodruff School of Mechanical Engineering.
‡Pathfinder Technologies.
§Pathfinder Technologies. Associate Fellow AIAA.
¶Assistant Professor, Woodruff School of Mechanical Engineering. Member AIAA.

I. Introduction

KINODYNAMIC planning for mobile systems requires solving complicated constraints while satisfying the system's dynamics. This is a challenging problem, especially when the system dynamics are complex, multiple degrees of freedom (DOF), and nonlinear. Kinodynamic planning can be simplified through the use of motion primitives. Motion primitives can be concatenated to quickly form dynamically feasible trajectories using the concept of the maneuver automaton (MA) [1,2]. This enables rapid kinodynamic motion planning using discrete planners [2–7].

The performance of primitive-based planners is fundamentally limited by the contents of the primitive library. Too few primitives can cause inefficient motion plans, while too many primitives can increase computational complexity. Primitives can be created from raw data generated by pilots [8], through the formulation of an optimal control problem [2], or by forward simulation of the system dynamics with a known control law [9]. However, each of these methods requires extensive human knowledge to pilot the system, formulate an optimal control problem, or develop a stable and effective controller. Currently there is no formalized method for autonomously generating an effective motion primitive library for a given dynamic system.

Reinforcement learning offers exciting new potential in this area. Reinforcement learning can be used to generate motion primitives without expert knowledge. Current machine learning techniques can

save time for human experts and remove potential biases. Reinforcement learning has shown the capability to learn original policies that match or exceed human capabilities when playing complex games [10–12].

In this work we present a framework for autonomously generating a library of motion primitives using reinforcement learning. This library can be generated with minimal human involvement and provides a way to combine the creativity of reinforcement learning with the rigorous performance and broad capability of the MA. The framework starts with an initial primitive library (Fig. 1a), the framework then intelligently generates and adds new primitives (Fig. 1b), and these primitives are used to provide improved planning performance (Fig. 1c).

These methods restrict human input to designing general objectives for a task rather than directly designing specific motion primitives. The overall approach has several key benefits. First, the process can run completely autonomously without human intervention or influence. Second, primitive-based planners can plan/replan very rapidly once the primitive library is generated. Third, the primitive library can be used in a range of environments that were not part of the problem formulation. For example, primitives generated through training on open environments can still provide substantial benefits in obstacle-rich or adversarial situations.

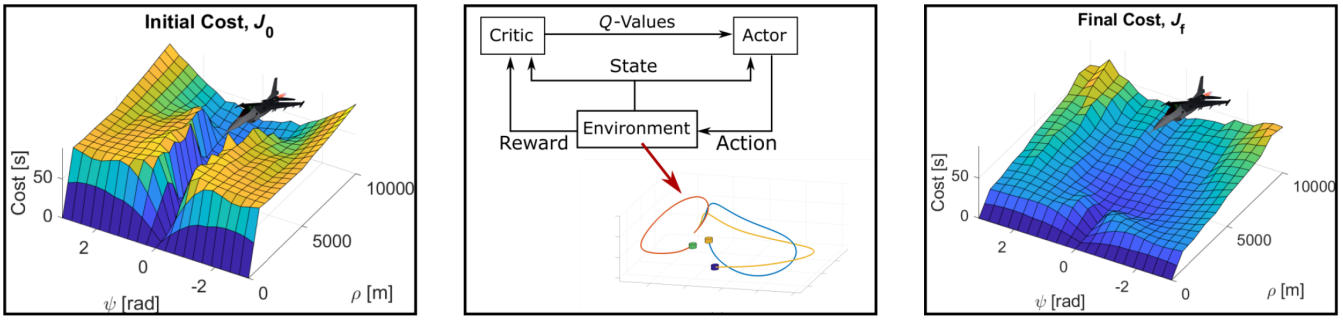
The core technical contributions of this work are twofold. First, there is development and quantitative evaluation of a new closed-loop framework that uses reinforcement learning to generate motion primitives. Second, there is creation of a modified Hybrid A* algorithm for motion planning with the MA. The framework is divided into four key steps: 1) learning end-to-end trajectories, 2) primitive extraction from entire trajectories, 3) pruning redundant primitives, and 4) passing back shaping rewards based on the new library for further improvement. The full framework is illustrated in Fig. 2. We have made unique technical contributions to each of the four key steps: 1) novel reward shaping aligned with the MA formulation, 2) a new primitive extraction algorithm, 3) use of value iteration for rapid library pruning, and 4) leveraging feedback to minimize library size. In addition, we have created an extended version of Hybrid A* built around the MA as well as a set of heuristic functions to improve motion planning performance. This modified algorithm adds the trim state to the search nodes and extends planning to 3D position. Each search step considers a maneuver as well as a set of discrete coasting times for the current trim state. The proposed heuristics replace the Reeds–Shepp heuristic used in the original Hybrid A* paper [13] with a set of precomputed cost functions describing the cost-to-go of less constrained subproblems using the same set of motion primitives.

The paper begins with a review of primitive-based path planning and motivates the use of using reinforcement learning over manually designing a library of motion primitives. Next, the paper describes the requirements for the presented framework. The paper then describes each step of the framework in detail. The value of the framework is illustrated using a range of simulated environments using a nonlinear 6-degree-of-freedom medium-fidelity F-16 flight model [14].

II. Prior Literature

This work builds on the MA, which is a motion primitive architecture that categorizes primitives into a set of trim states, Q_T, and a set of maneuvers, Q_M [1]. Trim states are equilibrium points in the system's dynamics and can be maintained for an undefined coasting time. This allows for flexibility in motion plans. Maneuvers are only constrained at their endpoints that must be trim states, but may otherwise encode any feasible trajectory. These motion primitives are also invariant to a symmetry group, $\mathcal{H}$, which does not affect the system's dynamics. Common symmetry states for aerial vehicles are lateral position, altitude (assuming constant atmosphere), and heading.

A range of methods can be used to create motion plans by leveraging the MA. This involves selecting a string of primitives and coasting times for the trim states. One method precomputes an interpolated cost table via value iteration, then chooses maneuvers and coasting times through a greedy policy [1]. Precomputing the cost table can allow for fast planning; however, the table may require significant



a) b) c)

Fig. 1 Overview of the proposed framework: a) the initial primitive library satisfies reachability and establishes baseline cost for the planner, b) reinforcement learning generates new policies from which primitives are extracted, and c) new primitives improve the performance of the planner.

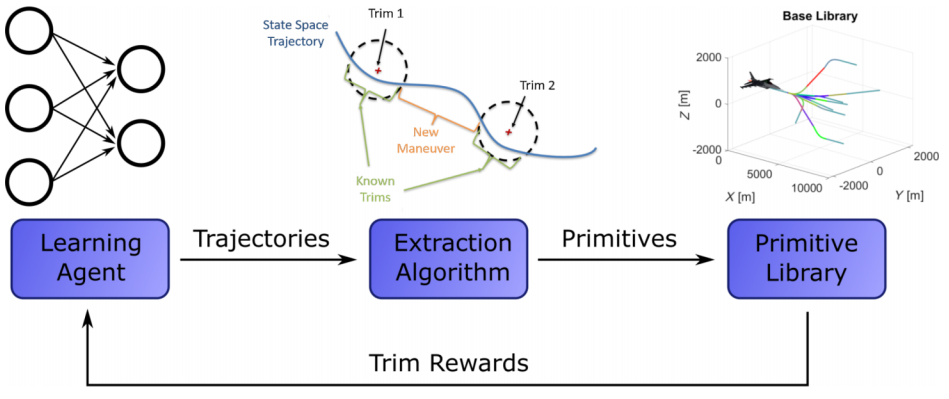


Fig. 2 Diagram of full framework process.

storage space. An alternative method solves the planning problem without a cost table by explicitly solving a combinatorial search and nonlinear program [2]. These obstacle-free planning algorithms can be combined with random trees to handle kinematic constraints, such as obstacles [15,16].

The MA has also been used in combination with an A*-based search algorithm [17]. One particularly relevant search algorithm is Hybrid A* [13]. Hybrid A* uses a set of curves to expand nodes along feasible trajectories. The algorithm has been demonstrated on autonomous cars navigating around obstacles and used Reeds–Shepp paths to calculate obstacle-free heuristic values [18]. This provides several advantages over Theta*, which is another common search algorithm in aerospace applications [19]. Theta* may produce paths with 90° angles, which limits the algorithm to long-distance planning where the cell size is significantly smaller than the turn radius of the vehicle [20]. The use of curves in Hybrid A* allows it to find feasible paths regardless of cell size and track changing velocity along the path. A few modifications to this approach can allow the algorithm to work with the MA. This work presents a modified Hybrid A* for this purpose as well as new heuristics to replace the Reeds–Shepp paths that would not be suitable for fixed-wing aerial vehicles.

The selection of motion primitives, especially maneuvers, heavily influences the effectiveness of the MA. A single well-designed maneuver can significantly improve the optimal path achieved by the MA [2]. The MA has been used for aggressive navigation of dense obstacles using carefully designed motion primitives for maneuvering in tight spaces [21]. These past works show that well-designed motion primitives can be powerful; however, they often require expert knowledge of useful behaviors for the given task. Relying on human intuition can be time-consuming and can introduce human biases into the planner. Therefore, it is desirable to automate the process of generating motion primitives.

The field of reinforcement learning offers new potential for creating effective behavior without human guidance. In particular, Deep Deterministic Policy Gradient (DDPG) and its variants, Twin Delayed DDPG (TD3) and Soft Actor Critic (SAC), can be used on systems with continuous state and action spaces [22–24]. These algorithms use neural networks to learn continuous policies, meaning that they can learn a navigation controller for mobile systems. DDPG in particular has been integrated into a framework for learning to navigate a ground robot and quadrotor [25]. SAC has also been applied to learn a control policy for a legged robot [26].

However, a common problem with neural-network-based algorithms is a lack of transferability to new problems. Extracting motion primitives from the learned policies allows some of that experience to be converted to a general form for use in multiple cases. Additionally the learned motion primitives could be mixed with manually designed primitives to supplement, rather than replace, existing knowledge.

Prior work has learned motion primitives through reinforcement learning for specific cases. One study showed improvement to a primitive-based planner for a quadrotor using Bézier curves as motion primitives [4,27]. Another work learns primitives for a hexapod as a set of neural network policies [28]. In both of these studies, motion primitives are learned individually outside the scope of the full task environment. Our work builds on these ideas by learning policies in the full environment and extracting motion primitives from these policies. Our methods also incorporate the MA directly into the learning framework. This allows our framework be applied to a wide class of vehicles.

III. Primitive Generation Framework Requirements

In this section we outline a set of requirements for an autonomous primitive generation framework. The learning process should be able to improve a motion primitive library with respect to a given planner without intermediate human input. The output of this process should be stored transparently, meaning that it can be easily interpreted by humans and transferred to new planners or applied to new tasks. The framework should also be modular to the type of vehicle, cost function, environment, planning algorithm, and learning agent to allow for a wide range of applications. These desired requirements are outlined below:

1) *Full autonomy:* The proposed framework should be able to discover motion primitives to improve the given planner autonomously. Motion primitives should be learned through independent exploration of the task and environment, so no human input is required in the design of each primitive. Autonomy alleviates the human resource cost of manually designing motion primitives. Additionally, learning the task from nothing frees the framework of biases in existing human knowledge.

2) *Transparency of learned experience:* The experience gained through the learning process should be easily interpreted by humans. Transparency helps apply learned experience to new tasks and environments and allows human experts to understand and learn from the new motion primitives.

3) *Additive learning:* Learned experience should be purely additive without the possibility for continued learning to reduce performance. The policy should be able to take advantage of all past experience such that the actions currently available will always be as good as or better than previously available actions.

4) *Vehicle and environment modularity:* The framework should be applicable to a variety of system dynamics to allow for application on multiple vehicles or environments. This modularity should be built-in such that application to new dynamics does not require fundamental changes to the learning architecture.

5) *Interchangeable cost function:* The objective function optimized by the learned primitives should be interchangeable. Different metrics may need to be prioritized in various scenarios. The framework should easily accept new cost functions to learn different sets of motion primitives for these scenarios.

To our knowledge, current literature does not provide an architecture for learning motion primitives that meets the above criteria. We therefore propose a new framework to achieve these requirements by combining reinforcement learning with the MA and a primitive-based planner. The proposed framework is outlined in Fig. 2.

This framework improves the planner on the given environment and reward/cost function autonomously via reinforcement learning. Experience gained during exploration is extracted as motion primitives, which may be easily visualized and interpreted as reference trajectories in the state and control spaces. These motion primitives can also be manipulated afterwards or applied to a new task. Since new primitives can be added without removing previous ones, the quality of paths generated by the learned library only improves on the initial library. The motion primitives learned are defined by the MA, which may describe primitives for a wide range of mobile systems, such as fixed-wing aircraft, rotorcraft, watercraft, and cars [1]. These primitives are also fed back as shaping rewards to help the discovery of new motion primitives in later iterations.

IV. Framework Overview

This section outlines the learning framework and describes integration of the MA with primitive-based planners and a reinforcement learning algorithm. The proposed architecture consists of three components: a primitive-based planner, a learning agent, and a motion primitive extraction algorithm. Trim states that are extracted from the learned policy are also fed back through shaping rewards to help the learning agent in subsequent iterations of the process.

A. Motion Planners

The first component of the proposed architecture is a planner that can solve the given task using motion primitives from an MA. This section covers an existing motion planner for obstacle-free navigation using value iteration. Additionally we propose modifications to the Hybrid A* search algorithm to integrate it with an MA for navigation around obstacles.

1. Value Iteration

The first proposed planner uses value iteration to build a table of costs with interpolation as described by [1]. This method computes

cost of a hybrid state, $(q, h) \in Q_T \times \mathcal{H}$, iteratively as

$$J_{i+1}(q,h) = \min_{p,\tau} \gamma_q \tau + \Gamma_p + J_i(\Phi_{p,\tau}(q,h)) \tag{1}$$

In the above equation, τ is the coasting time, and γ_q and Γ_p are the cost of the trim state and maneuver, respectively. The function $\Phi_{p,\tau}: Q_T \times \mathcal{H} \to Q_T \times \mathcal{H}$ transforms the hybrid state to a new state based on a maneuver and coasting time. Since the symmetry group is continuous, the cost function must be approximated by calculating it at a discrete set of points in $\mathcal{H}$ and interpolating between them. This cost function will converge to the optimal cost function J^*, which can be used for planning with a greedy policy. The optimal action is recovered by finding the maneuver and coasting time that achieve the optimal cost at a given state

$$p^*, \tau^* = \arg\min_{p,\tau} \gamma_q \tau + \Gamma_p + J^*(\Phi_{p,\tau}(q,h)) \tag{2}$$

The main downside to value iteration is the need to store the cost table for planning. This table can be large and its size increases exponentially with the number of discrete points used for the symmetry group. This method is manageable for low-dimensional problems, such as navigating in a 2D plane. In this case the size of the table would be $|Q_T|N^2$, where N is the number of points used to discretize each dimension of $\mathcal{H}$. Navigating to a goal in 3D space with a desired final heading would add two dimensions to the symmetry group, increasing the size to $|Q_T|N^4$.

Additionally the value iteration method is intended for obstacle-free navigation. A rapidly-exploring random tree (RRT) algorithm can use the value iteration to expand new nodes [16]; however, this requires solving many paths before reaching the goal. Due to these disadvantages, we propose the use of Hybrid A* as an alternative that does not require a precomputed cost function and can handle navigation around obstacles on its own. However, value iteration can still be useful for quickly evaluating the performance of maneuvers. We describe this feature in Sec. IV.C.

2. *Hybrid A**

Hybrid A* is a variant of the A* algorithm designed for creating feasible vehicle trajectories. In the original Hybrid A* paper [13], the algorithm used a range of curves representing feasible paths for an autonomous car to expand new nodes in the search [13]. The original algorithm searches over discrete nodes containing x and y position and heading, (x, y, ψ). These nodes are expanded by searching a set of feasible curves to find subsequent nodes that are reachable from the current state.

In this work, we present a version of Hybrid A* that is modified to be better suited for MA-based aircraft planning. The modified algorithm extends Hybrid A* to 3D position and includes the current trim state in the node. The format of the nodes for this extended Hybrid A* algorithm is (q, x, y, z, ψ). Instead of expanding nodes via kinematically feasible curves, the algorithm expands node using dynamically feasible trim trajectories and maneuvers from the MA as shown in Fig. 3. Since Hybrid A* only expands nodes via fixed length trajectories, trim primitives are limited to a discrete set of coasting times during the search, but can be altered for precision in postprocessing. A single discrete time could be considered; however, considering multiple allows for both good resolution and a potential reduction of the depth of the solution similar to jump points in regular A* [29].

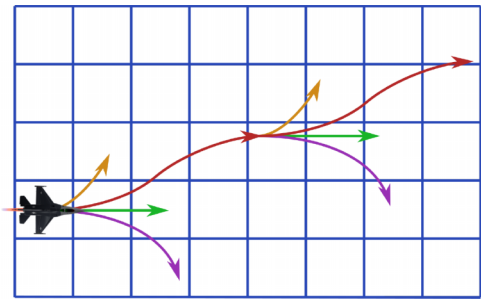


Fig. 3 The modified Hybrid A* uses maneuvers, and trim primitives are used to expand a path to new nodes from the endpoint of the last primitive.

When expanding a node in the search, the algorithm checks the endpoints of each motion primitive that is executable from the current trim state and gets the respective cell. If that cell has not been visited previously in the search, the endpoint is added to a priority queue along with its cost and heuristic value to be expanded at a later step of the algorithm. The cost and heuristic functions are defined by the user. The extended version of the algorithm is described in Algorithm 1. The function Succ(q) denotes successors of a primitive. The successors are either a set of possible maneuvers if $q \in Q_T$ or the successor trim state if $q \in Q_M$.

Hybrid A* attempts to find an optimal sequence of motion primitives. It is not guaranteed to return the optimal concatenation of primitives due to some deviations from the regular A* algorithm. The path does tend to fall within a neighborhood of the optimal, and the original paper uses some postprocessing to locally optimize the path [13]. Our version can easily optimize a path returned by Hybrid A* with a simplified version of the nonlinear program described in [2]:

$$\min \sum_{i}^{N} \gamma_{q_i} \tau_i \quad \text{s.t} \tag{3}$$

$$\left[\prod_{i}^{N-1} \exp(\hat{\xi}_{q_i} \tau_i) g_{p_i}\right] \exp(\hat{\xi}_{q_i} \tau_i) = g_f \tag{4}$$

$$\tau_i \geq 0 \tag{5}$$

Equations (3–5) describe a nonlinear program to minimize the cost associated with sum of coasting times under the constraint that the endpoint must reach the desired goal, as well as nonnegativity constraints on the coasting times. This assumes a fixed string of maneuvers of length N, which was found through Hybrid A*. This minimization can be accomplished via any suitable nonlinear program solver, such as trust region method [30]. Postprocessing the coasting times can improve suboptimal costs or final displacement from the goal due to the discrete nature of the Hybrid A* nodes.

The original Hybrid A* also made use of Reeds–Shepp paths to calculate an obstacle-free heuristic. The work proposes an alternative

Algorithm 1: Pimitive-based Hybrid A*

Inputs: Initial state (q_0, h_0), goal state (q_f, h_f), a discrete set of coasting times T, and a heuristic function
Frontier ← Empty Priority Queue
Closed-Set ← Empty Set
Push (q_0, h_0) to the Frontier with a value of Heuristic(q_0, h_0)
while Frontier $\neq \varnothing$ **do**
  $q, h \leftarrow$ Pop node from Frontier
  if Cell(q, h) = Cell(q_f, h_f) **then**
    Break
  end if
  for $t \in T$ **do**
    $h_{\text{new}} \leftarrow h e^{\hat{\xi}_q t}$
    Push (q, h_{new}) to the Frontier with a value of Heuristic(q, h_{new}) + Cost(q, h_{new})
  end for
  for $p \in$ Succ(q) **do**
    $h_{\text{new}} \leftarrow h + \delta h_p$
    $q_{\text{new}} \leftarrow$ Succ(p)
    Push $(q_{\text{new}}, h_{\text{new}})$ to the Frontier with a value of Heuristic$(q_{\text{new}}, h_{\text{new}})$ + Cost$(q_{\text{new}}, h_{\text{new}})$
  end for
end while

obstacle-free heuristic, which can be generalized to any less constrained goal of the search. A common approach to designing heuristic functions is to use a less-constrained version of the problem being solved [31]. For 3D motion planning, this can be achieved by decoupling the symmetry space, for example, finding a path to a given lateral position regardless of heading and altitude.

The solution to these lower-dimensional problems can be stored in cost tables similar to value iteration. While this requires storage space, the decoupled tables will be much smaller than the solution to the fully coupled problem.

This paper investigates the benefit of using decoupled cost tables as a heuristic function for Hybrid A*. When planning on the same task that the cost table was generated on, Hybrid A* should search mostly along a direct path to the goal similarly to how the value iteration method recovers the optimal policy from the cost table. In cases that are strictly more constrained than the original cost table, the heuristic will still be admissible since it will underestimate the real cost. For example, if obstacles are added to the environment Hybrid A* can still use the cost table to improve search efficiency. This benefit is an important advantage over value iteration that would need to compute a new cost table if new constraints are added.

B. Learning Agent

Our framework uses reinforcement learning to generate new motion primitives. The learning agent generates a new policy for the given environment from scratch using the full continuous action space for the system. Allowing the agent to learn raw inputs for the entire task gives freedom to the possible motion primitives learned rather than limiting the agent to learn specific motion primitives. This represents a fundamental difference from past works in this area [4,28]. Additionally by learning to solve the task instead of explicitly learning motion primitives, the agent finds behaviors that are specifically tailored to its objective. This gives significant advantages to the diversity of motion primitives that can be discovered over methods that learn explicit primitives one at a time. A range of reinforcement learning methods may be used for trajectory generation as long as they are capable of learning continuous state and action spaces. Some good algorithms for this phase are DDPG, TD3, and SAC [22–24]. The "learning agent" could also be a human pilot demonstrating trajectories to be input to the extraction algorithm later.

For this work we use SAC due to its success on similarly complex mobile systems [26]. Actor–critic algorithms train two neural networks simultaneously by exploring and environment as illustrated in Fig. 4. The actor network attempts to learn the optimal policy while the critic network attempts to approximate the value function [32]. SAC is an off-policy actor–critic deep reinforcement learning algorithm that uses maximum entropy reinforcement learning. An illustration of the actor–critic architecture is shown in Fig. 4.

One problem with this approach is that without constraining the algorithm to learn specific motion primitives it is likely to find a policy that that does not use primitives, since motion primitives are not necessarily optimal in the continuous space. We introduce a shaping reward to encourage the agent to stay in or near trim states when possible. Additionally we formulate some general shaping rewards to help with a goal-based navigation task:

$$R_{\text{obs}}(x') = \begin{cases} -k_{\text{obs}}, & \text{if } x' \in O \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

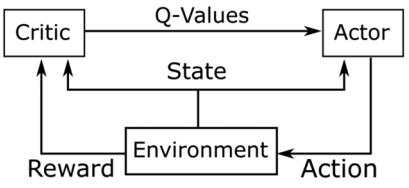


Fig. 4 In the generic structure of an actor–critic model the critic learns Q values from the environment, which the actor uses to improve its policy.

$$R_{\text{dist}}(x, x') = k_{\text{dist}}(d_1(x, x_g) - d(x', x_g)) \tag{7}$$

$$\tilde{R}_{\text{trim}}(x) = \max_q \frac{1}{1 + d_2(x, x_q)}, \quad \forall\, q \in Q_T \tag{8}$$

$$R_{\text{trim}}(x, x') = k_{\text{trim}}(\tilde{R}_{\text{trim}}(x) - \tilde{R}_{\text{trim}}(x')) \tag{9}$$

The values of k_{obs}, k_{dist}, and k_{trim} are constants that may be tuned to weight each reward. The functions $d_i: \mathcal{X} \times \mathcal{X} \to \mathcal{R}$ give the distance between two states. Distance to the goal, d_1, should a metric be relevant to the task such as Euclidean distance. The first shaping reward is a penalty for entering any state that is out of bounds or contains an obstacle [Eq. (6)]. The second shaping reward provides a small value for getting closer the goal and a small penalty for moving away to help with the sparse nature of the main reward for reaching the goal [Eq. (8)]. Both of these rewards are generic to any goal-based objective and do not specifically help with generating motion primitives.

The last shaping reward gives a small reward for moving closer to a known trim state and a penalty for moving away [Eq. (9)]. The maximum cumulative reward is defined by k_{trim}. This allows the reward to be easily scaled relative to the primary reward. This shaping reward's cumulative value should be relatively small, so that actions that reach the goal with a lower cost are favorable. If a trim state will reach the goal with a similar cost to any other feasible trajectory, the agent is encouraged to drive the system to and stay at that trim state.

This behavior will increase the frequency of trim states in the agent's trajectories with relatively small impact on the overall cost. Increased frequency of trim primitives will also increase the frequency of maneuvers, since by the definition of the MA any trajectory connecting two trim states is classified as a maneuver as mentioned in Sec. II.

Shaping rewards may also be added for various cost functions. Ideally, these rewards should be directly related to the cost functions used by the planning algorithm.

During the exploration phase, the reinforcement learning algorithm interacts with the environment and improves a policy based on the provided reward functions. After a given number of episodes or time steps training can be paused for the extraction algorithm. A set of trajectories are generated by forward simulating the learning agent's policy in the environment. While generating these trajectories, training-specific features of the learning algorithm can be disabled. In the case of SAC, for example, only the mean of the stochastic policy is used instead of sampling a control input from the Gaussian distribution. Some examples of these trajectories are shown in Fig. 5. These trajectories are passed to the extraction algorithm to identify new motion primitives.

C. Extraction

The trajectories generated by the learning agent may contain subtrajectories that match the MA's definition of either a trim primi-

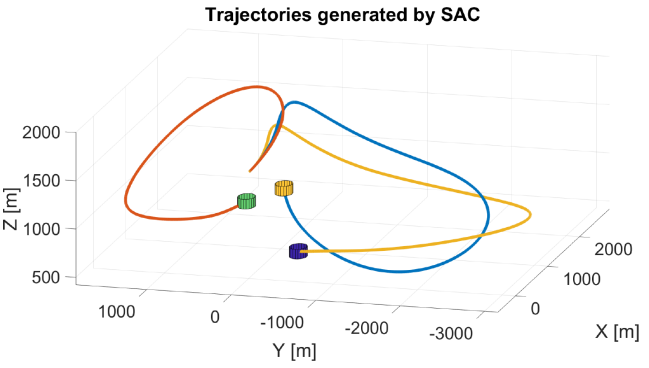


Fig. 5 Trajectories generated by the SAC with cylinders indicating the goals for each episode.

Algorithm 2: Add new trims

Inputs: A state trajectory, $\boldsymbol{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T\}$, a state time derivative trajectory, $\dot{\boldsymbol{X}} = \{\dot{\boldsymbol{x}}_1, \ldots, \dot{\boldsymbol{x}}_T\}$, threshold values, r_{eq}, r_{trim}.
for $0 < i \leq T$ **do**
  if $|\dot{\boldsymbol{x}}_i| < r_{\text{eq}}$ AND $d(\boldsymbol{x}_i, \boldsymbol{x}_q) < r_{\text{trim}} \; \forall \; q \in Q_T$ **then**
    $q_{\text{new}} \leftarrow$NEW − TRIM$(x_i)$
    $Q_T = Q_T \cup q_{\text{new}}$
  end if
end for

tive or a maneuver. Many methods exist for analyzing trajectory data including time series clustering and state clustering [33,34]. Our method directly leverages the definition of the MA. We describe a new algorithm to search raw trajectories for new motion primitives given the current MA's library. This algorithm passes over the data in two steps to add new trim primitives and then new maneuvers.

The first step calculates the time derivative of the state and thresholds it around zero. Points within this threshold may be considered as trim primitives. These points are also compared to other known trim states to avoid dense clusters of redundant trim states. If the distance between the state of a potential trim primitive and a known trim primitive is less than a desired threshold, it is ignored. Otherwise, the point and its corresponding equilibrium control input are added to the MA as a new primitive.

The second step identifies maneuvers by using known trim states to segment the raw trajectory as shown in Fig. 6. Points that belong to the maneuver start set C_q can be considered a hybrid state of the MA [35]. The set C_q is defined as all points from which a maneuver $p \in \text{Succ}(q)$ can be initiated and end in a set D_p, where $D_p \subset C_{\text{Succ}(p)}$ under a given control law. In other words, the start set represents the set of states that the feedback controller can drive toward the end trim of the maneuver. This start set can be calculated numerically based on a controller or a conservative estimate can be used. This step of the algorithm identifies segments of the raw trajectory for which all points are elements of one of these sets. Each point is labeled as the primitive they are closest to

$$\text{label}(x) = \begin{cases} \arg\min_{q \in Q_x} d_2(x, x_q), & \text{if } Q_x \neq \varnothing \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

where

$$Q_x = \{q | q \in Q_T \text{ and } x \in C_q\} \tag{11}$$

The label 0 in Eq. (10) denotes an unlabeled point, or a point that does not belong to any trim primitive's maneuver start set. After labeling, either every point in each segment is labeled as a trim state or every point is unlabeled. The first and last segments are discarded if they are not labeled as trim states, so the remaining segments are

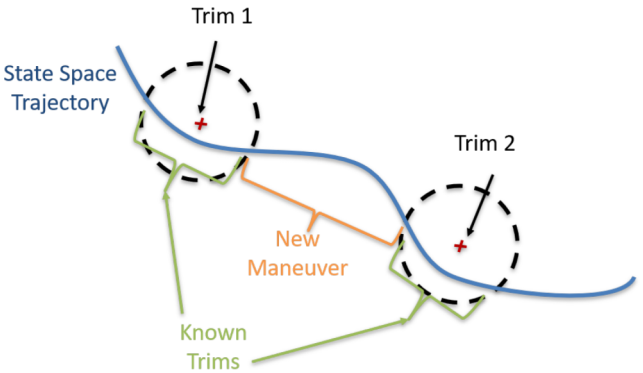


Fig. 6 Segments that start and end within the boundaries around a known trim state are extracted as reference trajectories for a new maneuver.

Algorithm 3: Find maneuvers

Inputs: A state trajectory, $\boldsymbol{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T\}$, where label$(x_1)$ and label(x_T) are not 0, and a control signal, $\boldsymbol{U} = \{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_T\}$
$i \leftarrow 0$
$M \leftarrow \{\}$
while $i \leq T$ **do**
  while label$(x_i) \neq 0$ **and** $i \leq T$ **do**
    $i \leftarrow i + 1$
  end while
  $m \leftarrow$NEW − MANEUVER()
  Pred $(m) \leftarrow$label(x_{i-1})
  $x_m \leftarrow$ Empty array
  $u_m \leftarrow$ Empty array
  while label$(x_i) = 0$ **and** $i \leq T$ **do**
    Append x_i to x_m
    Append u_i to u_m
  end while
  Succ $(m) \leftarrow$label(x_i)
  $M \leftarrow M \cup m$
end while

Algorithm 4: Prune maneuvers

Inputs: A set of candidate maneuvers M and an initial cost map J_0
for $m \in M$ **do**
  $J \leftarrow J_0$
  $q_0 \leftarrow$Succ(m)
  for $h \in H$ **do**
    $J_{\text{new}} \leftarrow \min_\tau \gamma_{q_0} \tau + \Gamma_m + J(\Phi_{\tau,m}(q_0, h))$
    $J(q_0, h) \leftarrow \min(J_{\text{new}}, J(q_0, h))$
  end for
  if $\sum_h J(q_0, h) - J_0(q_0, h) > 0$ **then**
    $Q_M \leftarrow Q_M \cup m$
  end if
end for

either labeled as trim states or are unlabeled and between two trim states. By the MA definition of motion primitives, these unlabeled subtrajectories are maneuvers and can be added to the primitive library. Algorithm 3 efines this process in more detail. The function Pred(m) denotes the predecessor of maneuver m.

During the extraction algorithm, any primitive that fits the criteria of the MA will be identified and added to the library, regardless of the benefit it adds to the planner. Adding large numbers of redundant motion primitives could be detrimental to the run time of the planning algorithm that would need to check a wider range of options at each step that may never be useful. A pruning step can help by eliminating motion primitives that would never be more useful than any currently available motion primitive. Algorithm 4 outlines an optional step that may be used to reduce the number of redundant motion primitives.

The pruning algorithm runs a simplified step of value iteration to check if a motion primitive could improve the cost-to-go from any initial condition. The initial guess for the cost function J_0 can be acquired by running value iteration with the base library of motion primitives, or by using Hybrid A* to calculate the cost-to-go on a sweep of initial positions. The pruning algorithm examines each maneuver m separately via a simplified value iteration equation:

$$J_m(q, h) = \min_\tau \gamma_q \tau + \Gamma_m + J(\Phi_{\tau,m}(q, h)) \tag{12}$$

If $J_m(q, h) < J_0(q, h)$ for any $q \in Q_T$ and $h \in \mathcal{H}$, the maneuver is kept, since it offers some improvement to the planner. Otherwise, the maneuver can be pruned.

Once the extraction algorithm is completed, the exploration rewards can be updated based on new trim primitives and the process can be repeated from the learning stage. This allows new maneuvers

to be added to better connect the new trim states to others in the MA that provides continued improvement.

V. Simulation

This section describes an experimental setup for testing the proposed framework in a simulated environment. The simulation uses JSBSim [36] and a medium-fidelity nonlinear model of an F-16 aircraft [14] for the flight dynamics. We combine these dynamics with custom environments in Python to handle reward functions and additional constraints. The framework is evaluated on an obstacle-free environment in which the agent is given a goal position and must navigate within a threshold of that position. A second environment is also used to demonstrate the transferability of motion primitives to new tasks. The second environment includes a goal position as well a set of obstacles. The framework is not trained on this environment, but we show that the planner's performance improves on this task using only the motion primitives learned in the first environment. This ability to transfer knowledge to a new task is not easily done with learned policies based purely on neural networks [37].

A. Framework Demonstration

The framework starts with an initial library of manually designed motion primitives. These primitives are only meant to cover basic behaviors and satisfy the reachability criteria of the MA. By satisfying reachability, a path is guaranteed to exist for any goal in the obstacle-free case that is useful for evaluating the first iteration of rollouts. This initial library includes nine trim primitives: a level cruise, ascending/descending trim states, and steady turns in both directions. The library also contains 16 maneuvers to connect all the trim primitives to the level cruise trim state. The initial library is illustrated in Fig. 7.

The exploration phase uses SAC for the learning agent to train in the obstacle-free environment. The observation vector for SAC includes the relative position of the goal, velocity, angular rate, and attitude. The velocity is given in the form of Mach number, angle of attack, and side slip angle. The action vector includes normalized body rate commands. The critic networks have two layers with 300 and 200 hidden units, respectively. The actor networks have two shared layers with 300 and 200 hidden units and two layers for the both the mean and standard deviation of the policy with 64 and 32 hidden units. All hidden layers use the ReLU activation function. The actor networks use tanh activation functions at their outputs, and the critic networks use linear activation functions at their outputs.

An SAC model is pretrained without the trim reward from Eq. (9) for 2000 episodes. The first iteration of the exploration phase starts with this model and begins learning with the trim reward included. Each subsequent iteration begins training with the model from the previous iteration. The model trains for 500 episodes each iteration. After training, the model generates 100 trajectories starting from uniformly sampled initial conditions. While generating these trajectories, the model uses the mean of its policy as a deterministic action.

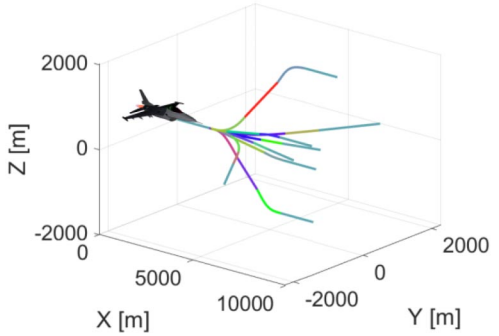


Fig. 7 Example trajectories of the base library showing transitions from the straight and level trim state to all other trim states and back.

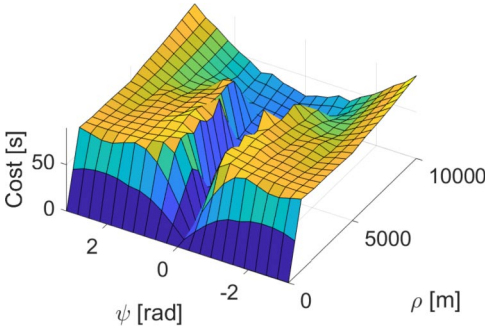


Fig. 8 Map of time-to-go for distance and direction of goal relative to vehicle shown for a single trim state.

The initial primitive library and its subsequent iterations are scored using the cost of the path calculated by Hybrid A* and the optimization step from Eqs. (3–5). The library is evaluated on a sweep of uniformly spaced goal locations to produce a cost table similar to value iteration. The goal positions are only lateral, though altitude is not constrained. Reaching the desired lateral position at any altitude is considered success. The goal positions are placed uniformly at 1 km intervals on a square grid in the xy plane. These scores and their average are reported as performance metrics for each iteration of the primitive library. A slice of the cost function for the base library calculated for the straight and level trim state is illustrated in Fig. 8.

During each iteration of the process, we record the new cost table and the number of motion primitives added to the library. The process stops when the new cost table makes insignificant improvement over the previous table.

B. Testing on Obstacles

We also demonstrate that the learned primitives can provide benefit in new environments and tasks that were not part of the original training. The learned primitives are evaluated on an obstacle field (recall that the learned library was only trained on obstacle-free environments). Performance is evaluated with a similar cost table sweeping over possible lateral positions of the goal. Positions that are infeasible due to being inside an obstacle are ignored. Minimum time cost is used to stay consistent with the objective the framework trained on.

The obstacle field is generated as a set of randomly positioned cylinders with infinite height. Each cylinder has a radius of 2 km.

We expect to see significant improvement in the paths generated to navigate around obstacles despite the original task being obstacle-free. Since the learning agent's experience is encoded into motion primitives, they should be easily applied to any environment that the planning algorithm can solve. Transferring the motion primitives to a new problem is an advantage over many reinforcement learning algorithms. For example, SAC alone would not be able to successfully apply its learned policy for obstacle-free navigation to steer through an obstacle field.

C. Validation

The methods described in this work were validated by attempting to create motion plans in environments with and without obstacles. A navigation feedback controller is used to help track the sequence of motion primitives. The full dynamics are simulated via JSBSim, which is an open source flight dynamics solver. JSBsim takes the body angular rate commands and the throttle command as inputs. The F-16 model features on onboard control augmentation system for tracking the angular rates. The navigation controller is used to track the aircraft position trajectory. The navigation controller is described by the following equations:

$$\boldsymbol{X_e} = \boldsymbol{X_r} - \boldsymbol{X} \tag{13}$$

$$A = \tan^{-1}\left(\frac{y_e}{x_e}\right) - \psi \tag{14}$$

$$B = -[(\psi + A - \psi_r] \tag{15}$$

$$\tilde{\psi}_e = K_A A + K_B B \tag{16}$$

$$T_c = K_T u_e \tag{17}$$

$$\tilde{\theta}_e = \theta_e + (K_z z_e + K_{\dot{z}} \dot{z}_e) \tag{18}$$

$$\tilde{\phi}_e = \phi_e + K_{\text{bank}} \psi_e - K_{\text{level}} \text{sign}(\phi) \theta_e \tag{19}$$

$$p_c = p_e + K_\phi \tilde{\phi}_e \tag{20}$$

$$q_c = q_e + K_\theta (\tilde{\theta}_e \cos(\phi) - \psi_e \sin(\phi)) \tag{21}$$

$$r_c = r_e + K_\psi (\tilde{\theta}_e \sin(\phi) + \psi_e \sin(\phi)) \tag{22}$$

The state vector $\boldsymbol{X}$ is in the format $[x, y, z, u_b, v_b, w_b, p_b, q_b, r_b, \phi, \theta, \psi]$. These states are position coordinates, (x, y, z), linear velocity in the body frame, (u_b, v_b, w_b), angular body rates, (p_b, q_b, r_b), and attitude angles, (ϕ, θ, ψ). Likewise, the reference state vector $\boldsymbol{X_r}$ and state error vector $\boldsymbol{X_e}$ share the same format, and their components are denoted by their respective subscripts (e.g., x_r is the reference x position). The reference state vector is based only on the states associated with the current motion primitive.

The navigation controller determines desired angular rates and desired forward velocity. These are based on the relative position and angle between the aircraft and the trajectory. Deviations in the angular rates and forward velocity are dealt with using the angular rate command and throttle commands that are sent to the F-16 control augmentation system. Error in the roll angle is reduced through the roll rate command and pitch angle is corrected through pitch and yaw rate commands based on the current roll angle. The navigation controller corrects for altitude errors by modifying the desired pitch angle. Position and heading errors are minimized by attempting to drive angles A and B from Fig. 9 to zero.

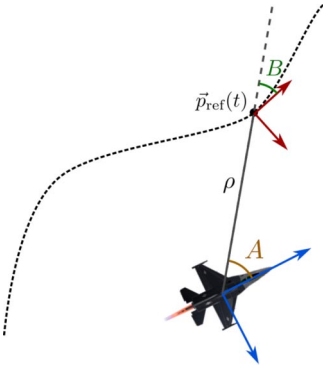


Fig. 9 Illustration of navigation controller angles.

D. Hybrid A* Heuristics

This paper proposes a set of heuristics based on the previously described cost tables, which may improve the performance of our version of Hybrid A*. The cost tables provide a precomputed cost for a less constrained planning problem. This cost can be directly used as an admissible heuristic for the fully constrained problem. The benefit of the proposed cost table heuristics is measured empirically by tracking depth of search and number of nodes expanded. A node is considered "expanded" when its neighbors are added to the frontier as described in Algorithm 1. The relation between the number of nodes expanded, N, search depth d, and effective branching factor b^* is described as follows [38]:

$$N = b^* + (b^*)^2 + \ldots + (b^*)^d \tag{23}$$

Values for N and d are recorded for several plans to different goals and used to calculate b^*. An overall average effective branching factor is also calculated from all the samples. The goals are given as desired positions in 3D coordinates and a desired heading, (x_g, y_g, z_g, ψ_g).

Two cost table heuristics are considered for this experiment. The first heuristic stores the cost to go to the lateral position of the goal only, regardless of altitude and heading. This cost table is the same as the one used in the demonstration of the learning framework. The second heuristic stores the cost to reach a desired altitude and heading. The values of these cost tables are visualized in Fig. 10. The two cost tables are combined into a single heuristic by taking the maximum of the two at a given node, which preserves admissibility.

The cost table heuristics are compared against a set baseline heuristics defined by Eqs. (24–26). These baseline heuristics are simple distance metrics for 3D motion planning.

$$h_{\text{lat}} = \frac{\sqrt{(x_g - x)^2 + (y_g - y)^2}}{V_{\max}} \tag{24}$$

$$h_z = \frac{|z_g - z|}{\dot{z}_{\max}} \tag{25}$$

$$h_\psi = \frac{|\psi_g - \psi|}{\dot{\psi}_{\max}} \tag{26}$$

Equation (24) is a heuristic estimating the time to fly in a straight line to the lateral position of the goal. Equation (25) estimates the time to climb or descend to the altitude of the goal. Equation (26) estimates the time to turn to face the desired heading. The constants $V_{\max}$, $\dot{z}_{\max}$, and $\dot{\psi}_{\max}$ are the maximum lateral speed, climb or descent rate (whichever is faster), and turn rate, respectively. These values are acquired from the set of motion primitives. These heuristics are also combined by choosing the max value of the three equations for a given node.

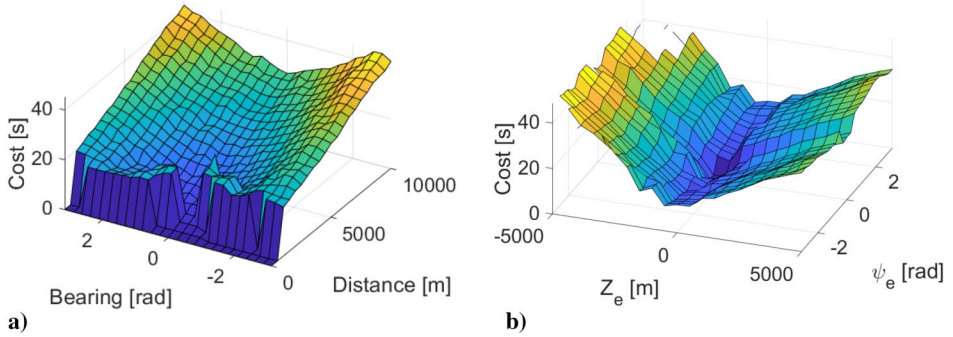


Fig. 10 Cost-map-based heuristics: a) the first heuristic for lateral distance and bearing to the goal, and b) the second heuristic for altitude and heading error.

VI. Data Analysis

The following section discusses the results of each of the above tests. The framework demonstrates autonomous improvement on the obstacle-free navigation task and transfers these improvements to a new environment with a set of obstacles. The cost tables generated for the framework also provide improvement to the efficiency of the Hybrid A* planning algorithm.

A. Framework Results

Figure 11 shows an overview of the improvement made to the cost table after the framework finished five full iterations as described by the previous sections. The initial cost is calculated using only the base library of manually designed motion primitives. Significant improvement can be seen in the final cost, especially in regions where the aircraft is close to the goal and facing away. Figure 11c explicitly illustrates this improvement, which is calculated as the initial cost minus the final cost, $J_0 - J_f$. The initial primitive library lacked maneuvers to quickly adjust heading, which makes it difficult to reach the goal from these positions. The learned library has found maneuvers that solve this issue, resulting in a much lower and smoother cost in these regions.

In total, 91 motion primitives were added to the primitive library during these five iterations. The average cost improvement was 21.6 s, and the maximum improvement between any given point on the initial and final cost map is 82.0 s. A few of the learned maneuvers are shown in Fig. 12. Qualitatively we can see that these primitives allow sharp turns from various trim states. Many of the learned turning maneuvers also end in one of the steady turn trim states. This may provide an advantage in planning by allowing the aggressive, fixed-time turns to be extended with a steady turn to give finer resolution when adjusting heading. Many learned motion primitives also have significant motion along the vertical axis. The kinetic energy is constrained at the endpoints by the initial and final trim states, but the exchange of kinetic and potential energy during the maneuver may improve some characteristics of the primitives (e.g., tighter turn radius).

Figure 13 shows the average cost at each iteration. The most significant overall improvement is made in the first few iterations. The average cost seems to converge to a final value after several iterations.

B. Application to Obstacle Field

A similar cost table was constructed for the obstacle field, shown in Fig. 14. Note that these cost maps are plotted in rectangular coordinates rather than cylindrical coordinates for better visualization of obstacle locations. The "holes" in these plots indicate locations within obstacles, which would be invalid goal positions.

Significant improvement is made between the cost maps generated with the base library and the learned library despite the learned library being trained only on the obstacle-free environment. This improvement demonstrates the transferability of the learned motion primitives to new tasks, in particular, more constrained tasks.

The average cost improvement in the obstacle field is 16.3 s. The maximum improvement between any given point on the initial and final cost map is 94.2 s.

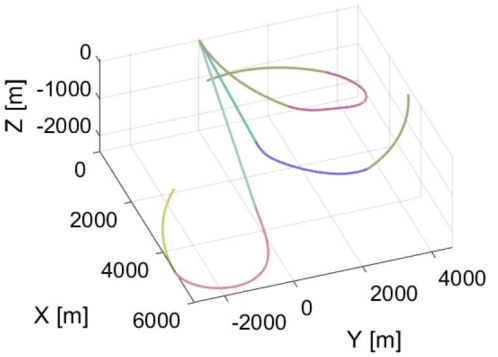


Fig. 12 Three of the learned motion primitives plotted with 10 s of their initial and final trim states.

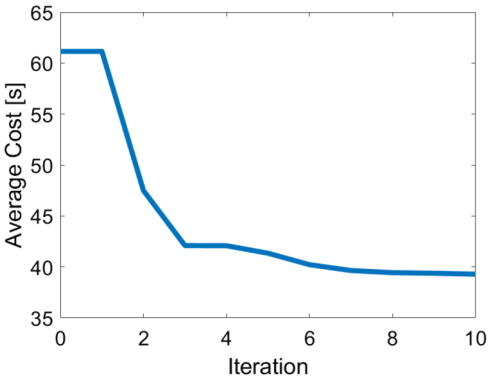


Fig. 13 Value of the integrated cost map for each iteration.

C. Validation

The learned library is validated by planning a path to a goal with Hybrid A* and executing the path in JSBSim with the controller described by Eqs. (13–22). An example of an obstacle-free path is shown in Fig. 15. Figure 15a shows the paths segmented into individual primitives by color, and Fig. 15b shows the reference trajectories and the actual trajectory taken by the controller. Additionally, these figures give a qualitative illustration of the difference between the base library path and the learned library path. The learned library is able to use sharper turns to take a more direct path to the goal, whereas the base library only has slow steady turns, which results in a longer path. The actual trajectories validate the feasibility of tracking the learned primitives within acceptable error margins.

Figure 16 shows similar trajectories for navigating obstacles. In these cases the radii of obstacles used in the Hybrid A* planning were expanded to leave a margin for tracking errors. The performance of the planner can also be illustrated with video animations [39].

D. Heuristic Analysis

A performance comparison of the Hybrid A* heuristics is shown in Table 1. The cost table heuristics result in a lower branching factor

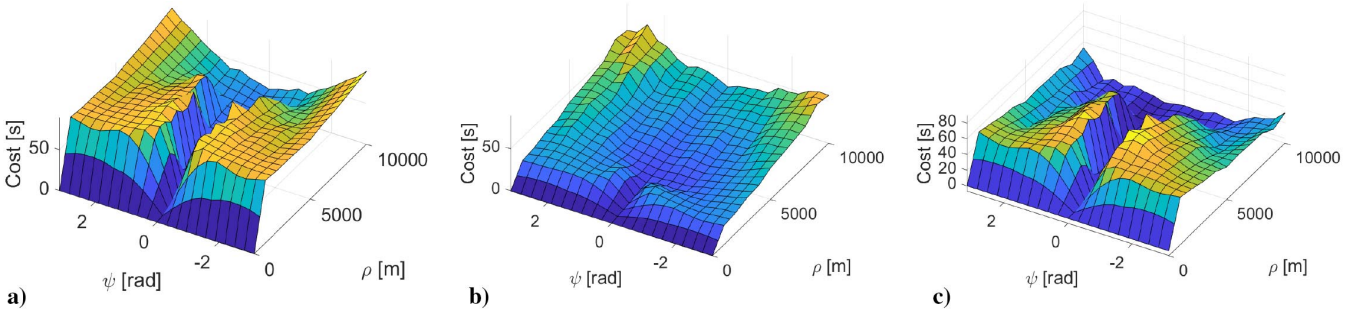


Fig. 11 Comparison of costs: a) original cost map, b) cost map after a full iteration of exploration, extraction, and re-evaluation, and c) difference between the original and improved maps.

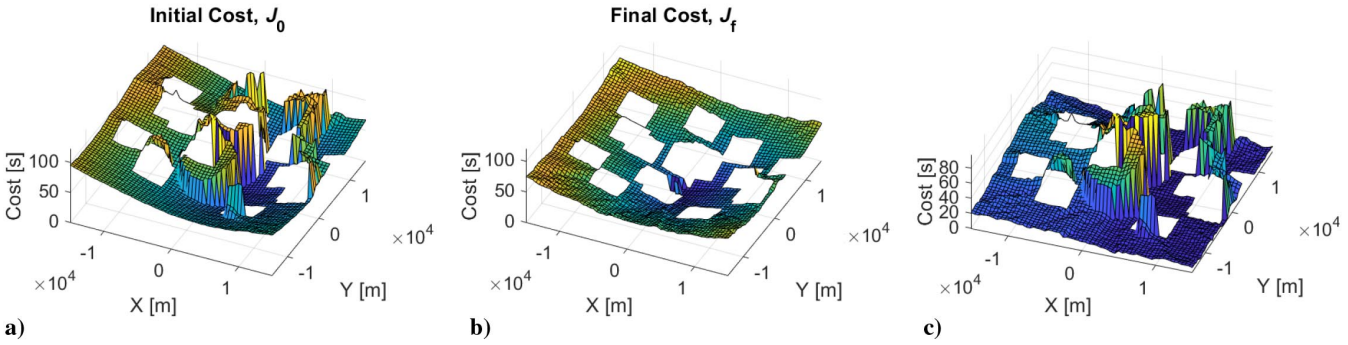


Fig. 14 Cost comparison for obstacle field: a) cost map in obstacle field with base library, b) cost map in obstacle field with learned library, and c) difference between the original and improved cost maps.

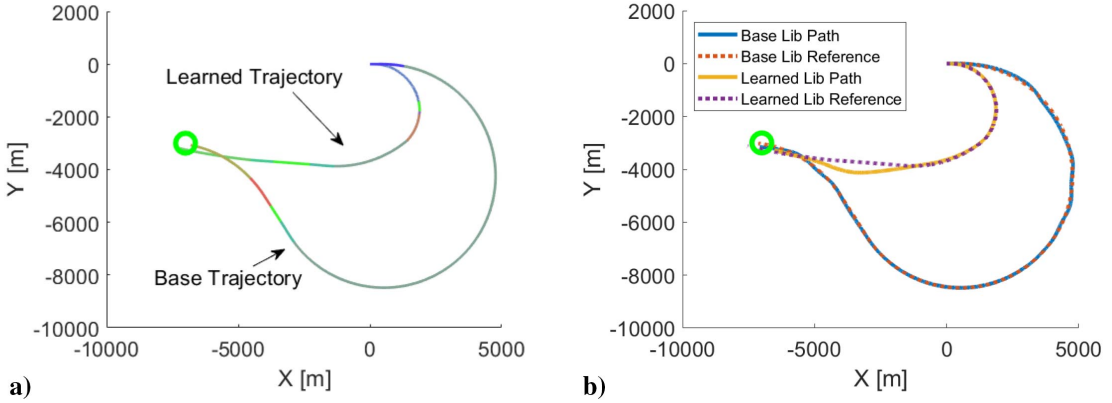


Fig. 15 a) Comparison of full trajectory generated by original primitives vs the learned primitives given the same goal position, and b) comparison of simulated trajectories following the planned path with the navigation controller.

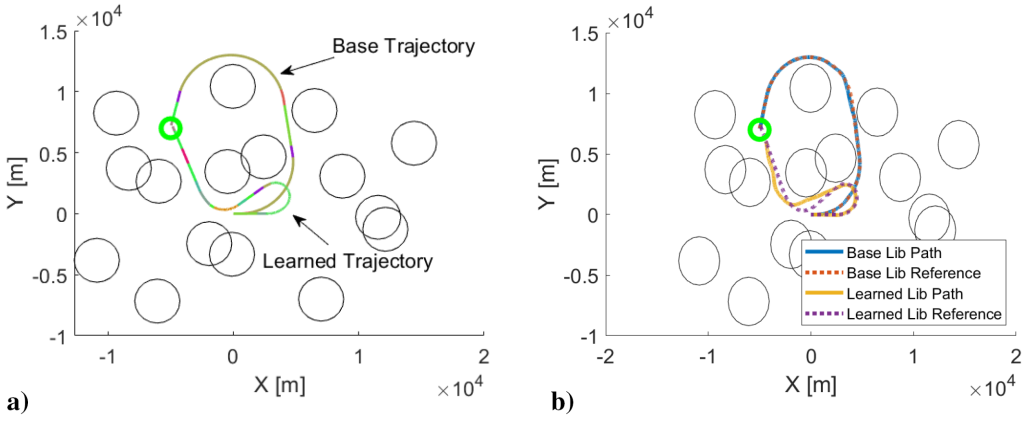


Fig. 16 a) Comparison of full trajectory generated by original primitives vs the learned primitives given the same goal position and obstacle field, and b) comparison of simulated trajectories following the planned path with the navigation controller.

and significantly fewer nodes expanded on average than the set of basic heuristics. The number of nodes expanded directly impacts the run time of the algorithm, though the run time is also effected by other factors such as processing power.

This reduction in nodes expanded demonstrates a significant advantage to the use of cost maps as heuristics. Additionally Hybrid A* allows for decoupling dimensions of the full problem to reduce storage space of the tables, which is not possible for value iteration. For example, in this case the goal is constrained in four dimensions: x position, y position, z position, and heading. If each of these dimensions is discretized into a vector of length N, then value iteration would need to compute a table of size $|Q_T|N^4$. As shown above, Hybrid A* can use the maximum of multiple decoupled tables as a heuristic instead. The size of these tables together is only $2|Q_T|N^2$, which scales much better with N than the fully coupled table. Hybrid A* can also still use a fully coupled table as a heuristic, which should improve its performance further, though that may not always be practical or necessary.

Table 1 Hybrid A* performance improvement with cost tables as heuristic functions

Heuristic	Nodes expanded	Branching factor	Runtime, s	Avg. path cost, s
Baselines	316,640	2.6318	4.6120	30.8516
Cost maps	52,005	2.2649	2.2649	28.2655

VII. Discussion

An initial set of motion primitives is not required for the framework to learn; however, no maneuvers can be learned until the framework

has found at least one trim. It may be beneficial to initialize the learning process with enough motion primitives to satisfy reachability when possible. Reachability is defined in [9] as the ability of the MA to steer the system from any initial hybrid state (q_0, h_0) to any desired hybrid state $(q_f, h_f) \in Q_T \times \mathcal{H}$, and requires that the set of trim primitives for a basis in $\mathcal{H}$ and the primitives form a strongly connected graph. This criterion can be met with only two trim states and two maneuvers connecting them if the trim states have different turn radii and nonzero climb rates with different signs. Additional trim states may help find more maneuvers earlier.

Deploying the library of learned motion primitives safely requires some conditions to be met. Satisfying reachability guarantees the MA can reach an arbitrary goal in obstacle-free space. If this criterion is not met in the base library, it can be checked by the turn radii and climb rates as stated in the previous paragraph. During training some trim states may be added without maneuvers to fully connect them to the rest of the library. These trim states and maneuvers leading into them may be removed to prevent some dead ends in the search. Additionally, runtime should also be short enough to plan in realtime. Runtime depends on the onboard hardware and the number of motion primitives. Since the hardware may be a fixed constraint, the learning process could be terminated at a given number of primitives.

The framework may produce a large number of motion primitives, which may include overspecialized or redundant maneuvers. The pruning method in Sec. IV.C may be modified to more aggressively remove maneuvers by requiring $J_m(q, h) < J_0(q, h) - \epsilon$, where ϵ is a nonnegative user parameter. This would require maneuvers to improve performance beyond a certain threshold.

The learning and planning methods in this work are intended to be broadly applicable for a range of aerospace planning problems. The proposed approach can produce kinodynamic motion plans from a start to a goal in the presence of hard (cannot enter) obstacles. Many civil and military applications can be abstracted to a known map with areas the aircraft should not enter. These can include corridors for other aircraft, areas with known weather, or defensive installations. The relative density and size of these obstacles will vary with the application. Civil transport applications may feature relatively empty maps, whereas military applications may have more numerous obstacles that must be avoided. The objective function may also vary with application. While minimum time was used in this work, this may lead to trajectories that are uncomfortable for passengers or reduce airframe life-time. Therefore, other objective functions such as minimum acceleration can be used. This can be introduced simply by changing the reward function for the RL agent and by changing the cost function for the Hybrid A* planner.

VIII. Conclusions

This paper presented an autonomous framework to improve the performance of a primitive-based motion planning algorithm through reinforcement learning. Improvement is made by adding new motion primitives learned through exploration of the given task. This autonomous framework was demonstrated on a point-to-point navigation task using a simulated F-16 model for the dynamics and the Hybrid A* search algorithm for the planner. Significant improvement was made to a small base library without human input or guidance during the training process.

The results show how the proposed framework can enable a primitive-based planner to be improved without human intervention. In addition, the results provide a new method for using reinforcement learning to improve the performance of deterministic planners. Reinforcement learning enables generation of new, complex, 3D primitives that can be used in a variety of environments.

Lastly, this paper demonstrated the value of the new primitive extraction framework and the modified Hybrid A* algorithm. The primitive extraction algorithm can be used to segment trajectories based on the MA definition. The modified Hybrid A* provides effective planning using motion primitives, and the updated heuristics reduce planning time. Additionally this work can be extended to more complex obstacle environments or even adversarial engagements. Future applications may allow this framework to learn novel primitives for such cases. These primitives could be used directly by a planner or transcribed into a human-readable library to gain qualitative insight for the given objective.

Acknowledgments

This work was supported by the Laboratory Directed Research and Development program at Sandia National Laboratories, a multimission laboratory managed and operated by the National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

References

[1] Frazzoli, E., Dahleh, M. A., and Feron, E., "A Hybrid Control Architecture for Aggressive Maneuvering of Autonomous Helicopters," *Proceedings of the 38th IEEE Conference on Decision and Control (Cat. No.99CH36304)*, Vol. 3, IEEE, New York, 1999, pp. 2471–2476. https://doi.org/10.1109/CDC.1999.831296

[2] Frazzoli, E., Dahleh, M. A., and Feron, E., "Maneuver-Based Motion Planning for Nonlinear Systems with Symmetries," *IEEE Transactions on Robotics*, Vol. 21, No. 6, 2005, pp. 1077–1091. https://doi.org/10.1109/TRO.2005.852260

[3] Barry, A. J., Florence, P. R., and Tedrake, R., "High-Speed Autonomous Obstacle Avoidance with Pushbroom Stereo," *Journal of Field Robotics*, Vol. 35, No. 1, 2018, pp. 52–68. https://doi.org/10.1002/rob.21741

[4] Camci, E., and Kayacan, E., "Planning Swift Maneuvers of Quadcopter Using Motion Primitives Explored by Reinforcement Learning," *2019 American Control Conference (ACC)*, Inst. of Electrical and Electronics Engineers, New York, 2019, pp. 279–285, https://ieeexplore.ieee.org/document/8815352/. https://doi.org/10.23919/ACC.2019.8815352

[5] Dharmadhikari, M., Dang, T., Solanka, L., Loje, J., Nguyen, H., Khedekar, N., and Alexis, K., "Motion Primitives-Based Path Planning for Fast and Agile Exploration Using Aerial Robots," *2020 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, New York, 2020, pp. 179–185. https://doi.org/10.1109/ICRA40945.2020.9196964

[6] Grymin, D. J., Neas, C. B., and Farhood, M., "A Hierarchical Approach for Primitive-Based Motion Planning and Control of Autonomous Vehicles," *Robotics and Autonomous Systems*, Vol. 62, No. 2, 2014, pp. 214–228, https://www.sciencedirect.com/science/article/pii/S0921889013001978. https://doi.org/10.1016/j.robot.2013.10.003

[7] Schouwenaars, T., Mettler, B., Feron, E., and How, J. P., "Robust Motion Planning Using a Maneuver Automation with Built-In Uncertainties," *Proceedings of the 2003 American Control Conference, 2003*, Vol. 3, IEEE, New York, 2003, pp. 2211–2216. https://doi.org/10.1109/ACC.2003.1243402

[8] Abbeel, P., Coates, A., and Ng, A. Y., "Autonomous Helicopter Aerobatics Through Apprenticeship Learning," *The International Journal of Robotics Research*, Vol. 29, No. 13, 2010, pp. 1608–1639. https://doi.org/10.1177/0278364910371999

[9] Frazzoli, E., Dahleh, M. A., and Feron, E., "Robust Hybrid Control for Autonomous Vehicle Motion Planning," *Proceedings of the 39th IEEE Conference on Decision and Control (Cat. No. 00CH37187)*, Vol. 1, Inst. of Electrical and Electronics Engineers, New York, 2000, pp. 821–826. https://doi.org/10.1109/CDC.2000.912871

[10] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al., "Mastering the Game of Go Without Human Knowledge," *Nature*, Vol. 550, No. 7676, 2017, pp. 354–359. https://doi.org/10.1038/nature24270

[11] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al., "Human-level Control Through Deep Reinforcement Learning," *Nature*, Vol. 518, No. 7540, 2015, pp. 529–533. https://doi.org/10.1038/nature14236

[12] Jaderberg, M., Czarnecki, W. M., Dunning, I., Marris, L., Lever, G., Castañeda, A. G., Beattie, C., Rabinowitz, N. C., Morcos, A. S.,

Ruderman, A., Sonnerat, N., Green, T., Deason, L., Leibo, J. Z., Silver, D., Hassabis, D., Kavukcuoglu, K., and Graepel, T., "Human-Level Performance in 3D Multiplayer Games with Population-Based Reinforcement Learning," *Science*, Vol. 364, No. 6443, 2019, pp. 859–865, https://science.sciencemag.org/content/364/6443/859. https://doi.org/10.1126/science.aau6249

[13] Dolgov, D., Thrun, S., Montemerlo, M., and Diebel, J., "Practical Search Techniques in Path Planning for Autonomous Driving," *Ann Arbor*, Vol. 1001, No. 48105, 2008, pp. 18–80.

[14] Nguyen, L., Ogburn, M., Gilbert, W., Kibler, K., Brown, P., and Deal, P., "Simulator Study of Stall/Post-Stall Characteristics of a Fighter Airplane with Relaxed Longitudinal Static Stability," Dec. 1979, https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19800005879.pdf.

[15] Frazzoli, E., Dahleh, M. A., and Feron, E., "Real-Time Motion Planning for Agile Autonomous Vehicles," *Proceedings of the 2001 American Control Conference. (Cat. No.01CH37148)*, Vol. 1, IEEE, New York, 2001, pp. 43–49. https://doi.org/10.1109/ACC.2001.945511

[16] Frazzoli, E., Dahleh, M. A., and Feron, E., "Real-Time Motion Planning for Agile Autonomous Vehicles," *Journal of Guidance, Control, and Dynamics*, Vol. 25, No. 1, 2002, pp. 116–129. https://doi.org/10.2514/2.4856

[17] Richards, N., Sharma, M., and Ward, D., "A Hybrid A*/Automaton Approach to On-Line Path Planning with Obstacle Avoidance," *AIAA 1st Intelligent Systems Technical Conference*, AIAA Paper 2004-6229, 2004. https://doi.org/10.2514/6.2004-6229

[18] Reeds, J., and Shepp, L., "Optimal Paths for a Car that Goes Both Forwards and Backwards," *Pacific Journal of Mathematics*, Vol. 145, No. 2, 1990, pp. 367–393.

[19] Nash, A., Daniel, K., Koenig, S., and Felner, A., "Thetâ*: Any-Angle Path Planning on Grids," *AAAI*, Vol. 7, July 2007, pp. 1177–1183.

[20] Garcia, M., Viguria, A., and Ollero, A., "Dynamic Graph-Search Algorithm for Global Path Planning in Presence of Hazardous Weather," *Journal of Intelligent & Robotic Systems*, Vol. 69, No. 1, 2013, pp. 285–295. https://doi.org/10.1007/s10846-012-9704-7

[21] Paranjape, A. A., Meier, K. C., Shi, X., Chung, S.-J., and Hutchinson, S., "Motion Primitives and 3D Path Planning for Fast Flight Through a Forest," *International Journal of Robotics Research*, Vol. 34, No. 3, 2015, pp. 357–377. https://doi.org/10.1177/0278364914558017

[22] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D., "Continuous Control with Deep Reinforcement Learning," arXiv preprint arXiv:1509.02971, 2015.

[23] Fujimoto, S., Van Hoof, H., and Meger, D., "Addressing Function Approximation Error in Actor-Critic Methods," arXiv preprint arXiv:1802.09477, 2018.

[24] Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S., "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," arXiv:1801.01290 [cs, stat], 2018, http://arxiv.org/abs/1801.01290.

[25] Faust, A., Oslund, K., Ramirez, O., Francis, A., Tapia, L., Fiser, M., and Davidson, J., "PRM-RL: Long-Range Robotic Navigation Tasks by Combining Reinforcement Learning and Sampling-Based Planning," *2018 IEEE International Conference on Robotics and Automation (ICRA)*, Inst. of Electrical and Electronics Engineers, New York, 2018, pp. 5113–5120, https://ieeexplore.ieee.org/document/8461096/. https://doi.org/10.1109/ICRA.2018.8461096

[26] Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., et al., "Soft Actor-Critic Algorithms and Applications," arXiv preprint arXiv:1812.05905, 2018.

[27] Camci, E., and Kayacan, E., "Learning Motion Primitives for Planning Swift Maneuvers of Quadrotor," *Autonomous Robots*, Vol. 43, No. 7, 2019, pp. 1733–1745. https://doi.org/10.1007/s10514-019-09831-w

[28] Li, T., Lambert, N., Calandra, R., Meier, F., and Rai, A., "Learning Generalizable Locomotion Skills with Hierarchical Reinforcement Learning," *2020 IEEE International Conference on Robotics and Automation (ICRA)*, Inst. of Electrical and Electronics Engineers, New York, 2020, pp. 413–419. https://doi.org/10.1109/ICRA40945.2020.9196642

[29] Harabor, D., and Grastien, A., "Online Graph Pruning for Pathfinding on Grid Maps," *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 25, AAAI, Menlo Park, CA, 2011.

[30] Yuan, Y.-X., "A Review of Trust Region Algorithms for Optimization," *International Council for Industrial and Applied Mathematics (ICIAM)*, Vol. 99, July 2000, pp. 271–282.

[31] Hart, P. E., Nilsson, N. J., and Raphael, B., "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE transactions on Systems Science and Cybernetics*, Vol. 4, No. 2, 1968, pp. 100–107. https://doi.org/10.1109/TSSC.1968.300136

[32] Konda, V. R., and Tsitsiklis, J. N., "Actor-Critic Algorithms," *Advances in Neural Information Processing Systems*, NeurIPS, San Diego, CA, 2000, pp. 1008–1014.

[33] Hallac, D., Vare, S., Boyd, S., and Leskovec, J., "Toeplitz Inverse Covariance-Based Clustering of Multivariate Time Series Data," *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)*, IJCAI, California, 2018, pp. 5254–5258. https://doi.org/10.1145/3097983.3098060

[34] Löw, T., Bandyopadhyay, T., and Borges, P. V. K., "Identification of Effective Motion Primitives for Ground Vehicles," *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, New York, 2020, pp. 2027–2034. https://doi.org/10.1109/IROS45743.2020.9341708

[35] Frazzoli, E., "Robust Hybrid Control for Autonomous Vehicle Motion Planning," Ph.D. Thesis, Massachusetts Inst. of Technology, Cambridge, MA, 2001, https://dspace.mit.edu/handle/1721.1/8703.

[36] Berndt, J., "JSBSim: An Open Source Flight Dynamics Model in C++," *AIAA Modeling and Simulation Technologies Conference and Exhibit*, AIAA Paper 2004-4923, 2004. https://doi.org/10.2514/6.2004-4923

[37] Zhu, Z., Lin, K., and Zhou, J., "Transfer Learning in Deep Reinforcement Learning: A Survey," arXiv preprint arXiv:2009.07888, 2020.

[38] Russell, S., and Norvig, P., *Artificial Intelligence: A Modern Approach*, Pearson Education, Inc., Upper Saddle River, NJ, 2002, pp. 103–104, Chap. 3.

[39] Goddard, Z., Wardlaw, K., Williams, K., Parish, J., and Mazumdar, A., "Improving Primitive-Based Planners Through Reinforcement Learning," 2021, https://youtu.be/4WnJ27p-0cs.

E. Atkins
Associate Editor