Inverse Optimal Planning for Air Traffic Control

Ekaterina Tolstaya[1], Alejandro Ribeiro[1], Vijay Kumar[1], Ashish Kapoor[2]

***Abstract*— We envision a system that concisely describes the rules of air traffic control, assists human operators and supports dense autonomous air traffic around commercial airports. We develop a method to learn the rules of air traffic control from real data as a cost function via maximum entropy inverse reinforcement learning. This cost function is used as a penalty for a search-based motion planning method that discretizes both the control and the state space. We illustrate the methodology by showing that our approach can learn to imitate the airport arrival routes and separation rules of dense commercial air traffic. The resulting trajectories are shown to be safe, feasible, and efficient.**

I. Introduction

Air traffic controllers (ATC) must follow a complex set of regulations, including requirements on spacing between airplanes, weather restrictions, and airport-specific departure and arrival protocols. Additionally, experienced ATCs often formulate strategies that balance various demands that arise from the complex interplay between these factors. The demand for ATC services, which are already stretched thin, will further increase due to the rapid progress in the field of aerial robotics.

We tackle the problem of building an Autonomous ATC that could significantly reduce the load on the human operators. In particular, we envision a system that concisely describes the rules of air traffic control and supports dense autonomous air traffic around commercial airports. There are several key challenges in building such an autonomous system. First, there are various deterministic and stochastic variables, such as traffic density, weather, regulatory requirements, and local geography. Often, there are multiple compliant ways, that can be qualitatively very different, to route air traffic. While a human operator can seamlessly optimize across these factors, it is not clear how an algorithmic system should weigh and jointly optimize across all these different criteria to mimic the choices of the human ATC. Secondly, most of the existing ATC services are developed for standard fixed-wing aircraft and helicopters. It is not clear how the current ATC system should be extended to novel aerial vehicles, such as micro UAVs and VTOL vehicles, which might have completely different dynamic behavior. Finally, from the algorithmic point of view, such a planning task requires optimization across multiple dynamic agents, which quickly becomes intractable as the number of vehicles increases.

Our novel approach combines search-based motion planning and inverse reinforcement leaning to address these challenges. The key technical insight is that, given a planner, we can learn a reward function that an ATC might be optimizing by leveraging aircraft traces available via the U.S. Federal Aviation Administration's Aircraft Situation Display to Industry feed. In particular, we learn the parameters of the reward function that correspond to how different factors are considered for trajectory selection.

The resulting trajectories attempt to imitate real air-traffic and are shown to be safe and feasible. The learned cost functions are interpretable and can be compared to existing standard procedures. Additionally, the decoupling between the planner and the reward functions means that we can change the dynamics to those of a new aircraft or aerial robot and the system can adjust without retraining. Further, leveraging robotic path-planners also helps with computational challenges and eliminates the need for directly learning a control policy.

A. Autonomous Air Traffic Control

There are significant efforts to redesign the air traffic control system to enable autonomous decision making. One popular research direction is focused on conflict resolution to prevent vehicles from entering unsafe states. For example, [1] aims to quantify the complexity of the current air traffic situation and provides a scheme for distributed control that ensures safety and eliminates the possibility of collisions. [2] develops a Markov Decision Process-based system for resolving conflicts in flight plans. There has been a significant thrust in the development of air traffic simulators and interactive systems for experiments with control algorithms and human factors. [3] describes a simulator developed by NASA and an example of a reinforcement learning approach for an airplane's greedy optimization of arrival times. Rather than manually designing a reactive system that eliminates conflicts, we seek to learn how to generate feasible trajectories for open-loop control in dense air traffic.

B. Control using Artificial Potential Fields

In mobile robot navigation, the penalty on unsafe states can be formalized as a spatial potential field. The artificial potential field is a popular method for efficient obstacle avoidance behaviors [4]. First, a potential field is generated based on known obstacles, and then the gradient of this potential field determines the direction of motion. One major drawback is the lack of guarantees for arrival at the origin

This work is supported by grants NSF DGE-1321851 and ARL DCIST CRA W911NF-17-2-0181.

[1]Department of Electrical and Systems Engineering, University of Pennsylvania, Philadelphia, PA 19104, USA [eig, aribeiro, kumar]@seas.upenn.edu

[2]Microsoft Corporation, Redmond, WA 98052, USA akapoor@microsoft.com

or obstacle avoidance. Getting stuck in local minima of the potential functions is a risk [4]. The evolutionary artificial potential function method has also been used for real-time robot path planning in the presence of dynamic obstacles [5].

Potential functions have been used extensively to model air traffic. [6] proposes a potential-field based system that can encode the effects of factors such as weather and air traffic density to manage en-route traffic. [7] focuses on the problem of re-routing of air traffic due to weather by generating potential functions and addresses the problems of local minima.

These works use heuristics to avoid local minima in the potential field. They also neglect the problem of path planning in dense airspace around airports immediately prior to landing. We follow the approach of [8], which uses an artificial potential function as a penalty so that an optimal planner avoids potentially unsafe states. We extend this approach to the Dubins Airplane model and learn the potential function.

C. Learning from Demonstrations

Inverse reinforcement learning is the problem of using expert demonstrations to learn the reward function that an expert is maximizing. This reward function can then be used to determine a controller that imitates the trajectories of the expert. Approaches such as behavior cloning [9] seek to directly find a mapping from states to experts' actions, but this may generalize poorly to new situations. Other approaches include maximum margin planning [10] and feature expectation matching [11], but these suffer from an ambiguity because one policy can be optimal for many different reward functions. We apply the maximum entropy inverse reinforcement learning algorithm described by [12]. This approach has been extended to deep reward functions and policies [13] and has been used in conjunction with planning for manipulation tasks [14].

II. Planning using the Dubins Airplane Model

We model the state of a single airplane using the Dubins airplane model, a four-dimensional system with a configuration space, $\mathcal{S} = \mathbb{R}^3 \times [-\pi, \pi]$, with $\mathbf{s} = (x, y, z, \phi)$, where x, y and z describe the coordinates of the airplane in three-dimensional Euclidean space, and $\phi \in [-\pi, \pi)$ is the bearing of the airplane relative to the $+x$ axis [15]. This airplane model assumes a fixed speed of v in the xy-plane.

The system is controlled through the first derivatives of altitude and bearing, $\dot{\phi}$ and $\dot{z}$, with control inputs denoted as u_z and u_ϕ, respectively. The system can be described as:

$$\dot{\mathbf{s}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} v\cos\phi \\ v\sin\phi \\ u_z \\ u_\phi \end{bmatrix} \tag{1}$$

This model can be used to plan a trajectory from an initial state $\mathbf{s}_0 = (x_0, y_0, z_0, \phi_0)$ to a given a goal configuration $\mathbf{s}_g = (x_g, y_g, z_g, \phi_g)$. This information is given to each airplane by air traffic control when the airplane approaches an airport. For example, $\mathbf{s}_g$ may be the location and orientation of an assigned runway. We assume that these assignments are provided prior to planning. The problem of planning a safe, minimum-length trajectory from the current state of the airplane to the goal can be formalized as follows:

$$\begin{aligned} &\underset{\mathbf{s}(t)}{\operatorname{argmin}} \int_{t_0}^{T} \|\dot{\mathbf{s}}(t)\|\, dt \\ &\text{s.t. } \dot{\mathbf{s}}(t) = \big(\cos\phi, \sin\phi, u_z(t), u_\phi(t)\big) \\ &\quad \mathbf{s}(t_0) = \mathbf{s}_0, \mathbf{s}(T) = \mathbf{s}_g \\ &\quad \mathbf{s}(t) \in \mathcal{S}^{safe}, \mathbf{u}(t) \in \mathcal{U} \end{aligned} \tag{2}$$

where $\mathcal{S}^{safe}$ is the set of safe states and $\mathcal{U}$ is the set of allowed control inputs. To enable learning the set of safe states $\mathcal{S}^{safe}$, we can reformulate (2) to use a soft penalty on possibly unsafe states, following the approach of [8], which uses hard constraints in addition to a potential function that guides the planned trajectory farther away from obstacles. The penalty term, $J\big(\mathbf{s}(t)\big)$, will be learned from data. For now, we express the new minimum path length planning problem as:

$$\begin{aligned} &\underset{\mathbf{s}(t)}{\operatorname{argmin}} \int_{t_0}^{T} \Big(1 + J\big(\mathbf{s}(t)\big)\Big) \|\dot{\mathbf{s}}(t)\|\, dt \\ &\text{s.t. } \dot{\mathbf{s}}(t) = \big(\cos\phi, \sin\phi, u_z(t), u_\phi(t)\big) \\ &\quad \mathbf{s}(t_0) = \mathbf{s}_0, \mathbf{s}(T) = \mathbf{s}_g \\ &\quad \mathbf{u}(t) \in \mathcal{U} \end{aligned} \tag{3}$$

The motion cost of a trajectory, $\tau : [0, \tau] \to \mathcal{S}$, is the sum of the path length and the line integral of the penalty:

$$C(\tau) = \int_{t_0}^{T} \Big(1 + J\big(\mathbf{s}(t)\big)\Big) \|\dot{\mathbf{s}}(t)\|\, dt. \tag{4}$$

Solving the motion planning problem (3) requires searching the space of all feasible trajectories. The continuous state and time problem is intractable, so we follow the approach of [16] by discretizing the system in time with an interval of $\Delta t = 30$ seconds. We also discretize the controls to obtain a set of fixed-time motion primitives. The bearing is chosen from the set $u_\phi \in \{-\Delta\phi, 0, \Delta\phi\}$ and the altitude change is chosen from the set $u_z \in \{-\Delta z, 0, \Delta z\}$. The motion primitives induce a discretization of states and enable the use of search-based motion methods for planning feasible and resolution-complete solutions [16]. While continuous states are denoted $\mathbf{s} \in \mathcal{S}$, we denote the discretized states as $\bar{\mathbf{s}} \in \mathcal{G}$, where $\mathcal{G}$ is a 4-dimensional grid, $\mathcal{G} \subset \mathcal{S}$. The induced discretization has a resolution of $\rho = [\rho_x, \rho_y, \rho_z, \rho_\phi]$, so the conversion from $\mathbf{s}$ to $\bar{\mathbf{s}}$ can be expressed using the floor function $\lfloor\ \rfloor$:

$$\bar{\mathbf{s}} = \left[\left\lfloor \frac{\mathbf{s}_x}{\rho_x} \right\rfloor, \left\lfloor \frac{\mathbf{s}_y}{\rho_y} \right\rfloor, \left\lfloor \frac{\mathbf{s}_z}{\rho_z} \right\rfloor, \left\lfloor \frac{\mathbf{s}_\phi}{\rho_\phi} \right\rfloor \right] \tag{5}$$

To solve the discretized problem, we apply the Anytime Repairing A* (ARA*) algorithm for finite-time path planning [17]. Please note the ε-suboptimality guarantee for the ARA* algorithm, which is expressed as:

$$C(\tau^*) \leq C(\tau) \leq \varepsilon C(\tau^*) \tag{6}$$

Algorithm 1 Dubins Airplane Heuristic

Input: Start $\mathbf{s}_0$, goal $\mathbf{s}_g$, forward speed v, max rate of climb Δz, turning rate $\Delta\phi$

Output: Minimum path length from start to goal, d_{min}

1: Compute Dubins car distance d_{xy} using the LSL, RSR, LSR, RSL, RLR, LRL paths with curvature $\kappa = \Delta\phi / v$

2: Compute the minimum time from start to goal in the xy plane

$$t_{min} = d_{xy}/v$$

3: Compute the minimum time for ascent/descent

$$t_z = |z_g - z_0|/\Delta z$$

4: **while** $t_z > t_{min}$ **do**

5: Add a helical ascent/descent to the trajectory

$$t_{min} = t_{min} + \frac{2\pi}{\Delta\phi}$$

6: **end while**

7: Compute the minimum path length from start to goal:

$$d_{min} = \sqrt{(v \cdot t_{min})^2 + (z_g - z_0)^2}$$

where $\varepsilon \geq 1$ and τ^* is the optimal path and $C(\tau^*)$ is its cost. The ability to sample suboptimal trajectories allows us to visit and learn about a larger variety of states during inverse reinforcement learning. Given the continuous trajectory defined by a series control inputs, we then use trajectory refinement to produce smoother trajectories using spline interpolation [16].

To use the ARA* algorithm for motion planning for fixed-wing vehicles, we must use a heuristic that provides a lower bound on the distance to the goal. For the non-holonomic Dubins airplane model, there are existing methods for computing the optimal path length from a start state to a goal state, but they are expensive to compute and require the consideration of low-altitude and high-altitude cases [15]. To simplify this computation, we always assume the high altitude case, which allows adding a helical descent or ascent without worrying about collisions with the ground. Our heuristic may therefore inadmissible, but is much more computationally efficient. The computation of this heuristic is summarized in Algorithm 1: First, we use the Dubins car model to compute the minimum path length from the start state to the goal state in the xy plane. Following the approach of [18], to find the shortest length path, we check each of six types of Dubins Car paths, with each path consisting of three segments: Right (R), Straight (S) and Left (L) [18]. Then, we compute the minimum time necessary for the ascent or descent. Then, if there is not enough time for the airplane to change altitude, we add helical sections to the trajectory to allow the airplane to descend in the z dimension and return to the same position and bearing in the xy plane.

III. Learning From Demonstrations

We assume that the set of safe states $\mathcal{S}$ in (2) is unknown. These conditions may be determined by the layout of the airspace at an airport, the motion of other airplanes, or weather. Our goal in this work is to learn about the unsafe conditions through inverse optimal control. To enable learning J using gradient descent, we use the soft penalty formulation in (3).

In order to learn the true penalty $J(\mathbf{s})$, we require a data set of expert demonstrations, $\mathcal{D} = \left\{s_i^e(t)\right\}_{i=1,\ldots,M}$. Although we do not know the true penalty $J(\mathbf{s})$, we have a current best estimate, $J_\theta(\mathbf{s})$. This cost, J_θ, is a function of non-linear features $\mathbf{f}$ of the state $\mathbf{s}$, defined in Section IV along with particular examples of cost functions. We parametrize J_θ as linear in the features of the state $\mathbf{f}(\mathbf{s})$:

$$J_\theta(\mathbf{s}) = \theta^T \mathbf{f}(\mathbf{s}). \tag{7}$$

The motion planner can use this estimated J_θ to plan trajectories satisfying (3). Our next goal is to formulate the loss function that will allow us to update J_θ using trajectories from the expert and learner.

We assume that the demonstrated expert trajectories follow the principal of maximum entropy [19], which states that the probability of an expert's trajectory τ with a lower cost is exponentially more likely to be selected than a trajectory with a higher cost:

$$\mathbb{P}(\tau) \propto e^{-C(\tau)} \tag{8}$$

Using this assumption, we can apply the maximum entropy inverse reinforcement learning algorithm described by [12], and its deep learning counterpart [13]. A detailed comparison of related works can be found in [14]. This approach seeks to find the cost function J_θ that *maximizes* the log likelihood of expert trajectories $\mathcal{D}$:

$$\mathcal{L}(\theta) = \log \mathbb{P}(\mathcal{D}, \theta | J_\theta) \tag{9}$$

If the cost function J_θ is a linear function the features, this problem is convex. [12] shows that the gradient of this objective is the difference in feature counts along the trajectories of the expert and the learner. To compute the gradient of $\mathcal{L}$ with respect to θ, we need to first introduce the state visitation counts of the expert computed using the data set $\mathcal{D}$:

$$\mathbf{f}_\mathcal{D} = \sum_{\tau \in \mathcal{D}} \sum_{\mathbf{s} \in \tau} \mathbb{P}(\tau) \mathbf{f}(\mathbf{s}) \tag{10}$$

In [12], the expert's empirical feature counts are compared to the expectation of the learner's state visitation counts:

$$\mathbb{E}[\mathbf{f}_\theta] = \sum_{\mathbf{s} \in \mathcal{S}} \mathbf{f}(\mathbf{s}) \mathbb{P}(\mathbf{s} | J_\theta) \tag{11}$$

Using these two quantities, we can express the gradient of $\mathcal{L}$ to be equal to the difference in feature counts along the trajectories of the expert and the learner [12], [13]:

$$\nabla_\theta \mathcal{L} = \mathbb{E}[\mathbf{f}_\theta] - \mathbf{f}_\mathcal{D} \tag{12}$$

As in [20], we assume that the distribution of the learner's sampled trajectories is uniform. Therefore, the expectation

Algorithm 2 Inverse Optimal Control for Air Traffic

for $n = 0, 1, \ldots, M$ **do**
  Receive expert trajectories ordered by arrival time
$$\mathcal{D} = \left\{\mathbf{s}_i^e(t)\right\}_{i=1,\ldots,M}$$
  for $i = 0, 1, \ldots, M$ **do**
    Obtain start state, $\mathbf{s}_0$, and end state, $\mathbf{s}_g$, of $\mathbf{s}_i^e(t)$
    Obtain other airplanes' trajectories, $\mathbf{s}_k^o(t)$ for $k < i$
    Use ARA* planner to minimize J_θ over $\mathbf{s}(t)$
    Compute stochastic gradient:
$$\hat{\nabla}_\theta \mathcal{L} = \sum_{t=t_0}^{T} f\big(\mathbf{s}(t)\big) - \sum_{t=t_0}^{T_i} f\big(\mathbf{s}_i^e(t)\big)$$
    Update cost parameters θ with step size α:
$$\theta_{t+1} = \theta_t + \alpha \hat{\nabla}_\theta \mathcal{L}$$
  end for
end for

$\mathbb{E}[\mathbf{f}_\theta]$ can be estimated using samples from learner with the current cost function J_θ and we can use the stochastic gradient computed using trajectories from the learner with the current cost J_θ, $\left\{\mathbf{s}_i(t)\right\}_{i=1,\ldots,N}$, and trajectories from the expert, $\left\{\mathbf{s}_i^e(t)\right\}_{i=1,\ldots,M}$:

$$\hat{\nabla}_\theta \mathcal{L} = \underbrace{\frac{1}{N}\sum_{i=1}^{N}\sum_{t=t_0}^{T_i} f\big(\mathbf{s}_i(t)\big)}_{\text{learner}} - \underbrace{\frac{1}{M}\sum_{i=1}^{M}\sum_{t=t_0}^{T_i} f\big(\mathbf{s}_i^e(t)\big)}_{\text{expert}} \tag{13}$$

Gradient ascent on the objective $\mathcal{L}(J)$ will increase the cost of the states that the learner visits, but the expert does not, so that the learner will avoid those states in the future. In practice, we set $M = N = 1$.

We summarize our approach in Algorithm 2. The learner obtains sets of time-synchronized expert trajectories for the landings of two or more airplanes. Then, for each trajectory in the expert's data set, the ARA* planner is used to plan a trajectory between the same start and end states. Given the expert's and learner's trajectories, we can then compute the stochastic gradient and perform a gradient step on the θ parameter of the cost function. If the planner fails to produce a solution within the time limit, we use only the expert's trajectory for the gradient computation.

IV. Safety in a Multi-agent System

Our next goal is to use prior knowledge to add structure to the cost function J to speed up learning. We assume that the centralized air traffic controller plans the landing trajectories for airplanes in the order of their arrival. When planning the trajectory for every new airplane, $\mathbf{s}(t)$, the trajectories of the n previous airplanes are known, denoted $\left\{\mathbf{s}_k^o(t)\right\}_{k=1,\ldots,n}$. We also know the location of the destination airport, $\mathbf{s}_a$.

Also, recall that the planning problem is computed in a discrete state space $\bar{\mathbf{s}} \in \mathcal{G}$, so we will only need to compute the cost of states on this discrete grid and can use the conversion from $\mathbf{s}$ to $\bar{\mathbf{s}}$ in (5). Therefore, rather than directly learning J, we only need to learn $\bar{J} : \mathcal{G} \to \mathbb{R}$. To relate this form to the previous notation, this is equivalent to choosing a feature extraction function $\mathbf{f}(s)$ that converts the continuous states $\mathbf{s}$ to discrete states $\bar{\mathbf{s}}$ and then applies additional non-linear operations.

We assume that $\bar{J}$ depends on two types of safety constraints: location relative to a specific airport, and the pairwise spacing between airplanes. The first component of the cost function $\bar{J}$ controls the airspace around airport a and can be expressed as $\bar{J}_a\big(\bar{\mathbf{s}}(t)\big)$. The second cost function controls the pairwise spacing of airplanes and can be written as $\bar{J}_o\big(\bar{\mathbf{s}}(t), \bar{\mathbf{s}}_k^o(t)\big)$, where $\bar{\mathbf{s}}_k^o(t)$ is the location of a nearby airplane at time t. If there are n other airplanes in the area, we must sum this objective for all other airplanes: $\sum_{k=1}^{n} \bar{J}_o\big(\bar{\mathbf{s}}(t), \bar{\mathbf{s}}_k^o(t)\big)$. Therefore, $\bar{J}\big(\bar{\mathbf{s}}(t)\big)$ can be written as a sum of the two types of soft penalties:

$$\bar{J}(\mathbf{s}(t)) = \bar{J}_a\big(\bar{\mathbf{s}}(t)\big) + \sum_{k=1}^{n} \bar{J}_o\big(\bar{\mathbf{s}}(t), \bar{\mathbf{s}}_k^o(t)\big) \tag{14}$$

Next, we parametrize $\bar{J}_a$ and $\bar{J}_o$ to allow us to learn these functions using gradient descent. Motivated by planning in the presence of potential functions [8], we choose to represent $\bar{J}_a$ as a spatial potential field, approximated using a fine discretization of the input space. This approximation represents the cost function as a value for each state $\bar{\mathbf{s}} \in \mathcal{G}$ on a discrete grid. A cost function used for motion planning must be non-negative. Also, the formulation must be piece-wise linear in features of state, $\mathbf{f}(\mathbf{s})$, to satisfy the assumptions of [12]. Therefore, we use a cost function of the following form:

$$\bar{J}_a\big(\bar{\mathbf{s}}(t)\big) = \sum_{\bar{\mathbf{s}}_i \in \mathcal{G}} \max\left\{w_i, 0\right\} \mathbb{1}_{\bar{\mathbf{s}}(t)=\bar{\mathbf{s}}_i}\big(\bar{\mathbf{s}}(t)\big), \tag{15}$$

where $\mathcal{G} \subset \mathbb{R}^4$ is a finite set of all states within the controlled airspace and $\mathbb{1}$ is the indicator function. By learning w_i for each discrete state in the space, we construct a look-up table of costs to enable efficient motion planning.

Next, we describe the penalty on pairwise distances between airplanes to control the airspace around each airplane. Each prior arrival is treated as a moving obstacle with a known trajectory, $\bar{\mathbf{s}}_k^o(t)$, and a cylindrical shape. We choose the penalty on getting too close to another airplane to be a linear drop-off potential function [21]:

$$\begin{aligned}\bar{J}_o\big(\bar{\mathbf{s}}(t), \bar{\mathbf{s}}_k^o(t)\big) =& u \max\left\{v_z - |\Delta_z(t)|, 0\right\} \\ & \cdot \max\left\{v_{xy} - \sqrt{\Delta_x^2(t) + \Delta_y^2(t)}, 0\right\},\end{aligned} \tag{16}$$

where $\Delta(t) = \bar{\mathbf{s}}(t) - \bar{\mathbf{s}}_k^o(t)$ and $\Delta_x(t)$, $\Delta_y(t)$ and $\Delta_z(t)$ are the components of the difference in the discrete positions of the two airplanes at time t. We consider the relative positions in the horizontal and vertical dimensions separately since the clearance requirements in altitude may be different. The v_{xy} and v_z thresholds are learned through gradient descent, while the scaling factor u is determined as a hyper parameter. The parameters for $\bar{J}_a$ and $\bar{J}_o$ can now be learned through

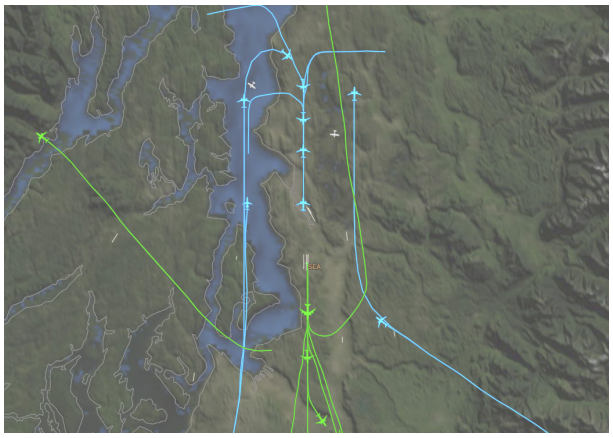


Fig. 1: A visualization of the Seattle-Tacoma airport data provided by FlightAware, with arrivals indicated in teal and departures in green [23].

stochastic gradient descent:

$$\theta = \left[\{w_i\}_{\bar{\mathbf{s}}_i \in \mathcal{G}}, v_{xy}, v_z\right] \tag{17}$$

The number of parameters w_i is large and equal to $|\mathcal{G}|$. In the next section, we describe the practical considerations of learning this set of parameters.

V. Methods and System Architecture

Now, we describe the implementation of the inverse optimal control system that learns from the air traffic data set.

A. Dataset Processing

In this work, we narrow our focus to airplane landings at the Seattle-Tacoma airport (SEA), but this approach can be generalized to take-offs or inter-airport routing. Specifically, our goal is to mimic the Standard Terminal Arrival Routes at the SEA airport [22]. We use recorded trajectories from the 11^{th} - 13^{th} of January, 2016 as provided by FlightAware [23]. A visualization of current data is displayed in Figure 1. The data set provides the location of airplanes asynchronously at a time interval of about 30 seconds.

The airplane locations were provided as GPS measurements of latitude, longitude and altitude from onboard instrumentation. Bearing was estimated from subsequent observed locations. The provided WGS-84 measurement (GPS latitude, longtiude and altitude location) were converted to a local east-north-up (ENU) Cartesian system centered on the location of the Seattle-Tacoma airport [24], [25]. All distances are provided in kilometers and bearing angles in radians from the $+x$ axis. The $+z$ axis indicates up and is perpendicular to the tangent plane.

By utilizing the Dubins airplane model, we make a constant velocity assumption. Real airplane trajectories accelerate and decelerate, especially during take-offs and landings. Extending our approach to acceleration or jerk-based motion primitives as in [26] is left to future work.

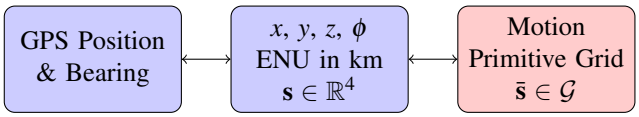


Fig. 2: Three parametrizations are used for expressing airplane states. The raw data provided by FlightAware is given in GPS latitude, longitude and altitude. We can convert from GPS to a local east-north-up (ENU) Cartesian frame. Then, we discretize the state with a resolution of ρ to obtain the motion primitive grid coordinates via (5). Continuous state representations are denoted in blue and discrete - in red.

B. Interpolation

The FlightAware data set provides waypoints at a time interval of about 30 seconds. We needed to perform interpolation, which is required for three reasons: 1) to generate time-synchronized trajectories for multiple airplanes, 2) to approximate the cost along a continuous trajectory, and 3) to generate dense trajectory data for SGD updates. The interpolation is computed for the x, y and z components of the trajectory separately using the Univariate Spline module of the SciPy library [27]. The bearing of the aircraft is then computed using subsequent x_t, y_t, z_t locations, $\phi_t = \arctan2(y_{t+1} - y_t, x_{t+1} - x_t)$.

C. Planning

The ARA* planner is given a fixed time limit of 30 seconds. The planner does not always find a solution within the time limit, in which case we assume that the learner obtained a trajectory with zero cost and no states. This provides a lower bound on the learner's cost and assumes that the learner is able to instantly move directly from that start to the goal. In this case, we can use only the expert's trajectory for the stochastic gradient to update the cost function. The parameters of the motion primitives, such as the maximum turning rate and maximum rate of climb, were chosen based capabilities of commercial aircraft and tuned through a grid search procedure by comparing with actual trajectories. The parameters were chosen to be: forward velocity $v = 100$ m/s, rate of climb $\Delta z = 6$ m/s, time discretization $\Delta t = 30$ s, angular velocity $\Delta\theta_1 = 0.025$ radians/s. An additional allowed turning rate of $\Delta\theta_2 = 0.0025$ radians/s mitigated some of the imprecision resulting from the coarse time discretization. The Dubins airplane heuristic still uses the largest turning speed to compute an approximate lower bound on the path length. The airplane states were discretized to a resolution of $\rho_x = \rho_y = 125$ m in the x and y dimensions and $\rho_z = 50$ m in the z dimension. The bearing was discretized to $\rho_\phi = 0.05$ radians. Also, due to the discretization of the control space, it is extremely unlikely that the planned trajectories end up at exactly the goal state. Therefore, rather than having a single goal state, we use a goal region of $\pm[500, 500, 25]$ meters and ± 0.125 radians centered around the original goal state.

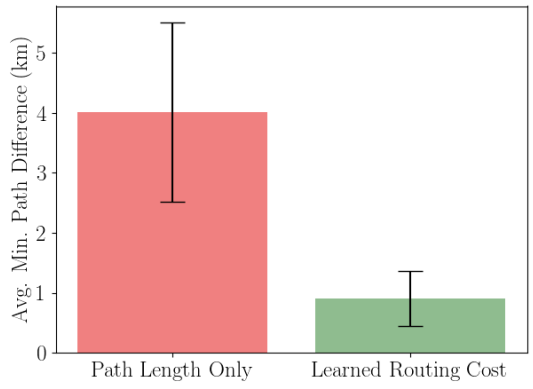


Fig. 3: We quantify the planning performance of a planner that uses the learned $\bar{J}_a$ function and a planner that minimizes the path length criterion only. The average minimum path difference is $\frac{1}{N}\sum_{i=1}^{N}\frac{1}{T}\sum_{t=1}^{T}\min_u ||\bar{\mathbf{s}}_i(t)-\bar{\mathbf{s}}_i^e(u)||_2$, a measure of how far the learner's trajectory strays from the expert's demonstration on average. Error bars denote a one standard deviation bound and $N = 300$.

D. Learning

To fit the cost function $\bar{J}_a$ as described in (15), we need to store cost function values for each discrete state in the state space. We observe that the true structure of the data is sparse, so rather than storing a dense look-up table of cost values, we implement a sparse hash-map approach, which allows a constant-time evaluation of a state's cost. $\bar{J}_a$ is initialized to a cost of $w_i = 100$ for all states. When we observe a particular state and perform a gradient step, we store the new value in the hash table. To further improve computational efficiency, the cost function was discretized on a slightly coarser grid that that of the motion primitive discretization, with resolution 0.25 in the x and y dimensions and 0.125 in the z dimension. The bearing was discretized to 0.125 radians. A step size of $\alpha = 10.0$ was used for learning.

The discretization of the cost function presents a challenge if the localization information is not exact. The provided dataset uses GPS locations, which are susceptible to drifts and sensor noise. One possible solution is evaluating the cost function at planning time by computing the average of multiple nearby cells of the cost function grid. This is expensive due to the sparse data structure used to store the values of the cost function. Instead, we add Gaussian noise to the state $\mathbf{s}_t$ before the gradient update of the cost function (13). A small amount of white Gaussian noise is added to the input state $\mathbf{s}_t$ before discretization: $\mathbf{w}_t \sim \mathcal{N}(0,\Sigma)$, where $\Sigma = [0.25, 0.25, 0.125]I$. The cost function $\bar{J}_o$ was initialized with overly conservative thresholds of $[v_{xy} = 60, v_z = 60]$ in discretized units, which is equivalent to $[7.5, 3]$ km. A step size of $\alpha = 0.01$ was used, with gradients larger than 100.0 clipped. The slope of the cost function is $u = 1.0$.

VI. Results

The arrival route function $\bar{J}_a$ has an extremely large number of parameters and therefore requires more data to train. We trained this function first without considering distances between airplanes. Then, we fixed the routing cost function while learning the inter-airplane safety function $\bar{J}_o$.

A. Learning the Airport Routing Cost

First, we train the airport routing cost, $\bar{J}_a$, that defines the allowed paths for airplanes arriving to the Seattle-Tacoma airport. Rather than the expensive computation of $\mathcal{L}(J_\theta)$ (9), we benchmark performance by the margin between the cost of the learner's trajectory and that of the expert, given an expert trajectory $\bar{\mathbf{s}}_i^e(t)$ and a corresponding planned trajectory $\bar{\mathbf{s}}(t)$:

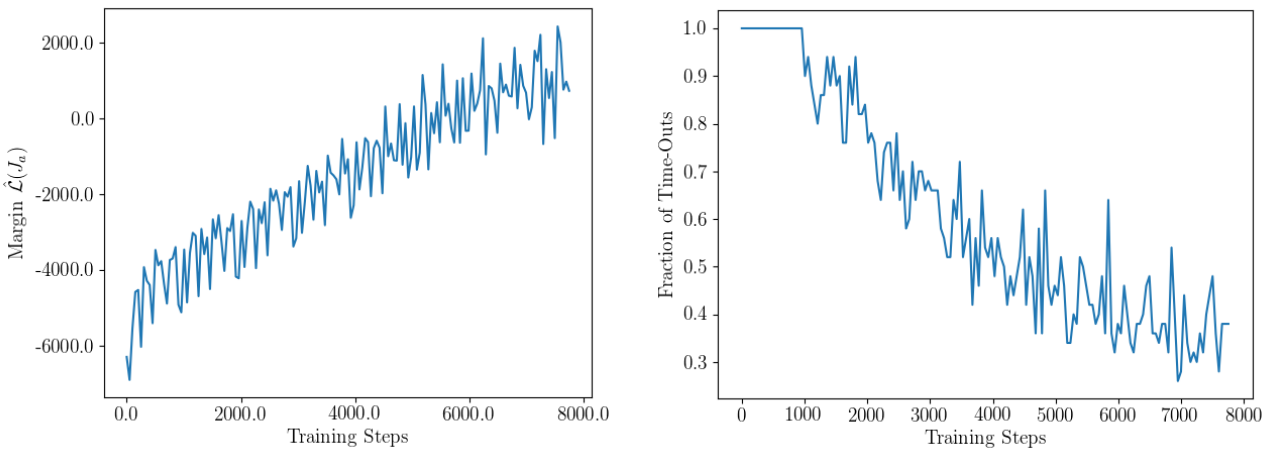
$$\hat{\mathcal{L}}(\bar{J}_a) = \sum_{t=t_0}^{T_i} \bar{J}_a\big(\bar{\mathbf{s}}(t)\big) - \sum_{t=t_0}^{T_i} \bar{J}_a\big(\bar{\mathbf{s}}_i^e(t)\big) \tag{18}$$

As the cost of the expert trajectory decreases relative to that of the expert, the objective increases and the probability of the expert's trajectory (9) increases. In Figure 4a, we observe that the benchmark $\hat{\mathcal{L}}(J_\theta)$ increases during training.

It is important to note that the use of a fixed time limit for the ARA* planner often produces time-outs. In this case, the gradient step is computed using only the expert trajectory, treating the learner's trajectory as a cost of zero. Also, to speed up learning for the first 1000 iterations, we used only the expert's trajectory for the gradient step since the fraction of time-outs is large at the beginning of training. It is interesting that the number of time-outs decreases during training, as we can see in Figure 4b. As the cost of the expert trajectories and the corresponding states decreases, the Dubins airplane heuristic becomes a tighter lower bound on the motion cost of the trajectories that move through those states. In Figure 5, we can see two examples of planned trajectories. We compare a trajectory planned using only the minimum path length objective to a trajectory planend using the learned $\bar{J}_a$ cost function. After training, the learner closely follows the expert's trajectory, with small oscillations in the z-axis. Figure 3 quantifies this comparison of the minimum path length planner and the routing cost planner over 1000 trajectories.

B. Learning the Inter-Airplane Safety Cost

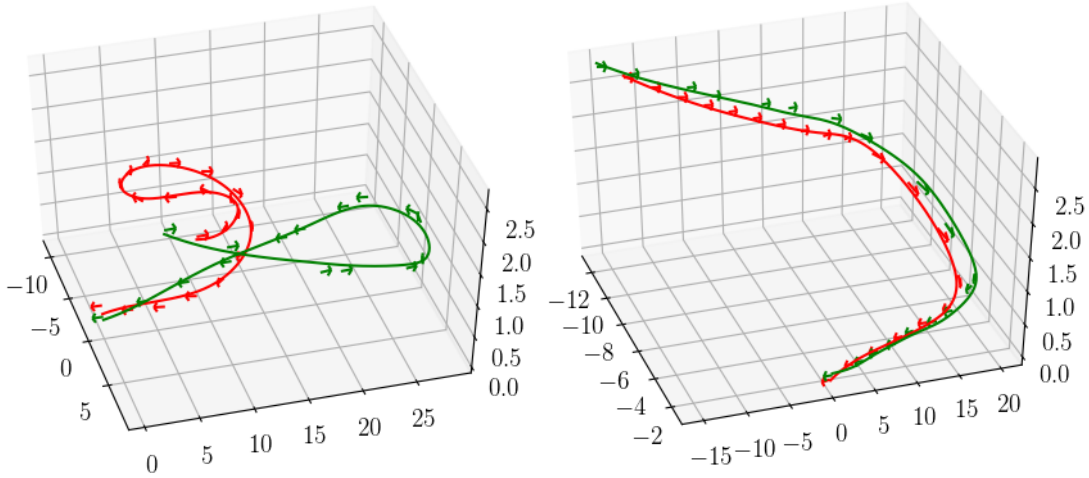
Now, we turn to learning the cost function that controls the spacing between pairs of airplanes, $\bar{J}_o$. We examine an example of five concurrent landing trajectories planned using the learned costs $\bar{J}_a$ and $\bar{J}_o$ in Figure 6. The cost function $\bar{J}_a$ is visualized in green, with darker values indicating lower cost, and lighter values indicating higher cost. The five airplane trajectories are indicated with colored curves, with the current location of each airplane marked with a small filled circle. The thresholds of the $\bar{J}_a$ function are marked with large unfilled circles of corresponding colors. If a sixth airplane approached the airport, it would consider these unfilled circles to be states with very high cost and would avoid these regions, eliminating the posibility of collisions.



(a) Cost Margin $\hat{\mathcal{L}}(\bar{J}_a)$

(b) Fraction of Time-outs

Fig. 4: First, we train the airport routing cost function $\bar{J}_a$. In Fig 4a, we observe that, $\hat{\mathcal{L}}(\bar{J}_a)$, the margin between the expert's cost and the learner's cost increases during training, which we use as a proxy for the true optimization objective $\mathcal{L}(\bar{J}_a)$. The ARA* planner is given a fixed time budget and sometimes exceeds that time budget without finding a solution. Fig 4b shows the fraction of time-outs, which decreases during training. In both plots, each data point is the average of 50 consecutive training steps.



(a) Minimum path length

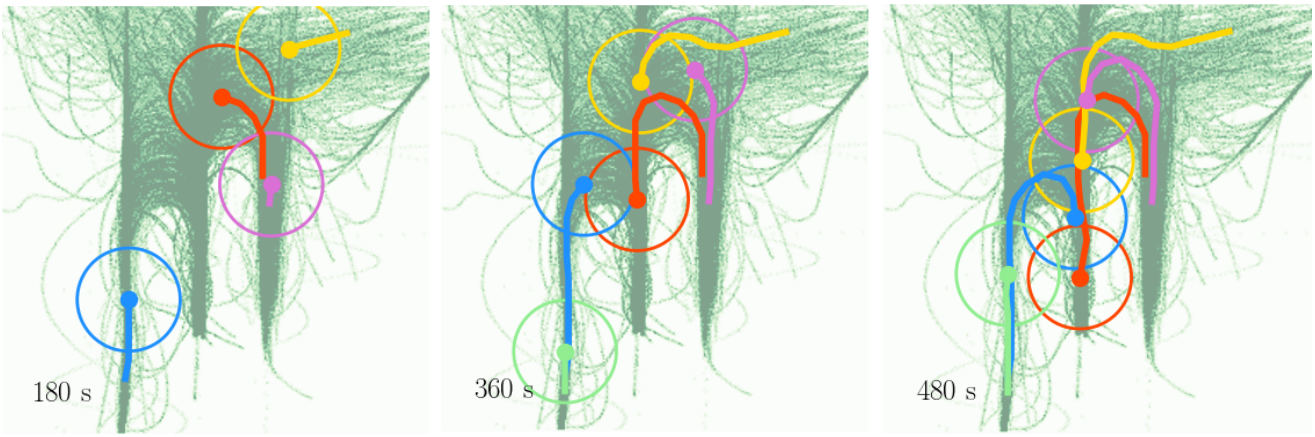
(b) Using $\bar{J}_a$

Fig. 5: A comparison of the expert's and learner's 3D trajectories for two different objective functions. The actual ATC trajectory is denoted with green arrows and a green spline. The planner's path is marked with red arrows and interpolated by the red curve. The units are in kilometers in the ENU coordinate system. In 5a, the planner minimizes the path length alone, and in Figure 5b, the planner uses the learned airport routing cost, $\bar{J}_a$.

VII. Conclusion

This work develops the method for search-based planning using cost functions learned through inverse optimal control. We demonstrate that the learned cost functions encode the implicit safety criteria of human ATC trajectories while maintaining high efficiency comparable to that of human operators. Our approach is well suited for planning trajectories over longer distances of tens of kilometers, but the Dubins airplane dynamics model has obvious limitations due to the coarse discretization of the control space, which are especially apparent during the last stages of a landing. Given a data set with more precise location measurements at a higher frequency, one possible solution could be to directly learn the motion primitives from data. This approach would still need to be verified in a high-fidelity air traffic simulation.

We hypothesize that the learned cost functions could enable distributed control in dense airspaces, as an alternative to completely centralized trajectory generation and air traffic control. The airplane spacing function $\bar{J}_o$ learns the relative importance of other vehicles during trajectory planning, indicating which other vehicles can be ignored during planning.



(a) $t = 180$ s (b) $t = 360$ s (c) $t = 480$ s

Fig. 6: A top down view of five trajectories produced by the learner using both $\bar{J}_a$ and $\bar{J}_o$, with the trajectories of the airplanes denoted in red, blue, yellow, purple and green, and their current locations denoted with a small filled circle. The large unfilled circles denote the threshold of the $\bar{J}_o$ cost, preventing airplanes from getting too close to each other. The learned cost function $\bar{J}_a$ is in the background indicated by green shading, with dark green indicating low cost states, and white indicating high cost states. We observe that the controller maximizes the efficiency of this landing sequence by pushing the trajectories of the airplanes as close together as allowed by the safety criterion.

The airport routing function $\bar{J}_a$ may be found to encode which areas of the airspace are unused and can be utilized by other autonomous air traffic.

References

[1] M. Prandini, L. Piroddi, S. Puechmorel, and S. L. Brázdilová, "Toward air traffic complexity assessment in new generation air traffic management systems," *IEEE transactions on intelligent transportation systems*, vol. 12, no. 3, pp. 809–818, 2011.

[2] Z. Mahboubi and M. J. Kochenderfer, "Autonomous air traffic control for non-towered airports," in *Proc. USA/Eur. Air Traffic Manage. Res. Develop. Seminar*, 2015, pp. 1–6.

[3] K. Tumer and A. Agogino, "Distributed agent-based air traffic flow management," in *Proceedings of the 6th international joint Conf. on Autonomous agents and multiagent systems*. ACM, 2007, p. 255.

[4] Y. Koren and J. Borenstein, "Potential field methods and their inherent limitations for mobile robot navigation," in *Proceedings. 1991 IEEE Int. Conf. on Robotics and Automation*. IEEE, 1991, pp. 1398–1404.

[5] P. Vadakkepat, K. C. Tan, and W. Ming-Liang, "Evolutionary artificial potential fields and their application in real time robot path planning," in *Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No.00TH8512)*, vol. 1, July 2000, pp. 256–263 vol.1.

[6] K. Ramamoorthy, T. Singh, and J. Crassidis, "Potential functions for en-route air traffic management and flight planning," in *AIAA Guidance, Navigation, and Control Conf. and Exhibit*, 2004, p. 4878.

[7] X. Xu, C. Li, and Y. Zhao, "Air traffic rerouting planning based on the improved artificial potential field model," in *Control and Decision Conf. (CCDC), 2010 Chinese*. IEEE, 2010, pp. 1444–1449.

[8] S. Liu, K. Mohta, N. Atanasov, and V. Kumar, "Towards search-based motion planning for micro aerial vehicles," *arXiv preprint arXiv:1810.03071*, 2018.

[9] D. A. Pomerleau, "Efficient training of artificial neural networks for autonomous navigation," *Neural Computation*, vol. 3, no. 1, pp. 88–97, 1991.

[10] N. D. Ratliff, J. A. Bagnell, and M. A. Zinkevich, "Maximum margin planning," in *Proceedings of the 23rd Int. Conf. on Machine learning*. ACM, 2006, pp. 729–736.

[11] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the twenty-first Int. Conf. on Machine learning*. ACM, 2004, p. 1.

[12] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning." in *AAAI*, vol. 8. Chicago, IL, USA, 2008, pp. 1433–1438.

[13] M. Wulfmeier, P. Ondruska, and I. Posner, "Maximum entropy deep inverse reinforcement learning," *arXiv preprint arXiv:1507.04888*, 2015.

[14] C. Finn, S. Levine, and P. Abbeel, "Guided cost learning: Deep inverse optimal control via policy optimization," in *Int. Conf. on Machine Learning*, 2016, pp. 49–58.

[15] H. Chitsaz and S. M. LaValle, "Time-optimal paths for a dubins airplane," in *Decision and Control, 2007 46th IEEE Conf. on*. IEEE, 2007, pp. 2379–2384.

[16] S. Liu, N. Atanasov, K. Mohta, and V. Kumar, "Search-based motion planning for quadrotors using linear quadratic minimum time control," in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ Int. Conf. on*. IEEE, 2017, pp. 2872–2879.

[17] M. Likhachev, G. J. Gordon, and S. Thrun, "Ara*: Anytime a* with provable bounds on sub-optimality," in *Advances in neural information processing systems*, 2004, pp. 767–774.

[18] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of mathematics*, vol. 79, no. 3, pp. 497–516, 1957.

[19] E. T. Jaynes, "Information theory and statistical mechanics," *Physical review*, vol. 106, no. 4, p. 620, 1957.

[20] M. Kalakrishnan, P. Pastor, L. Righetti, and S. Schaal, "Learning objective functions for manipulation," in *2013 IEEE Int. Conf. on Robotics and Automation*. IEEE, 2013, pp. 1331–1336.

[21] R. Murphy, R. R. Murphy, and R. C. Arkin, *Introduction to AI robotics*. MIT press, 2000.

[22] F. A. Administration. (2016) Terminal procedures publication. [Online]. Available: https://www.faa.gov/air_traffic/flight_info/aeronav/productcatalog/ifrcharts/terminalprocedures/

[23] FlightAware. (2019) Seattle-tacome intl airport. [Online]. Available: https://flightaware.com/live/airport_status_bigmap.rvt?airport=KSEA

[24] J. Farrell and M. Barth, *The global positioning system and inertial navigation*. Mcgraw-hill New York, 1999, vol. 61.

[25] G. van Drimmelen. (2018) Gps utils. [Online]. Available: https://gist.github.com/govert/1b373696c9a27ff4c72a

[26] S. Liu, K. Mohta, N. Atanasov, and V. Kumar, "Search-based motion planning for aggressive flight in se (3)," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2439–2446, 2018.

[27] S. Community. (2019) Univariate spline. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.UnivariateSpline.html