Adversarial Sampling-Based Motion Planning

Hayden Nichols[1], Mark Jimenez[2], Zachary Goddard[1], Michael Sparapany[3], Byron Boots[4], and Anirban Mazumdar[1]

***Abstract*—There are many scenarios in which a mobile agent may not want its path to be predictable. Examples include preserving privacy or confusing an adversary. However, this desire for deception can conflict with the need for a low path cost. Optimal plans such as those produced by RRT* may have low path cost, but their optimality makes them predictable. Similarly, a deceptive path that features numerous zig-zags may take too long to reach the goal. We address this trade-off by drawing inspiration from adversarial machine learning. We propose a new planning algorithm, which we title Adversarial RRT*. Adversarial RRT* attempts to deceive machine learning classifiers by incorporating a predicted measure of deception into the planner cost function. Adversarial RRT* considers both path cost and a measure of predicted deceptiveness in order to produce a trajectory with low path cost that still has deceptive properties. We demonstrate the performance of Adversarial RRT*, with two measures of deception, using a simulated Dubins vehicle. We show how Adversarial RRT* can decrease cumulative RNN accuracy across paths to 10%, compared to 46% cumulative accuracy on near-optimal RRT* paths, while keeping path length within 16% of optimal. We also present an example demonstration where the Adversarial RRT* planner attempts to safely deliver a high value package while an adversary observes the path and tries to intercept the package.**

***Index Terms*—Motion and Path Planning, Deep Learning Methods, Integrated Planning and Learning.**

I. Introduction

DECEPTIVE path planning (DPP) aims to minimize the probability that an observer will correctly identify the planner's intended destination before it has been reached. DPP is relevant for adversarial environments and for privacy protection. In this work, we present an example use case in which a DPP algorithm could be a security measure against piracy in the delivery of high-value cargo. While past works have examined deception, they have primarily focused on deceiving humans [1], [2] or deceiving domain-specific algorithms [3], [4]. However, deep learning methods have shown promise for predicting trajectories and navigation goals from time-series data [5]. Such methods can provide potential improvements over human experts such as speed and autonomous performance. In addition, recent results showed that deep learning could provide improved accuracy over domain-specific algorithms for navigational goal recognition [5]. Therefore, there is a need to examine how deceptive planners can deceive neural-network based observers.

One way to achieve deception is simply through randomness [6]. While Rapidly-Exploring Random Trees (RRT) rapidly generates highly random paths [7] that can be unpredictable, the path cost is often sub-optimal [8]. Moreover, RRT does not consider deception performance, and could occasionally produce plans that are not deceptive.

Optimized sampling-based algorithms such as RRT* [8] attempt to minimize a given cost function. However, path optimality is closely associated with indicating the intent of the actor [9]. Therefore, we seek to create an improved DPP algorithm by incorporating principles from adversarial machine learning [10]–[12]. Specifically, we assume the adversary observer is using a neural-network based classifier and we

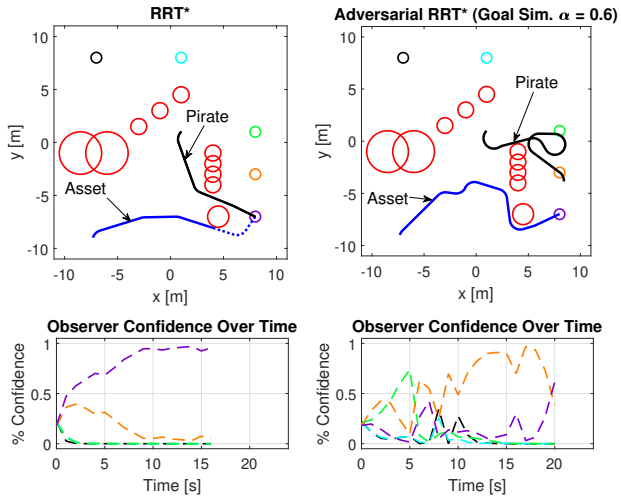


Fig. 1: Illustration of the "Pirate Deception Scenario". Red circles are obstacles. The 5 colored circles are possible goals. Below each is the observer confidence in each goal over time.

Manuscript received: September, 9, 2021; Revised December, 6, 2021; Accepted January, 13, 2022.

This paper was recommended for publication by Stephen J. Guy upon evaluation of the Associate Editor and Reviewers' comments.

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.

M. Jimenez was supported by the NSF SURE Robotics program (1757401).

[1]H. Nichols, Z. Goddard, and A. Mazumdar are with the George W. Woodruff. School of Mechanical Engineering, Georgia Institute of Technology, Atlanta, GA 30332, email: hnichols31@gatech.edu

[1]M. Jimenez is with the Department of Computer Science, University of Hawaii, Hilo, HI 96720, email: markjime@hawaii.edu

[2]M. Sparapany, is with Sandia National Laboratories, Albuquerque, NM 87123, email: mjspara@sandia.gov

[3]B. Boots is with the Paul G. Allen School of Computer Science and Engineering, University of Washington, Seattle, WA 98195, email: bboots@cs.washington.edu

Digital Object Identifier (DOI): see top of this page.

attempt to consider a deception metric while still minimizing path cost. We do so by incorporating a deception cost term into the RRT* algorithm using a Recurrent Neural Network (RNN). This produces low-cost paths that have deceptive properties. We term this approach "Adversarial RRT*."

An example environment is shown in Fig. 1. This features a starting position and a set of 5 potential goals. The planner in this work creates a plan that reaches one of the five goals and balances path cost (length) and deception. A neural-network based observer attempts to guess the goal of the planner using the observed time history of the vehicle trajectory. We assume the observer is a naive adversary with access to its own planner (RRT*) but unaware that any deceptive actions are being taken against it. We compare performance between standard RRT* and Adversarial RRT* on a Dubins vehicle model. We also study an adversary agent, termed a *pirate*. The *pirate* utilizes the observer predictions and RRT*, attempting to reach the intended goal before the planner. The left panel of Fig. 1 shows the planner and observer behavior with RRT*, while the right panel of Fig. 1 shows the planner and observer behavior with Adversarial RRT*. Note how the path qualitatively changes and the observer confidence fluctuates when Adversarial RRT* is used. Also note the effects of this observer confidence on the *pirate* behavior. We quantify performance by examining the RNN average cumulative prediction accuracy, as well as the success rate of each deceptive agent against the pirate.

The main contributions of this work are 1) the creation of two deception methods that leverage adversarial machine learning to deceive neural-network based observers, 2) incorporation of deception into an optimized kinodynamic planner (Adversarial RRT*), 3) the validation of the proposed method on a Dubins vehicle, and 4) the quantitative analysis of deception across planner and observer configurations.

This paper begins with an overview of relevant background principles and a discussion of our design choices. Next, we detail our representation of the Deceptive Path Planning problem, and our approach to the goal recognition problem. This is followed by a discussion of our Adversarial RRT* algorithm design. Finally, the paper concludes with an analysis of the Adversarial RRT* planning performance and results.

II. Background

This paper considers the scenario of traversal to one of a set of goals in an adversarial setting [3], [13]. One agent, the *planner*, is tasked with reaching the predetermined goal in a fixed environment, while avoiding obstacles. Meanwhile the opposing agent, the *observer*, is tasked with determining which goal the *planner* intends to reach as soon as possible.

A. Goal Recognition

Goal recognition consists of predicting the unobserved goal of an agent when given a sequence of its observed states [14], [15]. In the context of this work, goal recognition aims to predict the desired goal of a mobile agent (navigational goal recognition). Our proposed planner seeks to disrupt an observer performing navigational goal recognition. Existing work has made efforts to formally define and characterize deception and legibility of intent in robotic manipulator motions from the perspective of human observers [2], [9], [16]. A mathematical description of legibility and predictability of motion is given in Eq. (9) of [9]; legible behavior is defined as a sequence of actions which convey an agent's true intention. Moreover, legibility is considered closely correlated with optimality [9], [16], in that a path which is efficient from origin to goal more clearly conveys the agent's target.

Deep Learning, specifically neural networks, have been applied in goal recognition to automatically encode planning domain knowledge from raw data [17], [18]. Deep learning architectures have also shown efficacy in predicting object trajectories and navigation goals given sequential position measurements [5], [19]. Recurrent Neural Networks (RNN), are particularly suited to handle sequential inputs [20], [21]. Therefore, deep learning techniques hold promise for future navigational goal recognition applications.

B. Sampling-Based Planning

Sampling-based path planners are widely used due to their computational simplicity and speed. Examples include RRT and the optimized RRT* [7], [8]. These algorithms have been applied to a range of systems and have probabilistic completeness guarantees under general conditions [7]. While the basic RRT algorithm is implemented with line-segment edges, it has been adapted to non-holonomic dynamics using Dubins' paths [22]. Therefore, sampling based planners are a good starting point for an adversarial planner.

C. Adversarial Machine Learning

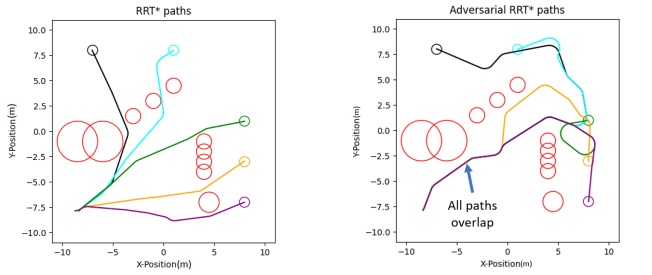
Machine learning offers great potential for classifying mobile vehicle trajectories. However, machine learning algorithms are vulnerable to adversarial attacks [10]. Such attacks create inputs that reduce the target agent's accuracy or confidence in prediction [12]. Analysis has shown that non-random perturbations of inputs can change the prediction of a neural network [10]. It has also been shown that image classification algorithms can be fooled with small changes to select input pixels [10]. Changes can be as simple as adding noise to an existing image [11] or altering a single input channel [23]. In the case of generating adversarial examples of robotic paths, the strategy of perturbing points on a viable path is complicated by the requirements to obey the robot's dynamic constraints, efficiently produce a plan, and avoid obstacles.

While Adversarial methods often utilize knowledge of the target network, they can also generalize well, meaning they can fool unknown networks as well [10]. This paper incorporates adversarial machine learning into an optimized sampling-based planner to deceive navigational goal recognition algorithms.

III. Mathematical Formulation

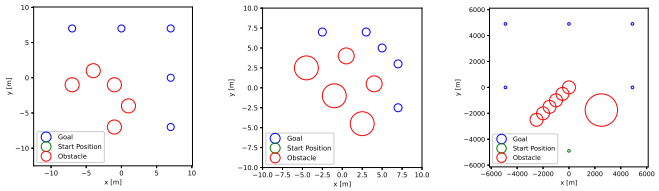
A. Planner

We assume a dynamically constrained mobile ground robot, whose state is given by $s_t = (x_t, y_t, \theta_t)$, must travel to a goal in one of a selection of goal sets, $g \in G_i \in \mathcal{G} = \{G_1, G_2, ..., G_n\}$, in a 3-dimensional environment, $\mathcal{X} \subset \mathbb{R}^3$,.



(a) RRT* (b) Adv. RRT* (Entropy $\alpha = 1$)

Fig. 2: Paths using 250 iterations of each algorithm.



(a) (b) (c)

Fig. 3: Additional environments simulated in this work.

This environment has obstacles, $\mathcal{O} \subset \mathcal{X}$, and goal sets $\mathcal{G}$ where $\mathcal{G}$ and $\mathcal{O}$ are disjoint.

A path is a continuous integral curve of the robot's dynamic constraints which connects the planner's initial state, $s_0 = (x_0, y_0, \theta_0)$, to its final state, $s_g = (x_f, y_f, \theta_f) \in g$. A path can be discretized into a sequence of states $S = (s_0, \ldots, s_f)$ where $||(s_{i+1}) - (s_i)||_2 < \epsilon$. A feasible path is any path, $\mathcal{P}$, which does not collide with any obstacles in the environment: $\texttt{Feasible}(\mathcal{P}) \iff \forall (x_i, y_i, \theta_i) \in \mathcal{P}, (x_i, y_i, \theta_i) \notin \mathcal{O}$. For our purposes, we take $s_g = (x_f, y_f, \theta_f), \theta_f \in \mathbb{R}$ such that the goal location does not depend on orientation. An example of such an environment are shown in Fig. 3. Some viable paths through that environment are shown in Fig. 2a.

The *planner* produces one viable path, $\mathcal{P}_{gt}$, to the selected goal. The *observer* makes a prediction at each time step for each goal candidate in the environment, represented as a multinoulli distribution [20], parameterized by the vector $\mathbf{c} \in [0, 1]^n$ over the set of n goal sets, $G_i \in \mathcal{G}$, where

$$\mathbf{c}_i(s_t) = P(G_i|s_t). \tag{1}$$

The focus of this paper is on the design of a *planner* which minimizes a cost, J. This cost should balance path performance and deception, and is described in Section V.B. In this work, we choose path performance to be the path length. We examine two deception metrics that are functions of path length and RNN predictions about the intended goal. The cost function in this work is a weighted sum of these two quantities.

B. Observer

The *observer* in this work performs navigational goal recognition [5]. The observer uses the current time history of the vehicle trajectory to predict its goal. In contrast, the *planner* attempts to "hide the real", by preventing its actions from showing its intended goal, as described in [2]. The *planner* is rewarded for producing a less legible path (a legible path conveys the planner's intended goal, described by Eq. (9) of [9]), but penalized for less efficient paths. The planner's optimization function treats deception as an inversion of probabilistic goal recognition as in [24]. Thus, our planner must devise a plan which is both efficient in terms of distance traveled, and deceptive in terms of causing the *observer* to have low confidence in the correct target goal.

C. Planner Model for Goal Recognition

In our planning/observation scenario it is necessary to approximate the goal recognition behavior of the observer. This enables the planner to reason about how to make a plan deceptive. We assume the planner has access to some function f^* which approximates the true distribution $P(G_i|S)$, modeling the observer's expectation of which goals are most likely. Our planner incorporates a neural network which has learned f^* from observing near-optimal paths, using this distribution to plan a path which is both efficient and deceives the observer. For this work, we utilize an RNN as our approximation for the observer. This is called the "planner model."

IV. Adversarial RRT* Algorithm Overview

This work modifies an RRT* framework to achieve deceptive path planning. This is done by incorporating a metric of deception into the RRT* cost function. We chose this combination because RRT* provides rapid, optimized motion plans. We then add deceptive considerations by approximating the observer behavior using the planner model RNN. This RNN takes path observations as an input and outputs confidences that the input path is going toward each output goal. This output is used to compute the deception score over the path. The following subsections describe the RRT* algorithm developed for this task, modifications to include the adversarial cost function, and platform-specific modifications for the Dubins' vehicle model.

A. RRT* Utilities and Data Structures

This subsection defines utility functions and data structures used in RRT* and our implementation of Adversarial RRT*.

The RRT* node is defined as $x = (s, \sigma, x_p, X_c, c)$, where s is the state, σ, is trajectory of states along the path to the node from its parent, x_p is the parent node, $x_c \in X_c$ references any children nodes, and c is the cost to reach the node.

We add fields to the base RRT* node structure for use with our adversarial planner. The planner model's hidden RNN state associated with the node is stored as h_{RNN}. This allows the RNN state to be set to that of a particular node for future predictions. We have also chosen to store references to any children nodes, $x_c \in X_c$ to facilitate cost re-propagation. The node is defined as $x = (s, \sigma, h_{RNN}, x_p, X_c, c)$.

The following are RRT* and Adv. RRT* helper functions.

CalcCost(x) : calculates the cost of a node according to the cost function. For our platform, RRT* node cost is the Dubins' path length to the node. Adv. RRT* cost is described

in Section V.B. For Adversarial RRT*, the planner model (RNN) is called and the node's internal RNN state is set here.

COST(x) : Returns the already-computed cost of a node.

EDGE(x) : Returns the edge tuple associated with a node (between the node and its parent).

OBSTACLEFREE(x) : Returns a boolean indicating if the path σ to a node from its parent is free of obstacles.

NEAREST(G,x) : Returns the nearest node (in Dubins path length) to x contained in the vertices, V (a component of the graph, G).

NEAR(G,x) : Returns a subset of the graph, G, containing nodes within a pre-configured radius (in Dubins path length) of the input node x.

STEER(...) : handles the creation of a path between two nodes. It also populates the data structure for the output node.

REPROPCOSTS(...) function is necessary for the cost function formulation, since it will depend on the planner model (RNN) output. When paths are rewired, it not only affects the cost of that rewired segment, but also of the down-branch segments as well. Future predictions from the planner model (RNN) rely on the hidden state, which is changed by rewiring. Thus, the cost must be recursively repropagated, starting with the rewired node and iterating through its children.

Algorithm 1 Adversarial RRT* Recursive Cost Repropagation

1: **function** REPROPCOSTS(x_{rewire})
2: x_{rewire}.cost $\leftarrow$ CALCCOST(x_{rewire});
3: **for** $x_{child} \in x_{rewire}$.children **do**
4: REPROPCOSTS(x_{child})
5: **end for**
6: **end function**

BESTPATH(...) function returns the best path by finding the goal node from V_g with the lowest cost. Starting with this node, the function steps back through the branch, collecting parent nodes in $\mathcal{P}$ until the start node is reached.

Algorithm 2 RRT* Best Path

1: **function** BESTPATH(G)
2: $\mathcal{P} \leftarrow \emptyset$
3: (V,E,V_g) $\leftarrow G$;
4: $x \leftarrow$ ARGMIN(V_g);
5: $\mathcal{P} \leftarrow \mathcal{P} \cup x$;
6: **while** x.parent is not None **do**
7: $x \leftarrow x$.parent;
8: $\mathcal{P} \leftarrow \mathcal{P} \cup x$;
9: **end while**
10: return $\mathcal{P}$
11: **end function**

B. RRT* Algorithm

Main Function:

The RRT* algorithm [8], shown in Algorithm 3, consists of a few steps: initialization, extending the graph, and building the final best path from the completed graph. The initialization process involves creating a node from the start position to populate the vertex set, V. Sets of edges, E, and goal nodes, V_g, are initialized as empty. These three sets, as a tuple, form the graph, G. Obstacle locations and the goal location are given at initialization. Extending the graph begins by randomly sampling a node. This node, the existing graph, and the set of acceptable goal states are passed to the EXTEND(...) function. This function handles the extension of the graph, and returns the updated sets of nodes, V', edges, E', and goal nodes V_g'. This graph extension is repeated for a predetermined number of iterations (denoted by N), after which, the BESTPATH(...) function is called to return the lowest cost path in the graph.

Algorithm 3 RRT* Algorithm

1: **function** RRT*(x_{init}, X_{goal})
2: $V \leftarrow \{x_{init}\}$; $E \leftarrow \emptyset$; $V_g \leftarrow \emptyset$;
3: **for** $i = 1$ to N **do**
4: $G \leftarrow (V,E,V_g)$;
5: $x_{rand} \leftarrow$ SAMPLE(i);
6: (V',E',V_g') $\leftarrow$ EXTEND(G,x_{rand},X_g);
7: $G \leftarrow (V',E',V_g')$
8: **end for**
9: return BESTPATH(G)
10: **end function**

Extend Procedure:

EXTEND(...), in Algorithm 4, takes a newly sampled node, x, attempts to add it to the graph G, and checks if the goal set, X_{goal} can be reached from the newly added node.

The function finds the nearest existing node using NEAREST(...). The STEER(...) function returns a node x_{new} at the newly sampled node location, reached from the nearest neighbor. This path to the new node is checked for obstacles using the OBSTACLEFREE(...) function. This condition must be satisfied before finding a best parent: if the path from nearest neighbor has an obstacle, searching over near neighbors may not yield a viable parent node, leading to inefficiencies.

If the path to x_{new} is obstacle-free, the algorithm checks the nearest neighbors within a specified radius, X_{near}, of the new node to see if x_{new} can be reached with a lower cost via another node. This process of the best-parent search is outlined in line 9-15 of Algorithm 4. This search involves using STEER(...) to steer from each near node in X_{near} (excluding $x_{nearest}$) to the sampled location. If obstacle-free, the cost of the resultant node (called $x_{new,near}$) is compared to x_{min}, which is the node reached by the current best parent. If the cost is reduced, x_{min} is updated with $x_{new,near}$.

At the conclusion of the best-parent search, an attempt is made to connect the new node, x_{min}, to the goal (lines 18-21 of Algorithm 4). This aims to determine if the goal is reachable from the new node, which has two implications. First, this enables tracking of all complete paths to the goal for constructing the final solution. This is important when running the algorithm for a limited number of iterations since the probabilistic completeness of RRT* only holds as the number of iterations gets very large. Second, this allows for using this RRT* implementation when constraints such as limited fuel or range are added to the vehicle model. If the vehicle cannot reach the goal from the newly added node due to such

constraints, this step allows for discarding the node. After this check, the graph is updated with the new information.

Algorithm 4 RRT* Extend

1: **function** EXTEND(G,x,X_{goal})
2: $(V,E,V_g) \leftarrow G$
3: $V' \leftarrow V$; $E' \leftarrow E$; $V'_g \leftarrow V_g$;
4: $x_{nearest} \leftarrow$NEAREST(G,x)
5: $x_{new} \leftarrow$ STEER($x_{nearest}$,x);
6: **if** OBSTACLEFREE(x_{new}) **then**
7: $x_{min} \leftarrow x_{new}$;
8: $X_{near} \leftarrow$ NEAR(G,x);
9: **for** all $x_{near} \in X_{near}/x_{nearest}$ **do**
10: $x_{new,near} \leftarrow$ STEER(x_{near},x);
11: **if** OBSTACLEFREE($x_{new,near}$) **and**...
12: ...COST($x_{new,near}$) $<$ COST(x_{min}) **then**
13: $x_{min} \leftarrow x_{new,near}$;
14: **end if**
15: **end for**
16: x_{min}.parent.children.APPEND(x_{min})
17: $V' \leftarrow V' \cup x_{min}$; $E' \leftarrow E' \cup$ EDGE(x_{min})
18: $x_{goal} \leftarrow$ STEER(x_{min},X_g)
19: **if** OBSTACLEFREE(x_{goal}) **and** $x_{goal} \in X_g$ **then**
20: $V'_g \leftarrow V'_g \cup x_{goal}$;
21: **end if**
22: **for** all $x_{near} \in X_{near}$ **do**
23: $x_{rewire} \leftarrow$ REWIRE(x_{min},x_{near})
24: **if** OBSTACLEFREE(x_{rewire}) **and**...
25: ...COST(x_{rewire}) $<$ COST(x_{near}) **then**
26: x_{near}.parent $\leftarrow x_{rewire}$.parent;
27: $x_{near}.\sigma \leftarrow x_{rewire}.\sigma$;
28: $E' \leftarrow E'/$EDGE(x_{near});
29: $E' \leftarrow E' \cup$EDGE(x_{rewire});
30: $G' \leftarrow$ REPROPCOSTS(G',x_{near});
31: **end if**
32: **end for**
33: **end if**
34: return (V', E', V'_g)
35: **end function**

Rewire Procedure:

The rewire procedure takes place in lines 22-33. This process iterates through the nearest neighbors in X_{near} and determines if the cost to reach each node can be reduced via the newest added node. To accomplish this, a path is created from x_{min} to x_{near} using REWIRE(...).

Then, the cost of the resultant node, x_{rewire} is compared to that of the original x_{near}. If the cost of x_{rewire} is lower, then rewiring occurs and the graph is updated.

REWIRE(...) and STEER(...) are identical for a classic RRT* implementation. There are, however, some platform-specific modifications to this, discussed later. After any potential rewiring occurs, EXTEND(...) returns the updated graph.

For Adversarial RRT*, the cost depends on the planner RNN (as discussed in detail later). Thus, rewiring not only affects the cost of the rewired node, but also future outputs of the planner RNN due to its recurrent nature. When rewiring, the costs down-branch must be recursively updated using the updated RNN hidden states. This is handled in the REPROPCOSTS(...) function in Algorithm 1. This is not necessary in RRT*.

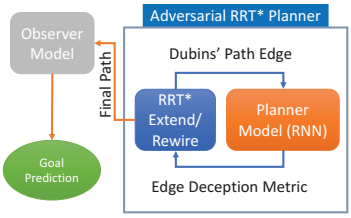


Fig. 4: The Adversarial RRT* planner architecture.

C. Platform Specific Modifications

In this work we use a Dubins vehicle model to test our planning methods. We assume the robot is only capable of traveling at constant forward velocity and with a non-zero minimum turning radius, $\rho_{min} > 0$ [25]. The Adversarial RRT* algorithm was tailored to the Dubins vehicle behavior by modifying the following cases.

In our Dubins' vehicle implementation, NEAREST(G,x) and NEAR(G,x) use Dubins' length for distance calculations.

In STEER(...), we define start state as $s_{from} = (x_i, y_i, \theta_i)$, and end state as $s_{to} = (x_f, y_f)$. Final heading θ_f is unspecified since only the sampled point in $\mathbb{R}^2$ is specified.

Finally, we draw a distinction between REWIRE(...) and STEER(...) for our Dubins' vehicle RRT* implementation. REWIRE(...) preserves the full state (x,y, and θ) of the "to" node, whereas STEER(...) ignores θ of the "to" node.

V. USING GOAL RECOGNITION IN AN ADVERSARIAL FRAMEWORK

We utilize two distinct neural networks in this work. The first is a neural network trained to classify the goal, which is used to calculate the deception-based cost function during planning. This network is referred to as the *planner* model, and is an RNN. The second is a network trained separately from a clean initialization which the planner attempts to fool. This model is only used for validation and is referred to as the *observer* model. These models were trained using mutually exclusive datasets of near-optimal paths. Figure 4 shows the architecture of the planner and the role of the two models.

A. Planner Model

The *planner* model is trained to perform a similar task to that of the *observer* model. The planner model is used during RRT* to predict deception in classification. A key contribution of this paper is a novel method of integrating this *planner* model into a sampling-based path planner. To accomplish this, we create a planner model (RNN) that predicts the vehicle's intended goal based on its trajectory. This is intended as an approximation of the *observer*, and is incorporated into the RRT* path planning process. This planner model computes the probability that each goal site is the intended target of the planner. These predictions follow the Softmax convention, where z contains the scores for each neural network target

class, z_i. These predictions are incorporated into the RRT* cost function.

$$softmax(z)_i = \frac{exp(z_i)}{\sum_j exp(z_j))} \tag{2}$$

B. Adversarial Cost Function for RRT*

1) Cost Formulation: We seek to create a motion planner that can balance between path cost and deception. We use J_{path} to refer to the cost associated with the path. This is often the total length of the path, as used in this work.

$$J_{path} = \int_{path} ds \tag{3}$$

The total cost, J, is a combination of the path cost and the deception score, $\mathcal{D}$, where $\alpha \in [0, 1]$ scales the weight of the deception score. Our planner attempts to minimize path cost while still accounting for deceptive properties.

$$J = J_{path} - \alpha\mathcal{D} \tag{4}$$

Existing literature provides several strategies for formulating optimization of deceptiveness [2]–[4]. Strategies include maximizing entropy and moving towards wrong goals.

In this work, we examine two deception scores: Shannon's Entropy [26], which represents ambiguity between goals; and a Goal Simulation [3] metric, which rewards high confidence in an incorrect goal. The Shannon's Entropy metric is:

$$D_{entropy}(X) = -\sum_{i=1}^{n} P(x_i) \log_n P(x_i) \tag{5}$$

Where X is the *planner's* goal prediction confidences, and $P(x_i)$ is confidence in the i-th goal location. Similar to [3], we formulate the Goal Simulation metric as:

$$D_{simulation}(X) = \max_{i \setminus i_{goal}} [P(x_i)] - P(x_{goal}) \tag{6}$$

The deception score can be computed for a path observation, s, using either metric, as:

$$\mathcal{D}(s) = \int_{path} (D(X|s))ds, \tag{7}$$

It should be noted that all integrals have been discretized into their equivalent summation in our implementation. We describe them as integrals for theoretical generality.

2) Cost Implementation: We implement our deception cost, $\mathcal{D}$, by leveraging adversarial machine learning. Specifically, we create a planner model that approximates the behavior of the observer. For this work, we use an RNN for the planner model. This allows the *planner* to query a prediction for a candidate path as it is planned, providing information about the value of the selected deception metric from a comparable observer. Our cost function implementation is shown in Eq. 8. The first term of the integral represents the path length. The second term, $\alpha D(X|s)$, represents the deceptiveness of the path input measured from the *planner* RNN output.

$$Cost(s) = \int_{path} (1 - \alpha D(X|s))ds, \quad \alpha \in [0, 1] \tag{8}$$

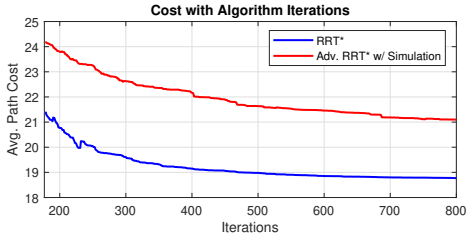


Fig. 5: Avg. cost over 100 paths with increasing iterations.

The α parameter is the deception weighting term. When $\alpha = 1$, the upper bound for both deception metrics of 1 will correspond to a path cost of 0. Conversely, $\alpha = 0$ will produce an identical result to RRT* with a path length cost.

It should be noted that our modifications to the cost function deviate from the original RRT* formulation [8] since the cost at the tail of the given path now depends on information from the prior path. Therefore, there is no guarantee Adversarial RRT* will converge on a path that is optimal with respect to our combined cost function, if such a feasible path exists. However, the aim of this work is not to optimize deception, but to provide dynamically feasible modifications to the near-optimal path in attempts to increase overall deceptiveness of the path. As such, RRT* provides a good planning framework.

VI. Simulated Results

A. Dataset Generation

In this work, the *observer* model uses a neural network (RNN) to predict the intended goal of the planner. The *planner* model aims to approximate an observer in order to deceive it. The *planner* also uses an RNN (but is trained on different data). We assume the application for the *observer* can be modeled by a simulation environment. The *observer* is assumed to be a naive adversary with access to its own planner (RRT*) to construct a training dataset. The *observer* is unaware that any deceptive actions are being taken against it. For training these neural networks, RRT* was used to produce two mutually exclusive training sets, each with 40,000 paths to 5 goals in a fixed map. In this work, we utilize 4 different fixed maps, shown in Figures 3 and 2a. The average path cost over 100 paths as iterations increase is shown in Figure 5.

We sample a distribution of points around the initial position to create a more diverse set of paths for training purposes. The initial position in each path was randomized using a normal distribution with 1-meter standard deviation from the nominal start location. Initial heading is randomized similarly, but with $\pm$45-degree perturbation from nominal initial heading. A similar procedure was used to generate validation data using RRT* and Adversarial RRT*. The Dubins vehicle was configured with 1-m/s velocity and 1-m minimum turning radius. Simulations were performed on an Intel Core i7-8700U 3.2GHz processor with 32GB RAM.

B. Observer and Planner Model Details

The *observer* model we consider in this work is an RNN based on the LSTM layer. This is also the architecture of

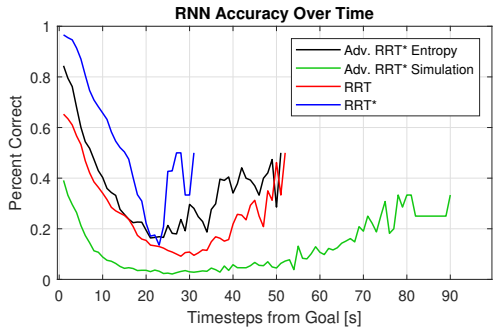


Fig. 6: Accuracy on 500 paths planned with each algorithm.

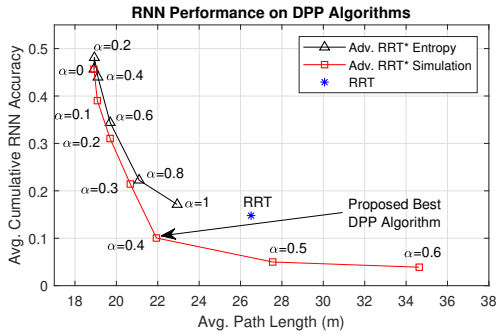


Fig. 7: Performance of each algorithm with varying α values.

the *planner* model. The *observer* model RNN architecture involves an LSTM layer with 1024 units and 4 fully-connected layers of decreasing dimension. These layers output into a Softmax layer. The benefit of the RNN is that it accepts variable input size, meaning it is well suited for sequential trajectory data. This architecture is similar to the goal prediction RNN in [21] that predicts goals from time-series observations. We include additional dense layers to provide more capacity and increased performance. The *planner* model is an RNN with architecture identical to that of the *observer* model.

C. Adversarial RRT* Performance Against RNN Observer

The Adversarial RRT* planner's performance is evaluated by comparing the paths it generates to the RRT* planner. Note that the observer was trained only using RRT* data. We assume that the observer is not aware of the adversarial attack.

Paths were generated for each planner using 250 iterations and a range of values of α for Adversarial RRT*. The initial conditions and sampling were seeded with the same set of random number generation seeds between the algorithms. A comparison of paths to each goal generated by RRT* (a) and Adversarial RRT* (b) is shown in Figure 2. The RRT* paths are near-optimal, whereas the Adversarial RRT* paths have increased length. The Adversarial RRT* paths maintain high ambiguity between the goals as long as possible.

We used the *observer* RNN to make predictions on RRT, RRT*, and Adversarial RRT* paths. Figure 1 shows a comparison of typical paths generated by each algorithm and what the RNN predicts over time. Note, as referenced in the accompanying video, that the Adversarial RRT* path differs from the RRT* path in that it travels in such a way as to increase the selected deception metric.

We use *observer* accuracy as a performance metric. A correct prediction is achieved when the highest confidence *observer* output corresponds to the correct goal location. Prediction accuracy over time is of interest to examine how quickly the observer typically converges on the correct goal. We also examine the average cumulative RNN accuracy, which is the average RNN accuracy across the entire path.

Fig. 6 shows the accuracy of the *observer* RNN on sets of paths from each algorithm over time. Note, the horizontal-axis is time in seconds *from* the goal, meaning as time from goal increases, the vehicle moves away from the goal, toward the start position. This is done to adjust for varying path lengths. The RRT* curve ends earliest since, moving backward from the goal, the start position is reached sooner due to a lower-cost path. This shows that both RRT and Adv. RRT* with Entropy ($\alpha = 1$) maintain low accuracy longer than RRT*, after which the RNN converges to higher accuracy. RRT slightly outperforms Adv. RRT* w/ Entropy. Adversarial RRT* w/ Goal Simulation ($\alpha = 0.6$) further decreases RNN accuracy and prevents convergence to high accuracy.

D. Comparison of Algorithm Performance

The adversarial cost functions allow the planner to select longer paths if they increase the chosen deception metric (entropy or goal simulation). This deviation from the optimal path allows for deceptiveness. Figure 7 shows performance curves for each Adversarial RRT* algorithm with a range of values of α. $\alpha = 0$ represents the RRT* algorithm. Observer RNN accuracy from RRT with the same Dubins' model is also shown. Each data point represents an average of results from 500 paths each from the three maps (1500 paths total). The vertical axis is average cumulative accuracy of the observer RNN on paths generated with that algorithm configuration. The horizontal axis is path length. As such, a high-performing DPP algorithm will be in the lower-left region of the graph.

Observer RNN performance on the RRT paths is degraded due to the highly random nature of paths generated. However, path length of RRT paths is 40% higher than optimal.

In contrast, Adv. RRT* with Entropy, while less deceptive in this regard, is still able to approach RNN cumulative accuracy of about 17% as α approaches 1, but at a much lower path length than RRT. Adv. RRT* with Goal Simulation outperforms in both deceptiveness and optimality. It is able to provide better deception (10% RNN accuracy) than the RRT paths, while keeping path length within 16% of optimal.

E. Pirate Deception Scenario Performance

We simulate a use case of Adversarial RRT* in which an agent must deliver an *asset* to a predetermined goal location (chosen from a set of candidates) before a *pirate* adversary can identify and reach the location to hijack or prevent the delivery. In this scenario, both the *asset* and the *pirate* are identical Dubins' vehicle models (1 m/s speed and 1-m turn radius). The *pirate* starts roughly in the centroid of the goals.

The *pirate* uses 1-Hz surveillance from the RNN to identify the most likely goal based on the *asset's* trajectory. The pirate plans a path using RRT* (150 iterations) to reach this goal from its current pose and executes this path until the predicted goal changes or the *asset* completes its path. The *asset* will use DPP to plan a deceptive path to delay the *pirate's* arrival.

Examples of this scenario are shown in Fig. 1. The left shows the *pirate* (black) reaction to an *asset* path planned with RRT* (blue). Here, the *pirate* reaches the goal before the *asset*. The portion of the *asset* path that occurs before the *pirate* reaches the goal is shown as solid. Afterwards, the path is dashed, indicating the *asset* failed to complete the path in time. The right shows the *pirate* reaction to an *asset* path planned with Adv. RRT* with Simulation ($\alpha = 0.4$). Here, the *asset* reaches the goal before the *pirate*, as indicated by the truncated pirate path. The *asset* employing Adv. RRT* delays the *pirate* by making it think it is targeting other goals. An animation of the pirate example is in the accompanying video.

We simulated 500 paths each of RRT*, and Adv. RRT* with each metric using the map in Fig. 2a. We found that 52.8% of RRT* paths reach the goal before the *pirate*, compared to 75.4% of paths from Adv. RRT* with Simulation ($\alpha = 0.4$) and 74% with Entropy ($\alpha = 1$). This example application shows how deceptiveness can improve performance.

F. Larger Drone Delivery Scenario

To illustrate the broader applicability of this work, we examined a similar navigational goal prediction scenario but with a more relevant vehicle and map. Specifically, we examined a drone delivery example which features a speed of 90m/s and a turning radius of 300m. The map, shown in Figure 3c, was 10-km by 10-km. RNN training and experimental protocols were similar to the other experiments. We achieved similar deception results. Average cumulative *observer* accuracy on RRT* paths in this experiment was 47%. Accuracy on Adv. RRT* paths was lower, at 19% with Simulation ($\alpha = 0.4$) and 27% with Entropy ($\alpha = 1$). Adv. RRT* path lengths increased from optimal by 29% for Simulation and 22% for Entropy.

VII. Conclusion

We have proposed a novel application of adversarial machine learning attacks by integrating two different deception cost functions into a sampling-based planner. We demonstrate this adversarial cost function in an RRT* planner for a Dubins' vehicle. We evaluated Adversarial RRT* against an observer model RNN. The results illustrate how Adversarial RRT* can reduce classification accuracy and delay the onset of high observer confidence. We also provide a simulated piracy case-study. In this example, Adversarial RRT* can improve the security of a high-value delivery. We envision that this planner could be utilized by a range of deceptive mobile systems.

References

[1] A. Dragan, R. Holladay, and S. Srinivasa, "An Analysis of Deceptive Robot Motion," in *Robotics: Science and Systems X*. Robotics: Science and Systems Foundation, Jul. 2014.

[2] ——, "Deceptive robot motion: synthesis, analysis and experiments," *Autonomous Robots*, vol. 39, no. 3, pp. 331–345, Oct. 2015.

[3] P. Masters and S. Sardina, "Deceptive Path-Planning," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*. Melbourne, Australia: International Joint Conferences on Artificial Intelligence Organization, Aug. 2017, pp. 4368–4375.

[4] Z. Liu, Y. Yang, T. Miller, and P. Masters, "Deceptive Reinforcement Learning for Privacy-Preserving Planning," *arXiv:2102.03022 [cs]*, Feb. 2021, arXiv: 2102.03022.

[5] M. Maynard, T. Duhamel, and F. Kabanza, "Cost-based goal recognition meets deep learning," 2019.

[6] M. Tambe, "Security and Game Theory: Algorithms, Deployed Systems, Lessons Learned." Cambridge: Cambridge University Press, 2011.

[7] S. M. Lavalle, "Rapidly-Exploring Random Trees: A New Tool for Path Planning," Tech. Rep., 1998.

[8] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, Jun. 2011, publisher: SAGE Publications Ltd STM.

[9] A. D. Dragan, K. C. Lee, and S. S. Srinivasa, "Legibility and predictability of robot motion," in *2013 8th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. IEEE, 2013, pp. 301–308.

[10] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[11] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2574–2582.

[12] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. D. Tygar, "Adversarial machine learning," in *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, 2011, pp. 43–58.

[13] P. Masters and S. Sardina, "Cost-based goal recognition in navigational domains," *Journal of Artificial Intelligence Research*, vol. 64, pp. 197–242, 2019.

[14] D. Pattison and D. Long, "Accurately determining intermediate and terminal plan states using bayesian goal recognition," D. Pattison, D. Long, and C. Geib, Eds. ICAPS, Jun. 2011, pp. 32–37.

[15] M. Vered and G. A. Kaminka, "Heuristic Online Goal Recognition in Continuous Domains," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*. Melbourne, Australia: International Joint Conferences on Artificial Intelligence Organization, Aug. 2017, pp. 4447–4454.

[16] B. Busch, J. Grizou, M. Lopes, and F. Stulp, "Learning legible motion from human–robot interactions," *International Journal of Social Robotics*, vol. 9, no. 5, pp. 765–779, 2017.

[17] L. Amado, R. F. Pereira, J. Aires, M. Magnaguagno, R. Granada, and F. Meneguzzi, "Goal Recognition in Latent Space," in *2018 International Joint Conference on Neural Networks (IJCNN)*. Rio de Janeiro: IEEE, Jul. 2018, pp. 1–8.

[18] L. Rosa Amado, R. Fraga Pereira, and F. Meneguzzi, "Combining LSTMs and Symbolic Approaches for Robust Plan Recognition," in *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, May 2021, pp. 1634–1636.

[19] T. Fernando, S. Denman, S. Sridharan, and C. Fookes, "Soft + Hardwired attention: An LSTM framework for human trajectory prediction and abnormal event detection," *Neural Networks*, vol. 108, pp. 466–478, Dec. 2018.

[20] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*, ser. Adaptive computation and machine learning. Cambridge, Massachusetts: The MIT Press, 2016.

[21] L. Amado, J. P. Aires, R. F. Pereira, M. C. Magnaguagno, R. Granada, and F. Meneguzzi, "LSTM-based goal recognition in latent space."

[22] S. Karaman and E. Frazzoli, "Optimal kinodynamic motion planning using incremental sampling-based methods," in *49th IEEE Conference on Decision and Control (CDC)*, Dec. 2010, pp. 7681–7687.

[23] J. Su, D. V. Vargas, and K. Sakurai, "One pixel attack for fooling deep neural networks," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 5, pp. 828–841, 2019.

[24] P. Masters, "Goal recognition and deception in path-planning," *Ph. D. dissertation, RMIT University*, 2019.

[25] 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.

[26] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.