

**LEVERAGING ADVERSARIAL MACHINE LEARNING TECHNIQUES FOR
DECEPTIVE SAMPLING-BASED MOTION PLANNING**

A Dissertation
Presented to
The Academic Faculty

By

Hayden Nichols

In Partial Fulfillment
of the Requirements for the Degree
Masters of Science in the
College of Engineering
Department of Mechanical Engineering

Georgia Institute of Technology

May 2022

© Hayden Nichols 2022

**LEVERAGING ADVERSARIAL MACHINE LEARNING TECHNIQUES FOR
DECEPTIVE SAMPLING-BASED MOTION PLANNING**

Thesis committee:

Dr. Anirban Mazumdar
Mechanical Engineering
Georgia Institute of Technology

Dr. Jonathan Rogers
Aerospace Engineering
Georgia Institute of Technology

Dr. Jun Ueda
Mechanical Engineering
Georgia Institute of Technology

Date approved: December 26, 2021

ACKNOWLEDGMENTS

Thanks to my advisor, Dr. Anirban Mazumdar for mentoring me through my graduate experience and helping to guide my research.

Also, I want to express gratitude to my colleagues in the DART Laboratory for providing support and companionship during my time here. Special thanks to Zachary Goddard and Sam Deal, who both provided invaluable insights and helped me work through many inevitable bugs.

Thanks to my family who have supported my endeavors and have encouraged me along the way. Their outspoken confidence in me helped me to persevere even when I had doubts. Lastly, thank you to my fiancée, Emily. She has given me endless support and sanity checks, celebrated my successes, motivated me through setbacks, and (most admirably) tolerated my complaining along the way.

This work was supported by the Sandia National Laboratories Laboratory Directed Research and Development (LDRD) Program. Sandia National Laboratories is a multi-mission laboratory managed and operated by 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 DENA0003525. 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.

TABLE OF CONTENTS

Acknowledgments	iii
List of Figures	vi
Summary	vii
Chapter 1: Introduction and Background	1
1.1 Introduction	1
1.2 Background	2
1.2.1 Sampling-Based Motion Planning	3
1.2.2 Goal Recognition	3
1.2.3 Deep Learning	4
1.2.4 Adversarial Machine Learning	4
1.2.5 Functional Requirements	5
1.2.6 Contributions	5
Chapter 2: Mathematical Formulation and Algorithmic Approach	7
2.1 Mathematical Formulation	7
2.1.1 Planner Formulation	7
2.1.2 Observer Formulation	8

2.1.3	Navigational Goal Recognition for Path Planning	8
2.2	Adversarial RRT* Algorithm	9
2.2.1	RRT* Utilities and Data Structures	10
2.2.2	RRT* Algorithm	11
2.2.3	Platform Specific Modifications	14
2.3	Using Goal Recognition in an Adversarial Framework	16
2.3.1	Planner Observation Model	16
2.3.2	Adversarial Cost Function	17
Chapter 3: Simulated Results		21
3.1	Overview of Simulated Experiments	21
3.1.1	Simulation Environments and Experiments	21
3.1.2	Dataset Generation	22
3.1.3	Observer and Planner Model Configuration	25
3.2	Results of Simulated Experiments	25
3.2.1	Adversarial RRT* Performance Against RNN Observer	25
3.2.2	Comparison of Algorithm Performance	28
3.2.3	Pirate Deception Scenario Performance	29
3.2.4	Larger Drone Delivery Scenario	30
Chapter 4: Conclusion and Future Works		32
References		33

LIST OF FIGURES

2.1	The Adversarial RRT* planner architecture.	17
2.2	An illustration of the two deception scores in this work.	19
3.1	Illustration of the “Pirate Deception Scenario”. Red circles are obstacles. The 5 colored circles are possible goals. Below each is the observer confi- dence in each goal over time.	23
3.2	Additional environments simulated in this work.	24
3.3	Avg. cost over 100 paths with increasing iterations.	25
3.4	Paths using 250 iterations of each algorithm.	26
3.5	Accuracy on 500 paths planned with each algorithm.	27
3.6	Performance of each algorithm with varying α values.	28

SUMMARY

There are many applications in which a mobile agent wants to avoid having its intent known to an observer. Additionally, a mobile agent may want to have deceptive actions that convey an intent other than its true objective. Examples of this include preserving privacy in a high-surveillance environment or confusing an opponent in an adversarial setting. However, this desire for deception can conflict with the need for an efficient path. Optimal plans such as those produced by RRT* may have low path cost. However, optimality can lead to predictability, in that observers can often predict the intent of an optimal agent. Similarly, a deceptive path that moves in such a way to confuse the observer may take too long to reach the goal. This work attempts address this trade-off by drawing inspiration from adversarial machine learning. Presented in this thesis is a novel planning algorithm, dubbed 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. Performance of Adversarial RRT* is demonstrated with two measures of deception, using a simulated Dubins vehicle. Adversarial RRT* can decrease RNN accuracy across paths to 10%, compared to 46% accuracy on near-optimal RRT* paths, while keeping path length within 16% of optimal. This work also presents two simulated use cases of Adversarial RRT* planner. In one use-case, the planner attempts to safely deliver a high value asset to a predetermined location while an adversary observes the vehicle's path and tries to interrupt the delivery. In the other use-case, fixed-wing delivery drone dynamics are approximated using a Dubins' vehicle model and Adversarial RRT* is demonstrated as a method to preserve the privacy of the recipient of an aerial delivery.

CHAPTER 1

INTRODUCTION AND BACKGROUND

This work presents a novel algorithm for deceptive path planning that leverages sampling-based motion planning and adversarial machine learning. This chapter introduces the aims of this work and relevant background information.

1.1 Introduction

Deceptive path planning (DPP) aims to minimize the probability of an external observer identifying the planner's intended destination. DPP has relevance in applications in adversarial environments and for preserving privacy. Some examples of relevant DPP applications are maintaining privacy when traversing an environment with heavy surveillance or confusing an opponent in a game. This work includes 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 are increasingly applied to difficult problems and predicting agents' intent based on observations is among these. Deep learning has been shown effective for trajectories forecasting and navigational goal recognition from time-series data [5]. Since they provide computational speed and autonomy, these methods may provide improvements over the performance of human experts in the future. Furthermore, recent research has shown that deep learning algorithms outperform domain-specific algorithms for navigational goal recognition in terms of prediction accuracy. [5]. This serves to illustrate the potential offered by deep learning techniques and motivate the likelihood that they will be used for goal recognition in the future. As such, there exists a need to investigate how deceptive measures can be taken against neural-network based observers.

Randomness has proven to often be an effective policy for deception [6]. Naturally, a randomized path planner may have some inherently deceptive characteristics. A highly-randomized path planner, such as Rapidly-Exploring Random Trees (RRT) [7], may provide paths that can be unpredictable, but the path cost (length, time, energy, etc.) is sub-optimal [8]. This means that an agent executing an RRT path might be deceptive, but excessive costs incurred by sub-optimality may be disadvantageous. Furthermore, RRT takes no consideration of deception performance, and could occasionally produce plans that are not deceptive. As such, the randomly generated path could be predictable.

Optimized sampling-based algorithms such as RRT* [8] obtain optimality by minimization of a given cost function. While this is advantageous for achieving objectives or reaching goals quickly or efficiently, optimality of actions often indicates the intent of the actor [9], which would degrade deceptiveness.

To address this trade-off, this thesis seeks to develop a deceptive path planning algorithm inspired by adversarial machine learning concepts [10, 11, 12]. Specifically, the adversary observer is assumed to be using a neural-network to classify the intended goal of an agent based on trajectory. The methods presented here consider actions to deceive this observer while still minimizing path cost. This is achieved by integrating a deception score term into the RRT* optimization function using a Recurrent Neural Network (RNN) that approximates the observer. This produces low-cost paths that have deceptive properties. The resulting algorithm is dubbed “Adversarial RRT*.”

1.2 Background

This thesis examines the scenario of traversal to one of a set of goals in an adversarial setting [3, 13]. One agent, the *planner*, must reach the predetermined goal in a fixed environment, while avoiding obstacles. Meanwhile the adversary agent, the *observer*, must determine the *planner's* intended goal as quickly as possible. This section reviews the relevant literature pertaining to DPP, motion planning, goal recognition, deep learning and

adversarial machine learning. Each section highlights potential gaps this work seeks to address.

1.2.1 Sampling-Based Motion Planning

Sampling-based path planners have become widely popular path planning methods because they offer computational simplicity in difficult, high-dimensional path planning problems. Some popular categories of sampling-based motion planners include RRT and the optimized RRT* [7, 8]. These algorithms have demonstrated applications for many systems and have probabilistic completeness guarantees under general conditions [7]. The most basic RRT algorithm is implemented with line-segment edges, but it can be adapted to kinodynamic systems, for instance, by using Dubins' paths [14]. Therefore, sampling based planners are a good starting point for an adversarial planner.

1.2.2 Goal Recognition

Goal recognition consists of predicting the unobserved goal of an agent when given a sequence of its observed states [15, 16]. This thesis focuses on navigational goal recognition, or predicting the desired goal of a mobile agent. The deceptive path planning algorithm proposed in this thesis seeks to disrupt a neural-network based navigational goal recognition observer.

Predictability and legibility of robot manipulator motion has been discussed and characterized in existing work [9, 2, 17]. These works focus on the development and characterization of deception paradigms effective in deceiving human observers. 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. This is notable since optimality is closely associated with legibility [9, 17], in that a path which is efficient from origin to goal more clearly conveys the agent's target.

Related work examines leveraging goal prediction paradigms to deceptive path plan-

ning [18, 4]. One goal prediction strategy that has been leveraged for deceptiveness is cost-based goal recognition [18, 13]. Cost-based goal recognition uses domain observations and cost-to-go predictions to determine if an agent is behaving rationally and to predict which goal is best for that agent. This paradigm is inverted to plan deceptive paths.

Metrics similar to those in [2] are leveraged as reward functions for Q-learning in a reinforcement learning framework in [4]. The objective in this work was to create deceptive policies for privacy-preserving path planning. The work in thesis is differentiated from the reinforcement-learning approach through the applicability to kinodynamic systems, as well as leveraging adversarial machine learning.

1.2.3 Deep Learning

Deep Learning, especially deep neural networks, have been used to automatically encode planning domain knowledge from observations in goal recognition problems [19, 20]. Deep learning has shown promise in predicting object trajectories and navigation goals given sequential position measurements [21, 5]. Recurrent Neural Networks (RNN), are particularly suited to handle sequential inputs [22, 23]. Therefore, deep learning techniques are promising candidates for navigational goal recognition applications.

1.2.4 Adversarial Machine Learning

Machine learning has shown promising results for classifying mobile vehicle trajectories [19, 24]. While promising for goal recognition, machine learning algorithms are vulnerable to adversarial attacks [10]. These adversarial attacks generate inputs that can degrade the target neural network's accuracy or confidence in prediction [12]. Analysis has shown that non-random perturbations of inputs are a method of modifying the output of a neural network to a chosen output class. Furthermore, image classification algorithms can be deceived by injecting small changes to select input pixels [10]. These injections can be addition of noise to an existing image [11] or alteration of a single, strategically selected

input channel [25].

However, for generating adversarial example equivalents for robotic paths, these strategies do not transfer well. The strategy of perturbing points on a precomputed feasible path is made difficult by the requirements to obey the robot's dynamic constraints, efficiently produce a plan, and avoid collisions with environmental obstacles.

While adversarial machine learning examples and techniques may leverage knowledge of the target network's architecture or parameters, they can also generalize well, meaning they can fool unknown networks as well [10]. This means that a DPP algorithm based on a particular adversarial technique or exploiting a particular network architecture may fool many types of network architectures. This thesis incorporates adversarial machine learning into an optimized sampling-based planner to deceive navigational goal recognition algorithms.

1.2.5 Functional Requirements

There is value in a deceptive path planning algorithm that can deceive a neural-network-based navigational goal recognition algorithm. It is additionally valuable for this method to extend to dynamical systems in that the deceptive considerations do not violate dynamic constraints or collide with obstacles. Furthermore, this algorithm should be able to balance between deceptiveness and path cost. As such the functional requirements for the DPP algorithm explored in this work are that the algorithm:

1. is able to fool neural network goal recognition algorithms;
2. produces dynamically feasible paths that avoid collisions; and
3. can balance between deceptiveness and path cost/optimalty.

1.2.6 Contributions

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 sampling-based 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.

CHAPTER 2

MATHEMATICAL FORMULATION AND ALGORITHMIC APPROACH

This chapter provides the formulation of the DPP problem and the algorithmic approach to addressing it. First, a mathematical formulation of the DPP problem is defined. Next, the Adversarial RRT* Algorithm is outlined, with particular focus on how it differs from RRT* and platform specific modifications made to adapt the algorithm to a Dubins' vehicle.

2.1 Mathematical Formulation

This section provides the formulation of the DPP problem that this work seeks to address. The formulation of the *planner*, *observer*, and the *planner* goal recognition model are given.

2.1.1 Planner Formulation

The *Planner* is one of two competing agents in the proposed deceptive path planning scenario. In this thesis, the *planner* is a dynamically constrained mobile ground robot, whose state is given by $s_t = (x_t, y_t, \theta_t)$. The *planner* will traverse to a goal in one of several goal regions, $g \in G_i \in \mathcal{G} = \{G_1, G_2, \dots, G_n\}$, in a 3-dimensional environment, $\mathcal{X} \subset \mathbb{R}^3$. The environment has obstacles, $\mathcal{O} \subset \mathcal{X}$, and goal sets \mathcal{G} where \mathcal{G} and \mathcal{O} are disjoint.

A path is defined to be 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 discretization of a path is described as $S = (s_0, \dots, s_f)$ 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: $\text{Feasible}(\mathcal{P}) \iff \forall (x_i, y_i, \theta_i) \in \mathcal{P}, (x_i, y_i, \theta_i) \notin \mathcal{O}$. For the purposes of this work, $s_g = (x_f, y_f, \theta_f), \theta_f \in \mathbb{R}$ such that the goal location is independent of orientation. An example of the environment is shown in Figure 3.2. Some viable paths through that

environment are shown in Figure 3.4a.

The *planner* produces one viable path, \mathcal{P}_{gt} , to the selected goal. 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.

2.1.2 Observer Formulation

While the *planner* attempts to “hide the real” by preventing its actions from showing its intended goal (as described in [2]), its adversary the *observer* performs navigational goal recognition [5]. The observer uses observation of the *planner* vehicle trajectory to predict its intended goal. At each time period, the *observer* generates a probability for each goal candidate in the environment, represented as a multinoulli distribution[22], parameterized by the vector $\mathbf{c} \in [0, 1]^n$ over the set of n goal sets, $G_i \in \mathcal{G}$, where

$$\mathbf{c}_i(s_t) = P(G_i|s_t). \quad (2.1)$$

The *planner*’s objective is to produce a less legible path (a legible path conveys the planner’s intended goal, described by Eq. (9) of [9]), but must also consider the optimality of the planned path. The *planner* treats deception as an inversion of probabilistic goal recognition as in [26], meaning 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.

2.1.3 Navigational Goal Recognition for Path Planning

It is necessary to approximate the goal recognition behavior of the *observer* for the proposed method of deceptive path planning. This work assumes the *planner* has access to some function f^* which approximates the true distribution $P(G_i|S)$, modeling the observer’s expectation of the likelihood of each goal being selected. The *planner* incorporates a neural network which has learned f^* from observing near-optimal paths, which can be

leveraged to plan a path which is both efficient and deceives the observer. For this work, an RNN is used as our approximation for the observer. This is called the “planner observation model.”

It was assumed that f was learned by observing near-optimal paths, and as such, f^* would approximate f by also learning from near-optimal paths. In general, an observer need not assume it is observing an optimal agent. However, creating an observer that is able to quantify agent optimality or rationality poses its own challenges. Generating datasets to train such a complex observer would require a diverse set of planning algorithm(s) capable of generating the behaviors of such agents. Moreover, the quality of this observer would be linked to the diversity of deceptive paradigms modelled in the dataset. Furthermore, the development of an observer that can accurately predict the intent of *both* optimal and deceptive agents is a field with ongoing research [26, 18]. This work chooses to build a naive observer that does not anticipate deceptive actions being taken against it. Future investigation should examine if Adversarial RRT* could generalize to fooling an observer that expects deceptive behavior, though it is likely that this is dependent on the ability to model f^* similarly to the underlying assumptions made by the observer.

2.2 Adversarial RRT* Algorithm

This work makes modifications to an RRT* framework to provide deception in path planning. This is accomplished by adding a deception metric into the RRT* cost function. RRT* was selected as a base algorithm because it provides rapid, optimized motion planning. Deceptive considerations are added to the planning process through approximating the targeted observer’s output using the planner observation model RNN. During the planning process, path observations of potential path segments are passed to this RNN, which outputs predicted likelihood that each goal is the selected goal given the observations. This prediction is used to compute the deception metric for the path segment.

The following subsections describe the RRT* algorithm developed for this task, mod-

ifications to include the adversarial cost function, and platform-specific modifications for the Dubins' vehicle model.

2.2.1 RRT* Utilities and Data Structures

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

For this work, a node for the RRT* algorithm is taken to be $x = (s, \sigma, x_p, X_c, c)$, where s is the vehicle state at the node, 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.

For Adversarial RRT*, the node datastructure is augmented with several new entries. The planner observation RNN hidden state at the node is stored as h_{RNN} . This allows the RNN to be re-initialized at that node for future predictions. References to children nodes, $x_c \in X_c$, are stored to facilitate cost re-propagation. The Adversarial RRT* 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 RRT* with a Dubins' vehicle, cost is Dubins' path length. The Adv. RRT* cost function is described in subsection 2.3.2. For Adversarial RRT*, the planner observation model is called and the node is populated with the corresponding hidden RNN state 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).

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

$\text{STEER}(\dots)$: creates a path between two nodes according to a steering procedure. It also populates the data structure for the output node.

$\text{REPROPCOSTS}(\dots)$ function is required for the adversarial cost function, which utilizes the planner observation model (RNN) output. When rewiring occurs in the RRT* framework, it not only modifies the cost of that rewired node, but also of the down-branch portion of the tree as well. Future predictions from the planner observation model (RNN) depend upon the hidden state, which is modified by rewiring. Thus, the cost and RNN hidden states of down-branch nodes must be recursively re-propagated, starting with the rewired node and iterating through its children.

Algorithm 1 Adversarial RRT* Recursive Cost Repropagation

```

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

```

$\text{BESTPATH}(\dots)$ returns the lowest-cost path by searching V_g for the goal node that reaches the goal with the lowest cost. From this node, the function steps back through the branch, adding subsequent parent nodes to the path, \mathcal{P} , until the start node is reached.

2.2.2 RRT* Algorithm

This section details the RRT* algorithm, which is used as a starting point for development of the Adversarial RRT* algorithm. Later sections will provide the details of these modifications.

Algorithm 2 RRT* Best Path

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

Main Function:

The RRT* algorithm [8], shown in Algorithm Algorithm 3, consists of initialization, iteratively extending the graph and, lastly, constructing the best path from the graph. To initialize the algorithm, the first node is created from the given start position, which is used to initialize the vertex set, V . The edge set, E , and goal nodes set, V_g , are initialized as empty. Together, these three sets form the graph, G . Also during initialization, obstacle locations and the goal region are given.

The Extend procedure is a single iteration of the process by which the graph is extended and the environment is explored. To start the Extend procedure, a node is sampled at random. This node, in addition to G , and the goal region are passed as arguments to the EXTEND(...) function. This function extends the graph, and returns the updated sets of nodes, V' , edges, E' , and goal nodes V'_g . Extension of the graph occurs for a predetermined number of iterations (denoted by N), then the BESTPATH(...) function is called to return the lowest cost path in the graph.

Extend Procedure:

EXTEND(...), in Algorithm Algorithm 4, takes a randomly sampled node, x , attempts to make a connection to the graph, G , and checks if the designated goal region, X_{goal} is

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:      $x_{rand} \leftarrow \text{SAMPLE}(i);$ 
6:      $(V', E', V'_g) \leftarrow \text{EXTEND}(G, x_{rand}, X_g);$ 
7:      $G \leftarrow (V', E', V'_g)$ 
8:   end for
9:   return BESTPATH( $G$ )
10: end function
```

reachable via the newly added node. It also check for collisions with obstacles.

EXTEND(...) begins by finding the nearest node in G using NEAREST(...). The STEER(...) function creates a path from the nearest neighbor to the sampled node and returns a node x_{new} at the sampled node. The path from the steer procedure is checked for collisions using the OBSTACLEFREE(...) function. This obstacle check is done to guarantee the node has at least one viable parent before trying to find the best parent.

If OBSTACLEFREE(...) finds that x_{new} is reached without collision, the set neighbors, X_{near} , within a specified radius of the new node are checked to see if the cost to reach x_{new} can be reduced via another parent. This “best-parent search” is outlined in lines 9-15 of Algorithm Algorithm 4. In this process, STEER(...) is used to create a path from each node in X_{near} to x_{new} . The lowest-cost obstacle-free path and its corresponding parent node is selected, which corresponds to the parent of x_{min} in the algorithm.

After the best parent for the new node is found, an attempt is made to connect x_{min} to the goal (lines 18-21 of Algorithm Algorithm 4). This determines if the goal set is reachable from x_{min} , which is important for two reasons. First, this allows the algorithm to keep track of possible paths to the goal for constructing the final solution. This is important because RRT* is only probabilistically complete. As such, a path is only guaranteed to be randomly sampled as the number iterations gets very large. Forcing the goal-set reachability check allows the algorithm to be run for fewer iterations while still finding a goal in most cases.

Second, this makes the RRT* more adaptable to applications in which constraints such as fuel or battery consumption or max range are added to the vehicle model. This step allows a sampled node to be discarded if the goal set is not reachable due to such constraints. After this check, the graph is updated with the new information.

Rewire Procedure:

The rewire procedure occurs in lines 22-33. Rewiring iterates through each node $x_{near} \in X_{near}$ and determines if the cost to reach x_{near} can be improved through x_{min} . This is done by creating a path from x_{min} to x_{near} using REWIRE(...). The cost of the rewired node, x_{rewire} is compared to the cost of x_{near} . The rewire occurs and the graph is updated if x_{rewire} achieves lower cost than x_{near} .

In a traditional RRT* implementation, REWIRE(...) and STEER(...) are interchangeable. This work specifies some platform-specific modifications to the Rewire function, discussed later, which is why the distinction is made. After the rewire process occurs, EXTEND(...) returns the updated graph.

In the specific case of Adversarial RRT*, cost is dependent on the planner observation RNN output (as discussed in detail later). Due to the recurrent nature of the RNN, a rewire not only changes the RNN output and costs of the rewired node, but also down-branch outputs of the planner observation RNN. As such, after rewiring, down-branch costs and states are recursively updated. This occurs in the REPROPCOSTS(...) function in Algorithm Algorithm 1. This is not necessary in RRT*.

2.2.3 Platform Specific Modifications

The work in this thesis relies on a Dubins' vehicle model to test the effectiveness of the discussed planning methods. The mobile platform used is assumed to travel at constant forward velocity and be constrained by a non-zero minimum turning radius, $\rho_{min} > 0$ [27]. RRT* and Adversarial RRT* algorithms were tailored to the Dubins' vehicle model by

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:    $x_{nearest} \leftarrow \text{NEAREST}(G, x)$ 
5:    $x_{new} \leftarrow \text{STEER}(x_{nearest}, x);$ 
6:   if OBSTACLEFREE( $x_{new}$ ) then
7:      $x_{min} \leftarrow x_{new};$ 
8:      $X_{near} \leftarrow \text{NEAR}(G, x);$ 
9:     for all  $x_{near} \in X_{near} / x_{nearest}$  do
10:       $x_{new, near} \leftarrow \text{STEER}(x_{near}, x);$ 
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:       $x_{min}.\text{parent.children}.\text{APPEND}(x_{min})$ 
17:       $V' \leftarrow V' \cup x_{min}; E' \leftarrow E' \cup \text{EDGE}(x_{min})$ 
18:       $x_{goal} \leftarrow \text{STEER}(x_{min}, X_g)$ 
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:         $x_{rewire} \leftarrow \text{REWIRE}(x_{min}, x_{near})$ 
24:        if OBSTACLEFREE( $x_{rewire}$ ) and...
25:          ...COST( $x_{rewire}$ ) < COST( $x_{near}$ ) then
26:             $x_{near}.\text{parent} \leftarrow x_{rewire}.\text{parent};$ 
27:             $x_{near}.\sigma \leftarrow x_{rewire}.\sigma;$ 
28:             $E' \leftarrow E' / \text{EDGE}(x_{near});$ 
29:             $E' \leftarrow E' \cup \text{EDGE}(x_{rewire});$ 
30:             $G' \leftarrow \text{REPROPCOSTS}(G', x_{near});$ 
31:          end if
32:        end for
33:      end if
34:    return ( $V', E', V'_g$ )
35:  end function
```

modifying the following cases.

In the Dubins’ vehicle specific implementation in this work, $\text{NEAREST}(G,x)$ and $\text{NEAR}(G,x)$ use distance in Dubins’ path space for distance calculations.

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

Finally, a distinction is made between $\text{REWIRE}(\dots)$ and $\text{STEER}(\dots)$ for the Dubins’ vehicle RRT* implementation. $\text{REWIRE}(\dots)$ produces a path that solves a full-state (x, y, θ) boundary value problem, taking into account the full state of the ”to” node, whereas $\text{STEER}(\dots)$ ignores θ portion of the ”to” node state.

2.3 Using Goal Recognition in an Adversarial Framework

This thesis uses an approximation of a goal recognition neural network to provide measures of deception in the planning process. Two separate, distinct neural networks are used in this work. One neural network, the *observer* model, is the target of the deceptive path planning. The planner is attemptin to fool this network. The other neural network discussed is the *planner observation* model. This is designed as an RNN trained (on separate data, from a clean initialization) to classify the goal similarly to the *observer* model. This will provide an expected measure of deception which can be used to compute the deceptive cost function. Both neural networks learn using mutually exclusive datasets of near-optimal paths. Figure Figure 2.1 shows the architecture of the planner and how each neural network factors into the work in this thesis.

2.3.1 Planner Observation Model

The *planner observation* is intended to model the function of the *observer* model. As such, it is leveraged during Adversarial RRT* to provide a quantification of how a given path may deceive the target *observer* model. This thesis describes a novel approach to incorporating the *planner observation* model into a sampling-based kinodynamic path planning

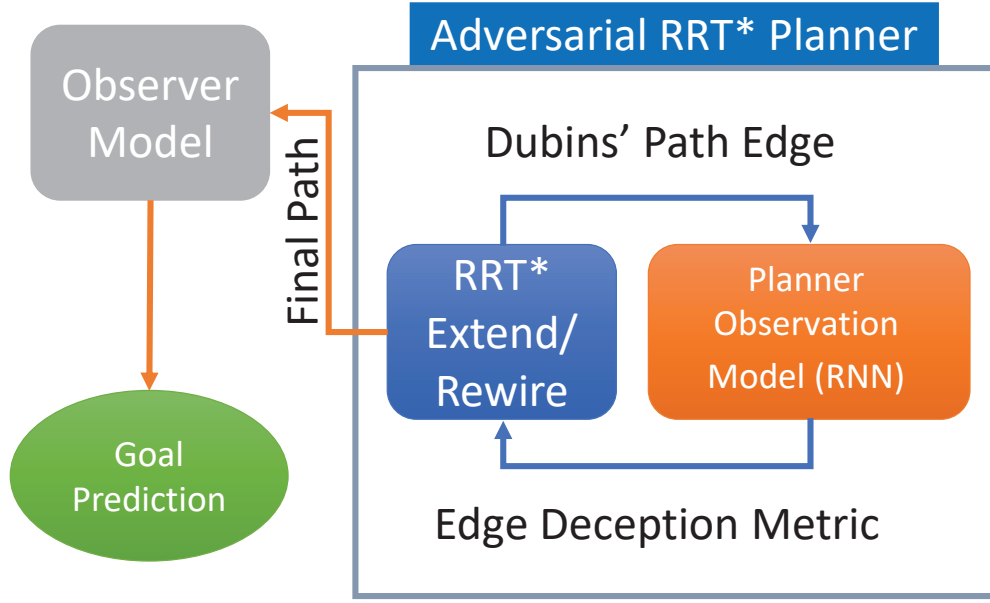


Figure 2.1: The Adversarial RRT* planner architecture.

algorithm.

This is accomplished by developing a *planner observation* model RNN that can be queried with a path input to provide a prediction of the vehicle’s intended goal. This will serve to approximate the *observer* RNN output, during path planning.

The *planner observation* model computes the likelihood that each goal region has been targeted by the *planner* agent.

The output is obtained using the Softmax convention in Equation 2.2, where z contains the scores for each neural network target class, z_i . The resulting predictions are leveraged to compute the deception score in Adversarial RRT*.

$$\text{softmax}(z)_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)} \quad (2.2)$$

2.3.2 Adversarial Cost Function

An objective of this work is to create a motion planner that can balance optimizing path cost with deceptiveness. As such, an optimization cost function has been developed to reflect

this functional requirement. J_{path} captures to the traditional path cost associated with the path. Typically this will be time or path length. This work selects path length as J_{path} , shown in Equation 2.3.

$$J_{path} = \int_{path} ds \quad (2.3)$$

The total cost in the adversarial cost function, J , combines path cost, J_{path} , and deception score, \mathcal{D} , where $\alpha \in [0, 1]$ scales the weight of the deception score. This is shown in Equation 2.4 The Adversarial RRT* planner employs this cost function to minimize path cost while still accounting for deceptive properties.

$$J = J_{path} - \alpha \mathcal{D} \quad (2.4)$$

Previous existing works give formulations for strategies to acheive deceptiveness [2, 3, 4]. Some strategies shown to be successful and applicable to this work include dissimulation (or maximizing entropy) and simulation of other potential objectives (or moving towards wrong goals).

Adversarial Cost Formulation

For the formulation of the cost function in this thesis, two deception scores are examined: Shannon’s Entropy [28], which captures observer ambiguity between candidate goals; and a Goal Simulation[3] metric, which tends to encourage high observer confidence in an incorrect goal. This is shown in Figure 2.2a. Note how the high-entropy path shown in green maintains observer ambiguity between the two goals.

The Shannon’s Entropy metric is:

$$D_{entropy}(X) = - \sum_{i=1}^n P(x_i) \log_n P(x_i) \quad (2.5)$$

Where X is the *planner observation* RNN’s goal predictions, and $P(x_i)$ is expected like-

likelihood in the *planner* has selected i -th goal location given the path. Similar to [3], the Simulation metric is formulated as:

$$D_{simulation}(X) = \max_{i \setminus i_{goal}} [P(x_i)] - P(x_{goal}) \quad (2.6)$$

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

$$\mathcal{D}(s) = \int_{path} (D(X|s)) ds, \quad (2.7)$$

An illustration comparing a path simulating a wrong goal to the optimal path is shown in Figure 2.2b. Note how the green path simulating goal B causes high observer confidence in the wrong goal.

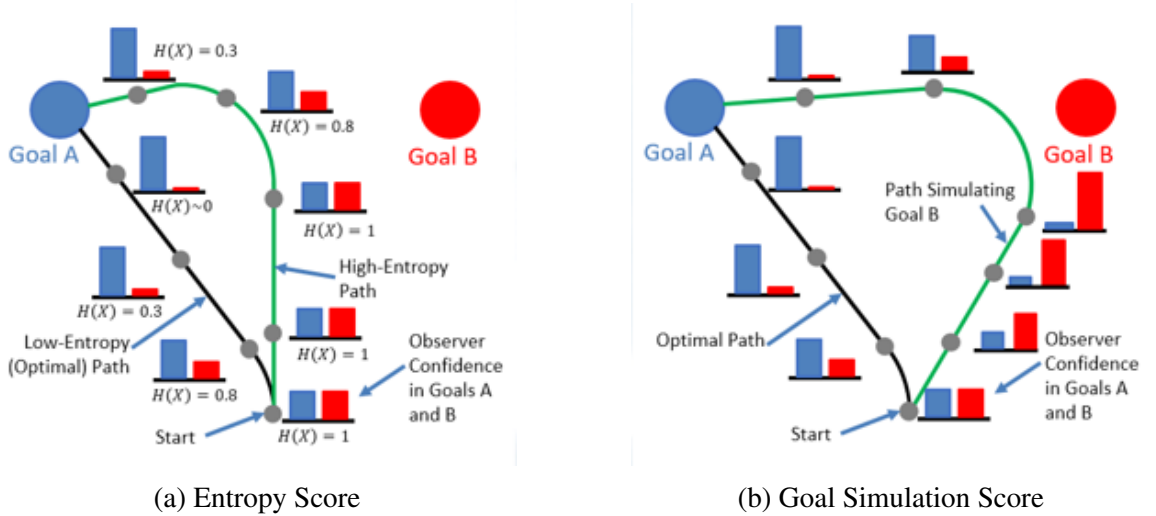


Figure 2.2: An illustration of the two deception scores in this work.

It should be noted that all integrals are discretized into their equivalent summation in the implementation of the Adversarial RRT* algorithm. They have been described here as integrals for theoretical generality.

Cost Function Implementation

The deception score, \mathcal{D} , is implemented by leveraging concepts from adversarial machine learning. A planner observation model is created to approximate the behavior of the observer the algorithm is targeting. In this thesis, an RNN architecture is used for both models. In this way, the *planner* can query a goal prediction for a path segment as it is planned, providing a measure of how deceptive the path could be to a potential observer. The cost function formulation is shown in Equation 2.8. The first term inside the integral is the path length. The $\alpha D(X|s)$ term, captures the predicted deceptiveness based on the *planner observation* RNN output.

$$Cost(s) = \int_{path} (1 - \alpha D(X|s)) ds, \quad \alpha \in [0, 1] \quad (2.8)$$

The α parameter is a tuning parameter that weights the deception cost. When $\alpha = 1$, the greatest possible value of both deception costs (upper bound of 1) will produce a path cost of 0. On the other hand, $\alpha = 0$ generates a path identical to that of RRT* with a path length cost.

The modifications made to the cost function deviate from the original RRT* formulation [8] because path 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. This is because the principle of optimality may be violated. 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.

CHAPTER 3

SIMULATED RESULTS

This chapter discusses the simulations used to evaluate the deceptiveness and overall performance of the Adversarial RRT* and the findings of these experiments.

3.1 Overview of Simulated Experiments

This section provides an overview of the simulated experiments used to evaluate the performance of Adversarial RRT*. First, an overview of the simulation environments across the various experiments is given. Then, the procedures used to generate training and experimental data is outlined. Finally, details about the *observer* and *planner observation* RNN architectures are given.

3.1.1 Simulation Environments and Experiments

An example environment is shown in Figure 3.1 with a start state and a set of 5 candidate goals. The *planner* plans a feasible path to one of the goals, selected at random, that balances path cost (length) and deception. A neural-network *observer* predicts the goal of the planner based on observations of the vehicle trajectory. The observer is assumed to be a naive adversary with access to its own planner (RRT*) but unaware that any deceptive actions are being taken against it. Performance comparisons are made between RRT, standard RRT*, and Adversarial RRT* on a Dubins vehicle model. RRT has potential to be highly deceptive, but is suboptimal. As such, it is anticipated that RRT paths will provide a benchmark for deceptiveness, but will have higher path costs. RRT* paths will serve as a baseline for optimality. No algorithm should produce paths that are, on average, lower cost than RRT* (since it is asymptotically optimal), but deceptiveness of RRT* should be very low.

This work also examines the performance of Adversarial RRT* against a vehicular adversary agent, referred to as a *pirate*. The *pirate* uses the *observer* predictions outputs and attempts to reach the most likely goal of the *asset/planner*. The left panel of Figure 3.1 shows the *planner* and *pirate* behavior with RRT*, while the right panel of Figure 3.1 shows the *planner* and *pirate* behavior with Adversarial RRT*. Note how the path qualitatively changes between RRT* and Adversarial RRT*. Also note how and the *observer* confidence has much greater variability when predicting from observations of Adversarial RRT* paths. Also notice the effects of the deceived observer confidence on the *pirate* behavior. Performance can be measured using the *observer* average prediction accuracy, as well as the rate at which the deceptive *asset* agent is able to defeat the *pirate* agent.

Finally, another scenario examines how this technique could scale to larger maps and more application-realistic vehicle dynamics. Specifically, a fixed-wing delivery drone is modelled, using simplifying assumptions, to be a Dubins' vehicle with a constant speed of 90 m/s and a minimum turning radius of 300 m. This drone operates in a 10-km square map (shown in Figure 3.2d) and must make a delivery to one of 5 delivery sites. Deceptive path planning will be employed in this scenario to maintain the privacy of the recipient of the delivery. Performance between RRT* and select Adversarial RRT* algorithms are compared using this platform.

3.1.2 Dataset Generation

In this work, the *observer* model uses a neural network (RNN) to predict the intended goal of the planner. The *planner* observation model neural network aims to approximate an observer in order to deceive it (but it is trained on different data). It is assumed the application scenario for the *observer* can be modeled by a simulation environment, in that the *observer* can simulate training examples to learn how to predict for the given scenario. Specifically in this work, the *observer* is assumed to be a naive adversary with access to its own planner (RRT*) to construct a training dataset. For training these neural networks,

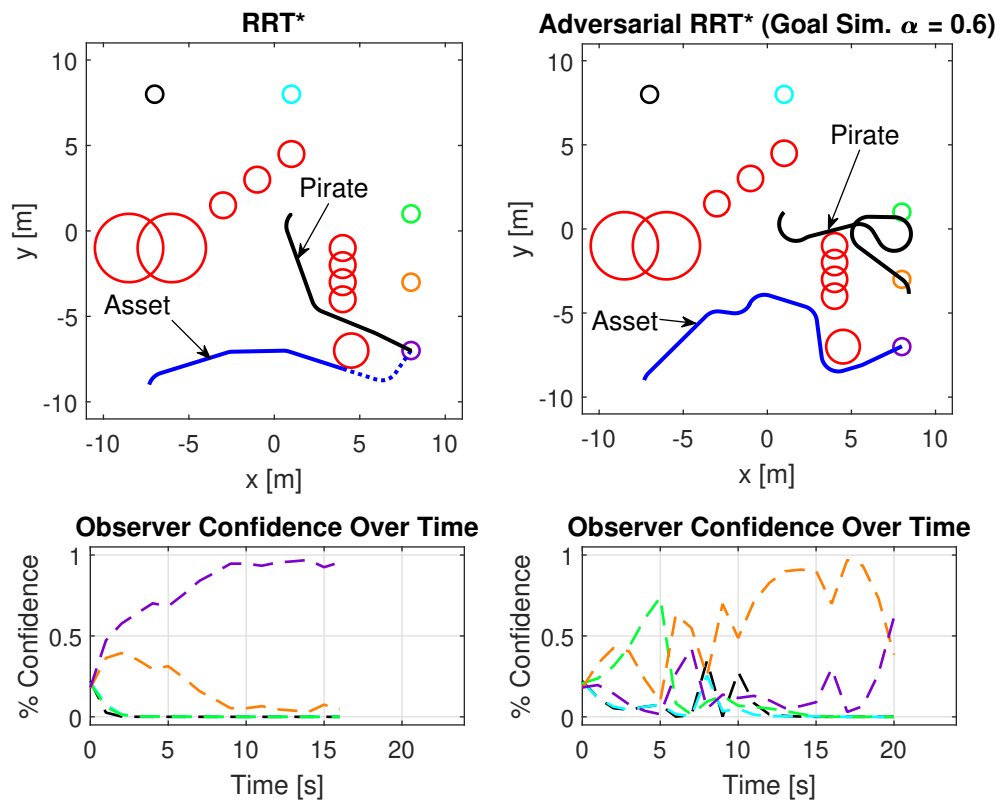


Figure 3.1: Illustration of the “Pirate Deception Scenario”. Red circles are obstacles. The 5 colored circles are possible goals. Below each is the observer confidence in each goal over time.

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, 4 different maps, shown in Figure 3.2, are utilized. The average best path cost over 100 paths as iterations increase is shown in Figure 3.3.

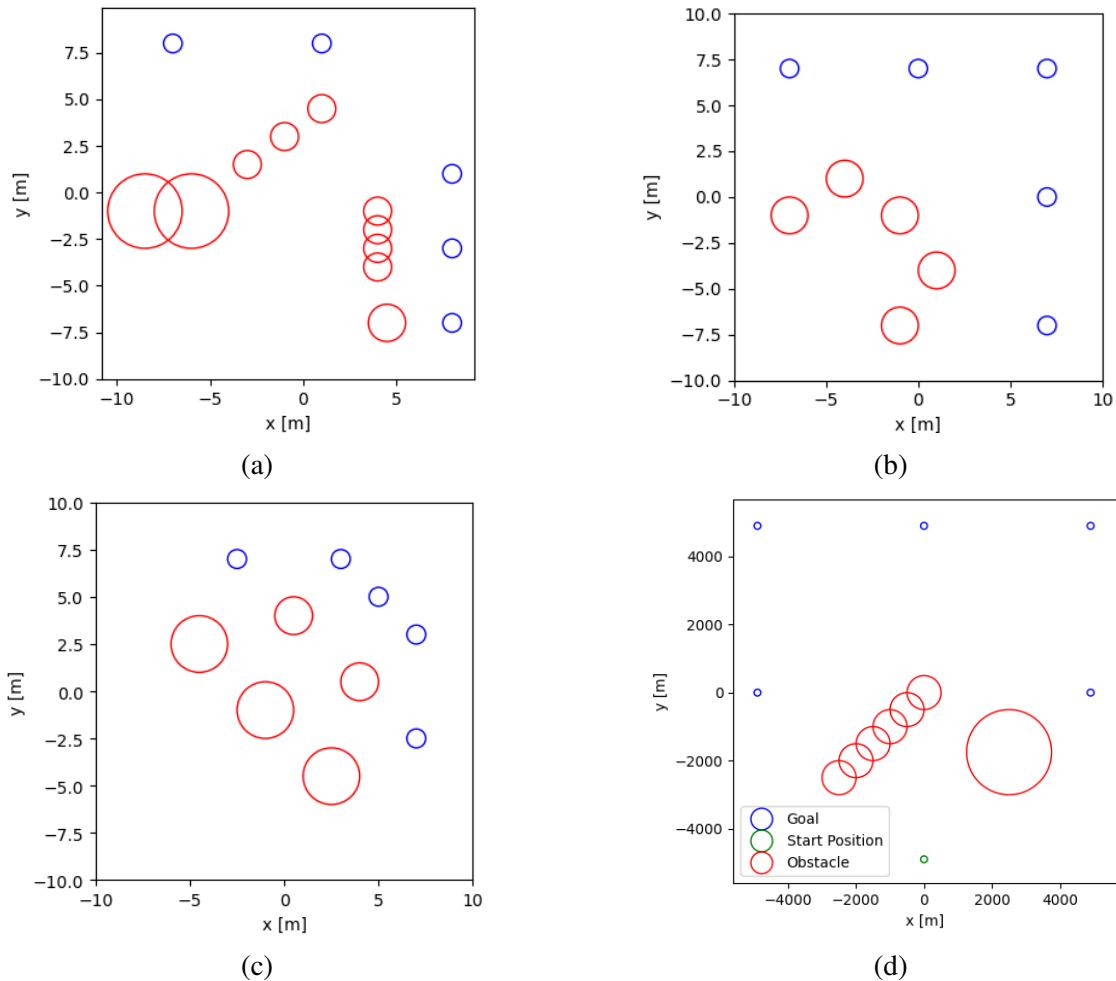


Figure 3.2: Additional environments simulated in this work.

The start state of the vehicle for each path is sampled from a multivariate normal distribution to generate a more diverse dataset. Initial position in each path was randomized around a nominal mean position with a 1-meter standard deviation from the nominal start location. Initial heading is randomized with a normal distribution with ± 45 -degree standard deviation about the nominal initial heading. The Dubins' vehicle had 1-m/s velocity and 1-m minimum turning radius, except in the aerial vehicle case. Simulations were performed on an Intel Core i7-8700U 3.2GHz processor with 32GB RAM.

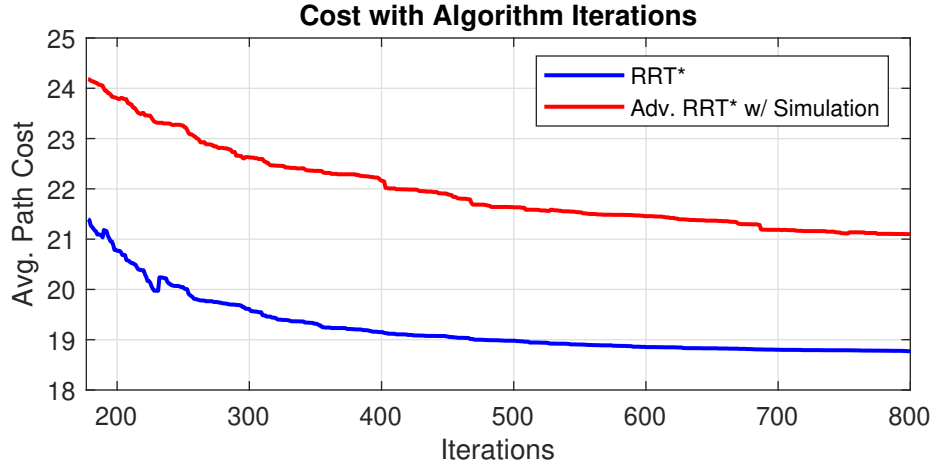


Figure 3.3: Avg. cost over 100 paths with increasing iterations.

3.1.3 Observer and Planner Model Configuration

The *observer* model considered in this work is an RNN using the Long Short-Term Memory (LSTM) layer. This is also the architecture of the *planner observation* model. The *observer* model architecture involves an LSTM layer with 1024 units followed by 4 fully-connected layers of decreasing dimension. The output is achieved through a Softmax layer. The RNN architecture is advantageous since it accepts variable input size, making it well-suited for the sequential nature of trajectory data. This architecture is similar to the goal prediction RNN in [23] that predicts goals from time-series observations. The RNN's used in this work include additional dense layers to provide more capacity and increased performance. The *planner observation* model has identical structure to that of the *observer* model.

3.2 Results of Simulated Experiments

This section presents and discusses the results from the simulated experiments.

3.2.1 Adversarial RRT* Performance Against RNN Observer

This section discusses the performance comparison between Adversarial RRT*, RRT*, and RRT. Paths from each algorithm were evaluated by comparing the performance of the

observer RNN on the paths. This naive *observer* was trained only using RRT* data and is not aware of any adversarial attack.

Paths were generated with each algorithm using 250 iterations for RRT* and Adv. RRT*. A range of α values between 0 and 1 were tested. The random sampling was seeded with the same sequence of random number generation seeds between the algorithms. Figure 3.4 shows a comparison of paths generated by RRT* (a) and Adversarial RRT* (b). The RRT* paths are near-optimal and take the most direct route to the goal. The Adversarial RRT* paths increase path length from optimal, but in doing so are able to maintain high ambiguity between the goals as long as possible. It can be seen that Adv. RRT* paths to different goals (shown using the Entropy cost function) tend to overlap until much later in the path before diverging to the respective goals.

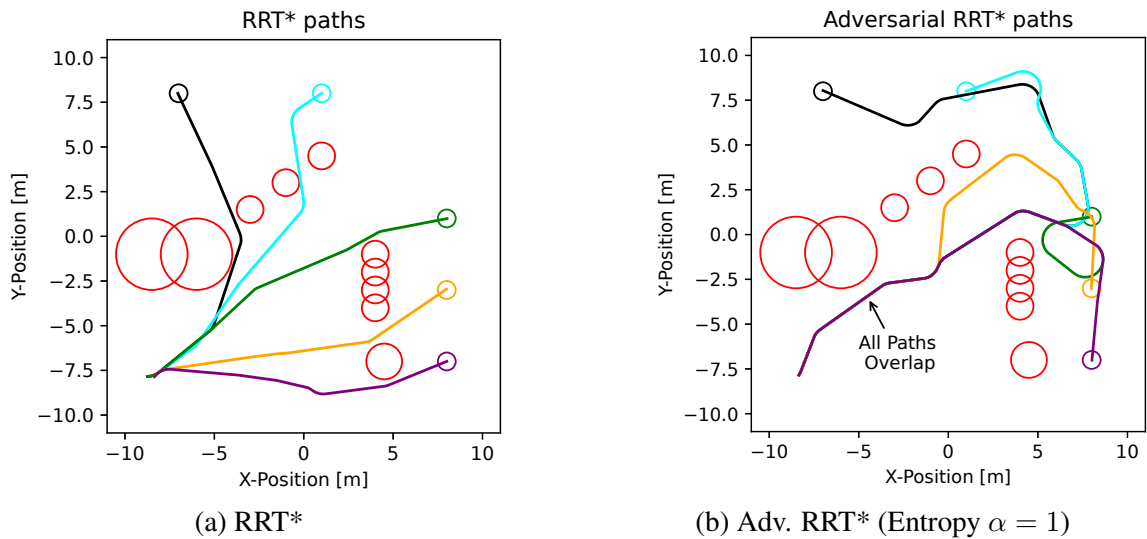


Figure 3.4: Paths using 250 iterations of each algorithm.

This analysis uses the *observer* RNN to make predictions on RRT, RRT*, and Adversarial RRT* paths. Figure 3.1 compares how the RNN performs on each algorithm’s paths over time. Note 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.

This analysis examines *observer* accuracy as a performance metric. The *observer*

achieves a correct prediction when its highest-confidence output class corresponds to the correct goal location. It is of interest to examine prediction accuracy over time to determine how early the observer typically converges on the correct goal. The average cumulative RNN accuracy, which is the average RNN accuracy across the entire path, is examined later.

Figure 3.5 shows the *observer* RNN accuracy on sets of paths from each algorithm over time. Is reversed from the intuitive direction (shown as timesteps *from* goal). This means as time from goal increases, the vehicle moves away from the goal, toward the start position. This adjustment allows the varying-length paths to be aligned according to a common condition (all paths reach the goal, but take different amounts of time to do so). The RRT* curve ends earliest since, moving backward from the goal, the start position is reached sooner due to a lower-cost path.

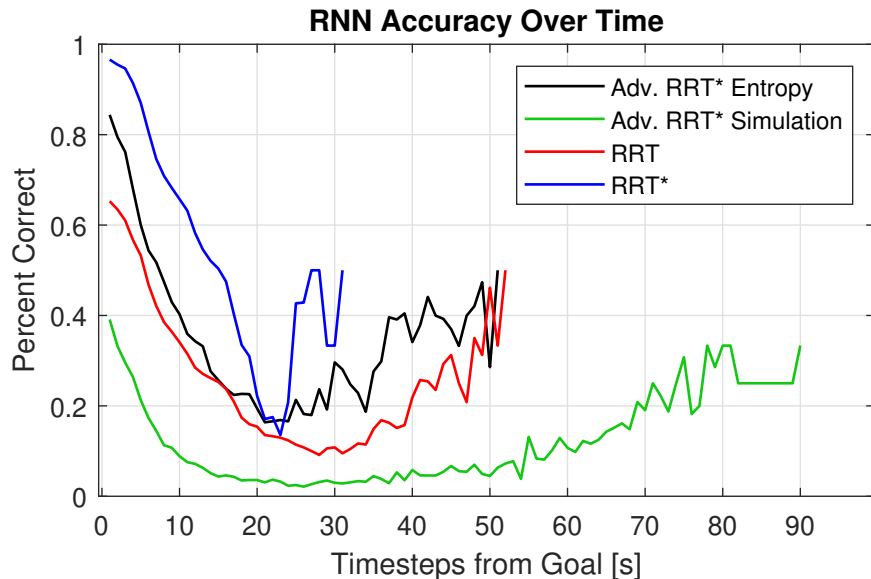


Figure 3.5: Accuracy on 500 paths planned with each algorithm.

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.

3.2.2 Comparison of Algorithm Performance

The cost function developed for Adversarial RRT* allow for longer paths to be selected as long as they increase the selected deception metric. In this case, deviation from the optimal path increases deceptiveness. Figure 3.6 shows performance frontiers for each Adversarial RRT* algorithm with varying α values. $\alpha = 0$ is equivalent to the RRT* algorithm. *Observer* RNN accuracy on RRT paths for the same Dubins' model is also shown as the blue star. Each data point represents an average of results from 500 paths each from the three maps (1500 paths total). The vertical axis is average accuracy of the observer RNN on paths generated with the indicated algorithm configuration. The horizontal axis is path length. A high-performing deceptive path planning algorithm will be found in the lower left portion of the graph, since it will have near-optimal path length, while causing low RNN accuracy.

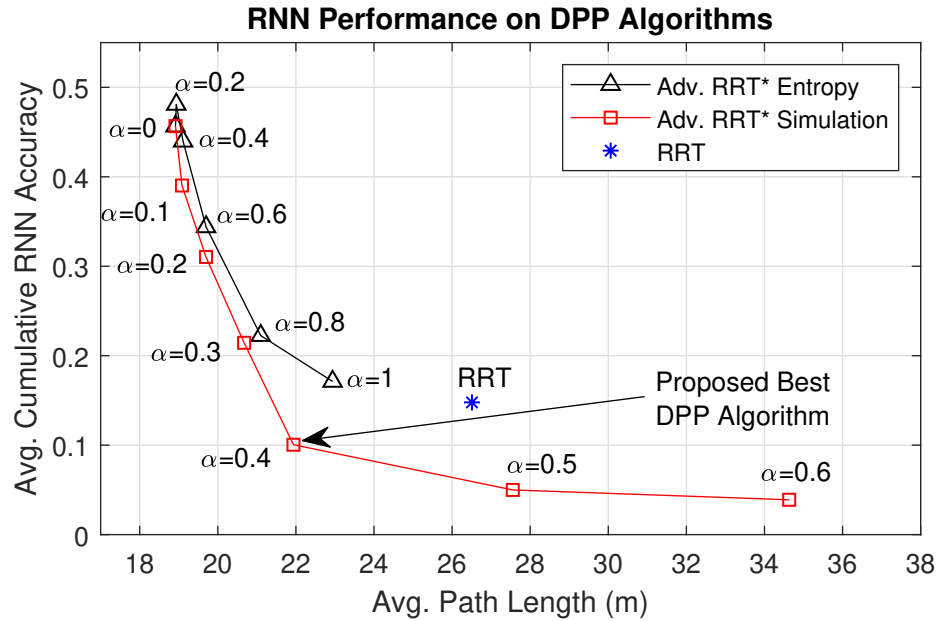


Figure 3.6: Performance of each algorithm with varying α values.

The paths produced by RRT are able to degrade *observer* RNN performance because of the highly randomized way in which RRT constructs paths. As a result of RRT suboptimality, however, average path length is 40% higher than optimal.

While slightly less deceptive overall Adv. RRT* with Entropy can still approach RRT-level deceptiveness (RNN accuracy of 17%) as α approaches 1, but is able to achieve closer-to-optimal average path length compared to RRT.

Adv. RRT* with Goal Simulation is able to provide higher performance in both path optimality and deceptiveness. It achieves higher levels of deception (10% RNN accuracy) than the RRT and Entropy paths, while maintaining average path length within 16% of optimal.

3.2.3 Pirate Deception Scenario Performance

This section discusses a use case for deceptive path planning in which an agent delivers an *asset* to a predetermined goal region (chosen from a set of candidates) before a *pirate* adversary can detect and reach the goal region to interrupt the delivery. Here, both the *asset* and the *pirate* are equally-maneuverable Dubins' vehicles (1 m/s speed and 1-m turn radius). The *pirate* has an initial position approximately in the centroid of the candidate goals.

The *pirate* is provided with 1-Hz surveillance from the *observer* RNN that gives the expected likelihood that the *asset* has selected each goal candidate based on its observed trajectory. The pirate uses RRT* with 150 iterations to reach the most likely goal from its current state and executes the planned path until the most-likely goal changes or the *asset* reaches its goal. On the other hand, the *asset* will employ various deceptive path planning techniques to fool the *observer*, in an attempt to delay or prevent the *pirate* reaching the goal.

A select example of this simulation is shown in Figure 3.1. The top-left panel shows the *pirate* (black) response to an RRT* *asset* path (blue). When the *asset* uses RRT*, the *pirate* is able to interrupt the delivery by reaching the goal before the *asset*. The segment of the *asset* path prior to the *pirate* reaching the goal is plotted as a solid blue line. After the *pirate* reaches the goal, the blue path is dashed, indicating the *asset* did not complete its

objective. The top-right panel demonstrates the *pirate* behavior in response to to an *asset* using Adv. RRT* with Simulation ($\alpha = 0.4$). In this case, the *asset* achieves its objective of reaching the goal before the *pirate*, as shown by the truncated pirate path. In this example, the *asset* employing Adv. RRT* is shown to fool the *observer* used by the *pirate* by using path segments that simulate paths to other goals (the “goal simulation” strategy).

500 paths of RRT*, Adv. RRT* with Entropy, and Adv. RRT* with Simulation were generated using the map shown in Figure 3.4a. 52.8% of RRT* paths are successful in reaching the goal region before *pirate*, while 75.4% of paths from Adv. RRT* with Simulation ($\alpha = 0.4$) and 74% with Entropy ($\alpha = 1$) are able to defeat the pirate successfully. This example application shows how deception of an adversary’s observation mechanism may improve an agent’s performance in adversarial games.

3.2.4 Larger Drone Delivery Scenario

To illustrate potential for this method to extend to various vehicle dynamics, a similar navigational goal prediction scenario modelling a vehicle and map suited to a particular real-world application. Specifically, this simulation models a drone delivery example in which a fixed-wing delivery drone must maintain privacy while delivering a package to one of several regions. To model the fixed wing plane, a Dubins’ vehicle with fixed speed of 90m/s and a turning radius of 300m was selected. The map, shown in Figure 3.2d, is roughly a 10-km square, representing a longer-range delivery scenario. The RNN training and experimental protocols were similar to the other experiments and scenarios presented in this thesis.

This simulated experiment compared privacy performance of the plane using RRT* to identical planes using two Adversarial RRT* configurations (Simulation with $\alpha = 0.4$, and Entropy with $\alpha = 1$). Average *observer* accuracy on RRT* paths in this simulation 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.

This demonstrates the applicability of the Adv. RRT* framework to other vehicles and applications.

CHAPTER 4

CONCLUSION AND FUTURE WORKS

This thesis proposes a novel deceptive path planning technique inspired by adversarial machine learning. This technique integrates two different deception cost functions based on querying an adversarial neural network into a sampling-based planner. A Dubins' vehicle dynamic model is used to demonstrate the kinodynamic nature of the planner in a simulated environment. This planner, called Adversarial RRT*, is evaluated against an *observer* RNN attempting to perform navigational goal recognition. In these experiments, Adversarial RRT* reduces classification accuracy and delays the onset of high observer confidence. Other scenarios in which Adversarial RRT* can be leveraged are presented. The pirate deception scenario demonstrates that Adversarial RRT* can improve the security of a high-value delivery. The drone delivery scenario shows the applicability of the algorithm to other dynamic systems.

This work reveals several areas of potential future investigation. One such avenue is exploring the extensions of Adversarial RRT* to platforms with more complex dynamics and constraints.

Another area worth investigating is a more comprehensive approach to deception that could scale to multi-agent scenarios and adversarial games. For example, in the pirate deception scenario, taking into account the pirate's dynamics and kinematics as part of the deception paradigm may provide significantly improved success rates. On the other hand, discovering ways to fool a pirate that follows a more complex policy for acting given the observer output is a potentially useful endeavor.

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, ISBN: 978-0-9923747-0-9.
- [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, ISBN: 978-0-9992411-0-3.
- [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. arXiv: 1911.10074 [cs.AI].
- [6] M. Tambe, “Security and Game Theory: Algorithms, Deployed Systems, Lessons Learned,” Cambridge: Cambridge University Press, 2011, ISBN: 978-0-511-97303-1.
- [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 *et al.*, “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] 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.
- [15] D. Pattison and D. Long, “Accurately determining intermediate and terminal plan states using bayesian goal recognition,” in D. Pattison, D. Long, and C. Geib, Eds., *ICAPS*, Jun. 2011, pp. 32–37.
- [16] 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, ISBN: 978-0-9992411-0-3.
- [17] 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.
- [18] P. Masters, M. Kirley, and W. Smith, “Extended Goal Recognition: A Planning-Based Model for Strategic Deception,” p. 9, 2021.
- [19] 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, ISBN: 978-1-5090-6014-6.
- [20] 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, ISBN: 978-1-4503-8307-3.
- [21] 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.

- [22] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*, ser. Adaptive computation and machine learning. Cambridge, Massachusetts: The MIT Press, 2016, ISBN: 978-0-262-03561-3.
- [23] L. Amado, J. P. Aires, R. F. Pereira, M. C. Magnaguagno, R. Granada, and F. Meneguzzi, “LSTM-based goal recognition in latent space,” Aug. 20, 2018.
- [24] L. Amado, J. P. Aires, R. F. Pereira, M. C. Magnaguagno, R. Granada, and F. Meneguzzi, “Lstm-based goal recognition in latent space,” *arXiv preprint arXiv:1808.05249*, 2018.
- [25] 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.
- [26] P. Masters, “Goal recognition and deception in path-planning,” *Ph. D. dissertation, RMIT University*, 2019.
- [27] 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.
- [28] C. E. Shannon, “A mathematical theory of communication,” *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.