Project: Learning Based RRT using a Visual Classifier as a Cost Function

Aaron Kampmeier akampmei@asu.edu
Brandon Evans bpevans@asu.edu
Braeden Woodard bmwoodar@asu.edu
Pavan Kumar Raja praja3@asu.edu

1 Introduction

Rapidly-Exploring Random Trees (RRT) [**?**] have proven to be a very useful sample-based path-planning algorithm to efficiently search high-dimensional spaces. However, RRT can be hindered in environments with obstacles or specific pathways to a goal. The algorithm may waste time exploring down irrelevant or blocked paths. In this paper, we will survey various learning approaches to problems encountered by RRT in such environments, evaluate a baseline modified RRT algorithm based on behavior cloning, present the results of our Visual Classifier assisted RRT algorithm, and discuss our future work.

2 Problem Statement

A basic RRT algorithm's performance can suffer in configuration spaces with obstacles by exploring down irrelevant or blocked paths. The surveyed learning improvements to RRT may help but they still need a way to discover where the goal states and relevant obstacles are in relation to the agent. This problem can arise a variety of different domains such as expansions through mazes, robotic control in constrained spaces [**?**], and systems testing of multi-step processes [**?**]. There exists previous work regarding trying to solve this problem via RRT algorithms that use Behavior Cloning to seed the RRT tree. This paper uses one of these algorithms, CA-RRT[**?**]. as a baseline and aims to develop a BC RRT algorithm that uses a visual classifier to guide RRT expansion.

3 Related Work

In the last few years, several contributions to robotic navigation and exploration have been proposed that incorporate variations of sampling-based costmap planners, such as RRT or RRT*, and learning from demonstrations. Zuo et. al. [**?**] proposes a method, Clone Assisted RRT (CA-RRT), that learns heuristics for a search-based algorithm from a limited number of human demonstrations. CA-RRT is an expansion of the human-seeded RRT (HS-RRT) proposed by Chang et. al. [**?**], which uses states from human demonstrations as seed configurations states within an RRT. This expansion includes adding a first-person image-state-based (behavioral) cloning approach over the HS-RRT. Zuo et. al. [**?**] method consists of collecting a set of human-created trajectories and training a behavioral cloning policy using the first-person state-action pairs from those trajectories to seed the RRT. These heuristics allow the search algorithms to automatically test other states in the game thus removing the necessity to test all possible scenarios in the game through human game play.

Perez et. al. [**?**] also proposes an approach to learn navigation behaviors from demonstrations with the use of sampling-based costmap planners, though they propose the use Inverse Reinforcement Learning (IRL) concepts to identify the RRT cost function that best fits the example trajectories. The proposed method, Rapidly-exploring random Trees Inverse Reinforcement Learning (RTIRL), makes use of the RRT* instead of a Markov Decision Process (MDP). This is due to the advantages that RRT* has over MDP, in which RRT* can handle continuous state and control spaces. Perez et. al. [**?**] RTIRL method is used to extract the proper weights of the cost function from demonstration

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

trajectories, which can then be used later in the RRT* process to allow the robot to reproduce the desired behavior at various scenarios.

Both papers propose learning algorithms that use sampling-based costmap planners for path planning and exploration. Our proposed method will also be expanding upon the RRT algorithm and involve collecting and utilizing human demonstrations to allow for learning from demonstrations. Our proposed algorithm will incorporate a similar behavior cloning (BC) approach to that of Zuo et. al. [**?**] as the BC policy will be act as a seed in the initial RRT configuration. In addition to BC, we aim to use a visual classifier to learn a cost function, by using an image to gather samples from various locations to determine the cost to reach the goal state. This approach is a development of and similar to the work by Majeed et al., who successfully used an image pre-processor to locate objects in an environment prior to performing path planning utilizing an A* algorithm. [**?**] They found that using an image processor to locate obstacles in combination with A* planning was successful in path finding in both static and dynamic environments. [**?**] We aim to further expand on this paper's work by implementing image processing into the cost function of RRT rather than pre-processing. We aim to train a behavior cloning policy on a 2D image of the environment, similar to the greyscale 2D images used in the paper by Majeed et al. as our image also uses high contrast to aid in the performance of our algorithm.

4 Problem Domain

Currently, the problem domain consists primarily of 2D pygame environment which contains static obstacles that create a maze that the agent must navigate. The background of this environment is black, the obstacles and agent blue, and the goal green to add needed constrast for the visual classifier. Currently, the domain can consist of both static and domain obstacles, starts, and goals based on user arguments when running the agent.

For our problem the agent is creating a path to navigate from start to goal. In order to do this, our agent uses a variety RRT algorithms.

Out agent is tested in a both static and dynamic 500x800 pixel environments using either basic RRT, BCRRT, or Image BCRRT. BCRRT is our baseline, and Image BCRRT is our visual classifier assisted BCRRT. For the Image BCRRT the input to the agent is a 2D image of the environment with the action the agent is taking in that state.

5 Learning Baseline

To create a baseline for our CARRT, we require a demonstration. As we planned to use static obstacle space, we ignored the location of obstacles in creating the demo. We used pygame to set up our environment and demonstrations were collected by us playing the game using WASD keys. Each demonstration is a trajectory with (state, action) pairs. Each state is represented as (the current position of the agent, goal position) and action is represented as ENUM of WASD keys. We used a simple MLP model as a prediction/action generator, a summary of the model is given in Table 1. For the next phase of the project, we'll keep this model as a baseline and expand it to accommodate dynamic world state space.

6 Experiment Metrics

The metrics being evaluated in the experiment include the following:

- Success Rate: The percentage of successes that the agent successfully reaches the goal.
- Time: The time taken by the agent from the start of RRT to the terminal node or condition i.e. when the limit of 1000 expansions is met.
- Number of Expansions: The number of nodes in the RRT graph at termination.

7 Experiment Results

As our goal was to develop a RRT algorithm that uses a visual classifier to learn a cost function. The baseline used for this paper required a learning-based approach, as such, we decided to utilize CARRT as our baseline rather than a standard RRT. The above metrics were used for an initial evaluatation of both our baseline algorithm, which was a version of CA-RRT (BCRRT), and a standard RRT algorithm. In order to ensure the baseline BCRRT was performing correctly, the metric results must reflect an improvement of BCRRT over the strandard RRT. The initial baseline results for BCRRT can be seen in Table 1, while the results for RRT are shown in Appendix A: Table 5. The experiment does not directly compare RRT with our proposed IMGBCRRT algorithm, as we are solely focusing on a learning-based approach. After training numerous models of IMGBCRRT, we proceeded with the experiment phase of this paper, this included the testing of our proposed algorithm. We performed testing on three different configurations of IMGBCRRT. The first configuration consisted of a static enviroment with static obstacles and static start/goal locations, which is labeled as IMGBCRRT SA, and is shown in Table 2. The second configuration consisted of a mixed enviroment with dynamic obstacles and static start/goal locations, which is labeled as IMGBCRRT SDO, and is shown in Table 3. The last configuration consisted of a dynamic enviroment with dynamic obstacles and dynamic start/goal locations, which is labeled as IMGBCRRT DA, and is shown in Table 4.

To properly evaluate the performance of IMGBCRRT, the experiment consisted of testing the three configurations of IMGBCRRT against the baseline BCRRT. The BCRRT configuration consisted of a static enviroment with static obstacles and static start/goal locations. The experiment was done over five iterations with the results of the metrics being collected during each iteration of the test. The mean and standard deviation of the metric results are provided in each results table as well as a box and whisker plot. The results of the baseline BCRRT compared to RRT reflect an expected outcome of BCRRT performing better overall than the RRT algorithm. The RRTprovides a mean success rate of .79, a mean time of 3.39 s $\pm$ 1.20 s, and a mean expansions of 744.48 $\pm$ 208.45. While the BCRRT provides a mean success rate of .90, a mean time of 1.94 s $\pm$ 1.73 s, and a mean number of expansions of 436.48 $\pm$ 375.33.

In regard to the results of the direct comparison of BCRRT and IMGBCRRT SA, we can see a significant improvement on all metrics. The results show a significant reduction of time with IMGBCRRT SA with a mean time of .92 s $\pm$ .31 s and a mean number of expansions of 22.09 $\pm$ 84.50. The evaluation of the two remaining configurations of IMGBCRRT in comparison to IMGBCRRT SA reflect an increase of mean time and number of expansions when dynamic obstacles are used though the success rate remains at .90. We can see from the results that the standard deviation (SD) seems rather large, this may be due to the small number of iterations that were used in testing, this can be addressed in future work by performing testing with a larger number of iterations. Due to the large SD values, we further analyzed the data by calculating the coefficient of variation (CV) for each of the results to determine the level of dispersion around the mean, shown in Appendix A: Table 6. Despite the large standard deviations seen in the results, we see a reasonable dispersion of mean time and number of expansions (with the exception of IMGBCRRT SA).

Table 1: Results from BCRRT Algorithm

Iteration	Success Rate	Mean Time(s)	Std. Dev Time	Mean Expansions	Std. Dev Expansions
1	.89	1.9586	1.8819	440.6875	378.6503
2	.91	2.0340	1.7459	442.09	372.4981
3	.92	1.6560	1.4974	439.7563	375.8521
4	.90	2.0366	1.8273	428.4387	376.2023
5	.89	2.0287	1.6985	431.4369	373.4285
AVG	.902	1.9428	1.7302	436.4819	375.3263

Table 2: Results from IMGBCRRT SA Algorithm

Iteration	Success Rate	Mean Time(s)	Std. Dev Time	Mean Expansions	Std. Dev Expansions
1	1.0	.8916	.2051	19.8276	66.0555
2	1.0	.8958	.1310	18.4576	63.0348
3	1.0	.8971	.1238	16.7414	54.4690
4	1.0	.9432	.4144	25.1466	101.5863
5	.98	.9743	.6745	30.3017	137.3698
AVG	.996	.9204	.3097	22.0950	84.5031

Table 3: Results from IMGBCRRT SDO Algorithm

Iteration	Success Rate	Mean Time(s)	Std. Dev Time	Mean Expansions	Std. Dev Expansions
1	1.0	1.4602	.8591	196.7556	194.4713
2	.92	2.6790	1.8534	374.2079	313.2433
3	.73	4.3425	1.8081	696.5349	293.5065
4	.98	1.8653	1.6107	208.3444	290.2712
5	.98	1.6502	1.0182	222.1038	249.5101
AVG	.922	2.3994	1.4299	339.5893	268.2005

Table 4: Results from IMGBCRRT DA Algorithm

Iteration	Success Rate	Mean Time(s)	Std. Dev Time	Mean Expansions	Std. Dev Expansions
1	.99	2.2373	.7730	283.5508	181.8569
2	.94	2.7234	1.1715	564.7465	246.4456
3	.81	2.2974	1.4021	360.9149	378.8530
4	.91	1.4934	1.0860	257.0303	303.3063
5	.86	3.6526	1.2020	637.5954	264.6904
AVG	.9020	2.4808	1.1269	420.7676	275.0304

8 Future Work

We plan to build on our baseline and accomplish these listed tasks.

- Creating and training a visual classifier to learn a cost function in the RRT.
  - Collecting demonstration for variable goal and obstacle environment.
  - Training a visual classifier to recognize a “goal region” in a 2D image.
  - Use the visual classifier as a cost function to aid in guiding the agent’s expansions towards the goal faster.
- Perform testing on the Visual-RRT against the CA-RRT behavior cloning baseline.
- Prepare data visualizations of the data collected in the comparison testing.
- Perform student’s t-test on the two algorithm results.

9 Appendix A: Supplementary Material

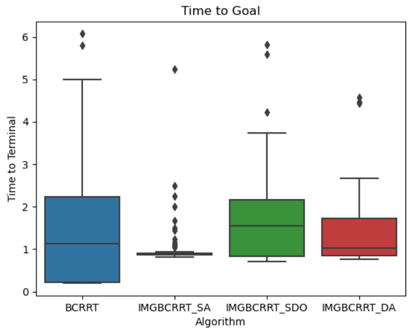

Figure 1: Time to reach goal

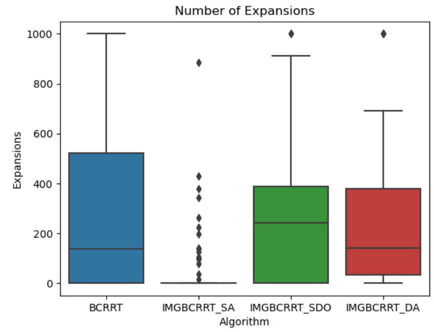

Figure 2: Number of nodes expanded

Figure 3: Expansion based solely on RRT

Figure 4: Expansion after behavior cloned roll-out

Table 5: Metrics from RRT Algorithm

Iteration	Success Rate	Mean Time(s)	Std. Dev Time	Mean Expansions	Std. Dev Expansions
1	.79	3.6179	1.2385	745.3562	211.5847
2	.81	3.5861	1.1871	752.03	202.2828
3	.79	2.9090	1.0498	751.5603	209.2586
4	.77	3.4672	1.2880	734.6838	213.4561
5	.79	3.3586	1.2298	738.7589	205.6531
AVG	.79	3.3878	1.1986	744.4778	208.4471

Table 6: Coefficient of Variation (CV)

Algorithm	CV Time(s)	CV Expansions
RRT	0.3544	.2800
BCRRT	.8916	.8600
IMGBCRRT SA	.3292	3.7147
IMGBCRRT SDO	.6354	.9527
IMGBCRRT DA	.4884	.7445