Title

Author 1 and Author 2
Address line
Address line
And
Author 3
Address line
Address line

Abstract

Probabilistic planners have demonstrated promising results in dealing with robotic navigation under uncertainty (Likhachev and Stentz 2008), for example, when parts of a map are unknown or the robot must navigate amongst other unpredictable agents. To compute trajectories, probabilistic planners rely on atomic actions which are strung together to form a path. Planners will often invoke a lower level controller to execute these actions. Existing work focuses on situations in which the humans in the scene move such that completion of the desired action is always feasible. Little attention has been paid to scenarios in which humans make reaching the goal waypoint impossible or prohibitively costly. Current methods may simply attempt to replan in these cases which can be too slow to avoid imminent collisions. Additionally, if the waypoint is unreachable or costly to reach, this approach may incur a higher cost than abandoning the waypoint and moving to a different location first before replanning. This paper presents a reinforcement learning based approach to developing *failure controllers* for exactly these situations which direct the robot to an optimal failure location if its original goal waypoint becomes unreachable. We also present a method for determining when this controller should be employed by the robot that leverages calibrated estimates of the uncertainty of the robot in its next move. We refer to this element as a *discriminator*. The combination of a failure controller with this discriminator allows the robot to determine when it is unable to reach its intended goal and to react to the situation accordingly, generating safer, more efficient trajectories.

Introduction

failure controller = what we follow when our goal is infeasible/costly.

success controller = what we follow by default when the planner invokes this controller for a given scene.

discriminator = function to determine when to switch from success to failure controller.

Background

Related Work

Problem Statement

The problem statement is two-fold. First, a failure controller that can direct the robot in cases where it has no navigation goal is sought. This is in the form of a policy selecting one of the robot's possible actions conditioned on the its current state. Second, we wish to determine at each time step whether or not the robot should start executing its failure controller for a given scene. This should occur if and only if the robot determines that its original navigation goal is infeasible (or is likely to be infeasible).

Failure Controller Consider a robot using a probabilistic planner to navigate in a pedestrian environment. The environment contains both static and dynamic obstacles in the form of humans and other objects. This robot has a set M of motion primitives which it can use to perform simple movements. The robot also has a set C of controllers which it can execute to perform more complex behaviors that may require cooperation from humans. Because these controllers have some probability of failing, the robot needs a contingency plan to follow that will return it to a safe location.

This paper demonstrates our framework by applying it to a *barge-in* controller. This controller addresses a situation in which humans block a narrow corridor and the robot wishes to move past them. Ideally, the robot would drive towards the group of people who would then part enough for the robot to reach its goal. However, if the people do not move—for example, if they do not notice the robot or they cannot infer its intention to move past them—the robot needs a contingency plan that results in safe behavior and brings the robot back to a location from which it can replan.

We model the problem of finding the optimal failure policy as solving a Markov Decision Process $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma \rangle$ with an infinite set of states $\mathcal{S}$, a finite set of actions $\mathcal{A}$, state transition matrix $\mathcal{P}(s_t, a_t, s_{t+1})$, reward function $\mathcal{R}(s_t, a_t, s_{t+1})$ and discount factor $\gamma \in (0, 1)$ (Sutton and Barto 1998). The robot seeks an optimal policy $\pi^*(a_t \mid s_t) = P(a_t \mid s_t)$ where optimal indicates that the policy maximizes the overall expected discounted reward the robot receives (referred to as the "return" of the policy) from the current time step t onwards:

$$\sum_{k=0}^{\infty} E[\gamma^k \mathcal{R}(s_{t+k}, a_{t+k}, s_{t+k+1})] \tag{1}$$

To aid in finding this policy, we define a $Q_\pi(a_t, s_t)$ function which estimates the return of executing action a_t from state s_t and following policy π from that point onwards:

$$Q_\pi(a_t, s_t) = \sum_{k=0}^{\infty} E[\gamma^k \mathcal{R}(s_{t+k}, \pi(a_{t+k} \mid s_{t+k}), s_{t+k+1})] \tag{2}$$

$$\begin{aligned} Q_\pi(a_t, s_t) = \sum_{s' \in \mathcal{S}} \mathcal{P}(s_t, a_t, s')[\mathcal{R}(s_t, a_t, s') \\ + \gamma \sum_{a' \in \mathcal{A}} \pi(a' \mid s') Q_\pi(a', s')] \end{aligned} \tag{3}$$

Assuming π is greedy, we can rewrite[1]

$$Q_\pi(a_t, s_t) = \mathcal{R}(s_t, a_t, s_{t+1}) + \gamma max_a Q_\pi(a, s_{t+1}) \tag{4}$$

We denote the Q function of the optimal policy as Q^*. Q^* gives a convenient way to determine the optimal action for any state given that the action space is finite and small because we can simply find the action with the largest Q^* value and execute it.

Discriminator At each time step, the discriminator indicates whether or not the robot should stop executing its current controller and transition to the corresponding contingency plan (failure controller). Once the robot has made this transition it cannot return to its original controller until the failure controller has finished its execution and the robot has determined it is safe to replan. Therefore, we wish to develop a discriminating criterion that is met if and only if the robot would truly be better off (would execute a safer or more efficient trajectory) if it switches to its failure controller.

In this paper, we specifically focus on navigation controllers represented by a neural network. Intuitively, we wish to know whether or not this network is confident that it can reach its intended goal. We judge this confidence by measuring the uncertainty of the network in its prediction of the next step for the robot to take and make the discriminator a function of this uncertainty.

Approach

Both the default and failure controllers are represented as neural networks.

Failure Controller **This is the same for the failure controller as it was in the RISS paper.**

Discriminator To determine whether or not the robot is likely to succeed in a given scene, we determine the uncertainty of the success controller and compare it to a threshold. If it ever exceeds a precomputed threshold (specific to each controller), then the robot transitions to its failure controller and continues executing it until it reaches its timeout. For particularly unstable scenes where the uncertainty fluctuates greatly between time steps, it may be appropriate to examine the rolling average of the uncertainties rather than the value itself at each time step.

In order to compute this threshold, we create a training set of success examples for a given scene. This training set consists of many (1000) trial runs in which the robot successfully reaches its goal in a given scene, stored as time series data of the locations and velocities of the robot and each human.

The success controller network has the same architecture as that of the failure controller, except it only has five outputs instead of 35. These outputs represent the expected next position of the robot, (p_x, p_y), the logarithm of the standard deviation of this prediction $(log\sigma_x, log\sigma_y)$, and the inverse hyperbolic tangent of the correlation between the x and y prediction $(tanh^{-1}(\rho))$. This network is trained using a negative two-dimensional Gaussian log-likelihood loss where the target at each time step is the ground truth robot position at the next time step. This network is trained with a high dropout probability of 0.4 for 300 epochs. At testing time, to determine the uncertainty of the network's prediction, we add its data uncertainty to its model uncertainty. To compute these values, we simply run the robot's observed state through the network some number of times (20 in our specific case) to find the average standard deviation in the x and y directions reported by the network (data uncertainty) as well as the standard deviations of the target positions reported by the network for both the x and y directions (model uncertainty). These two values are summed to find the total uncertainty in each direction (Gal 2016). To find the square of the overall magnitude of this uncertainty we sum the squares of these individual components.

At testing time, we simply see if this total uncertainty magnitude exceeds a precomputed threshold. To find this threshold, we simulated the success controller in 100 cases similar to the training data set that allowed the robot to reach its goal, and 100 cases in which the robot could not reach its goal due to human movement it the scene. Each trial ran for 100 time steps. We then examined the maximum uncertainty realized in each of these trials by the success controller and chose a threshold that minimized the total number of erroneous categorizations (the sum of the number of familiar trials with higher uncertainty and unfamiliar trials with lower uncertainty). This threshold is then set for this specific success controller.

Experiments

To test the effectiveness of our approach we ran three different types of trials for a *barge-in* controller (working on some of them, as well as overtake). This controller applies to a scene in which a group of humans are standing at the end of a hallway and the robot would like to barge past them to reach a goal on the opposite side. In a successful execution of this controller, the humans will move out of the way for the robot to pass through. In an unsuccessful execution, the humans may remain stationary or may even move into the corridor, preventing the robot from passing.

We test first on an unfamiliar scene in which the humans move into the corridor the robot occupies. In one set of trials, we enable the discriminator and failure controller so that the robot will respond to the novel scene by transitioning to its contingency plan once it realizes that its original goal is infeasible (specifically, once it is suitably uncertain that it will reach its intended goal). If the robot's uncertainty never

[1]This rewrite also makes use of the fact that the ground-truth next state is an unbiased estimator of the expected next state.

reaches this threshold after 100 time steps, the trial is simply ended. The failure controller was run for 100 time steps before being cut off.

(I haven't done this experiment yet but it should be quick, hopefully)In the next set of trials, we deactivated the discriminator such that the success controller would run until it was cut off after 100 time steps. The robot then follows this policy and ignores its uncertainty throughout the trial.

Finally, in the last set of trials we placed the robot in a familiar scene with the failure controller and discriminator activated. This was done to ensure that the robot does not aggressively trigger the failure controller: it should only be used when the robot is truly uncertain about its ability to reach the goal. These trials serve to verify that the uncertainty threshold is high enough: the robot will continue following its success controller as long as it is reasonably certain it can reach its goal.

Results

After carrying out 100 of each type of trial, the overall paths produced in each run were rated according to:

- Path length: The total distance traveled by the robot. In general we would like the robot to move less when possible (find the shortest path when executing a success controller and minimize backwards progress when executing the failure controller).
- Angular distance: The changes in heading the robot underwent. This is used to approximate the smoothness of the robot's path.
- Collisions: The sum of the number of people the robot is in collision with in each time step. We place the highest priority on minimizing this value.
- Intrusions: The sum of the number of people the robot intrudes on in each time step. An intrusion is defined as the robot being within $0.2m$ of a person. All other things equal, we would like to decrease the number of intrusions, but some number of them are acceptable if they are necessary to reach the goal.

(Insert table of results)

Conclusion

References

Gal, Y. 2016. *Uncertainty in deep learning*. Ph.D. Dissertation, PhD thesis, University of Cambridge.

Likhachev, M., and Stentz, A. 2008. Probabilistic planning with clear preferences on missing information. *Lab Papers (GRASP)* 25.

Sutton, R. S., and Barto, A. G. 1998. *Introduction to Reinforcement Learning*. Cambridge, MA, USA: MIT Press, 1st edition.