Learning over Subgoals for Efficient Navigation of Structured, Unknown Environments

Gregory J. Stein,* Christopher Bradley,* and Nicholas Roy
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
{gjstein, cbrad, nickroy}@mit.edu

Abstract:
We propose a novel technique for efficiently navigating unknown environments over long horizons by learning to predict properties of unknown space. We generate a dynamic action set defined by the current map, factor the Bellman Equation in terms of these actions, and estimate terms, such as the probability that navigating beyond a particular subgoal will lead to a dead-end, that are otherwise difficult to compute. Simulated agents navigating with our Learned Subgoal Planner in real-world floor plans demonstrate a 21% expected decrease in cost-to-go compared to standard optimistic planning techniques that rely on Dijkstra's algorithm, and real-world agents show promising navigation performance as well.

Keywords: Navigation, POMDP, Deep Learning

1 Introduction

We aim to navigate partially-revealed environments in minimum distance. Imagine a robot tasked with navigating a large office complex with a partial map, discovering the environment as it travels. To reach its goal, the agent must plan to enter unknown parts of the map, and in so doing, reason about the cost of trajectories through unknown space. Many approaches to planning attempt to avoid the difficulties associated with reasoning about unknown space by optimistically assuming all unobserved space is free of obstacles [1, 2]. Yet if instructed to travel to a conference room on the far side of the building, for example, such an optimistic planner is likely to make globally suboptimal decisions and guide the robot into countless offices as it navigates towards its goal.

Planning more intelligently requires making inferences about environmental topology, so that likely dead-ends may be predicted and avoided. For example, offices in a building usually only have a single entrance and measure no more than a dozen meters in either direction, so any proposed motion plans that attempt to reach a faraway goal via an unexplored office should have a higher expected cost than those that plan to navigate through hallways. In general, navigating in a partially-revealed environment can be modeled as a Partially Observable Markov Decision Process [3] (POMDP), which can be leveraged to compute the expected minimum cost path. However, using this model in practice to evaluate the expected cost of an action, like choosing to go left or right at a fork-in-the-road, requires not only enormous computational effort, but also access to a distribution over possible environments. Obtaining such a distribution in a form that is useful for navigating unknown space is difficult in practice for complex environments.

There are a number of learning-based approaches for navigating in unknown environments [4, 5, 6]. The key limitation of many such approaches so far has been in their ability to learn policies that reason over long time horizons, since previous efforts have focused predominantly on short-horizon costs. Richter *et al.* [4, 7] showed an improvement over optimistic planning heuristics by directly predicting the expected cost over a family of motion primitives; the predictions were limited to the near-term horizon — only a few time-steps into the future — and were ineffective for predicting the long-term implications of an action in topologically complex environments. Another approach to

*G. J. Stein and C. Bradley contributed equally to this work.

2nd Conference on Robot Learning (CoRL 2018), Zürich, Switzerland.

trajectory optimization implicit in the POMDP is to learn a good policy using reinforcement learning. Kahn *et al.* [8] learn a policy for navigating an office-like environment, but the model horizon for their experiments is limited to 3.6 meters. While effective for avoiding obstacles, planning only a few meters ahead is insufficient for recognizing and avoiding the long-term implications of actions, which may impact navigation over hundreds of meters. Reinforcement learning has been shown to be successful for navigation in known environments [9] and for short range navigation in unknown environments [5, 6]. However, practical limitations associated with data complexity [10, 11] or delayed rewards [12] make it difficult for such agents to learn policies that can effectively navigate large-scale, unknown environments.

In order to more easily reason about the structure of an unobserved environment beyond the short term, we introduce an abstraction that allows us to make predictions in a computationally efficient manner. Specifically, we define a dynamic action set by associating a subgoal with each boundary between free and unknown space and defining high-level actions corresponding to traveling to the goal through each subgoal. Beginning with the Bellman Equation, we derive an algorithm that uses the subgoals to define a set of actions; doing so allows us to estimate the expected cost of navigating to the goal via one of the subgoals and to avoid the computational challenges of reasoning over individual trajectories or motion-primitives. Representing its possible paths to the goal, a robot must pass through one of the subgoals to enter unknown space and reach the unseen goal. The set of dynamic actions (subgoals) is computed on-line and updates as the robot explores the environment. Key to our approach is learning the properties of the unknown environment beyond each subgoal: e.g., the likelihood that attempting to reach the goal via a subgoal will ultimately lead to the goal or to a dead end. By learning these terms, our algorithm is able to incorporate prior information and estimate the long-time-horizon cost of our actions, while avoiding the computational costs usually incurred when planning under uncertainty.

Our approach enables experience-guided navigation of unknown environments in real-time on an autonomous platform. Moreover, we use classical planning techniques to navigate toward each subgoal. Our planner is therefore is guaranteed to reach the goal if a feasible path exists, even in unfamiliar environments where learning may not provide accurate predictions. We demonstrate the effectiveness of our technique in simulation by showing that the expected cost of using our planner for navigation is 21% lower on real-world floor plans as compared to a standard optimistic heuristic. We also include real-world experiments on an RC car and show promising performance consistent with the simulated results, thus demonstrating that our algorithm is suitable for operation on a real-world robot.

2 Planning with Subgoals

Our objective is to minimize the expected cost of navigating a static, unknown environment. As mentioned previously, navigating an unknown environment can be modeled as a Partially Observable Markov Decision Process. The expected cost of an action under the optimal policy can be computed recursively using the Bellman Equation:

$$Q(b_t, a_t \in \mathcal{A}(b_t)) = \sum_{b_{t+1}} P(b_{t+1}|b_t, a_t) \left[R(b_{t+1}, b_t, a_t) + \min_{a_{t+1} \in \mathcal{A}(b_{t+1})} Q(b_{t+1}, a_{t+1}) \right] \quad (1)$$

where $R(b_{t+1}, b_t, a_t)$ is the expected cost of reaching belief state b_{t+1} from b_t by taking action a_t. Our belief state is a two-element tuple consisting of the partially observed map m_t and the robot pose q_t: $b_t = \{m_t, q_t\}$ In practice, computing the expected cost via Eq. (1) is intractable, since it requires taking an expectation over the distribution of all possible maps.

To simplify the calculation of the expected cost, we introduce our Learned Subgoal Planning algorithm, which we use to evaluate the expected cost of selecting a motion plan that passes through a particular subgoal, each of which corresponds to a boundary between free and unknown space, and navigates the unknown environment towards the goal. At every time step, the subgoals define our set of actions: each action corresponds to traveling to the goal via the unknown space associated with a particular subgoal. The action loop for our subgoal planning agent is as follows: (1) we obtain the set of subgoals (this yields a set of actions $\mathcal{A}(b_t)$) given the belief b_t, (2) we compute the expected cost of selecting an action a_t, (3) we choose the subgoal q_g^* that minimizes the expected cost, (4) we compute a motion plan passing through subgoal q_g^*, and (5) we select a low-level motion primi-

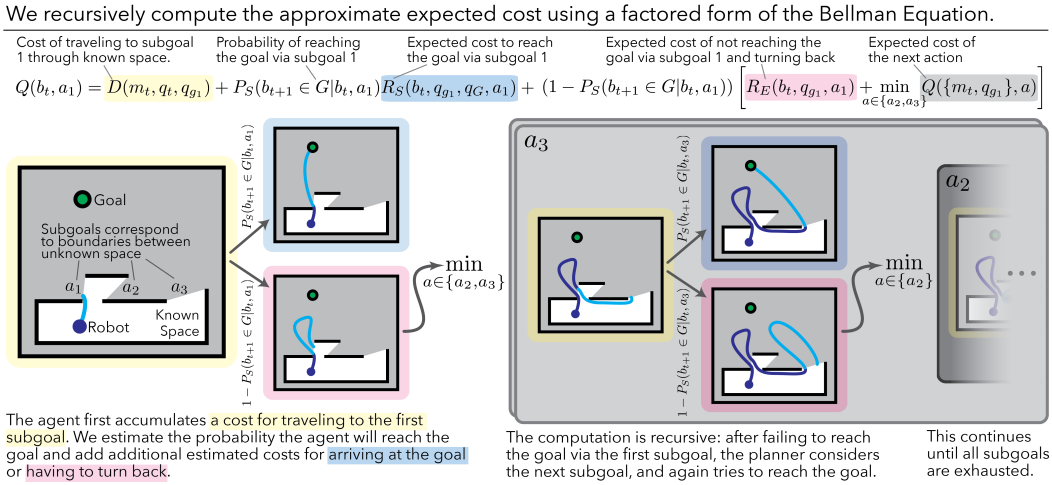

Figure 1: This schematic gives an overview of our subgoal planning algorithm, which allows us to plan through an unknown environment by computing the expected cost of each action in a way that is computationally feasible. Learning is used to estimate the terms P_S, R_S and R_E, thereby introducing prior knowledge about the environment class of interest into the decision-making procedure. This equation is derived and discussed further in Sec. 2.1.

tive that moves along the computed path. This repeats at each time step until the goal is within the revealed map, and the agent is able to plan to it.

2.1 Approximating Expected Cost via the Subgoal Planning Algorithm

We first factor the Bellman Equation according to our abstraction and introduce terms we later estimate with learning. Without loss of generality, we can split the future belief states b_{t+1} into two sets: future states in which the robot has reached the goal, which we write as $b_{t+1} \in G$, and future states in which it has not, $b_{t+1} \notin G$. Noting that the future cost is identically zero for states that reach the goal, we can eliminate $Q(b_{t+1}, a_{t+1})$ when $b_{t+1} \in G$ and rewrite Eq. (1) as follows:

$$\begin{aligned} Q(b_t, a_t \in \mathcal{A}(b_t)) = P_S(b_{t+1} \in G | b_t, a_t) \sum_{b_{t+1} \in G} P_G(b_{t+1} | b_t, a_t) R(b_{t+1}, b_t, a_t) + \\ (1 - P_S(b_{t+1} \in G | b_t, a_t)) \sum_{b_{t+1} \notin G} P_{\bar{G}}(b_{t+1} | b_t, a_t) \left[R(b_{t+1}, b_t, a_t) + \min_{a \in \mathcal{A}(b_{t+1})} Q(b_{t+1}, a) \right], \end{aligned} \tag{2}$$

where $P_S(b_{t+1} \in G | b_t, a_t) \equiv \sum_{b_{t+1} \in G} P(b_{t+1} | b_t, a_t)$ is the proportion of states in which the agent successfully reaches the goal after selecting action a_t from belief state b_t and $P_G(b_{t+1} | b_t, a_t) \equiv P(b_{t+1} | b_t, a_t) / P_S(b_{t+1} \in G | b_t, a_t)$ is normalized according to $\sum_{b_{t+1} \in G} P_G(b_{t+1} | b_t, a_t) = 1$, so that the first term in Eq. (2) can be thought of as the expected cost over states that reach the goal times the proportion of states that reach the goal (both of which, we later estimate using learning). $P_{\bar{G}}$ is defined similarly over states that do not reach the goal. Evaluating the recursive term in Eq. (2) remains particularly difficult to compute. We reduce this term with a simplifying assumption: that the expected cost of the future states can be approximated using the current observed map m_t, subject to the restriction that the set of future actions is simply the current set of actions minus the action that was just chosen: $\mathcal{A}(b_{t+1}) = \mathcal{A}(b_t) \setminus \{a_t\}$. This pair of assumptions makes recursively computing the expected cost of an action significantly easier, since the recursive term in the resulting approximate Bellman Equation only depends on the current observed map m_t:

$$\min_{a \in \mathcal{A}(b_{t+1})} Q(b_{t+1}, a) \approx \min_{a \in \mathcal{A}(b_t) \setminus a_t} Q(\{m_t, q_{t+1}\}, a), \tag{3}$$

where $\{m_t, q_{t+1}\}$ is a belief state consisting of the *current* observed map state m_t and the *next* robot pose q_{t+1}. To compute the expected cost-to-go at time t via our Subgoal Planning abstraction, we first compute the set of actions, $\mathcal{A}(b_t)$, which each correspond to planning to the goal through one of the available subgoals. Following Eqs. (2) & (3), and recognizing that we can use a Dijkstra's

algorithm heuristic D to navigate through known space, the expected cost can be written as,

$$Q(b_t, a_t) = \overbrace{D(m_t, q_t, q_g)}^{1.} + P_S(b_{t+1} \in G | b_t, a_t) \overbrace{R_S(b_t, q_g, q_G, a_t)}^{2.} + (1 - P_S(b_{t+1} \in G | b_t, a_t)) \left[\underbrace{R_E(b_t, q_g, a_t)}_{3.} + \min_{a \in \mathcal{A}(b_t) \setminus a_t} \underbrace{Q(\{m_t, q_g\}, a)}_{4.} \right], \tag{4}$$

where we define the following terms:

1. $D(m_t, q_t, q_g)$: the cost of traveling from q_t, the current location of the agent, to a subgoal q_g (as specified by an action a_t). Since this motion plan occurs entirely within known space, we use Dijkstra's algorithm as a heuristic to calculate this cost.
2. $R_S(b_t, q_g, q_G, a_t)$: the expected cost of success. Suppose the robot succeeds in reaching the goal via the subgoal specified by action a_t with a probability $P_S(b_{t+1} \in G | b_t, a_t)$. Then, with probability P_S, we accrue an expected cost of traveling from the subgoal q_g to the goal q_G via unknown space.
3. $R_E(b_t, q_g, a_t)$: the exploration cost. In the event that the agent does not reach the goal, we say that it instead *explores* the space beyond the subgoal of interest. With probability $1 - P_S$, the agent then accumulates an *exploration cost*, the expected cost of trying to navigate to the goal via a_t and returning to the subgoal q_g.
4. $Q(\{m_t, q_g\}, a)$: the future expected reward. Upon failing to reach the goal and exploring the chosen space beyond the subgoal q_g we consider a different action, $a \in \mathcal{A}(b_t) \setminus a_t$. We compute this final term recursively until all actions in $\mathcal{A}(b_t)$ have been exhausted.

An illustration of our procedure for a simple example can be found in Fig. 1.

2.2 Learning Properties of Unknown Space

The computation of a number of the terms in Eq. (4) — the probability of succeeding in reaching the goal P_S, the success cost R_S, the exploration cost R_E — are implicitly expectations over the future belief b_{t+1} and, as before, are computationally intractable. Rather than attempt to compute these terms, we learn them[1], allowing us to predict information about the unseen portions of the environment and incorporate this information into the calculation of the expected costs. These three terms are learned in a supervised manner: we train a fully-connected neural network to take as input (1) a single sensor observation, a 256-element vector of LIDAR range measurements, (2) the 2D robot-frame position of the goal, and (3) the robot-frame position of the subgoal, and predict the three terms specified above. As new observations are incorporated into the map and the set of subgoals changes — i.e., the boundary of observed space changes — any new subgoal is passed to the neural network, which then estimates the properties of the unknown space beyond that subgoal.

For all environments, the neural network is a fully connected neural network with hidden layers of width $\{256, 128, 48, 16\}$, each followed a batch normalization layer, 50% dropout, and a sigmoid activation function[2]. The output of the neural network is of dimension three, for each of the three desired outputs; a sigmoid activation and weighted cross-entropy loss is used to predict the success probability while a linear activation and a squared error loss is used for the two regression outputs. Loss for the success cost is only applied when a plan through a subgoal does lead to the goal and the loss for the exploration cost is only applied when it does not. One additional consideration is that the effective cost of misclassifying a subgoal strongly depends on a number of factors. For example,

[1] In fact, since using Dijkstra's algorithm to compute the distance between the goal q_G and each subgoal q_g provides a reasonable estimate of the success cost, we instead learn the *delta success cost*, the difference between the two: $\Delta R_S = R_S(b_t, a_t) - D(b_t, q_g, q_G)$.

[2] All networks were trained using the Adam optimizer in TensorFlow (default parameters) over 100k steps with a batch size of 256 and using roughly 300k training samples for each environment. The learning rate begins at 0.004 and decreases by a factor of 0.9 every 5k steps; the relatively fast learning rate decay and the dropout regularize the network and reduce overfitting. The performance is largely insensitive to changes in the random seed, adding additional layers, or changing the number of hidden nodes. We trained a few dozen different networks with variations on these parameters, and found that the standard deviation in the loss between these different networks was only 4.2%.

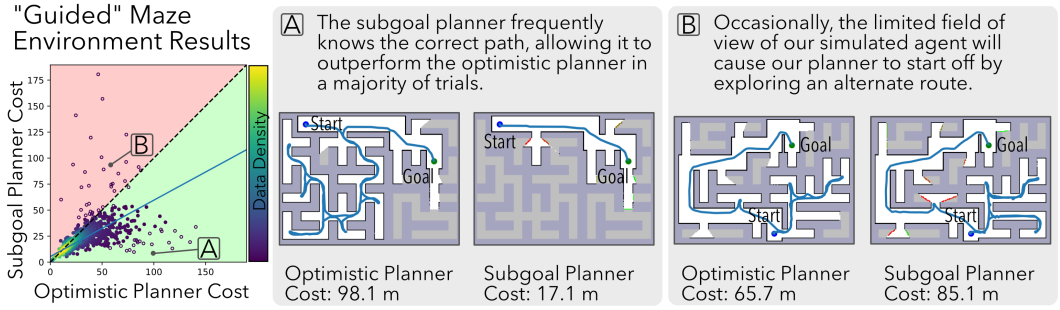

Figure 2: A comparison between the cost-to-go of the optimistic planner baseline and our subgoal planner for 1,200 simulated trials in the Guided Maze Environment. The blue line represents the calculated fit line of data points selected by RANSAC (filled circles). The Learned Subgoal Planning agent travels a total of 22.3% less distance than the optimistic baseline over these trials, which is shown in the scatter plot by the fit line being in the green colored region. In plots of the subgoal planner's path, boundaries associated with each subgoal are colored from green to red so as to visualize P_S, the estimated likelihood that it will lead to the goal.

if a subgoal constitutes the only path to the goal, then the penalty for misclassifying this subgoal is very high. By contrast, deciding to explore a small room near the goal will not likely take long and is therefore of relatively low cost. Computing the relative importance of correctly classifying a subgoal would require solving the full Bellman Equation and is therefore too expensive to compute. Instead, we provide a heuristic for computing this *misclassification cost* at training time. For subgoals that *do not* lead to the goal, misclassification results in the exploration cost R_E described above, which is roughly the cost of exploring the space beyond the chosen subgoal. For subgoals that *do* lead to the goal, the robot could travel across the entire map before returning; therefore, the penalty is the cost of traveling to the furthest point beyond a subgoal that does not lead to the goal and back. The computed costs are then used as weights for the cross-entropy loss used to train the classifier. Our heuristic matches intuition: subgoals that lead to the goal are naturally more important to correctly classify than those that do not. Finally, we also use *class reweighting* to compensate for the asymmetry in the proportion of subgoals that do and do not lead to the goal.

2.3 Training Data Generation

Since we learn the properties of the unknown space beyond each subgoal, we require training data generated from each environment class of interest. To obtain this data, we generate maps from the environment class of interest and have an agent navigate using the optimistic planner as if the map were unknown. As the agent explores, revealing new portions of the environment, we extract the newly updated subgoals. Using the underlying map, we determine if planning through each subgoal will lead to the goal and its cost of success (if the subgoal *does* lead to the goal) or its exploration cost (if the subgoal *does not* lead to the goal). Thus, each new subgoal corresponds to a single training datum. For each environment, we generate a few hundred maps and collect data from each run of the optimistic planner. The resulting data is then used to train the neural network described in the previous section. We note that the maps we use for training are distinct from those used for testing. Additionally, for our floor plan environment, the maps we use to generate our training data are from different buildings than the floor plans we use to evaluate performance, so that we may show that we learn generally informative features about human-structured environments instead of memorizing the environments themselves.

3 Simulation Results

To evaluate the performance of our approach to planning, we generate simulated occupancy grids from three different classes of environments: *"guided" mazes*, *random forests*, and *real-world floor plans*. We generate training data from these environments and train a neural network for each class, as described in Sec. 2.2 & 2.3. For each environment class, individual trials consist of our simulated RC Car agent navigating between a random start/goal location for each of the optimistic planner — which plans as if all unknown space is unoccupied — and Learned Subgoal Planner; the length of the resulting trajectories are compared.

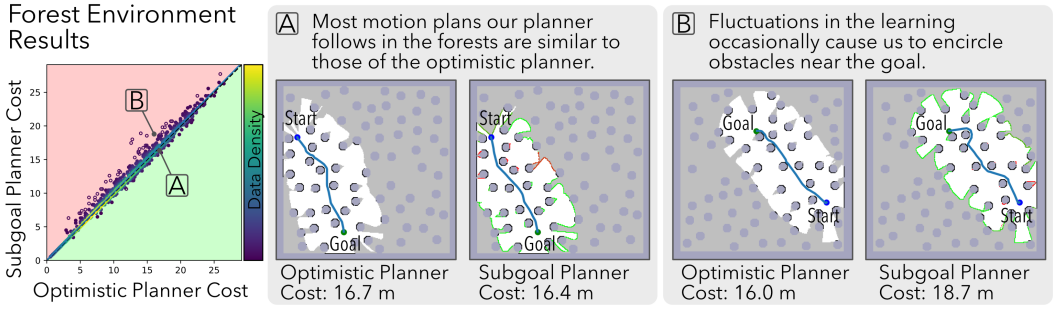

Figure 3: A comparison between the cost-to-go of the optimistic planner baseline and our Learned Subgoal Planner for 1,000 simulated trials in the Forest Environment. Our agent closely matches the performance of the optimistic baseline, which already achieves near-optimal navigation.

3.1 "Guided" Maze Navigation Results

We generate maze environments using *Kruskal's Algorithm* [13]. We use the term *guided* to mean that the maze is designed in such a way that local features give immediate guidance on which way to go: the hallways along the most-direct path between the start and the goal are twice as wide as any other branches of the maze. The Learned Subgoal Planner should therefore be able to use individual laser scans to estimate properties of the unknown space beyond each subgoal with reasonable reliability. We evaluate navigation performance of the simulated agents through 1,200 runs in our synthetic maze environment, and include a scatter plot in Fig. 2 that gives the net distance traveled of each trial by both the optimistic planner baseline and our subgoal planner.

The net cost incurred by the robot summed over all trials (each from start to goal) shown in Fig. 2 is 22.3% lower for our subgoal planner than for the optimistic planner. Furthermore, fitting a line to 95% of the data — as selected by RANSAC — compares the incurred cost of our planner to that of the optimistic planner, yielding a cost ratio of 0.570: the expected cost of the motion plan executed by our subgoal planner is 43% lower than that of the optimistic planner for the same environment configuration. Fig. 2A shows one example where the subgoal planner, using local features to estimate the expected cost of each subgoal, navigates directly to the goal, yet the optimistic planner unnecessarily explores much of the environment before eventually reaching the goal.

There are a handful of cases in which the optimistic planner outperforms the Learned Subgoal Planner. Such trials typically result from errors in the estimation near the start of motion: if our planner incorrectly predicts that the subgoal leading to the goal is a dead end, it will navigate a portion of the remaining environment before returning to correct its mistake. We show one such example in Fig. 2B. These cases are uncommon, however, (roughly 2.5% of results) and are outweighed by the trials in which our planner outperforms the optimistic planner.

3.2 Forest Navigation Results

Having shown that our algorithm is capable of using local features to aid navigation, we also demonstrate that the Learned Subgoal Planner does not appreciably lower performance in environments were the optimistic planner is already close to the best policy. We navigate in a set of random forests, generated in such a way that gaps between each individual cylinder is large enough to allow the agent to pass between them. We conducted 1,000 trials in the random forest environments and plot the results in Fig. 3, which shows a good agreement between the cost of the motion plans from the optimistic planner and our subgoal planner. Fitting a line to 95% of the data — as selected by RANSAC — yields a cost ratio of 1.006 between the planners, confirming the good agreement. In addition, the cost of using our subgoal planner summed over all trials is only 2% higher than the net cost from the optimistic planner, suggesting that using our subgoal planner does not substantively lower performance in environments for which the optimistic planner is already very effective. It is worth noting that outlying points, those visibly above the line, are uncommon and have only a small impact on the overall navigation performance; such points, such as the one shown in Fig. 3B, frequently occur near the goal, when small fluctuations in the learning output may cause our planner to change directions while encircling a cylindrical object. Other small deviations, as in Fig. 3A, are caused mostly by luck, when the two motion plans deviate around an obstacle.

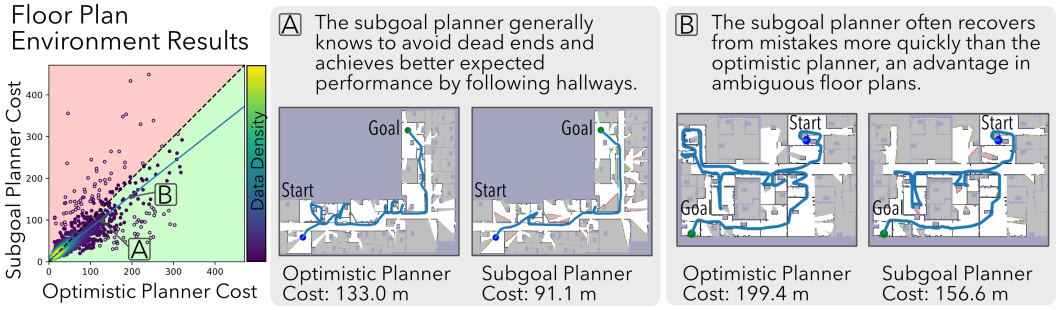

Figure 4: A comparison between the cost-to-go of our Learned Subgoal Planner and the optimistic planner baseline for 2,000 simulated trials in the Floor Plan Environment. The subgoal planning agent travels a total of 12.5% less distance than the optimistic baseline.

3.3 Floor Plan Environment Navigation Results

Finally, we conducted trials in the *floor plan environment*: maps of our floor plan environment are occupancy grids generated from blueprints of real-world buildings around the MIT campus with obstacles added at random, so as to mimic real-world furniture and clutter. A scatter plot showing the results of these trials can be found in Fig. 4. Our Learned Subgoal Planner achieves better performance in expectation for the floor plan environments. The net cost over all runs is 12.5% lower for the subgoal planner than for the optimistic planner. We again fit a line to 95% of the data, as selected by RANSAC; the cost ratio is 0.784, suggesting that the expected cost of a plan using the Learned Subgoal Planner is 21% shorter than that of the optimistic planner.

In Fig. 4A, we show a typical sample from the navigation trials, in which our Learned Subgoal Planner avoids many dead ends — which take the form of offices and lab spaces — that lead the optimistic planner astray. In most floor plan maps, hallways define the most likely routes between faraway start and goal locations; thus, subgoals that lead down hallways are frequently of lowest expected cost. Most outliers in Fig. 4 occur when the two planners travel in different directions at a branching point in the environment. Relatedly, we highlight in Fig. 4B one case in which following the hallway does not lead to the goal. After traveling much of the hallway, the subgoal planning agent recovers and turns back to seek more probable route to the goal. It reaches the goal 22% faster than the the optimistic agent, which goes on to explore the upper-left portion of the map.

4 Real-world Experiments

Finally, we integrated our Learned Subgoal Planner with a physical platform so as to observe its performance in the real world. For our experimental platform, we used a RC car [14] with a Hokuyo Lidar [15] and a Microstrain IMU [16]. The Robot Operating System was used for interprocess communication, run on a NUC computer [17]. The Octomap package [18] was used to build a 2D occupancy grid from the incoming laser scans and the robot pose, provided by an Extended Kalman Filter that fuses the output from a laser scan matcher and IMU measurements. Rather than collect a large-volume of real-world data, we instead trained in simulated floor-plan environments with a realistic distribution of open doorways. Using this data was a natural choice, since the real-world environments we use for testing are similar in structure to the floor plans.

We focused on locations known to cause problems for optimistic approaches to navigating unknown spaces, particularly hallways with classrooms or other rooms. As in our simulation results, agents using an optimistic planner frequently entered rooms, even when the goal was sufficiently far away that reaching the goal via entering the room was unlikely. In three distinct, real-world environments, we observed similar behavior: the subgoal planning agent would actively avoid unnecessarily entering dead-ends — including classrooms, a lecture hall, an apartment, and numerous closets — which the optimistic planner would enter. Yet if the goal was placed inside these regions, the subgoal planner would know to enter them, showing that the agent knew more than to simply follow hallway-like features. We highlight one of our real-world experiments in Fig. 5, in which the agent demonstrates the aforementioned behavior in an apartment complex. By avoiding the open rooms and dead-ends, the agent running our planner reached faraway goals more quickly than the optimistic planner, thus corroborating our simulation results with a real-world platform.

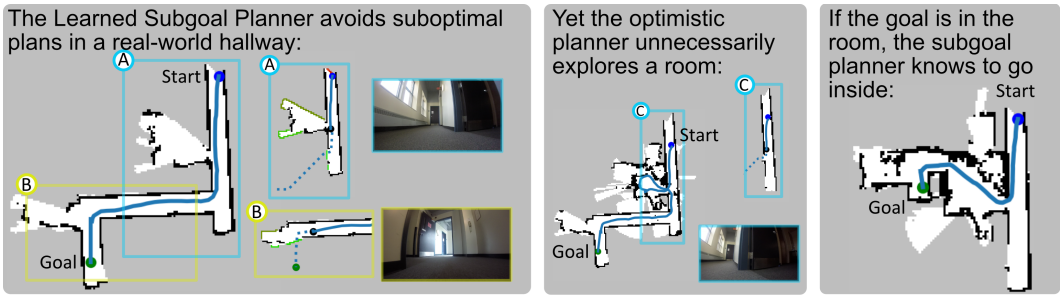

Figure 5: This figure shows an experimental run on our RC Car, in which our Learned Subgoal Planner (left) outperforms the optimistic planner (center) in navigating to an unseen goal. In (A), we observe the subgoal planner choose to pass the room and continue down the hallway. By contrast, the optimistic planner enters the room (C). On the far right, our Learned Subgoal Planner knows to enter the room when the goal is placed in a position where the lowest expected cost path is to go inside.

5 Related Work

Work on solving navigation and exploration tasks in uncertain environments using POMDPs dates to the mid-1990's [3, 19, 20]. Some recent work has instead focused on using a dynamic action set [21] or boundaries between free and unknown space [22, 23] for navigation/exploration of unknown environments, though none incorporate learning. Karaman and Frazzoli [24] navigate unknown random forest environments, yet make strong assumptions about the environment distribution. If the distribution over states is not known, learning is required to estimate the expected cost [7]. Most literature that uses learning to explore unknown environments focuses on short-time-horizon planning. Richter *et al.* [4] learn to solve an approximate POMDP for navigation of unknown environments, yet restrict their planning horizon to only a few time-steps. The deep reinforcement learning (Deep RL) community has been making rapid progress towards model-free navigation of unknown environments [5, 25, 6, 26, 27], yet most such research focuses on small environments or, as in Kahn *et al.* [8], use time-horizons insufficient for making reasoned decisions in environments of our size, which can require recall of information over thousands of steps. The MERLIN agent [11] uses a differentiable neural computer to tackle maze navigation and other goal-oriented tasks, which enables storage and recall of information over much longer time-horizons than is typically realizable by standard "end-to-end" Deep RL systems. However, their approach requires orders of magnitude more data than is used for our approach (billions of samples versus hundreds-of-thousands) and on environments considerably smaller than ours, owing to the difficulties associated with model-free reinforcement learning with delayed rewards [10, 12]. Like other Deep RL agents, they also lack guarantees their agent will ultimately succeed at the assigned task.

6 Conclusion and Future Work

In this work, we present Learned Subgoal Planning as an approach to use subgoals, each corresponding to a boundary between free and unknown space, and an accompanying factorization of the Bellman Equation for evaluating the expected cost of navigating through unknown space to reach an unseen goal in partially-revealed environments. With this planning framework, we can incorporate prior information into our decision making process in a way that deterministically reaches the goal and is computationally tractable. We show that our Learned Subgoal Planner is capable of making informed decisions in both simulated environments and on real-world hardware. One potential avenue to improve our work is augmenting the learning pipeline to use more than a single sensor observation, which could allow the system to update the properties of faraway subgoals as the robot explores. Our representation is also not without limitations. We make an independence assumption between our actions so that our model does not capture the impact of information gathering actions. Second, our neural network model returns only the maximum likelihood value for each learned subgoal property: our model may have trouble when higher-order moments of the distribution matter a great deal. Despite these limitations, our planner demonstrates reductions of over 20% in the expected cost-to-go in simulated floor plan environments. Furthermore, we demonstrate promising results on a real-world platform, showing that our Learned Subgoal Planning approach is a practical method for improving autonomous navigation of structured unknown environments.

Acknowledgments

This work was supported by the Office of Naval Research and Marc Steinberg under the PERISCOPE MURI Contract #N00014-17-1-2699. G. J. Stein acknowledges support by a NDSEG Graduate Fellowship. Their support is gratefully acknowledged.

References

[1] A. Bachrach. *Trajectory Bundle Estimation For Perception-Driven Planning*. PhD thesis, Massachusetts Institute of Technology, 2017.

[2] T. Fraichard. A short paper about motion safety. *International Conference on Robotics and Automation (ICRA)*, 2007.

[3] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998.

[4] C. Richter, J. Ware, and N. Roy. High-speed autonomous navigation of unknown environments using learned probabilities of collision. In *International Conference on Robotics and Automation (ICRA)*, 2014.

[5] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik. Cognitive mapping and planning for visual navigation. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[6] J. Zhang, J. T. Springenberg, J. Boedecker, and W. Burgard. Deep reinforcement learning with successor features for navigation across similar environments. In *International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[7] C. Richter. *Autonomous navigation in unknown environments using machine learning*. PhD thesis, Massachusetts Institute of Technology, 2017.

[8] G. Kahn, A. Villaflor, B. Ding, P. Abbeel, and S. Levine. Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation. In *International Conference in Robotics and Automation (ICRA)*, 2018.

[9] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *International Conference on Robotics and Automation (ICRA)*, 2017.

[10] M. Deisenroth and C. E. Rasmussen. Pilco: A model-based and data efficient approach to policy search. *International Conference on Machine Learning (ICML)*.

[11] G. Wayne, C. Hung, D. Amos, M. Mirza, A. Ahuja, A. Grabska-Barwinska, J. W. Rae, P. Mirowski, J. Z. Leibo, A. Santoro, M. Gemici, M. Reynolds, T. Harley, J. Abramson, S. Mohamed, D. J. Rezende, D. Saxton, A. Cain, C. Hillier, D. Silver, K. Kavukcuoglu, M. Botvinick, D. Hassabis, and T. P. Lillicrap. Unsupervised predictive memory in a goal-directed agent. *CoRR*, abs/1803.10760, 2018.

[12] J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *International Journal of Robotics Research*, 2013.

[13] *Introduction to Algorithms, Third Edition*. MIT Press, 2009.

[14] RACECAR/J. RACECAR/J Platform. https://racecarj.com/, 2018.

[15] Hokuyo. Hokuyo UTM-30LX Scanning Laser Rangefinder. https://www.hokuyo-aut.jp/search/single.php?serial=169, 2018.

[16] L. MicroStrain. 3DM-GX4-45 Industrial-Grade All-In-One Navigation Solution. https://www.microstrain.com/inertial/3dm-gx4-45.

[17] Intel. NUC 7 Enthusiast Mini PC. https://www.intel.com/content/www/us/en/products/boards-kits/nuc/mini-pcs/nuc7i7bnkq.html,.

[18] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013.

[19] M. L. Littman, A. R. Cassandra, and L. P. Kaelbling. Learning policies for partially observable environments: Scaling up. In A. Prieditis and S. Russell, editors, *Machine Learning Proceedings 1995*, pages 362 – 370. 1995.

[20] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT press, 2005.

[21] N. Roy and C. Earnest. Dynamic action spaces for information gain maximization in search and exploration. In *American Control Conference (ACC)*, Minneapolis, MN, 2006.

[22] I. Arvanitakis, K. Giannousakis, and A. Tzes. Mobile robot navigation in unknown environment based on exploration principles. In *Conference on Control Applications (CCA)*, Sept 2016.

[23] R. Nasir and A. Elnagar. Gap navigation trees for discovering unknown environments. *Intelligent Control and Automation*, 6(04):229, 2015.

[24] S. Karaman and E. Frazzoli. High-speed flight in an ergodic forest. *International Conference on Robotics and Automation (ICRA)*, 2012.

[25] L. Tai, G. Paolo, and M. Liu. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In *International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[26] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, et al. Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*, 2016.

[27] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel. RL^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.