NavTuner: Learning a Scene-Sensitive Family of Navigation Policies

Haoxin Ma[1], Justin S. Smith[2], and Patricio A. Vela[2]

***Abstract*— The advent of deep learning has inspired research into end-to-end learning for a variety of problem domains in robotics. For navigation, the resulting methods may not have the generalization properties desired let alone match the performance of traditional methods. Instead of learning a navigation policy, we explore learning an adaptive policy in the parameter space of an existing navigation module. Having adaptive parameters provides the navigation module with a family of policies that can be dynamically reconfigured based on the local scene structure and addresses the common assertion in machine learning that engineered solutions are inflexible. Of the methods tested, reinforcement learning (RL) is shown to provide a significant performance boost to a modern navigation method through reduced sensitivity of its success rate to environmental clutter. The outcomes indicate that RL as a meta-policy learner, or dynamic parameter tuner, effectively robustifies algorithms sensitive to external, measurable nuisance factors.**

I. Introduction

Autonomous navigation through static, unstructured environments has advanced in the past decades but fundamentally still relies on engineered approaches [1], [2]. Given an approximate map, the approaches use sensor data to inform updated estimates of the environment which are used to evaluate future trajectories in terms of safety and other characteristics with the aim of finding a collision-free, goal-attaining path. Traditionally designed systems involve manual parameter selection for general purpose navigation, which exhibits sensitivity to environmental conditions.

This paper investigates the use of machine learning to dynamically reconfigure the parameters of a hierarchical navigation system according to the immediate, sensed surroundings of the robot. We show that scene-dependent online tuning improves navigation performance and reduces sensitivity to environmental conditions. The final reinforcement learning solution, called *NavTuner*, addresses the problem of parameter sensitivity to operational variance. *NavTuner* is publicly available as open source [3].

A. Research Context

1) Navigation and Machine Learning: One candidate approach to learning and navigation is to replace the traditionally engineered system with an end-to-end sensor to decision neural network [4]–[7]. Empirical and limited benchmarking show some promise on this front. However, instead of directly solving the navigation problem itself, these methods solve some highly specific subset of it, typically equivalent to the local navigation problem. All argue for the need to integrate with existing hierarchical schemes, though few actually do so. Both [4], [8] show that reinforcement learning or reward-based approaches for local planning in a local-global hierarchical planner can work. Instead of mapping sensor input to navigation decisions directly, other learning based methods seek to map the sensor data to higher level information for decision making. Inverse reinforcement learning has been used to learn a reward function from RGB-D features and goal-directed trajectories [9]. A neural SLAM model has learned to output necessary information for global and local policies to control a robot [10]. Learning-based perception can be combined with model-based control methods to navigate in partially observable, unknown environments [11].

Overall, there is no substantive benchmarking of learning based methods with traditional navigation schemes [7]. Thus, the assertions that learning can overcome sensitivity to environmental conditions and can outperform traditionally engineered systems remain unconfirmed. As a preliminary investigation, we implemented a couple methods and benchmarked them in simulated ROS/Gazebo environments, see Figure 1 (and §IV-A for more details). When compared to a traditional hierarchical navigation approach [2], learning-based navigation methods have lower performance and equivalent or higher sensitivity to the environment. The sensitivity can be seen by the drop in performance (e.g., higher slope) as the environment becomes denser in the success rate vs obstacle quantity graphs of the second row. The variable performance across environments indicates poor generalization to world structure by the learning methods.

2) Reinforcement Learning-Image to Action: Tasks involving perception to action pipelines, such as visual navigation, can be implemented using end-to-end reinforcement learning (RL) based on deep convolutional neural network policy learners [12]–[16]. This is perhaps the most pervasive use of RL in vision-based navigation contexts. The learnt policy directly maps visual sensor input (laserscan, RGB image, or depth map) to actions, steering commands, or velocities [13], [17], [18]. Policy learning can be improved by using knowledge from previously learnt visual navigation tasks when learning new tasks [18]. Adding decision relevant auxiliary tasks can promote learning internal representations that support navigation [19] and improve the sample efficiency of RL while speeding up the training process [20]. In a similar vein, providing mid-level visual cues improves the learning, generalization, and performance of policies [21]. Also, learning subroutines instead of individual

*This work supported in part by NSF Award #1849333.

[1]H. Ma is with College of Computing, Georgia Institute of Technology, Atlanta, GA 30308, USA. haoxin.m@gatech.edu

[2]J.S. Smith, and P.A. Vela are with the School of Electrical and Computer Engineering and the Institute for Robotics and Intelligent Machines, Georgia Institute of Technology, Atlanta, GA 30308, USA. {jssmith, pvela}@gatech.edu

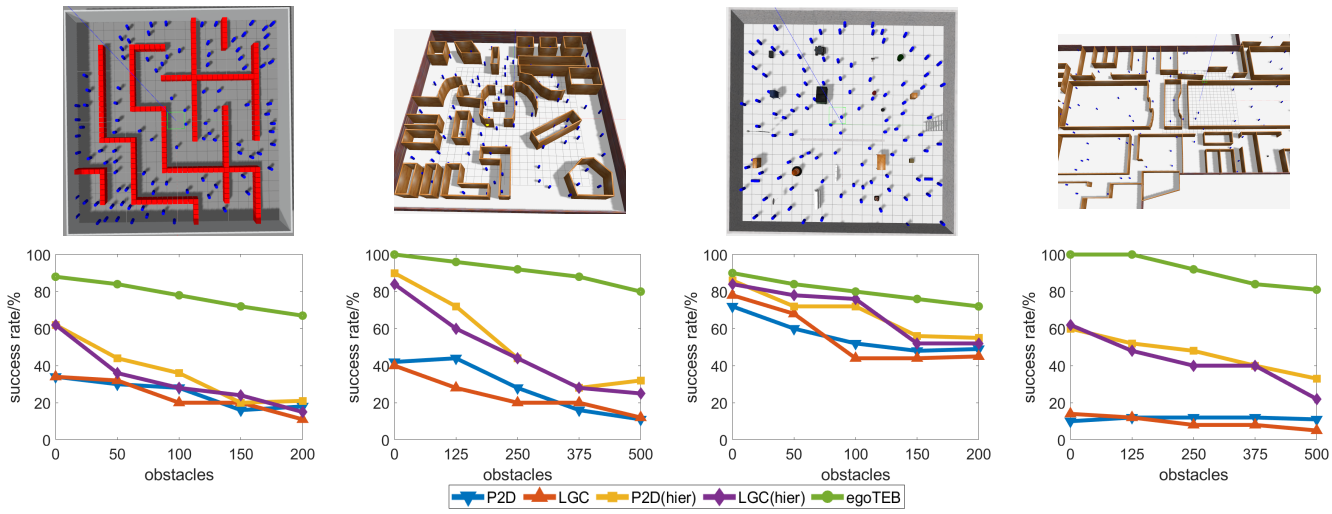

Fig. 1: Benchmark Worlds and Results. Top row: Depictions of the environments Maze, Campus, Sector, and Office with 100 obstacles. Bottom row: Navigation success rate versus obstacle count plots for egoTEB [2] and methods from [4], [5].

actions can boost the performance [22]. While navigation is a good task to demonstrate RL methods, in many cases the task focuses on navigation guidance more so than collision avoidance. Evaluation does not focus on embodied navigation nor realistic motion and collision models. Application of RL to robotics places less emphasis on the policy learner and more on the structure of the learning network [17], [23] and on the use of existing navigation methods to generate policy samples [5], [23], [24].

3) Vision-Based Navigation: Typical solutions for vision-based navigation rely on a combination of path planning and sensor-based world modeling to synthesize local paths through the world based on recovered structure. The most effective policies are hierarchical, with a *global planner* establishing potentially feasible paths based on an estimated map and a *local planner* following the global path as closely as possible, subject to collision-avoidance constraints in response to sensed obstacles missing from the map [25]–[27]. Though exhibiting strong performance in Figure 1, the variable maximum success rates and downward trends of egoTEB [2] across the graphs indicates that traditional methods are not immune to variation as induced by world structure. There is potential value in modifying them based on the local obstacle configuration space, i.e., in performing online parameter tuning for navigation.

4) (Hyper)-Parameter Tuning: The potentially negative impact of incorrect hyper-parameters or parameters is well established in machine learning. Hyper-parameter optimization improves learning outcomes without requiring major modifications to the underlying learning structure [28]–[30]. When feasible to implement, the additional computation needed is offset by the performance enhancement. Given that RL policies can exhibit high variability to the training process and parameter settings, the process of gradient-free hyper-parameter and reward tuning is recommended when the additional computational resources are available [31].

There is prior work on automatic parameter tuning applied to motion planning and navigation. Motion planning algorithms can perform better with random or Bayesian based [32] and model based [33] automatic parameter tuning algorithms. The benefits also apply to safety constraints [34]. Parameter tuning can also be incorporated into the learning process of a learning-based motion planning algorithm to reduce sensitivity to manual tuning [35]. Adaptive planner parameters of a planner can be learnt through intervention [36], demonstration [37], and reinforcement learning [38].

II. Scope of Investigation

This paper explores the performance impact of online tuning for traditional navigation strategies. Per Figure 1 and [36]–[38], they exhibit sensitivity to external world structure. Some of the sensitivity is a function of manually specified algorithm constants (i.e., parameters) that do not generalize well to different free-space circumstances. The objective is to reduce system sensitivity and improve performance outcomes during deployment by learning a more optimal tuning policy for the navigation parameters as a function of local geometry.

A. Hierarchical Navigation

The system to study is a hierarchical, vision-based navigation system consisting of two distinct spatial-and time-scales, *global* and *local*, with planning modules associated to each scale. The global planner computes a candidate path connecting the robot's current pose to the goal pose based on the current map. The local planner uses the global path to generate a series of sub-goals that should be feasible to sequentially achieve. Since the environment is unknown or uncertain, the local planner solves the current sub-goal based on a local map informed by the real-time integration of sensory information regarding the structure of the local environment. Its spatial and temporal scale is sufficiently small that real-time processing of the sensory data and short-time trajectory synthesis with collision checking and trajectory scoring is achievable. During navigation, information flow between the two planners influences the paths planned and

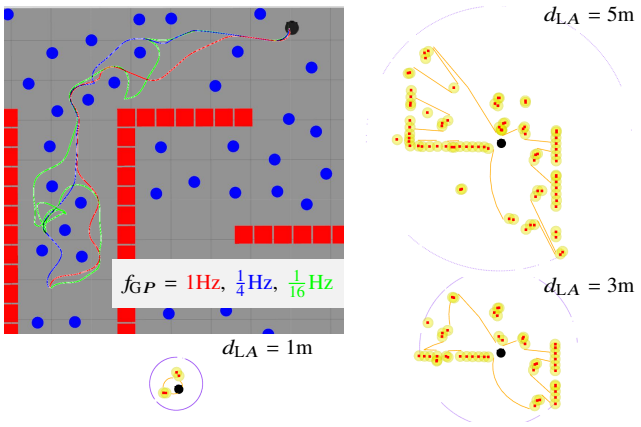

Fig. 2: Changing planning frequency f_{GP} impacts path taken (top-left). Changing look ahead distance d_{LA} impacts candidate navigable gaps found (orange curves).

the paths taken. Parameters internal to the global and local planners further influence their outputs. The parameters and the interacting modules should work to achieve collision-free navigation to the goal point while taking the shortest path possible given *a priori* known and *a posteriori* sensed information about the world. Here, the reference hierarchical planning system will utilize the *egoTEB* local planner [39], and the robot will be the differential drive Turtlebot.

B. Navigation Parameters

To illustrate the sensitivity of navigation performance to internal parameters, this section describes the impact of two: the *global planning frequency* and the local planner *look ahead distance*.

1) Global Planning Frequency (f_{GP}): The global planner recomputes the global path at a specified frequency to refresh its estimated best path based on data accumulated during navigation. Additionally, there are special events that trigger new plans (such as arriving at a dead-end). The tunable global planning frequency is upper bounded by the local planning frequency (or sensing rate) and the rate at which new global paths are generated. Other environmental events (not part of the detected special events) impact the need for new global plans. Sensory information accumulating in the local map may indicate that the current global path is no longer feasible and should be recomputed or that an unavailable path option may now be feasible and should be considered.

The effect of changing f_{GP} for a given environment and navigation goal is visually exemplified in Figure 2 (top-left). Table I quantifies its influence. The table reports the difference between the best performing and the worst performing outcomes in the *Maze* environment as a function of f_{GP} across the different obstacle densities tested (50 repeated trials each). The difference is a measure of parameter sensitivity to environment changes. SR stands for success rate and PL stands for path length. In the environment with the most obstacles (least spacing) there are almost 20 more failures cases for the worst f_{GP} setting versus the best. The path length increases by 7.68m which is roughly 14% of the average path length.

TABLE I: Performance changes vs. inter-obstacle spacing.

Spacing	f_{GP}		d_{LA}	
	ΔSR (%)	ΔPL (m)	ΔSR (%)	ΔPL (m)
0.75	19	7.68	8	8.65
1.0	8	1.56	5	2.93
1.25	4	1.04	7	2.34
1.5	5	0.48	6	1.8

2) Look Ahead Distance (d_{LA}): The source of local obstacle information for egoTEB is the *egocircle*. The egocircle populates an egocentric polar data structure with sensed obstacle points over time and uses it to synthesize a 1D laser scan of the local environment. This *egocircle scan* is processed by egoTEB to establish navigation gaps between obstacles through which the robot may traverse. The gaps represent different path opportunities to take to the goal state.

Gap processing uses d_{LA} to define a look ahead distance cut-off when detecting gaps. Measurements beyond d_{LA} are "ignored" in simulation of a shorter distance scanner. Local paths generated as part of egoTEB will not extend beyond this radius. Since only nearby obstacles are considered, lowering d_{LA} limits the quantity of gaps detected and candidate paths generated and tested. Figure 2 depicts the local map information and the detected gaps based on three different d_{LA} values. The parameter determines the spatial extent of the local map, which ultimately influences several parts of the local navigation strategy. The impact of d_{LA} on navigation is evident in Table I.

C. Scene Adaptive Policy Models

Learning a dynamic tuning policy can be done through a variety of mechanisms, with one main difference being batch learning versus reinforcement learning (RL). Both are explored and evaluated in this study. Batch learning requires sampling the input and output space to generate the training data. In contrast, RL involves rollouts that test output values chosen according to some training policy and record the measured input values to generate the training data. The input data is the egocircle scan and the output data is the parameter choices. The discretization for d_{LA} ranges over $[1.0, 5.5]$ meters in 0.5 increments (10 total), and for f_{GP} ranges over $[1/16, 1]$ Hz in factor of 2 increments (5 total).

1) Batch Learning: Several network models were chosen for the batch learning approach. They include: a linear network, a neural network (w/ReLu), and a convolutional neural network (w/ReLu). Two output structures are tested for each: a classifier structure over discretized values and a regression model for continuous prediction (trained with the discretized values). Training is performed by collecting the performance statistics of constant parameter value runs across a sweep of each discretized parameter individually.

2) Reinforcement Learning: The reinforcement learning model is a deep Q network (DQN) [40] with an action branching structure for multi-dimensional output [41], depicted in Figure 3. The reward structure uses knowledge

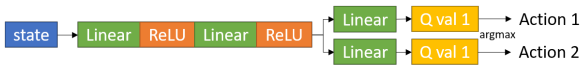

Fig. 3: Structure of a Double Action DQN (2D-DQN).

of the shortest path between the start and goal. For each training run the reference shortest path is obtained from the D^*-Lite algorithm [42] (other global planners would work). During navigation, the reward is the negative of the distance between the agent's current position and the nearest point on the shortest path. This reward is passed to the agent every two seconds for a policy update. The agent also updates at the end of a training run with the following terminal reward:

$$R_{goal} = \begin{cases} 1000(l/l_{min}) & \text{if goal attained} \\ -1000 & \text{otherwise} \end{cases}, \qquad (1)$$

based on the the Success Weighted by Path Length [43], where l and l_{min} are the lengths of the actual path taken by the robot and the reference shortest path, respectively.

III. Experiments and Methodology

All training methods use data generated from Gazebo simulations of robot navigation engagements in unknown environments. The robot used is a differential drive Turtlebot equipped with a Microsoft Kinect sensor ($60^\circ \times 45^\circ$ FoV). Depth images from the sensor are converted into laser scans that update the egocircle. The Maze environment in Figure 1 is used for training while the other environments are used for testing. For pose information, the true robot pose is accessed from Gazebo.

A. Experimental Methodology

1) The Maze Environment: To define navigation scenarios for which there is always a valid path between any random start and goal poses, we designed a maze environment consisting of maze walls placed within a $20m \times 20m$ square room. The maze walls are composed of $0.5m \times 0.5m$ blocks placed on a Manhattan grid within the maze free space. Several different wall configurations for the mazes exist. The occupancy maps of these different mazes are used as the initial global occupancy map for the robot.

During a trial, the initial map provided for the maze is incomplete because of the randomized placement of obstacle cyclinders with a radius of $0.15m$ within the world. There are two random placement strategies: uniform density and non-uniform density. A non-uniform placement density is achieved by dividing the environment into 5×5 subregions and assigning each region a random uniform density. The density specification for a maze region is given by a minimum inter-obstacle distance from the discrete set $\{0.75, 1.0, 1.25, 1.5\}\, m$.

2) Batch Data Collection: To generate training data for the supervised methods, we perform a single coordinate parameter sweep over the uniform density and robot configuration and run each configuration 50 times with valid random start and goal poses. The result is a total of $50 \times 5 \times 4 = 1000$ runs for f_{GP} and $50 \times 10 \times 4 = 2000$ runs for d_{LA}. Data recorded from each run includes all of the egocircle scans, whether it succeeded, the total path length, and the runtime.

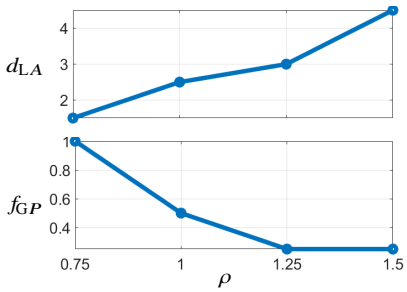

Fig. 4: Best hyper-parameter versus obstacle density ρ.

The binary success data provides a success rate for the navigation parameter and density pairing combination. For each density we choose the hyper-parameter value that maximizes the success rate. The average path length is used as a tie breaker (then the average runtime in the case of another tie). The best value versus the density ρ defines the selection functions $f_{GP}(\rho)$ and $d_{LA}(\rho)$, which are depicted in Fig. 4.

3) Training the Neural Networks: The classifiers are trained with cross entropy loss and the regressors are trained with mean squared error loss, both using the SGD optimizer. The training data consists of the known densities of the mazes, the best hyper-parameter values for said densities, and the recorded laser scan readings from the navigation scenarios.

Since training for reinforcement learning requires evaluating rollouts, it involves a different process. The DQN is trained in a non-uniform environment using ϵ-greedy with the reward function described in Section II-C.2, either from scratch or with a warm start using the collected batch data.

The model trained from scratch is trained for 1000 runs, and the warm started one is trained for 800 runs after first training with behavior cloning using the chosen best hyperparams for 200 runs. We compare the two models to see how previous knowledge acquired from experiments in uniform environments influences model performance.

B. Testing and Evaluation

The evaluation experiments are all done in non-uniform environments with the egoTEB planner set for dynamic parameters. Every two seconds the parameter prediction model predicts the optimal parameters and applies them. The baseline implementation of egoTEB uses the fixed parameters $d_{LA} = 3m$ and $f_{GP} = 1$Hz. The trained NavTuner is directly evaluated in unseen new environments, without any finetuning or additional online learning.

1) Testing Environments: Experiments are conducted in the four environments: maze, sector (dense), campus (dense), and office (dense). The last three are benchmark environments from [39]. The maze environment has two sets of evaluation experiments: with the same maze and with a maze different from the training maze. Changing the maze tests whether the network models have learned the specific maze wall setup instead of the local obstacle distribution.

To vary from the training data, we generate 4 different types of random obstacles, including 0.3m×0.3m and 0.15m×0.15m boxes and cylinders with diameters of 0.3m and 0.1m. The maximum number of obstacles is 200 for the maze and sector environments and 500 for the campus and office environments. In each environment, we run the 5 experiment configurations, consisting of 0, 25%, 50%, 75%, and 100% of the maximum number of obstacles.

2) Evaluation Criteria: For each run, the performance data collected is the result, the path length, and the runtime. For the maze environment, we also run experiments with an oracle version in which the parameters are updated using the best value curves (Figure 4) and the known density.

IV. Results and Analysis

This section reports and analyzes the results of several experiments. Of the metrics recorded, the success rate exhibited the most differences across the techniques, thus the discussion and analysis will revolve around this quantity. The experiments performed include (a) comparative performance of egoTEB versus end-to-end learning schemes, (b) a first pass evaluation of outcomes for the maze environment and a single parameter, (c) a more complete evaluation across environments, and (d) the extension of the best performing method to more parameters. Lastly, there is a comparative discussion regarding contemporary works with similar aims.

A. Comparison with End-to-End Learning

This experiment whose outcomes are in Figure 1 compares the fixed parameter egoTEB to Perception-to-Decision (P2D) [5] and to the local goal classifier (LGC) [4]. A third method, IntentionNet [6] was implemented to the best of our ability (including communication with the authors), but it would not provide good results in the benchmark environments. Thus, it is not included in the plots. The two methods implemented were run in an end-to-end manner, as well as within the same hierarchical navigation system as egoTEB. There are 25 runs per environment. As discussed in §I-A.1, the fixed parameter egoTEB implementation outperforms all of the learning-based implementations. The success rate is higher, there is a smaller gap between the max and min success rates, and the outcomes across environments are more consistent. If end-to-end learning is to be pursued as a navigation scheme, structurally different solutions than those explored will be needed. For a traditionally engineered solution, such as egoTEB, the environment sensitivity should be addressed.

B. Evaluation in Maze Environments

Moving to the learnt policy tuners, Table II contains the success rates for the maze environment with the maximum number of obstacles (200) and variation of d_{LA} only, for the same/different maze environments. The success rates are similar across all methods when examining the two maze types (compare down columns); they vary by 3% or less. The similarity indicates that the policy tuners are most likely learning the local obstacle distributions and not the wall setup. All tuner models outperform the default values (DV), but they do not all outperform the best values (BV). The only tuners which do are the RL implementations. This outcome suggests that it is beneficial to have a closed-loop, embodied learning process whereby the policy tuner evaluates and corrects its own performance. Doing so more effectively explores the parameter and environment space.

TABLE II: Success Rates for Maze Environments

	DV	BV	Classifier			Regressor			DQN	
			L	NN	CNN	L	NN	CNN	Sc	WS
Same	67	78	70	73	74	69	72	75	**81**	80
Different	68	75	69	71	74	69	71	73	**80**	**80**

C. Evaluation in All Benchmark Environments

The success rate outcomes across all navigation environments and for all obstacle configuration levels are plotted in the graphs of Figure 5 (top). Both parameters f_{GP} and d_{LA} vary. The first property to note is that all policy tuners operate at or above the success rate (SR) of the fixed parameters egoTEB implementation. The non-negative performance impact shows that the results from §IV-B translate across obstacle quantities and environments. The top two performers continue to be the RL tuners since their traces often lie above those of the others. Lower slopes for the policy tuners relative to those of the default egoTEB implementation is an indication of reduced sensitivity to the obstacle density. Several policy tuners have this property for some obstacle density ranges.

Figure 5 (bottom) quantifies the reduced sensitivity by plotting bar graphs of the difference between the highest and the lowest success rates (for a given method) across the different obstacle count implementations. Almost all methods have a lower difference versus the baseline egoTEB implementation, though some just barely. In a couple cases (mostly for the sector world), the higher difference is a function of a higher best performance that increased the size of the performance gap while still outperforming the default egoTEB. Though better by number, having a large difference is less than ideal. Again, the RL tuners exhibit the best performance, with the DQN trained models having reduced the difference to 35% and 46% of egoTEB's on average for training from scratch and warm start, respectively. One detail in favor of a warm start is that it improved the zero obstacle success rate for the maze and sector cases, which is where the default egoTEB does not have perfect performance. This boost is not seen for training from scratch.

D. Evaluation of a 7D-DQN NavTuner

Since the RL-based policy tuner has the best performance, we will denote any such implementation to be a *NavTuner* for short-hand. Given the good performance of the 2D NavTuner, a 7D version with a 7D-DQN is deployed and trained using the same training process but with 4000 rollouts (taking 4.2 days to train a model). The 2D-DQN is trained again from scratch for the same 4000 rollouts. The global planner parameter value to output is f_{GP}, and the egoTEB local planner parameter values to output are d_{LA}, *selection cost*

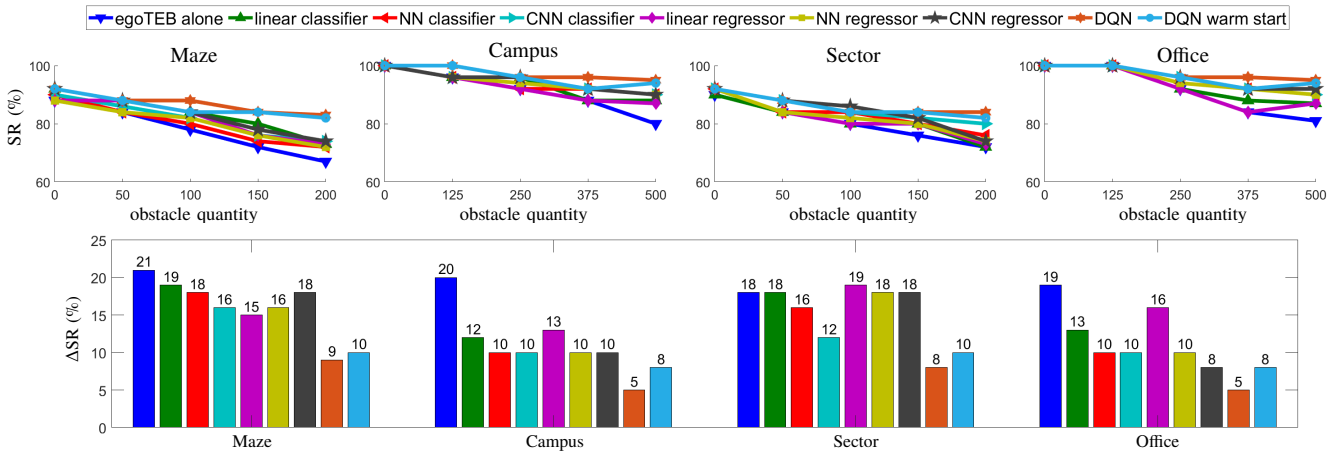

Fig. 5: Top: Success Rate vs Obstacle Quantity plots for the different environments. Bottom: Difference in Success Rate between best and worst cases for the different environments. Legend color coding applies to both figures.

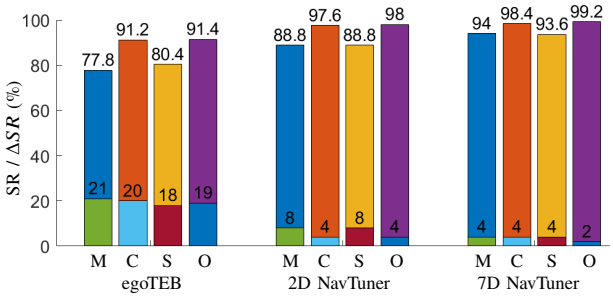

Fig. 6: Success Rate and Difference in Success Rate bar plots for different environments and navigation systems. The taller bars are success rates with the actual numbers printed above them. The shorter overlaid bars are the success rate differences with the actual differences printed above them.

hysteresis, path *switching blocking period*, *selection prefers initial plan*, *inflation distance*, and the *number of poses in the feasibility check* (integer). Comparative results between egoTEB, the 2D-DQN, and the 7D-DQN are in the bar charts of Figure 6 with the actual numerical values printed above the bars. The 7D NavTuner boosts the performance and reduces the difference, as seen by the higher bars and the lower difference values. The zero obstacle performance boost, not seen for the 2D NavTuner case with training from scratch, now occurs. The campus and office worlds have near perfect success rates across all implementations as evidenced by the high success rate and low difference. Recall that they start at 100% with no obstacles. Compared to egoTEB, the average success rate (marginalized across all environments) for egoTEB with a 7D-DQN NavTuner improves by 13%, from 85.2% to 96.3%. The sensitivity drops from an average difference of 19.5% for egoTEB to 3.5% for the 7D NavTuner version (82% lower). When compared to the 2D NavTuner, the sensitivity drop is from 6% to 3.5% (42% lower).

E. Discussion and Comparison to Other Tuning Models

The online tuning of navigation parameters using deep learning methods is relatively recent. This section discusses the outcomes of a few methods in relation to the NavTuner findings. The approach in [35] aimed to arrive at a differentiable model for the deep network training process to correct for structural deficiencies in soft-constraint optimal control solvers. While the method did lower sensitivity to obstacle configurations, it did not improve the success rate. It is doubtful that such an approach could work for the class of soft-constraint optimization solvers studied. Typically, soft constraints are addressed using a scale-space method that solves the problem multiple times under different soft-constraint parameters to incrementally approach the hard-constraint solution. Even then feasibility is not guaranteed. The egoTEB algorithm builds on TEB [1], which also employs a soft-constraint optimal control solver using factor graphs like [35]. By solving a highly localized problem for detected gaps, obtaining multiple solutions, and performing feasibility checking, many of the deficiencies of soft-constraint solvers are avoided. That NavTuner boosts performance and reduces sensitivity using a model-free and non-differentiable method suggests that trying to derive a solution like [35] may not be necessary when there are learning schemes that do not require it. Model-free RL will be more effective when the parameters are difficult to differentiate, as holds for the egoTEB navigation parameters.

The approaches in [36]–[38] (APPLI, APPLD, and APPLR) are the closest to NavTuner. For APPLI, rather than report success rate a time penalty was assigned to the run, and only the average completion times were reported. APPLI reported a 43% relative improvement compared to fixed parameters DWA [25], but also exhibited worse performance for a smaller percentage of trials versus DWA. APPLD discussed navigation failures in the text. Under APPLD, the designed maze was traversed every time, while DWA failed for the great majority. APPLD requires human demonstration and cannot leverage simulation to automatically learn a policy tuner. Hence the implementation of APPLR based on RL, much like NavTuner. Evaluation of APPLR versus DWA was mixed, in the sense that APPLR performed better for a subset of experiments, but worse or equivalently for another (with these latter being the harder scenarios). APPLR

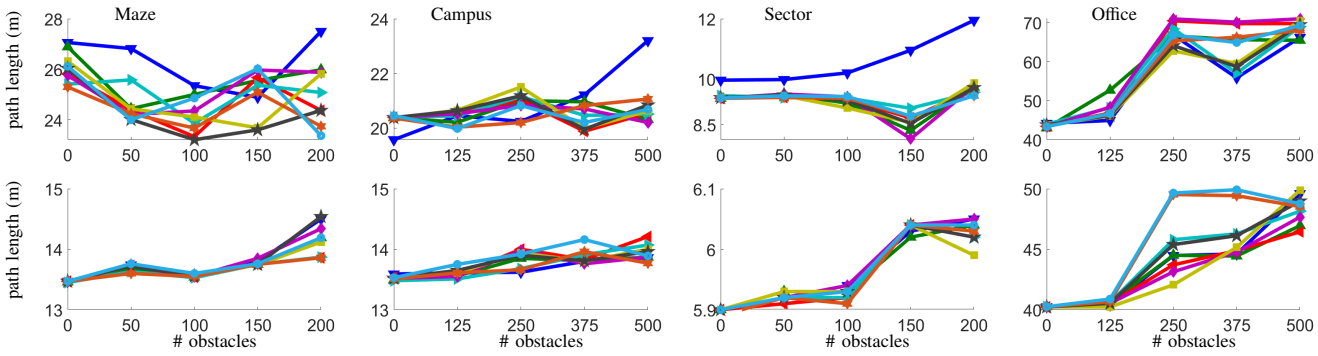

Fig. 7: **Top row:** Path length comparison across the different approaches based on successful runs for each approach. **Bottom row:** Path length comparison restricted to the common successful runs. Legend same as in Figure 5.

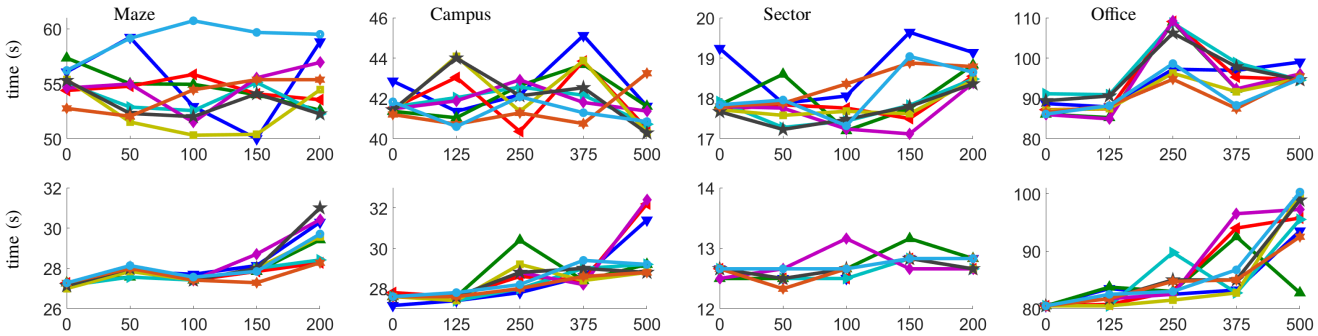

Fig. 8: **Top row:** Traversal time comparison across the different approaches based on successful runs for each approach. **Bottom row:** Traversal time comparison restricted to the common successful runs. Legend same as in Figure 5.

had high variance in traversal time versus DWA, which led to the mixed outcomes. Though we did record path length and runtime as performance metrics, we found that they did not capture performance as well the success rate did. Furthermore, it is more difficult to compare when the approaches do not have comparable success rates, which may explain the mixed outcomes of the APPLx implementations. For example, Figures 7 and 8 compare the path lengths and traversal times, respectively. There are two rows, with the top being the averages across all successful runs for each method and the bottom being the average for the common successful runs across all obstacle configurations, thus it compares comparable navigation scenarios and outcomes vs methods. Performance is comparable when looking at the runs for which all succeeded (bottom rows). Divergence of statistics happens when including the differing outcomes, usually in the form of more variance or a shift in performance. The increased path length for office by the RL methods in the bottom row may seem incorrect, but it persists in the top row and these methods have the best success rate. This shift most likely indicates that a longer, circuitous path is being taken, but that is more favorable. Being longer is not disadvantageous. For the sector case (top row), egoTEB has a monotonic increase while the others do not. The common dip suggests that they are also leveraging some of the global scene structure that egoTEB with fixed parameters cannot. Comparing path length and traversal time has more nuance than just needing to be shorter.

One potential difference is that the APPLx navigation scenarios resemble tunnel-like environments solvable using a forward motion wander navigation method, which might be the cause of the mixed outcomes. Additionally, AAPLx uses a high or full field of view laser scanner while NavTuner uses a limited field-of-view depth imaging sensor, leading to the possibility of passing near unsensed regions. NavTuner learns to modulate the parameters such that these partially observed situations do not significantly impact navigation performance. No sensitivity analysis for the APPLx algorithms illuminated clear differences in outcomes for a variable environmental parameter, thus it is more difficult to ascertain the overall benefits of APPLR over DWA beyond traversal time (the text does note more recovery behavior state events, but does not quantify them). In the sensitivity analysis here, the one variable that exhibited the most sensitivity, and also happens to be the most important, is the success rate. Using RL, NavTuner effectively learns a family of scene-sensitive navigation policies to select from during navigation, thereby leading to performance and robustness improvements.

V. Conclusion

This study provided experimental evidence using controllable simulations of navigation scenarios to show that online adaptive tuning of navigation parameters for engineered navigation systems can improve run-time performance and reduce sensitivity to environmental conditions. The sensitivity associated to traditionally designed systems is often used as a justification for replacing them with deep learning models. This paper started by showing that contemporary deep learn-

ing navigation methods could not match the performance of a fixed parameter navigation system. Furthermore, it showed that learning a family of navigation policies for online navigation parameter tuning further boosts performance. The improvements are more substantive in comparison to existing methods with similar contexts, suggesting that those studies were incomplete. The code is open-sourced [3].

The use of the free space measurements as the input data should permit some level of sim-to-real transfer since it is an intermediate representation. Deep learning on intermediate representations, or with them, is less sensitive to source signal noise since it has been processed out. The APPLx family of tuners provides evidence that transfer can happen. Future work aims to confirm this conjecture for NavTuner.

References

[1] C. Rösmann, W. Feiten, T. Wösch, F. Hoffmann, and T. Bertram, "Efficient Trajectory Optimization Using a Sparse Model," in *European Conference on Mobile Robots*, Sept 2013, pp. 138–143.

[2] J. S. Smith, R. Xu, and P. Vela, "egoTEB: Egocentric, perception space navigation using timed-elastic-bands," in *ICRA*, 2020, pp. 2703–2709.

[3] H. Ma, J. Smith, and P. Vela, "IVALab: NavTuner git repository," 2021. [Online]. Available: https://github.com/ivaROS/NavTuner

[4] J. Smith, J. Hwang, F. Chu, and P. Vela, "Learning to navigate: Exploiting deep networks to inform sample-based planning during vision-based navigation," *arxiv/CoRR*, vol. abs/1801.05132, 2018.

[5] M. Pfeiffer, M. Schaeuble, J. Nieto, R. Siegwart, and C. Cadena, "From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots," in *ICRA*, May 2017, pp. 1527–1533.

[6] W. Gao, D. Hsu, W. S. Lee, S. Shen, and K. Subramanian, "Intention-Net: Integrating Planning and Deep Learning for Goal-Directed Autonomous Navigation," *arXiv*, no. 1710.05627, 2017.

[7] X. Xiao, B. Liu, G. Warnell, and P. Stone, "Motion control for mobile robot navigation using machine learning: a survey," *arXiv*, no. 2011.13112, 2020.

[8] A. Faust, O. Ramirez, M. Fiser, K. Oslund, A. Francis, J. Davidson, and L. Tapia, "Prm-rl: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning," in *ICRA*, 2018, pp. 5113–5120.

[9] B. Kim and J. Pineau, "Socially adaptive path planning in human environments using inverse reinforcement learning," *International Journal of Social Robotics*, vol. 8, no. 1, pp. 51–66, Jan 2016.

[10] D. S. Chaplot, D. Gandhi, S. Gupta, A. Gupta, and R. Salakhutdinov, "Learning to explore using active neural slam," *arXiv preprint arXiv:2004.05155*, 2020.

[11] S. Bansal, V. Tolani, S. Gupta, J. Malik, and C. Tomlin, "Combining optimal control and learning for visual navigation in novel environments," in *Conference on Robot Learning*, 2020, pp. 420–429.

[12] G. Kahn, A. Villaflor, B. Ding, P. Abbeel, and S. Levine, "Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation," in *ICRA*, May 2018, pp. 1–8.

[13] 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 *ICRA*, May 2017, pp. 3357–3364.

[14] J. Bruce, N. Sünderhauf, P. Mirowski, R. Hadsell, and M. Milford, "One-shot reinforcement learning for robot navigation with interactive replay," *arXiv*, no. 1711.10137, 2017.

[15] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *JMLR*, vol. 17, no. 1, pp. 1334–1373, 2016.

[16] J.-M. Choi, S.-J. Lee, and M. Won, "Self-learning navigation algorithm for vision-based mobile robots using machine learning algorithms," *J. of Mech. Sci. and Tech.*, vol. 25, no. 1, pp. 247–254, Jan 2011.

[17] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," in *IROS*, Sep. 2017, pp. 31–36.

[18] J. Zhang, J. T. Springenberg, J. Boedecker, and W. Burgard, "Deep reinforcement learning with successor features for navigation across similar environments," in *IROS*, Sep. 2017, pp. 2371–2378.

[19] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, D. Kumaran, and R. Hadsell, "Learning to navigate in complex environments," *arXiv*, vol. 1611.03673, 2016.

[20] J. Ye, D. Batra, E. Wijmans, and A. Das, "Auxiliary tasks speed up learning pointgoal navigation," *arXiv*, no. 2007.04561, 2020.

[21] A. Sax, J. O. Zhang, B. Emi, A. Zamir, S. Savarese, L. Guibas, and J. Malik, "Learning to navigate using mid-level visual priors," *arXiv preprint arXiv:1912.11121*, 2019.

[22] A. Kumar, S. Gupta, and J. Malik, "Learning navigation subroutines from egocentric videos," in *Conference on Robot Learning*, 2020, pp. 617–626.

[23] W. Gao, D. Hsu, W. S. Lee, S. Shen, and K. Subramanian, "Intention-net: Integrating planning and deep learning for goal-directed autonomous navigation," *arXiv*, no. 1710.05627, 2017.

[24] Y. Kim, J. Jang, and S. Yun, "End-to-end deep learning for autonomous navigation of mobile robot," in *ICCE*, Jan 2018, pp. 1–6.

[25] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *RA-M*, vol. 4, no. 1, pp. 23–33, 1997.

[26] C. Roesmann, W. Feiten, T. Woesch, F. Hoffmann, and T. Bertram, "Trajectory modification considering dynamic constraints of autonomous robots," in *German Conf. on Robotics*, May 2012, pp. 1–6.

[27] C. Wang, L. Meng, S. She, I. M. Mitchell, T. Li, F. Tung, W. Wan, M. Q. . Meng, and C. W. de Silva, "Autonomous mobile robot navigation in uneven and unstructured indoor environments," in *IROS*, Sep. 2017, pp. 109–116.

[28] M. Feurer and F. Hutter, "Hyperparameter optimization," in *Automated Machine Learning*. Springer, Cham, 2019, pp. 3–33.

[29] T. Yu and H. Zhu, "Hyper-parameter optimization: A review of algorithms and applications," *arXiv*, no. 2003.05689, 2020.

[30] C. Huang, B. Yuan, Y. Li, and X. Yao, "Automatic parameter tuning using bayesian optimization method," in *CEC*, 2019, pp. 2090–2097.

[31] H. L. Chiang, A. Faust, M. Fiser, and A. Francis, "Learning navigation behaviors end-to-end with autorl," *RA-L*, vol. 4, no. 2, pp. 2007–2014, April 2019.

[32] J. Cano, Y. Yang, B. Bodin, V. Nagarajan, and M. O'Boyle, "Automatic parameter tuning of motion planning algorithms," in *IROS*, 2018, pp. 8103–8109.

[33] R. Burger, M. Bharatheesha, M. van Eert, and R. Babuška, "Automated tuning and configuration of path planning algorithms," in *ICRA*, pp. 4371–4376.

[34] F. Berkenkamp, A. Krause, and A. P. Schoellig, "Bayesian optimization with safety constraints: safe and automatic parameter tuning in robotics," *arXiv*, no. 1602.04450, 2016.

[35] M. Bhardwaj, B. Boots, and M. Mukadam, "Differentiable gaussian process motion planning," in *ICRA*, 2020, pp. 10 598–10 604.

[36] Z. Wang, X. Xiao, B. Liu, G. Warnell, and P. Stone, "Appli: Adaptive planner parameter learning from interventions," *arXiv*, no. 2011.00400, 2020.

[37] X. Xiao, B. Liu, G. Warnell, J. Fink, and P. Stone, "Appld: Adaptive planner parameter learning from demonstration," *arXiv*, no. 2004.00116, 2020.

[38] Z. Xu, G. Dhamankar, A. Nair, X. Xiao, G. Warnell, B. Liu, Z. Wang, and P. Stone, "Applr: Adaptive planner parameter learning from reinforcement," *arXiv*, no. 2011.00397, 2020.

[39] J. S. Smith, S. Feng, F. Lyu, and P. A. Vela, *Real-Time Egocentric Navigation Using 3D Sensing*. Cham: Springer International Publishing, 2020, pp. 431–484. [Online]. Available: https://doi.org/10.1007/978-3-030-22587-2_14

[40] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[41] A. Tavakoli, F. Pardo, and P. Kormushev, "Action branching architectures for deep reinforcement learning," in *AAAI Conf. on Art. Int.*, 2018.

[42] S. Koenig and M. Likhachev, "Fast replanning for navigation in unknown terrain," *IEEE Transactions on Robotics*, vol. 21, no. 3, pp. 354–363, June 2005.

[43] P. Anderson, A. Chang, D. S. Chaplot, A. Dosovitskiy, S. Gupta, V. Koltun, J. Kosecka, J. Malik, R. Mottaghi, M. Savva, *et al.*, "On evaluation of embodied navigation agents," *arXiv*, no. 1807.06757, 2018.