Generating Robust Supervision for Learning-Based Visual Navigation Using Hamilton-Jacobi Reachability

Anjian Li[1]　　　　　　　　　　　　　　　　　　　　　ANJIANL@SFU.CA

Somil Bansal[2]　　　　　　　　　　　　　　　　　　　SOMIL@BERKELEY.EDU

Georgios Giovanis[1]　　　　　　　　　　　　　　　　　GGIOVANI@SFU.CA

Varun Tolani[2]　　　　　　　　　　　　　　　　　　VTOLANI@BERKELEY.EDU

Claire Tomlin[2]　　　　　　　　　　　　　　　　　　TOMLIN@BERKELEY.EDU

Mo Chen[1]　　　　　　　　　　　　　　　　　　　　　MOCHEN@SFU.CA

[1]*School of Computing Science,*　　　　[2]*Department of Electrical Engineering and Computer Sciences,*
Simon Fraser University,　　　　　　　　*University of California, Berkeley,*
Burnaby BC, Canada, V5A 1S6　　　　　*Berkeley, CA 94720, USA*

Editors: A. Bayen, A. Jadbabaie, G. J. Pappas, P. Parrilo, B. Recht, C. Tomlin, M. Zeilinger

Abstract

In Bansal et al. (2019), a novel visual navigation framework that combines learning-based and model-based approaches has been proposed. Specifically, a Convolutional Neural Network (CNN) predicts a waypoint that is used by the dynamics model for planning and tracking a trajectory to the waypoint. However, the CNN inevitably makes prediction errors which often lead to collisions in cluttered and tight spaces. In this paper, we present a novel Hamilton-Jacobi (HJ) reachability-based method to generate supervision for the CNN for waypoint prediction in an unseen environment. By modeling CNN prediction error as "disturbances" in robot's dynamics, our generated waypoints are robust to these disturbances, and consequently to the prediction errors. Moreover, using globally optimal HJ reachability analysis leads to predicting waypoints that are time-efficient and avoid greedy behavior. Through simulations and hardware experiments, we demonstrate the advantages of the proposed approach on navigating through cluttered, narrow indoor environments.

Keywords: visual navigation, reachability analysis, optimal control, machine learning

1. Introduction

Autonomous navigation is fundamental to control and robotics. Following the success of deep learning, visual navigation has gained popularity. One appeal of visual navigation – which involves using one or more cameras and computer vision to perceive the environment to reach navigational goals – is that cameras are cheap, light weight, and ubiquitous.

Typically, a geometric map of the environment is used for navigation (Thrun et al. (2005); Fuentes-Pacheco et al. (2015); LaValle (2006)). However, real-time map generation can be challenging in texture-less environments or in the presence of transparent, shiny objects, or strong ambient lighting (Alhwarin et al. (2014)). In contrast, end-to-end (E2E) learning approaches have been used for locomotion (Gandhi et al. (2017); Kahn et al. (2017); Sadeghi and Levine (2017); Kang et al. (2019)) and goal-point navigation (Zhu et al. (2017); Gupta et al. (2017); Khan et al. (2018); Kim and Chen (2015); Pan et al. (2018)) that side-step this explicit map estimation step, but suffer from data inefficiency and lack of robustness (Recht (2018)). Consequently, a number of papers seek to combine the best of learning with optimal control for high-speed navigation (Richter et al. (2018);

Jung et al. (2018); Loquercio et al. (2018); Bansal et al. (2018); Müller et al. (2018); Meng et al. (2019)), race-track driving (Drews et al. (2017, 2019)), and drone racing (Kaufmann et al. (2019, 2018)). In particular, Bansal et al. (2019) combines the ideas from optimal control and computer vision by having a convolutional neural network (CNN) predict waypoints instead of control signals and using optimal control to obtain the control for reaching the waypoints. This hybrid approach greatly improves generalizability: a CNN trained in simulation could be successfully deployed on a real robot without any additional training or tuning. However, the inevitable errors in waypoint predictions during the test time may cause unintended robot trajectories, ultimately resulting in collisions with the obstacles. This is particularly problematic when the robot needs to navigate through cluttered environments or narrow openings, as the error margin in such scenarios is often small.

Contributions: Based on the framework in Bansal et al. (2019) we propose a novel reachability-based method to generate robust waypoints for supervising the CNN. Our key insight is to model the CNN prediction error as "disturbance" in robot's dynamics and generate waypoints that are optimal under worst-case disturbances, ensuring robustness despite the prediction errors. Compared to other works in the safe learning literature such as Fisac et al. (2019) and Bajcsy et al. (2019), which wrap reachability-based safety controllers around policies being learned, we provide an alternative that pre-emptively uses disturbances in training data generation.

Unlike Bansal et al. (2019), which relies on distance-based heuristics, our method involves solving static Hamilton-Jacobi (HJ) (Bansal et al. (2017)) partial differential equations (PDEs). The obtained value functions represent the time until goal-reaching and time until col-
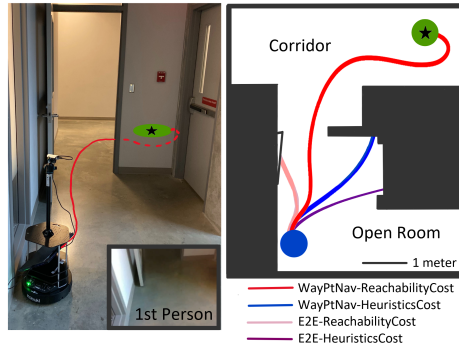

Figure 1. Hardware experiment 1. On the left, 3rd and 1st person views of the Turtlebot2 testbed are shown along with a trajectory (red) to the goal (green). On the right, a birds-eye view of the environment is displayed. The Turtlebot's starting location is shown as a blue circle.

lision despite worst-case disturbances, given system dynamics and a known environment. These value functions are combined into a cost map that precisely quantifies the quality of waypoints and considers all possible combinations of states by construction. This leads to less greedy navigation behavior and significant improvement in the success rate during test time.

Overall, our approach leads to less greedy navigation behaviors that are robust to CNN prediction errors. Through simulations and real-world experiments, we demonstrate that practical scenarios such as safely moving through narrow doorways become possible with the proposed approach.

2. Problem Setup

We consider the problem of autonomous navigation in *a priori* unknown static environment. Starting from an initial position, the robot needs to reach a goal position $p^* = (x^*, y^*)$. We model our ground vehicle as a four-dimensional (4D) system with the following dynamics:

$$\dot{x} = v \cos \phi, \quad \dot{y} = v \sin \phi, \quad \dot{v} = a, \quad \dot{\phi} = \omega, \tag{1}$$

where the state $z(t)$ at time t consists of the position $(x(t), y(t))$, speed $v(t)$, and heading $\phi(t)$. The control is acceleration and turn rate, $u(t) := (a(t), \omega(t))$. The robot is equipped with a forward-facing, monocular RGB camera mounted at a fixed height and oriented at a fixed pitch. At time t, the robot receives an RGB image of the environment $\mathcal{E}$, $I(t) = I(\mathcal{E}, z(t))$, the state $z(t)$, and the target position $p^* = (x^*, y^*)$. The objective is to obtain a control policy that uses these inputs to guide the robot to within a certain distance of p^*.

3. Background

We build upon the learning-based waypoint approach to navigation (LB-WayPtNav) proposed in Bansal et al. (2019). However, unlike LB-WayPtNav, we use a HJ Reachability-based framework to generate supervision data. We now provide a brief overview of LB-WayPtNav and HJ reachability.

3.1. LB-WayPtNav

LB-WayPtNav combines a learning-based perception module with a dynamics model-based planning and control module for navigation in *a priori* unknown environments (see Fig. 2).

Perception module. The perception module is implemented as a CNN that takes as input a 224×224 pixel RGB image, $I(t)$, captured from the onboard camera, the target position, p^*, specified in the vehicle's current coordinate frame, and vehicle's current linear and angular speed, $(v(t), \omega(t))$, and outputs the desired state or a waypoint $\hat{w}(t) := (\hat{x}(t), \hat{y}(t), \hat{\theta}(t))$.

Figure 2. LB-WayPtNav framework adapted from Bansal et al. (2019).

Planning and control module. Given a waypoint $\hat{w}(t)$, the system dynamics in Eqn. (1) are used to plan a spline-based trajectory to $\hat{w}(t)$, starting from the current state of the vehicle. This leads to a smooth, dynamically feasible, and computationally efficient trajectory to the waypoint. An LQR-based feedback controller tracks the trajectory. The commands generated by the LQR controller are executed on the system for H seconds, and then a new image is used to generate another waypoint and trajectory. This process is repeated until the robot is within a certain distance of p^*.
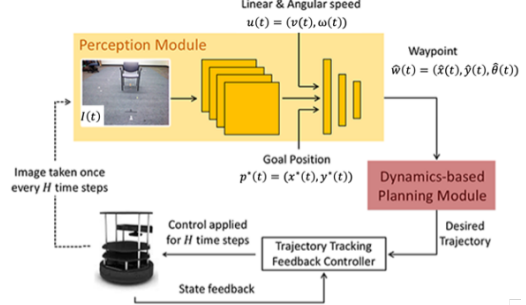
3.2. The Time-to-Reach Problem in Reachability Analysis

Consider a dynamical system described by $\dot{z}(t) = f(z(t), u(t), d(t))$, $z(0) = z_0$, where $z \in \mathbb{R}^n$ is the state, and $u \in \mathcal{U}, d \in \mathcal{D}$ are the control and disturbance respectively. In this paper, the control represents the actions a robot can take to change its state, and the disturbance primarily models CNN prediction errors. Following Yang et al. (2013) and Takei et al. (2010), we define the time-to-reach (TTR) value function denoted as $V_R(z)$, which represents the minimum time required to drive the system to the goal set Γ_G while avoiding the obstacle set Γ_O, despite worst-case disturbance. The control minimizes this time and disturbance maximizes: $V_R(z) = \max_{d(\cdot) \in \mathbb{D}} \min_{u(\cdot) \in \mathbb{U}} \min\{t | z(t) \in \Gamma_G \wedge \forall s \in [0, t], z(s) \notin \Gamma_O\}$. We also define the time-to-collision (TTC) value function, $V_C(z)$, which represents the maximum time until collision with the obstacle set Γ_O assuming that the control is optimally avoiding obstacles under worst-case disturbance: $V_C(z) = \min_{d(\cdot) \in \mathbb{D}} \max_{u \in \mathbb{U}} \min\{t | z(t) \in \Gamma_O\}$. Applying the dynamic programming principle, we can obtain $V_R(\cdot)$ and $V_C(z)$ respectively as the viscosity solution to the following stationary HJ PDEs:

$$V_R(z) = 0 \text{ in } \Gamma_G, \quad V_R(z) = \infty \text{ in } \Gamma_O, \quad \max_{u \in \mathcal{U}} \min_{d \in \mathcal{D}} \{-\nabla V_R(z)^\top f(z, u, d) - 1\} = 0 \text{ otherwise}$$

$$V_C(z) = 0 \text{ in } \Gamma_O, \quad \min_{u \in \mathcal{U}} \max_{d \in \mathcal{D}} \left\{ -\nabla V_C(z)^\top f(z, u, d) - 1 \right\} = 0 \text{ otherwise} \tag{2}$$

4. Reachability-based Supervision for Waypoints

In the perception module presented in Fig. 2, from the start position, the robot sequentially observes the image $I(t)$, linear speed $v(t)$ and angular speed $\omega(t)$ at time t to predict a waypoint $\hat{w}$ with a
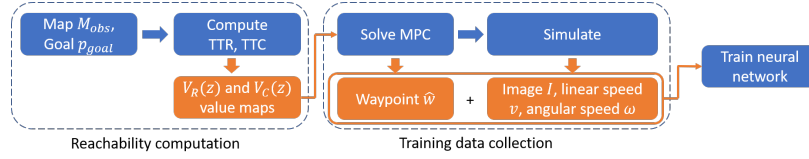
Figure 3.   Workflow of training data generation using reachability expert.

CNN. This waypoint prediction is conducted at a fixed replan frequency until the robot reaches the goal Γ_G, defined to be positions within certain distance to p^*.

To generate supervision for safe and efficient waypoints, we propose a novel reachability expert to autonomously navigate in simulation and collect training data. Specifically, given an obstacle map M_{obs} and a goal area Γ_G, one can compute corresponding TTR and TTC value maps, which are integrated in the cost function of a model predictive control (MPC) optimization problem. By solving this MPC problem, the optimal waypoint $\hat{w}$ is obtained, and at time t, the image $I(t)$ is rendered, and linear and angular speed $\{v(t), \omega(t)\}$ are measured in simulation environment. Finally, we repeat the above procedure in different navigation tasks until sufficient data-label pairs $\{(I, v, \omega), \hat{w}\}$ are obtained for the training dataset. The entire procedure is illustrated in Fig. 3.

After training, the neural network can robustly transfer to novel, unknown environments.

4.1. TTR and TTC Computations

We add disturbances to Eqn. (1) to describe the expert's system dynamics

$$\dot{x} = v\cos\phi + d_x, \quad \dot{y} = v\sin\phi + d_y, \quad \dot{v} = a, \quad \dot{\phi} = \omega + d_\phi, \tag{3}$$

$$v \in [0, \bar{v}], \quad a \in [-\bar{a}, \bar{a}], \quad \omega \in [-\bar{\omega}, \bar{\omega}], \quad d_x^2 + d_y^2 \le \bar{d}_{xy}^2, \quad d_\phi \in [-\bar{d}_\phi, \bar{d}_\phi] \tag{4}$$

with states $z = (x, y, v, \phi)$, controls $u = (a, \omega)$ and disturbances $d = (d_x, d_y, d_\phi)$. Also, $\bar{a}, \bar{\omega}$, and $\bar{d}_\phi$ are upper bounds for acceleration, angular speed and disturbances in turn rate, and $\bar{d}_{xy}$ is the circular upper bound for disturbances in x and y components of speed.

For every navigation task, we initialize a goal position p^* on an known obstacle map M_{obs}. We define the goal area Γ_G to be $\Gamma_G = \{(x, y, \phi, v) : \sqrt{(x - p^*)^2 + (y - p^*)^2} \le c\}$, and the map M_{obs} as Γ_O. $V_R(z)$ and $V_C(z)$ are then computed based on Eqn. (2). $V_R(z)$ and $V_C(z)$ guide the reachability expert in a 4D state space for goal reaching and collision avoidance.

Incorporating worst-case disturbances leads to more conservative values of $V_R(z)$ and $V_C(z)$, which result in less greedy expert trajectories in a nuanced manner. Greediness often makes the robot incapable of going around obstacles and having a view of the environment that informs the robot about possible routes to the goal. Crucially, the conservatively safe trajectories address the prediction errors from neural networks, since a minor deviation will not lead to collision. Note that M_{obs} is assumed to be known only during training; no such assumption is made during the test time when the robot only relies on onboard sensors for navigation.

4.2. Waypoint Supervision Generation

We use an MPC framework to generate waypoints and expert trajectories. To achieve efficient and safe navigation, we trade off between reaching the goal faster and staying further from obstacles. Thus we design a novel cost function, ReachabilityCost J, to be a combination of TTR and TTC:

$$J(z) = V_R(z) + \alpha(\bar{V}_C - V_C(z)), \tag{5}$$

where $\bar{V}_C$ is the computational upper bound for TTC, and α is the scale factor.

The expert uses MPC to plan, in a receding horizon manner, a trajectory of time horizon H until the goal is reached. Starting from $t = 0$, in every MPC optimization problem, we discretize the time horizon $[t, t + H]$ to be $\{t_i | t + i\Delta t, i \in \{0, 1, ..., N\}\}$. At any time index i, we denote $z^{[i]} := z(t_i)$ and $J^{[i]} := J(t_i)$. Then the following MPC problem is sequentially solved in $[t, t + H]$:

$$\underset{\substack{z^{[0]}, ..., z^{[N]}, \\ u^{[0]}, ..., u^{[N]}}}{\text{minimize}} \sum_{i=0}^{N} J^{[i]}(z^{[i]})$$

subject to $x^{[i+1]} = x^{[i]} + \Delta t v^{[i]} \cos \phi^{[i]}$, $\quad y^{[i+1]} = y^{[i]} + \Delta t v^{[i]} \sin \phi^{[i]}$, $\quad \phi^{[i+1]} = \phi^{[i]} + \Delta t w^{[i]}$,

$$v^{[i+1]} = v^{[i]} + \Delta t a^{[i]}, \quad v^{[i]} \in [0, \bar{v}], \quad \omega^{[i]} \in [-\bar{\omega}, \bar{\omega}],$$

$$z^{[0]} = z(t), \quad z^{[N]} = z(t + H), \quad u^{[0]} = u(t), \quad u^{[N]} = u(t + H) \tag{6}$$

To solve Eqn. (6) in $[t, t + H]$, the reachability expert first samples a local waypoint grid $\hat{w}$ in the heading direction as possible final states $z^{[N]}$: $\hat{w} := (\hat{\boldsymbol{x}}^{[N]}, \hat{\boldsymbol{y}}^{[N]}, \hat{\boldsymbol{\phi}}^{[N]})$, and compute dynamically feasible spline trajectories to each waypoint using differential flatness of Eqn. (3) (Walambe et al. (2016)). Next, the expert filters out the invalid waypoints whose trajectory violates control constraints. Finally, the solution trajectory $\boldsymbol{z}$ with the minimum cost is chosen, and the corresponding waypoint $\hat{w} = (\hat{x}^{[N]}, \hat{y}^{[N]}, \hat{\phi}^{[N]})$ is added to the training data set along with the image $I(t)$ and speeds $v(t), \omega(t)$ at time t. By solving the MPC problem many times, we obtain the expert dataset $S = \{(I_k(t), v_k(t), \omega_k(t)), (\hat{x}_k^{[N]}, \hat{y}_k^{[N]}, \hat{\phi}_k^{[N]})\}_{k=1}^{M}$, where k is the index of the MPC problem, and M is the total number of data points (and the total number of MPC problems solved).

5. Summary of Simulation Results

After training on the generated expert dataset S, we test our model in a novel environment in simulation without an *a priori* known map.

Dataset: We use Stanford large-scale 3D Indoor Spaces Dataset Armeni et al. (2016) as our simulation environment, which are 3D scans of real world buildings. Two buildings are used for data generation and training; the 3rd *held-out* building is used as the test environment, which has significant differences in the object appearance and layout. For navigation tasks in training and testing, we sample various start and goal positions that require the robot to go through narrow openings.

Implementation details: We train the CNN in Fig. 2 with 150k data points from the reachability expert, $M = 150$k. The mean squared error loss is used and optimized using the Adaptive Moment Estimation (ADAM) algorithm with a learning rate of 10^{-4} and weight decay of 10^{-6}.

Metrics: We use both statistics and trajectory plots to present the test results. For statistics, we use success rate, average time to reach the goal area (for successful tasks), acceleration and jerk to measure the quality of trajectories. With trajectory plots, we analyze the robot's specific behaviors.

Baselines: We compare our approach with the HeuristicsCost designed in Bansal et al. (2019) for the MPC framework:

$$J^{heuristic}(z) := (\max\{0, \lambda_1 - d^{obs}(x, y)\})^3 + \lambda_2 (d^{goal}(x, y))^2 \tag{7}$$

where $d^{obs}(x, y)$ is the distance to the nearest obstacle, $d^{goal}(x, y)$ is the distance to the goal, and λ_1 and λ_2 are scaling factors. We compare our ReachabilityCost in Eqn. (5) to the HeuristicsCost in Eqn. (7) for two different navigation frameworks: waypoint navigation (WayPtNav) and end-to-end (E2E) learning, resulting in a total of 4 methods. WayPtNav maps the current image and controls inputs to the waypoint (as in Fig. 2), where E2E learning directly outputs the control commands given the same inputs. We also compare against an additional baseline, ReachabilityCost-NoDstb, which does not incorporate disturbances in the system dynamics during the data generation.

Table 1. **Quantitative Comparisons in Simulation:** We compute four metrics-success rate, average time to reach the goal, acceleration and jerk-on 200 navigation tasks with a replan frequency of 4Hz. The proposed method, WayPtNav-ReachabilityCost, is most successful at completing novel navigation tasks. Without the disturbances incorporated in the dynamics, ReachabilityCost takes the shortest time to reach the goal, but the success rate largely drops because of prediction errors. For E2E learning, the success rate is generally lower and the trajectories are less smooth (indicated by high average jerk).

Agent	Success (%)	Time taken (s)	Acceleration (m/s^2)	Jerk (m/s^3)
WayPtNav-ReachabilityCost	**63.82**	21.00 ±8.00	**0.06 ±0.01**	**0.94 ±0.13**
WayPtNav-HeuristicsCost	52.26	18.82 ±5.66	0.07 ±0.02	1.06 ±0.15
WayPtNav-ReachabilityCost-NoDstb	49.24	**16.19 ±4.8**	0.07 ±0.01	0.98 ±0.16
E2E-ReachabilityCost	8.04	19.55 ±4.72	0.07 ±0.01	2.16 ±0.30
E2E-HeuristicsCost	31.66	25.56 ±9.85	0.26 ±0.06	9.06 ±1.94

5.1. Expert Performance

We select $\bar{v} = 0.6$ m/s, $\bar{\omega} = 1.1$ rad/s, and $\bar{a} = 0.4$ m/s^2 to match the specifications of the Turtlebot 2 used in the hardware experiments (Sec. 7). We set $\bar{d}_{xy} = 0.05$ m/s and $\bar{d}_{\phi} = 0.15$ rad/s to account for prediction errors, and $\alpha = 30$ to prioritize collision avoidance. All expert trajectories are generated according to Section 4.2, where the replanning is done every 1.5s to collect training data. In Fig. 6 (a) to (c), we compare the expert trajectories obtained by HeuristicsCost and Reachability-Cost. The reachability expert uses the full system dynamics for optimizing waypoints. As a result, it maintains an appropriate orientation and speed when going through the narrow openings. Moreover, due to the presence of disturbances, it takes a conservative path, always staying near the middle of narrow openings, resulting in collision-free trajetcories even when there is prediction error. In contrast, HeuristicsCost takes a greedier path to approach the goal. To address CNN prediction error, Bansal et al. (2019) use an obstacle padding which makes narrow openings impossible to enter.

5.2. Test Results

We compare the different methods in Table 1. WayPtNav-ReachabilityCost achieves the highest success rate and least acceleration and jerk. WayPtNav-ReachabilityCost-NoDstb takes the shortest time to reach the goal, but experiences a notable drop on the success rate.

E2E learning has lower success rate and more jerky trajectories in genreral. Notably, ReachabilityCost has a significant lower success rate with E2E learning compared to WayPtNav.

6. Analysis of Simulation Results

Comparison with WayPtNav-HeuristicsCost. Shown in Sec. 5.1, the reachability expert is less greedy and more robust, enabling it to navigate through cluttered environments
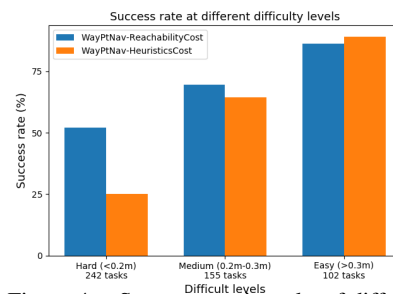

Figure 4. Success rate in tasks of different difficulties. Difficulties are assessed according to minimum distance to obstacles d_{min} along trajectories, divided into Hard ($d_{min} < 0.2$ m, 242 tasks), Medium (0.2 m $\leq d_{min} \leq 0.3$ m, 155 tasks) and Easy ($d_{min} > 0.3$ m, 102 tasks). ReachabilityCost has significant advantage in Hard tasks.

and narrow openings. This results in a higher success rate during test time, as shown in Fig. 4, where we compare the two methods on 500 navigation tasks with varying difficulties. Here, we measure the difficulty of a task by the opening size that the robot must navigate through on its way to the goal. ReachabilityCost has much higher success rate on the "Hard" level, indicating that it makes robot more adept at maneuvering in narrow environments.

Effect of Replanning Frequency. Replanning frequency represents how often the robot predicts new waypoints. Fig. 5 shows the success rate on 200 navigation tasks for 5 different replanning
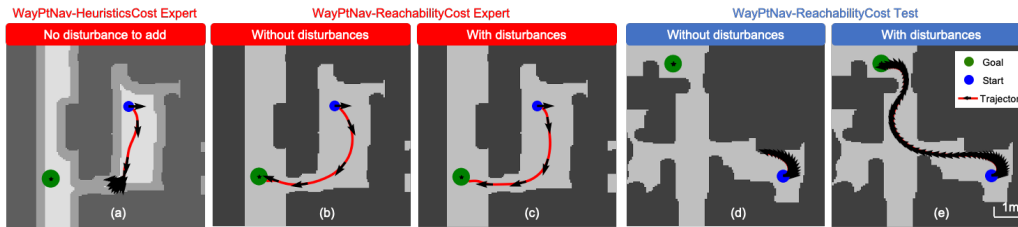
Figure 6. Trajectory comparison. (a), (b), and (c) are expert trajectories; (d) and (e) are test trajectories. In (a), the baseline expert starts with a greedier path and fails to enter the narrow opening due to hard obstacle padding (medium grey) used during waypoint optimization. Comparing (b) and (c), the reachability expert can safely go through the openings, but is more likely to stay in the middle of the road with disturbances incorporated in dynamics. This ability is transferred to test times in (d) and (e), where the robot with disturbances added is more resistant to the prediction errors from the neural network and manages to take a collision free path to the goal.

frequencies: 0.67 Hz, 1 Hz, 2 Hz, 4 Hz and 6.67 Hz. As the frequency increases, the success rate elevates to the peak at 4 Hz for both experts. Above 4 Hz, the success rate drops dramatically.

In general, a higher replanning frequency helps the system react faster to the unknown environment; however, if it is too high, the lack of visual memory in the WayPtNav framework results in myopic decisions, leading to a drop in the success rate. ReachabilityCost benefits more from the higher replanning frequencies compared to HeuristicsCost, as the greedy behaviors from the latter tend to drive the robot to cut corners, often leading to situations that are hard to recover from.

Effect of Adding Disturbances. Adding disturbances in the dynamics is crucial for improving test performance: it not only accounts for dynamics uncertainties, but also models neural network prediction errors. Without disturbances, trajectory can be "too optimal" so that a little deviation of robot's state or minor errors from the CNN results in collision. This is evident from Table 1, where the success rate drops from 63.82% to 49.24% in the absence of disturbances.
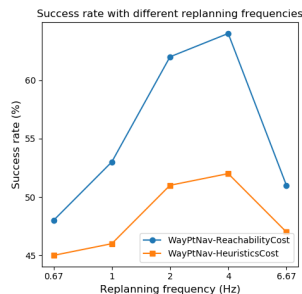

Figure 5. Success rates improve for both methods as the replanning frequency is increased (up to 4 Hz). ReachabilityCost benefits considerably more with higher replan frequencies.

We examine reachability expert trajectories with and without the disturbances in Fig. 6 (b) and (c). Without disturbances, the expert chooses a path close to the obstacles, while with disturbances, the expert stays near the middle of the road. Although both experts succeed in reaching the goal, disturbances lead to more robust trajectories, which translate to test time as shown in Fig. 6 (d) and (e)). Without disturbances, the robot tries to avoid the wall but fails, while with disturbances, the robot is able to stay in the middle of the opening, and pass through a very narrow doorway.

Comparison with E2E Learning. Our conclusions here are consistent with the findings in Bansal et al. (2019) – the model-based approach (WayPtNav) leads to a higher success rate and significantly smoother trajectories. The success rate of E2E learning declines further for the reachability expert as the control profiles are even more nuanced, making it challenging to learn them.

Failure Cases. Since we do not construct a map or have any visual memory in this framework, the robot struggles in the navigation scenarios where it needs to "backtrack". In addition, when the room layout is too different from the training time, the CNN fails to predict good waypoints.

7. Hardware Experiments

We tested our framework on a Turtlebot 2 hardware testbed (Fig. 1), using monocular RGB images from the onboard camera for navigation. We tested the ReachabilityCost and HeuristicsCost CNNs directly on the Turtlebot without any additional training or fine-tuning.

Experiments were carried out in 3 separate areas of Simon Fraser University, each of which were absent from the training set. An outline of each experiment and the trajectories taken by the Turtlebot with each CNN are shown in Figures 1 and 7. Video footage of all experiments can be found at https://www.youtube.com/playlist?list=PLUBop1d3Zm2uDGGfGrjWiSjrSlzo5vWMs.

Each scenario required the Turtlebot to traverse narrow spaces and doorways, both of which have shown a low success rate for the HeuristicsCost CNN. For the first scenario, the Turtlebot maneuvered through a narrow doorway, and then around the nook behind the door to reach the goal (Fig. 1). The second and third scenarios required the Turtlebot to move from an open room into a narrow corridor and from a cluttered environment into a hallway respectively (Fig. 7).

For each scenario, the HeuristicsCost CNN was unable to maneuver through the doorways, and collided with the wall at full speed, while the ReachabilityCost CNN successfully navigated through the doors and reached the goal. The ability to navigate through narrow environments and doorways is a key improvement seen in the ReachabilityCost CNN. For both methods, the neural networks trained using end-to-end learning were unable to reach the goal.
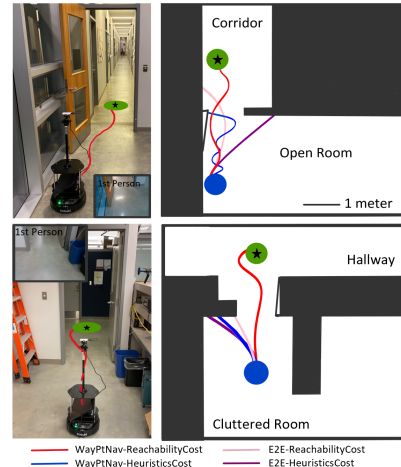

Figure 7. Experimental scenarios 2 (top) and 3 (bottom), shown in 3rd and 1st person along with birds-eye view. The goal is marked in green, and the trajectory taken by the ReachabilityCost CNN to the goal is shown in red. The Turtlebot's starting location is shown as a blue circle.

8. Discussion

The CNN prediction error is the deviation of the predicted waypoint away from the optimal one, resulting in a deviation of the robot trajectory. In the TTR and TTC value computations, worst-case disturbances are added to the dynamics, which lead to conservative TTR values (larger) and TTC values (smaller) that account for possible deviations in the robot trajectory. This conservatism can effectively compensate for the prediction error.

Currently we choose the disturbance bound based on empirical results from our experiments. In the future, more accurate bound can be learned using principled analytical and data-driven methods.

9. Conclusion

In this paper, we present a novel method to generate training data for waypoint prediction in visual navigation, using reachability analysis. Our method helps the neural network to learn to predict efficient and safe waypoints given system dynamics and observations of the environment. We also use disturbances in dynamics to model neural network prediction errors and greatly enhance its robustness. Simulation and real robot experiments demonstrate higher success rate and smoother trajectories in navigation tasks with our method, which crucially enables the robot to pass through narrow passages such as doorways. Immediate future work includes adding memory in our navigation framework, investigating the data mismatch between training and test scenarios to better transfer the expert performance in test time, and incorporating human agents.

Acknowledgments

References

Faraj Alhwarin, Alexander Ferrein, and Ingrid Scholl. IR stereo kinect: improving depth images by combining structured light with IR stereo. In *PRICAI*, 2014.

Iro Armeni, Ozan Sener, Amir R Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. 3d semantic parsing of large-scale indoor spaces. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

Andrea Bajcsy, Somil Bansal, Eli Bronstein, Varun Tolani, and Claire J. Tomlin. An Efficient Reachability-Based Framework for Provably Safe Autonomous Navigation in Unknown Environments. In *Proc. Conf. Decision and Control*, 2019.

Mayank Bansal, Alex Krizhevsky, and Abhijit Ogale. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. *arXiv preprint arXiv:1812.03079*, 2018.

Somil Bansal, Mo Chen, Sylvia Herbert, and Claire J. Tomlin. Hamilton-Jacobi reachability: A brief overview and recent advances. In *Proc. IEEE Conf. on Decision and Control*, 2017.

Somil Bansal, Varun Tolani, Saurabh Gupta, Jitendra Malik, and Claire Tomlin. Combining optimal control and learning for visual navigation in novel environments. *arXiv preprint arXiv:1903.02531*, 2019.

Paul Drews, Grady Williams, Brian Goldfain, Evangelos A. Theodorou, and James M. Rehg. Aggressive deep driving: Combining convolutional neural networks and model predictive control. In *CoRL*, 2017.

Paul Drews, Grady Williams, Brian Goldfain, Evangelos A Theodorou, and James M Rehg. Vision-based high-speed driving with a deep dynamic observer. *IEEE Robotics and Automation Letters*, 2019.

Jaime F. Fisac, Anayo K. Akametalu, Melanie N. Zeilinger, Shahab Kaynama, Jeremy Gillula, and Claire J. Tomlin. A General Safety Framework for Learning-Based Control in Uncertain Robotic Systems. *IEEE Transactions on Automatic Control*, 64(7):2737–2752, Jul. 2019. doi: 10.1109/TAC.2018.2876389.

J. Fuentes-Pacheco, J. Ruiz-Ascencio, and J. M. Rendón-Mancha. Visual simultaneous localization and mapping: a survey. *Artificial Intelligence Review*, 2015.

Dhiraj Gandhi, Lerrel Pinto, and Abhinav Gupta. Learning to fly by crashing. In *IROS*, 2017.

Saurabh Gupta, Varun Tolani, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. *arXiv preprint arXiv:1702.03920*, 2017.

Sunggoo Jung, Sunyou Hwang, Heemin Shin, and David Hyunchul Shim. Perception, guidance, and navigation for indoor autonomous drone racing using deep learning. *IEEE Robotics and Automation Letters*, 2018.

Gregory Kahn, Adam Villaflor, Vitchyr Pong, Pieter Abbeel, and Sergey Levine. Uncertainty-aware reinforcement learning for collision avoidance. *arXiv preprint arXiv:1702.01182*, 2017.

Katie Kang, Suneel Belkhale, Gregory Kahn, Pieter Abbeel, and Sergey Levine. Generalization through simulation: Integrating simulated and real data into deep reinforcement learning for vision-based autonomous flight. *arXiv preprint arXiv:1902.03701*, 2019.

Elia Kaufmann, Antonio Loquercio, Rene Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. Deep drone racing: Learning agile flight in dynamic environments. In *CoRL*, 2018.

Elia Kaufmann, Mathias Gehrig, Philipp Foehn, René Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. Beauty and the beast: Optimal methods meet learning for drone racing. In *ICRA*, 2019.

Arbaaz Khan, Clark Zhang, Nikolay Atanasov, Konstantinos Karydis, Vijay Kumar, and Daniel D Lee. Memory augmented control networks. In *ICLR*, 2018.

Dong Ki Kim and Tsuhan Chen. Deep neural network for real-time autonomous indoor navigation. *arXiv preprint arXiv:1511.04668*, 2015.

Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

Antonio Loquercio, Ana I Maqueda, Carlos R del Blanco, and Davide Scaramuzza. Dronet: Learning to fly by driving. *IEEE Robotics and Automation Letters*, 3(2):1088–1095, 2018.

Xiangyun Meng, Nathan Ratliff, Yu Xiang, and Dieter Fox. Neural autonomous navigation with Riemannian motion policy. *arXiv preprint arXiv:1904.01762*, 2019.

Matthias Müller, Alexey Dosovitskiy, Bernard Ghanem, and Vladen Koltun. Driving policy transfer via modularity and abstraction. *arXiv preprint arXiv:1804.09364*, 2018.

Yunpeng Pan, Ching-An Cheng, Kamil Saigol, Keuntaek Lee, Xinyan Yan, Evangelos Theodorou, and Byron Boots. Agile off-road autonomous driving using end-to-end deep imitation learning. In *RSS*, 2018.

Benjamin Recht. A tour of reinforcement learning: The view from continuous control. *Annual Review of Control, Robotics, and Autonomous Systems*, 2018.

Charles Richter, William Vega-Brown, and Nicholas Roy. Bayesian learning for safe high-speed navigation in unknown environments. In *Robotics Research*, pages 325–341. Springer, 2018.

Fereshteh Sadeghi and Sergey Levine. (CAD)^{2}RL: Real single-image flight without a single real image. In *RSS*, 2017.

R. Takei, R. Tsai, Haochong Shen, and Y. Landa. A practical path-planning algorithm for a simple car: a Hamilton-Jacobi approach. In *Proc. American Control Conference*, 2010.

Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

Rahee Walambe, Nipun Agarwal, Swagatu Kale, and Vrunda Joshi. Optimal trajectory generation for car-type mobile robot using spline interpolation. *IFAC*, 49(1):601 – 606, 2016.

Insoon Yang, Sabine Becker-Weimann, Mina J. Bissell, and Claire J. Tomlin. One-shot computation of reachable sets for differential games. In *Proc. ACM Int. Conf. Hybrid Systems: Computation and Control*, 2013.

Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *ICRA*, 2017.