Technical Report
Sim-to-Real: Virtual Guidance for Robot Navigation

Evan Luo*, Yu-Wen Chen*, Kai-Chen Lin*, Chao-Hsien Ting*, Hao-Kang Liu*, Chu-Chi Chiu*, Yu-Wei Chu*, Chien Liu, Hsin-Wei Hsiao, and Chun-Yi Lee

Abstract

We present an effective, easy-to-implement, and low-cost modular framework for completing complex navigation tasks. Our proposed method is based on a single monocular camera to localize, plan, and navigate. A localization module in our framework first localizes and acquires the robot's pose, which is then forwarded to our planner module to generate a global path and its intermediate waypoints. This information along with the pose of the robot is then reinterpreted by our framework to form the "virtual guide", which serves as a virtual lure for enticing the robot to move toward a specific direction. We evaluate our framework on a Husky robot in a number of virtual and real-world environments, and validate that our framework is able to adapt to unfamiliar environments and demonstrate robustness to various environmental conditions.

I. Introduction

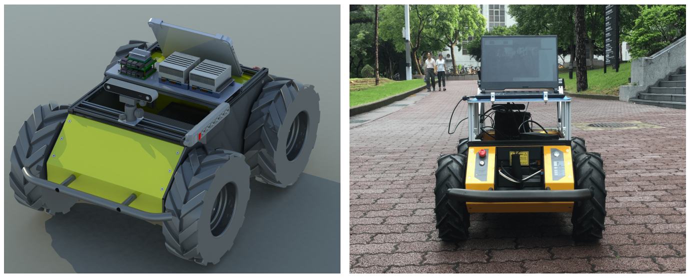

(a) Husky CAD model. (b) Outdoor navigation.

Fig. 1: The Husky platform used in this work.

Autonomous navigation has been attracting attention in recent years for controlling autonomous guided vehicles (AGV), as self-driving cars [1], autonomous drones, and domestic delivery robots have become increasingly prevalent in everyday life. Modern approaches for autonomous navigation have conventionally been accomplished with multiple sensor fusion, mostly splitting the task into two basic behaviors: local planning, which creates trajectories from the robot's current position to a given local target; and path following, which guides the robot to follow a general direction to reach a desired goal. A popular means for achieving such an objective is the adoption of LIDAR, which is able to deliver high accuracy and promises stability. Although LIDAR is widely used and studied, its high cost makes it infeasible for deployment or low-budget projects. On the other hand, vision-based navigation, which makes use of rich information from the unstructured physical world, offers an alternative for lowering the expenses. Interpreting and representing visual inputs to perform actions and interact with objects, however, is especially challenging in unstructured environments, as colored images are typically complex and noisy [2, 3]. This makes designing rule-based robots which satisfy the requirements of comprehending visual scene semantics and navigating to specific desired destinations especially difficult.

A number of learning-based approaches have been proven effective [4–6] in a variety of vision-based robotic navigation tasks. While these approaches have been demonstrated effective, they generally require huge amount of training samples. Collecting training data in real-world environments, however, has long been considered a challenging task, for preparing such amounts of labeled data is time-consuming and needs significant amounts of manual labor. In addition, learning-based navigation approaches, such as deep reinforcement learning (DRL) methods [6–9], typically require extensive trial-and-error experiences to learn an effective navigation policy. This limits the efficiency of the DRL algorithms, and restricts their applicability to fragile robots.

A popular approach for accelerating the training process and reducing the cost is to train the robots using simulators. There have been a number of recent research works in the literature on training navigation policies in virtual worlds [10–14]. However, the discrepancies between virtual and real worlds prohibit an agent trained in the virtual world from being transferred to the real world directly. To deal with this problem, researchers have adopted methods such as domain adaptation (DA) [15–17] and

* indicates equal contribution.

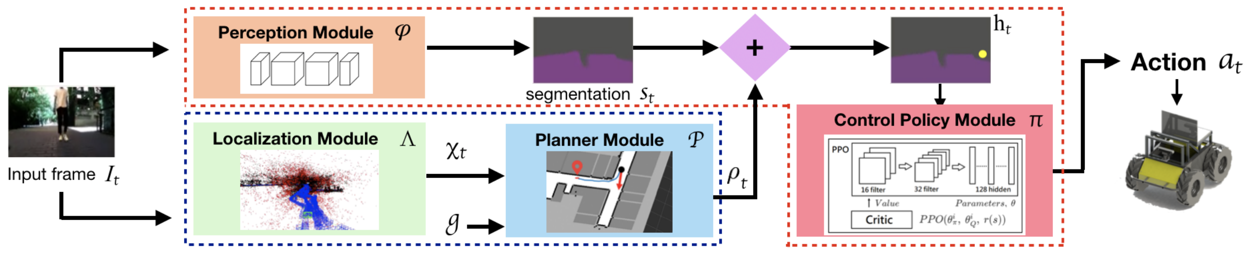

Perception Module φ
segmentation s_t
h_t
Action a_t
Input frame I_t
Localization Module Λ
χ_t
g
Planner Module $\mathcal{P}$
ρ_t
Control Policy Module π
PPO
16 filter
32 filter
128 hidden
Value
Parameters, θ
Critic
$PPO(\theta'_\pi, \theta'_Q, r(s))$

Fig. 2: Overview of the proposed framework.

domain randomization (DR) [4, 17–19] to close the sim-to-real reality gap [20, 21]. Unfortunately, collecting the real-world data required by DA for control policy learning is exceptionally time-consuming. Although DR does not require real-world data, the technique lacks a systematic way to determine which parameters are required to be randomized. Another branch of methods attempt to deal with the same problem by training their navigation agents in high-fidelity virtual environments [12, 22]. However, such approaches necessitate considerable time and efforts for building environments. Moreover, the policies trained by such approaches are only applicable to those pre-built environments in most cases, limiting the generalizability and transferability of the policies learned by the agents.

To resolve the above issues, we propose a new modular framework for addressing the reality gap in the vision domain and navigating a robot via virtual signals. Our robot uses a single monocular camera for navigation, without assuming any usage of LIDAR, stereo camera, or odometry information from the robot. The proposed framework consists of four modules: a localization module, a planner module, a perception module, and a control policy module. The localization module is responsible for estimating the current pose of the robot from the visual input, and conveys it to the planner module. The planner module then constructs a path between the robot's current location and the desired goal, defining a global direction for the robot to follow. This global direction is then reinterpreted by our framework as the tendency, which is rendered on the semantic segmentation (which serves as the meta-state representation relating the modules in our framework [20]) generated by the perception module as the "virtual guide" for the control policy module. The virtual guide is depicted as a yellow ball in this work, however, it is not restricted to any specific form of representation. In the proposed framework, the role of the virtual guide is similar to a carrot (i.e., a lure) for enticing the robot to move toward a specific direction. The control policy module is implemented as a DRL agent, and is trained completely in simulated environments, with an aim to learn a policy for chasing the virtual guide and avoiding obstacle collision. While in the execution phase, the control policy module receives actual segmented images with virtual guidance, enabling the agent to complete obstacle-avoidance and guide-following tasks simultaneously in the real world. Compared to conventional navigation approaches, our methodology is not only highly adaptable to diverse environments, but is also generalizable to complex scenarios.

- A straightforward, easy to implement, and effective modular learning-based framework for dealing with the challenging robot navigation problem in the real world.
- An architecture that separates the vision-based robotic learning model into a perception module, a control policy module, a planner module, and a localization module.
- A concept of altering the behavior policy of a robot by adjusting the meta-state representation of it (Section II).
- A virtual guidance scheme for transforming the instruction provided by the planner module to a virtual guide for navigating the robot to the target goal (Section II-C).
- An approach for balancing obstacle avoidance and path following so as to accomplish complex navigation tasks.

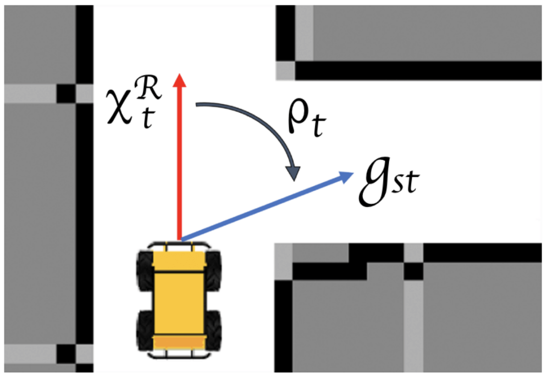
The rest of this paper is organized as follows. Section II walks through the proposed methodology in detail. Section III describes the evaluation setup. Section IV presents the experimental results and their implications. Section V concludes. The background material related to this paper is included in our supplementary material [23].

II. Proposed Methodology

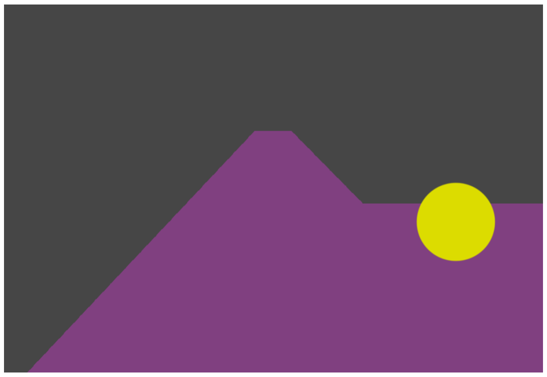
In this section, we present the architecture of the proposed methodology. We first provide an overview of the entire framework, followed by a detailed description of the components in it. Next, we introduce the proposed virtual guidance scheme. Finally, we walk through the training procedure of the control policy module as well as the pseudo-code of it.

A. *Overview of the Framework*

The main objective of the proposed framework is to navigate the robot to a designated target goal g while avoiding the obstacles on its way using the input RGB frames captured by a single monocular camera. Fig. 2 illustrates an overview of the proposed modular architecture, which comprises four distinct modules: a localization module Λ, a planner module P, a perception module ϕ, and a control policy module π. For an input RGB frame I perceived at time t (denoted as I_t),



(a) The robot's pose.



(b) The virtual guide.

Fig. 3: The proposed virtual guidance scheme. Please note that g_{st} represents the short term goal, as discussed in Algorithm 1.

ϕ produces a semantic image segmentation s_t, which serves as the meta-state representation for relating the latter three modules P, ϕ, and π. Meanwhile, Λ performs localization for the robot according to the information contained in I_t, and produces a pose χ_t (in terms of the coordinate χ_t^T and quaternion orientation χ_t^R of the robot) to be fed into P. Based on g and χ_t, P estimates a potential route to g and generates a tendency ρ_t, which serves as a high-level guided direction for the robot to navigate to g. The tendency ρ_t is then rendered on s_t as a yellow ball (denoted as the "virtual guide") to form a new representation h_t. The control policy module takes h_t as its input and reacts with an action $a_t \in A$ according to π at t, where A is the action space of the robot.

B. Components of the Framework

In this section, we walk through the implementation details of the four modules Λ, P, ϕ, and π in separate paragraphs.

Localization module (Λ). The main objective of Λ is to estimate the global pose $\chi_t = \Lambda(I_t)$ from I_t for P to utilize. The localization algorithm employed by Λ is ORB-SLAM2 [24]. To enable Λ to perform localization, the map is pre-built, and an independent tracking thread is used for localizing and estimating χ for every I. The localization algorithm deals with two scenarios. If feature tracking is successful for the previous frame (i.e., I_{t-1}), Λ uses a constant velocity motion model on the map points observed in I_{t-1} to predict χ_t [24]. Otherwise, Λ initializes pose estimation from all keyframes, converts I_t into a bag of words, and queries a database for the nearest keyframe to estimate χ_t.

Planner module (P). The objective of P is to generate a potential path according to χ_t and g, and derives a ρ_t to render the virtual guide for the robot (discussed in Section II-C). The path-planning strategy employed by P is Dijkstra's algorithm [25]. The generated path consists of a sequence of intermediate way-points in a known space, and is collision-free considering only static obstacles in the pre-built map. Dynamics obstacles, including moving objects and static objects not in the pre-built map, are avoided by π. Please note that in our work, P does not exploit any odometry information from the robot.

Perception module (ϕ). The main function of the perception module is to generate s_t from I_t. The perception module can be any semantic segmentation model, and is expressed as $s_t = \phi(I_t, \theta^\phi)$, where θ^ϕ denotes the parameters of ϕ. The architecture and training dataset of ϕ is described in Section III. The advantages of using s_t as the meta-state representation is threefold. First, although the visual appearances of the images from virtual and real worlds differ, their semantic segmentations are almost identical [20, 26]. This enables us to use s_t to bridge the reality gap [20]. Second, s_t preserves only the most important information of objects in I_t, resulting in a more structured representation. It also offers much richer information than other sensory modalities, as they typically fail to cover crucial high-level scene semantics. Third, ϕ only requires a single monocular camera, which is relatively cheaper than depth cameras or laser rangefinders.

Control policy module (π). The function of the control policy module is to generate a_t (in terms of linear and angular speeds) for the robot, with an aim to eventually navigate the robot to g. In this work, π is implemented as a DRL agent trained by proximal policy optimization (PPO) [27] that takes h_t as its input. During the training phase, π is trained completely in simulated environments, and learns an effective policy to chase the virtual guide and avoid static and dynamic obstacles directly from h_t rendered by a low-fidelity simulator. In the execution phase, π instead receives h_t rendered from s_t and ρ_t, enabling sim-to-real transferring of π. In the real world, our framework executes π as the local planner for avoiding obstacles, while following the instructions from P in the manner of virtual guidance. The scheme allows our robot to balance obstacle avoidance and path following so as to accomplish complex navigation tasks.

Algorithm 1 Virtual guidance scheme

```
Input: $g \in G_{space}$: goal, where $G_{space}$ is the goal space.
Input: $t$: current timestep.
Input: $I_t$: current input frame.
Input: $p_a \in$[0,1]: angle threshold.
Input: $N$: number of virtual guide in a candidate pool.
Output: $h_t$: rendered segmentation with virtual guidance.
 1: $\chi_t$ = ($\chi_t^T$, $\chi_t^R$) $\leftarrow \Lambda(I_t)$ /* Localization module estimates a pose $\chi_t$.*/
 2: /* $\chi_t^T$ denotes position and $\chi_t^R$ denotes orientation. */
 3: path $\leftarrow P(\chi_t^T, g)$
 4: $g_{st} \leftarrow$ choose a short term goal from the path.
 5: $\rho_t \leftarrow$ compute the rotation angle from $\chi_t^R$ to $g_{st}$.
 6: if $\rho_t > p_a$ then
 7:     VG $\leftarrow$ LEFT // VG stands for virtual guide.
 8: else if $\rho_t < -p_a$ then
 9:     VG $\leftarrow$ RIGHT
10: else
11:     VG $\leftarrow$ STRAIGHT
12: end if
13:
14: $s_t \leftarrow \phi(I_t, \theta^{\phi})$
15: for $i = 1, \cdots$ N do
16:     $c \leftarrow$ choose a region from [VG] candidate pool.
17:     if $c$ is free for the virtual guide. then
18:         $h \leftarrow s + c$
19:         break
20:     end if
21: end for
22: if there is no candidate. then
23:     $h_t \leftarrow s_t$ /* let RL agent drive a while. */
24: end if
25: return $h_t$
```

Algorithm 2 Training procedure of the control policy module

```
Property: $n_a$: Number of agents.
Property: $n_e$: Number of maps.
Property: $R$: Reward function.
Input: $H$: Horizon of timestep.
Input: $M$: Set of multiple maps.
Input: $T_1 \cdots T_{n_e}$: Set of target's goal positions for each map.
Input: $\mu$: Hyper-parameters configuration.
Input: $p \in [0, 1]$: Success threshold.
Output: $\pi$: DRL agent's Policy.
Output: $\theta_\pi, \theta_Q$: Parameters of DRL model.
 1: Copy $\theta_\pi, \theta_Q$ for each agent in $n_a$.
 2: for $i = 1, \cdots n_a$ running in parallel do
 3:     Choose a map $m \in M$ randomly.
 4:     Set target goal randomly $g \in T_m$, $s \leftarrow start\ point\ s_s$.
 5:     for $j = 1, \cdots H$ do
 6:         while agent not done do
 7:             Multinomial sampling $a$ from $\pi(s)$, $s_t \leftarrow m(a)$.
 8:             if $distance(s, g) < p$ then
 9:                 $R \leftarrow R + 1$
10:             else if $collision()$ then
11:                 $R \leftarrow R - 1$
12:             end if
13:         end while
14:         $\pi^i = PPO(\theta_\pi^i, \theta_Q^i, r(s))$ // Policy update.
15:     end for
16: end for
17: Perform asynchronous update $\theta_\pi$ using $d\theta$ of $\theta_\pi^i$ and $\theta_Q$ using $d\theta$ of $\theta_Q^i$.
18: return $\pi, \theta_\pi, \theta_Q$
```

C. Virtual Guidance Scheme

As π is trained to chase the virtual guide in simulated environments, the virtual guidance scheme in the real world aims at rendering a virtual guide on suitable locations of s_t such that the robot is able to follow the path derived by P and navigate to g. Fig. 3 depicts the proposed virtual guidance scheme. Fig. 3 (a) illustrates the robot, χ_t, and ρ_t, while Fig. 3 (b) shows the rendered h_t to be fed into the π. The virtual guide used in this work is a 2-D yellow ball. In order to dynamically adjust the position and size of the virtual guide, a pool of candidate locations are pre-defined. To determine where the virtual guide should be rendered, we combine χ_t estimated by Λ with the short term goal g_{st} that P produces to calculate ρ_t. The target location of the virtual guide is selected from the candidate pool accordingly based on ρ_t and whether obstacles exist in relevant areas. If no location on s_t is appropriate for placing the virtual guide (i.e., too many obstacles), we leave π to concentrate on obstacle avoidance first without rendering the virtual guide, until the desired direction becomes navigable again. The pseudo-code of the proposed virtual guide scheme is summarized in Algorithm 1.

D. Training Procedure of the Control Policy Module

The training procedure of our control policy is presented in Algorithm 2. The base algorithm used for training π is PPO [27]. A number of training maps are pre-designed as our training environments, and is discussed in detail in Section III. During the training phase, multiple agents are trained simultaneously with randomly chosen maps and randomly placed goals. The experiences collected by these agents are used to update a common central policy in an asynchronous fashion. Our reward function is designed such that a one-time reward of '$+1$' is awarded to the agent if the distance between the virtual guide and the agent is within a small threshold d_{th}. Otherwise, no reward is given to the agent. If the agent collides with an obstacle halfway, it is penalized by a reward of '-1' each time. Note that the virtual guide disappears immediately if the agent reaches it.

III. Evaluation Setup

In this section, we present the evaluation setup in detail. We evaluate the proposed framework both in virtual and real environments. We begin with explaining the robot and computational platforms, followed by a description of the simulator used for training and evaluation. Then, we provide an overview of the training environments used in this work. We suggest the interested readers to refer to the supplementary material [23] for the rest of our hyper-parameter settings.

A. Robot and Hardware Setups

Husky. We use Clearpath Husky Autonomous Ground Vehicle (AGV) as our robot platform. The size of Husky is 990 $\times$ 670 $\times$ 390 mm (L $\times$ W $\times$ H), and its maximum speed is 1.0 m/s, with a payload of 75 Kg and a climbing grade of 45 degrees. A single monocular camera and a laptop computer is mounted on top of it for executing our framework. An illustration of our Husky platform is presented in Fig. 1.

Computational platform. We run our framework on a laptop equipped with an Intel i7-9750H CPU and an NVIDIA GTX 1660Ti GPU with 16G of RAM. The operating system we use is Ubuntu 16.04. We use robot operating system (ROS) [28] as the interface between our framework and the Husky platform.

B. Training Environment Setup

Simulator. We use the Unity3D[2] engine to develop the virtual environments for training our DRL agent (i.e., π). In the simulated environments, the agent receives its observations in the form of semantic segmentation rendered by a segmentation shader from a first-person perspective. During the training phase, yellow balls are placed in the environments to train the agent to follow the virtual guide, The geometric size and shape of our simulated agent is specifically configured to be similar to that of the real Husky.

ML-Agents [29]. We train our agent with Unity ML-Agents[3], which is an open-source Unity plugin that enables games and simulations to serve as environments for training DRL agents.

Training environments. The environments used to train navigation tasks highly affect the behavior of the DRL agent. Large roadmaps and complex routes confuse agents, while small, relatively easy maps result in agent behavior overfitting, leading to poor performance when facing unseen maps. To address these shortcomings, our model is trained using twenty independent maps. At the beginning of each episode, a map is randomly selected, with the agent and the goal placed at randomly determined locations. The twenty random maps are depicted in Fig. 4. This training approach allows our agent to learn a policy that is robust to complex real-world scenarios. To resemble more closely to virtual guidance given in real-world navigation, we place a variety of obstacles such as furniture, trees, and pedestrians in our training maps, and set the agent's field of view in Unity to 60 degrees, which is the same as our monocular camera mounted on Husky. We did not apply time penalty during training, as it may cause a higher chance of colliding into obstacles when the reward goal is in sight. Last of all, despite that two-dimensional action space (consisting of right and left turns) converges faster, we use three-dimensional action space adding the choice of going forward. This enables the agent to learn a policy that can handle complex scenarios.

IV. Experimental Results

In this section, we present the experimental results and discuss their implications. We evaluate the performance of the proposed framework for virtual and real-world indoor and outdoor navigation tasks. In Section IV-A, we report our results evaluated in the simulated evaluation environments. Then, in Sections IV-B and IV-C, we demonstrate our results in real-world indoor and outdoor environments, respectively. Finally, we analyze the performance of our system in real-world range-based obstacle avoidance tasks in Section IV-E.

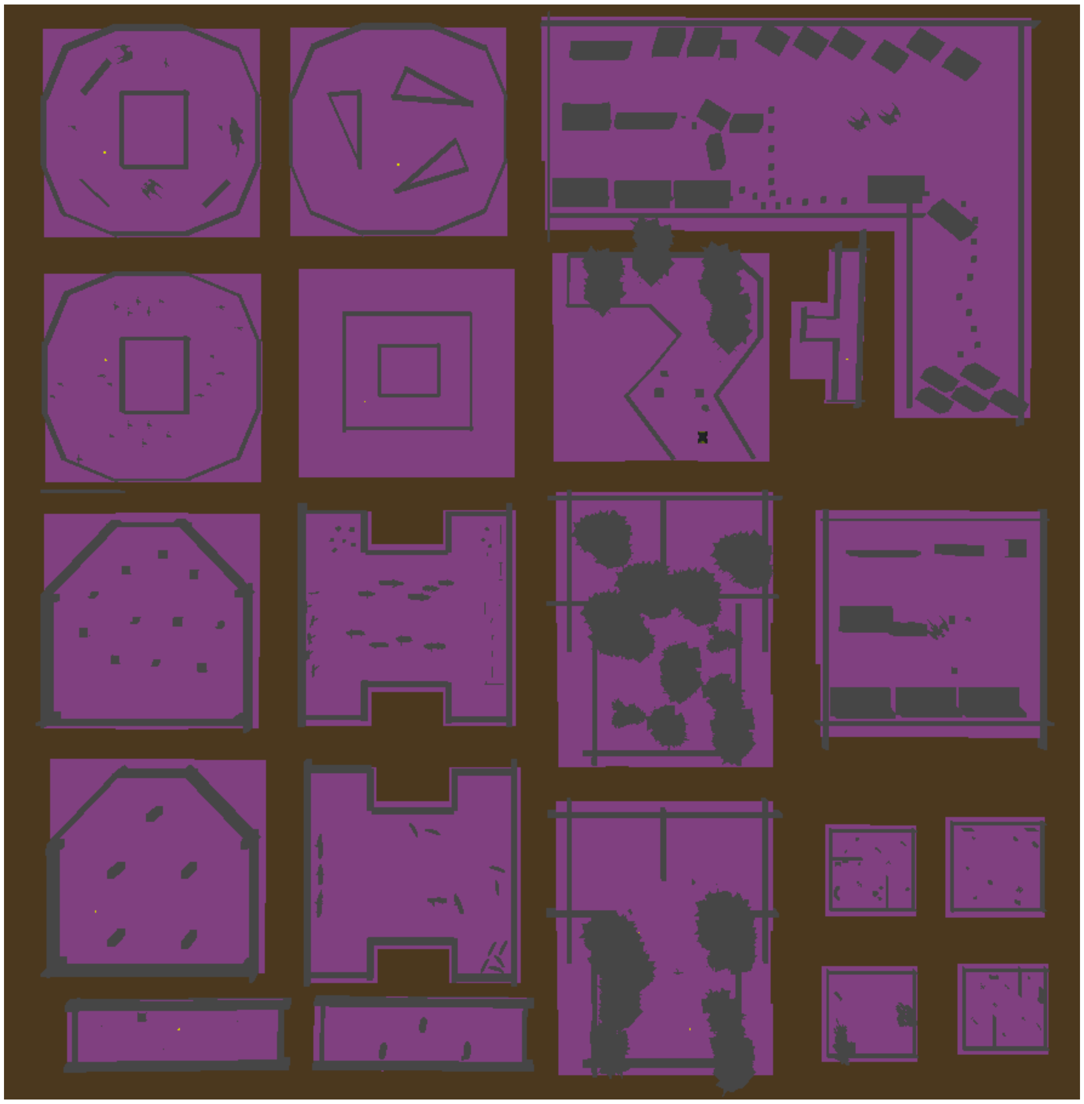


Fig. 4: The maps used for training our π. This figure illustrates the bird's-eye view of our twenty different training maps mentioned in Section III. We set up a variety of obstacles, such as chairs, desks, trees, and creatures to simulate real-world scenarios. We also adjust the color of the virtual environments in order to match the output of the perception module ϕ.

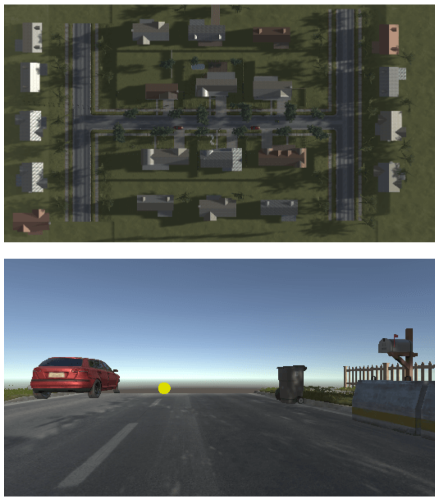
TABLE I: Evaluation in the virtual environments.

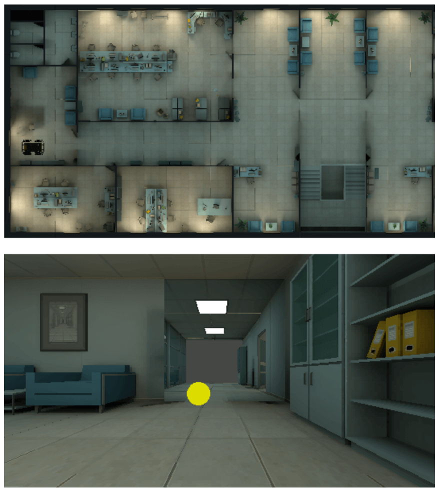
Environment	Success Rate (%)	Duration(s)	
		μ	σ
Suburban	76.56	405.4	7.486
Race track	92	335.8	2.39
Office	94	176.1	2.02

A. *Performance in the Simulated Environments*

The main objective of this experiment is to verify that our model can adapt to new, never-seen surroundings that are several times larger than our training environments, thus eliminating the need for retraining. We first evaluate our model in simulated environments, where s_t is directly provided by the segmentation shader. To achieve environmental diversity, we select three different virtual environments with various scenes shown in Fig. 5. The environments include: Fig. 5 (a) a suburban area, Fig. 5 (b) a dimly lit office, and Fig. 5 (c) a race track. The top row correspond to the bird's-eye view of the environments, and the bottom row correspond to them viewed from the first-person perspective. For each map, we test our model at least 100 times and summarize the results in Table I. Overall, our model achieves at least 75% success rates in

[2]Unity3D: https://unity3d.com/
[3]ML-Agents: https://unity3d.com/machine-learning


(a) Suburban


(b) Office

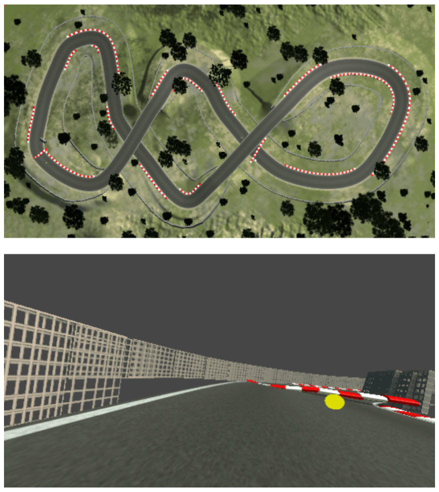
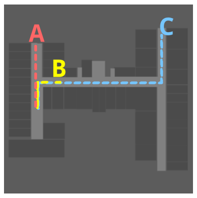
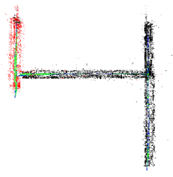
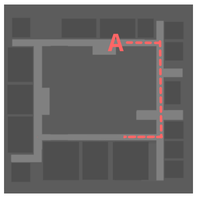

(c) Race track

Fig. 5: The virtual environments used for evaluation.

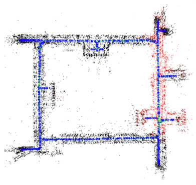


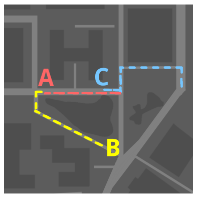
(a) Bldg 1 floorplan


(b) Bldg 1 orb-slam



(c) Bldg 2 floorplan


(d) Bldg 2 orb-slam



(e) Outdoor map

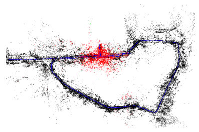

(f) Outdoor orb-slam

Fig. 6: Real-world indoor and outdoor evaluation environments.

each virtual environment. This validates that our model is able to simultaneously avoid obstacles and act accordingly with our virtual guidance scheme even in unfamiliar environments.

B. Performance in the Real-World Indoor Environments

The environments used for indoor navigation tasks consist of two buildings. Fig. 6 (a)(c) shows the buildings' floor plans with routes plotted with dotted lines. Our routes consist of straight, right turn, left turn, and multiple turn routes to demonstrate common agent behaviors. Fig. 6 (b)(d) shows the corresponding maps pre-built by ORB-SLAM2 [24] used for Λ to localize the robot. Three routes from Building 1 and one route from building two is selected for evaluation. Ten to fifteen dynamic obstacles are placed along the robot's route in building 2, while no dynamic obstacles is set in Building 1. Each route is evaluated ten times. The task is considered successful is the robot reaches the goal without colliding with any obstacles. The evaluation results are shown in Table II, including the path length, success rate, average duration, and its variance. Our system achieves an average success rate of 100% on all three routes of Building 1 and 81.82% on the route of Building 2. We assume that the drop in success rate is primarily due to the difficulty to avoid dynamic obstacles along the narrow corridor of Building 2. The results prove that our system is efficient enough to follow the desired path without colliding into obstacles, and possesses adaptability in different indoor environments that our Husky could fit in.

C. Performance in the Real-World Outdoor Environments

The environments used for outdoor navigation tasks include rugged terrains and sidewalk-road transitions. Fig. 6 (e) shows the map of our environment and the three evaluation routes (Routes A, B, and C). The map generated by ORB-SLAM2 [24] is shown in Fig. 6 (f). The outdoor environment is challenging due to the fact that the routes we selected for evaluation are not roads with straight turns and are often cluttered with pedestrians and other obstacles. Road size also fluctuates as roads transition to sidewalks. The evaluation results are presented in Table III, including the path length, success rate, average duration, and variation. Our system achieves a success rate of 99% on Route A, 80% on Route B, and 70% percent on Route C. Please note that our π is the same for both indoor and outdoor evaluation tasks.

TABLE II: Evaluation in the real-world indoor environments.

Environment		Length (m)	Success Rate (%)	Duration(s)	
				μ	σ
Building1	Route A	16.96	100	47.761	1.557
Building1	Route B	10.06	100	48.773	0.502
Building1	Route C	55.12	100	194.147	1.585
Building2	Route A	48.53	81.82	116.278	1.257

TABLE III: Evaluation in the real-world outdoor environments.

Environment	Length (m)	Success Rate (%)	Duration(s)	
			μ	σ
Route A	63.32	99	222.891	24.186
Route B	86.88	80	322.943	8.058
Route C	92.06	70	356.047	23.232

The success rates of the three routes demonstrate our system's ability to navigate through diverse terrains. Fig. 7 shows the example first-person views of the three routes. Route A features heavy pedestrian flow (Fig. 7 (a)), which tests competence in avoiding dynamic obstacles. Route B's large to small road transition (Fig. 7 (b)) shows the capability of handling changes in observation without driving off-road. Route C contains a wide roadblock that was not included in the map pre-built by ORB-SLAM2 [24] (Fig. 7 (c)), which inspects whether our local planner has the capability of finding an alternate path. Routes B and C both encounter forked roads (Figs. 7 (b)(d)), which tests whether the robot can opt for the right path. The routes selected for our test also have different shading, localization accuracy, and segmentation stability.

D. Analysis of the Trajectories Navigated by Husky

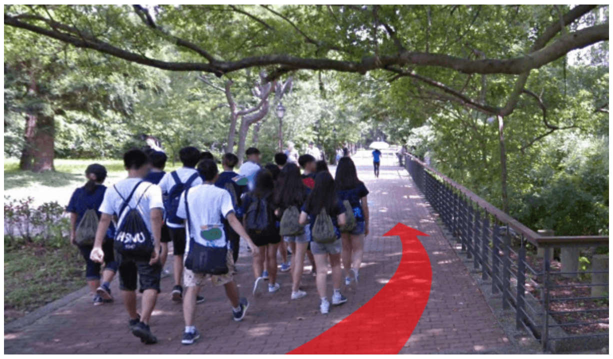
Fig. 8 plots the trajectories navigated by Husky for the real-world indoor and outdoor evaluation environments discussed in Sections IV-B and IV-C. Ten trajectories are plotted for each case. It is observed that the trajectories are similar for each case, qualitatively demonstrating the small variance in duration presented in Tables II and II. These show that our Husky can navigate stably in indoor and outdoor scenarios.

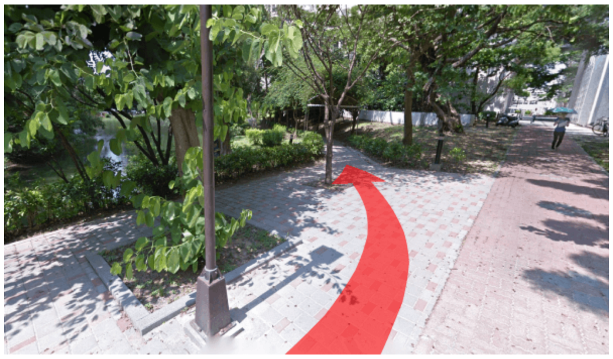
E. Performance in Range-Based Obstacle Avoidance Tasks

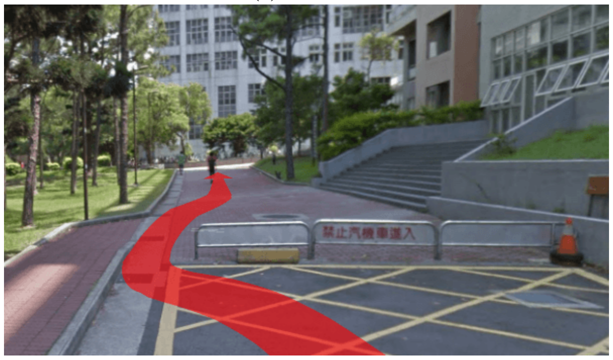
This experiment aims to demonstrate the robustness of our system when faced with dynamic obstacles and other variable conditions such as inconsistent lighting and pedestrian density. Our average pedestrian density is up to 5.4 people per minute on straight route tests from 10 to 100 meters. The results are reported in Table IV, including the success rate and the average duration Our model achieves at least 77.8% success rates in each cases. This conveys that when our robot strays from the original path to avoid dynamic obstacles, virtual guidance is able to lead it back and resumes its original task.

V. Conclusions

In this paper, we presented a straightforward, easy to implement, and effective modular framework using only a single monocular camera to handle the challenging robot navigation problem in the real world. We achieved this objective by introducing a virtual guidance scheme, which employs the virtual guide to navigate the robot's policy to its destination. We performed extensive experiments in diverse indoor and outdoor maps, and verified that our method is robust to various environmental conditions and generalizable to unfamiliar maps both in the virtual and real-world tasks.


(a) Route A


(b) Route B



(c) Route C

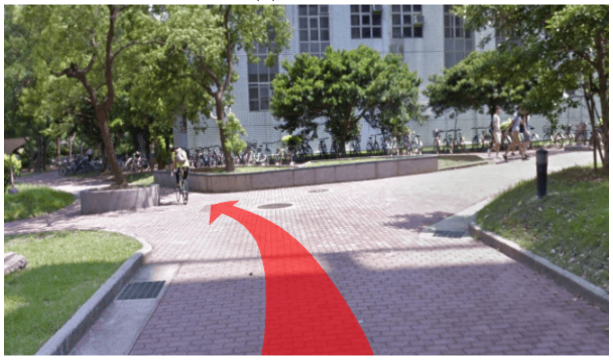
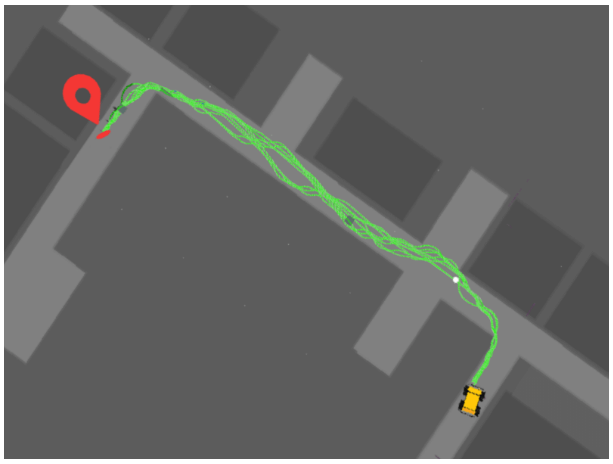

(d) Route C

Fig. 7: The first-person views of the three outdoor routes.


(a) Building 2 (Route A)

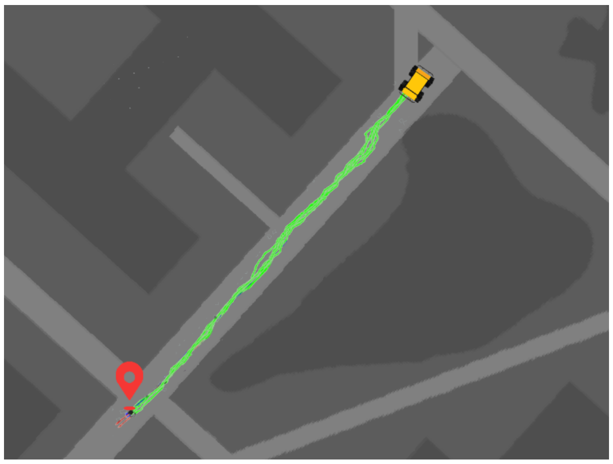

(b) Outdoor (Route A)

Fig. 8: Illustration of the trajectories navigated by Husky.

References

[1] A. Seff and J. Xiao. “Learning from Maps: Visual Common Sense for Autonomous Driving”. In: *ArXiv* abs/1611.08583 (2016).

[2] D. Maier, A. Hornung, and M. Bennewitz. “Real-time navigation in 3D environments based on depth camera data”. In: *Proc. Int. Conf. Humanoid Robots (Humanoids)*. Nov. 2012, pp. 692–697.

[3] J. Biswas and M. V. Manuela. “Depth camera based indoor mobile robot localization and navigation”. In: *Proc. Int. Conf. Robotics and Automation (ICRA)*. May 2012, pp. 1697–1702.

[4] F. Sadeghi and S. Levine. “CAD^{2}RL: Real Single-Image Flight without a Single Real Image”. In: *CoRR* abs/1611.04201 (2016).

TABLE IV: Evaluation in obstacle avoidance tasks.

Route Length (m)	Success Rate (%)	Duration(s) μ	Duration(s) σ
10	100	36.77	1.569
20	83	74.09	3.144
40	80	114.40	3.284
100	77.8	397.196	8.652

[5] C. Finn and S. Levine. "Deep Visual Foresight for Planning Robot Motion". In: *CoRR* abs/1610.00696 (2016).

[6] S. Gupta et al. "Cognitive Mapping and Planning for Visual Navigation". In: *CoRR* abs/1702.03920 (2017).

[7] M. Luber et al. "Socially-aware robot navigation: A learning approach". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Oct. 2012, pp. 902–907.

[8] S. Brahmbhatt and J. Hays. "DeepNav: Learning to navigate large cities". In: *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*. July 2017, pp. 3087–3096.

[9] Y. Duan et al. "RL^2: Fast Reinforcement Learning via Slow Reinforcement Learning". In: *ArXiv* abs/1611.02779 (2017).

[10] A. Faust et al. "PRM-RL: Long-range Robotic Navigation Tasks by Combining Reinforcement Learning and Sampling-based Planning". In: *CoRR* abs/1710.03937 (2017).

[11] A. Pokle et al. "Deep Local Trajectory Replanning and Control for Robot Navigation". In: *CoRR* abs/1905.05279 (2019).

[12] Y. Wang, H. He, and C. Sun. "Learning to Navigate Through Complex Dynamic Environment With Modular Deep Reinforcement Learning". In: *IEEE Transactions on Games* 10.4 (Dec. 2018), pp. 400–412.

[13] H.-T. L. Chiang et al. "Learning Navigation Behaviors End to End". In: *CoRR* abs/1809.10124 (2018).

[14] A. Francis et al. "Long-Range Indoor Navigation with PRM-RL". In: *CoRR* abs/1902.09458 (2019).

[15] A. Ghadirzadeh et al. "Deep Predictive Policy Training using Reinforcement Learning". In: *CoRR* abs/1703.00727 (2017).

[16] A. A. Rusu et al. "Sim-to-Real Robot Learning from Pixels with Progressive Nets". In: *CoRR* abs/1610.04286 (2016).

[17] F. Zhang et al. "Sim-to-real Transfer of Visuo-motor Policies for Reaching in Clutter: Domain Randomization and Adaptation with Modular Networks". In: *ArXiv* abs/1709.05746 (2017).

[18] X. B. Peng et al. "Sim-to-Real Transfer of Robotic Control with Dynamics Randomization". In: *CoRR* abs/1710.06537 (2017).

[19] J. Tobin et al. "Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World". In: *CoRR* abs/1703.06907 (2017).

[20] Z. W. Hong et al. "Virtual-to-real: Learning to control in visual semantic segmentation". In: *Proc. Int. Joint Conf. Artificial Intelligence (IJCAI)*. July 2018, pp. 4912–4920.

[21] J.Tobin et al. "Domain randomization for transferring deep neural networks from simulation to the real world". In: *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*. Sept. 2017, pp. 23–30.

[22] A. Mousavian et al. "Visual Representations for Semantic Target Driven Navigation". In: *2019 International Conference on Robotics and Automation (ICRA)*. May 2019, pp. 8846–8852.

[23] *Sim-to-Real: Virtual Guidance for Robot Navigation Supplementary Material*. https://drive.google.com/open?id=1QyGCQIGFcLEPGTFeaupNKVZo2SOmrcJa. Accessed: 2020-09-30.

[24] R. Mur-Artal and J. D. Tardós. "ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras". In: *IEEE Trans. Robotics* 33.5 (June 2017), pp. 1255–1262.

[25] E. W. Dijkstra. "A note on two problems in connexion with graphs". In: *Numerische Mathematik* 1.1 (Dec. 1959), pp. 269–271.

[26] X. Pan et al. "Virtual to real reinforcement learning for autonomous driving". In: *Proc. The British Machine Vision Conference (BMVC)*. Sept. 2017.

[27] J. Schulman et al. "Proximal policy optimization algorithms". In: *arXiv:1707.06347* (Aug. 2017).

[28] M. Quigley et al. "ROS: An open-source robot operating system". In: *ICRA workshop on open source software*. May 2009.

[29] A. Juliani et al. "Unity: A General Platform for Intelligent Agents". In: *ArXiv* abs/1809.02627 (2018).