Indoor Multi-Sensory Self-Supervised Autonomous Mobile Robotic Navigation

Junhong Xu, Hanqing Guo, Shaoen Wu
Department of Computer Science
Ball State University
{jxu7, hguo, swu}@bsu.edu

***Abstract*—Autonomous robotic navigation in indoor environments is fairly challenging and important to industrial environments. Traditional map-based or mapless navigation methods often fail because of the unstructured characteristics of the environments. Recently, imitation learning using DAgger algorithm has been successfully applied to many real-world robotic tasks. However, it needs human operators to give correct control commands without feedback to overcome data distribution mismatch problem, which is always prone to error and expensive. In this paper, we propose a novel solution to eliminate the need of human manual labeling after the initial data collection in the task of imitating to navigate in indoor environments. This solution introduces *an imperfect policy* based on multi-sensor fusion and *a recording policy* that only records the data giving the most knowledge to the navigation policy. The recording policy mitigates the affect of learning too much from an imperfect policy. With extensive experiments in indoor environments, we demonstrate that after several iterations of learning, the robot is able to navigate through real-world hallways in both seen and unseen situations safely. In addition, we show that our system achieves near human performance in most of the tasks and even surpasses human performance in one out of three tasks. To the best of our knowledge, this is the first work that utilizes imperfect sensor measurements to perform self-supervised imitation learning in robotic navigation tasks.** [1]

I. Introduction

Indoor autonomous robotic navigation is highly challenging, but crucial to industrial environments. Such a navigation system can be used in many applications such as delivering and transporting items in warehouses where moving obstacles are presented. It requires robots to perceive and understand environments and predict next possible actions to avoid exceptions such as collision.

Traditional indoor robotic navigation methods are widely based on SLAM [25] that is a two-stage approach including perception and action stages. In the perception stage, a global map is built using on-board sensors. In the action stage, the map is given to a motion planner to predict possible actions [19] [24]. While this method allows a robot to navigate to a target location in cluttered environments based on a perfect global map, it is difficult to maintain the map because of inaccurate measurements of sensors, dynamic obstacle changes, and feature representation errors.

Another approach to autonomous robotic navigation is the end-to-end imitation learning. It has been applied to many robotic tasks such as learning to manipulate objects [18], driving to a target position [16], and self-driving car [3]. Imitation learning simplifies robotic tasks in such a way that the agent only needs to know current observations. It predicts corresponding control commands based on observations. However, one major drawback of imitation learning is that it breaks the i.i.d. assumption because future observations depend on previous actions [20]. To address this problem, DAgger algorithm or its variants [28] [11] is used to ask an expert policy to label all observations encountered in each training iteration without iterative feedbacks [20]. This performs well in simulated settings [18], [28] where the expert policy is easy to access. However, real-world environments often consist of noisy and inaccurate sensor measurements, which often result in defective or inaccurate expert policy.

Recent autonomous robotic navigation solutions employed deep reinforcement learning(DRL) to learn from trail and error [15]. Applying DRL to real-world robots directly is however not practical because DRL robots have to explore dangerous states that may damage them. Therefore, many works only apply DRL to navigation in simulated settings [7] and then transfer the learned knowledge to the real world [22], [23].

In this work, we aim at the challenge of autonomous robotic navigation in *real world*, specifically in indoor environments. We propose an imitation learning framework called ***M**ulti-**S**ensory **S**elf-Supervised **L**earning (MS2L)*. A key innovation in this work is to employ both imaging sensors and non-imaging sensors in the deep learning framework. Instead of manual labeling, *MS2L* uses an **imperfect** *sensor policy* to label the obser encountered by a mobile agent after the initial iteration. In addition, based on a *safety policy* [28], we design a *recording policy* in our framework that determines whether the current observation is worth learning from. We show in our experiments that this *recording policy* is crucial to our framework because the *sensor policy* can not be perfect in real-world and learning too much from this imperfect policy yields performance degradation. Compared to literature solutions, this work has the following novelties:

- it does not require a single human correction after the initial iteration;
- it trains and tests in real-world environments without the need of transferring knowledge from simulation to real-world;

[1] Video can be found at https://youtu.be/jnmtr0B9hcg

- it does not require multiple-camera configurations to record observations from different viewpoints.

In the rest of this paper, Section II reviews the related work, particularly imitation learning and reinforcement learning. Then, Section III discusses the design of our proposed framework. Next, Section IV presents the extensive evaluations of the *MS2L* in real indoor environment. The paper is concluded by Section V.

II. Related Work

This work combines DAgger algorithm with self-supervised learning, which covers broad fields of machine learning and robotics. We describe the most relevant works to ours and their connections to our framework.

A. Imitation Learning

Ross et al. [20] proposed an iterative training procedure called DAgger algorithm. This algorithm is widely used in robotics such as control [21], indoor navigation [6], and grasping [10]. One common idea these methods sharing is that this algorithm requires some expert policies to be queried from. However, these expert policies (usually human experts) are expensive to be queried from in real-world because of the unknown dynamics. In [21], pilots (human operators) are provided with partial feedback labeled on the collected images to reduce the burden of correcting actions without feedback. This offline labeling method still requires a supervisor. In [10], the authors explore the idea of using a hierarchy of different supervisors to reduce the burden of human expert in the task of learning to grasp objects in clutter. They use a motion planner, crowd-sourced supervisor, and expert supervisor. In contrast to their works, our work only requires a supervisor to give action commands in the initial training of DAgger algorithm.

The second approach using imitation learning in navigation is setting up multiple cameras to capture different scenes at the same time. In [5] and [3], they setup several cameras on the vehicle or drone aiming at different directions. They label each observation according to different camera positions, e.g. the observation recorded by the left camera is labeled as right. Although this method breaks the data mismatch problem in imitation learning, it still has two drawbacks. First, this labeling strategy only works on the problem on lane following. Second, Equipping multiple cameras may not be feasible to small robots.

There are some works that do not use DAgger algorithm in robotic navigation. In [4], the authors uses a drone to collide into objects and collect these collision data. Then they train a neural network on these data to learn a policy that avoids dangerous states. This approach is not application to fragile hardware as it needs robots to run into dangerous states thousands of times. Similarly, in [8], they do not use DAgger algorithm to learn drone control commands. They only use a naive method that learns from human demonstrators, which causes many failure cases.

[28] and [11] try to reduce times of querying from an expert policy in DAgger algorithm. In [28], they use an additional neural network to restrict the number of times to query from an expert supervisor. Similar to their approach, we adapt the *safety policy* network. However, the policy our robot imitates from is not perfect and we need additional constrains in *safety network*.

Compared to our previous work [27], this work analyzes action distribution after each iteration of self-supervised learning. It shows that our self-supervised learning framework is able to balance the dataset distribution without any supervision after the first iteration.

B. Reinforcement Learning

There are a large amount of works using reinforcement learning to address navigation problem. In [7], they train a DQN(deep q learning) agent [15] to cross an intersection in simulation. Zhu et al. [29] proposed a simulated environment for a deep reinforcement learning agent to navigate in rooms and find target objects. Similarly, in [14] the authors use DDPG method to avoid obstacles in a continuous action space. While these methods do not need an expert policy to imitate from, their agents are only trained in simulated settings. These methods require agents to learn from trail and error which is not applicable to real-world robotic tasks. Transferring policies learned in simulation to real world is critical. There are some works that address transferring learned policy from simulation to real world [26] [22]. In [26], the authors use randomized environments to train their policy in a grasping task. They address that their agent see the real world as another randomized environment. In addition, Rusu et al. proposed a network architecture that reduces catastrophic forgetting in transfer learning. However, this requires an extra training step in the real world which may also result in encountering dangerous states [17]. An overview of reinforcement learning can be found in [13].

III. Indoor Multi-Sensory Self-Supervised Autonomous Mobile Robotic Navigation

MS2L framework allows the mobile robot to label and learn from data collected by itself after the initial iteration. This section describes the problem it is solving, robotic platform, and *MS2L* framework in detail.

A. Problem formulation

In this section, we introduce the task, basic concepts, and notations.

Our goal is to train a mobile robot navigating through dynamically changing indoor environments quickly and safely without collisions.

1) Learn to navigate using DAgger: The action space in our work consists of two continuous variables representing linear and angular velocity respectively. The navigation task is defined as follows: given a current observation x, the robot outputs an action a that indicates linear and angular velocity based on a learned navigation policy

$$a = \pi_{\theta_p}(x), \tag{1}$$

where θ_p is the parameters of navigation policy. π_{θ_p} is learned by imitating some reference policy π_{ref}.

We define training samples encountered by executing π_{ref} as $D_{ref} = \{(x_1, l_1), ..., (x_t, l_t), ..., (x_T, l_T)\}$, where x_t and l_t represents the observation and corresponding action taken by π_{ref} at timestep t. The imitation objective is defined as

$$C_{imitation}(D_{ref}, \pi_{\theta_p}) = 1/N \times \sum_{i=1}^{i=N} ||\pi_{\theta_p}(x_i) - l_i||_2^2, \quad (2)$$

where N is the mini-batch size.Robot minimizes the squared $l2$ norm of the difference between predicted action values and the ones taken by the reference policy. DAgger algorithm trains the policy that minimizes imitation objective through multiple iterations. At the initial iteration, $\pi_{\theta_p^0}$ is trained on the data executed by π_{ref}. Then, at iteration n, observations are collected by the current navigation policy $\pi_{\theta_p^{n-1}}$. Reference policy π_{ref} is used to generate actions given these collected observations $l = \pi_{ref}(x)$. Generated data are aggregated to the previous dataset. Finally, $\pi_{\theta_p^n}$ is trained on the aggregated dataset. Detailed explanations of data collection and training procedure are described in Section III-C3

2) Self-supervised learning: To alleviate the need for human supervision required by DAgger algorithm, we want the robot to learn from training samples not labeled by human supervisors, but from its own. To this end, we introduce an imperfect policy derived from its range sensors. There are two reference policies in our work: π_{human} and π_{sensor} which denote human policy and sensor policy respectively. Let $D^i_{\pi_{sensor}}$ represents the dataset collected by π_{sensor} at the i^{th} iteration. Similar to DAgger, the robot learns from π_{human} to initialize navigation policy. After this pre-training stage, the robot learns from both $D^i_{\pi_{sensor}}$ and $D_{\pi_{ref}}$. Instead of requiring humans to label observations encountered by executing the current navigation policy, an imperfect sensor policy is used. Because the limited performance of sensor policy, we need to constrain the number of data recorded. This constraint should select the most effective data to be learned. A recording policy π_{θ_r} decides which observations are most needed to be learned by the current navigation policy, where θ_r represents the parameters of recording policy.

B. Robotic platform

Before describing *MS2L* framework in detail, we introduce our robot configurations and hardware specifications.

We use an iRobot Create2[2] as the base of our robotic platform. The linear velocity is in the range of $-0.5m/s$ to $0.5m/s$ and the angular velocity is in the range of $-4.5rad/s$ to $4.5rad/s$. Two HCSR04 ultrasonic sensor are mounted on two sides of the platform. They can detect objects within a distance of 5 to $400cm$ (or 2 to $156in$). In addition, a ZED™ Stereo Camera[3], which captures RGB images and estimates a depth image using stereo images, is equipped in the front of the robotic platform. The valid depth estimation is between $0.5m$ and $20m$. These two types of depth sensors complement with each other to derive the sensor policy. We use a NVIDIA® Jetson TX1[4] as the main computation resource to make inference. It has 256 CUDA cores with 2GB memory which allows it to make fast inference. Fig. 1 shows a front view of our robot.

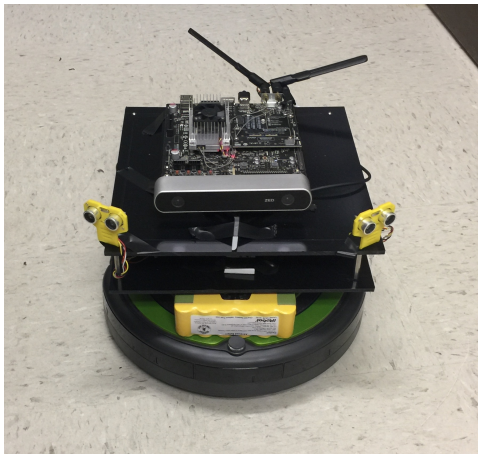


Fig. 1: ZED Stereo camera, two ultrasonic sensors and a NVIDIA® Jetson TX1 are mounted on iRobot Create2 with a Jetson TX1 computation board.

C. Methodology

In this section, we describe how our learning framework combines sensor policy, recording policy, and DAgger algorithm to enable the robot to learn in a self-supervised manner. Unlike DAgger algorithm used in [10] or [21], our learning framework does not require human supervision after the initial iteration. Since it learns from its own experiences, it alleviates the burden of human supervisor.

1) Sensor policy: While in simulated or controlled environments where the dynamic of environment is always known, it is not possible to obtain an accurate model of an unstructured real-world indoor environment. For example, in a simulated environment, requiring the distance between robot and wall is easy, but inaccurate and noisy measurements are often obtained from off-the-shelf range sensors like stereo cameras or ultrasonic sensors. In our work, we build our self-supervised learning system upon these unreliable sensor measurements. Our goal is not using an expensive and accurate range sensor like a laser range finder used in [16] to create an accurate depth control policy. We show that by using our proposed framework, the robot is able to successfully navigate through hallways in different conditions by imitating from an imperfect policy. Our sensor policy is derived from inaccurate depth estimation from off-the-shelf range sensors.

The key part of depth estimation from stereo cameras is finding correspondences of each pixel between left and right images. In practice, we find this estimation often fails when the correspondences can not be found while facing to plain

[2]http://www.irobot.com/About-iRobot/STEM/Create-2.aspx

[3]https://www.stereolabs.com/

[4]http://www.nvidia.com/object/embedded-systems-dev-kits-modules.html

Fig. 2: Two examples that stereo camera fails to estimate depth image. The left image shows a plain wall. The trash can in the right image only appears in the left camera.

surfaces or objects only appear in one camera. Two examples of failure cases are shown in Fig. 2. Based on this observation, we mounted two ultrasonic sensors on two sides of the robot to complement with failure cases.

The sensor policy is fully based on depth image and distance measurements from the two ultrasonic sensors. It is a deterministic algorithm that has zero variance but high bias. Algorithm 1 illustrates the policy in detail. The basic idea of it is using the valid points in a depth image received by the stereo camera and calculating the action correspondingly. If the number of valid points is smaller than the stereo threshold R, it uses ultrasonic sensor measurements to determine the action. It first splits a depth image M into three parts equally. It then counts the number of valid points inside each part using $count$ method. In this method, we filter out large values (> 20) because of sensor noise. In addition, values that are larger than some distance threshold D is considered a valid point. This calculates object occupancy in each part. The stereo camera confidence ratio r is calculated using the ratio between total number of valid points in each part and total number of pixels in the entire depth image M. Intuitively, r represents how much confidence the robot should trust the stereo camera. Rest of the algorithm is straight forward. If r is above R, the policy uses object occupancy to determine actions. Otherwise, ultrasonic sensor measurements are used.

This policy has a basic idea of when to turn, go backward, or go forward and fails in many cases. Because of high bias in this algorithm, it has a low performance. We show in our experiments, with an initial training combined recording policy, the robot is able to learn useful knowledge from this policy.

2) Recording policy: Recording policy π_{θ_r} is crucial to our learning framework. It takes the current observation x_t as input and outputs the probability indicating how much confidence of the current observation is needed for the navigation policy π_{θ_p} to learn. Unlike [28] where an expert policy exists and accessible, we use an imperfect sensor policy π_{sensor} to self labeling data after the pre-training stage. We use π_{θ_r} to constrain how much information the robot learns from π_{sensor}.

We use an additional neural network to model π_{θ_r}. Recording policy learns the deviation of labels output by navigation policy from the ones executed by reference policy. We define

Algorithm 1 Sensor policy

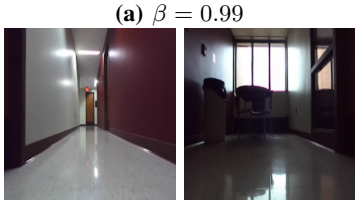
Input: Depth image M, right and left ultrasonic sensor measurements d_1, d_2, distance threshold D, stereo threshold R, and occupancy threshold T.
Output: Action that the robot needs to take to avoid collisions.

procedure SENSORPOLICY
$M_l, M_m, M_r = split(M)$
$O_l, O_m, O_r = count(M, D)$
$r = \frac{len(M_l) + len(M_m) + len(M_r)}{H \times W}$
if $r > R$ **then**
  if $O_m >= T$ **then**
    if $O_l < O_r$ **then**
      Turn right
    else
      Turn left
  else if $O_l \geq T$ and $O_m \geq T$ and $O_r \geq T$ **then**
    Decelerate and go backward
  else
    Go forward
else
  if $d_1 < D$ and $d_2 < D$ **then**
    Decelerate and go backward
  else if $d_1 < D$ **then**
    Turn right
  else if $d_2 < D$ **then**
    Turn left
  else
    Go forward

the error produces by the navigation policy π_{θ_p} with respect to reference policies (π_{human} and π_{sensor}) as

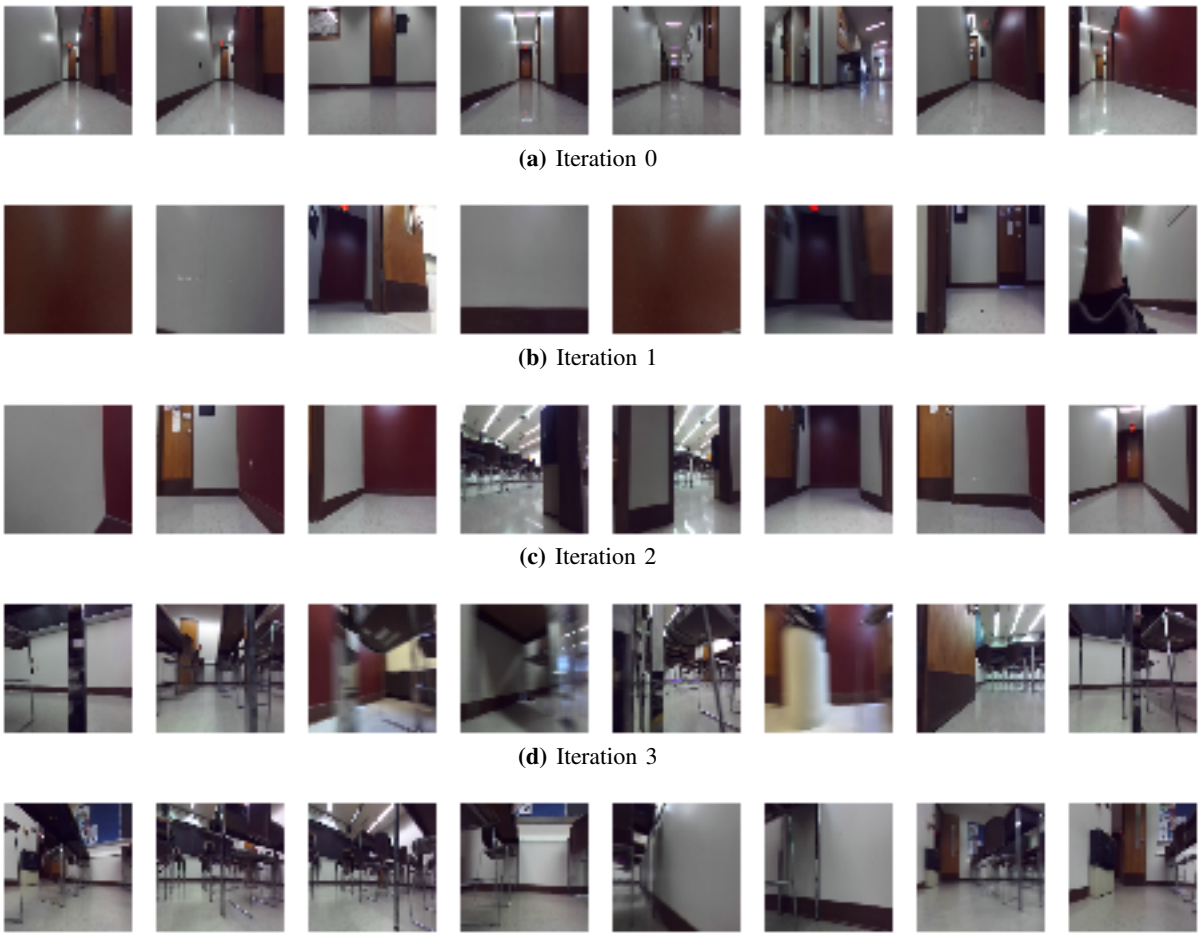
$$e(x_t, \pi_{\theta_p}, \pi_{human}, \pi_{sensor}) = \gamma||\pi_{\theta_p}(x_t) - \pi_{human}(x_t)||_2^2 + (1-\gamma)||\pi_{\theta_p}(x_t) - \pi_{sensor}(x_t)||_2^2, \quad (3)$$

(a) $\beta = 0.99$



(b) $\beta = 0.5$

Fig. 3: Images are uniformly sampled from two datasets that have different thresholds.



(a) Iteration 0

(b) Iteration 1

(c) Iteration 2

(d) Iteration 3

(e) Iteration 4

Fig. 4: An illustration of data collected in five iterations.

where γ is a constant between 0 and 1 that weights the relative importance of two reference policies. If γ is close to 1,the sensor policy is not accounted for the deviation. This error metric uses squared $L2$ distance of the difference between predicted values and reference values. In our experiment, we set it to a large value 0.8. With this error metric, a binary valued function indicating whether to record is defined as

$$\epsilon(x, \pi_{\theta_p}) = \begin{cases} 1, & \text{if } e(x_t, \pi_{\theta_p}, \pi_{human}, \pi_{sensor}) > \tau \\ 0, & \text{otherwise} \end{cases}, \quad (4)$$

where τ is a scalar indicating the degree of error tolerates by the recording policy.

π_{θ_r} is trained on a recording dataset $D_{\pi_{\theta_r}} = \{x_1, ..., x_N\}$. The objective function to be minimized by π_{θ_r} is a binary cross-entropy loss function defined as

$$C_{record}(D_{record}, \epsilon, \pi_{\theta_r}\pi_{\theta_p}, \pi_{ref}) = -\frac{1}{N} \times \sum_{n=1}^{N} \epsilon(x, \pi_{\theta_p}) log(\pi_{\theta_r}(x_n)) + (1 - \epsilon(x, \pi_{\theta_p})) log(1 - \pi_{\theta_r}(x_n)),$$

where π_{ref} is consisted of π_{human} and π_{sensor}. With π_{θ_r} and good choices of γ, β, and τ, the robot can balance the knowledge learning from π_{human} or π_{sensor}. This allows the robot to learn from imperfect real-world sensor policy in an unsupervised manner.

In inference mode, observations are recorded only if $\pi_{\theta_r}(x) > \beta$, where β is a threshold controls how much the robot trusts π_{θ_r}. We call observations recorded by π_{θ_r} as valid. Some recorded observations with different thresholds are shown in Fig. 3. In Fig. 3a, most images are recorded in dangerous states such as turning or too close to walls. In contrast, Fig. 3b shows most of the images are going forward.

3) Data collection and training: Our data collection and training process consist of two stages with 5 iterations in total. Fig. 4 shows some samples collected from 5 iterations. The data collection and training algorithm is shown in Algorithm 2. The first iteration is a pre-training stage, where a human operator controls the robot using a PlayStation controller to navigate through the hallway to collect navigation and recording policy dataset $D_{\pi_{\theta_r}}$ and $D^0_{\pi_{\theta_p}}$. These two datasets are used for initializing navigation policy and recording policy. The

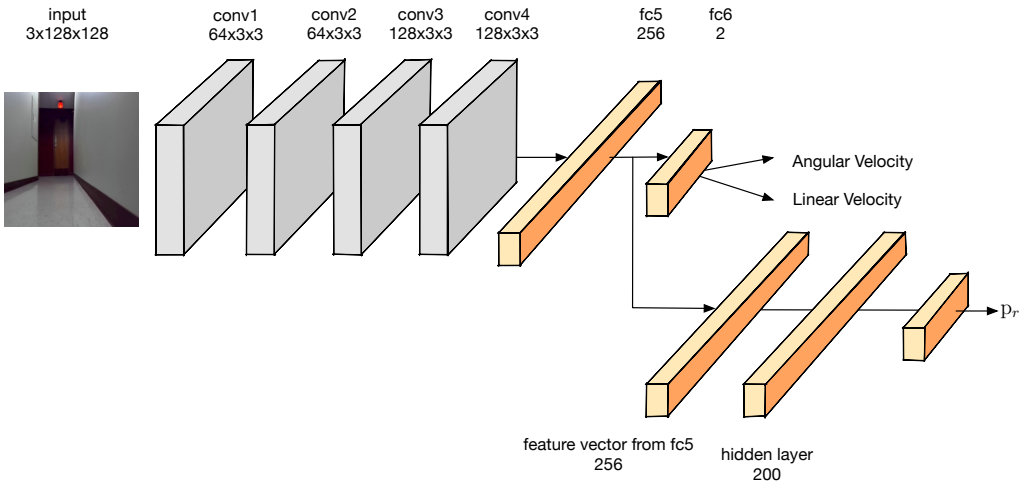


Fig. 5: Our model consists of two networks: policy network π_{θ_p} and recording network π_{θ_r}

Algorithm 2 Multi-Sensory Self-Supervised Learning

procedure DATA COLLECTION AND TRAINING

Random initialize π_{θ_r} and π_{θ_p}.

Pre-training

Collect $D_{\pi_{\theta_r}}$ and $D^0_{\pi_{\theta_p}}$ using π_{human}

$\pi_{\theta^0_p} = arg\,min_{\theta_p} C_{imitaion}(D^0_{\pi_{\theta_p}}, \pi_{\theta_p})$

$\pi_{\theta^0_r} = arg\,min_{\theta_r} C_{record}(D_{\pi_{\theta_r}} \cup D^0_{\pi_{\theta_p}}, \epsilon, \pi_{\theta_r}, \pi_{\theta^0_p}, \pi_{ref})$

Self-supervised learning

for $i = 1$ **to** 4 **do**

Collect and label observations using $\pi_{\theta^i_p}$ and π_{sensor} into dataset D.

Only keep (x, l) pair from D where $\pi_{\theta^i_r}(x) > \beta$

$D^i_{\pi_{\theta_p}} = D \cup D^{i-1}_{\pi_{\theta_p}}$

$\pi_{\theta^i_p} = arg\,min_{\theta_p} C_{imitaion}(D^{i-1}_{\pi_{\theta_p}}, \pi_{\theta^{i-1}_p})$

$\pi_{\theta^i_r} = arg\,min_{\theta_r} C_{record}(D_{\pi_{\theta_r}} \cup D^i_{\pi_{\theta_p}}, \epsilon, \pi_{\theta^{i-1}_r}, \pi_{\theta^{i-1}_p}, \pi_{ref})$

return $\pi_{\theta^4_p}$ and $\pi_{\theta^4_r}$

second stage is a self-supervised learning stage including four iterations. At this stage, Instead of querying π_{human}, π_{sensor} is used to generate labels for collected data. In addition, π_{θ_r} is used to constrain the data collection. Specifically, an observation x and its corresponding label $\pi_{sensor}(x)$ is only recorded when $\pi_{\theta_r}(x) > \beta$.

In pre-training stage, the robot collects navigation policy dataset $D^0_{\pi_{\theta_p}}$ and recording policy dataset $D_{\pi_{\theta_r}}$ in clockwise and counter-clockwise directions in RB 4th floor hallway. It ensures the recording policy correctly classifies valid observations by traveling in different directions. Although pre-training stage provides a relative good policy e.g. following a straight line, it still collides with walls that may cause hardware damage. To address this problem, we multiply linear velocity by the output of recording policy $v' = (1 - \pi_{\theta_r}(x)) \times \pi_{\theta_p}(x)_0$, where the subscript 0 is the index for linear velocity in action space. If a recording is needed, i.e. $\pi_{\theta_r}(x)$ is close to 1, also means the observation may be dangerous to the robot, so the robot should slow down to avoid collision with a fast speed.

As we show in our experiments, with this training and data collection methodology, we obtain a policy that surpasses human performance policy in one task and near-human performance in two tasks.

4) Network architecture: Fig. 5 shows an overview of our network model. Given an observation x, the navigation policy network predicts an action $a = \pi_{\theta_p}(x)$, where θ_p is the parameters of a five layer convolutional network. The recording policy is parameterized by two-layer fully connected neural network. It takes the feature vector f from the last layer (before final output layer) of navigation policy network fc_5 and predicts the recording probability $p_r = \pi_{\theta_r}(f)$, where θ_r is the parameters of the recording network. The design choice of the recording policy is two folds. First, because of the GPU memory constraint on Jetson TX1, we are not able to build two CNNs, where one for navigation policy and another for recording policy. Second, the feature extracted by CNN at layer fc_5 contains discriminant representations of the environment as discussed in [12]. These representations boost learning of the recording policy.

IV. PERFORMANCE EVALUATION

In this section, we describe our experiments and evaluations. Our experiments are performed to show:

- The performance of navigation policy improves as the number of iterations increases.
- Recording policy is crucial to our framework. It gives more accurate predictions on which observations to be recorded as the number of iterations increases.

- The navigation policy trained on all iterations surpasses π_{sensor} on all the tasks and achieves near human performance.

A. Experiment Settings

The experiments have been performed in the Robert Bell Hall building at Ball State University. Figure 6 and 7 are indoor environments where our robot has been trained.

Fig. 6: Robert Bell 4^{th} floor hallway

Fig. 7: Robert Bell 4^{th} floor classroom

In all experiments, we use tensorflow [1] to implement our neural networks. Adam optimizer [9] with learning rate 0.0001 and 0.001 is used for π_{θ_p} and π_{θ_r} respectively and a $L2$ weight decay of 0.0001 to all networks. We set $\gamma = 0.8$ and $\tau = 0.00025$ while training π_{θ_r}. We set $\beta = 0.99$ while collecting data using recording policy. Observations are RGB images received from stereo camera and resized to dimension of 128×128. For each iteration, we collect data for $250s$ at $30fps$. We use traveled distance and duration before colliding as evaluation metrics for test runs. We set the maximum travel duration to $250s$. In addition, the metric terminates test runs when they repeat the same trajectory.

We trained our robot in 2 indoor environments: Robert Bell 4^{th} floor hallway and a classroom during break time. The classroom is an extremely difficult environment, where human operator also collides into obstacles quickly in some test runs.

B. Comparing Navigation Policies

In this experiment, we show navigation policy improves over iterations. Fig. 8 and Fig. 9 presents two performance evaluations in terms of distance traveled and time duration. We notice that there is no gain between the pre-training stage and the first iteration in self-supervised learning stage. However, a performance gain is linearly increasing after the first iteration of the self-supervised learning stage. We hypothesis this is because the recording policy only records a small number of observations, which gives a relative small dataset to be aggregated (also shown in Table I).

The failures of the first two iterations happen at the first turning corner. The second and third iterations fail when the lightening is too bright or too dark. In the last iteration, the robot navigates through the hallway successfully and reach time limit. The result indicates $MS2L$ framework learns to navigate without human supervision.

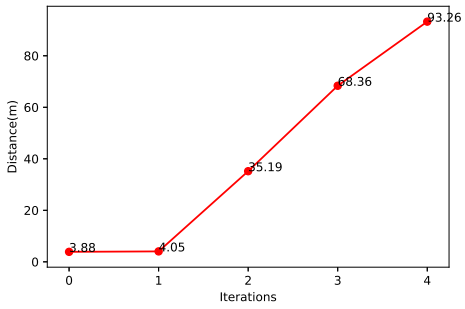


Fig. 8: Distance traveled before colliding in each iteration.

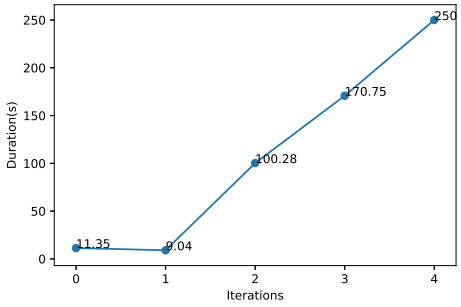


Fig. 9: Duration traveled before colliding in each iteration.

C. Datasets and Recording Policy

The performance of recording policy is evaluated by number of observations recorded as well as distributions over recorded actions. An optimal recording policy records small amount of observations with actions that have equal distributions. For example, a bad recording policy collects more forwarding actions, e.g. $a = (0.5, 0.0)$. This may cause the robot to only learn going forward.

Table I illustrates the amount of data collected during each iteration. Fig. 10 shows the distributions over linear and angular velocities in 5 iterations. We collect training data for $250s$ at $30fps$, so there are 7500 observations in each iteration. The ratio column is the ratio of collected observations and total observations. In the pre-training stage (first iteration), human operator collects all observations in 250 seconds. The number of observations collected in the rest four iterations dramatically reduce because of the recording policy. We notice that at

TABLE I: Results on training data collected during 5 iterations

Iteration	# of observations	Ratio
0	7500	1
1	351	0.005
2	853	0.114
3	790	0.105
4	335	0.045

TABLE II: Performance of navigation policies on different β

β value	Distance(m)	Time(s)
0.99	35.19	100.28
0.5	27.53	97.49
0.1	15.33	43.92

iteration 1, less observations are recorded than iteration 2 and 3. However, by looking at Fig. 10, most of the probability masses of linear and angular velocities at iteration 1 are on 0.5 and 0, which is going forward. In contrast, the probability masses of angular velocity in iteration 2 and 3 are mainly on left ($4.5rad/s$) and right ($-4.5rad/s$) turn. This indicates that the recording policy learns to distinguish between valid and invalid observations as the number of iteration increases (invalid observations are usually the ones with straight road). Note that linear velocity with less than $0.5m/s$ is caused by decelerating in Algorithm 1. Our goal is to train the robot navigating as quickly and safely as possible. Therefore, most forwarding velocity is at the maximum value ($0.5m/s$). Only a small part is in the dangerous observations where sensor policy needs to decelerate and go backward.

As shown in Fig. 10, most angular velocity is $0rad/s$ at iteration 0. This is because the hallway mainly consists of straight road with only several turning corners. Because of the distribution mismatch and training data unbalancing problems (only forwarding action in dataset), the robot is not able navigate without collision. The self-supervised learning stage breaks distribution mismatch and balances the dataset by recording valid observations.

While the previous results show that the recording policy's ability to identify valid observations, the importance of the recording policy still needs to be verified. As discussed in Section III-C2, β controls the strength of how many observations to be recorded. For example, if $\beta = 0$, all observations will be recorded. Table II shows the performance of navigation policy given different β values. Because of time limits, navigation policies are evaluated after the third iteration. As β decreases, the robot records more observations from sensor policy which causes the navigation policy learns more from wrong actions. The performance decreases as β decreases.

D. Comparing with Baselines

We compare our method with two baselines over three tasks. The first baseline is sensor policy and the second one is a human operator. The observation for human operator is a third person view which is the same as navigation policy. We show

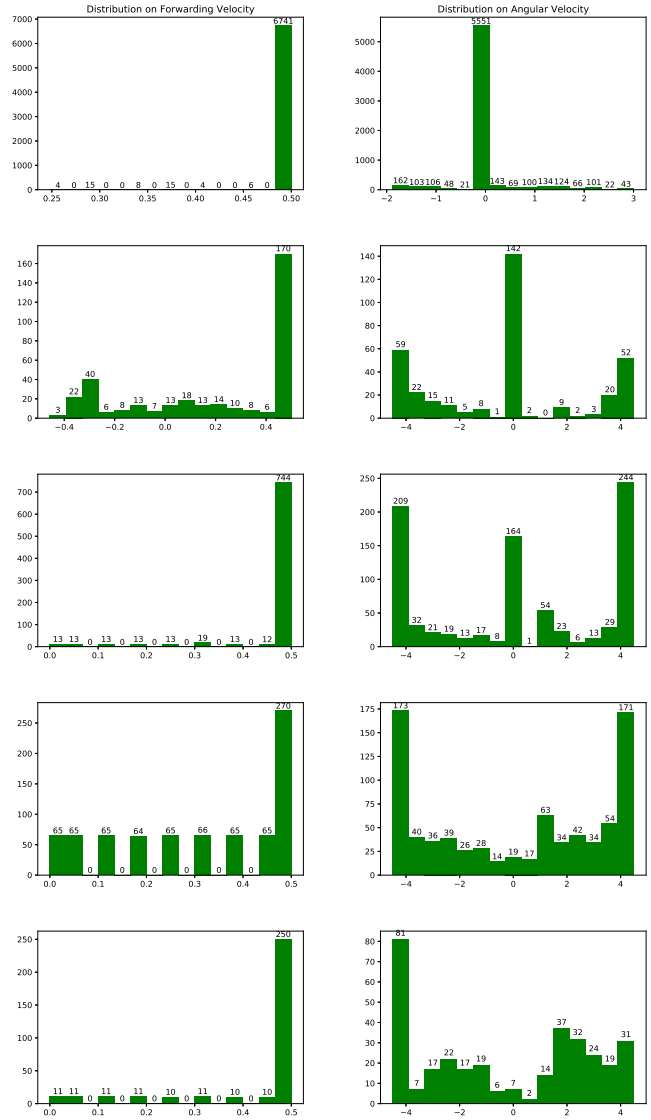


Fig. 10: Distributions of forwarding and angular velocities over 5 iterations. Each row represents one iteration.

that our framework is able to surpass sensor policy by a large margin in two out of three tasks. It is also able to achieve near human performance in two tasks and surpasses human performance in one task. The three tasks are described as follows:

- The first task is navigating through Robert Bell 4^{th} floor hallway during business time. The environment

TABLE III: Comparisons with baselines in three tasks

	Hallway		Classroom		Hallway with noise	
	Dist(m)	Time(s)	Dist(m)	Time(s)	Dist(m)	Time(s)
π_{sensor}	23.44	58.61	1.46	3.18	9.77	20.35
π_{human}	**114.58**	250	**3.43**	14.63	12.09	54.64
π_{θ_p}	89.96	224.97	3.04	12.26	**61.23**	196.37

Fig. 11: Visualizations of navigation policy. Blue masks indicate the visual cues navigation policy focus on.

is different from the the training data which is during break time.

- The second task is navigating through a classroom where chairs and tables are main obstacles. This environment is difficult even for human operator.
- The third task is navigating through Robert Bell 4^{th} floor with Gaussian noise added to the controller which simulates hardware malfunctioning. We would like to use this test to show the robustness of our method to hardware noise.

Table III shows the results of comparisons. The performance of the proposed framework exceeds sensor policy in every environment we tested on. This is noticeable because only one iteration of human examples are learned by the policy. This means that our framework is able to filter out defect training examples from sensor policy. Human policy baseline has a higher performance in two out of three tasks. However, human operator is not able to deal with sudden noise added to the controller. This allows our method surpasses human operator.

One thing we noticed is that the navigation policy has a lower performance in this experiment than in the previous ones. This is because the robot is testing during the business hour when the traffic is much heavier than during break time while previous ones are evaluated in the training environments.

E. Visualizing Navigation Policy

We use the method introduced in [2] to visualize our navigation policy. This is a fast and accurate method to visualize which sets of pixels make the most contributions for making CNN decisions. It is useful for us to interpret what our navigation policy has learned. Fig. 11 illustrates observations with overlaid blue masks indicating pixels that activate the navigation policy.

We notice that lights are the most important feature while going forward. Wall edges and light reflections are useful for navigation policy to determine when and where to take a turn action. This is reasonable since lights are aligned in a row and installed in the center of hallway and wall edges usually mean close to corners or walls. This demonstrates that navigation policy learns to navigate by looking at these visual cues to output correct actions.

V. Conclusions

In this work, we propose a $MS2L$ framework to alleviate human supervision by combining imperfect sensor policy and recording policy. To best of our acknowledge, This is the first time using imperfect sensor measurements to do self-supervised learning in the task of navigating in indoor environments. We perform extensive experiments qualitatively and quantitatively to show our framework learns to navigate without human intervene. It achieves near human performance and surpasses the sensor policy by a large margin. Since this framework do not require a perfect policy to learn from, it is applicable to many other robotic tasks that require expert supervision.

Acknowledgment

The authors would thank all anonymous reviewers for their precious time. This material is based upon work supported by the National Science Foundation under Grant No. #1408165 and #1726017. The TX1 and TX2 used for this research was donated by the NVIDIA Corporation.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] M. Bojarski, A. Choromanska, K. Choromanski, B. Firner, L. Jackel, U. Muller, and K. Zieba. VisualBackProp: visualizing CNNs for autonomous driving. 2016.

[3] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[4] D. Gandhi, L. Pinto, and A. Gupta. Learning to Fly by Crashing. 2017.

[5] A. Giusti, J. Guzzi, D. C. Cireşan, F.-L. He, J. P. Rodríguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. Di Caro, et al. A machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robotics and Automation Letters*, 1(2):661–667, 2016.

[6] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik. Cognitive Mapping and Planning for Visual Navigation. 2017.

[7] D. Isele, A. Cosgun, K. Subramanian, and K. Fujimura. Navigating Intersections with Autonomous Vehicles using Deep Reinforcement Learning. *arXiv preprint arXiv:1705.01196*, 2017.

[8] D. K. Kim and T. Chen. Deep Neural Network for Real-Time Autonomous Indoor Navigation. *CoRR*, abs/1511.04668, 2015.

[9] D. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *Computer Science*, 2014.

[10] M. Laskey, J. Lee, C. Chuck, D. Gealy, W. Hsieh, F. T. Pokorny, A. D. Dragan, and K. Goldberg. Robot grasping in clutter: Using a hierarchy of supervisors for learning from demonstrations. In *IEEE International Conference on Automation Science and Engineering*, pages 827–834, 2016.

[11] M. Laskey, S. Staszak, W. Y.-S. Hsieh, J. Mahler, F. T. Pokorny, A. D. Dragan, and K. Goldberg. Shiv: Reducing supervisor burden in dagger using support vectors for efficient learning from demonstrations in high dimensional state spaces. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 462–469. IEEE, 2016.

[12] Y. Lecun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[13] Y. Li. Deep Reinforcement Learning: An Overview. 2017.

[14] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *Computer Science*, 8(6):A187, 2015.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[16] 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. *arXiv preprint arXiv:1609.07910*, 2016.

[17] A. A. R. N. C. Rabinowitz". Progressive Neural Networks. 2016.

[18] R. Rahmatizadeh, P. Abolghasemi, A. Behal, and L. Bölöni. Learning real manipulation tasks from virtual demonstrations using LSTM. *arXiv preprint arXiv:1603.03833*, 2016.

[19] E. Rimon and D. E. Koditschek. Exact robot navigation using artificial potential functions. *IEEE Transactions on robotics and automation*, 8(5):501–518, 1992.

[20] S. Ross, G. J. Gordon, and D. Bagnell. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. In *AISTATS*, volume 1, page 6, 2011.

[21] S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. Bagnell, and M. Hebert. Learning monocular reactive uav control in cluttered natural environments. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 1765–1772. IEEE, 2013.

[22] F. Sadeghi and S. Levine. $(CAD)^2$ RL: Real Single-Image Flight without a Single Real Image. *arXiv preprint arXiv:1611.04201*, 2016.

[23] L. Tai, G. Paolo, and M. Liu. Virtual-to-real Deep Reinforcement Learning: Continuous Control of Mobile Robots for Mapless Navigation. *arXiv preprint arXiv:1703.00420*, 2017.

[24] R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts. LQR-trees: Feedback motion planning via sums-of-squares verification. *The International Journal of Robotics Research*, 29(8):1038–1052, 2010.

[25] S. Thrun and J. J. Leonard. Simultaneous localization and mapping. In *Springer handbook of robotics*, pages 871–889. Springer, 2008.

[26] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World. 2017.

[27] J. Xu, S. Zhu, H. Guo, and S. Wu. Avoidance of manual labeling in robotic autonomous navigation through multi-sensory semi-supervised learning. *arXiv preprint arXiv:1709.07911*, 2017.

[28] J. Zhang and K. Cho. Query-efficient imitation learning for end-to-end autonomous driving. *arXiv preprint arXiv:1605.06450*, 2016.

[29] 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. 2016.