Self-Correcting Online Navigation via Leveraged Gaussian Processes

Seunggyu Chang, Sungjoon Choi, Songhwai Oh
Department of Electrical and Computer Engineering and ASRI, Seoul National University, Seoul, 08826, Korea
(E-mail: {seunggyu.chang, sungjoon.choi, songhwai.oh}@cpslab.snu.ac.kr)

***Abstract*—In this paper, a novel online learning navigation algorithm is proposed to incorporate negative data generated from failure in an online manner. While existing methods require additional knowledge about *what to do* at failed situations, the proposed method alleviates this by utilizing failures as a clue of *what not to do* without requiring additional knowledge of *what to do*. By combining the benefits of leveraged Gaussian processes and sparse online Gaussian processes, we proposed an online learning framework for navigation and its update rule which instantly learns which actions to avoid from the failures while navigating. Our navigation method is successfully validated on a static planar world and dynamic worlds on both simulation and real-world dataset.**

***Keywords*—Online learning navigation, Learning from failure.**

1. INTRODUCTION

Recently, robots are becoming prevalent to our daily lives. It has become easier to find service robots in public places such as shopping malls, convention halls and airports. For a robot to provide a good service, it should navigate safely and harmoniously with the people in a crowded environment. However, the environment the robot encounters change over time, e.g., seasonal changes of structure arrangement in a mall. To perform good navigation, the robot should continuously modify its behavior adaptively to the current environment to prevent crashing with people or obstacles. To achieve this, *online learning* plays an important role.

Learning from demonstration (LfD) has been widely used for autonomous robot navigation [1]–[3]. LfD is an algorithm to teach a robot to perform a specific task by showing well-performed demonstrations instead of designing a hand-crafted controller for the robot to perform certain tasks. LfD models a policy function, π, which directly maps environments ($\mathcal{S}$: *states*) to robot behaviors ($\mathcal{A}$: *actions*). Hence, LfD is effective when defining the concept of best action is ambiguous or when a control function is highly complex function such as an autonomous navigation in a crowd[1]

However, LfD has risk of failing at accomplishing tasks due to the limitation of demonstration — scarcity of successful demonstrations in certain states or suboptimal property of demonstrations which are generated by non expert. There have been many attempts to handle this risk [4]–[9], however, most of them require extra knowledge, e.g., additional information in terms of reward function which is not usually given in LfD [9], or an external supervisor to inform optimal actions for queries [5].

In this paper, a novel online learning navigation algorithm is proposed by updating a policy function by utilizing additional information of *what not to do* from failures. The proposed method is inspired by human learning behavior. For example, once a human learner fails at performing a task, he/she memorizes the fault and takes precautions not to do similar actions again. Likewise, the proposed algorithm includes *memorizing* and *correcting* procedures for the failed experiences.

The proposed method utilizes leveraged Gaussian processes (leveraged GPs) [10] for incorporating both successful and failed data. However, since the computational complexity of leveraged GPs is $O(n^3)$ where n is the number of samples in the training set, it is not suitable for online learning. Hence, sparse online Gaussian processes (SOGPs) [11] parameterization technique is used for fast online update. By combining the benefits of both leveraged GPs and SOGPs, we proposed an online navigation framework and its fast update rule. To this end, two different techniques are proposed to learn from failures: *memorizing* and *self-correcting*. Intuitively, *memorizing* refers a process of adding additional failed data as a new training data while *self-correcting* refers a process of modifying existing false training data to negatives. The proposed methods are intensively validated by experiments including navigation in a static planar world and in a dynamic environments on both simulation and real-world dataset.

The remainder of this paper is structured as follows. The next section describes brief summary of literature about autonomous navigation and LfD algorithms and Section III describes preliminary knowledge for the proposed methods. Section IV describes the proposed online navigation framework and its novel online update rule. A quantitative results about simulation study and real-world experiments are given in Section V and VI.

2. RELATED WORK

There have been many efforts for a robot to navigate safely and efficiently in a dynamic environment. The classical way is to model environments using hand crafted features. i.e., modeling obstacles as sources of potential field [12] or sources of virtual force [13]. Recently, data-driven navigation models have begun to be proposed and LfD has been widely used [1]–[3] for intuitiveness and simplicity.

However, LfD has some drawbacks [14] as an agent may encounter unfamiliar states due to environmental changes. This risk often results in failures and accumulated failures can cause catastrophic situation which is harmful to the agent or other objects [5] and a number of approaches [5], [10], [15], [16] are suggested to handle failures of LfD. We grouped LfD algorithms into four categories depending on the learning method and the ability to incorporate negative data. We grouped LfD algorithms into batch learning and online learning in terms of capability of incorporating additional data sequentially. Furthermore, we grouped LfD algorithms into two in terms of ability of incorporating failure data as well

[1]What is optimal avoidance is ambiguous because there is no explicit rule for how, where, and when to avoid.

TABLE 1: Categorization of LfD algorithms

	Batch Learning	Online Learning
Successful	[1]–[3]	[5], [17]
Successful + Failure	[10], [15]	[16], This work

as successful data. Table 1 summarizes classification of LfD algorithms.

A. Successful Data + Batch Learning: A vast number of existing LfD methods can be included in this class. [1] presents autonomous navigation in a complex unstructured terrain with an expert's demonstration for constructing a cost map used for planning. [3] proposed an autoregressive Gaussian motion controller which is trained using expert's demonstration to navigate in a crowd.

B. Successful Data + Online Learning: Algorithms of this class are initially trained with successful demonstration. However, the agent may fall into a catastrophic situation and cannot recover from it [5], [17]. [5] proposed a meta learning algorithm called DAgger which queries *what actions to do* to an external expert for recovering from the catastrophic situation requiring constant interactions with an external supervisor.

C. Successful & Failure Data + Batch Learning: Algorithms in this class are capable of incorporating failed demonstrations adaptively without re-train the model from a scratch. The notable advantage of this group compared to group *B* is that it utilized failed demonstrations as well as successful demonstrations. [10], [15] proposed leveraged GPs to incorporate multiple correlated processes in a single framework. By modeling that negative data (failure data in terms of [10], [15]) are generated from negatively correlated process to the process generating positive (successful) data, leveraged GPs successfully exploit both positive and negative data for inference. However, this group have difficulties in incorporating additional data.

D. Successful & Failure Data + Online Learning: This class uses successful demonstrations initially but updates the model adaptively in an online way, especially when the agent encounters failure data in the middle of execution. We believe that the ability of incorporating failure data plays an important role for online navigation. [16] proposed a maxican hat style probability density function to model joint probability on state-action space as a density mixture model. According to the literature, [16] is the most similar work to ours. However, the probability density mixture model experiences severe performance degradation in high-dimensional space because it is unsupervised generative model. Hence, it may fail when state-action space lies in a high dimension [18]. The proposed method in this paper lies in this category.

3. PRELIMINARIES

3.1. Sparse Online Gaussian Processes

Gaussian Processes (GPs) are supervised nonparametric model which are widely used for modeling nonlinear function based on observations. In GPs, function values of every subset of data points exhibit joint Gaussian distribution. A Gaussian process is completely specified by a mean function $m(\mathbf{x})$ and a covariance function $k(\mathbf{x}, \mathbf{x}')$.

Function values of test data can be inferred from conditional distribution conditioned on the training data, requiring an inverse operation of covariance matrix of training data which costs $O(n^3)$ computational complexity. This high computational cost inhibits its usage for large-scale problems. Hence, a number of works [11], [19], called sparse GPs, have been proposed to tackle this problem assuming that a small number of data points called inducing points are enough to approximate the behavior of original GPs.

Online learning is a learning method that updates a model sequentially as training data is revealed sequentially. In case of naive GPs, the posterior update to incorporate new data with existing training data is computationally cumbersome and hinders GPs from online learning. For that reason, [20] measure the projection-induced error of a new data in the reproducing kernel Hilbert space (RKHS) to choose the representative subset resulting sparse GPs. Sparse online Gaussian processes (SOGP) [20] parameterize all the training data into a concise model parameters anchoring at representative subset of the training data called inducing points. The model is updated to be closest to true posterior distribution in the Kullback-Leibler divergence sense. Following [20], the online update of a posterior mean $\langle f_{\mathbf{x}} \rangle$ and posterior kernel function of a SOGP are

$$\begin{aligned} \langle f_{\mathbf{x}} \rangle_{t+1} &= \langle f_{\mathbf{x}} \rangle + q^{(t+1)} K_t(\mathbf{x}, \mathbf{x}_{t+1}) \\ K_{t+1}(\mathbf{x}, \mathbf{x}') &= K_t(\mathbf{x}, \mathbf{x}') + r^{(t+1)} K_t(\mathbf{x}, \mathbf{x}_{t+1}) K_t(\mathbf{x}, \mathbf{x}') \end{aligned} \tag{1}$$

where the scalars $q^{(t+1)}$ and $r^{(t+1)}$ are

$$\begin{aligned} q^{(t+1)} &= \frac{\partial}{\partial \langle f_{t+1} \rangle} \ln \langle P(y_{t+1} | f_{t+1}) \rangle_t \\ r^{(t+1)} &= \frac{\partial^2}{\partial \langle f_{t+1} \rangle_t^2} \ln \langle P(y_{t+1} | f_{t+1}) \rangle_t \end{aligned} \tag{2}$$

3.2. Leveraged Gaussian Prcoesses

Leveraged Gaussian Processes (leveraged GPs) [10] can model multiple correlated processes in a single Gaussian process framework by introducing leverage parameter, γ, to a general kernel function. By doing so, one can vary the level of influence of data which come from differently correlated processes. Prediction of leveraged GP tends to be close to the positively correlated samples and drift apart from the negatively correlated samples. This can help LfD model predict more reliable policy where the successful demonstration is sparse. Suboptimal or adversarial demonstrations can be effectively incorporated with the successful demonstrations in leveraged GPs. Leverage kernel function is defined as follows [10].

$$k_{LEV}(\mathbf{x}, \mathbf{x}') = \cos\left(\frac{\pi}{2}(\gamma_{\mathbf{x}} - \gamma_{\mathbf{x}'})\right) k_{PSD}(\mathbf{x}, \mathbf{x}') \tag{3}$$

Note that the first $\cos()$ term in Eq. (3), varying from $+1$ to -1, controls correlation between two processes. In particular, by setting leverage parameters γ of negative samples to induce $\cos()$ term in Eq. (3) to be negative, we can utilize information contained in negatively correlated processes rather than discarding it. In [10], robust navigation is successfully accomplished using leverage GP and both positive and negative data. However, they are batch mode algorithm which is not applicable to online learning.

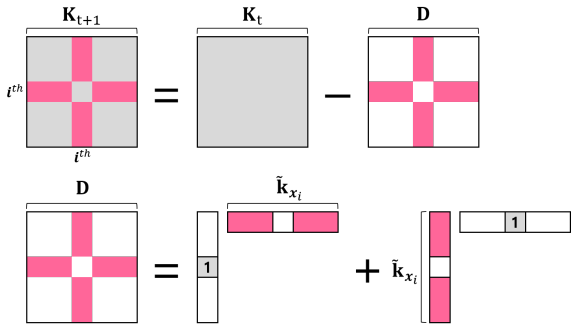


Fig. 1: Illustration of the changing pattern of the kernel matrix when leverage parameter of i-th sample changes.

4. SELF-CORRECTING NAVIGATION

In this section, we present a novel self-correcting online navigation (SCON) algorithm which can adaptively improves the quality of navigation from failed experiences in a online manner. In this work, a failure is defined to be occurred when the agent fall into an inevitable collision state (ICS) [21]. We treat a negative demonstration as the state-action pair previous to the failure and positive demonstration as successful demonstration. To incorporate both positive and negative demonstrations, we utilize leveraged Gaussian Processes (leveraged GPs) [10] which are able to effectively incorporate both positive and negative data. However, leveraged GPs are not suitable for online learning as it should be re-trained to incorporate additional data. To utilize leveraged GPs for online setting, the proposed method (SCON) exploits sparse online Gaussian processes (SOGP) [20]. By combining the benefits of both leveraged GPs and SOGP, SCON is proposed which can easily incorporate additional negative data.

4.1. Online Parameter Update Rule

Two methods of model parameter update rule specific to leveraged GPs for SOGP parameterization, *memorizing* and *self-correcting*, are proposed in this section. Consider a situation at time t when the negative data $(\mathbf{x}_t, \mathbf{u}_t)$ just being incorporated. Let $\mathcal{M}_t$ and $\mathcal{I}_t$ denote the SOGP model and corresponding inducing set at time t. One naïve approach is to directly apply basic SOGP update rule to the existing model $\mathcal{M}_t$ with negative data. However, this may destroy the model when the negative data conflict with existing trained data which have high correlations with the negative data. Let $\mathcal{S}$ denotes a subset of inducing set whose elements have high correlations in absolute values, which are higher than ϵ_{th}, with the negative data. If the negative data is unfamiliar to the current model $\mathcal{M}_t$, that is $\mathcal{S} = \varnothing$, it is directly added to the model $\mathcal{M}_t$ with negative label. This process is referred to as *memorizing* and can be conducted by SOGP learning rule with leverage parameter $\gamma_t = -1$. Otherwise, for stable learning of the SOGP, it is necessary to modify existing training data (inducing points) to be consistent with the negative data. To this end, we convert the leverage parameters of the highly correlated subset $\mathcal{S}$ to the opposite signs, from positives to negatives and vice versa. Using the property of a leverage kernel function, a fast update rule to convert leverage parameters of existing training data (inducing points) in the SOGP framework is proposed and this process is referred to as *self-correcting*.

Algorithm 1 Self_correcting

Input: $\mathbf{x}_t, \mathcal{S}$, $f^{(t)}$ composed of $\boldsymbol{\alpha}_t, \mathbf{C}_t, \mathbf{Q}_t$
Output: $f^{(t+1)}$ composed of $\boldsymbol{\alpha}_{t+1}, \mathbf{C}_{t+1}, \mathbf{Q}_{t+1}$

```
for i = 1 to |S| do
    u_i ← e_i
    x_i ← i-th element of S
    v_i ← k̃_{x_i}
    γ_{x_i} = −γ_{x_i}
end for
U ← [u_1, ... , u_{|S|}, v_1, ... , v_{|S|}]
V ← −2[v_1, ... , v_{|S|}, u_1, ... , u_{|S|}]
update α using (7)
update C using (8)
update Q using (9)
return f^(t)
```

Consider i-th inducing point is selected and is to change from positive to negative. Then the i-th row and i-th column of the kernel matrix $\mathbf{K}_t$ except the i-th diagonal entity $(\mathbf{K}_t)_{ii}$ change. As in the Fig. 1, the changing components form cross shape and can be decomposed into rank-2 downdate. Let $\tilde{\mathbf{k}}_{\mathbf{x}_i}^{(p)}$ denote kernel vector of $\mathbf{x}_i$ with leverage parameter $\gamma = +1$ w.r.t. inducing point set $\mathcal{I}_t$ with its i-th elements is replaced to 0. Similarly, let $\tilde{\mathbf{k}}_{\mathbf{x}_i}^{(n)}$ denotes a vector constructed as the same way as $\tilde{\mathbf{k}}_{\mathbf{x}_i}^{(p)}$ with $\gamma = -1$. Let $\mathbf{e}_i$ denotes standard unit vector and $\mathbf{D}$ denotes cross shape difference matrix. Then $\mathbf{D}$ can be decomposed into

$$\mathbf{D} = \mathbf{e}_i \left(\tilde{\mathbf{k}}_{\mathbf{x}_i}^{(p)} + \tilde{\mathbf{k}}_{\mathbf{x}_i}^{(n)}\right)^T + \left(\tilde{\mathbf{k}}_{\mathbf{x}_i}^{(p)} + \tilde{\mathbf{k}}_{\mathbf{x}_i}^{(n)}\right) \mathbf{e}_i^T \tag{4}$$

Moreover, if the leverage parameter of the i-th inducing point is to change from -1 to $+1$, the value of cross shape changes to minus of themselves. Then the difference matrix becomes

$$\mathbf{D} = 2\left(\mathbf{e}_i \tilde{\mathbf{k}}_{\mathbf{x}_i}^{(p)\,T} + \tilde{\mathbf{k}}_{\mathbf{x}_i}^{(p)} \mathbf{e}_i^T\right) \tag{5}$$

To update an SOGP model, the inverse of $\mathbf{K} - \mathbf{D}$ is required because SOGP model parameters incorporate inverse of kernel matrix of inducing set $\mathcal{I}_t$. We can quickly update model parameters using Woodbury formula [22]. Letting $\mathbf{U} = [\mathbf{e}_i, \tilde{\mathbf{k}}_{\mathbf{x}_i}]$ and $\mathbf{V} = -2[\tilde{\mathbf{k}}_{\mathbf{x}_i}, \mathbf{e}_i]$, update of inverse kernel matrix becomes

$$\begin{aligned} \mathbf{K}_{t+1}^{-1} &= (\mathbf{K}_t - \mathbf{D})^{-1} = \left(\mathbf{K}_t + \mathbf{U}\mathbf{V}^T\right)^{-1} \\ &= \mathbf{K}_t^{-1} - \mathbf{K}_t^{-1}\mathbf{U}\left(\mathbf{I} + \mathbf{V}^T\mathbf{K}_t^{-1}\mathbf{U}\right)^{-1}\mathbf{V}^T\mathbf{K}_t^{-1} \end{aligned} \tag{6}$$

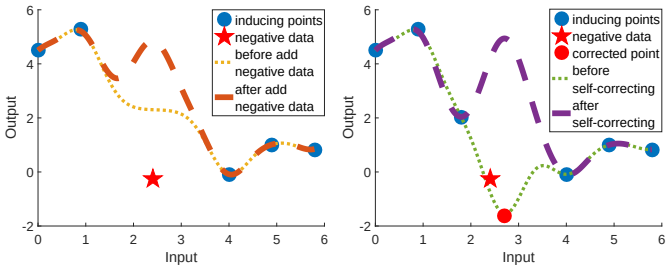
Using (6), online update rule of model parameters for *self-correcting* is proposed in Proposition 1.

Proposition 1. *The model parameter update rules for self-correcting are*

$$\boldsymbol{\alpha}_{t+1} = \boldsymbol{\alpha}_t + \mathbf{C}_t\mathbf{U}(\mathbf{I} - \mathbf{V}^T\mathbf{C}_t\mathbf{U})^{-1}\mathbf{V}^T\boldsymbol{\alpha}_t \tag{7}$$

$$\mathbf{C}_{t+1} = \mathbf{C}_t - \mathbf{C}_t\mathbf{U}(\mathbf{I} - \mathbf{V}^T\mathbf{C}_t\mathbf{U})^{-1}\mathbf{V}^T\mathbf{C}_t \tag{8}$$

$$\mathbf{Q}_{t+1} = \mathbf{Q}_t - \mathbf{Q}_t\mathbf{U}(\mathbf{I} + \mathbf{V}^T\mathbf{Q}_t\mathbf{U})^{-1}\mathbf{V}^T\mathbf{Q}_t \tag{9}$$



(a) Memorizing (b) Self-Correcting

Fig. 2: Effects of *memorzing* and *self-correcting*.

Proof. From [20], $\mathbf{Q}_t = \mathbf{K}_t^{-1}$. Replacing $\mathbf{K}_t^{-1}$ with $\mathbf{Q}_t$ in Eq. (6) obtains (9). From Eq. (8) in [20], $K_t(\mathbf{x}, \mathbf{x}') = K_0(\mathbf{x}, \mathbf{x}') + \mathbf{k}_{\mathbf{x}}^T \mathbf{C}_t \mathbf{k}_{\mathbf{x}'}$. Where $\mathbf{k}_{\mathbf{x}}$ denotes kernel vector of $\mathbf{x}$ w.r.t. training data. Because $K_t(\mathbf{x}, \mathbf{x}')$ is the covariance of GPs posterior, it is the same as $K_0(\mathbf{x}, \mathbf{x}') - \mathbf{k}_{\mathbf{x}}^T (\mathbf{K}_t + \sigma_n \mathbf{I})^{-1} \mathbf{k}_{\mathbf{x}}$ [23]. Hence, $\mathbf{C}_t = -(\mathbf{K}_t + \sigma_n \mathbf{I})^{-1}$ and replacing $\mathbf{K}_t^{-1}$ with $-\mathbf{C}_t$ obtains Eq. (8). From Eq. (8) in [20], $\langle f_{\mathbf{x}} \rangle_t = \boldsymbol{\alpha}^T \mathbf{k}_{\mathbf{x}}$ which is the mean of GPs posterior is represented as a linear combination of feature vectors of training data in RKHS. Hence $\boldsymbol{\alpha}_t = (\mathbf{K}_t + \sigma_n \mathbf{I})^{-1} \mathbf{y}_t$ [23] where $\mathbf{y}_t$ denotes output vector corresponding to $\{\mathbf{x}\}_{i=1}^t$. Multiplying $\mathbf{y}_t$ on both side of Eq. (8) yields Eq. (7). □

The computational complexity of *self-correcting* is $O(m^2|\mathcal{S}| + m|\mathcal{S}|^2)$ where m is the size of the inducing set $\mathcal{I}_t$ and the $|\cdot|$ denotes the cardinality operation. Since $|\mathcal{S}| \ll m$ in general, it is faster than re-training the model by changing leverage parameters. The pseudo code of *self-correcting* is described in Algorithm 1.

4.2. Effects of Memorizing and Self-Correcting

The effects of *memorizing* and *self-correcting* are illustrated in Fig. 2. Figure 2a shows *memorizing* effect. Dotted line shows posterior mean of the SOGP model trained on blue circles. When a negative sample occurs where training data is scarce, the proposed method (SCON) learns it as a negative. After learning, the prediction shifts apart from the newly added negative sample as dashed line. In Fig. 2b, SCON is initially trained on blue and red circles which are denser than Fig. 2a. Leverage parameters of inducing points are initially set to positive. When negative sample occurs near the inducing points, *self-correcting* occurs and the red circle is updated to be negative. The prediction shifts apart from the negatively changed inducing point. Note that both *memorizing* and *self-correcting* exhibit similar behavior to the same negative sample—drift apart from the negative sample. With this principle, the agent can avoid the malady actions while navigating.

4.3. Navigation Workflow

Consider a robot navigating in a dynamic environment. Let $\pi : \mathcal{A} \rightarrow \mathcal{S}$ denotes the policy function for navigation. The policy model π_0 is initially trained with positive data demonstrated by an expert. We assume that the robot is equipped with failure check routine which can discriminate the next state is ICS or not.[2] Let $\mathbf{x}_t$ denotes a current state and $\mathbf{u}_t$ denotes the action determined by current policy model π_t at time t. The failure check routine checks whether it is ICS or not. Once the action $\mathbf{u}_t$ is determined as a negative data, the proposed method robot learns it immediately. The robot computes a subset of inducing set, $\mathcal{S}$, according to Section 4.1. If $\mathcal{S} = \varnothing$, *memorizing* is conducted for learning and otherwise, *self-correcting* is conducted. Then the robot navigates according to updated policy π_{t+1}. If the action is not a negative, the robot skips learning procedure and navigates according to the current policy π_t. The pseudo code of the whole navigation process is described in Algorithm 2.

Algorithm 2 Self-correcting online navigation

Input: $\{(\mathbf{x}_i, \mathbf{u}_i)\}_{i=1}^N$ collected from demonstrator, $\mathbf{x}_0$ initial position, ϵ_{th}
Output: $f^{(T)}$

Initialization : $f^{(0)}$ trained via *Leveraged* SOGP with $\{\gamma_i = 1\}_{i=1}^N$

for $t = 0$ to T **do**
  $\mathbf{u}_t \leftarrow \langle f_{\mathbf{x}_t}^{(t)} | \, \gamma_t = 1 \rangle$
  if $(\mathbf{x}_t, \mathbf{u}_t)$ is negative **then**
    $\mathcal{S} \leftarrow \{m \mid |k(\mathbf{x}_m, \mathbf{x}_t)| \geq \epsilon_{th}, \; m \in \mathcal{I}\}$
    if $\mathcal{S} = \varnothing$ **then**
      $f^{(t+1)} \leftarrow$ Memorizing$(f^{(t)}, \mathbf{x}_t, \mathbf{u}_t, \gamma = -1)$
    else
      $f^{(t+1)} \leftarrow$ Self_correcting$(f^{(t)}, \mathbf{x}_t, \mathcal{S})$
    end if
    $u_t \leftarrow \langle f_{\mathbf{x}_t}^{(t+1)} | \, \gamma = 1 \rangle$
  end if
  $\mathbf{x}_{t+1} \leftarrow$ Navigate$(\mathbf{x}_t, \mathbf{u}_t)$
end for
return $f^{(T)}$

5. EXPERIMENTS

In this section, the performance of self-correcting online navigation (SCON) algorithm is validated by comparing with existing LfD algorithms on three different cases: navigation in a static planar world, in dynamic worlds on both simulation and real-world dataset.

5.1. Navigation in a Static World

The behavior of the proposed method (SCON) is studied in a static world. The obstacle configuration is static, however, the configuration at navigation time differs from the configuration when the demonstration is generated. Positive demonstrations are generated from the expert's reward map illustrated as a background in Fig. 3 without any obstacles. The paths of demonstrations proceed to the center of the map where the reward value is the highest.

The proposed method (SCON) is compared with naïve GPs algorithm. Simulations are conducted on 8 different starting positions. To evaluate the effect of SCON, it is tested twice for the same starting points keeping the updated model at the end of the first episode to an initial in the second episode. Each

[2]Failure check routine need not be a sophisticated function. Instead simple collision check algorithm based on linear dynamics prediction model works fine in this paper.

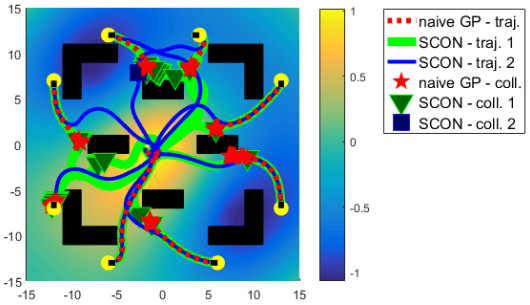


Fig. 3: Simulation result in the planner obstructed world.

trajectory consists of 100 time steps. The resulting paths are illustrated as lines and the positions where the collision occurs are marked as symbols in Fig. 3. At the first episode, SCON and naïve GPs collide at similar positions. However, SCON regards failure inducing actions as negatives and corrects its policies after experiencing few times of failures. As a result, the robot escapes from the failed situations and reach the highest reward region as illustrated in thick green lines in Fig. 3. At the second episode, the trajectories of SCON are bent apart from the collision points occurring at the first episode as described in thin blue lines in Fig. 3. The drifting behavior is due to the result of correcting policy.

TABLE 2: Average number of collision in the static world

	episode 1	episode 2
naïve GP	28.88	–
SCON	6.88	**0.13**

The average number of collision for 8 different starting points is summarized in Table 2. The proposed method (SCON) collides less than naïve GPs in the first episode as it escapes from the failures while naïve GP being stuck in repeating collisions. The collision rate of SCON gets lower at second episode as it learns what actions to avoid in the previous.

5.2. Navigation in a Simulated Dynamic World

Before testing the performance of SCON in a dynamic world with real world dataset, it is intensively validated with crowd motion generator. The simulator emulates Pioneer 3DX mobile robot equipped with two Microsoft Kinect cameras whose combined field of view (FOV) is 120° and eight sonar sensor to avoid an immediate collision as in [3]. The policy function for navigation is based on autoregressive Gaussian motion controller (AR-GPMC) [3]. It controls the robot based on observing three consecutive relative position of a moving obstacle as a state. The proposed method (SCON) is compared to naïve GPs, leveraged GPs [10], and Donut [16] controller. The training data is consisted of two types, positive and negative. Positive training data consist of behaviors of avoidance and negative data consists of adversarial behaviors which collide with obstacle. Naïve GP, Donut, and SCON are initially trained with 340 positive data whereas leveraged GP is initially trained with both 340 positive and negative data. The parameter ϵ_{th} is set to 0.4 for SCON. While naïve GP and leveraged GP models unchange through time, Donut and

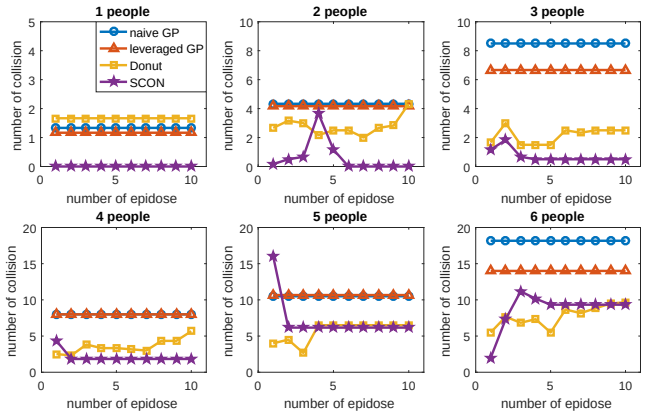


Fig. 4: Simulation result for simulated dynamic world

TABLE 3: Average number of collision in the simulated dynamic world

Number of people	1	2	3	4	5	6
naïve GP	1.33	4.33	8.50	8.00	10.50	18.17
leveraged GP	1.17	4.17	6.67	8.00	10.67	14.00
Donut	1.67	2.78	2.15	3.58	**5.67**	**7.77**
SCON	**0.00**	**0.62**	**0.72**	**2.08**	7.15	8.67

SCON learns and update the models in online to utilize failures encountered in execution. In simulations, different numbers of moving people, from 1 to 6, are tested. For each number of people, ten times of episode are tested to see cumulative effect of online learning. Each episode consists of four different types of moving patterns. The results are summarized in Table 3 and the learning curve is illustrated in Fig. 4. The number of collision of leveraged GP is not larger than that of naïve GP and remarkably smaller when the number of people is 3 and 6. The online learning algorithms, SCON and Donut, show better performances than non-online algorithms because they can adjust policies adaptively to the failure by not doing so. The proposed method (SCON) shows the best performance when the number of people is one to four and Donut shows the best performance when the number of people is 5 and 6. However, the more average number of collision of SCON is due to the high collision in the early stage of episodes. After all, the number of collision of SCON become comparable to that of Donut.

5.3. Navigation in a Real Dynamic World

The proposed algorithm is validated with naïve GP, leveraged GP, and Donut in a real dynamic world. The walking motion of a crowd is recorded from a sidewalk in front of a hotel [24]. As the video consists incomplete annotation, we tested algorithms on a selected frames for four times. The results is summarized in Fig. 6 and in Table 4. Similar to the result of Section 5.2, the proposed method (SCON) shows the lowest average number of collision followed by the Donut, leveraged GP in an order. Fig. 5 depicts four snapshots of running state at forth trial. First row of the Fig. 5 depicts the video frames for each time step augmented with the robot position symboled as a red square. Second row of the Fig.

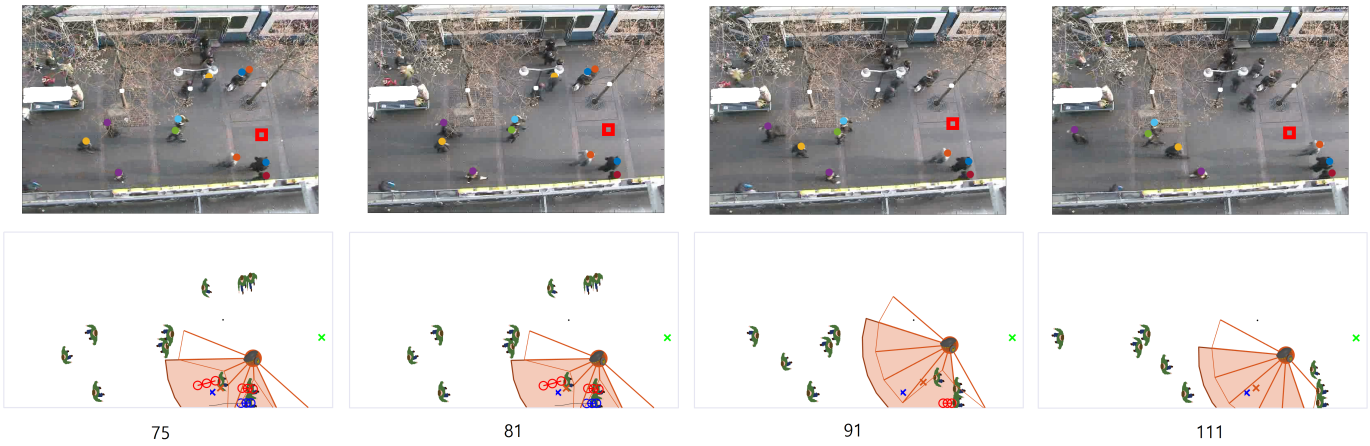


Fig. 5: Snapshots of navigation in the real dynamic world. The number below represents time step for the trajectory.

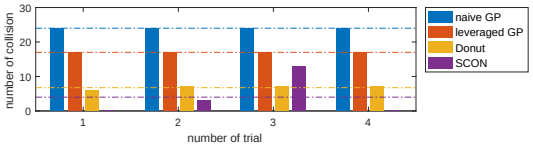


Fig. 6: Results of the navigation in the real dynamic world

TABLE 4: Average number of collision in the real dynamic world

naïve GP	leveraged GP	Donut	SCON
24.00	17.00	6.75	**4.00**

5 depicts internal perception states of a robot for each time step. It is remarkable that although the initial demonstration does not contain backward movement, SCON learns backward movement to avoid failure while navigating.

6. CONCLUSIONS

In this paper, the self-correcting online navigation framework is proposed by incorporating the leveraged GPs and SOGPs. The proposed method (SCON) can incorporate additional negative data generated from the failure and can correct its policy function while navigating. The performance of the proposed method is intensively validated in the simulated static world and dynamic world on both simulation and real-world dataset. The proposed method achieves the best performance on the simulated dynamics when the number of people is 1 to 4 and second best performance when the number of people is 5 to 6 in the sense of average number of collision. In real world dataset, SCON achieves the lowest average number collision. In most cases, SCON outperforms other LfD based navigation algorithms and it is expected to navigate successfully by learning itself to avoid failed experiences in an online manner.

ACKNOWLEDGMENT

This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIP) (NO.2014-0-00147, Basic Software Research in Human-level Lifelong Machine Learning).

REFERENCES

[1] D. Silver, J. A. Bagnell, and A. Stentz, "Learning from demonstration for autonomous navigation in complex unstructured terrain," *The International Journal of Robotics Research*, vol. 29, no. 12, pp. 1565–1592, 2010.
[2] P. Henry, C. Vollmer, B. Ferris, and D. Fox, "Learning to navigate through crowded environments," in *Proc. of the International Conference on Robotics and Automation (ICRA)*. IEEE, 2010.
[3] S. Choi, E. Kim, and S. Oh, "Real-time navigation in crowded dynamic environments using gaussian process motion control," in *Proc. of the International Conference on Robotics and Automation (ICRA)*. IEEE, 2014.
[4] S. Chernova and M. Veloso, "Multi-thresholded approach to demonstration selection for interactive robot learning," in *Proc. of the ACM/IEEE International Conference on Human-Robot Interaction*. ACM, 2008.
[5] S. Ross, G. J. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning." in *AISTATS*, vol. 1, no. 2, 2011, p. 6.
[6] B. Argall, B. Browning, and M. Veloso, "Learning by demonstration with critique from a human teacher," in *Proc. of the ACM/IEEE International Conference on Human-Robot Interaction*. IEEE, 2007.
[7] U. Nehmzow, O. Akanyeti, C. Weinrich, T. Kyriacou, and S. A. Billings, "Robot programming by demonstration through system identification," in *Proc. of the International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2007, pp. 801–806.
[8] A. Chella, H. Dindo, and I. Infantino, "A cognitive framework for imitation learning," *Robotics and Autonomous Systems*, vol. 54, no. 5, pp. 403–408, 2006.
[9] M. A. K. Jaradat, M. Al-Rousan, and L. Quadan, "Reinforcement based mobile robot navigation in dynamic environment," *Robotics and Computer-Integrated Manufacturing*, vol. 27, no. 1, pp. 135–149, 2011.
[10] S. Choi, E. Kim, K. Lee, and S. Oh, "Leveraged non-stationary gaussian process regression for autonomous robot navigation," in *Proc. of the International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 473–478.
[11] E. Snelson and Z. Ghahramani, "Sparse gaussian processes using pseudo-inputs," in *Advances in neural information processing systems*, 2006.
[12] S. S. Ge and Y. J. Cui, "Dynamic motion planning for mobile robots using potential field method," *Autonomous robots*, vol. 13, no. 3, pp. 207–222, 2002.
[13] K.-T. Song and C. C. Chang, "Reactive navigation in dynamic environment using a multisensor predictor," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 29, no. 6, pp. 870–880, 1999.
[14] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and autonomous systems*, vol. 57, no. 5, pp. 469–483, 2009.
[15] S. Choi, K. Lee, and S. Oh, "Robust learning from demonstration using leveraged gaussian processes and sparse-constrained optimization," in *Proc. of the International Conference on Robotics and Automation (ICRA)*, 2016.
[16] D. H. Grollman and A. G. Billard, "Robot learning from failed demonstrations," *International Journal of Social Robotics*, vol. 4, no. 4, pp. 331–342, 2012.
[17] S. Ross and D. Bagnell, "Efficient reductions for imitation learning." in *AISTATS*, vol. 3, no. 2, 2010, pp. 3–5.
[18] C. Bouveyron and C. Brunet-Saumard, "Model-based clustering of high-dimensional data: A review," *Computational Statistics & Data Analysis*, vol. 71, pp. 52–78, 2014.
[19] A. J. Smola, P. Bartlett, *et al.*, "Sparse greedy gaussian process regression," in *Advances in neural information processing systems*, 2001.
[20] L. Csató and M. Opper, "Sparse on-line gaussian processes," *Neural computation*, vol. 14, no. 3, pp. 641–668, 2002.
[21] T. Fraichard and H. Asama, "Inevitable collision statesa step towards safer robots?" *Advanced Robotics*, vol. 18, no. 10, pp. 1001–1024, 2004.
[22] M. A. Woodbury, "Inverting modified matrices," *Memorandum report*, vol. 42, p. 106, 1950.
[23] C. E. Rasmussen, *Gaussian processes for machine learning*. Citeseer, 2006.
[24] S. Pellegrini, A. Ess, K. Schindler, and L. Van Gool, "You'll never walk alone: Modeling social behavior for multi-target tracking," in *Proc. of the International Conference on Computer Vision*. IEEE, 2009, pp. 261–268.