Synthetic LiFi Channel Model Using Generative Adversarial Networks

Ardimas Andi Purwita[*], Anil Yesilkaya[†], and Harald Haas[†]
[*]Computer Science Department, Faculty of Computing and Media, Bina Nusantara University
[†]LiFi R&D Centre, Department of Electronic & Electrical Engineering, The University of Strathclyde
E-mail: ardimas.purwita@binus.edu, {a.yesilkaya, harald.haas}@strath.ac.uk

Abstract—**In this paper, we present our research on modeling a synthetic light fidelity (LiFi) channel model that uses a deep learning architecture called generative adversarial networks (GAN). A research in LiFi that requires the generation of many multipath channel impulse responses (CIRs) can benefit from our proposed model. For example, future developments of autonomous (deep learning-based) network management systems that use LiFi as one of its high-speed wireless access technologies might require a dataset of many CIRs. In this paper, we use TimeGAN, which is a GAN architecture for time-series data. We will show that modifications are necessary to adopt TimeGAN in our use case. Consequently, synthetic CIRs generated by our model can track long-term dependency of LiFi multipath CIRs. The Kullback–Leibler divergence (KLD) is used in this paper to measure the small difference between samples of synthetic CIRs and real CIRs. Lastly, we also show a simple demonstration of our model that can run on a small virtual machine hosted over the Internet.**

I. Introduction

Due to its high speed advantage while occupying the unlicensed spectrum, optical wireless communications (OWC) or light fidelity (LiFi) is frequently mentioned as a way to enable technologies for future wireless systems, such as 6G [1]. With the advancement of LiFi in many aspects, such as: data rates, capacity, miniaturisation, standardisation, security, etc., LiFi will soon be ready for mainstream industries and consumer applications [2].

Similar to any wireless communication system, the study of LiFi propagation channels are critical for the development of LiFi [3]. However, many researchers that focus on deterministic approaches, e.g., recursive or iterative models [4], [5], limit their assumptions to use at most three orders of reflections. For example, Wu *et al.* assume only an order of reflection [6], while Miramirkhani and Uysal assume three reflections [3]. The main reason for this is a long computation time and high memory requirement if a higher order of reflections are considered [5]. In addition, it is known that the order of reflections does not affect the direct current (DC) channel gain as much as having the higher order bounces does, which leads to low order modeling assumptions in the literature, e.g., [3]. Nevertheless, this justification can be inaccurate since the difference of DC channel gains of two channel impulse responses (CIRs) obtained by assuming three reflections and higher order of reflections can be 7 dB, see [7, Fig. 6]. In [8], Zhou *et al.* conclude that for a high-speed LiFi, it is required to assume a high order of reflections as magnitude responses of LiFi channels can fluctuate up to 10 dB at high frequency ranges (see [9, Fig. 7]), which cannot be captured by only considering a low order of reflections. Therefore, it is important to consider a high order of reflections for studies of high-speed LiFi for future wireless systems.

Similar to recursive or iterative models, good accuracy can be obtained by non-deterministic approaches, such as Monte Carlo ray tracing (MCRT) [10], [11], in exchange for computation time. It is reported in [5], [12] that the generation of optical CIRs could take anywhere from several hours to several days. The long simulation time might not be a problem for a study that only takes into consideration different situations on a case-by-case basis, e.g., evaluating error performances of a modulation technique in a fixed location with varying parameters of optics. However, in order to draw a general conclusion, e.g., that a modulation technique is better than another technique, one must show that the conclusion applies to all users' positions. For example, in [13, Fig 3], it is shown that error performances of a modulation technique are generally not good (i.e., the bit error ratio can be greater than 10^{-2} in a relatively wide ares of an indoor scenario), and they are only good in certain locations.

Generating many accurate CIRs, e.g., for studies of high-speed LiFi and deep learning, by using recursive or MCRT approaches will be time-consuming and often infeasible. Another option is to use geometric models, such as those explained in [14], [15]. However, geometric models also suffer from limited accuracy [16]. Thus, another approach is to use an approximation as in [17], but it is not straightforward to generalize and apply these to scenarios that include furniture, or consider a higher order of reflections. In order to overcome all the previously mentioned problems, in this paper, we explore a generative approach as in [18], where synthetic magnitude responses could be generated by using a deep learning architecture called generative adversarial networks (GAN) [19] and its variants. By using GAN, a synthetic CIR can be generated from a noise sample. The benefit of using GAN to generate CIRs is that the heavy computation is moved to the pre-training (i.e., generating a dataset of CIRs) and training phases (i.e., training GAN). We will later show that the proposed CIR generator can run on a single-core central processing unit (CPU) and 1 GB of memory, where the service is hosted over the Internet. Another benefit is that we do not need to store a big volume of CIR data. Instead, one can generate a CIR simply by drawing a realization from a distribution function. To the best of our knowledge, there is no similar work that has been published before.

The rest of this paper is organized as follows. In Section II, we will first briefly review LiFi channel model and discuss our lightweight simulator that can generate many CIRs, which will be used to train our GAN. Then, in Section III, we will discuss GAN and its variant, TimeGAN [20], to generate time-series data. In Section IV, we will discuss our methodology to generate a synthetic LiFi CIRs. Results and discussions will be discussed in Section V. Section VI concludes this paper.

II. LiFi Channel Model and Our Simulator

In this section, we will briefly review a general LiFi channel model and our simulator named `owcsimpy`[1], which will be used to generate a massive dataset of LiFi CIRs to train the GAN architecture later.

A. LiFi Channel Model

We will explain the LiFi channel model that is based on deterministic approaches following [4], [5] and [7]. Light Emitting Diodes (LEDs), photodiodes (PDs), and diffuse reflecting surfaces can be modeled by an elemental object denoted by ε having the following attributes:

$$\varepsilon = \{\mathbf{p}, \mathbf{n}, m, A, \Psi_c, R\}, \quad (1)$$

where $\mathbf{p}$ is a position vector, $\mathbf{n}$ is a normal vector, m is the mode number of the Lambertian radiation, A is a detection area, Ψ_c is field-of-view (FoV), and R is a reflectivity. An elemental object can act as a source (denoted by ε^s) or act as a receiver (denoted by ε^r). That is, ε^s and ε^r are defined as follows:

$$\varepsilon^s, \varepsilon^r \in \varepsilon, \text{ where } \varepsilon^s = \{\mathbf{p}, \mathbf{n}, m\}, \text{ and } \varepsilon^r = \{\mathbf{p}, \mathbf{n}, A, \Psi_c, R\}.$$

Let an LED be defined as $S = \varepsilon^s$ and a PD be defined as $R = \varepsilon^r$, then the CIR $h(t; S, R)$ of a source S and a receiver R is defined as:

$$h(t; S, R) = \sum_{k=0}^{\infty} h^{(k)}(t; S, R), \quad (2)$$

where $h^{(k)}(t; S, R)$ is the CIR that is calculated by only assuming k reflections. Based on [4], $h^{(k)}(t; S, R)$ can be approximated as:

$$h^{(k)}(t; S, R) \approx \sum_{i=0}^{N-1} h^{(0)}(t; S, \varepsilon_i^r) \circledast h^{(k-1)}(t; \varepsilon_i^s, R), \quad (3)$$

where N is the total number of elemental objects ε that are obtained by partitioning solid objects, and the notation $\circledast$ denotes the convolution operation. The initial condition of the above equations can be calculated as follows:

$$h^{(0)}(t; \varepsilon_i^s, \varepsilon_j^r) = \frac{(m_i + 1)A_j}{2\pi d_{ij}^2} \cos^{m_i}(\phi) \cos(\theta) \mathbb{1}(\theta) V_{ij}, \quad (4)$$

where $\mathbb{1}(\theta)$ is an indicator function for $0 \le \theta \le \Psi_{cj}$, and V_{ij} is the visibility function that evaluates if the path between ε_i^s and ε_j^r is not blocked. The distance between ε_i and ε_j is denoted by d_{ij}. The incident angle between ε_i^s and ε_j^r is denoted by ϕ, while θ is the angle between ε_j^r and ε_i^s.

B. Simulator to Generate Data

There are a few simulators that can be used to generate CIRs. However, to the best of our knowledge, most of them require us to install licensed programs, such as MATLAB®. Examples of such programs are introduced in [21], [22]. In addition to the need to install a licensed software, another limitation is that we can only define models that are aligned with the x-axis or y-axis with [21], [22], for example [5, Figs 1 and 10]. This limited capability restricts us to define our 3D objects that might face or move in any direction. Another licensed simulator is Zemax®[2], which is a well-known ray tracing tool. Another limitation of Zemax® is that only Windows operating systems are supported, which require another license to run.

In this paper, in order to train GAN, a massive number of CIRs is required, which is commonly found in many deep learning architectures [23]. We generated 10 million CIRs to train our GAN model. Therefore, it is imperative to have a lightweight simulator that can be installed and run on many small computing instances simultaneously. It means that we need to eliminate the options that depend on licensed programs. Hence, we use our open-source simulator called `owcsimpy` that is mainly written in Python and C languages and only depends on free libraries such as NumPy, Matplotlib, CBLAS, CLAPACK, OpenMP, etc. Note that `owcsimpy` currently supports the iterative model from [5], the frequency-domain model from [7], and the geometric model from [15]. We installed and deployed `owcsimpy` in hundreds of free-tier Amazon Elastic Compute Cloud (Amazon EC2) instances to generate the dataset. The fact that `owcsimpy` can run on a Cloud computing service means that we can directly store the generated CIRs to Cloud computing database services and connect it with deep learning containers to train our GAN.

III. Generative Adversarial Networks (GAN) and TimeGAN

In this paper, we will briefly review GAN and TimeGAN since they are foundational architectures for our model.

A. Generative Adversarial Networks (GAN)

The vanilla GAN [19] consists of two networks, namely the generative network (denoted by G) and discriminative network (denoted by D). Suppose that G is a multilayer perceptron with parameters or weights $\mathbf{w}_g$, then G is a function that maps a noise $\mathbf{z}$ drawn from density function $p_{\mathbf{Z}}(\mathbf{z})$ to a data space, e.g., images, audios, etc. Therefore, G can be denoted as $G(\mathbf{z}; \mathbf{w}_g)$. The discriminative network D can be also modeled by a multilayer perceptron $D(\mathbf{x}; \mathbf{w}_g) \in [0, 1]$, where $\mathbf{x}$ is a realization from the data space, and D outputs a probability

[1]https://github.com/ardimasp/owcsimpy

[2]https://www.zemax.com/products/opticstudio

of $\mathbf{x}$ being a real data or a synthetic data. Both networks are trained via the so-called adversarial process which is defined as:

$$\min_G \max_D \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}} [\log (D(\mathbf{x}))]\, \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{Z}}} [\log (1 - D(G(\mathbf{z})))] .$$

In the game theory, the optimization problem above is also called as a two-player minimax game since the network G is trained to fool the network D. Then, both networks can be trained by using the prominent backpropagation and dropout algorithms by Hinton *et al.* [24].

B. Time-Series GAN (TimeGAN)

TimeGAN is a GAN for time-series data. In this paper, we treat CIRs denoted in (2) as time-series data. Let a time-series data be denoted by a realization of a random process $\mathbf{X}$, i.e., $\mathbf{x}_{1:T}$, where T is a random variable. In addition, we can also generate a static data that corresponds to the temporal data $\mathbf{x}$, namely $\mathbf{s} \sim p_{\mathbf{S}}$.

TimeGAN is made up of four networks, i.e., an embedding function ($\mathtt{e}$), a recovery function ($\mathtt{r}$), a generator ($\mathtt{g}$), and a discriminator ($\mathtt{d}$). Suppose that $\mathcal{X}$ and $\mathcal{S}$ are vector spaces that contain $\mathbf{x}$ and $\mathbf{s}$, respectively. Furthermore, let $\mathcal{Z}$ be a vector space that contains noise $\mathbf{z}$. Then, the four networks are defined as follows:

$$\begin{aligned}
\mathtt{e} &: \mathcal{S} \times \prod_t \mathcal{X} \to \mathcal{H}_{\mathcal{S}} \times \prod_t \mathcal{H}_{\mathcal{X}}, \\
\mathtt{r} &: \mathcal{H}_{\mathcal{S}} \times \prod_t \mathcal{H}_{\mathcal{X}} \to \mathcal{S} \times \prod_t \mathcal{X}, \\
\mathtt{g} &: \mathcal{Z}_{\mathcal{S}} \times \prod_t \mathcal{Z}_{\mathcal{X}} \to \mathcal{H}_{\mathcal{S}} \times \prod_t \mathcal{H}_{\mathcal{X}}, \text{ and} \\
\mathtt{d} &: \mathcal{H}_{\mathcal{S}} \times \prod_t \mathcal{H}_{\mathcal{X}} \to [0,1] \times \prod_t [0,1],
\end{aligned}$$

where $\mathcal{H}$ is a latent vector space. Loss functions that are used to train the networks are:

$$\begin{aligned}
\mathcal{L}_{\mathrm{r}} =& \mathbb{E}_{\mathbf{s},\mathbf{x}_{1:T}} \left[||\mathbf{s} - \hat{\mathbf{s}}||_2 + \sum_t ||\mathbf{x}_t - \hat{\mathbf{x}}_t||_2 \right], \\
\mathcal{L}_{\mathrm{u}} =& \mathbb{E}_{\mathbf{s},\mathbf{x}_{1:T}} \left[\log y_{\mathcal{S}} + \sum_t \log y_t \right] + \\
& \mathbb{E}_{\mathbf{s},\mathbf{x}_{1:T}} \left[\log(1 - \hat{y}_{\mathcal{S}}) + \sum_t \log(1 - \hat{y}_t) \right], \text{ and} \\
\mathcal{L}_{\mathrm{s}} =& \mathbb{E}_{\mathbf{s},\mathbf{x}_{1:T}} \left[\sum_t ||\mathbf{h}_t - \mathtt{g}_{\mathcal{X}}(\hat{\mathbf{h}}_{\mathcal{S}}, \hat{\mathbf{h}}_{t-1}, \mathbf{z}_t)||_2 \right],
\end{aligned} \tag{5}$$

where:

$$\hat{\mathbf{h}}_t = \mathtt{g}_{\mathcal{X}}(\hat{\mathbf{h}}_{\mathcal{S}}, \hat{\mathbf{h}}_{t-1}, \mathbf{z}_t),\ \hat{\mathbf{h}}_{\mathcal{S}} = \mathtt{g}_{\mathcal{S}}(\mathbf{z}_{\mathcal{S}}), \tag{6}$$

and $||\cdot||_2$ denotes L2-norm. Let $\mathbf{w}_{\mathrm{e}}, \mathbf{w}_{\mathrm{r}}, \mathbf{w}_{\mathrm{g}},$ and $\mathbf{w}_{\mathrm{d}}$ be the weights of the embedding function, the recovery function, the generator, and the discriminator, respectively. Then, the networks are trained based on the following optimization problems:

$$\min_{\mathbf{w}_{\mathrm{e}},\mathbf{w}_{\mathrm{r}}} (\lambda \mathcal{L}_{\mathrm{s}} + \mathcal{L}_{\mathrm{r}}) \text{ and } \min_{\mathbf{w}_{\mathrm{g}}} (\eta \mathcal{L}_{\mathrm{s}} + \max_{\mathbf{w}_{\mathrm{d}}} \mathcal{L}_{\mathrm{u}}), \tag{7}$$

where $\lambda, \eta \geq 0$ are hyperparameters during the training phase. Intuitively speaking, according to [20] the above optimization problems try to approximate the following objectives:

$$\min_{\hat{p}} D\left(p_{\mathbf{S},\mathbf{X}} || \hat{p}_{\mathbf{S},\mathbf{X}}\right) \text{ and } \min_{\hat{p}} D\left(p_{\mathbf{X}_t|\mathbf{S},\mathbf{X}_{1:t-1}} || \hat{p}_{\mathbf{X}_t|\mathbf{S},\mathbf{X}_{1:t-1}}\right),$$

where D denotes a distance measure between the true distribution p and the estimated distribution $\hat{p}$, e.g., Kullback-Leibler divergence (KLD). Finally, Algorithm 1 summarizes a step-by-step method to generate a synthetic time-series data from a noise sample drawn from an independent and identically-distributed (IID) $p_{\mathcal{Z}}$.

Algorithm 1 Pseudocode of TimeGan

```
1: Sample $(\mathbf{z}_{\mathcal{S}}, \mathbf{z}_{\mathcal{X}}) \overset{\text{IID}}{\sim} p_{\mathcal{Z}}$
2: Generate synthetic latent codes by calculating (6)
3: Mapping to the feature space:
4:     $(\hat{\mathbf{s}}, \hat{\mathbf{x}}_{1:T}) = (\mathtt{r}_{\mathcal{S}}(\hat{\mathbf{h}}_{\mathcal{S}}), \mathtt{r}_{\mathcal{X}}(\hat{\mathbf{h}}_{\mathcal{X}}))$
```

IV. Methodology

In this section, we will explain our methodology in implementing TimeGAN for generating a synthetic LiFi CIR. First, we will discuss modifications that we apply to the vanilla TimeGAN so that it better fits our case. Then, we will elaborate our training methodology. We will also explain the performance metrics used in this paper.

A. Modifications

In [20], the authors use basic recurrent networks for all four networks, i.e., $\mathtt{e}$, $\mathtt{r}$, $\mathtt{g}$, and $\mathtt{d}$. However, it is well known that recurrent networks have problems tracking the long-term dependency problem [23]. The problem could affect our model in generating synthetic LiFi CIRs as the recursive calculation shown in (2) indicates a long-term dependency. That is, the impulse from, for example, the second-order reflection affects the calculation of impulses coming from a higher order of reflections, which will arrive much later compared to impulses coming from a lower order of reflections. In order to capture the long-term dependencies of time-series data, a typical network that is frequently used is the long short-term memory (LSTM) network. Therefore, in this paper we use the LSTM networks for $\mathtt{e}$, $\mathtt{r}$, $\mathtt{g}$, and $\mathtt{d}$. In the next section, we will provide results that use recurrent neural networks vs. LSTM and show that the former network cannot capture long-term dependency.

In our case, $\mathcal{S}$ is a set of geometry locations, e.g., locations of users, PDs, furniture, etc, and $\mathcal{X}$ is a set of LiFi CIRs. Based on (5), $\mathcal{L}_{\mathrm{r}}$ is computed based on samples $\mathbf{s}$, $\hat{\mathbf{s}}$, $\mathbf{x}$, and $\hat{\mathbf{x}}$. Meanwhile, $\hat{\mathbf{s}}$ and $\hat{\mathbf{x}}$ are calculated based on the estimated latent codes $\hat{\mathbf{h}}$ individually. Therefore, there could be an error in generating $\hat{\mathbf{x}}$ from $\hat{\mathbf{h}}_{\mathcal{X}}$. We will show later that by adding another loss function that takes an advantage of `owcsimpy`,

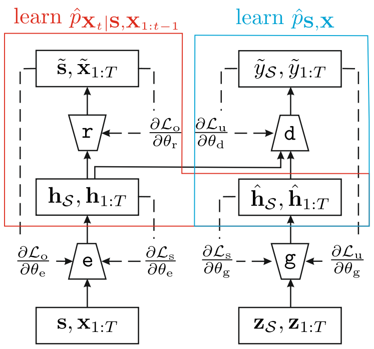

learn $\hat{p}_{\mathbf{x}_t|\mathbf{s},\mathbf{x}_{1:t-1}}$
learn $\hat{p}_{\mathbf{s},\mathbf{x}}$
$\tilde{\mathbf{s}}, \tilde{\mathbf{x}}_{1:T}$
$\tilde{y}_{\mathcal{S}}, \tilde{y}_{1:T}$
r
$\frac{\partial \mathcal{L}_\text{o}}{\partial \theta_\text{r}}$
$\frac{\partial \mathcal{L}_\text{u}}{\partial \theta_\text{d}}$
d
$\mathbf{h}_{\mathcal{S}}, \mathbf{h}_{1:T}$
$\hat{\mathbf{h}}_{\mathcal{S}}, \hat{\mathbf{h}}_{1:T}$
$\frac{\partial \mathcal{L}_\text{o}}{\partial \theta_\text{e}}$
e
$\frac{\partial \mathcal{L}_\text{s}}{\partial \theta_\text{e}}$
$\frac{\partial \mathcal{L}_\text{s}}{\partial \theta_\text{g}}$
g
$\frac{\partial \mathcal{L}_\text{u}}{\partial \theta_\text{g}}$
$\mathbf{s}, \mathbf{x}_{1:T}$
$\mathbf{z}_{\mathcal{S}}, \mathbf{z}_{1:T}$

Fig. 1. Training of TimeGAN, where solid lines show forward propagation, and dashed lines show backward propagation.

we can improve the performance. Specifically, $\mathcal{L}_\text{r}$ is modified into:

$$\mathcal{L}_\text{o} = \mathbb{E}_{\mathbf{s},\mathbf{x}_{1:T}} \left[||\mathbf{s} - \hat{\mathbf{s}}||_2 + \sum_t ||\mathbf{x}_t - \hat{\mathbf{x}}_t||_2 + \sum_t ||\mathbf{x}_t - \text{o}(\hat{\mathbf{s}})||_2 \right], \tag{8}$$

where o is either `owcsimpy` or a deep learning model that has been trained to model `owcsimpy`. Therefore, our optimization problem becomes:

$$\min_{\mathbf{w}_\text{e},\mathbf{w}_\text{r}} (\lambda \mathcal{L}_\text{s} + \mathcal{L}_\text{o}) \text{ and } \min_{\mathbf{w}_\text{g}} \left(\eta \mathcal{L}_\text{s} + \max_{\mathbf{w}_\text{d}} \mathcal{L}_\text{u} \right), \tag{9}$$

where $\mathcal{L}_\text{r}$ is replaced by $\mathcal{L}_\text{o}$.

B. Training

The training process of the TimeGAN that uses stochastic gradient descent (SGD) is summarized in Algorithm 2 and illustrated in Fig. 4. The main ingredient of the training is a dataset of geometry configurations ($\mathbf{s}$) and time-series CIRs ($\mathbf{x}$). In other words, $\mathbf{s}$ is a collection of $\mathcal{E}^\text{s}, \mathcal{E}^\text{r}$, furniture locations, human locations, and room dimensions, and $\mathbf{x}$ is the corresponding CIR $h(t; S, R)$. In this paper, we focus only on a specific environment in order to prove the fact the TimeGAN can be reliably used to generate synthetic CIRs for LiFi. Specifically, we assume a 4 m × 3 m × 3 m office room with a desk and a chair as depicted in Fig. 2(a). Solid objects are modeled with collections of rectangular planes as illustrated in Fig. 2(b). Orientations of the LiFi access point and the mobile device are represented by arrows. In `owcsimpy`, a red circle in Fig. 2(b) is used to distinguish the role of a mobile device, i.e., whether it acts as a receiver or not. Therefore, Fig. 2(b) illustrates a downlink scenario.

Table I summarizes our LSTM-based TimeGAN architecture. The input layer handles 1000 inputs, and we use multiple networks in the hidden layer. We use three LSTM cells followed by a fully connected layer. A rectified linear unit (ReLU) activation function is used and followed by a dropout layer with a 20% dropout rate to prevent overfitting. Another stack of LSTM cell and a fully-connected layer are used before the regression output layer.

Algorithm 2 Pseudocode of Training TimeGan

```
    Input: $\lambda = 1$, $\eta = 10$, batch size 128, learning rate 0.01
    Initialize: $\mathbf{w}_\text{e}$, $\mathbf{w}_\text{r}$, $\mathbf{w}_\text{g}$, $\mathbf{w}_\text{d}$
 1: while not converge do
 2:     (1) Map the feature space to the latent space
 3:     Sample $(\mathbf{s}_1, \mathbf{x}_{1,1:T_1}), \ldots, (\mathbf{s}_{n_\text{b}}, \mathbf{x}_{1,1:T_{n_\text{b}}})$
 4:     for $n = 1, \ldots, n_\text{b}$, $t = 1, \ldots, T_n$ do
 5:         $(\mathbf{h}_{n,\mathcal{S}}, \mathbf{h}_{n,t}) = (\text{e}_{\mathcal{S}}(\mathbf{s}_n), \text{e}_{\mathcal{X}}(\mathbf{h}_{n,\mathcal{S}}, \mathbf{h}_{n,t-1}, \mathbf{x}_{n,t})))$
 6:         $(\tilde{\mathbf{s}}_n, \tilde{\mathbf{x}}_{n,t}) = (\text{r}_{\mathcal{S}}(\mathbf{h}_{n,\mathcal{S}}), \text{r}_{\mathcal{X}}(\mathbf{h}_{n,t}))$
 7:     end for
 8:     (2) Generate synthetic latent codes
 9:     Sample $(\mathbf{z}_{\mathcal{S},1}, \mathbf{z}_{1,1:T_1}), \ldots, (\mathbf{z}_{\mathcal{S},n}, \mathbf{z}_{1,1:T_n}) \overset{\text{IID}}{\sim} p_{\mathcal{Z}}$
10:     for $n = 1, \ldots, n_\text{b}$, $t = 1, \ldots, T_n$ do
11:         $(\hat{\mathbf{h}}_{n,\mathcal{S}}, \hat{\mathbf{h}}_{n,t}) = (\text{g}_{\mathcal{S}}(\mathbf{z}_{\mathcal{S},n}), \text{g}_{\mathcal{X}}(\hat{\mathbf{h}}_{n,\mathcal{S}}, \hat{\mathbf{h}}_{n,t-1}, \mathbf{z}_{n,t})))$
12:     end for
13:     (3) Distinguish between real and synthetic data
14:     for $n = 1, \ldots, n_\text{b}$ do
15:         $(\hat{y}_{n,\mathcal{S}}, \hat{y}_{n,1:T_n}) = \text{d}_{\mathcal{S}}(\mathbf{h}_{n,\mathcal{S}}, \mathbf{h}_{n,1:T_n}, \hat{\mathbf{h}}_{n,\mathcal{S}}, \hat{\mathbf{h}}_{n,1:T_n})$
16:     end for
17:     (4) Calculate losses
18:     Calculate $\mathcal{L}_\text{o}$, $\mathcal{L}_\text{u}$, $\mathcal{L}_\text{s}$ according to (5) and (8)
19:     (5) Update the weights via SGD
20:     $\mathbf{w}_\text{e} = \mathbf{w}_\text{e} - \gamma \nabla_{\mathbf{w}_\text{e}} - (\lambda \mathcal{L}_\text{s} + \mathcal{L}_\text{o})$
21:     $\mathbf{w}_\text{r} = \mathbf{w}_\text{r} - \gamma \nabla_{\mathbf{w}_\text{r}} - (\lambda \mathcal{L}_\text{s} + \mathcal{L}_\text{o})$
22:     $\mathbf{w}_\text{g} = \mathbf{w}_\text{g} - \gamma \nabla_{\mathbf{w}_\text{g}} - (\lambda \mathcal{L}_\text{u} + \mathcal{L}_\text{o})$
23:     $\mathbf{w}_\text{d} = \mathbf{w}_\text{d} - \gamma \nabla_{\mathbf{w}_\text{d}} - \mathcal{L}_\text{u}$
24: end while
```

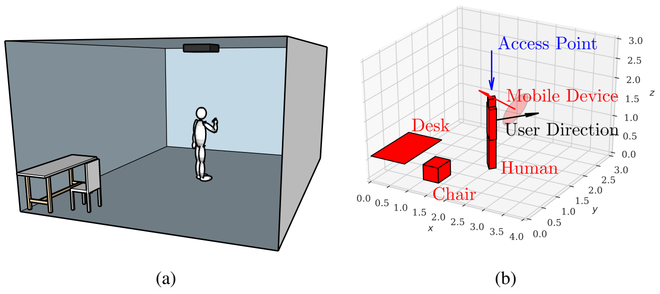


Fig. 2. (a) A sketch of a simple office scenario, and (b) its model by using `owcsimpy`.

C. Performance Metrics

In this paper, we use the KLD estimator from [25] to measure the difference between distributions of real data and synthetic data. Given a sample $\mathbf{x}_{1:T}$ drawn from $p_\mathbf{X}$, then the empirical distribution is:

$$P_\text{e}(x) = \frac{1}{T} \sum_{t=1}^{T} U(x - x_t), \tag{10}$$

where U is a step function with $U(0) = 0.5$. The empirical distribution P_e is then used as a reference to calculate a function as:

$$P_\text{c}(x) = \begin{cases} 0, & x < x_0 \\ a_t x + b_t, & x_{t-1} \le x < x_t \\ 1, & x_{T+1} < x \end{cases} \tag{11}$$

Table I. TimeGAN-LSTM Architecture

Name and Type	**Activation Properties**	**Details**	**States**
Sequence Input: 1×1000	N/A	-	N/A
lstm_1: LSTM Hidden Units: 1000	State activation function: tanh Gate activation function: sigmoid	Input Weights: 4000×1 Recurrent Weights: 4000×1000 Bias: 4000×1	Hidden state: 1000×1 Cell state: 1000×1
lstm_2: LSTM Hidden Units: 500	State activation function: tanh Gate activation function: sigmoid	Input Weights: 2000×1000 Recurrent Weights: 100×500 Bias: 2000×1	Hidden state: 500×1 Cell state: 500×1
lstm_3: LSTM Hidden Units: 250	State activation function: tanh Gate activation function: sigmoid	Input Weights: 1000×500 Recurrent Weights: 100×250 Bias: 1000×1	Hidden state: 250×1 Cell state: 250×1
fc_1: Fully connected	N/A	Weights: 100×250 Bias: 100×1	N/A
relu_1: ReLU	N/A	-	N/A
do: Dropout 20%	N/A	-	N/A
lstm_4: LSTM Hidden Units: 500	State activation function: tanh Gate activation function: sigmoid	Input Weights: 4000×100 Recurrent Weights: 4000×1000 Bias: 4000×1	Hidden state: 500×1 Cell state: 500×1
fc_2: Fully connected	N/A	Weights: 1×500 Bias: 1×1	N/A
Regression output	Loss functions	-	N/A

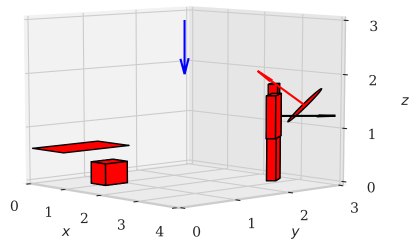
where a_t and b_t are defined to ensure that $P_c(x)$ takes the same value as $P_e(x)$ at the sampled values. Then, the KLD estimator between distribution P and Q is:

$$\hat{D}(P||Q) = \frac{1}{T}\sum_{t=1}^{T} \log \frac{\delta P_c(x_t)}{\delta Q_c(x_t)}, \tag{12}$$

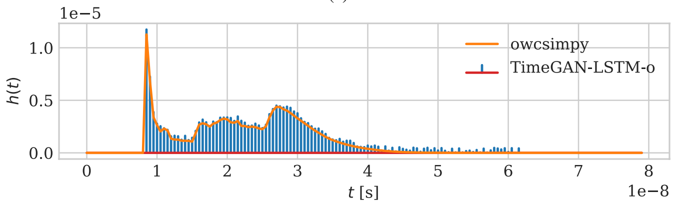
where $\delta P_c(x_t) = P_c(x_t) - P_c(x_t - \epsilon)$ for any $\epsilon < \min_t\{x_t - x_{t-1}\}$. We will use this estimator to measure the difference between the distribution of root mean square (RMS) of delay spread calculated from the dataset of CIRs and synthetic CIRs.

V. Results and Discussions

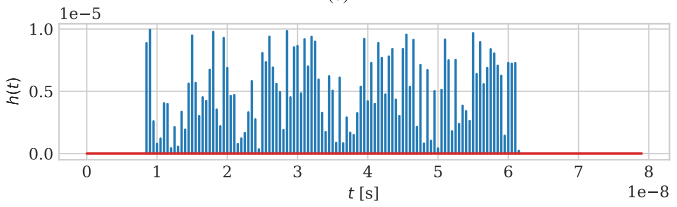
In this paper, we will investigate two LSTM-based models with different loss functions, i.e. the one with $\mathcal{L}_r$ and $\mathcal{L}_o$. (Recall that in Section IV, we hypothesized that by using $\mathcal{L}_o$, the performance can be improved as the network `r` also directly learns from `owcsimpy`, which generates the real data.) We will refer the first model as 'TimeGAN-LSTM-r', and the second model as 'TimeGAN-LSTM-o'. However, before focusing on this comparison, we would like to highlight our finding that standard recursive networks cannot capture the long-term dependency of LiFi CIRs. This fact can be shown by Fig. 3. Fig. 3(a) shows an estimated geometry configuration that is tied with the synthetic CIR shown in Fig. 3(b) by using the TimeGAN-LSTM-o model. Note that the synthetic CIR is compared by CIR obtained from `owcsimpy` having the configuration shown in Fig. 3(a). It can be seen that the TimeGAN-LSTM-o model can generate a synthetic CIR that is quite similar to the real one. Unlike the TimeGAN-LSTM-o model, recursive networks cannot generate a decent synthetic CIR as shown in Fig. 3(c). It appears that the impulses in synthetic CIR shown in Fig. 3(c) are random, which can be an indicator of a model that cannot track long-term dependency.



(a)



(b)



(c)

Fig. 3. (a) Estimated configuration from TimeGAN-LSTM-o, (b) CIR generated by TimeGAN-LSTM-o compared with CIR calculated by `owcsimpy`, and (c) CIR generated by TimeGAN with recursive networks.

Now, we will discuss the benefit of using our modified loss function $\mathcal{L}_o$ compared to $\mathcal{L}_r$. Fig. 4 shows the training curve of the total loss. It can be seen that the training performance is better if $\mathcal{L}_o$ is used. As the result, which can be seen in Fig. 5 the cumulative distribution function (CDF) of the RMS delay spread is closer to the CDF of CIRs from the dataset, which is generated by using `owcsimpy`. The estimated KLD

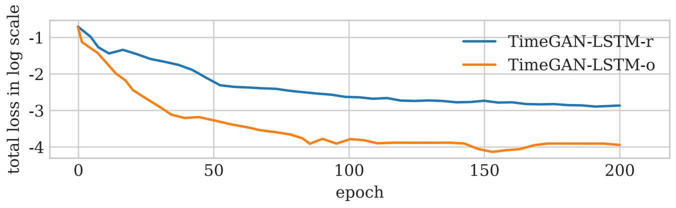


Fig. 4. Comparisons of training between different loss functions.

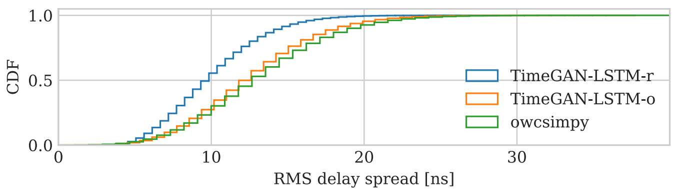


Fig. 5. Comparisons of CDFs.

.

of the TimeGAN-LSTM-o model is 0.0873, which is much lower compared to that of the TimeGAN-LSTM-r model that has 0.326 of the estimated KLD. We also tried to run the TimeGAN-LSTM-o model on a 1-vCPU AWS EC2 as our proof-of-concept[3]. For our future work, we will explore various environments, such as an open office and industrial environments, among others.

VI. Conclusions

In this paper, we discussed a LiFi channel model that can generate synthetic CIRs having similarity real data. We used TimeGAN as our base deep learning architecture. In order to generate a massive dataset of CIRs to be used to train our deep learning model, we used our lightweight simulator, `owcsimpy`, that can be deployed in many small computing instances. Then, we used LSTM as the recurrent network in the TimeGAN in order to capture the long-term dependency of LiFi CIRs. We also modified the loss function in order to improve the performance of our TimeGAN. Finally, we show that KLD of 0.0873 can be achieved by our proposed model compared to that of 0.326 of KLD with the original setup.

Acknowledgment

This research has been supported in part by EPSRC under Established Career Fellowship Grant EP/R007101/1, Wolfson Foundation and European Commission's Horizon 2020 research and innovation program under grant agreement 871428, 5G-CLARITY project.

References

[1] L. Bariah, L. Mohjazi, S. Muhaidat, P. C. Sofotasios, G. K. Kurt, H. Yanikomeroglu, and O. A. Dobre, "A prospective look: Key enabling technologies, applications and open research topics in 6G networks," *IEEE Access*, vol. 8, pp. 174 792–174 820, 2020.

[2] A. A. Banham, R. R. Schaeffer, and S. S. Scace, "LiFi is Ready for Mainstream," in *2021 OFC*. San Francisco, CA, USA: IEEE, 2021, pp. 1–4.

[3] F. Miramirkhani and M. Uysal, "Channel modelling for indoor visible light communications," *Philosophical Transactions of the Royal Society A*, vol. 378, no. 2169, p. 20190187, 2020.

[4] J. R. Barry, J. M. Kahn, W. J. Krause, E. A. Lee, and D. G. Messerschmitt, "Simulation of multipath impulse response for indoor wireless optical channels," *IEEE JSAC*, vol. 11, no. 3, pp. 367–379, 1993.

[5] J. B. Carruthers and P. Kannan, "Iterative site-based modeling for wireless infrared channels," *IEEE IEEE Trans. Antennas Propag.*, vol. 50, no. 5, pp. 759–765, 2002.

[6] Z.-Y. Wu, M. Ismail, J. Kong, E. Serpedin, and J. Wang, "Channel characterization and realization of mobile optical wireless communications," *IEEE Trans. on Commun.*, vol. 68, no. 10, pp. 6426–6439, 2020.

[7] H. Schulze, "Frequency-domain simulation of the indoor wireless optical communication channel," *IEEE Trans. on Commun.*, vol. 64, no. 6, pp. 2551–2562, 2016.

[8] Z. Zhou, C. Chen, and M. Kavehrad, "Impact analyses of high-order light reflections on indoor optical wireless channel model and calibration," *J. Lightw. Technol.*, vol. 32, no. 10, pp. 2003–2011, 2014.

[9] M. Uysal, F. Miramirkhani, T. Baykas, and K. Qaraqe. (2018, November) IEEE 802.11bb reference channel models for indoor environments. [Online]. Available: https://mentor.ieee.org/802.11/dcn/18/11-18-1582-04-00bb-ieee-802-11bb-reference-channel-models-for-indoor-environments.pdf

[10] F. Lopez-Hernandez, R. Perez-Jimeniz, and A. Santamaria, "Monte Carlo calculation of impulse response on diffuse IR wireless indoor channels," *Electronics Letters*, vol. 34, no. 12, pp. 1260–1262, 1998.

[11] M. S. Chowdhury, W. Zhang, and M. Kavehrad, "Combined deterministic and modified Monte Carlo method for calculating impulse responses of indoor optical wireless channels," *J. Lightw. Technol.*, vol. 32, no. 18, pp. 3132–3148, 2014.

[12] F. Miramirkhani, M. Uysal, and E. Panayirci, "Channel modeling for visible light communications," in *Optical Wireless Communications*. Springer, 2016, pp. 107–122.

[13] L. Zeng, D. C. O'Brien, H. Le Minh, G. E. Faulkner, K. Lee, D. Jung, Y. Oh, and E. T. Won, "High data rate multiple input multiple output (MIMO) optical wireless communications using white LED lighting," *IEEE JSAC*, vol. 27, no. 9, pp. 1654–1662, 2009.

[14] J. B. Carruthers and J. M. Kahn, "Modeling of nondirected wireless infrared channels," *IEEE Trans. on Commun.*, vol. 45, no. 10, pp. 1260–1268, 1997.

[15] V. Jungnickel, V. Pohl, S. Nonnig, and C. Von Helmolt, "A physical model of the wireless infrared communication channel," *IEEE JSAC*, vol. 20, no. 3, pp. 631–640, 2002.

[16] S. Yahia, Y. Meraihi, A. Ramdane-Cherif, A. B. Gabis, D. Acheli, and H. Guan, "A survey of channel modeling techniques for Visible Light Communications," *Journal of Network and Computer Applications*, p. 103206, 2021.

[17] C. Chen, D. A. Basnayaka, X. Wu, and H. Haas, "Efficient analytical calculation of non-line-of-sight channel impulse response in visible light communications," *Journal of Lightw. Technol.*, vol. 36, no. 9, pp. 1666–1682, 2017.

[18] D. Righini, N. A. Letizia, and A. M. Tonello, "Synthetic power line communications channel generation with autoencoders and GANs," in *2019 IEEE SmartGridComm*. Beijing, China: IEEE, 2019, pp. 1–6.

[19] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[20] J. Yoon, D. Jarrett, and M. van der Schaar, "Time-series Generative Adversarial Networks," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019.

[21] J. B. Carruthers and P. Kannan, "IRSIMIT– Infrared Impulse Response Simulator by Iterations," 2002. [Online]. Available: http://iss.bu.edu/bwc/irsimit/

[22] M. B. Rahaim, T. Borogovac, and J. B. Carruthers, "CandlES: Communication and Lighting Emulation Software," in *Proceedings of the Fifth ACM WiNTECH*. New York, NY, USA: Association for Computing Machinery, 2010, p. 9–14.

[23] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[24] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[25] F. Pérez-Cruz, "Kullback-Leibler divergence estimation of continuous distributions," in *2008 IEEE International Symposium on Information Theory (ISIT)*. Toronto, ON, Canada: IEEE, 2008, pp. 1666–1670.

[3]Please visit https://bit.ly/owcsimpy-gan-first-demo for a simple demo.