SINERGI Vol. 25, No. 2, June 2021: 141-152
http://publikasi.mercubuana.ac.id/index.php/sinergi
http://doi.org/10.22441/sinergi.2021.2.005

SELF-COLLISION AVOIDANCE OF ARM ROBOT USING GENERATIVE ADVERSARIAL NETWORK AND PARTICLES SWARM OPTIMIZATION (GAN-PSO)

Zendi Iklima[1*], Andi Adriansyah[1], Sabin Hitimana[2]
[1]Department of Electrical Engineering, Faculty of Engineering, Universitas Mercu Buana, Indonesia
[2]Department of Information Technology, Kigali Independent University, Rwanda

Abstract
Collision avoidance of Arm Robot is designed for the robot to collide objects, collide the environment, and collide its body. Self-collision avoidance was successfully trained using Generative Adversarial Networks (GANs) and Particle Swarm Optimization (PSO). The Inverse Kinematics (IK) with 96K motion data was extracted as the dataset to train data distribution of $D(x)$ 3.6K samples and 7.2K samples. The proposed method GANs-PSO can solve the common GAN problem such as Mode Collapse or Helvetica Scenario that occurs when the generator $\boldsymbol{G}$ always gets the same output point which mapped to different input z values. The discriminator $\boldsymbol{D}$ produces the random samples' data distribution, presenting the real data distribution (generated by Inverse Kinematic analysis). The PSO was successfully reduced the number of training epochs of the generator $\boldsymbol{G}$ only with 5000 iterations. The result of our proposed method (GANs-PSO) with 50 particles was 5000 training epochs executed in 0.028ms per single prediction and 0.027474% Generator Mean Square Error (GMSE).

This is an open access article under the CC BY-NC license

Keywords:
Arm Robot;
Collision Avoidance;
Generative Adversarial Network;
Inverse Kinematics;
Particle Swarm Optimization;

Article History:
Received: June 14, 2020
Revised: July 7, 2020
Accepted: July 19, 2020
Published: Feb 5, 2021

Corresponding Author:
Zendi Iklima
Department of Electrical Engineering, Faculty of Engineering, Universitas Mercu Buana, Indonesia
Email: zendi.iklima@mercubuana.ac.id

INTRODUCTION

Collision risk assessment and collision avoidance of manipulator robot has been an interesting topic in autonomous robotics [1]. The manipulator robot is designed to simply human task, to reduce human error (e.g., remote teleoperation robot NASA's Robonaut), to carry out tasks that humans cannot handle it, and to perform in dangerous tasks [2][3]. Manipulator robots must work in a crowded or cluttered workspace, and have the ability to colliding the workspace, colliding its body, and colliding with each other [4][5].

Several studies to solve collision used Oriented Bounding Boxes (OBBs), Extended Oriented Bounding Boxes (EOBBs), Artificial Potential Field (APF) [6], Virtual Link, Path Planning combine with Kinematic Analysis [5][7], Neural Network [8], etc.

Kinematic Analysis commonly used to analyze the motion of a manipulator robot. It converts the robot manipulator's position and orientation from cartesian space to joint space defined as Inverse Kinematics (IK). A redundant robot has an infinite number of IK solutions. The evolutionary methods have been used to solve the IK problem. Some way of evolutionary are using a feed-forward Neural Network (NN) and its Back Propagation (BP), Genetic Algorithm (GA) [9], Particle Swarm Optimization (PSO) [10], Firefly Algorithm (FA) [11], Quantum PSO (QPSO) [12], Path Planning with PSO [13], etc.

Srisuk et al. explain the IK solution using Neural Network for a robotic arm in three dimensions. The IK-NN is defined by the network's optimal weight with an error rate of 5% [14]. Improves the network architecture utilizes a cycle of consistency to learn and solve the IK problem with a supervised or unsupervised manner. The motion's quantitative evaluation has a Mean Square Error (MSE) of 7.10% and 8.51% using a Cycle Consistency Objective

approach by adversarial. This MSE value prevents regressing to the end-effector positions of the input motion [15]. IK-ANN [16], IK-ANN-PSO [8], IK-RNN [17] was simulated for solving the IK problem. Still, this technique requires a lot of data to perform the training process and approximation, especially inverse kinematics and a robot manipulator's dynamics. To mapping, this problem as a serious dimensional problem can be processed efficiently within the data distribution.

Recent research used analytical models embedded inside a physics-based simulation, instead of using the real world's data set. The Reinforcement learning approach has been used to learn a closed-loop predictive controller for a real robot as IK solver [18]. This approach requires a huge dataset as an important role in training neural networks to approximate the result. To collect real-world data, consume a lot of time [19]. Generative Adversarial Networks (GANs) introduced by Ian Good Fellow in 2014 [20], describes how to generate additional 'fake' data similar to real-world data, and thereby enlarge the total dataset available for training target neural network [21, 22, 23]. Closely, Hailin Ren and Pinhas Ben-Tzvi [26] implements IK solver for 4-DOF MICO Robotic Manipulator using four types of GANs, namely Conditional GAN (CGAN), Least Square GAN (LSGAN), Bidirectional GAN (BiGAN) and DualGAN [27]. The Approaches proposed to solve IK and ID problems with the desired degree of accuracy and achieve a lower loss in the Performance and avoid overfitting. The training process of the proposed method takes 23 mins to train whole 60.000 datasets. The generator neural network's execution time takes 0.17ms in a single prediction with the best of loss 0.91 by GANs.

The main contributions of this paper are as follows:

- Use of GANs toward learning as the IK solver's self-organized motion where the real-world data generated itself in which the data is high dimensional inputs and distributed.
- Experiment to compare GANs performance with/without PSO to test the efficiency of the proposed GANs-PSO that evaluated using different sizes of the partial dataset and different deviations for the generator in the GANs.
- Collision avoidance dataset created with an IK generator function.
- The proposed method GANs and PSO capable to solve the common GAN problem such as Mode Collapse or Helvetica Scenario that occurs when the generator $\boldsymbol{G}$ always get the same output point though mapped to different input z values. The role of PSO optimizes the generator $\boldsymbol{G}$ Cost and Loss to solve the Mode Collapse Problem.

METHOD

System Design

The neural network structured of the common GANs contains two artificial neural networks of the generator $\boldsymbol{G}$ and the discriminator $\boldsymbol{D}$ [20][21] shown in Figure 1.

The generator $\boldsymbol{G}$ learns to create fake data by incorporating optimized feedback from the Discriminator [26]. The training process of the generator requires tighter integration than the discriminator training process.

In our method shows the portion of the GANs that trains the generator to include [20][21]:

- Random input (the end-effector position coordinated in x, y, z) distributed to the generator networks.
- Sample data collected from the generator by exponentially increased 10^n (where chosen n = 1, 2, 3, and 4)
- Generated data will be classified by the discriminator network (labelled real/fake)
- Generator loss, which penalizes the generator for failing to fool the Discriminator.
- Backpropagate through both the Discriminator and generator to obtain gradients and change only the generator weights.
- PSO used to optimize the backpropagation process.

The generator transforms the random input noise into a meaningful output to produce a wide variety of data, sampling from different places in the target distribution. The generator also a neural network needs to be trained. The neural network weights perform to reduce the error or loss of its output. The generator feeds into the discriminator network, and it produces the output to affect the next training process. Generator loss penalizes the generator for producing a sample that the discriminator network classifies as fake data distribution. So, this network must be included in a backpropagation process.

Backpropagation adjusts each layer (contains weight and biases) to impact the next cost function. In GANs, the impact of a generator weight depends on the impact of the discriminator weights it feeds into. So, backpropagation starts at the output and flows back through the Discriminator into the generator [28].

The Discriminator $\boldsymbol{D}$ is simply a classifier tries to distinguish real data from the data created by the generator [26][27]. The discriminator training data contains;

- **Real data** instances, in our system generated by inverse kinematic equation (to produce joint angular given by joint space/joint cartesian), the Discriminator labelled as a positive sample during the training process.
- **Fake data** instances created by the generator network labeled as a negative sample during the training process.

As shown in Figure 1, the Discriminator obtains two loss functions (loss function of $\boldsymbol{D}$ and loss function of $\boldsymbol{G}$). During the Discriminator is training, the Discriminator ignores the loss of the generator and uses the discriminator loss. The procedure during discriminator training [31]:

- Classifies both real and fake data from the generator
- The Discriminator updates its weights through backpropagation from the discriminator loss through the discriminator network.

The objective of the Optimizer shown in Figure 1 to optimize both of GANs neural networks. Adam Optimizer used as an optimizer architecture for advanced gradient algorithms [18][26].

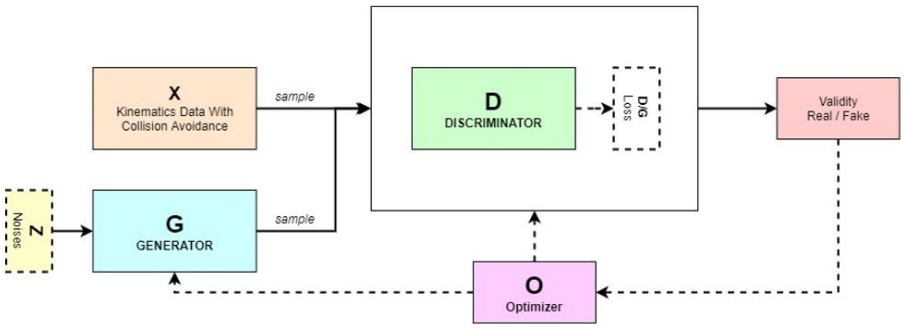

Figure 1. Network Structure of Generative Adversarial Networks

Kinematics Analysis

A manipulator robot's design is an important part of designing equations to assist in the feasibility of the manipulator robot. It is defined as transforming the geometrical equation to connect between joint spatial geometry concept and end-effector coordinates. The forward kinematics is described as transforming the joint space/joint angle to the cartesian space/joint variable (end-effector). Vice versa, inverse kinematics transform cartesian space / joint variable to joint space / joint angle [24] [25] as shown in Figure 2.

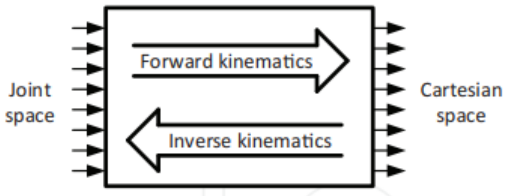

Figure 2. The Schematic of a Forward Kinematics and Inverse Kinematics [24]

Qinsheng explained Denavit-Hartenberg Convention is parameters related to a particular convention for enclosing the reference frames to the links of a kinematic chain/manipulator [24]. The DH has four parameters contain:

- Join offset (d_i): Offset distance among the common normal of the axis of $join\ i-1$ to $join\ i$. Figure 3 shows that d_i is the offset in a direction $X_0\ to\ X_1\ over\ Z_1$ on the other ways represented as $X_{i-1}\ to\ X_i\ over\ Z_i$.
- Join angle (θ_i): An angle measured among the common normal of the axis of $join\ i-1$ to $join\ i$. Figure 3 shown that θ_i is an angle rotated over Z_1 on the other ways represented as an angle measured among the common normal of the axis $X_{i-1}\ to\ X_i$ over Z_i.
- Link length (r_i): The length of the link is a common normal length between the axis of $join\ i-1$ and $join\ i$. Figure 3 shows that r_i is a link length measured from Z_1 to Z_2 over X_2 or might be represented as the link length of $Z_{i-1}\ to\ Z_i$ over X_i.
- Twist angle (α_i): Twist of the link is an angle measured between the axis of $join\ i-1$ and $join\ i$. Figure 3 shown that α_i is an angle measured between Z_0 and Z_1 over X_1 ($Z_{i-1}\ and\ Z_i\ over\ X_i$).

After determining the coordinates shown in Figure 3, the DH parameters can be represented in Table 1 [24].

Table 1. The Denavit-Hartenberg Parameters

Link-n	θ (radian)	d (mm)	α (radian)	r (mm)
1	θ_1	d_1	α_1	0
2	θ_2	0	0	r_1
3	θ_3	0	0	r_2

Table 1 used to define the DH matrix convention such as

$$ {}^{i-1}_{i}T = \begin{bmatrix} c(\theta_i) & -c(\alpha_i).s(\theta_i) & s(\alpha_i).s(\theta_i) & r_i.c(\theta_i) \\ s(\theta_i) & c(\alpha_i).c(\theta_i) & -s(\alpha_i).c(\theta_i) & r_i.s(\theta_i) \\ 0 & s(\alpha_i) & c(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1) $$

The join offset measured $d_i = 30mm$, twist angle measured $\alpha_i = 90^o$. The link length of joint-1 equal to zero (0), the link length of joint-2 is defined as r_1 measured $r_1 = 60mm$, and the link length of joint-3 is defined as r_2 measured $r_2 = 90mm$. $c(\theta_i)$ denotes as $cos(\theta_i)$, $s(\theta_i)$ denotes as $sin(\theta_i)$, $c(\alpha_i)$ denotes as $cos(\alpha_i)$, and $s(\alpha_i)$ denotes as $sin(\alpha_i)$.

Transformation of among the links from 0 to n mathematically defined:

$$ {}^{0}_{3}\boldsymbol{T} = {}^{0}_{1}\boldsymbol{T}\,{}^{1}_{2}\boldsymbol{T}\,{}^{2}_{3}\boldsymbol{T} \quad (2) $$

Equation (2) shows the connection of joint and links from the base frame to frame i, so the translation for each join can be defined by replacing the variables from (1) and Table 1, then [24]:

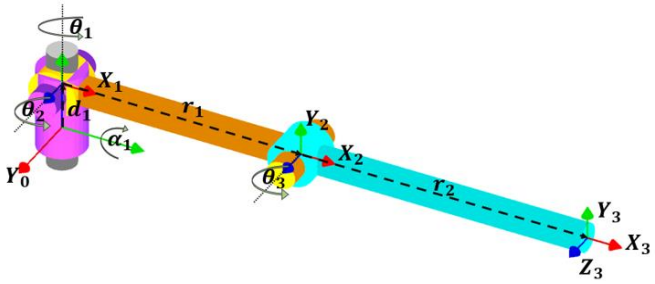

Figure 3. Local Reference Coordinate of 3 DoF Manipulator in Arm Robot [24]

$$ {}^{0}_{1}T = \begin{bmatrix} c(\theta_1) & 0 & s(\theta_1) & 0 \\ s(\theta_1) & 0 & -c(\theta_1) & 0 \\ 0 & 1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3) $$

$$ {}^{1}_{2}T = \begin{bmatrix} c(\theta_2) & -s(\theta_2) & 0 & r_1c(\theta_2) \\ s(\theta_2) & c(\theta_2) & 0 & r_1s(\theta_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4) $$

$$ {}^{2}_{3}T = \begin{bmatrix} c(\theta_3) & -s(\theta_3) & 0 & r_2c(\theta_3) \\ s(\theta_3) & c(\theta_3) & 0 & r_2s(\theta_3) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5) $$

The Forward Kinematic solution can be solved by multiplying (3), (4), and (5).

A representation of learning IK is a system would be to generate samples of (q, ξ), where, $\xi \in \Re^m$, is a vector containing the coordinates of the end-effector given by

$$ \xi = f(\boldsymbol{q}) \quad (6) $$

where $f(.)$ is a nonlinear direct kinematic function of the manipulator robot, and $q \in \Re^n$, is the vector containing the joint space configuration vector, and to learn the mapping $\xi \rightarrow q$. Refer to (2), $f(\boldsymbol{q})$ denotes as a Forward Kinematic solution of the manipulator robot in which calculating the joint space parameters (p_x, p_y, p_z) based on its input $\boldsymbol{q}$.

The end-effector linear velocity and angular velocity can be realized as formulized (7) [8]

$$ \dot{\xi} = \boldsymbol{J}\dot{\boldsymbol{q}} \quad (7) $$

Where $\boldsymbol{J}$ is the geometric Jacobian Matrix of the manipulator robot. The Jacobian $\boldsymbol{J}(\boldsymbol{q})$ of a forward kinematics is matrix 6x3 can be expressed as:

$$ \boldsymbol{J} = \begin{bmatrix} \frac{\partial x_0^D(\theta_1,\theta_2,\theta_3)}{\partial \theta_1} & \cdots & \frac{\partial z_0^D(\theta_1,\theta_2,\theta_3)}{\partial \theta_1} \\ \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots \\ \frac{\partial \omega x_0^D(\theta_1,\theta_2,\theta_3)}{\partial \theta_3} & \cdots & \frac{\partial \omega z_0^D(\theta_1,\theta_2,\theta_3)}{\partial \theta_3} \end{bmatrix} \quad (8) $$

The $\dot{\xi}$ is the velocity of end-effector in the base frame defined as

$$ \dot{\xi} = \begin{bmatrix} v_i^0 \\ \omega_i^0 \end{bmatrix} \quad (9) $$

v_i^0 is a linear velocity from the base frame $(\dot{x}_0^0, \dot{y}_0^0, \dot{z}_0^0)$ to end-effector $(\dot{x}_2^0, \dot{y}_2^0, \dot{z}_2^0)$ and can be derived as an angular velocity from the base frame $(\omega_{x0}^0, \omega_{y0}^0, \omega_{z0}^0)$ to frame-i $(\omega_{x2}^0, \omega_{y2}^0, \omega_{z2}^0)$,

$$ \dot{\xi} = \begin{bmatrix} \dot{x}_i^0 \\ \dot{y}_i^0 \\ \dot{z}_i^0 \\ \omega_{xi}^0 \\ \omega_{yi}^0 \\ \omega_{zi}^0 \end{bmatrix} \quad (10) $$

$\dot{\boldsymbol{q}}$ in the (7) is an angular parameter for each joint formulized as,

$$\dot{\boldsymbol{q}} = \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix} \quad (11)$$

These $\dot{\boldsymbol{q}}$ in (10) the angular joint from the base frame to frame-n to formalized revolute joint of $\dot{\theta}_1$, meanwhile for prismatic joint defined as $\dot{d}$.

Equation (7) and (10) $\dot{\xi}$ can be replaced as

$$\begin{bmatrix} \dot{x}_i^0 \\ \dot{y}_i^0 \\ \dot{z}_i^0 \\ \omega_{xi}^0 \\ \omega_{yi}^0 \\ \omega_{zi}^0 \end{bmatrix} = \dot{J} \cdot \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix} \quad (12)$$

Thus $\boldsymbol{J}$ has two parameters, as explained above, represented as:

$$\dot{J} = \begin{bmatrix} J_v \\ J_\omega \end{bmatrix} \quad (13)$$

where J_ω, denotes as the Jacobian matrices calculated by the partial derivative of the forward kinematic solution shown in (6) for the angular velocity of the manipulator robot that formulized as,

$$J_\omega = \begin{bmatrix} R_0^0.k & R_1^0.k & R_2^0.k \end{bmatrix} \quad (14)$$

The rotational in the base frame noticed as R_0^0, the rotational the base frame to frame 1 noticed as R_1^0, and the rotational the base frame to frame 2 noticed as R_2^0. k represents that the rotation always occurs in the z-axis then,

$$k = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad (15)$$

By looking (3), (4), and (5), the matrices R calculated as,

$$R_0^0 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad (16)$$

$$R_1^0 = \begin{bmatrix} s(\theta_1) \\ -c(\theta_1) \\ 0 \end{bmatrix} \quad (17)$$

$$R_2^0 = \begin{bmatrix} s(\theta_1) \\ -c(\theta_1) \\ 0 \end{bmatrix} \quad (18)$$

Therefore J_ω formulized as [24]

$$J_\omega = \begin{bmatrix} 0 & s(\theta_1) & s(\theta_1) \\ 0 & -c(\theta_2) & -c(\theta_2) \\ 1 & 0 & 0 \end{bmatrix} \quad (19)$$

$$J_v = \begin{bmatrix} R_{0.}^0 k \times (O_3^0 - O_0^0) & R_{1.}^0 k \times (O_3^0 - O_1^0) \\ & R_{2.}^0 k \times (O_3^0 - O_2^0) \end{bmatrix} \quad (20)$$

Furthermore, J_v denoted as the Jacobian matrices of the forward kinematic solution $f(.)$ shown in (6) to the linear velocity of the manipulator robot that formulized as, can be formalized as (24) where O is the origin from the base frame until end-effector, by looking (3) until (5), matrix O can be calculated as,

$$O_3^0 - O_0^0 = \begin{bmatrix} r_1 c(\theta_2) + r_2 c(\theta_3) \\ r_1 s(\theta_2) + r_2 s(\theta_3) \\ d_1 \end{bmatrix} \quad (21)$$

$$O_3^0 - O_1^0 = \begin{bmatrix} r_1 c(\theta_2) + r_2 c(\theta_3) \\ r_1 s(\theta_2) + r_2 s(\theta_3) \\ 0 \end{bmatrix} \quad (22)$$

$$O_3^0 - O_2^0 = \begin{bmatrix} r_2 c(\theta_3) \\ r_2 s(\theta_3) \\ 0 \end{bmatrix} \quad (23)$$

Equation (24) shows the initial value of J_v defined. Furthermore, (25) is the Jacobian Matrix of J contains J_ω and J_v.

Corresponding to (7), the linearity of the configuration Jacobian Matrix that allows solving the differential kinematics with an inversion of the following equations [8]

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{-1}\dot{\xi} \quad (24)$$

Equation (24) is valid only for the manipulators with the same dimension of the operational space and the join space $(m = n)$. When the manipulator is redundant $(m > n)$, the Jacobian Matrix has more columns than rows, and infinite number solutions exist for (7).

Generative Adversarial Network (GAN)

GANs can be utilized to produce new data in a limited situation. These data, sometimes be difficult and expensive and time-consuming to generate. The new data has to be realistic enough that whatever insights obtained from the generated data still applies to real data.

GANs was introduced in 2014 by Goodfellow et al. [30, 31, 32]. The original GANs can learn a generator to capture the distribution of real data by introducing an adversarial discriminator that evolves to discriminate between the actual data and the fake. GANs series widely proposed for a wide variety of problems. Figure 1 shows the original network structure of GANs.

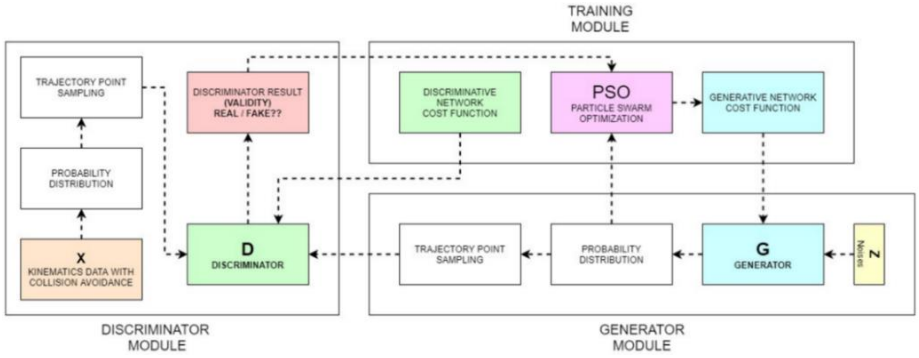

Figure 4. The Flowchart of Generative Adversarial Networks Modules

In the original formulation, GANs includes a generator $\boldsymbol{G}$ and a discriminator $\boldsymbol{D}$ [21]. Figure 4 shows the Self-Collision Avoidance System of Arm Robot's flowchart using GANs and PSO [26][34].

The adversarial framework is straightforward to implement when the models are both multilayers perceptron (MLP) but in this paper use the more than two layers of hidden layer called Artificial Neural Networks (ANNs).

$$J_v = \begin{bmatrix} -(r_1 s(\theta_2) + r_2 s(\theta_3)) & 0 & 0 \\ -(r_1 c(\theta_2) + r_2 c(\theta_3)) & 0 & 0 \\ 0 & r_1 c(\theta_1 - \theta_2) + r_2 c(\theta_1 + \theta_3) & r_2 c(\theta_1 - \theta_3) \end{bmatrix}$$

$$J = \begin{bmatrix} -(r_1 s(\theta_2) + r_2 s(\theta_3)) & 0 & 0 \\ -(r_1 c(\theta_2) + r_2 c(\theta_3)) & 0 & 0 \\ 0 & r_1 c(\theta_1 - \theta_2) + r_2 c(\theta_1 + \theta_3 & r_2 c(\theta_1 - \theta_3) \\ 0 & s(\theta_1) & s(\theta_1) \\ 0 & -c(\theta_2 & -c(\theta_2 \\ 1 & 0 & 0 \end{bmatrix} \quad (25)$$

The generator $\boldsymbol{G}$ learn the data distribution p_g over real data $\boldsymbol{x}$, the data distribution of $\boldsymbol{G}$ defined as input noise variables $p_z(Z)$, where $z \in Z$ is independent and identically distributed samples from a known prior p_z, to points in the space of real data $\boldsymbol{X}$. In this system variable z notice three parameters required as an input of the inverse kinematic equation (p_x, p_y, p_z). This $\boldsymbol{G}$ represents a mapping data space as $G(z;\theta_g)$, where $\boldsymbol{G}$ is a differentiable function represented by an ANNs with parameters θ_g.

This $G(z;\theta_g)$ is ANNs calculated consists of Feed-Forward step and Backpropagation step, formulized as [34, 35, 36];

$$y_j = f(net) \text{ then } net = \sum_{i=1}^{n} \omega_{ij} x_i - \theta_j \quad (26)$$

where y_j does weight give the network values ω_{ij} and threshold unit θ_j. This $f(.)$ is the activation function used to transform the incoming values from the previous layer to the successive layer. y_j represent the values from the input layer to the hidden layer y_i, hidden layer to next hidden layer y_h, and last hidden layer to output layer y_o [35][36].

The gradient descent required to calculated and minimize the error between observed data y_o and the desired data t_k. A measure of the error between both observed data y_o and the desired data t_k is

$$E = \frac{1}{2}\sum_{k=1}^{m}(t_k - y_o)^2 \quad (27)$$

Equation (27) represents the first step of backpropagation step, which the weights of the network changed each iteration of the training process [35][36]. The weights need to be updated each iteration by finding the values of the derivative of E in (27), which were given by

$$\triangle \omega_{ji} = -\gamma \frac{\partial E}{\partial \omega_{ji}} = \gamma \delta_j y_i \quad (28)$$

where the error δ_j is given as:

$$\delta_j = f'(\overline{y_o})(t_k - y_o) \quad (29)$$

f' is the derivative of the activation function $f(.)$ and γ is called learning rate. So, the weights updated when the input-hidden error calculated by [34]:

$$\delta_j = f'(\overline{y_i}) \sum_{i=1}^{n} \omega_{ji} \delta_j \quad (30)$$

The second ANNs $D(x;\theta_d)$ in which $D(x)$ represents the probability that x came from the data rather than p_g. By training the discriminator $\boldsymbol{D}$ to maximize the probability of assigning the correct label to training examples and samples from the generator $\boldsymbol{G}$.

That could be determined if a sample $x \in \mathcal{X}$ is from the real dataset $x{\sim}p_{data}(x)$ or generated from the generator $\boldsymbol{G}$, $x{\sim}G; z{\sim}p_z$. The training criterion of the discriminator $\boldsymbol{D}$, given any generator $\boldsymbol{G}$ expressed as,

$$V(D,G) = E_{x\sim P_{data}(x)}[logD(x)] + E_{z\sim P_z(z)}[\log(1-D(G(z)))] \quad (31)$$

Thus, the training process is to optimize the discriminator $\boldsymbol{D}$ to assign correct labels to both the real dataset and the noise sample from the generator $\boldsymbol{G}$, and simultaneously train the generator $\boldsymbol{G}$ to minimize,

$$minimize(\log(1-D(G(z)))) \quad (32)$$

When the generator $\boldsymbol{G}$ was unable to assign a label data as data distribution of $z{\sim}p_z$, in this case, $\log(1-D(G(z)))$ saturates. Rather than training the generator G to minimize shown in (30), the generator G can be maximized to provides the stronger gradients early in learning, formulized as:

$$maximize(\log(D(G(z)))) \quad (33)$$

Furthermore, minimax objective for both generator $\boldsymbol{G}$ and the discriminator $\boldsymbol{D}$ is formulated as follows [18, 20, 21, 26, 30, 32],

$$\min_G \max_D V(D,G) = E_{x\sim P_{data}(x)}[logD(x)] + E_{z\sim P_z(z)}[\log(1-D(G(z)))] \quad (34)$$

where, $D(.)$ denotes the discriminator $\boldsymbol{D}$ network's output, $G(.)$ denotes the generator $\boldsymbol{G}$ network's output, and $E[.]$ denotes the network expectation.

The generator $\boldsymbol{G}$ implicitly defines a probability distribution p_g as the distribution of the samples $G(z)$ obtained when $z{\sim}p_z$. A minibatch stochastic gradient descent training of GAN. A hyperparameter of k is the composer variables used to compute the generator loss and the discriminator loss $\boldsymbol{D}$ [31].

Algorithm 1

for number of training iterations **do**
  for k step **do**
  - Sample minibatch of m noise sample $\{z^{(1)},\dots,z^{(m)}\}$ from noise prior $p_g(z)$. z is noticed as the joint space in the end-effector provided by random values. So, z defines as an input (p_x, p_y, p_z).
  - Sample minibatch of m examples $\{x^{(1)},\dots,x^{(m)}\}$ from data generating distribution $p_{data}(x)$. x is noticed as the joint space in end-effector provided by inverse kinematic equations. So, x defines as an input (p_x, p_y, p_z).
  - Update the Discriminator by ascending its stochastic gradient [20]:

$$\nabla_{\theta_d}\frac{1}{m}\sum_{i=1}^{m}\left[\log D\left(\boldsymbol{x}^{(i)}\right)+\log\left(1-D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right]$$

  end for
  - Sample minibatch of m noise samples $\{z^{(1)},\dots,z^{(m)}\}$ from noise $p_g(z)$. z denotes as an input (p_x, p_y, p_z).
- Update the generator D by descending its stochastic gradient [20]:

$$\nabla_{\theta_g}\frac{1}{m}\sum_{i=1}^{m}\log\left(1-D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$$

end for

The gradient-based updates can use any standard gradient-based learning rule.

Particle Swarm Optimization (PSO)

The global optimum solution of the swarm particle defines as initial fitness value denotes by x. The swarm consists of n-particles travelling into n-dimensional search space with the epoch t. During each epoch, p particle produces a unique position vector x. The const function also calculated by each particle p which consider as the local best fitness (p_k^{best}) to find the best fitness called global best fitness (g_k^{best}) of the swarm. The vectors and velocities both formulated:

$$\mathrm{v}_{k+1}^{t} = \mathrm{v}_{k}^{t} + c_p\left(p_k^{best} - x_k^{t}\right) + c_g\left(g_k^{best} - x_k^{t}\right) \quad (35)$$

$$x_{k+1}^{t} = x_{k}^{t} + \mathrm{v}_{k+1}^{t} \quad (36)$$

x_k^t and v_k^t are the position vector and velocity vector of epoch t. c_p and c_g in (27) shows the coefficient factor that adjusts the particles' weights [33][34]. v_k^t is velocity vector formulized the fitness of backpropagation process in Artificial Neural Networks (ANNs). PSO was successfully integrated with ANNs training process to optimize the architecture of the network. This Algorithm (2) shows that PSO determines the optimized values of the ANNs learning rate γ and the number of nodes j in ANNs hidden layer architecture [35][36].

Algorithm 2

Initialize x_k^t swarms, velocity vector (v_k^t), local (p_k^{best}) and global (g_k^{best}) best positions.
  for number of iterations t **do**
    for number of swarms x_k^t **do**
      calculate $G(z;\theta_g)$ (26) until (30)
      evaluate fitness in (29)
      update p_k^{best}
    end for
  update g_k^{best}
  update v_{k+1}^t
  update x_{k+1}^t
  end for
find best solution of g_k^{best}
validation $rRMSE$ in (30)

Proposed Method

Referring to (31), $W(G(z))$ denotes as an Objective Function that optimized by PSO (explained in Algorithm 2).

$$V(D,G) = E_{x\sim P_{data}(x)}[logD(x)] + E_{z\sim P_z(z)}[\log(1 - D(W(G(z)))] \quad (37)$$

The training objective of the generator $\boldsymbol{G}$ and the discriminator $\boldsymbol{D}$ expressed as a loss function obtained:

$$G_{loss} = E_{z\sim P_z}[\log \prod_{i=1}^{n} G(z)_i . D\big(W\big(G(z)\big)\big] \quad (38)$$

$$D_{loss} = -E_{x\sim P_{data}(x)}[logD(x)] - E_{z\sim P_z(z)}[\log(1 - D(W(G(z)))] \quad (39)$$

where n denotes the number of the trajectory points generated by the generator $\boldsymbol{G}$, $G(z)_i$ denotes the probability occur of the i-th trajectory point, and $\prod_{i=1}^{n} G(z)_i$ denotes the probability occur of the generator output.

Table 2. Hyperparameter for neural networks and PSO [26]

Parameters	Choices	Selected Parameters
Number of Layers	[3,4,5,6,7,8]	3
Number of Neurons in Hidden Layers	[2^n] where $n = 2, 3, \ldots, 10$	2^6
Activation Functions	['ReLU', 'Sigmoid', 'TanH', 'Leaky ReLu']	'Leaky ReLu'
Activation Functions Rate	[0.1. 0.2, …. 1.0]	0.1
Acceleration Coefficients (c_p; c_g)	[1.0, 1.001, …, 2]	(0.5;0.9)

Table 2 shows the hyperparameter selected in this approach. The number of layers denotes that the architecture of the generator $\boldsymbol{G}$ and the discriminator $\boldsymbol{D}$ both are ANNs.

RESULTS AND DISCUSSION

The proposed approach was applied to avoiding collision of Arm Robot three degree of freedom (3 DoF). The datasets generated by inverse kinematic equation explained in (16). It is generated 7200 motion points to produce the joint space (p_x, p_y, p_z) and joint angular (q_1, q_2, q_3).

Figure 5 shows that the data distribution of real data $D(x)$ were used 3600 samples and 7200 samples. The batch noise of data distribution $G(z)$ generated using the same size with the data distribution of real data $D(x)$.

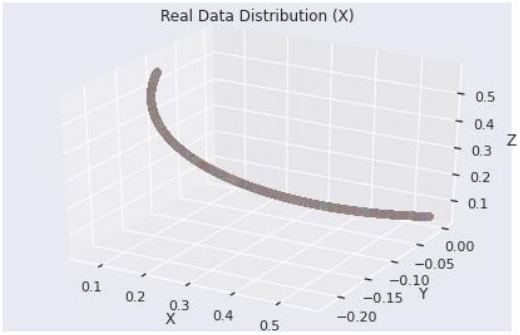

Figure 5. Data Distribution of Real Data $D(x)$

Training

The training process was performed using Intel(R) Core© i5-6300HQ CPU@2.30GHz, 12GB RAM and NVIDIA GeForce GTX 960M. The training process followed the hyperparameter [18] shown in Table 2. The training performance was captured in Figure 6, that means of our process time.

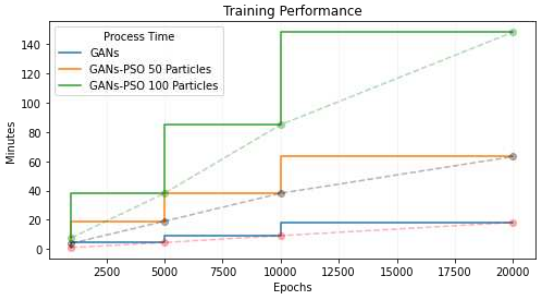

Figure 6. Training Generator G and Discriminator D Performance

GANs has 15.26 minutes for 20000 iterations. GANs-PSO with 50 particles has 63.17 minutes for 20000 training epochs. And, GANs-PSO with 100 particles has 149.78 minutes for 20000 training epochs. This Testing represents that our approach takes a lot of time than common GANs, because of generates 50 particles of $G(x)$ and 100 particles of $G(x)$ per data sample.

Figure 7 shows the data distribution of noise data $G(z)$ to match the data distribution of real data $D(x)$. A centroid distribution data shows the gap between real data distribution and noise data distribution in two cases before and after the generator $G(.)$ performed. Follows by the time process in Figure 6, the approaches tested in 1000 until 20000 training epochs.

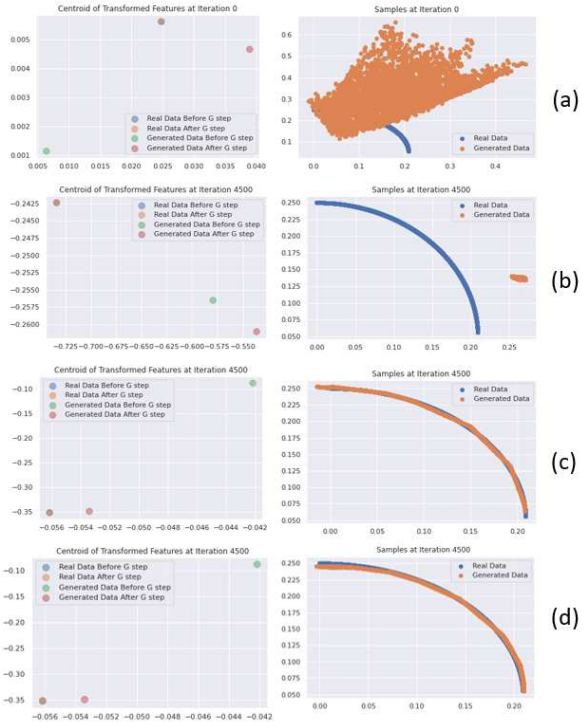

Figure 7. (a) Centroid of distribution data $G(z)$ and $D(x)$, (b) Centroid of GANs in 5000 training epochs, (c) Centroid of GANs-PSO 50 particles in 5000 training epochs, and (d) Centroid of GANs-PSO 100 particles in 5000 training epochs

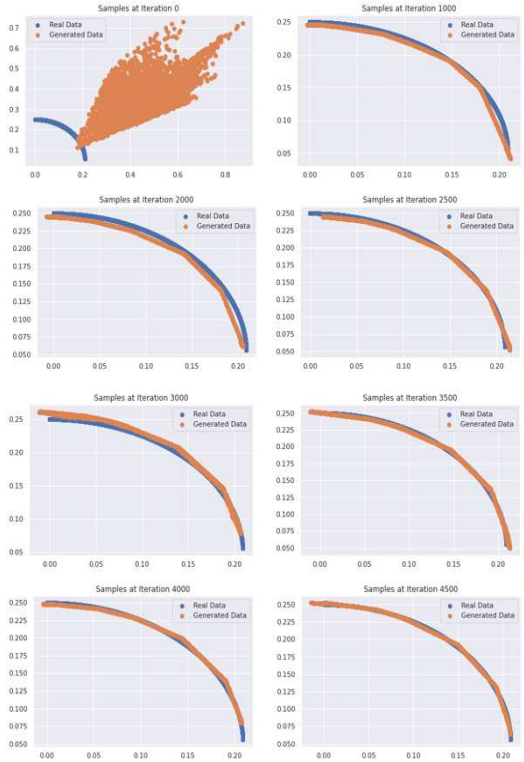

Figure 8. The Extraction of Figure 7, GANs-PSO 50 particles in 5000 training epochs

The centroid of data distribution $G(z)$ and $D(x)$ was successfully captured. GANs-PSO with 50 particles trained in 5000 training epochs was performed $G(z)$ to approximate the real data $D(x)$. The centroid generated real data $G(x)$ should be matched the centroid real data $D(x)$ shown in Figure 7.

The training process of GANs-PSO with 50 particles trained in 5000 training epochs was estimated 19 minutes. Figure 6 was transformed in 2D, representing the motion path of the generator result in Figure 8. The motion path was obtained by inputting various random numbers to G network as $G(z)$. The Arm Robot moved to a predefined position to caption the motion path, as shown in Figure 8.

The PSO has optimized the generator loss G_{Loss} by minimizing the cost function formulized in (38) and shown in Figure 9.

The Initial G Loss is calculated 1.215757084 after the end of the process (the num of the epoch multiplying with the num of the particles) G_{Loss} optimized is 1.32417093. Figure 9 also shows the minimax term GANs maximize generator $\boldsymbol{G}$ and minimize discriminator $\boldsymbol{D}$ (see (34)).

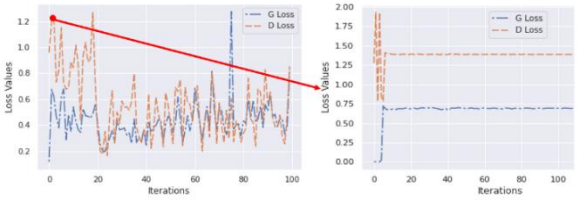

Figure 9. The Optimization of G_{Loss}

Figure 10 represents the generation of the data distribution of the generator $\boldsymbol{G}$ to reach the minimax expectation of $x{\sim}G; z{\sim}p_z$. It represents the four motions of the robot, and it takes 5000 iterations and 50 particles.

Figure 11 represents the generation of the data distribution of the generator $\boldsymbol{G}$ to reach the minimax expectation of $x{\sim}G; z{\sim}p_z$. Figure 11 also represents joint values in which contains the join value of theta 1, theta 2, and theta 3 and represents join velocities which contains J_{v1}, J_{v2}, and J_{v3}.

Testing Generator Network

Figure 12 shows the testing performance of our approach. The prediction of the proposed path tested in 1000 until 20000 training epochs.

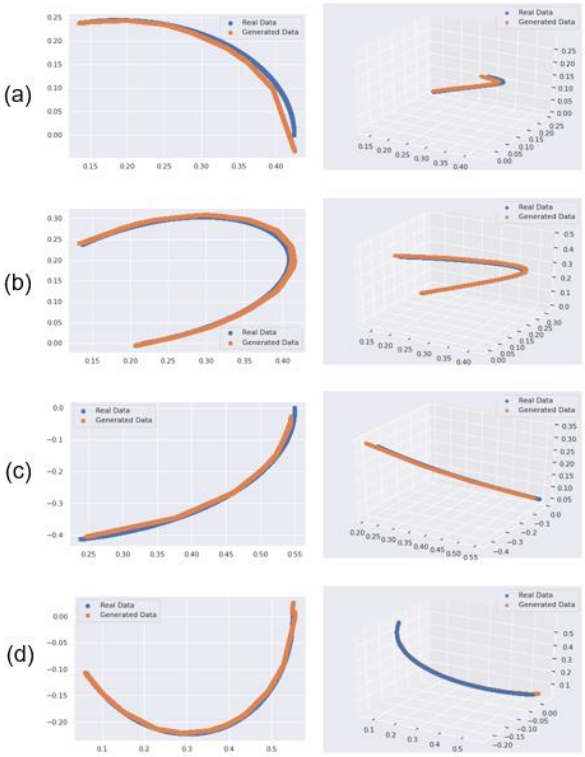

Figure 10. Minimax Expectation of $x \sim G$; $z \sim p_z$ (a) Motion-1, (b) Motion-2, (c) Motion-3, and (d) Motion-4

The generator G predict with the minimum value reached 0.0270 milliseconds, and the maximum value reached 0.0284 milliseconds. The processing load decreased if compared to the training load.

Each training process includes 1000 until 20000 training epochs, and each training epoch covers all the training dataset. At the end of each training epoch, the validation dataset is used to evaluate the well-trained neural network's Performance on the unseen dataset selected for training, which is relatively smaller than the test dataset. To test the overall Performance and the generalization of the well-trained neural network using the limited dataset, the test dataset was used to estimate its overall Performance over the whole actuation space.

Figure 13 show the Performance of GANs-PSO in predicting the proposed position and the predictive position. To see the comparison between the proposed method (GANs-PSO) and the conventional method (GANs) shown in Table 3. Our satisfied iterations selected in 5000 epochs comparing GANs and GANs-PSO the time estimation of the training process was counted in Table 3. Our proposed method GANs-PSO calculated the G_{RMSE} was tested 0.027475%.

Table 3. Performance of the proposed method

Training		
Method	**Time**	**Epoch**
GANs	6.43 minutes	5000
GANs-PSO	19.17 minutes	5000
Testing		
Method	G_{RMSE}	
GANs	6.402591%	
GANs-PSO	0.027475%	

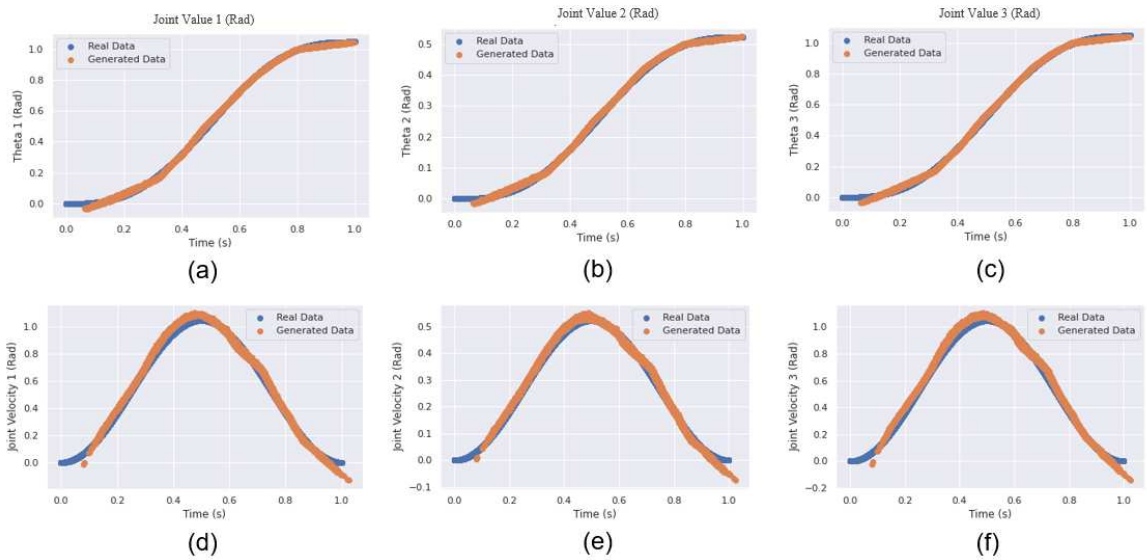

Figure 11. Joint Values: (a) Joint-1, (b) Joint-2, and (c) Joint-3, Joint Velocity: (d) Joint-1, (e) Joint-2, and (f) Joint-3,

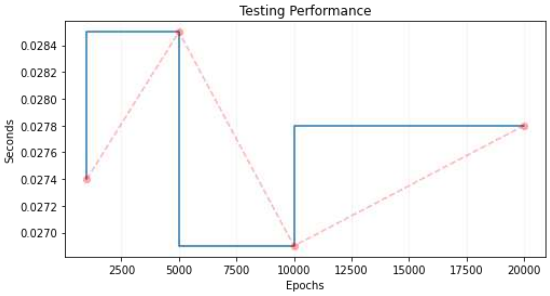

Figure 12. Testing Generator Performance

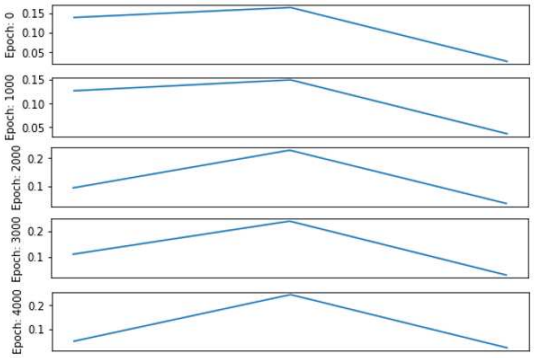

Figure 13. The GMSE Tested

CONCLUSION

This paper presented a hybrid method of Generative Adversarial Networks (GANs) and Particle Swarm Optimization (PSO) for Arm Robot three degree of freedom (3 DoF). The real data distribution $D(x)$ was generated by Inverse Kinematic equation to produce the joint space (p_x, p_y, p_z) and joint angular (q_1, q_2, q_3). The noise data distribution $G(z)$ was presented as the joint space $G(p_x, p_y, p_z)$. The PSO optimized the Performance of the generator G, in the case to avoid the occurrence of the mode collapse. Mode collapse occurs when the generator is unable to generate output as data distribution $x \sim G; z \sim p_z$. Therefore, PSO reduced the number of iterations differentially from conventional. The selected approach was GANs-PSO with 50 particles in 5000 training epochs, the training process of each proposed method takes around 19.17 minutes to train the whole 7200 datasets. The neural generator network's execution time takes around 0.028ms to perform a single prediction with the GMSE revealed 0.027475%.

REFERENCES

[1] T. Inan and A. F. Baba, "Particle Swarm Oprimization based Collision Avoidance," *Turkish Journal of Electical Engineering & Computer Science*, vol. 27, no. 3, pp. 2137-2155, 2019, DOI: 10.3906/elk-1808-63

[2] A. Adriansyah, Y. Gunardi, and E. Ihsanto, "Goal-seeking Behaviour-based Mobile Robot using Particle Swarm Fuzzy Controller," *TELKOMNIKA*, vol. 13, no. (2), pp. 528-538, 2015, DOI: 10.129281/ telkomnika.v13i2.1111

[3] A. Adriansyah and S. H. M. Amin, "Wall following Behavior-based Mobile Robot using Particle Swarm Fuzzy Controller," *Jurnal Ilmu Komuter dan Informasi (JIKI)*, vol. 9, no. 1, pp. 9-16, 2016, DOI: 10.21609/jiki.v9i1.367

[4] P. Bosscher and D. Hedman, "Real-time collision avoidance algorithm for robotic manipulators," *International Conference on Technologies for Practical Robot Applications*, Woburn, MA, 2009, pp. 113-122, DOI: 10.1109/TEPRA.2009. 5339635

[5] L. Zhao, J. Zhao, H. Liu and D. Manocha, "Efficient Inverse Kinematics for Redundant Manipulators with Collision Avoidance in Dynamic Scenes*," *International Conference on Robotics and Biomimetics (ROBIO)*, Kuala Lumpur, Malaysia, 2018, pp. 2502-2507, DOI: 10.1109/ROBIO.2018. 8664838

[6] T. Kivela, et al., "Redundant Robotic Manipulator Path Planning for Real-Time Obstacle and Self-Collision Avoidance," *Advances in Service and Industrial Robotics, Mechanisms and Machine Science,* vol. 49, 2018, DOI 10.1007/978-3-319-61276-8-24

[7] C. Park, "Self-Collision Detection & Avoidance Algorithm for a Robot Manipulator," *International Journal of Engineering and Innovative Technology (IJEIT)*, vol. 5, no. 4, pp. 139-142, 2015

[8] C. Yang, X. Wang, L. Cheng and H. Ma, "Neural-Learning-Based Telerobot Control with Guaranteed Performance," in IEEE Transactions on Cybernetics, vol. 47, no. 10, pp. 3148-3159, Oct. 2017, DOI: 10.1109/ TCYB.2016.2573837

[9] B. Durmus, H. Termutas and A. Gun, "An Inverse Kinematics Solution using Particle Swarm Optimization," *6th International Advanced Technologies Symposium (IATS'11)*, 2011

[10] N. Rokbani, and A. M Alimi, "Inverse Kinematics using Particle Swarm Optimization, A Statistical Analysis," *International Conference on Design and Manufacturing (IConDM)*, 2013, vol. 64, pp. 1602-1611

[11] N. Rokbani, A. Casals and A .M. Alimi, "IK-FA a New Heuristic Inverse Kinematics Solver using Firefly Algorithm," *Computational Intelligence Applications in Modeling and Control. Studies in Computational Intelligence*, vol 575. Springer, Cham, 2015

[12] M. Ayyildiz & K. Cetinkaya," Comparison of Four Different Heuristic Optimization

Algorithms for The Inverse Kinematics Solution of a Real 4-DoF Serial Robot Manipulator," *Neural Comput & Applic,* vol. 27, pp**.** 825–836, 2016
[13] T. Collins & W.M Shen, "PASO: An Integrated, Scalable PSO-based Optimization Framework for Hyper-Redundant Manipulator Path Planning and Inverse Kinematics," *ISI Tech Report,* 2016
[14] P. Srisuk, A. Sento and Y. Kitjaidure, "Inverse kinematics solution using neural networks from forward kinematics equations," *International Conference on Knowledge and Smart Technology,* Chonburi, 2017, pp. 61-65, DOI: 10.1109/KST.2017.7886084
[15] R. Villegas, et al. Neural Kinematic Networks for Unsupervised Motion Retargetting: IEEE, 2018
[16] A. R. J Almusawi., et al., "A New Artificial Neural Network Approach In Solving Inverse Kinematics of Robotic Arm (Denso VP242)," *Computational Intelligence and Neuroscience*, vol. 2016, ID: 5720163, 2016
[17] M. Schilling, "Setup of a Recurrent Neural Network as a Body Model for Solving Inverse and Forward Kinematics as well as Dynamics for a Redundant Manipulator," *Cluster of Excellence Cognitive Interaction Technology (CITEC)*, April 2019
[18] H. Ren & P. Ben-Tzvi, "Learning Inverse Kinematics and Dynamics of a Robotic Manipulator using Generative Adversarial Networks," *Robotics and Autonomous Systems*, vol. 124, ID: 103386, Feb 2019
[19] T. G. Thuruthel et al., "Model-Based Reinforcement Leaning for Closed-Loop Dynamic Control of Soft Robotic Manipulator: *IEEE Transactions on Robotics*, vol. 35, no. 1, pp. 124-134, Feb. 2019
[20] I. J. Goodfellow et al., "Generative Adversarial Nets," *arXiv*: 1406.2661, 2014
[21] I. J. Goodfellow, "NIPS 2016 Tutorial Generative Adversarial Networks, " *arXiv:* 1701.00160, 2017
[22] C. Ledig et al, "Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, 2017, pp. 105-114
[23] A. Radford et al., "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks," *arXiv*: 1511.06434, 2016
[24] L. Qingsheng and J. Andika," Analysis of Kinematic for Legs of a Hexapod using Denavit-Hartenberg Convention," *SINERGI*, vol. 22, no. 2, pp. 69-76, 2018. DOI: 10.22441/sinergi.2018.2.001
[25] O. Hock and J. Sedo, "Forward and Inverse Kinematics using Pseudoinverse and Transposition Method for Robotic Arm DOBOT," *Kinematics*, 2017
[26] F. Chao et al., "Generative Adversarial Nets in Robotic Chinese Calligraphy," *International Conference on Robotics and Automation (ICRA)*, Brisbane, QLD, 2018, pp. 1104-1110, DOI: 10.1109/ICRA.2018.8460787
[27] Z. Yi, H. Zhang, P. Tan and M. Gong, "DualGAN: Unsupervised Dual Learning for Image-to-Image Translation," *International Conference on Computer Vision (ICCV)*, Venice, 2017, pp. 2868-2876
[28] M. Alzantot, S. Chakraborty and M. Srivastava, "SenseGen: A deep learning architecture for synthetic sensor data generation," *International Conference on Pervasive Computing and Communications Workshops*, Kona, HI, 2017, pp. 188-193, DOI: 10.1109/PERCOMW.2017.7917555
[29] F. Yang and C. Peters, "AppGAN: Generative Adversarial Networks for Generating Robot Approach Behaviors into Small Groups of People," *International Conference on Robot and Human Interactive Communication (RO-MAN)*, New Delhi, India, 2019, pp. 1-8, DOI: 10.1109/RO-MAN46459.2019.8956425
[30] A. Lee, "Visual Dynamics Models for Robotic Planning and Control," *EECS*, Berkeley, 2019
[31] X. Qian et al.,"Hardness Recognition of Robotic Forearm Based on Semi-supervised Generative Adversarial Networks," *Frontiers in Neurorobotics*, September 2019
[32] H. T. Rauf et al., "Training of Artificial Neural Network Using PSO With Novel Initialization Technique," *International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)*, Sakhier, Bahrain, 2018, pp. 1-8
[33] B. A. Garro and R. A. Vazquez, "Designing Artificial Neural Networks using Particle Swarm Optimization Algorithms," *Computational Intelligence and Neuroscience*, ID: 369298, 2015, DOI: 10.1155/2015/369298
[34] M. Shariati et al.,"Application of a Hybrid Artificial Neural Network- Particle Swarm Optimization (ANN-PSO) Model in Behavior Prediction of Channel Shear Connectors Embedded in Normal and High-Strength Concrete," *Applied Sciences*, vol. 9, no. 24, pp. 5534, 2019, DOI: 10.3390/app9245534