Automatic navigation of wall-following mobile robot using a hybrid metaheuristic assisted neural network

Tirtharaj Dash\textsuperscript{a,*}, Rakesh Ranjan Swain\textsuperscript{b} and Tanistha Nayak\textsuperscript{c}

\textsuperscript{a} Data Science Research Group, Department of Computer Science and Information Systems, Birla Institute of Technology and Science Pilani, Goa Campus, Goa 403726, India
E-mail: tirtharaj@goa.bits-pilani.ac.in; ORCID: http://orcid.org/0000-0001-5965-8286
\textsuperscript{b} Parallel and Distributed Computing Lab, Department of Computer Science and Engineering, National Institute of Technology Rourkela, Odisha 769008, India
E-mails: rakeshswain89@gmail.com,
\textsuperscript{c} School of Information & Computer Sciences, Ravenshaw University Cuttack, Odisha 753003, India
E-mails: tanishthanayak@hotmail.com,

Abstract. This work proposes an attempt to control the navigation of a wall following mobile robot using a hybrid metaheuristic which is a combination of two different population-based heuristics such as Gravitational Search (GS) and Particle Swarm Optimization (PSO). The hybrid metaheuristic is called GSPSO which is used to train a multi-layered Artificial Neural Network (ANN) with the available benchmark sensor readings datasets obtained from the navigation of a wall following mobile robot SCITOS G5. The performance of the proposed model was evaluated with regard to different tuning parameters present in the metaheuristic as well as in the ANN. Simulated experimental results show satisfactory performance which makes the proposed method adoptable in practice for controlling a mobile robot for predicting its next direction based on sensed distance information from the wall or obstacles surrounding it.

Keywords: Robot navigation, Intelligent control, Artificial Neural Network, Particle Swarm Optimization, Wall-following robot, Gravitational Search Algorithm, Metaheuristics

1. Introduction

Mobile robots are widely being applied in human daily life with technological advancements and developments of enhanced computing technologies. The deployed service and surveillance mobile robots essentially have to cooperate with humans for performing a few tasks. As the name suggests, the robots are not static, rather they are mobile in nature and hence they have certain locomotion or movements for performing tasks. However, there are mainly two challenges in this field. First, the movements of these mobile robots become extremely difficult when the environment is full with obstacles such as walls, desks, doors, other fellow robots etc. Second, the decision related to direction and further movement has to be done in real time. Therefore, effective control of the locomotion of mobile robots, also called as ‘Robot Navigation’ along with obstacle avoidance is one of the very challenging and important tasks in

\*Corresponding author. E-mail: tirtharaj@goa.bits-pilani.ac.in.
many different intelligent industrial applications [1, 2]. A detailed description of the robot navigation problem is provided in the methodology section for a clear understanding of the work.

While navigating in an environment, the robot uses sensors to calculate the distance from obstacles or objects to its body. The sensors may be of different types such as infrared sensors, laser range finder, ultrasonic sensors or visible-light camera etc. Based on requirements and feasibility one should choose the appropriate type of sensors for the navigation tasks. During the navigation, mobile robots keep a safe distance from the obstacles or walls of the obstacle and these type of robots are called ‘wall following robots’ or ‘wall follower’ [3].

Many different classical models were proposed for controlling mobile robot navigation. Some recent studies include statistical modelling [4] and the use of Monte-Carlo sampling based visual perception [5]. However, it has been observed that in most of the cases, building an exact dynamical model is very difficult when information on the possible future environment is not known apriori. Furthermore, the developed robotics systems have no fixed dynamics for control. This makes the process more challenging in real time execution of industrial or household tasks. This type of problem could be observed as an optimization problem where learning real-time sensor readings are the principal objective based on which future decision could be made. To solve such complex optimization problems, metaheuristics could find their application in many different engineering disciplines. Furthermore, metaheuristics are problem-independent techniques. In other words, they do not consider much on the particular of the supplied problem and hence they could find their usefulness in many such application domains as black boxes. Often, the metaheuristics try their best to explore the solution space to give an optimum for the supplied objective function with regard to any problem. It should also be noted that some parameters related to any metaheuristic, however, require some fine tuning before they could be chosen as the best solution finder in the search space of any engineering or science problems.

The overall organization of this paper is as follows. section 1 introduced the problem statement and motivation behind the present research on automatic navigation of wall-following mobile robot. We give a brief survey of the current state of the art in robot navigation in section 2. Our proposed methodology is discussed in section 3. The evaluated and obtained results are presented and discussed in section 4. We conclude the work in section 5 along with possible future scope for extension of the current work.

2. Present state of the art

Nature inspired metaheuristics such as Genetic Algorithm (GA) [6], Particle Swarm Optimization (PSO) [7], Ant Colony Optimization (ACO) [8], Gravitational Search (GS) [9] and its variants such as Gradient Gravitational Search (GGS) [10] have been applied to solve complex optimization problems in the field of engineering as well as protein modeling. However, our focus would be on the application of metaheuristics for effective controlling of robot navigation tasks in upcoming portions of this paper.

The principal reason behind popularity and applicability of various metaheuristics is that they constitute a set of good solutions to maintain the dynamics used in the underlying control algorithm which closely mimics real-time dynamical systems. These algorithms have the capability to exploit and explore the global minimization of a supplied objective function. However, one can not always achieve a global minimum for all types of objective functions. Such example could be seeking global optimum (either maximum or minimum) in Neural Network (NN) learning which in turn has been applied to the problem of robot navigation. In this type of scenario, the goal is to reach a local optimum state which is closest to the global optimum. For better understanding, we should consider the convergence of any Artificial
Neural Network (ANN) during the process of training where the training error tries to get lowered in every iterative step in training.

In few recent studies, various hybrid variants of ACO could be applied to solve the problem of robot navigation. Hsu and Juang have implemented an interval type-2 fuzzy controller (IT2FC) with a modified continuous ACO, called SDE-CACO for wall following control of mobile robots. In their work, the speed and position of the robot could be controlled whose intermediate stage is minimizing a cost function of the wall following performance [11]. In another work, ACO was hybridized with GA for solving multi-objective mobile robot path planning problem [12]. In this work, the researchers could find a collision-free path for the mobile robot which closely resembles with wall following robot navigation problems [13].

The commonly used swarm based optimization algorithm PSO was used by Chen et al. [14] for solving wall following navigation of SIAT mobile robot. They measured the performance of their proposed navigation algorithm using classification accuracy. In another study, Dash et al. [15] proposed a different swarm based approach called GS for controlling wall following navigation of SCITOS G5 robot. In this work, a feed-forward neural network (FFNN) has been trained with GS algorithm and the generated model GS-FFNN was used to predict the next direction of the mobile robot in a closed room environment. They also used the performance measures accuracy and decision taking time for evaluating the developed model.

A biologically inspired metaheuristic called Artificial Immune System (AIS) algorithm was employed by Huang et al. [16] for mobile robot navigation in a structured environment with obstacles. They also incorporated B-spline technique to construct the optimal collision-free continuous path. However, this work might not be applied to robot navigation in an unknown clustered environment. This type of problem was solved by Mohanty and Parhi in their work [17] where they incorporated Cuckoo Search (CS) with Adaptive Neuro-Fuzzy Inference System (ANFIS). Furthermore, this work could solve the problem of multiple robots navigation in highly clutter terrains [18]. Many different metaheuristics have been used to solve robot navigation problem. A review of different metaheuristics approach in robotics could be obtained in [19].

Aforementioned approaches have been well appreciated and used in different contexts of problems and have been widely used in solving problems in various engineering and science domains. Motivated by the popularity and success of metaheuristics based learning models for solving real world challenging problems, we attempt to solve the popular wall following robot navigation problem in this work. In this work, we use a hybrid combination of two population-based metaheuristics such as Gravitational Search (GS) and Particle Swarm Optimization (PSO) and the resulting hybrid metaheuristic is called as GSPSO. The GSPSO algorithm is used to train an ANN with Input-Hidden-Output layer architecture, henceforth the developed model can be termed as GSPSO-ANN. The next section gives the details on the problem statement and our approach to solving the problem of robot navigation using GSPSO-ANN.

3. Methodology

In this section, we first explain the problem statement and subsequent subsections give the details on our approach to solving the problem.

Problem statement: Given an environment surrounded by walls and few obstacles inside, a robot has to decide on the next direction based on sensed distance information. The distances are basically the distance of the body of the robot from the wall so that it will avoid any collision with obstacle or wall
when it moves inside the environment. The problem is challenging considering the fact that the robot has to sense the distance from the wall to itself continuously as it moves and takes direction. This problem is stochastic in nature and makes future predictions or decisions difficult. Therefore, a deterministic or statistical approach would not solve the problem even though it worked with some historical data.

In this work, we train a widely used feed-forward ANN model with GSPSO metaheuristic for navigation of the wall following the mobile robot. The whole process in this work can be divided into following phases each of which is briefly explained in subsections for better understanding.

- Collecting data and preprocessing
- Designing ANN based on preprocessed data
- Training ANN with GSPSO metaheuristic
- Testing of trained ANN

3.1. Collecting data and preprocessing

As this work is simulation-based work, collection and preparation of data before processing is of importance so that it could emulate the real-time behavior when implemented in reality. The data is nothing but a collection of raw sensor readings containing numeric distance values from the SCITOS G5 robot to its surrounding walls. The sensor arrangement on the body of the robot has been shown in the Figure 1. Preprocessing refers to the process of conversion of non-numeric values to numeric values so that it could be used for the numeric computation. Further details on this phase are given in the section 4.

3.2. Designing ANN based on preprocessed data

In this work, we implement a multi-layered ANN with architecture Input–Hidden–Output layers. The number of inputs, denoted as $N$, in the ANN is equal to the number of feature attributes in the preprocessed input data. In this work, we use a single hidden layer with $M$ number of neurons in it. The number of neurons in the output layer, called predicting neuron is set to one which will give the next direction of the wall-following robot based on the presented sensor readings. Hence, the designed architecture of the ANN becomes $N : M : 1$. Let $V$ represents the set of the inter-layer weights of the ANN i.e. from synaptic connections from the input layer to the hidden layer; and $W$ represents the hidden layer to output layer weights i.e. synaptic connections from the hidden layer to the output layer. Hence, as per designed architecture, there are $N \times M$ number of weights in input to the hidden layer. Similarly, there is a single bias (denoted as $b_o$) in the output layer and $M$ biases (denoted as a set $b_h$) in the hidden layer. Figure 2 shows the overall architecture of the designed GSPSO-ANN.
3.3. Training ANN with GSPSO metaheuristic

In this subsection, we first discuss the forward processing of input sensor data by the ANN and then we discuss the GSPSO metaheuristic algorithm in details.

3.3.1. Forward computations in ANN:

As mentioned in the former subsection, the ANN has an Input–Hidden–Output layer architecture, we give a brief and general explanation of processing by the ANN for predicting the next direction of the wall following the mobile robot.

Consider a data instance \( x \) from training dataset which has \( N \) number of attributes. So, \( x \) can be represented as \( (x_1, x_2, \ldots, x_N) \). Consider that there are \( M \) number of neurons in the hidden layer. Lets denote the input–hidden layer weights as a set \( V = \{v_{ij} : 1 \leq i \leq N, 1 \leq j \leq M\} \), bias in the hidden layer as \( b_h = \{b_{hi} : 1 \leq i \leq M\} \). Similarly, lets denote the hidden layer–output layer weights as a set \( W = \{w_{j} : 1 \leq j \leq M\} \) and bias in the output layer as \( b_o \). In this work we use the sigmoid neurons in the hidden layer. Which means the fired strength from a neuron in hidden layer could be found out using Equation 1.

\[
f_{\text{sig}}(x) = \frac{1}{1 + e^{-x}} \tag{1}
\]

The input to \( j \)th neuron in the hidden layer is denoted as \( Z_{inj} \). The processed output from this \( j \)th neuron of the hidden layer is denoted as \( Z_j \) can be obtained using Equation 2.

\[
Z_j = f_{\text{sig}} \left( \sum_{i=1}^{N} (x_i \cdot v_{ij}) + b_{hi} \right) \tag{2}
\]
We do not apply any firing activation function in the output and hence the processed output at the output neuron can be obtained using Equation 3.

\[ Y = y_n = \sum_{j=1}^{m} (Z_j \ast w_j) + b_o \]  (3)

It should be noted that we used batch learning (epoch based) where one epoch refers to the processing of all the instances present in the training sensor dataset once for getting a cumulative error called mean-squared-error \((MS E)\). \(MS E\) can be calculated using the Equation 4. In the equation, \(n\) is the total number of patterns in the training dataset, \(Y_i\) and \(T_i\) are the output from the ANN and corresponding target respectively for pattern instance \(i\) in the dataset. Hence the Error in the Figure 2 refers to the \(MS E\).

\[ MS E = \frac{1}{n} \sum_{i=1}^{n} (Y_i - T_i)^2 \]  (4)

3.3.2. GSPSO algorithm for training the designed ANN:

The GSPSO metaheuristic based ANN, ‘GSPSO-ANN’ optimizes the neural network training based on the initial weights and biases of the ANN to obtain an optimal set of weights that is suitable for the ANN to learn the input-output transformation function available in the training data. Hence, the inputs to these algorithms will be weight sets such as \(V\), \(W\), \(b_h\), and \(b_o\) of the ANN. For clarity, it should be noted that optimizing the training of ANN refers to the minimizing the error computed during the training phase. Therefore, the considered metaheuristic GSPSO tries to minimize the \(MS E\) which has already been given in Equation 4 which in turn uses the weights and biases of the ANN i.e. \(V\), \(W\), \(b_h\), and \(b_o\) (refer Equation 2 and Equation 3). Here we describe the GSPSO algorithm and its application to ANN in detail. In this work, we employ the methodology adopted in [20–22] to solve the automatic mobile robot navigation problem.

The GSPSO metaheuristic is developed as a combination of two population-based metaheuristics called GS and PSO. The GS metaheuristic has a strong local search capability which exploits the search space [9] and the PSO is well known for its strength in social movement behavior which makes it a well-suited explorer of the search space [7]. It is believed that when exploitation and exploration could be combined together, it would result in better hybrid metaheuristic which would be having a strong potential of searching the high dimensional search space and optimize the objective function (i.e. \(MS E\) in this work).

The steps involved in the GSPSO algorithm are a combination of steps involved in GS algorithm and the steps involved in updating velocity and position in PSO algorithm. This, in some sense, guarantee that the GSPSO algorithm is superior than both GS and PSO by incorporating the exploration and exploitation capability during the search for the optimal parameters [22]. These steps are presented as an algorithm in Algorithm (1) [22].

Mathematically, the GSPSO system could be considered as an isolated system of agents similar to GS algorithm which is like a small artificial world of masses obeying the Newtonian laws of gravitation and motion. More precisely, masses or agents obey the two fundamental scientific laws: Law of gravity and Law of motion [9].
Algorithm 1 Steps involved in GSPSO algorithm:

1: \textbf{procedure} GSPSO–Algorithm
2: \hspace{1em} Initialize all agents with their positions, masses, accelerations
3: \hspace{1em} \textbf{while} Stopping criteria not satisfied \textbf{do}
4: \hspace{2em} Calculate gravitational force acting on each agent due to mutual attraction with neighbor agents
5: \hspace{2em} Calculate acceleration of each particle due to force of attraction
6: \hspace{2em} Update the best solution with new position of the agents (called candidate solutions)
7: \hspace{1em} \textbf{end while}
8: \textbf{end procedure}

Now, consider a system with \( N \) agents. We define the position of the \( ith \) agent by Equation 5

\[
X_i = (x^1_i, x^2_i, \ldots, x^d_i, \ldots, x^n_i); \quad 1 \leq i \leq N
\]  

where, \( x^d_i \) represents the position of \( ith \) agent in the \( dth \) dimensional search space.

At a time \( t \), the gravitational force acting on mass \( i \) from mass \( j \) can be defined as follows.

\[
F^d_{ij} = G(t) \frac{M_{pi}(t) \times M_{aj}(t)}{R_{ij}(t)} (x^d_j(t) - x^d_i(t)) + \epsilon
\]

In the Equation 6, \( G(t) \) is the gravitational constant at a time step \( t \), \( \epsilon \) is a small constant, \( M_{pi} \) is the passive gravitational mass related to agent \( i \), \( M_{aj} \) is the active gravitational mass related to agent \( j \). The value of \( G \) as a function of time and the masses of agents as a function of time can be obtained by using the following equations.

The gravitational constant, \( G \) can be represented as, \( G(t) \) where \( G_0 \) is the initial gravitational constant and \( T \) is the maximum iteration.

\[
G(t) = G_0 \times e^{-\frac{\eta \times t}{T}}
\]

While solving the global minimization problem during the training of the ANN by essentially minimizing the error function, fitness value \( fit_i \) is computed for all feasible agent positions in \( X \) where \( i = 1, 2, \ldots, N \) and \( N \) is the number of agents in the \( d \)-dimensional search space. The fitness value is used to find masses of the agents using following equations, where \( best(t) \) and \( worst(t) \) represents the minimum and maximum fitness values. In this work, it is assumed that gravitational mass and inertial mass are equal and hence the following equality condition is considered and further computations have been made.

\[
M_{ai} = M_{pi} = M_i; i = 1, 2, \ldots, N
\]

where, \( M_i \) represents the inertial mass of an agent \( i \) and \( M_i \) represents the gravitational mass of an agent \( i \).

\[
m_i(t) = \frac{fit_i(t) - worst(t)}{best(t) - worst(t)}
\]
\[ M_i(t) = \frac{m_i(t)}{\sum_{j=1}^{N} m_j(t)} \] (10)

In the above equations, \( fit_i(t) \) represents the fitness value of the agent \( i \) at time \( t \). For a global minimization problem, the \( \text{best}(t) \) and the \( \text{worst}(t) \) can be mathematically represented as,

\[ \text{best}(t) = \min\{ fit_j(t) \}; \forall j \in \{1, 2, \ldots N\} \] (11)

\[ \text{worst}(t) = \max\{ fit_j(t) \}; \forall j \in \{1, 2, \ldots N\} \] (12)

\( R_{ij} \) is the Euclidean distance between two entities \( i \) and \( j \) and is defined as given following equations.

\[ R_{ij}(t) = \| X_i(t), X_j(t) \|_2 \] (13)

\[ R_{ij}(t) = \left( \sum_{p=1}^{n} (X^p_i(t) - X^p_j(t))^2 \right)^{\frac{1}{2}} \] (14)

To give a stochastic characteristic to the GSPSO algorithm, it is considered that the total force that acts on agent \( i \) in a dimension \( d \) be a randomly weighted sum of \( d \)th components of the forces exerted from other agents:

\[ F^d_i(t) = \sum_{j=1, j\neq i}^{N} \alpha_j F^d_{ij}(t) \] (15)

where, \( \alpha_i \) is a small random number in the range \([0, 1]\) to give a randomized characteristic to the search. The new acceleration of each agent \( i \) at a time \( t \) can be obtained by using this field \( F_i \) and mass of an agent.

\[ a^d_i(t) = \frac{F^d_i(t)}{M_i(t)} \] (16)

Velocity and position of an agent \( i \) are calculated using the following equations respectively; where \( w \) is a weighting function, \( V_i(t) \) is the velocity of agent \( i \) at time \( t \), \( c_1' \) and \( c_2' \) are the acceleration coefficients, \( g\text{Best} \) is the best fitness so far, \( X_i(t) \) is the position of agent \( i \) at time \( t \), \( a_i \) is the acceleration of agent \( i \), \( \theta_1 \) and \( \theta_2 \) are random numbers in the range \([0,1]\).

\[ \chi_1 = (c_1' \times \theta_1 \times a_i \times c_i(t)) \] (17)
\[ \chi_2 = (c'_{2} \times \theta_2 \times (g_{Best} - X_i(t))) \]  

(18)

\[ V_i(t) = (w \times V_i(t)) + \chi_1 + \chi_2 \]  

(19)

The above-described GSPSO algorithm is used to train ANN by considering its weights \( V, W, b_h \) and \( b_o \) as input parameters or agents which are to be tuned. The training algorithm based on the GSPSO metaheuristic has been given in Algorithm (2) [22]. It should be noted that the \( MSE \) is the value of the fitness function which is essentially the cumulative error during training of the ANN.

Time complexity of the proposed algorithm: As the proposed ANN models are three layered architecture i.e. (Input–Hidden–Output), the computation will be carried out in two different passes; (i) Pass 1: the forward computation from input layer to the hidden layer will take \( \Theta(NM) \) time (\( \Theta \) denotes tight bound), (ii) Pass 2: the forward computation from the hidden layer to the output layer which takes \( \Theta(M) \) times. So total is \( T(N,M) = \Theta(NM) + \Theta(M) = \Omega(NM) \) (\( \Omega \) denotes lower bound). For the proposed GSPSO-ANN, \( \Omega(NM) \) computations will be carried out for each agent in the system. Hence, the overall algorithmic time complexity of the proposed models is \( \Omega(NM) \) multiplied with population size \( N_{pop} \). Therefore, the algorithmic time complexity of the model is \( \Omega(N_{pop} \times NM) \).

3.4. Testing of trained ANN

After the ANN is adequately trained with the training dataset, the trained ANN must be subject to testing for each instance available in the test dataset to evaluate the quality of prediction. In this testing phase, the predicted output would be checked with the closest match with any of the target navigation direction of the robot. Based on this output class, selective action could be taken for the currently tested instance. During the prediction or testing phase, a threshold was selected based on which a wise decision could be taken on the processed input instance. Further experimental details are provided in section 4.

4. Results and discussion

To evaluate the performances of GSPSO-ANN for navigation of wall following mobile robot, a set of experiments on different well-known benchmark sensor readings datasets were conducted. As per description on the datasets, it should be noted that the deployed robot takes a decision on its next direction by sensing the distances from the walls around it. Hence in this work, we evaluate the performance of our adopted GSPSO-ANN model based on two different parameters such as decision taking accuracy and time required for the model to be trained and tested including cross-validation. The proposed navigation algorithm was implemented in MATLAB R2010a on a Windows computer system with 8 processors (Intel Xeon with clock rate 3.50 GHz) and 16 GB main memory. The following subsections give the details of sensor data and the further simulation results.
Algorithm 2 Training of the model with sensor readings

1: procedure GSPSOASSISTEDANNFORROBOTNAVIGATION
2: Input the training robot sensor readings dataset
3: Preprocessing the input dataset (checking missing values and non-numeric values, if any)
4: Initialize ANN parameters based on input data matrix: \( N, M \), weights \((V, W, b_h\) and \(b_o)\)
5: Initialize number of populations of agents \((N_{pop})\)
6: Initialize training parameters: Max_Iteration, force \(F\), masses of agents: \(m, G_0\), initial position of agents: \(X_0\), inertia weights, \(c'_1, c'_2, d\) etc.
7: Initialize gBest, gBestScore to large value
8: while iteration \((t) \leq \text{Max}_\text{Iteration}\) do
9:     for each agents do
10:         Set weights of ANN \((V, W\) and biases) using position of agents
11:         for each epoch do
12:             Compute error \((E)\) of current agents
13:             \(Sum = Sum + E\)
14:         end for
15:         \(MSE = \frac{Sum}{NoP}\)
16:         Current Fitness of agent \(i\): \(CF(i) = MSE\)
17:         if gBestScore \(> CF(i)\) then
18:             gBestScore = CF\((i)\)
19:             gBest = \(X(i, :)\)
20:         end if
21:     end for
22:     for each agents do
23:         Update mass \(m\)
24:         Calculate force \(F\) exerted from other agents
25:         Calculate acceleration \(a\)
26:         Update Velocity \(V\)
27:         Update new position of agents \(X\)
28:     end for
29: end while
30: end procedure

4.1. Sensor datasets description

In this work we use the experimental sensor datasets used by Freire et al. [23] for wall following robot navigation. The datasets are available in the UCI Machine Learning Repository [24]. The datasets are a collection of sensor readings obtained by the mobile robot SCITOS G5 during its movement inside a room. All the sensor data were collected as the mobile robot SCITOS G5 was deployed to navigate in a room following the wall around it in a clockwise direction for 4 rounds. There were 24 ultrasound (US) sensors arranged circularly around its waist with an arc distance of 15 degrees. The conceptual numbering of the US sensors starts at the front of the robot and increases in a clockwise direction. There are three different sensor readings datasets namely: sensor_readings_24.data, sensor_readings_4.data and sensor_readings_2.data. The first file contains the raw values of the measurements of all 24 US sensors.
sensors and the corresponding class label (i.e. direction to where the robot should navigate next). All the sensor readings are sampled at a rate of 9 samples per second. The second dataset file contains four sensor readings with regard to ‘front distance’, ‘left distance’, ‘right distance’ and ‘back distance’ and the corresponding class label. They consist of minimum sensor readings among those within 60-degree arcs located at the front, left, a right and back portion of the robot. The third dataset file contains only the front and left distances and the corresponding class label. All the 24 US readings and the distances were collected at the same time step and hence each file as the same number of sensor readings instances. The possible direction of the robot (class labels in the dataset) is: ‘Move-Forward’, ‘Slight-Right-Turn’, ‘Sharp-Right-Turn’ and ‘Slight-Left-Turn’. Furthermore, the statistical information related to each sensor readings obtained from the all 24 US sensors could be obtained from [23, 24]. Table 1 depicts a summary of class distribution in the overall dataset. It should be noted that there is a total of 5456 number of sensor instances in each dataset.

<table>
<thead>
<tr>
<th>Direction</th>
<th>Number of samples</th>
<th>Proportion (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move-Forward</td>
<td>2205</td>
<td>40.41%</td>
</tr>
<tr>
<td>Sharp-Right-Turn</td>
<td>2097</td>
<td>38.43%</td>
</tr>
<tr>
<td>Slight-Right-Turn</td>
<td>826</td>
<td>15.14%</td>
</tr>
<tr>
<td>Slight-Left-Turn</td>
<td>328</td>
<td>6.01%</td>
</tr>
</tbody>
</table>

4.2. Preprocessing of dataset

In all the sensor readings dataset files, there were no missing values. However, the class labels are non-numeric. Hence, we converted the class labels to numeric values as following. We replaced ‘Move-Forward’ with 1, ‘Sharp-Right-Turn’ with 2, ‘Slight-Right-Turn’ with 3 and ‘Slight-Left-Turn’ with 4.

4.3. Simulation parameters

For any metaheuristic based approach, setting a basic set of parameters is a crucial task. In this work, we set different parameters in GSPSO-ANN as follows. For GSPSO metaheuristic, a number of agents in a population ($N_{pop}$) is set to two different values 30 and 50 to study if there is any effect on performance on navigation decision. The inertia weights ($w$), $c_1'$ and $c_2'$ are set to 2 each for good convergence behavior; inertia weight decreasing over time from ($w_{max} = 0.9$) to ($w_{min} = 0.5$) which has the effect of narrowing the search [9, 20]. Gravitational constant, $G_0$ is set to 1 and the constant $\eta$ is set to 20.

For ANN, deciding on how many neurons should be present in the hidden layer has been a difficult task. Hence, we set the number of neurons in the hidden layer, denoted as $M$, relative to the number of inputs ($N$). We studied the performance of GSPSO-ANN for robot navigation for three different values of $M$ which are $M = N$, $M = 2N$ and $M = 3N$. As the dataset is dimensionally large i.e. number of instances are very large, we assumed that number of epochs should be kept to 100. Hence, training was performed for 100 epochs for each parameter settings discussed above.
4.4. Obtained results

In any machine learning model, performance should not be judged just based on a single execution of the algorithm. To avoid any biasing towards the performance, in this work, we adopted 10-fold cross validation (10-CV) for evaluating the performance of our proposed robot navigation method based on ANN. For clarity, it is worth mentioning that 10-CV refers to dividing the whole dataset into 10 partitions, training the ANN with 9 partitions and testing with the rest 1 partition. Repeat this process 10 times for each independent test partitions. Finally, note the average performance obtained by the model. We present the results obtained by our proposed GSPSO-ANN for each dataset independently for clarity and readability. Table 2 and Table 3 depicts performance of GSPSO-ANN for the sensor_readings_2.data. Similarly, Table 4 and Table 5 depicts performance of GSPSO-ANN for the sensor_readings_4.data; Table 6 and Table 7 depicts performance of GSPSO-ANN for the sensor_readings_24.data.

From Table 2, it is seen that although the dataset is small (i.e. only two input sensor readings), the proposed model GSPSO-ANN could be able to achieve a minimum $MSE \approx 0.777$. However, the number of agents in population could be observed to have minimal effect on the achieved accuracy of decision making. The maximum decision accuracy achieved for this dataset is 79.6703% when the number of neurons in the hidden layer is set to the highest of the available settings i.e. $3N$ for the dataset sensor_readings_2.data, however with the expense of more simulation time. This is because, when the number of neurons in ANN increases relatively higher computations are required which makes it difficult for achieving quickest decision or prediction when input file dimension is large. However, the accuracy obtained for the sensor_readings_2.data file is smaller than the accuracy obtained by [15] where it was 86.3820%. A sample error convergence plot has been given in the Appendix.

<table>
<thead>
<tr>
<th>$M$</th>
<th>$MS E_{min}$</th>
<th>Accuracy(%)</th>
<th>Simulation time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.779523</td>
<td>79.1026</td>
<td>42.6632</td>
</tr>
<tr>
<td>4</td>
<td>0.777914</td>
<td>78.9194</td>
<td>56.0309</td>
</tr>
<tr>
<td>6</td>
<td>0.777138</td>
<td>79.6703</td>
<td>69.6747</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$M$</th>
<th>$MS E_{min}$</th>
<th>Accuracy(%)</th>
<th>Simulation time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.777805</td>
<td>79.4615</td>
<td>70.9680</td>
</tr>
<tr>
<td>4</td>
<td>0.777600</td>
<td>78.7546</td>
<td>93.6552</td>
</tr>
<tr>
<td>6</td>
<td>0.778960</td>
<td>79.3407</td>
<td>115.9977</td>
</tr>
</tbody>
</table>

In the case of sensor_readings_4.data dataset, some interesting observation could be made. The GSPSO-ANN could be seen to be degrading its performance of decision making with the increase in the number of hidden neurons ($M$) irrespective of the number of population ($N_{pop}$). From Table 4 it could be seen that the performance accuracy degrades from 73.6630% to 70.9707% however with no effect of $N_{pop}$ on minimum $MSE$. Similar behavior could also be seen in the case of higher number agents in the population. This type of behavior could be seen in GSPSO metaheuristic when the exploration in the search space is more than exploitation. In other words, the agents could be able to stick to each other while searching for a minimized state rather making the search process diversified which often
makes the search expensive when the search space is large. Moreover, one should note that the best result obtained by the proposed GSPSO-ANN for this dataset is 73.6630% which is better than 70.0880% accuracy achieved by previous work in [15].

Table 4
10-CV results for sensor_readings_4.data with $N_{pop} = 30$

<table>
<thead>
<tr>
<th>$M$</th>
<th>$MS E_{min}$</th>
<th>Accuracy(%)</th>
<th>Simulation time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0.735533</td>
<td>73.6630</td>
<td>140.9126</td>
</tr>
<tr>
<td>8</td>
<td>0.717203</td>
<td>72.8205</td>
<td>248.2709</td>
</tr>
<tr>
<td>12</td>
<td>0.739393</td>
<td>70.9707</td>
<td>355.9865</td>
</tr>
</tbody>
</table>

Table 5
10-CV results for sensor_readings_4.data with $N_{pop} = 50$

<table>
<thead>
<tr>
<th>$M$</th>
<th>$MS E_{min}$</th>
<th>Accuracy(%)</th>
<th>Simulation time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0.708280</td>
<td>73.1502</td>
<td>238.0684</td>
</tr>
<tr>
<td>8</td>
<td>0.702347</td>
<td>71.9414</td>
<td>414.3792</td>
</tr>
<tr>
<td>12</td>
<td>0.674267</td>
<td>68.8095</td>
<td>593.7492</td>
</tr>
</tbody>
</table>

The sensor_readings_24.data file is very large in dimension i.e. $5456 \times 25$ including target attribute in the dataset. Hence, it is more challenging for any machine learning model to achieve a satisfactory performance for large dataset so as to make the model to be adopted in practice. The evaluated performance of our proposed GSPSO-ANN for robot navigation are presented in Table 6 and Table 7. The proposed GSPSO-ANN could achieve satisfactory performance with different hidden neuron settings and population sizes. Here, the number of hidden neurons could be increased from 24 to as high as 72 which is a very high value and it required very high usage of computing power available for this work. However, we could achieve a maximum performance of 73.9927% with $M = 48$ and $N_{pop} = 50$. This performance is better than 69.7214% accuracy achieved by [15]. Unlike the sensor_readings_4.data file, degradation in performance (either minimum $MS E$ or accuracy) could not be seen for this dataset. There could following reasons for such behavior of GSPSO-ANN for this dataset. The number of neurons in the hidden layer is more as compared the former cases which enhances the processing and generalization capability of ANN. Again, the dataset is large due to which the network could be trained with a higher number of training features rather than a few features such as 2 or 4. When the number of features is more there is more knowledge associated with the network in the form of inter-layer weights which in-turn gives the network more predictive and regeneration capability.

Table 6
10-CV results for sensor_readings_24.data with $N_{pop} = 30$

<table>
<thead>
<tr>
<th>$M$</th>
<th>$MS E_{min}$</th>
<th>Accuracy(%)</th>
<th>Simulation time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>0.773349</td>
<td>71.3553</td>
<td>4647.7253</td>
</tr>
<tr>
<td>48</td>
<td>0.770749</td>
<td>66.8681</td>
<td>9246.5584</td>
</tr>
<tr>
<td>72</td>
<td>0.767751</td>
<td>71.8864</td>
<td>13869.0084</td>
</tr>
</tbody>
</table>

The performances reported in the above tables (Table 2 through to Table 7) show that the implemented GSPSO metaheuristic could be able to achieve a high-quality decision taking capability with regard to its exploration and exploitation policy while searching for the best solution in the search space during
Table 7

10-CV results for sensor_readings_24.data with \( N_{\text{pop}} = 50 \)

<table>
<thead>
<tr>
<th>( M )</th>
<th>( MSE_{\text{min}} )</th>
<th>Accuracy(%)</th>
<th>Simulation time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>0.767380</td>
<td>70.4029</td>
<td>7784.9553</td>
</tr>
<tr>
<td>48</td>
<td>0.759938</td>
<td>73.9927</td>
<td>15423.8735</td>
</tr>
<tr>
<td>72</td>
<td>0.760988</td>
<td>73.6630</td>
<td>23129.7800</td>
</tr>
</tbody>
</table>

The process of training the ANN. To support our claims in this discussion we incorporate a few existing algorithms which use the same datasets for benchmarking the models. We summarized the results obtained by application of GS-FFNN technique in [15] and gradient descent based ANN (GD-ANN) model in [25]. These two techniques use more or less similar parameter settings and the corresponding results are depicted in the Table 8. In [15], the parameters are \( M \) and \( N_{\text{pop}} \) where \( M \) is set between 2\( N \) and \( N_{\text{pop}} \) is set to 20 and 30 for each \( M \) setting. The GD-ANN [25] varies \( M \) and trains the ANN with different values of learning rate, denoted as \( \alpha \) in the range (0,1]. For better comparison among the models, we average the performances overall the \( \alpha \) values and present the summarized results as mean±standard deviation (std).

Table 8

Comparative summary of accuracy(%) achieved by various methods for same datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>( M )</th>
<th>GSPSO-ANN (this work)</th>
<th>GS-FFNN [15]</th>
<th>GD-ANN [25] ( \forall \alpha ) (mean±std)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sensor_reading_2.data</td>
<td>2</td>
<td>79.1026 79.1026</td>
<td>78.4615 78.4615</td>
<td>86.3820 85.1723 –</td>
</tr>
<tr>
<td>sensor_reading_2.data</td>
<td>4</td>
<td>78.9194 78.9194</td>
<td>78.7546 78.7546</td>
<td>85.7405 85.6855 87.80±2.49</td>
</tr>
<tr>
<td>sensor_reading_2.data</td>
<td>6</td>
<td>79.6703 79.6703</td>
<td>79.3407 79.3407</td>
<td>– – 88.77±2.70</td>
</tr>
<tr>
<td>sensor_reading_2.data</td>
<td>8</td>
<td>79.6703 79.6703</td>
<td>79.3407 79.3407</td>
<td>– – 89.56±3.71</td>
</tr>
<tr>
<td>sensor_reading_2.data</td>
<td>10</td>
<td>– – – –</td>
<td>– – 88.53±4.00</td>
<td></td>
</tr>
<tr>
<td>sensor_reading_4.data</td>
<td>4</td>
<td>73.6630 73.6630</td>
<td>73.1502 73.1502</td>
<td>70.0880 67.5037 –</td>
</tr>
<tr>
<td>sensor_reading_4.data</td>
<td>6</td>
<td>– – – –</td>
<td>– – 66.96±2.34</td>
<td></td>
</tr>
<tr>
<td>sensor_reading_4.data</td>
<td>8</td>
<td>– – – –</td>
<td>– – 67.42±3.43</td>
<td></td>
</tr>
<tr>
<td>sensor_reading_4.data</td>
<td>10</td>
<td>– – – –</td>
<td>– – 68.49±6.16</td>
<td></td>
</tr>
<tr>
<td>sensor_reading_4.data</td>
<td>12</td>
<td>– – – –</td>
<td>– – 71.59±8.16</td>
<td></td>
</tr>
<tr>
<td>sensor_reading_24.data</td>
<td>10</td>
<td>– – – –</td>
<td>– 53.5740 69.7214 –</td>
<td></td>
</tr>
<tr>
<td>sensor_reading_24.data</td>
<td>20</td>
<td>– – – –</td>
<td>– 53.5740 47.5073 –</td>
<td></td>
</tr>
<tr>
<td>sensor_reading_24.data</td>
<td>24</td>
<td>– – – –</td>
<td>– 70.4029 – –</td>
<td></td>
</tr>
<tr>
<td>sensor_reading_24.data</td>
<td>30</td>
<td>– – – –</td>
<td>– 45.4912 45.4362 46.50±0.00</td>
<td></td>
</tr>
<tr>
<td>sensor_reading_24.data</td>
<td>35</td>
<td>– – – –</td>
<td>– 54.7104 40.8908 46.50±0.00</td>
<td></td>
</tr>
<tr>
<td>sensor_reading_24.data</td>
<td>40</td>
<td>– – – –</td>
<td>– – 46.50±0.00</td>
<td></td>
</tr>
<tr>
<td>sensor_reading_24.data</td>
<td>45</td>
<td>– – – –</td>
<td>– – 46.50±0.00</td>
<td></td>
</tr>
<tr>
<td>sensor_reading_24.data</td>
<td>48</td>
<td>– – – –</td>
<td>– 66.8681 73.9927 69.7214 35.0806 –</td>
<td></td>
</tr>
<tr>
<td>sensor_reading_24.data</td>
<td>50</td>
<td>– – – –</td>
<td>– – 46.50±0.00</td>
<td></td>
</tr>
<tr>
<td>sensor_reading_24.data</td>
<td>72</td>
<td>– – – –</td>
<td>– – 71.8864 73.6630 – –</td>
<td></td>
</tr>
</tbody>
</table>

The GD-ANN technique constantly fails to achieve any better performance for the large dataset like sensor_readings_24.data even though the number of hidden neurons are increased from lower to higher. However, it could be seen that the method is capable of performing far better than other two methods.
for sensor_readings_2.data which is smaller in the number of attributes. It could be observed that for GD-ANN when the number of inputs is less, the underlying complexity of the model is less and hence it could be able to generate the adapted pattern during the testing of the models by utilizing the knowledge acquired during training as weights. Moreover, it should be noted that when dataset dimension grows the number of input in the NN model also increases and it would be difficult to capture the underlying behavior of patterns in the training set into a single layer of hidden neurons for algorithms like GD-ANN. This is the reason due to which for the large dimensional dataset like sensor_readings_24.data, GD-ANN could not regenerate the results during testing even though the number of hidden neurons are varied. Moreover, in reality, a robot might not rely on only two sensors for deciding on its next direction rather all distances from its body to the obstacles need to be taken care for better decision making. Moreover, the work in [15] could slightly achieve better performance as compared to [25]. But, if we consider its performances with different $M$ settings for $N_{pop} = 30$, the results are unreliable. But, the present work could demonstrate better performance for almost all the cases with the best performance of 73.9927% even though the dataset dimension is large. Which means that the proposed GSPSO-ANN could able to show better decision-making capability if adopted in practice.

5. Conclusion and future directions

In this work, we propose to use a hybrid metaheuristic based algorithm called GSPSO [22] for controlling wall following mobile robot navigation with ANN. We evaluated our proposed model with three different well-tested benchmark sensor readings datasets obtained from the navigation of the wall following mobile robot SCITOS G5. The performance reported by the proposed model is satisfactory with the highest decision-making the accuracy of 79.6703% for the smallest dataset and 73.9927% for the largest dataset.

In this work, the available sensor readings dataset is assumed to be fault free and there is no correlation among two different instances of the sensor readings. Which means that the occurrence of a set of sensor readings does not essentially depend on another set of sensor readings. However, in reality, this might not be always the case while considering the mechanical aspects of the robot and mechanical failures. Furthermore, it is believed that the sensor readings are completely certain and true and based on this assumptions the methods have been implemented and experimented. However, the data sensed by the sensors might not be always true or certain. In such a scenario, it would be interesting to incorporate the uncertainty in the automatic navigation control of the robot by using specific uncertainty handling techniques such as fuzzy control approach as discussed in [26].

Moreover, one could looked into the following two aspects as future extension of this work. Developing a robust method to work for cases when the sensors are faulty. Another considerable extension would be to take care of the cases when a few sensors get damaged during real-time navigation in the field. Solving the above questions are quite challenging and yet interesting.

Appendix

References

Fig. 3. A sample error convergence semilogy for GSPSO-ANN for sensor_readings_2.data dataset with $M = 2$ and $N_{pop} = 30$


