Soft Computing (2020) 24:1315–1333
https://doi.org/10.1007/s00500-019-03969-6

METHODOLOGIES AND APPLICATION

Neural collision avoidance system for biomimetic autonomous underwater vehicle

Tomasz Praczyk[1]

Published online: 5 April 2019

Abstract
Autonomous underwater vehicles (AUVs) are underwater robots which are able to perform certain tasks without the help of a human operator. The key skill of each AUV is the capability to avoid collisions. To this end, appropriate devices and software are necessary with the potential to detect obstacles and to take proper decisions from the point of view of both the task and safety of the vehicle. The paper presents a neural collision avoidance system (NCAS) designed for the biomimetic autonomous underwater vehicle (BAUV). The NCAS is a component of the path following and collision avoidance system (PFCAS), which as the name implies is responsible for safely leading the vehicle along a desired path with collision avoidance. The task of NCAS is to make decisions regarding vehicle maneuvers in the horizontal plane, but only in the close proximity of the obstacles. It is implemented as an evolutionary artificial neural network designed by means of a neuro-evolutionary technique called assembler encoding with evolvable operations (AEEO). The paper outlines operation and construction of the BAUV as well as the PFCAS, the role of the NCAS in the entire system, and briefly presents AEEO as well as reporting on the experiments performed in simulation.

Keywords Neural networks · Underwater vehicle · Biomimetic robot · Collision avoidance · Autonomy

1 Introduction

Currently, the underwater environment is increasingly being explored by humans. Previously, only submarines, submersibles, and divers in heavy, uncomfortable diving suits visited the marine depths. Today, underwater technology is becoming more and more accessible, even for the common man. Underwater vehicles, manned and unmanned, are currently used for various tasks, e.g., monitoring underwater infrastructure such as pipelines and oil rigs, monitoring the state of coral reefs, and for detection and neutralization of mines, and, of course, also for pleasure. The most frequently used type of underwater vehicles is the remotely operated vehicle (ROV), which is controlled from distance by a human operator. In addition to ROVs, autonomous vehicles are also in use and, in contrast to the ROVs, they have skills to operate independently of a human or other external supporting systems. The level of autonomy can vary; for example, it can only refer to the power system which means that the vehicle does not need external sources of energy, and it may also relate to operation in some emergency situations like lack of contact with the operator due to malfunction of communication system; usually, in this case, the vehicle has to safely reach a predefined nearby marine area. However, the autonomy may also mean complete independence of the vehicle in performing its mission from the very beginning to the end. Of course, the mission has first to be defined by the operator, but this is the only activity in which a human is involved, and the rest is the responsibility of the vehicle.

In order for the vehicle to be able to act without any external support, it has to be equipped with systems which provide operational independence. To decide for itself what actions has to be taken in order to achieve a desired goal, the vehicle has to be in possession of information describing the state of both the vehicle and its surroundings. The fundamental information required for safe underwater navigation is information about vehicle position, spatial orientation as well as the presence of obstacles. This information is necessary to lead the vehicle along a desired path specified as an ordered

Communicated by V. Loia.

✉ Tomasz Praczyk
t.praczyk@amw.gdynia.pl

[1] Institute of Naval Weapon, Polish Naval Academy, Gdynia, Poland

set of waypoints, and simultaneously to avoid collisions with obstacles located along this the path.

Generally, the task above can be divided into two main sub-tasks. The first one is to determine a global path from a starting point to a destination point based on the information from navigational charts with accurate bathymetry of the sea. Usually, the global path is planned either manually by a human operator or by means of well-known pathfinding optimization algorithms like A-star (Dechter and Pearl 1985; Hart et al. 1968), and this takes place before the start of the vehicle mission. The second task is dedicated to safe vehicle navigation between successive waypoints in the presence of obstacles laying in the vehicle's global path and is performed by vehicle systems.

The current paper is focused only on the latter task, and in general, three approaches can be used for that purpose (Campbell et al. 2012; Holtung Eriksen 2015; Tan 2006). The first of these is the so-called deliberative or sense-plan-act approach which is, in fact, the global path planning method applied locally. This means that the output product of the approach is the whole collision-free path between the current position of the vehicle and the next waypoint. To determine the path, different techniques can be used, e.g., heuristic search methods (Naeem et al. 2012; Tan 2006; Young-il et al. 2004; Yang et al. 2010) (A* algorithm, its modifications, and others), evolutionary (Kanakakis and Tsourveloudis 2007; Nikolos et al. 2007; Smierzchalski 1999; Smierzchalski and Michalewicz 2005; Szlapczynski and Szlapczynska 2012), and ant colony algorithms (Zeng et al. 2009). The condition for application of the approach above is a map of the underwater environment built based on the information gathered by vehicle sensors.

The main problem with applying these methods to collision avoidance is their computational complexity resulting from their inherent search or population nature. They generally cannot give a guaranty of finding a collision-free solution in an assumed short period; how long they work mostly depends on the conditions they are dealing with. In most situations at sea, they will work properly and without delay; however, there may be situations, e.g., a suddenly detected underwater object, when they may fail by providing a collision-free plan too late. All this makes the methods above suited rather for either offline global path planning or planning in conditions when there is no need to make rapid decisions.

The second approach, known as reactive or sense-act, has a local nature meaning that it relies only on local information directly from sensors or from a map, and it then generates a single maneuver for a next step, not the entire path to a next waypoint. Generally, the main drawback of this approach is locality and, in consequence, globally non-optimal paths and sometimes even becoming trapped in dead ends. On the other hand, the advantage over deliberative methods is high speed, which makes them appropriate for applications where a fast response is required.

The example of the reactive approach is the virtual force field (VFF) algorithm (Im and Oh 2000; Im et al. 2002; Kwon et al. 2005) which builds a model of attracting and repulsing forces affecting a vehicle moving in an environment. The former forces attract the vehicle to a next waypoint and sometimes also to a predefined path, whereas the latter ones push it back from obstacles. To determine the magnitude of forces, angle and distance parameters to all force sources are applied which are usually additionally preprocessed, for example, by means of fuzzy logic. The resulting force indicates the direction of movement for the vehicle.

An extension of the VFF is a vector field histogram (VFH) (Borenstein and Koren 1991) which models the surroundings of the vehicle in the form of three- or two-dimensional grids with appropriately situated obstacles. The grid representation is then converted into a histogram which shows density of the obstacles in different directions and in this way makes it possible to choose a direction with the lowest density.

The other example of the reactive approach is the dynamic window algorithm (DWA) (Holtung Eriksen 2015; Fox et al. 1997). In contrast to the above two approaches, it controls the vehicle by determining progressive and angular velocities instead of indicating directions. Each velocity pair corresponds to a predicted circular vehicle trajectory, and the task of the algorithm is to find a pair which maximizes an objective function including three different components. The first component is responsible for leading the vehicle toward a next waypoint, the second "repulses" it from the obstacles, whereas the third promotes fast movement of the vehicle.

In Wilson et al. (2003), a reactive algorithm is proposed called a line-of-sight counteraction navigation (LOSCAN) which is meant for two-ship encounter situations. As the authors of Wilson et al. (2003) mention, their algorithm takes its idea from missile proportional navigation with the difference that a missile objective is to capture the target, whereas the task of LOSCAN is to avoid it. The basic concept of the LOSCAN is to increase the misalignment between the ship relative velocity and the line of sight through providing appropriate acceleration commands. The algorithm works in two stages: first, a risk assessment is performed, and then, if a risk of collision exists, it generates navigation commands to avoid a collision.

A separate family of reactive approaches is based on fuzzy logic (Harris and Moore 1989; Kanakakis et al. 2004; Kim and Kim 2009). In Kanakakis et al. (2004), fuzzy logic framework is applied on three levels, that is, on sensor fusion, collision avoidance, and motion control levels. The first level is responsible for producing possibilities of collision in four cardinal directions, i.e., front, right, left, and back. Each direction can be described by the following possibilities: not possible, possible, and highly possible. Then, the collision

avoidance module supplied with the calculated possibilities, along with pitch and heading errors, determines how pitch and heading should be changed, for example, in the following way: left fast, left slow, down fast, down slow. The errors mentioned above point to a next waypoint. In addition to a new direction of movement, the collision avoidance module also indicates the surge speed by producing the following values: slow, normal, and high. In the final step, the motion control module transforms outputs of the previous level into specific controls for vehicle drive.

In Kim and Kim (2009), fuzzy reasoning is applied on two levels, i.e., collision avoidance and collision risk degree computation level. In the former case, the collision risk is calculated based on distance of the closest point of approach (DCPA) and time of the closest point of approach (TCPA), that is, two parameters which are commonly used by navigators to evaluate risk of collision with a tracked object. In turn, the collision avoidance is performed according to an algorithm which for a small obstacle runs fuzzy reasoning with the purpose of indicating a maneuver in the horizontal plane. Once a wide obstacle is detected, first, the vehicle goes up, and then, it tries again to apply a maneuver in the horizontal plane. To fix a collision-free maneuver, the fuzzy reasoning module indicates a narrow vertical sector in front of the vehicle toward which it should move.

Application of artificial neural networks (ANNs) for collision avoidance is presented in Chang and Gaudiano (1998),Guerrero-Gonzlez et al. (2011). In Chang and Gaudiano (1998), a general neural method is defined and tests on a small land vehicle are reported, whereas in Guerrero-Gonzlez et al. (2011) adaptation of the method to an underwater vehicle is given. The ANN proposed in the above papers is trained in an unsupervised manner to avoid obstacles. While training, the vehicle moves in a cluttered environment, collides with obstacles and, in this way, the ANN that controls the vehicle learns which signals on sensors and which controls lead to collisions. In the operation phase, the appearance of a "colliding" signal on sensors activates neurons which block controls that contribute to the collision.

Neural collision avoidance system (NCAS), presented in the current paper, is the next representative of the neural reactive approach for underwater vehicles. The NCAS is a component of the path following and collision avoidance system (PFCAS), and, like the solution described in Kim and Kim (2009), it is responsible for collision avoidance in the horizontal plane in the close proximity of obstacles. Changing depth and maneuvering away from the obstacles is the task of other PFCAS components.

The NCAS and PFCAS were designed for the AUV as presented in Fig. 1 which, due to its biomimetic drive (fins instead of a traditional screw propeller), is called biomimetic AUV, in short, BAUV. The task of this vehicle is generally to follow a predefined path with collision avoidance and provide simultaneous recording data from cameras and sonar. Construction of the vehicle, including the above-mentioned systems, started in simulation because of the high cost of tests with the application of the a real vehicle and the very high cost of mistakes committed in the early phases of vehicle construction. Moreover, the tests in simulation made it possible to examine key components of the BAUV, e.g., the PFCAS, in conditions which are hard to obtain in the real world.

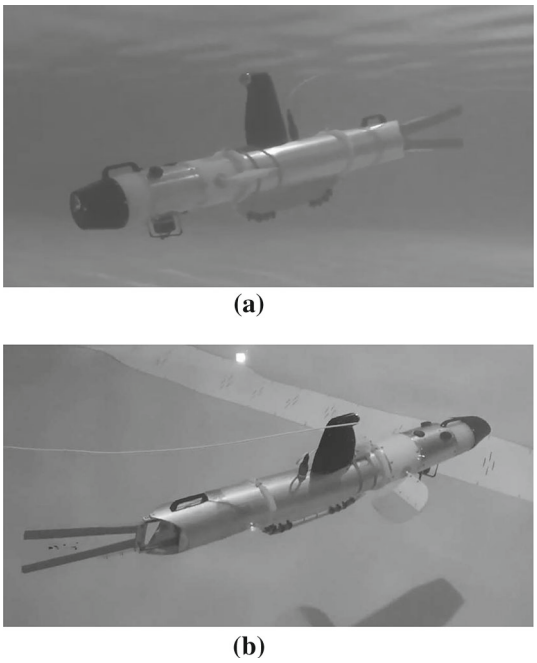


Fig. 1 BAUV used in preliminary experiments at a pool

Originally, the PFCAS had an algorithmic form and its parameters were determined in an evolutionary manner (Praczyk 2015a). Although the performance of the evolutionary tuned PFCAS was generally satisfactory, in most testing scenarios it was fixing safe paths; however, there were situations in which the system failed. The consequence was the decision to modify the PFCAS by changing the method for determining collision-free maneuvers in the horizontal plane. Due to the successful application of ANNs in a collision avoidance system designed for an autonomous surface vehicle (ASV) (Praczyk 2015b), the decision was made to apply ANNs in the BAUV. In the experiments reported in Praczyk (2015b), it appeared that ANNs used to control the ASV successfully coped with avoiding collisions with both stationary and moving obstacles. In addition to the effectiveness itself, a key feature of ANNs which decided about their use in ASV was their high processing speed compared to planning methods. As a result of a specific construction of a BAUV obstacle detection system[1] which requires a vehicle's

[1] Two narrow-sighted echo-sounders.

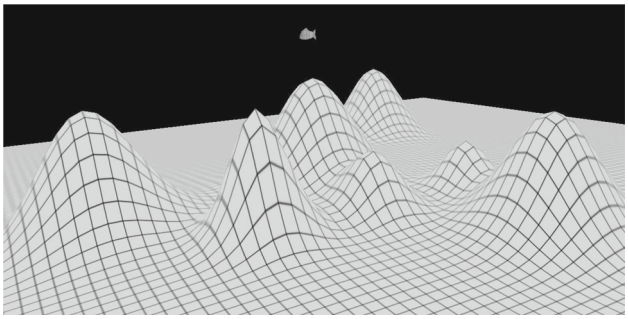


Fig. 2 Environment used in experiments

ability to make quick decisions, the high speed of ANNs is also a desirable feature which was taken into account when choosing them as the BAUV collision avoidance system.

To train collision avoidance ANNs, in Praczyk (2015b), a neuro-evolutionary method called assembler encoding with evolvable operations (AEEO) (Praczyk 2015c) was used. In the BAUV, the same method was applied to construct a collision avoidance ANN which in the paper is named neural collision avoidance system.

In order to appropriately prepare the NCAS, intense experiments in a simulated underwater environment were carried out. The environment presented in Fig. 2 was designed to include "mountains" of different heights and covered areas with the purpose of making it challenging for the NCAS and the entire PFCAS. In the experiments, two collision avoidance solutions were compared, that is, the original fully algorithmic PFCAS (APFCAS) and the PFCAS combined with the NCAS (PFCAS–NCAS). The current paper reports the experiments and presents all the above-mentioned systems. The rest of the paper is organized as follows: Sect. 2 outlines the real BAUV and its simulation model used in the experiments, Sect. 3 specifies APFCAS, PFCAS–NCAS, and other BAUV components that are involved in obstacle avoidance, Sect. 4 describes AEEO, that is, the neuro-evolutionary method applied to design the NCAS, Sect. 5 reports the experiments, and the final section summarizes the paper.

2 Vehicle and its model

The BAUV whose collision avoidance system is the main subject of the paper consists of a number of hardware and software components. They can all be divided into seven functional parts, i.e., drive, energy system, sensors, collision avoidance, navigation, communication, and control. From the point of view of collision avoidance, the most important elements of the vehicle are: drive, sensors, and navigation.

As for the drive, the BAUV is equipped with two pectoral fins and one tail fin with the engines. The pectoral fins are mainly used to control depth, whereas the tail fin is applied to control heading.[2] When the BAUV is on the surface, the pectoral fins can also be used to change heading, in which case they work in the opposite direction, i.e., one fin pushes the vehicle forward and the second pushes backward.

The motion of the pectoral fins is controlled with the three parameters, i.e., frequency, amplitude, and neutral position. Frequency and amplitude are parameters of motion, whereas the neutral position determines the position of fins around which their motion takes place, and each fin moves up and down from the neutral position. In turn, the tail fin is controlled with two parameters, i.e., neutral position and frequency, the amplitude is constant in this case.

Characteristic of a fin-propelled vehicle is oscillation of its motion in the horizontal plane which forced the project team to reduce the obstacle detection system to only three echo-sounders located at the top, bottom, and front of the vehicle and facing up, forward, and down. Application of a more advanced system equipped, for example, with a ring of sonar sensors covering a wider area in front of the vehicle would be at least problematic because of the above-mentioned oscillations.

In the experiments with the collision avoidance system reported in the paper, a simulation model of the BAUV was applied with the inclusion of oscillations in the horizontal plane. In order to maximally speed up calculations during evolutionary training of the NCAS, the model was not constructed in the traditional way, i.e., as a set of differential equations, but rather as a combination of recorded behavior of the traditional model (in the form of matrices with recorded parameters of the model in selected maneuvers, e.g., during turning left/right) and C++ implementation, whose task was to "glue" pieces of records into one coherent model. This way, calculations performed during simulations and associated with repeated, intense use of the model were shortened several times.

Construction of the model in the form of recorded behaviors of the vehicle and then "gluing" by means of the specialized software has yet another very important purpose. Namely, it enables quick and easy modeling of almost every underwater vehicle, including further versions of the BAUV and its successors. To perform simulations on another vehicle, it is necessary to record its behavior for selected maneuvers, to replace the matrices that currently represent the BAUV with matrices that specify this vehicle and finally to gently tune some parameters of the "gluing" software. It is assumed that it is a considerably easier and faster process than building the model of the vehicle according to the traditional approach.

To perform simulations with collision-avoiding BAUV, in addition to the vehicle itself, its vision system, i.e., its

[2] Because the vehicle is not equipped with a velocity sensor, the only controlled parameters are heading and depth.

sensors, had also to be modeled. Two virtual echo-sounders were applied as the vehicle sensors, the first looked straight ahead, whereas the second looked down and both of them were installed in the front of the BAUV. To simulate the operation of echo-sounders, the distance was measured between a current position of the BAUV and a first point located on the straight line being a prolongation of the observation line of the echo-sounder.

The last vehicle component crucial for collision avoidance is the navigational system which provides information about position (latitude, longitude, depth), and spatial orientation of the vehicle (roll, pitch, yaw). In the real vehicle, this information is acquired from GPS (on the surface), an inertial unit, a speed estimator, and a pressure sensor. In the simulations, the assumption was that all the navigational information provided by the above sensors/devices was available to the collision avoidance system. At each point of the simulation, the system knew an accurate vehicle position and spatial orientation.

3 Collision avoidance

Three different BAUV software components are involved in collision avoidance. The first is responsible for handling signals provided by echo-sounders, and then for their filtration and interpretation. The task of the next one is to build a map of the environment, to plan collision-free paths, and also to determine different variants of collision-free maneuver. The last component is responsible for collision avoidance near the obstacles and is implemented in the form of the aforementioned APFCAS and PFCAS–NCAS.

3.1 Obstacle detection and localization

The first layer in the entire collision avoidance system performs preprocessing of signals produced by echo-sounders. Since the echo-sounders may generate noisy output, Kalman filtration is used in the first stage of processing. In the simulations, this stage was omitted due to perfect indications of virtual echo-sounders. In the next stage, the filtered signal which is, in fact, the distance to an obstacle or the sea floor, is converted either into coordinates of the obstacle in the local coordinate system or into true distance to the floor. The front echo-sounder, depending on vehicle orientation, may point out either an obstacle or the floor, and the same applies to the bottom echo-sounder. To determine what echo-sounders "see," spatial orientation of the vehicle is necessary; moreover, outputs of both echo-sounders are compared to each other. When an obstacle is detected, its coordinates are compared to coordinates of a previously detected obstacle. When the distance between the coordinates is above a threshold, the detected obstacle is passed on to the mapping process; otherwise, it is neglected.

3.2 Mapping, path planning, collision-free maneuvers

After receiving the obstacle, the mapping process tests whether it overlaps with other obstacles added previously to the map. If so, the obstacle is neglected, if not, it is inserted into the map as a new obstacle. Since the obstacles are represented in the map as spheres of a definite radius, it is assumed that they overlap if the degree of the coverage exceeds thirty percent.[3]

An important responsibility of the mapping process is also to generate safe local paths to a next waypoint included in the global path. This task is performed in two different situations, and for that purpose, two distinct algorithms are used. A form of A-star algorithm is run periodically to propose a collision-free path when the vehicle is far away from all obstacles and no rapid vehicle response is necessary. When any safe maneuver cannot be found reactively due to all directions being blocked by obstacles, a quick planning algorithm described in Praczyk (2016) is used.

The decision to rely on the BAUV collision avoidance system mainly on a reactive basis is the consequence of sensors being applied for obstacle detection. Only one echo-sounder in front of the vehicle, which, as already mentioned, is the consequence of horizontal vehicle oscillations, causes frequent changes in the environment map. The vehicle observes only a very narrow strip ahead with the effect that the map initially includes only one obstacle, of course, if it exists. Each change of vehicle heading moves the front echo-sounder to a neighboring fragment of the environment and if an obstacle is in the way, it is added to the map. In a new environment region, each change of heading produces a new obstacle which often forces the vehicle to change direction of movement. Application of path planning algorithms as the main collision avoidance tool would lead, in that situation, to continuous re-planning, which would be ineffective and, in the close proximity of obstacles, also dangerous.

Since the mapping process is the only software component which is in possession of complete knowledge about the environment, its task is also to help the PFCAS in determining collision-free maneuvers or, more strictly, the directions of motion that avoid obstacles. To this end, the mapping process, on demand of the PFCAS, provides four different horizontal directions as presented in Fig. 3 (numbered from 1 to 4), including three which are collision-free. Direction no. 1 is the nearest collision-free maneuver from the current vehicle heading (direction no. 0) when turning left, direction no. 2 is the same except when turning right, direction no. 3 is

[3] This parameter was fixed arbitrarily.

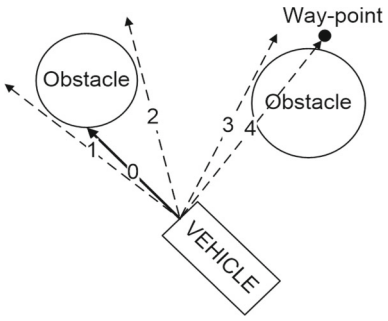


Fig. 3 Directions determined by mapping process

the nearest collision-free maneuver from heading to a next waypoint which is colliding and is numbered as 4.

3.3 APFCAS

The APFCAS leads the vehicle along a path specified as a list of waypoints. When necessary, the system avoids collisions by changing heading or depth. The high-level pseudo-code of the APFCAS is presented in Fig. 4.

The `moveBack` operation changes the direction of vehicle motion to the opposite direction, e.g., if the vehicle moves forward, `moveBack` changes the maneuver to move backward, if the vehicle goes up, `moveBack` changes the maneuver to go down. The reverse maneuver is stopped when the vehicle is a safe distance from obstacles and is able to continue its mission.

The task of the `moveForward` operation is high-level control of vehicle heading, depth, and speed which means that `moveForward` determines values of the parameters, whereas the low-level control system decides how to achieve them by means of drive. The general algorithm of `moveForward` is presented in Fig. 5 and shows that control of vehicle parameters is performed in three almost independent steps implemented by operations `setSpeed`, `setHeading`, `setDepth` whose pseudo-code is given in Figs. 6, 7, 8 and 9.

The operation of `setSpeed` is reduced to only adjusting the speed of the vehicle to the distance to the closest obstacle. If the distance is greater than an assumed threshold, the speed is set to a desired value attached to a description of a goal waypoint, otherwise it is reduced to a small value equal to SPEED_CLOSE_OBSTACLES which, as with other parameters of the APFCAS, was optimized in the evolutionary way during experiments reported in Praczyk (2015a).

`setHeading` changes the heading of the vehicle in two situations, the first being when the position of the vehicle is inside a virtual spherical obstacle and obstacles are detected above and below the vehicle, whereas the second is when the distance from the recent update of the heading is greater than an assumed threshold. The first situation may take place when collision with an obstacle is not detected by vehicle sensors; however, the vehicle position is inside a virtual obstacle added to the environmental map. In that case, when obstacles sensed by vehicle sensors make it impossible to change depth, the heading is altered, otherwise, the vehicle goes up or down. Periodical update of vehicle heading is performed to respond to different external forces which push the vehicle from a desired path to a waypoint and is performed every `thresholdDistanceFromLastTurn` meters and the value of `thresholdDistanceFromLastTurn` depends mainly on the distance from the obstacles, however, it can also be radically reduced to zero meters when a new obstacle is added to the map. Regardless of the situation, the heading value that is provided to the low-level control by means of `setParameter` is determined by the `getHeading` function which is supported by the mapping component of the PFCAS.

A new heading determined by `getHeading` is produced, in fact, by the mapping component and is either direction no. 3, no. 1 or no. 2, the mentioned directions are already specified in Fig. 3. First, direction no. 3 is tested (`heading := getDirection(3,wayPoint)`), that is, the collision-free direction toward the goal waypoint, and, if the change of the vehicle heading necessary to achieve the new heading is lower than `thresholdAngle`, the direction no. 3 is accepted and sent to the low-level control, otherwise, `getHeading` selects from directions no. 1 or 2 (`getDirection` function) and decides on the direction which needs the smallest change of vehicle heading.

`setDepth` function which, as the name implies, is responsible for control of the vehicle depth, on the one hand, maintains a safe distance from the sea bottom and, on the other hand, also minimizes the difference between a desired depth defined in goal waypoint and the current vehicle depth. When the vehicle is inside a virtual obstacle and there is no collision risk above or below the vehicle, the change of depth is dependent of `param`; otherwise, the vehicle depth is a balance between collision risk with the obstacles and the sea floor and the maintenance of desired vehicle parameters.

In total, the APFCAS has 18 parameters which affect its operation. All of them were tuned in the evolutionary way during experiments reported in Praczyk (2015a). The tuned system was applied as the point of reference for the proposed NCAS.

3.4 PFCAS–NCAS

The PFCAS–NCAS is, in fact, the APFCAS with a different `getHeading` implementation (see Fig. 10). In this case, determination of the vehicle heading is the responsibility of an evolutionary artificial neural network (EANN), a recurrent (REANN) or feed-forward version (FFEANN), designed by means of a neuro-evolutionary method called assem-

```
APFCAS::run(wayPoints)
begin
    currentWayPoint := 0;
    loop until currentWayPoint < wayPoints.size()
        if collision detected (vehicle hit obstacle)
            moveBack();
        else
            moveForward(wayPoints[currentWayPoint]);
        end if
        distanceToWayPoint := getDistance(vPosition,wayPoints[currentWayPoint]);
        if distanceToWayPoint < threshold
            currentWayPoint++;
        end if
    end loop
end
```

Fig. 4 Pseudo-code of APFCAS

Fig. 5 Pseudo-code of `moveForward`

```
moveForward::run(wayPoint)
begin
    distanceToObstacles := getDistance(vPosition,obstacles);
    setSpeed(distanceToObstacles);
    setHeading(wayPoint,distanceToObstacles);
    setDepth(wayPoint,distanceToObstacles);
end
```

Fig. 6 Pseudo-code of `setSpeed`

```
setSpeed::run(distanceToObstacles)
begin
    if distanceToObstacles < threshold
        setParameter(SPEED_CLOSE_OBSTACLES,SPEED);
    else
        setParameter(wayPoint.speed,SPEED);
    end if
end
```

bler encoding with evolvable operations (AEEO), described among other things in Praczyk (2015c). Regardless of the type, the neural network has three outputs corresponding to three different collision-free directions produced by the mapping component of the PFCAS. Selection of the direction to follow by the vehicle is indicated by the "strongest" neuron.

The input layer, depending on the network type, consists of either ten or twelve neurons which supply the EANN with the information about the surroundings of the vehicle, possible heading options, direction toward the goal waypoint, and in the case of the FFEANN, also with information about a short history of decisions taken.

The surroundings of the vehicle are divided into eight sectors of observation[4] specified in Fig. 11. The mapping component provides the EANN information about obstacles in six horizontal sectors, that is, sectors 3 to 8. This information takes the form of the closeness of the obstacles scaled to the range $\langle 0, 1\rangle$, where zero means either no obstacle or a very distant obstacle. When determining sector values, vehicle range of vision is reduced to a threshold which is a parameter of the NCAS. This way, the EANN is sensitive exclusively to obstacles which are in the vehicle proximity determined by the threshold. Formally, the sector values can be defined as follows:

$$\mathrm{SV}_2(n) = \begin{cases} 0 & \text{if } \texttt{distanceToObstacles} > D_{\max} \\ \frac{D_{\max} - \texttt{distanceToObstacles}}{D_{\max}} & \text{otherwise} \end{cases} \tag{1}$$

where n is a sector number, whereas $D_{\max}$ is the threshold for the vehicle range of vision.

In order to indicate possible options to select by the EANN, that is, directions which can be chosen by the vehicle when avoiding collisions, and direction toward the goal waypoint, the network is supplied with appropriately scaled numbers of sectors which correspond to individual directions. For example, when direction no. 1 is located in sector no. 3, the EANN input corresponding to the direction is fed with the value equal to 3/8 where 8 means the number of sectors.

Additional information provided to the FFEANN is information about a recent vehicle maneuver and the time elapsed since performing the maneuver. The NCAS assumes six vehicle maneuvers, i.e., move forward, move back, turn left, turn right, go up, and go down. Each maneuver has a number assigned, and as before, the network receives the numbers

[4] The number of sectors is a compromise between accuracy and simplicity of surrounding representation, too few sectors mean insufficient information to make proper decisions, whereas too many sectors mean many input neurons, complex architecture of the neural network and, in effect, problems with network training.

Fig. 7 Pseudo-code of `setHeading`

```
setHeading::run(wayPoint,distanceToObstacles)
begin
    if distanceToObstacles < 0 //vehicle inside obstacle
        if obstacles above and below vehicle
            setParameter(getHeading(wayPoint),HEADING);
            positionLastTurn := vPosition;
        end if
    else
        distanceFromLastTurn := getDistance(vPosition,positionLastTurn);
        if distanceFromLastTurn < thresholdDistanceFromLastTurn;
            setParameter(getHeading(wayPoint),HEADING);
            positionLastTurn := vPosition;
        end if
    end if
end
```

Fig. 8 Pseudo-code of `getHeading`

```
    getHeading::run(wayPoint)
    begin
        heading := getDirection(3,wayPoint,vPosition,obstacles);
        angle := getAngle(vHeading,heading);
        if angle > thresholdAngle
            heading1 := getDirection(1,vPosition,vHeading,obstacles);
            heading2 := getDirection(2,vPosition,vHeading,obstacles);
            angle1 := getAngle(vHeading,heading1);
            angle2 := getAngle(vHeading,heading2);
            if angle1 < angle2
                heading := heading1;
            else
                heading := heading2;
            end if
        end if
        return heading;
    end
```

Fig. 9 Pseudo-code of `setDepth`

```
     setDepth::run(wayPoint,distanceToObstacles)
     begin
         if distanceToObstacles < 0 //vehicle inside obstacle
             if no obstacles above vehicle
                 setParameter(vDepth/param,DEPTH);
             else
                 setParameter(vDepth*param,DEPTH);
             end if
         else
             if vDepth != wayPoint.depth;
                 //adjust depth to conditions
             end if
         end if
     end
```

Fig. 10 Pseudo-code of neural variant of `getHeading`

```
getHeading::run(wayPoint)
begin
    input[1-6] := getVehicleSurrounding(vPosition,vHeading,obstacles);
    input[7-9] := getDirections(wayPoint,vPosition,vHeading,obstacles);
    input[10] := getDirectionToWayPoint(wayPoint,vPosition,vHeading);
    if EANN is feed-forward
        input[11] := getLastManeuver();
        input[12] := getDurationLastManeuver();
    end if
    output := EANN.run(input);
    if output[1] > output[2] and output[1] > output[3]
        return getDirection(1,vPosition,vHeading,obstacles);
    else if output[2] > output[1] and output[2] > output[3]
        return getDirection(2,vPosition,vHeading,obstacles);
    else
        return getDirection(3,wayPoint,vPosition,obstacles);
    end if
end
```

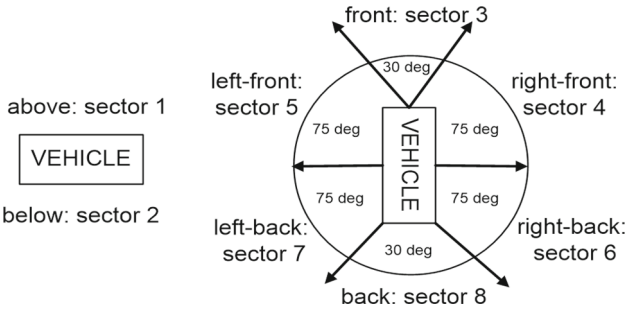


Fig. 11 Sectors around the vehicle

scaled to the range $\langle 0, 1\rangle$. The duration of the maneuver is passed on to the network as a value $1/k$ where k is the number of iterations performed since the start of the maneuver.

In total, the networks were supplied with the following input signals: input 1...6—information about obstacles around the vehicle, each input corresponds to a different sector, input 7...10—information about direction to a next waypoint and directions that correspond to possible vehicle maneuvers, each input is an encoded number of sector, input 11 (only FFEANN)—recent vehicle maneuver, input 12 (only FFEANN)—duration of the recent maneuver.

4 Assembler encoding with evolvable operations

Assembler encoding with evolvable operations (AEEO) originates from a generative neuro-evolutionary method called assembler encoding (AE) (Praczyk and Szymak 2011). In AE, an ANN is represented in the form of a linear program consisting of operations with predefined implementations and data. The task of the program is to create a network definition matrix (NDM) defining a single ANN. To form the program and, in consequence, an ANN, a cooperative coevolutionary algorithm (CCEGA) (Potter 1997; Potter and De Jong 2000) is used. The genetic algorithm generates operations (parameters of operations; as mentioned above, implementations are defined beforehand) and data which combined together form a program. Each program builds NDM which is then transformed into an ANN. To this end, the matrix has to store all the information necessary to construct a network. This information is included in both the size and the individual items of the matrix, scaled always to the range $\langle -1, 1\rangle$. The size of NDM determines a maximum number of neurons in an ANN, whereas individual items of the matrix define weights of interneuron connections, i.e., an item NDM$[i, j]$ determines a link from neuron i to neuron j. Apart from the basic part, NDM also contains additional columns that describe parameters of neurons, e.g., type of neuron (sigmoid, radial, linear, etc.) and bias (see Fig. 12) (Praczyk 2015d).

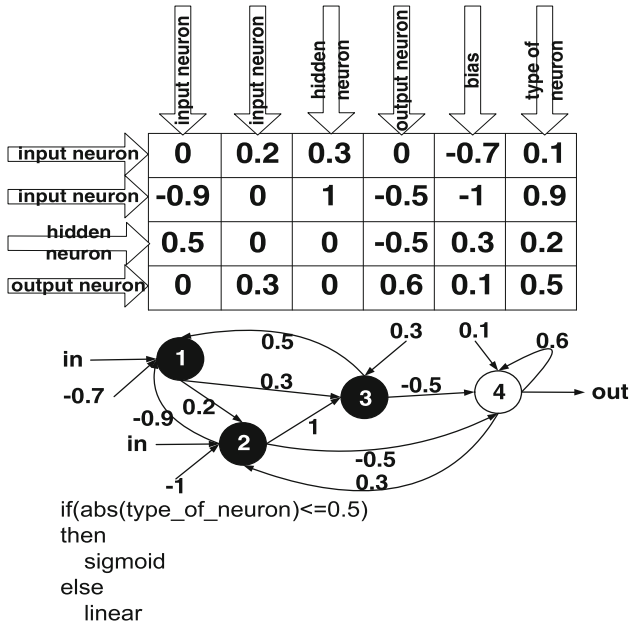


Fig. 12 The way of encoding ANN in the form of network definition matrix (NDM) (Praczyk 2015d)

In AEEO, the operations with predefined implementations are replaced with ANNs-operations with the same task as in the classic variant of the method. Moreover, the data used previously by the operations are completely removed from the programs which, in AEEO, include exclusively the ANNs-operations. The ANNs-operations operate together on one NDM which encodes one final ANN, and once the matrix is completely formed, it is then transformed into the network. To perform the task, ANNs-operations have two inputs, a number of hidden neurons, and three outputs. The inputs indicate an item in NDM updated by the operation (number of row and column), whereas the outputs are used for three different purposes, i.e., to determine a negotiation strength of each ANN-operation, to determine whether the item should be modified or not, and to determine a new value for the item. To indicate which ANN-operation should determine the value of a given item, negotiation outputs of all ANNs-operations are compared. An ANN-operation with the greatest output value is entitled to modify the item. In order for the item to be updated, the second output of the selected ANN-operation is tested. If the output value is greater than an assumed threshold, the item gets a value from the third output of the ANN-operation (see Fig. 13). Otherwise, the item remains intact and the corresponding connection between neurons in the final ANN is not established (Praczyk 2015d).

ANNs-operations evolve according to CCEGA. In CCEGA, each part of a solution evolves in a separate population. To form a complete solution, selected representatives (usually, the best ones) of each population are combined together. Application of this evolutionary scheme in AEEO consists

```
ANN-operation::run(L_{ANN-oper}, T, P_{max}^{AEEO})
begin
    initiateNDM(0)//all items in NDM are set to 0
    for i = 1 , i ≤ P_{max}^{AEEO}
        for j = 1 , j ≤ P_{max}^{AEEO} + 1
            for l = 1 , l ≤ L_{ANN-oper}
                ns_l = FANN_1^l(i,j) //negotiation strength
                //FANN_k^l(i,j) - kth output of lth ANN-operation
                //i and j are inputs to network
            end for
            win = Index(ns) //index of winner ANN-operation
            if FANN_2^{win}(i,j) > T
                NDM_{i,j} = FANN_3^{win}(i,j)
            end if
        end for
    end for
end
```

Fig. 13 Using ANN-operations to form NDM in AEEO (T-threshold, $P_{\max}^{\text{AEEO}}$-maximum number of neurons in resulting ANN, $L_{\text{ANN}-\text{oper}}$-number of ANN-operations) (Praczyk 2015d)

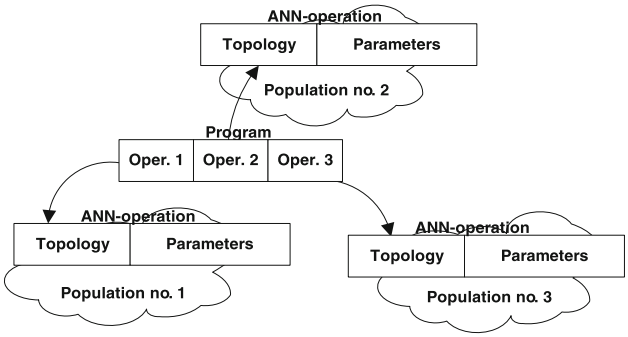


Fig. 14 Evolution of programs in AEEO (Praczyk 2015d)

in evolution of individual ANNs-operations in separate populations. The number of ANNs-operations in a complete program corresponds to the number of populations (see Fig. 14). Each population delegates exactly one representative ANN-operation to each program. During the evolution, programs expand gradually. In the beginning, they contain only one ANN-operation from one existing population. When the evolution stagnates, i.e., lack of progress in fitness of generated solutions is observed over some period, a set of populations containing ANNs-operations are enlarged by one population. This procedure extends all programs by one ANN-operation. Each population can also be replaced with a newly created population. Such a situation takes place when the influence of all ANNs-operations from a given population on fitness of generated solutions is definitely lower than the influence of ANNs-operations from the remaining populations (a population can be replaced when, for example, fitness of a population, measured as the average fitness of all ANNs-operations from the population, is definitely lower than the fitness of the remaining populations) (Praczyk 2015d).

In individual populations, the evolution proceeds according to Canonical GA (Goldberg 1989). At the genotypic level, each ANN-operation is represented in the form of a variable length chromosome consisting of two parts. The first short part defines the topology of an ANN-operation, whereas the second part includes parameters for the network (see Fig. 15). Construction of an ANN-operation proceeds in three phases. First, the topology of an ANN-operation is determined based on the information contained in the first part of the chromosome. To this end, binary values from this part are directly copied into NDM, and a single bit corresponds to a single item in the matrix. When the number of bits is insufficient to completely fill in the matrix, the whole sequence of bits is used again. In the following phase, the parameters from the second part of the chromosome are successively introduced into NDM. In this case, only items equal to one are modified. The remaining items, i.e., items equal to zero, remain intact. As before, transfer of the parameters is performed in a loop until all the elements in NDM have a value assigned. In the last phase, NDM is transformed into an ANN-operation (Praczyk 2015d).

Table 1 Parameters of PFCAS (thresholdDistanceFromLastTurn1—new obstacle is detected, thresholdDistanceFromLastTurn2—vehicle close to obstacles, thresholdDistanceFromLastTurn3—vehicle away from obstacles)

Parameter	Value
Threshold (Fig. 4)	5m
Speed of BAUV away from obstacles	1 m/s or 2 m/s
SPEED_CLOSE_OBSTACLES (Fig. 6)	0.5 m/s
ThresholdDistanceFromLastTurn1 (Fig. 7)	0 m
ThresholdDistanceFromLastTurn2 (Fig. 7)	1 m
ThresholdDistanceFromLastTurn3 (Fig. 7)	10 m
ThresholdAngle (Fig. 8)	40 deg
Param (Fig. 9)	2

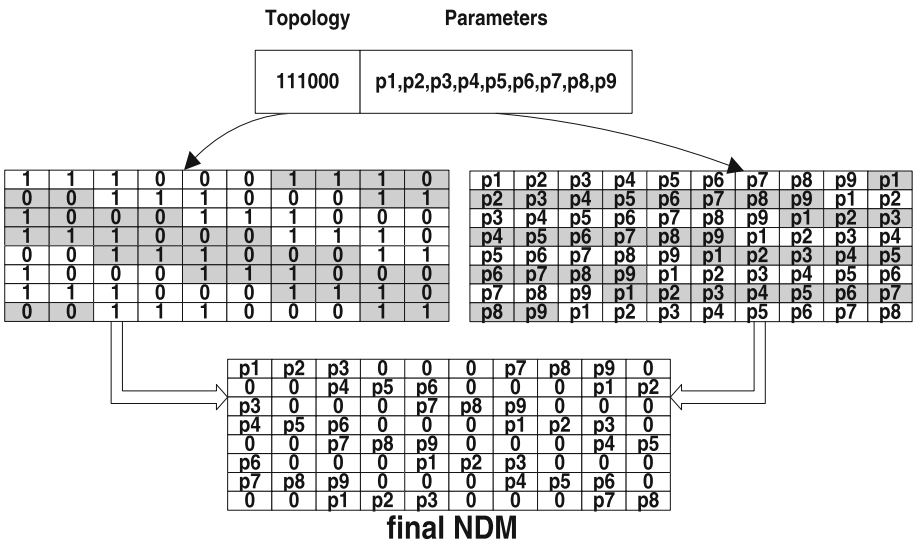


Fig. 15 The method of building NDM representation of ANN-operation (Praczyk 2015d)

5 Experiments

The experiments took place in simulation conditions in an environment of size: length = 100 m, width = 100 m, depth = 20 m—see Fig. 2. In the experiments, the task of the virtual BAUV, whose simulation model is outlined in Sect. 2, was to move from a starting point to a goal point located on two opposite sides of a mountainous area in the middle of the environment. To control the vehicle, APFCAS and PFCAS–NCAS were applied whose key parameters are given in Table 1.

In the first phase of the experiments, the AEEO was run many times to evolve different variants of the NCAS, both feed-forward and recurrent. To evaluate EANNs produced in the evolutionary process, fifty "learning" scenarios were applied which differed in the starting point, initial heading and speed of the vehicle, and in the goal point. Thirty of the scenarios were already used in the experiments reported in Praczyk (2015a) to evolve the APFCAS, which is the point of reference for the NCAS. In all thirty scenarios as well as in the remaining twenty, the APFCAS successfully led the vehicle to the goal point in an assumed time.

Each evolved EANN was tested maximally in all fifty scenarios, the tests were continued only up to the first failure which took place when the vehicle could not reach the goal point in a maximum number of iterations $I^i_{\max}$ where i is the number of the scenario. The value of $I^i_{\max}$ was twice the number of iterations which were needed by the APFCAS to lead the vehicle to the goal point.

The total evaluation of each EANN was a sum of evaluations obtained in scenarios in which the network was tested. The evaluation in a single scenario was different for success and failure in the scenario. Regardless of the scenario result, each emerging on the surface was penalized; this way, the EANNs were encouraged to operate underwater. In the case of the success in the scenario, the fewer iterations were necessary to reach the goal the higher, the evaluation was. The failure meant the evaluation that was the higher, the shorter the distance to the goal at the end of the scenario was. Formally, evaluation function applied in the learning phase can be defined as follows:

$$f(\text{EANN}) = \sum_{i=0}^{n} f_i \tag{2}$$

$$f^l_i = \begin{cases} (I^i_{\max} - I_{\text{surface}})/I^i_{\max} + 1/D_{\text{goal}}, & \text{failure in } i\text{th scenario} \\ (I^i_{\max} + (I^i_{\max} - I_{\text{surface}}) + (I^i_{\max} - I_{\text{goal}}))/I^i_{\max}, & \text{success in } i\text{th scenario} \\ 0, & \text{failure in previous scenario} \end{cases} \tag{3}$$

where

f^l_i, evaluation received in ith learning scenario
I_{surface}, number of iterations on surface
D_{goal}, distance to goal point at end of scenario
I_{goal}, number of iterations necessary to reach goal
n, number of learning scenarios: $n = 50$.

Table 2 Parameters of evolutionary process

Parameter	Value
Max number of evolutionary generations	60,000
Probability of crossover	0.7
Probability of cut-splice	0.1
Probability of mutation	0.06
Size of tournament	1
Number of elite individuals	1
Number of subpopulations (ANN-operations)	Min = 1, max = 5
Size of subpopulations	50
No. of integer genes in chromosomes	Min = 7, max = 20 (varied length chromosomes)
Type of neurons in ANN-operations	Sigmoid, radial, linear, sinusoid, cosinusoid
Hidden neurons in ANN-operations	4
Type of neurons in EANNs	Sigmoid, radial, linear, sinusoid, cosinusoid
Hidden neurons in EANNs	8

The learning phase of the experiments with the parameters specified in Table 2[5] was continued up to the point when a number of successful FFEANNs and REANNs were evolved, that is, the networks which successfully led the vehicle to the goal point in all the learning scenarios. The purpose was to produce neural solutions comparable to the APFCAS, that is, which were prepared in the same conditions and with the same effect as their algorithmic counterpart.

The learning phase showed, generally, that evolving an effective EANN is a highly challenging task. In order to achieve a single fully successful EANN, many AEEO runs were necessary, and statistically, it appeared that to produce ten successful networks, more than two hundred evolutionary runs were required. Moreover, it also turned out that effective feed-forward EANNs are much harder to evolve than recurrent ones. In order to produce a successful FFEANN, the AEEO needed about twice as many runs as in the case of REANNs. Since the complexity of the networks could not be the cause, REANNs had generally more parameters to evolve than their feed-forward counterparts and, in effect, they were harder to learn so, it seems that the only sensible explanation is insufficient/redundant or inappropriate input information provided to FFEANNs. The proof of that may be the most efficient feed-forward network which does not include an input neuron which supplies the network with the information about recent vehicle maneuver. As it appeared, in order to effectively control the vehicle, information about the duration of the recent maneuver is sufficient. It reflects the state of the vehicle and its readiness to changing the maneuver, and the BAUV simply has certain inertia which has to be taken into account when controlling the vehicle.

[5] Parameters of evolutionary process given in Table 2 and applied in the experiments are the most successful parameters applied in experiments reported in Praczyk (2015b).

The learning phase also showed that the sector information provided to the EANNs should take the form of scaled distance to an obstacle, binary information about the existence of an obstacle or its absence at some distance from the vehicle turned out to be simply too weak in order to efficiently avoid collisions.

The next phase of the experiments was devoted to comparisons between the APFCAS and the selected EANNs. The experiments took place in a modified underwater environment in relation to the previous phase. The modification consisted in a slight displacement of mountain obstacles and changing their height. In this phase, the next twenty-two generalization scenarios were applied. As before, they differed in the starting point, initial heading and speed of the vehicle, and in the goal point. This time, evaluation of the APFCAS and EANNs was performed based on all twenty-two scenarios, and even in the case of failure, the evaluation was continued up to the last scenario.

In order to compare all the collision avoidance solutions, two criteria were used. The first, more important criterion is the number of successful scenarios, whereas the second criterion is the total number of iterations in all the successful scenarios. Results achieved in that phase of the experiments are given in the following figures.

Figure 16 presents results of all compared solutions in all twenty-two generalization scenarios, and the NCAS is represented in the figure by networks that were the most effective in the second phase, one FFEANN and one REANN (NDMs of both networks are presented in Fig. 26 in "Appendix"). The figure shows that both the APFCAS and the FFEANN failed in three scenarios (negative values indicate failure in scenario), and they were unable to lead the vehicle to the goal point in the maximum number of iterations which in the generalization phase was very high, and for each scenario amounted to 10,000 iterations.

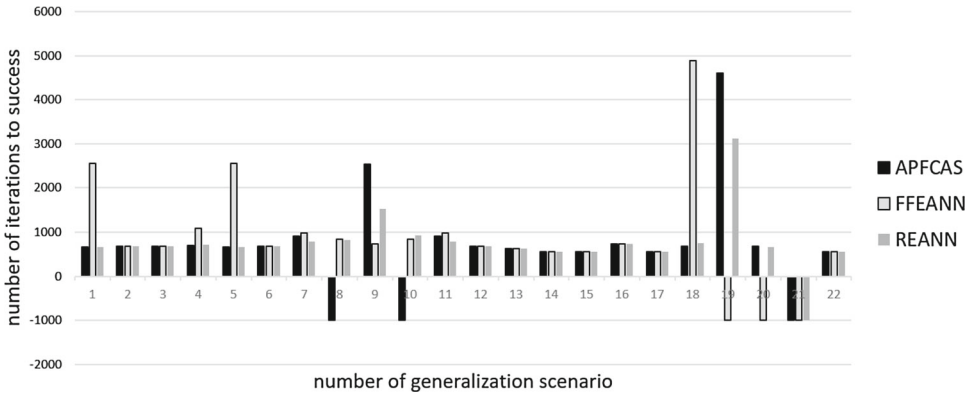


Fig. 16 Number of iterations necessary to lead vehicle to goal point. Negative value means lack of success in scenario

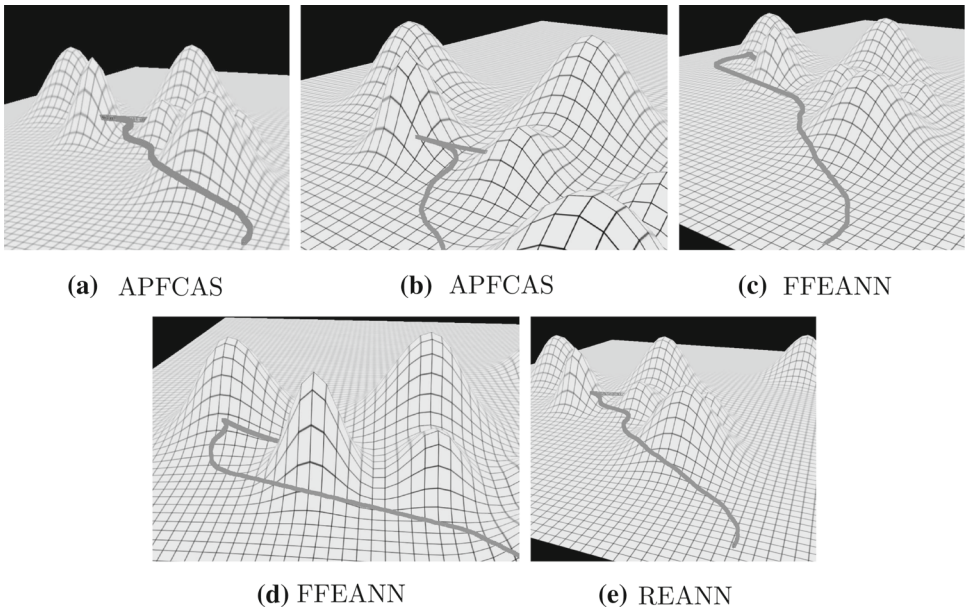


Fig. 17 Paths of the vehicle in generalization scenario no. 21. All methods failed in the scenario

In the case of REANN, only one scenario did not end in success, and the same scenario appeared, however, to be equally challenging for the remaining methods which also did not cope with it. Behavior of the vehicle in that scenario is depicted in Fig. 17 which shows that regardless of collision avoidance method the reason why the vehicle could not reach the goal was the same. As it appeared, the vehicle was trapped between two underwater hills. When it hit the first hill, it started to withdraw, resulting in it hitting the hill at its rear, which, in turn, forced the vehicle to move forward with the effect of collision with the first hill, and so forth. In addition to scenario no. 21, a similar situation also happened in other unsuccessful scenarios, for example, in scenario no 8 depicted in Fig. 18 in which the vehicle controlled by the APFCAS was also trapped between two obstacles.

Such behavior of the vehicle is due to the `moveBack` function (see Fig. 4) which, regardless of the method, moves the vehicle in the opposite direction to the previous maneuver when a collision is detected. The move is continued up to the point when a minimum distance to an obstacle is achieved which guarantees a safe collision-free maneuver. Unfortunately, as it turned out, such an after-collision vehicle strategy appeared to be ineffective and needs correction.

In scenarios successful for all the methods, the REANN was again the most effective. In the majority of them, it needed the least iterations to reach the goal point. When comparing the two remaining methods, Fig. 16 shows that the APFCAS outperforms the FFEANN, and there are a number of scenarios in which the latter method had to resort to surfacing the vehicle which always prolonged the way (see Fig. 19). In spite of the fact that surfacing the vehicle was penalized during the learning phase, this strategy of collision avoidance was in some circumstances applied by all the methods which is depicted in Figs. 20, 21, and 22.

When comparing decisions taken by the methods during the move to the goal point, it is necessary to state that they are often almost the same or very similar as shown, for example, in Figs. 23, 24 and 25. Such a result proves the correctness and effectiveness of solutions applied in the APFCAS in spite

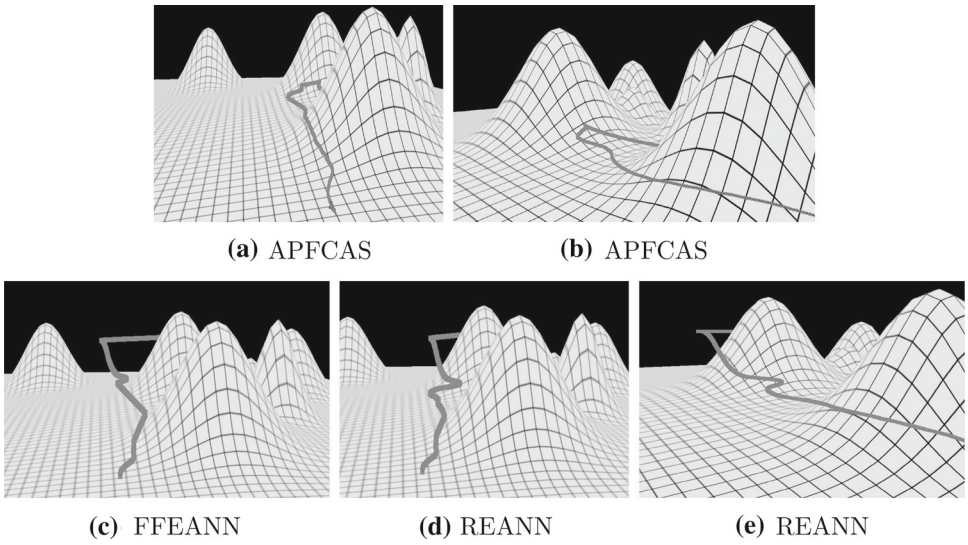


Fig. 18 Paths of vehicle in generalization scenario no. 8. Both EANNs were successful in scenario, whereas APFCAS failed because of entrapment between two obstacles

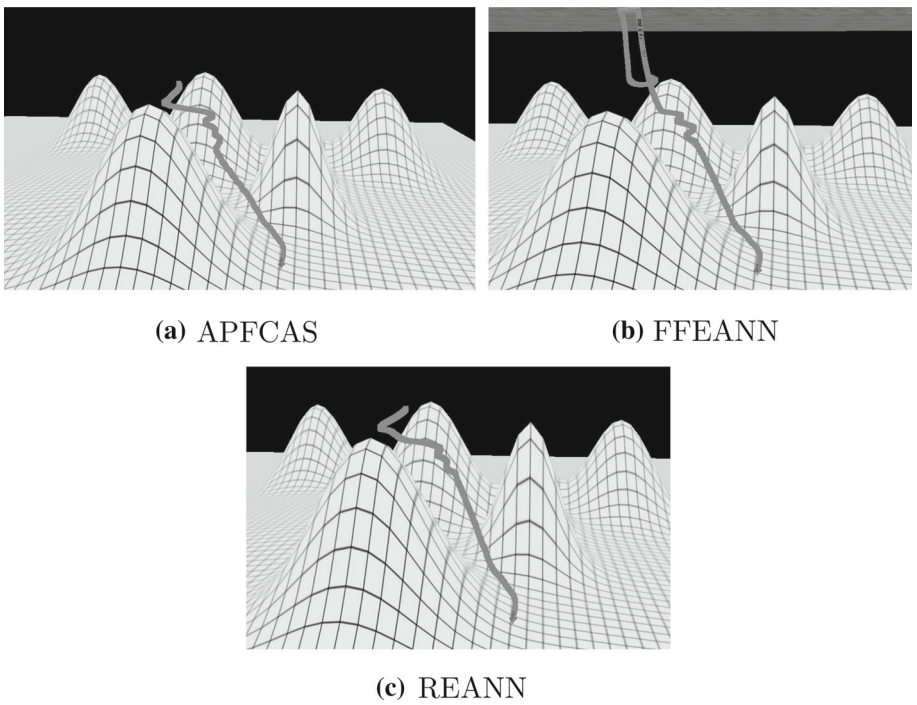
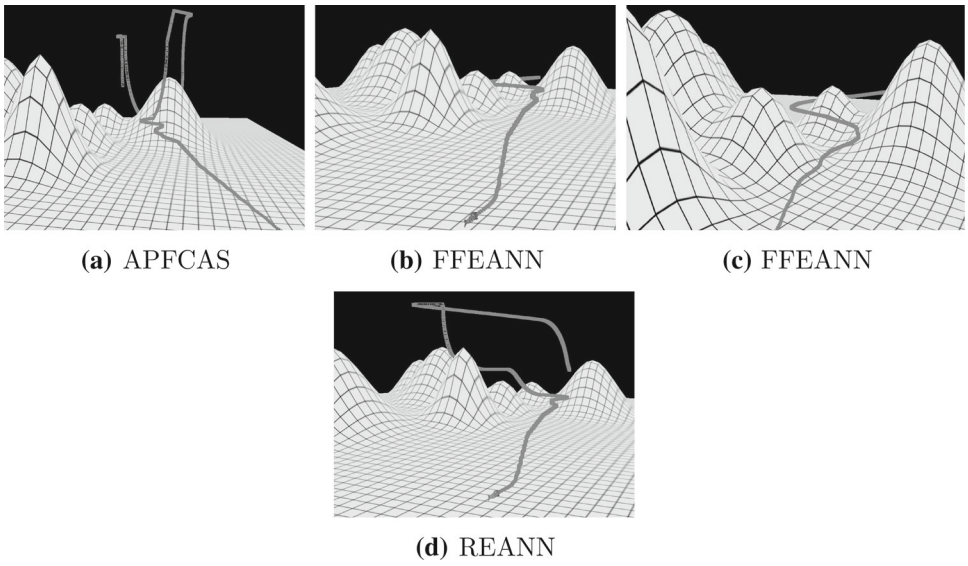


Fig. 19 Paths of vehicle in generalization scenario no. 4, surfacing the vehicle by FFEANN

of the fact that they were designed by a human and only tuned by means of evolutionary techniques.

Figure 23 also displays differences between APFCAS and REANN. The latter method steadily leads the vehicle toward the obstacle and avoids it by the maneuver to the right. There are no additional exploring moves observable in front of the obstacle which means that the collision-free maneuver was taken based on poor information about the surrounding world. The vehicle is equipped with a narrow-sighted echo-sounder which limits the ability of the vehicle to detect obstacles to a narrow strip directly in front of it. In order to acquire information about obstacles outside the strip and, in effect, to take a better decision, the vehicle has to perform exploring maneuvers right and left which is noticeable in the case of REANN. Before the final obstacle avoidance maneuver, the network gathers information about the world in front of the vehicle, and the maneuver is almost identical to the APFCAS one, with only a slight difference relating to the distance to the avoided obstacle. The APFCAS leads the vehicle at a greater distance from the obstacle than the REANN which seems to result from weaker information about the surrounding world. The consequence of this is a slightly longer way of the vehicle, in the case of APFCAS, which is confirmed by the detailed simulation results.

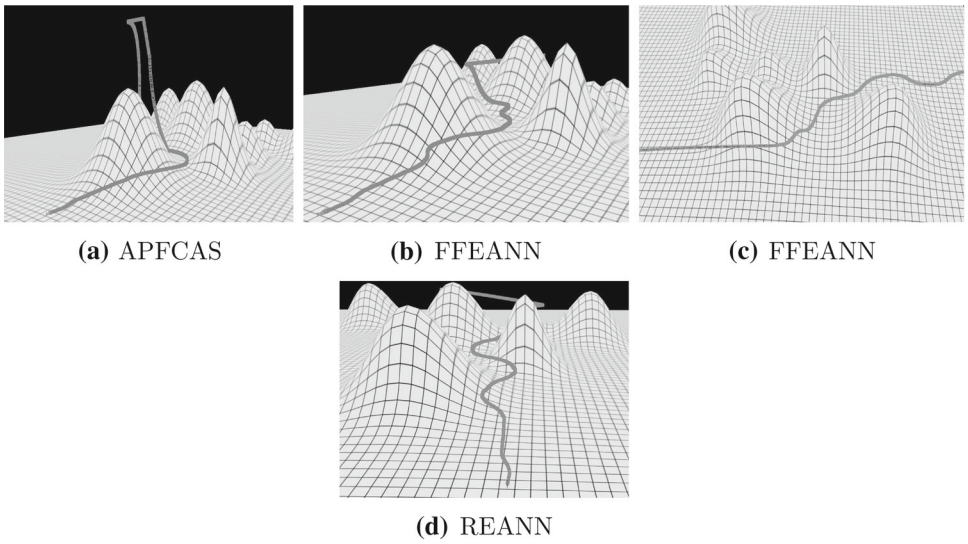
Analysis of paths produced by the FFEANN-controlled vehicle does not provide a clear answer to why this solu-



(a) APFCAS **(b)** FFEANN **(c)** FFEANN

(d) REANN

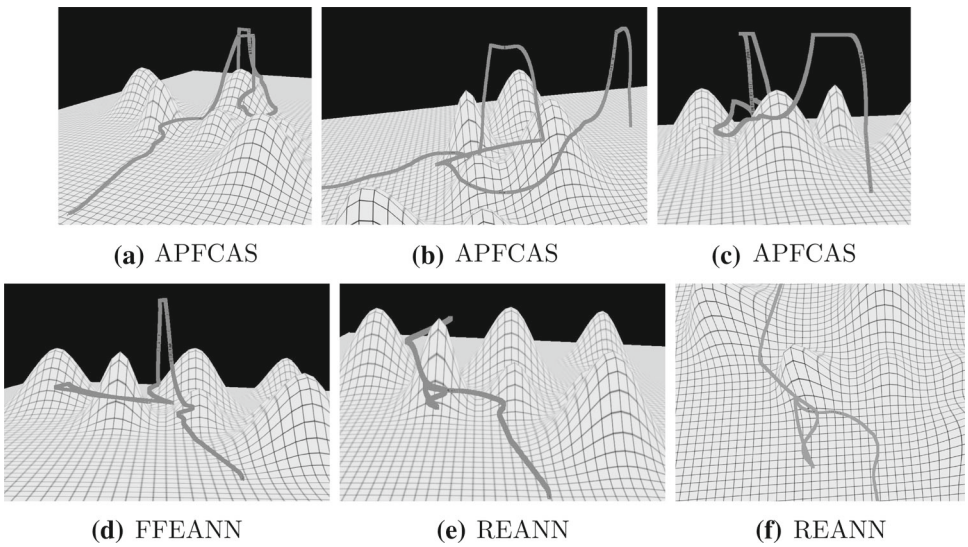
Fig. 20 Paths of vehicle in generalization scenario no. 9, surfacing the vehicle by APFCAS and REANN, in the case of neural network the surfacing is gradual



(a) APFCAS **(b)** FFEANN **(c)** FFEANN

(d) REANN

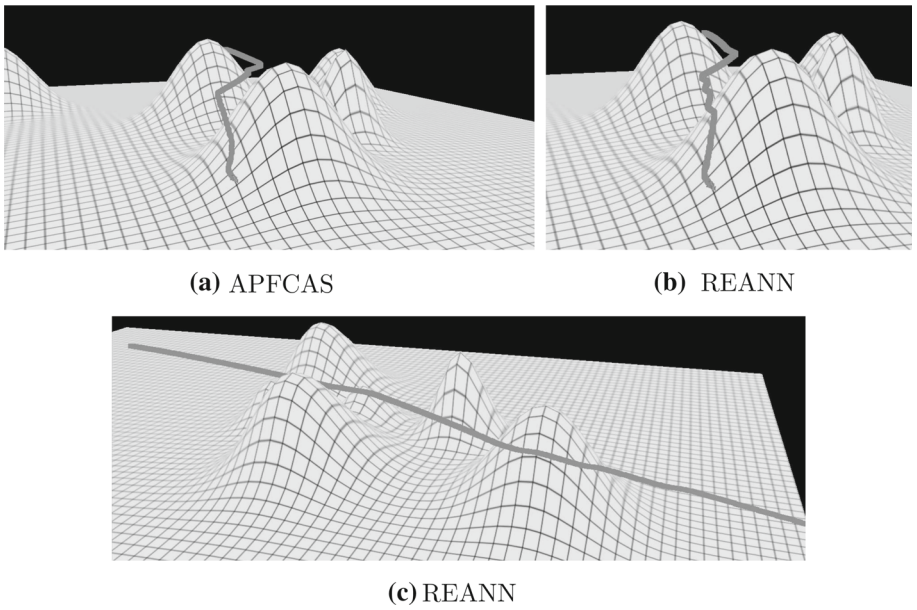
Fig. 21 Paths of vehicle in generalization scenario no. 10, surfacing the vehicle by APFCAS



(a) APFCAS **(b)** APFCAS **(c)** APFCAS

(d) FFEANN **(e)** REANN **(f)** REANN

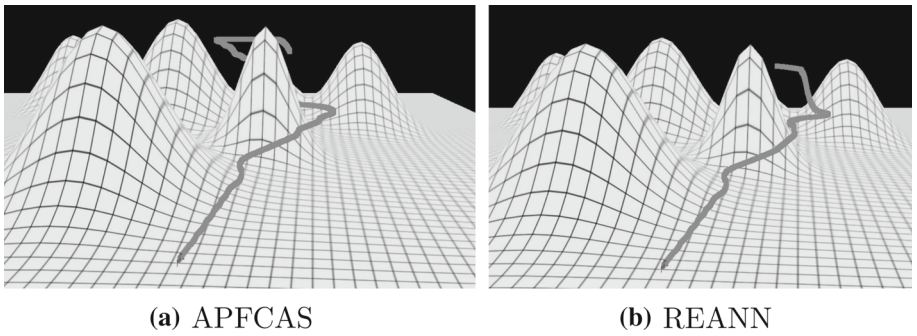
Fig. 22 Paths of vehicle in generalization scenario no. 19, surfacing the vehicle by APFCAS and FFEANN, neural network failed in the scenario



(a) APFCAS **(b)** REANN

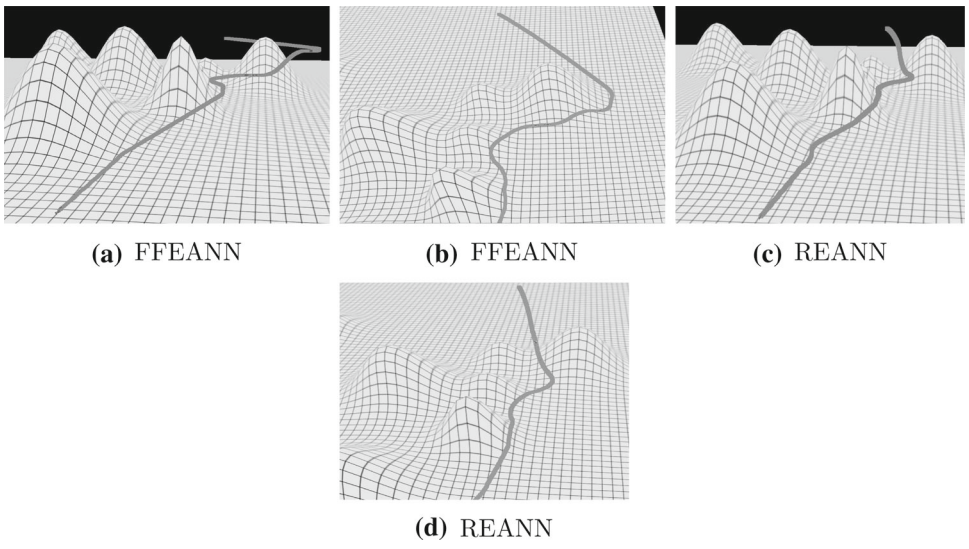
(c) REANN

Fig. 23 Paths of vehicle in generalization scenario no. 2



(a) APFCAS **(b)** REANN

Fig. 24 Paths of vehicle in generalization scenario no. 7



(a) FFEANN **(b)** FFEANN **(c)** REANN

(d) REANN

Fig. 25 Paths of vehicle in generalization scenario no. 11

tion appeared to be less effective than the remaining ones. Some maneuvers and the conditions in which they were run may suggest greater inertia of FFEANN compared to its rival methods. For example, Fig. 25 shows that the first turn to the right, whose purpose was to avoid the obstacle on the left-hand side of the vehicle, was performed closer to the obstacle and later in time than in the case of the REANN. The consequence of this is a less smooth path of the vehicle and in effect a longer route. A similar situation is visible in Fig. 21. In this case, the second turn to the left is performed too late compared to the same maneuver of the REANN. The effect is a change of vehicle depth and a further turn to the left which, as previously, prolongs the vehicle route.

6 Summary

The paper presents the application of evolutionary neural networks, both feed-forward and recurrent, to collision avoidance problem in an underwater environment. Neural networks evolved by means of a neuro-evolutionary method called assembler encoding with evolvable operations were used to control a biomimetic autonomous underwater vehicle in the close proximity of obstacles. An algorithmic collision avoidance system was applied as a point of reference for neural solutions. All the comparison tests were performed in a simulation underwater environment of a mountainous nature.

All the experiments revealed that recurrent networks outperform feed-forward versions and the algorithmic approach. As it turned out, the vehicle controlled by a recurrent network successfully, without collision, performed almost all testing scenarios. Moreover, most vehicle paths produced by that network were shorter than those obtained with the use of the remaining tested solutions.

The experiments also showed some drawbacks of the entire path following and collision avoidance system which need corrective actions. In the system, a separate component is responsible for control of the vehicle in the event of a collision. When a collision is detected, it takes over control from the collision avoidance component and its task is to bring the vehicle to a state in which a safe collision-free maneuver is possible. To this end, the vehicle performs the reverse maneuver to a previous maneuver; for example, when the recent maneuver before collision is "go ahead," the reverse maneuver is "go back." The duration of the reverse maneuver is constant in all cases regardless of the situation. Unfortunately, the experiments showed that this after-collision strategy is ineffective when obstacles are close to each other. In this case, the vehicle moves back and forth between the obstacles, always being in collision with one of them.

The other functionality which needs improvement is the change of vehicle depth. One situation in which the vehicle changes depth is with a high risk of collision for a longer period. In this case, the depth is adjusted to the current situation; the vehicle goes up when obstacles are below it, and down otherwise. The magnitude of a single depth change maneuver is a relatively small constant value determined in the evolutionary way. The assumption in this paper was that, in the case of persistent risk of collision, the vehicle should change depth gradually. However, as the experiments showed, that value may be too high. During the tests, a quite often situation was the emergence of the vehicle to the surface, even though a slight change of depth was enough to reduce the collision risk. Since one of the assumptions of the system is to keep the vehicle underwater whenever possible, the change depth maneuver has to be redesigned.

Despite all the aforementioned shortcomings, it seems that as a whole the system proved its effectiveness. It was verified in different conditions which often were highly demanding and rather difficult to encounter in the real world. Therefore, it seems that the system, after some improvements, can be successfully applied on the real BAUV.

Acknowledgements The paper is supported by the Project No. DOBR-BIO4/033/13015/2013, entitled "Autonomous underwater vehicles with silent undulating propulsion for underwater reconnaissance" financed by Polish National Center of Research and Development. Further development of the construction technology of biomimetic autonomous underwater vehicles whose the collision avoidance part is presented in the paper is intended to take place within the European Defense Agency SABUVIS project.

Compliance with ethical standards

Conflict of interest The authors declare that they have no competing interests.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

Appendix: Matrices representing neural networks that were the most effective in the second phase of experiments

See Fig. 26.

0	0	0	0	0	0.289	0	0	0	0.316	-0.927	0.326	0	0.334	0	0.339	0	0	0	0	0	0.340
0	0	0	-1.37	0.020	0.929	0.0888	-0.992	0	0	-0.223	0.617	0.914	-1.356	0	0	0	0	0	0.017	0	0
-2.132	-0.995	0.029	0.715	0.857	0.716	-0.987	-0.629	-0.108	0	-0.060	-0.923	-0.581	-0.395	-0.996	0.118	0.995	0.515	-1.788	0.458	-1.759	-0.546
-1.76	0.674	0.942	0	-1.568	0	-0.979	-0.529	0.894	0.457	-0.459	0.896	0	-0.097	0.212	-0.311	0	0	0	0	0.239	-0.609
-3.071	-0.665	0.691	-0.277	0	-2.367	0	0	0	0	0	0	0	0	-2.597	0.568	0	-2.591	0	0	-2.483	-0.030
-3.263	0	0	0	0	0	0	-3.089	0	0	0	0	0	-2.979	0	0	-3.397	0	-3.431	-2.876	0	0
-3.698	0	0	0	0	-3.997	0	0	0	0	0	-4.092	0	0	0	0	0	-4.174	0	0	0	0
0	-3.926	0	-3.92	0	0	0	0	0	0	0	0	0	-3.817	0	-3.802	0	0	0	0	0	0
-4.558	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-4.345	0
-6.28	0	0	0	0	0	0	0	0	0	0	0	-6.039	0	0	0	0	0	-5.987	0	0	0
-5.805	0	0	0	0	0	0	0	-5.639	0	-5.663	0	0	0	0	0	0	0	0	0	-5.811	0
-7.273	0	-7.042	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-7.22	0	0	-0.915	-0.902	-7.377	-0.604	-7.34	0	0	0	-0.432	0	0	0	0	0	-7.153	0	-7.116	0	0
-7.94	0	0.262	-7.938	-0.996	0.76	0.491	-8.012	0	0	0	0.852	0.845	0.83	0.992	-8.15	-0.684	0.704	0.63	0.986	0.144	-1
0.992	-0.939	-8.744	-0.998	-8.776	0.703	0.993	-8.076	-0.24	-8.049	-0.482	0.995	0.793	-7.998	-0.982	-0.691	-8.93	0	0.997	-0.047	-0.379	-0.992
-8.766	-0.946	-0.87	-9.504	0	0	-8.509	-0.941	0	0	0	0	-8.466	0	0	0.908	-0.392	0	0	0	-0.915	-0.656
-10.391	0	0	0	0	0	0	0	0	0	-10.188	0	-10.184	0	-10.178	0	0	0	0	-9.082	0	0
-9.868	0	-9.645	0	0	0	0	0	0	-10.72	0	0	0	0	-9.737	0	0	0	0	0	-9.812	-10.585
0	0	0	0	0	0	0	0	0	0	-11.143	0	0	0	0	0	0	0	0	0	0	0

(a) NDM of FFEANN

0	0	0.787	0	0	0	0.658	0	0	0	0.678	0	0	0	0.708	0	0	0	0.744	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.999	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-0.939	-0.889	-0.267	0.526	0.981	0.805	0.11	-0.655	-0.999	-0.7	0.049	0.767	0.991	0.577	-0.207	-0.859	-0.958	-0.44	0.361	0.93
-0.939	-0.889	-0.267	0.526	0.981	0.805	0.11	-0.655	-0.999	-0.7	0.049	0.767	0.991	0.577	-0.207	-0.859	-0.958	-0.44	0.361	0.93
-0.939	-0.889	-0.267	0.526	0.981	0.805	0.11	-0.655	-0.999	-0.7	0.049	0.767	0.991	0.577	-0.207	-0.859	-0.958	-0.44	0.361	0.93
-0.939	-0.889	-0.267	0.526	0.981	0.805	0.11	-0.655	-0.999	-0.7	0.049	0.767	0.991	0.577	-0.207	-0.859	-0.958	-0.44	0.361	0.93
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(b) NDM of REANN

Fig. 26 NDMs of neural networks used in the generalization tests

References

Borenstein J, Koren Y (1991) The vector field histogram—fast obstacle avoidance for mobile robots. IEEE Trans Robot Autom 7(3):278–288

Campbell S, Naeem W, Irwin GW (2012) A review on improving the autonomy of unmanned surface vehicles through intelligent collision avoidance maneuvers. Annu Rev Control 36:267283

Chang C, Gaudiano P (1998) Application of biological learning theories to mobile robot avoidance and approach behaviors. J Complex Syst 1:79–114

Dechter R, Pearl J (1985) Generalized best-first search strategies and the optimality of A*. J ACM 32(3):505536

Eriksen BOH (2015) Horizontal collision avoidance for autonomous underwater vehicles, Masters Thesis, Department of Engineering Cybernetics, Norwegian University of Science and Technology

Fox D, Thrun S, Burgard W (1997) The dynamic window approach to collision avoidance. IEEE Robot Autom Mag 4(1):2333

Goldberg DE (1989) Genetic algorithms in search, optimization and machine learning. Addison Wesley, Reading

Guerrero-Gonzlez A, Garca-Crdova F, Gilabert J (2011) A biologically inspired neural network for navigation with obstacle avoidance in autonomous underwater and surface vehicles, OCEANS, 2011 IEEE-Spain. IEEE

Harris CJ, Moore CG (1989) Intelligent identification and control for autonomous guided vehicles using adaptive fuzzy-based algorithms. Eng Appl Artif Intell 2(4):267–285

Hart PE, Nilsson NJ, Raphael B (1968) A formal basis for the heuristic determination of minimum cost paths. IEEE Trans Syst Sci Cybern SSC-4 2:100–107

Im K-Y, Oh S-Y (2000) An extended virtual force field based behavioral fusion with neural networks and evolutionary programming for mobile robot navigation. Evol Comput 2:1238–1244

Im K-Y, Oh SY, Han S-J (2002) Evolving a modular neural network-based behavioral fusion using extended VFF and environment classification for mobile robot navigation. Evol Comput 6(4):413–419

Kanakakis V, Tsourveloudis N (2007) Evolutionary path planning and navigation of autonomous underwater vehicles. In: Mediterranean conference on control and automation. MED 07, 16

Kanakakis V, Valavanis K, Tsourveloudis N (2004) Fuzzy-logic based navigation of underwater vehicles. J Intell Robot Syst 40:4588

Kim S-G, Kim Y-G (2009) An autonomous navigation system for unmanned underwater vehicle. In: Inzartsev AV (ed) An autonomous navigation system for unmanned underwater vehicle, underwater vehicles. InTech. ISBN: 978-953-7619-49-7. http://www.intechopen.com/books/underwater-vehicles/an-autonomous-navigation-system-for-unmanned-underwater-vehicle

Kwon K-Y, Cho J, Joh J (2005) Collision avoidance of moving obstacles for underwater robots. Syst Cybern Inform 4(5):86–91

Naeem W, Irwin GW, Yang A (2012) COLREGs-based collision avoidance strategies for unmanned surface vehicles. Mechatronics 22:669–678

Nikolos IK, Zografos ES, Brintaki AN (2007) UAV path planning using evolutionary algorithm. Stud Comput Intell: SCI 70:77111

Potter M (1997) The Design and Analysis of a Computational Model of Cooperative Coevolution, PhD thesis, George Mason University, Fairfax, Virginia

Potter MA, De Jong KA (2000) Cooperative coevolution: an architecture for evolving coadapted subcomponents. Evol Comput 8(1):1–29

Praczyk T (2015a) Using genetic algorithms for optimizing algorithmic control system of biomimetic underwater vehicle. Comput Methods Sci Technol: CMST 21(4):251–260

Praczyk T (2015b) Neural anticollision system for autonomous surface vehicle. Neurocomputing 149(Part B 3):559–572

Praczyk T (2015c) Assembler encoding with evolvable operations. Comput Methods Sci Technol: CMST 21(3):123–139
Praczyk T (2015d) Using evolutionary neural networks to predict spatial orientation of a ship. Neurocomputing 166:229–243
Praczyk T (2016) A quick algorithm for planning a path for biomimetic autonomous underwater vehicle. Sci J Marit Univ Szczec 45(117):23–28
Praczyk T, Szymak P (2011) Decision system for a team of autonomous underwater vehicles—preliminary report. Neurocomputing 74(17):3323–3334
Smierzchalski R (1999) Evolutionary trajectory planning of ships in navigating traffic areas. J Mar Sci Technol 4:1–6
Smierzchalski R, Michalewicz Z (2005) Path planning in dynamic environments. Stud Comput Intell: SCI 8:135153
Szlapczynski R, Szlapczynska J (2012) On evolutionary computing in multi-ship trajectory planning. Appl Intell 37:155174
Tan CS (2006) A collision avoidance system for autonomous underwater vehicles, PhD Thesis, The University of Plymouth
Wilson PA, Harris CJ, Hong X (2003) A line of sight counteraction navigation algorithm for ship encounter collision avoidance. J Navig 56(1):111–121
Yang A, Niu Q, Shao W, Li K, Irwin GW (2010) An efficient algorithm for grid-based robotic path planning based on priority sorting of direction vectors, In: Proceedings of international conference on life system modelling and simulation, LSMS 2010, and international conference on intelligent computing for sustainable energy and environment, ICSEE 2010. Wuxi, China
Young-il L, Yong-Gi K, Kohout L (2004) An intelligent collision avoidance system for AUVs using fuzzy relational products. Inf Sci 158:209–232
Zeng B, Yang Y, Xu Y (2009) Mobile robot navigation in unknown dynamic environment based on ant colony algorithm. In: WRI global congress on intelligent systems. GCIS 09, vol 3, p 98102