CMST 21(4) 251-260 (2015) DOI:10.12921/cmst.2015.21.04.009

Using Genetic Algorithms for Optimizing Algorithmic Control System of Biomimetic Underwater Vehicle

Tomasz Praczyk

Institute of Naval Weapon, Polish Naval Academy
81-103 Gdynia, ul. Śmidowicza 69
E-mail: t.praczyk@amw.gdynia.pl

Received: 24 November 2015; revised: 15 December 2015; accepted: 16 December 2015; published online: 29 December 2015

Abstract: Autonomous underwater vehicles are vehicles that are entirely or partly independent of human decisions. In order to obtain operational independence, the vehicles have to be equipped with a specialized control system. The main task of the system is to move the vehicle along a path with collision avoidance. Regardless of the logic embedded in the system, i.e. whether it works as a neural network, fuzzy, expert, or algorithmic system or even as a hybrid of all the mentioned solutions, it is always parameterized and values of the system parameters affect its effectiveness. The paper reports the experiments whose goal was to optimize an algorithmic control system of a biomimetic autonomous underwater vehicle. To this end, three different genetic algorithms were used, i.e. a canonical genetic algorithm, a steady state genetic algorithm and a eugenic algorithm.

Key words: biomimetic underwater vehicle, evolutionary computation

I. INTRODUCTION

Autonomous Underwater Vehicles (AUV), as the name implies, are vehicles which need the ability to operate autonomously or, in other words, independently of a human being. This ability can be used on different levels and in different situations, e.g. during the entire operation of the vehicle – a vehicle completely independent of an operator, or only when a communication system with the operator is broken – a remotely operated vehicle with a function to safely come back to the operator. To act autonomously, not only in emergency situations, each AUV has to be equipped with a specialized software that is able, to at least lead the vehicle along a path fixed by the operator and to avoid collisions with obstacles.

The software with the abilities outlined above was implemented for the Biomimetic Autonomous Underwater Vehicle (BAUV) [3, 9] constructed within the framework of the project no. DOBR-BIO4/033/13015/2013, entitled "Autonomous underwater vehicles with silent undulating propulsion for underwater reconnaissance" financed by the National Center of Research and Development. It was functionally divided into two separate parts, i.e. Low-Level Control System (LLCS) and High-Level Control System (HLCS). LLCS is responsible for executing commands provided by HLCS, i.e. its task is to implement decisions of HLCS by means of BAUV propellers. In other words, LLCS transforms high-level decisions of HLCS, e.g. to move forward, to turn left, to submerge to depth of 15 meters, into low-level decisions for BAUV propellers. To this end, different types of controllers, e.g. PID or fuzzy controllers, are used. In turn, HLCS is responsible for high-level decisions, i.e. decisions regarding direction of movement, turning on/off on-board devices, depth etc. Whereas LLCS performs tasks which have to be performed not only on autonomous vehicles, HLCS is characteristic exclusively for autonomous platforms.

Generally, HLCS can have different types of tasks; however, its basic responsibility is to lead the vehicle along a fixed path and to avoid obstacles encountered during the voyage. To perform this task, different control strategies and

different types of high-level controllers can be used, e.g. a neural network, a fuzzy system, an expert system, an algorithmic system or even a hybrid which can somehow combine all the mentioned solutions.

The current implementation of HLCS is algorithmic with the potential to include into the system other solutions, for example the ones mentioned above. To make the system effective in various collision and collision free situations, it was repeatedly tested, rebuilt and each next variant of the system was manually tuned. However, the manual tuning does not guarantee optimality of HLCS and BAUV behavior. In order to increase effectiveness of the current hand-tuned variant of HLCS, the decision was made to apply optimization techniques. Due to the complexity of BAUV control problem, the genetic algorithms (GA) were selected for that purpose. Their globality and discrete character influenced the decision.

The paper is a report on the experiments with the genetic algorithms used to optimize parameters of algorithmic HLCS. Its further part consists of four main sections: first, a control algorithm used in HLCS is shortly outlined, then, three types of genetic algorithms applied in the experiments are presented, in the next section, the experiments are reported, and finally, a summary is given.

II. THE ALGORITHM OF HLCS

The main idea of HLCS is to perform a mission of BAUV defined by the vehicle operator on a specialized computer application and provided to the vehicle by means of a radiomodem. The mission takes the form of a sequence of operations or orders for BAUV. The operations can determine successive movements of the vehicle but they also can control vehicle devices, e.g. sonars, cameras, and echo sounders. They can be very simple, e.g. stop, move forward, rotate, but also more complex, e.g. move to next waypoint, turn right at 20 deg, immerse 10 meters deep. In the latter case, the operations can be combined with simpler ones.

In normal conditions, i.e. when BAUV moves between successive waypoints with collision avoidance, HLCS performs one operation after another, but there can be situations when the mission has to be interrupted and an emergency action has to be taken. Collision with an obstacle is an example of such a situation. In this case, HLCS stops the operation which is currently run and starts an operation whose task is to lead BAUV to a position in which the mission can be safely resumed.

When navigating between waypoints and avoiding collisions, behavior of BAUV is, in principle, a sequence of operations: move forward, turn right/left and change the depth. The moment of activating specific operations depends on the situation under water and the logic embedded in HLCS. The situation observed by BAUV depends, in turn, on sensors installed on board. It is assumed that the BAUV sensors are used to form an environment map made up of simple elements like spheres and all the decisions on the vehicle movements are made based on the map, not based on direct indications of the sensors. In other words, HLCS "lives" in a virtual world which consists of a set of spheres and all system decisions are made in this world.

The logic of the system, if limited only to navigating between waypoints and avoiding collisions, can be reduced to one operation, say, `GoToWayPoint`. This operation, when possible, leads BAUV directly to a next waypoint, along a straight line, and when impossible, mainly because of obstacles, avoids collisions through changing direction of movement and depth of the vehicle. The choice of a specific collision-avoidance maneuver depends on location of the obstacles with reference to BAUV. When HLCS finds that there is a space for BAUV to directly move toward a next waypoint, the course to the waypoint is calculated and the vehicle continues its voyage according to the fixed course. The depth at which BAUV moves is equal to coordinate Z of the next waypoint. If BAUV is close to the goal waypoint, the operation stops and a next `GoToWayPoint` operation is run.

III. GENETIC ALGORITHMS

All the evolutionary techniques used in the experiments reported in the further part of the paper, regardless of the type, work according to the simple algorithm depicted in Fig. 1. First, a population of individuals is generated, typically at random. Each individual is then evaluated, i.e. a measure of goodness, called fitness, is assigned to it. In the next step, some individuals from the population are selected. The chosen individuals are parents for newborn offspring. To generate the offspring, different genetic operators are used on the parents. The newly generated offspring are then evaluated. The next activity is to replace the parents with the offspring. This way we obtain a new population of individuals. The population, depending on the replacement strategy, contains either exclusively newly generated individuals or both the parents and the children. The new population undergoes the same evolutionary procedure as the previous population, i.e. individuals are selected for reproduction, different operators are applied on selected individuals, offspring are generated, evaluated and finally a next population arises. This procedure is repeated until some stopping criterion is satisfied.

III. 1. Canonical Genetic Algorithm

The Canonical Genetic Algorithm (CGA) (see Fig. 2), like each GA, operates on the level of encoded solutions, i.e. genotypes represented as binary strings. In CGA, different selection schemes can be used to select genotypes for reproduction. The most frequently used schemes are: the proportional, ranking and tournament selection scheme [2, 6]. All the schemes present a selection philosophy according to which better individuals have a greater chance to be selected than the worse ones.

```
evolutionary_algorithm()
{
generate initial population at random;
evaluate population;
do{
     select parental individuals from population
          for reproduction;
     apply genetic operators to selected individuals
          and generate offspring;
     evaluate offspring;
     replace some parents with some offspring;
     }while(termination criterion is not satisfied)
}
```

Fig. 1. Pseudocode of Evolutionary Algorithm [6]

In order to select individuals for reproduction, the tournament selection, which is the only selection scheme used in the experiments presented further, organizes a cycle of tournaments. In each tournament, a number of randomly selected individuals participate. Selecting individuals for the tournament is uniform. Each tournament ends with selecting a winner individual which is better than the remaining individuals taking part in the competition. In order to fill out a newly created population, it is necessary to repeat the tournament many times.

The selected individuals undergo different genetic operators. Two commonly used genetic operators are crossover and mutation [2, 6]. The crossover recombines a genetic material from two parental chromosomes creating an offspring chromosome. The mutation introduces random changes into a chromosome flipping some bits (genes) to an opposite state. In CGA, the key operator is the crossover [2, 6] whereas the mutation usually plays a supporting role.

```
CanonicalGeneticAlgorithm()
{
generate initial population of random binary strings;
evaluate population;
do{
   select parental individuals for reproduction
     based on their fitness;
   generate offspring: apply crossover to parents;
   apply mutation to offspring;
   evaluate offspring;
   replace parents with offspring;
   }while(termination criterion is not satisfied)
}
```

Fig. 2. Pseudocode of Evolutionary Algorithm [6]

Generally, there are three basic forms of the crossover, i.e. the one-point, two-point and uniform crossover [8, 11]. The one-point crossover cuts two parental chromosomes into two segments. The left segment from the first chromosome is attached to the right segment from the second chromosome. This way, the first offspring is produced. The same procedure as above is applied with respect to the remaining unattached segments from both chromosomes. Thus, the second offspring is generated. The two-point crossover works similarly. However, this time, parental chromosomes are cut in two places. Accordingly, instead of two segments, as in the previous case, three separate segments are generated. To produce offspring, the two-point crossover swaps central segments of both chromosomes. The third version of the crossover is the uniform crossover. It exchanges corresponding bits from both chromosomes with the probability 0.5.

The mutation is the next evolutionary operator used in CGA. As mentioned above, the mutation is rather a supporting operator in relation to the crossover. Nevertheless, functioning of CGA and other GAs without this operator is rather impossible. The mutation helps in exploration of unknown regions of a genotype space. The lack of this operator would cause CGA to be not able to produce new, perhaps useful, combinations of bits. The mutation is a simple operator since it randomly flips bits from a chromosome to their opposite state.

III. 2. Steady State GA

Unlike CGA, Steady State GA (SSGA), in the reproduction phase, creates only one offspring which then replaces a single individual from the existing population. The newborn individual is generated from two randomly selected parents. As in CGA, better individuals have a greater chance to be parents than worse individuals. There are at least two different strategies to select an individual to replace. The first of them replaces the worst individual from a population. The next strategy randomly selects individuals to replace. In general, selecting individuals to replace is based on their fitness. In the experiments, the first mentioned replacement strategy was used.

III. 3. Eugenic Algorithm

The last GA used in the experiments is the Eugenic Algorithm (EuA) [1, 4, 5, 7]. The main difference between GAs presented so far and EuA is the way of constructing offspring. To generate offspring, the techniques presented previously always use two parents, i.e. they apply the sexual reproduction. In EuA, a single child is created basing on the information included in the whole population of individuals. Each individual from the population can affect the shape of a newborn individual. However, better individuals have a greater chance to introduce their genetic material into offspring than the worse ones. The pseudocode for EuA is presented in Figure 3.

EuA proceeds as follows. First, a randomly initialized population of binary individuals is generated. Then, a new individual is formed. To this end, EuA uses a temporary population of individuals, which in the beginning is a faithful copy of the original population. Initially, all genes from the new individual are unset. In subsequent steps, the genes are gradually fixed. The most significant gene, i.e. the gene which seems to have the greatest influence on fitness of individuals, is determined from the whole set of unset genes. To estimate the significance of the gene, the absolute difference between mean fitnesses fixed for various alleles (allele

is a value of a gene; in GAs allele is from the set {0,1}) is used [7]:

$$S_g = |\bar{f}(g,1) - \bar{f}(g,0)| \quad (1)$$

where S_g is the significance of the gene g and $\bar{f}(g,a)$ is the mean fitness of individuals whose gene g has the allele a assigned. An unset gene for which the difference S_g is the greatest is considered the most significant and it first gets the allele.

$$g^* = arg\max_{g \in u} S_g \quad (2)$$

In the above equation, g^* denotes the most significant gene while u is the set of unset genes in a newly created individual. To select a value for the gene, the following formula is used [7]:

$$x_{new}[g] = \begin{cases} 1 \text{ if } P(g,1) > P(g,0) \\ 0 \text{ otherwise} \end{cases} \quad (3)$$

$$P(g,a) = \frac{\bar{f}(g,a)}{\sum_{b \in \{'0','1'\}} \bar{f}(g,b)} \quad (4)$$

where $x_{new}[g]$ is a value of g^{th} gene in the newborn individual x_{new}, and $P(g,a)$ is the probability of the allele a to be chosen for the gene g.

In the next step, the chosen gene is removed from the set of unset genes and the process of restricting the temporary population is started. In order to make a decision about restricting the population, EuA estimates the amount of epistasis E, i.e. the level of interdependencies between individual genes. The estimate is then used to calculate the probability of restriction P_R [7]:

$$E = 1 - D_{max} \quad (5)$$

$$D_{max} = \max_{g \in u} |P(g,1) - P(g,0)| \quad (6)$$

$$P_R = E \quad (7)$$

In the case of the restriction, each individual whose value of gene g^* is different from $x[g^*]$ is removed from the temporary population.

```
eugenic_algorithm()
{
t:=0;
//generate initial population of binary strings
//at random
pop:=getRandomPop();
while(termination criterion is not satisfied)
    {
     u:={1,2,...,l};
     rpop:=pop;
     while(!u->isEmpty())
          {
          //select the most significant gene g*
          //out of all genes from u
          g*:=rpop->getTheMostSignificantGene(u);
          //set the gene g* of newborn individual x_new
          x_new[g*]:= rpop->setAllele(g*);
          u->removeGene(g*);
          //restrict rpop when epistasis is high
          //E denotes epistasis while P_R is
          //probability of restriction
          E:= rpop->getEpistasis(u);
          P_R:=E;
          r:=getRandom(0,1);
          if(r<= P_R)
               rpop:=rpop->restrictPopulation(g*,x_new[g*]);
          }
     //replace the worst individual
     //with the newborn individual
     w:= pop->getTheWorstIndividual();
     pop->replaceIndividual(w,x_new);
     t:=t+1;
     }
}
```

Fig. 3. Pseudocode for EuA (in the figure the following notation is used: `a:=b->op()` – means that `a` is a result of operation op() executed for an object b; usually, b is a set which means that op() is an operation over a set of elements) [7]

The process of creating the new individual described above is performed until the set of unset genes is empty. Once the new individual is completely formed it replaces the worst individual from the original population and the process of creating a next new individual starts once again. It is continued until some termination criterion is satisfied.

IV. EXPERIMENTS

IV. 1. Conditions of experiments

In order to optimize HLCS of BAUV, experiments were carried out in which GAs presented in the previous section were used as optimization tools. The experiments were performed in simulation, and to this end a simulation model of BAUV was implemented [10], which was the model of Remotely Operated Vehicle (ROV) "Gluptak" (Fig. 4) with inclusion of oscillation in horizontal plane. The oscillation made it possible to simulate undulating propulsion of BAUV[1].

In order to maximally speed up calculations during evolutionary processing, the model was not constructed in the

[1] At the time of writing the paper a real BAUV does not exist yet. In the project whose one element is reported in the paper, it was assumed that the work on physical construction of BAUV and HLCS has to be continued simultaneously. The purpose is to have HLCS almost ready to work once BAUV to be built. To this end, experiments in simulation have to be done, whose goal is to prepare HLCS possibly the most close to the variant of the system provided for work on real BAUV. To make HLCS easy to adjust to the real vehicle it was parametrized appropriately, the change of parameters will allow adaptation of HLCS to the real BAUV. In order for the adaptation to be quick and trouble-free, parameters determined for ROV mimicking BAUV should be close to the optimal ones.

Fig. 4. ROV "Gluptak" [12]

traditional way, i.e. as a set of differential equations, but 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 in one coherent model [10]. 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 "glued" by means of the specialized software has yet another very important purpose. Namely, it enables quick and easy modeling of almost every underwater vehicle which will be necessary in further stages of the project when physical construction of BAUV is ready. To perform simulations on the real BAUV, not on ROV "Gluptak" mimicking the real vehicle, it will be necessary to record behavior of BAUV for selected maneuvers, to replace the matrices that represent ROV "Gluptak" with matrices that represent BAUV, and finally to gently tune some parameters of the "gluing" software. It is assumed that it will be a considerably easier and faster process than building the model of BAUV according to the traditional approach.

To perform simulations with collision-avoiding BAUV, in addition to the vehicle itself, its vision system, i.e. its sensors, also had to be modeled. Two virtual echo sounders were applied as the vehicle sensors, the first of which looked straight ahead whereas the second one looked down, and each of them was installed in front of BAUV. To simulate operation of an echo sounder, the distance was measured between current position of BAUV and a first point located on the straight line being prolongation of the observation line of the echo sounder.

All the experiments were carried out in virtual simulation environment with obstacles of size: length=100m, width=100m, depth=20m (Fig. 5).

To optimize HLCS, the system was examined in thirty different testing scenarios. The scenarios were prepared manually in such a way that the complexity of the task which HLCS had to perform gradually increased. When preparing the scenarios, HLCS with parameters determined by system designers was used. In the preparation process, BAUV was run many times and each run was evaluated visually by the designers. To evaluate the scenarios, the complexity criterion was applied. The evaluation of each scenario depended on the path which BAUV selected to reach the goal point, the paths with a greater number of collision-avoidance maneuvers were generally considered to be more difficult. Note, however, that HLCS shaped in the evolutionary way might decide in a different way than its "hand-made" counterpart with the effect that scenarios considered by the designers as easy could be difficult, and conversely.

After preparing a number of scenarios, some of them were selected for experiments and ordered according to their complexity. First, BAUV had to get a goal point moving near a single obstacle. In the following scenarios, BAUV had to deal with a growing number of obstacles, and to get to the goal point, it had to do different maneuvers. In the final scenarios BAUV controlled by a "hand-made" HLCS cannot get to the goal point without collision with an obstacle. In such a case, HLCS ran the procedure which moved BAUV backward with respect to the obstacle so the vehicle could continue its voyage.

The scenarios generally differed in the following parameters: starting point, goal point, initial course. Moreover, in order not to prolong each scenario, each of them was parametrized by the maximum number of HLCS decisions which the system could make to succeed in the scenario. This number corresponded to the number of iterations which the

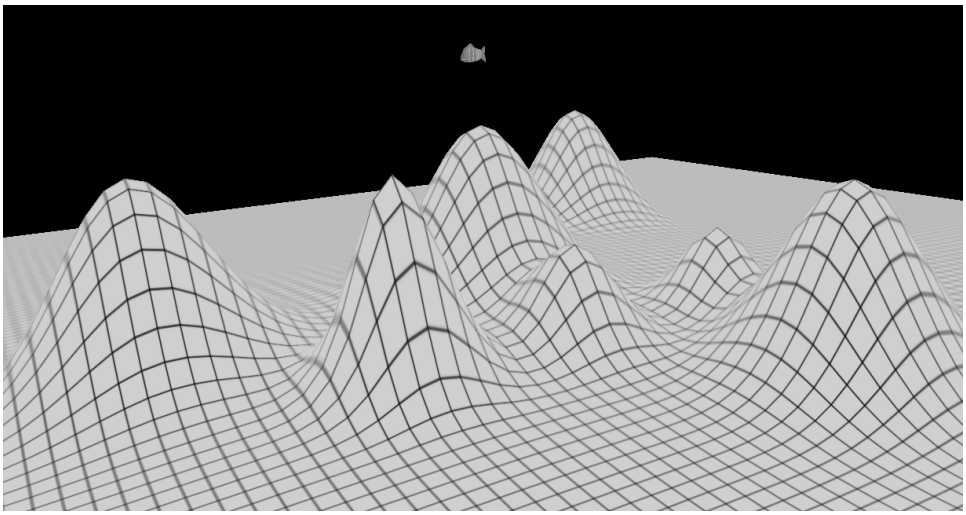


Fig. 5. Environment used in experiments

"hand-made" HLCS needed to lead BAUV to the goal.

To evaluate HLCSs produced during the evolutionary process, the following fitness function was used:

$$f(HLCS) = \sum_{i=0}^{n} f_i \tag{8}$$

$$f_i = \begin{cases} 1 + 1/N_i^{it}, \text{ BAUV reached goal in } i^{th} \text{ scenario} \\ 1/Distance_i(BAUV, goal), \text{ BAUV did not} \\ \qquad \text{reach goal in } i^{th} \text{ scenario} \\ 0, \text{ BAUV did not reach goal in previous scenario} \end{cases} \tag{9}$$

where

n – is the number of testing scenarios (n=30),

f_i – is fitness received by HLCS in i^{th} scenario,

N_i^{it} – is the number of iterations in which BAUV reached the goal point (the number of HLCS decisions) in i^{th} scenario,

$Distance_i(BAUV, goal)$ – is the Euclidean distance between the final position of BAUV in i^{th} scenario and the goal point

Each HLCS was evaluated maximally in thirty scenarios. If the goal point was not reached in a scenario, HLCS was not tested in next scenarios. In consequence, according to (8) and (9), fitness of HLCS is equal to the sum of fitnesses for reaching the goal and fitness received in one scenario in which BAUV did not arrive to the goal in the assumed maximum number of iterations.

In the experiments, the task of each GA was to determine eighteen different parameters of HLCS shortly outlined below:

1. P_1 – is a threshold which decides when BAUV, after the decision to turn, should stop this maneuver and start to move forward, in other words, P_1 is a parameter of a simple course controller which works as follows: once the difference between a desired course and actual course of the vehicle is less than or equal P_1 the vehicle starts to move forward. Such construction of the course controller was dictated by oscillations of BAUV course when the vehicle moves, constructing the course controller based on the average course fixed for some time appeared to be ineffective near obstacles when BAUV has to perform a quick collision avoidance maneuver, and there is no time for average course calculations,
2. P_2 – is the distance between successive corrections of the BAUV course. The vehicle moves toward a next waypoint and every P_2 meters corrects its course. In some situations, this distance can be intentionally shortened by HLCS, e.g. near obstacles – see P_7 and P_8,
3. P_3 – is the distance from the closest obstacle which, when reached, reduces the BAUV speed to 0.5m/s. The vehicle, when far away from obstacles, moves with the speed determined by the operator, when obstacles appear near the vehicle, its speed is reduced to make it more maneuverable,
4. P_4 – is a minimal BAUV depth for which it is possible to further reduce it when there is a great risk of collision with an underwater obstacle, it is assumed that BAUV should avoid the ascent to the surface of the sea,
5. P_5 – is the distance from the closest obstacle which enables an operational BAUV depth to be changed; changing the operational depth does not entail chang-

ing the actual BAUV depth immediately, the operational BAUV depth indicates a safe depth of BAUV,

6. P_6 – is the difference between the collision-free course of BAUV determined by a standard procedure used for that purpose and the current course of the vehicle. This difference is applied to detect a situation when the BAUV course should be calculated by a different procedure than the standard one,
7. P_7 – is the distance to the closest obstacle for which parameter P_2 is replaced with P_8, when the distance is less than or equal to P_7 corrections of BAUV course are made more often (every P_8 meters) with the effect that the vehicle is more maneuverable,
8. P_8 – see description of parameter P_7,
9. P_9 – is the distance to the closest obstacle which affects the way of how BAUV depth is changed: vertically or with simultaneous movement forward,
10. P_{10} – is the distance to the goal point which affects the way of BAUV depth change,
11. P_{11} – is a parameter which determines magnitude of BAUV depth change when obstacles are detected very close to BAUV,
12. P_{12} – is the difference between the actual BAUV depth and the operational depth which entails change of the former,
13. P_{13}, P_{14}, P_{15} – are parameters of simple BAUV depth controller which works approximately as follows: if the difference between the desired depth and the actual depth is less than or equal to P_{13} the vehicle starts to change the depth, the depth alteration maneuver is stopped and the vehicle starts to move forward if the difference between the desired depth and the actual depth is less than or equal to P_{14} or P_{15}[2], P_{14} is used when the depth is changed vertically, P_{15} otherwise,
14. P_{16}, P_{17}, P_{18} – are ranges of sensor vision applied when decisions on BAUV maneuvers are made.

All the parameters above were organized in binary chromosomes in which each parameter was encoded as a 7-bit gene. In consequence, the chromosomes included 126 bits in total. To obtain a specific value of a parameter, an integer value encoded in 7-bit gene was scaled to a range determined for the parameter.

In the experiments, the following parameters of GAs were used:

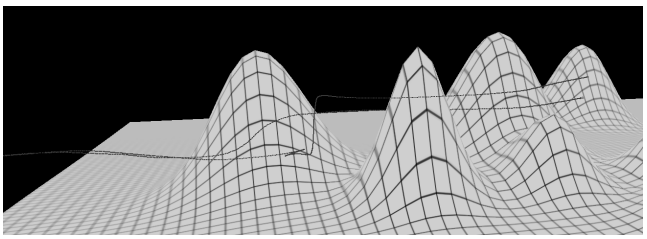
- a number of individuals in each population: 100,
- a maximum number of evolutionary iterations: 1000 (this value was the result of very time-consuming evolutionary calculations).

Parameters of CGA and SSGA:

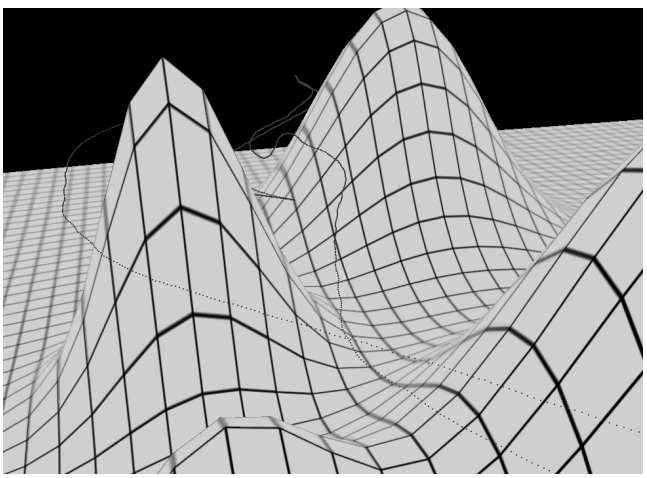
- crossover probability: 0.7
- per-bit mutation probability: 0.01, 0.1,
- size of tournament: 2.

Parameters of EuA:

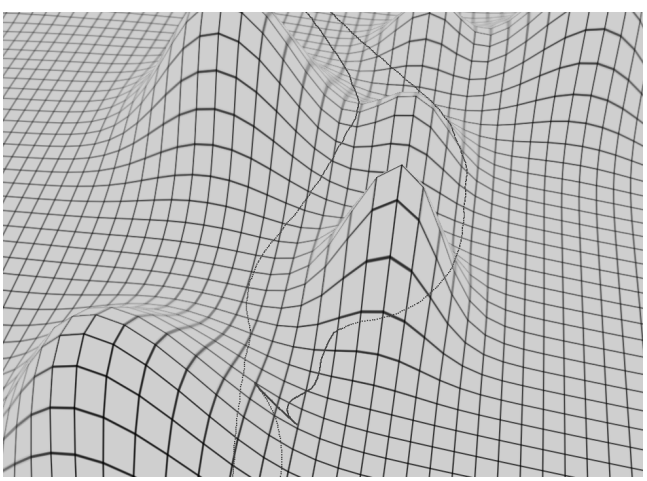
- selection noise: 0.01, 0.2,
- creation rate: 0.01, 0.2,
- restriction operator: on.



(a) the path which runs above is the path of "hand-made" HLCS



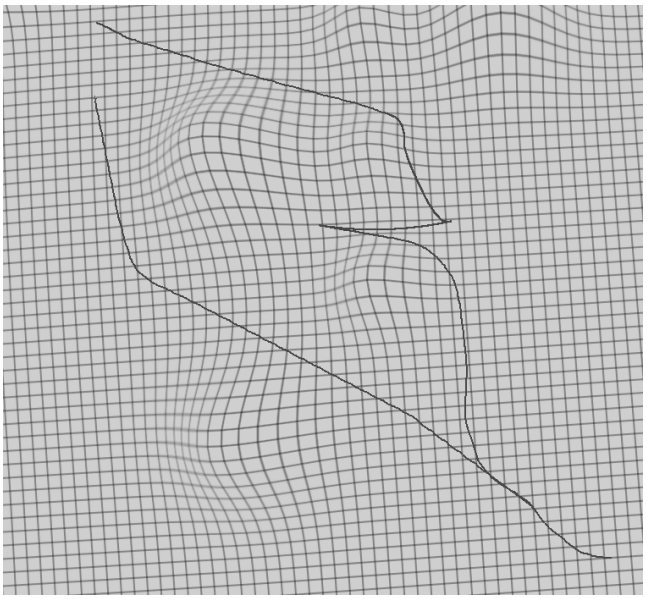
(b) the path on the left-hand side is the path of "hand-made" HLCS



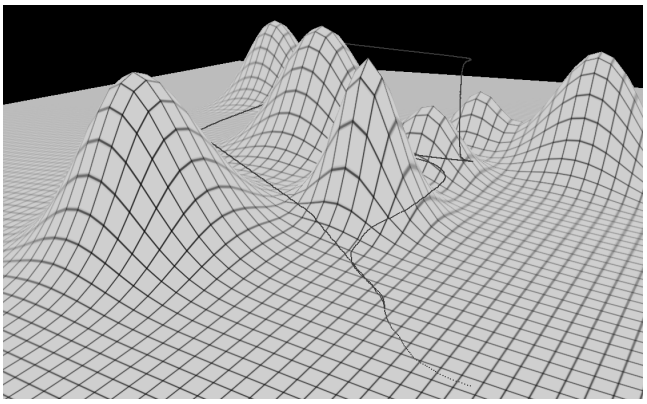
(c) the path on the right-hand side with visible move-backward maneuver is the path of "hand-made" HLCS

Fig. 6. Paths of BAUV for evolved and "hand-made" HLCS: (a),(b),(c) – the same paths visible from other observation points

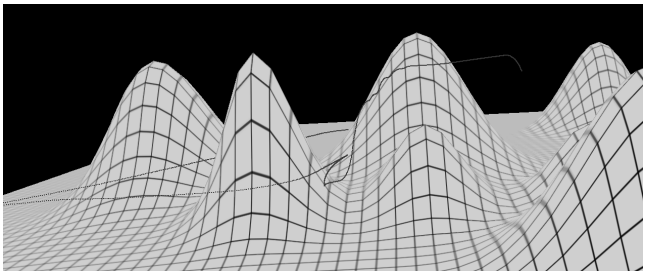
[2] in fact, to stop the depth change maneuver, other conditions also have to be satisfied



(a)



(b)

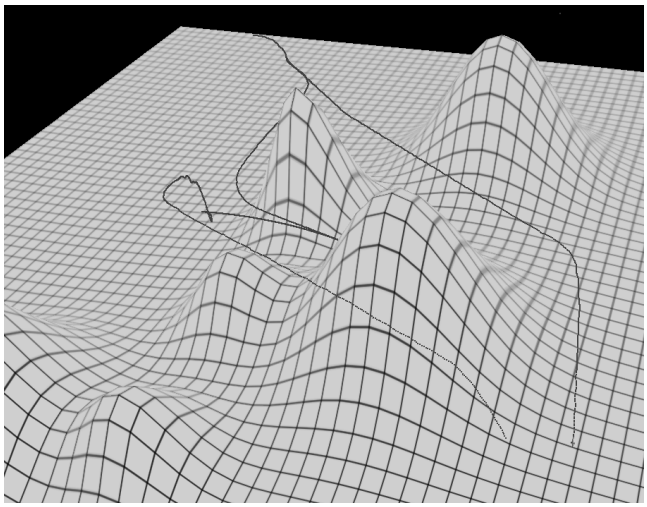


(c)

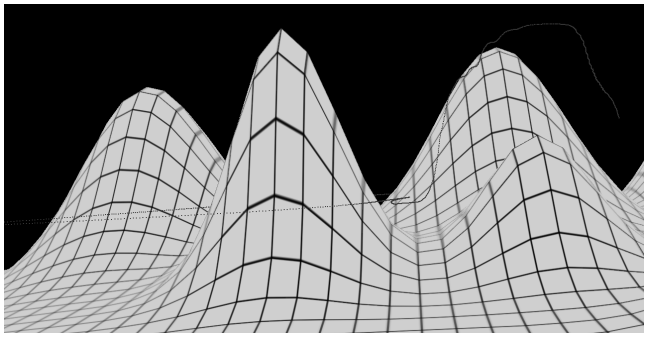
Fig. 7. Next paths of BAUV for evolved and "hand-made" HLCS: (a),(b),(c) – the same paths visible from other observation point, the path with move-backward maneuver is the path of "hand-made" HLCS

IV. 2. Experimental results

The optimization task imposed on GAs appeared to be quite challenging. Each algorithm was run thirty times, which means ninety runs in total, and only ten out of all the GA runs, i.e. six runs of EuA and four runs of CGA, ended with evolving HLCS which was more effective than "hand-made" HLCS: fitness of "hand-made" HLCS was equal to 30.04 whereas fitness of the best evolved HLCS was equal to 30.08[3]. When comparing individual GAs, it appeared that the most effective HLCSs are produced by EuA, then classical CGA and SSGA, on average.



(a)



(b)

Fig. 8. Another view of the paths from Figure 7

As it turned out, the objective of the experiments was achieved, i.e. GAs improved behavior of BAUV in relation to the vehicle equipped with "hand-made" HLCS. The improvement manifested mainly in visually more optimal paths and in a fewer number of HLCS decisions necessary to lead BAUV to the goal. The paths of BAUV with the "hand-made" HLCS, usually, included more changes of the vehicle depth compared to the paths produced by the evolved system. This, in effect, resulted in extension of BAUV way and a longer duration of voyage.

Another symptom of the improvement is also the lack of collisions for the optimized BAUV. In the case of "hand-made" HLCS, there were testing scenarios in which BAUV

[3] both HLCS's took the vehicle to the goal point in each scenario, the only difference between both types of HLCS's concerns N_i^{it} which has a little influence on fitness function – see (9)

could not get to the goal without a collision with an obstacle. As already mentioned, the collision obviously did not imply interruption of the scenario in such situation the vehicle could continue its voyage (provided that the maximum number of HLCS decisions assumed in a scenario was not exceeded), but after the moving-backward maneuver. The only visible consequence of the collision was a longer vehicle path and a longer duration of the vehicle voyage. The use of the optimized variant of HLCS eliminated completely collisions from all the testing scenarios, BAUV equipped with the evolved HLCS was more effective in its maneuvers and could successfully avoid all obstacles encountered on its way. Example paths produced by "hand-made" HLCS and its evolved counterparts are presented in Figures 6–8.

To confirm effectiveness of HLCSs produced by means of GAs, all of them were also tested in ten generalizing scenarios. As it turned out, eight out of the ten systems which were put to the extra tests succeeded, i.e. they led BAUV to the goal with no collision. What is more, they performed all the tasks faster than "hand-made" HLCS.

When comparing HLCS parameters set by the designers and the ones evolved and optimized by GAs, it appeared that in most cases they differ from each other. What is more, parameters set in separate evolutionary runs were often different as well. Example parameters evolved by CGA, EuA and the parameters determined by the designers are given in Tab. 1.

Tab. 1. Example parameters evolved by CGA (P_i^{CGA}), EuA (P_i^{EuA}) and the parameters determined by the designers (P_i^{oper})

	P_i^{CGA}	P_i^{EuA}	P_i^{oper}
P_1	4.6875	2.5	3
P_2	9.375	4.6875	10
P_3	2.8125	7.03125	15
P_4	3.59375	0.46875	0.46875
P_5	1.875	1.25	10
P_6	36.5625	5.625	50
P_7	5.15625	13.125	20
P_8	5.78125	3.4375	3
P_9	24.375	1.875	20
P_10	34.375	46.875	30
P_11	1.5625	3.125	1.5
P_12	7.34375	6.875	3
P_13	15.625	8.4375	8
P_14	4.0625	9.6875	5
P_15	7.96875	1.71875	3
P_16	10.3125	8.4375	5
P_17	17.5	15.4	3
P_18	18.4375	16.5625	5

The list of the parameters specified above indicates that some parameters have roughly the same value no matter who or what determined the value. It seems that parameters no. 1,2,4,8,10,11,12,14,16 can be assigned to that group of parameters. The remaining parameters differ in values depending on the method used to set them: GA or the designers. Usually, two methods agree on the choice of the value for the parameter and simultaneously they differ in their choice from the third method.

In the case of parameter no. 3 which decides when velocity of BAUV is reduced to 0.5m/s, different values of that parameter are due to the fact that for all the testing scenarios starting velocity of BAUV was set to 0.5 m/s, which means that this parameter had rather no influence on HLCS effectiveness.

Low values of parameter no. 5 that was produced by GAs imply that BAUV should change the operational depth as quickly as possible. Value 10m of the parameter which was indicated by the designers enabled HLCS to modify the operational depth considerably further from obstacles with the effect that the vehicle was more inert in this case.

As in the previous case, also for parameter no. 6, GAs proposed lower values than the designers. This time, it means that a special procedure for determining course of BAUV – a new course of the vehicle corresponds to a collision-free course that is closest to the current course of the vehicle, location of the goal does not affect the new course – is run more often than when the parameter value is set by the designers. In other words, when a new BAUV course, which takes into consideration the location of the goal point, requires turning at more than 36 (CGA) or 5 (EuA) degrees, HLCS chooses a more gentle collision-free maneuver affected only by the current BAUV course. This simply suggests that optimized HLCSs prefer more gentle maneuvers compared to their "hand-made" counterpart.

For parameter no. 7, the advice of GAs is again to decrease the value proposed by the designers. It means that the change of BAUV course correction frequency is made only in close proximity to obstacles. In further distance, the standard correction frequency is maintained.

In the case of parameter no. 9, GAs do not agree. One solution, supported also by the designers, is to vertically emerge/submerge BAUV close to obstacles and far away from them. Another solution is to use a vertical method for changing BAUV depth only in close proximity of obstacles.

The difference in opinion is also visible for parameters no. 13 and 15: CGA proposes greater values than EuA and the designers. Since these parameters, roughly speaking, decide about how fast BAUV changes its depth, the suggested values for parameters no. 13 and 15 can be shortly commented as follows: CGA advises slower changes than EuA and the designers.

Parameters no. 17 and 18 determine ranges of vision for BAUV sensors, and both GAs propose in this case to increase the ranges with respect to the values suggested by the designers. The consequence is that the vehicle can notice obstacles faster than for the ranges determined by the design-

ers. However, it also means that when BAUV is surrounded at all sides by obstacles located at different distances from the vehicle, HLCS will not be able to fix a collision-free course because each course will be found to be colliding, each of them will point at a noticed obstacle.

V. SUMMARY

The paper reports on the experiments whose objective was to determine parameters for the High-Level Control System (HLCS) of the Biomimetic Autonomous Underwater Vehicle (BAUV). All the experiments were carried out on a simulated vehicle and there were two reasons for this: first, the real vehicle did not exist at the point of starting the experiments, and second, once the real vehicle appears it has to be equipped in an effective, safe, intensely tested, and consequently, reliable HLCS as quickly as possible. Works on HLCS cannot be started and continued on the real vehicle because of a large cost of such an approach and inevitable delays in providing the final product of the project.

Generally, the experiments showed that GAs have in many points a different approach to controlling BAUV than the group of designers responsible for HLCS implementation. It appeared that HLCS parameters determined by GAs differ in many cases from the ones set by the designers. New optimized HLCS parameters allowed BAUV to behave more effectively, the paths of the vehicle movement were shorter, duration of the voyage decreased and, what is more, the vehicle successfully avoided collisions with obstacles, which was sometimes impossible for "hand-made" HLCS.

Acknowledgments

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 the National Center of Research and Development.

References

[1] Alden, M. and Van Kesteren, A. and Miikkulainen, R., *Eugenic Evolution Utilizing a Domain Model*, Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002), 2002

[2] D. E. Goldberg, *Genetic algorithms in search, optimization and machine learning*, Addison Wesley, Reading, Massachusetts, (1989)

[3] Malec M., Morawski M., Zając J. *Fish-like swimming prototype of mobile underwater robot*, Journal of Automation, Mobile Robotics & Intelligent Systems, Vol. 4, No 3, 2010, 25-30

[4] Polani, D. and Miikkulainen, R., *Fast Reinforcement Learning through Eugenic Neuro-Evolution*, The University of Texas at Austin, AI 99-277, 1999

[5] Polani, D. and Miikkulainen, R., *Eugenic Neuro-Evolution for Reinforcement Learning*, Proceedings of the Genetic and Evolutionary Computation Conference, 2000

[6] T. Praczyk, *Using Assembler Encoding to construct Artificial Neural Networks with a modular architecture*, Polish Naval Academy, 2011

[7] Prior, J. W., *Eugenic Evolution for Combinatorial Optimization*, The University of Texas at Austin, 1998

[8] G. Syswerda, *Uniform Crossover in Genetic Algorithms*, Proceedings of the 3rd International Conference on genetic Algorithms, 1989

[9] Szymak P., Malec M., Morawski M. *Directions of development of underwater vehicle with undulating propulsion*, Polish Journal of Environmental Studies, Hard Publishing Company, **19**(3), 107-110 (2010).

[10] P. Szymak, T. Praczyk, *Control-oriented Model of Biomimetic Underwater Vehicle Motion*, Solid State Phenomena **236** , 121-127 (2015).

[11] D. Whitley, A Genetic Algorithm Tutorial, Statistics and Computing **4**, 1994, 65-85, http://citeseer.ist.psu.edu

[12] http://cmtm.pg.gda.pl/systemy-techniki-glebinowe

Tomasz Praczyk is a senior lecturer at the Institute of Naval Weapon of Polish Naval Academy in Gdynia. He received his MSc degree in computer science in 1996. In 2001, he received his PhD degree; with thesis focused on using artificial neural networks to identify ships. His research interest is in neuro-evolution, artificial immune systems, and reinforcement learning.