Scientific Journals
of the Maritime University of Szczecin

Zeszyty Naukowe
Akademii Morskiej w Szczecinie

2016, 45 (117), 23–28
ISSN 1733-8670 (Printed)
ISSN 2392-0378 (Online)
DOI: 10.17402/081

Received: *31.08.2015*
Accepted: *10.12.2015*
Published: *25.03.2016*

A quick algorithm for planning a path for a biomimetic autonomous underwater vehicle

Tomasz Praczyk

Polish Naval Academy
69 Śmidowicza St., 81-103 Gdynia, Poland, e-mail: t.praczyk@amw.gdynia.pl

Key words: path planning, underwater vehicle, autonomy, collision avoidance, automatic control, effective planning algorithm

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 specialized software. The task of the software is to move the vehicle along a trajectory while avoiding collisions. In its role of avoiding obstacles, the vehicle may sometimes encounter situations in which it is very difficult to determine what the next movement should be from an ad hoc perspective. When such a situation occurs, a planning component of the vehicle software should be run with the task of charting a safe trajectory between nearby obstacles. This paper presents a new path planning algorithm for a Biomimetic Autonomous Underwater Vehicle. The main distinguishing feature of the algorithm is its high speed compared with such classic planning algorithms as A*. In addition to presenting the algorithm, this paper also summarizes preliminary experiments intended to assess the effectiveness of the proposed algorithm.

Introduction

Autonomous Underwater Vehicles (AUVs) are, as the name implies, vehicles which have the ability to operate autonomously, independently of human input. This ability can be used on different levels and in different situations, for example, as completely and continuously autonomous vehicle, or as a remotely operated vehicle with the capability of autonomously returning to its launch point when communication with the operator is lost. Continuous autonomous operation, as opposed to autonomy as an emergency recovery option, requires each AUV to be equipped with specialized software capable of guiding the vehicle along a path fixed by the operator while avoiding collisions with obstacles along the route.

Software with these abilities has been developed for the Biomimetic Autonomous Underwater Vehicle (BAUV) (Malec, Morawski & Zając, 2010; Szymak, Malec & Morawski, 2010). This vehicle is being constructed within the framework of project number 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. The work was functionally divided into two parts, a Low-Level Control System (LLCS) and a High-Level Control System (HLCS). The LLCS is responsible for executing commands provided by the HLCS; its task is to implement the decisions of the HLCS by means of BAUV propellers. In other words, LLCS transforms high-level decisions of HLCS, like moving forward, to turning left, or submerging to a depth of 15 meters, into low-level decisions for its propellers. To do so requires the use of both PID and fuzzy controllers. On the other hand, the HLCS system is responsible for high-level decisions regarding direction of movement, activating or deactivating on/off on-board devices, diving depth, and so on. Whereas the LLCS system performs tasks which must be performed during both autonomous or remotely controlled operation, the HLCS system is used only for autonomous platforms.

Although the HLCS can execute different types of tasks, its primary responsibility is to steer the vehicle along a fixed path while avoiding obstacles encountered along the way. The path is defined in terms of a set of predetermined waypoints created either manually or automatically, with the aid of a dedicated function incorporated into the operator management software. After determining the path, the BAUV comes under the control of the HLCS, moving from one waypoint to another. When the BAUV approaches an obstacle, the HLCS system activates the collision avoidance procedure. After the vehicle has reached a safe location, the HLCS resumes its function of leading the BAUV to the next waypoint. Unfortunately, preliminary experiments with a prototype HLCS have shown that certain conditions make it difficult for the HLCS to execute collision-free manoeuvres. These situations occur when the BAUV is surrounded on all sides by obstacles, some of which are closer than others. One solution to this problem is to use a path planning algorithm with the task of determining a "good", but not necessarily "optimal", collision-free path to the next waypoint. When the HLCS copes well in a given situation, it makes use of a standard collision avoidance algorithm. But when the HLCS system determines that the standard algorithm is incapable of charting a safe course, it activates a special planning algorithm that analyses additional data in search of a solution.

The planning algorithm described above is the main topic of the current paper. It is described in four sections: an introduction, a presentation of the basic algorithm, a report on algorithm verification experiments, and a concluding summary.

The algorithm

The algorithm incorporates a number of basic assumptions. First, it should run as rapidly as possible and be computationally simple. The speed requirement is due to the fact that the BAUV, when running the algorithm, is in the course of completing a mission that also has time requirements. The computational simplicity of the algorithm is important because of the limited capabilities of the on-board computers. The BAUV's computers are constrained by the small amount of physical space to house them as well as by the limited amount of electrical energy to power them. It is also necessary to remember that the task of computers is not only to chart a path for the BAUV, but also to control such additional behaviours as sensor control, camera operation, processing of sonar data, and so on. Second, there are no requirements that the path generated by the algorithm be rigorously "optimal", as opposed to simply being "good". The main thing is that it is determined quickly, and that it avoids collisions.

Third, the algorithm should work well at small distances. Either the operator or the planning algorithm, as a proxy for the operator, must be used to plan the whole path for the vehicle. The algorithm is based on information acquired from the sea charts, and distances between waypoints are rather large. The algorithm that deals with charting a path under complex collision situations has the additional task of determining the course from the current position to the next waypoint generated by the global path algorithm. Moreover, the fundamental responsibility of the collision-avoidance path planning algorithm (or the "local path algorithm") is to direct the BAUV to a position from which the next waypoint can be reached by a straight line course. All of these considerations imply that the collision-avoidance algorithm must work well when dealing with smaller areas.

Fourth, the course must be laid out in a simple virtual environment that makes use of data obtained by the vessel's sensors. These sensors are used to form an environment map comprised of such simple elements as spheres in space. The task of the planning algorithm thus becomes one of finding a path which avoids all the spheres and leads the BAUV to the next waypoint.

Finally, the planning algorithm should work in 2D space, that is to say, in XY space. This means that all paths should be paved at one depth. This assumption results from imperfect sensors which provide only limited data about the environment. Most of the time, the BAUV moves at the same depth and, as a consequence, has environmental information that is almost entirely restricted to a single depth. Therefore the planning algorithm must be able of gathering all the environmental information it needs from sensors operating at one depth.

All of the five assumptions presented above are met by the planning algorithm depicted in Figure 1. The algorithm is a "hill-climbing" algorithm (Gu, 1992; Rayward-Smith, Osman & Reeves, 1996; Selman & Gomes, 2006), which stores the path as a table of successive waypoints (`wayPoints` table). Working in "i" iterations, the path is first stored in the temporary table `wayPointsTmp`. Then the path is modified (`modifyPath()`) and evaluated (`evaluatePath()`). If the modification generates a better path than the best existing path (or sometimes a path that is just as good), the current best is replaced with the modified path;

```
pathPlanning(C,D,I,positionBAUV,P1,P2,P3)
    threshodlModificationCourse = C;
    numberModificationsFailure = 0;
    numberModificationsAllFailures = 0;
    course = random value from range 0..359;
    wayPoints[0] = getPoint(positionBAUV,D,course);//point is generated at distance D and at
                                              //course from current position of BAUV
    fitnessBest = evaluatePath(positionBAUV,wayPoints,D);
    repeat I times
        wayPointsTmp = wayPoints;//the path is stored in temporary path
        modification = modifyPath(threshodlModificationCourse,positionBAUV,D);//the path
                                                                  //is modified - see further
        fitnessCurrent = evaluatePath(positionBAUV,waypoints,D);//the modified path is
                                                            //evaluated - see further
        if((fitnessCurrent>fitnessBest and modification!=0)or(fitnessCurrent>=fitnessBest and
        modification==0)or(fitnessCurrent>=fitnessBest and modification!=0 and numberModifica-
        tionsAllFailures >= P1))
            fitnessBest = fitnessCurrent;
            threshodlModificationCourse = C;
            numberModificationsFailure = 0;
            numberModificationsAllFailures = 0;
            end if
        else
            wayPoints = wayPointsTmp;
            numberModificationsAllFailures++;
            if(modification==0)
                numberModificationsFailure++;

            if(numberModificationsFailure > P2)
                threshodlModificationCourse = threshodlModificationCourse + P3;
                numberModificationsFailure = 0;
                end if
            end else
        end repeat
    end function
```

Figure 1. The planning algorithm

otherwise, the modified path is deleted. Moreover, when the modification repeatedly (more than `P2` times) fails to produce a better path than the current best, the parameters of the modification function are changed (at `P3`) to increase the level of modification in the next iterations.

The evaluation function (`evaluatePath()`) included in the path planning algorithm works according to the implementation presented in Figure 2. It usually distinguishes two situations, one in which the path obtains an evaluation (or fitness) value equal to zero, and the other being all paths with non-zero evaluations. A fitness of zero is assigned to a path in the following two circumstances:

1. When any section between waypoints, the first waypoint and the current position of BAUV, or the last waypoint and the goal point of the algorithm, is in an obstacle. That is to say

```
evaluatePath(positionBAUV,wayPoints,D)
    repeat i=0, i<numberOfWaypoints
        if(i==0 and !isInObstacles(positionBAUV) and
        isSectionInObstacles(positionBAUV,wayPoints[i]))
            return 0;
        else
            if(i==0 and isInObstacles(positionBAUV) and isInObstacles(wayPoints[i]))
                return 0;
            else
                if(i==numberOfWaypoints-1 and isSectionInObstacles(wayPoints[i],pointGoal))
                    return 0;
                else
                    if(isSectionInObstacles(waypoints[i-1],wayPoints[i]))
                        return 0;
        end repeat

    distance = numberOfWaypoints*odlegloscPomiedzyWayPointami
    +getDistanceBetweenPoints(wayPoints[numberOfWaypoints-1],pointGoal);

    return 1/distance;
    end function
```

Figure 2. Evaluation function

```
modifyPath(threshodlModificationCourse,positionBAUV,D)
     operation = random value from range 0..2
     switch(operation)
     case 0://modification of selected waypoint
          numberOfPoint = 0;
          if(numberOfWaypoints > 1)
               numberOfPoint = random value from range 0..numberOfWaypoints-1;
          courseDelta = random value from range 0..threshodlModificationCourse-1;
          if(numberOfPoint==0)
               course = getCourse(positionBAUV,wayPoints[numberOfPoint]);
               wayPoints[numberOfPoint] = getPoint(positionBAUV,D,course+courseDelta);
               end if
          else
               course = getCourse(wayPoints[numberOfPoint-1],wayPoints[numberOfPoint]);
               wayPoints[numberOfPoint]=getPoint(wayPoints[numberOfPoint-1],D,course+courseDelta);
               end if
          end case
     case 1://addition a new waypoint
          if(numberOfWaypoints < maxNumberOfWayPoints)
               course = random value from range 0..359;
               wayPointy[numberOfWaypoints] = getPoint(wayPoints[numberOfPoint-1],D,course);
               numberOfWaypoints++;
               end if
          end case
     case 2://removal of last waypoint
          if(numberOfWaypoints > 1)
               numberOfWaypoints--;
          end case
     end switch
     return operation;
     end function
```

Figure 3. Modification function

whenever at least one point of the path lies inside a spherical obstacle.

2. When the current position of the BAUV is inside an obstacle, and any waypoint is also inside an obstacle.

Otherwise, the path gets is assigned a fitness that is inversely proportional to the length of the path.

The modification function (`modifyPath()`) which tries to produce better paths over a number of iterations is presented in Figure 3. It performs the following operations on a path:

1. Modification of a randomly selected waypoint (case 0 – in Figure 3) such that its location is moved to another point. The magnitude of the movement depends on the value of the parameter `threshodlModificationCourse`.
2. Addition of a new waypoint at the end of the path located a distance D from the previous

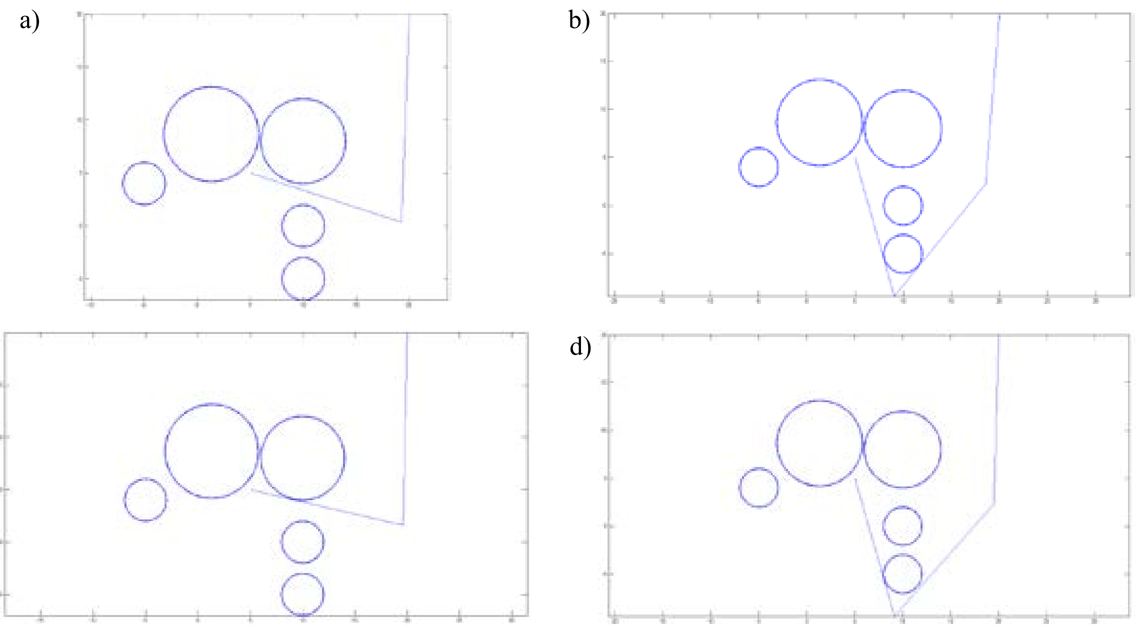

Figure 4. Example paths generated by the planning algorithm after 50 (a), 100 (b, c), and 200 (d) iterations

waypoint, and on a randomly selected course from the previous waypoint.
3. Removal of the last waypoint from the path.

The order of these operations is selected at random.

Experiments

Experiments were conducted to verify the intended performance of the algorithm as defined in the previous section. In the experiments, the algorithm was

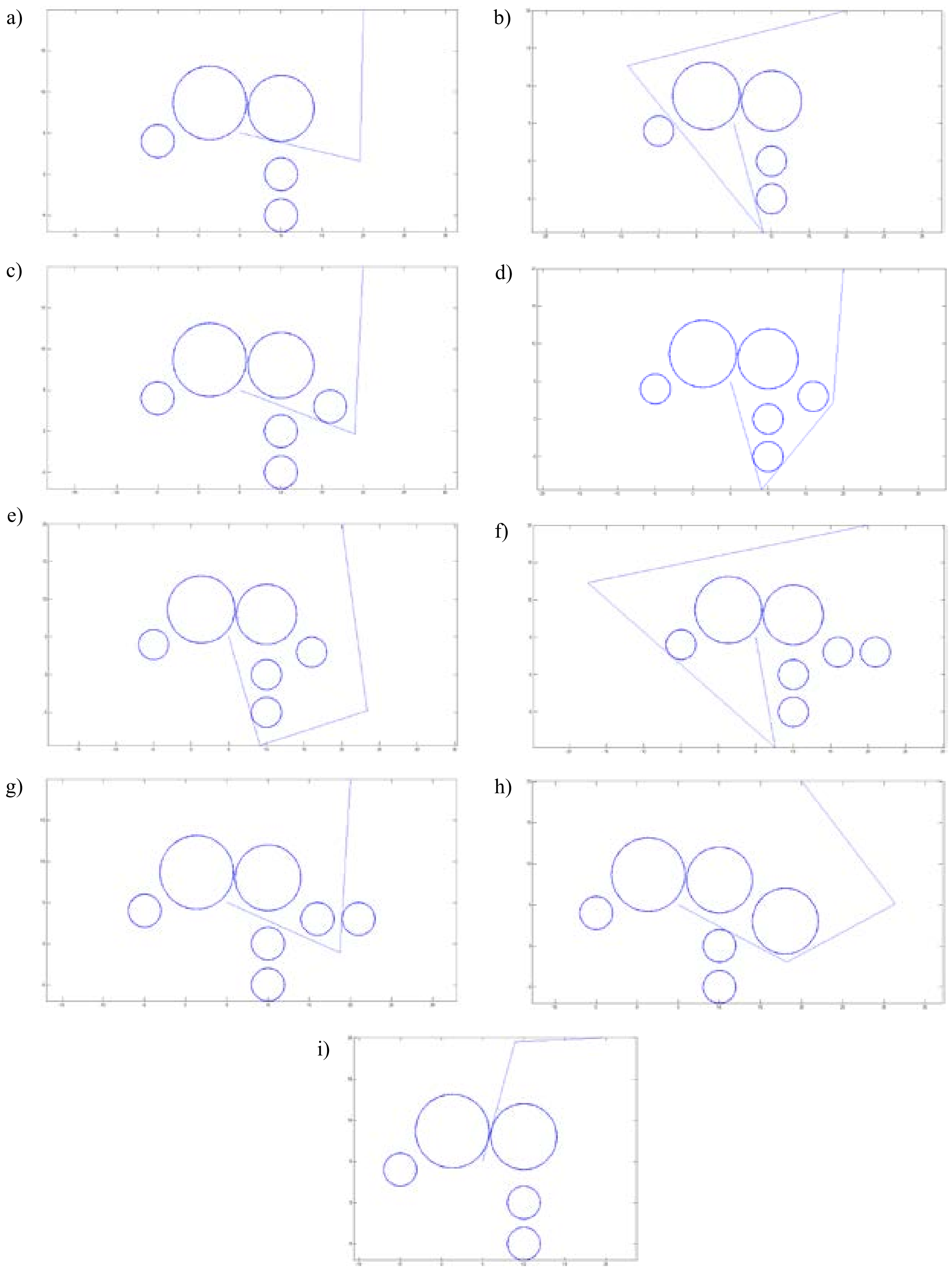

Figure 5. Example paths generated by the planning algorithm after 400 (a, b), 600 (c–h), and 1000 (i) iterations

tasked with finding the path for the BAUV under different testing scenarios. The tests simulated progressively more difficult path planning scenarios. First, the BAUV was surrounded by few spherical obstacles, and then more obstacles were added to make the task of the algorithm increasingly more challenging. In the experiments, the following values of the algorithm parameters were used: $C = 20$ deg, $D = 10$ m, $I = 100{,}1000$, P1 = 4, P2 = 5, P3 = 20. Results of the experiments are presented in Figures 4 and 5.

In general, the results showed that the algorithm is able to find even barely visible paths between obstacles quite rapidly. After only 50 iterations ($i = 50$), the algorithm appeared to be able to generated collision-free paths. However, about half of the paths produced, in this case, led through obstacles.

Increasing the value of *i* to 100, 200 and 400 iterations improved the situation, but did not entirely eliminate it; even with 400 iterations, some paths were generated that led to collisions with spherical obstacles. Indeed, it was not until *i* was set to 600 iterations that all paths produced by the algorithm were collision-free. Moreover, when *i* was set to 600, about 80% of the paths produced by the algorithm were very similar even though all modification operations were subjected to random modifications. Figures 5a, 5c, and 5g show the paths that were produced the most frequently. Figure 5i is also very interesting because it illustrates the path found by the algorithm for $i = 1000$, the path which goes through a barely visible hole between obstacles. As was the case for other highly iterated paths, the path depicted in Figure 5i was seen quite frequently, but only when *i* was set to 1000.

Conclusions

This paper presents a quick, hill-climb algorithm for planning a path for the BAUV. The algorithm is run on-board the BAUV board during vehicle operation, and is tasked with setting the course for the vehicle when the standard control strategies fail to indicate a safe direction of movement.

Experiments were conducted to evaluate the algorithm proposed here. In general, these experiments showed that the algorithm produces collision-free paths so long as a minimum of 600 iterations were performed. At 1000 iterations, the algorithm found paths that approximated optimal paths.

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 National Centre of Research and Development.

References

1. Gu, J. (1992) Efficient local search for very large-scale satisfiability problems. *Sigart Bulletin* 3 (1). pp. 8–12.
2. Malec, M., Morawski, M. & Zając, J. (2010) Fish-like swimming prototype of mobile underwater robot. *Journal of Automation, Mobile Robotics & Intelligent Systems* 4 (3). pp. 25–30.
3. Rayward-Smith, V.J., Osman, I.H. & Reeves, C.R. (eds) (1996) *Modern Heuristic Search Methods*. London, UK: John Wiley.
4. Selman, B. & Gomes, C.P. (2006) *Hill-climbing Search. Encyclopedia of Cognitive Science.*
5. Szymak, P., Malec, M. & Morawski, M. (2010) Directions of development of underwater vehicle with undulating propulsion. *Polish Journal of Environmental Studies* 19 (3). Hard Publishing Company, Olsztyn. pp. 107–110.