Autonomous navigation of an underwater robot (AUV) by genetic algorithm

Paul Pineau
Brest, France
Email: paul.pineau@ensta-bretagne.org

***Abstract*—There are multiple methods to navigate an AUV independently. Creating an autonomous mission controller and planner with FSM is of several interest for underwater missions. First of all the architecture is algorithmically simple. The positioning of an AUV is also problematic and this method of navigation offers a lot of room for maneuver if the robot is lost. This paper discusses how to find the optimal headings for the state machine in order to quickly validate the waypoints. For this purpose a genetic algorithm is used in order to optimize by a few iterations the optimal headings.**

Keywords: FSM, AUV, Genetic algorithm, autonomous

I. Introduction

The ocean remains an extremely unknown environment for man because it is dangerous and difficult to explore. If before one could study the surface with boats relatively easily, studying the seabed was (and remains) extremely complex. Several technological innovations have gradually allowed to replace divers by robots. The first were the ROV (Remotely Operated Vehicles) which made it possible to remove the human risk but being linked by a wire to an operator the missions are thus limited, in time as well as in space. This is why AUVs (Autonomous Underwater Vehicles) offer many advantages for underwater exploration. Their range is infinite (using the battery) and they can take readings on their own. However, the difficulty lies in the positioning of the robot that does not have access to GPS underwater. In a lake a robot often has to make a path between different waypoints. Using a finite state machine has many advantages, especially on the algorithmic complexity of the controller that results from it. However finding this controller can be complicated. In this topic, the goal is to find the optimal FSM for the path of the AUV through a genetic algorithm.

The purpose of the state machine is to "bounce" the AUV on isobaths. On a nautical chart, an isobath is a line joining points of equal depth. It is thus a curve of level, indicating the depth of a surface below the water level. At each isobath detection the AUV takes a new course in a predefined list. The goal is to find this list of heading in order to validate in an optimal way the waypoints given by the user at the beginning of the mission.

II. Genetic Programming for headings optimisation

The genetic programming method consist of iterate a list of data in order to find the optimal value. In this cas the data are a list of headings. For the algorithm[1] we need an initial population called M_0, composed of N members denoted $m_k, k \in [1, N]$. We will iterate this population until we find at least one member that satisfies our needs. Then we define $f(m_k)$ the fitness function that will define the quality of the member, in our case it will be the travel time. The principle of the algorithm is to keep the best members and combine them in order to improve the result in the image of the theory of evolution. We therefore define two operations, a unary one, the mutation that we will note $g_m(m_k)$ and another binary one, the crossover that we will note $g_c(m_k)$. We can then apply the genetic algorithm[1].

Algorithm 1 Mutation(m = [h$_1, h_2, .., h_n$])

```
Require: $k \leftarrow randint(1, N)$
Require: $p \leftarrow random(0, 1)$
1: if p $>$ 0.5 then
2:     $m[k] \leftarrow m[k] + 20$
3: else
4:     $m[k] \leftarrow m[k] - 20$
5: end if
```

Algorithm 2 Calculate m = g$_c(m1 = [h_{11}, h_{12}, .., h_{1n}], m2 = [h_{21}, h_{22}, .., h_{2n}])$

```
Ensure: m1 and m2 are sorted
1: for $iteration = 1, 2, k \ldots$ do
2:     $m[k] \leftarrow 1_{\overline{2}}.(m1[k] + m2[k])$
3: end for
```

A. Fitness Function

So we have $f(m_k)$ the fitness function. $f(m_k)$ calculates the time taken by the submarine to complete the route delimited by the waypoints. Thanks to an Euler method, the time taken by the submarine for each m_k of the population M is calculated. We need to check if all waypoints are validated. Let define $v(w_k)$ for $w_k \in W$, with W the list of waypoints. $v(w_k)$ return True if the waypoint is validated, false otherwise. A matrix M is then defined with $M(i, j) = 1$ if waypoint j is validated starting from i. We then have that all the waypoints are validated if $\forall i, \sum_j M(i, j) > 2$ and $\forall j, \sum_i M(i, j) > 2$ which means that at least the AUV came to the waypoint and leave. For example, if W = ([0,0],[10,0],[0,10]), we have M that can look like this :

$$\begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}$$

Here, the path is w1 to w2, w2 to w3 and finally w3 to w1.

B. The algorithm

Let define M the population,$PopSize$ the size, τ the probability to choose the crossover operation instead of the mutation, N_{max} the maximum iteration of the algorithm.

Algorithm 3 Compute optimal headings h

```
Require: $globalbest \leftarrow$ inf
Require: $bestheading \leftarrow []$
1: for $iteration = 1, \ldots, N_{max}$ do
2:     $globalbest \leftarrow$ best score in the population
3:     $bestheading \leftarrow$ the associate population member
Require: $M$ is sorted with the fitness
4:     for $1, \ldots, k, \ldots, PopSize$ do
5:         if $random(0,1) > \tau$ then
6:             $M[k] \leftarrow mutation(bestMembers)$
7:         else
8:             $M[k] \leftarrow crossover(bestMembers)$
9:
```

Where $bestMembers$ are the best members of the population at that time according to the fitness function.

III. Python simulation

I chose to implement the algorithm in python. To do so, I started by modeling the physics of a riptide submarine [2][3]. Then I had to apply a controller via FSM[4] (Finite State Machine). The principle is to measure a distance to the bottom that will have been modeled beforehand is to change state each time the $k-meter$ isobath is reached. I choose to use only a list of three headings.

A. FSM

The FSM look like this in my case.

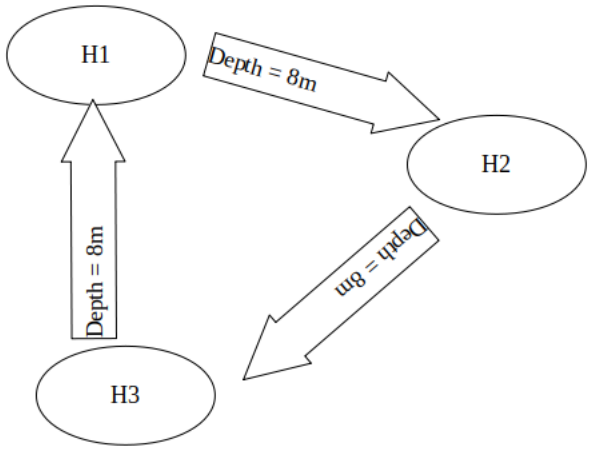


Fig. 1: FSM

There is one problem we need to take into account. When the AUV detect the isobath, he change heading and a few moment later detect almost the same isobath so we need to create special state. For example for the state $H1$ corresponding to heading 1. We need to have $H1in$ and $H1out$. the out state is useful for the transition and when we detect an isobath, we change state to the out one and when it's detected another time we change to the in state. With this tip there is no problem when the depth is rising or falling.

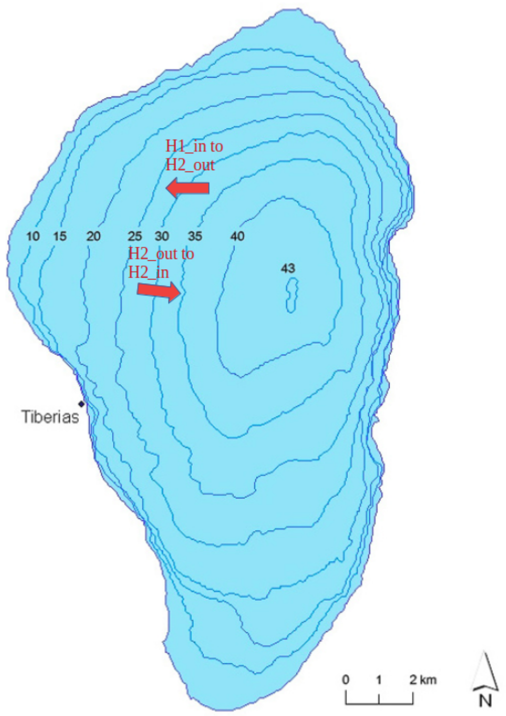
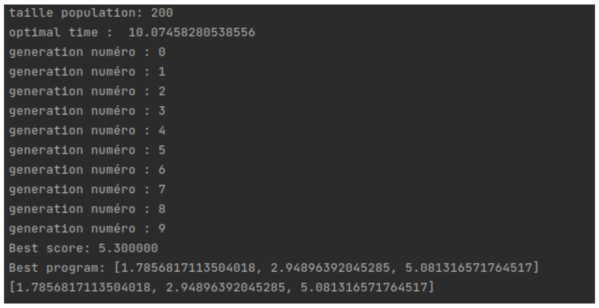


Fig. 2: In and Out States, source:"https://educalingo.com"

IV. Resulats

The algorithm makes it possible to find by having as waypoints $[[10, 20, -2], [20, 20, -2], [20, 10, -2]]$, with as background a giant bowl, optimal headings worth: $[1.7856, 2.9496, 5.0813]$ for an optimal time of 5.3.



```
taille population: 200
optimal time :  10.07458280538556
generation numéro : 0
generation numéro : 1
generation numéro : 2
generation numéro : 3
generation numéro : 4
generation numéro : 5
generation numéro : 6
generation numéro : 7
generation numéro : 8
generation numéro : 9
Best score: 5.300000
Best program: [1.7856817113504018, 2.94896392045285, 5.081316571764517]
[1.7856817113504018, 2.94896392045285, 5.081316571764517]
```

Fig. 3: Results

References

[1] Koza, J., 1998. Genetic programming 1998. San Francisco: M. Kaufmann Publishers

[2] Thor I Fossen.Handbook of marine craft hydrodynamics and motion control. John Wiley Sons, 2011

[3] Luc Jaulin.Mobile robotics. John Wiley Sons, 2019

[4] Timothy Kam, Tiziano Villa, Robert K. Brayton et Alberto Sangiovanni-Vincentelli, Synthesis of Finite State Machines : Functional Optimization, Springer, 1996, 282 p.