UDT 2019 – Future Underwater Weapon Tactical/Homing Algorithm Optimisation Using Machine Learning.

R. Larham[1]
[1]BAE Systems, Maritime Services, Broad Oak, Portsmouth, Hants, PO3 5PQ, UK

Abstract — This paper discusses the use of machine learning in the optimisation of underwater weapon tactical algorithm design. It proposes that in the immediate future this should utilise the ideas and structure of existing methods, being restricted to the optimisation of the design. It also considers the relative performance of ML tactics compared to those developed by traditional methods by looking at the state of ML in computer chess compared to traditionally developed chess engines. Also the HW requirements for ML tactical design are briefly discussed.

1 Introduction

The performance of an underwater weapon is determined by the available sensors, environment, target characteristics, tactics and countermeasures, weapon kinematic performance, targeting data and weapon homing algorithms. It is the design process of the last of these, weapon homing algorithms (which I will describe as weapon tactics from here on), that this paper is concerned with. Traditionally the design and tuning of such algorithms has been done by humans, but the time has come/fast approaching to adopt a level of automation of the process [1]. For weapon tactics a self-learning neural network approach is not appropriate as we require that the design/behaviour of the system be comprehensible to the design team, customer and operators. What is proposed here is that we adopt a machine learning/optimisation approach to tuning designs the basic structure of which is based on traditional designs.

2 Traditional Tactical Design

The traditional approach to the tactical design, as practiced by organisations that I have worked at, generally starts by creating a structure and populate using design adapted from previous project, or if new project create dummy structure and stubs for individual tactics. This could comprise a Tactical Decision Maker (TDM) and stubs for tactics that are thought appropriate. Figure 1 and Figure 2 show the position of tactics within the system and the structure of a TDM.

Using the system simulation/model as a test harness, we create an experimental TDM to call a particular tactic and hand fettle the tactic design until simulation results for that tactic are judged satisfactory.

Then repeat for other tactics

When the tactical system is sufficiently populated we hand optimise the TDM with the tactics.

Repeat until all tactics and TDM calling rules optimised and hopeful meet the performance requirements in the contract.

Then test on trials and alternative simulations, and re-tune if problems found.

One of the problems with this approach is its dependency on the availability of significant numbers of experienced competent staff, together with its cost.

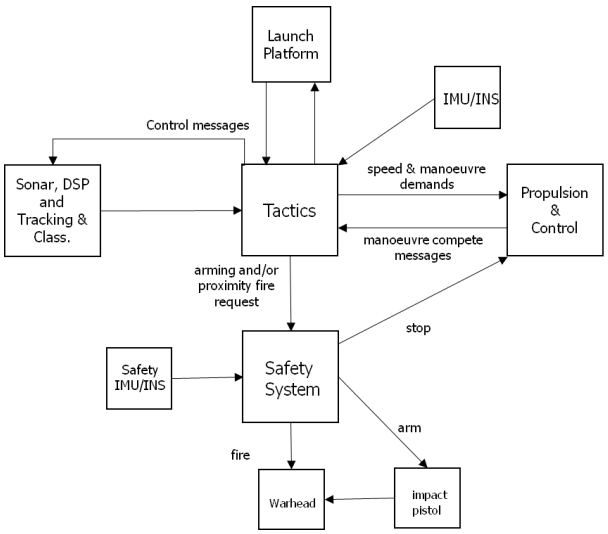

Figure 1: Simplified Torpedo Block Diagram Showing Where Tactics Sits With Respect To Other Parts of the System

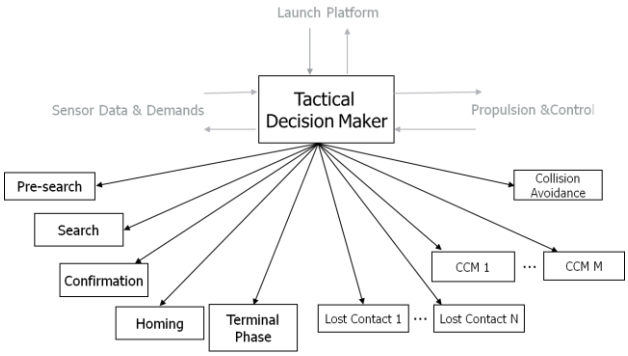

Figure 2: Structure of Tactics

3 Desirable Features Of Design Methods

In an ideal world we would like to specify the performance required of the system and use an automatic process to general weapon/tactical software which meets that performance requirement (as nearly as possible).

The specification of performance requirements should already be available in the traditional process, as it is part of the acceptance criteria for a project.

Completely automated design could be done in principle as long as we can measure how far from the target performance a design is. In principle, if fantastically impractically, this could be done using stochastic search, possibly more practically using other global optimization heuristics like simulated annealing, machine learning and/or evolutionary/genetic algorithms. But any method that makes no assumptions about the structure of the design will be impractical for the foreseeable future and also probably have the undesirable property of being completely opaque to humans.

In the near term it is desirable to retain the general structure of the existing designs, confining optimization to the parameters within the individual tactics, and the tactic call rules in the TDM.

Caveat: The design of the tactics must be robust against assumptions of target signatures and tactics … our systems should do something sensible even when the target/environment etc. are not something specified in the contract - we are not here just to "answer the exam question" (of Contract Acceptance Simulations). As a partial counter to the tendency to narrowly design against the requirement may be to include the target tactics in the optimisation process, making the optimisation process analogous to ML the game playing systems such as Alpha0 [3] and Lela0 (though asymmetric between players).

4. Expectations

An initial approach to the use of ML in the algorithm design process has been proposed, which is to retain the general structure of existing tactical processing systems, but to use the free parameters in the design as the components of the design vector with optimisation conducted using simulated engagements optimising subsystems sequentially in a kind of section search optimisation.

Several “toy” problems have been tested, where a single algorithm has been optimised against a training set of vignettes both by us and other organisations (see [2] for an example of area search). These have given what might be described as satisfactory results, in the sense that the toy problems may be tailored to show ML to good advantage, but the question remains of how well ML design of the complete system performs compared to a manually optimised design? Two examples are shown below in section 6, one from [2] and another from BAE experiments.

A field in which both highly hand optimised designs and ML designs exist and have been tried against one another is Computer Chess. The current star of ML chess systems is Alpha0 (A0) which uses a deep neural network trained in self play mode. The current champion among conventional (hand fettled) chess systems is Stockfish (Sf) [4]. In a tournament between A0 and Sf A0 achieved a significantly (spectacularly) better result than Sf. The validity of this is disputed as the two engines were not running on comparable HW nor were the time controls suited to Sf’s design.

Lela0 (L0) is an attempt to implement the ideas behind A0 on more usual HW for a chess engine. This allows L0 (actually LelaChess0 Lc0) and Sf to compete on a more level playing field under more normal time controls. Lc0, competed a number of rounds of the Top Chess Engine Championship in 2018, progressing from moderate results to coming second behind Stockfish in December 2018 (at time of writing the superfinal between Sf and Lc0 has just finished with victory going to Sf 50.5-49.5 but so close that they are for all intents and purposes of equal strength on this basis). This is probably the best comparison of the relative performance to be expected from highly optimised ML and hand-fettled systems for the same task on comparable HW. We may conclude that at present Hand-Fettled is still competitive, but the effort required to improve it is much greater than that for Lc0 in self play learning mode.

Extrapolating this (rather cavalierly) to torpedo tactic we may take this to indicate that the performance of ML optimised tactics will be comparable to hand-fettled tactics. Which approach will give greater performance will depend on the effort put into the optimisation.

Comment: There are several features of the approach to ML utilised in A0 and L0 that if they can be extended from a discrete to a continuous problem, could be of interest in underwater weapon algorithm design. In that while there might not be an identifiable set of tactics with transition rules would, given the state of the system, tell you the probability that it will pursue some course of action and the value of that action. The promise of quantum computing and its application to ML might make this an attractive long term line of research [6].

5. Hardware for ML Tactics Design

When I first started thinking about ML, automated optimisation of weapon algorithms it was quite clear that this would involve a massive amount of computation, perhaps as many as 100 replications per scenario for something like 100 scenarios per iteration of the optimisation loop. Each replication possibly comprising a simulation of an entire engagement on a detailed simulation. The only mitigating factor being the problem is that much of the computation is loosely coupled and highly parallel and suited to loosely couples multiprocessor architectures [7].

Fortunately even then (this is something like 15+ years ago) clusters of commodity level (single core) PCs could be configured suitably for the task. Today the hardware problem is more tractable with multi-core processors, or by farming the task out across the unused CPU cycles on a corporate network.

If we were to go down the route of Deep Neural Nets then the processing requirements might be reduced as we do not require complete evaluations of the objective during the training process. Also the emerging field of quantum machine learning needs to be watched.

6. Some Examples of ML Optimised Tactics Like Algorithms

The first example I want to show is from Kierstead & DelBalzo's paper [2] which uses a genetic algorithm to design an area search plan for a target in an area where the target detectability varies across the area. They have a search area approximately 46x165 nautical miles. The searcher searches at 15kts for a total of 40 hours. The target evades at 5kts with 3 hours between course changes and reflects off of the search area boundaries. They compare the Genetic Algorithm generated search path with a standard ladder search, and an optimised ladder search (see Figure 3 below). They find that the GA search has a CPD of 35% compared to 29% for the best ladder and 24% for the standard ladder. This shows the GA search to advantage.

It should be remarked that a more modern particle based approach + greedy algorithm would probably do at least as well as the GA does here.

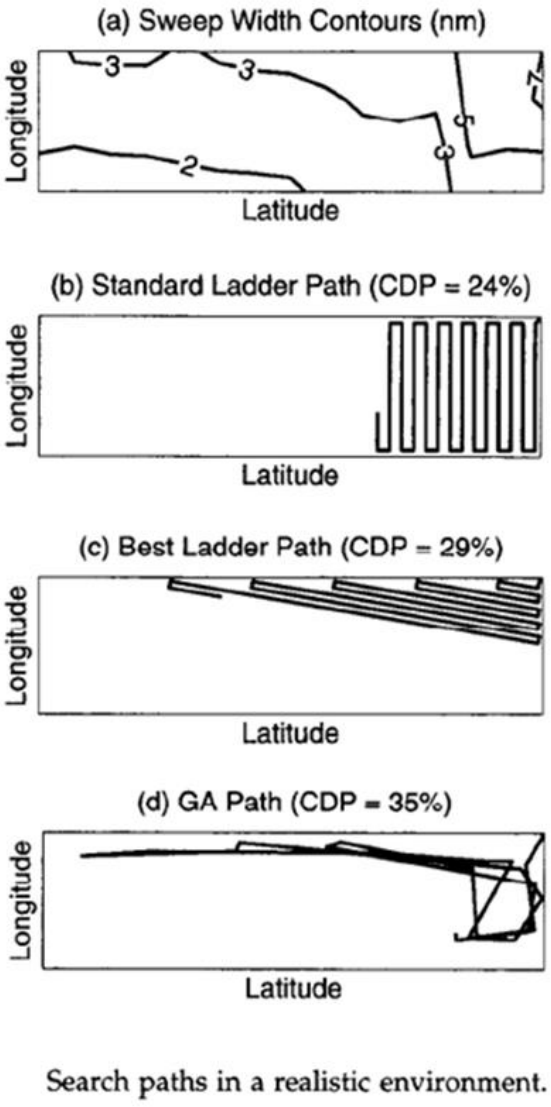

Figure 3: Plots of Sweep Width, and Search Patterns From [2] Figure 9.

The second example is from some experiments I conducted in 2007 on the optimisation of part of the terminal homing phase of a torpedo. The objective is to minimise a combination of the probability of

losing contact and accuracy of the delivery point over the space of target speeds, turn rates and initial geometry etc, and the weapon is manoeuvring to achieve this. In the sample plot one instance is shown with the target at 30kts and turning at 3°/s.. The plot shows the probability of achieving the desired result (remaining in contact and achieving an acceptable geometry at exit) for this instance. The big dip in probability of success seen in the hand optimised algorithm is due to the algorithm getting confused when the torpedo crosses the target axis. One might observe that a more competent designer would have realised this and modified tha algorithm accordingly, but even without the dip the hand optimised algorithm is worse than the ML optimised algorithm. In this example I used stochastic search as the optimisation/learning algorithm mainly because it is the one global optimisation algorithm with provable albeit slow convergence, and I was not particularly concerned with efficiency of the learning process due to a relatively small search space in this example.

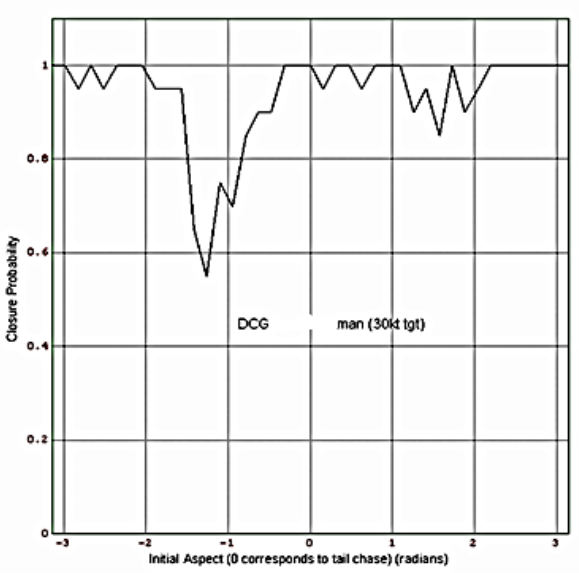

Figure 2: Closure probability against initial aspect current algorithm

Figure 4: Plots showing the performance of the hand optimised for the second example.

References

[1] G. Kendall, *Evolutionary computation has been promising self-programming machines for 60 years – so where are they?*, The Conversation, March 2016. https://theconversation.com/evolutionary-computation-has-been-promising-self-programming-machines-for-60-years-so-where-are-they-91872

[2] Kierstead & DelBalzo, *A Genetic Algorithm Applied to Planning Search Paths in Complicated Environments*, MOR 8 #2 2003 pp45-59

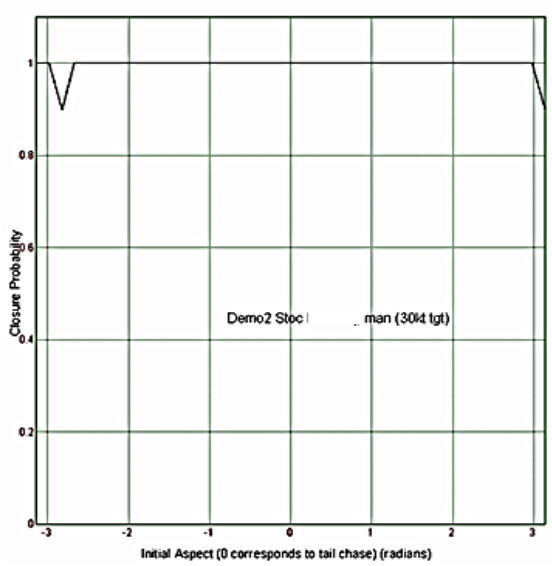

Figure 6: Closure probability against initial aspect final algorithm

Figure 5: Plots showing the performance of the ML optimised algorithm for the second example.

[3] Silver, D., Hubert, T., Schrittwieser, J., *A general reinforcement learning algorithm that masters chess, shogi, and go through self-play*. Science. 362 (6419): 1140–1144, Dec 2018.

[4] Wikipedia Contributers, *Stockfish (chess)*, Wikipedia, The Free Encyclopedia, Date retrieved: 29 January 2019 09:42 UTC, https://en.wikipedia.org/w/index.php?title=Stockfish_(chess)&oldid=880149074

[5] Silver A., *Leela Chess Zero: AlphaZero for the PC*, Chess Base News, 26th April 2018, https://en.chessbase.com/post/leela-chess-zero-alphazero-for-the-pc

[6] Shaw, R., *Quantum Machine Learning: An Overview*, KDnuggets, January 2018, https://www.kdnuggets.com/2018/01/quantum-machine-learning-overview.html

[7] Wikipedia Contributors, *Multiprocessing*, Wikipedia, The Free Encyclopaedia, 11 February 2019 10:48 UTC, https://en.wikipedia.org/w/index.php?title=Multiprocessing&oldid=881304958

.