J Intell Robot Syst (2009) 54:495–510
DOI 10.1007/s10846-008-9276-8

FPGA Implementation of Genetic Algorithm for UAV Real-Time Path Planning

François C. J. Allaire · Mohamed Tarbouchi · Gilles Labonté · Giovanni Fusina

Received: 15 March 2008 / Accepted: 30 June 2008 / Published online: 9 July 2008

Abstract The main objective of an Unmanned-Aerial-Vehicle (UAV) is to provide an operator with services from its payload. Currently, to get these UAV services, one extra human operator is required to navigate the UAV. Many techniques have been investigated to increase UAV navigation autonomy at the Path Planning level. The most challenging aspect of this task is the re-planning requirement, which comes from the fact that UAVs are called upon to fly in unknown environments. One technique that out performs the others in path planning is the Genetic Algorithm (GA) method because of its capacity to explore the solution space while preserving the best solutions already found. However, because the GA tends to be slow due to its iterative process that involves many candidate solutions, the approach has not been actively pursued for real time systems. This paper presents the research that we have done to improve the GA computation time in order to obtain a path planning generator that can recompile a path in real-time, as unforeseen events are met by the UAV. The paper details how we achieved parallelism with a Field Programmable Gate Array (FPGA) implementation of the GA. Our FPGA implementation not

F. C. J. Allaire (✉) · M. Tarbouchi
Electrical and Computer Engineering Department, Royal Military College of Canada, Kingston, ON K7K7L6, Canada
e-mail: francois.allaire@rmc.ca

M. Tarbouchi
e-mail: tarbouchi-m@rmc.ca

G. Labonté
Mathematics and Computer Science Department, Royal Military College of Canada, Kingston, ON K7K7L6, Canada
e-mail: labonte-g@rmc.ca

G. Fusina
Defence R&D of Canada–Ottawa, Ottawa, ON K1A0Z4, Canada
e-mail: Giovanni.Fusina@drdc-rddc.gc.ca

only results in an excellent autonomous path planner, but it also provides the design foundations of a hardware chip that could be added to an UAV platform.

Keywords UAV · Path planning · Genetic algorithms · FPGA · Real-time

1 Introduction

UAVs are used for many different services. In the military context, research has been performed to improve the autonomy of the UAV with different types of missions in view:

- Suppression of enemy air defence [1–3];
- Air-to-ground targeting scenario [4];
- Surveillance and reconnaissance [5];
- Avoidance of danger zones [6–15]; and
- Command, control, communication, and intelligence [16]

In the civilian context, UAVs could also be used for purposes such as:

- Weather forecast/ hurricane detection;
- Urbane police surveillance;
- Farm field seeding; and
- Border surveillance

All these scenarios require a common task: Path Planning. This task is crucial to the well being of the mission and to ensure the safety of both the mission and the UAV. Facing the reality of a dynamic world, UAVs need to re-plan their path in real-time. Currently, there is a human operator, dedicated to the UAV navigation, who is responsible for that function. This paper explains a solution that provides UAVs with an autonomous real-time path planning capability based on the Genetic Algorithms (GA) method implemented on a FPGA circuit board. This solution not only meets the real-time requirements of UAVs, but, with some additional works, would provide a hardware design ready to be inserted into UAV military and/or civilian platforms.

Firstly, the paper presents different existing path planning algorithms; it explains why the GA was selected. Secondly, the specific GA method used is detailed. Thirdly, the paper presents details of the GA design implementation on the FPGA circuit board. Finally, the results obtained will be discussed.

2 Path Planning Techniques

There are many path planning techniques used within the mobile robot world. Each one has its strengths and its weaknesses. Some techniques better fit an indoor environment, while others an outdoor environment. Some techniques better fit a fixed environment and others a dynamic environment. Some are better suited to

rover robots, other ones to flying robots. This section gives an overview of these existing techniques, which can be divided in three groups:

- deterministic techniques;
- probabilistic techniques; and
- heuristics techniques

This section's aim is to highlight why the GA method, from the heuristics techniques, is an excellent candidate for the path planning of the UAV, which is a flying robot within a dynamic environment.

2.1 Deterministic Algorithms

The deterministic group covers the techniques that are using fixed cost equations. These always provide the same results for a given scenario. They tend to strive to get the shortest path, which may not be the optimal solution.

2.1.1 "A-Star" Family

The best known member of this family is Disjkstra's algorithm, which computes the cost of going through a cell, and every cell surrounding the current position, until it reaches the goal position. A second member, A-Star, improved this intensive search by adding a heuristic constant that directs the search only towards cells that are in the direction of the goal position. A third member, D-Star, further improved the search process by adding a second heuristic to minimize the recalculation in case of meeting unforeseen obstacles. A fourth member, AD-Star uses an inflation factor to get to a sub-optimal solution quickly, meeting real-time requirements. This paper will not go deeper in details for these techniques; [17] provides a good comparison of these four different techniques. However, from Fig. 1 [18], we can extrapolate how the search could get mathematically heavier as we move from a 2D-rover environment to a 3D-UAV world. Moreover, UAVs are traveling through large areas;

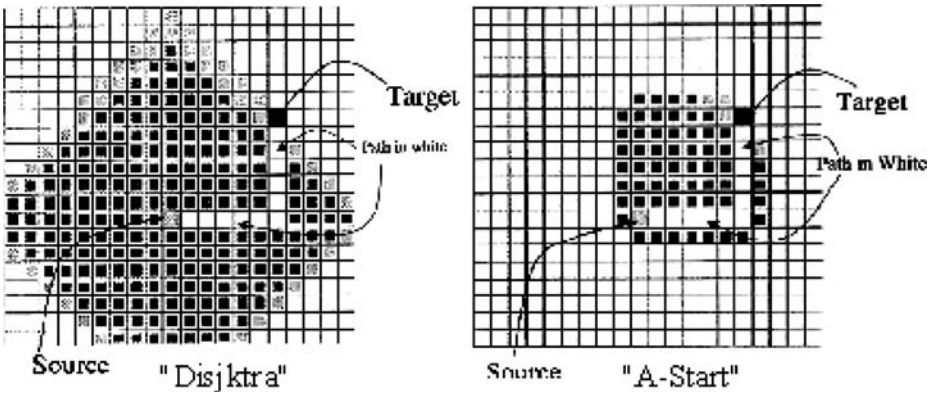

Fig. 1 Disjktra's and A-Star algorithm computation [18]. The Source is in an obstacle-free environment and tries to find the shortest path to go to the Target. *Black cells* represent those in which the algorithm had to evaluate the cost of passing through to find the best path

hence the algorithms have a larger number of cells to evaluate. As the evolution of the A-Star family shows, one needs heuristics to get away from this computational cumbersomeness.

2.1.2 Potential Field

The Potential Field concept involves representing obstacles by repulsive forces and the target by an attractive force. The robot needs to vector-sum the forces, and uses the resulting vector as its velocity direction command. The weakness of this concept is that there can be local minima of the potential field close to obstacles, in particular concave ones. Moreover, the computation complexity increases as we move to the 3D-UAV world, as vector summation has an additional component to deal with. Reference [19] elaborates some techniques (e.g., "the Mass–Spring–Damper System") that overcome the minima problem but introduce new problems (e.g., "high-resolution far from obstacles, but low-resolution near to obstacles").

2.1.3 Refined Techniques

Deterministic approaches tend to be computationally demanding because they perform many calculations to get exact predictable results with known situations. References [19, 20] developed refined techniques and demonstrated that a good way to get around the computational problem is to split the problem into two levels of path-definition: high path-definition for the neighbouring environment, within a specific radius from the UAV, and a coarse path-definition for environment outside that radius. These techniques aim at reducing the computation time as much as possible and to be able to re-compute the path in real-time to overcome unforeseen events.

2.2 Probabilistic Algorithms

The probabilistic group includes techniques that are using probabilities to model the uncertainties of the UAV world. Markov Decision Process (MDP) is one of these techniques. Reference [20] comments on the weakness of this method. What needs to be understood is that even if MDP addresses the uncertainties of the UAV world, MDP is a complex modeling process that does not take into consideration important constraints such as the time constraint, which is important when dealing with a team of UAVs. This research did not consider the probabilistic group because of the complexity of developing these models. This complexity will, most likely, prevent them from meeting the real-time requirement.

2.3 Heuristic Algorithms

The heuristic group covers the techniques that mix concepts from the deterministic and the probabilitic groups by using some heuristic representation of the problem. Two of these techniques used in the path planning are:

- Artificial Neural Networks, and
- Genetic Algorithm (GA)

2.3.1 Artificial Neural Networks

Artificial neural networks are based on the concept of the human cerebral cells, which can be conditioned or trained to behave as desired. The difficulty with this method is to get the proper training data to let the neural network learn the proper behaviour. One could want to train a neural network to be expert in path planning, but he would face the incapability, time wise or data wise, to train the system to overcome all possible unforeseen events.

2.3.2 Genetic Algorithm

The GA method is based on Darwin's theory of evolution where crossovers and mutations can generate better populations. GA uses randomness to cover the whole search area of potential solutions for the UAV path planning optimization problem and it uses determinism in keeping the best solutions. GA thus takes advantage of both deterministic and probabilistic approaches, while using simple operators to improve solutions: crossovers and mutations. GA provides high quality solutions [19, 21, 22]. Moreover, GA creates many possible solutions at a time and works on plain data without any a priori information [23]. However, because of the GA's slow iterative process (see the While-loop of Fig. 2), active research on this technique have not been pursued for the development of a real-time path planner.

Our research recognises the GA's advantages; and with its FPGA implementation, it overcomes the GA's computational disadvantage by considerably improving the processing time. This permits a GA-based real-time path planner for UAVs.

3 The Genetic Algorithm Used

An application of the Genetic Algorithm to path-planning, developed by Cocaud [24],—sponsored by the Defence R&D Canada (DRDC) in Ottawa—was used as an initial starting point for this research. This section describes this implementation by giving details on the following elements:

- Initial population characteristics,
- Selection technique,
- Crossover operator,
- Mutation operators,

Fig. 2 Pseudo-code of the genetic algorithm

```
GENETIC ALGORITHM;
    Initialize the population;
    Evaluate initial population;
    WHILE convergence criteria is NOT satisfied, DO
        Perform competitive Selection;
        Crossover solutions to generate new solutions;
        Mutate new solutions;
        Evaluate new solutions;
        Update old Population with new solutions;
    END-WHILE.
```

- Population update technique,
- Evaluation technique, and
- Convergence criteria

All details were set according to Cocaud's [24] experimentations and recommendations.

3.1 Population Characteristics

A population is composed of a certain number of solutions that are called chromosomes. Each chromosome represents a path-solution. It is composed of a certain number of genes that represent the transitional waypoints. It was found through experimentation that a population of path-solutions with around 30 to 40 different paths provided good solutions. The initial formulation of the Genetic Algorithm application also ensured that each potential path-solution does not exceed 20 transitional waypoints, but has at least 2 transitional waypoints. Each transitional waypoint is characterized by its three spatial coordinates. Starting waypoint and ending waypoint are fixed for all path-solutions, hence these are not considered as genes. Size of the population is fixed during the whole GA process.

The initial population is composed of a fixed number of path-solutions, each one having initially two randomly generated transitional waypoints. Thus, each initial path-solution has initially exactly four waypoints, starting and ending waypoints included.

3.2 Selection

The initial GA implementation of Cocaud [24] used the standard roulette-wheel method to select the parent path-solutions for the crossover phase. This selection method computes a selection probability for each path-solution. The probability is proportional to the path-solution fitness: the better is the path, the higher the probability is; the worse is the path, the smaller the probability is. Once the size of the probabilities has been set, a random–number–generator system provides a number that selects one of the path-solutions based on these probabilities.

3.3 Crossover

The crossover operation affects two selected path-solutions and permutes random sub-paths between the two solutions. A specific region is usually selected by one or two cutting-point(s) and only sub-paths within that region are permuted. The initial GA implementation always selects the middle of the path for the cutting-point, while cutting randomly before or after the middle waypoint for path with odd number of waypoints. Hence, each parent path is exchanging its transitional waypoints that are after the middle waypoint. Crossover is performed on 70% of the population, creating "children".

3.4 Mutation

Five different mutation methods were used by the initial GA implementation to speed-up the convergence: the perturb mutation, the insert mutation, the delete

mutation, the smooth-turn mutation, and the swap mutation. Mutations are done over 5% of the child population. Children are randomly selected with probabilities following a normal distribution.

3.4.1 Perturb Mutation

The perturb mutation selects randomly a waypoint of the selected child-path and modifies the values of its three coordinates randomly, while keeping them within an area of the size equal to 15% of the size of the full map (see Fig. 3). Before being accepted, the new waypoint is always checked to ensure it is not obstructed.

3.4.2 Insert Mutation

The insert mutation performs the same operation as the perturb mutation, except that it keeps the initial waypoint and adds a new waypoint after it in the path, while respecting the maximum number of waypoints per path (see Fig. 4).

3.4.3 Delete Mutation

The delete mutation randomly selects a waypoint and removes it from the current path, while respecting the minimum number of waypoints per path (see Fig. 5).

3.4.4 Smooth-Turn Mutation

The smooth-turn mutation evaluates the sharpness of each turn angle within a path. If it finds an angle that exceeds a specific threshold, a waypoint is added in such a way that the new angle is larger than the original one.

3.4.5 Swap Mutation

This mutation operator was used in [24] only when missions with multiple-objectives were considered. Because path planning usually addresses one objective at a time, we shall not consider this mutation hereafter.

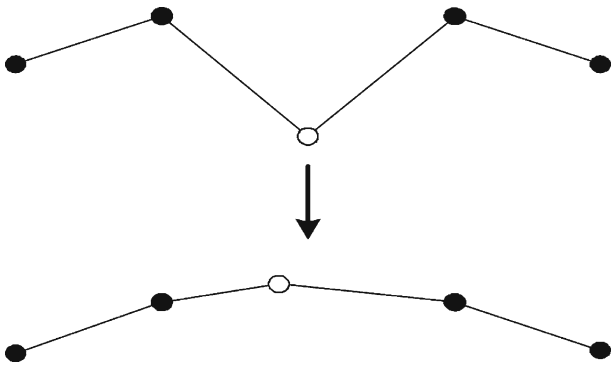

Fig. 3 Example of a perturb mutation [12]

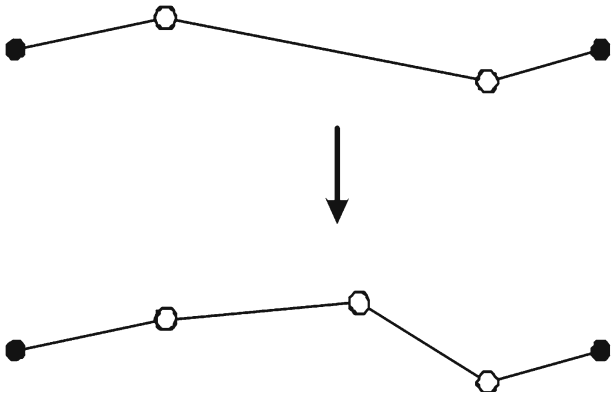

Fig. 4 Example of an insert mutation [12]

3.5 Population Update

This step consists in building a new population with the right number of path-solutions, using the children population and the old population. The new population used by the initial GA implementation is composed of: the entire children population; 5% of the best path-solutions from the old population; and it completes the population count by randomly selecting enough paths from the 95% leftover from the old population. This last selection uses the roulette-wheel technique. Each time one of the leftover solutions is selected, the roulette-wheel is recomputed without that selected solution.

3.6 Evaluation

Once the new population is created, each of its path-solutions has its fitness (goodness) reassessed. This evaluation phase was not modified from the original implementation of [24] in our research because the latter approach requires some improvements that will not be covered in this publication.

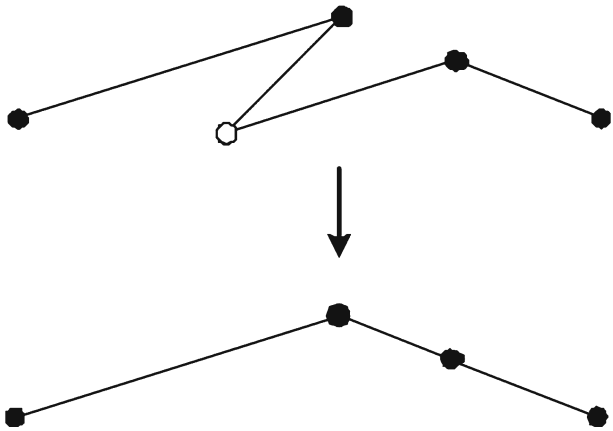

Fig. 5 Example of a delete mutation [12]

3.7 Convergence Criteria

The convergence criterion tells the GA when to stop its iterative process. More than one criterion could be used. The following three are commonly used:

- Fixed number of generations (iterations);
- Fixed level of fitness (goodness), calculated by the evaluation; and/or
- Fixed amount of time

The initial GA implementation stops the iterative process after 50 to 70 generations to have enough time for the GA to converge without wasting time with an already converged GA.

4 FPGA Design

A Field Programmable Gate Array (FPGA) consists of a programmable chip where the logic gates can be rearranged as needed by its user. This technology provides the flexibility to develop a processor dedicated to a specific operation instead of using a general purpose processor as used within personal computers. The chip area used by the processor can then be optimized by incorporating parallel processing. This is why some research [25–36] has already attempted different implementations of GA on FPGAs. However, none of them was applied to the UAV path-planning, where genes have a third dimension, namely the altitude or the Z axis, to take into consideration. None of them considered implementing elaborate mutation operators like the ones present in this research design. Nor did any of them use heavy parallelism while keeping the standard GA structure. On these three different points our research brings some new perspectives.

Section 4 will describe the details of the hardware design implementing the GA described in Section 3 on a FPGA circuit board. Figure 6 shows a block diagram of the design. The whole algorithm is running on the FPGA except for the evaluation phase. Evaluation runs on the PC, as does the simulation environment. This section will detail the design by covering the following points:

- Some structural modifications to the initial GA implementation;
- Control Unit;
- Receiving Unit;
- Selection Unit and Population Update;
- Crossover Unit;
- Mutation Unit; and
- Transmitting Unit

4.1 Structural Modification to the Initial GA Implementation

To integrate the FPGA hardware design shown in Fig. 6 to the initial PC-based GA executable, some modifications were required:

4.1.1 Fixed Population Characteristics

The first modification was to fix the population size to a predefined value of 32 path-solutions, in order to control the size of the hardware used from the FPGA board.

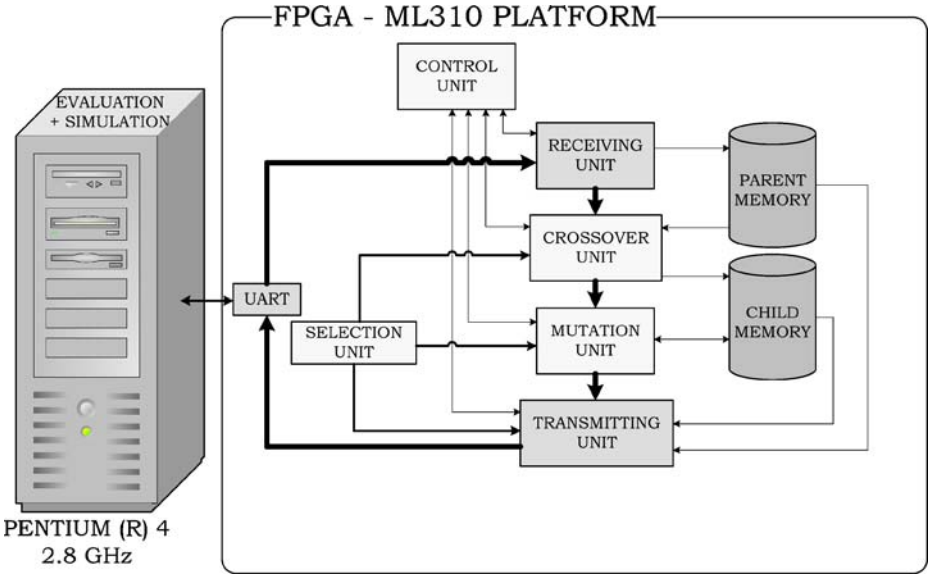

Fig. 6 This design overview shows the main hardware modules on the FPGA. The *dark thick arrows* provide an idea of the process flow

Fixing the size of the population also permitted fixing the number of the children generated at each iteration to 22 (69% of the old population). In the same line, the number of the best path-solutions kept from the old population, in the new population, was fixed to 2 (6% of the old population). Having these numbers fixed procured a time saving over having to dynamically calculate these values.

4.1.2 Sorting Algorithm

The second modification consisted in inserting a sorting algorithm after the evaluation phase. Having the path-solutions sorted based on their fitness score permits a reduction in the time required to find the best solutions within the population. This modification also permits fixing once and for all the roulette-wheel probabilities proportional to the rank of the solution; instead of re-computing the probabilities each time the fitness values were modified during the iterations.

After studying a number of sorting algorithms [37], the QuickSort algorithm was seen to be the fastest for the size of the population with which we dealt. Therefore, the QuickSort algorithm was implemented to minimize the amount of computations added to the evaluation phase.

4.2 Control Unit

The Control Unit turns "ON" one specific unit at a time. Once the specific unit completes its function, it sends back to the Control Unit an "Over" signal for the Control Unit to turn "OFF" that specific unit and turn "ON" the next unit in the GA process.

The Control Unit follows this process:

1. Waits on a "Listen" state to see if it will find a byte command on the UART that contain the number of chromosomes to be received;
2. When that command is received, it turns "ON" the Receiving Unit to receive the parent population from the PC;
3. The Crossover Unit is turned "ON" to crossover the parent population, hence creating the child population;
4. The Mutation Unit is turned "ON" to mutate 5 children from the child population; and
5. The Controller Unit turns "ON" the Transmit Unit to send the updated new population

Once the transmission is over, the PC computes the value of the individuals in the population; sorts the population based on the new fitness scores; and sends back the new population to the FPGA.

4.3 Receiving Unit

The Receiving Unit, once turned "ON" by the Control Unit, directly communicates with the UART to receive each path-solution of the parent population. The Receiving Unit saves the received population into the Parent Memory.

4.4 Selection Unit and Population Update

The Selection Unit is the only unit that is always "ON" independently from the Control Unit. The Selection Unit continuously provides 24 different random address numbers (11 for the crossover, 5 for the mutation and 8 for the Population Update), which change each clock cycle.

The Population Update consists of selecting the two best parents (predetermined addresses), the entire children population (predetermined addresses), and 8 random path-solutions (dynamic addresses) from the worst old parents to create the new parent population. Hence, the Selection Unit's random address numbers provide the only 8 dynamic addresses required for the Population Update to select the 32 path-solutions of the new parent population within one clock cycle.

4.5 Crossover Unit

The Crossover Unit is composed of 11 identical modules, which permute the way-points past the parent-paths middle-point (middle-point is rounded down for odd number of points) between the two parent-paths to generate two child-paths. The crossover module uses the same crossover technique as [24]; however, instead of performing the crossover sequentially, it is done entirely in parallel. In one clock cycle, the Crossover Unit uses the 11 outputs from the Selection Unit twice, and combines them differently. It results in having 22 parents that are available for the crossover within one clock cycle.

4.6 Mutation Unit

The Mutation Unit is composed of one Perturb Mutation module, two Inserter Mutation modules, and two Delete Mutation modules; all of these modules are running in parallel. The smooth mutation was not included because the insert mutation could produce a similar effect. The different mutation modules use the respective techniques from [24].

4.7 Transmitting Unit

The Transmitting unit (see the Transmitting Unit on Fig. 6) directly communicates with the UART to transmit all the 32 chromosomes of the new updated population.

5 Results Discussion

5.1 Testing Environment

To compare computation time between the initial GA implementation (described in Section 3) and this research's final FPGA implementation design (descried in Section 4), this research used a simulation environment of a canyon (see Fig. 7). The simulation environment provided a coarse atmospheric model with a constant unidirectional wind. The starting point was fixed at one extremity of the map and the

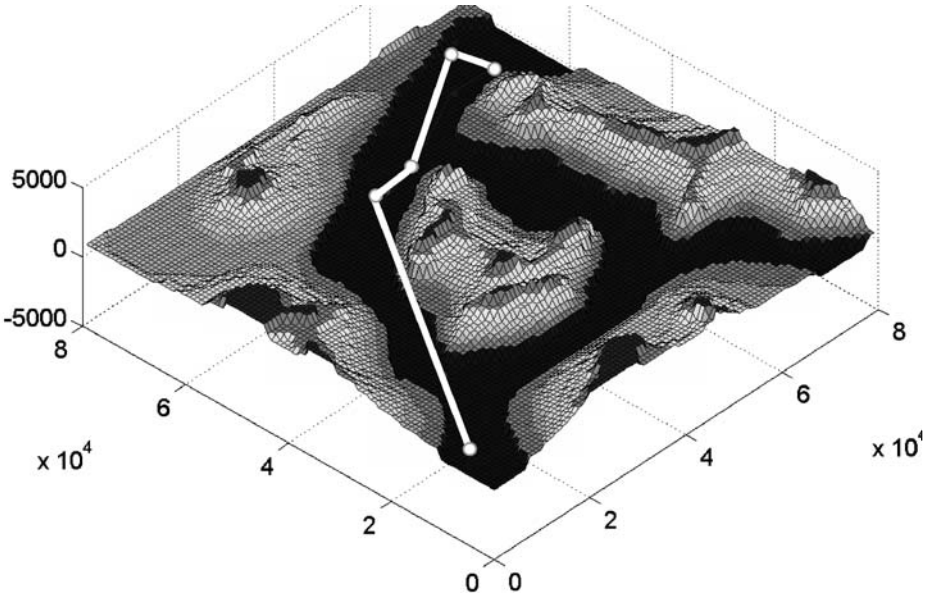

Fig. 7 Testing environment map (80 × 80 × 3.5 km), with a resolution of 500 × 500. The resolution of the Z axis was determined by the 16-bit integer used to define the height of each cell of the X and Y plan

Table 1 Timing results

GA phase	Initial PC-based GA	Modified GA with FPGA	Speed improvement
Selection and crossover	94 ms	8.85 μs	10,000×
Mutation	2 ms	18 μs[a]	111×
Population update	30 ms	600 ns[b]	50,000×
Evaluation	60 s/50 s	N/A[c]	1.2×[d]

The time values are averages captured after multiple trials. The PC used was a Pentium (R) 4 with a CPU of 2.8 GHz. The FPGA used was a Virtex-II Pro (2vp30ff896) with a system clock running at 100 MHz

[a]The Mutation was only tested within the simulation environment provided by Active-HDL version 7.1 (ALDEC, Inc's developing software). This restriction came from the fact that the current FPGA platform (the ML310) didn't have enough resources to host the full design; a FPGA ML410 (with twice the resources of the ML310) would have been required. However, before testing the Selection/Crossover system on hardware, testing was also performed with Active-HDL. Simulation results predicted properly the results obtained with the FPGA board connected to the C-code executable (running on the PC). We could therefore extrapolate this rectitude to the Mutation simulation prediction

[b]We can have access within on clock cycle to all the updated population (within 10 ns). Hence after 60 generations, we spent 600 ns to update the population. Some may want to multiply by the average number of waypoints-by-chromosome (5) this number since we can read only one waypoint by clock cycles (this is true only if we want to copy these information within a new memory space). Even with this multiplication, we still have a speed improvement of "10,000×"

[c]The evaluation phase was not implemented on FPGA in our research, because the original implementation approach [24] requires some improvements that will not be covered in this publication

[d]Sorting the population improved by 1.2 times the original computation time. This is due to the fact that a sorted population requires less search time to find the fitness statistic used to compute the resulting fitness score of each population member

ending point was fixed at the other extremity of the map. The scenario environment was fixed during the testing trials.

Multiple trials were first run with the C-code executable [24], as a stand-alone system. Average timing was extracted from these trials to provide the results exhibited in Table 1. Trials were run on a Pentium (R) 4 CPU 2.8 GHz.

The second set of trials incorporated the QuickSort algorithm within the C-code executable (running on the same PC); while the system, with the Selection and the Crossover Unit, was running on the FPGA (ML310 Xilink platform).

5.2 Timing Improvement Explanation

For both the crossover and the population update phases, the improvement reached a factor of 10,000. This is mainly due to the fact that the initial GA implementation had to sequentially select one solution at a time after generating a new random number for each selection, while this process is done in parallel in the FPGA. The recompilation of the selection-wheel was also a cumbersome task in the initial GA implementation. In the FPGA, these tasks are performed within one clock cycle (10 ns). For the crossover unit, the full parallelism used by duplicating the same crossover module (11 times) resulted in such a good improvement.

The mutation time improvement reaches a factor of 100. This is because the original GA implementation performed fewer mutations than the hardware design does. The difference in number of mutations comes from the fact that the initial GA used probabilities of a mutation to be applied to a child chromosome, while the hardware

design uses a fixed percentage of the child population to be mutated. However, being able to perform the 5 mutations simultaneously still provides an improvement by a factor of 100 in computation time over the initial GA implementation.

5.3 Future Work

The first step should be to optimize the evaluation phase for the members of the population. Secondly, a FPGA of the same calibre as the Virtex-4 LX200, which has 10 times more resources than the ML310 platform used here, should be acquired. If a hardware design could evaluate simultaneously the fitness of the 22 children solutions, we could theoretically improve the evaluation process time by a factor of at least 1,000 times. The whole GA path planning algorithm would then be totally running on the FPGA and would have the independence required to be connected to another UAV hardware platform. With that improvement, we would have a re-planning capability that provides a near optimal solution that, we estimate, would run in approximately 100 ms. Hence, a Sperwer (length of 4.2 m), flying at 150 km/h, would only cover a distance of 4.17 m before having its new directed path to follow. This distance is within the safety zone of the size of the aircraft considered by most of the path planning algorithms. We would then have a real-time path planner.

The third step would be to increase the confidence in the safety of the path planner by testing our system with a realistic tracker algorithm within a more sophisticated simulation environment. Then a live test with a real UAV would be realistic.

There is also room to optimize the current hardware design to save some area and some power dissipation, since our present design was oriented solely towards speed acquisition.

6 Conclusion

This research has successfully improved the processing time of a Genetic-Algorithm-based path planning algorithm. The FPGA implementation provides speed-ups of about 10,000 over the software executable. This improvement permits the UAV community to envisage research to incorporate a real-time path re-planning system, which can handle unforeseen events, into future UAV systems.

Acknowledgements F. C. J. Allaire thanks C. Cocaud, MEng from University of Ottawa, for his C code. It provided the foundation of this research project. F. C. J. Allaire thanks Dr. G. Labonté and Dr. M. Tarbouchi for their guidance throughout the research. F. C. J. Allaire also thanks J. Dunfield, MEng from Royal Military College of Canada, for the great support provided throughout the research. F. C. J. Allaire thanks M. Fricker, MEng from Royal Military College of Canada, for his support with paper redaction. Finally, F. C. J. Allaire thanks Dr. D. Al Khalili and J.-L. Derome for the technical support provided regarding FPGA technology.

References

1. Orgen, P., Winstrand, M.: Combining path planning and target assignment to minimize risk in a SEAD mission. In: AIAA Guidance, Navigation, and Control Conf., pp. 566–572. San Francisco, USA (2005)
2. Ye, Y.-Y., Min, C.-P., Shen, L.-C., Chang, W.-S.: VORONOI diagram based spatial mission planning for UAVs. Journal of System Simulation, China **17**(6), 1353–1355, 1359 (2005)

3. Yu, Z., Zhou, R., Chen, Z.: A mission planning algorithm for autonomous control system of unmanned air vehicle. In: 5th International Symposium on Instrumentation and Control Technology, pp. 572–576. Beijing, China (2003)
4. Vaidyanathan, R., Hocaoglu, C., Prince, T.S., Quinn, R.D.: Evolutionary path planning for autonomous air vehicles using multiresolution path representation. In: IEEE International Conf. of Intelligent Robots and Systems, vol. 1, pp. 69–76. Maui, USA (2001)
5. Rubio, J.C., Vagners, J., Rysdyk, R.: Adaptive path planning for autonomous UAV oceanic search missions. AIAA 1st Intelligent Systems Technical Conf., vol. 1, pp. 131–140. Chicago, USA (2004)
6. Shim, D.H., Chung, H., Kim, H.J., Sastry, S.: Autonomous exploration in unknown environments for unmanned aerial vehicles. In: AIAA Guidance, Navigation, and Control Conf., vol. 8, pp. 6381–6388. San Francisco, USA (2005)
7. Chaudhry, A., Misovec, K., D'Andrea, R.: Low observability path planning for an unmanned air vehicle using mixed integer linear programming. In: 43rd IEEE Conf. on Decision and Control. Paradise Island, USA (2004)
8. Theunissen, E., Bolderheij, F., Koeners, G.J.M.: Integration of threat information into the route (re-) planning task. In: 24th Digital Avionics Systems Conf., vol. 2, pp. 14. Washington, DC, USA (2005)
9. Rathinam, S., Sengupta, R.: Safe UAV navigation with sensor processing delays in an unknown environment. In: 43rd IEEE Conf. on Decision and Control, vol. 1, pp. 1081–1086. Nassau, USA (2004)
10. AIAA: Collection of technical papers. In: AIAA 3rd "Unmanned-Unlimited" Technical Conf., Workshop, and Exhibit, vol. 1, p. 586. Chicago, USA (2004)
11. Boskovic, J.D., Prasanth, R., Mehra, R.K.: A multi-layer autonomous intelligent control architecture for unmanned aerial vehicles. J. Aerosp. Comput. Inf. Commun., pp. 605–628. Woburn, USA (2004)
12. Zheng, C., Ding, M., Zhou, C.: Real-time route planning for unmanned air vehicle with an evolutionary algorithm. Int. J. Pattern Recogn. Artif. Intell. Wuhan, China **17**(1), 63–81 (2003)
13. Boskovic, J.D., Prasanth, R., Mehra, R.K.: A multi-layer control architecture for unmanned aerial vehicles. In: American Control Conference, vol. 3, pp. 1825–1830. Anchorage, USA (2002)
14. Rathbun, D., Kragelund, S., Pongpunwattana, A., Capozzi, B.: An evolution based path planning algorithm for autonomous motion of a UAV through uncertain environments. In: AIAA/IEEE Digital Avionics Systems Conf., vol. 2, pp. 8D21–8D212. Irvine, USA (2002)
15. Shiller, I., Draper, J.S.: Mission adaptable autonomous vehicles. Neural Netw. Ocean Eng., pp. 143–150. Washignton DC, USA (1991)
16. Fletcher, B.: Autonomous vehicles and the net-centric battlespace. In: International Unmanned Undersea Vehicle Symposium, San Diego, USA (2000)
17. Ferguson, D., Likhachev, M., Stentz, A.: A guide to heuristic-based path planning. In: International Conf. on Automated Planning & Scheduling (ICAPS), vol. 6, pp. 9–18. Monterey, USA, Work Shop (2005)
18. McKeever, S.D.: Path Planning for an Autonomous Vehicle. Massachusetts Institute of Technology, Massachusetts, USA (2000)
19. Judd, K.B.: Trajectory Planning Strategies for Unmanned Air Vehicles. Dept. of Mech. Eng., Brigham Young Univ., Provo, USA (2001)
20. Chanthery, E.: Planification de Mission pour un Véhicule Aérien Autonome, pp. 15. École Nationale Supérieur de l'Aéronautique et de l'Espace, Toulouse, France (2002)
21. Sugihara, K., Smith, J.: Genetic algorithms for adaptive planning of path and trajectory of a mobile robot in 2d terrains. IEICE Trans. Inf. Syst. **E82-D**(1), 309–317 (1999)
22. Pellazar, M.B.: Vehicle route planning with constraints using genetic algorithms. In: IEEE National Aerospace and Electronics Conf., pp. 111–119 (1998)
23. Mostafa, H.E., Khadragi, A.I., Hanafi, Y.Y.: Hardware implementation of genetic algorithm on FPGA. In: Proc. of the 21st National Radio Science Conf., pp. 367–375. Cairo, Egypt (2004)
24. Cocaud, C.: Autonomous Tasks Allocation and Path Generation of UAV's. Dept. of Mech. Eng., Univ. of Ottawa, Ontario, Canada (2006)
25. Tachibana, T., Murata, Y., Shibata, N., Yasumoto, K., Ito, M.: A hardware implementation method of multi-objective genetic algorithms. In: IEEE Congress on Evolutionary Computation, pp. 3153–3160. Vancouver, Canada (2006)
26. Tachibana, T., Murata, Y., Shibata, N., Yasumoto, K., Ito, M.: General architecture for hardware implementation of genetic algorithm. In: 14th Annual IEEE Symposium on Field Programmable Custom Computing Machine, pp. 2–3. Napa, USA (2006)

27. Tang, W., Yip, L.: Hardware implementation of genetic algorithms using FPGA. In: 47th Midwest Symposium on Circuits and Systems, vol. 1, pp. 549–552. Hiroshima, Japan (2004)
28. Hamid, M.S., Marshall, S.: FPGA realisation of the genetic algorithm for the design of grey-scale soft morphological filters. In: International Conf. on Visual Information Eng., pp. 141–144. Guildford, UK (2003)
29. Karnik, G., Reformat, M., Pedrycz, W.: Autonomous genetic machine. In: Canadian Conf. on Elect. and Comp. Eng., vol. 2, pp. 828–833. Winnipeg, Canada (2002)
30. Skliarova, I., Ferrari, A.B.: FPGA-based implementation of genetic algorithm for the traveling salesman problem and its industrial application. In: 15th International Conf. on Industrial and Eng. Application of Artificial Intelligence and Expert Syst. IEA/AIE, pp. 77–87. Cairn, Australia (2002)
31. Lei, T., Ming-cheng, Z., Jing-xia, W.: The hardware implementation of a genetic algorithm model with FPGA. In: Proc. 2002 IEEE International Conf. on Field Programmable, pp. 374–377. Hong Kong, China (2002)
32. Hailim, J., Dongming, J.: Self-adaptation of fuzzy controller optimized by hardware-based GA. In: Proc. of 6th International Conf. on Solid-State and IC Technology, vol. 2, pp. 1147–1150. Shanghai, China (2001)
33. Aporntewan, C., Chongstitvatana, P.: A hardware implementation of the compact genetic algorithm. In: Proc. of the IEEE Conf. on Evolutionary Computation, vol. 1, pp. 624–625. Soul (2001)
34. Shackleford, B., Snider, G., Carter, R.J., Okushi, E., Yasuda, M., Seo, K., Yasuura, H.: A high-performance, pipelined, FPGA-based genetic algorithm machine. Genetic Programming and Evolvable Machine **2**, 33–60 (2001)
35. Scott, S.S., Samal, A., Seth, S.: HGA: a hardware-based genetic algorithm. In: ACM 3rd International Symp. on FPGA, pp. 53–59. Monterey, USA (1995)
36. Emam, H., Ashour, M.A., Fekry, H., Wahdan, A.M.: Introducing an FPGA based genetic algorithms in the applications of blind signals separation. In: Proc. 3rd IEEE International Workshop on Syst.-on-Chip for Real-Time Appl., pp. 123–127. Calgary, Canada (2003)
37. Thiébaut, D.: Sorting algorithms. Dept. of Comp. Science, Smith College, Northampton, MA, USA (1997) Available: http://maven.smith.edu/~thiebaut/java/sort/demo.html (1997). Accessed 23 April 2008