Embedded harmonic control for trajectory planning in large environments

Cesar Torres-Huitzil, Bernard Girau, Amine Boumaza, Bruno Scherrer

Embedded harmonic control for trajectory planning in large environments

Cesar Torres-Huitzil

Polytechnic University of Victoria, Mexico
ctorresh@upv.edu.mx

Bernard Girau, Amine Boumaza,
Bruno Scherrer
LORIA – INRIA Nancy Grand Est, France
bernard.girau@loria.fr

Abstract

This paper presents an embedded FPGA–based architecture to compute navigation trajectories along a harmonic potential. The goals and obstacles may be changed during computation. Large environments are split into blocks. This approach, together with the use of an increasing precision, enables an optimization of the overall computation time that is theoretically and experimentally studied. Implementation results confirm outstanding speedup factors.

1. Introduction

Trajectory planning consists in finding a way to get from a starting position to a goal position while avoiding obstacles within a given environment or navigation space. Harmonic functions may be used as potential fields for trajectory planning [1]. Such functions do not have local extrema (unlike other potential based methods as in [5]), so that navigation algorithms may reduce to *locally* ascend the potential until they reach a *global* maximum, when obstacles correspond to minima and goals correspond to maxima.

Harmonic control has had some impact on the robotics community [1,2,4,6,8,9]. This paper presents an embedded implementation of this navigation method on reconfigurable digital circuits. After the iterated computation of the harmonic function, our implementation locally computes the direction to choose to get to the goal at any point of the environment. Dynamic changes in this environment may be taken into account. Our implementation has been designed to deal with very large environments while optimizing computation time. To do so, such environments may be split into several so-called blocks, and iterated updates are performed in a block-synchronous mode that takes advantage of large embedded SRAM memory resources. Moreover, an increasing precision is used throughout the convergence process, so as to further optimize computing times. Besides all implementation works, we have carefully justified our algorithmic and technological choices through both theoretical and empirical studies of the required precisions and convergence times. Section 2 describes the principles of harmonic functions and their use for trajectory planning. Section 3 introduces our block-synchronous algorithm, and its optimization with respect to precision and convergence rate. Section 4 describes its implementation architecture and results.

2. Harmonic control

A function u is harmonic when it satisfies Laplace's equation within its open definition domain:

$$\Delta u = \sum_{i=1}^{n} \frac{\partial^2 u}{\partial x_i^2} = 0 \qquad (1)$$

Harmonic functions have interesting properties: a non-constant harmonic function attains its maximum and minimum values on the boundary of its definition domain, and there can be no local minimum or maximum inside a bounded region this domain.

Planning trajectories with harmonic functions consists in finding the function u that is harmonic on the navigation space and that has value 0 on obstacle positions and value 1 on goal positions. Then a simple ascent along the gradient of u provides a trajectory towards a given goal from any starting position. The properties of harmonic functions ensure that such a path exists and it is free of local optima. Using a Taylor approximation of the second derivatives we derive the following discrete form:

$$0 = \Delta_\delta u(x,y) = \frac{1}{\delta^2}\left[u(x+\delta,y)+u(x-\delta,y)+u(x,y+\delta)+u(x,y-\delta)-4u(x,y)\right] \qquad (2)$$

where δ is the sampling of the grid nodes. This equation can be solved using relaxation methods that iteratively replace each node value with the simple

average of its four neighbours until convergence. Figure 1 shows different trajectories generated by simulations.

The main properties of harmonic control are:

Global navigation: Complete trajectories may be generated towards a goal position from anywhere in the environment, since there are no local minima.

Dynamic navigation: Unexpected updates of goals and obstacles (dynamic environments or on-line exploration [9]) may be considered, since harmonic functions are computed by iterative relaxation methods.

Parallel computation: These iterative computations are massively distributed. Computing node values only require local information of the neighbouring nodes

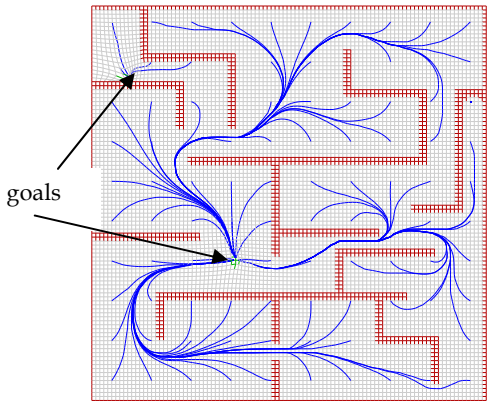

Fig. 1. Generated trajectories (100x100 grid, equally spread starting nodes, two goals)

3. Algorithm and optimization

Our aim is to design an embedded system for robot navigation, improving computation speed and power consumption. Besides, scalability and computation precision appear as critical issues for harmonic control.

3.1 Technological and algorithmic choices

3.1.1 Serial arithmetic and precision. Though our model needs to compute maxima that may only be computed in a MSBF mode, we mostly use standard LSBF serial operators to optimize the required area. Nevertheless, our implementation simultaneously handles a read access in MSBF mode to detect local maxima, taking advantage of the two simultaneous read addresses of the SRAM blocks. Serial arithmetics was choosen in order to privilige distributed low-area operators inside the FPGA and since bit-parallel hardware resources are not considerable in the computation of teh harmonic function for trajectory planning. Another advantage of serial operators is to be able to handle large precisions without an increased implementation area.

Precision has already been mentioned as a major limitation for analog implementations [8]. In case of insufficient precision, large areas of the grid may share the same value, hence a null gradient that results in incomplete trajectories. Connolly [1] argues that the precision should at least represent $1/N$, where N is the total number of grid nodes. We argue that $1/N$ may not be a sufficient precision. More precisely, the precision might have to represent at least $1/2^{O(L)}$ (therefore requiring some $O(L)$ bits), where L is the maximum trajectory length in the environment. To prove this, it is sufficient to develop equation (2) within a "corridor" of length L and width 1, with an obstacle on the left ($x_0 = 0$), and the goal at the other side ($x_L = 1$).

Nevertheless, the study of the required precision should take "likely" environments into account. We have carried out numerous experiments with large randomly generated mazes. It follows that in most environments, the maximum distance L to the goal is close to the square root of the environment size, and that a precision proportional to $1/L$ (i.e. a number of bits proportional to $\log(L)$) is generally sufficient to ensure that the computation of the harmonic function converges such that no local minimum or maximum exists (i.e. a trajectory is found from any node).

3.1.2 Block-parallel computation. Despite the use of small serial operators for low-area hardware solutions at the cost of performance, the size of the discretized environment we are able to map in a fully parallel way onto FPGAs is limited (around 50x50 nodes in our preliminary work in [3]). To handle much larger environments (or finer discrete resolutions), we propose a block-synchronous (or block-parallel) implementation: the environment is partitioned into several blocks, each block of nodes being implemented in a fully parallel way by the FPGA while the different blocks are sequentially handled. Moreover up to I consecutive iterations are performed for each block before handling the next block. As a mean to counterbalance the reduction in performance due to serial operators compared to bit-parallel resources, a detailed study to optimize I and the increasing scheme of the precision is performed.

3.1.3 Increasing precision. The computation time of a serial operator depends on the precision, and several iterations are required to let our system converge to a good approximation of the expected harmonic function. We propose to use an increasing precision to optimize the convergence time. The first iterations are performed

with a chosen reduced precision. When the whole system has converged for a given precision (potentials within all blocks of nodes have been stabilized), iterated updates start again with an increased precision. This is repeated until the necessary precision of the harmonic function estimation is reached (this is the case when *no local minimum or maximum exists*, therefore we stop the algorithm when no such local extremum is detected).

With this approach, first iterations are faster, since they handle reduced precisions with serial operators. Next iterations use an increased precision, and the additional convergence time only corresponds to computing the additional bits of the harmonic function estimation. Moreover the convergence of each block is reached sooner: *I* consecutive iterations are performed for each block before handling the next block, except if the computations within this block converge before the *I* iterations, which may happen (and is detected) more rapidly with reduced precisions.

3.2 Optimization

In this section, we study the computation time of this algorithm, so as to optimize *I* and the increasing scheme of the precision.

3.2.1 Convergence: theoretical results. Though not converging in general, we have proved that the iterated estimation H_k of the harmonic function derived from equation (2) converges in some weak sense. Namely it leads to oscillations around the fixed point. We have also determined the convergence rate: for any $\lambda > 0$

$$k \geq \frac{\log\left(\frac{\lambda e}{(1-\gamma)\|H_0 - H_*\|}\right)}{\log \gamma} \Rightarrow \|H_k - H_*\| \leq \frac{(1+\lambda)e}{1-\gamma}$$

where *e* is the maximum round-off error (precision), H_* is the fixed point of the process, H_0 is the starting point and γ is its contraction coefficient. We may consider λ as a margin that is added to the $\frac{e}{1-\gamma}$-wide asymptotical interval around H_*. This margin defines a wider interval where H_k finally lies.

Sketch of proof:
1. The computation of the harmonic function is contracting with coefficient $\gamma \approx 1 - \frac{\pi^2}{4}\left(\frac{1}{N^2} + \frac{1}{M^2}\right)$ in a *N*×*M* environment (proof outline: the update equation uses a matrix that is substochastic, and that corresponds to a vanishing Markov chain for which we know an upper bound of the biggest eigenvalue).
2. Taking into account both the round-off error (maximum *e*) at each iteration and the properties of a contracting function, we prove by recurrence that

$$\|H_k - H_*\| \leq \frac{e}{1-\gamma} + \gamma^k \|H_0 - H_*\|$$

so that the process is asymptotically $\frac{e}{1-\gamma}$-close to its fixed point. When it is $\varepsilon = \frac{(1+\lambda)e}{1-\gamma}$-close, we finally get the expected result.

3.2.2 Experimental convergence. Experimentally, the convergence of the iterated computation of the harmonic function (derived from equation (2)) not only converges in a weak sense, but fully converges whatever the fixed precision. To validate this assertion, we have carried out numerous experiments with various environments. They establish that the number of required iterations linearly depends on the number of nodes in the environment, which validates the above estimation of *k* (that depends on $1/\log(\gamma)$ which is roughly proportional to N^2 and M^2).

3.2.3 An increasing precision algorithm. We now consider the case where the harmonic function is iteratively computed using arithmetics with different precisions: $e_1 = 2^{-p_1}, e_2 = 2^{-p_2}, \ldots, e_T = 2^{-p_T}$. Write $H^{(0)}$ the initial estimate. Writing k_i the number of required iterations to reach precision p_i from precision p_{i-1}, we show that k_1 may be estimated as an affine function of the initial number of bits p_1:

$$k_1 \approx K\left(p_1 \log(2) + \log(1/\lambda) + \log(1/K) + \log\left(\|H^{(0)} - H_*\|\right)\right)$$

and each k_i may be estimated as an affine function of the increase in the number of digits: for $i \geq 2$

$$k_i \leq K\left((p_i - p_{i-1} + 1)\log(2) + \log(1/\lambda)\right)$$

From this, we derive that the increasing precision approach roughly divides the computation time by 2 when very large precisions are required, assuming that the precision increase is arithmetic $p_i = i\frac{p_{\max}}{T}$ (the detailed proof is not given here, it involves a first-order development of the sum of the k_i w.r.t. $p_{\max}$).

3.2.4 A block-synchronous algorithm. We have extended the above result when the environment is split into *B* blocks and all computations are performed with

serial arithmetics. Nevertheless, the optimization provided by our approach does not fully appear in this theoretical study, but when experimentally analyzing the overall convergence time. Indeed, besides allowing larger environments, this approach takes advantage of "still" blocks to converge faster: *I* iterations are performed only for blocks that have not converged so far, whereas in most experiments, large parts of the environment stay unchanged (still) for several iterations while distant blocks slowly propagate the changes.

We have performed tests on a PC in order to compare the overall number of computations when one uses several blocks, to the case where there is only one block. These experiments show that block-partitioning speeds up the computations, although the successive iterations are performed by a block without updating the neighbouring blocks. This speedup is observed provided that *I* is not too large and an early detection of stabilization is performed within each block. Moreover the speed-up increases with larger blocks. Following our experiments, the block-synchronous approach that uses an increasing precision finally divides the computation time by some coefficient between 2 and 4.

4. Hardware implementation

Though harmonic control has been widely used in robotics, few hardware implementations have been proposed. Their technological choices are mostly motivated by the fact that analog resistive grids may easily compute the harmonic function as in equation (2). For example in [7] an analog implementation of a 16x16 grid is proposed. The main limitation of this work is the precision (as for most analog implementations). To our knowledge, we propose here the first digital FPGA-based implementation.

4.1 General architecture

Figure 2 illustrates the general architecture of the implementation of harmonic control in a given environment. Since the environment is split into several blocks, this architecture mostly consists of a grid of *nxm* identical node modules surrounded by border node modules, a control module, a decision module, and a module to interact with the robot.

Each node module computes its corresponding node value within the currently handled block. The control of these computations are synchronized in the whole block so that node modules serially communicate their values to their neighbours. The node modules are split in groups of 3x6 nodes that share common storage resources (a single dual port SRAM block together with the counters used for address control).

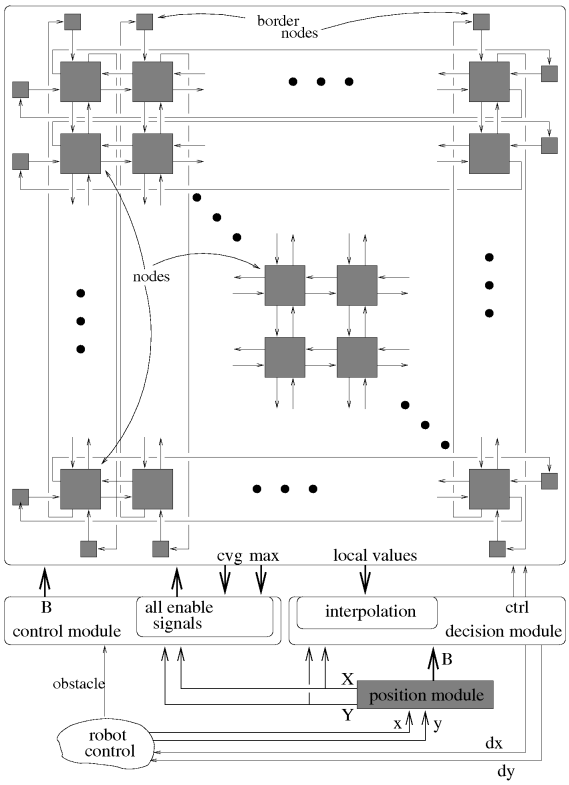

Fig. 2. General architecture

The border nodes are simpler. They store the values of the immediate neighbours of the most outer nodes within each block, and they serially generate these values when required. We handle the addressing scheme so that the values stored within each of the 4 possible borders are updated when the block that contains them is being computed. These updates require long-range connections from the node modules on each side of the block to the opposite border nodes.

The interaction with the robot includes a position modules, which role is mainly to compute the coordinates (B,X,Y) of the closest grid point (block, node) around the real coordinates (x,y) of the robot in its environment.

The control module generates the enable signals that are sent to all node modules to control their individual behaviour when an asynchronous event occurs (convergence of the computation of the harmonic function, or early detection of the convergence of the computation within the currently handled block, or detection of an unknown obstacle by the robot). It also computes the number B of the current block, and it handles the different counters such that an increasing precision is used until global convergence.

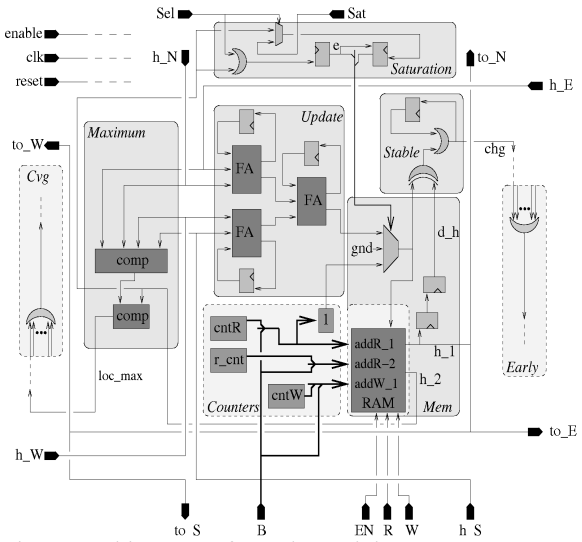

Fig. 3. Architecture of a node module

The decision module collects the navigation information that are provided by the node modules (according to block B). Then it extracts the information that corresponds to the neighbourhood of the current position coordinates (B,X,Y) of the robot, and it performs a linear interpolation of the potential values around (B,X,Y) to compute the navigation direction after convergence of the iterations: it corresponds to the maximum slope among the four triangles that are defined by the node and two of its immediate neighbours.

4.2 Node module implementation

The architecture of a node module uses 1-bit inputs and outputs to exchange data among nodes and with the global modules. Inputs are mainly used to receive the 4-connected neighbouring node values (signals `h_N`, `h_E`, `h_W`, `h_S`) and global control signals (standard signals `clk`, `reset`, `enable`, signals `Sel` and `Sat` to indicate obstacle/goal changes, and SRAM controls `EN`, `R`, `W`). The local value `h` of the harmonic function is sent to all neighbours (signals `to_N`, `to_E`, `to_W`, `to_S`) and to the global interpolation module so as to compute the navigation orientation if the robot is found to be located in the area that corresponds to the local node.

The proposed hardware node module is constituted by five main sub-modules. Figure 3 shows a block diagrama of this architecture, as well as its interaction with shared resources that are surrounded by dotted lines (the local counters and RAM modules are shared by a group/cluster of 3x6 node modules, the `Early` module and `Cvg` modules are shared by all nodes of the block).

`Update`: This module iteratively computes the harmonic function value $h_{(i,j)}$ where (i,j) are the coordinates of the node in the environment. As described in **2.**, each iteration computes:

$$h_{(i,j)}(t+1) = \frac{h_{(i-1,j)}(t) + h_{(i+1,j)}(t) + h_{(i,j-1)}(t) + h_{(i,j+1)}(t)}{4}$$

The output value is sent to the RAM with a write address delayed by 2 clock cycles (division by 4).

`Stable`: This module detects the local convergence of this computation (stabilization), by serially comparing the output of the iterated computation to the stored value. This local convergence test is then sent to a global `OR` gate (in the `Early` module) to disable the computation loop of the block when early stabilization has been detected before *I* iterations.

`Maximum`: This module checks for the presence of a local maximum. It uses a comparison between all neighbouring values and a comparison with the local value so as to determine whether the local node corresponds to a local maximum. This local information is sent to the `Cvg` module that uses a global `OR` gate so as to check for the presence of any local maximum in the current block.

`Mem and Saturation`: The node receives orders to behave as an obstacle or a goal through the `Saturation` module, and communication with the dual port SRAM block that stores the node value is controled by the `Mem` module. A multiplexer selects the correct value (output of `Update`, 0 or 1) with respect to a control given by the `Saturation` module that memorizes the `Sat` value to be the constant value of the grid point when the node is selected by the global control module (signal `Sel`). Since multiple blocks m handled, these constant values must also be stored in and retrieved from the RAM. In order to do that, we add a special bit (the MSB) to the values stored in memory (this bit is set to 1 when the local value is constant).

Counters: This module is shared by 18 nodes, since node modules are gathered together to form a 2D grid of 3x6 clusters. The main reason to group in such a configuration is due to the 18-bit width of the shared block ram. The depth of the block RAMs is 1K. It allows handling a wide range of arithmetic precisions such as 64-128 bits per word without modifying the memory organization. Module `Counters` generates the read (resp. write) addresses for the dual port RAM with counters `cntR`, `r_cnt` (resp. `cntW`).

4.3 Implementation results

This work uses a PCI bus board with three FPGAs, the largest one being a Virtex-4 XC4VLX160ff1513-12 FPGA from Xilinx, that contains 135,168 logic cells. The design was synthesized, placed and routed automatically in Xilinx Foundation ISE 9.2i. Each node module requires 26 logic cells, and each cluster of 18 node modules requires 458 logic cells (counters included). Using 264 dual port SRAM, the whole architecture may implement a 72x66 grid on less than 92 % of the logic cells. We use the remaining 24 SRAM blocks to implement the storage facilities of the 276 border blocks. A few slices are sufficient to handle addresses for these special blocks. Then the Control module (1934 slices) and the Decision module are added ($3p$ + 1235 slices for a p-bit precision). So that for example 97 % of this FPGA is finally used for the implementation of the algorithm with 4 blocks (19008 nodes) and p=255 (see below for the speed).

Software implementations of the harmonic function computation on a microprocessor based computer, Pentium 4,2 GHz, require around 200 μs per iteration with a 72x66 block. In the proposed hardware implementation, p+2 clock cycles are required per iteration for precision p, with an estimated clock frequency of 150 MHz. Thus, the implementation on the Virtex-4 provides a speed factor up to 200x (for a 128-bit precision that corresponds to some average-sized environments in our reported experiments). Moreover larger precisions may be handled by the proposed serial implementation when few blocks are used (up to 1K bits when only one block is used).

Following our experiments in section 3, our block-synchronous approach together with the increasing precision reduces the required number of iterations before convergence, so that the final speed factor is up to 800x. As an example, with a not too complex maze with 19008 nodes divided into 4 blocks, a p=255 precision, and I=6 consecutive iterations for each block at most, a speedup of 540x is obtained. Such environments experimentally require about 20000 iterations for the computation of the harmonic function (some 4 seconds on a PC). Therefore this acceleration fully enables the system to react to changes of goal and obstacles in real-time.

6. Conclusion

We have described the FPGA implementation of an architecture that computes trajectories along a harmonic potential, so as to solve the navigation problem in robotics. The goals and obstacles may be changed during computation. The proposed architecture uses a massively distributed grid of identical nodes that interact with each other within mutually dependant serial streams of data to perform iterative updates of the local harmonic function values until global convergence. When the environment size is too large for a fully parallel implementation on the used FPGA, our implementation takes advantage of the available SRAM to handle larger environments that are partitioned into blocks. The proposed architecture also introduces the use of an increasing precision. This approach enables an optimization of the overall computation time. We have justified our technological and algorithmic choices through both theoretical and experimental studies, with respect to both the block-synchronous approach and the increasing precision technique. Implementation results finally validate our approach in terms of parallelism, scalability, precision and speedup. The main perspective of this work is to extend it to optimal control, a more generic (and tunable) trajectory planning method, that uses similar computations without requiring such huge precisions, so that more blocks might be handled.

References

[1] Connolly, J.C. and Grupen, R. (1993). On the applications of harmonic functions to robotics. *Journal of Robotic and Systems*, 10(7):931–946.

[2] Feder, H.J.S. and Slotine, J.J.E. (1997). Real-time path planning using harmonic potentials in dynamic environments, *Proc. of the Int. Conf. Robotics and Automation*, IEEE CNF.

[3] Girau, B. and Boumaza, A. (2007). Embedded harmonic control for dynamic trajectory planning on FPGA, *Proc. of Int. Conf. on Artificial Intelligence and Applications.*

[4] Kazemi, M.; Mehrandezh, M. and Gupta, K. (2005). An incremental harmonic function-based probabilistic roadmap approach to robot path planning, *Proc. of the IEEE Int. Conf. Robotics and Automation*, IEEE CNF.

[5] Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotic Research*, 5(1):90–98.

[6] Masoud, A.A. and Masoud, S.A. (2002). Motion planning in the presence of directional and obstacle avoidance constraints using nonlinear anisotropic, harmonic potential fields. *IEEE Transactions on Systems, Man, & Cybernetics, Part A: systems and humans*, 32(6):705–723.

[7] Stan, M.; Burleson, W.; Connolly, C. and Grupen, R. (1994). Analog VLSI for robot path planning. *Journal of VLSI Signal Processing*, 8(1):61–73.

[8] Tarassenko, L. and Blake, A. (1991). Analogue computation of collision-free paths, *Proc. of the Int. Conf. on Robotics and Automation*, pages 540–545, IEEE CNF.

[9] Zelek, J.S. (1998). Complete real-time path planning during sensor-based discovery, *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and systems*