Embedded harmonic control for dynamic trajectory planning on FPGA

Bernard Girau, Amine Boumaza

EMBEDDED HARMONIC CONTROL FOR DYNAMIC TRAJECTORY PLANNING ON FPGA

Bernard Girau
Cortex team - LORIA INRIA-Lorraine
Nancy - France
email: Bernard.Girau@loria.fr

Amine Boumaza
Maia team - LORIA INRIA-Lorraine
Nancy - France
email: Amine.Boumaza@loria.fr

Abstract

This paper presents a parallel hardware implementation of a well-known navigation control method on reconfigurable digital circuits. Trajectories are estimated after an iterated computation of the harmonic functions, given the goal and obstacle positions of the navigation problem. The proposed massively distributed implementation locally computes the direction to choose to get to the goal position at any point of the environment. Changes in this environment may be immediately taken into account, for example when obstacles are discovered during an on-line exploration. The implementation results show that the proposed architecture simultaneously improves speed, power consumption, precision, and environment size.

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 have been proposed as potential fields for trajectory planning in [3]. This "harmonic control" approach was motivated by the fact that harmonic functions do not have local extrema (unlike other potential based methods as in [9]). In this approach, obstacles correspond to maxima of the potential, while goals correspond to minima. Control algorithms then reduce to *locally* descend the potential until they reach a *global* minimum.

Harmonic control has had some impact on the robotics community [10, 16, 1, 5, 7, 8, 12, 14]. Nevertheless, very few hardware implementations have been proposed. They are usually analog, therefore they suffer from a very long and complex design process, and a lack of flexibility (environment size, precision). This paper proposes an embeddable digital implementation on reconfigurable circuits, so as to reduce design time and to improve flexibility.

The proposed implementation maps the massively distributed structure of the space-discretized estimation of the harmonic function onto the circuit. An area-saving serial arithmetic is used, within a global scheme that simultaneously ensures pipelining and parallelism of the iterative computations. The decision process is also taken into account, through local estimations of the direction to take from any point.

Though the speed performance reported in this paper outperform software implementations, the main advantages of the described architecture is to provide low-power and scalable solutions that are able to handle the very large precisions required by harmonic control within large navigation environments.

Section 2 describes the principles of harmonic functions and of their use for trajectory planning. Section 3 summarizes the advantages of hardware parallel implementations on FPGAs for embedded navigation in autonomous systems, and it justifies the choice of serial arithmetics, especially with respect to the precision requirements of harmonic control. The proposed hardware architecture and its implementation results are described in section 4.

2 Harmonic control

In this part, we begin by a brief reminder of harmonic functions and some of their properties after which we will discuss their application to the navigation problem.

2.1 Harmonic functions

Let Ω be an open subset of $\mathbb{R}^n$, $\partial\Omega$ its boundary and $\bar{\Omega}$ its closure such that $\bar{\Omega} = \Omega \cup \partial\Omega$.

Definition Let $u : \bar{\Omega} \rightarrow \mathbb{R}$ be a real function, twice continuously differentiable, and $\Omega \subseteq \mathbb{R}^n$ with $n > 1$. The function u is harmonic iff $\Delta u = \sum_{i=1}^{n} \frac{\partial^2 u}{\partial x_i^2} = 0$.

This equation is known as Laplace's equation. Harmonic functions satisfy interesting properties:

- The maximum principle states that: if u is a non-constant continuous function on $\bar{\Omega}$ that is harmonic on Ω, then u attains its maximum and minimum values over $\bar{\Omega}$ on $\partial\Omega$.
- Applying the divergence theorem on harmonic functions, the following equation holds: $\int_s \vec{\nabla u} \cdot \vec{n}\, ds = 0$ where s is the boundary of a closed region strictly in Ω and $\vec{n}$ is a normal vector of s. This equation expresses that the normal flux of the gradient vector field through the region bounded by s is zero. It follows that there can be no local minimum or maximum of the potential inside a bounded region of Ω.

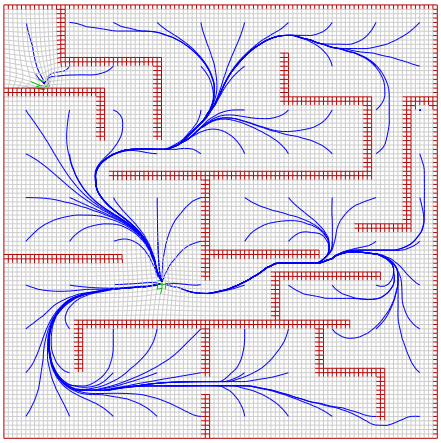

Figure 1. Trajectories generated by harmonic control (100×100 grid, equally spread starting points, two goals).

2.2 Application to navigation

To solve the navigation problem using harmonic functions, we consider the problem as a Dirichlet problem: its solution is to find the function u that is harmonic on Ω (the navigation space) and that satisfies boundary conditions on $\partial\Omega$ (obstacles and navigation goal):

$$\begin{cases} \forall x \in \Omega, & \Delta u\,(x) = 0 \\ \forall x \in \partial\Omega, & u\,(x) = g\,(x) \end{cases}$$

where the function $g : \partial\Omega \rightarrow \mathbb{R}$ represents boundary conditions on $\partial\Omega$. These conditions define the values of the navigation function on obstacles and goals. Without loss of generality we choose $g(x) = 1$ for obstacles and $g(x) = 0$ for goals. Solving the Dirichlet problem consists in finding the function u that is harmonic on Ω and that has value 1 on obstacle positions and value 0 on goal positions.

The navigation problem is then solved as follows: a simple descent 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.

2.2.1 Numerical method to solve Laplace's equation

In this part we consider the case where $\Omega = \mathbb{R}^2$. Traditionally, Laplace's equation is solved using methods from finite differences on a regular grid (discrete sampling of Ω). Using a Taylor approximation of the second derivatives we obtain the following discrete form of $u\,(x_1, x_2)$:

$$\Delta_\delta u\,(x_1, x_2) = \frac{1}{\delta^2}\,[u(x_1 + \delta, x_2) + u(x_1 - \delta, y_1) + u(x_1, x_2 + \delta) + u(x_1, x_2 - \delta) - 4u\,(x_1, x_2)]$$

where δ is the sampling of the grid that represents Ω. In this form, the equation can be solved using relaxation methods such as Jacobi or Gauss-Seidel whose principle is to iteratively replace each grid point value with the simple average of its four neighbors. Figure 1 shows different trajectories generated by simulations using this numerical scheme.

2.2.2 Properties of harmonic navigation functions

Harmonic navigation functions have many interesting properties which motivated their use in numerous applications especially in robotics [13, 4, 7, 5, 16, 14, 10, 12, 1, 8]:

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

Dynamic trajectory planning: Unexpected updates of the environment may be taken into account, since harmonic functions are computed by iterative relaxation methods. Therefore newly detected obstacles may be integrated in the model as new boundary conditions during computation, so that harmonic control is available in dynamic environments or in environments explored on-line [16, 2].

Parallel computation: An interesting property of the computations described above is their massively parallel distribution. Computing grid point values only requires local information of the neighboring cells. Fine-grain parallel implementations appear as an opportunity, as discussed in the next section.

3 Towards an embedded implementation

The results shown in figure 1 were obtained from software simulations carried out on a PC. The aim of our work is to design an embedded system for robot navigation. Computation speed is not the only criterion (trajectory decision must be performed in real time). Power consumption is also essential for autonomous systems. Moreover, computation precision and scalability appear as critical issues for harmonic control, as discussed below. These combined aspects motivate the design of a parallel hardware implementation. In such a work, the number of inputs/outputs and above all the level of parallelism have a direct influence on the obtained implementation consumption and speed. A massively parallel implementation is a real challenge, taking into account constraints such as precision, grid size, dynamic updates, etc.

3.1 Implementation environment

Since the appearance of programmable circuits, such as *field programmable gate arrays* (FPGAs), algorithms may be implemented on fast integrated circuits with software-like design principles. Usual VLSI designs lead to very high performances. But the time needed to realize an ASIC (application specific integrated circuit) is too long, especially when different configurations must be tested. The chip production time is usually very long (up to 6 months).

FPGAs, such as Xilinx FGPA ([15]), are based on a matrix of *configurable logic blocks* (CLBs). Each CLB is able to implement small logical functions (4 or 5 inputs) with a few elementary memory devices (flip-flops or latches) and some multiplexors. Each CLB is independently programmable. Similarly, the routing structure that connect the CLBs can be configured. A FPGA approach simply adapts to the handled application, whereas a usual VLSI implementation requires costly rebuildings of the whole circuit when changing some characteristics. A design on FPGAs requires the description of several operating blocks. Then the control and the communication schemes are added to the description, and an automatic "compiling" tool maps the described circuit onto the chip.

3.2 Technological choices

The main issues when a massively distributed model is mapped onto a FPGA are the huge number of operators, and the routing problems due to the dense interconnections. A first standard technological choice to solve these problems is to use serial arithmetics: smaller operators may be implemented, and they require less connection wires. Another essential technological choice is to estimate the minimum precision required to keep satisfactory results with as small as possible operators and memory resources.

3.2.1 Serial arithmetics

Serial arithmetics correspond to computation architectures where digits are provided digit after digit. Serial arithmetics lead to operators that need small implementation areas and less inputs/outputs, and that easily handle different precisions, without an excessive increase of the implementation area. Serial systems are characterised by their delay, i.e, the number δ such that p digits of the result are deduced from $p + \delta$ digits of the input values.

Two main kinds of serial arithmetics are available: LSBF or MSBF (least/most significant bit first). Standard serial operators work in a LSBF mode. Though our model requires the computation of a minimum value (gradient descent), that may only be computed in a MSBF mode, we use standard serial operators to optimize the required area[1].

3.2.2 Computation precision

Software simulation are usually performed to study the precision that is required by an application before its hardware implementation. Precision issues appear as a critical problem for harmonic control, that has already been mentioned as a major limitation for analog implementations [13]. Computing a harmonic potential over a large grid may result in gradients that are too small to use because the allowable precision is easily reached. Connolly [4] presents a relationship between the precision required for floating point representation on a computer and the number of nodes on the grid. He argues that the precision should at least represent $\frac{1}{N}$, where N is the total number of grid points, to circumvent precision problems. This further motivates the use of an embeded implementation in which we can allow very high precisions (see 4.3).

We have carried out experimental simulations with different navigation spaces. They have shown that for some 50×50 grids, a 22-bit precision is required at least to ensure significant differences between neighboring grid point values so as to generate correct trajectories. Other simulations have shown that very large grids may require more than 64-bit precision numbers. It should be pointed out that in case of insufficient precision, large areas of the grid may share the same value, which results in wrong trajectories.

Implementations based on serial arithmetics may be more easily extended to larger precisions than implementations based on parallel arithmetics. Since the size of serial adders and comparators does not depend on precision (unlike multipliers and elementary functions), our implementation may handle large precisions by means of rather simple changes in the control modules.

4 Hardware implementation of harmonic control

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 2.2.1. For example in [11] an analog implementation of a 16×16 grid is proposed. The main limitation of this work is the precision (as for most analog implementations). To our knowledge, digital FPGA-based implementations have not yet been proposed.

In this work, the discretized computation of the harmonic function is performed to make navigation decisions for a robot in an environment that may be explored on-line. Navigation orders are also discretized: at each position, the robot is ordered to move north, east, west or south. This simplification implies that the optimal trajectory along the gradient is only roughly estimated by elementary movements along the axis'. It still results in global trajectories towards the goal position from any starting point.

4.1 General architecture

The fine grain pipelined internal structure of the proposed digital architecture was implemented using fixed point arithmetic. Since grid point values range from 0 to 1, an unsigned representation with 25 bits is chosen, with a 24-bit fractional part. As mentioned above, larger wordlengths may easily be handled.

Figure 2 illustrates the general architecture of the implementation of harmonic control for a $n \times n$ grid. It con-

[1] The only existing radix-2 MSBF serial arithmetics is called *on-line* arithmetics. It uses a redundant number representations system, which induces less area-saving operators.

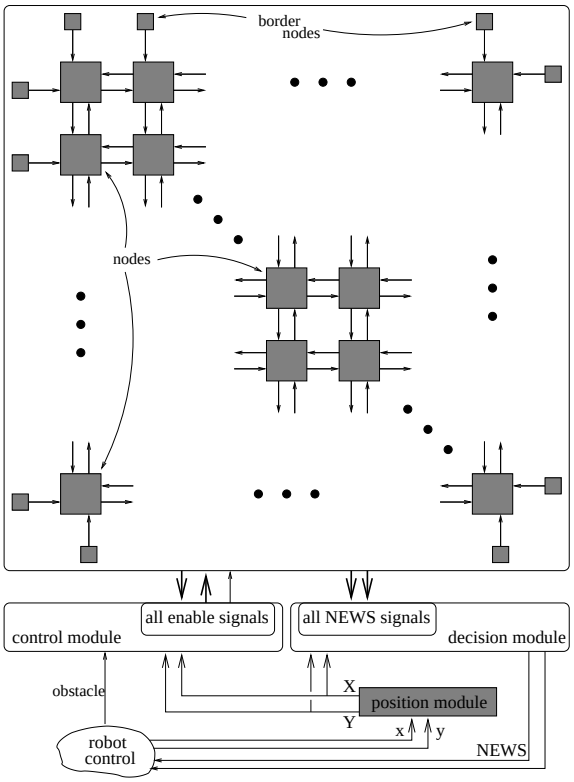

Figure 2. General architecture

sists of a grid of $n \times n$ identical node modules (gathered 16 by 16 to handle on-chip data storage and access) surrounded by border node modules, a control module, a decision module, and a module to interact with the robot.

The role of each component is the following:

○ Each node computes its corresponding grid point value, as well as the direction (north, east, west or south) to follow to get to the goal position. All nodes use loop computations within an internal pipeline scheme. This scheme is particularly efficient for serial implementations of iterated computations within massively distributed models ([6]). The control of these computations are synchronized in the whole grid so that nodes may serially communicate their grid point values to their neighbors. In order to simplify the block-diagram of figure 2, only few buses and wires are shown: the signals that carry the grid point values from and to any neighboring node.

○ The nodes are split in groups of 4×4 nodes that share common storage resources: a single dual port SRAM block stores the values of the 16 grid points, and its R/W accesses are controlled by a single set of counters (see 4.2).

○ The border nodes are very simple. They serially generate the grid point value of obstacles.

○ The interaction with the robot has not yet been implemented. It strongly depends on the exact configuration of the robot and of the FPGA board. It includes a position modules, which role is mainly to compute the coordinates `(X,Y)` of the closest grid point around the real coordinates `(x,y)` of the robot in its environment.

○ The control module generates the enable signals that are sent to all nodes to control their individual behaviour when a non-synchronized event occurs:

- convergence of the computation of the harmonic function (it depends on the local convergence signals computed by all nodes, see 4.2)
- detection of an unknown obstacle (at node `(X,Y)`)

○ The decision modules collects the navigation directions locally indicated by each node, and it forwards the selected direction of node `(X,Y)` to the robot.

In the following subsections, the hardware architecture for the node model and its main components will be described in some detail.

4.2 Node implementation

The architecture for the node model is shown in the simplified block diagram on figure 3. It uses 1-bit inputs and outputs to exchange data among nodes and with the global modules. Inputs are mainly used to receive the neighboring grid point values (signals `h_N,h_E,h_W,h_S`) and global control signals (standard signals `clk,reset,enable`, signals `sel,sat` to indicate obstacle/goal changes, and SRAM controls `EN,R,W`). Outputs are twofold: the local value `h` of the harmonic function is sent to all 4 neighbors (signals `to_N,to_E,to_W,to_S`) and signals `NE` and `WS` code the 4 possible directions to which a robot located around this grid point should move.

The proposed hardware node model is constituted by five main modules: the iterative computation of the harmonic function is performed by the `Update` module, the local convergence of this computation is detected by the `Cvg` module, the local navigation decision is performed by the `Direction` module, 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 grid point value is controled by the `Mem` module. Figure 3 shows this architecture, as well as its interaction with shared resources (the local `Counters` and `RAM` modules are shared by a group of 4×4 nodes, and the `Enable` module is part of the global control module). The functionality of the main modules and their implementation details are described below.

Update: This module performs the iterated computation of the harmonic function value $h_{i,j}$ where (i, j) are the coordinates of the node in the grid. As described in 2.2.1, each iteration computes:

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

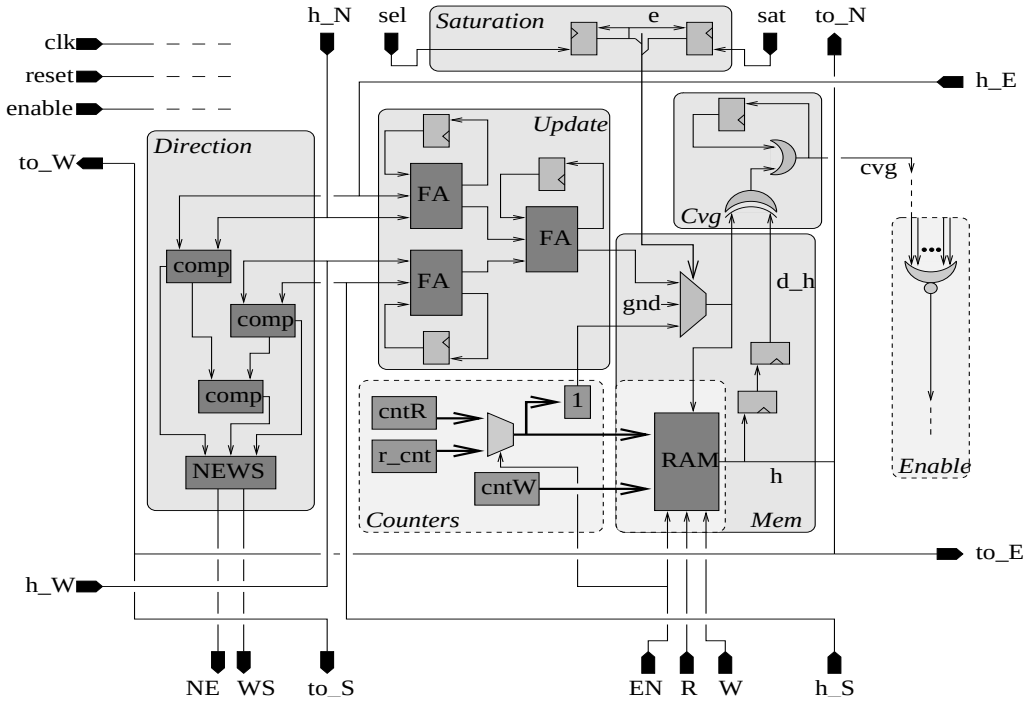

Figure 3. Architecture of a node

Three standard Full-Adder cells compute this average, without any shift or division operator, since the output value is sent to the RAM with a write address that is delayed by 2 clock cycles (division by 4). Additional flip-flops are required to store the carry values.

Cvg: This module serially compares the output of the iterated computation to the stored value (delayed by two flip-flops in the `Mem` module). This local convergence test is then sent to a global NOR gate (see the `Enable` module) to disable the computation loop after convergence.

Direction: This module operates after convergence of the iterations. It computes the minimum grid point value among the 4 neighbors. Its output is the 2-bit code of the direction to choose. This code is computed by the `NEWS` module that receives the results of the three comparators. This computation is performed in a MSBF mode, thanks to the reverse address counter `r_cnt`.

Counters: This module is shared by 16 nodes. It generates the read (resp. write) addresses for the dual port RAM by means of counters `cntR` (resp. `cntW`). The read address is multiplexed with the reverse counter `r_cnt` to change from LSBF to MSBF mode. When handling d-bit precision data, these counters are reset each $d + 2$ cycles (the RAM is written with a 2-clock cycles delay). Value 1 (for obstacles) is computed as a logical function of `cntR`.

Mem and Saturation: The local grid point value is not directly the output of the `Update` module. It may also be a constant 0 or 1 (goal or obstacle). A multiplexer selects the correct value 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`).

4.3 Implementation results

This work uses a PCI bus board equiped with a Virtex XC2V6000-4FF1517 FPGA from Xilinx, with up to 6,000,000 system gates. Such a FPGA contains 67,584 logic cells, to be compared with the 200,448 ones of the current largest Virtex-4. The design was synthesized, placed and routed automatically in Xilinx Foundation ISE 7.1i. Each node requires 20 logic cells, so that each block of 16 nodes requires 360 logic cells (counters included). On a XC2V6000, 144 dual port SRAM blocks are available, that may be configured as $1K \times 18$ RAMs to be shared by blocks of 16 nodes. The whole architecture implements a 48×48 grid on less than 77 % of the XC2V6000 logic cells. Larger grids (up to 72×72) may be implemented on the current largest FPGAs with this approach (the available SRAM blocks being the critical resource).

Software implementations of the harmonic function computation on a microprocessor based computer, Pentium 4, 2 GHz, require around 100 μs per iteration with a 50×50 grid (100 to 1000 iterations are required to converge, depending on the environment). In the proposed hardware implementation, 27 clock cycles are required per iteration, with an estimated clock frequency of 140 MHz. Thus, the

architecture provides a speed factor up to 500×, that would even increase with the number of nodes in the grid (sequential vs parallel implementation). But the implementation speed is not the main advantage of our implementation: real-time computation is easily reached by software implementations for such precisions and size grids. Power consumption is a key factor for embedded implementations, and above all very large precisions may be easily handled by the proposed serial implementation (up to 1K bits when fully using the SRAM blocks).

5 Conclusion and future work

We have presented an embedded architecture to solve the navigation problem in robotics, that computes trajectories along a harmonic potential, using a FPGA implementation. This architecture includes the iterated estimation of the harmonic functions. The goals and obstacles of the navigation problem may be changed during computation. The trajectory decision is also performed on-chip, by means of local computations of the preferred direction at each point of the discretized environment. 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 pipelined iterative updates of the local harmonic function values until global convergence.

The proposed architecture enables us to handle very large precisions using SRAM blocks, which is a benefit over computer resolution of harmonic funtions. Furthemore, a greatly improved speed and a low power consumption are other non-negligible advantages, since the goal is to embed this implementation on mobile robots. The current hardware model is already able to handle significantly large discretized environments.

Future architecture improvements are already considered to extend the capacities of the model to navigation within larger and more complex environments. A studied extention efficiently uses the available SRAM to handle very large grid, by means of an iterated computation mode that is both globally asynchronous and block-synchronous. Current efforts are also made to extend our implementation to optimal control, a more generic (and tunable) trajectory planning method.

References

[1] D. Alvarez, JC Alvarez, and RC Gonzalez. Online motion planning using laplace potential fields. In *Proceedings of the IEEE Int. Conf. Robotics and Automation, 2003.*.

[2] A. Boumaza and J. Louchet. Mobile robot sensor fusion using flies. In *Applications of Evolutionary Computing*, volume 2611 of *LNCS*, pages 357–367, 2003.

[3] C. I. Connolly, J. B. Burns, and R. Weiss. Path planning using laplace's equation. In *Proceedings of the 1990 IEEE Int. Conf. on Robotics and Automation*, pages 2102–2106. 1990.

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

[5] HJS Feder and J.J.E. Slotine. Real-time path planning using harmonic potentials in dynamic environments. In *Proceedings of the IEEE Int. Conf. Robotics and Automation, 1997*.

[6] Bernard Girau and Cesar Torres-Huitzil. FPGA implementation of an integrate-and-fire legion model for image segmentation. In *European Symp. on Artificial Neural Networks*, 2006.

[7] M. Huber, WS MacDonald, and RA Grupen. A control basis for multilegged walking. In *Proceedings of the IEEE Int. Conf. Robotics and Automation, 1996*.

[8] M. Kazemi, M. Mehrandezh, and K. Gupta. An incremental harmonic function-based probabilistic roadmap approach to robot path planning. In *Proceedings of the IEEE Int. Conf. Robotics and Automation, 2005*.

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

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

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

[12] JD Sweeney, H. Li, RA Grupen, and K. Ramamritham. Scalability and schedulability in large, coordinated, distributed robot systems. In *Proceedings of the IEEE Int. Conf. Robotics and Automation, 2003*.

[13] L. Tarassenko and A. Blake. Analogue computation of collision-free paths. In *International Conference on Robotics and Automation*, pages 540–545. IEEE, 1991.

[14] Y. Wang and GS Chirikjian. A new potential field method for robot path planning. In *Proceedings of the IEEE Int. Conf. Robotics and Automation, 2000*, volume 2, pages 977–982, 2000.

[15] Xilinx, editor. *The Programmable Logic Data Book*. Xilinx, 2002.

[16] J.S. Zelek. Complete real-time path planning during sensor-based discovery. In *IEEE/RSJ Int. Conf. on Intelligent Robots and systems*, 1998.