

# Robot Path Planning Using Field Programmable Analog Arrays

Scott Koziol, Paul Hasler and Mike Stilman

**Abstract**—We present the successful application of reconfigurable Analog-Very-Large-Scale-Integrated (AVLSI) circuits to motion planning for the AmigoBot robot. Previous research has shown that custom application-specific-integrated-circuits (ASICs) can be used for robot path planning. However, ASICs are typically fixed circuit designs that require long fabrication times on the order of months. In contrast, our reconfigurable analog circuits called Field Programmable Analog Arrays (FPAAs) implement a variety of AVLSI circuits in minutes. We present experimental results of online robot path planning using FPAAs, validating our assertion that FPAAs-based AVLSI design is a feasible approach to computing complete motion plans using analog floating-gate resistive grids. We demonstrate the integration of FPAAs hardware and software with a real robot platform and hardware in the loop simulations, present the trajectories developed by our planner and provide analysis of the time and space complexity of our proposed approach. The paper concludes by formulating metrics that identify domains where analog solutions to planning may be faster and more efficient than traditional, digital robot planning techniques.

## I. FPAAs EMBEDDED SYSTEM FOR PATH PLANNING

Path planning is a critical task for robots. Given states, actions, an initial state, and a goal state, path planning can be summarized with the following three tasks: First, find a sequence of actions that take the robot from its Initial state to its Goal state. Second, find actions that take the robot from *any* state to the Goal state, and third, decide the *best* action for the robot to take now in order to improve its odds of reaching the Goal. Path planning computations are typically executed on *Digital* microprocessors. This work will show that using *Analog* VLSI circuits instead of *Digital* circuits for path planning can potentially provide: Improved Time and Space Complexity metrics, improved computation times, and finally the potential for lower power processing capabilities [1], [2]. Low power processing capabilities provide a significant advantage for small ground robot applications where the power budget for Guidance, Navigation, and Control is limited and the battery is a significant portion of the robot’s mass, Fig 1. Micro Aerial Vehicles (MAV) or Autonomous Underwater Vehicles (AUV) may be potential future applications of this planner.

Custom ASIC designs could be used for path planning, but revisions incur a long design cycle/fabrication time. FPAAs, however, allow the designer to reconfigure the analog

S. Koziol and P. Hasler are with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, USA [skoziol,phasler@ece.gatech.edu](mailto:skoziol,phasler@ece.gatech.edu)

M. Stilman is with the School of Interactive Computing, Georgia Institute of Technology, Atlanta, GA, USA [mstilman@cc.gatech.edu](mailto:mstilman@cc.gatech.edu)

The authors would like to thank Stephen Williams of Georgia Institute of Technology for robot hardware support.

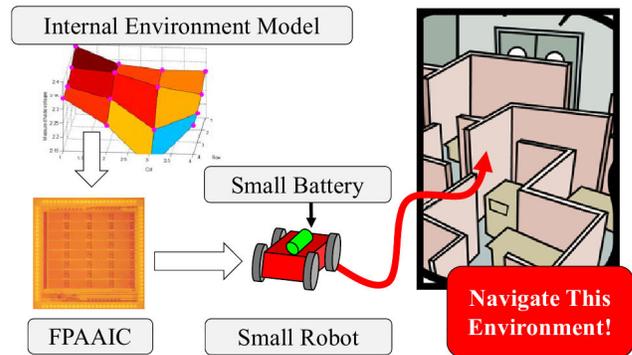


Fig. 1. This shows a goal of this research: To use reconfigurable analog circuits called Field Programmable Analog Array Integrated Circuits (FPAAs IC) to plan a path for small robot through an environment in an effort to conserve limited battery resources and extend operation times.

connections within the IC using software and hardware infrastructure. This allows quick design changes and re-use of a single IC [4]. Fig 2 shows the FPAAs and programming and control hardware infrastructure developed at Georgia Tech [3]. This embedded system programs the desired circuit onto an FPAAs IC.

This work builds upon [5]. Two significant contributions of this new paper are first, the FPAAs planner is integrated with a robot, and second the FPAAs planner is compared to a digital method in regards to space and time complexity. In summary, the answer to the motivation question *Why plan using analog?*: Because with a large enough grid size analog may be faster and more efficient than traditional, digital robot planning techniques. Also, it may provide a potential power savings. The answer to a second motivating question *Why FPAAs?*: Because reconfigurable AVLSI systems provide circuit tunability and flexibility that custom ASICs do not provide.

## II. RELATED WORK

In 1985, Khatib [6] introduced the idea of real time robot path planning using potential fields. One of the drawbacks to this method is that it is *not complete* because local minima in the search space may lead to solutions which do not end in the goal. One of the earliest references to using Laplace’s equation for path planning is [7]. This method eliminates the local minima problem of potential fields. Harmonic functions are solutions to Laplace’s equations. Harmonic potential fields are explored to eliminate local minima of potential fields, [8], [9], [10]. Tarassenko, et. al. build upon [7] and present the idea of using AVLSI resistive grids for

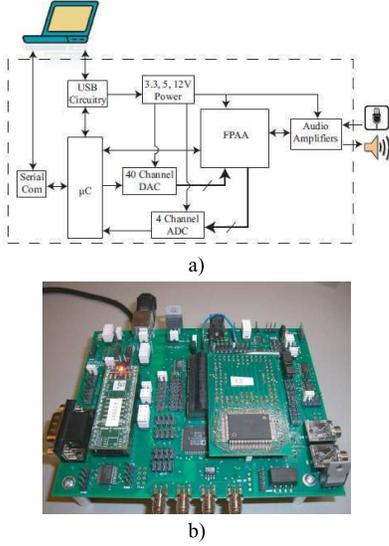


Fig. 2. (a) Block Diagram of the FPAA programming and control board (b) Custom embedded system implementing the block diagram and used for the experiments in this paper. [3]

the computation in [11]. Other research involving AVLSI or resistive grids for planning is found in [12], [13], [14], [13], [15], [16], and [17]

As evidenced by the above references, the idea of using Laplace’s equation and analog circuits for path planning is not new, but there is limited circuit measurement data in the references, and few if any examples of integrating an AVLSI planner with robot systems. Finally, the referenced circuits use resistors or standard transistors to implement the circuits. This work is new for a few reasons: First, because it provides measured data from a fabricated AVLSI IC that is actually integrated into a closed loop robot system. Second, our analog circuit implementation is different from the existing literature because it uses floating-gate transistors which provide, among other things, a non-volatile way to store the environment map, and third, initial calculations comparing time and space complexities for Analog vs Digital are presented.

### III. MATHEMATICAL ANALYSIS OF ANALOG PLANNER

The Laplacian is a differential operator that provides the partial derivatives of the gradient. In this path planning problem, it is the second partial derivatives of the voltage function [18]. *Laplace’s equation*, (1), uses the Laplace operator on a function and sets it to zero [18]. According to this method, the voltage at each node is the average of its four neighbors.

$$\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = 0 \quad (1)$$

Fig 3 shows how an office environment is modeled using a transistor based resistive grid. The transistors in the free path regions were turned fully on by programming the floating gates to conduct current, and the transistors representing

obstacles (in red) were turned off by programming the floating-gate transistors such that they would not conduct current. This resistive grid is then used to solve a path planning problem. As in [11], the path from start to goal is found by 1) placing a constant current source at the start node 2) waiting until the resistive grid settles into a steady state, 3) reading the node voltages 4) finding the path from start to goal using voltage measurements from successive nodes. The transistors used for planning in the FPAA are a special type of transistor called *floating-gate* transistors. A transistor can be operated in above threshold or subthreshold regimes. Subthreshold operation results in much lower power use because less current is being conducted. The current in a pFET transistor in subthreshold operation may be described in (2) [19].

$$I = I_0 e^{\frac{V_{dd}(\kappa-1)}{U_T}} e^{\frac{-\kappa V_G + V_S}{U_T}} \left[ 1 - e^{\frac{-V_{sd}}{U_T}} \right] \quad (2)$$

Where in (2),  $I_0$  is a constant representing pre-exponential factors,  $\kappa$  is a constant representing the capacitive coupling ratio from gate to channel, and  $U_T$  is the thermal voltage [20] [19]. From (2) the pFET’s resistance is calculated in (3).

$$R = 1 \left/ \frac{\partial I}{\partial V_d} \right|_{V_{ds}=0} = \frac{U_T}{I_0 e^{\frac{V_{dd}(\kappa-1)}{U_T}} e^{\frac{-\kappa V_G + V_S}{U_T}}} \quad (3)$$

Floating gate transistors are unique. Unlike a conventional transistor, floating gate transistors have an isolated gate terminal which can hold charge. This provides two important properties: First, one does not need to actively maintain a voltage on each gate terminal of the grid when using the circuit. Second, once programmed, the gate is *set* and the circuit will hold its configuration even if the power is removed from the IC. More details about the FPAA planner can be found in [5].

### IV. INTEGRATION OF FPAA AND ROBOT

We are using an AmigoBot robot to demonstrate the analog planning system, Fig 4(a). The FPAA and AmigoBot robot are integrated such that the FPAA acts as a “planning co-processor” for the robot. A block diagram showing how the analog resistive grid’s planner fits into the larger robot system is found in Fig 5(a). The Executor’s function is to act as an interface between the FPAA and the low level digital controller. An example software flow is found in Fig 5(b). This simple, proof of concept flow assumes four things: a known map, a static environment, the robot’s starting location is known, and the goal location is known. More complex flows could move the current source to follow the robot in the grid [11] and can also incorporate re-planning. The task of the Navigation block in Fig 5(a) is to convert high level plans such as “Move from the window to the desk at grid (4,3)” in Fig 3(a) to low level commands. A position-stabilizing controller adjusts the robot’s forward and angular velocity and is used to drive the robot to points in the grid [21]. The

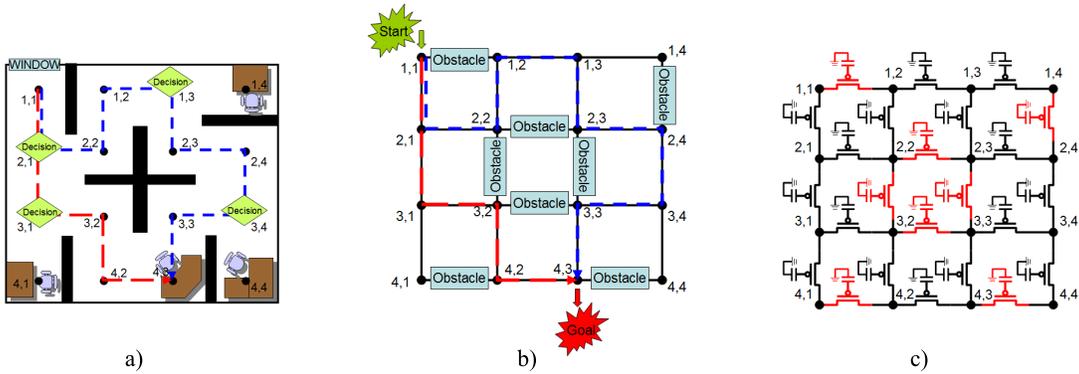


Fig. 3. Converting the office grid world into an AVLSI representation a) Office with walls as obstacles b) Simplified grid representation of office c) Floating-gate pFET transistors used to implement the obstacles.

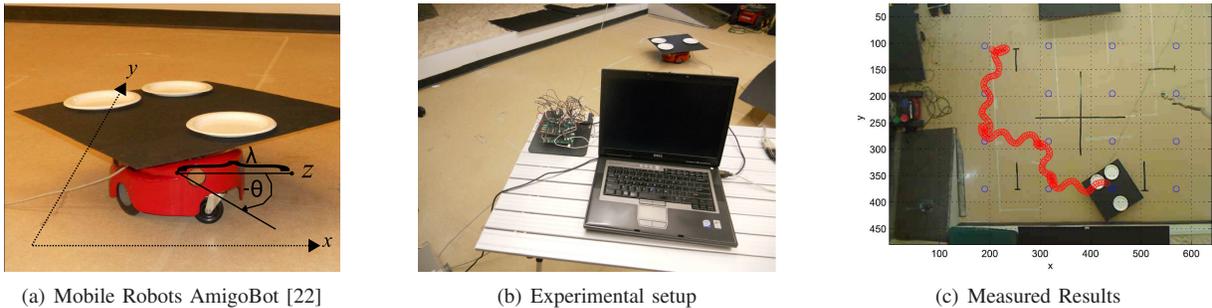


Fig. 4. Our experimental environment showing (a) the robot with coordinate axes, (b-c) the implementation of an FPAA generated plan.

control equations are based on feedback linearization. The kinematic equations of motion are shown in (4).

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\lambda} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -c(\lambda - \varepsilon) \end{bmatrix} + \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \quad (4)$$

This uses a coordinate transformation where a  $\lambda$  offset is chosen from the center of rotation (see axis overlaid over Fig 4(a)). An overhead camera is used for localization. Image processing routines segment three dots on the back of the robot and these are used to locate the robot in  $(x, y, \theta)$  image space [23].

For this proof of concept system, two programming environments were integrated: the Matlab of the FPAA, and the C++ code for *Player*. An extensive body of Matlab code has been developed by the Integrated Computational Electronics (ICE) Lab at Georgia Tech to program and communicate with the FPAA board. Open source *Player* software, [24], is used for interfacing with the FPAA (via Matlab) and for controlling the AmigoBot robot. The FPAA Matlab code is called by *Player* using Matlab engine functions [25].

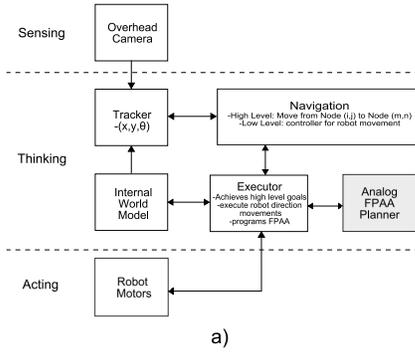
## V. EXPERIMENTAL RESULTS

This section presents our initial results of integrating a robot with an AVLSI co-processor. The experimental setup is shown in Fig 4(b). The robot is shown in the background, the FPAA is shown on the left corner of the desk, and the

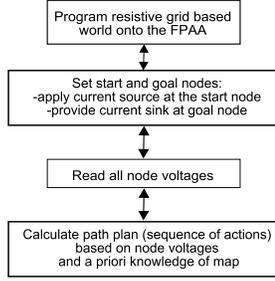
overhead camera is above (not shown). All are tethered to the laptop running *Player* via USB cables. Fig 4(c) shows our initial experimental results from a robot in a four by four grid world. The AVLSI FPAA hardware has been integrated into *Player* as a co-planner for the AmigoBot robot. This is an image taken from the overhead camera used for localization. The results are for an experiment in a 4x4 office grid world. The cubicle partitions are marked in black tape on the floor. The overlaid red dots are the recorded trajectory of the robot moving from node (1,1) to node (4,3). The overlaid blue circles mark the grid nodes. At each node traversed by the robot, the FPAA was consulted for the next node.

Although this is a trivial planning problem, it demonstrates two major goals: First, our system can make complete plans using floating-gate resistive grids (based on our limited experiments), Second, the supporting FPAA hardware and software are at a level of sophistication where they can be reliably integrated into robot platforms. This system has three modes of operation. These three methods provide useful options when debugging various parts of the system. A brief discussion of each of the modes is now presented.

**Real Robot, Real FPAA Results:** The FPAA and an AmigoBot robot were integrated together and localization was performed using an overhead camera (640x480 pixel resolution). The robot successfully navigated its path on the floor as directed by the FPAA. Fig 4(c) shows in red the path the robot made from its start to its goal. At each node (represented by a blue circle), *Player* queried the FPAA co-



a)



b)

Fig. 5. a) High level control system block diagram b) Software flow of the Executor designed to integrate the analog planner and the robot. .

processor to help decide whether to go straight, or turn left or right at each node in order to reach the goal.

**Real FPAA, Simulated Robot Results:** This is a Hardware in the Loop (HWIL) environment where the actual FPAA hardware is being called by *Player* and is interacting with a virtual robot in a three dimensional robot simulator with dynamics. This environment is called *Gazebo* and interacts with *Player* using the same control code. Ideally, one can take the same *Player* control code to control a virtual robot or the real thing. Fig 7 shows an image from a *Gazebo* simulation. Virtually identical software is used for this HWIL simulation as in the section regarding *Real Robot, Real FPAA Results*.

**Simulated FPAA, Simulated Results:** Finally, it is possible to simulate the FPAA results by using Matlab to solve for node voltage values using, for example, Kirchoff's laws.

## VI. ANALYSIS

This path planning problem can be formulated as a tree search problem. These problems are typically evaluated with four metrics: Completeness, Optimality, Time Complexity, and Space Complexity [26]. Time and Space Complexity are addressed further in the following sections. Time complexity is typically measured by the number of nodes generated [26]. Space complexity is measured in terms of the maximum number of nodes stored in memory [26].

### A. Time Complexity

Time complexity is not as simple as number of nodes generated with the FPAA. There are three items to consider when calculating the total time cost of the FPAA planner: FPAA grid programming time, solution computation time,

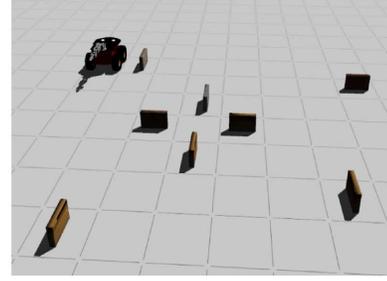


Fig. 7. This is an image of a simulated office environment used in a FPAA Hardware in the Loop (HWIL) test.

|  | Type 1 | Type 2 | Type 3 |
|--|--------|--------|--------|
| Measured Erase and Initialize grid (sec) | 35     | 35     | 35     |
| Measured Program time per path (sec)     | 0      | 0.0486 | 4.4332 |
| Expected Program times (sec) [4]         | 0      | 0.001  | .050   |

TABLE I

GRID PROGRAMMING TIMES ACCORDING TO PATH TYPE

and time to read the solution from the grid. Each of these are addressed below.

1) *FPAA Programming Time Measurements:* Programming a grid map onto the FPAA is done in two main phases: First, the FPAA is erased and prepared for programming, second, the new map is programmed onto the FPAA. With our current software, it takes approximately 35 seconds to erase and prepare the FPAA for programming. This amount of time is independent of the size of map that will be programmed. The time needed to program the map is a function of two parameters: size of map, and type of paths which connect the nodes on the map. There are three types of paths that we will consider: Type 1: impassable paths, Type 2: completely passable paths, and Type 3: a path which is passable, but with some degree of difficulty. This may be due to terrain such as sand, an incline, etc. The programming times for each of these paths is summarized in Table I. As the state of the art in floating-gate programming advances, these times are expected to decrease.

In the 4x4 grid example, 38 floating-gate switches were used in the circuit. Of these, 22 switches were *overhead*. That is, they were needed to program the grid, but were not *path* elements. This overhead number changes according to grid size. The switches were generated automatically using *GRASPER* software [4]. In this example, this overhead represents about 58% of the total number of switches. Due to obstacles, the number of free paths was only about 67% of the total paths possible in a 4x4 grid. If we consider  $N$  as the number of nodes on the side of an  $N \times N$  square grid, the number of possible paths is  $O(2N^2)$ .

2) *Solution computation time:* The computation time for the FPAA is based on the time it takes for all of the grid's node voltages to settle to steady state in response to a current step input. For the 4x4 grid example, the computation time of this grid based FPAA planner is approximately 4.5ms. Fig 9 shows the transient response of each of the sixteen nodes in

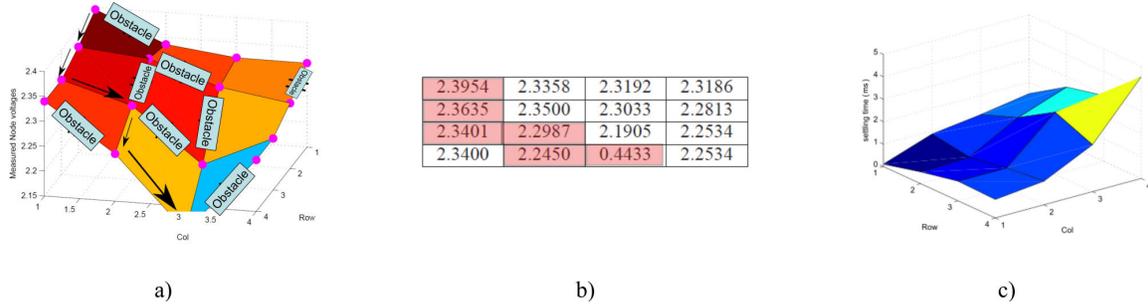


Fig. 6. a) Measured FPAA hardware results for a 4x4 grid like the configuration of the robot start, goal, and obstacles in Fig 3(c). b) A table of the measured voltages with path identified by the pink squares. c) Measured node voltage settling times of the example office 4x4 resistor grid as a function of grid location.

the grid. The limiting factor in this case is Node 15 which took about 4.5ms to settle. Fig 6(a-b) shows node voltage measurements from a 4x4 grid world, (c) shows the node settling times for each of the nodes. A step input voltage was placed on the pFETs gate at node (1, 1) and this implemented a step input current to represent the robot's location at this node. A current sink was implemented at node (4, 3) to represent the goal. The last node to reach steady state took 4.5ms.

3) *Solution access time:* In the FPAA, once the nodes have settled, the solution is found by reading  $d$  nodes, where  $d$  is the depth of the shallowest goal node. We could say then, that the FPAA has Time complexity of  $O(d)$ . For comparison, Breadth-First Search (BFS) has Time complexity of  $O(b^{d+1})$ , where the branching factor  $b = 4$ , and  $d$  is the depth of the shallowest solution. Fig 8(a) compares Analog to Digital Time Complexity as a function of shallowest solution.

### B. Space Complexity

To calculate a final path solution the FPAA planning system needs to maintain an adjacency list. This lists, for each node, all nodes that are one step away through Type 2 or Type 3 paths. This list can have the form  $[source\ node, list\ of\ adjacent\ nodes]$ . For example, in the 4x4 grid of Fig 3, the robot can reach nodes (2,2), and (3,1) from node (2,1). The corresponding adjacency list would be  $[(2,1) (2,2) (2,1)]$ . This information is contained in MATLAB and combined with the node voltages read from the FPAA to choose a path. Assuming no obstacles for maximum space complexity, the worst case space complexity of the FPAA is  $O(4N(N-1) + 1)$ , where  $N$  is the number of nodes on a side of a square map, i.e.  $N \times N$  map. This is calculated using (5) where  $N_x$  terms are numbers i.e.  $N_{middle-nodes}$  is the number of middle nodes, and  $A$  are numbers of adjacencies. BFS worst case Space Complexity is  $O(b^{d+1})$ , where  $b = 4$  and  $d$  is the depth of the DEEPEST solution.

$$\begin{aligned}
 Space\_complexity_{FPAA} = & (N_{corners} * A_{corner}) \\
 & + \left( N_{non-corner-nodes-on-grid-edge-side} \right. \\
 & \quad \left. * A_{non-corner-nodes-on-grid-edge-side} * 4 \right) \\
 & + (N_{middle-nodes} * A_{middle-nodes}) + 1
 \end{aligned} \tag{5}$$

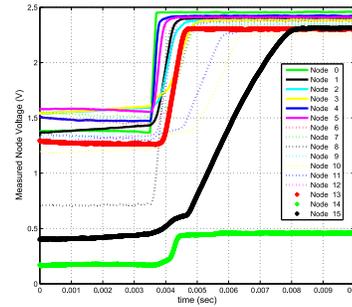


Fig. 9. Measured transient responses for node voltages.

| Criterion | FPAA                              | Breadth First |
|-----------|-----------------------------------|---------------|
| Complete? | Yes (based on limited experiment) | Yes           |
| Time      | $O(d)$                            | $O(b^{d+1})$  |
| Space     | $O(4N(N-1) + 1)$                  | $O(b^{d+1})$  |

TABLE II  
COMPARING FPAA TO BFS

Table II summarizes the Time and Space complexity comparisons between the FPAA and Breadth-first-search (BFS) [26].

### C. Calculation time estimate

Ideally, one would like to compare the actual solution times of the digital and analog solutions and not just operation numbers like Time complexity. As an estimate, assume that the BFS algorithm is being executed on an a processor such as the ATMEL ARM7TDMI RISC processor operating at 55MHz max clock speed. Further assume that the solution is at the deepest solution of the grid. If one multiplies the BFS Time Complexity number by the inverse of the ARM7 clock then we can have a crude estimate of the digital computation time. To estimate the computation time of the FPAA, we extract a curve from the diagonal delay times of Fig 6(c). Since BFS is in terms of  $b$  and  $d$ , and the FPAA settling time estimate is in terms of  $N$ , we use  $N = (d/2) + 1$  as the transformation. A comparison plot is shown in Fig 8(c). Estimate of BFS Computation Time for 55 MHz processor :  $O(b^{d+1}) * (\frac{1}{55MHz})$  where  $b = 4$ ; Estimated

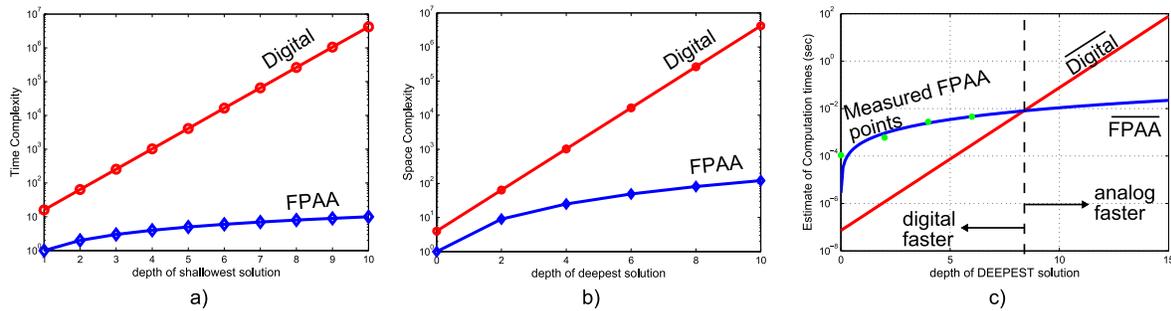


Fig. 8. (a) Comparing the Time complexity of the FPAA to BFS (b) Comparing worst case Space complexities of the FPAA to BFS (c) Comparing Computation Time of the FPAA to an estimate for BFS

FPAA Computation Time is based on extrapolation of the diagonals of the 4x4 delay measurement data in Fig 6(c). Based on this graph, the prediction is that an FPAA solution may be faster than digital for solution depths greater than 8. This plot estimates that the FPAA will be quicker at solving plans where the solution depth is greater than 8. This corresponds to the deepest solution of a 5x5 grid.

## VII. CONCLUSION

Fig 8(a) and (b) have shown that the Time Complexity and Space Complexity of the FPAA is orders of magnitude lower than that of BFS. Fig 8(c) also describes the solution depth at which FPAAs may find a solution quicker than BFS. Finally, the FPAA embedded planning system was successfully integrated with a real robot.

## REFERENCES

- [1] R. Sarpeshkar, "Analog versus digital: extrapolating from electronics to neurobiology," *Neural Computation*, vol. 10, no. 7, pp. 1601–1638, 1998.
- [2] C. M. Twigg, "Floating gate based large-scale field-programmable analog arrays for analog signal processing," Ph.D. dissertation, Georgia Institute of Technology, Atlanta, 2006. [Online]. Available: <http://hdl.handle.net/1853/11601>
- [3] S. Koziol, C. Schlottmann, A. Basu, S. Brink, C. Petre, B. Degnan, S. Ramakrishnan, P. Hasler, and A. Balavoine, "Hardware and software infrastructure for a family of floating-gate based fpaas," IEEE International Symposium on Circuits and Systems (ISCAS) 2010, May 2010.
- [4] A. Basu, S. Brink, C. Schlottmann, S. Ramakrishnan, C. Petre, S. Koziol, F. Baskaya, C. Twigg, and P. Hasler, "A floating-gate-based field-programmable analog array," *Solid-State Circuits, IEEE Journal of*, vol. 45, no. 9, pp. 1781–1794, 2010.
- [5] S. Koziol and P. Hasler, "Reconfigurable analog VLSI circuits for robot path planning," in *Adaptive Hardware and Systems (AHS), 2011 NASA/ESA Conference on*, June 2011, pp. 36–43.
- [6] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, vol. 2, Mar. 1985, pp. 500–505.
- [7] C. Connolly, J. Burns, and R. Weiss, "Path planning using laplace's equation," in *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*, May 1990, pp. 2102–2106 vol.3.
- [8] J.-O. Kim and P. Khosla, "Real-time obstacle avoidance using harmonic potential functions," in *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, Apr. 1991, pp. 790–796 vol.1.
- [9] —, "Real-time obstacle avoidance using harmonic potential functions," *Robotics and Automation, IEEE Transactions on*, vol. 8, no. 3, pp. 338–349, June 1992.
- [10] C. Connolly and R. Grupen, "The applications of harmonic functions to robotics," *Journal of Robotic Systems*, vol. 10, no. 7, pp. 931–946, June 1993.
- [11] L. Tarassenko and A. Blake, "Analogue computation of collision-free paths," in *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, Apr 1991, pp. 540–545 vol.1.
- [12] M. Stan and W. Burleson, "Analog VLSI for robot path planning," in *Signals, Systems and Computers, 1992. 1992 Conference Record of The Twenty-Sixth Asilomar Conference on*, Oct. 1992, pp. 915–919 vol.2.
- [13] G. Marshall and L. Tarassenko, "Robot path planning using VLSI resistive grids," in *Artificial Neural Networks, 1993., Third International Conference on*, May 1993, pp. 163–167.
- [14] M. Stan, W. Burleson, C. Connolly, and R. Grupen, "Analog VLSI for robot path planning," *The Journal of VLSI Signal Processing*, vol. 8, no. 1, pp. 61–73, 1994.
- [15] G. Marshall and L. Tarassenko, "Robot path planning using VLSI resistive grids," *Vision, Image and Signal Processing, IEE Proceedings -*, vol. 141, no. 4, pp. 267–272, Aug 1994.
- [16] K. Althofer, D. Fraser, and G. Bugmann, "Rapid path planning for robotic manipulators using an emulated resistive grid," *Electronics Letters*, vol. 31, no. 22, pp. 1960–1961, Oct. 1995.
- [17] M. Kanaya, G.-X. Cheng, K. Watanabe, and M. Tanaka, "Shortest path searching for robot walking using an analog resistive network," in *Circuits and Systems, 1994. ISCAS '94., 1994 IEEE International Symposium on*, vol. 6, May–2 Jun 1994, pp. 311–314 vol.6.
- [18] D. Powers, *Boundary value problems: and partial differential equations*. Academic Press, 2010.
- [19] C. Mead, "Analog VLSI and neural system," *Ed. Addison Wesley. USA*, 1989.
- [20] S. Liu, *Analog VLSI: circuits and principles*. The MIT press, 2002.
- [21] R. Olfati-Saber, "Near-Identity Diffeomorphisms and Exponential  $\epsilon$ -Tracking and  $\epsilon$ -Stabilization of First-Order Non-holonomic SE (2) Vehicles," in *Proceedings of the American Control Conference*. Citeseer, 2002.
- [22] M. Robots. (2011, Feb.) Mobile Robots Website. [Online]. Available: [www.mobilerobots.com](http://www.mobilerobots.com)
- [23] R. Borras. (2005, May) Blob Detection. . [Online]. Available: [www.ros.org/doc/api/cvblobslib/html/classCBlob.html](http://www.ros.org/doc/api/cvblobslib/html/classCBlob.html)
- [24] B. Gerkey, R. Vaughan, K. Stoy, A. Howard, G. Sukhatme, and M. Mataric, "Most valuable player: a robot device server for distributed control," in *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, 2001.
- [25] S. Koziol, D. Lenz, S. Hilsenbeck, S. Chopra, P. Hasler, and A. Howard, "Using floating-gate based programmable analog arrays for real-time control of a game-playing robot," in *Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on*, Oct. 2011, pp. 3566–3571.
- [26] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2003.