Volume 2, Issue 4, April 2012 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

An Efficient Architecture for Robotic Path Planning

M.Vijay
P.G. Scholar
Dept. of VLSI DESIGN
SRI RAMAKRISHNA ENGINEERING COLLEGE
vijayseessi@gmail.com,

Dr.M.Jagadeeswari
Professor & Head
Dept. of VLSI DESIGN
SRI RAMAKRISHNA ENGINEERING COLLEGE

Abstract-- **There are many path planning algorithms designed for mobile robots with software implementation. In the case of dynamic environments high-speed planning and recomputation of paths is necessary to avoid collision of robots with moving objects. A hardware-efficient algorithm is presented for finding a path of a mobile robot on image of an environment captured by an overhead camera. The algorithm computes the shortest path to identify collision free region for the robot. If multiple paths are provided, the path is traced to nearest path. 2-D cell architecture has been presented for the Euclidean distance Transform (EDT) and Nearest Neighbor Transform (NNT) for operating at high speed. The simulation result shows that the architecture is suitable for path planning in a dynamic environment containing obstacles with arbitrary shape and motion.**

***Keywords*— Euclidean distance transform(EDT), nearest neighbor transform(NNT), image, path planning, dynamic environment.**

I. INTRODUCTION

The development of robots that are able to assist humans in their day-to-day tasks has become a popular research area over the last few years [6], [11]. Mobile manipulator systems public service robots (PSRs1 and 2) perform delivery, patrol, and floor-cleaning jobs[8]. The guide robot provides exhibition guide services at a museum. These robots are equipped with a few laser range finders (LRFs) for localization and obstacle detection. The PSR1 and PSR2 are driven by active caster-typed holonomic omni directional wheels. The guide robot uses conventional two-wheel differential-type wheels. Basically, these robots share common control architecture. Some software components were partially modified according to the target robot system.. The control architecture design, navigation system, and behavior selection framework were proposed in [16] for each robot. The focus on the localization problem whose solution will enable robots to carry out dependable navigation in dynamic indoor environments depends upon algorithm. Localization as in [16] is one of the most important issues for successful autonomous navigation, and a great number of localization methods have been proposed so far.

Many studies have addressed about position tracking problems. According to the International Federation of Robotics (IFR), ‘‘a service robot is a robot which operates semi or fully autonomously to perform services useful to the well being of human and equipment, excluding manufacturing operations’’ [6]. These devices are typically complex systems requiring the input of knowledge from numerous disciplines. The authors have been using different software engineering techniques for the last 15 years, integrating new paradigms in the service robot development process as they emerged. This has made it possible to achieve rapid development of applications and subsequent maintenance.

Path planning is a fundamental task for a mobile robot by which it guides itself through the environment on the basis of sensory information. The potential of computational vision for robotic navigation is enormous, and vision-based path planning has been actively studied in the last decade [4]. The work has progressed on two separate fronts: 1) vision-based navigation of indoor robot where the complete knowledge of the environment is available [9],[13] and 2) vision-based navigation of outdoor robots where partial knowledge of the environment is only often available. Many existing path-planning algorithms have been designed for implementation in software. In the case of dynamic environments, high-speed planning and recomputation of paths is necessary to avoid collisions of robots with moving objects, particularly when new objects enter the environment suddenly or when moving objects change their predicted course. In such cases, the computational requirement exceeds the computing power of present-day general-purpose processors that implement the path-planning algorithm. It is desirable to develop specialized hardware-directed solutions which operate at high speed and which offer additional advantages, such as reconfigurability and portability. Programmable hardware devices such as field-programmable gate arrays (FPGAs) nowadays provide advanced features and resources to allow rapid prototyping of system-on-chips[12][14].

A new vision-based hardware-directed algorithm for tracing a path for an indoor mobile robot that is translating as well as rotating is proposed.
The proposed path planning scheme has the following features.

1) The algorithm involves simple local neighborhood operations on integer data for constructing the distance map and for tracing the path on an environment image captured by an overhead camera.
2) The algorithm can be easily mapped onto a cellular architecture with locally interconnected 2-D array of cells.
3) A cell has simple computational units and a few registers.
4) All the pixels can be processed in parallel by the cells.
5) The architecture can be operated at high speed due to its local interconnections.
6) The architecture is scalable.

This paper is organized as follows. Section II describes the construction of a distance map and the path planning on the constructed distance map. A new parallel algorithm for the path planning on an environment image is discussed Section III. The results for path planning are presented and explained in Section IV. Section V concludes this paper.

II. Methodology

Several algorithms for path planning are available for implementation on general-purpose processors [3], [10]. However, hardware-directed approaches are relatively recent. In particular, hardware-directed schemes have concentrated on two approaches for map construction. One approach is based on visibility graphs, while the other is based on Voronoi diagrams.

A popular structure to represent the environment in which a robot operates is the visibility graph. The graph has among its nodes, the vertices of the objects[7]. Arcs in the graph connect the points x and y that can see one another. Construction of the graph precedes path finding for the robot. For shortest path computations for mobile robots and autonomous vehicles, one typically considers the reduced visibility graph instead of the complete visibility graph. Traditionally, visibility graph computation has focused on sequential algorithms and software implementations. A direct method to construct the complete graph in an environment with n nodes would require $O(n^3)$ time. Since on a general- purpose computer, all tasks are executed in a sequential fashion, the visibility graph computation is time consuming and not appropriate for real-time applications. A new hardware-directed method for various phases of tangent construction, a central component in visibility graph construction is presented in [17]. In [7], a hardware-directed algorithm has been proposed for the construction of the complete visibility graph. However, the visibility-graph-based approaches assume the approximation of robot and obstacles by circumscribing polygons. The approximation requires considerable preprocessing. The visibility graph and related approaches are model-based, and they require modeling the environment before computing the graph. Moreover, the obstacles are approximated by polygonal shapes.

Another useful geometrical structure for path planning is the Voronoi diagram. Some designs of array-type architectures for the construction of Voronoi diagram on an environment image are available in the literature [17]. The authors have applied it to path planning for a diamond-shaped robot on a synthesized image containing simple obstacles. The diagram is constructed considering the interactions between features belonging to the same obstacle as well as those from different obstacles, and therefore, it has extra branches. Geometric data structures play an important role in various applications. Among them is the Voronoi diagram, whose applications include robot path planning, pattern classification, and image processing. The construction of Voronoi diagram is a fundamental problem in computational geometry, and the one that is constructed typically is the continuous one based on assumption of a model for the objects (polygon, curve, etc.). The Voronoi region of an object consists of all pixels which are closest to that object. The Voronoi diagram is quite complex when an image of real obstacles is considered, and the path planning on such a diagram is difficult. A new very large scale integration (VLSI) algorithm for construction of the Euclidean distance-based Voronoi diagram is proposed in [17]. The algorithm has linear time complexity (linear in image size). The algorithm involves simple computations based on local information, and is, therefore, amenable for hardware implementation. The algorithm considers the entire obstacle and not its features for the construction of a Voronoi diagram and so it has no extra branches. However, path planning has not been attempted using the diagram in any of these works.

Very recently, an algorithm has been presented in [16] for computing the actual path on a binary image of an environment. The method constructs the path from the start point to the goal but on the Euclidean distance transform (EDT) and Nearest Neighbor Transform (NNT) of the image. A straightforward realization of the method in hardware has been presented. In an earlier work [16], hardware architecture for the distance transform is available. In order to achieve a complete path-planning solution in hardware, interfacing strategies have to be designed to integrate these architectures. A distance transformation converts a binary image which consists of foreground and background pixels into an image where every foreground pixel has a value corresponding to the minimum distance from the background. A nearest neighbor transformation assigns the identity of the nearest background pixel to each pixel of the image. The distance transform (DT) and nearest neighbor transform (NNT) have applications in image processing, machine vision and other domains. In image processing, for instance, distance transforms find applications in image analysis. It is used for the shape analysis of objects in an image. It is also used to compute the discrete skeleton, the discrete Voronoi diagram and the Hausdorff distance for images. In this paper, a novel array-architecture-based hardware solution is proposed for complete path planning on the binary image of the environment. The different operations in path planning are decomposed into simple local neighborhood operations, and these local neighborhood operations are combined to design a processing element of the architecture. The path obtained from a start point to the goal is the shortest in terms of the number of steps.

III. Path planning on an Environment Image

Given the image of an environment captured by an overhead camera, the method first constructs a distance map for the binary form of the image to determine the collision-free region. The shortest path is then constructed in the collision free region.

For a binary image of obstacles, the EDT and NNT of the image is first computed. The EDT converts the binary image to a multivalued image in which each pixel p is assigned the Euclidean distance between p and the nearest obstacle pixel. The salient feature of the algorithm is that the computation of EDT involves only integer arithmetic operations within a small neighborhood of each pixel. The algorithm computes the distance vector (Δx,Δy) for each pixel, where Δx and Δy are the number of rows and columns by which a pixel is displaced from its nearest object pixel. The Euclidean distance d is given by d = $\sqrt{\Delta x^2 + \Delta y^2}$. The (Δx,Δy) of object pixels are initialized at (0,0), and those of free-space pixels are computed iteratively starting from the pixels near the object and moving toward the far away pixels. At any iteration k, the (Δx(p),Δy(p)) of those pixels p whose nearest integer approximation to their Euclidean distance d(p) equals k are computed. That is, the values of d(p) for these pixels lie within (k − 0.5, k + 0.5]. d(p) is not an integer, and hence, $d^2(p)$ is considered. $d^2(p)$ lies within $(k^2 - k, k^2 + k]$ since $d^2(p)$ is an integer. However, $d^2(p)$ is quite large in magnitude, and it requires a large storage space in a hardware. A new integer quantity δ(p), which is much smaller than $d^2(p)$, is defined as $(k^2 + k) - d^2(p)$, and it lies in the range [0, 2k). δ is used for the computation of (Δx,Δy). $d^2(p)$ can be derived using the already computed (Δx,Δy) of eight neighbors p_i, i = 1 to 8, surrounding p, as shown in Fig. 1.

6	2	5
3	0	1
7	4	8

Fig. 1 Neighborhood N

It is given by $\min[\Delta x_i^2 + \Delta y_i^2]$ where

$$\Delta xi = \begin{cases} \Delta x(pi), & for\ i = 1\ and\ 3 \\ \Delta x(pi) + 1, & otherwise. \end{cases} \quad (1)$$

The increment by one is due to p being displaced from pi by one row. Similarly, Δy_i is given in terms of $\Delta y(p_i)$. $d^2(p)$ is now expressed as $\min[d^2(p_i)+\Delta X_i + \Delta Y_i]$ where

$$\Delta Xi = \begin{cases} 0, & for\ i = 1\ and\ 3 \\ 2\Delta x(p_i) + 1, & otherwise. \end{cases} \quad (2)$$

δ(p) is finally expressed as follows:

$$\begin{aligned} \delta(p) &= k^2 + k - d^2(p) \\ &= \max[k^2 + k - d^2(p_i) - \Delta X_i - \Delta Y_i] \\ &= \max[\delta(p_i) - \Delta X_i - \Delta Y_i] = \max[\delta_i] \end{aligned} \quad (3)$$

δ(p) ≥ 0 implies that d(p) is less than k + 0.5. The iterative computation of (Δx,Δy) proceeds as follows. δ values of the object pixels are initialized to zero. At each iteration k, the δ values of free-space pixels whose (Δx,Δy) are not yet known are computed using the already computed δ, Δx, and Δy of the neighbors. If δ(p) ≥ 0, then (Δx(p),Δy(p)) corresponds to $(\Delta x_i, \Delta y_i)$, where the subscript i pertains to the pixel pi that gives maximum δ_i in (3).p can be assigned an integer distance d_I which is equal to k. This integer distance is sufficient to determine the collision-free region for the given robot. Once the (Δx,Δy) of a pixel is known, its δ should be updated for at least two successive iterations, as it depends on k. The updating allows the use of δ for the computation of (Δx,Δy) of the neighbors. $\delta_k(p)$ at iteration k is derived from $\delta_{k-1}(p)$ at iteration k − 1 as follows:

$$\begin{aligned} \delta_k(p) &= k^2 + k - d^2(p) \\ &= 2k + [(k-1)^2 + (k-1) - d^2(p)] \\ &= 2k + \delta_{k-1}(p). \end{aligned} \quad (4)$$

To keep track of pixels whose (Δx,Δy) have been computed, a flag done is assigned to each pixel, whose value is set to one when the transform values of pixels are computed at any iteration. The d_I (p) given by the iteration number k, when done is set, forms the distance map. The distance map is used to find first the collision-free region for the robot. Let d_{far} be the distance between the center of rotation and the farthest point of the robot from the center. The collision-free region consists of those pixels whose distance values (d_I) are greater than d_{far}. Consider the collision-free region as a graph. Each pixel is represented as a node, and it is connected to the eight neighboring pixels surrounding it by edges. The construction of the shortest collision-free path involves the construction of the breadth-first search (bfs) tree on the collision-free graph with its root being the goal pixel[16]. The construction of the bfs tree is continued until the start pixel is encountered. The construction involves storing the parent of each pixel. The path is then traced by following the parent nodes from the start pixel until the root node is reached. The path constructed by this method on the image is the shortest path in terms of the number of pixels. The robot is moved from one pixel to the next along the path by aligning its center of rotation to every pixel. The path planning can be extended to a scenario with multiple goals. In this case, bfs trees are constructed simultaneously with each goal pixel as a root. The path from a start pixel is traced to the nearest goal.

The pseudo code of the proposed algorithm Parallel_Path_Plan is as follows.

A. Algorithm: Parallel_Path_Plan

Inputs: An n × n binary image, dfar, ps, and pg

Outputs: Sequence of moves for the robot from ps to pg

```
Step 1: Initialization
    $\Delta x = \Delta y = \delta = 0$ & done = 1 for object pixels
    visited(p) = $\begin{cases} 1, & p = p_g \\ 0, & otherwise. \end{cases}$
```

```
        parent(p) = 0 for all p
Step 2: Find EDT
        repeat for k = 1, 2, . . .
        for all pixels p do in parallel
        Compute $\Delta x_i$, $\Delta y_i$, and $\delta_i$, i = 1 to 8
        $\delta_m = \max\{\delta_i$ with done($p_i$) = 1$\}$.
        if (done(p) = 0 ^ $\delta_m \geq 0$)
        $\Delta x(p) = \Delta x_m$
        $\Delta y(p) = \Delta y_m$
        $\delta(p) = \delta_m$
        done(p) = 1
        else if done(p) = 1
        $\delta(p) = \delta(p) + 2k$
        end if
        end for
        until done = 1 for all pixels
Step 3: Find collision-free region
        for every pixel p do in parallel
        if ($d_I$ (p) > $d_{far}$) cfr(p) = 1
        else cfr(p) = 0
        end for
Step 4: Construction of bfs tree
        repeat
        for every pixel p do in parallel
        if (visited(p) = 0)  ^ ($\exists p_i \,\varepsilon$  N[visited(p) = 1])
        visited(p) = 1
        parent(p) is pointed to $p_i$ with minimum i
        end if
        end for
        until (visited = 1 for all p where cfr = 1)
Step 5: Find sequence of moves
        p = $p_s$
        while (parent(p) $\neq$0)
        print parent(p)
        p = parent pixel of p
        end while
```

The sequences of moves are given by the pointers. The robot can take horizontal, vertical, or diagonal step based on the pointer value. It should be noted that once the bfs tree is constructed in Step 4, the shortest path from any pixel to the given goal pixel can be found out.

B. Complexity Analysis

The time and space complexities are given via Propositions 1and 2.

Proposition 1: The time complexity of Algorithm Parallel_Path_Plan is O(n^2).

Proof: The time complexity of the proposed algorithm depends mainly on the repeat-until and while loops. The statements within for-end for are executed in parallel for all pixels. The repeat-until loop in Step 2 runs until done is set to one for all pixels. In the worst case, the loop is executed for _dmax_ times, where dmax is the maximum possible distance value, which is $\sqrt{}$ 2n in the case of an n × n image. The cfr computation in Step 3 takes only constant time. The repeatuntil loop in Step 4 is executed close to n2 times in the worst case when the path obtained is a zigzag one. The while loop in Step 5 also runs close to n2 times in the worst case. Hence, the overall time complexity of the algorithm is O(n^2).

Proposition 2: The space complexity of Algorithm Parallel_Path_Plan is also O(n^2).

Proof: In the algorithm, the values of Δx, Δy, δ, cfr, visited, and parent of each pixel are stored. Since there are n2 pixels, the space complexity is O(n^2).

C. Simulation Studies

Results of VHDL code simulated using ModelSim followed by synthesis and implementation using Xilinx ISE are presented as in [15]. The design of an n × n algorithm was coded in Verilog hardware description language (HDL), and its functional behavior was tested.

The integer distance value, d(p), computed by the algorithm for every pixel p has been transformed into intensity levels. The maximum frequency of operation is shown in the Table I.

TABLE I
COMPARISON OF OPERATING FREQUENCY FOR DIFFERENT TECHNIQUES

Techniques	**Frequency(MHz)**	**Computation Time(µs)**
Voronoi Diagram	50	200
Visibility graph	160	67
BFS	232	43.1

The computation time for collision free path is less when compared to the other two techniques. The identical local operations performed in a local neighborhood of each pixel make the algorithm feasible for VLSI implementation in a two-dimensional array of locally connected identical cells.

IV. Proposed Cell Architecture

The different modules of EDT and NNT are shown in Fig. 2. The ADD-SUB module computes d_f, *i*= 1-8. The computation involves an addition and a subtraction for *i* = 2, 4, 6 & 8 and only a subtraction for *i* = 1, 3, 5 & 7. Therefore, eight subtractors and four adders are required to realise this computation. The INC module computes Δx_i and Δy_i given by eqn. 1. The computation requires twelve incrementers, six for implementing Δxi=│IΔx(pi)│ + 1, *i*= 1, *2, 4, 5, 6* & 8, and another six for implementing Δy_l = │Δy(pi)│ + 1, i=2, 3, 4, 6, 7 & 8.

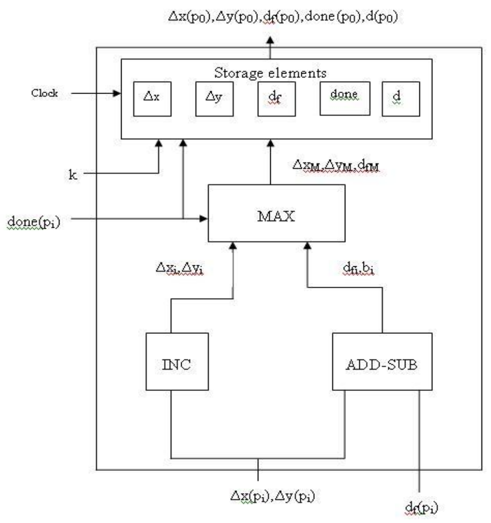

Fig. 2 EDT and NNT Architecture

Once d_{fi},Δx_i and Δy_i for *i*= 1-8 have been computed, these values are given to a MAX module along with done(p,) and borrow bits b_i from the subtractors of the ADD-SUB module. The MAX module has seven CMP-MUX (comparator-multiplexer) modules arranged in three levels to compute max[d_{fi}], *i*= 1-8,[17]. In the Figure, max[d_{fi}] is denoted by d_w. Further, the MAX module allows the values of Δx_i and Δy_i, corresponding to the *i* that yields dm. These values are denoted by Δx_M and Δy_M.

The CMP-MUX module takes two sets of inputs, set_j={d_{fi}, Δx_j, Δy_j, done(p_j), b_j} and set_k={d_{fk}, Δx_k, Δy_k, done(p_k), b_k}. It has a comparator to compare $d_{fi} > d_{fk}$ and a multiplexer that outputs either set_j or set_k depending on the value of se1 input of multiplexer. set_j is output when se1 = 0 while set_k is output when sel = 1. The sel is generated using the output cmp of comparator and the values of done and borrow bits. The comparator is designed for the simple case of comparing two unsigned binary numbers. done(p_j)= 1 means the value of d_{fi} is valid. b_j = 1 means d_{fj} is negative and cmp= 1 means $d_{fi} > d_{fk}$. The d_m, output by the MAX module, is added to *2k* where *k* is the iteration number generated by an external counter. The output $d_f(p_0)$ of the adder is given as input to the register d_j. The outputs, Δx_M and Δy_M, of the MAX module are given as inputs to the registers Ax and Ay. The *done* flip-flop is input with logic 1. In the design, the registers and flip-flop are loaded with the available inputs during the rising edge of the clock. From the switch and if statements of the algorithm in Section 3, it is clear that the clock is activated only when the following conditions are satisfied.

1. *done* of cell is not set.
2. At least a value of *done* of neighbors is logic 1.
3. $d_f(p_0)$is positive. In 2's complement representation of $d_f(p_0)$ means that the MSB is 0.

The done of cells corresponding to obstacle pixels are initialized at one and the visited of goal pixels are set to one. The initialization is done by feeding the done and visited of each row to the corresponding leftmost cell in the pipeline. When clocked, the inputs are passed through the array from left to right in each row. There are three modes of operation. EDT computation is done first. Using the initial done values, distance values d_I are computed first. Once EDT is computed, cfr is computed in the next phase using the input d_{far} and the computed d_I. In the third phase, the bfs tree is constructed using cfr and visited values. The gate counts for visibility graph and bfs and nnt are compared in Table II.

TABLE II
COMPARISON OF AREA

Techniques	Gate Counts
Visibility Graph	33026
BFS and NNT	1188

V. Conclusion

An algorithm for tracing the shortest path of a translating and rotating robot on the binary image of an environment has been given. The algorithm first constructs the distance map of the image to obtain a collision-free region and then constructs the bfs tree of pixels in the collision-free region, which defines the shortest path from any start pixel to a specific goal pixel. If multiple goals are provided, the path is traced to the nearest goal. The actual Euclidean distance value and the nearest neighbor of each pixel can be obtained from the vector Euclidean distance. Owing to simple, identical and local neighborhood operations, the proposed EDT architecture of computation is suitable for VLSI implementation. The architecture is capable of processing images at video rate for real-time path planning in a dynamic environment. Extensions to the work presented in this paper could involve handling a large indoor environment via multiple overhead cameras. Path planning can be performed by processing the images of portions of the environment in sequence and integrating the partial results. The block RAMs (BRAMs) and reconfigurability feature of FPGA can be exploited to achieve this. The FPGA will be reconfigured with the design for integration of these results to provide the complete bfs tree.

Acknowledgement

The authors thank the Management and Principal of Sri Ramakrishna Engineering College, Coimbatore for providing excellent computing facility and encouragement.

References

[1] Bellotto, N. and Huosheng, H. (2009) 'Multisensor-Based Human Detection And Tracking For Mobile Service Robots', IEEE Trans. Syst., Man, Cybern.B,Cybern., Vol.39, No.1, pp.167–181.
[2] Chen, K.H. (1997) 'Vision-Based Autonomous Land Vehicle Guidance In Outdoor Road

Environments Using Combined Line And Road Following Techniques', J. Robot. Syst., Vol.14, No.10, pp.711–728.

[3] Choset, H. Thrun, S. et al(2005) 'Principles of Robot Motion: Theory, Algorithms, and Implementations.Cambridge', MA: MIT Press.

[4] Desouza, G. and Kak, A. (2002) 'Vision For Mobile Robot Navigation: A Survey', IEEE Trans.Pattern Anal. Mach. Intell., Vol.24, No.2, pp.237– 267.

[5] Dezhen, S. (2007) 'Vision-Based Motion Planning For An Autonomous Motorcycle On Ill-Structured Roads', Auton. Robots, Vol.23, No.3, pp.197– 212.

[6] Iborra, A. Caceres, D. et al (2009) 'Design Of Service Robots', IEEE Robot. Autom. Mag., Vol.16, No.1, pp.24–33.

[7] Kei, L. S. Sridharan, K. et al (2004) 'Hardware-Efficient Schemes For Logarithmic Approximation And Binary Search With Application To Visibility Graph Construction', IEEE Trans. Ind. Electron., Vol.51, No.6, pp.1346–1348.

[8] Kim, M. Kim, S. et al (2009) 'Servicerobot For The Elderly', IEEE Robot. Autom. Mag., Vol.16, No.1, pp.34–45.

[9] Kim, D. and Nevatia, R. (1994) 'Representation And Computation Of The Spatial Environment For Indoor Navigation', in Proc. Int. Conf. Comput. Vis. Pattern Recog., pp.475–482.

[10] Latombe, J. C. (1991) 'Robot Motion Planning. Norwell', MA: Kluwer.

[11] Lee, D. and Chung, W. (2006) 'Discrete-Status-Based Localization For Indoor Service Robots', IEEE Trans. Ind. Electron., Vol.53, No.5, pp.1737–1746.

[12] Monmasson, E. and Cirstea, M. N. (2007) 'FPGA Design Methodology For Industrial Control Systems—A Review', IEEE Trans. Ind. Electron., Vol.54, No.4, pp.1824–1842.

[13] Rodrigo, R. Zouqi, M. et al (2009) 'Robust And Efficient Feature Tracking For Indoor Navigation', IEEE Trans. Syst., Man, Cybern. B, Cybern., Vol.39, No.3, pp.658–671.

[14] Rodriguez-Andina, J. J. Moure, M. J. et al (2007) 'Features, Design Tools, And Application Domains Of FPGAs', IEEE Trans. Ind. Electron., Vol.54, No.4, pp.1810–1823.

[15] Sridharan, K. and Priya, T. K. (2005) 'The Design Of A Hardware Accelerator For Real-Time Complete Visibility Graph Construction And Efficient FPGA Implementation', IEEE Trans. Ind. Electron., Vol.52, pp.1185–1187.

[16] Sudha, N. Aruppukottai Rajarathinam Mohan, (2011) 'Hardware-Efficient Image-Based Robotic Path Planning In A Dynamic Environment And Its FPGA Implementation' IEEE Transactions On Industrial Electronics, Vol.5.

[17] Sudha, N. and Sridharan, K (2004) 'A High-Speed VLSI Design And ASIC Implementation For Constructing Euclidean Distance-Based Discrete Voronoi Diagram', IEEE Trans. Robot. Autom., Vol.20, No.2, pp.352–358.