FPGA-based Design and Implementation of Real-time Robot Motion Planning

Ruige Li, Xiangcai Huang, Sijia Tian, Rong Hu and Dingxin He
Huazhong University of Science and Technology,
Wuhan, China
Email: whutddk@hust.edu.cn

Qiang Gu
PS-Micro,
Wuhan, China
Email: qiangg@ps-micro.com

***Abstract*—Collision detection in motion planning is time-consuming for robotic arm applications, especially real-time motion planning in a dynamic environment. It is necessary to design a collision detection accelerator for real-time robot motion planning. In this paper, an optimized FPGA-based design is proposed to reduce the hardware cost. Static random access memory (SRAM) is used to store encoded collision detection data for saving the FPGA look up tables (LUTs). The collision detection accelerator implementations based on SRAM and LUT are analyzed and compared in terms of hardware cost and time consumption. Experimental results are shown to prove the efficiency of the proposed design and implementation.**

***Keywords*—motion planning; collision detection; field programmable gate array; accelerator; static random access memory**

I. Introduction

Nowadays, robot motion planning is mainly based on three solutions: the artificial teaching [1], the algorithm based on mechanical and environmental model [2], and the algorithm based on randomly sampling [3]. The artificial teaching is widely used in industrial applications. The environment is static around artificial-teaching-based robots for safety. The algorithm based on mechanical and environmental model is efficient in dynamic environment, especially for automated guided vehicle control. In 6-degree-of-freedom (6-DOF) robotic arm applications, it's time-consuming to finish the model computation, because it costs much more resources to solve higher dimensional models. The algorithm based on randomly sampling is more suitable in high dimensional robot motion planning.

The probabilistic roadmap (PRM) is one of the widely used randomly sampling algorithms in robot motion planning. Kavraki et al. presented a solution of path planning in high dimensional configuration spaces by using PRM firstly in 1996 [3]. Since that, scholars have made various improvements and implementations of PRM, such as Lazy-PRM [4], Fuzzy-PRM [5] and lazy spatial network PRM [6]. It is time- and resource-consuming to apply a real-time PRM algorithm, especially in time-varying environment conditions. Pan et al. gave out a PRM solution in GPU [7]. Amato presented a PRM implementation in parallel with the help of c++ m_fork parallel micro tasking library [8]. FPGA [9] was used to accelerate the PRM for its hardware parallelism in [10]. This paper proposes an optimized PRM solution with the embedded system technology, which considers hardware cost, time consumption, flexibility and accuracy.

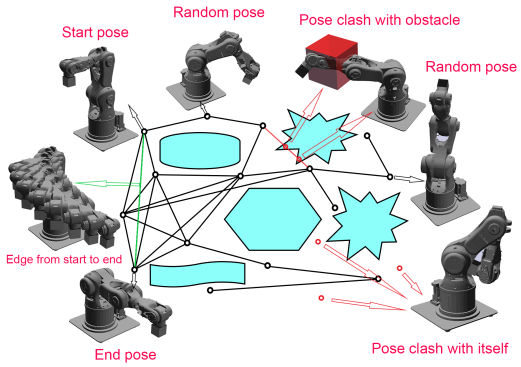


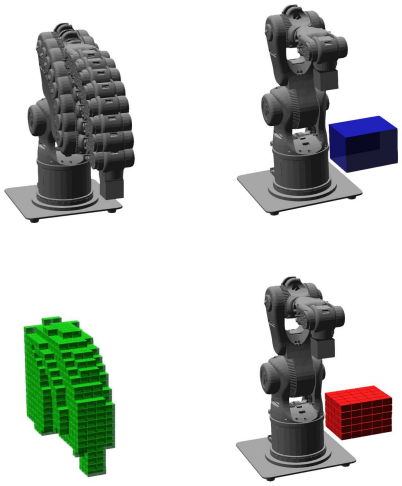
Fig. 1. Collision-free checking in the PRM for 6-DOF robotic arm.

The organization of this paper is shown as follows. Section II describes the background of PRM implementation and improvement. Section III presents the design and optimization of the proposed accelerator. The implementation and experimental results are given in Section IV. Final summary part is drawn in Section V.

II. Background

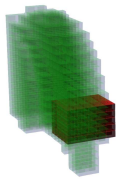
The PRM flow is described as follows. Firstly, collision-free configuration poses are sampled in the surrounding environment, where nodes and graph present the poses and environment respectively. Each node finds neighbor nodes with a distance function and a threshold. If a path from one node to its neighbor is collision-free, an edge will be added into the graph to connect those two nodes. Edges present the robot paths. Secondly, if collision-free edges exist, the graph-based algorithm Dijkstra [11] [12] can be used to find a route to connect start pose and end pose. Finally the robot moves along the edges to prevent clashing with the obstacles or itself. The collision-free checking is shown in Fig. 1. A large number of random poses and edges are generated to find an available route from start pose to end pose.

To apply a real-time PRM in time-varying environment conditions, the PRM calculation should be finished in every control period. The collision-free checking computation



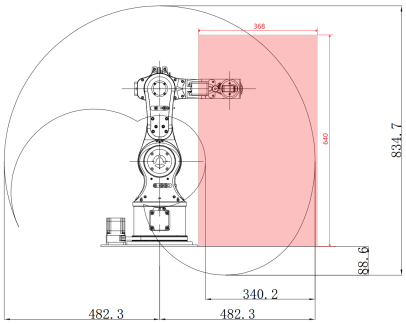
(a) Encode each edge in system preprocessing stage

(b) Detect and encode the location of obstacles in real-time operation stage

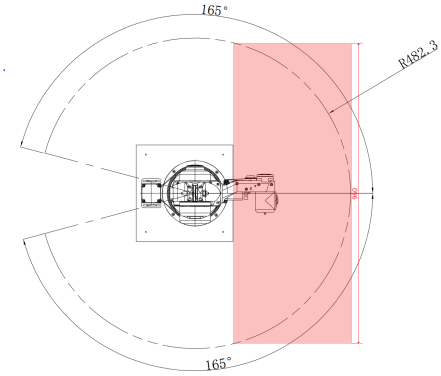


(c) Collision detection in real-time operation stage

Fig. 2. Steps to accelerate the collision detection.



(a) Front View



(b) Top View

Fig. 3. Reachable space of robotic arm [15], where obstacles are set in red area.

consumes most of computing resources in the PRM flow. And more sampling poses and edges require more time and resources in the collision-free checking.

Sean Murray et al. [13] [14] provided a solution to ease the contradiction between computing time and resource consumption. It accelerates the collision detection process in Fig. 2. In the system preprocessing stage, thousands of edges are selected by simulator. The workspace is divided into a serial of cube grids marked with indexes of "X", "Y", "Z". The indexes depend on the location in Cartesian coordinates of the workspace. Each edge is encoded with the grids' indexes. A combinational logic expression is created to present the encoded data of each edge. When the robotic arm moves along these edges, all intersected grids' entities in the workspace can be figured out by the expression. The geometric computation for collision-free checking is finished in the system preprocessing stage.

In real-time operation stage, obstacles detected by camera are encoded as a serial of cube grids' indexes in the same way. The index data is streamed over a PCIe bus to a parallel collision detection circuit. The algorithm in the collision detection circuit is to check if there is any obstacle grid coinciding with edges' grids. If a collided grid is detected, the edge is unavailable in following searching steps.

III. Optimized Embedded PRM Solution

In this paper, an optimized embedded design based on ARM plus FPGA is proposed to save cost and improve accuracy for the PRM implementation. Collision detection units (CDUs) are designed based on FPGA. The design based on [13] (hereinafter referred to as "the original design") is analyzed for comparison purpose.

The robotic arm is a 6-DOF series arm named Anno [15] in this paper. Reachable space in front of the robotic arm is sampled to detect collision. The obstacles are set in the red area of $368mm$ $(millimeter)$ * $960mm$ * $640mm$ in Fig. 3.

The edges are encoded as a serial of grids based on [13]. The obstacles are detected by camera and encoded by the ARM processor. The encoded data is transferred to CDUs in the FPGA to accelerate the collision-free checking. When

the checking process is finished in the FPGA, the results are transferred back to the ARM processor.

A. Encoding Accuracy Optimization

The encoding accuracy depends on the volume of grids. The encoding method with smaller grids can provide higher edge resolution and accuracy [13]. The pivotal area where the robotic arm often reaches can use more and smaller grids, while other area can use less and bigger grids.

The brief encoding rules are shown below: (1) cover the reachable area for both robotic arm and obstacles; (2) delete the unreachable area for either robotic arm and obstacles; (3) use denser and smaller grids in pivotal area to improve resolution.

The input resolution is set as 14 bits. Considering the above rules, three kinds of encoding methods are given as follows.

a) Cartesian Coordinate: Fig. 4 (a) shows the grids are arranged in Cartesian Coordinates. Every grid has same volume in the cube-shape. It doesn't follow the rule (3) well.

b) Spherical Coordinate: Distance, azimuth angle and zenith angle in spherical coordinate are shown in Fig. 4 (b) instead of length, width and height in Cartesian coordinates. Each grid becomes four-prism-shape. It improves the accuracy in center. But mass of useless area is encoded. It doesn't follow the rule (2) well.

c) Polar Coordinate: Encoding edges in polar coordinates in horizontal section is used in Fig. 4 (c). The radius ranges from $120mm$ to $488mm$ with 4 bits. Each bit presents $23mm$. The angle ranges from $-1.3197rad$ to $1.3197rad$ with 5 bits. Each bit presents $0.0825rad$. Each cube-shape grid becomes trapezoid-shape in horizontal section. The grids far from origin in horizontal cross section have bigger volume. The resolution in pivotal area is higher than Cartesian coordinates. The encoding method based on polar coordinates is chosen in the proposed design. The conversion equation to Cartesian coordinates is given below.

$$x = r\cos\theta \tag{1}$$

$$y = r\sin\theta \tag{2}$$

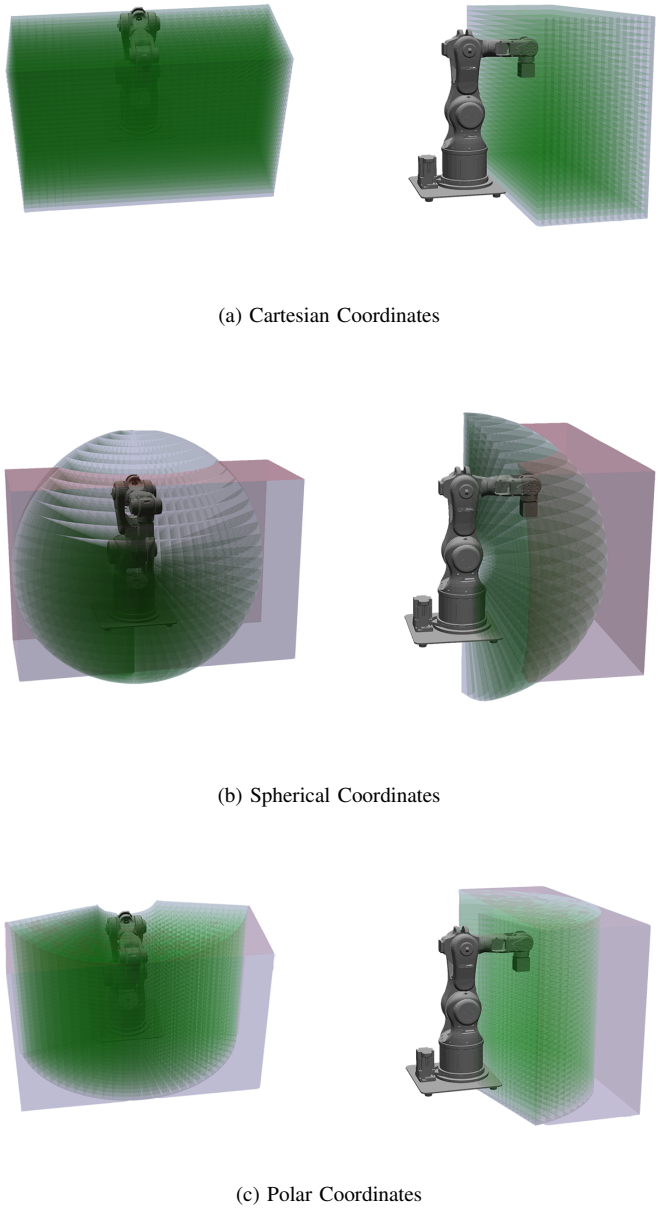
where x is width, y is length, r is radius and θ is angle.

B. Optimized Implementation

Three modules are proposed to complete the process: 1) input grids' indexes module, 2) CDU and 3) return result and reset module.

1) Input Grids' Index Module: Obstacles are detected through camera and encoded by the ARM processor. Encoded data is transferred into the FPGA to accelerate the collision detection process. The encoded data size depends on the number of grids and the resolution of input indexes.

2) CDU: The CDU function is to check whether a grid entity of obstacle intersects with the grids' entities of edges. The CDU input is index of an obstacle grid. The CDU output is a serial of bit-masks showing whether these edges are available. The bit width of CDU output depends on the number of edges. For example, the output has 4096 bits if 4096 edges are used to create the encoded data.



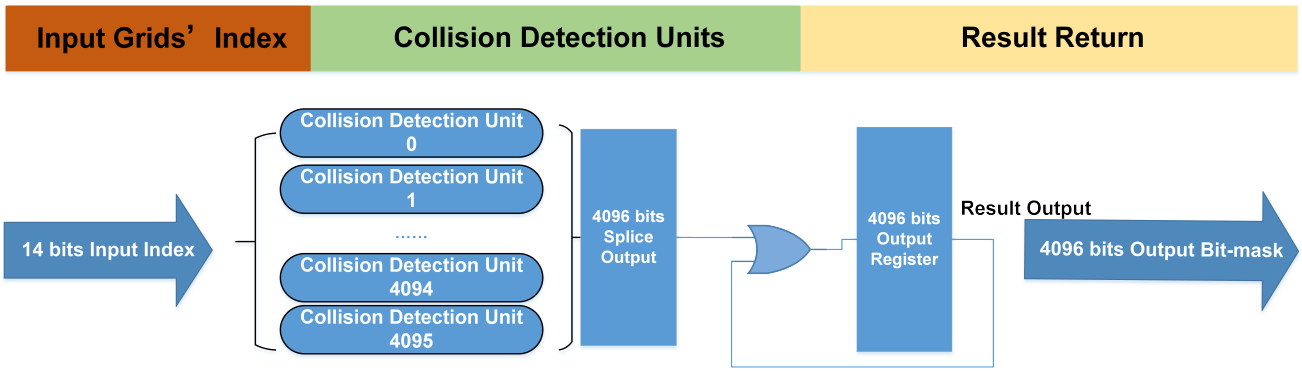
(a) Cartesian Coordinates

(b) Spherical Coordinates

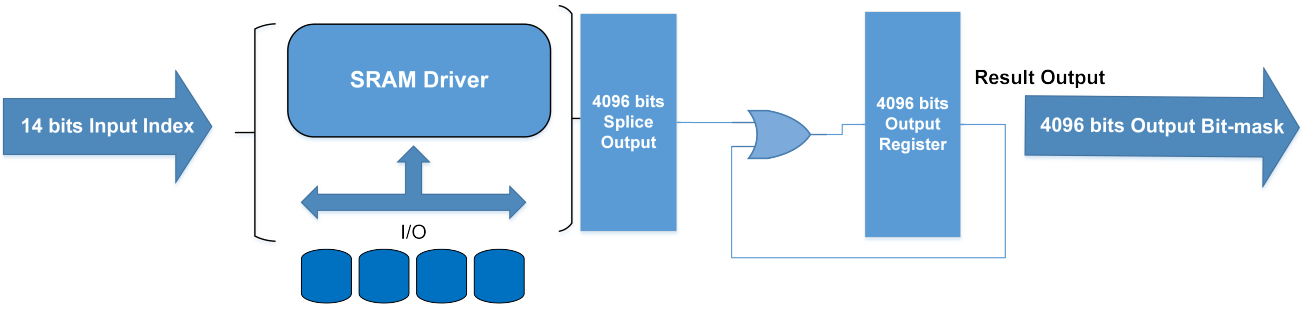
(c) Polar Coordinates

Fig. 4. Encoding methods based on Cartesian coordinates, spherical coordinates and polar coordinates, where (a) 16384 cube-shape grids (14 bits input) present whole area that obstacles can appear in Cartesian coordinates, (b) 16384 four-prism-shape grids (14 bits input) present part of area that obstacles can appear in spherical coordinate, (c) 16384 trapezoidal-cube-shape grids (14 bits input) present part of area that obstacles can appear in polar coordinates in horizontal section.

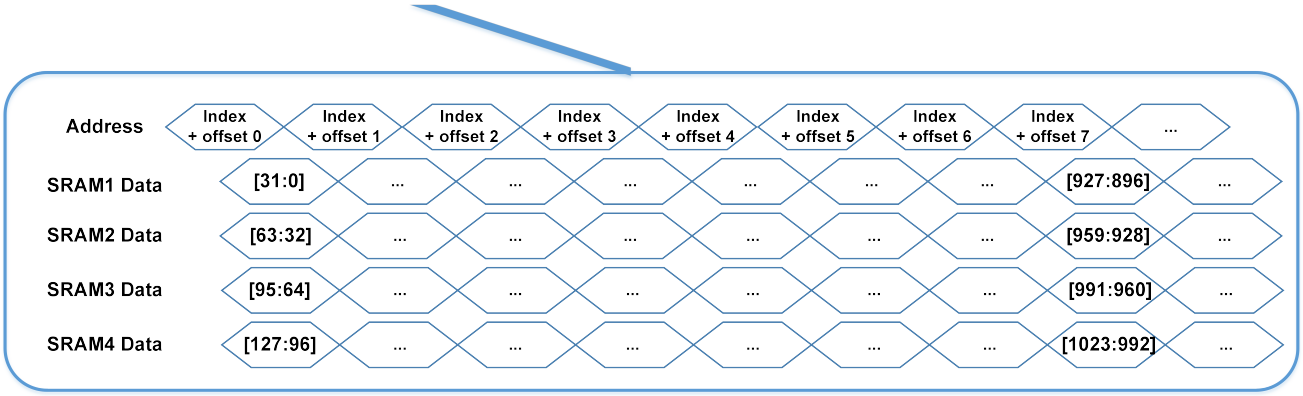
The logical function in FPGA mainly based on look up table (LUT). There are 2^n possible outputs with n-bit input in a FPGA-based logical function. Actually the input and output bit widths of LUT are usually much smaller than the requirement of encoded data (14 bits of input, more than 1024 bits of output). Encoded data is stored in SRAM instead of FPGA



(a) Original design based on LUTs



(b) Proposed design based on SRAMs



(c) Timing diagram of SRAMs to read out collision detection result

Fig. 5. Difference between the original design in [13] and proposed design.

in the proposed CDU. SRAM has bigger input and output bit widths. It is more efficient than LUT to store encoded data. Fig. 5 shows the difference between the proposed design and original design in [13]. In the proposed design, the encoded data is stored without compression in SRAM for real-time purpose.

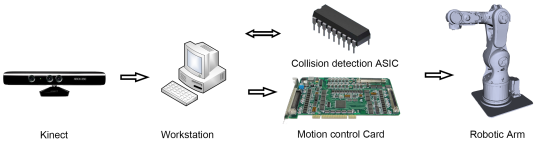
3) Return Result and Reset Module: OR logic calculation is operated with the CDU output and previous results. The results are saved in the FPGA registers. Finally, a serial of bit-masks are sent back to the ARM processor. They present whether the edges are available.

IV. Implementation and Experiment

A. Implementation

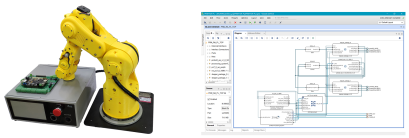
Fig. 6 shows the hardware of the proposed design and original design. The Zynq SoC XC7Z020 [16] are selected to implement the proposed design. There are four external SRAMs (IS61WV51232) in the hardware platform where a 32-bit result can be read in one clock cycle. And the maximum read frequency is 100MHz. The experiment is done with using Vivado 2018.3. The whole design structure is smaller and more concise by integrating the workstation, collision detection module, and motion control unit together.

In the experiment, The available number of LUTs is $53.2K$. And the maximum execution time to finish collision detection



(a) Original Design

(b) Proposed Design



(c) Hardware Platform

Fig. 6. Hardware of the proposed design and original design.

is set to $1ms$. The following expression shows the required memory space in the proposed design.

$$M = \frac{2^n b}{8} Byte \tag{3}$$

where n is bits of input, b is bits output, M is required memory space.

FPGA in the proposed design is used as an external SRAM controller to provide a parallel logic operator for high throughput. The SRAMs are driven by 80MHz clock in the proposed implementation. The SRAM timing diagram is shown in Fig. 5 (c). The CDUs work in 80MHz clock frequency. The original design is implemented based on [13] with some modifications for Zynq resource constraint.

B. Result and Analysis

The LUTs and time consumption are evaluated with 1024, 2048, 3072 and 4096 edges in the two designs. The result is shown in Fig. 7. Considering the restriction of LUTs and time consumption, the proposed design satisfies the requirement though multiple times of cycles are needed to complete one collision detection computation. But only 1024 edges can be applied in the original design. The results indicate that the implementations with 1024, 2048, 3072 and 4096 edges are all usable in the proposed design with fewer LUTs.

V. Conclusions

In this paper, a hardware accelerator and its optimized implementation are proposed for collision detection. Polar coordinate is selected to encode the grids compared with Cartesian coordinate and spherical coordinate. SRAMs are used in the proposed design to save FPGA LUTs without encoded data compression. The proposed design is implemented on Zynq SoC. The experimental results have validated the

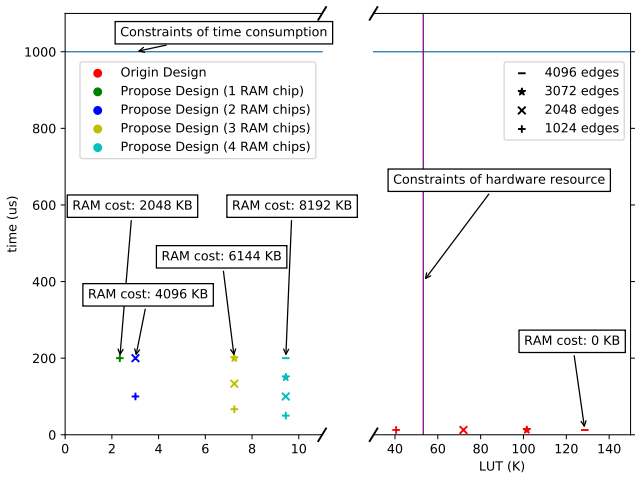


Fig. 7. Experimental results of the proposed design and original design.

good balance between the time consumption and hardware resource in the proposed design. The future study will be focused on the system optimization of the proposed design on SoC with machine vision.

Acknowledgment

This work was supported by National Natural Science Foundation of China under grant No.61672244.

References

[1] C. Park, K Park, D. I. Park, J. H. Kyung, "Dual arm robot manipulator and its easy teaching system", *IEEE International Symposium Assembly and Manufacturing*, pp. 242-247, 2009.

[2] T. Chen, Q. Zhang, "Robot motion planning based on improved artificial potential field", *3rd International Conference on Computer Science and Network Technology*, pp. 1208-1211, 2013.

[3] L. E. Kavraki, P. Svestka, J. C. Latombe, M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces", *IEEE Transactions on Robotics and Automation*, pp. 566-580, 1996.

[4] R. Bohlin, L. E. Kavraki. "Path planning using Lazy PRM", *IEEE International Conference on Robotics and Automation*, pp. 521-528, 2000.

[5] C. L. Nielsen, L. E. Kavraki, "A two level fuzzy PRM for manipulation planning", *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1716-1721, 2000.

[6] A. Gasparri, G. Oliva, S. Panzieri, "Path planning using a lazy spatial network PRM", *17th Mediterranean Conference on Control and Automation*, pp. 940-945, 2009.

[7] J. Pan, D. Manocha, "GPU-based parallel collision detection for fast motion planning", *International Journal of Robotics Research*, 2012.

[8] N. M. Amato, L. K. Dale, "Probabilistic roadmap methods are embarrassingly parallel", *IEEE International Conference on Robotics and Automation*, pp. 688-694, 1999.

[9] E. Monmasson, M. N. Cirstea, "FPGA design methodology for industrial control systems -- a review", *IEEE Transactions on Industrial Electronics*, pp. 1824-1842, 2007.

[10] N. Atay, B. Bayazit, "A motion planning processor on reconfigurable hardware", *IEEE International Conference on Robotics and Automation*, pp. 125-132, 2006.

[11] H. I. Kang, B. Lee, K. Kim, "Path Planning Algorithm Using the Particle Swarm Optimization and the Improved Dijkstra Algorithm", *IEEE Pacific-Asia Workshop on Computational Intelligence and Industrial Application*, pp. 1002-1004, 2008.

[12] N. Makariye, "Towards shortest path computation using Dijkstra algorithm", *International Conference on IoT and Application*, pp. 1-3, 2017.

[13] S. Murray, W. Floyd-Jones, Y. Qi, G. Konidaris, D. J. Sorin, "The microarchitecture of a real-time robot motion planning accelerator", *49th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 1-12, 2016.
[14] S. Murray, W. Floyd-Jones, Y. Qi, D. Sorin, G. Konidaris, "Robot motion planning on a chip", *Robotics: Science and Systems*, 2016.
[15] Anno, Inc., http://www.robotanno.com.
[16] Xilinx, Inc., "XMP100 Cost-Optimized Portfolio Product Tables and Product Selection Guide(v1.9.1)", 2018.
[17] S. Lian, Y. Han, X. Chen, Y. Wang, H. Xiao, "Dadu-P: A Scalable Accelerator for Robot Motion Planning in a Dynamic Environment", *55th ACM/ESDA/IEEE Design Automation Conference*, pp. 1-6, 2018.
[18] Xilinx, Inc., "Low-Cost EasyPath FPGAs Offer Promise to ASSP Companies", 2005.
[19] Xilinx, Inc., "Comparing and Contrasting FPGA and Microprocessor System Design and Development", 2004.
[20] D. A. Patterson, P. Chen, G. Gibson, R. H. Katz. "Introduction to Redundant Arrays of Inexpensive Disks (RAID)", *IEEE Computer Society International Conference*, pp. 112-117, 1989.
[21] N. A. Rodrguez-Olivares, A. Gmez-Hernndez, L. Nava-Balanzar, H. Jimnez-Hernndez, J. A. Soto-Cajiga, "FPGA-Based Data Storage System on NAND Flash Memory in RAID 6 Architecture for In-Line Pipeline Inspection Gauges", *IEEE Transactions on Computers*, pp. 1046-1053, 2018.