Hindawi Publishing Corporation
Advances in Fuzzy Systems
Volume 2012, Article ID 698062, 11 pages
doi:10.1155/2012/698062

Research Article

Speedup of Interval Type 2 Fuzzy Logic Systems Based on GPU for Robot Navigation

Long Thanh Ngo, Dzung Dinh Nguyen, Long The Pham, and Cuong Manh Luong

Department of Information Systems, Le Quy Don Technical University, No 100, Hoang Quoc Viet St., Cau Giay, Hanoi, Vietnam

Correspondence should be addressed to Long Thanh Ngo, ngotlong@gmail.com

Received 23 March 2012; Accepted 31 July 2012

Academic Editor: Oscar Montiel Ross

As the number of rules and sample rate for type 2 fuzzy logic systems (T2FLSs) increases, the speed of calculations becomes a problem. The T2FLS has a large membership value of inherent algorithmic parallelism that modern CPU architectures do not exploit. In the T2FLS, many rules and algorithms can be speedup on a graphics processing unit (GPU) as long as the majority of computation a various stages and components are not dependent on each other. This paper demonstrates how to install interval type 2 fuzzy logic systems (IT2-FLSs) on the GPU and experiments for obstacle avoidance behavior of robot navigation. GPU-based calculations are high-performance solution and free up the CPU. The experimental results show that the performance of the GPU is many times faster than CPU.

1. Introduction

Graphic processing units (GPUs) give a new way to perform general purpose computing on hardware that is better suited for the complicated fuzzy logic systems. However, the installation of these systems on the GPUs is also difficult because many algorithms are not designed in a parallel format conducive to GPU processing. In addition, there may be too many dependencies at various stages in the algorithm that will slow down GPU processing.

Type 2 fuzzy logic has been developed in theory and practice to obtain achievement for real applications [1–10]. A review of the methods used in the design of interval type 2 fuzzy controllers has been considered [11]. However, the complexity of T2FLS is still large and many researches focus to reduce these problems on the approach to algorithm or hardware implementation. Some proposals implement type 2 FLS focus on the design, and software development for coding a high-speed defuzzification stage based on the average method of two type 1 FLS [12] or the optimization of an incremental fuzzy PD controller based on a genetic algorithm [13]. More recent works, where an interval type 2 FIS Karnik-Mendel is designed, tested and implemented based on hardware implementation [14]. Using GPUs for general purpose computing is mentioned in many researches, recently, to speed up complicated algorithms by parallelizing to suitable GPU architecture, especially for applications of fuzzy logic. Anderson et al. [15] presented a GPU solution for the fuzzy C-means (FCMs). This solution used OpenGL and Cg to achieve approximately two orders of magnitude computational speedup for some clustering profiles using an nVIDIA 8800 GPU. They later generalized the system for the use of non-Euclidean metrics [16]. Further, Sejun Kim [17] describes the method used to adapt a multilayer trees structure composed of fuzzy adaptive units into CUDA platforms. Chiosa and Kolb [18] present a framework for mesh clustering solely implemented on the GPU with a new generic multilevel clustering technique. Chia et al. [19] proposes the implementation of a zero-order TSK-fuzzy neural network (FNN) on GPUs to reduce training time. Harvey et al. [20] present a GPU solution for fuzzy inference. Anderson et al. [21] present a parallel implementation of fuzzy inference on a GPU using CUDA. Again, over two orders of speed improvement of this

naturally parallel algorithm can be achieved under particular inference profiles. One problem with this system, as well as the FCM GPU implementation, is that they both rely upon OpenGL and Cg (graphics libraries), which makes the system and generalization of its difficult for newcomers to GPU programming.

Therefore, we carried out fuzzy logic systems analysis in order to take advantage of GPUs processing capabilities. The algorithm must be altered in order to be computed fast on a GPU. In this paper, we explore the use of nVIDIA's Compute Unified Device Architecture (CUDA) for the implementation of an interval type 2 fuzzy logic system (IT2FLS). This language exposes the functionality of the GPU in a language that most programmers are familiar with, the C/C++ language that the masses can understand and more easily integrate into applications that do not have the need otherwise to interface with a graphics API. Experiments are implemented for obstacle avoidance behavior of robot navigation based on nVIDIA platform with the summarized reports on runtime.

The paper is organized as follows: Section 2 presents an overview on GPUs and CUDA; Section 3 introduces the interval type 2 fuzzy logic systems; Section 4 proposes a speedup of IT2FLS using GPU and CUDA; Section 5 presents experimental results of IT2FLS be implemented on GPUs in comparing with on CPU; Section 6 is conclusion and future works.

2. Graphics Processor Units and CUDA

Traditionally, graphics operations, such as mathematical transformations between coordinate spaces, rasterization, and shading operations have been performed on the CPU. GPUs were invented in order to offload these specialized procedures to advanced hardware better suited for the task at hand. Because of the popularity of gaming, movies, and computer-aided design, these devices are advancing at an impressive rate. Classically, before the advent of CUDA, general purpose programming on a GPU (GPGPU) was performed by translating a computational procedure into a graphics format that could be executed in the standard graphics pipeline. This refers to the process of encoding data into a texture format, identifying sampling procedures to access this data, and converting the algorithms into a process that utilized rasterization (the mapping of array indices to graphics fragments) and frame buffer objects (FBO) for multipass rendering. GPUs are specialised stream processing devices.

This processing model takes batches of elements and computes a similar independent calculation in parallel to all elements. Each calculation is performed with respect to a program, typically called a kernel. GPUs are growing at a faster rate than CPUs, and their architecture and stream processing design makes them a natural choice for many algorithms, such as computational intelligence algorithms that can be parallelised.

nVIDIA's CUDA is a data-parallel computing environment that does not require the use of a graphics API, such as OpenGL and a shader language. CUDA applications are created using the C/C++ language. CPU and GPU programs are developed in the same environment (i.e., a single C/C++ program), and the GPU code is later translated from C/C++ to instructions to be executed by the GPU. nVIDIA has even gone as far as providing a CUDA Matlab plugin. A C/C++ program using CUDA can interface with one GPU or multiple GPUs can be identified and utilized in parallel, allowing for unprecedented processing power on a desktop or workstation.

CUDA allows multiple kernels to be run simultaneously on a single GPU. CUDA refers to each kernel as a grid. A grid is a collection of blocks. Each block runs the same kernel but is independent of each other (this has significance in terms of access to memory types). A block contains threads, which are the smallest divisible unit on a GPU. This architecture is shown in Figure 1.

The next critical component of a CUDA application is the memory model. There are multiple types of memory and each has different access times. The GPU is broken up into read-write perthread registers, read-write perthread local memory, read-write per-block shared memory, read-write per-grid global memory, read-only per-grid constant memory, and read-only per-grid texture memory. This model is shown in Figure 2.

Texture and constant memory have relatively small access latency times, while global memory has the largest access latency time. Applications should minimize the number of global memory reads and writes. This is typically achieved by having each thread read its data from global memory and store its content into shared memory (a block level memory structure with smaller access latency time than global memory). Threads in a block synchronize after this step. Memory is allocated on the GPU using a similar mechanism to malloc in C, using the functions cudaMalloc and cudaMallocArray. GPU functions that can be called by the host (the CPU) are prefixed with the symbol "*global*", GPU functions that can only be called by the GPU are prefixed with "*device*", and standard functions that are callable from the CPU and executed on the CPU are prefixed with "*host*" (or the symbol can be omitted, as it is the default). GPU functions can take parameters, as in C. When there are a few number of variables that the CPU would like to pass to the GPU, parameters are a good choice; otherwise, such as in the case of large arrays, the data should be stored in global, constant, or texture memory and a pointer to this memory is passed to the GPU function. Whenever possible, data should be kept on the GPU and not transferred back and forth to the CPU.

3. Interval Type 2 Fuzzy Logic Systems

3.1. Type 2 Fuzzy Sets. A type 2 fuzzy set in X is denoted $\widetilde{A}$, and its membership grade of $x \in X$ is $\mu_{\widetilde{A}}(x,u)$, $u \in J_x \subseteq [0,1]$, which is a type 1 fuzzy set in $[0,1]$. The elements of domain of $\mu_{\widetilde{A}}(x,u)$ are called primary memberships of x in $\widetilde{A}$, and memberships of primary memberships in $\mu_{\widetilde{A}}(x,u)$ are called secondary memberships of x in $\widetilde{A}$.

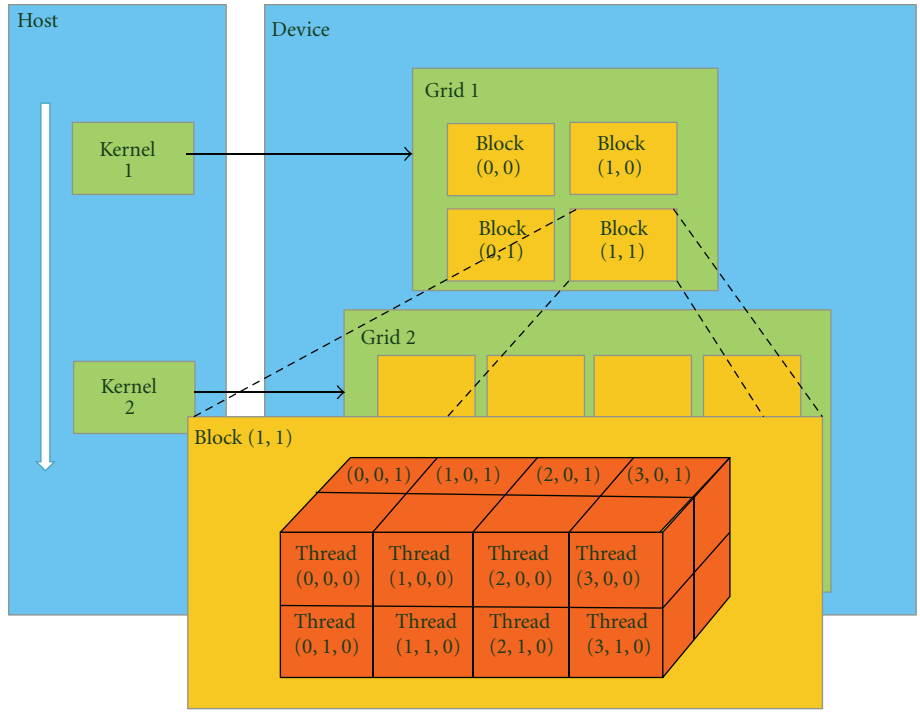

Figure 1: CUDA-processing model design [22].

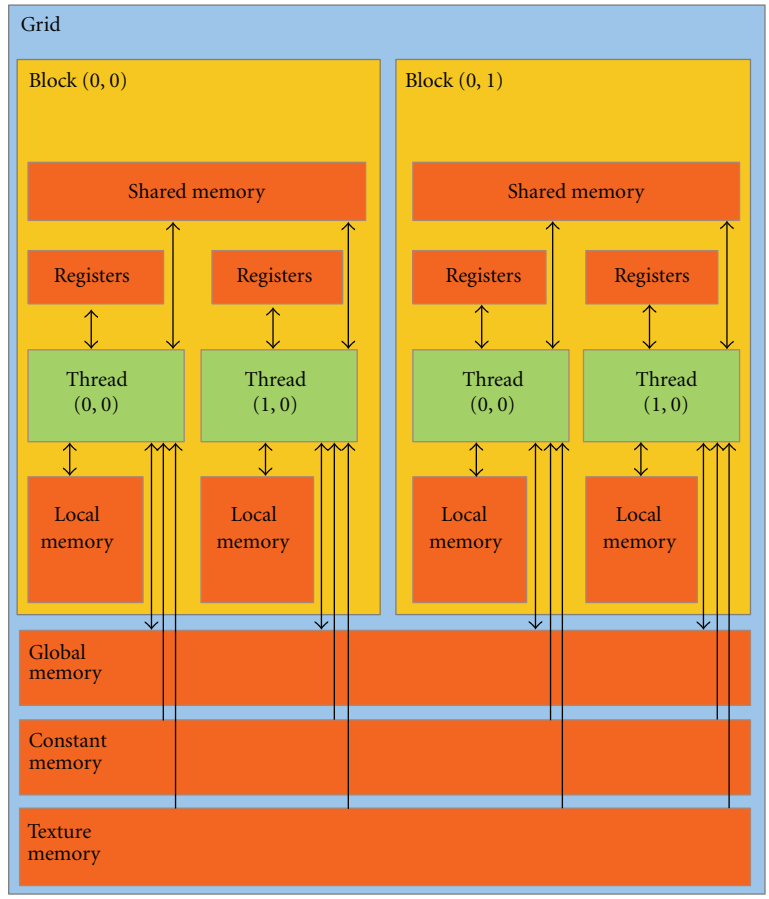

Figure 2: CUDA GPU memory model design [22].

Definition 1. A *type 2 fuzzy set*, denoted $\widetilde{A}$, is characterized by a type 2 membership function $\mu_{\widetilde{A}}(x,u)$ where $x \in X$ and $u \in J_x \subseteq [0,1]$, that is,

$$\widetilde{A} = \left\{ \left((x,u), \mu_{\widetilde{A}}(x,u) \right) \mid \forall x \in X, \forall u \in J_x \subseteq [0,1] \right\} \quad (1)$$

or

$$\widetilde{A} = \int_{x \in X} \int_{u \in J_x} \frac{\mu_{\widetilde{A}}(x,u)}{(x,u)}, \quad J_x \subseteq [0,1] \quad (2)$$

in which $0 \le \mu_{\widetilde{A}}(x,u) \le 1$.

At each value of x, say $x = x'$, the 2D plane whose axes are u and $\mu_{\widetilde{A}}(x',u)$ is called a *vertical slice* of $\mu_{\widetilde{A}}(x,u)$. A *secondary membership function* is a vertical slice of $\mu_{\widetilde{A}}(x,u)$. It is $\mu_{\widetilde{A}}(x = x',u)$ for $x \in X$ and for all $u \in J_{x'} \subseteq [0,1]$, that is,

$$\mu_{\widetilde{A}}(x = x',u) \equiv \mu_{\widetilde{A}}(x') = \int_{u \in J_{x'}} \frac{f_{x'}(u)}{u}, \quad J_{x'} \subseteq [0,1] \quad (3)$$

in which $0 \le f_{x'}(u) \le 1$.

In manner of embedded fuzzy sets, a type 2 fuzzy sets [1] is union of its type 2 embedded set, that is,

$$\widetilde{A} = \sum_{j=1}^{n} \widetilde{A}_e^j, \quad (4)$$

where $n \equiv \prod_{i=1}^{N} M_i$ and $\widetilde{A}_e^j$ denoted the jth type 2 embedded set of $\widetilde{A}$, that is,

$$\widetilde{A}_e^j \equiv \left\{ \left(u_i^j, f_{x_i}\left(u_i^j\right) \right), \ i = 1,2,\ldots,N \right\}. \quad (5)$$

where $u_i^j \in \{u_{ik}, k = 1,\ldots,M_i\}$.

Type 2 fuzzy sets are called interval type 2 fuzzy sets if the secondary membership function $f_{x'}(u) = 1$, for all $u \in J_x$, that is, a type 2 fuzzy set is defined as follows.

Definition 2. An *interval type 2 fuzzy set* $\widetilde{A}$ is characterized by an interval type 2 membership function $\mu_{\widetilde{A}}(x,u) = 1$ where $x \in X$ and $u \in J_x \subseteq [0,1]$, that is,

$$\widetilde{A} = \{((x,u),1) \forall x \in X, \forall u \in J_x \subseteq [0,1]\}. \quad (6)$$

Uncertainty of $\widetilde{A}$, denoted FOU, is union of primary functions that is $\mathrm{FOU}(\widetilde{A}) = \bigcup_{x \in X} J_x$. Upper/lower bounds of membership function (UMF/LMF), denoted $\overline{\mu}_{\widetilde{A}}(x)$ and $\underline{\mu}_{\widetilde{A}}(x)$, of $\widetilde{A}$ are two type 1 membership function and bounds of FOU.

3.2. Interval Type 2 Fuzzy Logic Systems (IT2FLSs). The general type 2 fuzzy logic system is introduced as Figure 3. The output block of a type 2 fuzzy logic system consists of two blocks that are type-reduced and defuzzifier. The type-reduced block will map a type 2 fuzzy set to a type 1 fuzzy set, and the defuzzifier block will map a fuzzy to a crisp. The membership function of an interval type 2 fuzzy set is called FOU which is limited by two membership functions of a type 1 fuzzy set that are UMF and LMF (see Figure 4).

The combination of antecedents in a rule for IT2FLS is called firing strength process represented by the Figure 5.

In the IT2FLS, calculating process involves 5 steps to getting outputs: fuzzification, combining the antecedents (apply fuzzy operators or implication function), aggregation, and defuzzification.

Because each pattern has a membership interval as the upper $\overline{\mu}_{\widetilde{A}}(x)$ and the lower $\underline{\mu}_{\widetilde{A}}(x)$, each centroid of a cluster is represented by the interval between c_L and c_R. Now, we will represent an iterative algorithm to find c_L and c_R as follows.

Step 1. Calculate θ_i by the following equation:

$$\theta_i = \frac{1}{2}\left[\overline{\mu}(x_i) + \underline{\mu}(x_i)\right]. \quad (7)$$

Step 2. Calculate c' as follows:

$$c' = c(\theta_1, \theta_2, \ldots, \theta_N) = \frac{\sum_{i=1}^{N} x_i * \theta_i}{\sum_{i=1}^{N} \theta_i}. \quad (8)$$

Step 3. Find k such that $x_k \le c' \le x_{k+1}$.

Step 4. Calculate c'' by following equation: in case c'' is used for finding c_L

$$c'' = \frac{\sum_{i=1}^{k} x_i \overline{\mu}(x_i) + \sum_{i=k+1}^{N} x_i \underline{\mu}(x_i)}{\sum_{i=1}^{k} \overline{\mu}(x_i) + \sum_{i=k+1}^{N} \underline{\mu}(x_i)}. \quad (9)$$

In case c'' is used for finding c_R, then

$$c'' = \frac{\sum_{i=1}^{k} x_i \underline{\mu}(x_i) + \sum_{i=k+1}^{N} x_i \overline{\mu}(x_i)}{\sum_{i=1}^{k} \underline{\mu_A}(x_i) + \sum_{i=k+1}^{N} \overline{\mu}(x_i)}. \quad (10)$$

Step 5. If $c' = c''$ go to Step 6 else set $c' = c''$, then back to Step 3.

Step 6. Set $c_L = c'$ or $c_R = c'$.

Finally, compute the mean of centroid, y, as

$$y = \frac{c_R + c_L}{2}. \quad (11)$$

4. Speedup of IT2FLS Using GPU and CUDA

The first step in IT2FLS on the GPU is selection of memory types and sizes. This is a critical step, the choice of format and type dictate performance. Memory should be allocated such that sequential access (of read and write operations) is as possible as the algorithm will permit.

Let the number of inputs be N, the number of parameters that define a membership function be P, the number of rules be R and the discretization rate be S. Inputs are stored on the GPU as a one-dimensional array of size N (see Figure 6).

The consequences are a CPU two-dimensional array of size $R \times P$. They are used only on the CPU when calculating the discrete fuzzy set membership values.

The antecedents are a two-dimensional array on the GPU of size $R \times (N \times P)$.

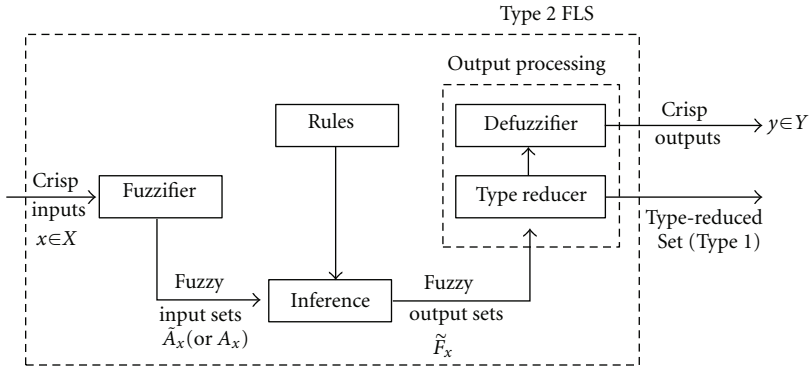

Figure 3: Diagram of type 2 fuzzy logic system [4].

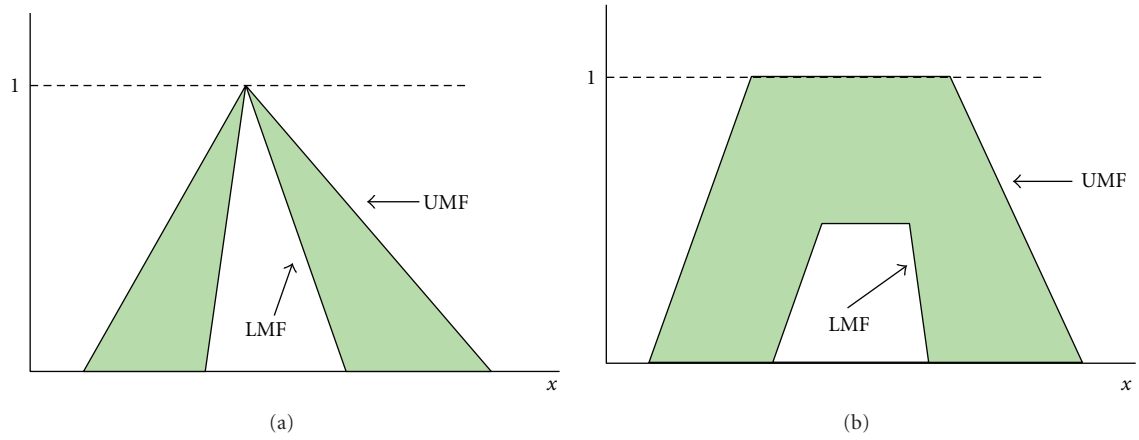

Figure 4: The membership function of an interval type 2 fuzzy set [1].

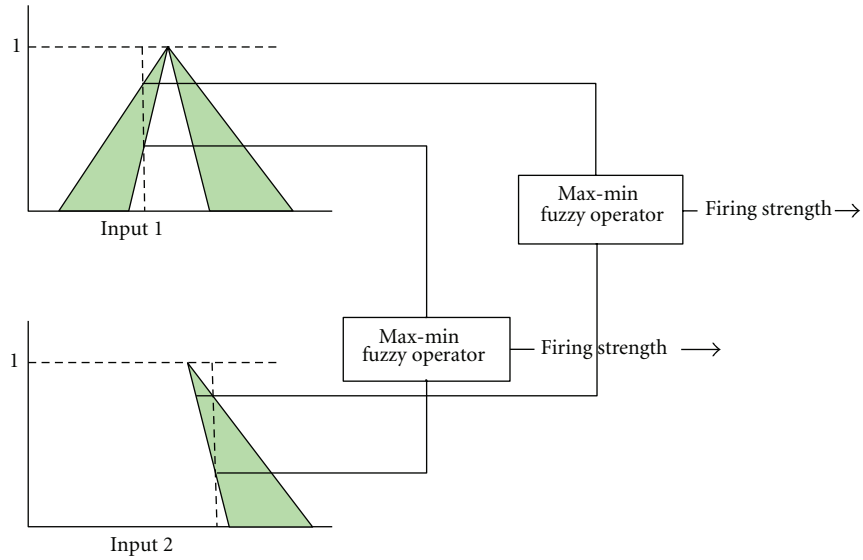

Figure 5: The combination of antecedents in a rule for IT2FLS [23].

	N			
1	X_1	X_2	$\cdots$	X_N

Figure 6: Input vector.

The fired antecedents are an R one-dimensional array on the GPU, which stores the result of combining the antecedents of each rule (see Figures 7, 8, and 9). The last memory layout is the discretized consequent, which is an $S \times R$ matrix created on the GPU.

The inputs and antecedents are of type texture memory because they do not change during the firing of a FLS, but could change between consecutive firings of a FLS and need to be updated. We proposed the GPU program flow diagram for a CUDA application computing a IT2FLS in Figure 10.

In IT2FLS, we have to calculate two values for two membership functions that are UMF and LMF. The first step is a kernel that fuzzifies the inputs and combines the antecedents. The next steps are implication and a process which is responsible for aggregating the rule outputs. The last GPU kernel is the defuzzification step.

The first kernel reads from the inputs and antecedents textures and stores its results in the fired antecedent's global memory section. All inputs are sampled for each rule, the rth rule samples the rth row in the antecedent's memory, membership values are calculated, and the minimum of the antecedents is computed and stored in the rth row of the fired antecedent's memory region. There are B blocks used by this kernel, partially because there is a limit in terms of the number of threads that can be created per block (current max is 512 threads). Also, one must consider the number of threads and the required amount of register and local memory needed by a kernel to avoid memory overflow. This information can be found per each GPU. We limited the number of threads per block to 128 (an empirical value found by trying different block and thread profiles for a system that has two inputs and trapezoidal membership functions). The general goal of a kernel should be to fetch a small number of data points, and it should have high arithmetic intensity. This is the reason why only a few memory fetches per thread are made, and the membership calculations and combination step is performed in a single kernel.

The next steps are implication and rule aggregation kernels. At first, one might imagine that using two kernels to calculate the implication results and rule aggregation would be desirable. However, the implication kernel, which simply calculates the minimum between the respective combined antecedent results and the discretized consequent, is inefficient. As stated above, the ratio of arithmetic operations to memory operations is important. We want more arithmetic intensity than memory access in a kernel. Attempt to minimize the number of global memory samples, we perform implication in the first step of reduction. Reduction, in this context, is the repeated application of an operation to a series of elements to produce a single scalar result. In the case of rule aggregation, this is the application of the maximum

	P_1			
	C_{11}	C_{12}	$\cdots$	C_{1P1}
	C_{21}	C_{22}	$\cdots$	C_{2P1}
R	$\cdots$	$\cdots$	$\cdots$	$\cdots$
	C_{R1}	C_{R2}	$\cdots$	C_{RP1}

Figure 7: Consequent.

	$M = N \times P$			
	A_{11}	A_{12}	$\cdots$	A_{1M}
	A_{21}	A_{22}	$\cdots$	A_{2M}
R	$\cdots$	$\cdots$	$\cdots$	$\cdots$
	A_{R1}	A_{R2}	$\cdots$	A_{RM}

Figure 8: Antecedent.

	S = sample rate			
	D_{11}	D_{12}	$\cdots$	D_{1S}
	D_{21}	D_{22}	$\cdots$	D_{2S}
R	$\cdots$	$\cdots$	$\cdots$	$\cdots$
	D_{R1}	D_{R2}	$\cdots$	D_{RS}

Figure 9: Discretized consequent.

operator over each discrete consequent sample point for each rule. The advantage of GPU reduction is that it takes advantage of the parallel processing units to perform a divide and conquer strategy. And the last step is defuzzification kernel. As described above, rule output aggregation and defuzzification reduction are used for IT2FLS on the GPU.

The output of rule output aggregation is two rows in the discretized consequent global memory array. The defuzzifier step is done by the Karnik-Mendel algorithms with two inputs that are rule_combine_UMF and rule_combine_LMF with two outputs y_l and y_r, respectively. The crisp output y is calculated by the formula $y = (y_l + y_r)/2$.

The steps for finding y_l and y_r on GPU (Notation: Rule_Combine_UMF (i) = $\overline{\mu_A}(x_i)$ and Rule_Combine_LMF (i) = $\underline{\mu_A}(x_i)$, N = sample rate) as follows

Step 1. Calculate θ_i on GPU by the following equation:

$$\theta_i = \frac{1}{2}\left[\overline{\mu_A}(x_i) + \underline{\mu_A}(x_i)\right]. \tag{12}$$

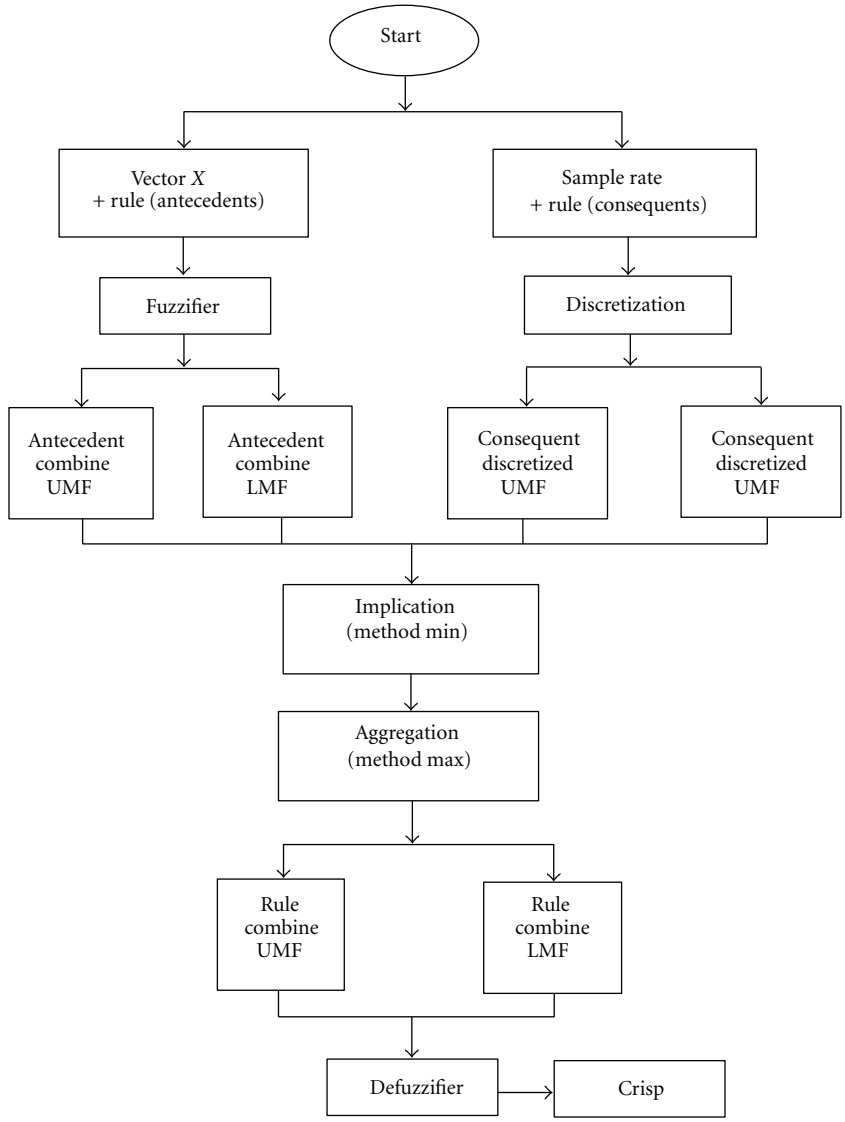

Figure 10: IT2FLS diagram for a CUDA application on the GPU.

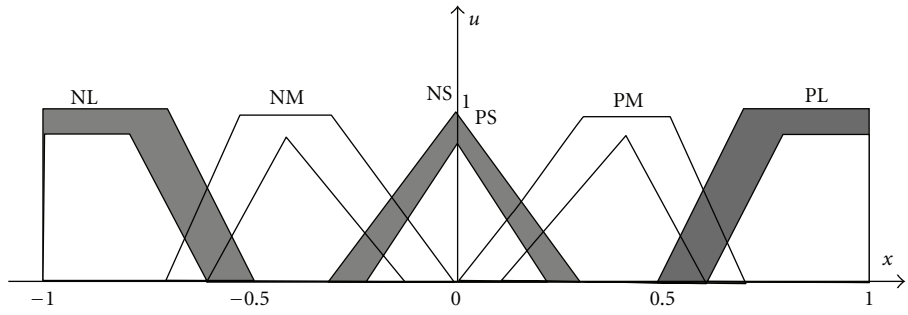

Figure 11: Membership grades of FDR.

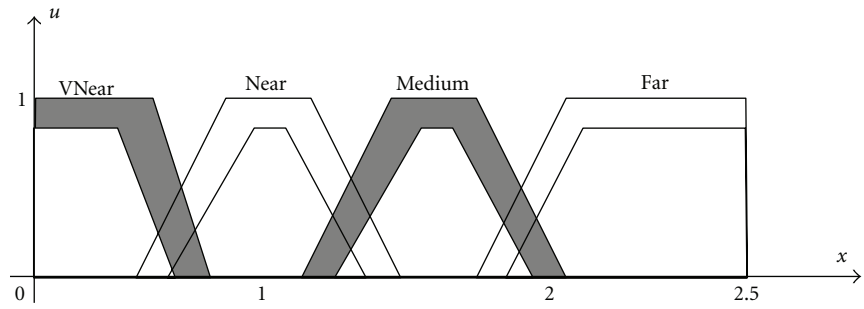

Figure 12: Membership grades of Range.

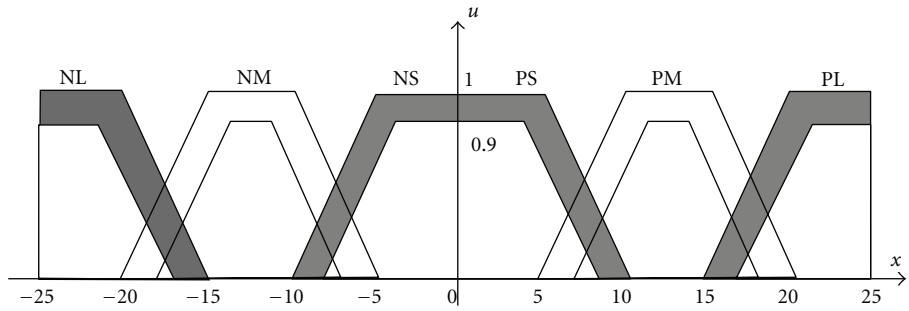

Figure 13: Membership grades of AoD.

Table 1: The rule base of collision avoidance behavior.

FDR	Range	AoD	FDR	Range	AoD
NS	VN	PL	PS	VN	NL
NS	N	PL	PS	N	NL
NS	M	PM	PS	M	NM
NS	F	PS	PS	F	NS
NM	VN	PM	PM	VN	NM
NM	N	PM	PM	N	NM
NM	M	PM	PM	M	NM
NM	F	PS	PM	F	NS
NL	VN	PM	PL	VN	NM
NL	N	PM	PL	N	NM
NL	M	PS	PL	M	NS
NL	F	PS	PL	F	NS

Step 2. Calculate c' on GPU as follows:

$$c' = c(\theta_1, \theta_2, \ldots, \theta_N) = \frac{\sum_{i=1}^{N} x_i * \theta_i}{\sum_{i=1}^{N} \theta_i}. \tag{13}$$

Next, copy c' to host memory.

Step 3. Find k such that $x_k \le c' \le x_{k+1}$ (calculated on CPU).

Step 4. Calculate c'' on GPU by following equation. In case c'' is used for finding y_l

$$c'' = \frac{\sum_{i=1}^{k} x_i \overline{\mu_A}(x_i) + \sum_{i=k+1}^{N} x_i \underline{\mu_A}(x_i)}{\sum_{i=1}^{k} \overline{\mu_A}(x_i) + \sum_{i=k+1}^{N} \underline{\mu_A}(x_i)}. \tag{14}$$

In case c'' is used for finding y_r, consider

$$c'' = \frac{\sum_{i=1}^{k} x_i \underline{\mu_A}(x_i) + \sum_{i=k+1}^{N} x_i \overline{\mu_A}(x_i)}{\sum_{i=1}^{k} \underline{\mu_A}(x_i) + \sum_{i=k+1}^{N} \overline{\mu_A}(x_i)}. \tag{15}$$

Next, copy c'' to host memory.

Step 5. If $c' = c''$ go to Step 6 else set $c' = c''$, then back to Step 3 (calculated on CPU).

Step 6. Set $y_l = c'$ or $y_r = c'$.

5. Experiments

5.1. Problems. We implement IT2FLS with collision avoidance behavior of robot navigation. The fuzzy logic systems have two inputs: the extended fuzzy directional relation (FDR) [24] and range to obstacle; the output is angle of deviation (AoD). The fuzzy rule has the form as follows.

IF FDR is $\widetilde{A}_i$ AND Range is $\widetilde{B}_i$ THEN AoD is $\widetilde{C}_i$, where $\widetilde{A}_i$, $\widetilde{B}_i$, and $\widetilde{C}_i$ are type 2 fuzzy sets of antecedent and consequent, respectively.

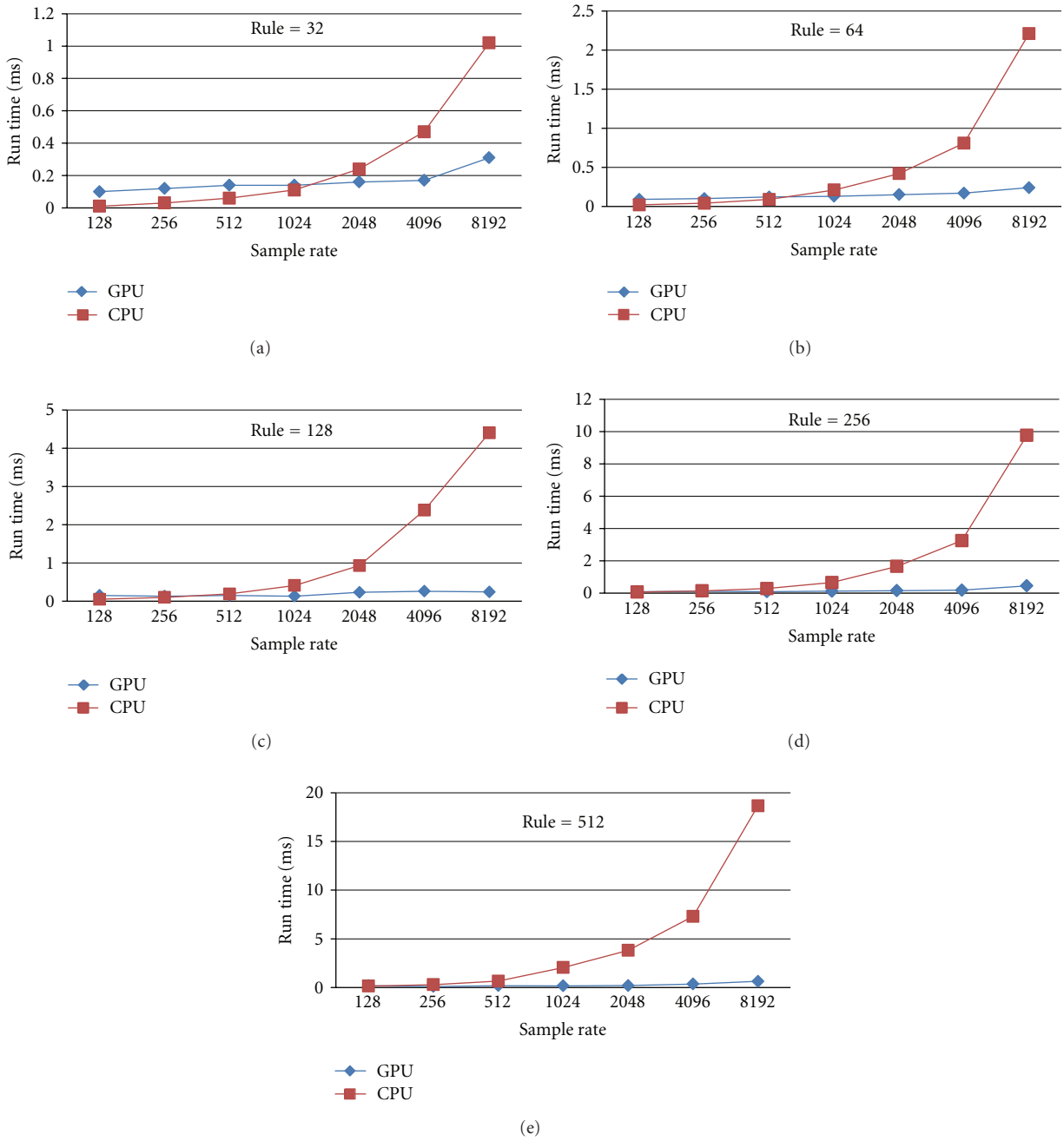

Figure 14: Run-time graph of the problem implementation.

The fuzzy directional relation has six linguistic values (NLarge, NMedium, NSmall, PSmall, PMedium, and PLarge). The range from robot to obstacle is divided into four subsets: VNear, Near, Medium, and Far. The output of fuzzy if-then is a linguistic variable representing for angle of deviation and has six linguistic variables the same the fuzzy directional relation with the different membership functions. Linguistic values are interval type 2 fuzzy subsets that membership functions are described in Figures 11, 12, and 13. The problem is built with 24 rules given by the following Table 1.

5.2. Experiments. The performance of the GPU implementation of a IT2FLS was compared to a CPU implementation. The problem is written in C/C++ console format and be installed on the Microsoft Visual Studio 2008, and it was performed on computers with the operating system windows 7 32 bit and nVIDIA CUDA support with specifications.

CPU was the Core i3-2310 M 2.1 GHz, the system had 2 GB of system RAM (DDR3).

GPU was an nVIDIA Gerforce GT 540 M graphics card with 96 CUDA Core, 1 GB of texture memory, and PCI Express X16.

Table 2: CPU/GPU performance ratio.

R	S						
	128	256	512	1024	2048	4096	8192
32	0.1	0.24	0.41	0.76	1.47	2.64	3.22
64	0.21	0.39	0.73	1.58	2.7	4.69	8.99
128	0.32	0.76	1.25	3.07	3.95	9.01	18.26
256	0.72	1.19	2.95	5.35	10.7	17.56	21.3
512	0.77	2.43	3.32	12.57	18.43	20.51	29.3

The number of inputs was fixed to 2, the number of rules was varied between 32, 64, 128, 256, and 512, and sample rate was varied between 256, 512, 1024, 2048, 4096, and 8192.

We take the ratio of CPU versus GPU performance. A value below 1 indicated that the CPU is performing best, and value above 1 indicates the GPU is performing best. The CPU/GPU performance ratios for the IT2FLS are given in Table 2 and run-time graph of the problem implementation was shown in Figure 14.

6. Conclusion

As demonstrated in this paper, the implementation of interval type 2 FLS on a GPU without the use of a graphics API which can be used by any researcher with knowledge of C/C++. We have demonstrated that the CPU outperforms the GPU for small systems. As the number of rules and sample rate grow, the GPU outperforms the CPU. There is a switch point in the performance ratio matrices (Table 2) that indicates when the GPU is more efficient than the CPU. In the case that sample rate is 8192 and rule is 512, the GPU runs approximately 30 times faster on the computer.

Future work will look at to extend interval type 2 FLS to the generalised type 2 FLS and applying to various applications.

Acknowledgment

This paper is sponsored by Intelligent Robot Project at LQDTU and the Research Fund RFit@LQDTU, Faculty of Information Technology, Le Quy Don University.

References

[1] N. N. Karnik, J. M. Mendel, and Q. Liang, "Type-2 fuzzy logic systems," *IEEE Transactions on Fuzzy Systems*, vol. 7, no. 6, pp. 643–658, 1999.

[2] N. N. Karnik and J. M. Mendel, "Centroid of a type-2 fuzzy set," *Information Sciences*, vol. 132, no. 1–4, pp. 195–220, 2001.

[3] Q. Liang and J. M. Mendel, "Interval type-2 fuzzy logic systems: theory and design," *IEEE Transactions on Fuzzy Systems*, vol. 8, no. 5, pp. 535–550, 2000.

[4] J. M. Mendel, R. I. John, and F. Liu, "Interval type-2 fuzzy logic systems made simple," *IEEE Transactions on Fuzzy Systems*, vol. 14, no. 6, pp. 808–821, 2006.

[5] F. Liu, "An efficient centroid type-reduction strategy for general type-2 fuzzy logic system," *Information Sciences*, vol. 178, no. 9, pp. 2224–2236, 2008.

[6] L. T. Ngo, L. T. Pham, P. H. Nguyen, and K. Hirota, "On approximate representation of type-2 fuzzy sets using triangulated irregular network," in *Foundations of Fuzzy Logic and Soft Computing*, vol. 4529 of *Lecture Notes in Computer Science*, pp. 584–593, Springer, 2007.

[7] L. T. Ngo, L. T. Pham, P. H. Nguyen, and K. Hirota, "Refinement geometric algorithms for type-2 fuzzy set operations," in *Proceedings of the IEEE International Conference on Fuzzy Systems*, pp. 866–871, August 2009.

[8] L. T. Ngo, "Refinement CTIN for general type-2 Fuzzy logic systems," in *Proceedings of the IEEE International Conference on Fuzzy Systems (IEEE-FUZZ '11)*, pp. 1225–1232, Hanoi, Vietnam, 2011.

[9] J. T. Starczewski, "Efficient triangular type-2 fuzzy logic systems," *International Journal of Approximate Reasoning*, vol. 50, no. 5, pp. 799–811, 2009.

[10] H. A. Hagras, "A hierarchical type-2 fuzzy logic control architecture for autonomous mobile robots," *IEEE Transactions on Fuzzy Systems*, vol. 12, no. 4, pp. 524–539, 2004.

[11] O. Castillo and P. Melin, "A review on the design and optimization of interval type-2 fuzzy controllers," *Applied Soft Computing*, vol. 12, no. 4, pp. 1267–1278, 2012.

[12] R. Sepúlveda, O. Montiel, O. Castillo, and P. Melin, "Modelling and simulation of the defuzzification stage of a type-2 fuzzy controller using VHDL code," *Control and Intelligent Systems*, vol. 39, no. 1, pp. 33–40, 2011.

[13] Y. Maldonado, O. Castillo, and P. Melin, "Optimization of membership functions for an incremental fuzzy PD control based on genetic algorithms," *Soft Computing for Intelligent Control and Mobile Robotics*, vol. 318, pp. 195–211, 2011.

[14] R. Sepúlveda, O. Montiel-Ross, O. Castillo, and P. Melin, "Embedding a KM type reducer for high speed fuzzy controller into an FPGA," *Applied Soft Computing*, vol. 12, no. 3, pp. 988–998, 2012.

[15] D. T. Anderson, R. H. Luke, and J. M. Keller, "Speedup of fuzzy clustering through stream processing on graphics processing units," *IEEE Transactions on Fuzzy Systems*, vol. 16, no. 4, pp. 1101–1106, 2008.

[16] D. Anderson, R. H. Luke, and J. M. Keller, "Incorporation of non-euclidean distance metrics into fuzzy clustering on graphics processing units," in *Proceedings of the Inertial Fusion Sciences and Applications (IFSA '07)*, vol. 41, pp. 128–139, 2007.

[17] K. Sejun and C. Donald, "A GPU based parallel hierarchical fuzzy ART clustering," in *Proceedings of the International Joint Conference on Neural Networks*, vol. 1, pp. 2778–2782, Rolla, Mo, USA, 2011.

[18] I. Chiosa and A. Kolb, "GPU-based multilevel clustering," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 2, pp. 132–145, 2011.

[19] F. Chia, C. Teng, and Y. Wei, "Speedup of implementing fuzzy neural networks with high-dimensional inputs through parallel processing on graphic processing units," *IEEE Transactions on Fuzzy Systems*, vol. 19, no. 4, pp. 717–728.

[20] N. Harvey, R. Luke, J. M. Keller, and D. Anderson, "Speedup of fuzzy logic through stream processing on graphics processing units," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '08)*, pp. 3809–3815, June 2008.

[21] D. Anderson, "Parallelisation of fuzzy inference on a graphics processor unit using the compute unified device architecture," in *Proceedings of the UK Workshop on Computational Intelligence (UKCI '08)*, pp. 1–6, 2008.

[22] M. Harris, 18 March 2008, Optimizing Parallel Reduction in CUDA, NVIDIA Whitepaper, http://www.nvidia.com/object/cuda/sample/dat/a%20-%20parallel.html.

[23] R. Imam and B. Kharisma, "Design of interval type-2 fuzzy logic based power system stabilizer," *International Journal of Electrical and Electronics Engineering*, vol. 3, no. 10, pp. 593–600, 2009.

[24] L. T. Ngo, L. T. Pham, and P. H. Nguyen, "Extending fuzzy directional relationship and applying for mobile robot collision avoidance behavior," *International Journal of Advanced Computational Intelligence & Intelligent Informatics*, vol. 10, no. 4, pp. 444–450, 2006.