Field Programmable Gate Array (FPGA) Based Collision Avoidance Using Acceleration Velocity Obstacles

Roopak Dubey*, Neeraj Pradhan*, K. Madhava Krishna and Shubhajit Roy Chowdhury

Abstract—**This paper presents a Field Programmable Gate Array (FPGA) based implementation of Acceleration Velocity Obstacle based Collision Avoidance for an omni-directional robot with acceleration constraint. Specifically a parallel architecture for collision avoidance is proposed that portrays the advantages of FPGA implementation over the sequential implementation for same processor or clock speed. FPGA based robotics is seen to gain popularity due to low cost, portability, seamless interface to hardware and most importantly due to inherent parallelism enshrined in various robotic algorithms. FPGA realization of the algorithm in a simulation test bed vindicates its efficacy and comparison with sequential implementation is also highlighted. The paper proposes three different architectures for the implementation of the proposed algorithm viz. sequential architecture; a resource constrained pipelined architecture and a hybrid pipeline parallel architecture. The performances of those three architectures have been evaluated.**

I. INTRODUCTION

Obstacle avoidance is one of the most basic problems in mobile robotics. In those systems where there are no other moving objects other than the robot itself static obstacle avoidance is sufficient. There are numerous algorithms for static obstacle avoidance [1][11][12]. But there are environments where there are moving objects as well as other robots in the vicinity of the robot under consideration. In those cases static obstacle avoidance is of not much avail. In most of the multi-robot and real world problems we need to deal with dynamic objects. An algorithm is needed which takes the dynamic nature of objects and other robots under consideration. Such an algorithm is termed as Dynamic Obstacle Avoidance algorithm [10]. There are again numerous approaches for Dynamic Obstacle Avoidance. Velocity Obstacle (VO) approach is one of the common approaches used for collision avoidance in dynamic environment [2]. When the acceleration constraints are considered in the VO it is called Acceleration Velocity Obstacle (AVO) [3]. When robots share the responsibility of Avoidance among themselves equally then it is termed as Reciprocal Collision Avoidance (RCA) [4].

Field Programmable Gate Array (FPGA) is gaining popularity in robotics recently and it can be seen through FPGA implementations of image processing descriptors [5][6]. The credit for this popularity goes to the unique combination of qualities like small size, ability to reconfigure both offline and online, low power dissipation, low cost and high speed. Small sized robots with constrained resources are the need of the day and hence algorithms are needed which can run on systems with lesser memory and size footprint. A base station PC has advantages of accuracy and speed but is limited by the wireless range to the robot and hence restricts its operating area. A laptop cannot be mounted on very small robots and can be expensive too. A cell phone processor which can do such amount of processing is expensive and increases the cost of robot also a full-fledged operating system is to be installed to control it which in turn increases memory requirement. A micro-controller does not provide for accuracy and speed, whereas in principle an FPGA has advantages of all. Moreover all these processing units have *Complex Instruction Set Computing* (CISC) or *Reduced Instruction Set Computing* (RISC) architecture and there is an extensive instruction set. The operations are based on Fetch-Decode-Execute cycle and hence largely sequential in nature while on FPGA fully parallel architectures can be designed and executed. All the processing units mentioned above follow *Arithmetic Logic Unit* (ALU) based architecture for arithmetic and logical operations which involve a lot of instructions and hence clock cycles [7]. FPGA offers to develop gate level implementations of arithmetic circuits and hence a significant improvement in operating speed is inevitable. We show that with FPGA as the processing unit, the execution time of the algorithm, the size and cost of the robot can be reduced.

FPGA is largely used for application specific design. FPGA contains an array of Configurable Logic Blocks (CLBs), Look-up Tables (LUTs) and programmable interconnects between them. By programming these inter-connects, same CLBs can be used for several different functions. Power dissipation is reduced because out of all available paths the shortest path is chosen for the implementation.

The paper focuses on the novel hardware implementation of the well known Reciprocal Collision Avoidance with Acceleration Velocity Obstacles.This hardware implementation reduces the power dissipation and cost of implementation of the robot. Parallelism has been incorporated into the system model by making multiple copies of same elementary level modules and using them in parallel.

We also show the comparison of sequential and pipelined design of same algorithm with the hybrid design in terms of clock cycles required to bring out vividly the advantages of FPGA over other processing devices which generally follows sequential approach while processing.

*Equal Contribution

II. Velocity obstacle based Collision Avoidance

Collision avoidance is one of the fundamental problems inmobile robotics. Numerous algorithms exist which avoid collision in various types of environment. Velocity Obstacle approach is one of the algorithms which is very efficient in the dynamic environments with multiple robots and moving objects [2]. In this algorithm the current velocities and position of various objects in the space is used along with constraints of the robot to find the set of velocities which can avoid the obstacles as well as satisfy the constraints [2][3]. The set of velocities other than the aforementioned velocities is known as the Velocity Obstacle [2]. This set of velocities forms a collision cone [2] [3] [4]. Figure 1 shows a collision cone ($C_{1/2}$) when there is one robot and one dynamic obstacle. Any velocity outside this cone is the velocity which robot can take to avoid obstacles successfully [2] [3] [4].

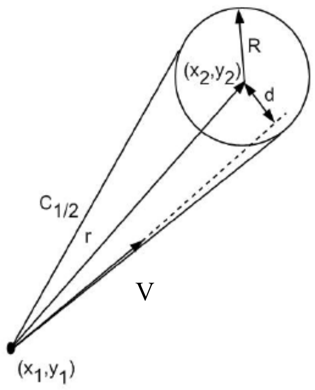

Figure 1: Collision Cone

Equation (1) [8] is the governing equation of the collision cone. The current work aims to find the velocities which satisfy these equations. In these equations, robots other than one which is controlled are considered as passive. All the velocities satisfying this equation are the avoidance velocities. Finally the velocities and position of the robot using the velocities from the set is updated.

$$d^2 = |\vec{r}|^2 - \frac{(\vec{r}.\vec{V})^2}{|\vec{V}|^2} \geq R^2 \tag{1}$$

Here,$\vec{r}$ is the relative position and $\vec{V}$ is the relative velocity of the robot 1 w.r.t robot 2 respectively.

$$\vec{v}_{n+1} = \vec{v}_n + \vec{a}t \tag{2a}$$

$$\vec{r}_{n+1} = \vec{r}_n + \vec{v}_n t + \frac{1}{2}\vec{a}_n t^2 \tag{2b}$$

Where,

$$\vec{r} = x\hat{\imath} + y\hat{\jmath} \tag{2c}$$

$$\vec{v} = v_x\hat{\imath} + v_y\hat{\jmath} \tag{2d}$$

$$\vec{a} = a_x\hat{\imath} + a_y\hat{\jmath} \tag{2e}$$

All velocities of the set are not achievable by robot due to acceleration constraints. So there is a need to take care of these constraints too. A velocity is chosen which is achievable by robot. So this reduces the set to a smaller set [2]. Equations at (2a) and (2b) are the update equations. An implementation of this algorithm has been developed on FPGA.

III. FPGA Architectures For Collision Avoidance

The problem of collision avoidance using Velocity Obstacle involves finding solution for equation (1). Let there be *n* robots which are similar with similar constraints on the map viz. Robot_1, Robot_2… Robot_*n*. Let Robot_1 be the robot for which collision avoidance is to be done. Then for each robot from Robot_2 to Robot_*n* an inequality like one shown in equation (1) can be obtained. So in total there are *n-1* inequalities which need to be solved.

There are two approaches to solve this problem. In one,linear programming can be used to solve the system of *n-1*inequalities with acceleration constraints and find the velocity of Robot_1 which falls out of collision cone. In the other, a velocity is sampled which is achievable under the acceleration constraints and verify whether it satisfies all *n-1* inequalities and the velocity which satisfies all is the chosen one.

The latter method has been used in this paper. This is because solving inequalities is more expensive in terms of resource and clock utilization when done on FPGA than the exhaustive method [9].

In this architecture a velocity is sampled out of achievable velocities. Robot_1 has access to the velocities and positions of other *n-1*robots. Using this data it is verified whether these velocities and positions satisfy the aforementioned *n-1* inequalities or not.

The algorithm consists of two main parts. In first part of algorithm, whether the robot reached the target or not, is checked. This is implemented by measuring the Euclidean distance between robot center and the target. In the second part a velocity is selected which satisfies system of inequalities under acceleration constraints.

In the second part, where inequalities are to be satisfied, first the positions are updated for a small time period using the velocities and then inequalities are verified at the updated position. If any one of the inequalities fails then another achievable velocity is sampled and tested.

Here the sampling of velocity is done using “to-goal” strategy [2]. In this strategy we always start looking first at the velocity for which velocity vector points towards the target. If that velocity is inside collision cone then we rotate our velocity vector by small values towards both clockwise and anti-clockwise.

In any CISC or RISC based processor all the operations are sequential and hence this verifying process takes the robots one by one and their positions are first updated using the update equations in (2a) and (2b). The Robot_1, the one for which the algorithm is implemented, gets its velocity verified in all *n-1* inequalities one by one.

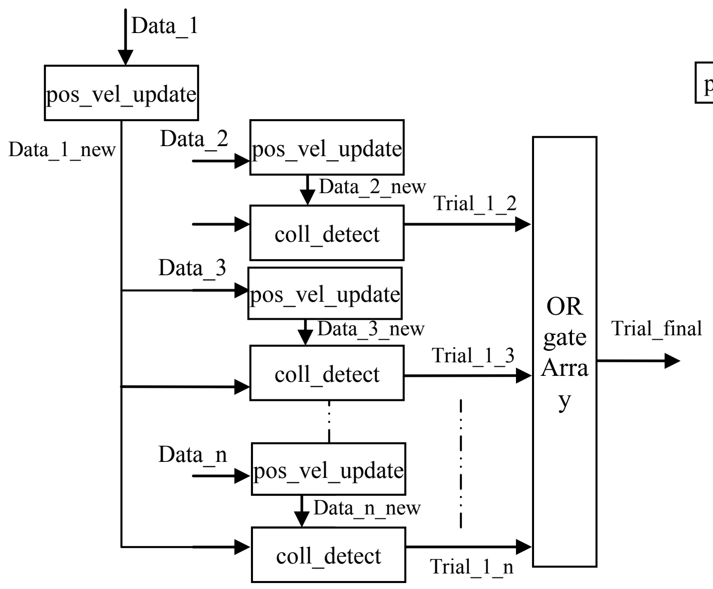

Figure 2: Sequential architecture forthe algorithm

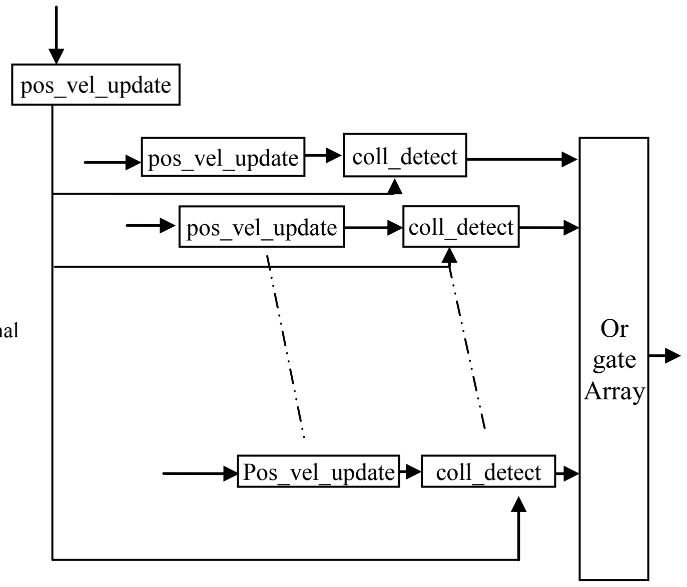

Figure 3: Resource constrained pipelined architecture for the algorithm(inputs outputs and intermediate values are same as in figure 2)

Three architectural design styles have been proposed for the implementation of the algorithm and the performances of these architectures have also been compared. Figure 2 shows the architecture for the sequential implementation of the algorithm.

In sequential architecture update of Robot_1 is done first. Then in next stage update of Robot_2 is done and data from these two updates are checked for collision using the inequality at (1) and this process is repeated for all the robots. So clock cycles are added up for each block. *Pos_vel_update* module takes 7 clock cycles to provide the output and *coll_detect* module takes 10 cycles to provide the output. Total clock cycles required for sequential operation is 7n + 10(n-1) = 17n -10, where *n* is number of robots. This implementation is used when resources are really meager and it is difficult to design even two components on same chip. Data_1, Data_2 etc. denotes the current position and velocity of Robot_1, Robot_2 and so on. Data_1_new, Data_2_new etc. are the updated position and velocity of the robots if we assume sampled velocity is chosen. Or_gate_array will give the output as '1' if any of the inequality fails.Trial is the signal which is asserted when robots are on collision course.

Figure 3 shows a resource constrained pipelined architecture for the implementation of algorithm. In this architecture the update of next robot could be performed while the inequality for previous robot is being verified. In this implementation, for *n*-robot problem,the clock cycles which are added are the only clock cycles which are required for *coll_detect* module to provide output. The cycles for the initial two updates are also added. The resource utilization is more than sequential implementation as two components that can work in parallel are to be implemented. Total clock cycles required is 10(n-1) + 14= 10n + 4. This number is much smaller when compared with sequential architecture, when number of robots increase.

When there is no limit on number of resources to be used hybrid architecture as shown in Figure 4 can be designed. In this architectureupdate for all the robots is made parallelly in one stage and in next stage the inequalities are verified for all parallelly. This is the fastest of all the implementations and takes only 7 + 10 = 17 clock cycles irrespective of number of robots. This is the unique advantage which is offered by FPGA. It is possible on FPGA to design parallel architectures at certain stages. Advantage of partial and full re-configurability is offered by FPGA and using this ability parallel architecture for large number of robots can be implemented.

In the presented work a hybrid design is used which saves a large number of clock cycles as number of robots increases. The exact numbers are shown in section VI: Results and Discussions.

If the FPGA has less number of resources, pipelined architecture is needed which is always better than the conventional sequential implementation in processors.

Table I
Comparative study of resource utilization of 2-robot problem

Architecture	CLBs	LUTs	Power Dissipation (mW)
Sequential	52%	32%	96
RC Pipelined	63%	41%	104
Hybrid	79%	70%	128

The comparative study of resource utilization and power dissipation in all these three implementations has been done on the basis of synthesis report. We show one such result for two robots in Table I. A more detailed comparison is tabulated in section VI.

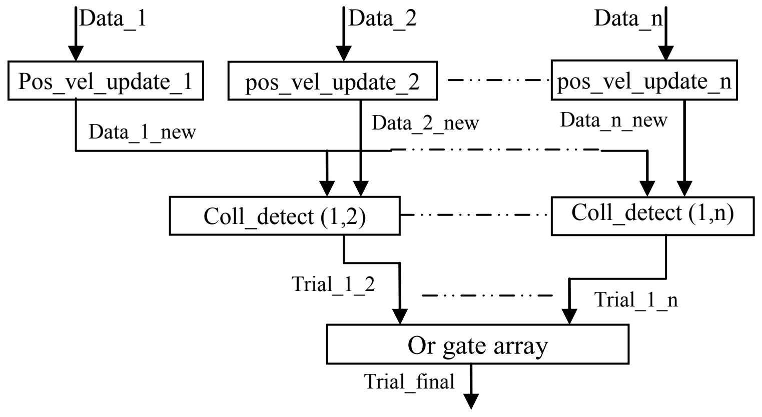

Figure 4: Hybrid architecture for the algorithm

IV. DETAILED ARCHITECTURAL DESCRIPTION

To ensure the parallelization of the operations, we used a hierarchical model for implementing the algorithm. The equations (1), (2a) and (2b) are vector equations. While implementation on FPGA all these equations are resolved in x and y components. The computations for magnitude and direction are done separately and then results are used. The following sub-sections provide a brief description of the modules. Architectural details are illustrated only for the key modules viz. *pos_vel_update* and *coll_detect* due to brevity of space.

A. Update Module (pos_vel_update)

The update module has been used for all types of position and velocity updates. It is used to find the point where robot will reach and what will be the velocity of robot at that point. The outputs of this module will go to collision detect module. The equations involved in implementation of this module are the resolved form of those given in equations (2a) and (2b) above. We calculated the new position and velocities in horizontal and vertical components separately. The hardware implementation is shown in Figure 5.

B. Collision Detection Module (coll_detect)

This module evaluates the condition given in equation-(1) using the resolved *x* and *y* components and determines whether robot is on collision course or not. Its output is a signal which tells the velocity update block to stop if condition is satisfied. The velocity for which condition is satisfied will be given to the Proportional Integro-Derivative (PID) control module to control the robot accordingly. The architecture is shown in Figure 6.

Equation-(1) is slightly altered for proper implementation on the hardware. The altered equation is given below.

$$|\vec{r}|^2|\vec{V}|^2 - (\vec{r}.\vec{V})^2 \geq R^2|\vec{V}|^2 \qquad (4)$$

The output trial = ‘1’ denotes there is a collision and trial = ‘0’ denotes that path is safe.

In Figure 6, r_sq denotes $|\vec{r}|^2$; vab_sq stands for $|\vec{V}|^2$ and dotp_sq stands for $(\vec{r}.\vec{V})^2$. *(X1, Y1)* and *(X2, Y2)* are positions of Robot-1 and Robot-2 respectively. *(VX1, VY1)* and *(VX2, VY2)* are components of velocities of Robot 1 and Robot 2. *R* is combined radius of robots. *R = R1 + R2* where *R1* and *R2* are radii of the robots.

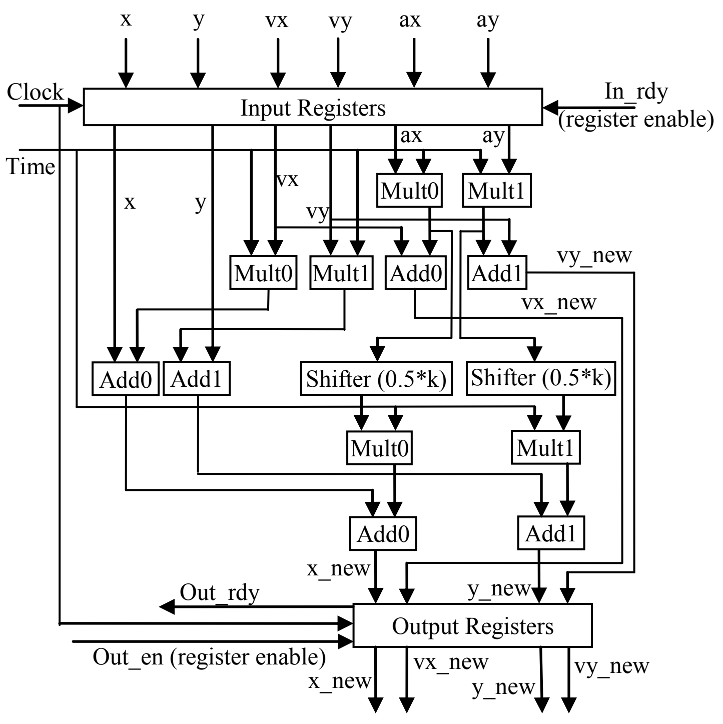

Figure 5: Update Module architecture

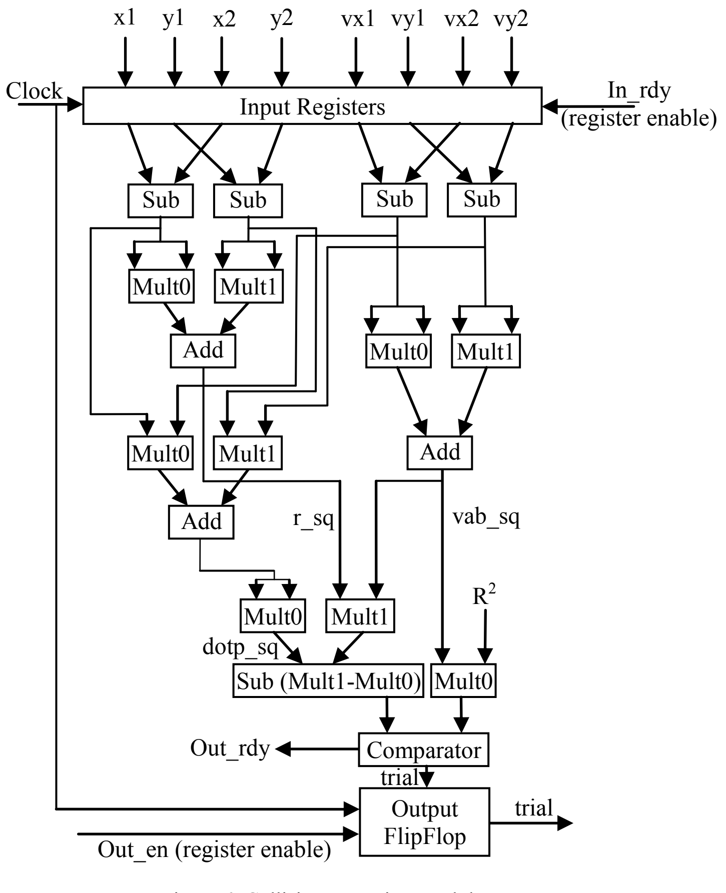

Figure 6: Collision Detection Module

C. Other Modules

Besides update module and collision detection module there are two more modules viz. Velocity Selector Module and Acceleration Check Module.

Velocity selector module selects one velocity out of an array of velocities using the "to goal" strategy mentioned earlier. The required direction is determined using target location and present location of robot. Components of velocities are selected to get the resultant in that direction.

Once the components of velocity is obtained acceleration check module checks whether the selected velocity is achievable within acceleration constraints or not? If not, then velocity selector module is signaled to select some other velocity: if yes, then the selected velocity is sent to update module.

V. FPGA Implementation

Each of the three implementations viz. sequential, resource constrained pipelined and hybrid in section III can be implemented on FPGA. If FPGA has fewer resources, then pipelined architecture should be used instead of parallel architecture. If a high-end FPGA with large number of resources is used or partial reconfiguration can be used then a wholly parallel design is possible which uses a small number of clock cycles for the algorithm which is 17 in this case.

Each Robot has a FPGA mounted on it and same design is implemented on each FPGA. So all robots work at the same time and hence share the responsibility of avoiding collision. All other parts of the overall design require same number of clock cycles and same amount of resources irrespective of number of robots. The exact values are given in result section.

The Nexys 2 board which is used to implement the design is shown in Figure 7. The actual robot is shown in Figure 8. The actual robot has 3 levels. First level is for wheels, second level is FPGA and third level is the interfacing circuit which provides interface of FPGA to the motors. The unique arrangement of wheels facilitates the omni-drive motion of the robot.

To reduce number of resources, serial operation is used at certain places. The design of system comprises both serial and parallel operations to maintain an optimum level in utilization of resources as well as clock cycles required. The exact values involved are presented in results section.

This hardware architecture is easily scalable to more number of robots and dimensions. As a very low end FPGA is used for experiment the results are obtained up to 8 robots in two dimensional maps.

VI. Results And Discussions

The algorithm is implemented using Xilinx Spartan 3E-500 FG320 FPGA using the Nexys-2 board that has over 10476 logic cells, 20 dedicated multipliers, 20 blocks of BRAM and 50 MHz clock speed. The test is run on various sources and target scenarios with 2, 4 and 8 robots. For the 8-robot problem, resource constrained pipelined architecture is used to obtain the results since hybrid design is not realizable on the aforementioned device. For more robots we need an FPGA with more resources on it.

Experimental Setup: - The design is dumped on the FPGA board using Digilent Adept Programmer interface and executed. The co-ordinates of all the robots are being transmitted through an on-board serial port and getrendered through a standard graphic package. The entire algorithm runs on FPGA, while the processor is used only for rendering.

Figure-9, 10shows the collision avoidance in 4-robot and 8-robot case respectively. A 2-dimensional map of size 300 x 300, without any static object is shown in figures. Four snapshots of the path followed by robots from source to destination are presented in figures of all the aforementioned three scenarios.

TABLE II
COMPARISON OF CLOCK CYCLES REQUIRED (*f* = 50MHz)

	2-robots (n=2)	4-robots (n=4)	8-robots (n=8)
Sequential (17n – 10)	24	58	126
RC Pipelined (10n + 4)	24	44	84
Hybrid	17	17	17

TABLE III
Power Dissipation (in mW) in various scenarios

Architecture	2 robots	4 robots	8 robots
Sequential	96	112	136
RC pipelined	104	127	153
Hybrid	128	161	-

Note: As mentioned earlier, the hybrid architecture for 8-robot problem is not feasible on the FPGA being used. Power Dissipation information is unknown for the same.

A lot of clock cycles can be saved when we go for hybrid design rather than sequential or pipelined design. Table II shows the number of clock cycles required to complete the crucial part of collision detection in all three types of implementation. The values shown in the table IIare the number of clock cycles required per update of position and velocity.

Figure 7: Xilinx Nexys 2 Development Board with Spartan 3E FPGA

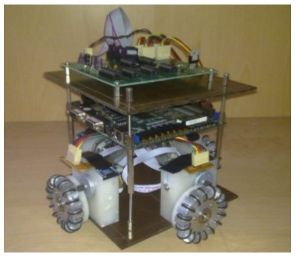
Figure 8: The robot with FPGA at second level

It can be clearly seen from the table II that a significant amount of speed up is obtained when we go for hybrid architecture. A full run of algorithm requires nearly 500 of such updates and hence there is a large gain in terms of speed of execution of algorithm when hybrid architecture is used. Hybrid architecture has high resource utilization but this issue can be addressed by the ability of FPGA to dynamically reconfigure. At a particular instance we can have only those resources which are required for processing and FPGA can be reconfigured before the next step. In this way highly parallel architectures can be designed for FPGA.

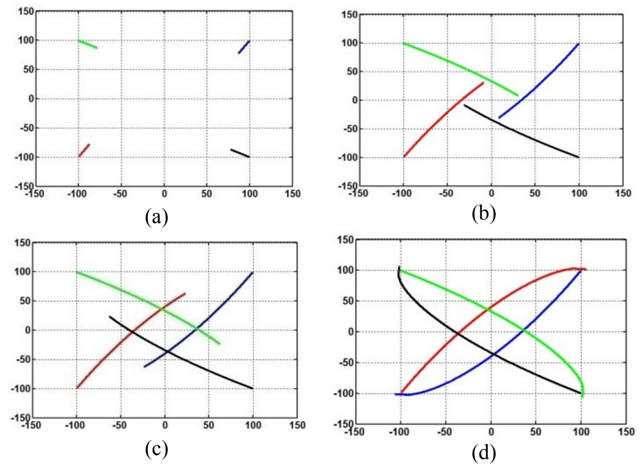

Figure 9: Obstacle avoidance in 4-robot problem

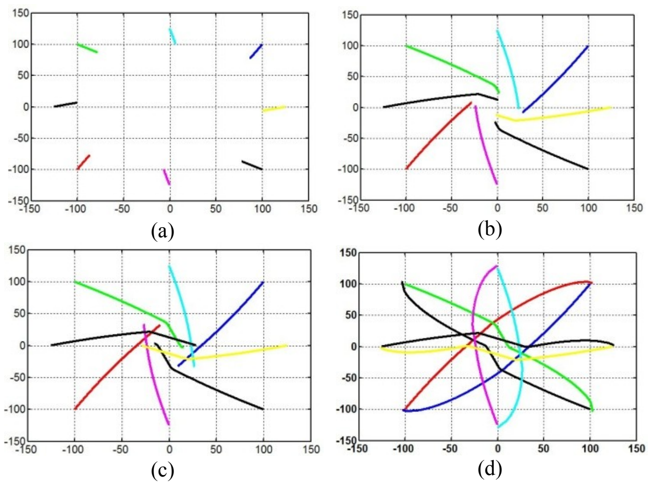

Figure 10: Obstacle avoidance in 8-robot problem

Simulation of power dissipation has also been done using XPower Analyzer tool of Xilinx ISE 13.4. The results are compiled in table III. As evident from the table III, sequential implementation has lowest power dissipation and hybrid implementation has highest. This is because in hybrid architecture, more cells and interconnects are active at any instance of time In RC pipelined architecture the number of active interconnects are more than that in sequential architecture and hence power dissipation is more.

The power dissipation in FPGA is in the order of milli-watt which is nominal when compared to power dissipation in a typical CISC or RISC processor for which power dissipation is in the order of Watt [13].

A significant reduction in power dissipation can be obtained when a scenario with large number of robots is implemented on FPGA.

VII. Conclusion and Future Scope

This paper presented two new architectures implemented on Spartan-3, Nexys-2 FPGA platform for multi robot collision avoidance. The architectures are particularly helpful when the number of robots in an environment increases and sequential implementation of avoidance routines tend to become unwieldy. The RC-pipelined architecture exploits the pipeline implementation feasible on an FPGA, shows tangible reduction in computation time without extensive consumption of resources. On the other hand the hybrid architecture provides for a full fledged parallel constant time implementation even as the number of robots increases. The hybrid architecture is the most sought after when the FPGA resources are not constrained. FPGA implementations on a Nexys-2, Spartan-3 platform confirm the efficacy of the proposed architectures. Comparative tabulations between sequential, pipelined and hybrid architectures vividly portray the benefits of a FPGA implementation. The efforts are on to develop such FPGA implementations of robotic algorithms invoking the specific advantages provided by FPGA platforms is a way forward towards low cost FPGA based robotic systems.

References

[1] Lozano-Perez, T.; , "Spatial Planning: A Configuration Space Approach," *Computers, IEEE Transactions on* , vol.C-32, no.2, pp.108-120, Feb. 1983

[2] P. Fiorini and Z. Shiller,"Motion planning in dynamic environments using velocity obstacles", *Int. J. Robot. Res.*, vol. 17, no. 7, pp.760 - 772 1998

[3] J. van den Berg, J. Snape, S. Guy, D. Manocha. Reciprocal collision avoidance with acceleration-velocity obstacles. *IEEE Int. Conf. On Robotics and Automation*, 2011.

[4] van den Berg, J., Guy, S. J., Lin, M. C., Manocha, D.: Reciprocal n-body Collision Avoidance.In: *Proc. Int. Symp. Robot. Res.,* (2009)

[5] J. Fischer, A. Ruppel, F. Weisshardt, and A. Verl, "A rotation invariant feature descriptor o-daisy and its fpga implementation," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on. IEEE*, 2011, pp. 2365–2370.

[6] J. Svab, T. Krajnik, J. Faigl, and L. Preucil, "Fpga based speeded up robust features," *in Technologies for Practical Robot Applications*, 2009. *TePRA* 2009.

[7] Maya B. Gokhale and Paul S. Graham,"Reconfigurable Computing, Accelerating Computation with Field Programmable Gate Arrays", 1st Ed. pp. 1-10, ISBN: 978-0387261058, Netherlands, Springer Publications 2005.

[8] C. Carbone, U. Ciniglio, F. Corraro and S. Luongo "A Novel 3D Geometric Algorithm for Aircraft Autonomous Collision Avoidance" In Proceedings of the 45th *IEEE Conference on Decision and Control*, USA, 2006

[9] Youshen Xia; Jun Wang; Hung, D.L.; , "Recurrent neural networks for solving linear inequalities and equations," *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on* , vol.46, no.4, pp.452-462, Apr 1999

[10] C. R. Karr, M. A. Craft, and J. E. Cisneros, "Dynamic obstacle avoidance", in *Proc. Conf. Distrib. Interact. Simul. Syst. Simul. Train. Aerosp. Envir.*, *Int. Soc. Opt. Eng.,* Orlando, USA, 1995,pp. 195–219.

[11] Khatib, O.;, "Real-time obstacle avoidance for manipulators and mobile robots," *Robotics and Automation. Proceedings. 1985 IEEE International Conference on* , vol.2, no., pp. 500- 505, Mar 1985

[12] Bruce, J.; Veloso, M.; "Real-time randomized path planning for robot navigation," *Intelligent Robots and Systems, 2002. IEEE/ RS International Conference on* , vol.3, no., pp. 2383- 2388 vol.3, 2002

[13] Gonzalez, R.; Horowitz, M.; "Energy dissipation in general purpose microprocessors," *Solid-State Circuits, IEEE Journal of* , vol.31, no.9, pp.1277-1284, Sep 1996.