FPGA BASED MULTI-ROBOT COLLISION AVOIDANCE SYSTEM

Amritha S B
National Institute of Technology, Tiruchirapalli
E-mail: b.amritha5@gmail.com

Bhavya Krishna
National Institute of Technology, Tiruchirapalli
E-mail: bhavyakrishna24@gmail.com

Sujana Ramesh
National Institute of Technology, Tiruchirapalli
E-mail: sujanaramesh90@gmail.com

Vivek D C
National Institute of Technology, Tiruchirapalli
E-mail: dc28vivek@gmail.com

N Ammasai Gounden
National Institute of Technology, Tiruchirapalli
E-mail: ammas@nitt.edu

Disclaimer All authors contributed equally to this paper.

Abstract. This paper presents a Multi-robot collision avoidance system suitable for FPGA implementation. The proposed scheme attempts a novel method of Reciprocal Collision Avoidance (RCA) along with Accelerating Velocity Obstacles (AVO). Parallel architecture is used for developing the collision analysis of robots with different targets. Each of these robots moves and reaches its target, avoiding collisions on its path. Simulations for the digital design hardware have been carried out in Vivado software and the robotic movements have been demonstrated using V-rep software. The proposed scheme would result in reduced cost and power dissipation while deploying in FPGAs. The details of the collision detection algorithm and the working of different modules used in the scheme along with test results are presented in the paper.
1. Introduction

The field of autonomous Multi-robot systems is expanding every day with the increase in automation in industries. Obstacles always pose a great threat in Multi-robot systems. Consequently, the obstacle avoidance system is a major focus of such systems. In those systems, where there are no other moving objects other than the robot itself, static obstacle avoidance algorithms are used. Various algorithms for static obstacle avoidance have been found efficient in the field of robotics [1]. Spatial planning is one such algorithm based on characterizing the position and orientation of each robot as a single point in the workspace. Other algorithms include time-varying artificial potential fields and rapidly exploring random trees which have gained popularity in static obstacle avoidance [2,3].

Multi-robot environments involve moving obstacles. Hence, dynamic obstacle avoidance algorithms are in need to have a collision free traversal. [4]. Omnidirectional vision sensor system with catadioptric panoramic cameras can be used for collision avoidance [5]. After applying some existing image processing techniques to the images obtained, the real image is restored and the environmental information around a robot is identified. However real-time application of computer vision requires high computational power and is expensive. A* algorithm can be used to plan the path for collision avoidance by using dynamic real-time monitoring and iterative move-evaluate-move cycles [6]. This can be used to avoid static as well as dynamic obstacles. Distributed Autonomous Robotic System (DARS) uses multi-layered reinforcement learning methods. By dividing a learning curriculum into multiple layers, the number of expected situations can be reduced [7]. The aforementioned methods are computationally intensive, require extensive usage of memory, and may not be feasible for small scale systems.

The hardware implementation of Multi-robot collision avoidance systems has been done by various methods with the help of different sensors such as RFID [8], ZigBee communication, but these have practical difficulties such as Dead corners in sensing. Long distance transmission of data also poses a problem in hardware implementation which can be improved with repeaters, this would increase the cost of the system.

Unlike a hardware system, a simulated system enables testing of the algorithm for a range of system parameters like the number of bots, robot constraints and nature of robot used. Simulation system based research on mid-air collision avoidance between pairwise non-cooperative aircraft presents an extended 3D case method to the geometric approach of collision cone based algorithms which are generally used in planar mobile robots. The algorithm presents avoidance maneuvers both in horizontal and vertical planes and the effectiveness is checked using numerical simulations in proper conflict scenarios [9].

A very popular approach of collision avoidance is based on velocity and is known as the Velocity Obstacle (VO) approach. Moving robots are known as Velocity Obstacles. When acceleration constraints are included, it is called AVO [10]. When every robot takes the responsibility to avoid collisions, it is termed as RCA [11]. In the aforementioned algorithm, the current position and velocities of each robot in the workspace are used to find the set of velocities that can avoid the obstacles by satisfying the constraints. The set of velocities that will result in a collision forms the collision cone [10,11,12].

The collision cone inequality can be solved using various approaches. One of these uses linear programming to solve all the inequalities with the acceleration constraints and obtain the velocity that falls out of the collision cone. The linear inequalities can be solved using recurrent
neural networks [13]. The main disadvantage of this approach is that it is expensive in terms of clock utilization and resources when executed in FPGA.

The need for FPGA arises from the fact that it is highly reconfigurable. This feature is particularly helpful in the robotic industry, as each robot would have different requirements, different tasks to perform, different sensor data to collect [14]. So finding processors for each robot’s provision would be tedious, and also customized products would lead to an increase in overall system cost. As FPGA is flexible and scalable, it has the potential to revolutionize the already rapidly changing robotics industry [15].

One of the critical factors to note is that most of the microcontrollers currently used in robotics are sequential in nature, whereas parallel architecture can be designed and executed in FPGAs. With FPGA as the processing unit, the size, the cost of the robot, and execution time decrease. FPGAs allow development in gate-level architecture, which makes them highly reliable for speed and accuracy [16]. This gives them an added advantage over the existing microcontrollers. The use of FPGA reduces the power dissipation as only the shortest path is implemented. Parallelism can be achieved into the system by making multiple instantiations of the same elementary level modules and using them in parallel, ensuring fast computation with minimum hardware. When a network of a large number of robots is deployed, a significant reduction in power dissipation can be witnessed [17].

1.1. Contributions of this paper

This work primarily focuses on multiple robots reaching their respective targets by avoiding collisions with each other. The proposed scheme has utilized the V-rep platform for the simulation which allows the design of customized bots with proper handling of kinematics/inverse kinematics. RCA with AVO is used in the present scheme which primarily works with velocity as a parameter for avoiding obstacles. This technique makes the system more robust to the dynamic changes in the environment besides consuming lesser power, faster and less computationally intensive in contrast to the already existing methods.

The novelty of this paper lies in the execution of the velocity selection strategy, communication between the software, and limited-time path planning.

- A hybrid of to goal and maximum velocity strategy is proposed in velocity selection to reduce the traversal time.
- Handshake signals have been interspersed with the communication modules to counter the mismatch in the rate of data transfer between the two software
- The path planner deployed is made of primitive logic structures thus making optimal use of resources.

Two major platforms are analyzed for the implementation of the scheme - namely ModelSim and Xilinx Vivado. Vivado is an integrated design environment (IDE) with system-to-IC level tools built on a shared, scalable data model and a common debug environment. The software also contains an elaborate IP catalog that can be used for multiple protocol designs and allows direct synthesis onto an FPGA system. Hence this software was chosen over ModelSim. The present work has utilized the un-paralleled runtime and memory capabilities of the software to perform computations at timings closest to runtime.
2. Theory

2.1. Collision cone

This paper focuses on the novel implementation of RCA with AVO. This implementation is cost-effective and has lower power dissipation. In this algorithm, the current position and velocities of each robot in the workspace are used along with constraints of the robot to find the set of velocities that can avoid the obstacles by satisfying the constraints. The set of velocities other than those used to avoid collision are known as the Velocity Obstacle. This set of velocities forms the collision cone. Figure 1 shows a collision cone (C1/2) where the collision between a robot at (x1,y1) and a dynamic obstacle at (x2,y2) is predicted. Any velocity outside this cone is the velocity which the robot can take to avoid obstacles successfully.

![Figure 1: Collision cone](image)

Equation mentioned below is the governing equation of the collision cone. In these equations, robots other than one which is controlled are considered as passive. All the velocities satisfying this equation are the avoidance velocities. The collision cone forms the basis for collision avoidance in this paper [18].

\[ d^2 = \frac{r^2 - (\vec{r}.\vec{V})^2}{V^2} \geq R^2 \]

2.2. Update module

The update module predicts the positions and velocities of the robots at the next instant. Throughout this paper, the robots have uniform acceleration in the direction of motion. Thus, the equations of kinematics are applicable and are used to predict the state of the robot in the next instant.

\[
\vec{v}_{n+1} = \vec{v}_n + \vec{a}_n t \\
\vec{r}_{n+1} = \vec{r}_n + \vec{v}_n t + \frac{1}{2} \vec{a}_n t^2
\]

where,

\[
\vec{r} = x\hat{i} + y\hat{j} \\
\vec{v} = v_x\hat{i} + v_y\hat{j}
\]
\[ \vec{a} = a_x \hat{i} + a_y \hat{j} \]

This module was implemented at the gate level, with the same multipliers and adders used repeatedly over different clock cycles. Figure 2 shows the architecture of the update module. Each horizontal level in the diagram represents a clock cycle. The clock frequency is set according to the constraints of gate propagation delays. The major inputs to the modules are the current position, velocity, acceleration, and sampling time and the module outputs the predicted position and velocity of the robot for the next instant.

Figure 2: Update module architecture. Takes current velocities and coordinates as the input and computes the velocities and coordinates at the next instant of time. Uses different stages to minimize the amount of hardware resources used.

2.3. Collision detection module

This module evaluates the inequality in the equation below and predicts whether there will be a collision or not. If the velocity falls outside the collision cone, the inequality holds true and there will be no collision at the next instant. If a collision is detected, the velocity is modified and the inequality is rechecked with the updated value of velocity.

\[ r^2 \vec{v}^2 - (\vec{r} \cdot \vec{v})^2 \geq R^2 \vec{v}^2 \]

Where \( \vec{r} \) is the relative position, \( R \) is the sum of the radii of both the robots and \( \vec{v} \) is the relative velocity of the robot 1 with respect to robot 2 respectively. Similar to the update module, the collision detection module is also implemented at the gate level with recursive usage of multipliers and adders. The inputs to this module are the velocity and position of two different robots in cartesian form. The collision detection module implemented has two 32 bit multipliers, four 32 bit subtractors, one 64 bit subtractor, three 32 bit adders, and one 64 bit comparator. This architecture is shown diagrammatically in Figure 3.
3. Methodology

To achieve the given objective we propose a three-module scheme.

- **Vivado Modules** - consisting of Verilog modules implementing the necessary computations and algorithms for collision avoidance
- **VREP Modules** - consisting of modules required to create robots, control the kinematics of the bot and configuring the initial and simulation setup
- **Communication Modules** - consisting of both Vivado and Vrep modules along with text files to facilitate the communication between the above-mentioned modules.

**Figure 3:** Collision detection module. Takes updated velocities of the bots and computes if there will be collision in the next instant of time. Uses different stages to reduce the hardware used.
3.1. Velocity Selector Module

Once a collision is detected by any of the robots, it has to take steps to avoid the collision. The velocity selector module changes the current velocity either by changing the magnitude and keeping phase constant or by changing both magnitude and phase. This module needs to access the coordinates in polar form, hence rectangular to polar conversion and vice-versa are implemented using CORDIC IP[19]. Coordinate rotational digital computer (CORDIC) algorithm was initially developed to solve non-linear trigonometric equations, then developed to solve other non-linear equations such as square root and hyperbolic equations. The CORDIC CORE of the CORDIC IP implements a various range of equations based on the set configuration. Xilinx’s CORDIC v6.0 IP is being used in this paper[20]. The CORDIC architecture is shown in Figure[5].

Figure 4: Three module scheme. Verilog modules implementing the necessary computations and algorithms for collision avoidance, Vrep modules controls the robot kinematics and configures the environment and Communication modules to aid communication between Vrep and Vivado modules.

Figure 5: Cordic IP
Figure 6: Velocity selector flowchart. Takes the current velocity in rectangular coordinates as the input, uses CORDIC IP for rectangular to polar and polar to rectangular conversions. Follows hybrid to goal and maximum velocity strategy. Gives the modified velocity as the output.

Figure 6 shows the flowchart of the velocity selector module. Once the necessary conversions have been made, the module has two steps of action. First, the magnitude of the velocity is decreased by a predetermined step size (to be designed based on the robot’s speed and torque constraints). The new velocity is passed through the update and collision detection modules to recheck for collisions. If there is no collision, the velocity is fed to the robots. If a collision is detected, the velocity is changed again. However, just decreasing the magnitude of velocity may eventually stop the robot and also increase the traversal time of the robot. This paper uses a novel method which is a hybrid of the to goal and velocity selection strategies. This method involves modifying the phase of the robot once the velocity reduces below a given threshold. Once the phase is modified, the magnitude of velocity is set to the maximum possible velocity. Hence the bots try to reach the target in the least possible time. The CORDIC IP then converts the polar values to rectangular values and the output of the module is the modified velocity of the bot.

3.2. Path planner and target checker
The velocity that has been changed by the velocity selector need not be in the direction of the bot’s target destination. In order to make the bot reach its target destination, a path planning algorithm is being implemented.

The path planning algorithm is implemented as a part of the integrator module. For each iteration when no collisions are detected for a robot, a count variable is incremented. When the count goes above eight, the robot is given velocity in the direction towards its target from its current position. The velocity vector is calculated by assigning the rectangular components, values proportional to the difference in the cartesian coordinates of the robot’s current position,
and its target coordinates. Since such an approach will yield very small velocities when the robot is close to the target, the proportionality constant is scaled up. The count is reset whenever a collision is detected with another robot to allow velocity selector to take in-charge and assign a velocity that does not result in a collision with the other robots. The surface of the traversal of the robots is assumed to be smooth and accelerations are held approximately constant throughout a unidirectional motion.

The accelerations (ax and ay) are set such that the net acceleration, $\vec{a}$ is in the direction of current velocity and stays the same till the velocity is changed.

While the path planner makes sure the robot is moving in a direction towards the target, the target checker ensures the robot stops once it has reached. The target checker module of each robot takes as inputs the current coordinates of the robot and compares it with its target coordinates to decide if it has reached the target. Keeping in mind the robot’s dimensions with an 11cm*19cm chassis, an error of 2cm is permitted currently and can be changed if high precision is required according to the application. Each time the positions and velocity feedback from the robot reaches the Verilog script, the target checker operates. Once the target checker module establishes that the robot has reached its target, it sets both the robot velocities (Vx and Vy) and accelerations (Ax and Ay) to zero to bring it to a stop immediately. The integrator module corresponding to the robot is disabled as it has completed its journey. The update modules of all the robots and integration modules of other robots continue to operate until all the robots have reached their targets after which simulation is stopped.

3.3. Collision detection

The collision detection module constraint explained in the theory section is valid only when there is non zero relative velocity and for a non zero angle between relative velocity and relative displacement between the two bots. It has to be modified as follows to suit the application of the system presented in this paper. When the relative velocity and relative displacement are in the same direction, relative displacement can be directly compared with the R as shown in the below equation.

$$r^2 > R^2$$

If the relative velocity is zero, then the entire collision detection module is bypassed.

3.4. Kinematics of the robot

The kinematic module implemented in the robotic simulation platform V-Rep comprises the computation of the motor speed of the left and right wheels of the robot and setting it to the dynamic joints actuating the wheels in simulation. Since the robot design used is a differential drive, there are certain non-holonomic constraints on establishing its position. For example, the robot cannot move laterally sideways along its axle. A simple approach to achieve any direction of the target velocity vector is to first turn the robot in the direction required and then move straight according to the magnitude of the target velocity vector. Since the orientation of the robot at any given instant need not align with the global axes, the relative rotation required is calculated before rotating. The orientation of the robot at any instant is calculated by setting it to align with the global axes initially before the motion starts and updating it after each rotation. Angle of target velocity vector is found by

$$\theta = \tan^{-1} \frac{vy}{vx}$$
But since the solutions of \( \tan^{-1} \) are in the range \((-90^\circ, 90^\circ)\) only, while the angle of target velocity vector can be in the range \((-180^\circ, 180^\circ)\), based on the signs of \(V_X\) and \(V_Y\), \(\theta\) is modified to increase range and achieve rotation to any angle. Then, required relative angle of rotation is

\[
\psi = \theta - \phi
\]

where, \(\phi\) is the current orientation of the robot in global axes.

Based on whether \(\psi\) is negative or positive, the robot turns right or left respectively. Once the relative rotation is computed, the robot has to be rotated accordingly. The rotation is achieved by stopping the wheel in the direction of rotation and actuating the other wheel to move along a circular path with the stopped wheel as the center and the axle length as the radius like in Figure 7 [21].

![Figure 7: Robot turning left](image)

3.5. Communication

Communication is an essential module between any two entities. In this work, efficient communication between the two software is a requirement for the smooth functioning of the implementation. The following methods have been used for communication purposes.

This work employs text files to pass data between the two software. Six text files (two for each bot) have been created and stored in the appropriate folders. For every bot, the first text file carries information from the V-rep software indicating the cartesian velocities and positions of the bot at any given instance. The second text file contains the computed values of velocities written by the computational software after performing the necessary calculations and checking the conditions.

The first step of the simulation is to write the initial velocities and positions of the bots to the first text files. This operation is performed simultaneously for all the bots from the simulation software. Vivado software then reads these files simultaneously. Once the values are read and stored, the computations take place, and based on whether a collision is detected or not the magnitude of velocity is decreased or remains stable.

The computations and writing of values to the text files are performed independently of each other for all three bots. The time delay of one bot does not affect the other as the operations take place normally. Once the computations are completed, the second textfile for each bot is...
opened. The values of velocities are written on to the file which is then read by V-rep. The new velocities are then given to the bots in the simulation.

These two software, however, are not synchronized. The rate at which Vivado writes to the text file is different from the rate at which V-rep reads the same and vice-versa. Hence, a problem of asynchronization between the two software occurs. The reading of text file by Vivado must take place only when the velocities and positions are updated by the bots simulated in V-rep. Similarly, the bots must read the velocities from the text file and update them to the simulation only when all the computation in Vivado has been completed and the revised velocities have been updated. To overcome these problems handshake signals have been used.

Handshaking is defined as “The establishment of synchronization between sending and receiving equipment by means of exchange of specific character configurations.” Handshaking is used as one of the major methods for the establishment of a synchronized communication between two channels. It allows the receiver to know when the input is available and allows the sender to know when the outputs have been received successfully by the receiver. The sender waits until the handshake signal is set before sending the next set of information to the receiver. In case of an error, the handshake signal isn’t set so the sender either waits for some time or resends the data.

This paper uses characters as handshake signals between the two software. As seen in Figure 8, the first line of the text file contains a character. This character is chosen as ‘r’ or ‘w’ for simplicity. Once the velocity and positions of the robot are known to the simulation in V-rep, the values are written to the text file. The character ‘w’ in the first line indicates that the values of velocity and position have been written and are ready for computation. The computational software Vivado then reads the file and updates the character in the first line to ‘r’ to indicate that the values have been read successfully. Only when this is done is the file updated with further velocities and positions of the robot.

**Figure 8:** Velocities and positions written to text file

**Figure 9:** Velocities and positions read from text file
The Figure 10a and Figure 10b represent the reverse handshake phenomenon between the computation and simulation software. Once the computational software i.e. Vivado receives the initial velocities and positions of the bot, it performs the analysis of whether there will be a collision or not and what to do in each case. Finally, it writes the updated velocities in the file along with the character ‘w’ in the first line, indicating that the new velocities have been updated. V-rep then reads the new velocities from the file, updating the character to ‘r’ indicating to the computational software that the data has been communicated successfully.

3.6. Integration of modules

As shown in the block diagram in Figure 4, the Integrator module integrates various other modules required for computing the collision detection algorithm for the bot under consideration. Each of the modules discussed above is designed with input_rdy signal as an input and output_rdy signal as an output. These signals are used in coordinating the modules one after another. Input_rdy signals are held high until the complete module’s execution is completed, while output_rdy signals are pulses that mark the end of the module’s execution.

As the objective is to prevent any collision in a three bot system, each bot’s integrator module is equipped with two collision detection modules. (Say bot 1 is under consideration, two collision detection modules are needed to prevent 1-2 and 1-3 collisions.) Velocity has to be changed even if one of the collision detection modules among CD1 and CD2, detects a collision. On detecting a rising edge in the input_rdy of the Integrator, CD1_in_rdy and CD2_in_rdy signals are being set to logic 1 hence activating the CD1 and CD2 modules. After the execution of the CD1 and CD2 modules, CD1_out_rdy and CD2_out_rdy pulses are being generated by the respective CD modules. Integrator on detecting the output_rdy pulses from CD modules deactivates the CD modules by writing logic 0 to CD1_in_rdy and CD2_in_rdy signals.

The output of the CD modules are stored in the coll_detect1 and coll_detect2 variables. When both coll_detect1 and coll_detect2 are logic 0 (implies no collision), the write_test module is being activated by holding write_check signal high. After the current velocities are written into the file by the write_test module, IG_output_rdy pulse is being generated to mark the end of execution of the integrator module.

If either coll_detect1 or coll_detect2 is high (implies there is a collision with at least one bot), the velocity selector module(VS) is being activated by holding VS_in_rdy signal high. The current velocity is being passed on to the VS module. After the execution of the module, VS
generates VS\_out\_rdy pulse conveying the end of its execution. Once the Velocity selection is completed, the current chosen speed has to be updated to check if it would result in a collision in the next instant.

Therefore, on detection of VS\_out\_rdy pulse, VS\_in\_rdy is held low, deactivating the VS module and UM\_in\_rdy is held high, activating the update module (UM). After the velocities are being updated, UM\_out\_rdy pulse is being generated in the update module. On detection of the UM\_out\_rdy pulse, CD modules are being activated again. This process continues until a condition of no collision with all the bots is being reached.

**Figure 11:** Integrator source code hierarchy

The top module is the main module that controls each bot’s activity and is the topmost in the hierarchy. The block diagram of the top module is given in Figure. [4] To read the data written by the VREP modules, the read\_test module is being employed. Current velocities and current coordinates are read from the text files in cartesian form and loaded on to the variables for each of the bot. On completion of the read\_module, input\_read pulse is being generated.

In the top module, on detection of the input\_read pulse, Update modules of 1,2 and 3 are activated with the respective input signals(UM\_in\_rdy1,UM\_in\_rdy2 and UM\_in\_rdy3). To reduce the number of update modules and hence to reduce the hardware required for synthesis, the first set of Update modules are being implemented in the top rather than in the integrator. Update modules completion is marked with logic 1 in UM\_out\_rdy1, UM\_out\_rdy2 and UM\_out\_rdy3. Once all the update modules are completed, UM\_all\_done signal is being set.

On the rising edge of UM\_all\_done, all the integrator blocks IG\_in\_rdy1, IG\_in\_rdy2 and IG\_in\_rdy3 are set. The number of integrator blocks equals the number of robots in the environment. Once the integrator block is completed, it generates IG\_out\_rdy pulses. On detecting the IG\_out\_rdy pulse, the respective integrator is deactivated by writing 0 to its IG\_in\_rdy.

Target checker is being used to check if the bot has reached its target destination. On reaching the target, top activates the write\_test module of the respective bot to write zero velocity into it.
4. Testing and results

Three levels of testing were performed and results were obtained. Level 1 testing contains every module testing parameters and results. Level 2 testing gives input via textfiles, hardcoding velocities and positions. The outputs are observed on the waveforms. The final stage of testing contains outputs obtained from running both the simulation and computational software simultaneously.

4.1. Module testings
4.1.1. Velocity selector module

Figure 13 shows the velocity selector operation where phase is held constant and magnitude is changed.

![Velocity selector operation](image)

**Figure 13:** Wave output of velocity selector with constant phase and reduced magnitude

\[ x_{\text{real}}, y_{\text{real}} \] are the input rectangular coordinates from the integrator. On receiving an active signal, the conversion process starts. The rectangular coordinates are converted to polar and stored in the variables mag and phase. From Figure 13, it is clear that the magnitude is changed while keeping phase constant (\( \text{mag1} \) is reduced from mag and phase1 is held same as phase). \( x_{\text{rot}} \) and \( y_{\text{rot}} \) are the final rectangular coordinates having \( \text{mag1} \) and phase1 as the magnitude and phase. The end of the velocity selector module is marked with the vs_done pulse.
Figure 14: Wave output of velocity selector with constant magnitude and reduced phase

Figure 14 shows the velocity selector operation where magnitude is held constant and phase is changed. This is done when the reduced magnitude is lesser than the threshold. (mag1 is held constant and phase1 is changed)

4.1.2. Integration module
The following figures show the testing of the integrator module that involves generating signals to activate and deactivate different modules computing for each robot.

Figure 15: Wave output of integrator module

Figure 15 shows a complete execution of the integrator module. This can be inferred from the input_rdy signal going 0. At the rising edge of the input_rdy, CD1_in_rdy and CD2_in_rdy goes high. After the execution of the collision detection module is complete, CD1_out_rdy and CD2_out_rdy pulses are being generated. Figure 16 shows the zoomed in version of the first collision detection output.

As coll_detect1 gives a high output, Velocity selector is being activated by making VS_in_rdy high. From Figure 15, at the end of Velocity selector operation, VS_out_rdy pulses are being generated and Velocity selector module is being deactivated by holding VS_in_rdy to 0. The
VS_out_rdy pulse also triggers the Update module by making UM_in_rdy high. The end of the update module is marked by UM_out_rdy pulse, which activates the collision detection modules by making CD1_in_rdy and CD2_in_rdy high. At the end of CD modules with CD_out_rdy pulses are being generated. Figure 17 shows the zoomed in version of the second collision detection output. As both coll_detect1 and coll_detect2 are 0, output_in_rdy pulse is being generated, triggering the write modules to write the updated velocity in to the bots.

![Figure 16: Wave output of integrator module with collision](image1.png)

![Figure 17: Wave output of integrator module with no collision](image2.png)

This example shows the complete working of the integrator changing the velocity with collision condition to the one with no collision.

4.1.3. Top module

The following figures show the testing of the top module that involves generating signals to activate and deactivate different modules in it.
Figure 18: Wave output of top module showing input_read function

Figure 18 shows the file read operation, just when the file is being read and the variables are being loaded with the values, input_read pulse is being generated.

Figure 19: Wave output of top module showing UM activation and deactivation

Figure 19 shows the rising edge of the input_read signal triggering the Update module 1, 2 and 3. The completion of the Update module is marked with UM_out_rdy pulses, that deactivates the corresponding update module by writing 0 to the UM_in_rdy signals.

Figure 20: Wave output of top module showing IG activation and deactivation

Figure 20 shows UM_all_done pulse triggers the integrator modules, by holding IG_in_rdy1, IG_in_rdy2 and IG_rdy3 pulse high. The completion of the integrator module is marked with the IG_out_rdy pulses, which the deactivates the corresponding integrator module by writing 0 to the IG_in_rdy signal.
### 4.2. Overall computational testing

In this Level, we combine all the individual modules and run an overall module testing to check the integration between each different modules. Table 1 displays the input parameters for the 3 Robots, and Table 2 indicates the target values for each of the Robot.

| Parameters | Bot 1 | Bot 2 | Bot 3 |
|------------|-------|-------|-------|
| $x$ | 1 | 800 | 0 | 0.5 | 400 |
| $y$ | 1 | 800 | 0 | 0.5 | 400 |
| $v_x$ | -0.04 | FFFFFAF | 0.02 | 28 | 0.02 | 28 |
| $v_y$ | -0.04 | FFFFFAF | 0.02 | 28 | 0.02 | 28 |
| $a_x$ | -0.3 | FFFFFD9A | 0.5 | 400 | 0.3 | 266 |
| $a_y$ | -0.3 | FFFFFD9A | 0.5 | 400 | 0.3 | 266 |
| $R^2$ | 0.0484 | 318FC | 0.0484 | 318FC | 0.0484 | 318FC |

**Table 1:** Input values for test case 1

| Parameters | Bot 1 | Bot 2 | Bot 3 |
|------------|-------|-------|-------|
| $x$ | 0 | 0 | 0.5 | 400 | 1 | 800 |
| $y$ | 0 | 0 | 0.5 | 400 | 1 | 800 |

**Table 2:** Target values for test case 1

For the above mentioned test case, we expect the Robots to encounter collision, and the Overall Module behaves accordingly and detects collision and also takes steps to avoid it and reach their respective targets.

Table 3 and Table 4 displays the values for the next test case. Here, in this case the robots does not encounter any collision and they reach their respective targets in their initial course without any deviation.

| Parameters | Bot 1 | Bot 2 | Bot 3 |
|------------|-------|-------|-------|
| $x$ | 10 | 5000 | 9 | 0 | -10 | FFFFFB000 |
| $y$ | 1 | 800 | 8 | 0 | -10 | FFFFFB000 |
| $v_x$ | 0 | 0 | -0.76 | FFFFF9EC | 0.4 | 333 |
| $v_y$ | 0.36 | FFFFFAF | -0.72 | FFFFFFA3E | 0.4 | 333 |
| $a_x$ | 2 | 1000 | 1 | 800 | 3 | 1800 |
| $a_y$ | 1 | 800 | -1 | FFFFF800 | 2 | 1000 |
| $R^2$ | 0.0484 | 318FC | 0.0484 | 318FC | 0.0484 | 318FC |

**Table 3:** Input values for test case 2
Table 4: Target values for test case 2

| Parameters | Bot 1 | Bot 2 | Bot 3 |
|------------|-------|-------|-------|
|            | Value | Value | Value |
| $x$        | 10    | -10   | 0     |
| $y$        | 5000  | -10   | 0     |

4.3. Simulation testing

CASE 1: When Robot1 and Robot2 are on the way to collision, the robot is expected to slow down. This can be seen in the values of $v_x$ and $v_y$ read from Vivado and printed in console log of VREP shown in Figure 21 below.

**Figure 21:** Robot1 slowing down when collision with robot2 is predicted

CASE 2: Robot 2 has reached its target as it can be seen in the following Figure 22. The target checker in Vivado found that the robot 2 has reached target and stopped it by writing $v_x$ and $v_y$ as zero as shown in Figure 23 and Figure 24.

**Figure 22:** VREP scene showing robot2 in its target subspace
Figure 23: Target checker finding out that robot2 has reached target

Figure 24: Text file showing VX and Vy written as zeros by Vivado

Figure 25 shows the trajectory and paths followed by the robots when there is a collision and when collision is avoided.

Figure 25: Trajectories

5. Conclusion
The paper has implemented a novel method of Reciprocal Collision Avoidance (RCA) along with Acceleration Velocity Obstacles (AVO); this implementation has been effective in reducing cost and power dissipation while deploying in FPGAs. This implementation would be a significant contribution to the field of low-cost robotic systems.

In this paper two software namely Vivado and V-REP have been used for computation and simulation respectively. The communication between the two software takes place using text files. Synchronization has been achieved using handshake signals. The surface of the traversal of the robots is assumed to be smooth and accelerations are held approximately constant throughout a unidirectional motion. Once the simulation begins, the positions and velocities of the bots are written to the respective text files which are then read by the simulation software and given to
the bots. If the bot is in a direction different from that of its target, the path planning algorithm comes into effect. If there is no collision for a substantial amount of time, this algorithm changes the velocities of the bots in the direction of their respective targets. The target checker algorithm checks if the bots have reached their targets and once reached it writes zero velocity to the file which stops the robots.

This paper contributes to the wide range of applications that Multi-robot systems have from emergency response and rescue, natural resource monitoring, warehousing, military missions, outdoor industrial operations, and home-care.

Appendix

The link to the video of our simulation is given below.

https://drive.google.com/file/d/1KCDbS_rChEolmB5wayEiHxstO75krfR/view

References

[1] Lozano-Perez. “Spatial Planning: A Configuration Space Approach”. In: IEEE Transactions on Computers C-32.2 (1983), pp. 108–120.
[2] J. Bruce and M. Veloso. “Real-time randomized path planning for robot navigation”. In: IEEE/RSJ International Conference on Intelligent Robots and Systems. Vol. 3. 2002, pp. 2383–2388.
[3] O. Khatib. “Real-time obstacle avoidance for manipulators and mobile robots”. In: Proceedings. 1985 IEEE International Conference on Robotics and Automation. Vol. 2. 1985, pp. 500–505.
[4] Clark R. Karr, Michael A. Craft, and Jaime E. Cisneros. “Dynamic obstacle avoidance”. In: Distributed Interactive Simulation Systems for Simulation and Training in the Aerospace Environment: A Critical Review. Ed. by Thomas L. Clarke. Vol. 10280. International Society for Optics and Photonics. SPIE, 1995, pp. 192 –216.
[5] C. Cai, C. Yang, Q. Zhu, and Y. Liang. “Collision Avoidance in Multi-Robot Systems”. In: 2007 International Conference on Mechatronics and Automation. 2007, pp. 2795–2800.
[6] Fan Liu and A. Narayanan. “Real time replanning based on A* for collision avoidance in multi-robot systems”. In: 2011 8th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI). 2011, pp. 473–479.
[7] Yoshikazu Arai, Teruo Fujii, Hajime Asama, Hayato Kaetsu, and Isao Endo. “Collision avoidance in multi-robot systems based on multi-layered reinforcement learning”. In: Robotics and Autonomous Systems 29.1 (1999), pp. 21 –32. ISSN: 0921-8890.
[8] Joyashree Bag. “Design and VLSI Implementation of Anticollision Enabled Robot Processor Using RFID Technology”. In: International Journal of VLSI Design Communication Systems 3 (Dec. 2012), pp. 51–65.
[9] C. Carbone, U. Ciniglio, F. Corrarlo, and S. Luongo. “A Novel 3D Geometric Algorithm for Aircraft Autonomous Collision Avoidance”. In: Proceedings of the 45th IEEE Conference on Decision and Control. 2006, pp. 1580–1585.
[10] J. van den Berg, J. Snape, S. J. Guy, and D. Manocha. “Reciprocal collision avoidance with acceleration-velocity obstacles”. In: 2011 IEEE International Conference on Robotics and Automation. 2011, pp. 3475–3482.
[11] Jur van den Berg, Stephen Guy, Ming Lin, and Dinesh Manocha. “Reciprocal n-Body Collision Avoidance”. In: vol. 70. Apr. 2011, pp. 3–19.

[12] Paolo Fiorini and Zvi Shiller. “Motion Planning in Dynamic Environments Using Velocity Obstacles”. In: The International Journal of Robotics Research 17 (July 1998), pp. 760–768.

[13] Youshen Xia, Jun Wang, and D. L. Hung. “Recurrent neural networks for solving linear inequalities and equations”. In: IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications 46.4 (1999), pp. 452–462.

[14] M. Martínez-Prado, J. Rodríguez-Reséndiz, R. Gómez-Loenzo, G. Herrera-Ruiz, and L. Franco-Gasca. “An FPGA-Based Open Architecture Industrial Robot Controller”. In: IEEE Access 6 (2018), pp. 13407–13417.

[15] M. Martínez-Prado, J. Rodríguez-Reséndiz, R. Gómez-Loenzo, G. Herrera-Ruiz, and L. Franco-Gasca. “An FPGA-Based Open Architecture Industrial Robot Controller”. In: IEEE Access 6 (2018), pp. 13407–13417.

[16] Tarek Ould Bachir, J.-P David, Christian Dufour, and Jean Bélanger. “Effective FPGA-based electric motor modeling with floating-point cores”. In: Dec. 2010, pp. 829–834. DOI: 10.1109/IECON.2010.5675179

[17] S.P.B. Muthusamy and V. Subramaniam. “Low-power dissipation using FPGA architecture”. In: Mar. 2012, pp. 418–421. ISBN: 978-1-4577-1545-7. DOI: 10.1109/ICDCSyst.2012.6188752

[18] R. Dubey, N. Pradhan, K. M. Krishna, and S. R. Chowdhury. “Field Programmable Gate Array (FPGA) based Collision Avoidance using acceleration velocity obstacles”. In: 2012 IEEE International Conference on Robotics and Biomimetics (ROBIO). 2012, pp. 2333–2338.

[19] Neeraj Pradhan, Roopak Dubey, K. Madhava Krishna, and Shubhajit Roy Chowdhury. “Low Power Two-Tier GALS Architecture for Multi Robot Collision Avoidance”. In: Proceedings of Conference on Advances In Robotics. AIR ’13. Pune, India: Association for Computing Machinery, 2013, 1–6. ISBN: 9781450323475.

[20] “CORDIC v6.0 LogiCORE IP Product Guide, Vivado design suit”. In: (Dec. 2017).

[21] Michael Jenkin Gregory Dudek. Computational Principles of Mobile Robotics. Cambridge University Press, July 2010.