FPGA BASED MULTI-ROBOT COLLISION AVOIDANCE SYSTEM

A thesis submitted in partial fulfillment of the requirements for the award of the degree of

B.Tech.

in

Electrical and Electronics Engineering

By

Amritha S B (107116008)

Bhavya Krishna (107116014)

D C Vivek (107116023)

Sujana Ramesh (107116088)

ELECTRICAL AND ELECTRONICS ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY
TIRUCHIRAPPALLI – 620015

MAY 2020

BONAFIDE CERTIFICATE

This is to certify that the project titled **FPGA BASED MULTI-ROBOT COLLISION AVOIDANCE SYSTEM** is a bonafide record of the work done by

Amritha S B (107116008)

Bhavya Krishna (107116014)

D C Vivek (107116023)

Sujana Ramesh (107116088)

in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Electrical and Electronics Engineering** of the **NATIONAL INSTITUTE OF TECHNOLOGY, TIRUCHIRAPPALLI**, during the year 2019-20.

Dr. N. Ammasai Gounden
Guide

Dr. V. Sankaranarayanan
Head of the Department

Project Viva-voce held on _____

Internal Examiner

External Examiner

ABSTRACT

This project presents a Multi-robot Collision Avoidance based on FPGA. The proposed method implements a novel method of Reciprocal Collision Avoidance along with Acceleration Velocity Obstacles. FPGA based implementations have been gaining popularity in the robotics field. This implementation has been effective in reducing cost and power dissipation while deploying in FPGAs. Parallel Architecture is used for implementing the collision analysis of three different robots. Each of these robots moves and tries to reach its target while avoiding collisions on its path. Simulations for the hardware have been done in Vivado software. Robotic simulations have been demonstrated using V-rep software. This project contributes to the wide range of applications of Multi robot Collision Avoidance Systems such as industrial robots.

Keywords : *FPGA*, *collision detection*. *robotic simulation*, *parallel architectiure*, *obstacle avoidance*

ACKNOWLEDGEMENTS

We would like to express our deepest gratitude to the following people for guiding us through this course and without whom this project and the results achieved from it would not have reached completion.

Dr. N. Ammasai Gounden, Professor, Department of Electrical and Electronics Engineering, for helping us and guiding us in the course of this project. Without his guidance, we would not have been able to successfully complete this project. His patience and genial attitude is and always will be a source of inspiration to us.

Dr. V. Sankaranarayanan, the Head of the Department, Department of Electrical and Electronics Engineering, for allowing us to avail the facilities at the department.

We are also thankful to the faculty and staff members of the Department of Electrical and Electronics Engineering, our individual parents and our friends for their constant support and help.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

1.1.1 Need for multi-robot systems

Multi-Robot System consists of multiple robots which can coordinate and communicate with each other to accomplish the tasks assigned to each one of them. The need for implementation of Multi-robot systems in industries comes with the great benefits it offers compared to its single robot counterparts, which includes, improved coverage, reduced manufacturing costs,and performing complex tasks. But these benefits come with a price such as, communication constraints and interference between the robots. Deploying Multi-robot systems gives the manufacturers a massive technological advantage as they are very productive, coordinate and perform tasks which are impossible to achieve with a single robot in the specified time frame.

In Multi-robot System by data fusion and sharing of information among the robots, the robustness of the system is improved; also, information redundancy can be eliminated. For example, whenever the robots exchange information about their position, velocities when they sense each other, they can localise themselves more efficiently.

Multi-Robot System is cost-effective as several simple, cheap robots can be used than using a single powerful robot that is complex and expensive to accomplish a task. These systems are fault-tolerant, robust and reliable, as these systems are usually decentralised and distributed. The inter-robot coordination and collaboration provide a broad set of exciting new applications for the robotics industry, from sensor networks for environmental monitoring, security and defence, to satellite communications, search and rescue, and much more.

1.1.2 Project statement

Obstacle Avoidance is a vital aspect considered in the robotics industry. Static Obstacles which obstruct the path of a robot can be easily avoided with just the sensor data, and no further computation would be required. In contrast, when there are dynamic obstacles in the environment, it would require additional computation, practical algorithms for path planning, which opens up many opportunities for research in this domain.

In this project, we primarily focus on Multiple robots reaching their respective targets by avoiding collisions with each other. To achieve this, we have several mobile robots in our environment, each with their own physical constraints (velocity, acceleration, dimensions of the robot). The objective of the robot is to reach their specified targets by avoiding collision in the least time possible. To implement this, we have categorised the project into many crucial modules. As most of the modules focus on collision avoidance, it is not to be forgotten that the primary object of the robots to reach their targets. The target checker algorithm checks whether the robot has reached its goal or not, and makes the system run till the robots reach their assigned targets.

1.1.3 Objectives

The objective of the project is to implement robotic algorithms on FPGA, leading to the development of low-cost FPGA based robotic systems. Our project focuses on the collision avoidance algorithm of multiple mobile robots implemented in an FPGA system. For each robot, every other robot is a dynamic obstacle. For collision avoidance, we use velocity based collision avoidance, where the algorithm primarily works with velocity as a parameter for avoiding obstacles.

The first task is to find the position and velocity of the robot at its next instant of time. The Update Module generates these positions and velocities and gives it to the next module. Our next task is to predict whether a collision will occur or not between any two robots, with the information received from the Update Module, The collision Detection Module computes and gives a Boolean result for “collision”. Based on the result, the next step is classified, if a collision occurs, suitable velocity is computed using the Velocity Selector Module for the robot to follow, else the robot continues to move in its path. The Path planning algorithm always ensures that the robot is still in its motive to reach the desired target, although if the robot changes its direction midways.

1.1.4 Need for FPGA in robotics

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

One of the critical factors to note is that most of the microcontrollers currently used in robotics are of sequential in nature, whereas parallel architecture can be

made and executed in FPGAs. With FPGA as the processing unit, the size and cost of robot, execution time come down and with FPGAs, gate-level architecture can be developed, which makes them highly reliable for speed and accuracy. This gives them an added advantage over the existing microcontrollers.

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, and this helps in saving a lot of resources.

The power dissipation of FPGA is in the order of milliwatt, which is minimal when compared to other processors used in robotic systems for which power dissipation is in the order of Watt. When a network of a large number of robots is deployed, a significant reduction in Power dissipation can be witnessed.

1.2 LITERATURE REVIEW

The research in the field of Multi-Robot systems is increasing day by day, and it is evident from the vast number of research publications in this domain. Obstacle avoidance always poses a greater threat in Multi Robot Systems. In those systems, where there are no other moving objects other than the robot itself, static obstacle avoidance algorithms are used. There are various algorithms on static obstacle avoidance as mentioned in [1]. The approach here on spatial planning is based on characterizing the position and orientation of each robot as a single point in the workspace. The other approaches for static obstacle avoidance include time varying artificial potential field, rapidly exploring random trees [2]. There have been many such interesting approaches and these are with respect to static obstacles in the workspace.

An algorithm is needed for avoiding collisions by taking other robots as dynamic ones, this algorithm is known as Dynamic Obstacle Avoidance algorithm. There are various approaches for this algorithm, one of them is based on velocity and is known as Velocity Obstacle (VO) approach [3]. It is one of the most effective algorithms when it comes to dynamic environments with multiple robots. When acceleration constraints are included, it is called Acceleration Velocity Obstacle [4]. When robots try to avoid them, it is termed as Reciprocal Collision Avoidance [5].

Our project focuses on novel implementation of Reciprocal Collision Avoidance with Acceleration Velocity Obstacles.This implementation has been effective in reducing cost and power dissipation. In this algorithm, the current position and velocities of each robot in the workspace is 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 [3] [4] [5]. Figure 1.1 shows a collision cone (C1/2) when there is a robot and another dynamic obstacle. Any velocity outside this cone is the velocity which the robot can take to avoid obstacles successfully.[3] [4]

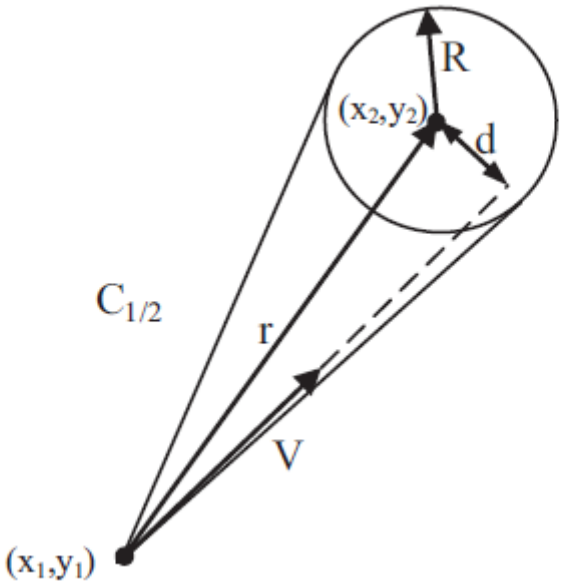


Figure 1.1: Collision cone

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.

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

The hardware implementation of these systems have been done by various methods with the help of various sensors such as RFID [6], zigbee communincation, but these have practical difficulties such Dead corners in sensing and could not meet the requirement of this study due to short sensing distance and small sensing angle. Long Distance transmission of data also poses a problem in hardware implementation which can be improved with repeaters. But that would increase the cost of the system.

There are various methods for hardware implementation and they have their own shortcomings which result in increased costs, affects the performance of the system. Thus there is no perfect combination of sensors to implement it practically, and some compromises have to be made for it.

1.3 SOFTWARES USED

This project focuses on collision avoidance of multiple robots. As seen in the aforementioned sections, the FPGA has been chosen as the primary processor for computation. In order to synthesise the project in an FPGA, the coding has to be implemented in a hardware description language. The hardware description language chosen for this project is Verilog. Two major platforms were analysed for the implementation of the project namely ModelSim and Xilinx Vivado. Visual simulations have been used to provide an easier understanding of the implementation of the project. The Virtual Robot Experimentation Platform commonly known as V-rep has been used for real-time visual and demographic simulation.

ModelSim is a multi-language HDL simulation environment which uses a unified kernel for simulation of all supported languages. It supports coding in Verilog, VHDL, and SystemC. However, the task of file management is a cumbersome process in ModelSim. Also, managing multiple modules within a single framework proved to be a daunting task in this software. Hence this project uses Xilinx Vivado as the major software for HDL coding.

Xilinx Vivado

Vivado Design Suite is one of the top software used for design, synthesis and analysis of HDL designs. 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. It delivers a robust performance using low power. The integrated design and synthesis environment provides ease of use to the programmer. This project has utilised the unparallelled runtime and memory capabilities of the software to perform computations at timings closest to runtime.

V-rep

V-REP is a 3D robot simulation software, with an integrated development environment, that allows you to model, edit, program and simulate any robot or robotic system (e.g. sensors, mechanisms, etc.). This software allows the design of customized bots with proper handling of kinematics/inverse kinematics. Multiple simulation settings and ease of use have made this software a grand success. This platform uses Lua scripting for coding. The versatility of this software has been instrumental in it being a primary choice for this project.

1.4 PROPOSED WORK

This section explains the flow of the thesis. In Chapter 2, the schematic of the workflow of the project is discussed. The complete flow chart is shown and an overview of the project is briefly explained. The various terminologies used are explained in this chapter. In Chapter 3, the major focus is on the methodologies adopted in the implementation of the project. It is subdivided based on the softwares used and the communication between them. Each module is delineated and the input and output signals to every module have been mentioned along with the various signals used. In Chapter 4, the three stages of testing done are elaborated. The results of every module as well as the complete simulation are shown with images and waveforms to support the same. Chapter 5 provides the overall summary of the project along with its future scope.

CHAPTER 2

SCHEMATIC OF WORK

2.1 OVERVIEW

This chapter contains the major terminologies used in the course of this project as well as thesis. The overall schematic of the project is presented in the form of a flowchart. The first section contains the important terminologies. The second section is further subdivided into subsections for Vivado, V-REP and communication modules. A brief introduction about the various modules used is also presented.

2.2 TERMINOLOGIES

I. Differential drive robot : A robot which is driven by two wheels mounted on a common axis, and where each wheel can independently be driven either forward or backward.

II. Collision: An instance or event in which two or more objects(at least one in motion) exert force on each other in about a relatively short time.

III. Sampling time : It is the interval between two successive samples.

IV. Next instant : Next instant is current time added to sampling time.

V. Handshaking : The establishment of synchronization between sending and receiving equipment by means of exchange of specific character configurations

VI. Velocity selection : Selection of a new velocity to prevent the occurrence of an event (in this case collision.)

VII. Cordic : 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.

VIII. Path planning:- Planning the path of a moving object in order to make it reach the target position is known as Path planning.

IX. Target space : Target space is a subspace of free space which denotes where we want the robot to reach at the end of its journey.

X. Scene : A scene is an independent collection of models ,an environment , a main script and pages and views which can be simulated by itself.

XI. Child scripts : A child script is a collection of routines written in Lua attached to (or associated with) scene objects to allow handling a particular function in a simulation.

XII. Threaded child script : Threaded child script is a script whose launch and resume is controlled by the default main script and will launch in a thread in a precise order.

XIII. Arena / Floor : The plane on which all the robots under consideration move about and all the targets are located on.

XIV. Trigger Pulse : A trigger pulse is an asynchronous event that causes a specific change in logical state. Typically, it is wired to the ”set” or ”reset” input of a stateful circuit element.

XV. Fixed point representation : Fixed point representation of fractional or decimal numbers consists of fixed number of integer bits and fractional bits. The data is processed as a fixed point representation in 16 bit 2's complement, with 5-bit integer in this project.

XVI. Mobile Robot: The robot is considered to be a rigid object of known geometry that is capable of moving freely in an area called the workspace that contains a number of fixed rigid objects called obstacles of known geometry and location.

XVII. Obstacles: The obstacles are areas in the configuration space where the robot cannot move.

XVIII. Trajectory: Trajectory is a function of the location of the robot with respect to time. A trajectory also implicitly gives information about the velocity and acceleration of the robot.

2.3 PROPOSED SCHEME

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.

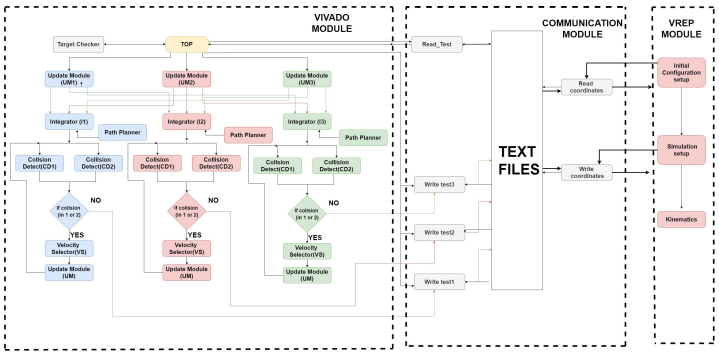


Figure 2.1: Three module scheme

2.3.1 Vivado modules

Top is the highest module of the hierarchy. Hence it is responsible for controlling each bot's activity. Read_Test reads the data consisting of x-coordinate,y-coordinate, velocity in x-direction, and velocity in the y-direction of all the three bots. On completion of loading the values from the file to the variables, the Input_read pulse is generated. The top module on receiving the pulse activates the update modules for each bot. Update module carries out the equations of motion by computing the final velocities and coordinates reached by the bots in the next instant of time. The algorithm relies on finding whether the bot will collide at the next instant. Hence this is a required step.

After updating the velocities and coordinates, the UM_all_done signal activates the Integrator modules of each bot by holding IG_in_rdy signal high. The Integrator module is the one that is responsible for integrating various modules namely collision detection, velocity selector, update module, and write_test module. In a three robot system, we need two collision detection modules in one integrator to find if there is a collision between any of the other 2 bots with the bot under consideration(in an 'n' robot system, we need n-1 collision detection modules). On the rising edge of the IG_in_rdy signal, the collision detection modules are activated by holding CD_in_rdy signals high.

The collision detection module computes and classifies whether there will be a collision or not at the next instance. If there is no collision with all the bots, the velocities are written to the respective write file through the write_test module. If there is a collision with at least one bot, the Velocity selector module is activated by holding VS _in_rdy signal high. Velocity selector takes the current velocity as the input and changes the velocity by either reducing the magnitude or changing the phase. Rectangular coordinates are being used in the computation process of various modules. Hence it is necessary to convert rectangular to polar coordinates and vice-e-versa for the functioning of the velocity selector. The CORDIC algorithm

is being used for facilitating the conversion.

The output of the velocity selector is loaded to the update module to get the next instant velocities and coordinates which are then loaded again in the collision detect modules. This process is repeated until no collision with any of the bots occurs.

When the bot under consideration experiences no collision for a given amount of time, the path planner algorithm realigns the bot in the direction of its target and increases its velocity.

Target checker algorithm, which is present in the top file, continuously checks whether the bots have reached their target destination. If reached, it stops the bot by writing zero velocities in the corresponding file. The targets are being set in the init module.

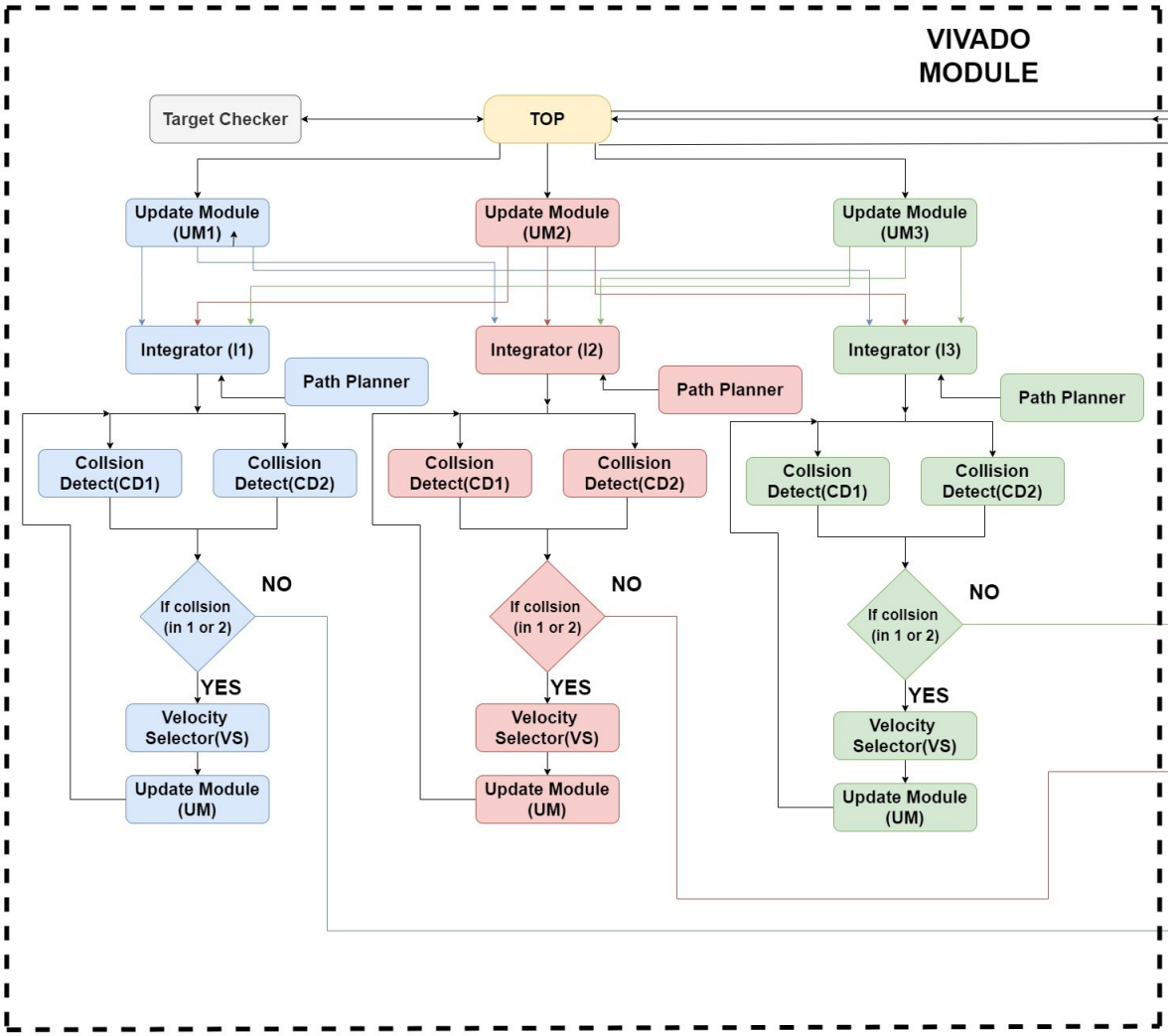


Figure 2.2: Vivado module

2.3.2 VREP modules

There are four modules implemented in the V-REP software which are kinematics, robot design selection, enable signal, and simulation settings. Since the digital circuit simulation running in Vivado gives the VX and VY values, the RPM required for both the left and right motors have to be calculated and set, this is done by the kinematics module. The bot is first turned in the direction of VX + jVY with constant magnitude and then the magnitude of speed is set as the magnitude of VX + jVY. Bot was changed from the initial proposal of an omnidirectional platform driven by a belt drive to a differential drive robot for ease of controlling.

In order to establish a one to one correspondence between one feedback write operation from VREP and a control value read operation from Vivado, a flag variable is used whose state keeps toggling between 0 and 1 and the state determines the operation. To control all the robots simultaneously, we have used threaded child scripts for each robot.

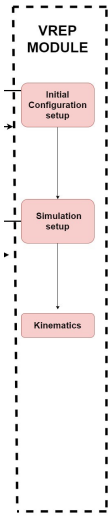


Figure 2.3: VREP module

2.3.3 Communication modules

V-rep and Vivado simulations run asynchronously. To facilitate the communication between both the simulations, handshake signals are being used. Separate modules that read data and write data are being employed in both the modules. Each bot has two text files for simulations in Vrep and Vivado to write/read data.

Write_coordinates module from V-rep writes the bot's current velocity in xy direction and coordinates in xy direction along with handshake symbol 'w' in the read text files of each bot. Read_test module in Vivado reads the same text files and on reading the handshake symbol w(for data written), it reads the rest of the file, loads the data into the variables, and writes the handshake symbol r(for data read).

Write_test module from Vivado, writes the velocities of the bot in xy directions along with the handshake symbol ‘w’ in the write text file of the concerned bot. Read_coordinates in Vrep reads the same text files and on reading the handshake symbol w(for data written), it reads the rest of the data, loads the velocities into the kinematics module, and writes handshake symbol r(for data read)

An enable signal is being used in Vrep simulation, to write only after each read operation. The next set of values is written by Vrep simulation only after the previously written values are being computed and processed by the Vivado simulation. This is being implemented mainly to account for the computation time in Vivado modules.

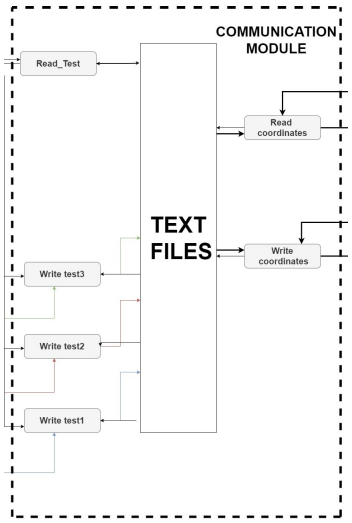


Figure 2.4: Communication module

2.4 SUMMARY

The various important terminologies used in the project were defined in this chapter. Section 2.3 contained a schematic of the project with a brief explanation on the various modules used. The different modules used in both softwares as well as the communication modules were presented in the form of a flow chart.

CHAPTER 3

METHODOLOGY

3.1 OVERVIEW

Chapter 3 contains the methodology used for implementation of the project. Section 3.2 describes the modules used in the computational software such as the update module, collision detection module and velocity selector module. The modules used in the robotic simulation software and simulation settings are mentioned in Section 3.3. The final section elucidates the communication modules that is used for synchronising Vivado and V-REP.

3.2 VIVADO MODULES

3.2.1 Update module

The velocity and position of the robot in the time before next sampling has to be calculated to give as inputs to the collision detection module. This module calculates those with vector forms of the equations of uniformly accelerated motion which are :

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

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

where,

$$\vec{r} = x\hat{i} + y\hat{j}$$

$$\vec{v} = v_x\hat{i} + v_y\hat{j}$$

$$\vec{a} = a_x\hat{i} + a_y\hat{j}$$

To implement the equations with minimal hardware, as multiple instances of this module are needed for each robot, the same multipliers and adders are used repeatedly on consecutive clock cycles to multiply/add different quantities respectively. This is shown diagrammatically in the below flow diagram of the architecture. [7]

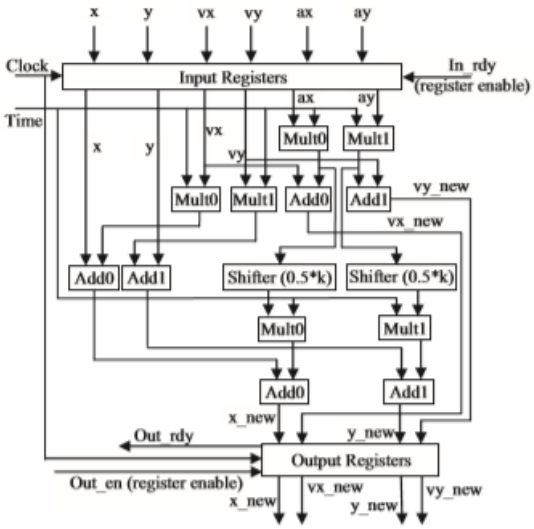


Figure 3.1: Update module architecture

The major inputs to the modules are the current position, velocity, acceleration and sampling time which are loaded to input registers in the first clock cycle after In_rdy is high. Each horizontal level in the diagram represents a clock cycle. So in the second clock cycle, Mult0 and Mult1 get inputs ax and t and ay and t respectively. The computed products are read and given to the adders Add0 and Add1 in the next clock cycle and so on. The clock period is set in such a way that the propagation delays of the modules are accounted for. The module outputs the predicted position and velocity of the robot for the next instant. Check appendix for the complete code used to implement the module. Figure 3.2 is a screenshot of the source code hierarchy of the update module.

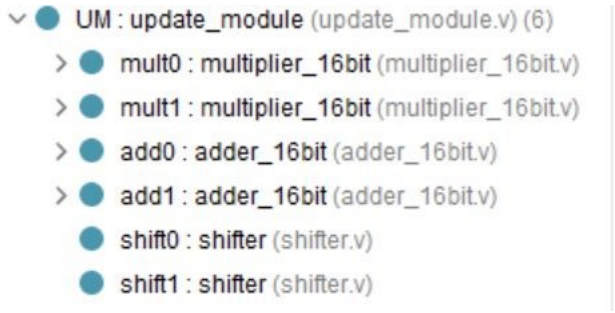


Figure 3.2: Update module source hierarchy

3.2.2 Collision detection module

This module evaluates the inequality mentioned below and predicts whether there will be a collision or not. [7] This inequality tells whether the present velocities fall inside or out of the collision cone between two robots, based on which the next task is performed. If the inequality is satisfied, the robot follows its current course

towards the target without any disruption, else Velocity Selector module is initiated, which helps the robot to avoid collision.

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

This module receives the output of the update module of two different robots as input. The inputs to this module are the velocity and position of two different robots as x and y components. 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 the below flow diagram.

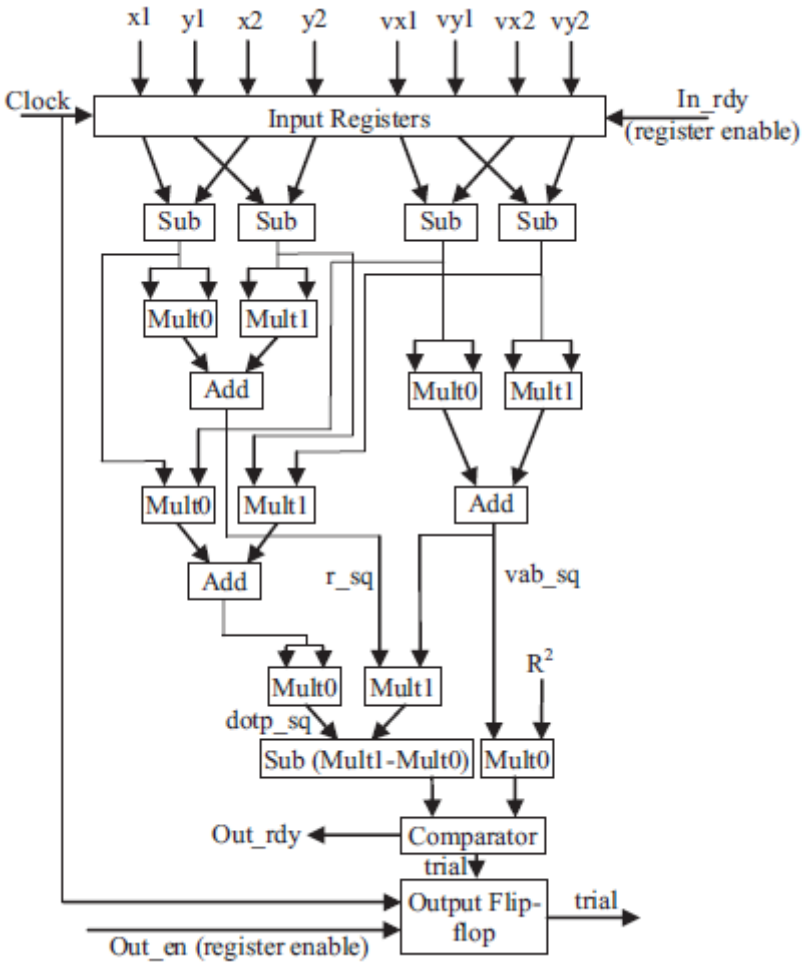


Figure 3.3: Collision detection module

Multiple copies of the same modules are instantiated to perform operations with different quantities in the upcoming instants of time. This module works in similar to the update module, with adders, multipliers being repeatedly used for the next clock cycles. This module takes ten clock cycles to complete it's one single run of operations.

The inputs are loaded in the input register when *In_rdy* is high and when the output is ready to be displayed, *Out_rdy* goes high. The output *trial* = 1 indicates collision, *trial* = 0 indicates that the path is safe. Refer appendix for the Verilog code used to design this module.

Figure 3.4: Collision detection module source hierarchy

3.2.3 Velocity selector module

Velocity selector changes the current velocity either by changing the magnitude and keeping phase constant or by changing both magnitude and phase. The rest of the Vivado modules use rectangular coordinates for processing and computations. But it is necessary for Velocity selector to access the coordinates in polar form, hence rectangular to polar conversion and vice-e-versa are implemented using CORDIC IP.

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 various range of equations based on the set configuration. Xilinx's CORDIC v6.0 IP is being used.[8]

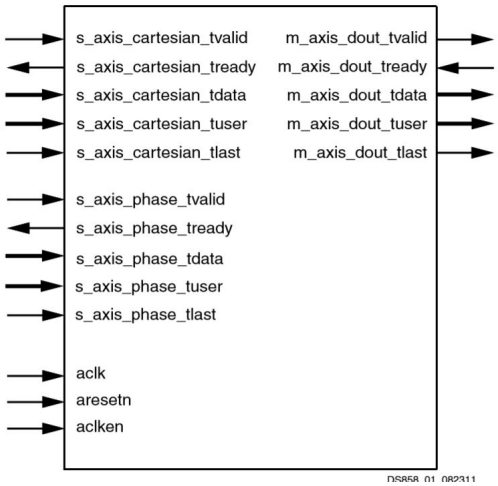


Figure 3.5: Cordic IP

The input control signals to CORDIC CORE are aclk, ACLKEN and ARESETn, signals needed to facilitate AXI4-Stream channels protocol. The channel is made up of tvalid and tdata ports. Additional ports and fields are configured depending on the function for which the CORDIC is employed. In the CORDIC core, the optional ports supported are tready, tlast and tuser. Together, tvalid and tready perform a handshake to transfer a message, where the payload is tdata, tuser and tlast. The CORDIC core operates on the operands contained in the tdata fields and outputs the result in the tdata field of the output channel.

Polar to rectangular coordinates conversion is being achieved by using the CORDIC IP in vector rotation mode, while Rectangular to Polar coordinates conversion is achieved through vector translation mode. The CORDIC IP is being configured with Coarse rotation and LUT based compensation (for reducing the effect of CORDIC scaling factor).

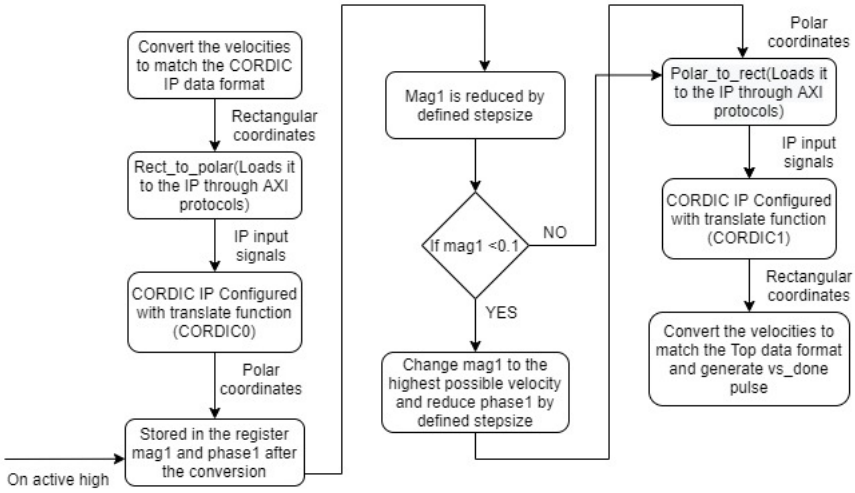


Figure 3.6: Velocity selector flowchart

The major input to the velocity selector are current velocity in rectangular coordinates, clock and active signal. Outputs are the reduced velocity in rectangular coordinates along with vs_done signal.

The Cordic input/output data format is represented by fixed-point two's-complement numbers with an integer width of 2 bits. As the rest of the module uses the fixed-point two's-complement numbers with an integer width of 5 bits, the first step involves the conversion of the input velocity to CORDIC compatible form. This is done by bitwise left shift operation on the inputs by 3.

The converted input is then loaded in the Rect_to_polar module. Rect_to_polar module takes the coordinates inputs, converts them into necessary input tdata fields and generates the control signals required for the AXI4 stream protocol, hence loading the values in the CORDIC IP configured with translation function. The output tdata fields are received in the Rect_to_polar module which then converts it back to magnitude and phase format and outputs it to the velocity selector.

On active signal being high, the magnitude and phase obtained in the previous step is loaded in the mag1 and phase1 registers. After 5 clock cycles, the mag1 is reduced by a defined step size (to be designed based on the robot's speed and torque constraints). If the mag1 is less than 0.1, then it is set back to a high value and phase1 is reduced by the step size. Mag1 and phase1 are then loaded to the Polar_to_rect module.

Polar_to_rect module similar to Rect_to_polar module takes the coordinates inputs, converts them into necessary input tdata fields and generates the control signals required for the AXI4 stream protocol, hence loading the values in the CORDIC IP configured with rotation function. The output tdata fields are received by the Polar_to_rect module which then converts it back to rectangular coordinates x_rot

and y_rot outputs it to velocity selector.

The data format is converted back to the 5-bit width integer format by first converting the x_rot and y_rot variables into real numbers and then converting the real number into a signed fixed-point two's-complement representation with 5-bit integer width. After all the processes vs_done pulse of about 10ns is being generated to mark the end of the velocity selector process. Refer Appendix for the code.

Figure 3.7: Velocity selector source code hierarchy

3.2.4 Integrator module

As shown in the block diagram in 2.2, 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 are 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 collision in a three bot system, each bot's integrator module is being 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(CD1 and CD2) detects 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 1 hence activating the CD1 and CD2 modules. After the execution of the CD1 and CD2 modules, CD1_out_rdy and CD2_out_rdy pulse are being generated by the respective CD modules. Integrator on detecting the output_rdy pulses from CD modules, deactivates the CD modules by writing 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 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. 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 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 3.8: Integrator source code hierarchy

3.2.5 Top module

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 Fig. 2.2. To read the data written by the VREP modules, the read_test module is being employed. Current velocities in xy direction and current coordinates in xy directions are read from the text files 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, Update modules are being implemented in the top rather than in the integrator. Update modules completion is marked with 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. 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

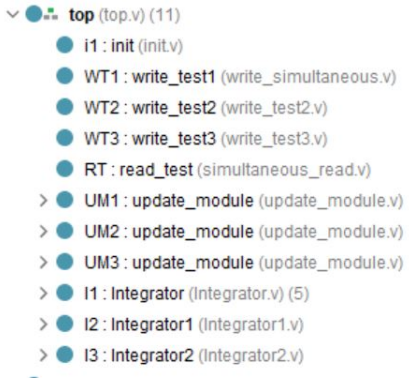


Figure 3.9: Top source code hierarchy

3.2.6 Path planning

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 simulation time in which the change in velocity of the concerned bot results in the transition from collision to no collision case is being stored in the variable time1(This is done at the end of the update module inside the integrator). The simulation time at which velocities are being loaded into the write_test modules is stored in the variable time2. (This is done in the block that starts the file-write process inside the integrator)**.

If enough time passes by with no collision cases, the velocity is changed to the maximum value in the direction of the bot's target destination. This is achieved by comparing the difference between the variables time1 and time2. If the difference is more than a few seconds(can be tuned depending on the motor speed and torque constraints) maximum velocity(which is 1m/s in this simulation) is given to the bot.

This velocity is not directly loaded. It is being passed onto the update module and then to collision detection module. Only if this velocity doesn't result in a collision, it is loaded to the write_test module.

3.2.7 Target checker

While most of the modules and work is on avoiding collision of the robots, it's not to be forgotten that the robots starting from initial positions have a destination to reach in order to carry out the task assigned to it. 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 a 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 till all the robots have reached their targets after which simulation is stopped.

3.3 V-REP MODULES

3.3.1 Kinematics

As mentioned in the proposed scheme, kinematic module comprises of the computation of the motor speed of the left and right wheels of the robot and also 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 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° , 90°) only, while the angle of target velocity vector can be in the range (-180°, 180°), based on the signs

of VX and VY , theta is modified to increase range and achieve rotation to any angle Then, required relative angle of rotation is

$$\psi = \theta - \phi$$

where, ϕ is the current orientation of the robot in global axes.

Based on whether ψ 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 the figure

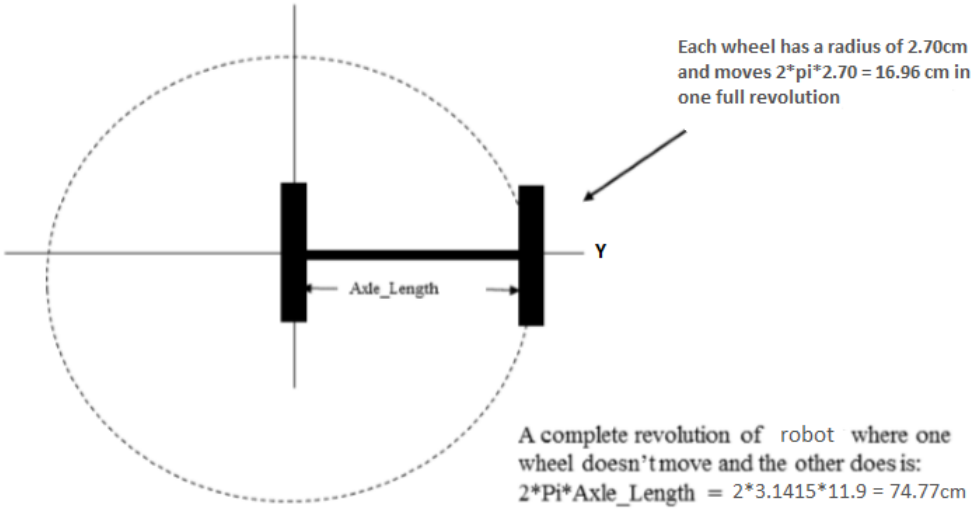


Figure 3.10: Robot turning left
[9]

In the figure, a left rotation is depicted, so the left wheel is stopped and right wheel rotated at a fixed speed (which has been chosen as 150rpm considering the rating of commercially available motors) for a time duration t, which is calculated proportional to ψ by the formula

$$t = \left| \frac{Axle\_Length * \psi}{wheel\_Radius * 15.7} \right|$$

After the rotation, magnitude of the running rate of both the wheels is given based on magnitude of target velocity.

3.3.2 Components of VREP scene

The VREP scene has 3 differential robots named as LineTracer, LineTracer0 and LineTracer1 each with an associated threaded child script and targets for each of the

robots visible as circular disks all placed on a floor of dimensions 25*25 metres.

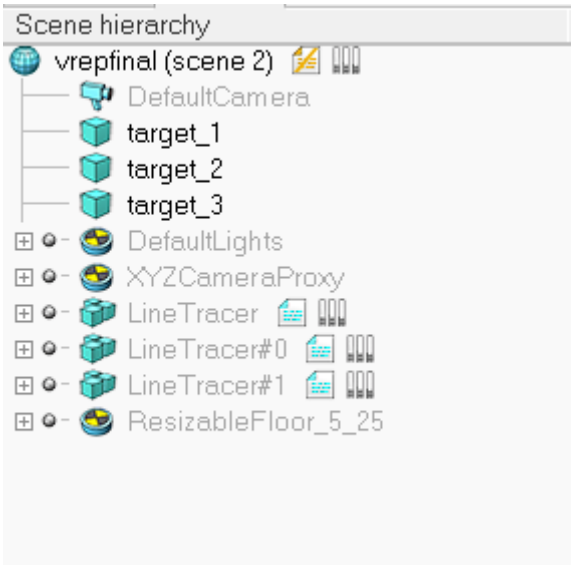


Figure 3.11: Scene hierarchy in VREP

The robots are actuated by rotating the left and right revolute joints connected to the chassis and wheels at rates calculated in the kinematics module. The body of the chassis is 11*19 cm and wheel radius is 2.7cm. While the two driven wheels are on the same axis, there is a third undriven wheel, a castor wheel to support the front portion of the robot and prevent it from falling to the floor.

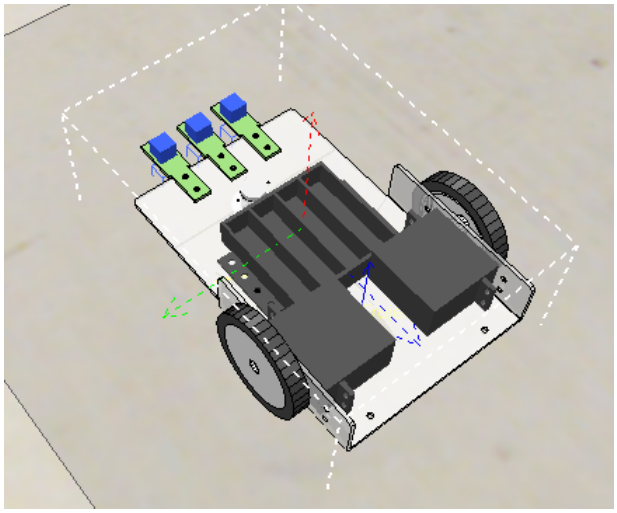


Figure 3.12: Differential drive robot model

3.3.3 Enable signal

Text files have been used as a major mode of communication between the computational and simulation softwares. Handshake signals have been used to synchronise the transfer of data from one to another. However, just using a handshake

signal causes a timing mismatch. The simulation software writes the velocities and positions of the bot as soon as Vivado reads the first textfile. The time taken for computation is not accounted for and in this time duration, there is a slight change in the position of the bot.

This issue has been resolved by the use of an enable flag within the simulation software. Once the computation has been completed, the new set of velocities is read from the text file by V-rep. A flag variable acts as an enable signal. The variable is initially set to one. Write instructions take place only when the flag is high and read takes place only when it is low. As soon as the write module is completed the flag is pulled to zero paving the way for the execution of read instructions. Similarly, once the read modules are executed the flag is pulled to one. This process accounts for any timing mismatch between the modules and also saves the computational power which will otherwise be consumed in the opening and closing of textfiles constantly. The flowchart presents the working of the enable signal.

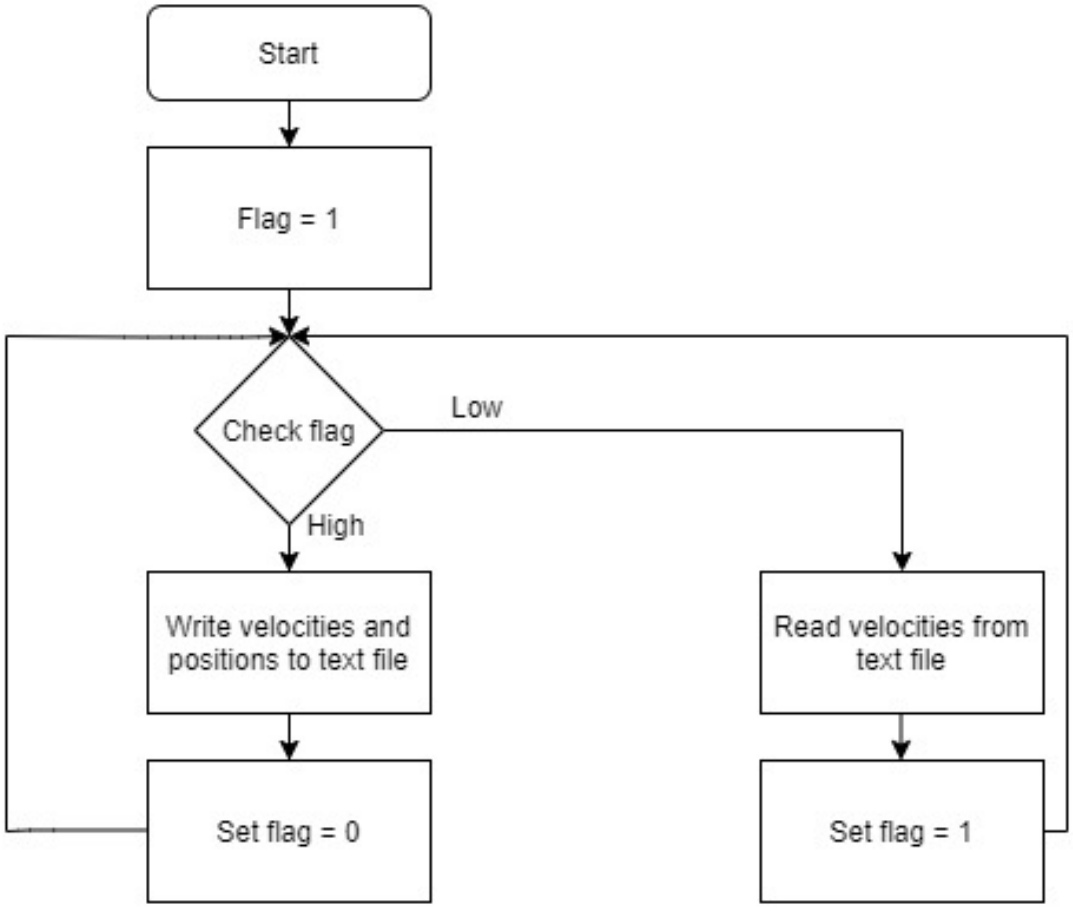


Figure 3.13: Flowchart for enable signal

3.3.4 Simulation settings

Any simulation software requires it's own simulation settings to run smoothly. This project uses simulation software V-rep. The three robots have been created as threaded child scripts in the software and the threads are switched every 100ps. The simulation was run on non-real time to match the time taken for computation. The speed has been reduced further to get a better view of the movement of robots on the arena.

3.4 COMMUNICATION MODULES

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

3.4.1 Read And write modules

The easiest way of communication between the two softwares is by using text files. This project employs the same method to pass data between the two files. 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 independent of each other for all the 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 open. 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 modules have been instantiated into the main program. Since the read module is the first module in the program it does not require any input signals. The output signals are the velocities of all the bots present in the system as well as a

read_done variable which is set to high after all the data has been read to indicate to the program to move on to the next module. The write module is in contrast to the read module. It is the last module to be instantiated in the program. The input variables consist of the updated velocities of the bots which are to be updated to the text files. There is also an output_in_rdy variable which acts as a signal to the module showing that the computations are complete and the velocities can be written. An output variable is pulled to one to indicate that the values have been written to the text files successfully.

3.4.2 Handshake signal

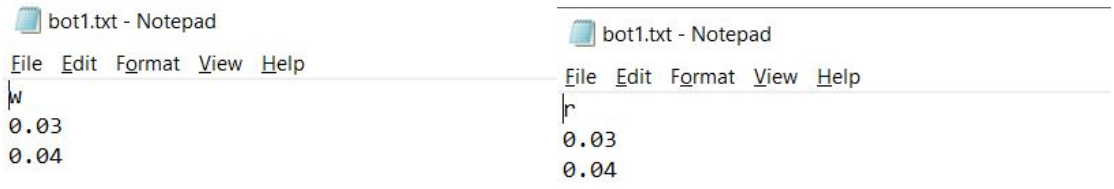
The project uses two softwares for the implementation of collision avoidance of multiple robots. These softwares, however, are not synchronised. The rate at which Vivado writes to the text file is different from the rate at which V-rep reads the same and vice-versa. Hence, a problem of asynchronisation between the two softwares 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 synchronised 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.

Figure 3.14: Velocities and positions written to text file

Figure 3.15: Velocities and positions read from text file

This project uses characters as handshake signals between the two softwares. As seen in Fig.3.14 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.



(a) Write (b) Read

Figure 3.16: Reverse handshake

The figures 3.16a and 3.16b represent the reverse handshake phenomenon between the computation and simulation softwares. 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.

3.5 SUMMARY

This chapter explains the various modules used for implementation of the project. Each section elaborates on the modules used in the two softwares as well the communication between them. The equations used to compute and detect collision

are also mentioned. The IPs used for conversions are explained. The chapter concludes with the method of synchronisation of the softwares using handhake signals.

CHAPTER 4

TESTING AND RESULTS

4.1 OVERVIEW

This chapter talks about the different levels of testing and the results 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 softwares simultaneously.

4.2 LEVEL 1: TESTING OF INDIVIDUAL MODULES

4.2.1 Vivado modules

Update module

3 sets of inputs were forced as inputs along with a high in In_rdy and the outputs of updates positions and velocities were compared with theoretical values and found to be the same. Table 4.1 shows test cases and Fig 4.1 shows the waveforms of the module obtained after simulation of test case 1

Parameters	Input Values		Expected Result		Obtained Result	
	Value	Hex Form	Value	Hex Form	Value	Hex Form
a_x	0.01	14				
a_y	0.01	14				
$t(noscaling)$	1	1				
x	2	1000	2.405	133D	2.405	133D
y	3	1800	3.405	1B3D	3.405	1B3D
v_x	0.4	333	0.41	347	0.41	347
v_y	0.4	333	0.41	347	0.41	347

Table 4.1: Test case inputs along with expected and obtained outputs

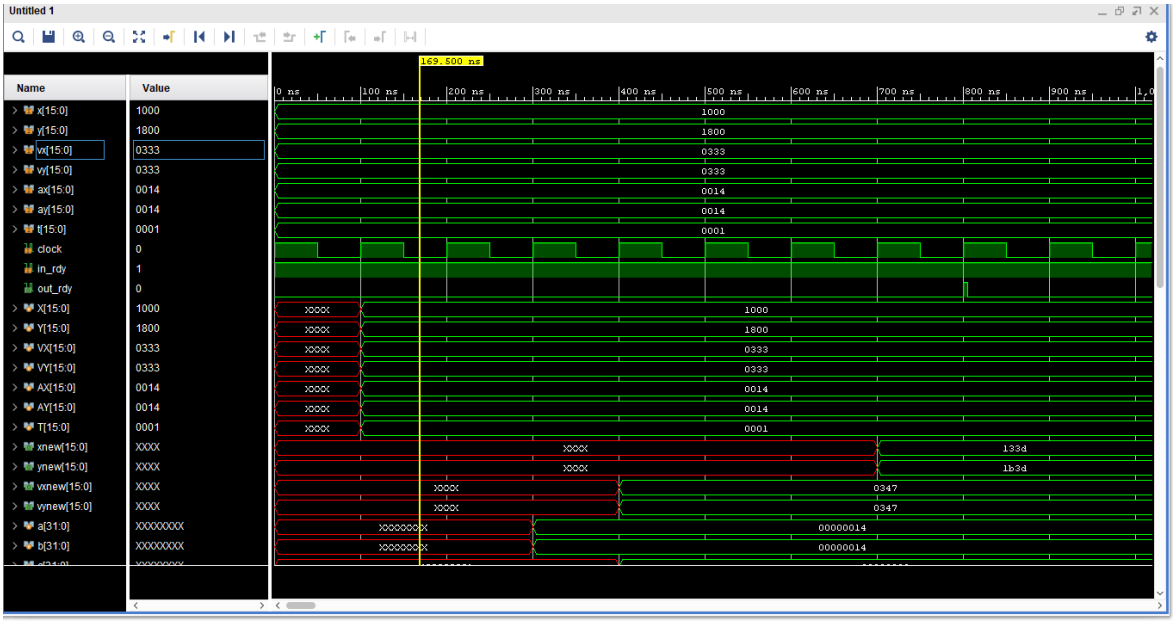


Figure 4.1: Waveform of update module input and output ports for test case 1

The out_rdy can be seen to go from 0 to 1 at the end of seven clock periods, after the output values have been loaded to output registers

Collision detection module

The inputs parameters (velocity, position) of two robots are forced, and *In_rdy* is kept high. Two different cases are forced into the module, and the output obtained in the designed module matches with the mathematical calculation performed for all the cases.

In the first case shown below, we take a case of collision between two robots.

Parameters	Bot 1		Bot 2	
	Value	Hex Form	Value	Hex Form
x	2	1000	0	0
y	2	1000	0	0
v_x	-0.2	FFFFFE67	0.2	199
v_y	-0.2	FFFFFE67	0.2	199

Table 4.2: Input values for collision detection test case 1

Parameters	Expected Result		Obtained Result	
	Value	Hex Form	Value	Hex Form
r_sq	8	2000000	8	2000000
vab_sq	0.32	147AE1	0.319	146B88
dot_sq	2.56	28F5C28B2F00	2.5525	28D710000000
LHS	0	0	0	0
RHS	0.015488	FFFFFE67	0.015442	3F40C699E0
$Collision$	Collision		Collision	

Table 4.3: Expected and obtained values for collision detection test case 1

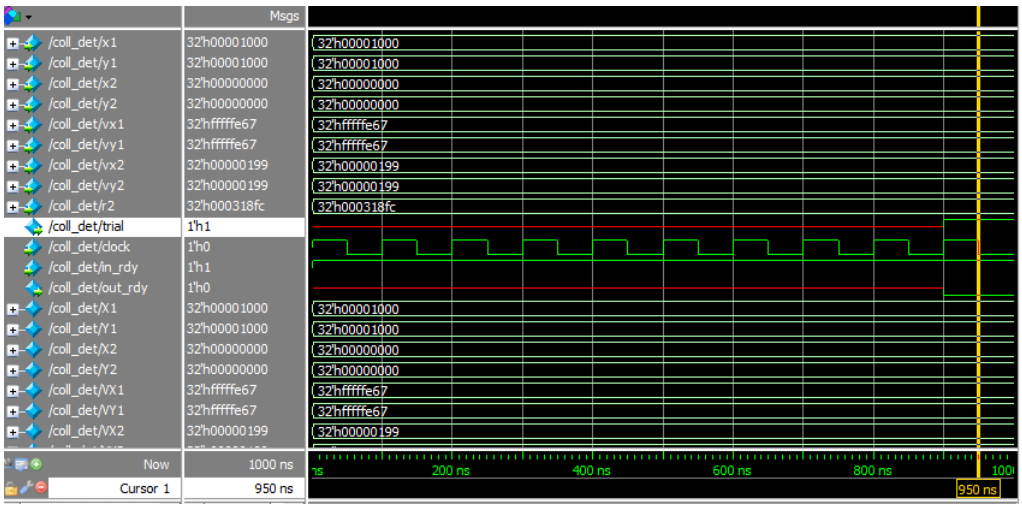


Figure 4.2: Waveform for test case 1

It can be seen that the *Out_rdy* pin goes low when the output is being generated and the output is *high* which matches with the calculation performed manually.

In the second case shown below, we take a case of no collision between two robots.

Parameters	Bot 1		Bot 2	
	Value	Hex Form	Value	Hex Form
x	0.1	CC	0.5	400
y	0.2	199	0.1	CC
v_x	0.1	CC	0.1	CC
v_y	0.4	333	0.2	199

Table 4.4: Input values for collision detection test case 2

Parameters	Expected Result		Obtained Result	
	Value	Hex Form	Value	Hex Form
r_sq	0.17	AE147	0.1703	AE6B9
vab_sq	0.04	28F5C	0.4007	290A4
dot_sq	0.0004	1A36E2EB2	0.000401	1A5123a44
LHS	0.0064	1A36E2EB1C	0.006425	1A5123a440
RHS	0.001936	7EE0B0AF5	0.001939	7F201C170
$Collision$	No Collision		No Collision	

Table 4.5: Expected and obtained values for collision detection test case 2

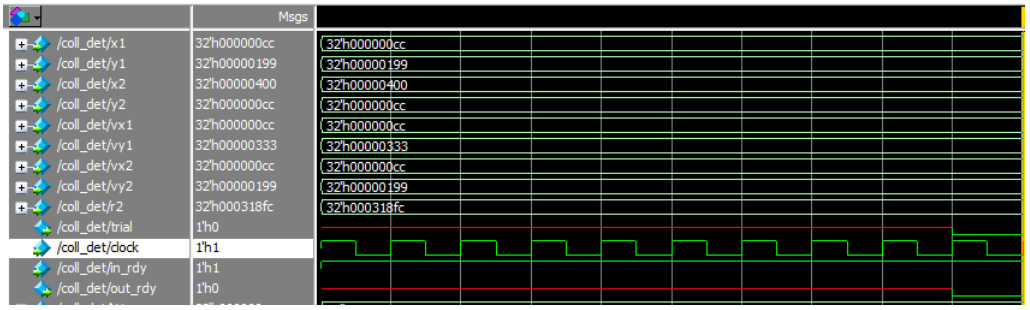


Figure 4.3: Waveform for test case 2

It can be seen that the *Out_rdy* pin goes low when the output is being generated and the output is *low* which matches with the calculation performed manually.

Velocity selector module

Figure 4.4 shows the velocity selector operation where phase is held constant and magnitude is changed.

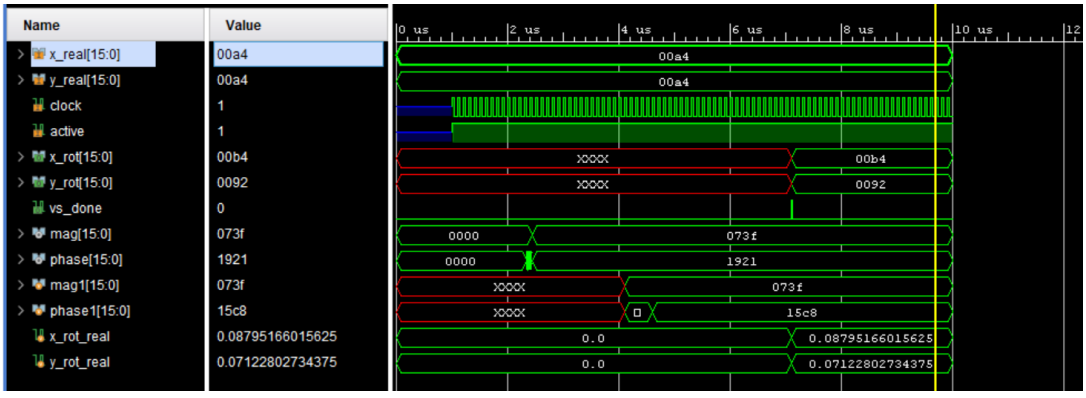


Figure 4.4: Wave output of velocity selector with constant phase and reduced magnitude

x_real , y_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 mag1 and phase1. From Figure 4.4, it is clear that the magnitude is changed while keeping phase constant(mag1 is reduced and phase1 is held constant). x_rot and y_rot are the final rectangular coordinates having mag1 and phase1 as the magnitude and phase. The end of the velocity selector module is marked with the vs_done pulse.

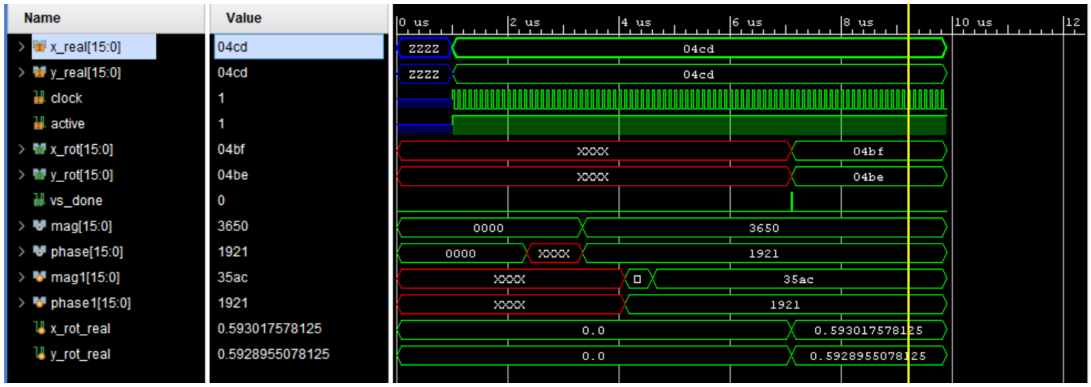


Figure 4.5: Wave output of velocity selector with constant magnitude and reduced phase

Figure 4.5 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)

Read And write modules

The reading of text files takes place as soon as the simulation begins. The positions and velocities are read from the file and necessary computations take place. Once the calculations are complete the velocities are written onto the file.

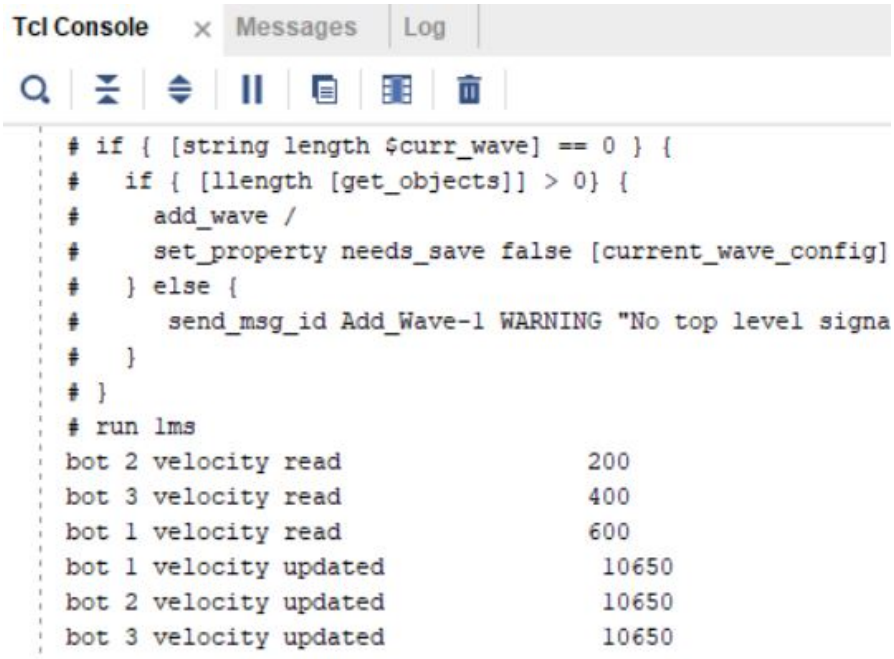


Figure 4.6: Console output of vivado read and write

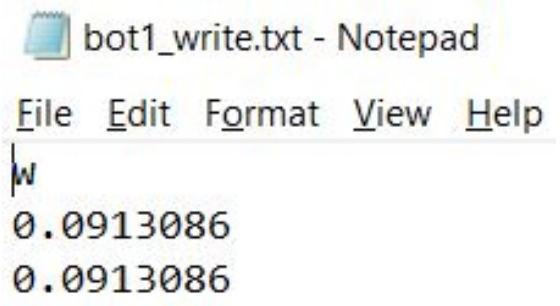


Figure 4.7: Text file after successful write

The above Fig.4.6 shows the console output after the running of the simulation. The first three statements indicate that the file has been read successfully. The next three statements show that the new values of velocity have been updated successfully to the file. The time at which the operation has taken place is also mentioned next to the statement. It can be seen that the last read takes place at 600 ns and the first write statement at 10650 ns. The time difference between them shows the time that has been utilised for computation. This also shows that the write instructions take place only after the computation has been successfully finished. The Fig.4.7 shows the text file after the values have been written.

Integration module

The following figures show the testing of the top module that involves generating signals to activate and deactivate different modules in it.

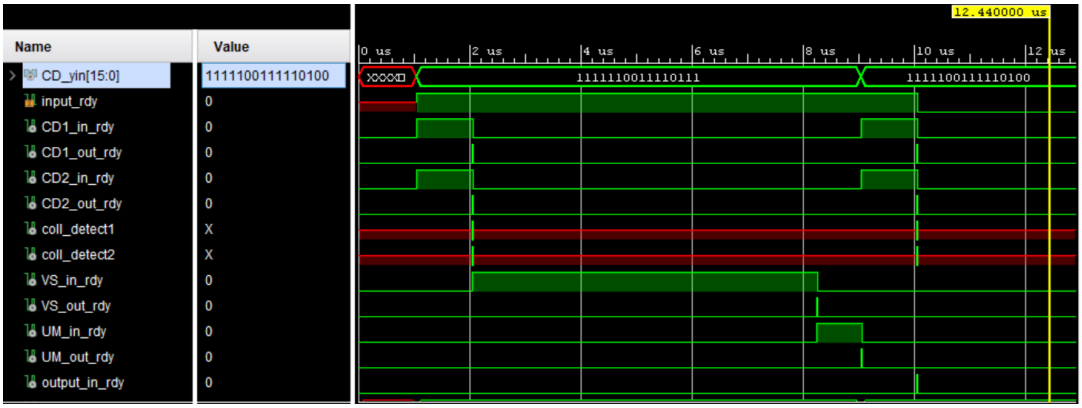


Figure 4.8: Wave output of integrator module

Figure 4.8 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 4.9 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 4.8, 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 4.10 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.

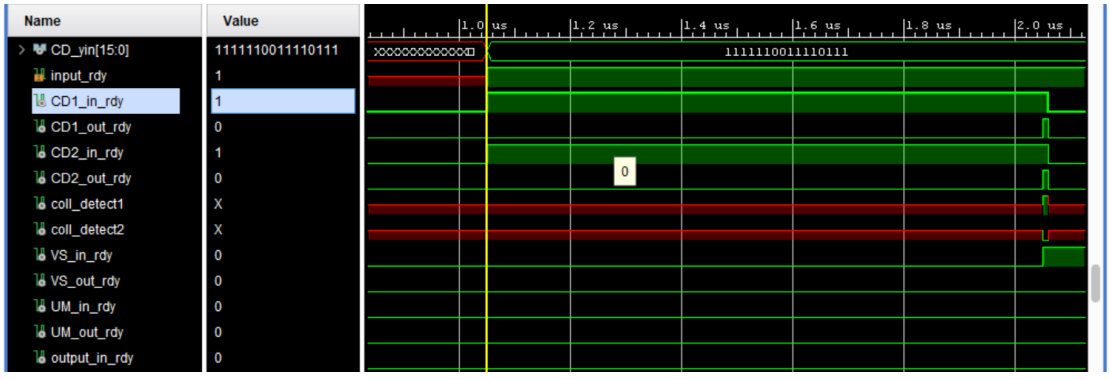


Figure 4.9: Wave output of integrator module with collision

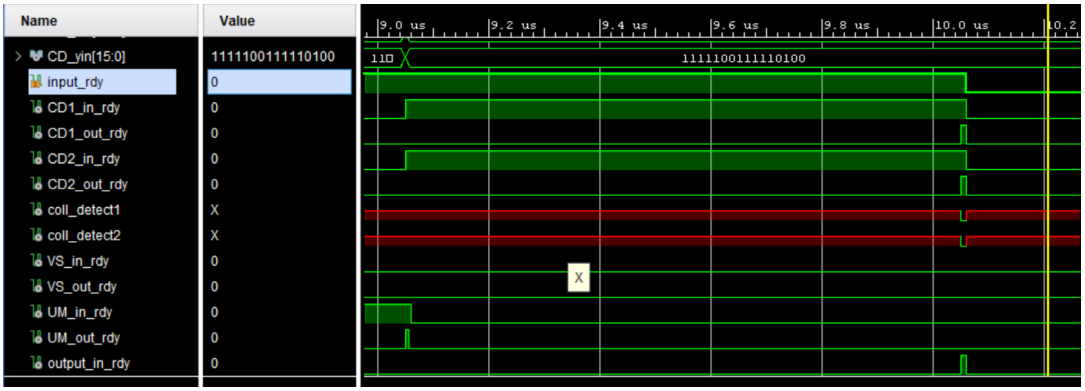


Figure 4.10: Wave output of integrator module with no collision

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

Top module

The following figures show the testing of the top module that involves generating signals to activate and deactivate different modules in it.

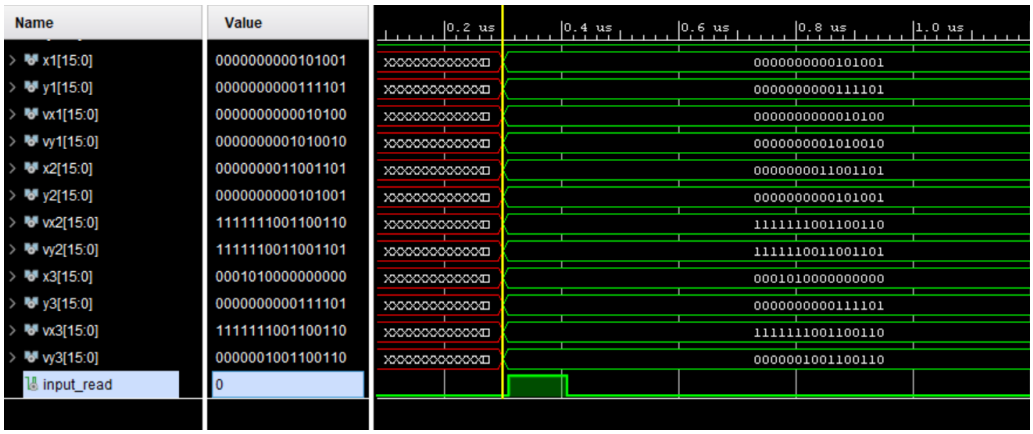


Figure 4.11: Wave output of top module showing input_read function

Figure 4.11 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.

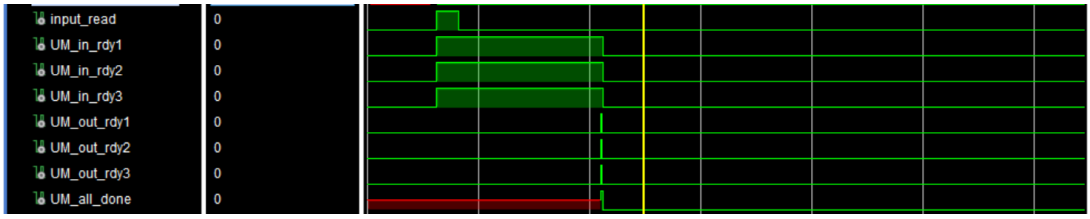


Figure 4.12: Wave output of top module showing UM activation and deactivation

Figure 4.12 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.

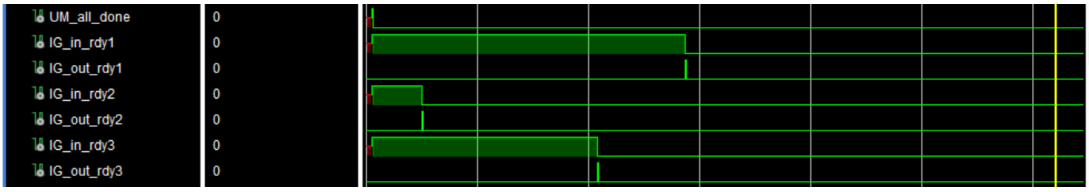


Figure 4.13: Wave output of top module showing IG activation and deactivation

Figure 4.13 shows UM_all_done pulse triggers the integrator modules, by holding IG_in_rdy1, IG_in_rdy2 and IG_rdy_3 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.

4.2.2 V-Rep Module

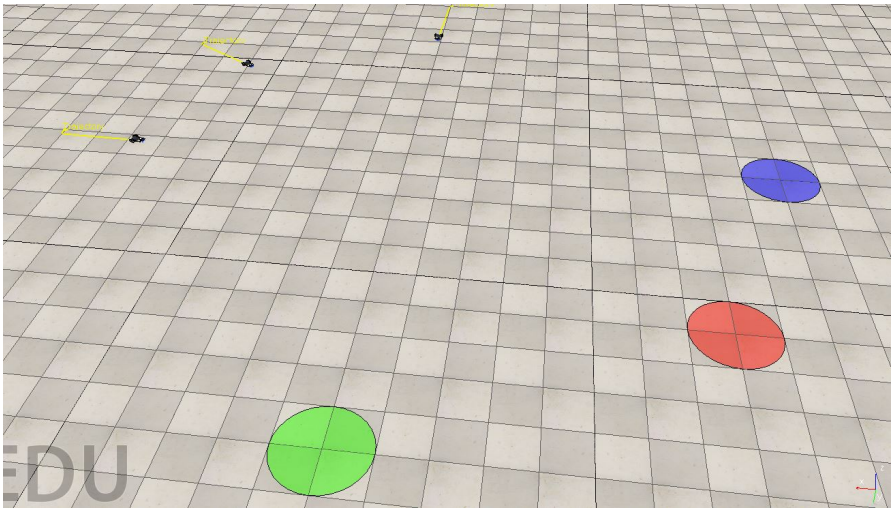


Figure 4.14: Simulation

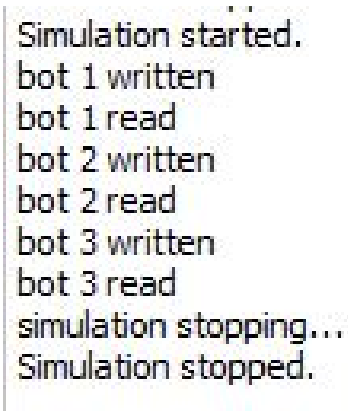


Figure 4.15: Console output of V-rep

The Fig.4.14 shows the simulation software in motion. The red, blue and green circles represent the targets of bots 1, 2 and 3 respectively. The bots start with an initial velocity which is set in the direction of the straight path to reach the target. The bots are seen moving with this velocity following the straight-line trajectory. The textfiles containing read and write have been hardcoded with the necessary information. Testing was done for basic movement of bots as well as successful reading and writing from text files. The Fig.4.15 is a snap of the console indicating the successful reading and writing of text files.

4.3 LEVEL 2: OVERALL MODULE TESTING

In this Level, we combine all the individual modules and run a overall module testing to check the integration between each different modules. Table 4.6 displays the input parameters for the 3 Robots, and Table 4.7 indicates the target values for each of the Robot.

Parameters	Bot 1		Bot 2		Bot 3	
	Value	Hex Form	Value	Hex Form	Value	Hex Form
x	1	800	0	0	0.5	400
y	1	800	0	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	FFFFD9A	0.5	400	0.3	266
a_y	-0.3	FFFFD9A	0.5	400	0.3	266
R^2	0.0484	318FC	0.0484	318FC	0.0484	318FC

Table 4.6: Input values for test case 1

Parameters	Bot 1		Bot 2		Bot 3	
	Value	Hex Form	Value	Hex Form	Value	Hex Form
x	0	0	0.5	400	1	800
y	0	0	0.5	400	1	800

Table 4.7: 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 4.8 and Table 4.9 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	
	Value	Hex Form	Value	Hex Form	Value	Hex Form
x	10	5000	9	0	-10	FFFFB000
y	1	800	8	0	-10	FFFFB000
v_x	0	0	-0.76	FFFFF9EC	0.4	333
v_y	0.36	FFFFFAF	-0.72	FFFFFA3E	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 4.8: Input values for test case 2

Parameters	Bot 1		Bot 2		Bot 3	
	Value	Hex Form	Value	Hex Form	Value	Hex Form
x	10	5000	-10	FFFFB000	0	0
y	10	5000	-10	FFFFB000	0	0

Table 4.9: Target values for test case 2

4.4 LEVEL 3: FINAL 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 vx and vy read from Vivado and printed in console log of VREP shown in figure 4.16 below

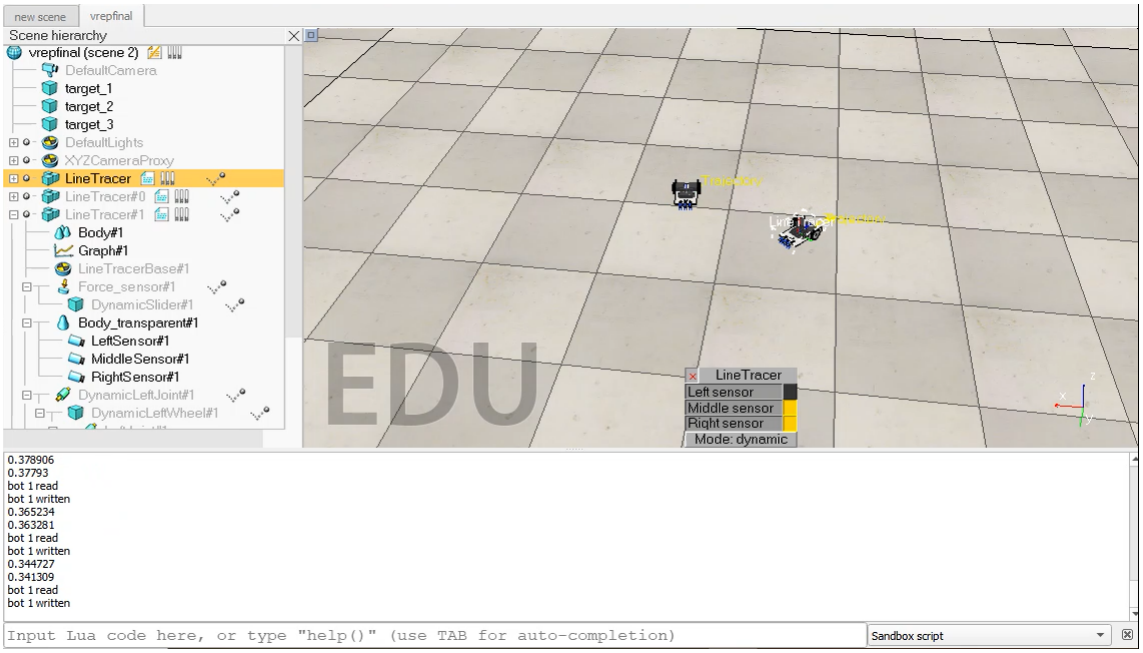


Figure 4.16: Robot1 slowing down when collision with robot2 is predicted

CASE 2 : Robot 2 has reached its target as it can be seen in the following figure 4.17. The target checker in Vivado found that the robot 2 has reached target and stopped it by writing vx and vx as zero as shown in figure 4.18 and figure 4.19 .

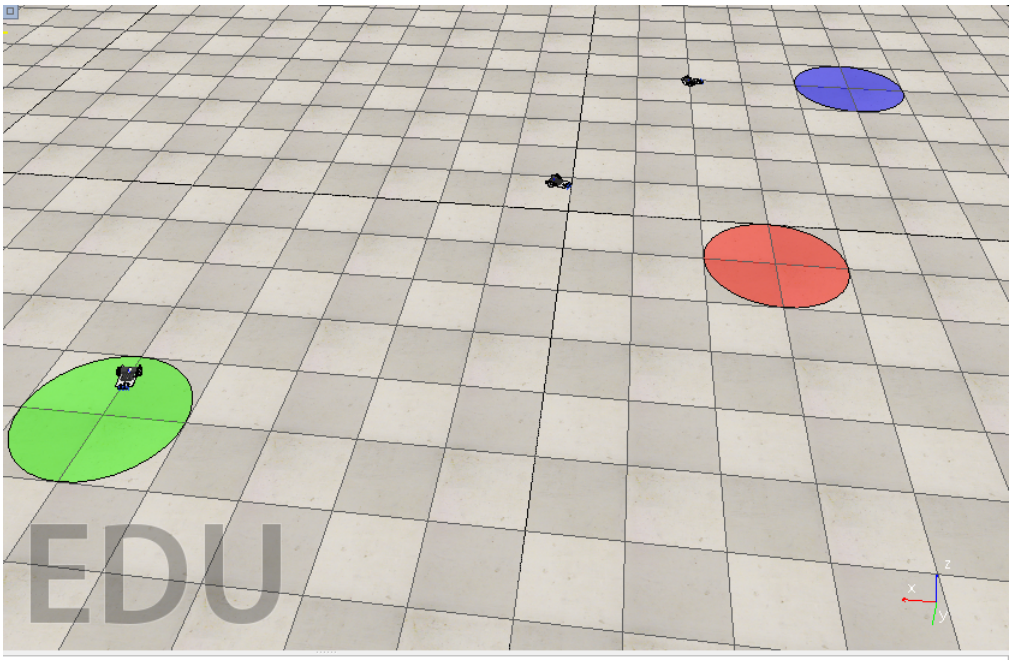


Figure 4.17: VREP scene showing robot2 in its target subspace

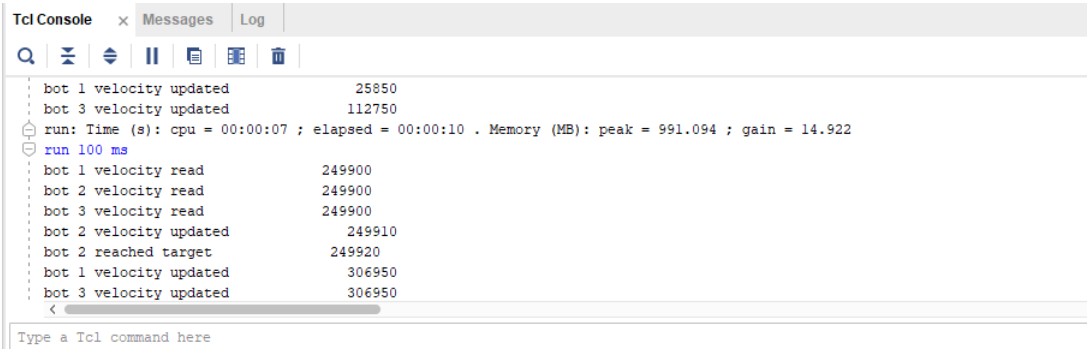


Figure 4.18: Target checker finding out that robot2 has reached target

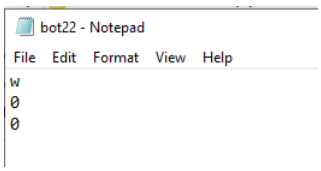
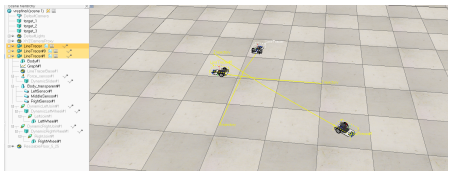
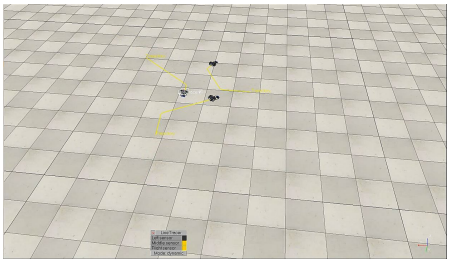


Figure 4.19: Text file showing VX and Vy written as zeros by Vivado

Fig.4.20 shows the trajectory and paths followed by the robots when there is a collison and when collision is avoided.



(a) With collision



(b) Collision avoidance

Figure 4.20: Trajectories

4.5 SUMMARY

Chapter 4 presents the consolidated results obtained through various stages of testing of the project. Waveforms showing the change in variable values are shown. Comparisons between expected and actual outpus are also presented for validation purposes. Simulation scene images show the movements and trajectories of the three bots along with their targets. Console outputs show the changes in velocities and angles in order to avoid collision.

CHAPTER 5

CONCLUSION

5.1 MAJOR CONTRIBUTIONS OF THE PROJECT

The project has implemented a novel method of Reciprocal Collision Avoidance along with Acceleration Velocity Obstacles; this implementation has been effective in reducing cost and power dissipation while deploying in FPGAs. Robotic algorithms are successfully implemented on FPGAs leading to the development of low-cost FPGA robotic systems. This implementation would be a significant contribution to the field of low-cost robotic systems.

In this project two softwares namely Vivado and V-REP have been used for computation and simulation respectively. The communication between the two softwares takes place using text files. Synchronisation has been achieved using handshake signals. Once the simulation begins, the positions and velocities of the bots are written to the respective text files. The initial velocities of the bot are set in the direction of the target. The read module in vivado reads the values and then proceeds to the update module. This module calculates the positions and velocities of the bots at the next instance using kinematics equations. These values are given to the collision detection module which then checks whether any combination of bots will collide. If there is no collision detected the bots continue to move in the direction of the target with their initial velocities. If a collision is detected, the velocity selector module is called and the bots are slowed down i.e the magnitude of velocities are decreased with constant phase. If collision can still not be avoided the directions of the bots are changed by the module until all collisions have been avoided successfully. These new velocities are then written to the 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 project 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.

5.2 SCOPE FOR FURTHER WORK

The future scope of this project is to extend the algorithm for real time scenarios. This work can be extrapolated into a hardware environment and sensors can be fixed on the robot to detect obstacles and get position and velocities of the bots. It can be programmed to avoid both static and dynamic obstacles.The future scope for architecture could be implementing the project using different architectures and performing a timing analysis by synthesising the program onto an FPGA.

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] 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.

[4] 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.

[5] Jur van den Berg, Stephen Guy, Ming Lin, and Dinesh Manocha. “Reciprocal n-Body Collision Avoidance”. In: vol. 70. Apr. 2011, pp. 3–19.

[6] 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.

[7] 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.

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

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

APPENDIX A

CODE ATTACHMENTS

A.1 READ MODULE

```
module read_test1(x1_bin,y1_bin,vx1_bin,vy1_bin,read_done1);
 output [15:0] x1_bin,y1_bin,vx1_bin,vy1_bin;
 output read_done1;

 integer i,code,k,l,m,n;
 integer file_id;
 wire y;
 integer j = 1;
 integer char1;

 real vx1, vy1, x1, y1;
 reg [15:0] x1_bin_reg,y1_bin_reg,vx1_bin_reg,vy1_bin_reg;
 reg read_done_reg1=1'b0;
 assign x1_bin    = x1_bin_reg;
 assign y1_bin    = y1_bin_reg;
 assign read_done1 = read_done_reg1;
 assign vx1_bin   = vx1_bin_reg;
 assign vy1_bin   = vy1_bin_reg;

 always
     begin
         #300 file_id = $fopen("bot1_read.txt","r");
         code=$fscanf(file_id , "%c\n",char1);
         $fclose(file_id);
         if(char1 =="w")
         begin
         file_id = $fopen("bot1_read.txt","r");
         code=$fscanf(file_id , "%c\n",char1);
         k = $fscanf(file_id , "%f\n",vx1);
         l = $fscanf(file_id , "%f\n",vy1);
         m = $fscanf(file_id , "%f\n",x1);
         n = $fscanf(file_id , "%f\n",y1);
         $fclose(file_id);
         file_id = $fopen("bot1_read.txt","w");
         $fseek(file_id , 0, 0);
         $fwrite(file_id , "r\n");
         $display("bot_1_velocity_read", $time);
         $fclose(file_id);
         end
          x1_bin_reg  <= x1  * 2**(11);
          y1_bin_reg  <= y1  * 2**(11);
          vx1_bin_reg <= vx1 * 2**(11);
          vy1_bin_reg <= vy1 * 2**(11);
 end
      always @(x1_bin or y1_bin or vx1_bin or vy1_bin)
```

```
46          begin
47                  #10;
48                  read_done_reg1=1'b1;
49                  #100;
50                  read_done_reg1=1'b0;
51              end
52  endmodule
```

A.2 UPDATE MODULE

```
1   module update_module(x, y, vx, vy, ax, ay ,t,clock, in_rdy,out_rdy,
        xnew, ynew, vxnew, vynew);
2
3   input [15:0] x,y,vx,vy,ax,ay,t;
4   output [15:0] xnew,ynew,vxnew,vynew; //confirm size of t
5   input clock,in_rdy;
6   output out_rdy;
7
8   reg [15:0] X,Y,VX,VY,AX,AY,T;      //input registers
9
10  reg [15:0] xnew,ynew,vxnew,vynew; //output registers
11
12  reg [31:0] a,b,c,d,i,j;           //intermediate registers
13
14  reg [15:0] e,f,g,h,k,l,M0A,M0B,M1A,M1B,P0,Q0,P1,Q1,SHI,SFI;
15  wire [0:10] carry;
16  integer count=0;
17  wire[31:0] M0P,M1P;
18  wire [15:0] S0,S1,SHO,SFO;
19  reg output_ready=1'b0;
20  //instantiations
21  assign out_rdy=output_ready;
22
23  multiplier_16bit mult0(M0P, M0A, M0B);
24  multiplier_16bit mult1(M1P, M1A,M1B);
25
26  adder_16bit add0(carry[0],S0,P0,Q0,1'b0);
27  adder_16bit add1(carry[1],S1,P1,Q1,1'b0);
28
29  shifter shift0(SHI,SHO);
30  shifter shift1(SFI,SFO);
31
32  always @(posedge clock)
33
34   if(in_rdy)
35    begin
36
37    case(count)
38    0: begin
39         output_ready=1'b0;
40         X<=x; Y<=y; VX<=vx; VY<=vy; AX<=ax; AY<=ay; T<=t;      //input
               registers
41       end
42
43    1:begin
```

```
44          M0A<=AX; M0B<=T;
45          M1A<=AY; M1B<=T;
46        end
47      2: begin
48          a=M0P;
49          b=M1P;
50          M0A<=VX; M0B<=T;
51          M1A<=VY; M1B<=T;
52          P0<=VX; Q0<=a;
53          P1<=VY; Q1<=b;
54        end
55      3: begin
56          c=M0P;
57          d=M1P;
58          k=S0;
59          l=S1;
60          P0<=X; Q0<=c;
61          P1<=Y; Q1<=d;
62          SHI=a;
63          SFI=b;
64          vxnew=k;
65          vynew=l;
66        end
67      4: begin
68          g=SHO;
69          h=SFO;
70          e=S0;
71          f=S1;
72          M0A<=g; M0B=T;
73          M1A<=h; M1B=T;
74        end
75      5: begin
76          i=M0P;
77          j=M1P;
78          P0<=e; Q0<=i;
79          P1<=f; Q1<=j;
80        end
81      6: begin
82          xnew=S0;
83          ynew=S1;          //out_rdy signal is raised once all outputs
                are ready
84        end
85      7: begin
86            count=-1;              //so that after updation it goes to 0
87            output_ready=1'b1;
88            #5;                    //Output pulse
89            output_ready=1'b0;
90        end
91    endcase
92  count=count+1;
93  end
94
95 endmodule
```

A.3 COLLISION DETECTION MODULE

```
1  module coll_det(x1, y1, x2, y2, vx1, vy1, vx2, vy2, r2, trial, clock,
       in_rdy,out_rdy);
2
3  input [15:0] x1, y1, x2, y2, vx1, vy1, vx2, vy2;
4  input [31:0] r2;
5  output trial;
6  input clock,in_rdy;
7  output out_rdy;
8
9  reg [31:0] X1,Y1,X2,Y2,VX1,VY1,VX2,VY2,R2;      //input registers
10 reg trial; //output registers
11 reg [31:0] e,f,g,h,i,j;
12 reg [63:0] dot_sq,l,m,n;                        //intermediate registers
13 reg [31:0] a,b,c,d,r_sq,vab_sq,k,M0A,M0B,M1A,M1B,P0,Q0,P1,Q1,P2,Q2,P3
       ,Q3,P4,Q4,P5,Q5,P6,Q6;
14 reg [63:0] C1,C2,P7,Q7;
15 wire [0:10] carry;
16 integer count=0;
17 wire C0;
18 wire[63:0] M0P,M1P,S7;
19 wire [31:0] S0,S1,S2,S3,S4,S5,S6;
20 reg output_ready=1'b0;
21 assign out_rdy=output_ready;
22 //instantiations
23
24 multiplier_32bit mult0(M0P, M0A, M0B);
25 multiplier_32bit mult1(M1P, M1A, M1B);
26
27 adder_32bit add0(carry[0],S0,P0,Q0,1'b0);
28 adder_32bit add1(carry[1],S1,P1,Q1,1'b0);
29 adder_32bit add2(carry[2],S2,P2,Q2,1'b0);
30
31 subtractor_32bit sub0(carry[3],S3,P3,Q3,1'b1);
32 subtractor_32bit sub1(carry[4],S4,P4,Q4,1'b1);
33 subtractor_32bit sub2(carry[5],S5,P5,Q5,1'b1);
34 subtractor_32bit sub3(carry[6],S6,P6,Q6,1'b1);
35 subtractor_64bit sub4(carry[7],S7,P7,Q7,1'b1);
36 comparator comp1(C0,C1,C2);
37
38 always @(posedge clock)
39
40  if(in_rdy)
41   begin
42
43   case(count)
44   0: begin
45     if(x1[15])
46       begin
47       X1[31:16]<=16'b1111111111111111;
48       X1[15:0]<=x1;
49       end
50       else
51       begin
```

```
52          X1[31:16]<=16'b0;
53          X1[15:0]<=x1;
54          end
55
56          if(y1[15])
57          begin
58          Y1[31:16]<=16'b1111111111111111;
59          Y1[15:0]<=y1;
60          end
61          else
62          begin
63          Y1[31:16]<=16'b0;
64          Y1[15:0]<=y1;
65          end
66
67          if(x2[15])
68          begin
69          X2[31:16]<=16'b1111111111111111;
70          X2[15:0]<=x2;
71          end
72          else
73          begin
74          X2[31:16]<=16'b0;
75          X2[15:0]<=x2;
76          end
77
78          if(y2[15])
79          begin
80          Y2[31:16]<=16'b1111111111111111;
81          Y2[15:0]<=y2;
82          end
83          else
84          begin
85          Y2[31:16]<=16'b0;
86          Y2[15:0]<=y2;
87          end
88
89          if(vx1[15])
90          begin
91          VX1[31:16]<=16'b1111111111111111;
92          VX1[15:0]<=vx1;
93          end
94          else
95          begin
96          VX1[31:16]<=16'b0;
97          VX1[15:0]<=vx1;
98          end
99
100         if(vy1[15])
101         begin
102         VY1[31:16]<=16'b1111111111111111;
103         VY1[15:0]<=vy1;
104         end
105         else
106         begin
107         VY1[31:16]<=16'b0;
```

```
108         VY1[15:0]<=vy1;
109         end
110
111         if(vx2[15])
112         begin
113         VX2[31:16]<=16'b1111111111111111;
114         VX2[15:0]<=vx2;
115         end
116         else
117         begin
118         VX2[31:16]<=16'b0;
119         VX2[15:0]<=vx2;
120         end
121
122         if(vy2[15])
123         begin
124         VY2[31:16]<=16'b1111111111111111;
125         VY2[15:0]<=vy2;
126         end
127         else
128         begin
129         VY2[31:16]<=16'b0;
130         VY2[15:0]<=vy2;
131         end
132         R2<=r2;       //input registers getting their values
133
134       end
135
136     1:begin
137       P3<=X1; Q3<=X2;
138       P4<=Y1; Q4<=Y2;
139       P5<=VX1; Q5<=VX2;
140       P6<=VY1; Q6<=VY2;
141     end
142
143     2:begin
144       a=S3;
145       b=S4;
146       c=S5;
147       d=S6;
148
149       M0A<=a; M0B<=a;
150       M1A<=b; M1B<=b;
151     end
152
153     3:begin
154       e=M0P;
155       f=M1P;
156       M0A<=c; M0B<=c;
157       M1A<=d; M1B<=d;
158     end
159
160      4:begin
161       g=M0P;
162       h=M1P;
163       M0A<=a; M0B<=c;
```

```
164         M1A<=b; M1B<=d;
165         end
166
167       5:begin
168         i=M0P;
169         j=M1P;
170         P0<=e; Q0<=f;
171         P1<=g; Q1<=h;
172         P2<=i; Q2<=j;
173        end
174
175       6:begin
176         r_sq=S0;
177         vab_sq=S1;
178         k=S2;
179         M0A<=k; M0B<=k;
180         M1A<=r_sq; M1B<=vab_sq;
181        end
182
183       7:begin
184         dot_sq=M0P;
185         l=M1P;
186         P7<=l; Q7<=dot_sq;
187         M0A<=vab_sq; M0B<=R2;
188        end
189
190       8:begin
191        if(S7<64'd17592186040)
192         begin
193         m=r_sq;
194         n=R2;
195         C1<=m; C2<=n;
196         end
197
198         else
199         begin
200         m=S7;
201         n=M0P;
202         C1<=m; C2<=n;
203         end
204        end
205       9:begin
206             trial=C0;
207             count=-1;                //so that after updation it goes to 0
208             output_ready=1'b1;       //out_rdy signal is raised once
                    all outputs are ready
209             #100;
210             output_ready=1'b0;  //Should be given as a pulse
211             trial =1'bX;        //Same with the trial case :P
212        end
213
214     endcase
215    count=count+1;
216    end
217
218  endmodule
```

A.4 VELOCITY SELECTOR MODULE

```
1  module Velocity_selector(x_real,y_real,clock,active,x_rot,y_rot,
       vs_done);
2
3      input  [15:0]x_real;
4      input  [15:0]y_real;
5      input  clock;
6      input  active;
7      output [15:0]x_rot;
8      output [15:0]y_rot;
9      output vs_done;
10
11     wire [15:0]theta_real;
12     wire [15:0]mag_real;
13     wire [15:0]mag;
14     wire [15:0]phase;
15     reg   rtp_in_rdy,ptr_in_rdy;
16     wire  rtp_out_rdy1,ptr_out_rdy1;
17     reg   set;
18     wire  rtp_out_rdy,ptr_out_rdy;
19     reg  [15:0]mag1;
20     reg  [15:0]phase1;
21     wire [15:0] x_real_nm,y_real_nm,x_rot_nm,y_rot_nm;
22     reg  [15:0]step=16'd1638;
23     reg  [15:0]stepphase=16'd6430;
24     reg  done=1'b0;
25     reg [15:0] x_rot_reg,y_rot_reg;
26     integer count=-1;
27
28     Rect_to_polar RP(x_real_nm,y_real_nm,mag,phase,rtp_in_rdy,
         rtp_out_rdy1);
29     Polar_to_rect PR(mag_real,theta_real,x_rot_nm,y_rot_nm,ptr_in_rdy
         ,ptr_out_rdy1);
30
31     assign rtp_out_rdy = rtp_out_rdy1 & set;
32     assign ptr_out_rdy = ptr_out_rdy1 & set;
33
34     assign vs_done=done;
35     assign mag_real=mag1;
36     assign theta_real=phase1;
37
38     assign x_real_nm =(x_real<<3);        //To make it compatible with
           overall Fixed point representation
39     assign y_real_nm =(y_real<<3);
40     assign x_rot     =x_rot_reg;
41     assign y_rot     =y_rot_reg;
42     real x_rot_real,y_rot_real;
43
44     always @(posedge active)
45     begin
46     set=1'b1;
47     rtp_in_rdy =1'b1;
48     end
49
```

```
50
51      always @(posedge rtp_out_rdy)
52      if(active)
53      begin
54          #10;
55          rtp_in_rdy =1'b0;
56      end
57
58      always @(posedge ptr_out_rdy)
59      if(active)
60      begin
61          #10;
62          ptr_in_rdy=1'b0;
63       end
64
65       always @(negedge ptr_out_rdy)
66            begin
67            if(active)
68            begin
69                #10;
70                x_rot_real=$signed(x_rot_nm)*(1.0/((2**14)*1.0));//For
                      signed conditions, converting it to real and then to
                      binary should be done rather than shifting
71                y_rot_real=$signed(y_rot_nm)*(1.0/((2**14)*1.0));
72                x_rot_reg=x_rot_real*(2**11);
73                y_rot_reg=y_rot_real*(2**11);
74                done=1'b1;
75                set=1'b0;
76                #100;
77                done=1'b0;
78            end
79            end
80
81       always @(negedge rtp_out_rdy)
82            if(active)
83            begin
84            #10;
85            mag1=mag;
86                phase1=phase;
87                ptr_in_rdy=1'b1;
88                mag1=mag1-step;
89                if(mag1<16'd3277)
90                begin
91                    phase1=phase1-stepphase;
92                    mag1=16'd16384;
93                end
94             end
95  endmodule
```

A.5 INTEGRATOR MODULE

```
1   module Integrator(tx1,ty1,x1_not_update,y1_not_update, vx_not_update,
        vy_not_update,x1,y1,vx1,vy1,x2,y2,vx2,vy2,x3,y3,vx3,vy3,ax,ay,
        input_rdy,clock,ig_done);
2
```

```
3  input [15:0] tx1,ty1,x1_not_update,y1_not_update,vx_not_update,
       vy_not_update,x1,y1,vx1,vy1,x2,y2,vx2,vx3,vy2,x3,y3,vy3,ax,ay;
4  input input_rdy,clock;
5  output ig_done;
6
7  reg ig_done_reg=1'b0;
8  assign ig_done=ig_done_reg;
9  reg [15:0] count1=0;
10 reg [15:0] diffx,diffy;
11 wire [15:0] t;
12 wire [31:0] R;
13 reg [31:0] R_reg=32'd203004;
14
15 reg [15:0] vx1_WT,vy1_WT;
16 wire [15:0] WT_vx1,WT_vy1;
17
18 reg input_CD1=1'b0;
19 reg input_CD2=1'b0;
20 reg input_UM=1'b0;
21 reg input_VS=1'b0;
22
23 reg output_check_reg=1'b0;
24 reg input_check_reg=1'b1;
25
26 wire output_in_rdy;
27 wire output_written;
28 wire input_in_rdy;
29
30 reg vs_ip_sel=1'b0;// One from file else from UM
31
32 //Update module Variables
33 wire [15:0] UM_Vxin,UM_Vyin,xnew,ynew,vxnew,vynew;
34 wire UM_in_rdy,UM_out_rdy;
35 reg [15:0] Vxnew_UM,Vynew_UM;
36 update_module UM (x1, y1, UM_Vxin, UM_Vyin, ax, ay ,t,clock,
       UM_in_rdy,UM_out_rdy,xnew, ynew, vxnew, vynew);
37
38 //Velocity Selector Variables
39 wire [15:0] VS_Vxin,VS_Vyin,VS_Vxout,VS_Vyout;
40 wire VS_in_rdy,VS_out_rdy;
41 reg  [15:0] Vxnew_VS,Vynew_VS;
42 Velocity_selector VS (VS_Vxin,VS_Vyin,clock,VS_in_rdy,VS_Vxout,
       VS_Vyout,VS_out_rdy);
43
44 //Update module Variables
45 wire [15:0] CD_Vxin,CD_Vyin,CD_xin,CD_yin;
46 wire CD1_in_rdy,CD1_out_rdy;
47 wire CD2_in_rdy,CD2_out_rdy;
48 reg [15:0] Vxnew_CD,Vynew_CD,xnew_CD,ynew_CD;
49 wire coll_detect1,coll_detect2;
50 coll_det CD1(CD_xin,CD_yin,x2, y2, CD_Vxin, CD_Vyin, vx2, vy2, R,
       coll_detect1, clock, CD1_in_rdy, CD1_out_rdy);
51 coll_det CD2(CD_xin,CD_yin,x3, y3, CD_Vxin, CD_Vyin, vx3, vy3, R,
       coll_detect2, clock, CD2_in_rdy, CD2_out_rdy);
52
53 //Write Test for bot1
```

```
54  write_test1 WT(WT_vx1,WT_vy1,output_in_rdy,output_written);
55
56  assign R     = R_reg;
57  assign t = 16'd1;          //Time should not be scaled :) for LHS and
       RHS being okay.
58
59  assign CD_xin=xnew_CD;
60  assign CD_yin=ynew_CD;
61  assign CD_Vxin=Vxnew_CD;
62  assign CD_Vyin=Vynew_CD;
63
64  assign UM_Vxin=Vxnew_UM;
65  assign UM_Vyin=Vynew_UM;
66
67  assign VS_Vxin=Vxnew_VS;
68  assign VS_Vyin=Vynew_VS;
69
70  assign VS_in_rdy     = input_VS;
71  assign UM_in_rdy     = input_UM;
72  assign CD1_in_rdy     = input_CD1;
73  assign CD2_in_rdy     = input_CD2;
74
75  assign output_in_rdy =  output_check_reg;
76  assign input_in_rdy  =  input_check_reg;
77
78  assign WT_vx1        =  vx1_WT;
79  assign WT_vy1        =  vy1_WT;
80
81  always @(posedge input_rdy)   //Make input_check high for writing
       into UM from file
82  begin
83          input_CD1 = 1'b1;
84          input_CD2 = 1'b1;
85          Vxnew_CD  =  vx1;
86          Vynew_CD  =  vy1;
87          xnew_CD   =  x1;
88          ynew_CD   =  y1;
89          vs_ip_sel = 1'b0; //from file
90
91  end
92  always @(posedge CD1_out_rdy)
93  begin
94      #10;
95      input_CD1 = 1'b0;
96  end
97  always @(posedge CD2_out_rdy)
98  begin
99      #10;
100     input_CD2 = 1'b0;
101 end
102
103 always @(posedge UM_out_rdy)
104 begin
105     input_CD1 = 1'b1;
106     input_CD2=  1'b1;
107     xnew_CD  = xnew;
```

```
108       ynew_CD  = ynew;
109       Vxnew_CD = vxnew;
110       Vynew_CD = vynew;
111       #10;
112       input_UM = 1'b0;
113   end
114
115   always @(coll_detect1 | coll_detect2 )   //Load only when collision
          exist
116   begin
117       if((coll_detect1==1)||(coll_detect2 ==1))
118           begin
119           count1=1'b0;
120           input_VS = 1'b1;
121           case(vs_ip_sel)
122               0:begin
123                   Vxnew_VS = vx_not_update;
124                   Vynew_VS = vy_not_update;
125                 end
126               1:begin
127                   Vxnew_VS  = UM_Vxin;
128                   Vynew_VS  = UM_Vyin;
129                 end
130           endcase
131       end
132
133       if((coll_detect1==0)&&(coll_detect2 ==0))
134       begin
135           count1=count1+1'd1;
136           output_check_reg=1'b1;
137           $display("count1 : ", count1);
138           if(count1>=4'd8)
139           begin
140               diffx = tx1-x1_not_update;
141               diffy = ty1-y1_not_update;
142               if ($signed(diffx)<$signed(16'd2048)&&$signed(diffx)>
                      $signed(-16'd2048)&&$signed(diffy)<$signed(16'd2048)&&
                      $signed(diffy)>$signed(-16'd2048))
143               begin
144               vx1_WT  =$signed(diffx)*(1.0);
145               vy1_WT  =$signed(diffy)*(2.0);
146               end
147               else
148               begin
149               vx1_WT  =$signed(diffx)*(1.0/(10.0));
150               vy1_WT =$signed(diffy)*(1.0/(10.0));
151               end
152               $display("No collision for long so moving towars target",
                       $time);
153           end
154           else
155           begin
156               case(vs_ip_sel)
157                   0:begin
158                       vx1_WT = vx_not_update;
159                       vy1_WT = vy_not_update;
```

```
                    end
                  1:begin
                      vx1_WT   = UM_Vxin;
                      vy1_WT   = UM_Vyin;
                    end
              endcase
          end
          #10;
          output_check_reg=1'b0;
          ig_done_reg=1'b1;
          #10;
          ig_done_reg=1'b0;
      end
end

always @(posedge VS_out_rdy)
begin
      input_UM = 1'b1;
      Vxnew_UM = VS_Vxout;
      Vynew_UM = VS_Vyout;
      vs_ip_sel= 1'b1;
      #10;                          //Delay necessary to copy the outputs
          before deactivating the module
      input_VS = 1'b0;          //To turn the module of once output is
          ready(simillarly with the rest);
end

endmodule
```

A.6 TOP MODULE

```
module top;

wire clock;
reg clock_reg;
assign clock=clock_reg;
reg[15:0] diff=0;
wire [15:0] tx1,ty1,tx2,ty2,tx3,ty3;// target co-ordintes

init i1(tx1,ty1,tx2,ty2,tx3,ty3);

wire [15:0] x1,y1,vx1,vy1,t,x2,y2,vx2,vy2,x3,y3,vx3,vy3;

reg [15:0] ax1,ay1,ax2,ay2,ax3,ay3;

reg [15:0] WT_vx1,WT_vy1,WT_vx2,WT_vy2,WT_vx3,WT_vy3;
reg wt1=1'b0,wt2=1'b0,wt3=1'b0;
wire stopped1,stopped2,stopped3;

write_test1 WT1(WT_vx1,WT_vy1,wt1,stopped1);
write_test2 WT2(WT_vx2,WT_vy2,wt2,stopped2);
write_test3 WT3(WT_vx3,WT_vy3,wt3,stopped3);


//Read test
```

```
25 wire input_read1,input_read2,input_read3;
26
27
28 //read_test RT(x1,y1,vx1,vy1,x2,y2,vx2,vy2,x3,y3,vx3,vy3,input_read);
29 read_test1 RT1(x1,y1,vx1,vy1,input_read1);
30 read_test2 RT2(x2,y2,vx2,vy2,input_read2);
31 read_test3 RT3(x3,y3,vx3,vy3,input_read3);
32
33 reg flag1=1'b0,flag2=1'b0,flag3=1'b0;//target reached flags set to 0
       initially
34
35 assign t = 16'd1;
36
37 //UM1 variables and instantiations
38 wire [15:0] xnew1,ynew1,vxnew1,vynew1;
39 wire UM_out_rdy1;
40 reg [15:0] Vxnew_UM1,Vynew_UM1;
41 wire UM_in_rdy1;
42 reg input_UM1 =1'b0;
43 assign UM_in_rdy1= input_UM1;
44 /*assign UM_Vxin1  = Vxnew_UM1;
45 assign UM_Vyin1  = Vynew_UM1;*/
46 update_module UM1 (x1, y1, Vxnew_UM1, Vynew_UM1, ax1, ay1 ,t,clock,
       UM_in_rdy1,UM_out_rdy1,xnew1, ynew1, vxnew1, vynew1);
47
48 //UM2 variables and instantiations
49 wire [15:0] xnew2,ynew2,vxnew2,vynew2;
50 wire UM_out_rdy2;
51 reg [15:0] Vxnew_UM2,Vynew_UM2;
52 wire UM_in_rdy2;
53 reg input_UM2 =1'b0;
54 assign UM_in_rdy2 = input_UM2;
55 /*assign UM_Vxin2   = Vxnew_UM2;
56 assign UM_Vyin2   = Vynew_UM2;*/
57 update_module UM2 (x2, y2, Vxnew_UM2,Vynew_UM2, ax2, ay2 ,t,clock,
       UM_in_rdy2,UM_out_rdy2,xnew2, ynew2, vxnew2, vynew2);
58
59 //UM3 variables and instantiations
60 wire [15:0] xnew3,ynew3,vxnew3,vynew3;
61 wire UM_out_rdy3;
62 reg [15:0] Vxnew_UM3,Vynew_UM3;
63 wire UM_in_rdy3;
64 reg input_UM3 =1'b0;
65 assign UM_in_rdy3 = input_UM3;
66 /*assign UM_Vxin3  =Vxnew_UM3;
67 assign UM_Vyin3 = Vynew_UM3;*/
68 update_module UM3 (x3, y3,Vxnew_UM3,Vynew_UM3, ax3, ay3 ,t,clock,
       UM_in_rdy3,UM_out_rdy3,xnew3, ynew3, vxnew3, vynew3);
69
70 reg  UM1_done,UM2_done,UM3_done;
71
72
73 //IG1 variables and instantiation
74 wire IG_in_rdy1,IG_out_rdy1;
75 reg IG_in_reg1;
76 assign IG_in_rdy1=IG_in_reg1;
```

```
77  Integrator I1(tx1,ty1,x1,y1,vx1,vy1,xnew1,ynew1,vxnew1,vynew1,xnew2,
        ynew2,vxnew2,vynew2,xnew3,ynew3,vxnew3,vynew3,ax1,ay1,IG_in_rdy1,
        clock,IG_out_rdy1);
78
79  wire IG_in_rdy2,IG_out_rdy2;
80  reg IG_in_reg2;
81  assign IG_in_rdy2=IG_in_reg2;
82  Integrator1 I2(tx2,ty2,x2,y2,vx2,vy2,xnew2,ynew2,vxnew2,vynew2,xnew1,
        ynew1,vxnew1,vynew1,xnew3,ynew3,vxnew3,vynew3,ax2,ay2,IG_in_rdy2,
        clock,IG_out_rdy2);
83
84  wire IG_in_rdy3,IG_out_rdy3;
85  reg IG_in_reg3;
86  assign IG_in_rdy3=IG_in_reg3;
87  Integrator2 I3(tx3,ty3,x3,y3,vx3,vy3,xnew3,ynew3,vxnew3,vynew3,xnew2,
        ynew2,vxnew2,vynew2,xnew1,ynew1,vxnew1,vynew1,ax3,ay3,IG_in_rdy3,
        clock,IG_out_rdy3);
88
89  wire ig_trigger;
90  initial
91  begin
92  ax1     = 16'd64921;
93  ay1   = 16'd64921;
94  ax2     = 16'd1024;
95  ay2   = 16'd1024;
96  ax3     = 16'd614;
97  ay3   = 16'd614;
98  end
99  assign ig_trigger = UM1_done | UM2_done | UM3_done;
100
101 always @(posedge input_read1)    //Make input_check high for writing
        into UM from file
102 begin
103         //diff=$(x1-tx1)>$signed((-16'd1024));
104         if ($signed(x1-tx1)<$signed(16'd900)&&$signed(x1-tx1)>$signed
                (-16'd900)&&$signed(y1-ty1)<$signed(16'd900)&&$signed(y1-
                ty1)>$signed(-16'd900))    begin
105         flag1=1'b1;
106         Vxnew_UM1=16'b0;
107         Vynew_UM1 =16'b0;
108         ax1=16'b0;
109         ay1=16'b0;
110         WT_vx1 = Vxnew_UM1;
111         WT_vy1 = Vynew_UM1;
112         wt1 = 1'b1;
113         #10 wt1=1'b0;
114         input_UM1 = 1'b1;
115         $display("bot_1_reached_target", $time);
116         end
117
118         else  begin
119         input_UM1 = 1'b1;
120         Vxnew_UM1 = vx1;
121         Vynew_UM1 = vy1;
122         end
123   end
```

```
124     always @(posedge input_read2)    //Make input_check high for writing
            into UM from file
125     begin
126             if ($signed(x2-tx2)<$signed(16'd900)&&$signed(x2-tx2)>$signed
                    (-16'd900)&&$signed(y2-ty2)<$signed(16'd900)&&$signed(y2-
                    ty2)>$signed(-16'd900))     begin
127             flag2=1'b1;
128             Vxnew_UM2=16'b0;
129             Vynew_UM2 =16'b0;
130             ax2=16'b0;
131             ay2=16'b0;
132             WT_vx2 = Vxnew_UM2;
133             WT_vy2 = Vynew_UM2;
134             wt2 = 1'b1;
135             #10 wt2=1'b0;
136             input_UM2 = 1'b1;
137             $display("bot 2 reached target", $time);
138             end
139
140             else        begin
141             input_UM2 = 1'b1;
142             Vxnew_UM2 = vx2;
143             Vynew_UM2 = vy2;
144             end
145 end
146 always @(posedge input_read3)    //Make input_check high for writing
        into UM from file
147     begin
148             if ($signed(x3-tx3)<$signed(16'd900)&&$signed(x3-tx3)>$signed
                    (-16'd900)&&$signed(y3-ty3)<$signed(16'd900)&&$signed(y3-
                    ty3)>$signed(-16'd900))     begin
149             flag3=1'b1;
150             Vxnew_UM3=16'b0;
151             Vynew_UM3 =16'b0;
152             ax3=16'b0;
153             ay3=16'b0;
154             WT_vx3 = Vxnew_UM3;
155             WT_vy3 = Vynew_UM3;
156             wt3 = 1'b1;
157             #10 wt3=1'b0;
158             input_UM3 = 1'b1;
159
160             $display("bot 3 reached target", $time);
161             end
162
163             else          begin
164             input_UM3 = 1'b1;
165             Vxnew_UM3 = vx3;
166             Vynew_UM3 = vy3;
167             end
168
169 end
170
171
172
173 always @(posedge UM_out_rdy1)
```

```
174 begin
175     UM1_done  = 1'b1;
176     #10;
177     input_UM1 = 1'b0;
178     UM1_done  = 1'b0;
179 end
180
181 always @(posedge UM_out_rdy2)
182 begin
183     UM2_done  = 1'b1;
184     #10;
185     input_UM2 = 1'b0;
186     UM2_done  = 1'b0;
187
188 end
189 always @(posedge UM_out_rdy3)
190 begin
191     UM3_done  = 1'b1;
192     #10;
193     input_UM3 = 1'b0;
194     UM3_done  = 1'b0;
195 end
196 always @(UM3_done or UM2_done or UM1_done)
197 if(ig_trigger)
198 begin
199     IG_in_reg1=(!flag1);
200     IG_in_reg2=(!flag2);
201     IG_in_reg3=(!flag3);
202 end
203
204 always @(posedge IG_out_rdy1)
205 begin
206 IG_in_reg1=1'b0;
207 end
208
209 always @(posedge IG_out_rdy2)
210 begin
211 IG_in_reg2=1'b0;
212 end
213
214 always @(posedge IG_out_rdy3)
215 begin
216 IG_in_reg3=1'b0;
217 end
218
219 always
220     #50 clock_reg=~clock_reg;
221
222 initial
223     clock_reg=0;
224 always @(flag1 or flag2 or flag3)
225     if((flag1==1)&&(flag2==1)&&(flag3==1))
226         #10
227         $stop;
228
229
```

```
230 endmodule
```

A.7 ROBOT SIMULATION

```
1  function sysCall_threadmain()
2
3
4      objHandle=sim.getObjectAssociatedWithScript(sim.handle_self)
5      result,robotName=sim.getObjectName(objHandle)
6      leftJoint=sim.getObjectHandle("LeftJoint")
7      rightJoint=sim.getObjectHandle("RightJoint")
8      leftJointDynamic=sim.getObjectHandle("DynamicLeftJoint")
9      rightJointDynamic=sim.getObjectHandle("DynamicRightJoint")
10     nominalLinearVelocity=0.3
11     wheelRadius=0.027
12     interWheelDistance=0.119
13     initialVehicleZpos=sim.getObjectPosition(objHandle,sim.
           handle_parent)[3]
14     previousSimulationTime=sim.getSimulationTime()
15
16     file1 = io.open("C:/Users/sujan/Desktop/project final files/
           textfiles/bot1_read.txt", "r+")
17     file2 = io.open("C:/Users/sujan/Desktop/project final files/
           textfiles/bot1_write.txt", "r+")
18     objectHandle = sim.getObjectHandle("LineTracer")
19
20 phi=0
21 theta=0
22 angle=0
23 flag = 1
24
25
26 --initial positions
27 --sim.setObjectPosition(objHandle,-1,{0,0,0})should not set position
28
29 --initial velocities
30 vx=-0.32
31 vy=0.24
32
33 --initial orientation
34
35 sim.setObjectOrientation(objHandle,-1,{0,-1.57,0})
36
37     while sim.getSimulationState()~=sim.
           simulation_advancing_abouttostop      do
38         file1:seek("set",0)
39         status1 = file1:read('*l')
40
41
42         --print(status)
43         if ((status1=='r')and(flag == 1)) then
44
45             file1:seek("set",0)
46             file1:write("w")
47             file1:write('\n')
```

```
48              file1:write(vx)
49              file1:write('\n')
50              file1:write(vy)
51              file1:write('\n')
52              position=sim.getObjectPosition(objectHandle,-1)
53              file1:write(string.format("%.3f", position[1]))
54              file1:write('\n')
55              file1:write(string.format("%.3f", position[2]))
56
57
58              print("bot 1 written:")
59              flag = 0
60
61          end
62          --file1 = io.open("F:/FYP-FINAL/synchronisation/bot1.txt", "r
                +")
63          file2:seek("set",0)
64          status2 = file2:read('*l')
65
66          --print(status)
67          if ((status2=='w')and (flag == 0)) then
68
69              vx = file2:read('*n')
70              vy=file2:read('*n')
71              file2:seek("set",0)
72              file2:write("r")
73              print("bot 1 read", vx, vy)
74              flag = 1
75
76         end
77          -- We want next while-loop to be executed exactly once every
                main script pass, but since
78          -- this script runs in a thread, we explicitely switch
                threads at the end of the while-loop
79          -- Next instruction makes sure one full pass of the while-
                loop can be executed before switching threads:
80          sim.setThreadSwitchTiming(300)
81       theta=math.atan(vy/vx)
82  if(theta==0)then
83  if(vx<0)then
84      theta=3.1415
85  end
86  end
87     if(theta>0) then
88      if(vx<0) then
89        theta  =theta-3.1415
90      end
91
92     elseif(theta<0) then
93          if(vx<0) then
94            theta  =theta+3.1415
95          end
96     end
97
98
99   --   print("theta : ",theta*180/3.1415)
```

```
100
101     angle=theta-phi
102
103
104     if (math.abs(angle)>3.1415) then
105      if(angle<0) then
106       angle =6.283-math.abs(angle)
107      elseif(angle>0) then
108       angle = math.abs(angle)-6.283
109      end
110     end
111 --print("angle : ",angle*180/3.1415)
112 if(angle>0) then
113      --print("angle of rotation: ",angle*180/3.1415)
114      t= math.abs((interWheelDistance*angle)/(wheelRadius*15.7))
115      sim.setJointTargetVelocity(leftJointDynamic,0)
116      sim.setJointTargetVelocity(rightJointDynamic,15.7)
117      sim.wait(t)
118 end
119 if(angle<0) then
120      --print("angle of rotation : ",angle*180/3.1415)
121      t= math.abs((interWheelDistance*angle)/(wheelRadius*15.7))
122      sim.setJointTargetVelocity(leftJointDynamic,15.7)
123      sim.setJointTargetVelocity(rightJointDynamic,0)
124      sim.wait(t)
125 end
126 phi=theta
127
128 --print("phi :",phi*180/3.1415)
129
130
131         v=math.sqrt((vx*vx)+(vy*vy))
132       --print("velocity magnitude : ",v)
133         vel1=v/wheelRadius
134         sim.setJointTargetVelocity(leftJointDynamic,vel1)
135         sim.setJointTargetVelocity(rightJointDynamic,vel1)
136
137
138
139 end
140 end
```