International Journal of Recent Technology and Engineering (IJRTE)
ISSN: 2277-3878 (Online), Volume-7, Issue-6, March 2019

Collision Avoidance using Gazebo Simulator

Sandeep Jose, Kavitha R

Abstract: ***Autonomous cars will make its complete presence on roads in the future. A major feature of autonomous cars currently under research is collision avoidance on roads. Better collision avoidance systems could result in a decrease in number of accidents. Smart collision avoidance systems could handle the increasing amount of vehicles on roads. Collision avoidance system provides alert to the autonomous vehicles if an unavoidable collision is detected. When the collision is definite to happen, collision avoidance system takes action by its own without any driver input (by braking or steering or both). Collision avoidance system does the obstacle avoidance by gathering information about the environment with the help of sensors embedded in the system. The effectiveness of collision avoidance system depends upon the speed at which the system reacts from the gathered inputs. This paper uses the Gazebo simulation to design and implement collision avoidance. This paper also present a simple and effective obstacle avoidance algorithm for a simulated robot. Turtlebot's Obstacle Avoider algorithm is attached to the robot in the simulator with the support of ROS(Robotic operating system) to implement collision avoidance.***

Index Terms**: *Autonomous cars, Gazebo simulator, LIDAR, Smart transport system, Turtle bot.***

I. INTRODUCTION

Smart transport systems rule the present decade. Smart transport system is an improved system which provides users with better knowledge on functionalism of transport and traffic system and empower them with information for a secure, organized and better utilization of transport network. Roads in India are not able to handle the increasing number of vehicles thus increasing the need for intelligent vehicles. An increased amount of smart vehicles can decrease the amount of accidents. Technologies associated with smart systems are divided as computational and sensing. Computational technologies include floating car data. Floating car data are data collected from various transport routes. Floating car data includes triangulation method, vehicle desertification, GPS based methods, and smart phone based rich monitoring [1]. Sensing technologies include inductive loop detection, video vehicle detection and Bluetooth detection [1].

Revised Manuscript Received on 30 March 2019.
***** Correspondence Author
Sandeep Jose, Masters in Computer Applications from CHRIST (Deemed to be University). Rajagiri College of Management and Applied Sciences Kakkanad (Kerala), India.
Kavitha R, Assistant Professor, Department of Computer Science, Madurai Kamaraj University Madurai (Tamil Nadu), India.

Ordinary vehicles are fitted with LIDAR [2] (light detection and ranging) sensor, camera sensor and radar sensor which gather the information necessary for vehicles to navigate. As the vehicles navigate LIDAR [2] sensor gives information about adjacent vehicles and objects on the road, cameras give information about traffic lights, street signs and other signs on the road. Most of the autonomous cars are there to help drivers. For testing of autonomous cars developers use high-quality simulators. Simulators provide a workspace for new algorithms to be created. It provides an opportunity to simulate new sensors which have not arrived in the market. In this paper collision avoidance model using Turtlebot's Obstacle Avoidance algorithm is proposed. The motivation for using Turtlebot's Obstacle Avoidance algorithm in the collision avoidance model is to reduce the collision risk rate for autonomous vehicles in the future. Smart transport systems vary in technologies. Common technologies associated with smart transport systems include car navigation, traffic signal control systems, container management systems, variable message signs, automatic number plate recognition or speed cameras to monitor applications such as security and to more advanced applications that integrate live data and feedback from a number of other sources such as parking guidance and information systems, weather information systems, and etc [1]. Autonomous cars cannot be directly tested with new algorithms since chance of collision is high. So new algorithms are generally tested on a simulator using robotic models before being implemented in autonomous vehicles. There is an indeed demand on every robotic system to find the alternative path when a collision is definite to happen [3]. Navigation and obstacle avoidance is one of the fundamental problems in mobile robotics, which are being solved by the various researchers in the past two decades [4]. Navigation algorithms are divided as deterministic, non-deterministic (stochastic) and evolutionary algorithms. Determinable algorithms are divided into fuzzy logic and neural network fuzzy. Non deterministic algorithms are divided into Generic algorithm, particular swarm optimization, simulated annealing and ant colony optimization. Evolutionary algorithms include a combination of fuzzy logic and non-deterministic algorithms and neural networks. Collision avoidance techniques are applied after a collision leading obstacle is found. There are many features that need to be taken into consideration by robots when possible obstacles are found in the path. Dynamics of obstacle, location and trajectory are some of the features that robot checks on seeing an obstacle.

Decision making is a most important phase in obstacle avoidance in robotic models. Robot takes a decision based on the trajectory. Trajectory can either be determined upon the robot's realization of the obstruction or in case all of the obstructions positions are known, after the map is loaded [5]. Robots are becoming a vital part in our day to day life. Major concerns of robotic cars are complexity of collision avoidance algorithm, compute power, sensor accuracy and sensor data interpretation.

Table 1: Review of papers related to collision avoidance

Year	Implementation	Sensors	Algorithm /method	Features used
2011 [7]	Matlab Simulation environment.	Sensor-LRF, UTM-30 LX (HOKUYO DENKI) -	ERIE laser range simulation algorithm, configuration space method	laser range finder (LRF) was used for obstacle avoidance and environmental magnetic field for navigation to learn all the environment of the whole course.
2016 [8]	Tong University Driving Simulator		FCW algorithm	Improve the handling of extreme high-collision-risk scenarios
2016 [9]	Kuznetsk simulator		probabilistic robot localization algorithm	Training on the intricacies of the algorithm and to develop idea by factors such as the degree of sustained sensory noise.
2017 [10]	Gazebo simulator		Potential field algorithm	Improvement in robot steering and obstacle

Algorithmic complexity is directly proportional to the collision avoidance algorithm. Best algorithms actually lead to the formation of smarter collision avoidance system. Smart systems can drive the vehicles independently on its own from starting point to stopping point. Autonomous cars navigate on the path with less human intervention. Autonomous cars functions independently by collecting data from sensors working along with machine learning and neural networks. Artificial intelligence form the core of autonomous cars. Machine learning algorithms are fed with the patterns in the data by neural networks [4]. Autonomous cars reduce the human errors but face collision due to mechanical failures that happen in it. The world famous car manufactures such as Audi, BMW, Ford, and Tesla working on success of on road autonomous cars [6].

II. LITERATURE REVIEW

Obstacle avoidance is an important function of Autonomous cars. Autonomous cars detect the environment around it using the sensors and plan the trajectory avoiding all the obstacles. Algorithms developed for obstacle avoidance in autonomous cars can be classified into four categories: graph search-based methods, virtual potential and navigation function-based methods, meta-heuristic-based methods and mathematical optimization-based methods [16]. Obstacle avoidance algorithms in autonomous cars help in having a quick and uninterrupted motion of the car to the destination. Most of the obstacle avoidance algorithms are based on controlling the car's velocity. Obstacle avoidance algorithms are tested in simulators before being implemented in real time. Most of the collision avoidance algorithms in robots use a sensory output for getting the position of the obstacle. Sensors help the robot to know the exact position of obstacle and allows for smooth navigation for robots avoiding all obstacles in the path. Table 2 illustrate the review of some of the motivating works related to obstacle avoidance done within the past 18 years. There are many changes happening to autonomous cars over the years. Table 2 also discusses the sensors used and their implementation in obstacle avoidance algorithms. It also discusses whether the implementation was done in simulators or real time and some unique features related to each implementation. Avoiding obstacles appearing in front of autonomous vehicles is called collision avoidance. In recent research collision avoidance has become increasingly important in autonomous cars. Basic collision avoidance systems alert the autonomous cars about the difficulties on a road using in-built sensors and execute an urgent action (such as applying brakes) to avoid a potential accident [16]. Turtlebot's Obstacle Avoider algorithm is used in the collision avoidance model to prevent the collision with static obstacles. An important feature of Turtlebot's Obstacle Avoider is its ability to navigate from start, identify the obstacles on the way take a diversion, and go to the destination through the map. The presented Turtlebot's Obstacle Avoider algorithm is designed for a robot. The collision avoidance algorithm is implemented in python language and is simulated using Gazebo simulator.

Gazebo's realistic environments and simple interface can drastically reduce the turnaround time for efficient design and implementation of new algorithms [17]. There are different types of collisions that can occur in autonomous vehicles. Some of the most common types of collisions happening include side impacts, rear impacts, back up collisions, rollovers, speeding accidents and single car accidents [18].

Table 2: Review of papers related to autonomous cars

Year	Implementation	Sensors	Algorithm /method	Other features
2001 [15]	Real time implementation	Ultrasound sensors, laser range finders, GPS antenna, Gyroscope, Speed meter	adaptive-fuzzy-network-based C-measure algorithm	matching method introduced to solve the problem of car positioning
2008 [16]	Real time implementation			deals with robust localization in outdoor environments, approaches to deal with task allocation and execution, comparative study of Internet protocols for NRS.
2014 [11]	Simvista simulation	Time-to-collision sensor		Evaluates the performance of collision warning system using brake reaction time, maximum deceleration and lane deviation
2016 [12]	TIARAS Simulation	Velodyne HDL 32-E, LIDAR, RTK,GPS, Sens MTi IMU, Point XB2, XB3 stereo cameras	IARA'S Obstacle Avoider Algorithm	IARA'S Obstacle Avoider receives receives updated map of the environment around the car, car's state relative to the map, and a trajectory from the current car's state to the next goal state
2016 [13]	Carnetsoft's simulation		Proactive collision avoidance algorithm	expressed intent of human driven cars and minimize the need for sudden braking or other purely reactive sudden actions
2016 [14]	Simulation was used		Model Predictive Control Algorithm	Avoiding obstacle and ensuring vehicle's protection are primary Factors of MPC algorithm

III. TURTLEBOT'S OBSTACLE AVOIDER

A. Outline

Avoiding obstacles appearing in front of autonomous vehicles is called collision avoidance. In recent research collision avoidance has become increasingly important in autonomous cars. Basic collision avoidance systems alert the autonomous cars about the difficulties on a road using in-built sensors and execute an urgent action (such as applying brakes) to avoid a potential accident [16]. Turtlebot's Obstacle Avoider algorithm is used in the collision avoidance model to prevent the collision with static obstacles. An important feature of Turtlebot's Obstacle Avoider is its ability to navigate from start, identify the obstacles on the way take a diversion, and go to the destination through the map. The presented Turtlebot's Obstacle Avoider algorithm is designed for a robot. The collision avoidance algorithm is implemented in python language and is simulated using Gazebo simulator. Gazebo's realistic environments and simple interface can drastically reduce the turnaround time for efficient design and implementation of new algorithms [17]. There are different types of collisions that can occur in autonomous vehicles. These collisions can actually create huge impact on autonomous cars. Some of the most common types of collisions happening include side impacts, rear impacts, back up collisions, rollovers, speeding accidents and single car accidents [18].

B. Turtlebot's Obstacle Avoider Algorithm

The trajectory of an autonomous car could be described as follows. Let *Linx* be variable denoting linear velocity, *threshold* represents threshold value for laser scan, *Kp* represents kappa constant, *angz* represents angular value. Each of these variables are initialized globally as *Linx* = 0.0, *threshold* = 1.5, *pi*=3.14, *Kp* = 0.05, *angz* = 0.0.

The maximum gap between robot and obstacle is assigned as 40 m. Turtlebot's Obstacle Avoider algorithm shows a clear description of collision avoidance technique Turtlebot has used to avoid collision. Turtlebot's Obstacle Avoider algorithm shows a clear description of collision avoidance technique Turtlebot has used to avoid collision. At each stage with the movement of Turtlebot, the algorithm receives input of an updated map of environment, the current robot's position relative to the map of origin, and trajectory that robot should follow to reach the goal. Turtlebot is repeatedly informed with the trajectory it should take to reach the destination. In Turtlebot's Obstacle Avoider algorithm maximum gap (maxgap) is assigned 40 meters which is the minimum distance that robot maintains with obstacle not to avoid a collision. Range angles (rangeangles) refer to list of range angles Turtlebot can rotate. Range angles can vary from - 0.5(anticlockwise) to threshold value 1.5. A threshold value is a maximum limit of range angle set for robot to move through the map. List of range angle values is assigned to variable largest gap. The minimum (minangle) and maximum angle (maxangle) assigned are calculated using the formula.

*minangle = ((index element value in largest gap) * (angular value of range)*180) / PI)* (1)

*maxangle = ((last element value in largest gap) * (angular valueofrange)*180)/PI)* (2)

Minimum angle is the smallest angle at which Turtlebot rotate. Maximum angle is the largest angle at which Turtlebot rotate. Angular value is used for getting corresponding angular value when a robot faces an obstacle. Average gap is the difference of maximum angle and minimum angle robot should rotate whole divided by 2. Average gap is computed below

Average gap = (maxangle - minangle) / 2 (3)

Turn angle (turn angle) is the angle Turtlebot takes to turn on seeing the obstacle. Turn angle is a sum of both the average gap and minimum angle robot takes to turn on seeing the obstacle. Turn angle depends on the size of obstacle appearing in front of the Turtlebot. Size of obstacles differ from one to another. Obstacles can be dragged and placed in the given square shape in the simulator. If average gap (averagegap) is less than maximum gap (maxgap) the robot would take a journey in an anticlockwise direction. Turtlebot adjust the trajectory according to the position of the obstacle. If an average gap is greater, then maximum gap, Turtlebot would take a linear path and angle Turtlebot takes while travelling linearly is computed using formula

*(Kp * -1) * (90-turnangle).* (4)

Algorithm 1: Turtlebot's Obstacle Avoider Algorithm

1: $Set\ values\ for\ LINX, Pi, Kp, angz, threshold\ and\ max\_gap$
2: ***function*** $LASERSCANPROCESS(data)$//definition of Laserscan Process for Laser subscriber
3: $ranges = list(range\ angles)$ //ranges is assigned list of angles turtlebot can rotate.
4: ***for*** k, g ***in*** $ranges(x)$ ***do***
5: ***append*** *gaplist* // gaplist is assigned distances of obstacles from robot
6: ***sort*** *gap list* // sort gaplist
7: *largestgap = gaplist[-1]* // largest among gaplist is assigned to gaplist[-1]
8: $minangle = largestgap[0] * ((data.angleincrement) * 180/PI)$ // minimum angle that turtlebot should rotate
9: $maxangle = largestgap[-1] * ((data.angleincrement)$ *180/PI // maximum angle that turtlebot should rotate
10: $averagegap = (maxangle - minangle)/2$ // average gap is calculated
11: $turnangle = minangle + averagegap$ // turn angle is calculated
12: ***if*** $(averagegap == max\ gap)$ ***then***
13: $angz = -0.5$ // angle is assigned value - 0.5
14: ***else do***
15: $LINX = 0.5$ // Linear velocity is assigned value 0.5
16: $angz = Kp * (-1) * (90 - turn\_angle)$ // angle value is calculated

Obstacles differ in shape as well. They can come in circular, cylindrical, and some default shapes available in the gazebo simulator. These shapes can be easily pulled to the simulator.

IV. EXPERIMENTAL SETUP

A. Turtlebot's Hardware and Software

Fig. 1 illustrate an overview of how Gazebo simulator works. ROS-Gazebo provides a standard simulation framework based on its modular architecture in robotics research and it facilitates the integration of contributions by other researchers [17]. Sensors used in Turtlebot are Hokuyo laser sensor and the camera sensor. The camera sensor is embedded on top of Turtlebot. Camera sensor allows detecting all the obstacles within its range. Laser sensor is attached to chassis of Turtlebot. It sends the laser beam from chassis. Laser beam gets reflected from some obstacle and thus allows Turtlebot to track the position of an obstacle from it. ROS (robot operating system) provided medium for the development of Turtlebot in Gazebo. Catkin package installed provides workspace for development of robots. Gazebo supports both c++ and python programming language. Environment setup is very flexible. URDF (Unified Robot Description Format) is a standardized XML format file in ROS used to describe all robot elements [21]. URDF is used to describe kinetic features of a robot. Obstacle avoidance code is currently attached to the joints of obstacle avoider.

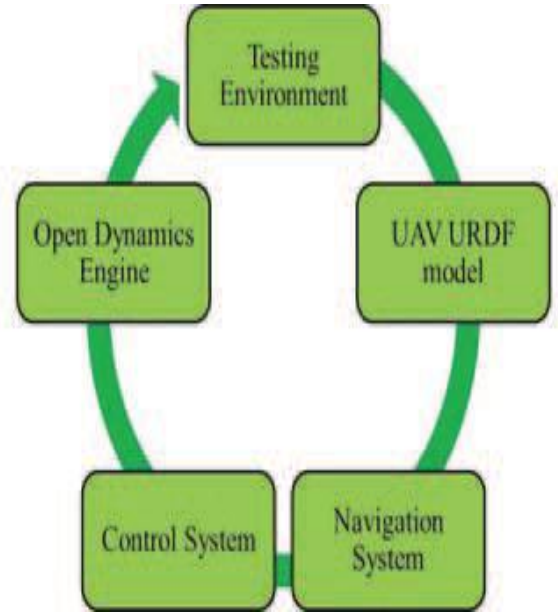

Fig 1: Overview of Gazebo simulator [2]

B. Experimental Methodology

Turtlebot is placed in the center of the square grid in Gazebo. The square grid is defined in the environment by default. Simulator allows to drag in shapes. Turtlebot model to the square grid. Fig.2 illustrate a predefined map in Gazebo simulator. Turtlebot can move bidirectional at 720 degrees in clockwise and anticlockwise direction. Turtlebot is attached with a laser sensor which can detect how far obstacle is away from Turtlebot. Turtlebot has a camera attached which allows seeing the static obstacles that confront it. Turtlebot's wheel is fitted with obstacle avoider algorithm. Custom Turtlebot made in simulator uses message *rospy.Subscriber("scan",sensor_msgs.msg.LaserScan,Laser ScanProcess)* to communicate to rostopic. Rostopic's include publishers, subscribers, publishing rate and ROS messages. Rospy is python client library for ROS [19]. Rospy allows to interact with Rostopic and Rosservice. The '/scan' Subscriber block of the navigation model is subscribing the /scan topic of the ROS[20]. There will be no stoppage of Turtlebot once it has started to move until and unless the controller decides to pause the simulation. When the command *rospy.Publisher('/cmd_vel', Twist, queue_size=10)* is executed, robot starts to move. Rostopic command lists the vehicle's velocity and angles at which the robot is moving, which can be seen at the terminal. Virtual obstacles have been created in the Gazebo simulator, which the robot would detect as possible obstacle. The goal of the experiment is that Turtlebot should avoid the obstacles, detect a clear path and move into that clear path. If another obstacle is found, then the robot should deviate from the obstacle immediately, and scan for the free path. Based on the angular distance which robot captures, a robot would make a decision which path to turn to.

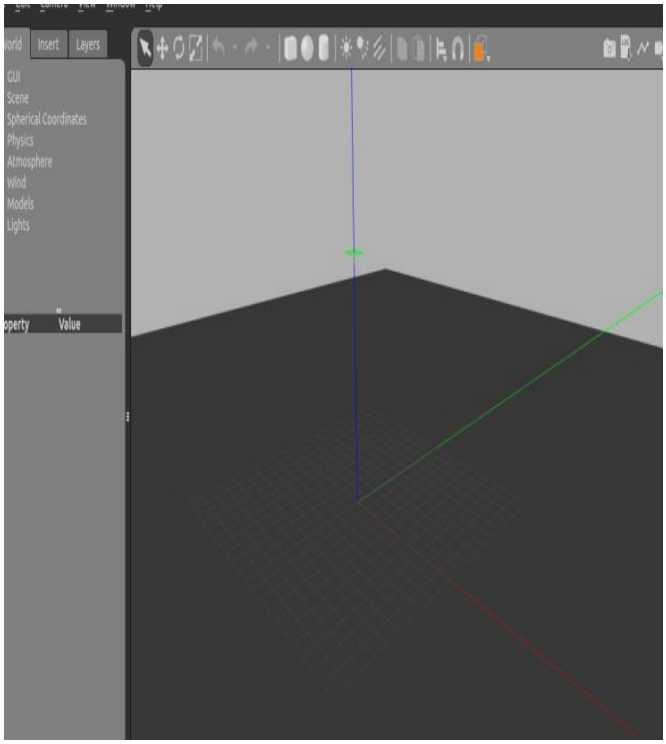

Fig 2**:** Review of papers related to autonomous cars

V. SIMULATION AND RESULTS

The Gazebo simulator used for study is able to simulate collision avoidance which is not possible to demonstrate real time. Fig.3 illustrate the block diagram avoiding a collision in Gazebo simulator. The simulator provides options to develop new robots, collision avoidance controllers and sensors. Simulator has left panel which consists of 3 options world, insert and layers. Sdf (Simulator description format) is used to format world files. World files has got an extension .world. A custom built Turtlebot is created by editing URDF (Unified Robot Description Format) of Catkin package. Catkin package was integrated with ROS during the development of collision avoidance model. Catkin contain all the options for creating a new custom Turtlebot. Catkin package was put into the models section under world files in the simulator. Inside catkin three subfolders were made *testbotdescription*, *testbotGazebo and testbotcontrol*. Entire robot structure including links and joints is mentioned in the *testbotdescription* folder. Entire models are launched into the Gazebo simulator using *testbotGazebo* folder. Control over wheels in Turtlebot is done using *testbotcontrol* model. Turtlebot is attached with a Hokuyo laser sensor and a camera sensor. Turtlebot's joints are attached with the obstacle avoider algorithm. Turtlebot's Obstacle Avoider algorithm is made in the *testbotcontrol* folder.The predefined map is available in the simulator. Custom made Turtlebot is dragged to a required position in the map from world's folder. Insert option is used to insert available models in Gazebo simulation. Models in Gazebo characterize a physical element with dynamic, kinematic, and visual properties. Some available models within simulators for collision avoidance are grey wall, jersey barrier, standing person,

walking person, my robot. These models are dragged into the simulation environments. Simulation environment created is shown in Figure 4. In the implementation ROS was integrated with Gazebo using *Gazebo_ros_pkgs*. Ros_pkgs provides Gazebo with ROS messages and services. Gazebo-ros-pkgs builds only with catkin. It treats URDF (unified robot description format) and SDF (simulator description format) as equally as possible. Gazebo can be run in following ways using following nodes. To launch both the Gazebo server and GUI, the command *rosrun gazebo_ros gazebo* is used. Similarly the command *rosrun Gazebo_ros gzclient* launches the Gazebo GUI. Gazebo server can be launched by using command *rosrun_ gazebo_ros_gzserver*.

Collision avoidance environment created is saved using *.world* extension. Environment is launched using command *roslaunch testbot_launch Gazebo.launch*. Turtlebot is made to move using the command *roslaunch testbotdescription sensor_data_listener.py*. Turtlebot's Obstacle Avoider code is attached in sensor_data_listener.py file. Fig.5 illustrates the angular and linear velocity data generated via simulator. Angular and linear velocity is adjusted according to the position of the obstacles in the simulator.

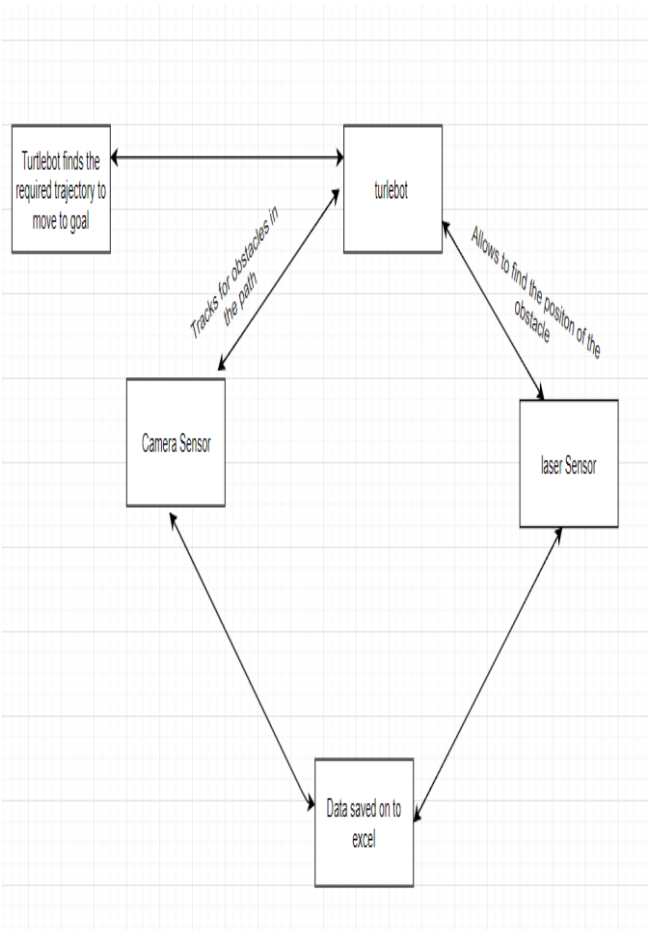

Fig 3: Block diagram illustrating collision avoidance in Gazebo simulator

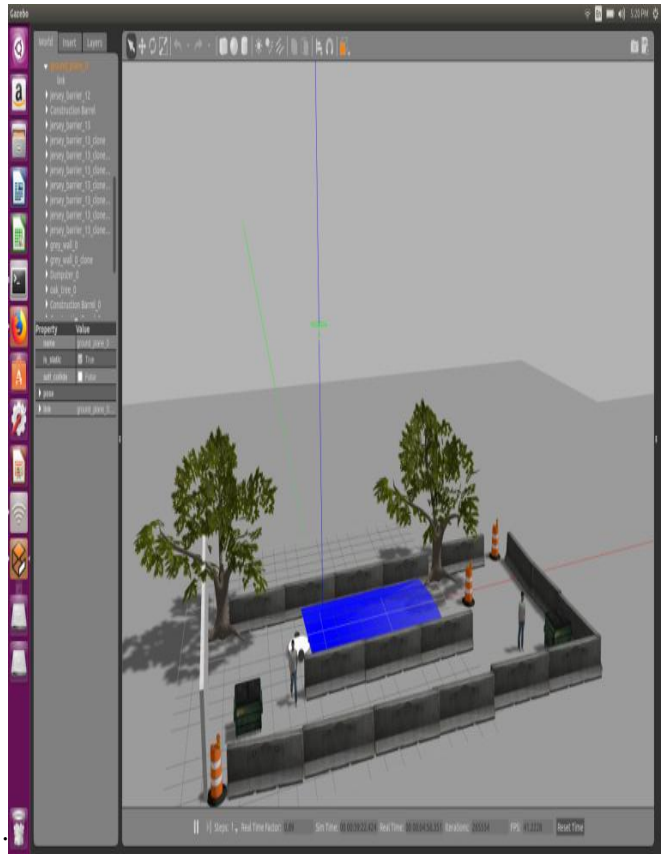

Fig 4: Turtlebot moving in Gazebo simulator

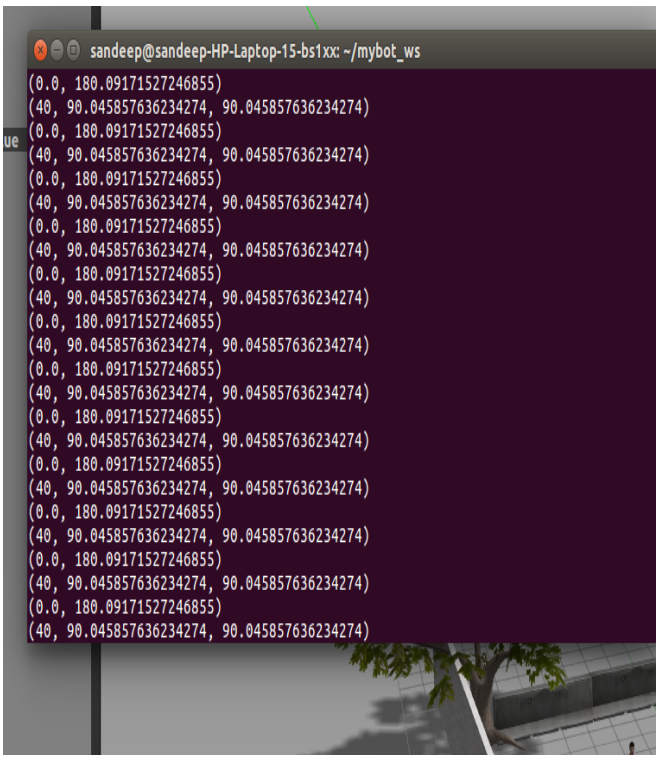

Fig 5: Angular and Linear velocity of Turtlebot

Fig.6 illustrate the Turtlebot avoiding the obstacles on the simulator. Minimum distance to avoid collision between Turtlebot and obstacle is set with a predefined value in

Turtlebot's Obstacle Avoider algorithm. Turtlebot keeps on changing the trajectory according to the position of the obstacle in the map.

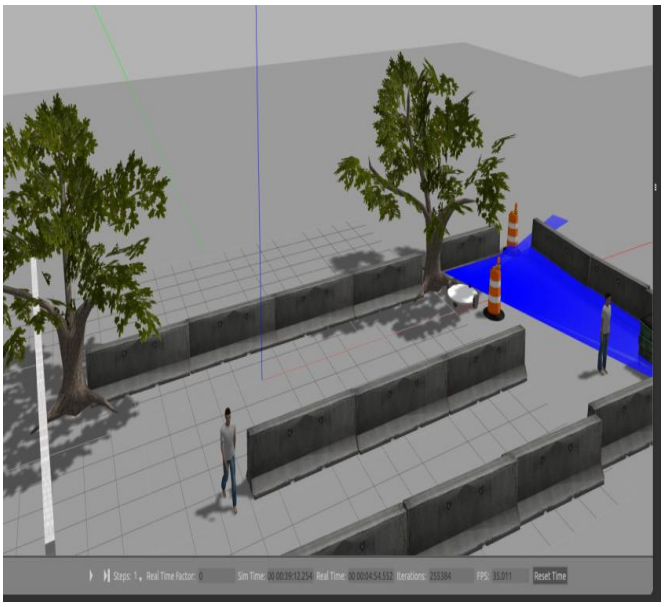

Fig 6: Turtlebot avoiding obstacles on the road

VI. CONCLUSION

Collision avoidance system in autonomous cars helps its users with smart and safe transportation. Effective collision avoidance systems could reduce the number of road accidents by large extent. In this paper Turtlebot's Obstacle Avoider algorithm is implemented using Gazebo simulator. Turtlebot's Obstacle Avoider computes minimum angle, maximum angle, average gap and turn angle to find right path and avoid obstacle. In the experiment only static obstacles are taken into consideration. When Turtlebot detects an obstacle on its path, minimum and maximum angle varies. This helps obstacle avoider to take the right path according to the situation. The sensors such as LIDAR, and Camera from Gazebo are used to detect the right path in this work. Gazebo supports more sensors such as Odometry, Imu, Collision, Gps etc. Gazebo provides option to change lighting conditions such as ambience, background, temperature etc. As a future enhancement, algorithm will be tested for Turtlebot with different sensors, dynamic obstacles, and various ambience conditions. Effectiveness of Turtlebot's Obstacle Avoider algorithm depends on the complexity and performance of algorithm. Based on the performance of Turtlebot's Obstacle Avoider, the algorithm could be used in autonomous cars.

ACKNOWLEDGEMENT

I would like to thank Prof. Joy Paulose for providing a substantial contribution of guidance, encouragement and valuable suggestion during the entire course of research work.

REFERENCES

1. https://en.wikipedia.org/wiki/Intelligent_transportation_system
2. Mengmi Zhang, Hailong Qin , Menglu Lan, Jiaxin Lin, Shuai Wang, Kaijun Liu, Feng Lin and Ben M. Chen, A High Fidelity Simulator for a Quadrotor UAV using ROS and Gazebo, IECON2015-Yokohama, November 9-12, 2015
3. Gunnar Gullstrand, Obstacle Avoidance for Mobile Robot, Kth Numerical Analysis and Computer science
4. Anish Pandey, Shalini Pandey and Parhi DR, Mobile Robot Navigation and Obstacle Avoidance Techniques**,** International Robotics & Automation Journal, Volume 2 Issue 3 - 2017
5. https://en.wikibooks.org/wiki/Robotics/Navigation/Collision_Avoidance
6. https://www.trafficsafetystore.com/blog/autonomous-car-technology/
7. Sam Ann Rahok, Koichi Ozaki, 2D Simulator of Obstacle Avoidance Using LRF for Mobile Robots, Second International Conference on Intelligent Systems, Modelling and Simulation, 2011
8. Xuesong Wang, Ming Chen, Meixin Zhu, and Paul Tremont, Development of a Kinematic-Based Forward Collision Warning Algorithm Using an Advanced Driving Simulator, IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, 2016
9. Robin Han, Dominik Auer, Sarah Edenhofer, Sebastian von Mammen, SkyNetz: A Playful Experiential Robotics Simulator, IEEE 2016
10. 10. Vision Based Control and Simulation of a Spherical Rolling Robot based on Ros and Gazebo, IEEE 4th International Conference on Knowledge-Based Engineering and Innovation, 2017
11. Xuedong Yan , Qingwan Xue, Lu Ma and Yongcun Xu , Driving-Simulator-Based Test on the Effectiveness of Auditory Red-Light Running Vehicle Warning System Based on Time-To-Collision Sensor, www.mdpi.com/journal/sensors,2014
12. Rânik Guidolini, Claudine Badue, Mariella Berger, Lucas de Paula Veronese, Alberto F. De Souza, A simple yet effective obstacle avoider for the IARA Autonomous Car, 2016 IEEE 19th International Conference on Intelligent Transportation Systems, November 1-4, 2016
13. Denis Osipychev, Duy Tran, Weihua Sheng and Girish Chowdhary, Human Intention-Based Collision Avoidance for Autonomous Cars, 2017 American Control Conference
14. Jiechao Liu, Paramsothy Jayakumar, Jeffrey L. Stein, and Tulga Ersal, Combined Speed and Steering Control in High Speed Autonomous Ground Vehicles for Obstacle Avoidance Using Model Predictive Control, IEEE Transactions on Vehicular Technology 2016, 0018-9545
15. Sinn Kim and Jong-Hwan Kim, Adaptive Fuzzy-Network-Based C-Measure Map-Matching Algorithm for Car Navigation System, IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS , VOL. 48, NO. 2, APRIL 2001, 0278–0046
16. Alberto Sanfeliua, Norihiro Hagita b,c, Alessandro Saffiotti , Robotics and Autonomous Systems , 2008 Elsevier, 56 (2008) ,793–797, www.elsevier.com/locate/robot
17. Nathan Koenig, Andrew Howard, Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator ,Proceedings of 2004 IEEE/RSJ Intelligent conference on robots and systems, September 28 - October 2, 2004,
18. https://www.herrmanandherrman.com/blog/types-vehicle-accidents/
19. http://wiki.ros.org/rospy
20. Neerendra Kumar and Zoltan Vlamossy, Obstacle recognition and avoidance during robot navigation in unknown environment, International Journal of Engineering And Technology, 7 (3) (2018), 1400-1404, www.sciencepubco.com/index.php/IJET
21. http://Gazebosim.org/tutorials?tut=model_structure&cat=build_robot

AUTHORS PROFILE

Sandeep Jose is currently pursuing his Masters in Computer Applications from CHRIST (Deemed to be University). He has completed his Bachelors in Computer Applications from Rajagiri College of Management and Applied Sciences. His research interests include Data Analytics, IOT

Kavitha R is Assistant Professor of Computer Science with CHRIST (Deemed to be University) from 2008. She holds a Masters in Computer Applications from Madurai Kamaraj University and an MPhil from Mother Teresa Women's University. She has completed her PhD in Computer Science in 2018 from CHRIST. She has had nearly half a decade's teaching experience prior to joining CHRIST. Her research interests include Data Analytics, IOT, Smart Home Wireless Sensor Networks and Mobile Technology.