Implementation of Autonomous Driving Algorithms on a Miniature Robot

Shreyash G. Patil and Kishanprasad G. Gunale
Dr. Vishwanath Karad MIT World Peace University, School of ECE, Pune, India
Email: shreyashpatil199601@gmail.com; Kishanprasad.gunale@mitwpu.edu.in

***Abstract*—The proposed work mainly attempts to implement numerous image processing algorithms on TurtleBot3 Waffle Pi for autonomous driving using Robot Operating System (ROS). The image processing algorithms used in this work for developing a vision-based system for autonomous driving. Before actual implementation on hardware, the algorithms are first tested in simulation. The time-synchronized data feed is collected from the sensors of the robot. Then the collected data is further processed to perform controlling actions using actuators. The robot's performance is tested and optimized for the same algorithms by varying different parameters in simulation and on a real robot.**

***Index Terms*—ROS, Gazebo, image processing, TurtleBot3, robotics**

I. Introduction

Robotics is becoming a popular area of research and development for intelligent autonomous driving. Autonomous driving has significant challenges in the detection of objects in a natural environment. Based on future prospects, autonomous driving must pass through vehicle testing and validation depending on all possible scenarios. A prototype robot can be used to test the autonomous driving algorithms to know all potential challenges in a physical environment. TurtleBot3 is a standard robot for ROS, which is an open-source framework for robotic applications. TurtleBot3 Waffle Pi comes along with open control board for ROS and Raspberry Pi 3 as Single Board Controller (SBC). It is extensible and can be interfaced with various ranges of modular actuators. Raspberry Pi, Raspberry Pi camera, OpenCR 1.0 board, Bluetooth module, 360º LiDAR and two Dynamixels are the main hardware components of the TurtleBot3 Waffle Pi.

ROS supports code reuse in robotics and ready to use development environment. It has extensive community support, comprehensive tools, and client libraries that help complete the robot process cycle that involves sensing the environment and processing information and action. ROS node is a process that performs computations on some data for sensors and actuators. ROS nodes can communicate with each other by sending ROS messages through topics, services, and actions. ROS topics are used to pass ROS messages among publisher nodes and subscriber nodes by setting connections through a master node in this proposed work. With the help of TCPROS, the transport layer for ROS messages and services, it is based on TCP/IP protocol.

In this proposed work, the TurtleBot3 Waffle Pi model is used. The autonomous driving algorithms are first tested in the simulated world with TurtleBot3 using Gazebo and Rviz tools that ROS supports. Gazebo plugin is a ROS-compatible simulator tool that can create a real-time 3D scenario for any robot [1]. The performance of TurtleBot3 Waffle Pi in the Gazebo simulator is tested on a simulated track that is designed for testing autonomous driving. Simultaneous localization and mapping (SLAM), navigation, lane detection, lane following, collision avoidance, lane changing, traffic light detection, traffic stop sign detection, and parking are the algorithms tested in simulation. Image processing algorithms using OpenCV are implemented on some nodes to detect objects in the environment, and necessary control actions are taken depending on the type of detected object [2]. Edge detection, masking, threshold, hough transform is applied to the image to test the detection of lane markings. SLAM is used to generate map of an unknown environment. The waypoint based navigation is used to provide the reference points to avoid moving out of the lane. Feature matching technique is used to detect the traffic signs. A simple blob detector is used for the detection of traffic lights. All these algorithms are implemented in the real world for TurtleBot3 after testing in the simulated world.

II. Related Work

Previous studies based on vision-based systems that detect and recognize objects in natural environments use image segmentation and feature extraction for object identification and provide a background for comparing different ML algorithms on a robot [3]. Based on the detected obstacle or object, a neural network controller can be used to control the trajectory with the help of a neural-network based velocity controller [4]. There are approaches to create a map of an unknown environment. In the presence of a sensor like a laser scanner, the GMapping, cartographer, or frontier exploration approach can be used to obtain a map by scanning the environment concerning the robot's pose [5]. Turtlebot has the

Manuscript received July 9, 2021; revised August 25, 2021; accepted September 16, 2021.
Corresponding author: Shreyash G. Patil (email: shreyashpatil199601@gmail.com).

doi: 10.18178/ijeetc.11.2.138-145

capability to track an object and follow it in real-time [6]. Further modification in TurtleBot3 can be done by using Zynq7000 SOC that comes along with FPGA for faster real-time processing of high-quality images and NVIDIA Tesla GPU to utilize Tensorflow capabilities MNSTbot [7]. Various sensor fusion frameworks can be implemented to estimate better data from the sensors. Kalman filter can be used for better estimation of the data from two or more sensors providing similar types of data [8]. Detection of lanes can be done by two methods: model-based and feature-based. Hough transform can be used easily to detect straight lane markings, while the curved lane detection can be done by least-square fitting [9]. The Advanced Driver Assistance System (ADAS) features also include lane departure warnings while detecting the lane. The concept of vanishing point can be used to estimate lane departure based on whether the lane curving is towards the right or towards the left. The vanishing point remains at the center of the image frame only if the lane is straight. It moves towards the left if the lane is curving towards the right and vice versa [10]. In most of the Hough transform-based lane detection research, the image is processed to select the region of interest, conversion to grayscale, and edge detection followed by drawing hough lines. The processing time and the data storage limit can be reduced by using Hough space and fast Hough transform, parallel Hough transform and improved Hough transform for better results [11]. Another method to detect lane departure is by using the Euclidean distance. The Euclidean distance among origin, the midpoint of left lane marking, and the midpoint of right lane marking help calculate departure from either the left or right side of the road [12]. There are two methods for lane changing, i.e., free lane change and forced lane change. Free lane change operates according to the driver command [13]. The process of changing lanes or avoiding obstacles can be made easier by using SLAM, especially if the surrounding environment of the ego robot or vehicle is static. SLAM can be used to generate a map of an unknown environment with the help of sensors like Kinect depth-sensing camera, LRF or LiDAR [14]. The next critical functionality in autonomous driving is avoiding the obstacle and changing lanes by avoiding the collision. This is because the lane following algorithm keeps the system in the lane, and the lane changing algorithm tries to move the robot to another lane. It is complicated to have another lane in the same field of view. It becomes a challenging task when there are more obstacles to be avoided from collision [15]. Occupancy grid map is a very commonly used method in ROS to generate a map using SLAM. Using waypoints or multiple goal locations to the navigation stack, the TurtleBot3 can be made to move in a saved map to follow a given trajectory [16]. The official manual by Robotis for Tutlebot3 provides details of the basic functionalities to start with TurtleBot3. The detailed information for basic operations of TurtleBot3 like teleoperation, SLAM, navigation, simulation, manipulation, autonomous driving, machine learning is available in the Robotis manual for TurtleBot3 [17]. While autonomous driving, it is necessary to continuously keep track of each frame if any traffic sign or signal is present. Various researches have already been implemented, e.g., using machine learning classification for traffic sign detection or feature matching techniques to detect particular signs. Feature detection techniques using SIFT, SURF detectors of OpenCV give a fast response for real-time feature matching with Brute Force or FLANN-based matchers [18]. It is necessary to test all algorithms, which is not always a cost-effective solution if directly implemented on the hardware. Hence there are simulation tools that can be used with ROS. Gazebo plugin is a ROS-compatible simulator tool that can create a real-time 3D scenario for any robot. It also resembles a real-world scenario for the robot along with the environment [19]. A complete robot description using Unified Robot Description Format (URDF) files is possible with Gazebo with necessary sensors and actuators, which can be visualized in another tool called Rviz. Further, all the algorithms to be implemented on the real robot can be first tested in simulation. Once the simulation results give an optimized expected response, it can be implemented on the real robot.

III. Proposed Work

The scope of this study is to use ROS for TurtleBot3 Waffle Pi and implement autonomous driving algorithms like lane following, traffic light detection, stop sign detection, parking sign detection, parking, obstacle avoidance and lane changing. The Autorace package for TurtleBot3 burger is modified to perform additional functionalities using another model, i.e., TurtleBot3 waffle pi.

A. Lane Following

The lane detection is done using a camera and OpenCV. However, image processing cannot be directly applied in ROS. Image pre-processing is required for the lane following process, where most lane markings are straight, shown in Fig. 1.

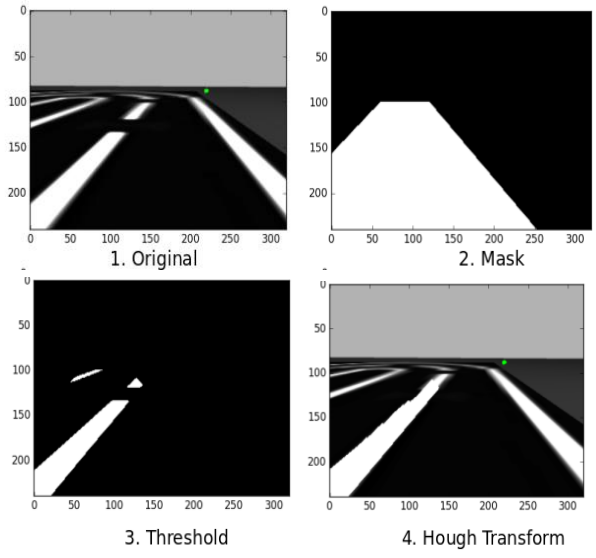

Fig. 1. Image pre-processing.

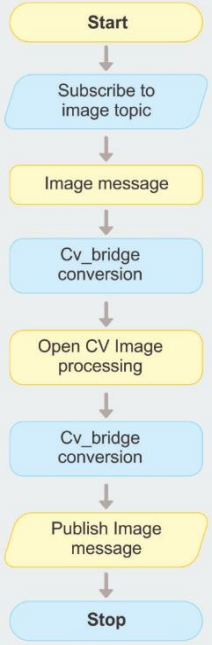

Fig. 2. Flowchart showing ROS image conversion to OpenCV image.

In ROS, the nodes can send and receive ROS messages, so the image format is not the same as the OpenCV image. Here the cv_bridge library is used to convert ROS image message to OpenCV image shown in Fig. 2. The lane following must work very efficiently such that no other functionalities get disturbed due to improper lane detection [20]. By default, the Autorace package uses yellow lane markings in the left lane and white lane markings on the robot's right side. But it is not always possible to have continuous left lane marking in yellow color. In the case of actual road scenarios across the globe, there is a possibility of having dotted lane marking. For this, a small test track with dotted lane marking is prepared for lane detection. The first step of lane detection is to select the region of interest by preparing an appropriate mask for the input grayscale image. Further image processing is done on the area designated by the mask, which is trapezoidal. Then binary thresholding of the image is performed to find the presence of lane marking.

Hereafter, the Hough transform is applied directly to the thresholded image without edge detection to connect the discontinuity in all the lines. Hough transform converts the points from the Cartesian image space to curves in the hough space or parametric space. The number of possible lines passing from each point conversion is mathematically given by parametric equation Eq. (1):

$$x\cos\theta + y\sin\theta = \rho \quad (1)$$

The performance was best for parameters set to $\rho = 1$ and $\theta=\pi/180$ with a maximum 30 number of lines passing through one point and a maximum line gap of 250. The bird's eye view can be obtained by changing the perspective of the image to get an almost infinite slope for both the lane lines. Finding the peaks of the histogram at the lines helps to get the location of the lane lines. This position can be implemented with a sliding window search for getting the information about the frames following the lines. Further, the center of left and right is calculated by getting the mean value of both the line locations. This center lane value is used as the input to the Proportional Integral Derivative (PID) controller. Output is given to the angular Z value of the topic responsible for the turning motion of the robot. The PID tuning is performed by varying the K_p and K_d constants until the robot moves in an optimized way to follow the lane Eq. (2).

$$\text{angular\_z} = K_p \times \text{error} + K_d \times (\text{error} - \text{last\_error}) \quad (2)$$

where the error value is the deviated value from the center of the lane.

B. Obstacle Avoidance and Lane Changing

The most challenging task is to perform obstacle avoidance and lane changing since two separate nodes cannot be used to control the same ROS topics simultaneously. It will cause clashes between the ROS messages, which will make the robot show oscillatory behavior. Since one ROS message will try to move the robot to another lane, other ROS messages will force the robot to keep running in the same lane. Another issue happens if one node subscribes to two different topics and publishes the message to the same topic having different behaviors. The same results are obtained, which causes the robot to show clashes between two other behaviors. The solution uses a message filters library that takes the messages of various types from multiple sources as input and outputs the message only if it has received it on each of those sources with the same timestamp [21]. This lane-changing algorithm is added to the node for controlling the robot by using a PID controller.

An approximate time synchronizer is used in this particular application for subscribing to messages from ROS topics "/scan" and "/desired_center data". The topic "/scan" provides information from the 360° laser scanner or LiDAR required to detect obstacles and the "/desired_center" data provides the information from the detect_lane node required to follow the lane. The threshold is set to 30°, so the scanning angle range is between –30° to +30°. After that, PID tuning can be required based on the curvature present in the track in the same node. Fig. 3 shows a flowchart for obstacle avoidance.

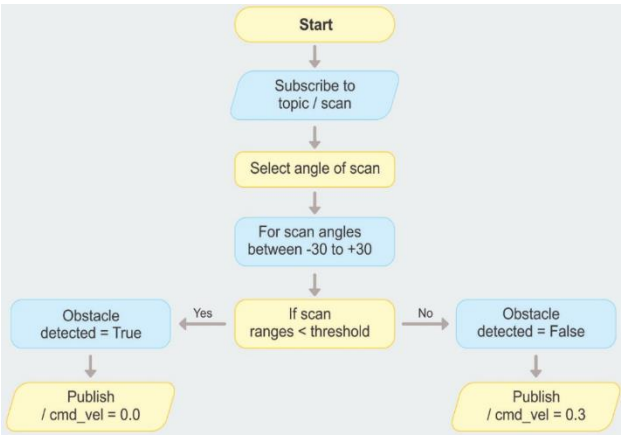

Fig. 3. Flowchart showing obstacle avoidance.

C. Traffic Light Detection

The detection of traffic lights requires calibration based on hue, saturation, and value of colors for continuous detection of the traffic signals. The separate node for traffic light detection converts the BGR format of the image to HSV, then it uses a simple blob detector from OpenCV and matches it with the circular mask with color detection. If red light gets detected, then it publishes a "max_vel" message value equal to zero to the control lane node, and if green light is detected, then "max_vel" message value 0.12 is published to the control lane node.

D. Traffic Sign Detection

Traffic sign detection node uses Fast Library for Approximate Nearest Neighbor (FLANN) based on feature matching technique with Scale Invariant Feature Transform (SIFT). For each traffic sign, one reference image is kept in a directory. The same traffic sign is used on track for detection. Once the traffic sign is visible in the camera feed, the feature matching algorithm detects similar pixels from the camera feed and reference image. If the number of pixels matching is more than a threshold value, then the sign is classified as parking or stop sign.

After detecting a particular sign, the message is published through the sensor messages, which are then subscribed by a respective node. Based on the sign detection, necessary control actions are performed. If a parking sign is detected, the robot will search for vacant parking and stop for the given time. If the stop sign gets detected, all the processes will be stopped after making the velocity of TurtleBot3 zero.

IV. Simulation and Results

A. Lane Following

The discontinuous lane markings are joined to form a continuous lane marking using hough transform by applying a mask to a particular region of interest, as shown in Fig. 4. HoughlinesP method of OpenCV is used to draw lane marking at discontinuities that uses probabilistic Hough transform. It is observed that the Hough transform works better for straight lines; however, it cannot effectively join the discontinuous lane markings for curvature. For curvature, a 2nd order polynomial:

$$ax^2 + bx + c = 0 \quad (3)$$

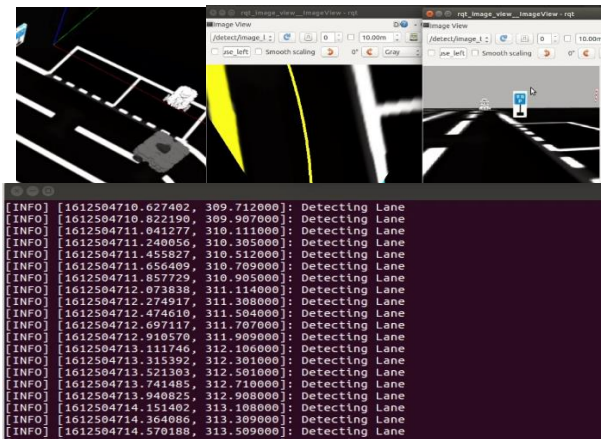

Fig. 4. Lane following simulation.

Equation (3) is used for the line formed by peaks of the histogram. In the simulation, it performs well for the curvature with proper PID tuning at the controller end. The PID controller is tuned such that the deviation from the lane results in an increased error value; it is then compared with the previous error value and fed to the turn function to bring the robot back to the lane. The desired rise time and setting time for smooth motion without any oscillation were achieved by manually varying the K_p and K_d constant values. Finally, K_p=0.0025 and K_d=0.007 were finalized for better performance. Thus, whenever the error value increases, it gets compared with the previous error and multiplied by the K_p and K_d constants, increasing the turning in opposition to the deviation from the lane.

B. Obstacle Avoidance and Lane changing

1) Obstacle avoidance

As the obstacle gets detected precisely in the scan range of 0 degrees and the distance range of 0.5m, the message is published to the "cmd_vel" topic. It changes the velocity of TurtleBot3 to zero until the obstacle remains detected in the gazebo simulator. The robot stops at a distance of approximately 0.5m with an expected tolerance of –0.1m, as shown in Fig. 5.

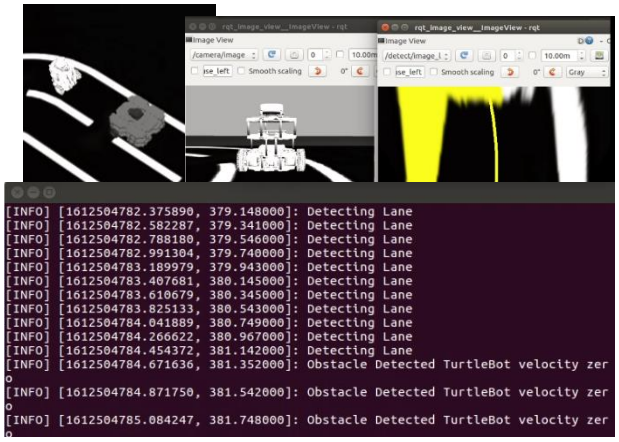

Fig. 5. Obstacle avoidance simulation.

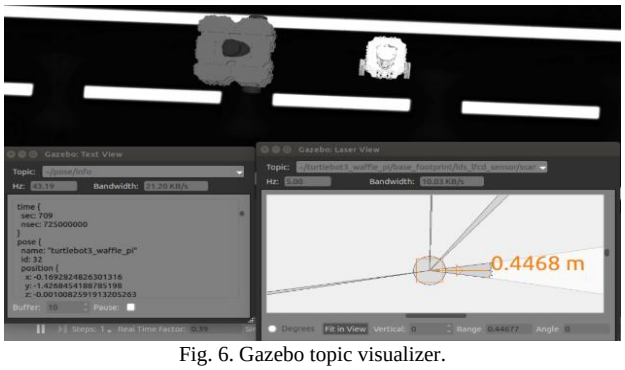

Fig. 6. Gazebo topic visualizer.

2) Lane changing

The exact distance between Turtlebot3 and obstacle can be seen in the Gazebo topic visualizer tool shown in Fig. 6. The performance of the lane changing algorithm in simulation is not entirely efficient because there are two different sensor messages used for lane changing. The first one belongs to the image message coming from the

camera to keep in the same lane, and the other is a scan message coming from the laser scanner to change the lane due to a detected obstacle. The ROS message filters property is used to work in a time-synchronized manner. The approximate time synchronizer method is used to allow multiple subscribers for ROS messages of two different topics. But if the message's arrival is before the expected time in a callback, then that message is ignored, which sometimes causes the robot to turn at an undesired angle. The problem does not occur for 8 out of 10 trials, and the robot successfully changes lane. Successful lane changing is shown in Fig. 7.

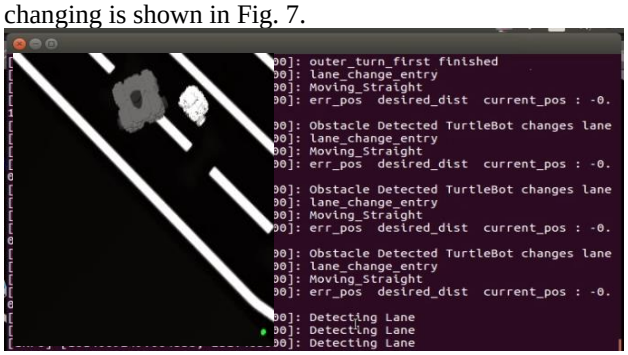

Fig. 7. Obstacle avoidance and lane changing.

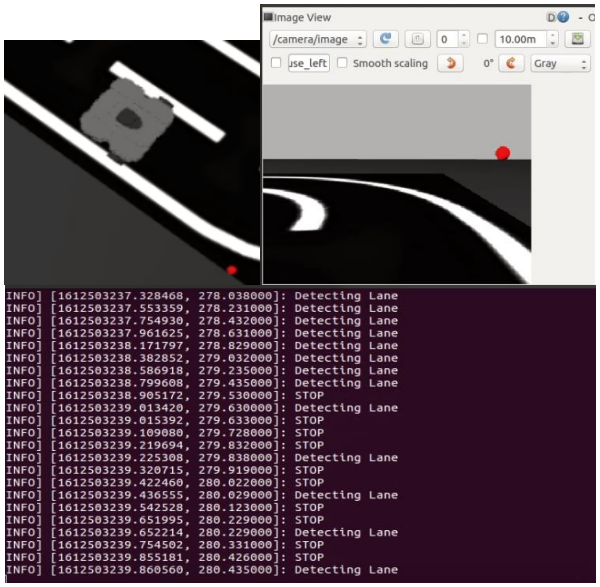

Fig. 8. Red traffic signal simulation.

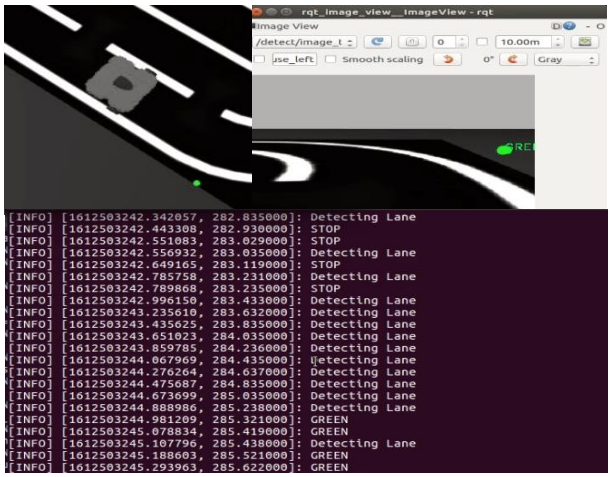

Fig. 9. Green traffic signal simulation.

C. Traffic Light Detection

1) Red traffic light

After converting the image from BGR to HSV, it becomes easy to differentiate the colors in an image by applying the mask for a range of particular colors. The traffic light detection algorithm performs well by using a simple blob detector of openCV by finding the circularity within a fixed range. In the case of red color, each object in the range of red color is further checked whether the circularity is present within a certain radius and if it is in the desired degree of HSV format for red.

2) Green Traffic Light

Similarly, for green color, the desired range is set using proper HSV values for green. In the simulation, the successful detection of traffic lights for red and green colors is shown in Fig. 8 and Fig. 9, respectively.

D. Traffic Sign Detection

1) Parking sign

The traffic sign detection in simulation performs remarkably well for both parking and stop signs. It is to be noted that as the FLANN-based feature matching technique is used in sign detection, the images of objects used in simulation should match the image of the reference directory. It limits the traffic sign detection to exactly the two traffic signs which are used in this proposed work.

2) Stop sign

The parking sign detection is shown in Fig. 10, and stop sign detection is shown in Fig. 11 in the simulation.

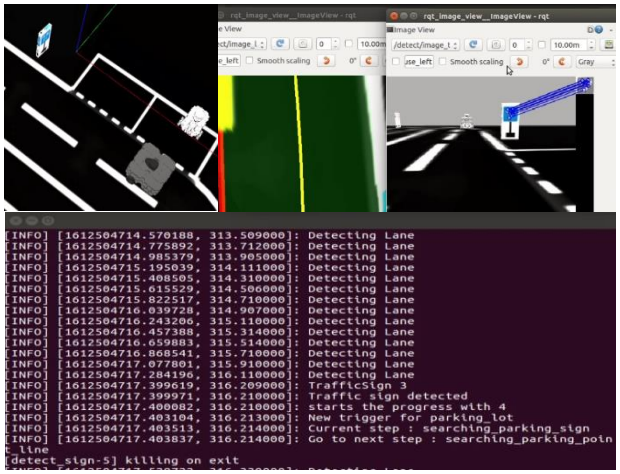

Fig. 10. Parking sign detection simulation.

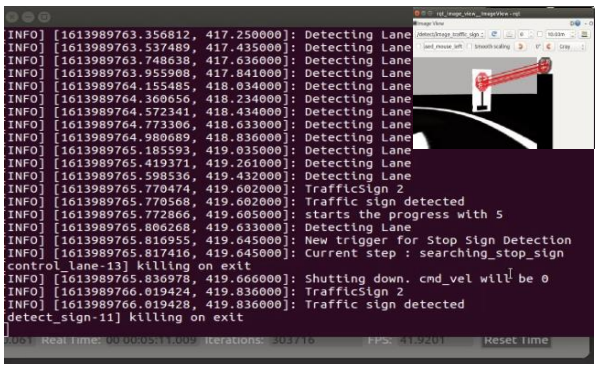

Fig. 11. Stop sign detection simulation.

V. Implementation and Results

A. Lane Following

The lighting conditions of the environment affect the performance of the image processing algorithms. Therefore, camera calibration must be done to set the desired parameters. Intrinsic calibration involves parameters like focal length, distortion, skew, which is done using the checkerboard. Extrinsic camera calibration involves lightness, hue, saturation, selection of the region of interest, and left and right lane visibility. All these calibration parameters are saved in YAML files. The lane following algorithm is shown in Fig. 12. Actual implementation uses a similar approach which is used in the simulation for TurtleBot3 waffle pi. In the simulation, the Hough transform is applied on a bird's eye view. However, in the case of actual implementation, it is applied to the camera feed as it produced better response.

The presence of lane marking is detected by using the histogram peaks and fed to draw sliding windows at the points generated by histogram peaks. These points are followed to estimate the curvature. Using the polyfit function, the lane lines are drawn on curved lanes, as shown in Fig. 13. Depending on the remote PC processing speed, the number of sliding windows can be varied to obtain better results, such as n=40 and n=20, as shown in Fig. 14 and Fig. 15, respectively. Further, the center of the lane is calculated and used as an input to the PID controller for lane following.

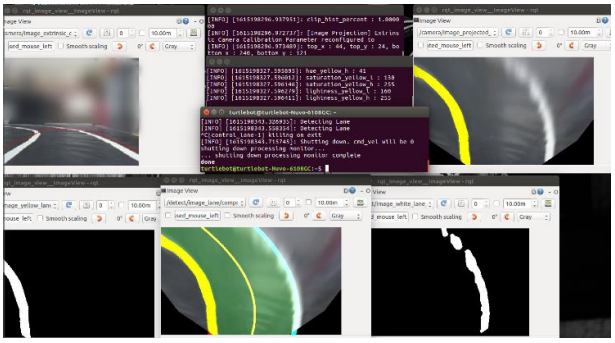

Fig. 12. Lane following.

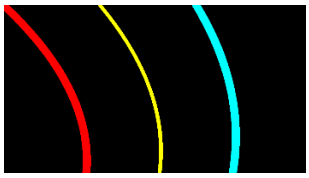

Fig. 13. Drawing lane lines to follow.

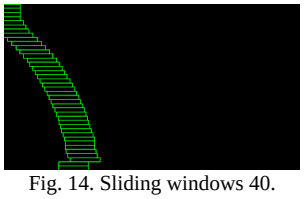

Fig. 14. Sliding windows 40.

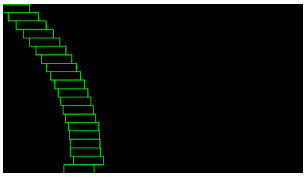

Fig. 15. Sliding windows 20.

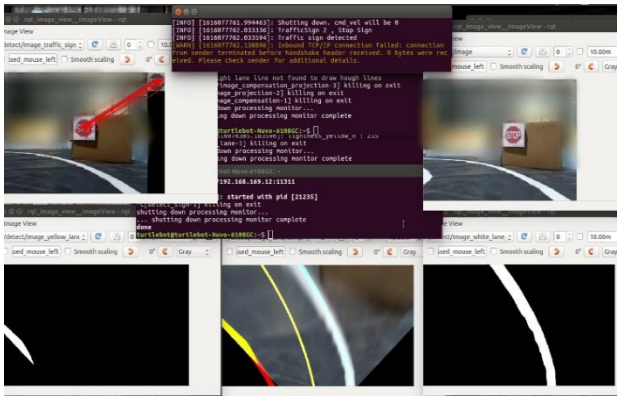

Fig. 16. Stop sign detection.

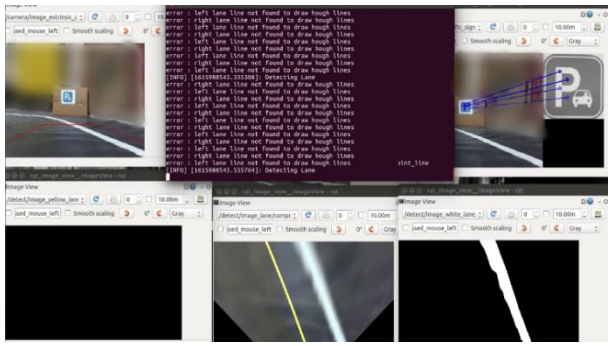

Fig. 17. Parking sign detection.

B. Traffic Sign Detection

The image of the traffic sign used for actual implementation must be kept in the reference directory used for matching 8images from the camera frame. The fixed distance between the matched pixels while feature matching can be varied by using a multiplier set to 0.7 to maintain the ratio. Parking is executed by keeping track of the scan ranges 60º to 120º. If there is no obstacle in the scan ranges, then the robot performs the parking function. Fig. 16 and Fig. 17 show the stop sign detection and parking sign detection, respectively.

C. Traffic Light Detection

The algorithm used in actual implementation remains the same as the simulation. However, the radius of the circular light parameter is modified to recognize the traffic light from the desired distance.

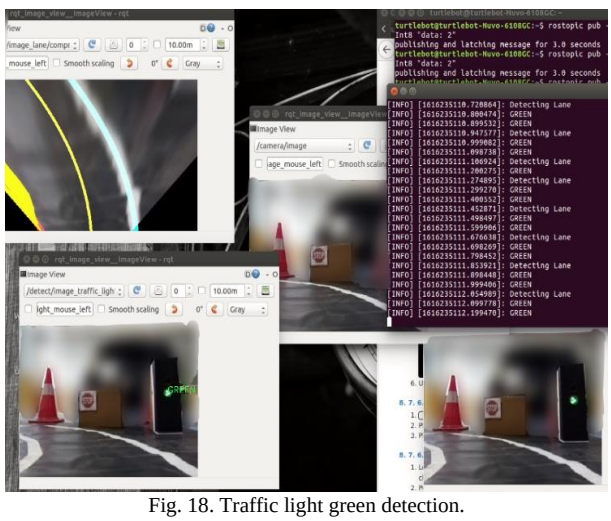

Fig. 18. Traffic light green detection.

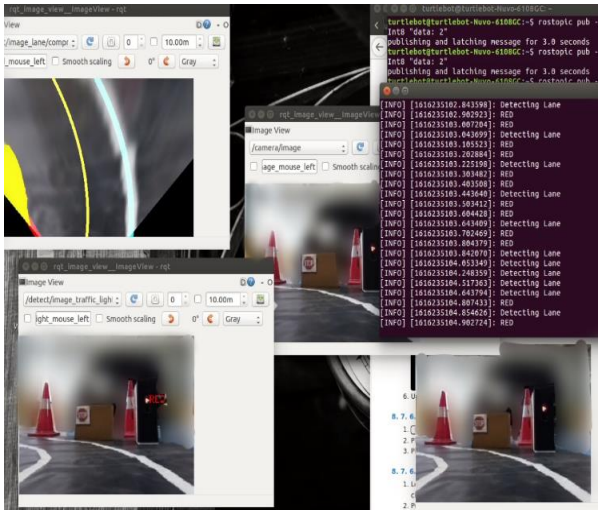

Fig. 19. Traffic light red detection.

The green and red traffic light signals are detected as shown in Fig. 18 and Fig. 19, respectively. All the necessary parameters like hue, saturation, and value for red and green colors are calibrated while performing extrinsic calibration for traffic light detection.

D. Waypoint Based Navigation

In [22] Zandra B. Rivera et. al. simulated waypoint based navigation system for Wheeled Mobile Robot (WMR) using the Gazebo simulator. Comparison between Gazebo Simulation of WMR with MATLAB Simulink model is presented by author.

In the proposed approach various lane detection algorithms are first simulated in Gazebo envirnoment and then real time testing is carried out using Turtlebot3. Further lane changing algorithm is also implemented if obstacle gets detected. However sharp curvatures may cause robot to move out of the lane, so to avoid this waypoint references can guide the robot to keep in same lane.

The SLAM node is used to generate a map of unknown environment. The red arrows are the waypoints that are set using a pose array using visualizing tool Rviz. The path travelled by the robot is shown in Fig. 20.

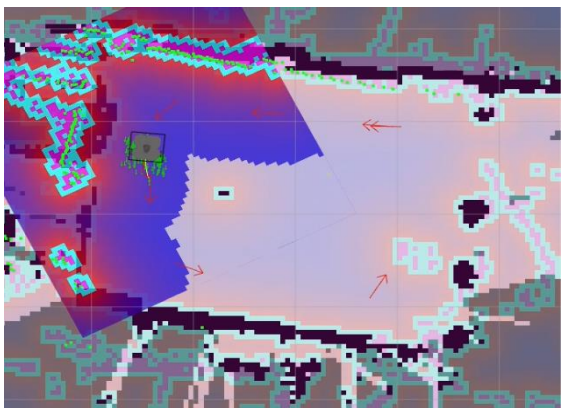

Fig. 20. Waypoint based navigation.

VI. Limitations and Future Directions

The actual TurtleBot3 must be tested in an environment with controlled lighting conditions since most detection algorithms require image processing. Reinforcement learning approach and adding reference waypoints while following lanes can help tackle problems related to lighting conditions in image processing. The lane markings are not visible if the obstacle in front of the robot is significant. Due to this, the robot avoids obstacles; however, it cannot change the lane as the lane marking is not visible in the camera frame. Path planning can be used to implement proper lane changing. The processing speed of SBC is limited to 30 FPS for high-quality frames, and sometimes if the frames get dropped, it can miss the object that needs to be detected. The robot fails to turn for curvature with a significantly smaller radius during lane finding. It can move out of the lane for such cases. Kalman filters using parabolic and circle equations for estimating the curvatures in lanes can reduce these problems. This system can be implemented on different SBC with higher processing speeds to achieve better response. The traffic signs and the traffic lights used must match the predefined type. Image classification using machine learning may be preferred to detect all possible traffic signs. Variation in these conditions and signs leads to changes in image files and methods.

VII. Conclusion

In this research work, we have implemented autonomous driving algorithms on TurtleBot3 waffle pi with ROS. Various algorithms like lane following, obstacle avoidance, and lane changing were implemented and verified. Detection of the traffic light, stop sign, parking sign algorithms were tested in simulation, and on an actual robot, subject to the objects placed in the surroundings must be static. The Hough transform is a proven technique to eliminate discontinuity in lane markings. Under the simulation environment, if the gap between the lane markings in a lane increases, then the robot tends to oscillate. Proper PID tuning is necessary to prevent oscillatory behavior and to improve stability. ROS tools and packages are used to access the data efficiently from sensors and process it to generate a proper response for the actuators to perform necessary actions. However, the performance of these algorithms is limited to the processing power of the computer. Image processing that is used is sufficient to perform proper detection of surrounding objects but is limited to the environmental lighting conditions. These studies are required for a prototype robot to know all potential challenges in a physical environment for autonomous driving.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

Authors Shreyash Patil and Kishanprasad Gunale contributed to the research; Shreyash Patil worked on simulation and implementation of the proposed work.

Conceptualization, Shreyash, Kishanprasad; formal analysis, Shreyash; software, Shreyash; supervision,

Kishanprasad; writing—original draft, Shreyash, writing—review and editing, Kishanprasad.

ACKNOWLEDGMENT

The authors acknowledge the infrastructure and support offered by Automotive Research Association of India (ARAI), the interdisciplinary R&D organization of in India. The authors would like to thank Dr.V.D. Karad MIT World Peace University, School of Electronics and Communication Engineering for lab facilities and a highly encouraging work environment, which helped in the completion of this proposed work.

REFERENCES

[1] R. Mishra and A. Javed, "ROS based service robot platform," in *Proc. of 4th Int. Conf. on Control, Automation and Robotics*, 2018, pp. 55-59.

[2] OpenCV Python Tutorials. [Online]. Available: https://opencv-python-tutroals.readthedocs.io/en/latest/

[3] M. Mainampati and B. Chandrasekaran, "Evolution of machine learning algorithms on autonomous robots," in *Proc. 10th Annual Computing and Communication Workshop and Conf.*, 2020, pp. 0737-0741.

[4] K. Khnissi, C. Seddik, and H. Seddik, "Smart navigation of mobile robot using neural network controller," in *Proc. Int. Conf. on Smart Communications in Network Technologies*, 2018, pp. 205-210.

[5] W. A. Syaqur, A. S. A. Yeon, A. H. Abdullah, *et al*., "Mobile robot based simultaneous localization and mapping in UniMAP's unknown environment," in *Proc. Int. Conf. on Computational Approach in Smart Systems Design and Applications*, 2018, pp. 1-5.

[6] I. K. E. Purnama, M. A. Pradana, and Muhtadin, "Implementation of object following method on robot service," in *Proc. Int. Conf. on Computer Engineering, Network and Intelligent Multimedia*, 2018, pp. 172-175,

[7] K. Mori, Y. Saitoh, and N. Nakasato, "Introduction of MNSTbot," in *Proc. Int. Conf. on Field-Programmable Technology*, 2018, pp. 397-399.

[8] S. Gangadhar, "Sensor fusion framework and simulation on a TurtleBot3 robotic vehicle," M.S. thesis Electrical Engineering, North Carolina University, 2017.

[9] G. Deng and Y. Wu, "Double lane line edge detection method based on constraint conditions Hough transform," in *Proc. 17th Int. Symp. on Distributed Computing and Applications for Business Engineering and Science*, 2018, pp. 107-110.

[10] S. Gupta and D. Yadav, "Lane-finding based on structure analysis of lane and computer vision," in *Proc. Int. Conf. on Communication and Electronics Systems*, 2019, pp. 1513-1519.

[11] P. Maya and C. Tharini, "Performance analysis of lane detection algorithm using partial Hough transform," in *Proc. 21st Int. Arab Conf. on Information Technology*, 2020.

[12] P. N. Bhujbal and S. P. Narote, "Lane departure warning system based on Hough transform and Euclidean distance," in *Proc. Third Int. Conf. on Image Information Processing*, 2015, pp. 370-373.

[13] Y Liu and D. Wang, "Vehicle lane changing model to safety avoid obstacles," in *Proc. World Automation Congress*, 2012, pp. 1-4.

[14] X. Li, S. Li, S. Jia, and C. Xu, "Mobile robot map building based on laser ranging and kinect," in *Proc. IEEE Int. Conf. on Information and Automation*, 2016, pp. 819-824.

[15] M. Haris and J. Hou, "Obstacle detection and safely navigate the autonomous vehicle from unexpected obstacles on the driving lane," *Sensors* vol. 20, no. 17, 2020.

[16] Alberto Ezquerro. Sending goals to the navigation stack using waypoints. [Online]. Available: https://theconstructsim.com

[17] Manual by robotics for Turtlebot3. [Online]. Available: https://emanual.robotis.com/docs/en/platform/turtlebot3/applications/#applications

[18] R. Renjith, R. Reshma, and K. V. Arun, "Design and implementation of traffic sign and obstacle detection in a self-driving car using SURF detector and Brute force matcher," in *Proc. IEEE Int. Conf. on Power, Control, Signals and Instrumentation Engineering*, 2017, pp. 1985-1989.

[19] R. K. Megalingam, D. Nagalla, R. K. Pasumarthi, V. Gontu, and P. K. Allada, "ROS based, simulation and control of a wheeled robot using gamer's steering wheel," in *Proc. 4th Int. Conf. on Computing Communication and Automation*, 2018.

[20] ROS Tutorial. [Online]. Available: http://wiki.ros.org/ROS/Tutorials

[21] ROS tutorial on message filters property. [Online]. Available: http://wiki.ros.org/message_filters

[22] Z. B. Rivera, M. C. De Simone, and D. Guida, "Unmanned ground vehicle modelling in gazebo/ROS-based environments," *Machines* vol. 7, no. 2, 2019.

Shreyash Patil received the electronics and telecommunication engineering degree from Rashtrasant Tukdoji Maharaj Nagpur University, India, in 2018. Currently, he is pursuing a master of technology in VLSI and Embedded systems from Dr. Vishwanath Karad MIT World Peace University, Pune, India. He worked as a project intern at the Automotive Research Association of India in Pune, India in the year 2020-21 for project work based on ROS 1 for autonomous driving of TurtleBot3.

Kishanprasad Gunale received a Ph.D. degree in electronics and telecommunication from SPPU, Pune. Currently, he is working as an asst. professor at School of Electronics and Communication Engineering at Dr. Vishwanath Karad MIT World Peace University, Pune, India. His areas of research is computer vision, automotive electronics. He is a member of SAE, IETE, and IET. To his credit six journal publications and six conference publications and two published patents are present.