Application of Image Processing Techniques for Autonomous Cars

Shaun Fernandes, Dhruv Duseja, Raja Muthalagu*

Department of Computer Science, Birla Institute of Technology and Science Pilani, Dubai Campus, Dubai, UAE

Received 16 July 2020; received in revised form 28 October 2020; accepted 17 December 2020

DOI: https://doi.org/10.46604/peti.2021.6074

Abstract

This paper aims to implement different image processing techniques that will help to control an autonomous car. A multistage pre-processing technique is used to detect the lanes, street signs, and obstacles accurately. The images captured from the autonomous car are processed by the proposed system which is used to control the autonomous vehicle. Canny edge detection was applied to the captured image for detecting the edges, Also, Hough transform was used to detect and mark the lanes immediately to the left and right of the car. This work attempts to highlight the importance of autonomous cars which drastically increase road safety and improve the efficiency of driving compared to human drivers. The performance of the proposed system is observed by the implementation of the autonomous car that is able to detect and classify the stop signs and other vehicles.

Keywords: image processing, autonomous cars, self-driving, object detection

1. Introduction

Image processing is a field in a computer that deals with processing and extracting meaningful and useful data from images. Since Images represent unstructured data, they are significantly more difficult than processing structured data such as tables and forms. Depending on the application, image processing can be applied in numerous fields, and via many different methods and algorithms. It is a large and growing branch that is currently in use in multiple areas such as the medical field, Video Processing, Pattern Recognition, Machine/Robot Vision, Driverless cars, etc.

In a recent paper by Viswanathan and Hussein [1-2], much study was done in the field of image detection for Autonomous cars. Features such as lane detection, traffic signal detection, and speed bump detection were studied and the method of implementation also were reported in by many researchers around the world. The lane detection system prescribed here talks about a few major steps involved. Selecting the region of interest (ROI) and discarding unwanted data, then image pre-processing, Canny edge detection, and Hough transforms applied on the final output. As mentioned, these were methods originally used in lane detection, but in more recent times, spiking neural networks have become popular, being able to detect edges faster and more efficiently than traditional methods. Traffic signal detection was achieved by looking for a certain shape in the image, and then looking at the color in that image if it was found. The paper also proposes a steer by wire implementation in cars, where instead of having the drive shaft directly connected to the axil, it added an electronic control unit (ECU) in between. This ECU would then take user steer as input and based on external conditions decide whether to execute the command to steer or not. Using these methods would prove extremely helpful in future implementations of autonomous cars.

An interesting collision detection system is proposed by Guha and the team [3-4]. This system aims to focuses on other objects along the road instead of looking for lanes, such as stop signs, Traffic lights, and taillights of cars ahead are looked for. upon seeing those, instructions are given to the car to stop. It has been implemented using HARR cascades, which provide light

* Corresponding author. E-mail address: raja.m@dubai.bits-pilani.ac.in
Tel.: +971-562249227886-5

weight and efficient solution to the problem of object detection. Using multiple images of positive examples of their objectand many more negative examples, they were able to accurately classify these important objects and react accordingly. After calibrating the camera, calculated the camera tilt angle, and considered its height off of the ground, they were also able to predict the distance of the object with a fair amount of accuracy.

Shah, Adhikari, and Maheta [5-6] were also able to produce similar results while attempting to tackle the same problem. Edge detection was implemented for lane detection. But for street signs like left, and right arrows, matrix matching algorithms were used instead of HARR cascades, and similar results were shown. This goes to show that different algorithms may produce similar output and that each should be tested, in order toselect the best on a case by case basis.

Xue Li [7] and the team worked on drastically improving the efficiency of the edge detection algorithm. Instead of using the conventional means of Canny edge detection, which many claim to be slower and at times inaccurate as a means of highlighting edges in the image, the team decided to use a spiking neural network. Spiking neural networks are built to emulate real neural networks more precisely than simple MLP's. They do this by not letting all neurons fire at the same time but instead firing only when a certain threshold is met, and they also incorporate time into the model. The output of the edges from the SNN was passed through a Hough transform and using the SNN, Xue Li was able to gain accuracy and efficiency far greater than conventional methods of Canny edge detection. Consequently improve efficiency of the lane detection algorithm overall. Once the input images are preprocessed, features that define a lane like colors and edges can be extracted using color space exploration and edge detection algorithms like Canny edge detector [8]. The Hough transform algorithm [8], is one of the most widely used algorithms for lane detection in previous studies.

Conventional vision-based lane detection techniques can be roughly divided into two main classes, feature-based techniques [9] and model-based techniques [9]. Feature-based techniques mainly rely on the detection of features describing lane markings such as lane color, texture, and edges. Most of the previous works used a technique like Sobel filters or canny edge detectors to detect the lanes using pixel intensities and edge information with lane markings. The lane markings are being extracted using different preprocessing steps. Instead of using the RGB color images directly, the researchers have converted the RGB images to other color formats. The general idea of color space transformation being that the colors most used for lane markings, yellow and white, are more distinct in other color domains. Once the edges are detected, then they are combined using a lane hypothesis which can be used to filter out outliers. In general, feature-based methods have better computational efficiency and can accurately detect lanes when lane markings are clear, but quickly lose robustness depending on many factors.

A weakly-supervised adversarial domain adaptation is implemented to improve the segmentation performance from synthetic data to real scenes, which consists of three deep neural networks [10]. The detection and segmentation (DS) model focus on detecting objects and predicting a segmentation map; a pixel-level domain classifier (PDC) tries to distinguish the image features from which domains; and an object-level domain classifier (ODC) discriminates the objects from which domains and predicts object classes [11]. Siamese fully convolutional networks (FCNs) (named "s-FCN-loc"), which can consider RGB-channel images, semantic contours, and location priors simultaneously to segment the road region elaborately [11]. Vision technology is implemented that allows traffic engineers to analyze traffic safety based on image processing applications [12-13]. In this paper, a system for image processing in autonomous cars has been implemented by applying a multistage preprocessing to the images for lane detection, the lanes can be detected quickly and accurately. The implementation of the lane detection model has been created and was in a working state but was not integrated with the current model. It was designed to be used as an aid for a future autonomous car model that run on real roads. The output of the model could be passed as a secondary input to the neural network to aid it in detecting lanes and making decisions

The remainder of the paper is organized as follows. Section 2 describes the problem formulation and objective. Section 3 describes design methodology. Section 4 describes the lane detection by using image processing techniques. Experimental results of the object detection are presented in Section 5. Finally, section 6 concludes our work.

2. Problem Formulation and Objective

One of the greatest causes of death, besides diseases, is car accidents. According to a recent survey [14], 1 out of 80 people, or approximately 1.2%, die in a car accident. Most of these accidents can be attributed to human error. Some of the major causes of road deaths are distraction, over speeding, drunk driving, and recklessness. Autonomous cars aim to provide a solution to this massive problem of loss of life from car accidents. Since these systems are programmed to drive efficiently and safely, they minimize, if not, eliminate the need for a human driver, they would also eliminate the aforementioned human errors. This would lead to a significantly safer driving experience and would hence decrease the road accident rate. This project aims to take a step in the right direction and move from traditional human-driven cars to fully autonomous cars that make driving much safer.

This paper aims at multiple objectives. First, all hardware components of the wheel-e car have to be tested and calibrated to make sure that everything runs smoothly. The wheel-e, which is being interfaced via an Arduino, had to communicate with the computer via Bluetooth and a program written to give different driving instructions to the car from the computer. This would then be tested to see if all controls sent from the computer were properly received and executed by the car. Having the controls sent directly from the computer would later make it easier to gather training data for the neural network that will control the car.

Unfortunately, due to limitations on the processing and storage capabilities of the Arduino, the idea of using the wheel-e was canceled. Instead, a simple remote-controlled car was used for this paper. The remote controller and the internal wiring have to be turned on to be able to control by an Arduino. Wires has to be soldered onto the controller's circuit board and then connected to the Arduino. The Arduino has to be interfaced to the computer via a python program, allowing controls to be sent from the computer and directly executed by the R/C car.

Lane detection is a very important step as the car moves along the road, it will need to know exactly where the lanes are, so it knows the boundaries and can obtain information on how to steer the car. To do this, the images sent from the car had to be cropped to display only the region of interest (ROI), preprocessed to increase accuracy, Canny edge detection applied on the image to only display the edges, and finally, Hough transform used to detect and mark the lanes immediately to the left and right of the car.

Besides lanes, other objects also has to be recognized using image processing techniques: stop signs which would tell the car to wait for a few seconds. Other cars in front, which pose as obstacles, also needed to be detected to avoid collisions. An algorithm to measure the distance to each of these objects. All these objects were classified using haar cascades, which is fast and efficient, and can detect objects no matter how big or small and no matter where they are placed in the image. The haar cascades need to be trained on many training images from each class, and the images has to be taken from many different points of view. A lot of images of negative examples has also been provided to properly train the classifier. The distance measurement of the classified objects was done using monocular methods. Having the camera calibrated previously, and measured the width of the objects used, distance to these objects could be measured and could provide useful information for the autonomous car.

The image classifiation need to be integrated along with the car. In this final stage, the autonomous car would be set up with a camera that would constantly stream live videos of the road to a computer with the image processing software installed. The Lane detection system would aid the car, staying in lane and steering in the right direction. Upon detecting a stop sign at a

certain distance the car will be told to stop for a few seconds before proceeding with caution. And detection of cars ahead would tell the car to slow down or stop completely to avoid collisions. Using these images processing tools provide much-needed aid to autonomous cars in making decisions.

3. Design Methodology

3.1. Hardware implementation of wheel-e

This work initially made use of a wheel-e as its base. The wheel-e is an Arduino controlled model car. The model is composed of a large chassis base, 2 DC motors, 2 large wheels, optical interrupt sensors, a small castor wheel at the front, a battery holder for 4 batteries, breadboard, dual H-Bridge motor driver, Arduino Camera, Bluetooth module, and an Arduino Uno as shown in Fig. 1.

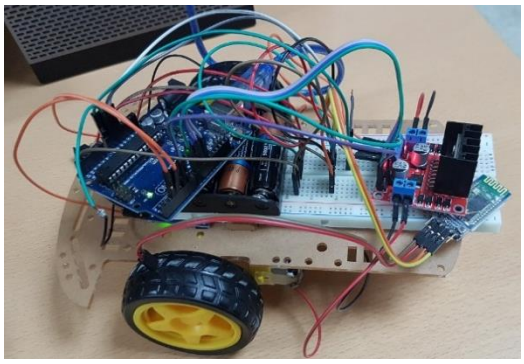

Fig. 1 Arduino based wheel-e car

The chassis base is a large precut acrylic base that is specifically designed with the wheel-e in mind. The 6-volt DC motors are the powerhouses of the system, they drive the car forwards. Since no two motors are identical, these motors spin at different speeds even when supplied with the same amount of power, and this is a problem that needs to be solved later. These motors attach to 2 large plastic wheels, and each wheel has an optical encoder wheel. By reading how many times the slots of the optical encoder wheel have passed by the optical interrupt sensor, we can determine how much the wheel has rotated and how much distance has been covered by the car. The castor wheel is not electronically powered that means it cannot be rotated. It is simply used to provide support and stability to the car. The Battery holder is capable of holding 4x 1.5-volt batteries, supplying the motors and the Arduino with up to 6-volts of power.

The L298N Dual H-Bridge motor driver is used to interface the Arduino with the motors. It provides the DC motors with power, receives and executes commands for speed and PWM sent from the Arduino. The main function of the H-Bridge is to provide the ability to switch polarities of the wheel input and hence change the direction of rotation of the wheel from forwards to backward. The Arduino Camera is one of the most important parts of this project. It is used to get a live feed of images taken from in front of the wheel-e. The Bluetooth module is used to relay images back to the computer and to receive directions on where and when to turn from the computer.

3.2. Rectifying steering problems of wheel-e

We had initially started with only the model of the wheel-e and the base code. The first thing to be done was to learn how to use and program the Arduino. Once the basics of Arduino coding were understood, many changes need to be made. The existing model had a few flaws that have to be corrected to be able to properly work with the model in the later stages of the project. The first stage of work was to set up the hardware properly and to ensure that it work correctly.

The main thing that need to be corrected was the turning of the Arduino. Since no two motors are alike, and many factors such as the thickness of connecting wires, build of the motors, the strength of the motor magnets, etc., affect their speed.

Because of this, the motors were never spinning at the same speed. This posed a problem since the car would tilt to the side when it was supposed to go straight and would make sharp right turns and very wide left turns. These irregularities in turning would affect the car later when it was being controlled autonomously since the control algorithm would assume a perfectly stable and symmetrical turning system.

To address this problem the optical interrupt sensors need to be used. Since every time the wheel turned, it also turned the optical encoder wheels which had 20 holes in it, the optical interrupt sensor was able to read every time a new hole passed by the sensor. It would then send an interrupt to the Arduino which would count the number of pulses it received from the sensor. The following is the general formula for calculating rpm:

$$rpm = \frac{1}{times\ in\ \min utes} \times \frac{No.\ of\ \ pulses}{pulses\ \ per\ turns} \tag{1}$$

But since the Arduino provides time in milliseconds, and the number of pulses per turn is 20 (20 holes in the encoder wheel), the modified formula bellow was uses instead:

$$rpm = \frac{60 \times 1000}{times\ in\ \min utes} \times \frac{No.\ of\ \ pulses}{20} \tag{2}$$

Now being able to accurately calculate the speed that each wheel is spinning at, we could proceed to adjust the power sent to the motors. This would be used to compensate for one motor is faster than the other. This was done by lowering the PWM for the faster motor (*b*) till it match the speed of the slower one (*a*). We used a proportional method for adjusting the speed of the motors. The proportion of the speed of one motor to another was calculated. (assume *a* to be the slower motor)

$$percentage\ difference_b = \frac{RPM_a - RPM_b}{RPM_b} \tag{3}$$

The PWM of the slower motor was kept at its maximum, and PWM of the faster motor was reduced at first, after that, it constantly readjusted proportional to the difference between the two motor speeds to try and make it match the exact speed of the slower motor.

$$new\ PWM_b = PWM_b + (PWM_b \times percentage\ difference_b) \tag{4}$$

By implementing this function, we were able to adjust the motor speed of the car to match each other with some accuracy. This function is constantly being called, hence, the car is always constantly, dynamically adjusting its speed to be able to drive better than it did before. This solution is unfortunately not enough to get the car to drive straight. The car would instead of following a straight line, veer off to one side or the other. Even slight differences in a rotation of one wheel would set the car oversteering to the opposite direction. Once the speeds for the motors were adjusted and calibrated, the next step was to create a turning function. This function simply took in as arguments which direction the car should turn in and how much it should turn. Turning was then achieved by having both wheels moving forward but having one wheel move a certain degree faster than the other wheel. So, if we want a sharp left turn, we make the right wheel spin much faster than the left wheel, on the other hand, if we want a wide turn right, we would make the left wheel spin lightly faster than the right wheel.

Even though the steering had been slightly improved, there were other unforeseen problems. These came mostly from the camera part. As the Arduino is a very small and low powered microcontroller, there was a severe lack of required memory and processing power. It was unable to handle taking and storing many images, as it lacked the memory needed. Since the processor was also lacking power, any pictures that were taken would take time to be processed and sent over Bluetooth. This

caused a large lag over half a second, which would not allow for real-time image processing and hence was the largest bottleneck in the control pipeline. Another drawback is that the Arduino would often not respond to the commands sent over Bluetooth. The commands would be passed but it was noticed that for the backward and forward command, the Arduino would not respond until it was restarted. Without the ability to reliably send steering commands to the car, the entire project would be brought to a standstill. Hence the idea of using the Arduino-based wheel-e was dropped and instead a remote-controlled car was used.

3.3. Hardware implementation of remote-controlled car

As the wheel-e had proven to be a failure for the purposes of this project, the R/C car was selected as its replacement. The project required the car to be controlled remotely from the computer, and so many changes would have to be made to be able to achieve such control. Firstly, the remote control of the R/C car was opened. Inside was a circuit board (Fig. 2) that consisted of its IC, antenna, and 4 x 4-pin push buttons, one button each for forward, backward, left and right [15].

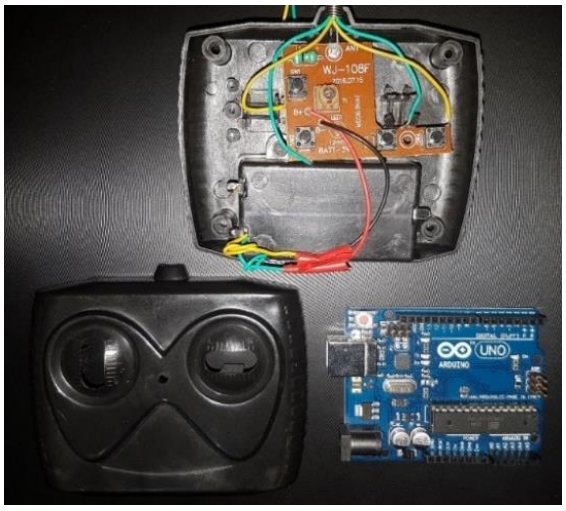

Fig. 2 Arduino and remote controller after soldering wires

Using the Arduino and an LED for testing, all 4 pins of all 4 push buttons were tested and the different kinds of pins were found. On an average 4 pin push button, 2 adjacent pins on one side are connected as are the 2 pins opposite to them. Current only flows from one side to the other when the button is held down. After finding which pins were always on, and which pins would only turn on when the button was held down, it was realized that sending a high signal to one of those pins would act the same as physically pressing the button. So, 4 wires were soldered, and each one on the side of the button that turned high upon button press and was low otherwise. This allowed simulating button presses for the controller's forward, backward, left and right commands simply by providing a high signal from an Arduino to the corresponding pin of the controller.

3.4. Interfacing the car with the computer

After setting up the hardware connections from the Arduino to the controller as in Fig. 3, the software side of things needed to be implemented. The Arduino program was straightforward, simply responding to certain input in the serial monitor and performing the corresponding task ('f' for forward, 'l' for left, 's' for stop, and so on).

Once the Arduino was set up with code to steer the car based on commands received, the whole setup had to be interfaced via python, and optimized there. To be able to control the car, the program needed to be able to read multiple key presses simultaneously, remember which keys are currently being held down, and send an appropriate message to the Arduino which drives the car. To get multiple keypresses at once, the *pynput* python keyboard input library was used. Multiple threads were created for each function, so they could all run efficiently side by side. This also allowed the program to keep track of multiple distinct key presses simultaneously. The commands were then sent to the Arduino using the python *Serial* library.

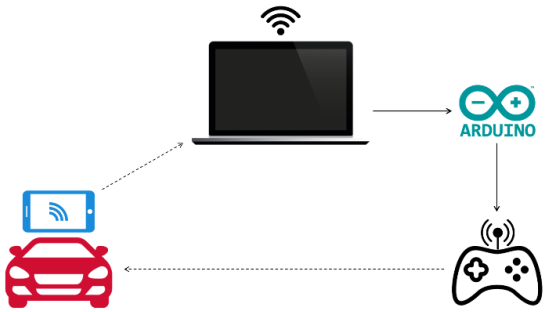

Fig. 3 Proposed design architecture

After having completed the Arduino setup, the next task is to stream images from the car to the computer, which would then be processed, sending drive instruction to the car. To get these images, an android phone was used. A mobile application allowed a video stream to be created on the local network, so as long as the computer and the phone were connected to the same network, they could communicate with each other. The PC would then connect to this stream using the python *OpenCV* library. This same library could also convert the video stream into individual frames of images, a necessary step for image processing.

4. Lane Detection by Using Image Processing Techniques

Lane detection is one of the most important aspects of an autonomous car. A camera, usually installed in the center of the dashboard behind the windshield, provides a constant stream of images of the road ahead. Lane detection is necessary, as it provides the car with information about whether it is maintaining lane discipline, and the angle of the lanes can tell it about upcoming turns. It is responsible for keeping the vehicle in the middle of the lane at all times.

The process implemented goes through a few steps:

- Gray scale conversion
- Yellow and white filter
- Gaussian blur
- Canny edge detection
- Region of interest
- Hough transform

4.1. Gray scale conversion

The image provided by the camera will likely be in color, but most of the information about the lanes can be found from a gray image itself. As shown in Fig. 4, the original image will have 3 dimensions of information: red, green, and blue. But a grayscale image contains only a single dimension: intensity as in Fig. 5. This reduces the size of the data being worked with a third of its original size. Such a large amount of improvements in efficiency is achieved by losing small amount of information lost by choosing grayscale image.

Fig. 4 Original image used as an input

Fig. 5 Gray scale converted image

4.2. Yellow and white filter

Roads all over the world employ a standard color scheme for lane marking. All lanes are either pure white or bright yellow. This simplifies the process of lane detection. Instead of looking at the entire image, only those parts that are yellow or white can be filtered out and further analyzed. To get the yellow regions of the image, it is first converted to HSV (hue, saturation, value) format. This is done because it is difficult to obtain the correct range for bright yellow when using the red, green, blue format. Once converted, a mask is created containing only yellow portions of the HSV image, and another mask containing only White portions. These masks contain the original image where their respective colors are present, and the rest of the pixels of the image are set to black. These two masks are then combined by using a bitwise and operator. Finally, the combined mask is applied on the grayscale image and all that remains are white and bright yellow parts of the image as in Fig. 6.

Fig. 6 Output image after applied white and yellow filter

4.3. Gaussian blur

Before applying edge detection on the image, it needs to be processed further. At this stage, the image is at a fairly high resolution, so small flaws in otherwise straight lines would be recognized by the algorithms. Hence these images need to have some amount of detail taken away, which is achieved by a Gaussian blur. The Gaussian blur algorithm reduces the resolution of the image by setting the value of a pixel as the average of the value of all the pixels around it. This helps suppress noise that may be present in the image, and smooth out lines and make them appear straighter as in Fig. 7.

Fig. 7 Output image after applied Gaussian blur

4.4. Canny edge detection

Now that the image has been completely pre-processed, edge detection can be applied to it. Even at this stage, the image has a little too much data to use the Hough transform on as shown in Fig. 8. So, to optimize it even further, only the edges of the image will be taken. An edge is any point where the average intensity of pixels on one side is drastically different when compared to the average intensity on the other side. The Canny edge detection algorithm works by applying a few filters on the image and then calculating the directional gradient of each pixel and checking whether it satisfies the criteria for an edge or not. Compared to other strictly defined methods, Canny seems to be one of the best when both speed and accuracy are concerned.

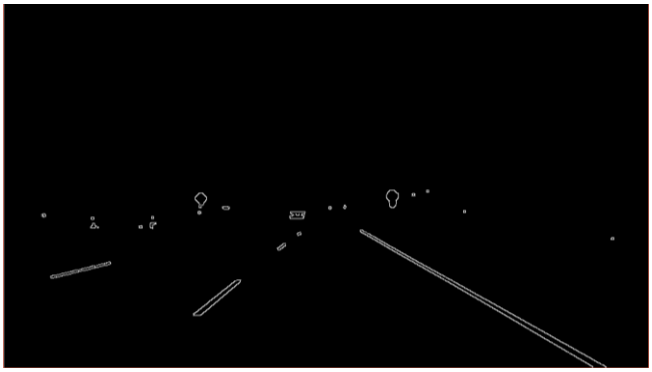

Fig. 8 Output image after applied Canny edge detection

4.5. Spiking neural networks for edge detection

While Canny edge detection is one of the most popular algorithmically defined methods of edge detection, there is still room for improvement. Using a spiking neural network is a much more efficient method of edge detection. Spiking neural networks are built to emulate real neural networks more closely than simple MLP's. They do this by not having all neurons fire at the same time but instead firing only when a certain threshold is met, and they also incorporate time into the model. Xue Li and team [4] trained a spiking neural network and were able to achieve very high accuracy and speed while a SNN was not used for the purpose of this work, it still remains one of the best methods to improve the currently standing algorithms and is highly recommended for future works.

4.6. Region of interest

While the image provided by the camera is necessary for lane detection, a large region of the image is unimportant. The only region that is required is the road straight ahead, and the immediate flanking lanes as shown in Fig. 9. The rest of the image consists of vehicles ahead, the horizon and the sky above it, and lanes other than the one the car is currently driving on. Discarding all this unhelpful data from the image, at an early stage helps save a lot of computation time for the onboard computer, which can then be used by other important processes. Also, it helps the further algorithms pinpoint the location of the lanes instead of looking at other unnecessary parts of the image.

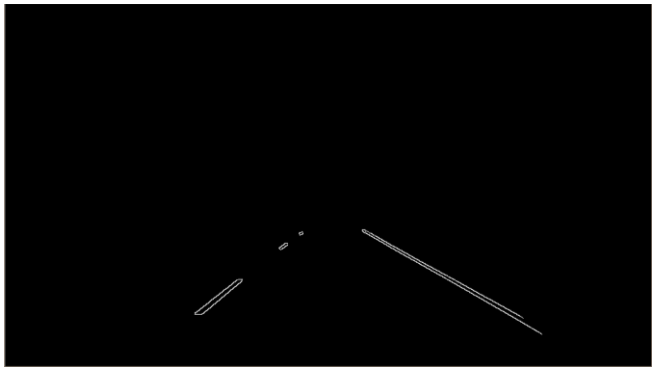

Fig. 9 Region of interest

4.7. Hough transform

A Hough transform is used in the end to get the two lanes out of the image of edges. The Hough transform is a mathematical function that is used to find the lines in an image. It transforms all lines into points in the Hough space, and points into lines, and these points are calculated by the following equation.

$$r = x\cos\theta + y\sin\theta \tag{5}$$

Observing the curves produced in the Hough space will show that there are 2 major clusters of points of intersection in the Hough space, representing all the lines that comprise the 2 major lanes on either side of the road. These lines have been plotted onto the original image as illustrated in Fig. 10.

Fig. 10. Hough transform lines

Fig. 11. Lanes output from Hough transform

The image shows about 6 lines on the left lane and about 7 on the right. Each of these groups needs to be considered separately, but the number of lines must be reduced to one each. This can be achieved by taking the average of each set of Hough lines produced. This leaves a final image as illustrated in Fig. 11 that considers the average length and direction of each line. These lines may be extrapolated to account for areas of the road where there are no lanes.

5. Results and Discussion

5.1. Detection of stop sign and distance measurement

While lane detection is important for an autonomous car to be able to drive according to the rules and maintain lane discipline, it is also important for them to be able to recognize street signs. Street signs have been around for decades as they provide an easy, fast, and convenient way for a driver to look at instructions and warnings even when driving at high speeds. Such signs often provide very valuable information to the driver and could similarly aid an autonomous car as it drives, especially along smaller, unknown roads. As a proof of concept, this project implements the detection and response to a stop sign. As the R/C car moves along the specified path, its attached camera keeps a lookout for such signs and obstacles. Upon seeing a stop sign, it calculates the distance to the sign, based on previously done calibration, and stops if it is within a certain distance of said sign. After stopping for a few seconds, it then proceeds with caution.

Fig. 12 Stop sign detection with distance measurement

To train the stop sign classifier, images were taken of a small, cutout stop sign as shown in Fig. 12. These were taken from many different angles to make the classifier robust to changes in camera angles. They were then cropped to only contain the desired region. Many negative samples were also provided to the classifier, so it knew what and what not to look for. The classifier was based on haar cascades. The haar cascade classifier learns by superimposing the positive samples over the negative samples and generating features out of the superimposition. The training algorithm creates thousands of features and

then applies each of these features over every pixel of the image. Only those features that have a low loss value when classifying the given set of images are retained, while the rest are discarded. In the end, those features are combined in the form of a cascade. Larger, general features are kept higher up and checked sooner, while smaller, specific features are low in the tree and checked only if the ones above them proved successful.

Along with being able to classify a stop sign for a given image, the distance to the stop sign also needs to be calculated. Only once the car came within a certain radius of the stop sign was it supposed to stop; hence distance was necessary. To calculate the distance, a monocular method was implored as 2 cameras could not be fit onto the car. There is a direct correlation between actual size, perceived size, the focal length of the camera, and the distance. Simply using geometric triangle similarity, it is known that the ratios of the sides of two similar triangles are equal [16]. During calibration, the focal length was found by obtaining concurrent measurements using the following equation.

$$Focal\ length = \frac{width\ in\ pixels \times dis\tan ce\ \ from\ camera}{actual\ \ width\ of\ \ objects} \tag{6}$$

Once the focal length was obtained, all that need to be done was rearrange the formula so that the distance could be calculated as follows

$$Dis\tan ce\ \ from\ camera = \frac{focul\ \ length \times actual\ \ width}{width\ in\ \ pixels} \tag{7}$$

While this method is not perfect, it is able to provide a rough estimate of the distance of the object. This method is also resistant to changes in the angle of images, as long as the horizontal angle is not changed by much.

5.2. Vehicle detection with distance measurement

Besides stop signs, the algorithm was also trained to recognize vehicles. In this case it was trained on one specific car that would be used as an obstacle for the autonomous car. The algorithm again made use of haar cascades as its classifier. Using many positive examples of the vehicle, and a set of negative examples without the vehicle. The classifier was trained to recognize the car with high accuracy, using the same monocular vision method of triangular similarity, the distance to the car was also measured, and the camera had to be recalibrated on the new object. The result is illustrated in Fig. 13.

Fig. 13 Vehicle detection with distance measurement

6. Conclusions

A system for image processing in autonomous cars has been implemented. By applying a multistage pre-processing to the images for lane detection, the lanes can be detected quickly and accurately. Selecting the region of interest as the first step saves at least 2-3 times the amount of processing required. Selecting only white and yellow regions further reduces processing

time and drastically improves the detection of the relevant edges. Classification of stop signs and detection of vehicles ahead was also achieved. This would be greatly useful for recognizing and responding to not only road signs but also vehicles ahead.

Not only were these objects recognized, but their distance from the camera was also calculated. The algorithm responds to a stop sign by stopping the car for a few seconds if it is within a certain distance from the stop sign. After the wait time had elapsed, it continues along. It also responds to vehicles ahead by stopping completely if it gets too close. These implementations ensure that the autonomous car follows the rules of the road, and does its best to prevent accidents, preventing loss of life. In the future, the proposed algorithm can be further trained to recognize and respond to many more road signs.

Conflicts of Interest

The authors declare no conflict of interest.

References

[1] V. Viswanathan and R. Hussein, "Applications of Image Processing and Real-Time Embedded Systems in Autonomous Cars: A Short Review," International Journal of Image Processing, vol. 11, no. 2, pp. 35-49, April 2017.

[2] J. V. Brummelen, M. O'Brien, D. Gruyer, and H. Najjaran, "Autonomous Vehicle Perception: The Technology of Today and Tomorrow," Transportation Research Part C: Emerging Technologies, vol. 89, pp 384-406, April 2018.

[3] J. Joel, G. Fatma, B. Aseem, and G. Andreas, "Computer Vision for Autonomous Vehicles: Problems, Datasets and State of the Art," Foundations and Trends® in Computer Graphics and Vision, vol. 12, no. 1-3, pp. 1-308, July 2020.

[4] Y. Xing, C. Lv, L. Chen, H. Wang, H. Wang, D. Cao, et al., "Advances in Vision-Based Lane Detection: Algorithms, Integration, Assessment, and Perspectives on ACP-Based Parallel Vision," IEEE/CAA Journal of Automatica Sinica, vol. 5, no. 3, pp. 645-661, May 2018.

[5] A. Jain, H. Reddy, and S. Dubey, "Automated Driving Vehicle Using Image Processing," International Journal of Computer Science and Engine and Engineering, vol. 2, no. 4, pp. 138-140, May 2014.

[6] R. Muthalagu, A. Bolimera, and V. Kalaichelvi, "Lane Detection Technique Based on Perspective Transformation and Histogram Analysis for Self-Driving Cars," Computers & Electrical Engineering, vol. 85, 106653, July 2020.

[7] X. Li, Q. Wu, Y. Kou, L. Hou, and H. Yang, "Lane Detection Based on Spiking Neural Network and Hough Transform," 8th International Congress on Image and Signal Processing (CISP), October 2015.

[8] Y. Huang, Y. Li, X. Hu, and W. Ci, "Lane Detection Based on Inverse Perspective Transformation and Kalman Filter," KSII Transactions on Internet and Information Systems, vol. 12, no. 2, pp. 643-661, February 2018.

[9] J. M. Yang and S. H. Wei, "Spatial and Spectral Nonparametric Linear Feature Extraction Method for Hyperspectral Image Classification," Advances in Technology Innovation, vol. 2, no. 3, pp. 68-72, July 2017.

[10] Q. Wang, J. Gao, and X. Li, "Weakly Supervised Adversarial Domain Adaptation for Semantic Segmentation in Urban Scenes," IEEE Transactions on Image Processing, vol. 28, no. 9, pp. 4376-4386, September 2019.

[11] Q. Wang, J. Gao, and Y. Yuan, "Embedding Structured Contour and Location Prior in Siamesed Fully Convolutional Networks for Road Detection," IEEE Transactions on Intelligent Transportation Systems, vol. 19, no. 1, pp. 230-241, January 2018.

[12] W. Suh, J. R. Kim, and J. Cho, "Vision Technology Based Traffic Safety Analysis Using Signal Data," Proceedings of Engineering and Technology Innovation, vol. 13, pp. 20-25, July 2019.

[13] W. H. Suh, J. W. Park, and J. R. Kim, "Traffic Safety Evaluation Based on Vision and Signal Timing Data," Proceedings of Engineering and Technology Innovation, vol. 7, pp. 37-40, December 2017.

[14] Injury Facts, "Historical Fatality Trends," https://injuryfacts.nsc.org/motor-vehicle/historical-fatality-trends/deaths-and-rates/, October 23, 2018.

[15] J. Wan, "Working of a 4-pin Push Button," http://www.jobiwan.net/XFP/button.png, December 10, 2018.

[16] T. Emara, "Pinhole Camera Distance Measurement," http://emaraic.com/assets/img/posts/computer-vision/distance-measurement/pinhole-camera.gif, December 10, 2018.