Lane Detection and Object Detection algorithms for Self-Driving RC Car

Done by: Abdelaziz Al-Najjar ID: 2269512
Department: Electrical and Electronics Engineering
Middle East Technical University, Northern Cyprus Campus
Email: e226951@metu.edu.tr

Advisor: Dr. Cem Direkoğlu
Department: Electrical and Electronics Engineering
Middle East Technical University, Northern Cyprus Campus

Keywords: ***Self-Driving Car***
RC Car
Computer vision
Lane detection
Object detection
Haar features
Hough transform
Machine learning
Histogram

ABSTRACT:

With the increase in numbers of car traffic collisions caused by the human driver, many car companies are now moving towards developing intelligent vision systems to help the car navigate itself safely. These systems are mainly concerned about two things, detecting objects around the car and keeping the car between the lanes. For detecting objects, most systems include sensor subsystems that surround the car, such as lidar, sonar, IMU, and odometry which can be costly and not efficient since these sensors alone cannot fully identify the objects and extract information from surroundings, such as colors in a traffic light, reading signs…etc. In this work, we addressed these issues by developing algorithms for detecting objects that surround the car using machine learning and Haar feature-based cascade classifier. Also, this work includes algorithms for lane detection using Hough line transform and Canny edge detection and improves these algorithms by using histogram method for identifying the lanes. Moreover, these algorithms are optimized to work on a Raspberry Pi 3 B+ as the master device which will be responsible for sending information to the Arduino UNO which will be responsible for controlling the motors of the RC car.

I. INTRODUCTION:

The existence of cars started to become dangerous in man-kind life, especially with the massive increase in cars. In 2019, The Insurance Information Institute believes that this increase result in 6.8 million car accidents in the United States alone, 1.9 million of these accidents result in injury or fatality [1]. According to studies by [2], 90% of these accidents are caused by human errors. From this point, many Intelligent Transportation Systems (ITSs) have been developed to reduce this number and increase road safety. ITS is an active research area that includes tasks like lane detection and object detection and collision prevention.

The lanes are a very important part of the car transporting system since it helps to control and guide drivers and reduces the collisions that could happen due to traffic conflicts. Therefore, lane detection plays a very important role in any ITS related system, especially for improving self-driving car performance.

The objective of the lane detection algorithm is to separate the lanes from the background and identify their specific location using hardware devices or computer vision. The proposed lane detection algorithm uses machine vision techniques to detect the lanes and identify the left and right lanes. The algorithm will also be able to send the result of lanes status to the car motors control subsystem and specify if the lanes are going right, left, or forward.

Another aspect of this work is concerned about is car surroundings and object detection. The self-driving car needs to read the traffic signs and perform the necessary action. For example, stop the car in case of a red traffic light, move when green traffic light. stop the car temporarily when stop sign, and not to exceed the speed limit…etc. More importantly, the car should also detect if there is an obstacle in front of it, such as cars, humans, cats, dogs… etc. This obstacle detection can be done using hardware (sensors) or software (machine vision) techniques.

The proposed algorithm uses machine learning algorithms to train models for each object and then apply these models to a program that will be connected to the HD camera of the car. The algorithm will also send information to the motor control subsystem which then will perform the necessary action.

For the sake of testing these algorithms, they are optimized to be applied on a demo RC car that will include a Raspberry Pi 3 B+ as the master device that gets the frames from the HD camera on top of the car, and then sends the result of the applied lane or object detection to the slave subsystem which is represented by Arduino UNO which will then control the motors and perform the necessary action.

This paper will start by giving an overview of the system in section 2, then will describe the algorithm for lane and object detection in sections 3 and 4 respectively. In section 5 the results will be given. Lastly, the deductions will be given in section 6.

II. OVERVIEW OF THE MACHINE VISION SYSTEM

The system of the machine vision for the self-driving car starts by getting the frame image of the road shown in fig.1 from the RPI HD camera that is installed on the car. After that, the Raspberry Pi will be dedicated to doing two main operations in parallel.

The first operation is lane detection, where the image will go through multiple stages of analyzing and manipulating the image, then send the result to the Arduino that will be responsible to perform the necessary action with controlling the physics of the car in our case.

The second operation is object detection, where the objects in the image such as cars, stop signs, traffic light, persons, cats, and dogs will be detected. The detected object name along with its distance from the camera will be sent to the controller to do the necessary operation with the car.

Fig. 1 a road example image for lane and object detection

III. LANE DETECTION

A. Related work

[3] proposed a method that uses two-stage feature extraction. Their approach includes extracting the line segments and the line segments clusters. Afterward, it will go through an Identifying lanes stage where the lane model and curve fitting will be applied.

[4] proposed a method that makes use of the deformable template model to the expected lane boundaries in the image.

B. Used algorithm

The idea used for the lane detection approach in this work is much simpler compare to the related work shown in the previous section and can be explained as follows:

Step 1: RoI generation (Fig. 2)

Considering the Raspberry Pi capabilities and to reduce the unnecessary details in the frame, the idea was that the RC car is not concerned about knowing the status of the lane after 30 meters. Instead, it is concerned about the current lane status and what is the information that needs to be sent to the Arduino right now for the coming 5-15 meters based on the camera position and quality. Hence, the algorithm starts by generating a Region of Interest (RoI) that will be used for detecting the lanes.

Step 2: Warping the RoI and removing noises (Fig. 3)

Using the ready OpenCV warp function on the RoI will ease the process of detecting lanes by removing unnecessary details of the frame. On the grayscaled image, apply modified gaussian blur to remove noises from the RoI image.

Step 3: Canny edge detection (Fig. 4)

Using Canny detector function available in OpenCV and modifying it to have a low threshold and detect all edges in the warped image.

Step 4: Dilating the edges

To avoid undetected lanes of the image in the case of bad lighting conditions, the edge images will go through a dilating procedure.

Step 5: Hough Transform (Fig. 5)

Now that the edges are dilated and clear, it is safe to apply the Hough line transform which will detect the left and right lanes as a binary image.

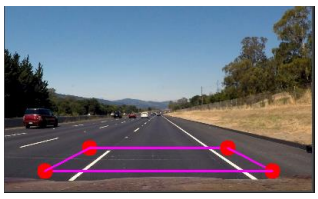
Fig. 2 The frame with the RoI

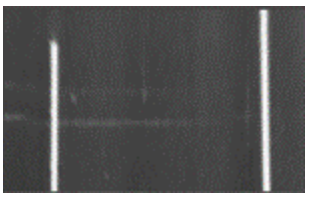
Fig. 3 warped RoI


Fig. 4 Canny edges of warped image


Fig. 5 Detected lanes using Hough line

Improvement on the algorithm:

Applying Histogram:

Now that the lanes are detected, all that is left is to define these lanes as if it was the left or right lane, and if the road is going right or left, and what information will be sent to the controller. For that, the frame will be divided vertically into two halves. Using the histogram method to define the maximum value for each half and store its position. The maximum value in the left half of the frame will represent the left lane position, and the maximum value of the right half will represent the right lane position.

Applying the left and right lanes positions to the following formula to get the lane center:

$$lane\ center = \frac{left_{max} + right_{max}}{2} + left_{max} \quad (1)$$

Using the lane center then subtracting it from the frame center which depends on the camera position to get the direction that will be sent to the Arduino.

$$Direction = frame\ center - lane\ center \quad (2)$$

If the Direction is less than zero, then the information to be sent to the Arduino that the lanes are moving right. Likewise, Arduino will receive a move left order if the Direction is larger than 0. Otherwise, the car will move forward. The Direction value will have a relation with the degree of the steering wheel rotation that will be determined in the Arduino code.

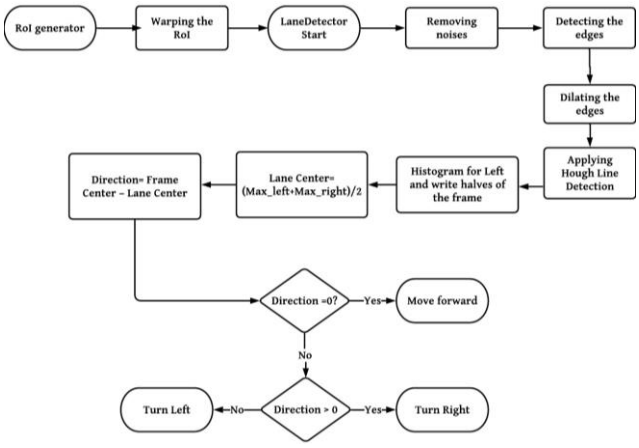

Fig. 6 lane detection flow chart

IV. OBJECT DETECTION:

So far, the algorithm can detect cars, stop signs, persons, cats, and dogs. To detect these objects, Haar feature-based cascade classifiers models have been trained on almost 300 positive images and 2000 negative images for each item.

The reason behind using Haar cascade classifier and not a new method like Yolo v4 is Yolo v4 requires a high-performance GPU/CPU and large RAM, which after some research, turned out that the Raspberry pi model is not capable of handling. [5]

Moreover, the OpenCV library provides a Cascade Classifier reader, which also be useful in implementing this algorithm. The algorithm starts by loading the XML trained model into the program.

For the program to detect each object, it will make a region of interest for each object to reduce the noises and false positives. For example, for it to detect cars, it will be concerned about a smaller region directly in front of the camara. Moreover, for it to detect traffic lights, it will be concerned about the top, right and left sides of the frame only. Same for the stop signs.

After detecting the objects in each region of interest, it will draw a box around each object. Using the following formula to determine the distance that will be sent to the Arduino to take the necessary action:

$$Object\ Distance = m*(diameter\ of\ the\ box) + c \quad (3)$$

Where *m* and *c* parameters are determined by finding the relation between the diameter of the box in pixels and the distance in meters.

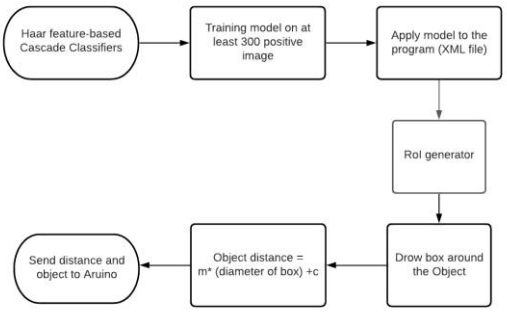

Fig. 7 Object Detection flow chart

V. RESULTS

A. Lane detection

Having described the details and workability of the algorithm, a video dataset obtained from [6] is used to apply this method.

The result images of Fig. 8 represent the overlap of the lane center and frame center. In this image, the direction result is 0. Which then will be sent to the controller to operate the car in the forward direction.

As can be seen in Fig. 7 below, the direction of the lanes is slightly moving toward the left direction. The blue line represents the frame center, while the green lane shows the lane's center. The result is obtained to be positive with a relatively small value, hence the controller will turn the car to the left according to this value.

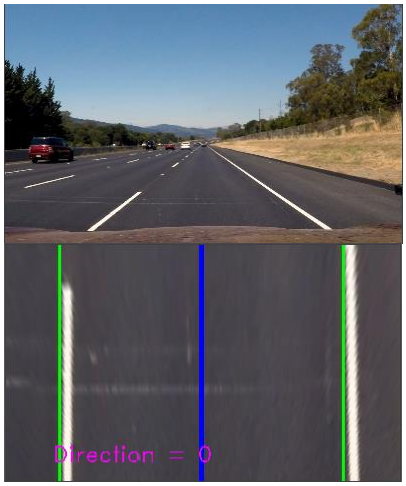

Fig. 6 Forward direction result

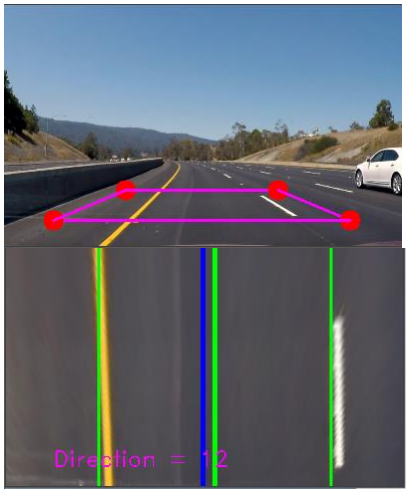

Fig. 7 Left direction result

B. Object detection

Using datasets from [7] and [8] to train the Haar feature-based cascade classifier models to detect cars, traffic lights, persons, and stop signs. These models were applied to the program to start detecting these objects.

The resulting image in Fig. 8 shows that these Haar features models performed well in detecting the car that is in the region of interest for the front car, and ignored the other cars in the frame. The model also detected the traffic light in the top part of the frame. The algorithm also shows the approximate distances of the car and the traffic light.

Fig. 9 shows the result of detecting stop signs Haar feature model successfully detected the stop sign and returned its distance.

Fig. 8 Car and traffic light detection

Fig. 9 Stop sign detection

VI. DIFFICULTIES:

A. Technical limitations:

When these algorithms were in the design phase it has been decided that a Raspberry Pi 3 B+ model will be used as a master module and will perform the image processing for the self-driving car as well as sending data to the controller to control the motors. However, the Raspberry Pi has only 1 Gigabyte of RAM and limited CPU and GPU capabilities, hence it will not be able to perform high accuracy object detection technologies such as Yolo, R-CNN. This was the reason to move to an older and simpler method such as Haar features based cascade classifiers which has acceptable accuracy for the purpose of building a simple self-driving RC car.

B. Haar False Positives:

Even though the Haar models are lighter on the Raspberry Pi, they have a drawback which is the many false positives. Fig. 10 shows that the Haar features detected the car mirror as another car that is 40 meters away from the camera, this information will distract the Arduino that controls the motors. This problem is solvable

by adding these false positives to the negative datasets while training the models.

Fig. 10 car detection false positive

C. Environmental problems:

As can be seen in Fig. 11, there are two different lighting conditions between the left and right lane which could distract the edge detection procedure since each lighting condition requires a different threshold approach. To solve this issue 2 measures have been taken, first, to add trackbars that will change the canny edge threshold based on the lighting condition. The second measure was to add the dilating stage in the algorithm to dilate the detected edges and ease the task on the Hough line detection.

Fig. 11 Different lighting condition

VII. CONCLUSION

This work addressed the problem of machine vision algorithms for self-driving RC car that can be then integrated to real cars. It first started by describing the importance of lanes in guiding the car, then it explained the steps for detecting lanes using a modified canny edge detector and Hough line transform. Afterward, identifying these lanes using the histogram method and sending the status of the lane to the controller. Moreover, the importance of object detection for the self-driving car was explained and how it could replace nowadays car sensors. The paper explained how Haar features models were used to detect cars, traffic lights, stop signs, persons, cats and dogs. Later, the formula for measuring the object distances that will be sent to the controller was introduced. The results were satisfying for the purpose of self-driving RC car, however could be improved by solving the difficulties which are technical limitation due to Raspberry Pi module, false positives due to Haar features and environmental lighting difficulty.

VIII. REFERENCES:

[1]: Facts + Statistics: Highway safety. (n.d.). Retrieved from https://www.iii.org/fact-statistic/facts-statistics-highway-safety

[2]: Hancock, P. (2018, February 03). Are Autonomous Cars Really Safer Than Human Drivers? Retrieved from https://www.scientificamerican.com/article/are-autonomous-cars-really-safer-than-human-drivers/

[3]: Niu, J., Lu, J., Xu, M., Lv, P., & Zhao, X. (2015, December 31). Robust Lane Detection using Two-stage Feature Extraction with Curve Fitting. Retrieved January 18, 2021, from https://www.sciencedirect.com/science/article/pii/S0031320315004690

[4]: Zhou, Y., Xu, R., Hu, X., & Ye, Q. (2006, February 21). A robust lane detection and tracking method based on ... Retrieved January, from https://iopscience.iop.org/article/10.1088/0957-0233/17/4/020

[5]: (n.d.). Retrieved January 12, 2021, from https://www.raspberrypi.org/forums/viewtopic.php?t=219601

[6]: Ross Kippenbrock. (2017, June 30). Rkipp1210/pydata-berlin-2017. Retrieved from rkipp1210/pydata-berlin-2017/blob/master/test_video/

[7]: Jensen, M. (2018, February 28). LISA Traffic Light Dataset. Retrieved from https://www.kaggle.com/mbornoe/lisa-traffic-light-dataset

[8]: Nypd. (2017, March 09). Vehicle Collisions in NYC, 2015-Present. Retrieved from https://www.kaggle.com/nypd/vehicle-collisions

[9]: Caltech Pedestrian Detection Benchmark. (n.d.). Retrieved January 19, 2021, from http://www.vision.caltech.edu/Image_Datasets/CaltechPedestrians/index.html

APPENDIX :

The code functions used for lane and object detection:

```
void ROI_generator()
{
    float height = img.size().height;
    float width = img.size().width;
    /*namedWindow("ROItrackbars", (640, 200));  Trackbar used for selecting the ROI
    createTrackbar("widthTop", "ROItrackbars", &widthTopint, 400);
    createTrackbar("hightTop", "ROItrackbars", &hightTopint, 240);
    createTrackbar("widthBottom", "ROItrackbars", &widthBottomint, 400);
    createTrackbar("hightBottom", "ROItrackbars", &hightBottomint, 240);*/
    float widthTop = widthTopint, hightTop = hightTopint, widthBottom = widthBottomint, hightBottom = hightBottomint;
    Point2f src[4] = { {widthTop,hightTop} ,{width-widthTop,hightTop}, {width-widthBottom,hightBottom},{widthBottom,hightBottom} };
    Point2f dst[4] = { {0.0f,0.0f},{w,0.0f},{w,h},{0,h} };
    matrix = getPerspectiveTransform(src, dst);

    // drowing the region of interest on the image
    warpPerspective(img, imgWarp, matrix, Point(w, h));
    for (int k = 0; k < 4; k++) {
        circle(img, src[k], 10, Scalar(0, 0, 255), FILLED);
        if ((k + 1) != 4) {
            line(img, src[k], src[k + 1], Scalar(255, 0, 255), 2);
        }
        if ((k + 1) == 4) line(img, src[0], src[3], Scalar(255, 0, 255), 2);
    }

}
```

Figure 12 RoI generator and Image warping Code

```
void LaneDetector()
{
    //int x = 79, y = 220;

    //****** Canny edge thresholds trackbars*******
    //namedWindow("CannyTrackbars", (640, 200));
    //createTrackbar("x", "CannyTrackbars", &x, 500);
    //createTrackbar("y", "CannyTrackbars", &y, 500);
    ////
    //
    //namedWindow("HoughTrackbars", (640, 200));
    //createTrackbar("m", "HoughTrackbars", &m, 500);
    //namedWindow("Bluring", (640, 200));
    //createTrackbar("blursize", "Bluring", &blursize, 15);

    GaussianBlur(imgGray, imgGray, Size(5, 5), 0);
    Canny(imgGray, imgCanny, x, y);
    dilate(imgCanny, imgDil, kernel);
    //HoughLinesP(imgCanny, lines, 1, CV_PI / 180, 50, 50, 10);
    HoughLines(imgDil, lines, 1, CV_PI / 180, m, 0, 0);
    imgc = imgDil.clone();
    /*for (size_t i = 0; i < lines.size(); i++) {
        Vec4i l = lines[i];
        line(imgc, Point(l[0], l[1]), Point(l[2], l[3]), Scalar(255, 0, 255), 4, LINE_AA);
    }*/
    for (size_t i = 0; i < lines.size(); i++)
    {
        float rho = lines[i][0], theta = lines[i][1];
        Point pt1, pt2;
        double a = cos(theta), b = sin(theta);
        double x0 = a * rho, y0 = b * rho;
        pt1.x = cvRound(x0 + 1000 * (-b));
        pt1.y = cvRound(y0 + 1000 * (a));
        pt2.x = cvRound(x0 - 1000 * (-b));
        pt2.y = cvRound(y0 - 1000 * (a));
        line(imgc, pt1, pt2, Scalar(250, 250, 250), 3, LINE_AA);
    }

}
```

Figure 13 Lane Detector function

```
void Histrogram() {
    histrogramLane.resize(400);
    histrogramLane.clear();
    Mat imgc1 = imgc.clone();
    for (int d = 0; d < 400; d++)       //frame.size().width = 400
    {
        ROILane = imgc1(Rect(d, 140, 1, 100));
        divide(255, ROILane, ROILane);
        histrogramLane.push_back((int)(sum(ROILane)[0]));
    }
}

void LaneFinder()
{
    vector<int>::iterator LeftPtr;
    LeftPtr = max_element(histrogramLane.begin(), histrogramLane.begin() + 150);
    LeftLanePos = distance(histrogramLane.begin(), LeftPtr);

    vector<int>::iterator RightPtr;
    RightPtr = max_element(histrogramLane.begin() + 250, histrogramLane.end());
    RightLanePos = distance(histrogramLane.begin(), RightPtr);

    line(imgWarp, Point2f(LeftLanePos, 0), Point2f(LeftLanePos, 240), Scalar(0, 255, 0), 2);
    line(imgWarp, Point2f(RightLanePos, 0), Point2f(RightLanePos, 240), Scalar(0, 255, 0), 2);
}

void LaneCenter()
{
    laneCenter = (RightLanePos - LeftLanePos) / 2 + LeftLanePos;

    /*namedWindow("frame center", (640, 200));
    createTrackbar("frame center position", "frame center", &frameCenter, 400);*/
    line(imgWarp, Point2f(laneCenter, 0), Point2f(laneCenter, 240), Scalar(0, 255, 0), 3);
    line(imgWarp, Point2f(frameCenter, 0), Point2f(frameCenter, 240), Scalar(255, 0, 0), 3);
    Result = laneCenter - frameCenter;
    ss.str(" ");
    ss.clear();
    ss << "Result = " << Result;
    putText(imgWarp, ss.str(), Point2f(47, 220), FONT_HERSHEY_DUPLEX, 0.75, Scalar(255, 0, 255), 1);
}
```

Figure 14 Histogram, Lane finder and lane center functions

```
void Stop_detection()
{
    if (!Stop_Cascade.load("Resources/stopsign_classifier.xml"))
    {
        printf("Unable to open stop cascade file");
    }

    RoI_Stop = frame_Stop(Rect(0, 0, 400, 240));
    cvtColor(RoI_Stop, gray_Stop, COLOR_RGB2GRAY);
    equalizeHist(gray_Stop, gray_Stop);
    Stop_Cascade.detectMultiScale(gray_Stop, Stop);

    for (int i = 0; i < Stop.size(); i++)
    {
        Point P1(Stop[i].x, Stop[i].y);
        Point P2(Stop[i].x + Stop[i].width, Stop[i].y + Stop[i].height);

        rectangle(RoI_Stop, P1, P2, Scalar(0, 0, 255), 2);
        putText(RoI_Stop, "Stop Sign", P1, FONT_HERSHEY_PLAIN, 1, Scalar(0, 0, 255, 255), 1);

        Stop_distance = (-1.07) * (P2.x - P1.x) + 72.4;
        ss.str(" ");
        ss.clear();
        ss << "StopDistance = " << Stop_distance <<"meters";
        putText(RoI_Stop, ss.str(), Point2f(230, 130), 0,0.4, Scalar(255, 0, 255), 1);

    }

}
```

Figure 15 Stop sign object detection code

Similar codes will be obtained for detecting other objects but using different names other than Stop.