An Algorithm Using Dynamic Geometric Constraints for Detecting and Marking Roads for Autonomous Golf Cart

Luodai Yang, Qin Hu, Lijuan Zhang, Jonathon Lin
College of Technology
Eastern Michigan University
Ypsilanti, MI 48197
Email: lyang15@emich.edu, qhu1@emich.edu, lzhang20@emich.edu, jonathon.lin@emich.edu

Abstract

Lane detection and marking is the first step for Driver Assistance System (DAS) and Autonomous Vehicle technology. Over the years, people have done a great deal of research in this area and have developed different methods and technologies to solve real world problems. However, most methods are based on highway or local road environment. Usually, using straight lines or fitted curves to model roads has been proven to be effective. However, these methods are not suitable for golf courses with multiple-curves. In this paper, we propose a new lane detection and marking algorithm with a series of simplified filters to help identifying multiple-curved roads in golf courses. This algorithm can detect the whole road surface and get sufficient number of edge points. After processed by the proposed filter for dynamic geometric constraints, the left and right edges of a lane can be precisely marked for an autonomous golf cart.

Introduction

Driver Assistance System (DAS) and Autonomous Vehicle technology are a combination of many advanced technologies that helps people to reduce the risk of driving and improve the experience [1]. Usually, this system is embedded and integrated into the vehicle with various sensors such as camera, LiDAR and radar to collect information from surrounding environment. The collected data are then used to understand and predict their surroundings in real time [2]. Based on such set of information, a road model with surrounding information can be built. The vehicle, then, has the ability to take the initiative to make an action by pre-programed strategies developed by computer vision and AI algorithms. Lane detection and marking is the first step for the driver assistance and autonomous vehicle system. It plays an essential role in this autonomous system and technology [1]. The results from lane detection and marking can be used as the input data for subsequent processes like building a road model for analysis or creating a visible image to assist driving. In this paper, we propose a new lane detection and marking algorithm with a series of simplified filters for golf course environment.

A great deal of related researches using different methods for modeling, detection,

tracking and departure of lanes [1, 3, 4, 5] has been conducted to solve real world problems. But most of the researches are based on highway environments with lane marks. No effort has been devoted to the lane detection and marking in golf courses. Most of the time, the terrain of the highway is gentle and the close area is almost straight which means that the road is always located in front of vehicle [1]. Therefore, people can use region of interest (ROI) to choose a special area as the target [6, 7]. But for a golf course, it has a dramatic terrain which means that it may include multiple curves, and even not the whole road in one single image.

In this paper, we present a new lane detection and marking algorithm based on HSV (hue, saturation, value) color space and dynamic geometric constraints which can identify lanes in golf courses with multiple curves.

1. Methodology

Our algorithm includes two main parts: road detection and extraction, and lane marking as shown in Figure 1. The algorithm is based on an HSV color model and Canny edge detection [8, 9]. First, we use HSV color model to identify all pixels of the grass and road of a golf course. After processing by a simple filter using Gaussian Blur and logic addition [10], a Canny edge detector is utilized to extract all possible edge points of the road [9]. A dynamic geometric constraint filter is then used to determine which side the edge points belong to, the left lane or right lane. Finally, we use different colors to mark the right lane (blue) and left lane (red), and return them as a visible result.

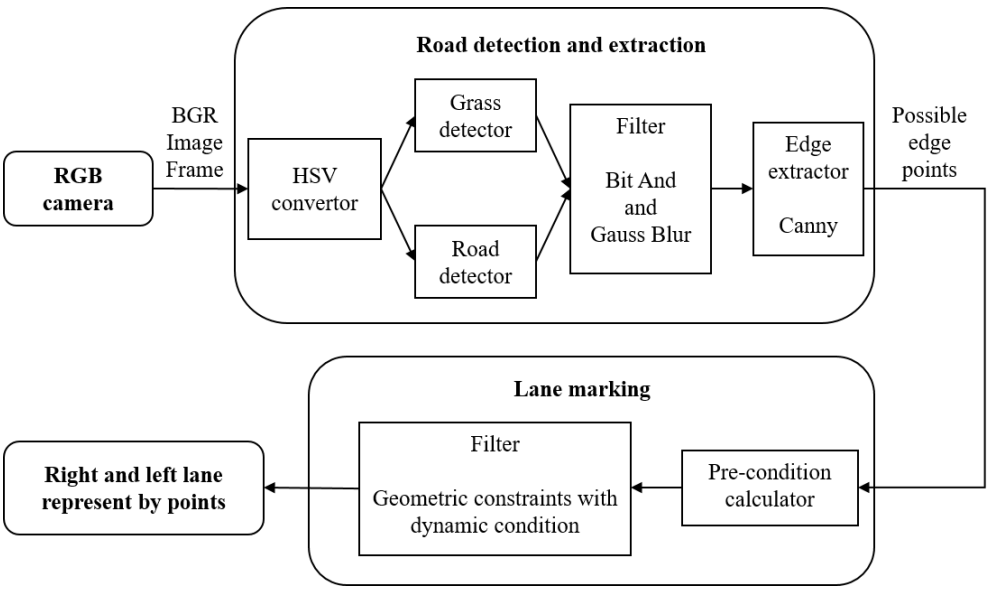


Figure 1: Flowchart of processing algorithms

For road detection and extraction, we use the program based on Python programming language and OpenCV (Open Source Computer Vision Library) to implement all processes for detection and feature extraction [11], then use a MATLAB program to do the points iterating and lane marking.

1.1. Road detection and extraction

1.1.1. HSV convertor

The images collected from a forward-looking camera mounted on the front window of the golf cart are stored in RGB (red, green, blue) format. The field of view is resized to a resolution of 640 x 480 in order to reduce the time and resource of computing. We use the HSV color model for color identification, HSV is a color model which is used to describe colors in computer graphics [12]. It can be transformed from popular RGB color space by Equation (1) [8]. HSV is better than RGB color model for color distinction [13, 14].

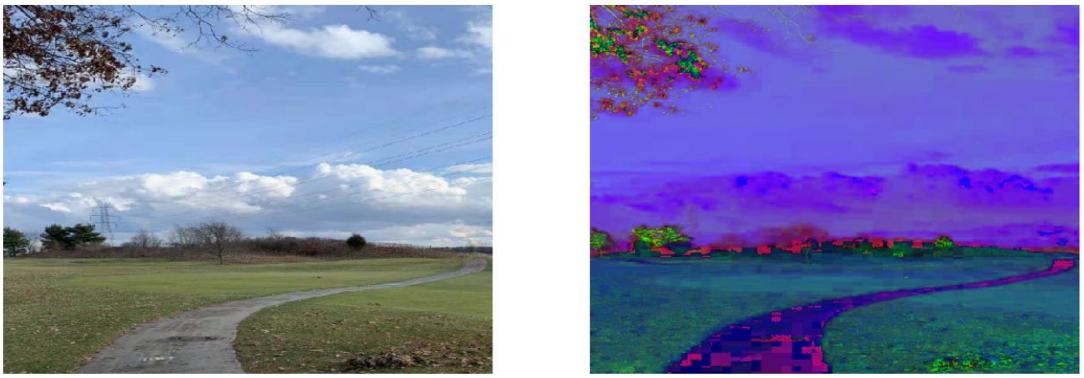
$$H = \begin{cases} 0°, & \Delta = 0 \\ 60° * \left(\dfrac{G' - B'}{\Delta} \ mod\ 6\right), & C_{max} = R' \\ 60° * \left(\dfrac{B' - R'}{\Delta} + 2\right), & C_{max} = G' \\ 60° * \left(\dfrac{R' - G'}{\Delta} + 4\right), & C_{max} = B' \end{cases} \quad (1a)$$

$$S = \begin{cases} 0, & C_{max} = 0 \\ \dfrac{\Delta}{C_{max}}, & C_{max} \neq 0 \end{cases} \quad (1b)$$

$$V = C_{max} \quad (1c)$$

Here *R'*, *G'*, and *B'* are *R/255*, *G/255* and *B/255*, respectively. $C_{max} = max(R',G',B')$, $C_{min} = min(R',G',B')$, and $\Delta = C_{max} - C_{min}$ [8]. The *mod* means modulo operation.

The RGB image after the conversion becomes a HSV image of 640 x 480 x 3 matrix form Figure 2. Here 640 and 480 are the width and height by pixels of the image, and 3 is the different color channels of the image.



(a) (b)

Figure 2: Two visible images for (a) RGB and (b) HSV models

1.1.2. *Classify road pixels and grass pixels*

We choose different HSV value ranges to determine each pixel being either a grass pixel or a road pixels. For example, the HSV values for those pixels for grass will fall in the green value range. We need to analyze and determine the range of HSV values for both road and grass pixels. Figure 3 shows that a grass sample is taken from the original image, then use histograms to illustrate the distribution for three channels of hue, saturation and value as shown in Figure 4.

(a) (b)

Figure 3: (a) The raw image; (b) A grass patch sampled from (a)

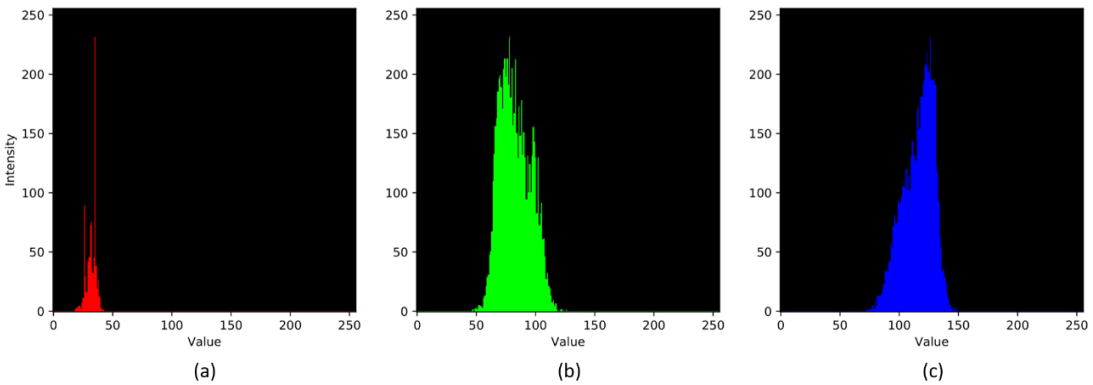
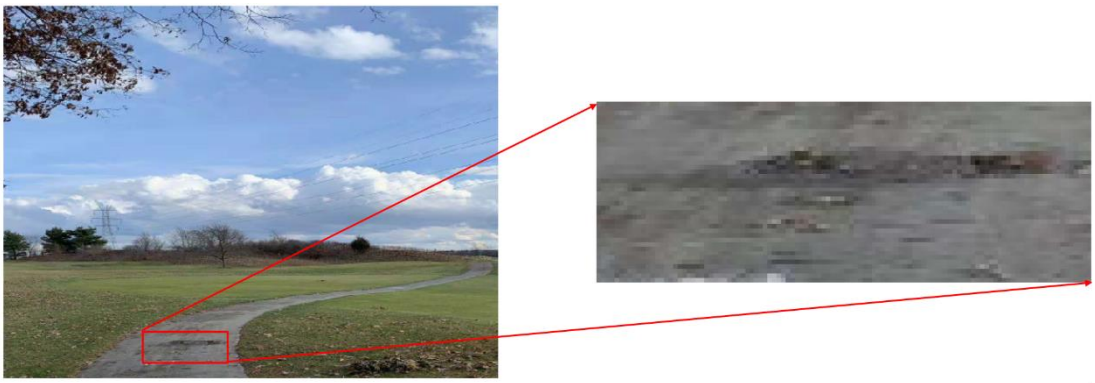


Figure 4: Histograms for (a) H, (b) S and (c) V channels in grass

Similarly Figure 5 shows that a road sample is taken from the original image, then use histograms to illustrate the distribution for three channels of hue, saturation and value as shown in Figure 6.



(a) (b)

Figure 5: (a) The raw image; (b) A road patch sampled from (a).

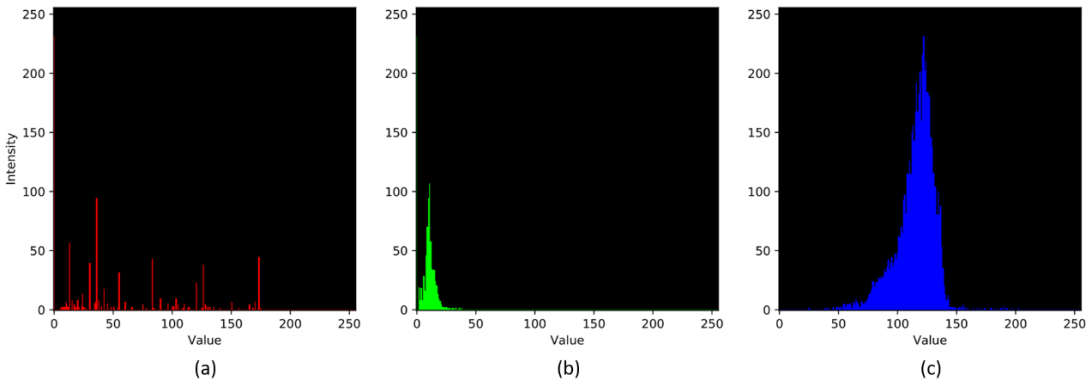
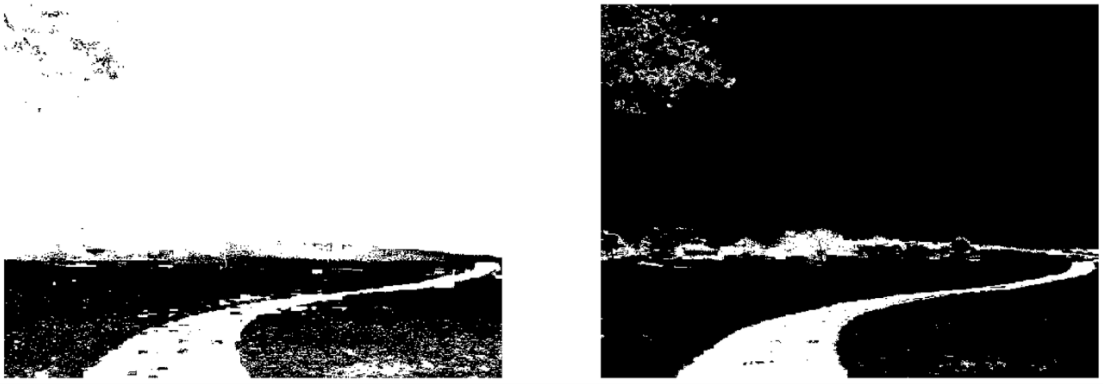


Figure 6: Histograms for (a) H, (b) S and (c) V channels of the road sample

In Figure 4 and Figure 6, the horizontal axis represent different values for H, S, and V from 0 to 255, and the vertical axis represent the number of pixels with a particular H, S or V value.

Based on the above results, we choose the HSV range (0, 20, 70) to (40, 125, 160) for grass, and (0, 0, 50) to (180, 40, 150) for road. These ranges of values are used as a simple filter to determine which pixels are for possible road and grass, respectively. The results are two binary maps as shown in Figure 7. Figure 7(a) is the binary map for the grass identified in Figure 3. Each possible grass pixel is black with a value of "0" in this map. Figure 7(b) is the binary map for the road identified in Figure 5. Each possible road pixel is white with a value of "1".



(a) (b)

Figure 7: Two binary maps for (a) the grass and (b) the road from Figure 5(a)

1.1.3. Bit addition and Gaussian blur filter

Since noises and rough edges are often found in the processed images, another filter is used to reduce noises and smooth the edges. This filter includes two parts, logic addition and Gaussian blur. Logic addition is used to reduce the noises and mix two images into one for a better result as shown in Figure 8.

Figure: 8 Binary map after bit addition

Gaussian blur can offer a 2-D convolution operation that blurs the image in order to obtain smoother edges for Canny edge detector and thus reduce the noise [15, 9]. The kernel of Gaussian blur uses standard deviations as the weights which can keep more details after blurring [15, 10]:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (2)$$

In equation (2), G(x, y) calculates the value from 2D normal distribution based on *x* and *y* of different positions. Here *x* is the distance between the original point and the

horizontal axis, *y* is the distance between the original point and the vertical axis, and σ is the standard deviation. The original point is the central pixel in the image matrix. The values of *G(x, y)* are used to build a convolution matrix, they then are applied to the original images.

In our program, a 13x13 convolution matrix is created as the kernel to convolute image matrixes with the same size from Figure 8. The standard deviation *σ(x)* and *σ(y)* for *x* and *y* directions are computed from width and height by following formula [16]:

$$\sigma(a) = 0.3\left\{\frac{(2a-1)}{2} - 1\right\} + 0.8 \tag{3}$$

Here, width and height are 13 x 13. Therefore, x and y have the same value 6.5 based on *2x=13* and *2y=13*. The standard deviation σ is 2.3 for both *σ(x)* and *σ(y)* in Figure 9.

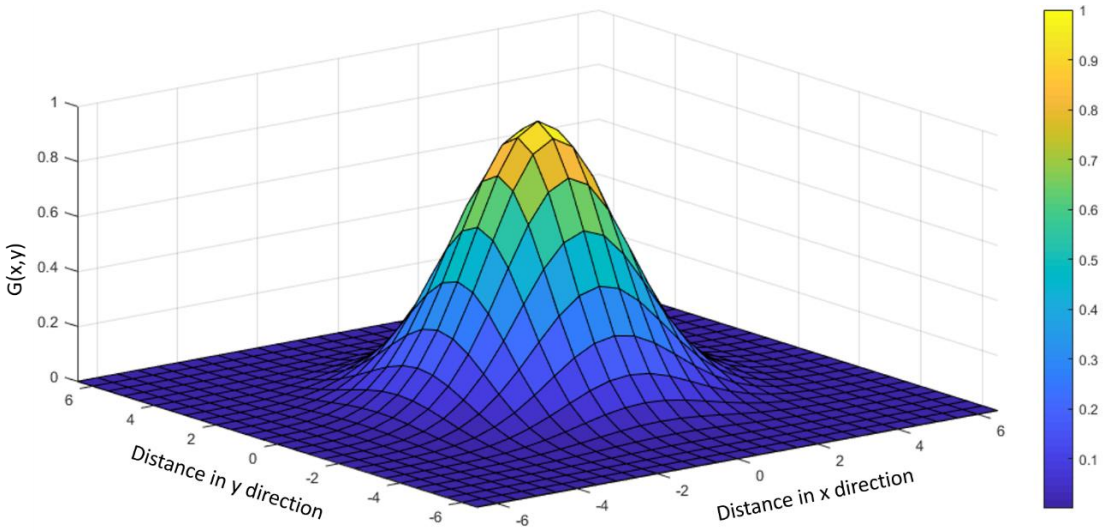


Figure 9: 2D Normal distribution.

Therefore, *G(x, y)* returns different normal distribution values as weights from Figure 9 for each pixels based on x and y distance, then does multiplication with 13 x 13 image matrixes from Figure 8 and add them together to get the a single value. This value is used to replace this matrix as one single pixel: $S = \sum G(x,y) * z(x,y)$, where *z(x, y)* is the value from each pixel, *G(x, y)* is the weight for each pixel. Finally, the whole 13 x 13 matrix will be replaced by *S*. Figure 10 shows the blurred result which misses some unnecessary details in the image.

Figure 10: Image after Gaussian blur filter.

1.1.4. Extract edge features

After Gaussian blur, the Canny edge detector is used to find all possible edge points of the image because Canny edge detector can be used to find the image gradient [1, 7]. As shown in Figure 10, the edges are located in the borderline between the white and black colors, which means between the value '0' and '1'. The Canny edge detector helps to find the locations, with the maximum gradient from color value changes, which are the edges.

To find all gradients in the image, the Sobel operator uses two 3 x 3 convolution matrix as the filters to find the approximate absolute gradient magnitude for each pixel. One estimates the gradient in the vertical direction G_x and the other for the horizontal direction G_y [15, 7, 17].

$$\boldsymbol{G_x} = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \boldsymbol{A} \tag{4a}$$

$$\boldsymbol{G_y} = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \boldsymbol{A} \tag{4b}$$

Here G_x and G_y are Sobel operators, A is the original image matrix. This process returns the first derivatives in both horizontal direction G_x and vertical direction G_y. The gradient and direction of edges can be determined as [15, 7]:

$$G = \sqrt{{G_x}^2 + {G_y}^2} \tag{5a}$$

$$angle° = tan^{-1}(G_y/G_x) \quad (5b)$$

After this process, the non-maximum suppression is used to help us to locate the local maximum and suppress other gradient values by setting them to 0. Local maximum indicates that the location has the sharpest change, the maximum gradient. Each pixel is processed by first comparing the gradient of the pixel with the gradient of the pixel in both positive and negative gradient directions. If the gradient of the pixel is the largest after comparing to other pixels in the mask matrix in the same direction, the value will be preserved. Otherwise, the value will be suppressed.

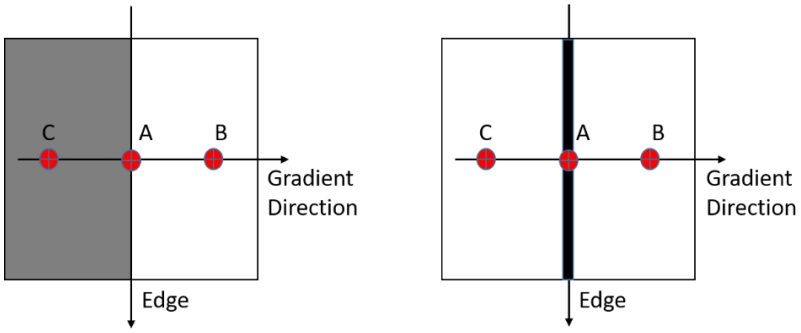


Figure 11: Point A has the maximum gradient value.

As shown in Figure 11, points B and C have different color value '1' and '0' that are represented by white and black color. Here point A have the largest gradient, thus we preserve point A and mark it as possible edge point.

Two thresholds, high and low values, then are used as a filter to ignore pixels with a smaller gradient value and preserve edge pixels with a higher gradient value. Therefore, we have three scenarios: The gradient value of an edge pixel is (1) higher than the high threshold value; (2) lower than the low threshold value; (3) between two threshold values. For (1), the pixel will be marked as a strong edge pixel and keep this one. For (2), the pixel will be marked as a weak edge pixel and ignore this one. For (3), if the pixel is close to a strong edge pixel, keep it. Otherwise, ignore it.

As shown in Figure 12, the orange edge in the binary map has part A and B. The gradient value of part A is higher than the maximum threshold value, therefore part A will be considered as edge. The gradient value of part B is in between maximum and minimum value, but the part B is close to part A which is already be considered as edge, thus, part B is also considered as edge. Part C and part D are in between the maximum and minimum thresholds. They are not close to any exist edge points, so these two parts are considered as noises. The gradient value of part E is lower than the minimum value, so it is considered as a noise, too.

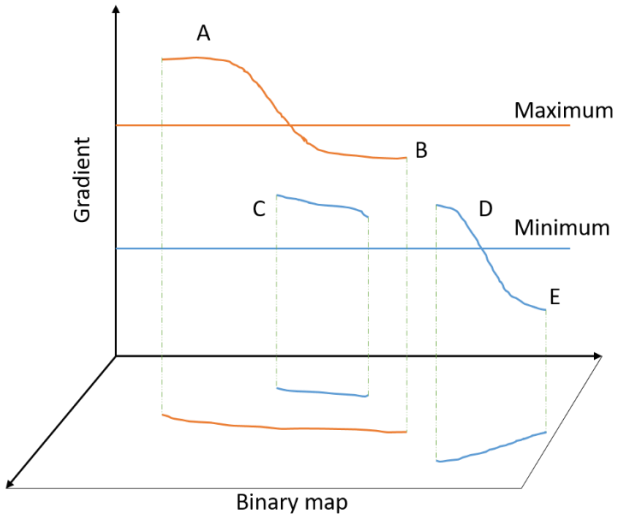
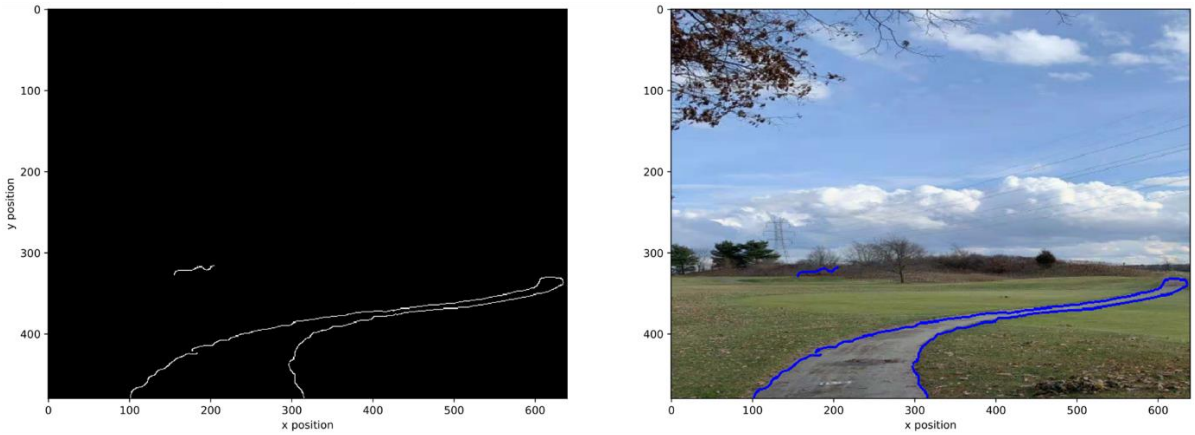


Figure 12: Three situations for all gradients

In this algorithm, the Canny function from OpenCV library is used to implement this process with the low threshold = 150 and the high threshold = 450 [15, 9]. Figure 13a shows the result of edge point binary map of the raw image with blue points in Figure 13b after using the Canny method.



(a) (b)

Figure 13: (a) The edge points identified from the binary map; (b) The edges detected on the original image.

The next step is to reduce the density of points because the original result has too many points that could cause sharp changes and noises. By reducing the density of points, the change in slopes can be smooth and the complication of computing is significantly reduced.

The final step is to convert the result from Canny into a collection of points which is presented by [x, y] positions. The original format for the position of points in the binary map only has the value of '1' and '0'. The value '1' means a possible edge point, conversely '0' means no edge point. Therefore, we use the column number and

raw number as [x, y] space position to indicate the location for each edge point. Then arranged them by [y] value, and saved all position as .csv format file which can be imported by MATLAB program.

1.2. Lane marking

1.2.1. Identify initial conditions

Lane marking uses the slope and distance between two edge points as geometric constrains to determine whether a point belonging to the right or the left edge of the lane. Two sets of three initial conditions (initial slope, distance, and start points) are needed to help identifying whether a point is in the right edge or in the left edge of the lane. Since the denominator $x_2 - x_1$ in the slope equation, $m = (y_2 - y_1)/(x_2 - x_1)$, can't be 0, and the numerator $y_2 - y_1$ is expected not to be 0 for reducing noises, therefore, adjacent points with the same *x* or *y* value are deleted in the point set. In this paper, the first 30 points from .csv file are used to calculate the three initial conditions for the sample set. To calculate initial conditions we need sample points for both left edge and right edge of the lane. A simple rule, *weight/2,* is used to determine if a point being in the left edge or in the right edge. Those points having x value greater than *weight/2* are right edge sample points, and points having x value less than *weight/2* are left edge sample points.

For the initial slopes, the algorithm calculates the slopes in right edge points and in left edge points. Then choose the median slope value, *Slope = median(*$m = \frac{y_2 - y_1}{x_2 - y_1}$*)*, to find the two initial slopes for right edge and left edge of the lane.

For the initial distances, it is similar to the calculation of the initial slopes. The algorithm use the same first 30 points to calculate their distances for both right edge and left edge of the lane. Then choose the maximum distance, *max(*$distance = \sqrt{(y_2 - y_1)^2 * (x_2 - x_1)^2}$*)*, as the initial distance. The results are two initial distances for right edge and left edge of the lane.

For the initial points, we simply choose the last point in right edge and the last point in left edge of the sample point set as the first point for slope and distance calculation.

1.2.2. Test target points by geometric constraints with dynamic initial conditions

Figure 14 is the algorithm flowchart for implementing geometric constraints. In this process, [y] values of points in .cvs file are used to identify the order of points from lowest to highest values. The given initial slope is used as the base value for slope change range, and initial distance as the base value for distance change range. In this algorithm, we simply choose ±5 °(slope) and ±5 (distance) as the change ranges. If the

slope and distance between target point and initial point satisfy these two conditions for left lane or right lane, storage the current target point in the corresponding point set and become the new initial point for the next target point. If the current target point can't satisfy these two conditions for right or left edge, then just ignore it, and change the initial conditions for left or right edge to have a large tolerance for next target point. For example, change $\pm 5°$ to $\pm 10°$ in the left edge conditions if this run did not find the left edge point, but if at the same time, the target point satisfied the right edge conditions, save it as the right edge and do not change the condition for the right edge. Once the program finds a point belonging to left or right lane, then resets the corresponding initial conditions back to the original value for the next target point.

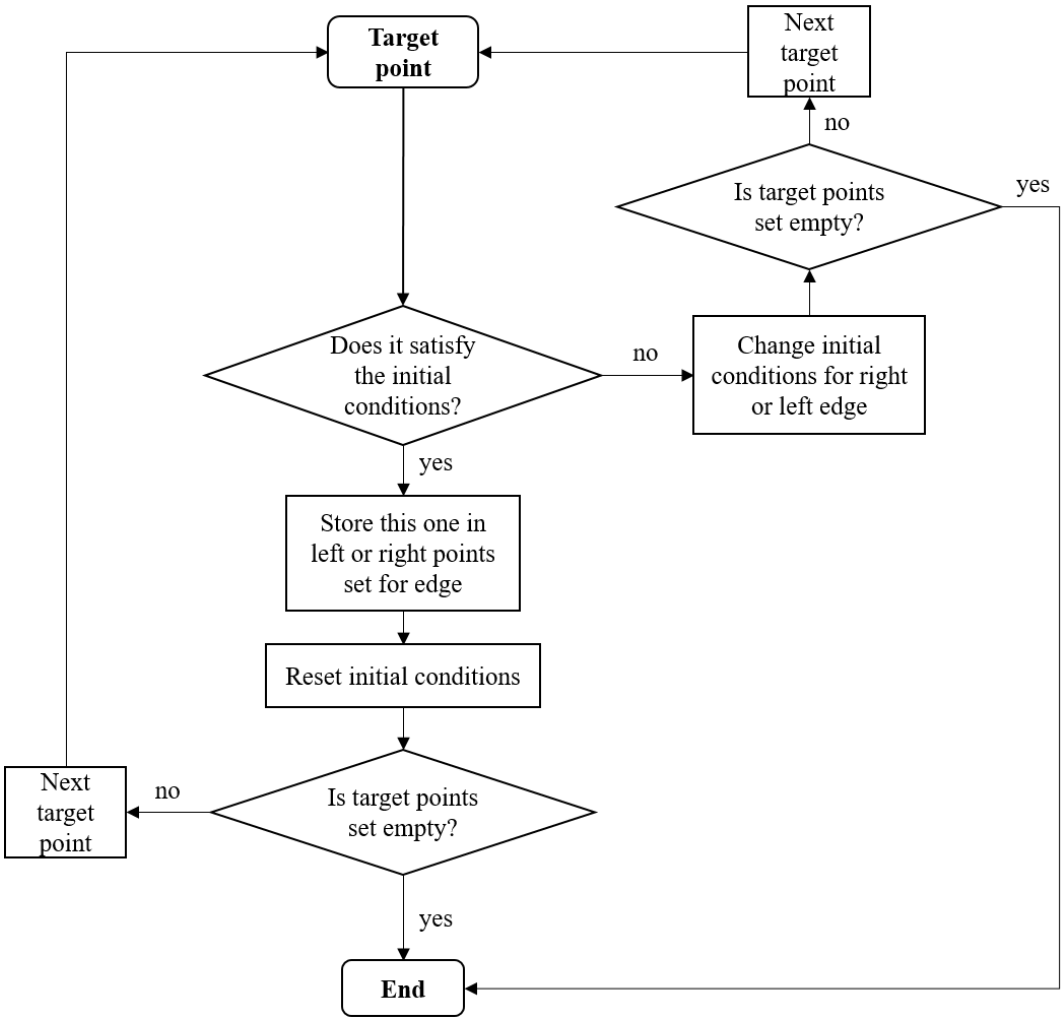


Figure 14: Flowchart for geometric constraints with dynamic initial conditions process

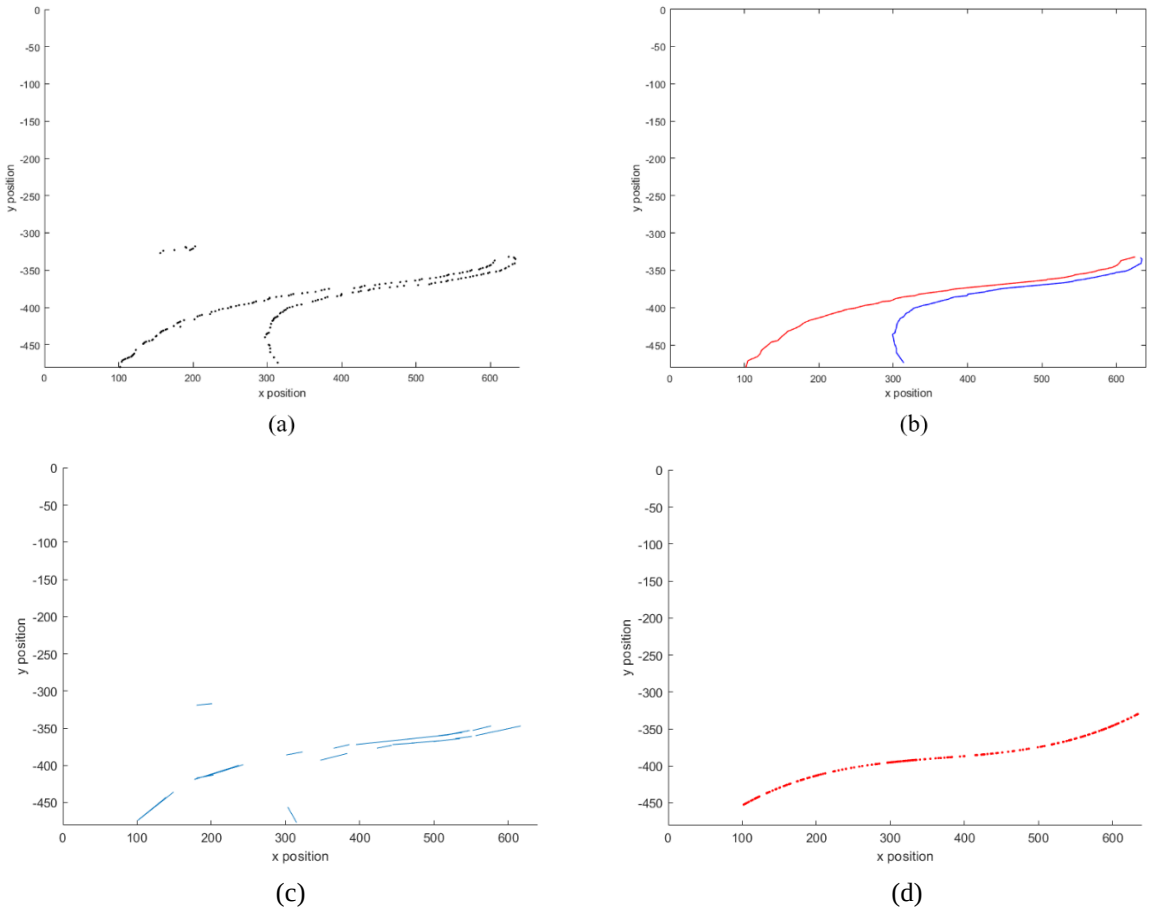


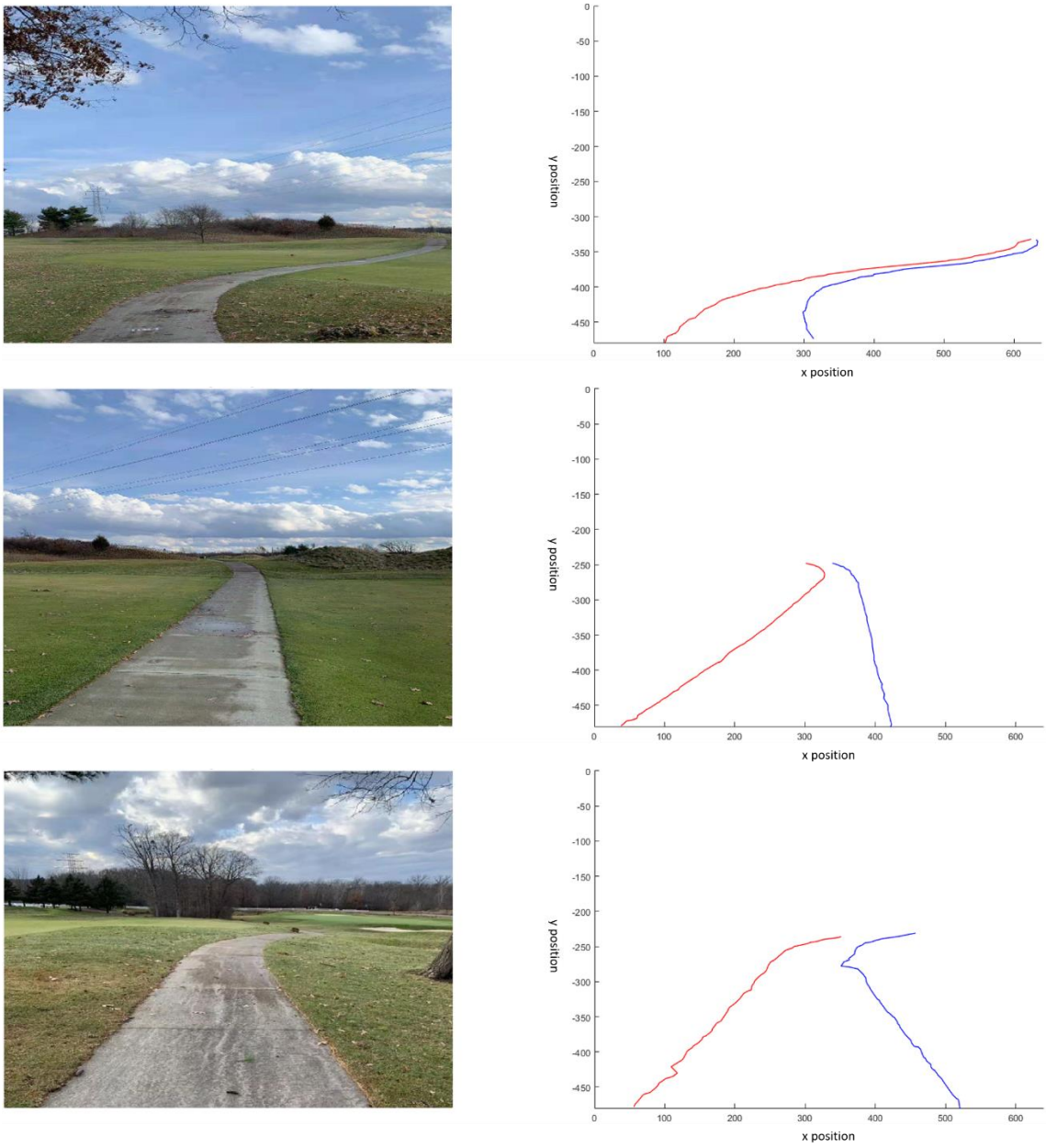
Figure 15: (a) All possible edge points after Canny operation; (b) Geometric Constraints - Left edge of the lane is in red and the right edge of the lane is in blue after the lane marking algorithm; (c) Lane detection result by Hough Transform method; (d) Lane detection result by Polynomial Curve Fitting method.

We also compare our results with the results from two popular lane detection algorithms, Hough Transform and Polynomial Curve Fitting. Figure 15 shows the results obtained from the three algorithms. All possible edge points after Canny operation (importing from .csv file) is shown in Figure 15 (a). Figure 15(b) shows the result using geometric constraints algorithm process. The left edge of lane is marked as red line and the right edge of lane is marked as blue line. Figure 15(c) shows the result by using Hough Transform method [18, 19]. The outline of lane has been marked, but it did not filter out the noises and the edge is not continued. Figure 15(d) shows the result by Polynomial Curve Fitting. It only marked the general trend and it can't identify the left and right edge of the lane [20, 21]. It can be clearly seen that our geometric constraints algorithm performs better than the other two methods.

2. Results and Discussions

Figure 16 shows four sets of images with various road profiles and their edge markings generated from our lane marking algorithm. The results show that the algorithm can

successfully perform these five tasks: (1) detect and mark left and right road edges in golf course environment using geometric constraints, (2) provide the tolerance for some noises such as trees, clouds, etc., (3) filter out unrelated elements such as rainwater and leaves, (4) add differentiated colors (red for left edge and blue for right edge) on the road, and (5) detect and mark multiple curves. This algorithm did not use ROI (region of interest) to limit the region of view because it does not need to keep the whole road in front of it all the time.



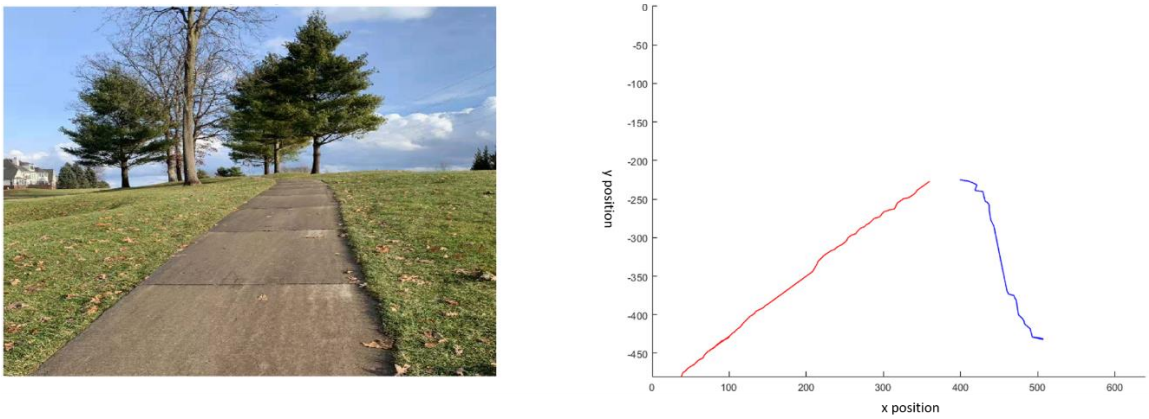
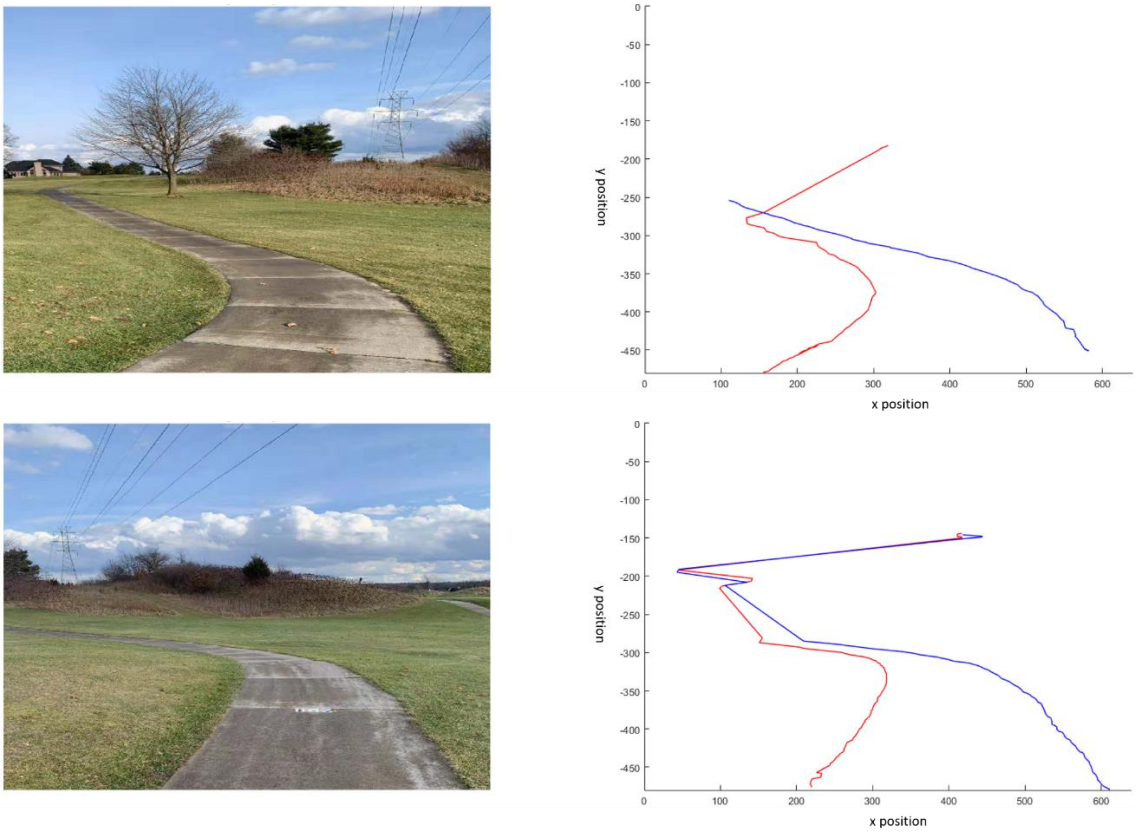


Figure 16: Results have good visible lane mark.

Figure 17 shows three sets of images with more sophisticated road profiles and their edge markings. The edge marking results indicate that the algorithm has successfully detected and marked the edges for the first half of the roads, but not for the second half of the roads. The problem with the second half of the roads may come from the images having the targeted segments of the roads extended too long so that the distance between two edges of the road becoming too small to be detected and marked by the algorithm. This problem can be improved by using aerial view of images to reduce such distortions in the images or just use the first half of edge markings as images continuously updated while the golf cart is moving forward.



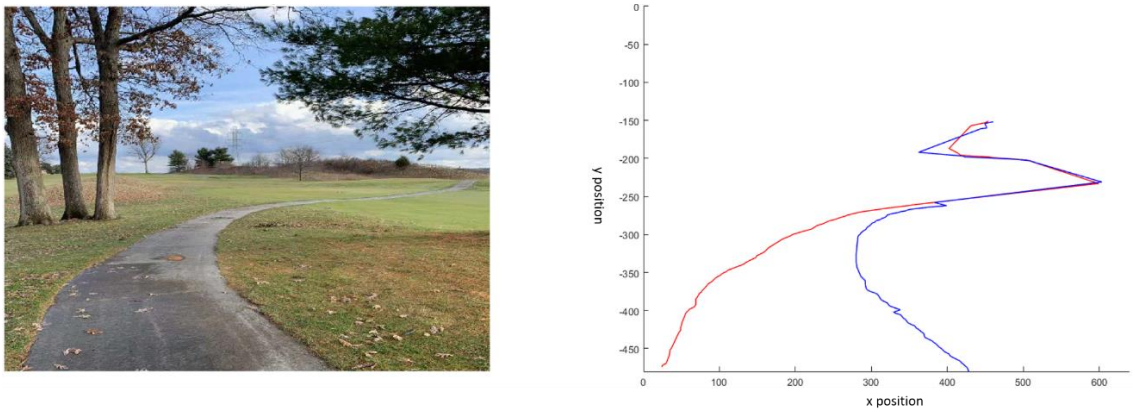


Figure 17: Results need to be improved.

3. Conclusion

In this paper, we present a new algorithm using dynamic geometric constraints with HSV model and Canny method for detecting and marking roads for golf course environment. Based on the feature of dynamic geometric constraints, the algorithm has successfully solved the problems encountered in using traditional edge detection method for golf courses. The results show that this algorithm has a good tolerance for noises and an expected robustness. The results can be used in autonomous golf car to help develop its driver assistance system or autonomous technology for improving the safety and driving experience. Further studies would be on improving road modeling, by using aerial images and neural network to identify various signs and obstacles to enhance the system.

Reference

[1] S. P. Narote, P. N. Bhujbal, A. S. Narote, et al, "A Review of Recent Advances in Lane Detection and Departure Warning System," *Pattern Recognition*, vol. 73, Jan. 2018.

[2] H. Zhu, K. V. Yuen, L. Mihaylova, H. Leung, "Overview of Environment Perception for Intelligent Vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 10, Oct. 2017.

[3] U. Franke, D. Gavrila, S. Görzig, et al, "Autonomous Driving Goes Downtown," *IEEE Intelligent systems*, vol. 13, no. 6, Dec. 1998.

[4] C. Mu, X. Ma, "Lane Detection Based on Object Segmentation and Piecewise Fitting," *TELKOMNIKA Indonesian Journal of Electrical Engineering*, vol. 12, no. 5, May. 2014.

[5] P. Y. Hsiao, C. W. Yeh, S. S. Huang, L. C. Fu, "A Portable Vision-Based Real-Time Lane Departure Warning System: Day and Night," *IEEE Transactions on Vehicular Technology*, vol. 58, no. 4, May. 2009.

[6] V. Gaikwad, S. Lokhande, "Lane Departure Identification for Advanced Driver Assistance," *IEEE*

Transactions on Intelligent Transportation Systems, vol. 16, no. 2, Apr. 2015.
[7] P. C. Wu, C. Y. Chang, C. H. Lin, "Lane-Mark Extraction for Automobiles under Complex Conditions," *Pattern Recognition*, vol. 47, no. 8, Aug. 2014.
[8] A. R. Smith, "Color Gamut Transform Pairs," *ACM Siggraph Computer Graphics*, vol. 12, no. 3, Aug.1978.
[9] J. Canny, "A Computational Approach to Edge Detection," *IEEE Transactions on pattern analysis and machine intelligence*, vol. PAMI-8, no. 6, Nov. 1986.
[10] Fisher, Perkins, Walker & Wolfart, "Spatial Filters - Laplacian of Gaussian," https://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm, (accessed March 2019).
[11] G. Bradski, A. Kaehler, *Learning OpenCV: Computer Vision with the OpenCV library*, O'Reilly Media, Inc., Sebastopol, 2008.
[12] G. H. Joblove, D. Greenberg, "Color Spaces for Computer Graphics," *ACM SIGGRAPH Computer Graphics*, vol. 12, no. 3, 1978.
[13] A Mammeri, A Boukerche, Z Tang, "A Real-Time Lane Marking Localization, Tracking and Communication System," *Computer Communications*, vol. 73, Jan. 2016.
[14] M. W. Schwarz, W. B. Cowan, J. C. Beatty, "An Experimental Comparison of RGB, YIQ, LAB, HSV, and Opponent Color Models," *ACM Transactions on Graphics (TOG)*, vol. 6, no. 2, Apr. 1987.
[15] B. Green, "Canny Edge Detection Tutorial," http://masters.donntu.org/2010/fknt/chudovskaja/library/article5.htm, (accessed March 2019).
[16] "Open Source Computer Vision Library," https://docs.opencv.org/master/d4/d86/group__imgproc__filter.html#gac05a120c1ae92a6060dd0db190a61afa, (accessed March 2019).
[17] I. Sobel, G. Feldman, "A 3x3 isotropic gradient operator for image processing," *Pattern Classification and Scene Analysis*, 1973, pp. 271-272.
[18] R. O. Duda, P. E. Hart, "Use of the Hough Transformation to Detect Lines and Curves in Pictures," *Communications of the ACM*, vol. 15, no. 1, Jan. 1972.
[19] L A F Fernandes, M. M. Oliveira, "Real-Time Line Detection Through an Improved Hough Transform Voting Scheme," *Pattern recognition*, vol. 41, no. 1, Jan. 2008.
[20] L. A. Sandra, *PHB Practical Handbook of Curve Fitting*, CRC Press, Boca Raton, 1994.
[21] P. G. Guest, *Numerical Methods of Curve Fitting*, Cambridge University Press, Cambridge, 2012.