Investigation of edge detection to navigating an autonomous vehicle

juncheng

May 2024

Abstract

In autonomous vehicle navigation, precise edge detection plays a pivotal role in interpreting the vehicle's surroundings and enabling safe path planning. This paper investigates the application of edge detection techniques, including the Canny algorithm and the Hough Line Transform, for recognizing key road features such as lanes and obstacles. Through OpenCV-based interactive parameter tuning with trackbars, optimal detection thresholds are identified to adapt edge detection to varying road conditions. The study emphasizes the importance of fine-tuned edge detection in real-time navigation and obstacle avoidance.

1 Introduction

Autonomous vehicles rely on computer vision to analyze their environment and navigate safely. A core challenge lies in accurately detecting road features such as lanes, edges, and obstacles amidst varying lighting conditions and diverse road structures. Edge detection is crucial in differentiating relevant road features from irrelevant noise.

The Canny edge detection algorithm remains one of the most effective methods for detecting edges due to its noise reduction, gradient calculations, and edge tracking via hysteresis. Once edges are detected, the Hough Line Transform helps identify linear features like lane markings.

This paper explores the use of these edge detection techniques to enhance autonomous vehicle navigation. Using Python and OpenCV, we implemented an interactive approach with trackbars to adjust detection thresholds dynamically. The code provided allows users to refine the Canny algorithm's lower and upper threshold values in real-time. Key components of this implementation include:

- **Grayscale Conversion**: Converts the original image to a single-channel intensity image for simpler processing.
- **Gaussian Blurring**: Reduces noise to improve edge detection accuracy.
- **Canny Edge Detection**: Identifies significant edges based on the tuned threshold values.

- **Hough Line Transform**: Detects road lines using the probabilistic Hough transform method.
- **Interactive Parameter Adjustment**: Employs trackbars to adjust Canny thresholds and improve detection.

The resulting image is visually inspected, and the best thresholds are saved for further testing on real-world data. By combining these techniques, we illustrate the critical importance of optimized edge detection in safe autonomous vehicle navigation.

2 Line Detection Methods

2.1 Grayscale Conversion

Converting the input image to grayscale is essential to reduce computational complexity while focusing on intensity values. This step is crucial because many edge detection algorithms work best on single-channel images. OpenCV's `cvtColor` function with the `COLOR_BGR2GRAY` flag simplifies the image to a single intensity channel. Grayscale conversion eliminates color distractions, making edge detection consistent.

2.2 Gaussian Blurring

Noise can interfere with accurate edge detection, leading to false positives and a cluttered edge map. Applying Gaussian blurring with a 5x5 kernel reduces noise while preserving crucial features. The blurring maintains smooth gradients where lines exist.

2.3 Canny Edge Detection

The Canny edge detection algorithm involves:

- **Gradient Calculation**: Canny computes the gradients in the blurred grayscale image to identify significant intensity changes.
- **Edge Tracking via Hysteresis**: With two adjustable threshold values (lower and upper), Canny separates strong edges from noise and tracks weak edges connected to strong edges. Interactive trackbars allow fine-tuning of these thresholds to correctly identify lines.

2.4 Hough Line Transform

After edge detection using Canny, the probabilistic Hough Line Transform (`cv2.HoughLinesP`) identifies line segments:

- **Parameter Space**: The function accumulates edge points in a parameter space defined by distance (ρ) and angle (θ).

- **Thresholding and Voting**: A minimum number of votes (accumulated edge points) determines whether a line is present. Parameters like `threshold`, `minLineLength`, and `maxLineGap` influence line detection sensitivity.
- **Drawing Lines**: The function outputs detected lines as sets of endpoints (x_1, y_1, x_2, y_2), which are drawn on the original image using `cv2.line`.

3 Impact of Threshold Changes on Edge Detection

4 Edge Detection Parameters Used

Threshold Set	Lower Threshold	Higher Threshold
1	88	179
2	119	133
3	120	138

Table 1: Threshold Parameters for Edge Detection

5 Observations and Analysis

Each threshold set produced varying results, highlighting different road features in the provided images.

5.1 Lower Threshold: 88, Higher Threshold: 179

Figure 1: Detected edges using Lower Threshold 88, Higher Threshold 179

Results: The image generated using this set resulted in more edges being detected due to the low threshold value of 88, capturing most edges around the sidewalk, the road, and other regions.
Analysis: A lower threshold value allows more subtle gradients to be detected. However, this approach is also prone to picking up noise, especially on textured surfaces like sidewalks. The higher threshold (179) ensures that strong gradients remain highlighted.

5.2 Lower Threshold: 119, Higher Threshold: 133

Figure 2: Detected edges using Lower Threshold 119, Higher Threshold 133

Results: With these thresholds, edges are detected more selectively. The lines demarcating the sidewalk are clear, while unnecessary noise from textured regions is minimized.
Analysis: The higher *lower threshold* ensures that weaker edges are filtered out. The *higher threshold* value of 133 ensures that strong edges are not overlooked. As a result, the edges of the sidewalk are cleanly detected.

5.3 Lower Threshold: 120, Higher Threshold: 138

Figure 3: Detected edges using Lower Threshold 120, Higher Threshold 138

Results: Similar to the second set, these thresholds provide selective detection, further filtering weak gradients and focusing on the primary edges.
Analysis: A slight increase in the *higher threshold* (to 138) makes this set even more selective, showing clearer lines around the sidewalk borders.

5.4 Comparison on Blurring Size

side-walk-2-True-33-193-204.png

Figure 4: Blurring size (3, 3). Detected edges using Lower Threshold 193, Higher Threshold 204.

Figure 5: Blurring size (5, 5). Detected edges using Lower Threshold 153, Higher Threshold 165.

Figure 6: Blurring size (7, 7). Detected edges using Lower Threshold 89, Higher Threshold 128.

Results: We change the blurring size from 3 to 7, finding that the thresholds decrease with blurring size increasing.
Analysis: Blurring size increasing will make the edge become blurred, which makes edge detection hard.

5.5 Indoor Conditions

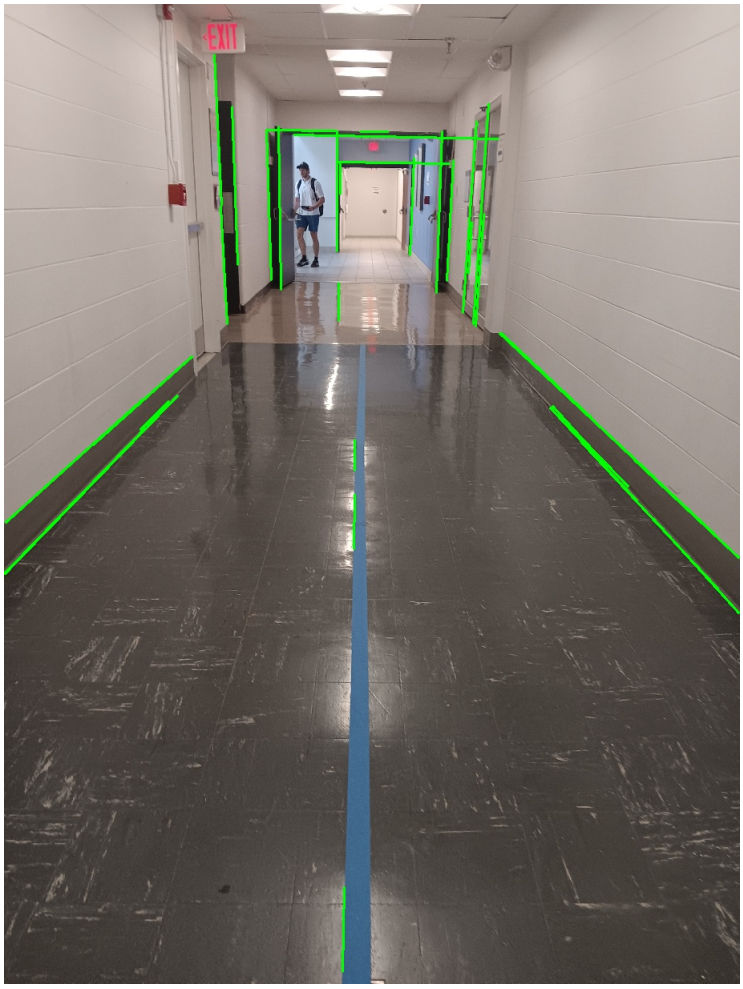

Figure 7: Detected edges using Lower Threshold 102, Higher Threshold 153.

Results: We experiment on indoor image, and we find the edge detected clearly. **Analysis:** From the above result, we can find that the canny edge detection works well in indoor conditions, which verifies the effectiveness of this algorithm.

5.6 Handle Different Conditions

When dealing with different conditions, such as variations in lighting, while applying Canny edge detection, we can use several approaches to handle these challenges:

- Preprocessing: Before applying the Canny edge detection, you can perform preprocessing techniques to enhance the image quality and mitigate the effects of lighting variations. Techniques like histogram equalization, contrast stretching, or adaptive thresholding can improve the overall image quality.
- Lighting Correction: To specifically address lighting variations, you can apply lighting correction techniques to normalize the image's illumination. Histogram matching, gamma correction, or homomorphic filtering can adjust the image's lighting conditions and improve the edge detection results.
- Multiple Thresholds: Instead of using a single global threshold for Canny edge detection, you can consider using multiple thresholds tailored to different lighting conditions. Adaptive thresholding algorithms can dynamically determine the appropriate thresholds based on local image properties, allowing for better adaptation to varying lighting conditions.

5.7 Suggestions on Running on Pi.

The aspberry Pi is a piece of hardware with a smaller processor and less image processing power. We can resize the image or compress the pixel size to make our application run successfully on the platform. We use the cv2.resize API in our codes to achieve this.