Canny Edge Operator-based Smart Traffic Control System for Intelligent Traffic Management

S. Spandana[1], Kalleda Appanolla Architha[2], Patlola Suraj Reddy[2], Vadde Indu[2], Tirumala Teja Harshitha[2], Munugala Bhanu Prakash[2]

[1]Assistant Professor,[2]UG Student,[1,2]Department of Computer Science and Engineering
[1,2]Malla Reddy Engineering College and Management Science, Kistapur, Medchal-501401, Hyderabad, Telangana, India

ABSTRACT

Traffic congestion has become a pervasive issue in urban areas, leading to increased travel times, fuel consumption, and environmental pollution. The Smart Traffic Control System utilizing the Density-Based Canny Edge Detection Algorithm addresses the challenge of inefficient traffic management. Traditional traffic control systems often lack adaptability to dynamic traffic conditions, resulting in suboptimal signal timings and increased congestion. The project seeks to leverage advanced image processing techniques to provide real-time insights into traffic patterns, enabling more intelligent and responsive signal control. The significance of the Smart Traffic Control System lies in its potential to significantly improve the efficiency of traffic flow and reduce congestion in urban environments. By incorporating the Canny Edge Detection algorithm, the system can accurately detect and analyze traffic density, leading to more informed decisions about signal timings. The adaptive nature of the system allows it to respond dynamically to changing traffic conditions, optimizing the allocation of green signal time. This adaptability is crucial in addressing the unpredictable and fluctuating nature of urban traffic, ultimately contributing to reduced travel times, fuel consumption, and environmental impact.

Keywords: Road traffic monitoring, Traffic congestion, OpenCV, Edge detection, Canny operator.

1. INTRODUCTION

As the population of modern cities increases day by day, the number of vehicles correspondingly increases, causing traffic congestion problems. Heavy traffic creates many problems and challenges in cities or areas with a high population, including wasted time, fuel and money. It also indirectly affects the environment by increasing pollution. Moreover, congestion problems cause stress to individuals as a result of missed opportunities and lost time. The traffic load is mainly dependent on parameters such as special events, construction, unpredictable accidents, weather, time, day, and season. If these parameters are not considered in a traffic control system, further delays can result. Often, to solve heavy traffic congestion problems, new roads are constructed [8]. However, one major drawback of new roads is that they can actually make their surrounding areas even more congested. As we know, transportation systems are the backbone of the economic growth and development of many countries. It is therefore important to have an efficient, fast, and economical traffic management system to promote national development. A suitable traffic monitoring system should use new techniques for counteracting the increasing number of vehicles on roads. In the present study, we examined automated and intelligent control systems because of their ease of implementation according to the traffic situation and applications for safely improving traffic flow in the current transportation system. Also, due to limited resources, intelligent traffic control systems will likely come to play an increasingly important role in traffic management [1]. The traffic congestion at a busy junction, different traffic management techniques are available. For continuously changing real-time situations, no one technique is perfect. Herein, we focus on highlighting a few traffic management techniques that are applicable to continuously varying real-time traffic. Specifically, our proposal is first based on

observing traffic on the road from received images: A red light indicates an empty road, and a green light indicates traffic on load. From here, we can adjust the duration of the green light according to traffic density. So that we might be able to resolve the problem of heavy traffic with proper management. The overall aim of the smart traffic control system which we have developed in the present study may help to reduce the waiting time of each lane of vehicles.

Applications

The applications of the Smart Traffic Control System are diverse and impactful.

- Firstly, it can be deployed at intersections and road junctions in urban areas to enhance traffic signal control, reducing wait times for vehicles and improving overall traffic flow.
- Additionally, the system can be integrated into larger smart city initiatives, contributing to comprehensive urban planning and management. The real-time traffic data generated by the system can be valuable for transportation authorities, allowing them to make data-driven decisions to optimize citywide traffic patterns.
- Furthermore, the project's adaptability and scalability make it applicable to various urban settings, from smaller municipalities to larger metropolitan areas, addressing the common challenge of traffic congestion in diverse contexts.

2. LITERATURE SURVEY

Anjani researched on the smart traffic control system availing image processing and using canny edge detection as an instrument for measuring the density of the traffic. the image of CCTV is taken and processed through the canny edge detection and counting the white pixels and calculating the green time using matching probability with reference image [1]. Taqi Tahmid proposes a traffic control system have been presented For this purpose, four sample images of different traffic scenario. Upon completion of edge detection, the similarity between sample images with the reference image has been calculated. Using this similarity, time allocation has been carried out for each individual image in accordance with the time allocation algorithm [3]. S. Lee's research on the system uses a camera-based density measurement method to estimate traffic flow and detect traffic congestion. The density measurement method uses the number of vehicles passing through a certain area over a period of time to estimate traffic flow. The system also uses Canny edge detection to detect the presence of vehicles at intersections and estimate their speed and direction of movement [4].

P. Ramya research on the system consists of three main modules: image processing, fuzzy logic control, and signal control. The image processing module uses Canny edge detection to detect the edges of vehicles in real-time video footage captured by cameras installed at intersections. The fuzzy logic control module then uses the edge density information to determine the appropriate signal timings based on the traffic volume and congestion level [5]. H. Jahanbakhsh research on The Canny edge detection algorithm is used to extract the edges of vehicles in the video input, which are then passed to an ANN to estimate the number of vehicles in each lane and their speeds. This information is then used to dynamically adjust the signal timings to optimize traffic flow [6]. V. Rajkumar proposes a smart traffic control system that uses Canny edge detection and image processing techniques to detect and classify vehicles, estimate traffic flow, and optimize traffic signal timings. The system is designed to improve traffic flow and reduce congestion on urban roads. The proposed system includes a camera-based vehicle detection module that uses Canny edge detection and morphological operations to extract vehicle features and classify them using machine learning algorithms [7]. The Sobel operator, also referred to as Sobel – Feldman Operator or Sobel Filter [8]. Sobel Operator uses 3x3 convolution kernels as where each of the masks responsible in calculating the gradient in both vertical and horizontal direction.

Based on the mask [9], it can be seen that the horizontal (x) orientation is expanding in the “right” direction while the vertical (y) orientation is increasing in the “up” direction. The Sobel operator has larger convolution kernel; it smoothens the input image to a greater extent making the operator to be less sensitive to noise [10]. However, due to its larger convolution kernel, it is much slower to compute as compared to the Roberts edge detector operator. Laplacian of Gaussian (Log) was invented by Marr and Hildreth back in 1980 [12]. It belongs to second spatial derivative in which an image is smoothed with Gaussian filter to get rid of the noise sensitivity [13]. It is then using the zero-crossing method operated by Laplacian operator to extract edges for the smoothed image. One of the drawbacks of the operator is it tends to weaken the image features at the same time which results in some edges are unable to be detected efficiently [14]. In 1986, John F. Canny developed an edge detection operator using a multi-stage algorithm to detect a wide range of image edges called the Canny edge detector [15]. Canny edge operation detector takes a grayscale image as input and produces an image showing the positions of tracked intensity discontinuities as an output [15]. It works by smoothing the image first, then uses the gradient of the image to highlight regions with high spatial derivatives. It then tracks those regions to delete any pixels that are not at the maxi. Active contours, also known as snakes was first introduce by Kass in 1988. Active contour widely used in image processing to locate object boundaries. This method used to energy-minimize model to detect the edges forming an object boundary with given an initial guess [16]. The model minimizes the energy function from an external and internal force to form the boundary edge of an object in the image. The model is efficient but it sensitive to image noise. It has the difficulty of the initial curve placement due to the local characteristic of image gradients.

3. EXISTING SYSTEM

Traditional traffic control systems typically rely on fixed signal timings and pre-defined traffic signal plans. These systems use sensors, such as induction loop detectors or cameras, to monitor traffic conditions at intersections. Based on the collected data or predetermined schedules, the traffic signals cycle through red, green, and yellow phases. These systems often operate on fixed time intervals and lack the adaptability to respond dynamically to changing traffic patterns. As a result, they may lead to inefficient signal timings, increased congestion, and longer wait times for vehicles, particularly during peak hours or unexpected traffic incidents.

Limitations

- Static Signal Timings: Traditional systems use static signal timings, which are predetermined and do not adapt to real-time traffic conditions. This can result in inefficient signal control during periods of varying traffic density.
- Limited Responsiveness: These systems lack the ability to respond dynamically to changing traffic patterns, unexpected events, or emergencies. As a result, they may not optimize signal timings to address immediate traffic needs.
- Inefficiency in Congested Situations: During peak traffic hours or in congested scenarios, fixed signal timings may contribute to increased congestion and longer wait times for vehicles, leading to frustrated commuters and potential safety concerns.
- Dependency on Predictive Models: Traditional systems often rely on predictive models based on historical data. While useful, these models may not accurately reflect real-time fluctuations in traffic patterns, limiting the system's responsiveness.
- Challenges in Scalability: As urban areas grow and traffic patterns evolve, traditional systems may face challenges in scaling to meet the increasing complexity of traffic management, making them less adaptable to the dynamic nature of modern cities.

— Environmental Impact: Inefficient traffic flow contributes to increased fuel consumption and emissions, negatively impacting the environment. Traditional systems may not effectively address these environmental concerns.

The limitations of traditional traffic control systems highlight the need for more intelligent and adaptive approaches, such as the Smart Traffic Control System utilizing the Density-Based Canny Edge Detection Algorithm. This modern approach aims to overcome these shortcomings by incorporating advanced image processing techniques to provide real-time insights and dynamic signal control, ultimately contributing to more efficient and environmentally friendly urban traffic management.

4. PROPOSED METHODOLOGY

This project aims to provide a user-friendly interface for analyzing traffic images, applying image processing techniques, and making decisions about traffic signal timings based on the detected density using the Canny Edge Detection algorithm. The system assesses the image pixel counts to determine the traffic conditions and suggests appropriate green signal times.

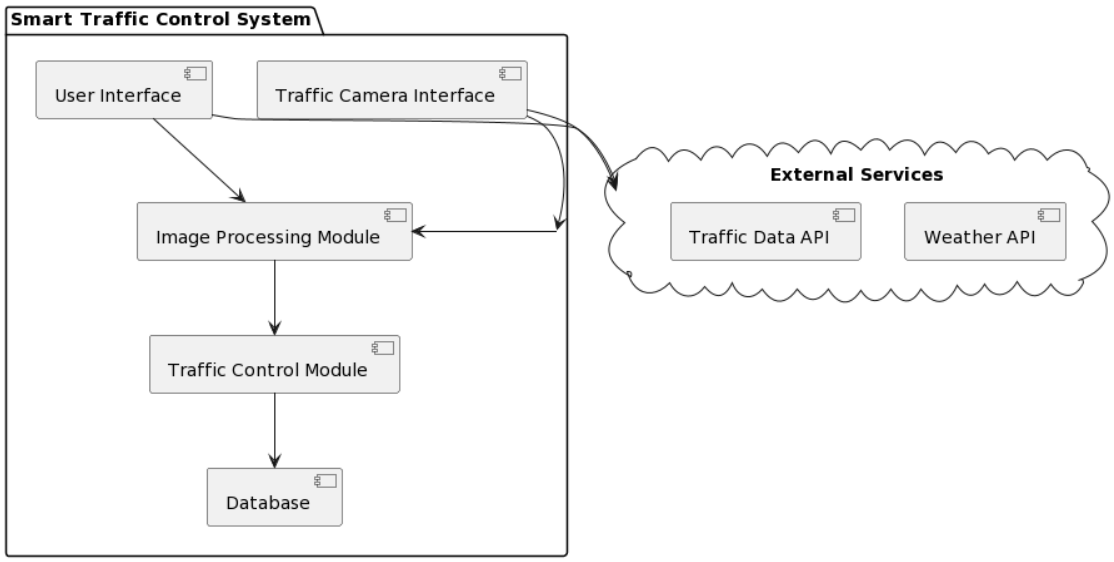


Figure 1: System architecture of proposed STCS.

The overview of the project is as follows:

1. GUI Setup:

— The Tkinter library is used to create a graphical user interface (GUI) for the project.
— The main window (main) is created with a specified size and title.

2. Functionality: The GUI provides several buttons for different functionalities related to traffic image processing.

3. Image Upload:

— Users can upload a traffic image using the "Upload Traffic Image" button.
— The selected image's path is displayed on the GUI.
— Image Processing with Canny Edge Detection:
— The "Image Preprocessing Using Canny Edge Detection" button triggers the application of the Canny Edge Detection algorithm to the uploaded image.

— The processed image is saved to the disk, and both the sample and reference images are displayed using Matplotlib.

4. White Pixel Count:

— The "White Pixel Count" button calculates the total count of white pixels in both the sample and reference images.
— The counts are then displayed using a message box.

5. Green Signal Time Allocation:

— The "Calculate Green Signal Time Allocation" button uses the white pixel counts to determine the traffic density.
— Depending on the density, different messages are displayed indicating the recommended green signal time.

6. Exit Button: The "Exit" button closes the Tkinter application.

4.2 Canny edge detector

The Canny Edge Detection algorithm, as implemented in the provided code, follows a multi-step process to identify edges in an image. It begins by smoothing the input image using a Gaussian filter to reduce noise and enhance relevant features. Subsequently, Sobel filters are applied to compute the image's gradient in both the x and y directions. Non-maximum suppression is then utilized to highlight local maxima in the gradient direction, effectively thinning the edges. Following this, double thresholding is applied to classify pixels as strong or weak edges based on intensity values. Finally, hysteresis is employed to connect weak edges to strong edges, forming continuous edge contours. The entire process is encapsulated in a class called CannyEdgeDetector, allowing the algorithm to be applied to a list of input images. The resulting list contains images where edges have been accurately detected and highlighted, providing valuable information for subsequent image analysis and processing in applications such as the Smart Traffic Control System mentioned in the larger code context.

4.3 Preprocessing

Image preprocessing is a critical step in computer vision and image analysis tasks. It involves a series of operations to prepare raw images for further processing by algorithms or neural networks. Here's an explanation of each step-in image preprocessing:

Step 1. Image Read: The first step in image preprocessing is reading the raw image from a source, typically a file on disk. Images can be in various formats, such as JPEG, PNG, BMP, or others. Image reading is performed using libraries or functions specific to the chosen programming environment or framework. The result of this step is a digital representation of the image that can be manipulated programmatically.

Step 2. Image Resize: Image resize is a common preprocessing step, especially when working with machine learning models or deep neural networks. It involves changing the dimensions (width and height) of the image. Resizing can be necessary for several reasons:

- Ensuring uniform input size: Many machine learning models, especially convolutional neural networks (CNNs), require input images to have the same dimensions. Resizing allows you to standardize input sizes.
- Reducing computational complexity: Smaller images require fewer computations, which can be beneficial for faster training and inference.

- Managing memory constraints: In some cases, images need to be resized to fit within available memory constraints.

When resizing, it's essential to maintain the aspect ratio to prevent image distortion. Typically, libraries like OpenCV or Pillow provide convenient functions for resizing images.

Step 3. Image to Array: In this step, the image is converted into a numerical representation in the form of a multidimensional array or tensor. Each pixel in the image corresponds to a value in the array. The array is usually structured with dimensions representing height, width, and color channels (if applicable).

For grayscale images, the array is 2D, with each element representing the intensity of a pixel. For color images, it's a 3D or 4D array, with dimensions for height, width, color channels (e.g., Red, Green, Blue), and potentially batch size (if processing multiple images simultaneously).

The conversion from an image to an array allows for numerical manipulation and analysis, making it compatible with various data processing libraries and deep learning frameworks like NumPy or TensorFlow.

Step 4. Image to Float32: Most machine learning and computer vision algorithms expect input data to be in a specific data type, often 32-bit floating-point numbers (float32). Converting the image array to float32 ensures that the pixel values can represent a wide range of intensities between 0.0 (black) and 1.0 (white) or sometimes between -1.0 and 1.0, depending on the specific normalization used.

This step is essential for maintaining consistency in data types and enabling compatibility with various machine learning frameworks and libraries. It's typically performed by dividing the pixel values by the maximum intensity value (e.g., 255 for an 8-bit image) to scale them to the [0.0, 1.0] range.

Step 5. Image to Binary: Image binarization is a process of converting a grayscale image into a binary image, where each pixel is represented by either 0 (black) or 1 (white) based on a specified threshold. Binarization is commonly used for tasks like image segmentation, where you want to separate objects from the background.

The process involves setting a threshold value, and then for each pixel in the grayscale image, if the pixel value is greater than or equal to the threshold, it is set to 1; otherwise, it is set to 0.

Binarization simplifies the image and reduces it to essential information, which can be particularly useful in applications like character recognition or object tracking, where you need to isolate regions of interest.

5. RESULTS AND DISCUSSION

5.1 Implementation

This project implements a Python script that uses the Tkinter library to create a graphical user interface (GUI) for a Smart Traffic Control System. The system aims to enhance traffic information through a Density-Based Canny Edge Detection Algorithm. Here is the implementation of code:

Step 1 – Imports:

- messagebox, Tk, simpledialog, filedialog: Classes from the Tkinter library for displaying message boxes and handling file dialogs.
- numpy: Library for numerical operations.
- askopenfilename: Function for opening a file dialog to select a file.

- CannyEdgeDetector: A custom module (presumably in a file named CannyEdgeDetector.py) for implementing the Canny Edge Detection algorithm.
- skimage, mpimg, os, sm, cv2, plt: Various libraries for image processing, file handling, and visualization.

Step 2 – Tkinter Setup:

- Creation of the main Tkinter window (main).
- Setting the title and geometry of the window.

Step 3 – Global Variables:

- filename: Holds the path of the selected traffic image file.
- refrence_pixels, sample_pixels: Variables to store pixel counts for reference and sample images.

Step 4 – Functions:

- rgb2gray: Converts an RGB image to grayscale.
- uploadTrafficImage: Opens a file dialog to select a traffic image and displays the selected file path.
- visualize: Visualizes images using Matplotlib.
- applyCanny: Applies the Canny Edge Detection algorithm to the selected image, saves the result, and visualizes the original and processed images.
- pixelcount: Counts the number of white pixels in the processed images and displays the counts using a message box.
- timeAllocation: Calculates and displays the green signal allocation time based on the ratio of sample to reference white pixels.
- exit: Closes the Tkinter window.

Step 5 – GUI Elements:

- Labels, buttons, and other GUI elements are created and configured with specific functionalities.
- The GUI provides options to upload an image, process it using Canny Edge Detection, count white pixels, calculate green signal time allocation, and exit the application.

Step 6 – Main Loop: The Tkinter main loop (main.mainloop()) is initiated to start the GUI application.

CannyEdgeDetector implements the Canny Edge Detection algorithm. The Canny Edge Detector is a popular edge detection algorithm used in image processing. This process involves smoothing with a Gaussian filter, gradient calculation, non-maximum suppression, double thresholding, and edge tracking by hysteresis. The resulting images with detected edges are stored in the imgs_final list.

1. Imports:

- ndimage, convolve: Functions from scipy for convolution operations and image filtering.
- misc: Module from scipy for miscellaneous image processing functions.
- numpy: Library for numerical operations.

2. Class Definition:

— CannyEdgeDetector: The class takes several parameters during initialization, including a list of images (imgs), standard deviation of the Gaussian kernel (sigma), kernel size, weak and strong pixel values, low and high thresholds for edge detection.

3. Initializer (__init__): Initializes various parameters, including input images, sigma, kernel size, pixel values, and thresholds.

4. Gaussian Kernel Function (gaussian_kernel): Generates a 2D Gaussian kernel with a given size and standard deviation.

5. Sobel Filters Function (sobel_filters):

— Applies Sobel filters to calculate image gradients in the x and y directions.
— Returns gradient magnitude (G) and gradient direction (theta).
— Non-Maximum Suppression Function (non_max_suppression):
— Performs non-maximum suppression to thin the edges.
— Compares pixel values in the gradient direction and keeps only the local maxima.

6. Threshold Function (threshold):

— Applies double thresholding to identify potential edge pixels.
— Pixels with values above the high threshold are considered strong edges, and those below the low threshold are ignored.

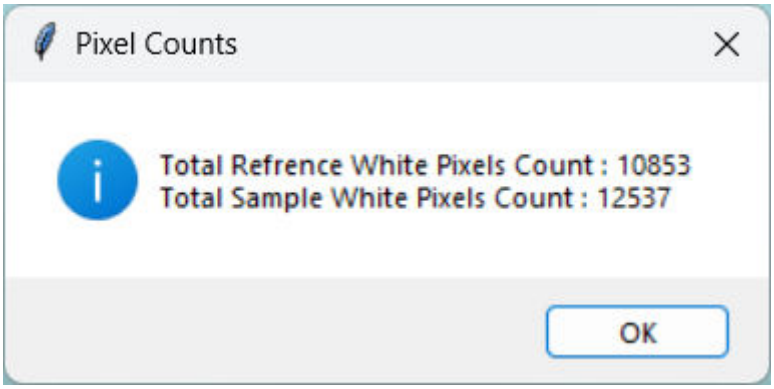
7. Hysteresis Function (hysteresis):

— Performs hysteresis to link weak edges to strong edges.
— If a weak edge pixel is connected to a strong edge pixel, it is considered part of the edge.

8. Edge Detection Function (detect):

— Applies the entire Canny Edge Detection process to each image in the input list (imgs).
— Returns a list of processed images.

5.2 Simulation results



6. CONCLUSION

The Smart Traffic Control System employing the Density-Based Canny Edge Detection Algorithm, as demonstrated in the provided code, represents a significant step towards enhancing traffic information processing and signal control. The utilization of the Canny Edge Detection algorithm allows for precise edge identification in traffic images, and the subsequent analysis, including white pixel counting and signal time allocation calculation, contributes to a more informed and adaptive traffic management approach. In conclusion, the project showcases the potential to significantly improve traffic flow and reduce congestion through advanced image processing techniques. Looking ahead, several avenues for future development present themselves. Real-time traffic monitoring can be implemented to enable continuous and dynamic adjustments to signal timings. Additionally, efforts to ensure fault tolerance, energy efficiency, and integration with broader smart city initiatives would contribute to the system's robustness and effectiveness. Ultimately, ongoing research and development in these areas will position the Smart Traffic Control System as a key player in the evolution of intelligent and adaptive urban traffic management solutions.

REFERENCES

[1] A Anjani, "Density Based Smart Traffic Control Using Canny Edge Detection Algorithm", JOURNAL OF ALGEBRAIC STATISTICS Volume 13, No. 3, 2022.

[2] A.J. Shakadwipi, "Smart Traffic Control System by Using Image Processing", IOSR Journal of Computer Engineering Volume 22, Issue 6, 2020.

[3] Taqi Tahmid, "Density Based Smart Traffic Control System Using Canny Edge Detection Algorithm for Congregating Traffic Information", EICT, 2017.

[4] S. Lee, "Density Based Smart Traffic Control System Using Canny Edge Detection Algorithm", 2021.

[5] P. Ramya, "Smart Traffic Control System Based on Canny Edge Detection and Fuzzy Logic", 2017.

[6] H. Jahanbakhsh, "Real-Time Traffic Control Using Canny Edge Detection and Artificial Neural Networks", 2017.

[7] V. Rajkumar, "Smart Traffic Control System Based on Canny Edge Detection and Image Processing", 2019.

[8] B. C. C. Meng, U. K. Ngah, B. E. Khoo, I. L. Shuaib, and M. E. Aziz, “A Framework of MRI Fat Suppressed Imaging Fusion System for Femur Abnormality Analysis,” Procedia Comput. Sci., vol. 60, pp. 808–817, 2015.

[9] D. Srivastava, R. Kohli, and S. Gupta, “Implementation and statistical comparison of different edge detection techniques,” in Advances in Computer and Computational Sciences, Springer, 2017, pp. 211–228.

10] M. A. Ansari, D. Kurchaniya, and M. Dixit, “A Comprehensive Analysis of Image Edge Detection Techniques,” Int. J. Multimed. Ubiquitous Eng., vol. 12, no. 11, pp. 1–12, 2017.

[10] T. Nagasankar and B. Ankaryarkanni, “Performance Analysis of Edge Detection Algorithms on Various Image Types,” Indian J. Sci. Technol., vol. 9, no. 21, pp. 1–7, 2016.

[11] S. Veni, “Image Processing Edge Detection Improvements and Its Applications,” Int. J. Innov. Sci. Eng. Res., vol. 3, no. 6, pp. 51–54, 2016.

[12] J. Hu, X. Tong, Q. Xie, and L. Li, “An Improved, Feature-Centric LoG Approach for Edge Detection,” in International Conference on Computational Science and Its Applications, 2016, pp. 474–483.

[13] S. Yuan, S. E. Venegas-Andraca, C. Zhu, Y. Wang, X. Mao, and Y. Luo, “Fast Laplacian of Gaussian Edge Detection Algorithm for Quantum Images,” in 2019 IEEE International Conferences on Ubiquitous Computing & Communications (IUCC) and Data Science and Computational Intelligence (DSCI) and Smart Computing, Networking and Services (SmartCNS), 2019, pp. 798–802.

[14] W. Rong, Z. Li, W. Zhang, and L. Sun, “An improved CANNY edge detection algorithm,” in 2014 IEEE International Conference on Mechatronics and Automation, 2014, pp. 577–582.

[15] L. Wang, Y. Chang, H. Wang, Z. Wu, J. Pu, and X. Yang, “An active contour model based on local fitted images for image segmentation,” Inf. Sci. (Ny)., vol. 418, pp. 61–73, 2017.