Object Detection and Segmentation in Urban Scenes using MATLAB

Muhammad Zauraiz Ahmad(F2023266573)[1], Syed Muhammad Adan Mehdi(F2023266584)[2], Hamza Irshad(F2023266483)[3], Fatima Nawaz(F2023266486)[4]

[1] Department of Computer Science, University of Management And Technology

Abstract :

This project explores various classical image processing techniques for object detection and segmentation in real-world urban scenes using MATLAB. Our approach includes spatial and frequency domain filtering, edge detection, global and adaptive thresholding, and segmentation methods like marker-controlled watershed and connected components. Adaptive thresholding and Canny edge detection provided the best results. To validate our pipeline, simple built-in images were also tested, where these methods worked perfectly, confirming the complexity of real urban data as the main challenge.

Index Terms: Image Segmentation, Urban Scenes, Thresholding, Object Detection, MATLAB

1 Introduction

Object detection and segmentation play a vital role in modern computer vision applications, particularly in urban environments where real-time analysis of visual data is required. From autonomous driving and surveillance systems to infrastructure planning, the ability to detect and isolate objects such as pedestrians, vehicles, and road signs is essential. While deep learning methods have recently become dominant in this field, traditional image processing techniques still offer valuable insights and remain important for low-resource environments and educational purposes.

This project explores classical image processing methods for detecting and segmenting objects in urban scenes using MATLAB. The focus is on evaluating how well spatial filtering, frequency domain analysis, edge detection, and thresholding techniques perform when applied to real-world images. The project was conducted as part of the CS406 Digital Image Processing course.

Urban scenes are inherently complex due to factors such as uneven lighting, object occlusion, cluttered backgrounds, and scale variation. These challenges often make it difficult for basic segmentation algorithms to isolate objects accurately. To test our pipeline, we selected images from the Cityscapes dataset — a popular collection of urban street views — and applied a structured sequence of techniques including adaptive thresholding, Canny edge detection, watershed segmentation, and connected components analysis.

While some methods like adaptive thresholding and Canny edge detection produced clear and reliable results, others such as watershed and connected components yielded mixed outcomes due to the complexity of the scenes. To verify the correctness of our implementation, we also applied the same techniques to simple test images (like coins.png) where the results were nearly perfect, demonstrating that the methods are valid but sensitive to scene complexity.

This report documents each stage of the project: from dataset selection and preprocessing to segmentation, object detection, and final labeling. It also highlights what worked, what didn't, and how image complexity impacted the outcomes.

2 Dataset Description

The primary dataset used in this project is the Cityscapes Dataset, a widely recognized benchmark in computer vision research focused on semantic understanding of urban street scenes. It provides high-resolution images captured from vehicle-mounted cameras across 50 different cities in Germany. These images depict real-world environments including roads, pedestrians, vehicles, buildings, and street signs — all under various weather and lighting conditions.

For this project, we worked specifically with the leftImg8bit image subset, which contains RGB images with a resolution of 2048×1024 pixels. We deliberately chose to retain the original resolution without any downscaling in order to preserve visual clarity and detail. This helped ensure that smaller objects, fine textures, and boundary features remained distinguishable. The only modification made was the conversion from RGB to grayscale for most processing steps.

We manually selected six images from the train subset based on visual diversity:

- Cologne - 2 images
- Bremen - 1 image
- Hamburg - 2 images
- Hanover - 1 image

These images reflect a range of scene complexities including intersections, construction zones, mixed lighting, and both sparse and dense object arrangements. Our goal was to evaluate how classical segmentation methods perform under varying urban conditions.

Additionally, we used MATLAB's built-in image coins.png, a simple grayscale image with well-separated circular shapes, to validate our segmentation pipeline. It served as a control test to show that our methods work effectively on clean, low-noise data — confirming that any performance limitations observed in Cityscapes images were due to scene complexity rather than algorithm flaws.

Examples of these scenes are shown in Figure 1.

Figure 1. Sample images selected from the Cityscapes dataset. Left: Bremen, Center: Cologne, Right: Hanover.

3 Methodology

This project was implemented as a step-by-step image processing pipeline using MATLAB. We focused on classical image processing techniques applied in a structured order to understand how well traditional methods perform in complex urban scenes. All images were kept at their original resolution (2048×1024) to preserve detail, and grayscale conversion was applied where necessary. The methodology was broken down into the following major components, reflecting the folder-wise structure and logic used during the project.

3.1 Spatial Filtering

Spatial filtering helps reduce noise and smooth images before segmentation. We applied three commonly used filters:

- **Averaging Filter:** A mean filter using a square mask:

$$g(x,y) = \frac{1}{n^2} \sum_{i=-k}^{k} \sum_{j=-k}^{k} f(x+i, y+j)$$

- **Gaussian Filter:** A weighted smoothing filter:

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

- **Median Filter:** Replaces a pixel with the median of its neighborhood, excellent for removing impulse noise.

These filters allow for suppression of small-scale variations while preserving object edges to prepare for segmentation.

3.2 2. Frequency Domain Filtering

We explored the frequency domain using the 2D Discrete Fourier Transform:

$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) \cdot e^{-j2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)}$$

Filtering was performed by multiplying the transformed image with frequency masks:

$$g(x,y) = \mathcal{F}^{-1}[F(u,v) \cdot H(u,v)]$$

We tested:

- **Low-pass filters** – blurred the image and removed noise
- **High-pass filters** – enhanced edges and sharp transitions
- **Band-pass filters** – kept mid-frequency structures

Low frequencies represent *slow variations* (smooth backgrounds), while high frequencies correspond to *rapid changes* (edges, textures, noise). Frequency domain filtering allows us to selectively enhance or suppress different spatial features of an image based on these variations.

Low-Pass Filters (LPF): These retain slowly changing regions and suppress edges and noise. A low-pass mask allows the center of the frequency spectrum to pass, removing high-frequency detail:

$$H_{LP}(u,v) = \begin{cases} 1, & D(u,v) \le r \\ 0, & D(u,v) > r \end{cases}$$

where $D(u,v) = \sqrt{(u - M/2)^2 + (v - N/2)^2}$.

High-Pass Filters (HPF): These remove smooth regions and enhance sharp transitions:

$$H_{HP}(u,v) = \begin{cases} 0, & D(u,v) \le r \\ 1, & D(u,v) > r \end{cases}$$

Band-Pass Filters (BPF): keeps only medium-level details (removes both very smooth and very sharp parts).

$$H_{BP}(u,v) = \begin{cases} 1, & r_1 < D(u,v) < r_2 \\ 0, & \text{otherwise} \end{cases}$$

After applying these masks, the filtered image is returned to the spatial domain using the inverse Fourier Transform:

$$g(x,y) = \mathcal{F}^{-1}\left[F(u,v) \cdot H(u,v)\right]$$

3.3 3. Noise Addition and Restoration

To simulate realistic conditions, we added:

- **Salt & Pepper Noise** – random black and white pixels
- **Gaussian Noise** – smooth noise across the image
- **Speckle Noise** – looks grainy, common in radar images

For restoration:

- **Median Filtering** – effective for impulse noise (salt and pepper)
- **Adaptive Wiener Filtering** – effective for Gaussian and speckle noise

Each method was applied to demonstrate how classical filters can restore corrupted images and prepare them for further analysis.

3.4 Segmentation Techniques

The goal of segmentation is to isolate foreground objects from the background. We used several methods to evaluate their effectiveness:

- **Canny Edge Detection:** Finds clear edges by detecting changes in brightness.
- **Otsu's Thresholding:** Automatically finds the best value to make the image black-and-white.

$$\sigma_b^2(t) = \omega_1(t)\omega_2(t)[\mu_1(t) - \mu_2(t)]^2$$

- **Adaptive Thresholding:** Finds the threshold locally for different image areas.
- **Watershed Segmentation:** Sees the image as a 3D surface and "floods" regions.
- **Connected Components Analysis:** Finds and labels separate white regions in binary images.

Each method had different strengths. Adaptive thresholding provided the clearest object boundaries, while Canny was useful for edge visualization. Watershed and connected components worked best in simpler scenes.

3.5 Object Labeling and Detection

Once we segmented the images, we counted and labeled the objects. We used:

- `bwconncomp()` to count the number of objects
- `regionprops()` to draw rectangles and mark centers
- Text labels like "Obj 1", "Obj 2", etc. on each detected object)

This helped us see which methods actually separated meaningful objects.

3.6 Validation Using Simple Images

To confirm that segmentation methods were implemented correctly, we applied them to MATLAB's built-in `coins.png` image. This image contains distinct, well-separated circular shapes.

All segmentation methods produced ideal results on this simpler image. Watershed segmentation, which struggled on complex urban scenes, worked correctly on this case. This demonstrated that limitations in object detection on our selected dataset were primarily due to image complexity, not implementation issues.

4 Experiments and Results

To test the methods described in the previous section, we applied each technique to multiple images taken from the Cityscapes dataset. These images included a variety of urban scenes, including streets, cars, pedestrians, traffic lights, and signs. We selected scenes with different brightness levels, clutter, and complexity to observe how well each method performed.

We saved each output image and visually compared the results for different filters and segmentation techniques.

4.1 Spatial Filtering Results

We applied averaging, Gaussian, and median filters on selected images. These helped reduce noise and smooth the image. Median filter gave the best visual output as it preserved edges while removing sharp dots.

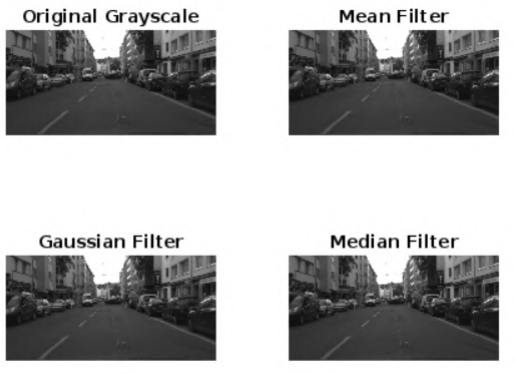

Figure 2. Spatial Filtering: (Top-Left) Original, (Top-Right) Mean Filter, (Bottom-left) Gaussian Filter , (Bottom-Right) Median Filter.

4.2 Frequency Filtering Results

We tested low-pass, high-pass, and band-pass filters with different radius values. Low-pass filters blurred the image, high-pass filters enhanced edges, and band-pass filters highlighted medium-level details.

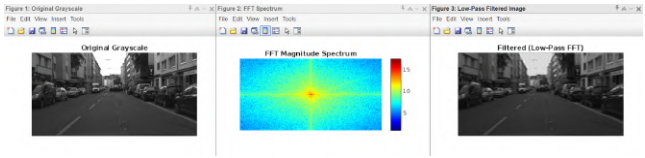

Figure 3. Low-pass Filter (Radius 50) on Cologne Image.

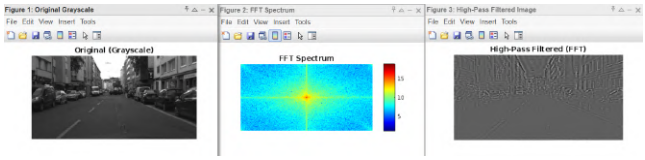

Figure 4. High-pass Filter (Radius 30) on Cologne Image.

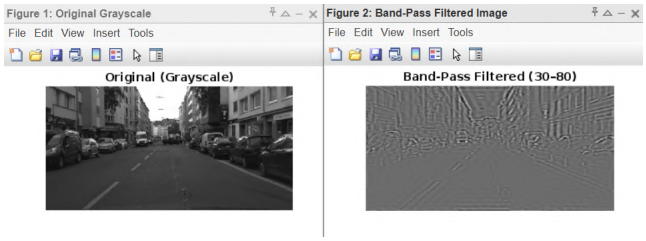

Figure 5. Band-pass Filter on Cologne Image.

4.3 Noise and Restoration Results

We added different types of noise and restored the image using suitable filters.

Salt and Pepper Noise + Median Filter:

Figure 6. Salt and Pepper noise and restoration on Bremen Image. (Top) Original(RGB), (Center) Noisy(Grayscale), (Bottom) Restored(Grayscale)

Gaussian Noise + Wiener Filter:

Figure 7. Gaussian noise and restoration using Wiener filter. (Top) Original(RGB), (Center) Noisy(Grayscale), (Bottom) Restored(Grayscale)

Speckle Noise + Wiener Filter:

Figure 8. Speckle noise and restoration using Wiener filter. (Top) Original(RGB), (Center) Noisy(Grayscale), (Bottom) Restored(Grayscale)

4.4 Segmentation Results

We applied three main segmentation methods:

Canny Edge Detection: Worked well on simple edges but struggled in very busy images.

Figure 9. Segmentation Methods: (Top) Original, (Bottom) Edge Detected Image.

Otsu's Thresholding: Struggled under variable lighting.

Figure 10. Segmentation Methods: Otsu's Thresholding (Top) Original, (Bottom) After Thresholding

Adaptive Thresholding: Worked best — it identified most objects clearly even in shadows.

Figure 11. Segmentation Methods: Adaptive Thresholding (Top) Original, (Bottom) After Thresholding

4.5 Object Detection and Counting

We tested object detection by labeling and counting objects using two main techniques:

- **Connected Components Analysis:** This technique detects groups of white pixels in a binary image and labels them as separate objects. It worked reasonably well in simpler images like Bremen, where objects are spaced out. However, in complex images like Cologne, it detected many small, irrelevant regions such as texture patches — resulting in false positives.

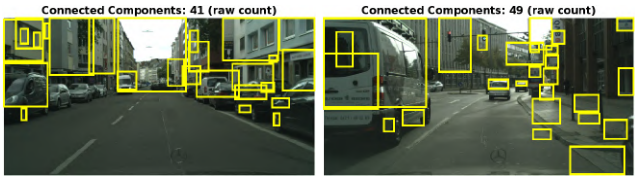

Figure 12. Labeled Output: (Left) Cologne (41 objects), (Right) Bremen (49 objects)

- **Watershed Segmentation:** This method views the image as a 3D landscape and tries to "flood" it from low-intensity areas to find boundaries between objects. It performed very poorly on complex urban scenes like Cologne and Bremen because of overlapping and touching objects, shadows, and background clutter. However, it worked much better on simpler images like `coins.png`, where objects are clearly separated.

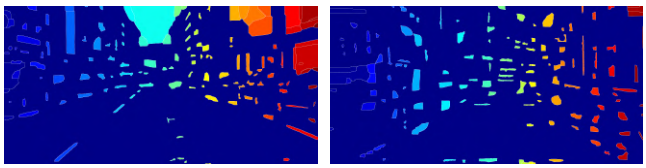

Figure 13. Watershed segmentation results: (Left) Cologne - overlapping patches, (Right) Bremen - incomplete separation.

4.6 Testing on Simple Images

To show our methods worked, we tested them on MATLAB's built-in 'coins.png' image. All segmentation methods worked perfectly on it, proving that the earlier challenges were due to image complexity.

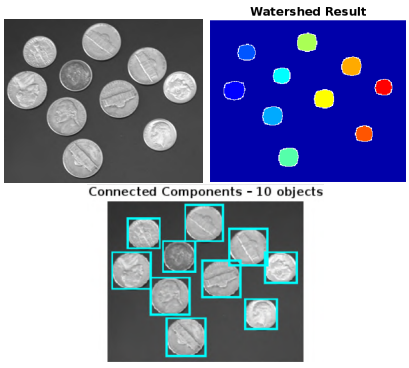

Figure 14. Testing on simple image: Coins. (Left) Original, (Center) Watershed, (Right) Connected Components

5 Discussion

Throughout the project, we applied classical image processing techniques to complex urban images from the Cityscapes dataset. While many methods worked well in theory and on simpler images, their performance varied significantly when tested on real-world scenes.

Performance Comparison

Among all segmentation methods, **adaptive thresholding** gave the best results.

Table 1
Comparison of Segmentation Methods

Method	Accuracy	Limitation
Canny	Medium	Detects edges, not full objects
Otsu	Low	Fails in uneven lighting
Adaptive	High	May over-segment small details
Watershed	Low	Fails in cluttered/overlapping scenes
Conn. Comp.	Low	False positives in complex scenes

Challenges Faced

A major challenge was the nature of the dataset — real-world images contain shadows, overlapping objects, perspective distortion, and a high amount of detail. Classical methods were not built for such conditions, and results were often cluttered or confusing.

Another issue was object size variation and background complexity. Some methods could not distinguish between people, vehicles, or road markings, especially when they were visually similar or very close to each other.

Lessons Learned

This project showed us both the strengths and the limitations of classical image processing techniques. While these methods are lightweight and easy to implement, they require careful tuning and are highly sensitive to image conditions. Adaptive approaches that consider local information performed better overall.

We also learned that sometimes even working methods (like watershed or connected components) may produce poor results if the image complexity is too high. Testing with simpler images helped confirm that the logic was correct — the limitation was with the input, not the method.

6 Conclusion

In this project, we explored a complete pipeline of classical image processing techniques using MATLAB, focusing on real-world urban scenes. We applied smoothing filters, frequency-based filtering, noise simulation, restoration, and various segmentation methods to understand their effectiveness on complex images.

From our experiments, it became clear that not all traditional methods are equally suited for real-world conditions. Adaptive thresholding consistently outperformed other techniques when it came to identifying distinct objects under varying lighting and background complexity. In contrast, global thresholding methods like Otsu struggled in scenes with shadows or uneven exposure. Watershed and connected components techniques worked well in simpler cases but produced cluttered or misleading results in crowded urban environments.

By comparing all results side-by-side, we were able to highlight the limitations of certain methods and validate our approach using built-in simpler images. We also learned the importance of preprocessing and image conditions in achieving reliable results.

Overall, this project helped us gain practical experience with foundational image processing concepts and showed that even basic methods, when properly applied, can still be effective in selective scenarios. For more robust results in real-world applications, however, a shift toward modern machine learning or deep learning methods would likely be necessary.

Group Members and Acknowledgements

Muhammad Zauraiz Ahmad	**Syed Muhammad Adan Mehdi**	**Hamza Irshad**	**Fatima Nawaz**
BSCS – Semester 4	BSCS – Semester 4	BSCS – Semester 4	BSCS – Semester 4

Acknowledgements: This project was completed as part of the CS406 course under the guidance of **Sir Jameel Ahmad**, Department of Computer Science, Spring 2025.