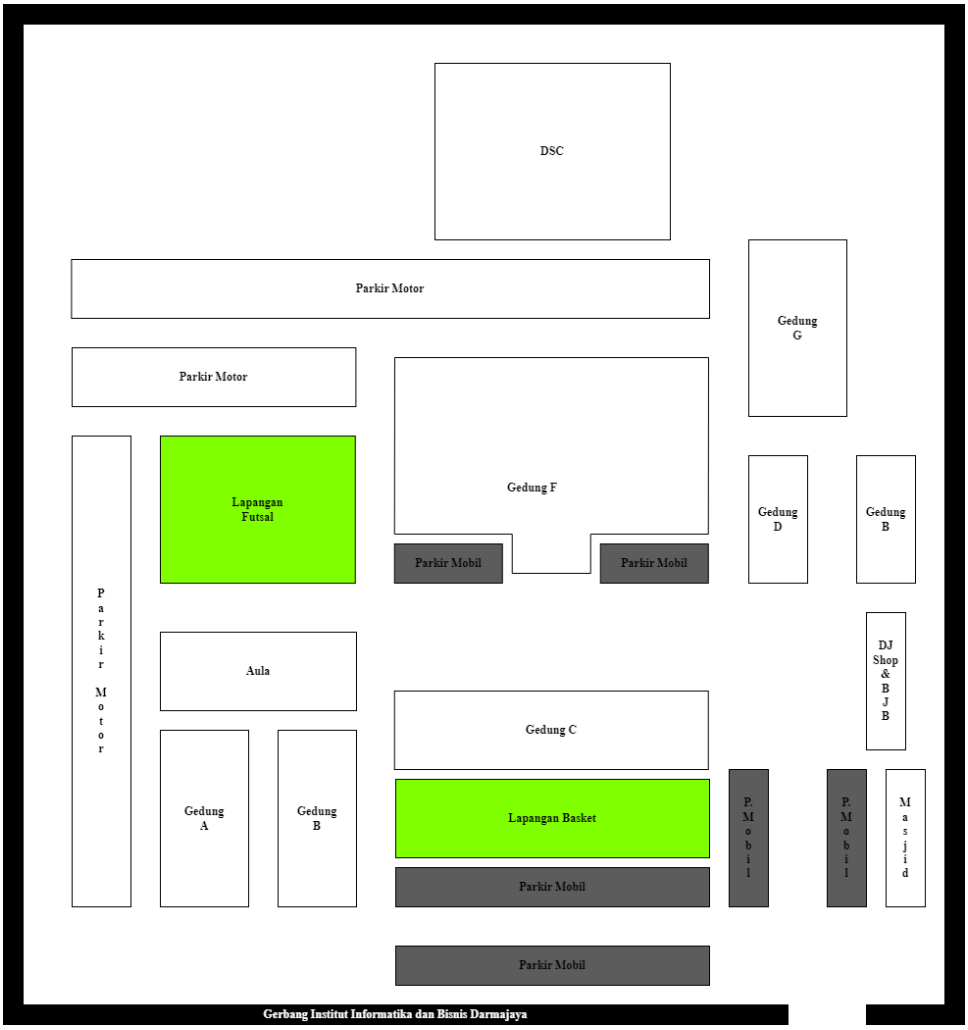
BAB II
LANDASAN TEORI

2.1 Parkir Mobil di Institut Informatika dan Bisnis Darmajaya

Area parkir mobil yang disediakan oleh Institut Informatika dan Bisnis Darmajaya memiliki beberapa spot, yaitu sebagai berikut:

a. Di sekitar gedung B dekat pintu masuk Institut Informatika dan Bisnis Darmajaya;
b. Di depan gedung F Institut Informatika dan Bisnis Darmajaya;
c. Di depan masjid Institut Informatika dan Bisnis Darmajaya;
d. Di lapangan futsal Institut Informatika dan Bisnis Darmajaya.



Gambar 2.1 Denah Institut Informatika dan Bisnis Darmajaya

2.2 Penelitian Terkait

Berikut ini adalah tabel penelitian terkait yang relevan dengan topik penelitian tentang deteksi ketersediaan lahan parkir kosong:

Tabel 2.1 Penelitian Terkait

No	Judul	Penulis	Tujuan	Metode/Algoritma	Kesimpulan	Akurasi
1	Aplikasi Penghitung Kapasitas Ruang Parkir pada Lahan Parkir Kosong Menggunakan *Library* OpenCV pada Bahasa Pemrograman Python	(Devi Ayu Lestari et al., 2023)	Menghitung ketersediaan area parkir pada suatu lahan parkir terbuka	*Image Processing*	Tahap uji coba dilakukan dengan metode *User Acceptance Testing* (UAT) menggunakan sampel berupa video dengan 10 responden, aplikasi ini dapat berfungsi sesuai dengan tujuannya dan ditemukan tidak ada masalah dalam perhitungan pada aplikasi serta aplikasi dapat berjalan dengan baik	92.6%

2	Prototipe Sistem Deteksi Ketersediaan Lahan Parkir Menggunakan Metode Algoritma *Canny Edge*	(Ferry Pradana Putra & Indah Susilawati, 2021)	Merancang sistem yang dapat mendeteksi ketersediaan lahan parkir secara *realtime* agar para pengendara tidak kebingungan mencari lahan parkir yang kosong untuk kendaraan mereka	*Canny Edge*	Berdasarkan pengujian yang dilakukan dengan posisi kamera dengan ketinggian 40cm dengan menggunakan *Threshold* 510 memiliki akurasi yang bagus dalam mendeteksi citra untuk lahan parkir.	77%
3	Rancang Bangun Sistem Informasi Lahan Parkir Kendaraan Roda Empat di Unikom Berbasis *Image*	(Leonard Satrio Tegar & Jana Utama, 2016)	Merancang sistem informasi lahan parkir secara otomatis dengan menggunakan pengolahan citra yang dapat	*Thresholding* dan *Background Subtraction*	Dari hasil pengujian, sistem mampu menghitung jumlah dan menentukan lokasinya dengan keberhasilan 100% dari setiap kondisi yang telah di uji coba	100%

	Processing		berperan sebagai pendeteksi lahan parkir yang masih tersedia dan yang sudah terisi pada gedung tersebut secara *realtime*			
4	*Parking Lots Space Detection with Floyd Warshall Algorithm at Kartini Shopping Mall Bandar Lampung*	(Yuni Puspita Sari et al., 2021)	Merancang aplikasi untuk menampilkan informasi mengenai tempat parkir dalam menghitung jalur terpendek	*Floyd Warshall*	Aplikasi ini menampilkan informasi lahan parkir yang kosong dengan menerapkan algoritma *Floyd Warshall* sebagai pencarian yang dapat digunakan dalam menghitung lintasan terpendek, dan mampu membandingkan semua jalur yang mungkin dalam graf untuk setiap sisi dari semua simpul yang ada	-
5	Deteksi	(Anwar	Merancang sebuah	*Image Processing*	Tahap uji coba dilakukan	-

	Ketersediaan Lahan Parkir Dengan Menggunakan OpenCV	Muzaki et al., 2024)	sistem deteksi ketersediaan lahan parkir dengan menggunakan *library* bahasa pemrograman Python yang diharapkan bisa membantu pengendara yang ingin memarkirkan kendaraannya pada tempat parkir sesuai dengan ketersediaan parkir		pengujian Black Box dengan data uji tampilan pengguna, perubahan citra, tampilan deteksi tersedia, dan tampilan deteksi penuh yang sesuai dengan yang diharapkan	

Berdasarkan tabel 2.1 beberapa penelitian terdahulu telah menjalani pendekatan serupa dalam metode dan tujuan penelitiannya. Hasil penelitian terdahulu menunjukkan bahwa penggunaan metode *Image Processing* dan *Adaptive Thresholding* memberikan tingkat akurasi yang baik dalam mendeteksi slot parkir. Penelitian ini mengadopsi *Adaptive Thresholding* dalam konteks simulasi deteksi ketersediaan lahan parkir mobil di Institut Informatika dan Bisnis Darmajaya. Oleh karena itu penelitian ini diharapkan mampu memberikan gambaran dan inovasi teknologi bagi institusi.

2.3 Simulasi

Simulasi adalah replika dari proses sebuah sistem yang dibuat dengan tujuan untuk mengamati karakteristik sistem nyatanya. Terkadang simulasi dibuat untuk sesuatu yang belum ada sistem nyatanya, sehingga pembuatan simulasi dalam hal ini untuk menguji sistem rancangan.

Shannon (1975), simulasi adalah proses perencanaan model dari sistem nyata yang dilanjutkan dengan pelaksanaan eksperimen terhadap model untuk mempelajari perilaku sistem atau evaluasi strategi [3].

2.4 *Aerial-Image Processing*

Aerial-Image Processing merujuk pada proses pengolahan citra yang diambil dari udara, khususnya dari *platform* seperti pesawat terbang, *drone*, atau satelit.

Pengolahan citra atau *image processing* adalah suatu sistem dimana proses dilakukan dengan masukan (*input*) berupa citra (*image*) dan hasilnya (*output*) juga berupa citra (*image*). Pada awalnya pengolahan citra ini dilakukan untuk memperbaiki kualitas citra, namun dengan berkembangnya dunia komputasi yang ditandai dengan semakin meningkatnya kapasitas dan kecepatan proses komputer, serta munculnya ilmu-ilmu komputer yang memungkinkan manusia dapat mengambil informasi dari suatu citra maka *image processing* tidak dapat dilepaskan dengan bidang *computer vision* [4].

2.5 Eksperimen

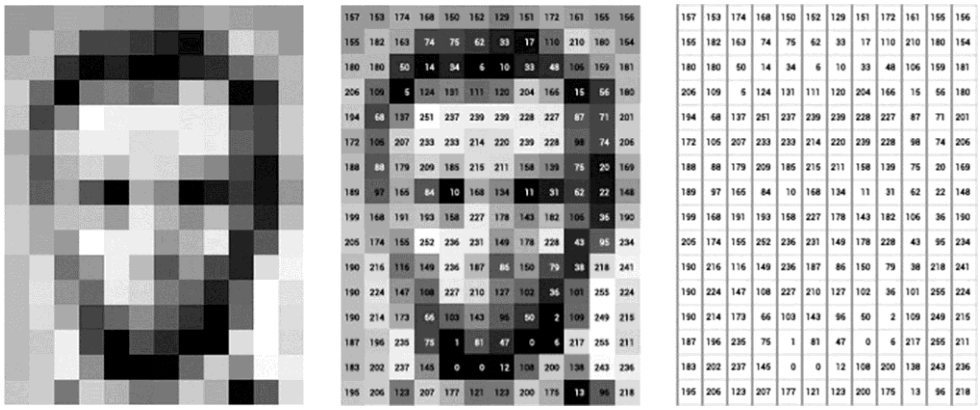
Eksperimen dilakukan untuk mengetahui kelebihan dan kekurangan suatu metode atau teknik, serta untuk menemukan cara-cara baru yang lebih baik dalam menyelesaikan masalah. Oleh karena itu, eksperimen sangat penting dilakukan agar hasil yang diperoleh bermutu dan dapat dipertanggungjawabkan [5].

2.6 OpenCV

Open Computer Vision (OpenCV) merupakan pustaka sumber terbuka yang berfokus untuk pemrosesan gambar. Tujuannya adalah agar komputer dapat memproses informasi visual seperti halnya manusia [6].

2.7 *Grayscale*

Grayscale adala representasi citra di mana setiap piksel hanya memiliki satu tingkat keabuan. Pada citra 8 bit, setiap piksel dipresentasikan oleh nilai yang berada dalam rentang 0 sampai 255. Semakin kecil nilainya maka tingkat kecerahannya akan semakin rendah (gelap), sementara semakin besar nilainya maka tingkat kecerahannya akan semakin tinggi (terang) [7].



Gambar 2.2 Konsep *Grayscale*

Berikut merupakan persamaan (1) dan (2) yang digunakan untuk menghitung nilai intensitas *Grayscale* :

$$Gray = 0.299^{*}R + 0.587^{*}G + 0.114^{*}B \quad (1)$$

atau

$$Gray = \frac{(R + G + B)}{3} \quad (2)$$

Keterangan:

$Gray$	:	nilai intensitas kecerahan piksel *Grayscale*
R	:	komponen merah (*red*) dari piksel berwarna
G	:	komponen hijau (*green*) dari piksel berwarna
B	:	komponen biru (*blue*) dari piksel berwarna

Nilai koefisien (0.299, 0.587, 0.114) dalam rumus ini adalah bobot relatif untuk setiap komponen warna. Rumus ini didasarkan pada kenyataan bahwa mata manusia lebih sensitif terhadap perubahan dalam komponen hijau, sehingga komponen hijau memiliki bobot yang lebih tinggi dalam menghitung intensitas *Grayscale.*

Berikut merupakan perhitungan manual untuk mengubah citra bewarna (RGB) ke dalam citra *Grayscale*, akan diambil contoh citra dengan satu piksel tertentu, dimana:

Diketahui : R = 150
G = 75
B = 200

Ditanya : Berapa nilai intensitas *Grayscale* jika dihitung dengan menggunakan persamaan (1)?
Berapa nilai intensitas *Grayscale* jika dihitung dengan menggunakan persamaan (2)?

Jawab : Nilai intensitas *Grayscale* jika dihitung dengan menggunakan persamaan (1) adalah:

$$Gray = 0.299^{*}R + 0.587^{*}G + 0.114^{*}B$$
$$Gray = 0.299^{*}150 + 0.587^{*}75 + 0.114^{*}200$$
$$Gray = 44.85 + 44.025 + 22.8$$
$$Gray = 111.675$$
$$Gray \approx 112$$

∴ Untuk piksel dengan nilai warna R = 150, G = 75, dan B = 200 yang dihitung dengan persamaan (1) didapatkan bahwa nilai intensitas *Grayscale*

yang sesuai adalah 112.

atau

Jawab : Nilai intensitas *Grayscale* jika dihitung dengan menggunakan persamaan (2) adalah:

$$Gray = \frac{(R + G + B)}{3}$$

$$Gray = \frac{(150 + 75 + 200)}{3}$$

$$Gray = \frac{425}{3}$$

$$Gray = 141.67$$

$$Gray \approx 142$$

∴ Untuk piksel dengan nilai warna R = 150, G = 75, dan B = 200 yang dihitung dengan persamaan (1) didapatkan bahwa nilai intensitas *Grayscale* yang sesuai adalah 142.

Perbedaan dari persamaan (1) dan (2) adalah:

a. Persamaan (1)
   - Menggunakan bobot yang berbeda untuk setiap komponen warna;
   - Bobot tersebut didasarkan pada penelitian psikovisual dan memberikan lebih banyak kontribusi pada warna hijau karena mata manusia lebih sensitif terhadap komponen warna hijau.

b. Persamaan (2)
   - Menggunakan pendekatan rata-rata sederhana dengan bobot yang setara untuk setiap komponen warna, sehingga memberikan kontribusi yang setara dari ketiga komponen warna.

2.8 *Gaussian Blur*

Gaussian blur adalah metode yang digunakan untuk menghilangkan *noise* (derau) pada citra dengan cara memberikan efek *blur* yang halus dan merata tanpa mengubah struktur dasar objek [8].

Berikut merupakan persamaan (3) yang digunakan untuk menghitung *Gaussian Blur*:

$$G(x,y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (3)$$

Keterangan:

$G(x,y)$	:	elemen matriks *Gauss* pada posisi (x,y)
σ	:	*filter* radius atau standar deviasi
(x,y)	:	ukuran matriks *Gauss* yang jangkauannya dari $-x$ sampai $+x$ dengan titik tengah $x = 0$ dan $y = 0$

Di bawah ini merupakan contoh perhitungan manual pada metode *Gaussian Blur* menggunakan persamaan (3):

Diketahui:

- Citra awal (*Grayscale*):

10	20	30
40	50	60
70	80	90

- Menerapkan *Gaussian Blur* dengan kernel 3×3

$$Kernel = \frac{1}{16} \begin{matrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{matrix}$$

Langkah-langkah perhitungan *Gaussiam Blur*:

- Ambil area sekitar piksel

$$Area = \begin{matrix} 10 & 20 & 30 \\ 40 & 50 & 60 \\ 70 & 80 & 90 \end{matrix}$$

- Lakukan konvolusi antara kernel dan area tersebut

  $Gaussian\ Blur = \frac{1}{16}\ (1 \times 10 + 2 \times 20 + 1 \times 30 + 2 \times 40 + 4 \times 50 + 2 \times 60 + 1 \times 70 + 2 \times 80 + 1 \times 90)$

$$Gaussian\ Blur = \frac{1}{16}\ (10 + 40 + 30 + 80 + 200 + 120 + 70 + 160 + 90)$$

$$Gaussian\ Blur = \frac{1}{16}\ (900)$$

$$Gaussian\ Blur = 56.25$$

$$Gaussian\ Blur \approx 57$$

- Ganti nilai piksel di pusat area dengan nilai *Gaussian Blur*

10	20	30
40	57	60
70	80	90

- Langkah ini diulang untuk setiap piksel dalam citra untuk menghasilkan citra yang telah diberlakukan *Gaussian Blur*.

Gambar 2.3 merupakan contoh penerapan metode *Gaussian Blur* pada sebuah citra *Grayscale*:

Gambar 2.3 *Gaussian Blur*

2.9 *Adaptive Thresholding*

Adaptive Thresholding adalah metode untuk mengonversi citra ke dalam citra biner (hitam dan putih) dengan cara menyesuaikan ambang batas (*Threshold*) secara lokal berdasarkan blok atau wilayah tertentu pada citra yang digunakan untuk memisahkan bagian objek dari latar belakang dalam citra digital [9].

Cara kerja *Adaptive Thresholding* melibatkan beberapa langkah umum:

a. Sebelum menerapkan metode *Adaptive Thresholding*, langkah pertama yang perlu dilakukan adalah mengonversi citra bewarna menjadi citra *Grayscale* untuk memudahkan perhitungan intensitas piksel.
b. Citra dibagi menjadi area kecil yang disebut *window* atau kernel yang akan digunakan untuk menghitung nilai ambang lokal.
c. Untuk setiap window, nilai ambang dihitung.
d. Setelah nilai ambang lokal dihitung untuk setiap piksel, *Thresholding* ditetapkan untuk memisahkan piksel menjadi dua kelas, yaitu: objek dan latar belakang.
e. Setelah *Thresholding* diterapkan pada setiap *window*, hasilnya digabungkan untuk menghasilkan citra *Adaptive Thresholding* yang lengkap.

Kelebihan dari *Adaptive Thresholding*:

a. *Adaptive Thresholding* memungkinkan penyesuaian dengan kondisi pencahayaan dan latar belakang yang tidak merata pada citra.
b. *Adaptive Thresholding* dapat meningkatkan kualitas citra dengan mengurangi derau pada citra.

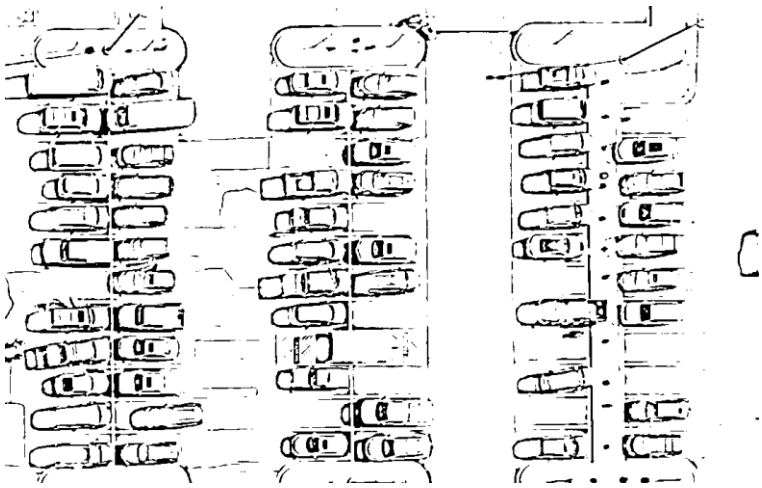
Terdapat lima jenis parameter yang dapat digunakan dalam fungsi "`cv.THRESHOLD`" di OpenCV untuk memberikan *Thresholding* yang berbeda, yaitu:

a. `cv2.THRESH_BINARY`

   Semua piksel dengan nilai di atas ambang batas akan diatur menjadi nilai maksimum (misalnya, 255), dan yang di bawah amban batas akan diatur menjadi nilai minimum (misalnya, 0).

   Rumus:

$$dst(x,y) = \begin{cases} maxval, & jika\ src(x,y) > thresh \\ 0, & lainnya \end{cases}$$



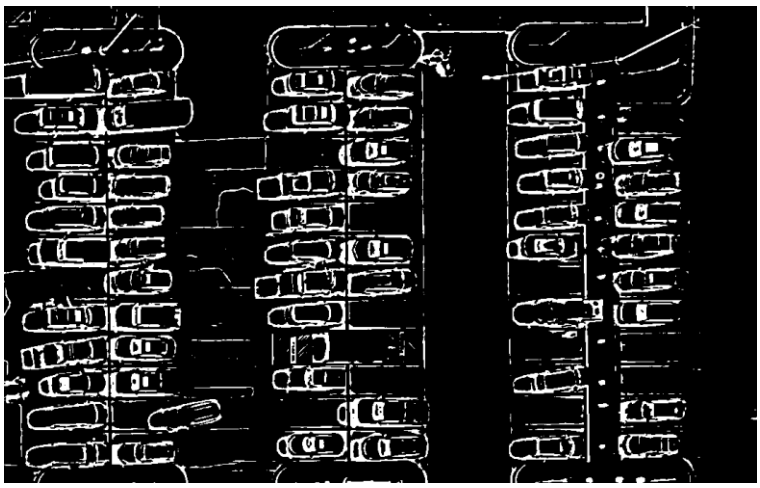
Gambar 2.4 `cv2.THRESH_BINARY`

b. `cv2.THRESH_BINARY_INV`

Kebalikan dari (`cv2.THRESH_BINARY_INV`). Semua piksel dengan nilai di atas ambang batas akan diatur menjadi nilai minimum, dan yang di bawah ambang batas akan diatur menjadi nilai maksimum.

Rumus:

$$dst(x,y) = \begin{cases} 0, \\ maxval, \end{cases} jika\ src(x,y) > thresh\ lainnya$$



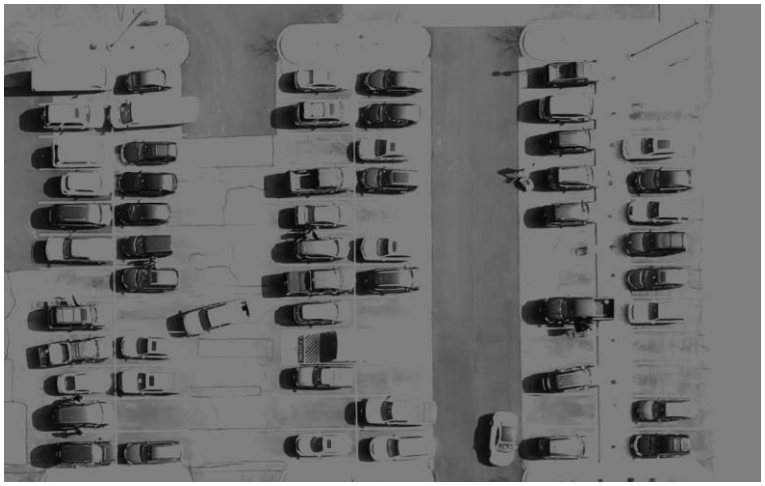
Gambar 2.5 `cv2.THRESH_BINARY_INV`

c. `cv2.THRESH_TRUNC`

Semua piksel dengan nilai di atas ambang batas akan diatur menjadi nilai ambang batas, dan yang di bawah ambang batas tetap tidak berubah.

Rumus:

$$dst(x,y) = \begin{cases} thresh, \\ src(x,y), \end{cases} jika\ src(x,y) > thresh\ lainnya$$



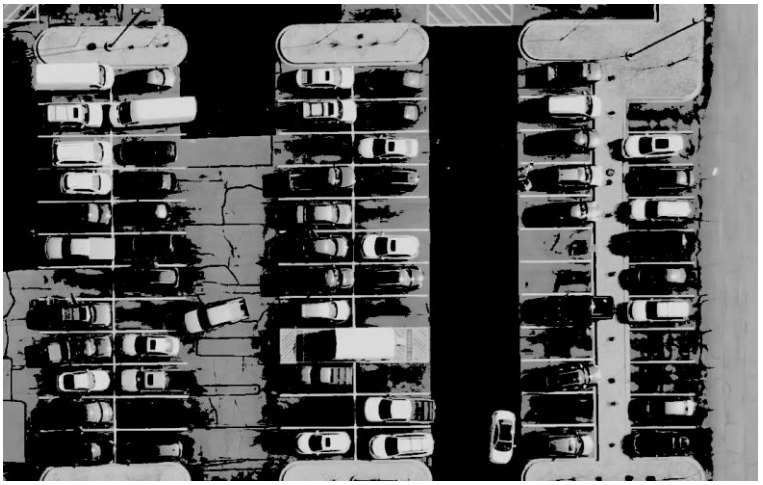
Gambar 2.6 `cv2.THRESH_TRUNC`

d. `cv2.THRESH_TOZERO`

Semua piksel dengan nilai di atas ambang batas tetap tidak berubah, dan yang di bawah ambang batas akan diatur menjadi nilai minimum.

Rumus:

$$dst(x,y) = \begin{cases} src(x,y), \\ 0, \end{cases} jika\ src(x,y) > thresh\ lainnya$$



Gambar 2.7 `cv2.THRESH_TOZERO`

e. `cv2.THRESH_TOZERO_INV`

Kebalikan dari (`cv2.THRESH_TOZERO`). Semua piksel dengan nilai di atas ambang batas akan diatur menjadi nilai minimum, dan yang di bawah ambang batas tetap tidak berubah.

Rumus:

$$dst(x,y) = \begin{cases} 0, \\ src(x,y), \end{cases} jika\ src(x,y) > thresh\ lainnya$$

Gambar 2.8 `cv2.THRESH_TOZERO_INV`

Selain paramenter yang dijelaskan sebelumnya, metode "`cv.adaptiveThreshold`" membutuhkan tiga paramenter *input*:

a. `adaptiveMethod`: menentukan bagaimana nilai ambang batas dihitung.
   - (`cv2.ADAPTIVE_THRESH_MEAN_C`): nilai ambang batas adalah rata-rata dari area tetangga dikurangi konstanta C;
   - (`cv2.ADAPTIVE_THRESH_GAUSSIAN_C`): nilai ambang batas adalah hasil jumlah tertimbang Gaussian dari nilai-nilai area tetangga dikurangi konstanta C.

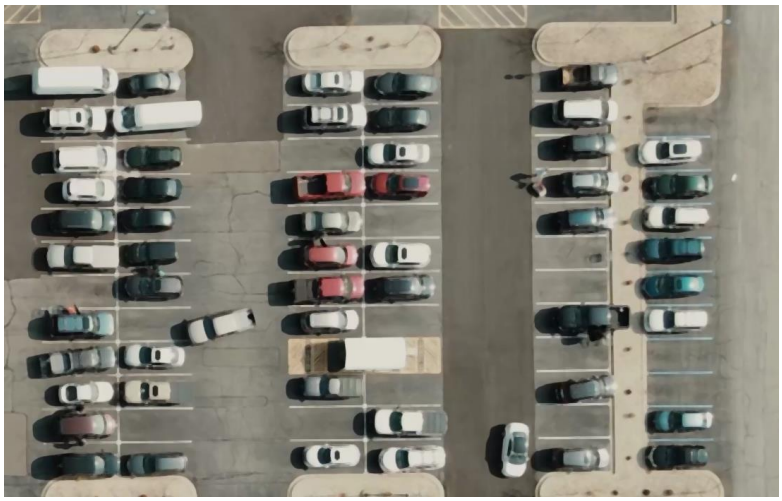
b. blockSize: menentukan ukuran area tetangga.

c. C: merupakan konstanta yang dikurangkan dari rata-rata atau jumlah tertimbang untuk mendapatkan nilai ambang batas.

2.10 *Median Blur*

Median Blur merupakan teknik pemrosesan citra yang digunakan untuk mengurangi jenis "*salt-and-pepper noise*" dengan cara mengambil nilai *median* dari piksel-piksel dalam wilayah tetangga [10].

Gambar 2.9 Citra asli sebelum dilakukan metode *Median Blur*



Gambar 2.10 Citra setelah dilakukan metode *Median Blur*

Di bawah ini merupakan contoh perhitungan manual dalam metode *Median Blur*:

Diketahui:

- Citra awal (*Grayscale*):

150	120	130	110	140
100	90	80	70	60
120	110	100	90	80
40	30	20	10	0
70	60	50	40	30

- Menerapkan *Median Blur* dengan kernel 3×3

Langkah-langkah perhitungan *Median Blur*:

- Pilih area pertama yang akan di-*blur* (berukuran 3×3)

150	120	130
100	90	80
120	110	100

- Urutkan nilai piksel dalam area secara *ascending*

  [80, 90, 100, 100, 110, 120, 120, 130, 150]
- Pilih nilai piksel yang berada di tengah setelah diurutkan (*median*)

  [80, 90, 100, 100, 110, 120, 120, 130, 150]
- Ganti nilai piksel di pusat area dengan nilai *median*

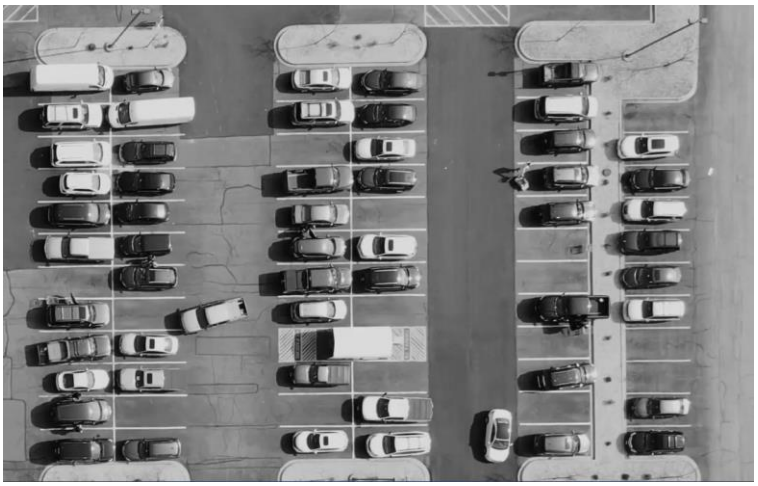
150	120	130
100	110	80
120	110	100

- Lakukan proses tersebut pada proses area dalam citra, dan hasil akhirnya adalah citra yang telah mengalami *Median Blur*.

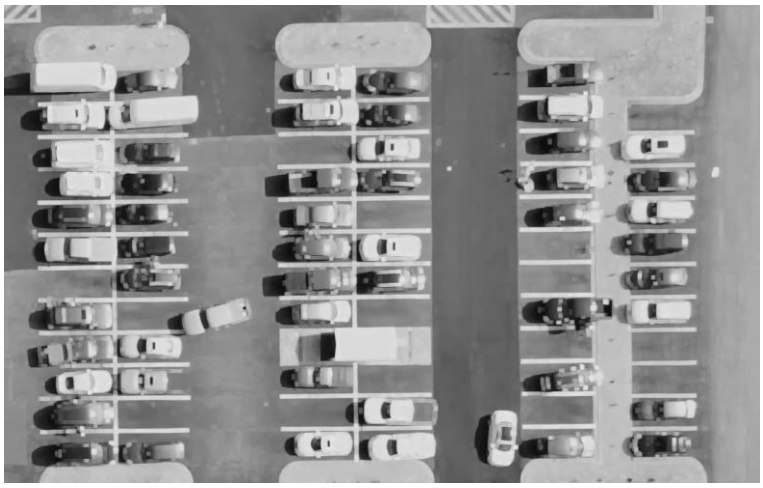
Dalam praktiknya, dapat menggunakan fungsi atau *library* seperti OpenCV untuk menerapkan *Median Blur*, sehingga tidak perlu dilakukan perhitungan manual.

2.11 *Dilation*

Dilasi (*Dilation*) *adalah* suatu operasi dalam pemrosesan citra yang dilakukan untuk memperluas atau memperbesar wilayah objek dalam citra dengan menambah lapisan di sekeliling objek sehingga citra hasil dilasi cenderung menebal [11].



Gambar 2.11 Citra asli sebelum dilakukan metode *Dilation*



Gambar 2.12 Citra setelah dilakukan metode *Dilation*

2.12 Python

Python adalah salah satu bahasa pemograman tingkat tinggi yang bersifat *interpreter*, *interactive*, *object-oriented*, dan *open source* yang dapat digunakan tanpa lisensi dan dapat dokembangkan semampu yang dapat dilakukan [12].

2.13 *Black Box*

Black Box juga disebut dengan pengujian prilaku yang berfokus pada persyaratan fungsional perangkat lunak. Artinya, teknik pengujian *Black Box* untuk membuat beberapa kumpulan kondisi masukan yang sepenuhnya akan melakukan semua kebutuhan fungsional untuk program [13].

2.14 PyCharm

PyCharm merupakan *Integrated Development Environment* (IDE) yang dirancang khusus untuk menulis kode dalam bahasa pemrograman Python. Fitur-fitur di dalamnya sudah cukup banyak, mulai dari *auto-complete code*, *coding assistance and analysis*, *syntax and error highlighting*, dan banyak lagi [14]. PyCharm dibuat oleh perusahaan asal Ceko, JetBrains, dan merilis versi beta pertamanya pada bulan Juli 2010. Kini PyCharm sudah memiliki 2 versi, yaitu PyCharm *Professional Edition* yang membutuhkan lisensi untuk menggunakannya dan PyCharm *Community Edition* yang sebagai *opern-source*-nya (https://www.jetbrains.com/pycharm/ diakses pada tanggal 18 Oktober 2023).