

## BAB II

### LANDASAN TEORI

#### 2.1 License Plate Recognition

*License Plate Recognition* (LPR) merupakan sebuah aplikasi untuk dapat mengenali posisi Tanda Nomor Kendaraan Bermotor (TNKB) serta dapat membaca TNKB itu sendiri. Aplikasi ini dibangun dengan bahasa pemrograman C++ dan didukung oleh *library* OpenCV untuk algoritma *image processing* dan *tesseract* OCR untuk pengenalan karakternya. Sudah ada beberapa penelitian yang membahas LPR, beberapa penelitian membahas mendapatkan posisi TNKB berdasarkan *aspect ratio* yang dilakukan oleh Sanjay Goel dengan judul *Vehicle License Plate Identification and Recognition* (2009) maka TNKB dapat dideteksi dengan *aspect ratio*, ada juga yang berdasarkan *histogram* gambar tersebut dengan menggunakan *vertical edge histogram* dan *horizontal edge histogram* yang dilakukan oleh Priyanto Hidayatullah dengan judul *License Plate Detection and Recognition for Indonesian Cars*.

#### 2.2 OpenCV

OpenCV adalah sebuah *computer vision library* yang bersifat *open source* yang dibuat untuk infrastruktur aplikasi *computer vision* secara umum. Lebih dari 2500 algoritma yang sudah di optimalisasi oleh *library* ini. Algoritma tersebut adalah *face detection*, *face recognition*, *identify objects*, *classify human actions*, *track camera movements*, *track moving objects*, dan lainnya.

OpenCV merupakan sebuah *open source* yang lebih dari 47 juta orang pengguna dan pengembang. Para pengguna tersebut mengembangkan dan menggunakan OpenCV dengan berbagai macam bahasa pemrograman seperti C++, C, Python, Java, dan MATLAB. OpenCV ini juga dapat digunakan ke berbagai *platform* seperti Windows, Linux, Mac OS, dan Android.

OpenCV mendukung CUDA, yaitu sebuah *library image processing* yang disediakan oleh NVIDIA dan tentunya menggunakan *core* dari *graphic card* NVIDIA untuk memprosesnya, dan untuk bagian tampilan aplikasinya, OpenCV didukung dengan OpenCL. Dan yang terakhir adalah OpenCV ini ditulis secara *native* dengan bahasa pemrograman C++.

## 2.3 Tesseract

Menurut jurnal “*An Overview of the Tesseract OCR Engine*”, 2007, *In Proceedings of the International Conference on Document Analysis and Recognition* yang ditulis oleh Ray Smith, tesseract adalah sebuah *Optical Character Recognition (OCR) engine*. Tesseract ini dapat dijalankan secara *multiplatform*. Ada tiga buah operasi system yang didukung yaitu Windows, Linux, dan Mac OS X. Tesseract ini ditulis dengan bahasa pemrograman C dan C++.

Tesseract akan melakukan beberapa tahapan sebelum dapat mengenali karakter. Ray Smith menjelaskan beberapa langkah bagaimana *tesseract* bekerja, tahapannya adalah *line and word finding*, *word recognition*, dan *static character classifier*.

### **2.3.1 Line and Word Finding**

Tesseract akan menggunakan *Connected Component Labelling* (CCL) untuk mendapatkan segmentasi yang disebut sebagai *binary large object* atau biasa disebut sebagai *blob* pada sebuah gambar. Hasil dari CCL adalah *blob*. *Blob* tersebut akan terorganisir menjadi *text lines*.

### **2.3.2 Word Recognition**

Tesseract akan melakukan *word recognition* dengan memanfaatkan *text lines* untuk mendapatkan kalimat dan kata-kata, dengan memanfaatkan metode *chopping joined characters*, maka akan menghasilkan segmentasi untuk setiap karakter yang ada di kalimat tersebut.

### **2.3.3 Static Character Classifier and Adaptive Classifier**

Pada tahap ini, *tesseract* menggunakan metode *polygonal approximation* untuk mengenali karakter. Tetapi teknik ini tidak berjalan dengan baik saat mengenali karakter yang bentuknya rusak, atau terputus. Oleh karena itu *adaptive classifier* adalah cara yang tepat mengenali karakter yang terputus atau rusak, perbedaan mencolok dari *adaptive* dan *static character* terletak pada *training data*, *adaptive classifier* menggunakan *isotropic baseline/x-height normalization*, metode ini memudahkan *tesseract* mengenal huruf besar atau kecil serta dapat lebih tahan terhadap *noise*.

## 2.4 Qt Creator

Qt Creator adalah sebuah *Integrated Development Environment* (IDE) yang dapat berjalan di berbagai *platform* seperti di Windows, Linux, dan Mac OS. Qt Creator ini menggunakan *compiler* C++, baik GNU *Compiler Collection*, MinGW atau MSVC sebagai *default compiler*. Qt Creator memiliki *framework* sendiri untuk memudahkan pengguna dalam mengembangkan aplikasinya.

Ada bagian dari Qt Creator untuk membuat desain *Graphical User Interfaces* (GUI) yaitu adalah Qt Designer, IDE ini memudahkan pengguna untuk melakukan kostumisasi terhadap tampilan aplikasi yang ingin dibangun.

## 2.5 Grayscale Image

*Grayscale image* merupakan sebuah gambar yang hanya memiliki satu *value* di setiap *pixel*, berbeda dengan *blue green red* (BGR) *image* yang memiliki tiga buah *value*, yaitu *red*, *blue*, dan *green*. Untuk mendapatkan nilai *value* di setiap *pixel* maka diterapkan rumus menghitung rata-rata di ketiga *value* tersebut.



Gambar 2.1 Gambar *BGR Image* dan *Grayscale Image*

Gambar 2.1 merupakan perubahan dari BGR *image* ke *grayscale image*, terlihat pada BGR *image* yang memiliki banyak jenis kombinasi warna sedangkan *grayscale image* yang memiliki satu jenis warna dengan level intensitas yang berbeda.

## 2.6 Binary Image

*Binary image* merupakan sebuah gambar yang hanya memiliki dua nilai pada setiap *pixel*, yaitu putih atau hitam. *Binary image* ini bisa didapatkan dengan melakukan proses *thresholding*, yaitu proses menjadikan *grayscale image* menjadi *binary image*, dengan menghitung perbandingan *value* setiap *pixel* pada *grayscale image* menjadikannya dengan kondisi tertentu bernilai putih atau hitam. Ada beberapa cara dalam melakukan *thresholding* yaitu, *manual thresholding* dan *adaptive thresholding*.



Gambar 2.2 *Binary Image*

Gambar 2.2 merupakan hasil dari *binary image*, terlihat hanya terdapat dua jenis warna, yaitu hitam dan putih. Warna putih memiliki nilai *pixel* sebesar 1, sedangkan hitam adalah 0. *Binary image* ini memiliki *single channel*, yang hanya bisa diberikan nilai antara 1 dan 0.

### 2.6.1 Manual Thresholding

Pada buku yang berjudul “*Learning OpenCV*”, 2013. Menjelaskan jika *Manual Thresholding* ini dilakukan dengan melakukan pengecekan setiap *value*. Rumus untuk mendapatkan nilai *binary* ditampilkan pada Gambar 2.3, jika *value* pada *pixel* tersebut lebih dari nilai *thresholding* yang ditetapkan maka akan bernilai *maxVal* yang artinya adalah hitam, dan jika tidak maka akan bernilai 0 yaitu putih.

$$\text{dst}(x, y) = \begin{cases} \text{maxVal} & \text{if } \text{src}(x, y) > \text{thresh} \\ 0 & \text{otherwise} \end{cases}$$

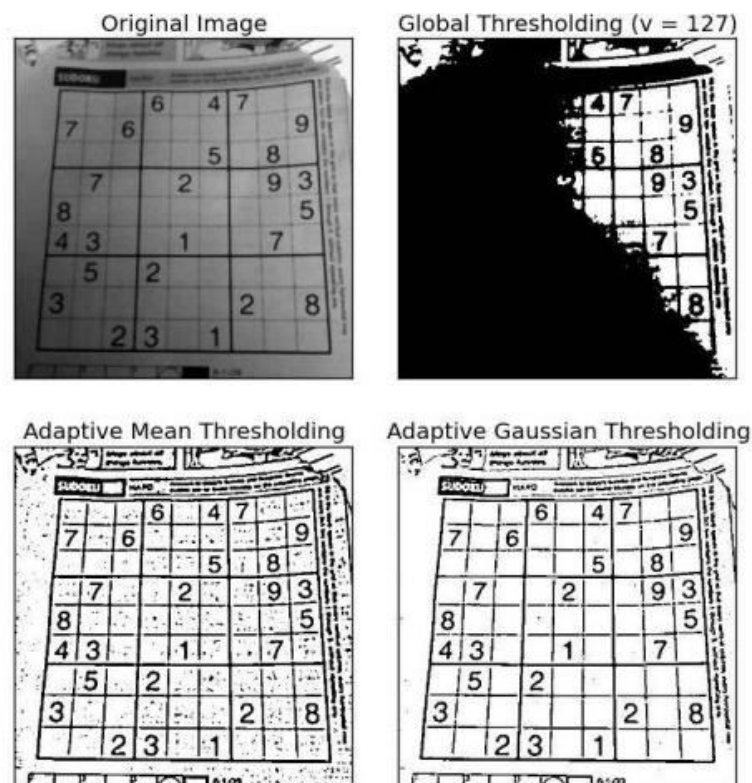
Gambar 2.3 Rumus Perhitungan *Manual Thresholding*

Pada Gambar 2.3, *x* dan *y* menyatakan koordinat pada sebuah gambar sedangkan *src* menyatakan objek target gambar yang akan dilakukan pengecekan terhadap jumlah nilai intensitas *level grayscale* dan *dst* adalah tujuan menuliskan hasil dari pengecekan terhadap nilai intensitas *level grayscale*. Oleh karena itu, pada *src(x,y)* akan dilakukan pengecekan terhadap nilai intensitas *level grayscale*, jika nilai intensitas *level grayscale* diatas nilai *thresh* yang telah ditetapkan maka hasil *dst(x,y)* bernilai *maxVal*, *maxVal* pada *binary image* bernilai 1, yaitu warna putih. Jika nilai intensitas *level grayscale*

dibawah nilai *thresh*, maka hasil  $dst(x,y)$  bernilai 0, yaitu warna hitam. Hal ini dilakukan untuk setiap koordinat gambar.

### 2.6.2 Adaptive Thresholding

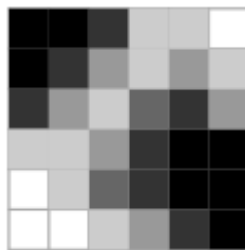
Pada buku yang berjudul “*Learning OpenCV*”, 2013. *Adaptive thresholding* ini dilakukan dengan melakukan perhitungan rata-rata minimal  $3 \times 3$  *pixel* di ambil dari sekeliling *pixel* itu sendiri, hal ini bertujuan untuk mendapatkan hasil *binary image* sebaik mungkin, karena jika menggunakan *manual thresholding* hasil gambar yang memiliki tingkat cahaya yang kurang akan tidak optimal hasilnya. Gambar 2.4 merupakan perbedaan hasil dari *manual thresholding* dan *adaptive thresholding*



Gambar 2.4 Perbandingan Hasil *Global Thresholding* dan *Adaptive Thresholding*  
 Sumber : ([http://docs.opencv.org/trunk/d7/d4d/tutorial\\_py\\_thresholding.html](http://docs.opencv.org/trunk/d7/d4d/tutorial_py_thresholding.html))

### 2.6.3 Otsu Thresholding

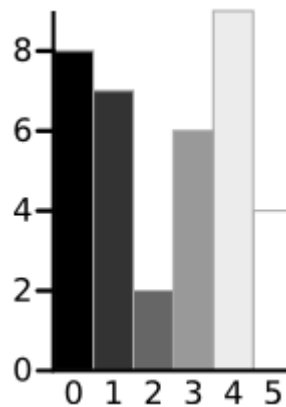
Teknik *otsu thresholding* ini ditemukan oleh peneliti bernama Nobuyuki Otsu dalam jurnal “A Threshold Selection Method from Gray-Level Histograms”, 1979, *IEE Transactions on Systems*. nilai *thresholding* yang didapatkan dengan cara melakukan perhitungan terhadap *histogram* pada gambar *grayscale*. *Histogram* menunjukkan nilai intensitas dari setiap *pixel* dalam 1 dimensi, sumbu x menyatakan level intensitas dan sumbu y menyatakan jumlah *pixel* yang memiliki nilai level intensitas tersebut. Dengan demikian teknik ini bertujuan mendapatkan nilai *thresholding* yang optimal, untuk mendapatkan nilai *thresholding* secara otomatis, ada beberapa tahapan yang harus dilakukan. Sebagai contoh ada sebuah *histogram* yang terdiri dari 6 level *grayscale*.



**Gambar 2.5 Grayscale Image dengan 36 pixels**

**Sumber :** (<http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html>)





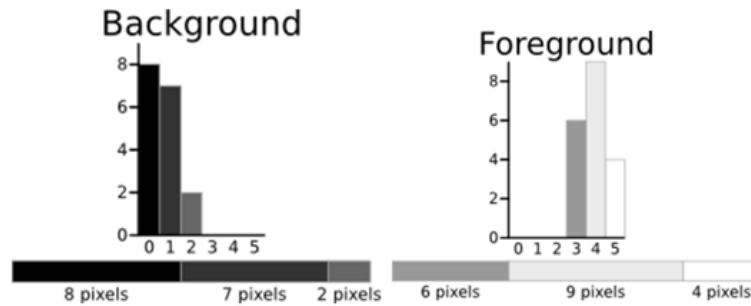
*Gambar 2.6 Histogram*

Sumber : (<http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html>)

Gambar 2.5 merupakan sebuah gambar yang terdiri dari 36 piksel, dan Gambar 2.6 merupakan tampilan *histogram* dari Gambar 2.5 yang merupakan gambar yang terdiri dari 6 level *grayscale*, dan dengan jumlah 36 piksel.

Hal yang dilakukan dalam teknik *otsu* ini adalah menghitung *weight*, *mean*, dan *variance* setiap level *grayscale*, jika level maksimal *grayscale* adalah 255, maka perhitungan dilakukan 255 kali, dalam artian perhitungan ini dilakukan secara iterasi sebanyak level *grayscale* yang ada. Untuk *weight* ditulis dalam  $W$ , *mean* ditulis dalam  $\mu$  dan *variance* ditulis dalam  $\sigma$ . Langkah-langkah yang akan dilakukan sebagai berikut.

1. Hitung nilai *weight*, sebagai contoh, nilai *threshold* yang diambil adalah 3, maka *histogram* akan dibagi menjadi 2 bagian, yang satu disebut sebagai *background*, dan satunya disebut sebagai *foreground*, *background* dengan simbol  $b$  dan *foreground* dengan simbol  $f$ .



**Gambar 2.7 Histogram Background dan Foreground**

Sumber : (<http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html>)

Gambar 2.7 merupakan pembagian dari nilai *threshold* dengan nilai 3, dengan jumlah data sebanyak 17 piksel. Untuk perhitungan *weight background* dan *weight foreground* adalah

$$\text{Weight } W_b = \frac{8 + 7 + 2}{36} = 0.4722$$

$$\text{Weight } W_f = \frac{6 + 9 + 4}{36} = 0.5278$$

**Gambar 2.8 Perhitungan untuk Weight**

Sumber : (<http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html>)

Pada Gambar 2.8 terdapat perhitungan untuk menentukan nilai *weight* dengan melakukan perhitungan distribusi data *background* atau *foreground* yang dibagi dengan jumlah piksel yang berada pada Gambar 2.7. Perhitungan di atas adalah pembagian jumlah piksel di *foreground* dan *background* dengan jumlah piksel pada Gambar 2.5.

2. Tahap berikutnya adalah menghitung *mean* pada *background* dan *foreground* dengan perhitungan seperti ini

$$\text{Mean } \mu_b = \frac{(0 \times 8) + (1 \times 7) + (2 \times 2)}{17} = 0.6471$$

$$\text{Mean } \mu_f = \frac{(3 \times 6) + (4 \times 9) + (5 \times 4)}{19} = 3.8947$$

**Gambar 2.9 Perhitungan untuk Mean**

Sumber : (<http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html>)

Perhitungannya yaitu dengan menghitung level *grayscale* dikali dengan jumlah *pixel* yang dimiliki nilai level *grayscale* tersebut. Sebagai contoh pada Gambar 2.9, (0 x 8) adalah 0 sebagai nilai level *grayscale*, dan nilai 8 adalah jumlah *pixel* yang memiliki nilai level *grayscale* 0. Semua dijumlahkan dan dibagi dengan jumlah total piksel yang dimiliki oleh *background* dan *foreground*.

- Selanjutnya adalah tahap menghitung *variance*, perhitungan *variance* akan dilakukan dengan melakukan perhitungan terhadap *mean* dan jumlah piksel.

$$\begin{aligned} \text{Variance } \sigma_b^2 &= \frac{((0 - 0.6471)^2 \times 8) + ((1 - 0.6471)^2 \times 7) + ((2 - 0.6471)^2 \times 2)}{17} \\ &= \frac{(0.4187 \times 8) + (0.1246 \times 7) + (1.8304 \times 2)}{17} \\ &= 0.4637 \end{aligned}$$

$$\begin{aligned} \text{Variance } \sigma_f^2 &= \frac{((3 - 3.8947)^2 \times 6) + ((4 - 3.8947)^2 \times 9) + ((5 - 3.8947)^2 \times 4)}{19} \\ &= \frac{(4.8033 \times 6) + (0.0997 \times 9) + (4.8864 \times 4)}{19} \\ &= 0.5152 \end{aligned}$$

**Gambar 2.10 Perhitungan untuk Variance**

Sumber : (<http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html>)

Sebagai contoh  $((0 - 0.6471)^2 \times 8)$  pada Gambar 2.10, nilai 0 adalah level dari *grayscale*, 0.6471 adalah nilai dari *mean*, dan nilai 6 adalah jumlah piksel yang memiliki level *grayscale* sebanyak 0. Semua akan dijumlah

setiap level *grayscale* pada *background*. selanjutnya akan dibagi dengan nilai 17 yaitu nilai total *pixel* pada *background*. Untuk perhitungan *foreground* juga sama dengan perhitungan pada *background*.

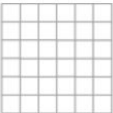
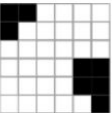
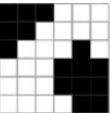
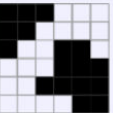
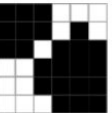
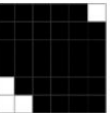
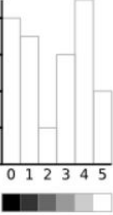
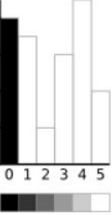
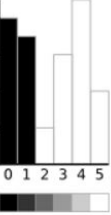
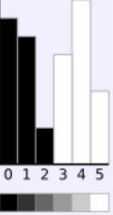
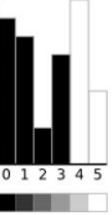
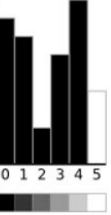
- Selanjutnya adalah menghitung *within class variance*, yaitu dengan cara melakukan perhitungan terhadap kedua *variance* yang akan dikali dengan kedua *weight* tersebut.

$$\text{Within Class Variance } \sigma_W^2 = W_b \sigma_b^2 + W_f \sigma_f^2 = 0.4722 * 0.4637 + 0.5278 * 0.5152 = 0.4909$$

**Gambar 2.11** Perhitungan untuk *Within Class Variance*

Sumber : (<http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html>)

- Selanjutnya dilakukan perhitungan untuk setiap level *grayscale*, jika pada contoh 6 level *grayscale*, maka dilakukan perhitungan langkah 1 sampai 4 hingga 6 kali.
- Tahap terakhir adalah memilih hasil *within class variance* yang terendah

| Threshold             | T=0   | T=1   | T=2   | T=3   | T=4   | T=5   |
|-----------------------|---|---|---|---|---|---|
|                       |  |  |  |  |  |  |
|                       |  |  |  |  |  |  |
| Weight, Background    | $W_b = 0$   | $W_b = 0.222$   | $W_b = 0.4167$  | $W_b = 0.4722$  | $W_b = 0.6389$  | $W_b = 0.8889$  |
| Mean, Background      | $M_b = 0$   | $M_b = 0$   | $M_b = 0.4667$  | $M_b = 0.6471$  | $M_b = 1.2609$  | $M_b = 2.0313$  |
| Variance, Background  | $\sigma_b^2 = 0$  | $\sigma_b^2 = 0$  | $\sigma_b^2 = 0.2489$   | $\sigma_b^2 = 0.4637$   | $\sigma_b^2 = 1.4102$   | $\sigma_b^2 = 2.5303$   |
| Weight, Foreground    | $W_f = 1$   | $W_f = 0.7778$  | $W_f = 0.5833$  | $W_f = 0.5278$  | $W_f = 0.3611$  | $W_f = 0.1111$  |
| Mean, Foreground      | $M_f = 2.3611$  | $M_f = 3.0357$  | $M_f = 3.7143$  | $M_f = 3.8947$  | $M_f = 4.3077$  | $M_f = 5.000$   |
| Variance, Foreground  | $\sigma_f^2 = 3.1196$   | $\sigma_f^2 = 1.9639$   | $\sigma_f^2 = 0.7755$   | $\sigma_f^2 = 0.5152$   | $\sigma_f^2 = 0.2130$   | $\sigma_f^2 = 0$  |
| Within Class Variance | $\sigma_W^2 = 3.1196$   | $\sigma_W^2 = 1.5268$   | $\sigma_W^2 = 0.5561$   | $\sigma_W^2 = 0.4909$   | $\sigma_W^2 = 0.9779$   | $\sigma_W^2 = 2.2491$   |

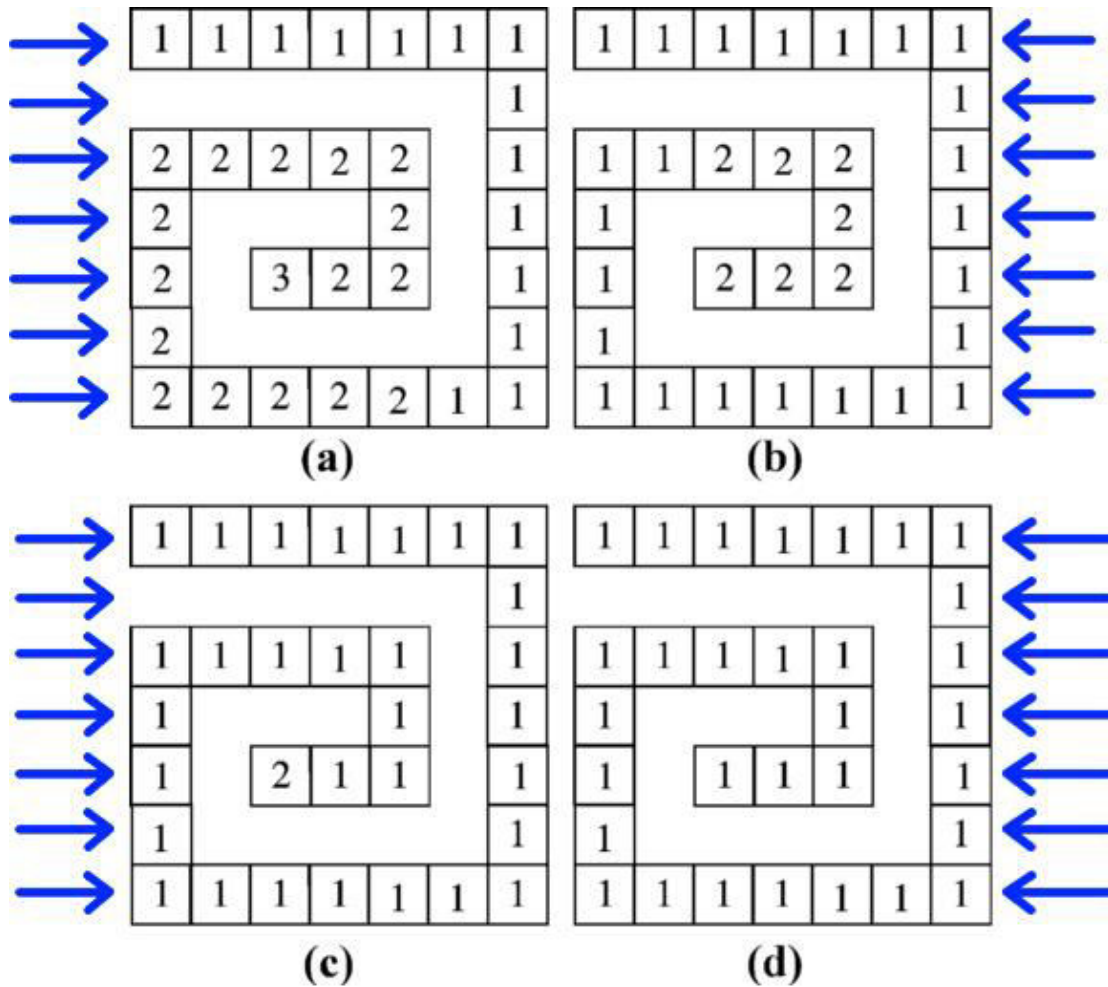
**Gambar 2.12** Hasil perhitungan 6 Level *Grayscale*

Sumber : (<http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html>)

Sebagai contoh Gambar 2.12, bahwa *threshold* dengan nilai 3 menghasilkan *within class variance* dengan nilai terendah, dengan demikian nilai *threshold* yang paling optimal adalah 3.

## **2.7 Connected Component Labeling**

*Connected Component Labeling* (CCL) merupakan sebuah algoritma untuk melakukan segmentasi dengan memanfaatkan *binary image*, hasil *segmentasi* dari CCL ini bisa di sebut *binary large object* atau *blob*, *blob* tersebut memiliki informasi yang dapat dihitung jumlahnya, ukurannya, dan letak posisinya.



Gambar 2.13 Merupakan Teknik Konvensional *Connected Component Labelling*. Pada Gambar (a) Merupakan Hasil dari *Forward Scanning*, Gambar (b) Merupakan Hasil dari *Backward Scanning*, Gambar (c) Merupakan Hasil dari *Forward Scanning* dan Gambar (d) Merupakan Hasil dari *Backward Scanning* yang terakhir

Sumber : (*Learning OpenCV*, 2013)

Cara kerja CCL ini adalah melakukan *scanning* perbaris, yang dilakukan pemindaian adalah piksel berwarna putih atau bernilai 1, pemindaian yang pertama disebut dengan *forward scanning*, yaitu melakukan pengecekan setiap piksel dimulai dari kiri atas ke arah kanan bawah. Dilakukan *forward scanning* dengan secara perbaris, baris pertama dilakukan *scanning* secara *horizontal* kearah kanan. Pada Gambar 2.12 (a) merupakan hasil dari *forward scanning*,

setiap piksel yang ditemukan akan diberi penamaan *label* angka, yaitu 1 untuk piksel yang pertama kali ketemu. Selanjutnya pemindaian akan bergeser satu piksel ke arah kanan, dan melakukan cek terhadap 3x3 piksel sekelilingnya. Jika ada *label* nama pada wilayah 3x3 piksel tersebut, piksel yang dipindai akan mendapatkan penamaan *label* yang paling rendah. Sebagai contoh jika pada area 3x3 piksel tersebut ada 3 *label*, yaitu *label* 1, *label* 3, dan *label* 4. Maka piksel yang dipindai tersebut akan mendapatkan nama *label* 1. Dan jika pada wilayah 3x3 *pixel* tersebut tidak ada *label* apapun. Maka penamaan *label* akan bertambah, jika sebelumnya *label* angka terakhir yang diberikan saat proses pemindaian adalah 1, maka piksel hasil pemindaian tersebut akan diberi nama *label* 2, karena wilayah 3x3 di sekitarnya tidak ada *label* apapun. Proses tersebut dilakukan secara iterasi sehingga mencapai titik kanan bawah.

Pemindaian kedua disebut sebagai *backward scanning* yang dilakukan pada Gambar 2.12 (b), tekniknya sama dengan *forward scanning*, yaitu penamaan terhadap *label*, bedanya *backward scanning* ini dimulai dari titik kanan bawah, dilakukan pemindaian secara *horizontal* ke arah kiri atas. Sama seperti sebelumnya, pemindaian ini dilakukan secara perbaris, akan tetapi arah dari pemindaian pun terbalik, yaitu dari sisi kanan ke sisi kiri.

CCL ini bekerja melakukan *forward scanning*, dan *backward scanning* secara berulang-ulang sehingga menemukan kondisi yang tidak ada perubahan penamaan *label*, saat mencapai kondisi tersebut, maka CCL ini sudah melakukan *segmentasi* terhadap objek-objek gambar tersebut. *Segmentasi label* tersebut disimpan pada

data yang disebut *blobs*, sehingga data *blobs* ini dapat dikelola untuk kepentingan langkah selanjutnya.

## 2.8 *Dilation* dan *Erode*

*Dilation* dalam bahasa Indonesia berarti pelebaran, *dilation* adalah proses konvolusi dengan menggunakan *external masker*, sebagai contoh ada gambar yang disebut sebagai A, dan *kernel* yang disebut sebagai B, *kernel* ini merupakan *external masker*. *Kernel* ini bisa dalam ukuran dan bentuk tertentu tergantung kebutuhan, dan memiliki titik pusat yang disebut *anchor point*. Cara kerjanya adalah *kernel* ini melakukan pemindaian terhadap gambar A, setiap piksel akan dilakukan pemindaian, lalu menghitung maksimal nilai untuk setiap piksel yang saling tumpang tindih oleh *kernel* B, dan kemudian mengganti *pixel* yang dibawah nilai *anchor point* dengan maksimal nilai piksel yang ada. Hasil dari proses *dilation* ini ditampilkan pada Gambar 2.14.



Gambar 2.14 *Dilation process*  
Sumber : (*Learning OpenCV*, 2013)





**Gambar 2.15 Erosion process**  
**Sumber : (Learning OpenCV, 2013)**

Sedangkan untuk *erode* atau *erosion* dalam bahasa Indonesia berarti erosi atau pengikisan, konsepnya sama, yang adalah berbeda adalah menghitung minimal nilai setiap piksel yang saling tumpang tindih oleh *kernel B*, dan kemudian mengganti piksel yang di bawah nilai *anchor point* dengan minimum nilai piksel yang ada. Secara umum *dilation* adalah proses untuk *expand region A*, sedangkan *erosion* adalah proses untuk *reduce region A*.

## 2.9 Kajian Literatur

Untuk mendapatkan informasi tanda nomor kendaraan bermotor dari sebuah gambar mobil dilakukan dua hal utama, yaitu deteksi dan pengenalan. Untuk tahap deteksi ada beberapa metode untuk mendapatkan lokasi dari TNKB ini.

Pada penelitian yang dilakukan Hidayatullah dalam judul “*License Plate Detection and Recognition for Indonesian Cars*” (2016), dalam tahap pendeteksian menggunakan pendekatan dengan cara memanfaatkan *histogram* dari gambar tersebut. Dilakukannya proses perhitungan pada *vertical* dan *horizontal edge detection* untuk melakukan validasi terhadap objek-objek yang tidak dibutuhkan. Dan pada penelitian yang dilakukan oleh Sanjay Goel dalam

judul “*Vehicle License Plate Identification and Recognition*” (2009) memanfaatkan aspek rasio dari TNKB untuk mendapatkan lokasi TNKB itu sendiri.

Kedua cara ini berhasil mendapatkan lokasi TNKB, akan tetapi tingkat akurasi yang didapatkan pada penelitian yang dilakukan Hidayatullah adalah 78.75% dan penelitian yang dilakukan oleh Sanjay Goel adalah 88%. Perbedaan yang cukup signifikan ini menjadi alasan utama untuk menerapkan metode yang akan digunakan dalam studi ini.

Metode yang akan digunakan pada penelitian ini berdasarkan penelitian yang dilakukan oleh Sanjay Goel, yaitu memanfaatkan aspek rasio dari TNKB. Akan tetapi perbedaan bentuk TNKB Indonesia dengan TNKB India yang dilakukan penelitian tersebut menjadi permasalahan untuk penerapan metode aspek rasio. Pada penelitian yang dilakukan Sanjay Goel, bentuk dari TNKB India memiliki warna latar belakang berwarna putih, sehingga melakukan proses segmentasi lebih mudah. Sedangkan TNKB Indonesia memiliki latar belakang hitam dan tidak sedikit warna latar belakang TNKB dengan warna mobil memiliki kesamaan, baik karena mobil berwarna hitam atau hal lainnya. Sehingga akan mengalami kesulitan saat melakukan proses segmentasi.

Untuk menyelesaikan permasalahan tersebut, metode aspek rasio ini akan mengalami beberapa perubahan sebelum dilakukan segmentasi. Dengan memanfaatkan ciri-ciri TNKB Indonesia yang memiliki tepi garis putih, tepi garis putih ini menjadi pembatas antara latar belakang TNKB dan objek lainnya. Akan

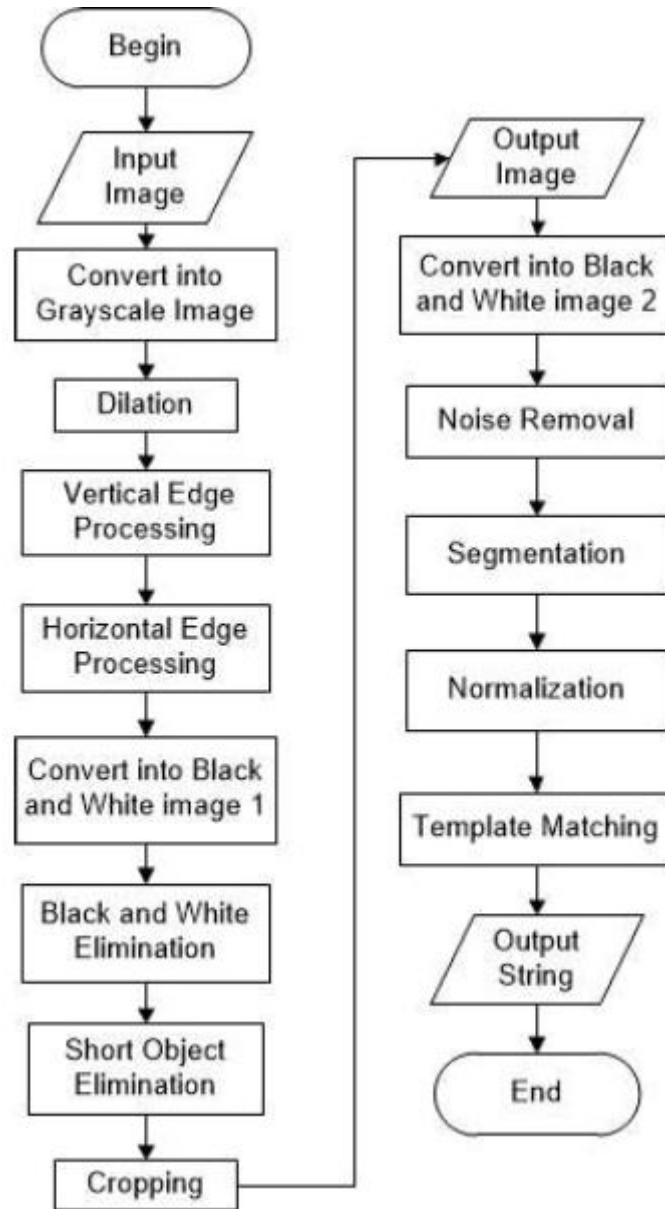
tetapi beberapa data memiliki tepi garis putih yang terputus, sehingga dibutuhkan metode lainnya untuk memperbaiki tepi garis yang terputus ini.

Untuk tahap pengenalan, pada penelitian yang dilakukan oleh Hidayatullah menggunakan tesseract dengan persentase keberhasilan sebesar 57.14%. Penelitian ini akan melakukan hal yang sama, akan tetapi akan diberikan penambahan metode untuk menyajikan gambar TNKB sebaik-baiknya agar tesseract dapat mengenali karakter TNKB dengan baik. Hal ini dikarenakan pada penelitian Hidayatullah tidak dilakukan proses *unwanted object removal* pada objek-objek yang berada dalam gambar TNKB, sehingga banyaknya karakter-karakter yang tidak dibutuhkan muncul dari hasil pengenalan dan membuat hasil pengenalan memiliki kualitas yang tidak baik. Untuk melakukan *unwanted object removal* adalah dengan memanfaatkan aspek rasio karakternya, hal ini dibahas pada journal yang ditulis oleh Samuel Mahatmaputra Tedjojuwono dengan judul “*Ornament Problem Suppression in Indonesian License Plate Recognition System*” (2017). Penelitian tersebut melakukan deteksi karakter-karakter pada TNKB dengan memanfaatkan aspek rasionya, sehingga dapat menghilangkan objek-objek lain yang tidak diperlukan.

### **2.9.1 License Plate Detection and Recognition for Indonesian Cars**

Penelitian sebelumnya yang memiliki judul *License Plate Detection and Recognition for Indonesia Cars* ini ditulis oleh Priyanto Hidayatullah, Ferry Feirizal, Hadi Permana, Qori Mauluddiah, dan Andhika Dwitama, penelitian ini membahas sebuah algoritma yang dipersiapkan untuk mengenal TNKB

Indonesia, karena menurut penulis bahwa setiap negara memiliki ciri-ciri TNKB sendiri, mulai dari warna, ukuran, bentuk huruf, dan format penulisan.



**Gambar 2.16 Diagram Blok**

**Sumber :** (*License Plate Detection and Recognition for Indonesian Cars, 2016*)

Gambar 2.16 merupakan diagram yang dikerjakan oleh penelitian tersebut, langkah awal yang dilakukan adalah mengubah *RGB Image* ke *Grayscale Image* dan dilakukannya *Vertical dan Horizontal Edge Processing*.



**Gambar 2.17** Hasil dari *Vertical Edge Processing*

**Sumber :** (*License Plate Detection and Recognition for Indonesian Cars, 2016*)

Gambar yang pertama pada Gambar 2.17 adalah *histogram* hasil dari *vertical edge*, di sana ada nilai yang akan divalidasi, yaitu jika nilai *histogram* kurang dari rata-rata nilai akan dihilangkan. Hasilnya mendapatkan tiga buah kandidat yaitu x, y, dan z. Penelitian yang dilakukan Hidayatullah ini mengasumsikan jika posisi TNKB tidak akan berada di tengah gambar maupun di posisi atas gambar, maka x dan y akan otomatis dihapus dan hasilnya berupa z, hasil dari eliminasi ini diperlihatkan pada Gambar 2.18.



**Gambar 2.18** Hasil dari *Vertical Edge Processing*

**Sumber :** (*License Plate Detection and Recognition for Indonesian Cars, 2016*)

Setelah melakukan *vertical edge processing* akan dilakukan kembali *horizontal edge processing*, metode yang dilakukannya sama dengan *vertical edge processing*, yaitu jika *value* dari hasil *histogram* di bawah rata-rata total *value histogram* maka akan dihapus, metode ini akan ditampilkan pada Gambar 2.19.



**Gambar 2.19** Hasil dari *Horizontal Edge Processing*

**Sumber :** (*License Plate Detection and Recognition for Indonesian Cars, 2016*)

Dengan cara melakukan eliminasi untuk beberapa objek yang diasumsi TNKB tidak pada di posisi dan memanfaatkan wilayah yang tersisa sebagai dasar dari melakukan *cropping* terhadap gambar, hasil dari *cropping* ini ditampilkan pada Gambar 2.20.

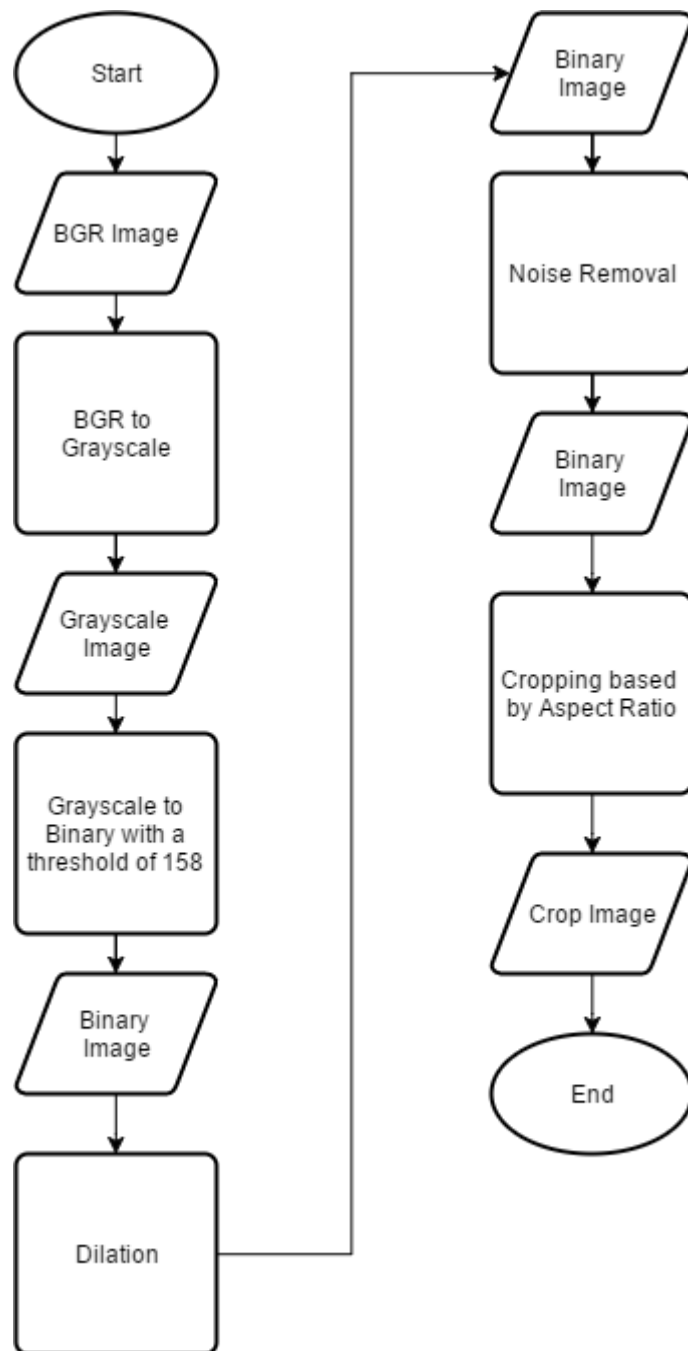


Gambar 2.20 Hasil dari proses *cropping*

Sumber : (License Plate Detection and Recognition for Indonesian Cars, 2016)

### **2.9.2 Vehicle License Plate Identification and Recognition**

Penelitian selanjutnya dilakukan oleh Sanjay Goel, Priyank Singh dan S. Batra et al ini melakukan pengenalan terhadap TNKB negara India, penelitian ini ditulis dengan bahasa pemograman C# .NET. Langkah-langkah yang akan dilakukan penelitian ini akan diperlihatkan pada Gambar 2.21.



**Gambar 2.21 Diagram blok dari penelitian**

**Sumber :** (*Vehicle License Identification and Recognition, 2009*)

Langkah awal yang dilakukan adalah langkah yang umum, yaitu melakukan perubahan terhadap RGB *image* pada Gambar 2.22 menjadi *grayscale image*, selanjutnya pada penelitian ini menggunakan *manual thresholding* untuk mendapatkan *binary image*, nilai optimum *thresholding*

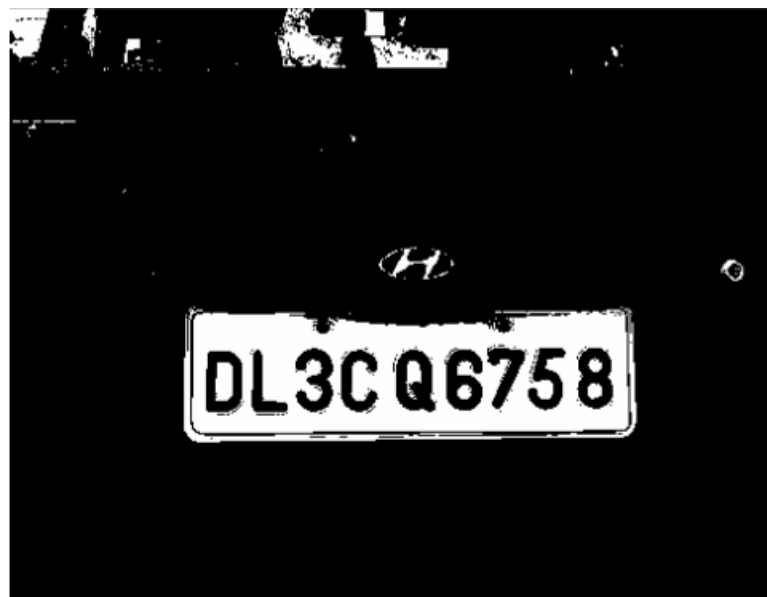


yang diterapkan di penelitian tersebut adalah 158. Hasil *thresholding* ini akan ditampilkan pada Gambar 2.23.



**Gambar 2.22 BGR Image**

**Sumber :** (*Vehicle License Identification and Recognition, 2009*)

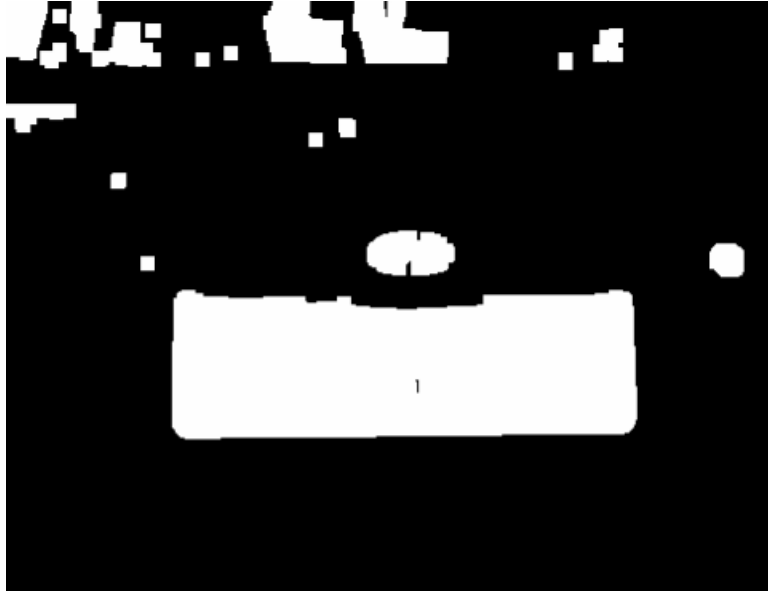


**Gambar 2.23 Binary Image**

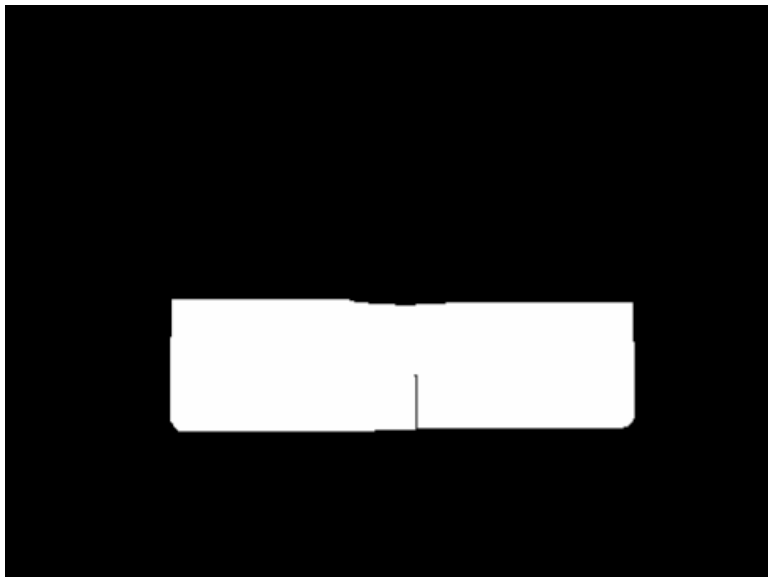
**Sumber :** (*Vehicle License Identification and Recognition, 2009*)

Gambar 2.22 dan Gambar 2.23 merupakan perubahan dari BGR *image* ke *binary image*, metode *thresholding* yang digunakan adalah *manual*

*thresholding* dengan nilai 158, selanjutnya akan dilakukan *dilation* dan *noise removal* untuk menghapus *white region* yang berukuran kecil.



**Gambar 2.24 Binary Image after Dilated**  
**Sumber : (Vehicle License Identification and Recognition, 2009)**



**Gambar 2.25 Binary Image after Noise Removal**  
**Sumber : (Vehicle License Identification and Recognition, 2009)**

Langkah selanjutnya adalah melakukan pemotongan *license plate* pada gambar dengan memanfaatkan *aspect ratio* dan koordinatnya ditemukan dengan proses *connected component labelling*.



**Gambar 2.26 Hasil dari proses *cropping***

**Sumber :** (*Vehicle License Identification and Recognition, 2009*)

Hasil di atas merupakan hasil *crop image*, terlihat disini ukuran yang dipotong melebihi ukuran *license plate*, hal ini dilakukan karena proses sebelumnya yaitu *dilation*. Penelitian ini memiliki tingkat keberhasilan pendeteksian dengan persentase 88%.

## **2.10 Kesimpulan**

Dengan melakukan literatur penelitian terhadap dua buah jurnal yaitu "*License Plate Detection and Recognition for Indonesian Cars*", 2016 dan jurnal "*Vehicle Identification and Recognition*", 2009. Dilihat dari hasil akurasi pendeteksian TNKB bahwa penelitian yang memanfaatkan aspek rasio memiliki persentase keberhasilan yang lebih tinggi, yaitu sebesar 88%, sedangkan yang memanfaatkan *vertical and horizontal edge detection* sebesar 78.75%. Oleh karena itu studi ini akan menggunakan metode yang dilakukan oleh jurnal "*Vehicle Identification and Recognition*" yaitu memanfaatkan aspek rasio dari TNKB untuk mendapatkan lokasi TNKB. Akan tetapi dilihat dari bentuk TNKB pada Indonesia dan India memiliki beberapa perbedaan, terutama untuk latar belakang TNKB, sehingga

dibutuhkan beberapa penyesuaian dalam menerapkan metode tersebut. Pada penelitian “*License Plate Detection and Recognition for Indonesian Cars*”, 2016, akurasi dalam pengenalan karakter TNKB sebesar 57.14%, hal ini akan diperbaiki agar mendapatkan hasil yang lebih baik.