TUGAS AKHIR – TE 141599

RANCANG BANGUN PENGENALAN CITRA RAMBU LALU LINTAS DENGAN METODE *LOCAL BINARY PATTERN*

Kristopher Lukas
NRP 07 1113 4000 0105

Dosen Pembimbing
Dr. Ir. Hendra Kusuma, M.Eng., Sc.
Ir. Tasripan, MT.

DEPARTEMEN TEKNIK ELEKTRO
Fakultas Teknologi Elektro
Institut Teknologi Sepuluh Nopember
Surabaya 2018

FINAL PROJECT – TE 141599

DESIGN AND IMPLEMENTATION TRAFFIC SIGN IMAGE RECOGNITION BASE ON LOCAL BINARY PATTERN METHOD

Kristopher Lukas
NRP 07 1113 4000 0105

Supervisor
Dr. Ir. Hendra Kusuma, M.Eng., Sc.
Ir. Tasripan, MT.

DEPARTMENT OF ELECTRICAL ENGINEERING
Faculty of Electrical Technology
Institut Teknologi Sepuluh Nopember
Surabaya 2018

LEMBAR PERNYATAAN KEASLIAN

PERNYATAAN KEASLIAN
TUGAS AKHIR

Dengan ini saya menyatakan bahwa isi sebagian maupun keseluruhan Tugas Akhir saya dengan judul "Rancang Bangun Pengenalan Citra Rambu Lalu Lintas dengan Metode *Local Binary Pattern*" adalah benar-benar hasil karya intelektual mandiri, diselesaikan tanpa menggunakan bahan-bahan yang tidak dijinkan dan bukan merupakan karya pihak lain yang saya akui sebagai karya sendiri.

Semua referensi yang dikutip maupun dirujuk telah ditulis secara lengkap pada daftar pustaka. Apabila ternyata pernyataan ini tidak benar, saya bersedia menerima sanksi peraturan yang berlaku.

Surabaya, 20 Juni 2018

Kristopher Lukas
NRP. 0711 13 4000 0105

RANCANG BANGUN PENGENALAN CITRA RAMBU LALU LINTAS DENGAN METODE *LOCAL BINARY PATTERN*

TUGAS AKHIR

Diajukan Guna Memenuhi Sebagian Persyaratan Untuk
Memperoleh Gelar Sarjana Teknik
Pada
Bidang Studi Elektronika
Departemen Teknik Elektro
Institut Teknologi Sepuluh Nopember

Menyetujui

Dosen Pembimbing I,	Dosen Pembimbing II,
Dr. Ir. Hendra Kusuma, M.Eng.Sc.	Ir. Tasripan, MT.
NIP: 196409021989031003	NIP: 196204181990031004

KEMENTERIAN RISET, TEKNOLOGI, DAN PENDIDIKAN TINGGI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
FAKULTAS TEKNOLOGI ELEKTRO
DEPARTEMEN TEKNIK ELEKTRO

SURABAYA
JULI, 2018

ABSTRAK

Rancang Bangun Pengenalan Citra Rambu Lalu Lintas dengan Metoda *Local Binary Pattern*

Kristopher Lukas
07 11 13 4000 0105

Dosen Pembimbing I : Dr. Ir. Hendra Kusuma, M.Eng., Sc.
Dosen Pembimbing II : Ir. Tasripan, MT.

Abstrak:

Pada Tahun 2013, data kecelakaan yang didapat dari Departemen Perhubungan Indonesia, terdapat setidaknya 8 kecelakaan yang 6 diantaranya disebabkan oleh kesalahan dari manusia. Hal inilah yang menyebabkan pengembangan teknologi pada keselamatan berkendara. *Autonomus Vehicle* merupakan teknologi yang mengambil alih peran manusia dalam mengemudi kendaraan.

Metode *Local Binary Pattern* mengubah sebuah citra untuk didapatkan nilai *histogram vector* yang tidak dipengaruhi oleh intensitas cahaya pada citra. Pada program akan dilakukan pengambilan citra untuk *data learning* dan pendeteksian rambu dengan *Hough Transformation*. Pada pengenalan dilakukan komparasi nilai histogram citra dengan *data learning* menggunakan metode *Chi-Square.* Dari hasil data yang didapatakan, bahwa pengujian pengenalan sebuah citra dengan menggunakan metode LBP menghasilkan tingkat akurasi pengenalan 96%, dengan menggunakan proses segmentasi sebanyak 100 *subdivide*.

Pada penelitian tugas akhir ini didapatkan bahwa pengenalan rambu lalu lintas menggunakan metode LBP sangat baik. Dari penelitian ini akan dilakukan pengembangan terhadap pengenalan rambu lalu lintas pada *autonomous vehicle*, sehingga dapat berkendara di jalan raya tanpa melanggar hukum.

Kata kunci: *Local Binary Pattern*, Pengenal Objek, OpenCV, *Hough Transformation*.

Halaman ini sengaja dikosongkan

ABSTRACT

Design and Implementation Traffic Sign Image Recognition base on Local Binary Pattern Method

Kristopher Lukas
07 11 13 4000 0105

Supervisor I : Dr. Ir. Hendra Kusuma, M.Eng., Sc.
Supervisor II : Ir. Taripan, MT.

Abstract:

There were 8 traffic accident, 6 of them were road coallison caused by human error in 2013, from Indonesia Departemen of Traffic. This kind of accident trigger some concern in safety feature in traffic. Autonomus Vehcile Technology is one of the technology that can reduce or eliminate people role to drive.

In this final project a system has been designed and built to recognize traffic sign using Local Binary Pattern method. This program consists of learning phase with collecting image for data learning and detect object using Hough Transformation. Second phase is recognition, this consist of LBP procese and compare histogram from input images with data learning using Chi-Sqr. To get better recognition, this method combines with segmentation to get vector histogram. After testing this program, it shows a result that the recognize has 96% accuracy with using 100 segmentations for vector histogram.

In this research, recognize the traffic sign using LBP method show great result. This method doesn't affect the illumination of image so the image can still be recognized when its night. Further work for this program is new development of LBP method to be use as the recognizer in fast image recognizer and development for traffic sign recognizer for autonomous vehicle so it can run on traffic.

Keywords: *Local Binary Pattern*, *Object Recognition*, OpenCV, Hough Trasnformation.

Halaman ini sengaja dikosongkan

KATA PENGANTAR

Segala puji syukur kepada Tuhan YME , atas segala nikmat, berkat dan karunia-Nya yang tak terkira kepada penulis, hingga penulis mampu menyelesaikan Tugas Akhir dengan judul:

Pengenalan Citra Uang Kertas Rupiah Dengan Metoda *Local Binary Pattern*

Tujuan utama tugas akhir ini adalah sebagai salah satu persyaratan untuk menyelesaikan jenjang pendidikan pada Bidang Studi Elektronika Teknik Elektro Institut Teknologi Sepuluh Nopember.

Atas selesainya penyusunan tugas akhir ini, penulis ingin mengucapkan terima kasih kepada:

1. Dr. Ir. Hendra Kusuma, M.Eng., Sc. dan Ir. Tasripan, MT. selaku dosen pembimbing Tugas Akhir yang telah memberi bimbingan, penjelasan, nasehat dan kemudahan dalam penyelesaian Tugas Akhir ini.
2. Dr. Ir. Djoko Purwanto, M.Eng., Ir. Haris Pringadi, MT., Ronny Mardiyanto, ST., MT., Ph.D., Muhammad Attamimi selaku dosen penguji yang telah mengoreksi Tugas Akhir ini.
3. Dr. Eng. Ardyono Priyadi, ST., M.Eng. selaku ketua Departemen Teknik Elektro ITS.
4. Bapak Ir. Posman Sitorus dan drg, Tuti Herawati selaku orang tua penulis yang selalu mendukung dan mendoakan penulis.
5. Halum Ghulami, M Kukuh Prayogo, Heriyanto bin Afandi, Eber Wonda dan teman-teman E53 yang tidak dapat disebutkan satu persatu.
6. Serta semua pihak yang telah membantu penulis dalam menyelesaikan Tugas Akhir ini.

Penulis berharap para pembaca Tugas Akhir ini bersedia memberikan kritik, saran dan masukan yang membangun agar selanjutnya dapat menambah manfaat untuk kedepannya. Semoga laporan tugas akhir ini dapat bermanfaat dan bisa dijadikan referensi bagi Tugas Akhir selanjutnya.

Halaman ini sengaja dikosongkan

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

Halaman ini sengaja dikosongkan

BAB I

PENDAHULUAN

1.1 Latar Belakang

Data kecelakaan pada tahun 2013 dari Departemen Perhubungan Indonesia terdapat 8 kecelakaan, 6 diantaranya disebabkan oleh kesalahan manusia. Hal ini yang menyebabkan perhatian pada sistem keselamatan berkendara untuk di kembangkan agar dapat mengurangi angka kecelakaan yang disebabkan oleh factor manusia. Teknologi Kendaraan Otonomus atau *Autonomus Vehicle* dikembangkan untuk mengurangi peran manusia dalam mengendarai kendaraannya. Teknologi *Autonomus Vehicle* dikembangkan oleh beberapa negara seperti German, America dan Jepang. Riset yang dilakukan oleh Universitas dan Industri Otomotif menggunakan sistem yang berbasis teknologi navigasi satelit dan teknologi kamera.

Penelitian teknologi *Autonomus Vehicle* dimulai pada tahun 1920. Pada tahun 1920 digunakan oleh Houdian Radio Control pada kendaraan dengan mendemonstrasikan kendaraan yang dapat dikontrol dengan menggunakan teknologi radio. Pada tahun 1980, dilakukan percobaan kendaraan yang dapat dijalankan dengan menggunakan teknologi berbasis kamera oleh Ernst Dickmanns dan timnya di Universitas Bundeswehr Munich pada mobil dengan kondisi jalan raya yang kosong. Setelah dilakukan riset ini beberapa Industri Otomotif besar memulai untuk melakukan riset tentang kendaraan otonomus. Pada juli tahun 2013, VisLab mendemonstrasikan BRAiVE, kendaraan yang dapat bergerak secara ontonom dengan kondisi rute yang ditentukan sampai dengan jalan raya. Pada tahun 2013, pemerintah Amerika Serikat telah melegalkan hukum mengenai kendaraan otonomus. Dengan adanya riset dalam pengembangan teknologi kendaraan otonomus, maka muncul ide untuk pengembangan sistem kendaraan otonomus untuk membaca rambu lalu lintas di jalan raya agar kendaraan dapat mematuhi peraturan jalan raya.

Perancangan sistem pada tugas akhir ini menggunakan kamera dan *mini PC* untuk melakukan pemrosesan data dan citra. Kamera digunakan untuk menangkap citra yang ada di jalan raya dan akan mendeteksi objek berupa rambu lalu lintas. Selanjutnya citra yang didapatkan akan diproses untuk dicari rambu lalu lintas dan diproses untuk diambil nilai *Local Binary Pattern* yang kemudian akan di komparasi dengan *histogram* dari

data learning, sehingga citra rambu lalu lintas dapat dikenali. Keseluruhan proses dilakukan pada *mini PC*. Program yang digunakan adalah Microsoft Visual Studio 2015 dengan menggunakan *library* dari OpenCV untuk memproses citra secara digital.

Metode pengenalan citra pada sistem ini dengan metode *Local Binary Pattern*, yaitu sebuah operasi yang memberikan sebuah label pada *pixel* di citra dengan menggunakan satu angka decimal sebagai pusat, kemudian dari nilai pusatnya akan dikomparasi dengan nilai disekitarnya sehingga didapatkan sebuah nilai histogram sebesar 8 bit. Metode ini dipilih karena memiliki akurasi pengenalan yang lebih besar dan waktu yang digunakan untuk memproses sebuah citra lebih cepat. Metode *Local Binary Pattern* memiliki algoritme yang dapat mengurangi pengaruh pencahayaan pada objek yang akan dikenali.

Hasil penelitian pengenalan rambu lalu lintas pada tugas akhir ini diharapkan dapat menjadi bahan untuk pengembangan fitur dalam mengenali sebuah objek terutama pada pengembangan kendaraan otonomus, sehingga diharapkan sistem ini dapat membantu pengenalan sebuah citra lebih baik lagi.

1.2 Perumusan Masalah

Pengenalan rambu lalu lintas telah dikembangkan dengan menggunakan beberapa metode pengenalan citra. Pada tugas akhir ini dilakukan penelitian dengan menggunakan metode *Local Binary Pattern*, karena algoritme pemrosesan sebuah citra digital memiliki tingkat pemrosesan yang sederhana dan pengaruh *illumination* dapat dikurangi, sehingga memiliki kelebihan dalam waktu pemrosesan yang lebih singkat dan akurasi pengenalan yang lebih baik dibandingkan metode lainnya. Permasalahan yang harus di selesaikan dalam penelitian tugas akhir ini yaitu:

1. Bagaimana menentukan teknik pengambilan citra dari kamera untuk menentukan apakah ada rambu lalu lintas atau tidak agar citra rambu lalu lintas dapat dikenali dengan baik.
2. Bagaimana melakukan pemrosesan citra digital, agar citra yang didapatkan dapat dikenali dengan baik.
3. Bagaimana menerapkan metode *Local Binary Pattern*, agar citra rambu lalu lintas dapat dikenali dengan baik.

4. Bagaimana mengetahui kelebihan dan kelemahan dari metode *Local Binary Pattern* untuk mengenali citra rambu lalu lintas terhadap metode lain pada penelitian yang sudah dilakukan.

1.3 Tujuan dan Manfaat Penelitian

Tujuan dari penelitian ini adalah:

1. Mengetahui teknik pengambilan citra rambu lalu lintas dengan kamera agar didapatkan citra rambu lalu lintas yang baik.
2. Mengetahui proses pengolahan citra, agar citra yang dihasilkan kamera mempunyai kualitas gambar yang lebih jelas dan baik.
3. Mengetahui cara mendeteksi dan mengenali citra rambu lalu lintas dengan metode *Local Binary Pattern*.

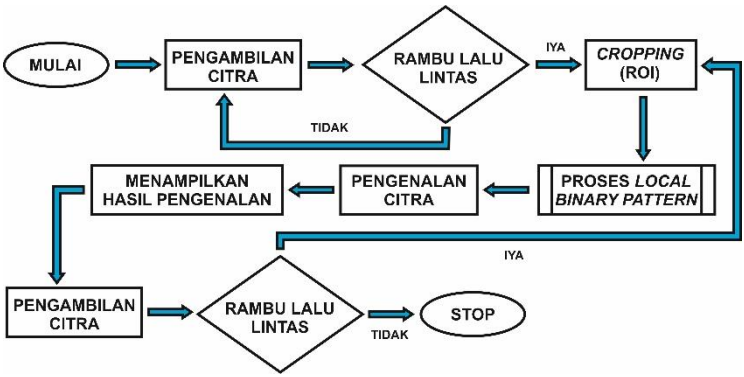
Manfaat dari penelitian tugas akhir ini adalah dihasilkan sebuah sistem yang dapat mendeteksi dan mengenali citra rambu lalu lintas yang ada di jalan raya. Sehingga dapat memudahkan pengembangan teknologi keselamatan dalam dunia otomitif dan dapat menjadi sebuah acuan untuk penelitian dalam pengenalan sebuah objek dengan menggunakan metode *Local Binary Pattern*.

1.4 Metodologi Penelitian

Dalam penyelesaian tugas akhir ini digunakan metodologi sebagai berikut:

1. Studi literatur
   Pada tahap ini dilakukan pengumpulan dan pembelajaran dasar teori yang dapat menunjang penelitian tugas akhir. Dasar teori yang digunakan dapat diambil dari buku, jurnal dan artikel di internet dan forum – forum diskusi yang relevan dan dapat di pertanggung jawabkan.
2. Perancangan perangkat keras
   Perancangan perangkat keras pada tugas akhir ini terdiri dari 2 (dua) bagian yaitu kamera untuk mengambil citra dan *mini PC* untuk melakukan pemrosesan data dan citra. *Mini PC* yang digunakan minimal memiliki spesifikasi untuk menjalankan program Microsoft Visual Studio 2015. Kamera yang digunakan adalah *Webcam Camera* yang memiliki resolusi minimal 1080 HD px.
3. Perancangan perangkat lunak

Perancangan perangkat lunak pada penelitian tugas akhir ini meliputi perancangan program pengambilan dan mendeteksi citra rambu lalu lintas agar citra yang didapatkan dapat di proses ke tahap selanjutnya. Program selanjutnya adalah pengubahan citra agar dapat dikenali dengan menggunakan metode *Local Binary Pattern* untuk di dapatkan *histogram*. Program selanjutnya adalah *histogram* yang didapatkan akan diproses untuk dikenali dengan cara mengkomparasi *histogram* dari citra rambu lalu lintas dengan *histogram data learning*. Kemudian akan didapatkan hasil dari pengenalan citra pada layer. Secara garis besar perancangan perangkat lunak dapat digambarkan menggunakan *flowchart* berikut:



Gambar 1.1 Diagram alur algoritme perangkat lunak.

1. Pengambilan citra
   Algoritma pengambilan citra dirancang untuk mengambil citra dari kamera untuk di deteksi.
2. Apakah ada rambu lalu lintas
   Pada bagian ini citra yang diambil dari kamera akan di deteksi apakah ada objek deteksi berupa rambu lalu lintas, jika iya akan dilanjutkan ke proses selanjutnya, jika tidak akan dilakukan pengambilan ulang citra.
3. *Region Of Interest* (ROI)

Pada proses ini citra yang didapat akan di deteksi dan di crop dengan menggunakan metode *Region Of Interest* untuk mencari rambu lalu lintas agar bias di proses.

4. *Local Binary Pattern* (LBP)
   Pada proses ini citra yang didapatkan akan di lakukan pemrosesan citra dengan menggunakan metode *Local Binary Pattern* dan akan di dapatkan *histogram* untuk di proses selanjutnya.
5. Pengenalan citra
   Pada proses ini citra yang sudah di proses pada proses sebelumnya akan di ambil nilai *histogram* untuk di komparasi dengan *histogram data learning* untuk dikenali.
6. Menampilkan hasil pengenalan
   Pada proses ini hasil dari komparasi *histogram* akan ditampilkan untuk mengetahui hasil deteksi citra rambu yang diambil.
7. Pengujian alat
   Pengujian alat dilakukan untuk menentukan tingkat akurasi deteksi rambu lalu lintas dari citra dan akurasi pengenalan citra rambu lalu lintas. Pengujian dilakukan untuk mengetahui apakah program dan algoritme yang digunakan berjalan dengan baik.
8. Analisa
   Analisa dilakukan terhadap hasil pengujian sehingga dapat ditentukan kasalahan dan karakteristik dari algoritme serta program deteksi dan pengenalan yang telah digunakan. Kesalahan dan karakteristik yang diuji adalah akurasi pendeteksi dan pengambilan citra rambu lalu lintas dan pengenalan citra rambu lalu lintas. Jika terdapat banyak kesalahan maka diperlukan perbaikan pada sistem.
9. Penarikan kesimpulan dan saran
   Penarikan kesimpulan mengacu pada hasil data pengujian sistem, analisis data dan referensi terkait. Kesimpulan menunjukkan hasil kerja sistem secara garis besar apakah sesuai rumusan masalah yang telah dibuat.
10. Penyusunan laporan
    Proses terakhir adalah membuat dokumentasi pelaksanaan tugas akhir yang meliputi dasar teori, proses perancangan, pembuatan dan pengujian sistem pengenalan.

1.5 Sistematika Penulisan

Laporan tugas akhir ini terdiri dari lima bab dengan sistematika penulisan sebagai berikut:

Bab 1: PENDAHULUAN
Pada bab ini meliputi latar belakang masalah yang mendasari pembuatan penelitian, perumusan masalah yang berkaitan dengan penelitian pada tugas akhir, tujuan, manfaat penelitian, metodologi penelitian yang di gunakan dalam penelitian tugas akhir, sistematika penulisan dan relevansi penelitian tugas akhir.

Bab 2: DASAR TEORI
Pada bab ini meliputi dasar – dasar teori penunjang yang digunakan dalam pengerjaan tugas akhir. Dasar teori terdiri dari rambu lalu lintas, *mini PC*, Microsoft Visual Studio, Opencv, pengenalan objek dan dasar teori mengenai *Local Binary Pattern*.

Bab 3: PERANCANGAN SISTEM
Pada bab ini meliputi penjabran mengenai perancangan sistem yang akan dibangun seperti perancangan perangkat keras dengan menggunkan kamera dan *mini PC*. Perancangan perangkat lunak dengan menggunakan OpenCV dan Microsoft Visual Studio 2015.

Bab 4: PENGUKURAN DAN ANALISIS SISTEM
Pada bab ini meliputi hasil dari rancang bangun sistem dan rancangan program yang telah di realisasikan.

Bab 5: PENUTUP
Bab ini menjelaskan tentang kesimpulan dari hasil penelitian meliputi kekurangan-kekurangan pada kerja alat, sistem, algoritma dan metoda yang digunakan serta dijelaskan pula saran untuk pengembangan penelitian ke depan.

1.6 Relevansi

Hasil dari penelitian tugas akhir ini diharapkan akan dapat memberi manfaat dan dampak sebagai berikut:

1. Dapat digunakan sebagai sistem untuk mengenal rambu lalu lintas yang baik.
2. Hasil dari penelitian pada tugas akhir diharapkan dapat menjadi acuan dalam penelitian mengenai pengenalan dan deteksi objek dan juga dapat menjadi pengembangan fitur keselamatan dalam kendaraan otonomus.
3. Sebagai dasar atau acuan penelitian lebih lanjut, agar dapat dikembang menjadi lebih baik lagi.

Halaman ini sengaja dikosongkan

BAB II

DASAR TEORI

2.1 Rambu Lalu Lintas

Rambu lalu lintas merupakan bagian dari perlengkapan jalan raya untuk memberikan petunjuk atau perintah kepada pengguna jalan raya. Rambu lalu lintas terdiri dari lambing, huruf, angka, kalimat atau perpaduan diantaranya. Rambu lalu lintas di jalan raya sudah diatur didalam Peraturan Mentri Perhubungan No.13 Tahun 2014, diamana telah diatur bentuk, bahan, cara pemasangan dan fungsi dari rambu lalu lintas. Agar rambu lalu lintas dapat dikenali dan dibaca dengan baik oleh pengendara, maka bahan yang digunakan adalah bahan *retro-reflektif* pada rambu konvensional.

Rambu lalu lintas sudah diterapkan pada zaman Kerajaan Roma, dimana rambu lalu lintas yang digunakan berupa sebuah pillar yang disebut *milestone*. *Milestone* terdiri dari tulisan untuk menunjukan arah tempat yang dituju dan waktu tempuh. Semenjak dunia otomotif memulai untuk melakukan pengembangan dan penjualan kendaraan bermotor, tingkat kepadatan kendaraan menjadi meningkat. Pada tahun 1930, setiap negara memulai untuk menerapkan rambu lalu lintas yang dibuat untuk menjaga ketertiban berkendara di jalan raya. Rambu yang digunakan terdiri dari simbol atau tulisan agar memudahkan pengendara menerjemahkan isi dari rambu lalu lintas.

Rambu lalu lintas di beberapa negara memiliki bentuk atau symbol yang berbeda, tetapi masih memiliki pengertian yang sama, seperti yang dicontohkan pada gambar 2.1 a dan b.

(a)

(b)

Gambar 2.1 Rambu lalu lintas dilarang Parkir
(a) Singapura (b) Indonesia

Tujuan dari pembuatan rambu lalu lintas yang sedikit berbeda dikarenakan adanya adaptasi dari peraturan departemen perhubungan tiap negara dan kondisi jalan, tetapi tetap memiliki arti yang sama sehingga mudah dipahami oleh pengendara beda negara atau daerah.

2.1.1 Jenis rambu lalu lintas

Di Indonesia rambu lalu lintas telah diatur fungsi dan pengelompokannya oleh Peraturan Mentri Perhubungan No.13 Tahun 2014. Rambu lalu lintas dibagi menjadi 4 (empat) jenis yang disesuaikan dengan peruntukan dan kondisi jalan raya. Jumalh rambu lalu lintas yang ada di Indonesia adalah 292 rambu lalu lintas, dan rata – rata penggunaan rambu yang ada di dalam kota adalah sebanyak 60 jenis rambu lalu lintas. Data rambu lalu lintas yang ada di Indonesia dapat di lihat pada lampiran.

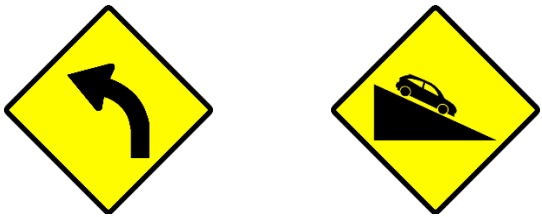
2.1.1.1 Rambu Peringatan

Rambu peringatan digunakan untuk memberi peringatan kepada pengendara mengenai adanya kemungkinan bahaya yang akan dihadapi atau untuk mengingatkan pengendara untuk lebih waspada dengan kondisi jalan yang akan dihadapi. Kemungkinan adanya bahaya merupakan sebuah kondisi diamana pengendara dituntut kewaspadaannya. Beberapa kondisi yang menuntut untuk kewaspadaan dari pengendara jalan raya adalah sebagai berikut:

1. Kondisi prasaranan pada jalan raya.
2. Kondisi lingkungan.
3. Kondisi alam sekitar.

4. Lokasi rawan terjadinya kecelakaan.

Ciri ciri yang terdapat dari rambu lalu lintas peringatan yaitu:
1. Memiliki warna dasar kuning.
2. Memiliki garis tepi berwarna hitam.
3. Terdapat simbol berupa huruf atau angka yang berwarna hitam.
4. Lambang pada rambu ini berwarna hitam.



(a) (b)

Gambar 2.2 Rambu peringatan.
(a) Tikungan kekiri. (b) Turunan landai

2.1.1.2 Rambu Petunjuk

Rambu petunjuk merupakan rambu lalu lintas yang digunakan untuk menunjukan arah tujuan dari pengendara. Rambu lalu lintas ini memberikan manfaat kepada pengendara agar pengendara tidak salah memasuki jalur lajur kendaraan dan tidak salah arah dalam mencapai tujuan. Pada rambu ini terdapat ciri ciri sebagai berikut:

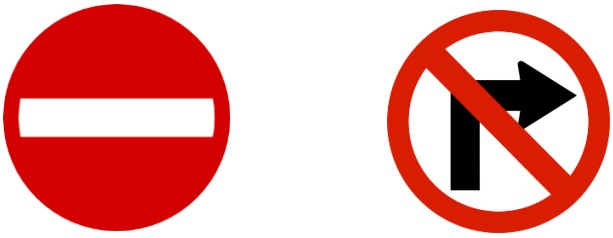
1. Rambu memiliki warna dasar hijau atau biru.
2. Tulisan, lambang, atau angka memiliki warna putih.
3. Terdapat arah petunjuk lokasi dan nama kota yang dituju.

Gambar 2.3 Rambu lalu lintas petunjuk.

2.1.1.3 Rambu Larangan

Rambu larangan merupakan sebuah rambu lalu lintas yang berisi perintah atau larangan agar dipatuhi oleh pengendara jalan raya. Rambu larangan memberikan mafaat kepada pengendara agar tidak terjadi kemacetan atau kecelakaan yang akan terjadi di jalan raya. Sifat dari rambu ini yaitu harus di patuhi dan di ikuti oleh pengendara. Ciri - ciri dari rambu ini adalah sebagai berikut:

1. Rambu memiliki warna dasar merah.
2. Untuk beberapa rambu memiliki warna dasar putih dengan garis tepi warna merah.
3. Terdapat angka, huruf dan simbol yang memiliki warna hitam.



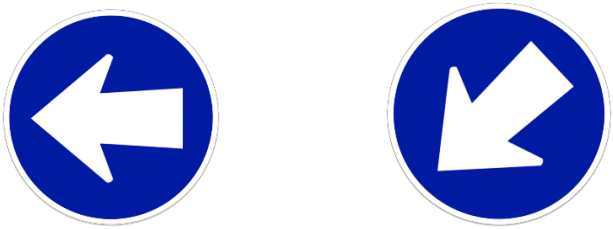
(a) (b)

Gambar 2.4 Rambu lalu lintas larangan.
(a) Rambu dialarang masuk. (b) Rambu dilarang belok kanan

2.1.1.4 Rambu Perintah

Rambu perintah merupakan sebuah rambu lalu lintas dimana pengendara harus memtuhi isi dari rambu lalu lintas. Perbedaan rambu lalu lintas ini adalah, rambu lalu lintas perintah dibuat dengan manfaat agar terjadinya keserasian antara pengendara di jalan raya, sedangkan rambu larangan lebih mengarah kepada hal yang tidak boleh dilakukan pengendara jalan raya dan lebih merujuk kepada Undang Undang Lalu Lintas dan Angkutan Jalan. Ciri – ciri yang dimiliki oleh rambu perintah adalah sebagai berikut:

1. Memiliki warna dasar biru tua
2. Terdapat lambang, huruf, angka yang memiliki warna putih.
3. Memiliki garis tepi berwarna putih.



(a) (b)

Gambar 2.5 Rambu lalu lintas perintah.
(a) Rambu wajib belok kiri. (b) Rambu wajib masuk lajur kiri

2.2 *Local Binary Pattern*

Metode *Local Binary Pattern* digunakan untuk menganalisa tekstur dari sebuah citra dengan ukuran yang lebih kecil dari citra sebelumnya. Selanjutnya dikembangkan menjadi metode pengenalan yang dapat diterapkan pada pengenalan objek lainnya. Dengan menggunakan metode LBP struktur sebuah citra didapatkan dengan cara membandingkan nilai *pixel* sebuah citra dengan beberapa nilai *pixel* di sekitarnya. Keuntungan yang didapat dari metode ini adalah perubahan *ilumination* pada sebuah

objek tidak memiliki pengaruh yang besar dan komputasi yang sederhana sehingga dapat dicapai tingkat akurasi dan waktu komputasi lebih cepat.

Algoritme pengoperasian yang digunakan oleh LBP adalah algoritme yang memberikan label pada suatu *pixel* di citra dengan sebuah integer yang disebut dengan *local binnary pattern code*, yang mana label tersebut adalah hasil dari *encode* struktur lokal *pixel* - *pixel* yang berada di sekitar *pixel* citra tersebut. Algoritme pengoperasian paling dasar pada LBP dilakukan dengan cara setiap *pixel* pada citra dikomparasikan dengan 8 (delapan) *pixel* tetangga disekitarnya dalam jangkauan 3 x 3*px* dengan membandingkan nilai *pixel* tetangga dengan nilai *pixel* pada bagian tengah dari *window* 3 x *3px*. Sehingga jika nilai *pixel* tetangga lebih besar dari pada nilai *pixel* ditengah maka nilai *pixel* tetangga akan diubah menjadi angka 1, jika nilai *pixel* tetangga lebih kecil dari nilai *pixel* ditengah maka nilai *pixel* tetangga akan diubah menjadi 0. Selanjutnya akan didapat 8 (delapan) angka biner dengan cara menyatukan hasil pengubahan komparasi pada 8 *pixel* tetangga mengikuti arah jarum jam dimulai dari *pixel* yang berada pada kiri atas dan hasil dari nilai binner akan diubah menjadi nilai desimal 8*bit* untuk label LBP sehingga didapatkan nilai *histogram*.

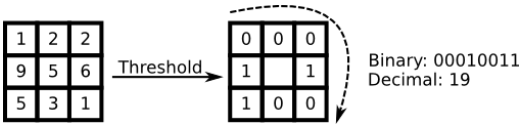
Pada pengembangan metode LBP *pixel* sekitar dapat lebih dari delapan dan dapat memiliki radius yang bervareasi sesuai dengan kebutuhan dan hasil yang diinginkan, secara umum nilai LBP *pixel* (Xc,Yc) pada citra dapat dirumuskan sebagai berikut:

$$LBPp,r\ (Xc,Yc) = \sum_{p=0}^{p-1} s(ip - ic)2^p \qquad (2,1)$$

Parameter yang ada yaitu:
p = jumlah *pixel* tetangga
r = jarak radius *pixel* tetangga
Xc = koordinat (x) *pixel* tengah
Yc = koordinat (y) *pixel* tengah
ip = nilai *grayscale pixel* tetangga atau *neighbour*
ic = nilai *grayscale pixel* tengah, dan fungsi s(x) didefinisikan sebagai:

$$s(x) = \begin{cases} 1, & if\ x \geq 0 \\ 0, & if\ x\ < 0 \end{cases} \qquad (2,2)$$

Yang menunjukkan bahwa jika hasil pengurangan bernilai negatif maka nilai *pixel* tetangga dikodekan dengan nilai 0 dan jika hasil pengurangan bernilai 0 atau positif maka nilai *pixel* tetangga dikodekan dengan nilai 1.



Gambar 2.6 Operasi *Local Binnary Pattern* sederhana.

Citra yang telah melalui proses LBP, memiliki *channel* sebanyak 1 *channel*. Hasil pengubahan citra LBP dapat dilihat pada gambar 2.7. Pada citra tersebut terdapat warna *colorspace grayscale*, dan pada citra yang dihasilkan terdapat beberapa *feature* yang nanti akan dapat diambil nilai histogramnya. Untuk mendapatkan hasil histogram yang terbaik, maka citra tersebut akan melalui proses segmentasi untuk menambah histogram dari citra dan menambah tekstur baru pada citra.

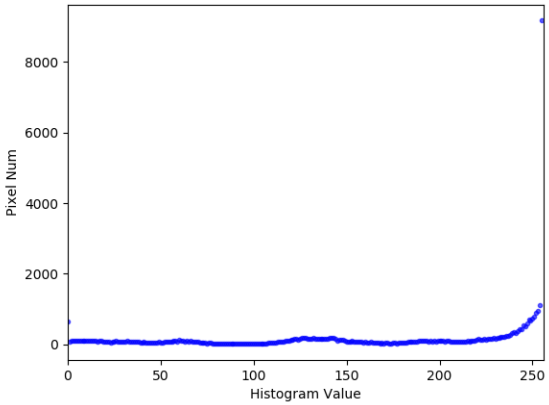
(a) (b)

Gambar 2.7 Citra yang mengalami proses LBP
(a) Citra sebelum proses LBP. (b) Citra sesudah proses LBP

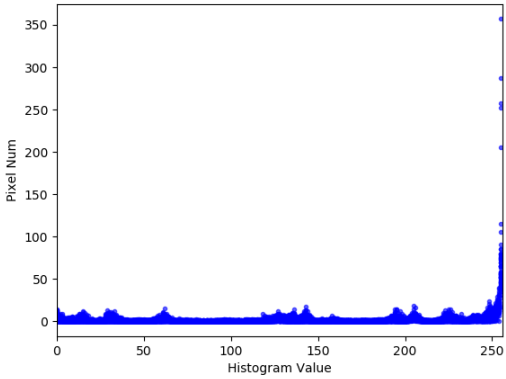
Limitasi yang dimiliki operasi LBP sederhana adalah ukuran 3x3 *pixel* tetangga tidak dapat menghasilkan sifat dominan pada struktur berskala besar. Untuk itu pengaturan ukuran radius tetangga dapat diatur sehingga dapat dihasilkan sifat dominan pada struktur berskala besar.

Pada hasil dari nilai LBP, citra akan memiliki nilai *pixel* yang lebih kecil dan hanya memiliki 1 *channel* saja. Hal ini disebabkan karena adanya pengurangan nilai dari pixel yang sudah di lewati oleh *kernel* dari LBP, dimana nilai *pixel* sekitarnya diubah menjadi sebuah nilai desimal pada nilai tengahnya dan citra yang diubah hanya memiliki 1 *channel* karena adanya pengubahan *colorspace* dari citra masukan pertama menjadi *grayscale*.

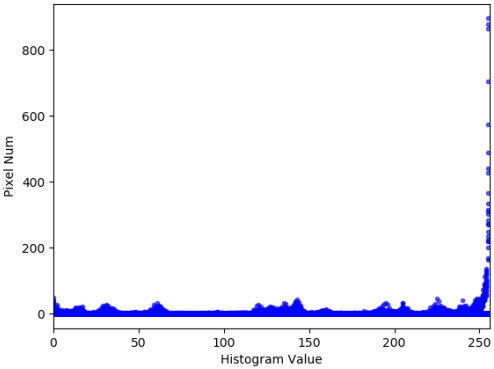
Untuk mendapatkan sebuah *feature* pada citra untuk dapat dikenali, maka diperlukan segmentasi pada citra masukan. Proses segmentasi dilakukan sesaat citra akan diubah menjadi sebuah citra LBP. Dengan melakukan segmentasi, citra akan memiliki *feature* yang nantinya akan dapat menambah tingkat ketelitian dalam pengenalan citra. Histogram yang akan didapat memiliki nilai yang berbeda beda ketika adanya penambahan besarnya segmentasi. Pada gambar 2.8 merupakan nilai histogram dari sebuah citra yang langsung diubah dengan proses LBP, gambar 2.9 (a) merupakan nilai histogram dari pembagian segementasi 5x5, (b) merupakan nilai histogram dari pembagian segmentasi 10x10.



Gambar 2.8 Histogram citra tanpa *subdivide*.



(a)



(b)

Gambar 2.9 Histogram dari LBP.
(a) Histogram 5x5 segmentasi. (b) Histogram 10x10 segmentasi.

Pembagian segmentasi untuk mencari nilai LBP tidak boleh memiliki ukuran yang sangat kecil dari nilai ukuran citra awal. Hal ini dapat menyebabkan citra tidak mendapatkan nilai *histogram feature*. Ketika citra telah diubah menjadi citra LBP, maka histogram akan dapat disatukan menjadi sebuah *feature histogram*. Histogram yang didapat akan di normalisasi dengan menggunakan rumus:

$$Ni = \frac{Hi}{\sum_{j=0}^{n-1} Hj} \qquad (2,3)$$

2.3 *Mini PC*

Mini PC merupakan sebuah perangkat keras yang terdiri dari *processor, motherboard, Video Graphic Array,* dan *RAM*. *Mini PC* memiliki cara kerja dan proses yang sama dengan computer pada umumnya, terdiri dari 3 (tiga) komponen utama, komponen pertama adalah *Hardware,* komponen kedua adalah *Software*, komponen ketiga adalah *Brainware*. Komponen ini merupakan bagian terpenting untuk dapat mengoperasikan *mini PC*.

Komponen pertama pada *mini PC* adalah *hadware*, dimana terdiri dari *motherboard* untuk meletakkan semua perangkat keras seperti *processor, VGA Card,* dan *RAM*. Pada *mini PC* jenis *motherboard* yang digunakan adalah jenis *Mini Motherboard*, hal ini dikarenakan ukuran sebuah *mini PC* yang tidak terlalu besar dan pada *mini PC* hanya dapat melakukan beberapa proses dengan menggunakan sistem yang kecil sehingga *motherboard* pada *mini PC* tidak memiliki *clock* yang tinggi dan *socket* yang banyak. *Processor* yang digunakan dapat menggunakan *processor* pada umumnya seperti Intel core i3 ataupun Intel core i7. Mini PC dengan dimensi dan ukuran yang lebih kecil dari *PC* memudahkan untuk dibawa kemana saja.

Software yang digunakan pada *mini PC* dapat menggunakan *Windows-based operation* atau dengan menggunakan *Linux-based operation*. Pada *mini PC* digunakan *Windows-based operation*, dikarenakan program yang digunakan untuk menjalankan program adalah Microsoft Visual Studio 2015. *Minimum system requirement* pada Visual Studio 2015 adalah sebagai berikut :

1. 1.6 GHZ *Processor.*
2. 1GB *RAM.*
3. Kapasitas minimal *hard disk* adalah 10GB.
4. 600MB *space* pada *hard disk.*
5. DirectX 9 yang dapat memproses citra dengan ukuran minimal 1024 x 768*pixel*.

Dengan menggunakan Microsoft Visual Studio 2015 maka akan menggunakan bahasa pemrograman dengan C / C++ API. Pada Microsoft Visual Studio diperlukan sebuah *library* untuk dapat memproses sebuah citra secara digital. *Library* yang digunakan adalah OpenCV 3.3. Untuk dapat menggunakan *library* dari Opencv diperlukan proses *compailing* pada Visual Studio.

Pada komponen penyusun ketiga adalah *brainware*, merupakan pengguna dari *mini PC* atau yang disebut sebagai *user*. Jika *user* tidak memasukan atau menjalankan sebuah program pada *mini PC*, maka *mini PC* tidak dapat berjalan atau beroperasi dengan sendirinya. *Brainware* dapat melakukan proses *Input* / *Output*, *input* dilakukan oleh *user* melalui sebuah *hardware* seperti *keyboard*, *mouse*, ataupun kamera. Setelah *mini PC* mendapatkan sebuah *input* dari *user*, maka nilai *input* akan diubah menjadi sebuah *command* pada program *operation* dan akan di proses di *Processor*. Pada tahap selanjutnya adalah *command* akan diproses kedalam sebuah *software*, pada *software* akan dilakaukan pengolahan data dari *command* sehingga didapatkan sebuah *output*. *Output* akan dikeluarkan melalui *hardware* yang memiliki fungsi pengingeraan pada *user* seperti *monitor* untuk memperlihatkan proses yang sedang berjalan atau aplikasi yang sedang bekerja, *speaker* untuk *user* dapat mendengarkan sebuah proses yang menggunakan suara.

2.4 *OpenCV*

OpenCV adalah *library* yang dapat digunakan untuk aplikasi pengindraan visual, *library* ini tidak berbayar. OpenCV dikembangkan dan ditulis menggunakan bahasa C dan C++ dan dapat dijalankan pada sistem operasi Linux, Windows dan OS X. Selain dapat digunakan menggunakan bahasa pemrograman C dan C++, OpenCV juga dapat digunakan menggunakan *interface* lainnya seperti Python, Ruby, Matlab dan beberapa bahasa pemrograman lain.

Untuk keperluan penelitian tugas akhir ini akan dijelaskan beberapa fungsi yang memiliki relevansi dengan penelitian tugas akhir yang dikerjakan diantaranya fungsi untuk mengambil citra dari *file* atau kamera, pengubahan citra dari satu warna BGR menjadi HSV lain, *thresholding*, *morphology*, *Region Of Interest*, deteksi *contour* untuk analisa *blob* dan fungsi-fungsi yang lainnya. Salah satu stuktur citra yang digunakan untuk seluruh operasi yang dilakukan oleh fungsi-fungsi yang dimiliki openCV adalah truktrur citra IplImage dan Mat, pada stuktur IplImage atau dapat menggunakan operasi Mat data citra akan dialokasikan secara otomatis pada memori dan akan me-*return* nilai pada data citra jika suatu *pointer* digunakan untuk mengakses data tersebut. Struktur IplImage suatu citra harus dideklarasikan pada awal program sebelum dapat digunakan untuk menampung data citra, contoh penulisan deklarasi IplImge seperti berikut:

```
IplImage* citra;
Mat citra;
```

Di mana citra adalah nama *pointer* yang digunakan untuk menunjuk alamat data citra pada memori. Struktur IplImage pada akhir program harus di-*release* agar memori yang digunakan untuk data citra pada IplImage struktur dapat dialokasikan untuk keperluan lainnya. Untuk mereset struktur IplImage atau Mat dapat mengunakan fungsi:

```
cvReleaseImage(&citra);
```

2.4.1 Akses citra dari *file* dan kamera

Fungsi yang digunakan untuk mengakses citra pada *file* yang telah ada adalah fungsi cvLoadImage() pada C dan cv::imread() untuk menggunakan C++, yang memiliki struktur berikut:

```
Untuk C:
IplImage* cvLoadImage(Const char* filename,
Int iscolor = CV_LOAD_IMAGE_COLOR);

Untuk C++ :
Mat imread(const string&filename, int flags=1);
```

Terdapat dua bagian pada fungsi cvLoadImage dan Mat imread yaitu *filename* dan *iscolor* atau int flags, bagian *filename* adalah nama *file* dari citra yang ingin dimuat, pengisiannya dengan menyertakan ekstensi format citra yang didahului dan diakhiri dengan tanda petik ganda, contohnya seperti ("image.bpm"). Bagian kedua yaitu *iscolor*, bagian ini digunakan untuk mengalokasikan warna pada citra yang akan dimuat dengan mengisi antara CV_LOAD_IMAGE_COLOR untuk mengambil warna citra dalam warna BGR dengan menggunakan 3 *channels* dengan kedalaman warna 8bit, untuk CV_LOAD_IMAGE_GRAYSCALE maka citra yang dimuat akan diubah menjadi citra dengan 1 *channel* dengan kedalaman warna 1bit, sedangkan jika menggunakan CV_LOAD_IMAGE_ANYCOLOR memori akan dialokasikan sesuai dengan jenis citra yang ingin dimuat dan kedalaman warna 8bit, dan jika menggunakan CV_LOAD_IMAGE_UNCHANGED maka memori akan dialokasikan untuk citra dengan *channel* dan kedalaman warna sesuai dengan format awal pada citra.

Terdapat *flag* yang bersifat opsional pada fungsi cvLoadImage() yaitu *flag* CV_LOAD_IMAGE_ANYDEPTH, jika *flag* ini digunakan maka kedalaman warna citra yang dimuat pada struktur IplImage akan menyesuaikan dengan kedalaman warna citra awal. *Flag* opsional ini dituliskan setelah *flag iscolor* jika diinginkan.

Untuk fungsi menyimpan citra maka digunakan fungsi cvSaveImage() pada C dan Mat imwrite() pada C++, yaitu fungsi yang digunakan untuk menyimpan citra ke dalam sebuah *file* dari struktur citra IplImage atau Mat. Dengan struktur fungsi sebagai berikut:

Untuk C:

```
Int cvSaveImage(Cont char* filename, Conts CvArr* image );
```

Untuk C++:

```
Bool imwrite(const strin&filename, InputArray img, constvector<int>&params=vector<int>());
```

Parameter yang terdapat adalaha *filename* adalah nama citra yang ingin disimpan pada *file* penulisannya seperti pada parameter *filename* fungsi cvLoadImage() atau imread(), dan CvArr* image atau InputArray img adalah nama *pointer* struktur IplImage atau Mat yang ingin disimpan kedalam sebuah file.

Terdapat beberapa fungsi yang dapat digunakan untuk mengakses citra yang berasal dari kamera atau *input* citra dari kamera, salah satunya cvCaptureFromCAM(int ID CAM) untuk C dan VideoCapture::VideoCapture(int device), fungsi ini digunakan untuk mengaktifkan dan mengambil citra dari kamera yang nanti akan dsimpan citranya kedalam sebuah *frame*, hanya terdapat satu bagian pada fungsi tersebut yaitu bagian ID CAM atau device, untuk menentukan kamera yang akan digunakan. Berikut contoh program yang digunakan apabila akan mengabil citra dari kamera dan menyimpannya kesebuah *file*:

```
Untuk C++:
Mat image;
Mat capture;
VideoCapture capture;
capture.open(0);
while(1) {
capture.read(capture);
image = capture;
imwrite("Image.jpg", image);
return 0;
}
```

Tugas yang dikerjakan contoh program diatas adalah menyiapkan stuktur IplImage dan Mat dengan nama *pointer* image, baris kedua menyiapkan struktur cvCapture atau capture.read dengan nama *pointer* capture, baris ke tiga mengaktifkan kamera, mengambil data citra dari kamera dan mengalokasikan pada struktur capture, baris ke empat membaca data citra pada capture kemudian mengirim data tersebut ke struktur IplImage dan Mat image, baris ke lima menyimpan data citra pada image ke dalam *file* dan dua baris selanjutnya bertugas untuk merelease struktur cvCapture dan IplImage secara berurutan. Untuk C++ tidak membutuhkan *release* karena dibutuhkan pengambilan gambar dalam sebuah loop

2.4.2 Mengubah *colorspace* citra menjadi *grayscale* dan HSV

Citra berwarna adalah citra yang memiliki 3 *channel* yaitu *channel Blue Green Red* (BGR). Terdapat perbedaan pada OpenCV dimana mengunakan format BGR sedangkan citra 8bit menggunakan format *Red Green Blue* (RGB). Perbedaan penggunaan format juga berpengaruh pada

penyimpanan nilai warna pada citra, untuk BGR nilai *Blue* akan disimpan pada *array* pertama, *Green* pada *array* kedua dan *Red* pada *array* ketiga. Hal ini juga berlaku keblikan dari RGB. BGR digunakan untuk penyimpanan pada citra dengan format bitmap, dimana nilai bitmap disimpan berdasarkan baris terlebih dahulu, dan ketika *height* memiliki nilai *positive*, *array* pada baris terkhir akan dibaca paling pertama. Pemanggilan fungsi untuk mengubah dari BRG menjadi RGB adalah CV_BRG2RGB.

Pada citra dengan *colorspace grayscale* adalah citra yang hanya memiliki 1 (satu) *channel* warna dan memiliki nilai tiap *pixel*nya dari *range* 0 (putih) – 255 (hitam) untuk 8bit citra. Secara manual citra *grayscale* didapatkan dari citra berwarna dengan mencari nilai rata-rata setiap *pixel* dari ketiga *channel*. Contohnya jika terdapat citra berwarna dengan nilai pada *pixel* (0,0) B = 130, G = 200 dan R = 60, maka nilai pada *pixel* (0,0) citra *grayscale* jika didapat dari konversi citra berwarna pada contoh adalah (200+200+60) / 3 = 130.

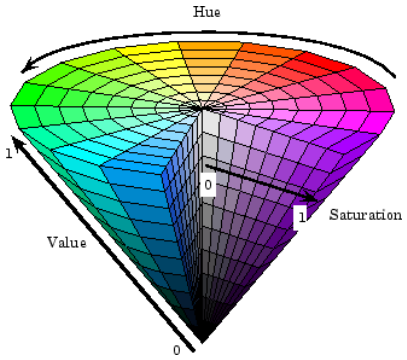
Terdapat beberapa fungsi pada *opencv* yang digunakan untuk mengkonversikan citra berwarna ke citra *grayscale*, diantaranya dengan menggunakan fungsi cvLoadImage() dengan *flag iscolor* diisikan dengan CV_LOAD_IMAGE_GRAYSCALE, dengan fungsi tersebut semua jenis citra akan dikonversikan kedalam sebuah citra *grayscale* 8bit. Selain cvLoadImage() dan imread() fungsi lain yang digunakan untuk mengkonversi citra berwarna menjadi citra grayscale adalah fungsi cvCvtColor() dengan struktur fungsi cvCvtColor(cons CvArr* src, CvArr* dst, Int code);

Di mana src adalah pointer Iplimage atau Mat citra yang ingin dikonversi kedalam *colorspace grayscale*, dst adalah pointer IplImage atau Mat citra yang dibuat untuk menyimpan nilai hasil pengubahan, dan code adalah *code* yang dipilih untuk jenis konversi yang diinginkan, untuk mengkonversi dari citra berwarna ke citra *grayscale* maka *code* yang dituliskan adalah CV_BGR 2GRAY. Fungsi ini digunakan untuk membuat struktur IplImage atau Mat baru untuk menyimpan citra hasil konversi adalah cvCreateImage() untuk C, untuk C++ tidak diperluka fungsi ini karena sudah dibuat kedalam sebuah Mat dst.

Di mana cvSize merupakan ukuran *width* dan *height* dari citra yang diinginkan, depth adalah kedalam warna dan *channel* adalah jumlah *channel* yang diinginkan. Jika digunakan untuk menyimpan hasil pengubahan dari citra BGR menjadi citra *grayscale* maka depth diisikan menggunakan integer 8 yang melambangkan kedalaman warna 8bit dan

channel diisikan dengan integer 1 yang melambangkan 1 *channel*. Berikut contoh program untuk mengkonversi citra berwarna ke *grayscale* menggunkan fungsi cvCvtColor():

Selain menggunakan *colorspace grayscale*, Opencv memberikan beberapa fungsi seperti CV_BGR2HSV, yang akan mengubah citra kedalam sebuah dasar penggunaa warna *Hue Saturation Value*. Untuk pemodelan warna HSV digunakan sebuah corong yang akan diperjelas pada gambar 2.10



Gambar 2.10 Pemodelan warna HSV

Untuk algoritme pengubahan nilai dari RGB menjadi nilai HSV adalah sebagai berikut :

$$H = \tan\left(\frac{3(G-B)}{(R-G)+(R-B)}\right) \qquad (2,4)$$

$$S = 1 - \left(\frac{\min(R,G,B)}{V}\right) \qquad (2,5)$$

$$V = \frac{R+G+B}{3} \qquad (2,7)$$

Parameter yang digunakan adalah:

- H = Nilai *Hue* dalam bentuk angka
- S = Nilai *Saturation* dalam bentuk persentase

- V = Nilai *Value* dalam bentuk persentase
- G = Nilai besaran *Green*
- R = Nilai besaran *Red*
- B = Nilai besaran *Blue*

Agar lebih mudah dipahami dengan menggunakan fungsi pada Open CV yaitu cv::cvtColor(InputArray src, OutputArray dst, CV_BGR2HSV). Pada HSV, nilai *Hue* dinyatakan sebagai nilai warna secara decimal untuk menetukan pelabelan warna, *Saturation* dinyatakan sebagai tingkat kemurnian warna, *Value* dinaytakan sebagai nilai kecerahan warna, dengan memiliki *range* dari 0 – 100%.

2.4.3 *Threshold*

Fungsi *threshold* pada *opencv* adalah fungsi yang digunakan untuk mengeliminasi *pixel-pixel* yang berada pada nilai dibawah atau diatas nilai batas ambang tertentu, fungsi ini sangat berguna untuk membuang *pixel-pixel* citra yang tidak diingikan atau *background*. Terdapat dua macam fungsi *threshold* pada opencv yaitu cvThreshold() dan cvAdaptiveThreshold(), perbedaan antara keduanya adalah metoda penentuan nilai yang menjadi batas ambang atau nilai *threshold*, kedua fungsi *threshold* tersebut hanya dapat diaplikasikan pada citra yang memiliki satu *channel* atau citra *grayscale*.

Fungsi cvThreshold() memberikan nilai ambang batas tertentu dengan memberikan nilai integer pada salah satu *flag* di fungsi cvThreshold() keuntungan dari fungsi ini adalah nilai *threshold* yang tidak berubah pada seluruh nilai *pixel* citra dan kelemahannya nilai ambang batas tidak dapat beradaptasi pada tingkat iluminasi citra sehingga proses pembeda antara objek dan *background* pada citra tidak maksimal untuk citra yang memiliki tingkat iluminasi tidak merata. Struktur pada fungsi cvThreshold sebagai berikut:

double cvThreshold(CvArr* src, CvArr* dst, double threshold, double max_value, int threshold_type);

Terdapat 5 flag yang dimiliki fungsi cvThreshold, yaitu *flag* src yang merupakan *flag* berisi nama *pointer* struktur IplImage citra yang akan dilakukan operasi *threshold*, kemudian *flag* dst yang merupakan *flag* yang berisi nama *pointer* struktur IplIlmage citra yang menampung hasil

dari operasi *threshold*, kemudian *flag threshold* yang merupakan *flag* berisi nilai ambang batas yang ingin diterapkan pada citra, kemudian *flag* max_value yang merupakan *flag* yang berisi nilai yang diberikan pada citra hasil selain nilai 0 yang tergantung dari tipe *threshold* yang dipilih pada *flag* terakhir, dan yang terakhir adalah *flag* threshold_type yang berisikian nilai integer atau nama tipe *threshold* yang ingin digunakan. Terdapat 5 pilihan tipe *threshold* yang dapat dipilih dan memiliki karakteristik yang berbeda-beda.

Jika dipilih tipe CV_THRESH_BINARY maka jika nilai *pixel* citra src_i lebih besar dari nilai ambang batas maka nilai *pixel* citra dst_i sama dengan nilai maksimum yang telah ditentukan dan jika nilai *pixel* citra src_i kurang dari nilai ambang batas maka nilai *pixel* citra dst_i sama dengan 0 atau hitam. Jika dipilih tipe CV_THRESHOLD_BINARY_ INV maka hasil pada citra dst_i berkebalikan dengan tipe CV_ THRESH_BINARY yaitu jika nilai *pixel* citra src_i lebih besar dari nilai ambang batas maka nilai *pixel* citra dst_i sama dengan 0 dan jika nilai *pixel* citra src_i kurang dari nilai ambang batas maka nilai *pixel* citra dst_i sama dengan nilai maksimum yang telah ditentukan sebelumnya. Jika dipilih tipe CV_THRESH_TRUNC maka jika nilai *pixel* citra src_i lebih besar dari nilai ambang batas maka nilai *pixel* citra dst_i sama dengan nilai maksimum yang ditetapkan dan jika nilai *pixel* citra src_i kurang dari nilai ambang batas maka nilai *pixel* citra dst_i sama dengan nilai *pixel* citra src_i atau tidak berubah.

Jika dipilih tipe CV_THRESH_ TOZERO_INV maka jika nilai *pixel* citra src_i lebih besar dari nilai ambang batas maka nilia *pixel* citra dst_i sama dengan 0 dan jika nilai *pixel* citra src_i kurang dari nilai ambang batas maka nilai *pixel* citra dst_i sama dengan nilai *pixel* citra src_i atau tidak berubah. Jika dipilih tipe CV_THRESH_TOZERO maka jika nilai *pixel* citra src_i lebih besar dari nilai ambang batas maka nilai *pixel* citra dst_i sama dengan nilai *pixel* citra src_i atau tidak berubah dan jika nilai *pixel* citra src_i kurang dari nilai ambang batas maka nilai *pixel* citra dst_i sama dengan 0 atau hitam.

2.4.4 *Morphology*

Morphology merupakan sebuah proses transformasi citra yang memiliki dua fungsi dasar yaitu *dilate* dan *erode*. Operasi *dilate* adalah operasi konvolusi antara suatu citra dengan sebuah *kernel* yang memiliki titik tengah. Operasi *dilate* dilakukan dengan memindai seluruh *pixel* pada citra dengan *kernel* yang sudah diberikan, dan menggantikan nilai

pixel yang dipindai oleh *kernel* dengan nilai maksimal yang didapat dari nilai *pixel* sekitar dari titik tengah *kernel*, sehingga operasi *dilate* menghasilkan daerah terang lebih luas. Dengan hal ini sebuah citra yang memiliki luasan *pixel* yang kecil akan diperbesar dan dibuat lebih jelas. Operasi *dilate* dapat diterapkan pada citra untuk memperjelas sebuah citra yang meiliki garis atau luasan *pixel* kecil.

Operasi *erode* adalah operasi kebalikan deri operasi *dilate*, hanya nilai *pixel* citra yang dipindai titik tengah *kernel* digantikan dengan nilai minimum *pixel* yang bedara disekitar titik tengah *kernel*, sehingga citra yang dihasilkan lebih banyak warna gelap. Operasi pada *erode* akan menghilangkan *noise* pada citra jika luasan *pixel* yang dicari lebih besar dari nilai kernel pada fungsi *erode*. Jika diterapkan pada citra *binary* maka operasi *dilate* dan *erode* sangat efektif untuk menghilangkan *noise* yang tidak diinginkan pada citra.

$$dilate(x,y) = [\min(x',y')\, kernel]\, src(x+x', y+y') \qquad (2,8)$$

$$erode(x,y) = [\max(x',y')\, kernel]\, src(x+x', y+y') \qquad (2,9)$$

Struktur fungsi *dilate* dan *erode* sebagai berikut:

void cvDilate(IplImage* src, IplImage* dst, IplConvKernel* B = NULL, Int iterations = 1);

void cvErode(IplImage* src, IplImage* dst, IplConvKernel* B = NULL, Int iteration = 1);

Baik fungsi cvDilate() maupun cvErode() memiliki 4 parameter, parameter pertama adalah *flag* yang berisi nama *pointer* struktur IplImage atau Mat citra *operand*, parameter yang kedua adalah *flag* yang berisi nama *pointer* struktur IplImage citra hasil operasi, *flag* yang ketiga atau *flag* B adalah *flag* yang berisi nama *pointer* struktur *kernel* yang digunakan, jika diisi dengan NULL (0) maka *kernel* yang digunakan adalah *kernel* bawaan yang memiliki luas 3x3 *pixel*. Dan *flag* yang ke empat adalah *flag* yang berisi nilai integer yang memberikan instruksi berapa kali iterasi yang diingninkan, jika tidak diisi maka iterasi akan diisi dengan nilai bawaan yaitu 1.

Pada operasi *morphology* dapat diatur berapa nilai *pixel* yang akan diperbesar dan nilai *pixel* yang akan diperkecil dengan menggunakan

fungsi cv::getStructuringElement(), yang akan dicontohkan pada code berikut:

```
getStructuringElement(MORPH_RECT, Size(3, 3));
```

2.4.5 *Region Of Interest*

Region Of Interest merupakan sebuah metode pengambilan citra yang dimana pengambilan citra tersebut telah melalui sebuah proses penyimpanan nilai *vector* atau dapat menetukan titik koordinat x dan y. OpenCV tidak memberikan fungsi untuk memanggil ROI, tetapi kita dapat menggunakan beberapa fungsi seperti cv::Rect rect untuk menentukan bahwa ROI akan berbentuk sebuah kotak, pada fungsi ini terdapat satu parameter yaitu rect dimana parameter ini dapat diisikan dengan Rect(koordinat x, koordinat y, *width*, *height*). Fungsi lainnya adalah untuk menggunakan *Hough Transformation*. Untuk penggunaan code ROI yang berbentuk kotak:

Untuk C++:

```
Mat image;
Mat Image ROI;
Rect ROI = rect(x,y,Size());
Image ROI = image(ROI);
```

Pada program ini untuk nilai X dan Y dapat kita tentukan koordinatnya di citra yang akan diproses, untuk Size() akan diisikan sebuah ukuran *width* dan *height* dari ROI citra yang ingin kita ambil. Untuk menyimpan citra setelah melalui proses pemotongan dapat digunakan fungsi imwrite().

2.4.6 *Contour*

Contour adalah area pada citra yang memiliki nilai *pixel* yang sama dengan nilai *pixel* sekitarnya, dalam *opencv* terdapat fungsi yang digunakan untuk menemukan *contour* yang terdapat pada citra, fungsi tersebut adalah cvFindContours() yang akan memberikan nilai integer jumlah *contour* yang ditemukan pada citra, fungsi cvFindContours() memiliki struktur fungsi:

Untuk C++ :

findConotours(InputOutputArray image, OutputArrayOfArays contours, OutputArray hierarchy, int mode, int method, Point offset=Point());

Fungsi cvFindContours() memiliki 6 parameter yaitu img, storage, first Contour, headerSize, mode dan method. *Flag* img adalah *flag* yang berisi nama *pointer* struktur IplImage citra yang akan dicari jumlah *contour*-nya, citra tersebut harus memiliki kedalaman warna 8 bit dan satu *channel*. Parameter yang kedua adalah *storage* yang merupakan struktur CvMemStorage yang digunakan untuk mengalokasikan nilai dari *contour* yang ditemukan pada citra, struktur tersebut harus di alokasikan menggunkan fungsi cvCreateMemStorage(). Pada C++ akan dibutuhkan deklarasi sebuah vetor untuk penyimapanan nilai kareana menggunakan cv::Mat. Parameter yang ketiga adalah firstContour yang merupakan *flag* yang berisi nama *pointer* struktur CvSeq yang merupakan struktur yang mengalokasikan memori yang digunakan menyimpan *sequence*, struktur CvSeq harus dideklarasikan terlebih dahulu pada bagian awal program. Parameter yang ke empat adalah *flag* headerSize yang merupakan *flag* yang berisi informasi ukuran objek yang akan dialokasikan, *flag* ini dapat diisi dengan sizeof(CvContour). Parameter yang ke lima adalah mode yang merupakan *flag* yang berisi pilihan susunan yang digunakan untuk menyusun hasil contour yang ditemukan pada citra, terdapat empat pilihan yaitu CV_RETR-EXTERNAL, CV_RETR_LIST, CV_RETR_CCOMP dan CV_RETR_TREE. Parameter yang terakhir adalah *flag* method yang berisi pilihan metoda yang ingin digunakan, terdapat 5 pilihan metode yang dapat dipilih yaitu CV_CHAIN_CODE, CV_CHAIN_APPROX_NONE, CV_CHAIN_AP PROX_SIMPLE, CV_CHAIN_APPROX_TC89_L1 atau CV_CHAIN_ APPROX_TC89_KCOS dan CV_LINK_RUNS.

Pemrograman untuk memanggil atau mencari sebuah *contour* dapat dilakukan sebagai berikut:

```
Mat canny_output;
vector<vector<Point> > contours;
vector<Vec4i> hierarchy;

Canny( src_gray, canny_output, thresh, thresh*2, 3 );
```

```
findContours( canny_output, contours, hierarchy,
CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE, Point(0, 0) );
Mat drawing = Mat::zeros( canny_output.size(), CV_8UC3 );
for( int i = 0; i< contours.size(); i++ ){
Scalar color = Scalar( rng.uniform(0, 255), rng.uniform(0,255),
rng.uniform(0,255) );
drawContours( drawing, contours, i, color, 2, 8, hierarchy, 0, Point()
);
}
```

Pada program ini bergungsi untuk mencari sebuah *contour* dan selanjutnya menggambarkan *contour*. Pada bagian pertama adlah mendeklarasi Mat untuk menyimpan citra atau memanggil sebuah citra. Pada bagian kedua adalah meyimpan niali *contour* menjadi sebuah vector, agar nilai *contour* dapat dengan mudah dipanggil dan disimpan kedalam sebuah vector. Bagian ketiga adalah mencari *contour* yang akan disimpan kedalam nilai vector *contour*. Bagian keempat adalah menggambarkan garis *contour*.

BAB III

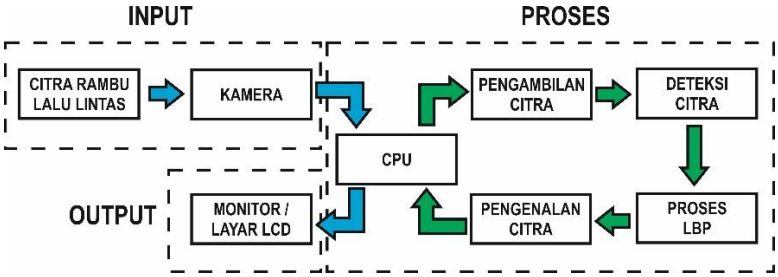
PERANCANGAN SISTEM

Secara garis besar tugas akhir ini, terdapat 2 (dua) bagian utama penyusunnya, yaitu perangkat lunak dan perangkat keras. Pada perangkat keras menggunakan 2 (dua) komponen utama yaitu kamera untuk mengambil citra dan *mini PC* untuk melakukan pemrosesan citra digital agar dapat dikenali. Untuk penggunaan kamera menggunakan kamera yang memiliki minimal resolusi sebesar 1080HD *pixel*, sedangkan pada *mini PC* digunakan *software* Microsoft Visual Studio 2015 dengan menggunakan *library* dari OpenCV.

Pada sistem perangkat lunak digunakan Microsoft Visual Studio 2015 untuk pemrogramannya dan OpenCV sebagai *library* untuk memproses citra rambu lalu lintas. Pada sistem di perangkat lunak terdapat beberapa bagian yaitu pengambilan citra, *thresholding* dan *morphology*, deteksi gambar dan *Region Of Interest* untuk mendeteksi rambu lalu lintas. Sedangkan untuk mengenali citra rambu lalu lintas digunakan program *Local Binary Pattern*, komparasi *histogram* dengan *histogram data learning*. Dari keseluruhan sistem ini dibangun untuk dapat mendeteksi dan mengenali citra rambu lalu lintas.

Pada gambar 3.1 akan dijelaskan mengenai blok diagram dari sistem yang akan dirancang, tanda panah biru menunjukan proses secara langsung antar komponen, untuk tanda panah hijau merupakan proses yang ada pada CPU yang dilakukan oleh program. Bagian pertama adalah bagian *input*. Pada bagian ini digunakan kamera sebagai pengambil dari citra rambu lalu lintas. Citra rambu lalu lintas akan di ambil dan disimpan kedalam data digital dengan menggunakan kamera, setelah itu akan diproses CPU.

Bagian kedua merupakan proses, proses akan dilakukan pada CPU. Pada CPU akan digunakan program Visual Studio dengan *library* dari OpenCV. Pada program akan dilakukan pengambilan citra dan pengenalan citra rambu lalu lintas. Program pada sistem ini akan dijelaskan lebih pada bab ini. Setelah dikenali citra rambu lalu lintas, akan ditampilkan dan disimpan dalam data di CPU. Bagian ketiga adalah *output*, data hasil dari pengenalan akan ditampilkan ke *monitor* atau layer LCD, terdapat *file* data hasil pengenalan yang akan dapat di akses untuk dilakukan proses selanjutnya.



Gambar 3.1 Blok Diagram Sistem

3.1 Perancangan perangkat keras

Pada perancangan perangkat keras, terdapat 2 (dua) komponen utama untuk menjalankan prosess dari sistem ini. Desain alat yang digunakan seperti pada gambar 3.1. Pengambilan citra rambu lalu lintas menggunakan kamera yang diambil secara *live streaming*, sedangkan pemrosesan citra dengan menggunakan *mini PC*.

Desain pada sistem perangkat keras ini, kamera digunakan untuk mengambil citra secara terus menerus, setelah itu citra akan diproses pada *software* yang terdapat pada *mini PC*. Pada *mini PC* terdapat *software* untuk memproses citra yang didapat dengan menggunakan *library* pemrosesan citra.

Alat yang digunakan dalam penelitian ini merupakan:

1. Kamera
   - 1.1. Nama : Logitech C270
   - 1.2. Video : *High Definition* 720 *pixel*
   - 1.3. Koneksi : USB 2.0

2. CPU
   - 2.1. Jenis : Lenovo DALZ2AMB8F0
   - 2.2. Prosesor : Intel *Core i5-3210M*
   - 2.3. VGA : NVIDIA GT650M
   - 2.4. RAM : 8 GB
   - 2.5. Koneksi : 2 USB 2.0 dan 2 USB 3.0
   - 2.6. Daya : 240 Volt 1.5 Amper

2.7. OS : Windows 8.0 64-*bits*
2.8. Frekuensi : 2.50 GHz

3. Monitor
   3.1. Jenis : Lenovo LCD Monitor
   3.2. Tipe Layar : LED
   3.3. Ukuran : 14" *inches*
   3.4. Resolusi : 1366 x 768 *pixels* HD

3.1.1 Kamera

Kamera pada sistem perangkat keras ini berfungsi untuk mengambil citra secara terus menerus atau *live streaming*. Citra yang dibaca memiliki besar 1080HD *pixel* dikarenakan semakin besar *pixel* dari kamera maka akan didapatkan citra yang memiliki kerapatan *pixel* yang lebih kecil dan ketajaman citra yang lebih baik untuk dilakukan proses pengenalan dan proses *thresholding*.

Proses pada kamera akan mengambil gambar secara *frame per frame*. Setiap *frame* yang diambil akan selalu diproses untuk dikenali dan diperlihatkan citra yang diambil dengan cepat. Standard pada kamera *webcam* menggunakan 30 *Frame Per Second* (FPS), dimana selama 1 detik pada *live stream* akan dihasilkan 30 citra. Untuk pengambilan citra secara *live stream* akan menggunakan memori pemrosesan yang sangat besar. Pengambilan citra secara *live stream* sangant dibutuhkan karena citra rambu lalu lintas yang diambil secara langsung dan data yang dikenali waktunya akan sangat cepat.

3.1.2 *Mini PC*

Mini PC digunakan untuk memproses citra yang diambil dari kamera yang kemudian akan dikenali oleh sistem perangkat lunak yang terdapat pada *mini PC*. Pada *mini PC*, digunakan dasar pemrograman dengan bahasa C/C++ yang diambil dari Visual Studio 2015. Untuk pemrosesan digunakan *mini PC* yang memiliki *system requirement* untuk menjalankan program Microsoft Visual Studio 2015, dengan minimum *processor* core i3 dan *RAM* sebesar 4GB.

Mini PC akan memerintahkan Kamera untuk mengambil citra secara *live stream* melalui *Universal Serial Bus* (USB). Kamera yang terkoneksi ke *mini PC* akan mengambil citra yang kemudia akan di proses pada perangkat lunak. *Mini PC* akan terus memanggil citra dari kamera secara

continuously. Hasil citra yang didapat dan hasil pengenalan akan disimpan pada *directory* program Microsoft Visual Studio.

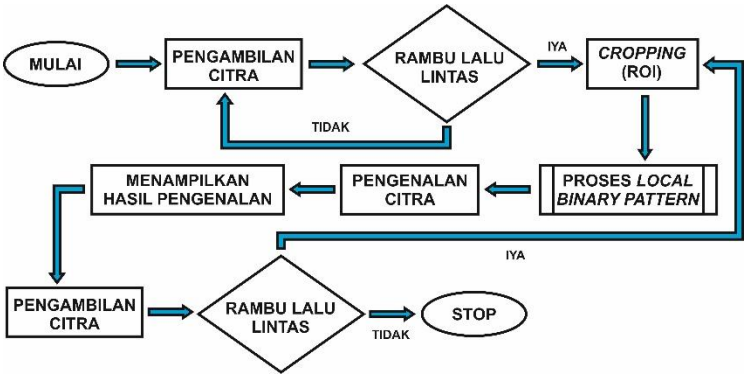
3.2 Perancangan perangkat lunak

Pada sistem perangkat lunak digunakan pemrograman dari Microsoft Visual Studio yang mengambil *library* dari OpenCV untuk dilakukan proses pengolahan data citra. Pada Visual Studio diperlukan *compile library* OpenCV sehingga bahasa pemrograman yang digunakan dengan OpenCV. Pada sistem perangkat lunak dilakukan 3 (tiga) proses utama yang pertama adalah proses pengambilan dan pengenalan citra rambu lalu lintas, kedua adalah proses pengubahan citra dengan metode *Local Bianry Pattern* yang kemudian akan diambil nilai *histogram* dari citra yang sudah dikenali, ketiga adalah proses pengenalan citra rambu lalu lintas yang sudah melalui proses *Local Binary Pattern* dan akan dikomparasi dengan *histogram* dari *data learning*.

Pada perancangan pengenalan rambu lalu lintas pertama kali adalah menuliskan program untuk mengambil citra dari kamera secara *live stream*, kemudian citra yang diambil akan disimpan dalam *frame* untuk dicari apakah ada citra rambu lalu lintas di dalam *frame*. Setelah mengambil citra dalam *frame* akan dilakukan proses penyimpanan frame menjadi sebuah citra. Dimana citra yang diambil akan diproses menjadi sebuah citra *threshold* yang nantinya akan dikenali dengan menggunakan ROI. Pada proses *thresholding*, digunakan *range* untuk mengetahui warna yang akan dikenali. Dengan menggunakan *range* warna HSV untuk merah dan biru, maka didapatkan sebuah citra yang dapat dikenali dengan menggunakan *contour*.

Setelah didapatkan *threshold* pada citra, maka citra akan di ubah dengan menggunakan *morphology*. Pada *morphology*, citra dari *threshold* akan difilter agar citra lebih mudah dikenal dan mengurangi *noise* dengan cara mengubah ukuran citra. Untuk pertama citra akan dilakukan proses *erode*, dimana untuk ukuran *pixel* yang memiliki besar kurang dari 3 x 3*px* akan diperkecil sehingga *noise* pada pengenalan range akan diperkecil atau dihilangkan. Setelah dilakukan proses *erode* maka akan dilakukan proses *dilate*, dimana proses ini akan membesarkan pixel yang berukuran lebih dari 8 x 8*px* akan diperbesar. Proses selanjutnya adalah akan dilakukan *boundingbox* untuk mendapatkan nilai *Region Of Interest*. Pada *Region Of Interest* citra yang didapatkan akan dipotong dan akan disimpan dalan sebuah *file.jpg*. Proses selanjutnya adalah citra yang didapatkan dari hasil *Region Of Interest* akan di ubah menjadi citra

grayscale yang akan di proses dengan menggunakan algoritme *Local Binary Pattern*. Setelah citra di proses dengan menggunakan algoritme *Local Binary Pattern*, akan didapatkan nilai *histogram* untuk di komparasi dengan *histogram data learning*. Hasil kecocokan komparasi *histogram* akan disimpan kesebuah *file* sebagai *back up* dan *data recorder*. Setelah proses berhasil maka akan diambil citra rambu lalu lintas terbaru untuk dikenali.



Gambar 3.2 *Flowchart* pengenalan rambu lalu lintas.

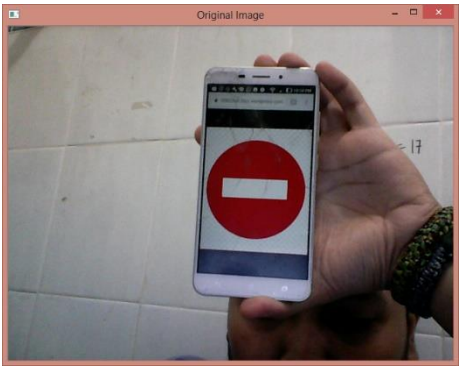
3.2.1 Pengambilan citra

Pengambilan citra dengan menggunakan metode *live stream*, dimana setiap citra yang didapat akan disimpan ke sebuah *frame* untuk dapat diproses. Data dari *frame* diambil langsung dari kamera dan disimpan ke sebuah *matrix* nilai berupa cv::Mat pada program.

Proses pengambilan data citra akan digunakan fungsi *class* dari OpenCV yaitu VideoCapture(const String &filename). Pada fungsi ini akan diambil citra secara langsung dari kamera yang akan disimpan ke Mat. Pada VideoCapture, akan ditentukan parameter pengambilan citra darikamera dengan menggunakan fungsi set(int propld, double value), fungsi ini dapat mengatur besarnya aspek *pixel* dari *width and height window* kamera. Untuk menggunakan kamera maka digunakan fungsi dari class VideoCapture berupa fungsi open(const String &filename), fungsi ini akan memanggil kamera yang telah tersimpan pada *mini PC*,

nilai yang digunakan bergantung dari jumlah banyaknya kamera yang digunakan.

Untuk membuat kamera untuk selalu mengambil citra maka diperlukan while (1). Selama program while (1) dipanggil kamera akan terus mengambil citra secara *continuously*. Untuk menyimpan data citra dari kamera maka diperlukan fungsi dari *class* VideoCapture berupa fungsi read(OutputArray image), fungsi ini untuk menyimpan hasil pengambilan dari kamera menjadi sebuah *array* pada Mat. Pada pengambilan citra melalui kamera akan di tentukan besar *window* dengan ukuran 1080 x 720*px*.



Gambar 3.3 Citra yang didapatkan melalui proses pengambilan citra dari kamera.

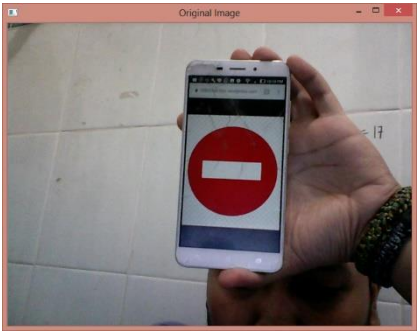
3.2.2 *Thresholding* dan *Morphology*

Thresholding merupakan sebuah metode untuk mengubah sebuah citra menjadi citra hitam dan putih dengan parameter yang sudah ditentukan. Untuk memanggil fungsi cv::threshold (InputArray src, OutputArray dst, double thresh, double maxval, int type). Niali dari *threshold* dapat bervariasi dari 0 sampai 255, jika memasukan nilai 0 sampai 255 maka citra yang didapatkan akan menjadi putih dikarenakan nilai yang mendekati besaran dari nilai thresh akan dibuat menjadi angka 0 yang berarti hitam, jika nilai berada lebih dari nilai thresh maka akan diubah menjadi nilai maxval 255 yaitu putih.

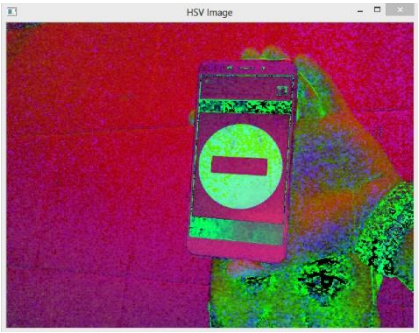
Pada fungsi *thresholding* citra yang didapat akan diubah menjadi sebuah citra dengan menggunakan *colorspace*. *Colorspace* yang tersedia

pada OpenCV adalah *Blue Green Red* (BGR), *Lab Colorspace*, *Hue Saturation Value* (HSV), *Grayscale*, dan *YCrCb Colorspace*. Cara mengubah *colorspace* dari citra dengan menggunakan fungsi cv::cvtColor(InputArray src, OutputArray dst, int code, int dstCn = 0). Pada OpenCV disediakan fungsi CV_BGR2RGB untuk mengubah BGR mejadi RGB, CV_BGR2GRAY untuk mengubah BGR menjadi *Grayscale*, CV_BGR2HSV untuk mengubah BGR menjadi HSV.

Proses ini menggunakan metode *thresholding* dengan mengambil *range* warna HSV sebagai nilai acuan untuk menentukan warna yang diinginkan. Jika warna selain dari *range* maka akan dibuat menjadi warna hitam, sedangkan warna yang berada dalam *range* akan menjadi putih, seperti yang dicontohkan pada gambar 3.4. pada gambar yang digunakan mengambil range warna merah dengan fungsi cv::inRange (InputArray src, InputArray lowerb, InputArray upperb, OutputArray dst), dimana menggunakan fungsi Scalar untuk menentukan nilai HSV pada InputArray lowerb dan InputArray upperb. Untuk merah *range* terendah dimulai dari Hue = 0 , Sat = 100, Val = 100, *range* terbesar dari Hue = 10, Sat = 255, Val = 255.



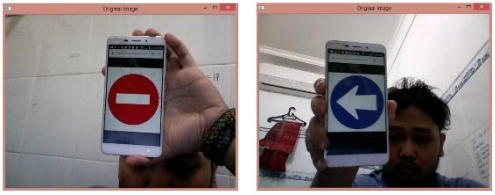
(a)



(b)

Gambar 3.4 *Threshold* dengan warna HSV.
(a) gambar sebelum di *threshold.* (b) gambar sesudah di *threshold* HSV.

Penggunaan *threshold* dengan metode HSV dapat membantu untuk menetukan warna objek secara spesifik. Dikarenakan rambu lalu lintas memiliki warna dasar merah dan biru dan kungin maka diambil *range* nilai HSV untuk setiap warna sehingga hanya warna merah, biru dan kuning yang dapat dideteksi. Seperti pada gambar 3.5 diamana hanya objek berwarna merah biru dan kuning yang terdeteksi.



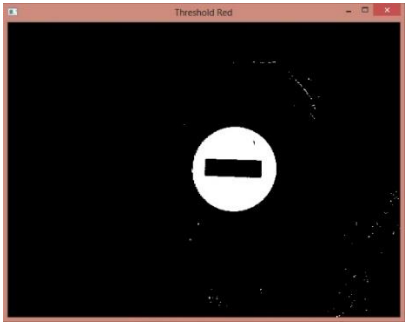
(a)

(b)

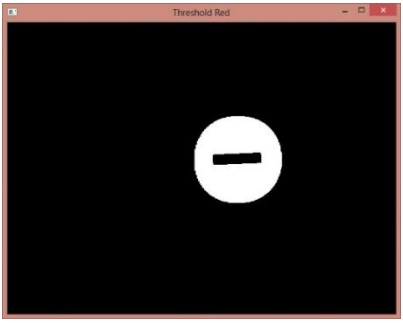
Gambar 3.5 *Threshold* warna merah, biru.
(a) gambar awal. (b) gambar setelah melalui proses *threshold.*

Morphology digunakan untuk menghilangkan noise pada citra, *isolation* citra menjadi sebuah individu, mendapatkan intesitas pada gambar. Pada *morphology* terdapat dua fungsi utama yaitu fungsi *erode* dan *dilate*, setiap fungsi dapat digabungkan untuk mencegah adanya noise pada citra setelah melalui proses *threshold*. *Erode* dan *dilate* menggunakan nilai *kernel* 3 x 3*px*.

Pada *morphology* jika nilai pixel kurang dari 20 x 20*px* akan danggap sebagai noise. Proses pertama dengan menggunakan *erode* dengan fungsi cv::erode(InputArry src, OutputArray dst, getStructuringElement(MORPH_RECT, Size(3, 3))). Fungsi ini digunakan jika nilai luasan *pixel* kurang dari 3 x 3*px* akan diperkecil dan kemudian akan dilakukan proses *dilate* setelah *erode*. Dengan menggunakan cv::dilate(InputArry src, OutputArray dst, getStructuringElement(MORPH_RECT, Size(8, 8))). Hasil citra *threshold* yang telah melalui proses *morphology* seperti pada gambar 3.6.



(a)



(b)

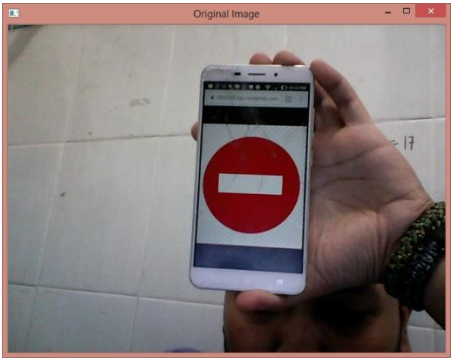
Gambar 3.6 Citra *threshold* setelah melalui proses *morphology.*
(a) citra *threshold* sebelum *morphology*. (b) citra *threshold* setelah *morphology.*

3.2.3 *Region Of Interest*

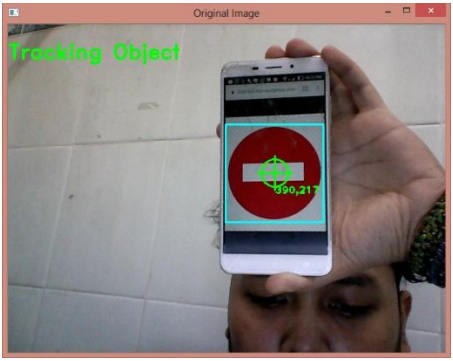
Pada proses ini citra yang telah di *threshold* akan didapatkan *contour*. Dengan menggunakan metode *moment* untuk mendapatkan nilai koordinat x dan y dari *contour*. Untuk nilai *contour* disimpan di sebuah vector dengan menggunakan fungsi definisi vector <point> contour. Nilai *contour* akan disimpan sehingga bias didapatkan besar luas area dari nilai *contour*. Rumus untuk menghitung nilai x = moment.m01 / area, untuk mendapatkan nilai y = moment.m10 / area. Setelah di dapatkan nilai koordinat dari *contour* maka akan di gunakan fungsi ROI.

Untuk memanggil fungsi ROI maka dibuatkan sebuah definisi bahwa ROI yang digunakan berbentuk sebuah kotak atau *rectangle*, dengan mendefinisikan Rect ROI serta untuk image dapat di deteksi dan *tracking*, maka didefinisikan fungsi Rect objectBoundingRectangle = Rect(0, 0, 0, 0). Setelah dinyatakan fugsinya maka dapat ditentukan besar nilai ROI x dan ROI y dengan menggunakan fungsi ROI.x = xPost - (objectBoundingRectangle.height / 2) dan fungsi ROI.y = yPost - (objectBoundingRectangle.height / 2). Pada fungsi ini digunakan hanya tinggi dari objectBoundingRectangle dikarenakan bentuknya yang akan menjadi kotak. Setelah mendapatkan koordinat ROI x dan ROI y, maka fungsi ROI.width = objectBoundingRectangle.height untuk mendapatkan nilai lebar dari ROI untuk nilai tinggi dari ROI maka digunakan fungsi

ROI.height = objectBoundingRectangle.height. Maka akan didapatkan hasil seperti pada gambar 3.7, dimana hanya pemotongan citra menjadi ukuran objek yang diinginkan.



(a)



(b)

Gambar 3.7 Citra yang sudah di potong untuk siap di proses. (a) citra asli. (b) citra setelah melalui proses ROI.

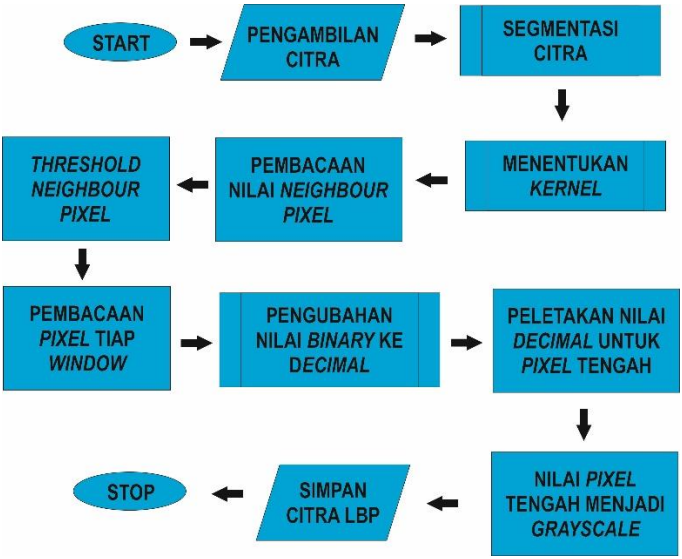
3.2.4 *Local Binary Pattern*

Pada proses pengubahan citra menjadi sebuah citra LBP, akan dilakukan dengan menggunakan algoritma dasar dengan melakukan pembacaan nilai *pixel* dengan menggunakan *kernel* 3x3. Pergerakan pembacaan nilai *pixel* dimulai dari ujung kiri atas menuju ujung kanan

bawah, dan bergeser 1 *pixel* kebawah. Pada Kernel akan dilakukan pembacaan nilai *pixel* pada *window*, dan nilai *pixel* ditengah akan dikomparasi dengan nilai *pixel* pada *window* sekitar yang disebut dengan *neighbour*.

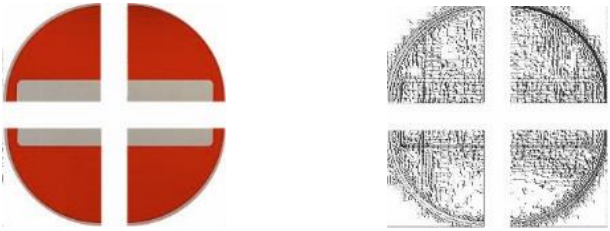
Setelah dilakukan pembacaan nilai, maka akan dilakukan proses *threshold* dimana jika nilai *neighbour pixel* lebih besar dari nilai *pixel* pada *window* ditengah, maka nilai *neighbour pixel* akan menjadi 1, dan jika nilai *neighbour pixel* lebih kecil dari nilai *pixel* pada *window* ditengah, maka *neighbour pixel* disekitarnya akan menjadi 0. Setelah dilakukan pengubahan nilai menjadi 1 atau 0, maka nilai *pixel* akan dibaca secara arah jarum jam. Nilai *pixel* yang didapatkan berupa sebuah nilai *binary* yang akan diubah menjadi sebuah nilai decimal dan akan disimpan pada nilai *pixel* pada *window* di tengah. Nilai *pixel* akan menjadi sebuah nilai *grayscale*.

Untuk mendapatkan nilai *feature histogram* maka diperlukan segmentasi pada citra sebelum proses LBP. Fungsi dari segmentasi selain untuk menambahkan *feature histogram* adalah menambah tingkat akurasi pengenalan dari citra rambu lalu lintas. Pada program segmentasi, mengunakan cv::Rect sebagai penentu dari besarnya nilai segmentasi dengan menentukan koordniat x dan koordinat y, serta lebar dan panjang dari segmentasi. Penggunaan segmentasi pada citra tidak boleh terlalu kecil, jika terlalu kecil, maka nilai *feature histogram* akan lebih susah didapat dan nilai iterasi akan menjadi lebih kecil, sehingga LBP tidak dapat mengenali adanya perubahan nilai *pixel*. Pada gambar 3.8 akan dijelaskan *flowchart* dari proses *feature histogram*.



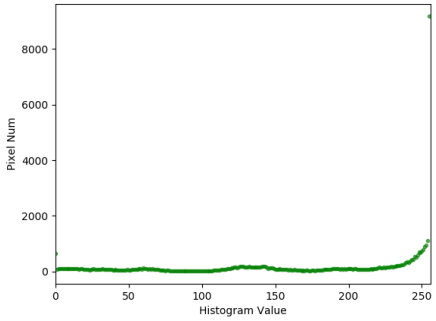
Gambar 3.8 Diagram alur dari *feature histogram* LBP

Hasil dari pengubahan citra menjadi sebuah citra LBP akan didapatkan sebuah citra yang memiliki tekstur khusus dimana hanya perubahan warna dan garis dari citra yang akan terbaca. Untuk mendapatkan *feature histogram*, diperlukan adanya segmentasi pada citra sebelum dilakukan pengubahan LBP pada citra. Pada gambar 3.9 citra yang disegmentasi akan diubah menjadi citra LBP. Gambar 3.10 merupakan perbandingan hasil dari *feature histogram* antara citra yang mengalami segmentasi dengan citra yang tidak disegmentasi.

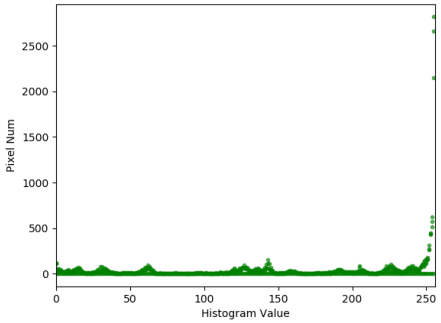


(a) (b)

Gambar 3.9 Citra proses LBP.
(a) citra sebelum proses LBP. (b) citra setelah proses LBP.



(a)



(b)

Gambar 3.10 Hasil dari nilai histogram pada citra LBP.

(a) Nilai histogram tanpa segmentasi. (b) Nilai histogram dengan 2x2 segmentasi

Pada program untuk melakukan proses pengubahan citra menjadi sebuah citra LBP seperti pada program dibawah.

```
for (int y = 0; y < ff; y++) {
for (int x = 0; x < ff; x++) {
int center = CV_IMAGE_ELEM(img_part, uchar, y, x);
lbp_nb[0] = CV_IMAGE_ELEM(img_part, uchar, y - 1, x - 1);
lbp_nb[1] = CV_IMAGE_ELEM(img_part, uchar, y - 1, x);
lbp_nb[2] = CV_IMAGE_ELEM(img_part, uchar, y - 1, x + 1);
lbp_nb[3] = CV_IMAGE_ELEM(img_part, uchar, y, x + 1);
lbp_nb[4] = CV_IMAGE_ELEM(img_part, uchar, y + 1, x + 1);
lbp_nb[5] = CV_IMAGE_ELEM(img_part, uchar, y + 1, x);
lbp_nb[6] = CV_IMAGE_ELEM(img_part, uchar, y + 1, x - 1);
lbp_nb[7] = CV_IMAGE_ELEM(img_part, uchar, y, x - 1);

for (int i = 0; i < 8; i++) {
if (lbp_nb[i] < center) {
lbp_nb[i] = 0;
}
else {
lbp_nb[i] = 1;
}
}
center = lbp_nb[7] * 128 + lbp_nb[6] * 64 + lbp_nb[5] * 32 +
lbp_nb[4] * 16 + lbp_nb[3] * 8 + lbp_nb[2] * 4 + lbp_nb[1] * 2 +
lbp_nb[0] * 1;
CV_IMAGE_ELEM(img_lbp, uchar, y - 1, x - 1) = center;
}
```

Program LBP ini menggunakan algoritma dasar pengubahan citra dengan menggunakan *kernel* 3x3. Pertama untuk menentukan pergerakan *kernel* maka akan dilakukan penentuan koordinat y dan kemudian akan dilakukan penentuan koordinat x, sehingga pergerakan *kernel* akan mulai dari ujung atas kiri ke ujung atas kanan, kemudian akan bergeser 1 *pixel* kebawah dari ujung kiri ke ujung kanan. Setelah menentukan pergerakan dari *kernel*, maka akan diambil nilai *pixel* dari *window* untuk dikomparasi

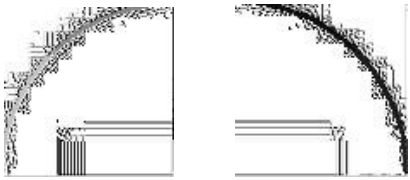
dengan nilai tengah dari *window*, dimulai dari *window* ujung kiri atas dan bergerak searah jarum jam.

Untuk mendapatkan nilai *pixel* ditengah dan *neighbour pixel*, pada program dilakukan dengan menggunakan cv:: CV_IMAGE_ELEM. Setelah dilakukan pembacaan nilai *pixel*, maka akan dilakukan komparasi nilai *neighbour pixel* dengan nilai *pixel* ditengah. Setelah semua nilai *neighbour pixel* telah didapatkan, maka akan diubah menjadi nilai *decimal*. Setelah dilakukan perhitungan, maka nilai pembacaan akan diubah menjadi nilai *pixel* tengah dengan menggunakan cv::CV_IMAGE_ELEM.

3.2.5 *Data learning*

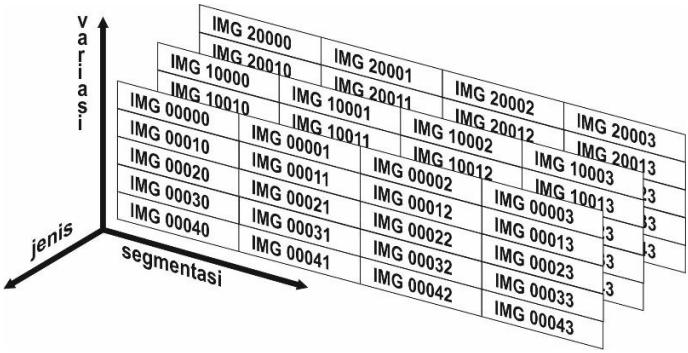
Data learning digunakan untuk mendapatkan nilai *data base* pengukuran *histogram* agar dapat dibandingkan dengan citra rambu lalu lintas yang didapatkan melalui kamera. Dari nilai *histogram data learning* dapat ditentukan citra rambu lalu lintas dikenali sebagai rambu lalu lintas tertentu pada proses selanjutnya.

Data learning yang digunakan adalah nilai *histogram* citra rambu lalu lintas yang telah melewati proses LBP dengan menggunakan sebanyak 5 jenis rambu lalu lintas dengan 10 variasi rambu lalu lintas sehingga keseluruhan terdapat nilai sebesar 50 *data learning*. Setiap citra rambu lalu lintas yang akan dikenali akan dilakukan proses segmentasi, proses segmentasi akan membantu komparasi citra *data* learning dengan citra yang akan dikenali. Segmentasi yang digunakan sebanyak 10x10, sehingga total *data learning* 5000 data. Untuk mendapatkan nilai kedekatan rambu lalu lintas, setiap citra rambu lalu lintas yang telah disegmentasi dan diubah kedalam LBP akan digabungkan menjadi sebuah nilai histogram kemudian di bandingkan dengan 50 citra *data learning* sehingga menghasilkan nilai terdekat dari rambu lalu lintas tersebut dengan nilai *histogram* pada citra *data learning*.



Gambar 3.11 Citra pada *data learning* (2x2 *subdivide*)

Pada proses pengenalan citra rambu lalu lintas akan ada proses pemanggilan *data learning* sebagai *data base* yang akan dikomparasi nilai *histogram*. Total citra *data learning* sebanyak 50 dan segmentasi tiap citra sebanyak 100 yang telah melalui prsoses LBP. Nilai *histogram data learning* akan disimpan kedalam sebuah *array* 3 dimensi dengan ukuran [5][10][100] untuk pelabelan dan menentukan banyak variasi dan segmentasi pada tiap citra *data learning*. Nilai *histogram* yang disimpan akan disimpan didalam aray sebesar [256]. Pada nilai *array* [5] digunakan untuk pengenalan jenis dari citra rambu lalu lintas. Pada nilai *array* kedua [10] untuk menentukan variasi dari setiap jenis rambu lalu lintas yang digunakan. Pada *array* ketiga [100] digunakan untuk segmentasi citra rambu lalu lintas. Jika digambarkan akan terlihat seperti gambar 3.12



Gambar 3.12 Ilustrasi nilai histogram *data learning* pada *array* 3 dimensi.

Pada gambar 3.11 nilai *data learning* di ilustrasikan sebagai nilai histogram setiap rambu lalu lintas yang disimpan dalam *array* 3 dimensi

dengan ukuran [5][10][100], dimana pada program akan dilakukan penulisan seperti berikut :

```
for (int a = 0; a < aa; a++) {
for (int b = 0; b < bb; b++) {
for (int c = 0; c < cc; c++) {
snprintf(data_learning, 1107, "LBP_DL/DL_SEG
%02d/LBP_DL_SEG/%01d%01d%03d.jpg",dd, a, b, c);
img_mat = imread(data_learning,
CV_LOAD_IMAGE_ANYCOLOR);
calcHist(&img_mat, 1, &channels, noArray(), hist_mat, 1,
&numBins, &ranges);
for (size_t i = 0; i < numBins; ++i) {
float val = hist_mat.at<float>(i);
datalearning[a][b][c][i] = val;
}
}
}
}
```

Pada program akan digunakan *integer* ‘a’ sebagai penentu jenis dan *integer* ‘b’ sebagai penentu variasi dari tiap jenis rambu lalu lintas dan *integer* ‘c’ untuk penentu jumlah dari segmentasi yang digunakan pada tiap citra untuk diambil nilai histogram. Untuk *integer* ‘i’ akan digunakan sebagai penyimpan nilai dari histogram citra yang telah disegmentasi.

3.2.6 Pengenalan rambu lalu lintas

Proses terakhir adalah pengenalan citra rambu lalu lintas dengan cara membandingkan nilai *histogram*. Fungsi calcHist(const Mat*images, int nimages, const int* channels, InputArray mask, OutputArray hist, int dims, const int* histSize, const float** ranges, bool uniform=true, bool accumulate=false). Fungsi ini untuk menghitung nilai dari citra yang akan dibuat komparasi pada proses selanjutnya. Komparasi citra rambu lalu lintas dari kamera dengan nilai *histogram data learning* menggunakan metode *Chi-square*, metode ini menggunakan komparasi dengan pendekatan nilai terkecil dari hasil perbandingan yang merupakan indikator citra yang memiliki nilai kesamaan *histogram* terdekat. Metode *Chi-square* dengan algoritme sebagai berikut:

$$chi\_square(h1, h2) = \sum_i \frac{(h1(i) - h2(i))^2}{h1(i) + h2(i)} \quad (3,1)$$

Di mana memiliki parameter:

- h1(i) = nilai *histogram* citra 1 ke i
- h2(i) = nilai *histogram* citra 2 ke i

Histogram citra rambu lalu lintas akan dibandingkan dengan semua nilai *histogram data learning*, sehingga didapatkan nilai terkecil dari hasil perbandingan, dan citra rambu lalu lintas dapat dikenali dengan melihat nilai terkecil dari nilai komparasi pada label yang ditentukan pada *data learning*. Hasil pengenalan citra rambu lalu lintas berupa sebuah nilai *integer* dengan *range* dari 0-4 yang melambangkan jenis citra rambu lalu lintas pada kamera. Setelah citra rambu lalu lintas dapat dikenali maka akan disimpan ke dalam sebuah text hasil data pengenalan citra.

Halaman ini sengaja dikosongkan

BAB IV

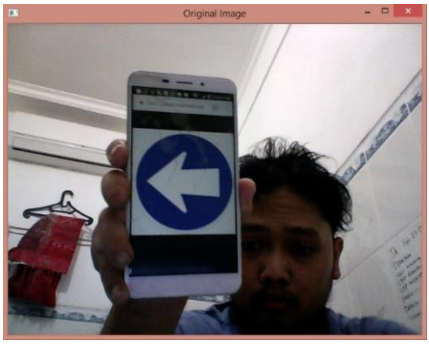
PENGUKURAN DAN ANALISA SISTEM

4.1 Pengujiaan agloritma deteksi objek

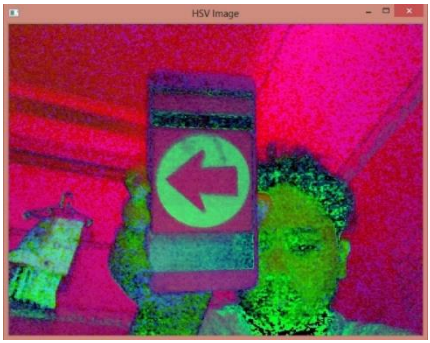
Pada pengujian algoritma deteksi objek terdiri dari pengujian hasil pengambilan gambar, *thresholding* dari HSV *Colorspace*, *morphology* citra *threshold*, dan deteksi gambar dengan ROI.

4.1.1 Pengambilan gambar

Rambu lalu lintas diambil dengan menggunakan satu buah kamera *webcam*, dengan pengaturan besar pixel adalah 640 x 480 *pixel*. Rambu lalu lintas akan diproses untuk dicari letak pada kamera, kemudian setelah di dapatkan letaknya maka akan dilakukan proses ROI. Pada proses ini citra dari kamera akan diubah *color spacenya* menjadi HSV. *Color Space* akan digunakan selanjutnya untuk mendapatkan *threshold* dari citra yang dicari.



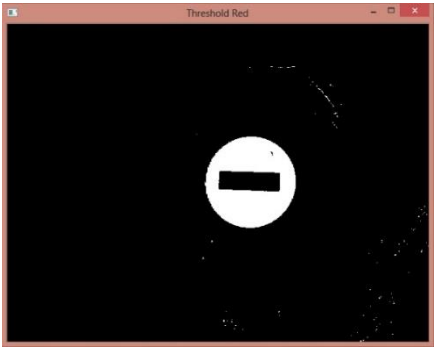
Gambar 4.1 Hasil pengambilan citra dari kamera



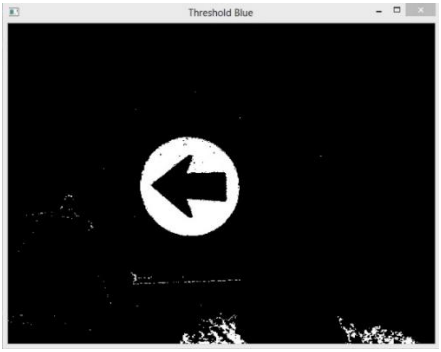
Gambar 4.2 Hasil pengubahan citra menjadi *colorspace* HSV

4.1.2 *Thresholding* dari HSV *Colorspace*

Pada proses *thresholding* digunakan proses pengambilan citra dengan pengubahan *colorspace* dari BGR menjadi HSV. Diambil *range* warna HSV untuk merah dengan nilai minimal Scalar(156, 137, 51) dan nilai maksimal Scalar(255, 255, 255), untuk warna biru dengan nilai *range* minimal Scalar(98, 132, 86) dan nilai maksimal Scalar(119, 255, 255).



Gambar 4.3 Hasil *threshold* dengan *range* merah.

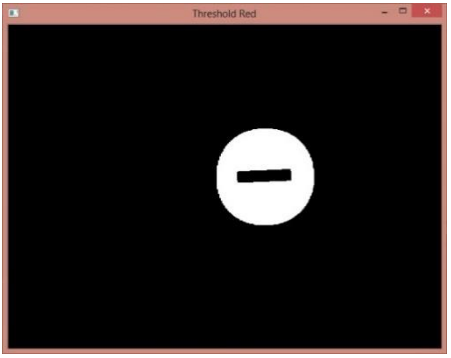


Gambar 4.4 Hasil *threshold* dengan *range* biru.

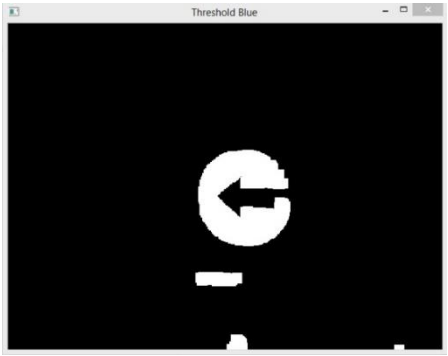
Masih terdapat *noise* pada gambar hasil *thresholding*, *noise* tersebut dapat dihilangkan pada proses selanjutnya yaitu *morphology*.

4.1.3 *Morphology*

Untuk menghilangkan *noise* dilakukan proses *morphology* yang terdiri dari *dilate* dan *erode*. Proses *erode* digunakan untuk menghilangkan *noise* dengan ukuran *pixel* yang memiliki ukuran *contour* lebih kecil dari 3 x 3*px* akan dihilangkan, kemudian dilakukan proses *dilate* untuk memperbesar *pixel* untuk *contour* yang berukuran lebih dari 8 x 8*px*. Sehingga didapatkan hasil *threshold*



Gambar 4.5 Hasil *morphology* citra *threshold* warna merah.



Gambar 4.6 Hasil *morphology* citra *threshold* warna biru.

Pada hasil akhir morfologi citra sudah tidak terdapat *noise* atau sudah jauh berkurang.

4.1.4 *Region Of Interest*

Proses ROI digunakan untuk mengetahui dan melacak gambar yang telah di *threshold* dan sudah diproses *morphology*. Sehingga didapatkan rambu yang akan dideteksi. Pada proses ini hanya ROI yang memiliki ukuran *width* lebih dari 200 x 200*px* yang akan di proses. Setelah melalui proses ini akan didapatkan sebuah file "IMG.jpg" yang kemudian akan diproses menjadi sebuah image LBP.

Gambar 4.7 Hasil citra ROI

4.2 Pengujian proses pengenalan

Pada pengujian proses pengenalan ini terdiri dari pengujian hasil pengoperasian LBP, *initiation data learning*, pengenalan rambu lalu lintas

4.2.1 Operasi *Subdivide*

Pada operasi *subdivide*, citra hasil ROI yang didapatkan akan dibagi menjadi beberapa bagian. Dengan menggunakan ukuran citra awal dan dibagi menjadi beberapa ukuran kecil yang dinginkan. Pada proses *subdivide* digunakan sebanyak pembagian 5 x 5 *subdivide*, 8 x 8 *subdivide*, 10 x 10 *subdivide*, 20 x 20 *subdivide*. Pembagian ini dapat dilakukan karena citra awal yang berukuran 200 *pixel* x 200 *pixel*, sehingga didapatkan hasil *subdivide* sebanyak 25, 64, 100, dan 400 *subdivide*. Pada gambar 4.8 a, b, c, dan d digunakan dengan *subdivide* 2 x 2.

(a)

(b)

(c)

(d)

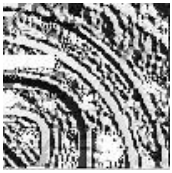
Gambar 4.8

(a) citra pertama. (b) citra kedua. (c) citra ketiga (d) citra keempat

4.2.2 Operasi *Local Binary Pattern*

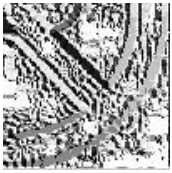
Pada operasi *Local Binary Pattern* dilakukan pada hasil *subdivide* dari citra awal. Hasil dari citra LBP adalah sebuah citra yang memiliki fitur berupa tekstur dari rambu lalu. Kemudian hasil citra LBP akan di ambil nilai *histogram* dan digabungkan menjadi satu bagian untuk dikomparasi dengan nilai *histogram* dari *data learning*. Pada gambar 4.9 a, b, c, dan d merupakan citra LBP dari tiap *subdivide*.

(a)



(b)

(c)



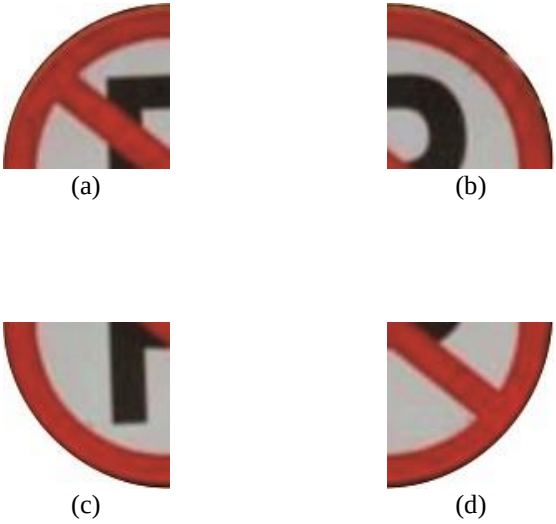
(d)

Gambar 4.9

(a) citra pertama. (b) citra kedua. (c) citra ketiga. (d) citra keempat.

4.2.3 *Initiation data learning*

Hasil inisiasi *data learning* merupakan *array* 4 dimensi yang berisi nilai *histogram* dari citra rambu lalu lintas yang digunakan untuk *data learning*. Terdapat 5 jenis citra rambu lalu lintas, 6 sampel dan 10 sample dari tiap jenis citra rambu lalu lintas, dan *subdivide*. Pada *subdivide* digunakan 25 *subdivide*, 64 *subdivide*, 100 *subdivide*, dan 400 *subdivide*. Berikut ini adalah contoh dari salah satu jenis dan sample dengan menggunakan 4 *subdivide*. Pada gambar 4.10 dan 4.11 menggunakan jenis rambu dilarang parkir.



(a) (b)

(c) (d)

Gambar 4.10
(a) citra pertama. (b) citra kedua. (c) citra ketiga. (d) citra keempat.

(a) (b)

(c)

(d)

Gambar 4.11
(a) citra pertama. (b) citra kedua. (c) citra ketiga. (d) citra keempat.

Setiap citra yang ada pada *data learning* akan di lakukan pengambilan nilai *histogram* dan disimpan kedalam sebuah label dengan format *array* [5][6][25][256], [5][6][64][256], [5][6][100][256], [5][6][400][256] dan [5][10][25][256], [5][10][64][256], [5][10][100][256], [5][10][400][256]. Pada *array* pertama digunakan untuk pelabelan nama jenis rambu lalu lintas, pada *array* kedua digunakan untuk variasi tiap jenis rambu lalu lintas, pada *array* ketiga digunakan untuk menentukan banyaknya *subdivide* yang digunakan sebagai pengenalan, pada *array* keempat digunakan untuk menempatkan nilai histogram dari 0 - 255. *Array data learning* inilah yang digunakan untuk membandingkan nilai *histogram* rambu lalu lintas.

4.2.4 Pengenalan rambu lalu lintas

Data pengujian terdapat 5 (lima) citra rambu lalu lintas yang terbagi menjadi 2 (dua) kelompok. Pada kelompok pertama terdapat 2 citra rambu lalu lintas yang tidak memiliki *background* dan satu citra yang di ubah tingkat kegelapannya. Citra pengujian dapat dilihat pada gambar 4.12 dan gambar 4.13.

Gambar 4.12 rambu dilarang parkir 1.

Gambar 4.13 rambu dilarang parkir 2.

Pada kelompok selanjutanya terdapat 3 citra yang memiliki *background*, Citra pengujian dapat dilihat pada gambar 4.14, gambar 4.15, dan gambar 4.16.

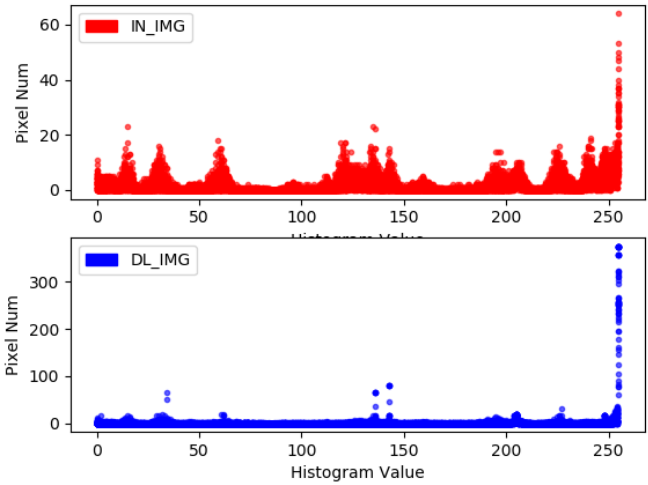
Gambar 4.14 rambu dilarang parkir 3.

Gambar 4.15 rambu dilarang parkir 4.

Gambar 4.16 rambu dilarang parkir 5.

Pengujian dilakukan dengan 5 kali pengulangan untuk tiap jenis rambu lalu lintas. Rambu lalu lintas yang digunakan sesuai dengan 5 jenis rambu pada *data learning*. Rambu pertama adalah rambu dialarang masuk, rambu kedua adalah rambu dilarang parker, ketiga adalah rambu dilarang belok kanan, keempat adalah rambu harus belok kiri, dan kelima adalah rambu putar balik.

Pada pengenalan rambu lalu lintas ini dilakukan pengujian dengan melakukan penggantian pada jumlah *subdivide* saat sebelum citra di LBP dan juga jumlah sampel tiap jenis rambu lalu lintas pada *data learning*. Pada pengujian yang pertama menggunakan sebanyak 6 sampel dari tiap rambu lalu lintas. Data hasil pengujian dengan menggunakan 6 sample dapat dilihat pada tabel 4.1, untuk pengujian degan menggunakan 10 sample dapat dilihat pada table 4.2.



Gambar 4.17 Komparasi nilai histogram citra dengan *data learning*.

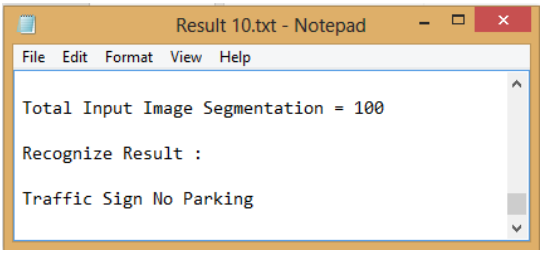
Jenis Rambu	Sample Rambu	Jenis, Sample, *Subdivide*			
		5,6,25	5,6,64	5,6,100	5,6,400
Dilarang Masuk	Citra 1	1	1	1	1
	Citra 2	1	1	1	1
	Citra 3	1	1	1	1
	Citra 4	0	1	1	1
	Citra 5	1	1	1	1
Dilarang Parkir	Citra 1	0	0	1	0
	Citra 2	1	1	1	1
	Citra 3	1	1	1	1
	Citra 4	0	0	0	0
	Citra 5	1	1	1	0
Dilarang Belok Kanan	Citra 1	1	0	1	0
	Citra 2	1	1	1	0
	Citra 3	0	1	1	0
	Citra 4	1	1	1	1
	Citra 5	0	0	1	0
Harus Belok Kiri	Citra 1	1	1	1	1
	Citra 2	1	1	1	0
	Citra 3	0	1	1	1
	Citra 4	1	1	1	1
	Citra 5	1	1	1	1
Putar Balik	Citra 1	1	1	1	0
	Citra 2	1	1	1	1
	Citra 3	1	1	1	1
	Citra 4	1	1	1	0
	Citra 5	1	1	1	0

Table 4.1 Pengujian dengan 6 sample *data learning*

Jenis Rambu	Sample Rambu	Jenis, Sample, *Subdivide*			
		5,10,25	5,10,64	5,10,100	5,10,400
Dilarang Masuk	Citra 1	1	1	1	1
	Citra 2	1	1	1	1
	Citra 3	1	1	1	1
	Citra 4	0	1	1	1
	Citra 5	0	1	1	1
Dilarang Parkir	Citra 1	0	0	1	0
	Citra 2	1	1	1	1
	Citra 3	1	1	1	1
	Citra 4	0	0	0	0
	Citra 5	1	1	1	0
Dilarang Belok Kanan	Citra 1	1	0	1	0
	Citra 2	1	1	1	0
	Citra 3	0	1	1	0
	Citra 4	1	1	1	1
	Citra 5	1	1	1	0
Harus Belok Kiri	Citra 1	1	1	1	1
	Citra 2	0	1	1	1
	Citra 3	1	1	1	1
	Citra 4	1	1	1	1
	Citra 5	1	1	1	1
Putar Balik	Citra 1	1	1	1	0
	Citra 2	1	1	1	1
	Citra 3	1	1	1	1
	Citra 4	1	1	1	0
	Citra 5	1	1	1	0

Table 4.2 Pengujian dengan 10 sample *data learning*

Setelah dikomparasi nilai *histogram* citra rambu lalu lintas dengan nilai *histogram data learning* maka didapatkan hasil seperti berikut :



Gambar 4.18 Hasil pengenalan rambu lalu lintas.

Pengujian yang dilakukan adalah dengan melakukan pengujian citra rambu lalu lintas dengan rambu lalu lintas dari *data learning*. Pengujian dilakukan dengan 2 *variable* pengujian yang berbeda, pertama adalah pengujian dengan pengaruh banyaknya *subdivide* dengan menggunakan 5x5, 8x8, 10x10, dan 20x20 *subdivide*. *Variable* kedua adalah pengujian terhadap penambahan sampel dari tiap jenis rambu lalu lintas, pengujian pertama dengan 6 sampel dan kedua dengan 10 sampel jenis rambu lalu lintas pada *data learning*.

Pada pengujian ini terdapat 5 jenis rambu lalu lintas yang digunakan dan pada tiap rambu lalu lintas terdapat 5 kali pengujian dengan sampel yang berbeda-beda. Sampel pertama merupakan rambu lalu lintas yang didapat secara sempurna dan tidak memiliki *background*. Sampel kedua, ketiga, dan kelima merupakan citra rambu lalu lintas yang diambil dengan menggunakan kamera pada situasi jalan raya. Sampel keempat merupakan citra pada sampel pertama yang mengalami perubahan tingkat pencahayaan. Pengujian kedua dilakukan dengan menambahkan sampel dari jenis rambu lalu lintas sebanyak 4 sampel, total sampel dari tiap jenis menjadi 10 sampel jenis rambu lalu lintas.

Hasil pengujian pertama, pengaruh pengubahan *subdivide* terhadap pengenalan citra rambu lalu lintas. Pada table 4.3, didapatkan bahwa pada citra 1 sampai citra 5, dengan adanya kenaikan *subdivide* pada citra sebelum proses LBP, citra akan semakin baik dikenali. Tetapi pada *subdivide* 400, didapatkan hasil pengenalan citra yang lebih buruk, dikarenakan citra yang dipotong ukurannya menjadi terlalu kecil dan tidak memiliki nilai *histogram vector* yang lebih baik.

Setelah dilakukan pengujian pengenalan rambu lalu lintas didapatkan bahwa dari 25 kali pengujian didapatkan hasil pengenalan sebesar pada 6 *data learning* dengan 25 *subdivide* 76%, pada 6 *data*

learning dengan 64 *subdivide* 84%, pada 6 *data learning* dengan 100 *subdivide* 96%, pada 6 *data learning* dengan 400 *subdivide* 60%. Untuk hasil pengujian dengan 10 *data learning* dengan 25 *subdivide* 76%, pada 10 *data learning* dengan 64 *subdivide* 88%, pada 10 *data learning* dengan 100 *subdivide* 96%, pada 10 *data learning* dengan 400 *subdivide* 64%.

Subdivide	*Data Learning*	Citra 1	Citra 2	Citra 3	Citra 4	Citra 5
25 SEG	6 DL	80%	60%	60%	80%	100%
	10 DL	60%	60%	80%	80%	100%
64 SEG	6 DL	100%	60%	60%	100%	100%
	10 DL	100%	60%	80%	100%	100%
100 SEG	6 DL	100%	80%	100%	100%	100%
	10 DL	100%	80%	100%	100%	100%
400 SEG	6 DL	100%	40%	80%	40%	40%
	10 DL	100%	40%	40%	100%	40%

Tabel 4.3 Pengujian terhadap banyaknya *Subdivide* dan *data leraning* terhadap pengenalan.

Pada pengujian kedua merupakan penambahan 4 sampel pada *data learning*, sehingga menjadi 10 sampel. Penambahan *data learning* berpengaruh kepada tingkat pengenalan sebuah citra. Dimana pada 64 *subdivide* dan 400 *subdivide* mengalami kenaikan pengenalan citra sebesar 2% dan 4%. Pada 64 *subdivide* dengan 6 *data learning*, didapatkan hasil pengenalan sebesar 80%, setelah penambahan *data learning* didapatkan akurasi pengenalan menjadi 82% tingkat kebenaran pengenalan. Pada 400 *subdivide* dengan 6 *data learning*, didapatkan hasil pengenalan sebesar 64%, setelah penambahan *data learning* didapatkan akurasi pengenalan menjadi 68% tingkat kebenaran pengenalan.

Setelah dilakukan pengujian pada table 4.1 dan table 4.2 terdapat perubahan tingkat akurasi dari pengenalan citra rambu lalu lintas, pada [5][6][64] nilai tingkat akurasi adalah 84%, pada saat penambahan sampel dari rambu lalu lintas menjadi 10 sampel, didapatkan angka akurasi menjadi 88%. Pada [5][6][400] nilai tingat akurasi awal adalah 88%, ketika ditambahkan sampel menjadi 10 sampel, didapatkan peningkatan sebear 4% menjadi 64%, peningkatan yang besar ini dapat dipengaruhi dengan semakin banyaknya fitur dari *data learning* yang akan dapat di komparasikan.

Pada pengujian citra rambu dilarang belok ke kanan, terdapat perubahan nilai deteksi, pada penggunaan 25 *subdivide* dan 64 *subdivide*. Dengan penambahan *data learning* maka citra dapat dikenali walau dengan jumlah *subdivide* yang lebih kecil. Perubahan deteksi juga terjadi pada citra rambu harus belok kiri, dimana pada penggunaan 25 *subdivide*, citra tidak dapat dikenali. Dengan adanya penambahan *data learning*, citra dapat dikenali dengan baik.

Tingkat akurasi pembacaan yang paling baik dengan menggunakan 100 *subdivide*, naik dengan 6 sampel citra dan 10 sampel citra rambu lalu lintas. Tingkat pengenalan yang didapatkan sebesar 96%, sedangkan tingkat akurasi pengenalan yang terburuk dengan menggunakan 400 *subdivide*, yaitu sebesar 60% dengan 6 sampel dan 64% dengan 10 sampel.

Pengujian ketiga adalah dengan pembuktian algoritma LBP yang tidak terpengaruh oleh tingkat pencahayaan pada citra. Pada table 5.2 didapatkan bahwa tidak ada perubahan pengenalan. Sehingga dapat dibuktikan bahwa pengaruh tingkat pencahayaan pada citra tidak berpengaruh pada akurasi pengenalan. Pengujian digunakan sampel pengujian citra pertama yang diubah pencahayaannya menjadi lebih gelap. Ada beberapa pengenalan pada citra 1, dan citra 2 yang mengalami pengenalan yang lebih buruk pada perubahan tingkat pencahayaan. Tetapi pada citra 3, mengalami kenaikan tingkat pengenalan pada citra yang lebih gelap. Dengan membandingkan pengubahan *data learning*, tingkat pengenalan tidak terpengaruh oleh jumlah *data learning*.

Citra	*Data Learning*	25 DEG		64 SEG		100 SEG		400 SEG	
		W	B	W	B	W	B	W	B
Citra 1	6 DL	1	0	1	1	1	1	1	1
	10 DL	1	0	1	1	1	1	1	1
Citra 2	6 DL	0	0	0	0	1	0	0	0
	10 DL	0	0	0	0	1	0	0	0
Citra 3	6 DL	1	1	0	1	1	1	0	1
	10 DL	1	1	0	1	1	1	0	1
Citra 4	6 DL	1	1	1	1	1	1	1	1
	10 DL	1	1	1	1	1	1	1	1
Citra 5	6 DL	1	1	1	1	1	1	0	0
	10 DL	1	1	1	1	1	1	0	0

Tabel 4.4 Pengujian tingkat pencahayaan pada citra rambu lalu lintas yang dikenali.

Untuk pengujian algorithma dari LBP dengan pengubahan *brightness* pada pengujian citra 4, didapatkan hasil bahwa *brightness* tidak berpengaruh pada pengenalan citra rambu lalu lintas. Pada pengujian ini juga didapatkan bahwa ada atau tidaknya *background* yang ditunjukan pada Gambar 4.19 a dan b, citra dapat dikenali dengan baik. Hal ini dapat ditunjukan dengan penambahan sampel *data learning*. Pada 6 sampel *data learning*, semua sampel tidak memiliki *background*. Sedangkan dengan penambahan 4 sampel yang memiliki *background*, didapatkan hasil yang sama dan tidak ada perubahan yang *significant*.

(a) (b)

Gambar 4.19
(a) sampel dengan *background*. (b) sampel tanpa *background*.

Pengujian Keempat dilakukan dengan menggunakan 3 kategori rambu lalu lintas. Kategori pertama adalah rambu larangan, kategori kedua adalah rambu perintah, dan ketiga adalah kategori rambu peringatan. Pada table 4.5 akan dijelaskan hasil dari pengujian rambu lalu lintas. Pengujian akan dilakukan dengan menggunakan 5 rambu lalu lintas untuk tiap kategori.

Citra Rambu	Rambu Larangan		Rambu Perintah		Rambu Peringatan	
	Konsisten	Terbaca	Konsisten	Terbaca	Konsisten	Terbaca
Citra 1	1	1	1	1	1	1
Citra 2	1	1	0	0	0	1
Citra 3	1	1	0	1	1	1
Citra 4	0	1	1	1	1	1
Citra 5	1	1	1	1	1	1

Tabel 4.5 Pengujian citra rambu lalu lintas secara *real time*.

Pada pengujian ini akan dilakukan dengan pendeteksian rambu lalu lintas secara *real time* dengan kamera. *Subdivide* yang digunakan adalah 100 *subdivide* Untuk citra rambu lalu lintas kategori pertama dapat dilihat pada gambar 4.20, untuk citra rambu lalu lintas kategori kedua dapat dilihat pada gambar 4.21, dan untuk citra rambu lalu lintas kategori ketiga dapat dilihat pada gambar 4.22. Pada pengujian ini dilakukan selama 30 detik pengenalan. Untuk kategori rambu larangan, pada citra keempat yaitu citra rambu dilarang berhenti, pengenalan selama 30 detik didapatkan bahwa pengenalan tidak selalu konsisten.

Gambar 4.20 Rambu lalu lintas larangan

Gambar 4.21 Rambu lalu lintas perintah

Gambar 4.22 Rambu lalu lintas peringatan

Pada rambu perintah, untuk rambu harus belok kanan tidak terbaca dan pengenalan selalu konsisten, sehingga pengenalan tidak dapat mengulang atau memperbaiki pengenalan rambunya. Pada rambu perintah putar balik, pengenalan tidak konsisten sehingga pengenalan pada saat tertentu dapat memberikan arti yang berbeda. Untuk rambu peringatan, pada rambu peringatan ada banyak pejalan kaki, pengenalan tidak konsisten sehingga pengenalan pada saat tertentu dapat memberikan arti yang berbeda.

Dari hasil pengujian didapatkan bahwa pengujian citra secara *real time* dapat membaca sebesar 80% dengan error sebesar 10%. Error ini didapat dikarenakan faktor perangkat keras dalam mengambil citra yang menggunakan *auto white balance*. Jika dapat dihilangkan, maka pengenalan akan mendapat tingkat pengenalan sebesar 90%. Hal ini sesuai dengan pengujian pertama yang menunjukan bahwa pengenalan rambu lalu lintas dengan menggunakan metode *Local Binary Pattern*, mendapatkan pengenalan sebesar 96%. Untuk pengenalan menggunakan program dan sistem ini mendapatkan tingkat akurasi pengenalan sebesar 80% dan tingkat ketelitian pengenalan 90%.

Halaman ini sengaja dikosongkan

BAB V

PENUTUP

5.1 Kesimpulan

Pada penelitian ini telah dibuat pengenalan rambu lalu lintas dengan menggunakan metode *Local Binary Pattern*. Pada sistem ini dibagi menjadi 3 bagian program. Program pertama dalah deteksi citra rambu lalu lintas dengan HSV dan *Hough Transformation*, program kedua adalah proses *Local Binary Pattern*, program ketiga adalah pengenalan ditra rambu lalu lintas dengna metode *Chi-Square*.

Dari hasil pengujian untuk pengenalan citra rambu lalu lintas didapatkan bahwa:

1. Hasil tingkat akurasi pengenalan ramb lalu lintas paling baik pada 100 *subdivide* dengan 96% tingkat akurasi pengenalan.
2. Penambahan *data learning* berpengaruh kepada lama waktu pengenalan dan tingkat pengenalan citra rambu lalu lintas
3. Pengaruh tingkat pencahayaan pada rambu lalu lintas tidak berpengaruh pada pengenalan rambu lalu lintas. Karena algoritma LBP tidak berpengaruh besar pada warna dari citra, tetapi tekstur.
4. Untuk waktu komputasi pengenalan yang lebih cepat, dapat menggunakan *subdivide* yang lebih sedikit, atau dapat menggunakan *data learning* yang sedikit.

Data yang didapat dari hasil pengujian menunjukan bahwa pengenalan menggunakan metode LBP dapat mengenali rambu lalu lintas dengan baik di kondisi jalan raya atau secara *real time*. Hasil dari penelitian ini akan dapat di implementasikan kedalam sebuah alat pengenalan rambu lalu lintas pada kendaraan autonomous dengan menggunakan *On Board Computer*, sehingga kendaraan autonomous dapat berkendara di jalan raya tanpa harus melanggar peraturan rambu lalu lintas dan dapat mengetahui kejadian yang ada di sekitarnya.

5.2 Saran

Saran agar program ini dapat berjalan lebih baik yaitu.

1. Penambahan pada *data learning* agar tingkat akurasi pengenalan program lebih tinggi.
2. Menggunakan metode komparasi *histogram* dengan komputasi yang lebih cepat dan lebih akurat.
3. Melakukan perbaikan pada program deteksi rambu lalu lintas agar rambu berbentuk prisma dapat di deteksi.

DAFTAR PUSTAKA

[1] Guo, Zhenhua. Lei Zhang. David Zhang. 2010. *A Completed Modeling of Local Binary Pattern Operator for Texture Classification*, Hong Kong Polytechinc University.

[2] Liberty, Jesse. 2002. *Learning Visual Basic .NET : Introducing the Language, .NET Programming & Object Oriented Software Development*, California: O'Reilly Media, Inc.

[3] Bradski, Gary. Adrian Kaehler. 2008. *Learning OpenCV Computer Vision with the OpenCV Library*, California: O'Reilly Media, Inc.

[4] V. A. Olivera. A. Conci. 2009. *Skin Detection using HSV Color Space*, Brazil : Universidade Federal Fluminese

[5] Pietikanen, Matti. Abdenour Hadid. Guoying Zhao. Timo Ahonen. 2011. *Computer Vision Using Local Binary Patterns.* Chester: Spinger.

[6] Szeliski, Richard. 2011. *Computer Vision: Algorithms and Applications.* Chester: Spinger.

[7] Guo, Zhenhua. Lei Zhang. David Zhang. 2010. *A Completed Modeling of Local Binary Pattern Operator for Texture Classification*, Hong Kong Polytechinc University.

[8] Liberty, Jesse. 2002. *Learning Visual Basic .NET : Introducing the Language, .NET Programming & Object Oriented Software Development*, California: O'Reilly Media, Inc.

[9] Bradski, Gary. Adrian Kaehler. 2008. *Learning OpenCV Computer Vision with the OpenCV Library*, California: O'Reilly Media, Inc.

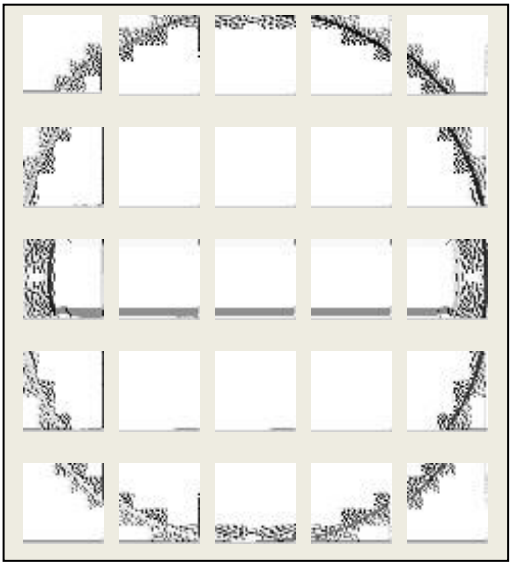
[10] V. A. Olivera. A. Conci. 2009. *Skin Detection using HSV Color Space*, Brazil : Universidade Federal Fluminese

[11] Pietikanen, Matti. Abdenour Hadid. Guoying Zhao. Timo Ahonen. 2011. *Computer Vision Using Local Binary Patterns.* Chester: Spinger.

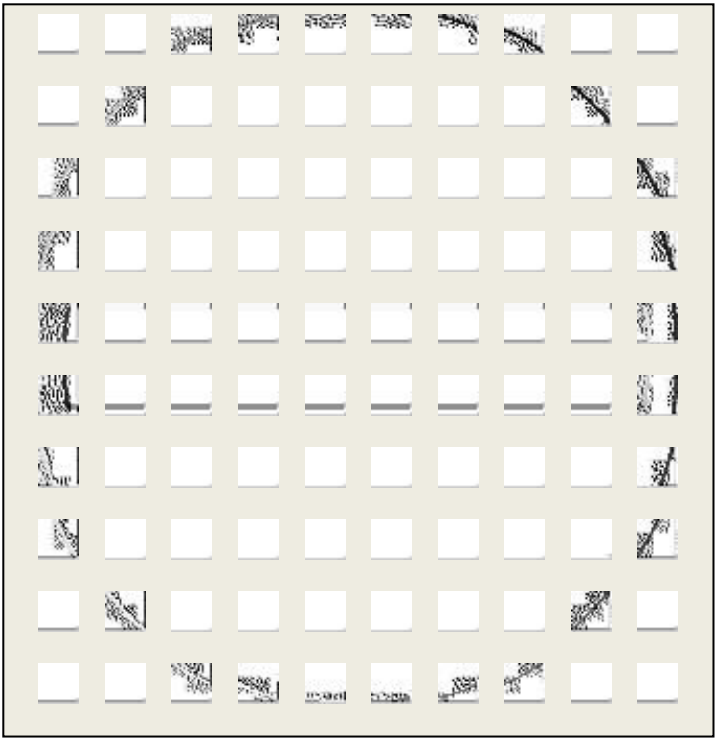
Halaman ini sengaja dikosongkan

LAMPIRAN

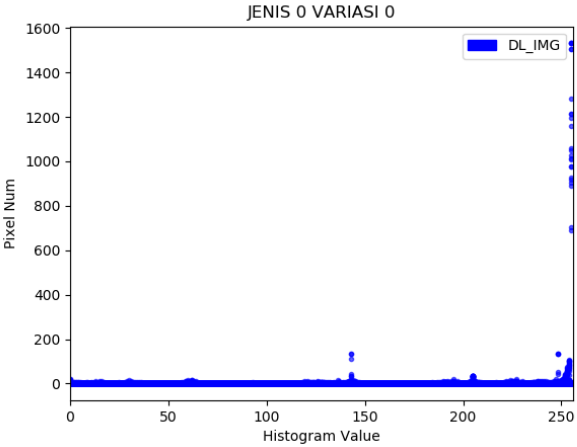
Citra LBP pada *data learning*:



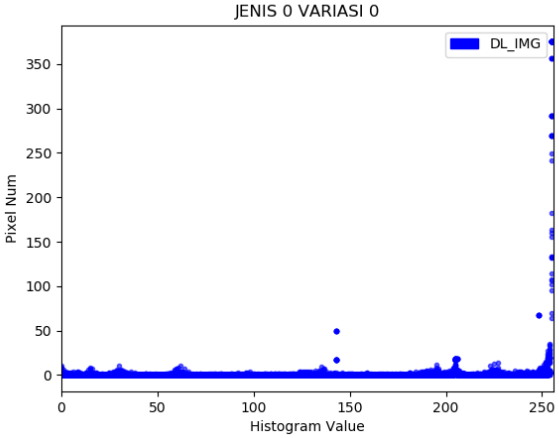
Citra LBP *data learning* dengan 25 segmentasi



Citra LBP *data learning* dengan 100 segmentasi

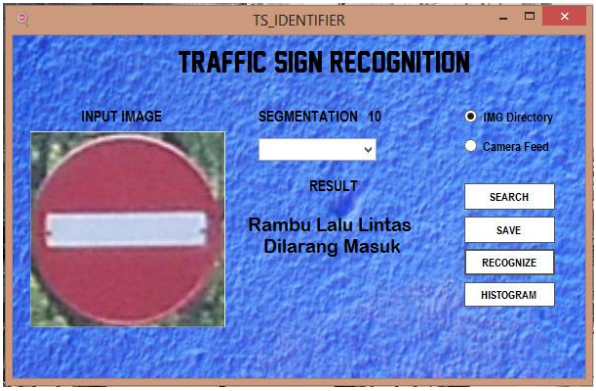


Nilai histogram pada *data learning* dengan 25 segmentasi



Nilai histogram pada *data learning* dengan 100 segmentasi

Graphic User Interface :



Program pengenalan citra :

```
#include <opencv2/core/core.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/core/mat.hpp>
#include <opencv2/core/types_c.h>
#include <opencv2/core/core_c.h>
#include <opencv2/highgui.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <stdio.h>
#include <iostream>
#include <fstream>
#include <math.h>
#include <opencv2/core/core.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/core/mat.hpp>
#include <opencv2/core/types_c.h>
#include <opencv2/core/core_c.h>
#include <opencv2/highgui.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <stdio.h>
#include <iostream>
```

```
#include <fstream>
#include <math.h>

using namespace cv;
using namespace std;

// MENENTUKAN JUMLAH DARI HISTOGRAM
int aa = 5; /// JENIS
int bb = 10; /// VARISAI
int dd = 10; // 5,8,10,20 /// BANYAKNYA SEGMENTASI YANG
DIINGINKAN
int ee = dd*dd; /// JUMLAH PENGULANGAN PARTISI
int cc = ee; /// PARTISI
int ff = 200 / dd; /// UKURAN DARI WINDOW

float datalearning[5][10][100][256]; /// UBAH NILAI UNTUK [JENIS]
[VARIASI] [PARTISI]
float hist_char[100][256];

int main(int argc, char* argv[]) {
Mat img_mat;
Mat hist_mat;

int label;
const int channels = 0;
const int numBins = 256; // nilai histogram
int char_recog[1107];
const float rangevals[2] = { 0.f,256.f };
const float* ranges = rangevals;
char lbp_imagestemp[1107];
char data_learnings[1107];
char data_learning[1107];
char partitions[512];

//================================================
================================================//

// ---------- INPUT IMAGE ---------- //
```

```
Mat input_gambar = imread("LBP_IN/IMG.jpg",
CV_LOAD_IMAGE_COLOR); /// IMAGE YANG DIINGINKAN
Size small_size(ff, ff);
Mat gambar_kecil = Mat(input_gambar, Rect(0, 0, ff, ff)).clone();
//imshow("Input Image", input_gambar);

//================================================
==============================================//

// ---------- SEGMENTATION INPUT IMAGE ---------- //

cout << "L O A D I N G ...." << endl;

for (int c = 0; c < ee; c++) {
for (int y = 0; y < input_gambar.rows; y += small_size.height) {
for (int x = 0; x < input_gambar.cols; x += small_size.width) {

//------ ROI PROGRAM ------//
Rect ROI = Rect(x, y, small_size.width, small_size.height);

//------ PENAMAAN SEGMENTASI ------//
snprintf(partitions, 512, "LBP_IN/IN_SEG %02d/%05d.jpg", dd, c);
gambar_kecil = input_gambar(ROI);
imwrite(partitions, gambar_kecil);
c++;
}
}
}

//================================================
==============================================//

// -------------------- LBP IMG ----------------------//

int lbp_nb[8];
char character_images[1107];
char lbp_images[1107];

for (int c = 0; c < ee; c++) {
```

```
snprintf(character_images, 1107, "LBP_IN/IN_SEG %02d/%05d.jpg",
dd, c);
IplImage* img_part = cvLoadImage(character_images,
CV_LOAD_IMAGE_GRAYSCALE);
IplImage* img_lbp = cvCreateImage(cvSize(ff, ff), 8, 1);
//cvShowImage("INPUT IMG", img_part);

for (int y = 0; y < ff; y++) {
for (int x = 0; x < ff; x++) {
int center = CV_IMAGE_ELEM(img_part, uchar, y, x);
lbp_nb[0] = CV_IMAGE_ELEM(img_part, uchar, y - 1, x - 1);
lbp_nb[1] = CV_IMAGE_ELEM(img_part, uchar, y - 1, x);
lbp_nb[2] = CV_IMAGE_ELEM(img_part, uchar, y - 1, x + 1);
lbp_nb[3] = CV_IMAGE_ELEM(img_part, uchar, y, x + 1);
lbp_nb[4] = CV_IMAGE_ELEM(img_part, uchar, y + 1, x + 1);
lbp_nb[5] = CV_IMAGE_ELEM(img_part, uchar, y + 1, x);
lbp_nb[6] = CV_IMAGE_ELEM(img_part, uchar, y + 1, x - 1);
lbp_nb[7] = CV_IMAGE_ELEM(img_part, uchar, y, x - 1);

for (int i = 0; i < 8; i++) {
if (lbp_nb[i] < center) {
lbp_nb[i] = 0;
}

else {
lbp_nb[i] = 1;
}
}
center = lbp_nb[7] * 128 + lbp_nb[6] * 64 + lbp_nb[5] * 32 + lbp_nb[4]
* 16 + lbp_nb[3] * 8 + lbp_nb[2] * 4 + lbp_nb[1] * 2 + lbp_nb[0] * 1;
CV_IMAGE_ELEM(img_lbp, uchar, y - 1, x - 1) = center;
}
}
//cvShowImage("LBP IMG", img_lbp);
snprintf(lbp_images, 1107, "LBP_IN/IN_SEG
%02d/LBP_IN_SEG/%05d.jpg", dd, c);
cvSaveImage(lbp_images, img_lbp);
}
```

```
//=================================================
================================================//

// ---------- HISTOGRAM DATA LEARNING ---------- //


ofstream segfile;
segfile.open("LBP_IN/Histogram DL.txt"); // MELIHAT HASIL
HISTOGRAM
for (int a = 0; a < aa; a++) {
for (int b = 0; b < bb; b++) {
for (int c = 0; c < cc; c++) {
snprintf(data_learning, 1107, "LBP_DL/DL_SEG
%02d/LBP_DL_SEG/%01d%01d%03d.jpg",dd, a, b, c);
img_mat = imread(data_learning, CV_LOAD_IMAGE_ANYCOLOR);
calcHist(&img_mat, 1, &channels, noArray(), hist_mat, 1, &numBins,
&ranges);
snprintf(data_learnings, 1107, "Histogram %01d%01d%03d.jpg =", a, b,
c);
segfile << data_learnings << endl;
for (size_t i = 0; i < numBins; ++i) {
float val = hist_mat.at<float>(i);
datalearning[a][b][c][i] = val;
segfile << datalearning[a][b][c][i] << "|";
if ((i == 50) || (i == 100) || (i == 150) || (i == 200) || (i == 250)) {
segfile << "\n";
}
}
segfile << "\n";
}
segfile << "\n";
}
segfile << "\n";
}
segfile.close();

//=================================================
================================================//
```

```
// ---------- RECOGNIZION TRAFFIC SIGN ---------- //

segfile.open("Result.txt");

//------ HISTOGRAM INPUT IMAGE ------//

for (int c = 0; c<cc; c++) {
snprintf(lbp_imagestemp, 1107, "LBP_IN/IN_SEG
%02d/LBP_IN_SEG/%05d.jpg", dd, c);
img_mat = imread(lbp_imagestemp,
CV_LOAD_IMAGE_ANYCOLOR);
//imshow("Input", img_mat);
calcHist(&img_mat, 1, &channels, noArray(), hist_mat, 1, &numBins,
&ranges);

for (size_t i = 0; i<numBins; ++i) {
float val = hist_mat.at<float>(i);
hist_char[c][i] = val;
}
}

//------ COMPARE HISTOGRAM DATA ------//

float comp = 3.40282347E+38;
for (int a = 0; a<aa; a++) {
float comp_newval = 0;
for (int b = 0; b<bb; b++) {
float comp_val = 0;

for (int c = 0; c<cc; c++) {
float chi_sqr = 0;

for (size_t i = 0; i < numBins; ++i) {
float pembilang = datalearning[a][b][c][i] - hist_char[c][i];
float penyebut = datalearning[a][b][c][i];
if (penyebut != 0) {
chi_sqr = chi_sqr + pow(pembilang, 2) / (penyebut);
}
}
```

```
comp_val = comp_val + chi_sqr;
}

if (comp_val<comp) {
comp = comp_val;
label = a;
}
}
}

//------ LABEL NAMING ------//

string Char_label[5] = {
"Dilarang Masuk","Dilarang Parkir", "Dilarang Belok Kanan","Putar
Balik","Harus Belok Kiri"
};
if (label == 0) {
//cout << "Traffic Sign " << Char_label[0] << endl;
segfile << "Rambu Lalu Lintas " << endl;
segfile << Char_label[0] << endl;
segfile << "\n";
}
if (label == 1) {
//cout << "Traffic Sign " << Char_label[1] << endl;
segfile << "Rambu Lalu Lintas " << endl;
segfile << Char_label[1] << endl;
segfile << "\n";
}
if (label == 2) {
//cout << "Traffic Sign " << Char_label[2] << endl;
segfile << "Rambu Lalu Lintas " << endl;
segfile << Char_label[2] << endl;
segfile << "\n";
}
if (label == 3) {
//cout << "Traffic Sign " << Char_label[3] << endl;
segfile << "Rambu Lalu Lintas " << endl;
segfile << Char_label[3] << endl;
segfile << "\n";
```

```
}
if (label == 4) {
//cout << "Traffic Sign " << Char_label[4] << endl;
segfile << "Rambu Lalu Lintas " << endl;
segfile << Char_label[4] << endl;
segfile << "\n";
}
segfile.close();
cout << "\n";
cout << "F I N I S H ...." << endl;

waitKey(0);

}
```

Halaman ini sengaja dikosongkan

BIODATA PENULIS

Kristopher Lukas, lahir di Jakarta 30 September 1995. Anak pertama dari dua bersaudara. Penulis memulai pendidikan jenjang dasar di sekolah dasar swasta SDK Mater Dei (2001), Pamulang. Setelah lulus sekolah dasar tahun 2007 penulis melanjutkan ke jenjang menengah di sekolah menegah pertama swasta SMPK Mater Dei, Pamulang (2007). Kemudian penulis melanjutkan jenjang pendidikan di sekolah menengah atas SMAN 2 Kota Tangerang Selatan, Cisauk (2010). Setelah menyelesaikan pendidikan di jenjang sekolah menengah atas penulis melanjutkan jenjang pendidikannya di Institut Teknologi Sepuluh Nopember departemen Teknik Elektro dengan bidang studi Elektronika.

Email: kristopher.sitorus@gmail.com

Halaman ini sengaja dikosongkan