SKRIPSI

PERANCANGAN PROGRAM PENGHITUNG JUMLAH KENDARAAN DI LINTASAN JALAN RAYA SATU ARAH MENGGUNAKAN BAHASA PEMROGRAMAN C++ DENGAN PUSTAKA OPENCV

Diajukan Untuk Memenuhi Persyaratan

Memperoleh Gelar Sarjana Teknik

Disusun Oleh:

FAJAR MIT CAHYANA

NIM. 0610630037-63

JURUSAN TEKNIK ELEKTRO

FAKULTAS TEKNIK

UNIVERSITAS BRAWIJAYA

MALANG

2014

LEMBAR PERSETUJUAN

PERANCANGAN PROGRAM PENGHITUNG JUMLAH KENDARAAN DI LINTASAN JALAN RAYA SATU ARAHMENGGUNAKAN BAHASA PEMROGRAMAN C++ DENGAN PUSTAKA OPENCV

SKRIPSI

KONSENTRASI TEKNIK ELETRONIKA

DiajukanUntukMemenuhiPersyaratan

MemperolehGelarSarjanaTeknik

DisusunOleh:

FAJAR MIT CAHYANA

NIM. 0610630037-63

Telah diperiksa dan disetujui oleh :

Dosen Pembimbing I	Dosen Pembimbing II
M. Julius St., Ir., MS	**R. Arief Setiawan ST., MT**
NIP. 19540720 198203 1 003	**NIP. 19750819 199903 1 002**

LEMBAR PENGESAHAN

PERANCANGAN PROGRAM PENGHITUNG JUMLAH KENDARAAN DI LINTASAN JALAN RAYA SATU ARAH MENGGUNAKAN BAHASA PEMROGRAMAN C++ DENGAN PUSTAKA OPENCV

SKRIPSI

KONSENTRASI TEKNIK ELETRONIKA

Diajukan Untuk Memenuhi Persyaratan
Memperoleh Gelar Sarjana Teknik

Disusun Oleh:

FAJAR MIT CAHYANA

NIM. 0610630037-63

Skripsi ini telah diuji dan dinyatakan lulus pada
Tanggal 11 Agustus 2014

Majelis Penguji :

Ir. Ponco Siwindarto, M.Eng.Sc	**Ir. Nanang Sulistyanto, M.T.**
NIP. 19590304 198903 1 001	**NIP. 19700113 199403 1 002**

Akhmad Zainuri, S.T., M.T.
NIP. 19840120 201212 1 003

Mengetahui :

Ketua Jurusan Teknik Elektro

M. Aziz Muslim, S.T., M.T., Ph.D
NIP. 19741203 200012 001

ABSTRAK

Fajar Mit Cahyana**, Perancangan Program Penghitung Jumlah Kendaraan Di Lintasan Jalan Raya Satu Arah Menggunakan Bahasa Pemrograman C++ Dengan Pustaka Opencv.**

Dosen Pembimbing : M. Julius, ST., MS & R. Arief Setyawan, ST., MT

Untuk menentukan kepadatan rata-rata lalu-lintas diperlukan adanya survey penghitungan jumlah kendaraan yang melintas di jalan raya. Pelaksannan survey tersebut biasanya dilakukan oleh seorang pengamat yang dimungkinkan terjadinya *human error* dalam proses penghitungan karena terlalu padatnya jumlah kendaraan yang lewat, pengaruh lingkungan atau kondisi internal peneliti itu sendiri sehingga mengakibatkan kurang akuratnya proses penghitungan yang dilakukan langsung oleh seorang peneliti. Selain rentan terjadinya *human error*, penghitungan yang dilakukan oleh manusia memerlukan biaya tersendiri untuk setiap pelaksanaanya sehingga kurang efisien.

Oleh karena itu, sebuah perancangan dan analisis program penghitung jumlah kendaraan di lintasan jalan raya dilakukan menggunakan bahasa pemrograman C++ dengan pustaka OpenCV. OpenCV digunakan karena memiliki banyak subprogram atau *library* yang dapat dikombinasikan sehingga memiliki berbagai fungsi dalam pemrograman yang berkaitan dengan pengolahan citra digital. Sedangkan bahasa pemrograman C++ digunakan karena Pustaka OpenCV 2.4.4 hanya dapat dijalankan dengan menggunakan bahasa pemrograman C dan C++.

Dari hasil pengujian didapatkan kecepatan rata-rata proses *background subtraction* 483,33 *frame* per detik dengan menggunakan metode *frame difference*. Dari hasil tiga kali pengujian *blobtracking* menunjukkan dari 100 kendaraan yang diproses oleh program dengan sudut pandang kamera 45^0 sebanyak 75 unit kendaraan berhasil dideteksi, dengan sudut pandang kamera 60^0 sebanyak 79 unit kendaraan berhasil dideteksi dan dengan sudut pandang kamera 90^0 sebanyak 75 unit kendaraan berhasil dideteksi.

PENGANTAR

Assalamu'alaikum warahmatullahi wabarakatuh.

Puji syukur kehadirat Allah Yang Maha Kuasa atas limpahan karunia-Nya berupa kemampuan berpikir, sehingga hanya dengan pertolongan-Nya penulis dapat menyelesaikan skripsi dengan judul **PERANCANGAN PROGRAM PENGHITUNG JUMLAH KENDARAAN DI LINTASAN JALAN RAYA SATU ARAH MENGGUNAKAN BAHASA PEMROGRAMAN C++ DENGAN PUSTAKA OPENCV.** Sholawat dan salam semoga tetap tercurah kepada Nabi Muhammad SAW, keluarga, para sahabat dan orang-orang tetap berada di jalan-Nya.

Tujuan penulisan skripsi ini adalah untuk memenuhi syarat dalam mencapai gelar sarjana teknik pada Jurusan Teknik Elektro Universitas Brawijaya.

Ucapan terima kasih saya sampaikan pada segenap pihak yang membantu dalam penulisan skripsi ini atas dukungan baik yang bersifat moral maupun material sehingga penulis mampu menyelesaikan penulisan skripsi ini.

Penulis senantiasa menyadari bahwa tulisan ini masih jauh dari sempurna, baik dari segi materi, sistematika pembahasan maupun susunan bahasanya. Oleh karenanya, penulis senantiasa terbuka terhadap saran dan kritik yang konstruktif. Dengan iringan doa mudah-mudahan karya tulis ini dapat bermanfaat dalam pengembangan pendidikan serta wawasan berpikir kita bersama.

Wassalamu'alaikum warahmatullahi wabarakatuh.

Malang, 23 Juli 2014

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I

PENDAHULUAN

1.1 LATAR BELAKANG

Untuk menentukan kepadatan rata-rata lalu-lintas diperlukan adanya survey penghitungan jumlah kendaraan yang melintas di jalan raya yang biasanya dilakukan pada jam-jam sibuk, yaitu dimulai pada pukul 06.00 – 09.00 dan pukul 15.00 – 18.00. Pelaksannan survey tersebut biasanya dilakukan oleh seorang peneliti yang memungkinkan terjadinya *human error* dalam proses penghitungan karena terlalu padatnya jumlah kendaraan yang lewat, pengaruh lingkungan atau kondisi internal peneliti itu sendiri sehingga mengakibatkan kurang akuratnya proses penghitungan yang dilakukan langsung oleh seorang peneliti. Selain rentan terjadinya *human error*, penghitungan yang dilakukan oleh peneliti memerlukan biaya tersendiri untuk setiap pelaksanaanya sehingga mengakibatkan kurang efisien dalam pelaksanaan survey yang dilakukan.

Untuk meningkatkan tingkat akurasi dan efisiensi biaya maka penggunaan kamera digital sebagai sensor penghitung kendaraan dapat meningkatkan tingkat akurasi dan efisiensi dalam proses penghitungan. Kamera tersebut tentunya membutuhkan sebuah perangkat lunak yang digunakan untuk mendeteksi, mengenali dan menghitung benda yang melintas dihadapan kamera digital tersebut.

Oleh karena itu, penulis berusaha melakukan sebuah perancangan dan analisis program penghitung jumlah kendaraan di lintasan jalan raya menggunakan bahasa pemrograman C++ dengan pustaka OpenCV. OpenCV digunakan karena memiliki banyak subprogram atau *library* yang dapat dikombinasikan sehingga memiliki berbagai fungsi dalam pemrograman yang berkaitan dengan pengolahan citra digital. Sedangkan bahasa pemrograman C++ digunakan karena Pustaka OpenCV 2.4.4 hanya dapat dijalankan dengan menggunakan bahasa pemrograman C dan C++.

1.2 RUMUSAN MASALAH

Rumusan masalah pada skripsi ini ditekankan pada :

1. Bagaimana menentukan metode pemisahan latar belakang yang bersifat statis dengan objek yang bergerak pada video.
2. Bagaimana menentukan metode mengunci objek yang bergerak pada video yang diamati.
3. Bagaimana menentukan metode menghitung objek yang bergerak pada video yang diamati.

1.3 BATASAN MASALAH

Mengacu pada permasalahan yang ada, maka skripsi ini memiliki batasan masalah sebagai berikut :

1. Bahasa pemrograman yang digunakan dalam skripsi ini menggunakan bahasa pemrograman C++ dengan menggunakan aplikasi Microsoft visual studio C++ 2010.
2. Pustaka yang digunakan dalam skripsi ini menggunakan pustaka OpenCV
3. Metode pemisahan latar belakang dengan objek yang bergerak menggunakan metode pengurangan latar belakang
4. Video yang akan dihitung objek bergeraknya berupa kendaraan beroda empat yang didapatkan dari proses perekaman menggunakan kamera digital.

1.4 TUJUAN

Tujuan penelitian ini adalah membuat program program yang mampu menghitung jumlah kendaraan di lintasan jalan raya satu arah menggunakan bahasa pemrograman c++ dengan pustaka opencv 2.4.4

1.5 SISTEMATIKA PENULISAN HASIL SKRIPSI

Sistematika penulisan yang digunakan dalam penyusunan laporan penelitian ini adalah sebagai berikut.

BAB I : PENDAHULUAN

Memuat latar belakang, rumusan masalah, ruang lingkup, tujuan, dan sistematika penulisan.

BAB II : TINJAUAN PUSTAKA

Tinjauan pustaka yang membahas dasar teori tentang pemrosesan citra digital menggunakan pustaka OpenCV dan metode yang digunakan untuk memisahkan antara latar belakang video dengan objek yang bergerak.

BAB III : METODOLOGI

Berisi metode penelitian yang akan dilakukan , terdiri dari objek penelitian, studi literatur, pengumpulan data, perancangan program, analisis, serta penarikan kesimpulan.

BAB IV : PEMBAHASAN

Melakukan perancangan program dengan menggunakan data dari objek yang diteliti kemudian dijalankan dan dianalisis.

BAB V : PENUTUP

Berisi kesimpulan dan saran yang diperoleh dari hasil analisis.

BAB II

TINJAUAN PUSTAKA

Pada bab ini dijelaskan kajian pustaka dan dasar teori yang digunakan untuk menunjang penyelesaian skripsi ini. Kajian pustaka diperlukan untuk melakukan kajian terhadap karya ilmiah yang berkaitan dengan skripsi ini. Tinjauan pustaka yang dipaparkan meliputi: *Computer Visual, OpenCV,* pemisahan latar belakang, *blobtracking,* dan beberapa teori perhitungan mengenai algoritma yang dipakai.

2.1. *Computer Vision*

Computer vision adalah ilmu pengetahuan yang mempelajari bagaimana komputer dapat mengenali objek yang diamati atau diobservasi. Computer vision merupakan salah satu disiplin ilmu dari Artificial Intelligence yang biasa juga di sebut kecerdasan buatan, computer vision berfokus pada informasi-informasi yang dimiliki oleh data gambar. Data gambar memiliki bentuk seperti video (gambar yang berjalan) dan gambar dari kamera.

Salah satu proses dari computer vision adalah image processing atau lebih dikenal pengolahan citra. Sebagai disiplin teknologi, computer vision berusaha untuk menerapkan teori dan model untuk pembangunan sistem computer vision. Definisi computer vision dari beberapa ahli sebagai berikut :

- Menurut Andian Low (1991), computer vision berhubungan dengan perolehan gambar, pemrosesan, klasifikasi, pengenalan, dan menjadi penggabungan pengurutan pembuatan keputusan menuju pengenalan.

- Menurut Michael G. Fairhurst (1995), computer vision sesuai dengan sifatnya, merupakan suatu subyek yang merangkul berbagai disiplin tradisional secara luas guna mendasari prinsip-prinsip formalnya, dan dalam mengembangkan suatu metodologi yang berlainan dari apa yang dimilikinya, pertama-tama harus menggabungkan dan secara berurutan membangun materi yang mendasari ini.

- Menurut Saphiro dan Stockman (2001), computer vision merupakan suatu bidang yang bertujuan untuk membuat suatu keputusan yang berguna mengenai objek fisik nyata dan keadaan berdasarkan atas sebuah citra. Computer vision merupakan kombinasi antara pengolahan citra dan pengenalan pola. Hasil keluaran dari proses computer vision adalah pengertian tentang citra.

Contoh aplikasi dari computer vision mencakup sistem untuk :

- Pengendalian Proses
- Mendeteksi peristiwa
- Mengorganisasi informasi (misalnya untuk pengindeksan database foto dan gambar urutan
- Modeling benda atau lingkungan
- Interaksi (misalnya input ke perangkat untuk interaksi dengan manusia)

2.2. *OpenCV*

OpenCV adalah sebuah library yang berisi fungsi-fungsi pemograman untuk teknologi computer vision secara real time. OpenCV bersifat open source, bebas digunakan untuk hal-hal yang bersifat akademis dan komersial. Di dalamnya terdapat interface untuk C++, C, Python, dan nantinya Java yang dapat berjalan pada Windows, Linux, Android, dan Mac. Terdapat lebih dari 2500 algoritma dalam OpenCV, digunakan di seluruh dunia, telah lebih dari 2,5 juta kali diunduh dan digunakan lebih dari 40 ribu orang. Penggunaannya antara lain pada seni interaktif, inspeksi tambang dan menampilkan peta di web melalui teknologi robotik.

Pada awalnya OpenCV ditulis dengan menggunakan bahasa C namun sekarang secara menyeluruh sudah menggunakan antarmuka bahasa C++ dan seluruh pengembangannya terdapat dalam format bahasa C++. Contoh aplikasi dari OpenCV yaitu interaksi manusia dan komputer, identifikasi, segmentasi dan

pengenalan objek, pengenalan wajah, pengenalan gerakan dan penelusuran dapat dalam OpenCV antara lain:

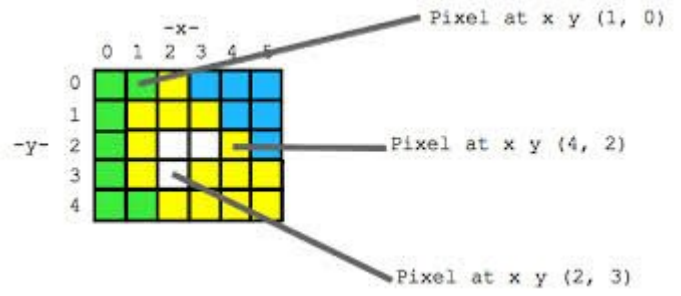
- Manipulasi data image (alokasi, rilis, duplikasi, pengaturan, konversi).
- Image dan I/O video (masukan berbasis file dan kamera, keluaran image/video file).
- Manipulasi matriks dan vektor serta aljabar linear (produk, solusi, eigenvalues, SVD).
- Beragam struktur data dinamis (daftar, baris, grafik).
- Dasar pengolahan citra (filter, deteksi tepi, deteksi sudut, pengambilan sampel dan interpolasi, konversi warna, operasi morfologi, histogram).
- Analisis struktur (komponen yang berhubungan, pengolahan kontur, transformasi jarak, variasi momen, transformasi Hough, perkiraan polygonal, menyesuaikan garis, Delaunay triangulation.
- Kalibrasi kamera (menemukan dan menelusuri pola kalibrasi, kalibrasi, dasar estimasi matriks, estimasi homografi, korespondensi stereo).
- Analisis gerakan (optical flow, segmentasi gerakan, penelusuran).
- Pengenalan objek (metode eigen, HMM).
- Dasar Graphical User Interface atau GUI (menampilkan image/ video, penanganan mouse dan keyboard, scroll-bars).
- Pelabelan image (garis, poligon, gambar teks).

2.3 Citra Digital

Seperti yang dikemukakan oleh Anvil K. Jain (1989), Citra digital adalah representasi dari suatu objek nyata baik dalam bentuk 2 dimensi maupun 3 dimensi menjadi bentuk gambar digital yang dikenali oleh komputer.

Citra digital dapat didefinisikan sebagai fungsi dua variabel, f(x,y), x dan y adalah koordinat spasial dan nilai f(x, y) adalah intensitascitra pada koordinat tersebut dan nilai dari fungsi yang merupakan tingkat intensitas warna atau tingkat keabu-

abuan dari titik tersebut (Robert J.schalkoff, 1989). Hal ini ditunjukkan dalam Gambar 2.1.



Gambar 2.1 Citra Digital

2.3.1 Pengolahan Citra Digital

Pengolahan citra merupakan bidang studi yang mempelajari proses pengolahan gambar di mana baik masukan maupun keluarannya berbentuk berkas citra digital (Aniati Murni Arymurthy, 1992).

Pengolahan citra digital dapat dikelompokkan dalam dua jenis kegiatan :

1) Memperbaiki kualitas suatu gambar, sehingga dapat lebih mudah diinterpretasi oleh mata manusia.
2) M engolah informasi yang terdapat pada suatu gambar untuk keperluan pengenalan objek secara otomatis.

Kebutuhan untuk memproses sebuah gambar dengan cepat dalam satu aplikasi merupakan salah satu masalah utama dalam melakukan pengolahan citra, sedangkan untuk aplikasi yang bekerja secara real time lebih bergantung pada pemrosesan piksel atau sinyal yang cepat daripada metode optimisasi lain yang rumit dan memakan waktu (Chun Jen Chen, 2003).

Pengolahan citra dan pengenalan pola menjadi bagian dari proses pengenalan citra. Kedua aplikasi ini akan saling melengkapi untuk mendapatkan ciri khas dari suatu citra yang hendak dikenali. Secara umum tahapan pengolahan citra digital

meliputi akusisi citra, peningkatan kualitas citra, segmentasi citra, representasi dan uraian, pengenalan dan interpretasi.

2.3.2 Konversi RGB To Gray

Proses yang sering dilakukan pada saat *image processing* adalah konversi citra berwarna menjadi *grayscale*. Input citra yang diterima dari *web cam* berupa gambar RGB (Red Green Blue), sehingga perlu dilakukan proses konversi untuk menyederhanakan citra, dengan tujuan mengurangi tingkat kesalahan pada tahap-tahap berikutnya.

Citra RGB terdiri dari 3 *layer* matrik, yaitu *Red layer, Green layer* dan *Blue layer*, maka pada proses konversi RGB *to Gray* dilakukan tranformasi dari sebuah image berwarna 24 bit dengan 3 *channel* menjadi sebuah *image grayscale* 8 bit dengan *single channel*. Konversi ini dilakukan dengan melakukan penjumlahan nilai bobot dari warna merah, biru, dan hijau dari citra berwarna. Rumus yang digunakan untuk mengkonversi warna ditunjukkan dalam Persamaan (2-1).

$$Y = 0.299R + 0.587G + 0.114B \quad (2\text{-}1)$$

2.4 Background Subtraction

Background subtraction, juga dikenal sebagai deteksi gambar depan (*foreground)* adalah teknik di bidang pengolahan citra dan komputer visi, latar depan gambar yang diekstrak untuk diolah lebih lanjut (untuk pengenalan obyek, dll.). Pada mumnya daerah gambar menarik berupa sebuah objek (manusia, mobil, teks) yang terdapat pada gambar depannya (*foreground)*. Setelah tahap *preprocessing* gambar, lokalisasi objek diperlukan dengan menggunakan teknik ini. *Background subtraction* adalah pendekatan yang secara luas digunakan untuk mendeteksi benda-benda

bergerak dalam video dari kamera statis. Salah satu metode dari *background subtraction* adalah pengurangan *frame*. Pengurangan nilai absolut dari frame ditunjukkan dalam Persamaan (2-2):

$$? (? + 1) = |? (?, ?, ? + 1) ? ? (?, ?, ?)| \quad (2\text{-}2)$$

Dalam persamaan *background* diasumsikan terdapat pada *frame* pada waktu *t* dan yang objek dianggap bergerak *(foreground)*dinyatakan dengan *D*. Selisih dari *foreground* dan *background* dapat diperoleh dengan hanya melibatkan dua *frame.* Metode ini hanya dapat digunakan dalam kasus ketika *foreground*selalu dalam posisi bergerak dan *background*yang didapatkan dalam posisi statis.

Sebuah nilai ambang *Th* dapat digunakan untuk memperbaiki kualitas pemisahan *foreground* dan *background.* Nilai ambang ditunjukkan dalam Persamaan (2-3).

$$|? (?, ?, ?) - ? (?, ?, ? + 1)| > ? h \quad (2\text{-}3)$$

Dalam persamaan tersebut selisih dari intensitas nilai *pixel* difilter berdasarkan nilai *Th* yang ditetapkan. Tingkat akurasi dalam metode pengurangan *frame* bergantung pada kecepatan pergerakan dalam suatu *scene*. Semakin cepat pergerakan yang terjadi maka diperlukan nilai ambang yang lebih besar.

2.5 Image Morphology

OpenCV menyediakan antar muka yang cepat dan mudah untuk melakukan proses transformasi morfologi (J. Sierra, 1983). Robert Laganiere (2009:115) menyebutkan dasar transformasi morfologi disebut dilasi dan erosi. Keduanya

muncul dalam konteks variasi yang luas seperti menghilangkan *noise,* mengisolasi elemen individual, dan menggabungkan elemen yang terpisah dalam sebuah gambar.

Dilasi adalah sebuah konvulsi dari sebuah gambar dengan sebuah elemen yang disebut *kernel. Kernel* tersebut memiliki sebuah titik jangkar yang terdefinisi, sering berupa sebuah persegi atau cakram dengan titik jangkar di tengahnya dan merupakan sebuah *template* yang digunakan untuk memindai sebuah gambar. Efek dari dilasi adalah melokal-maksimalkan elemen.

Operasi dilasi sering digunakan ketika berusaha menemukan komponen yang terhubung (seperti area diskrit yang luas dengan intensitas atau warna piksel yang sama). Proses dilasi pada ukuran beberapa elemen yang kecil dan berdekatan akan meleburkan elemen-elemen tersebut menjadi satu.

Erosi adalah operasi kebalikan dari dilasi. Erosi adalah operasi yang ekivalen dengan menghirung sebuah area lokal-minimum dari *kernel.* Erosi menghasilkan gambar baru yang lebih kecil daripada dari gambar asal. Pada umumnya Dilasi memperbesar gambar asal sedangkan erosi menyempitkan gambar asal. Operasi erosi sering digunakan untuk mengurangi bintik *noise* yang muncul dalam gambar. Noise tidak akan muncul jika berupa bintik dan dilakukan proses erosi yang mengurangiukurannya, sedangkan gambar yang ukurannya besar tidak akan terpengaruh dengan proses tersebut.

Morfologi terbuka adalah kombinasi dari proses yang didahului operasi erosi dan dilanjutkan dengan operasi dilasi. Morfologi ini sering digunakan untuk menghitung daerah pada gambar biner (Robert Laganiere, 2009:221).

2.6 Blob Detection

Dalam pengolahan citra, mendeteksi objek *low-level* dalam sebuah gambar merupakan hal yang sangat penting. Objek dalam bentuk dua atau tiga dimensi

tersebut biasa disebut *blob*. Bentuk *blob* timbul dalam cara yang berbeda bergantung pada ukuran dan dapat dideteksi dengan menggunakan metode sederhana dalam sebuah representasi gambar.

Ada beberapa teori mengenai definisi *blob* yang dikemukakan oleh beberapa orang, yaitu antara lain:

1) Lindenberg (1993) mendefinisikan sebuah *blob* sebagai sebuah daerah yang terkait setidaknyasatu daerah yang ekstrim, baik maksimum maupun minimum atau sebuah daerah terang maupun daerah gelap.
2) Hinz (2005) mendefinisikan sebuah *blob* sebagai sebuah bujur sangkar dengan daerah yang homogen.
3) Rosenfeld dan Sher (1998) mendefinisikan sebuah *blob* sebagai persimpangan dari garis tegak lurus ke daerah tepi tangennya, dikelilingi oleh enam arah atau lebih.
4) Yang dan Parvin mendefinisikan sebuah *blob* 3D sebagai fitur berbentuk bulat dalam skala ruang.

Untuk berbagai macam tipe *blob,* diperlukan berbagai metode yang berbeda yang harus memenuhi syarat:

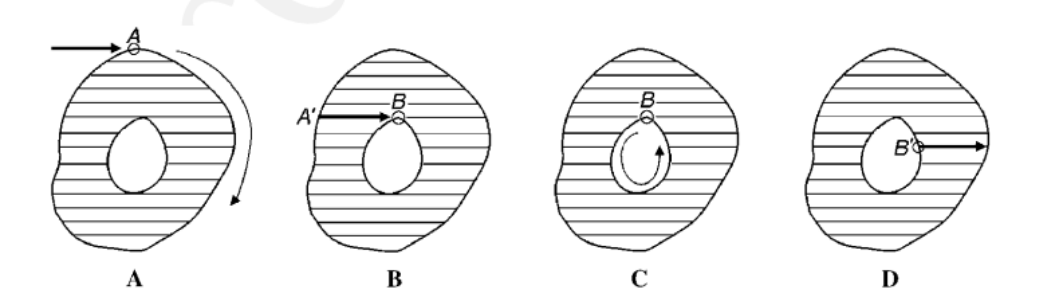
1) Handal. Metode pada citra *low-level* harus tahan terhadap *noise.*
2) Akurasi. Dalam metrologi, citra hasil dengan akurasi yang tinggi sangat diperlukan.
3) Selektabilitas. *Blob* dalam ukuran yang berbeda harus dapat terdeteksi.
4) Kecepatan. Metode tersebut harus mendekati proses *real-time.*
5) Mempunyai parameter yang sedikit dan semantik. Metode tersebut harus mudah dimengerti agar mudah disesuaikan.

Dalam metode *blobtracking* yang ditulis oleh Fu Chang, dkk (2003) pada bidang dua dimensi citra biner dipindai dari atas ke bawah dan dari kiri ke kanan per setiap baris. Secara konseptual, operasi dapat dibagi menjadi empat langkah utama yang ditunjukkan dalam Gambar. 2.2 (A-D). Dalam Gambar. 2.2(A), ketika titik

kontur eksternal A bertemu pertama kalinya, jejak kontur dibuat secara lengkap sampai pemindaian kembali ke A. Label tertentu ditetapkan untuk titik A dan semua titik kontur tersebut.

Dalam Gambar. 2.2 (B), ketika label kontur titik eksternal A' ditemukan, garis pindai diikuti untuk menemukan semua piksel hitam berikutnya, s (jika ada), dan diberikan label yang sama dengan A'.

Dalam Gambar. 2.2 (C), ketika titik kontur internal B ditemukan pertama kalinya, label B ditetapkan sama seperti kontur eksternal dari komponen yang sama. Kontur internal yang mengandung B kemudian dilacak dan ditetapkan ke semua titik kontur label yang sama seperti B. Proses pendeteksian *blob* ditunjukkan dalam Gambar 2.2.



Gambar 2.2 Proses pendeteksian Blob

Dalam Gambar 2.2 (D) , ketika titik kontur internal berlabel B' ditemukan, garis *scan* ditelusuri untuk menemukan semua piksel hitam berikutnya , s, (jika ada) dan diberikan label yang sama dengan B ' .

Dalam prosedur di atas, *single pass* atas gambar hanya dibuat dan ditetapkan ke setiap titik komponen baik label baru atau label yang sama sebagai titik sebelumnya di garis pindai.

Untuk mempermudah, diasumsikan bahwa piksel pada baris paling atas semua putih (jika tidak, akan ditambahkan baris *dummy pixel* berwarna putih). Untuk

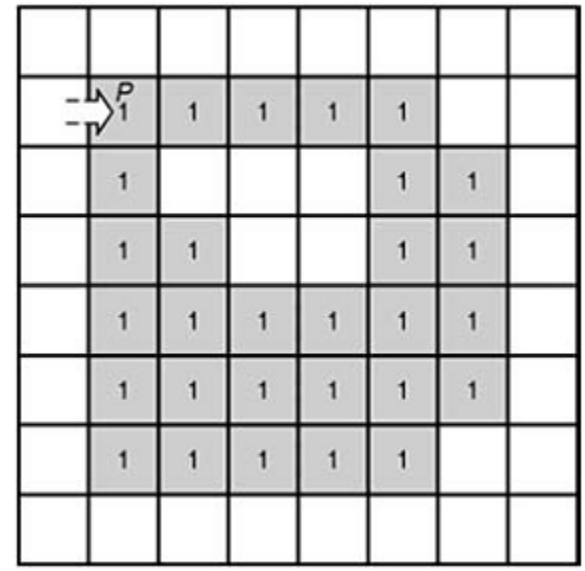
gambar dokumen yang diberikan, I, akan dikaitkan dengan gambar L yang menyertainya , yang menyimpan informasi label.

Pada Awalnya, semua titik diatur ke 0 (yaitu belum terlabeli). Kemudian dimulai pememindaian untuk menemukan piksel hitam. Variabel C dijadikan indeks label untuk komponen. Awalnya C diatur dengan nilai 1.

Empat langkah konseptual tersebut di atas dapat dikurangi menjadi tiga langkah logis. Langkah pertama berhubungan dengan pertemuan antara titik eksternal dengan semua titik pada kontur tersebut. Langkah kedua berhubungan dengan pertemuan titik internal dengan semua titik pada kontur tersebut, dan langkah ketiga berhubungan dengan titik yang tidak ditangani pada dua langkah pertama.

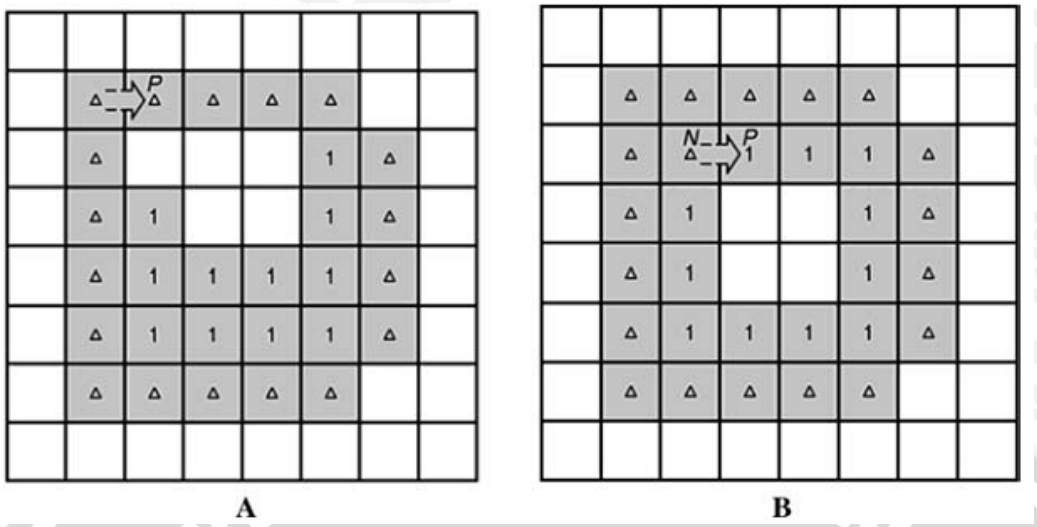
Misalkan P adalah titik tertentu yang berkaitan dengan algoritma pemrograman *blobtrack* maka:

1) Langkah pertama: Jika P belum terlabeli, dan piksel di atasnya adalah piksel putih dalam Gambar 2.3, P seharusnya terletak pada sebuah kontur eksternal dari sebuah komponen pertemuan baru. Kemudian P diberikan label C, ketika mengeksekusi *countur tracing* (sebuah prosedur pelacakan kontur yang akan dibahas kemudian) untuk menemukan semua eksternal kontur tersebut dan mendandainya dengan label C untuk semua piksel pada kontur tersebut. Kemudian label C dberikan nilai 1. Langkah pertama proses *blobtracking* ditunjukkan dalam Gambar 2.3.



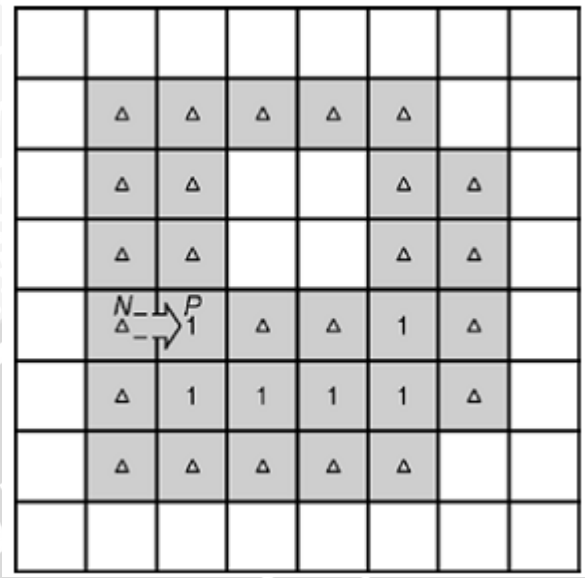
Gambar 2.3 P Merupakan Titik Awak Dari Kontur Luar. 1 Merupakan *Pixel* Hitam yang Belum Diberi Label

2) Langkah kedua: Jika piksel di bawah P adalah piksel putih yang belum diberi tanda, P kemungkinan adalah pertemuan internal kontur baru. Ada dua kemungkan. Pertama, P sudah diberi label dalam Gambar 2.4 (A). Pada kasus ini, P juga merupakan sebuah piksel kontur eksternal. Kedua, P belum diberi label, ditunjukkan dalam gambar 2.4 (B). Dalam kasus ini, titik N pada garis pindai sebelumnya (titik sebelah kiri P) seharusnya sudah terlabeli. Kemudian dilakukan penugasan P memiliki label yang sama dengan N. Pada kedua kasus tersebutdilakukan proses eksekusi *contour tracing*untuk menemukan kontur internal berisi P, dan melakukan penugasan dengan label yang sama kepada semua piksel pada kontur tersebut. Langkah kedua proses *blibtracking* ditunjukkan dalam Gambar 2.4.



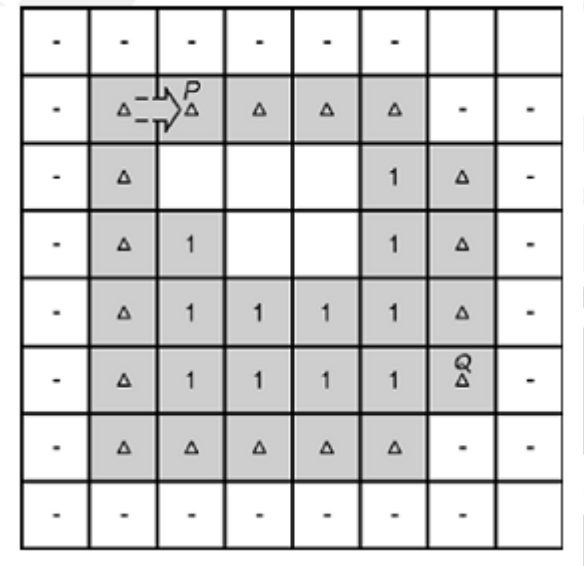
Gambar 2.4 (A) P Merupakan Titik Awal Kontur Internal yang Terletak pada Kontur Luar. (B) P titik Awal Kontur Internal Namun Bukan Merupakan Kontur Luar. Δ Merupakan *Pixel* Hitam Yang Terlabeli

3) Langkah ketiga: Jika P bukanlah titik yang terdapat pada langkah pertama atau kedua (P bukan sebuah titik kontur), maka sebelah kiri P bukanlah piksel yang telah terlabeli, oleh karena itu dilakukan penugasan P memiliki nilai sama dengan N. Langkah ketiga proses *blobtracking* ditunjukkan dalam Gambar 2.5



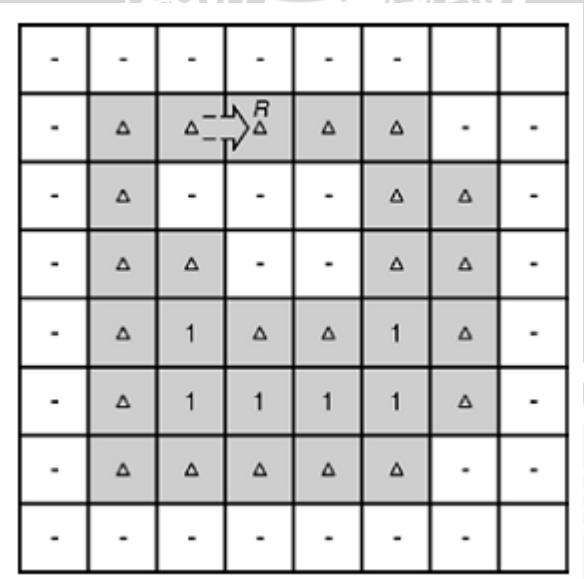
Gambar 2.5 P Merupakan *Pixel* Hitam yang Belum Terlabeli dan N Sudah Terlabeli

Seperti ditunjukkan dalam gambar 2.5, untuk menghindari eksekusi *counter tracing* pada titik Q, titik putih di sekeliling titik Q ditandai dengan integer negatif. Dengan demikianpada waktu garis pindai melintasi Q, piksel di bawah Q bukanlah sebuah piksel putih yang belim diberi tanda. Di lain sisi, titik di bawah piksel pertemuan kontur internal pertama P (gambar 5) masih belum tertandai karena kontur internal yang berisi piksel tersebut belum ter-*trace*. Penandaan titik Q ditunjukkan dalam Gambar 2.6.



Gambar 2.6 *Pixel* Putih yang Mengelilingi Kontur Diberi Tanda Integer Negatif

Dengan menandai piksel putih di sekelilingnya dapat dipastikan bahwa masing masing kontur internal dilacak sekali saja. Seperti ditunjukkan dalam Gambar 2.6, ketika kontur internal telah ter-*trace*, piksel disekitar bawah R bukan lagi merupakan piksel yang belum diberi tanda. Dengan demikian penjejakan kontur internal sekali lagi ketika R bertemu dengan garis pindai dapat dihindari. Penandaan di sekitar piksel putih dengan integer negatif ditunjukkan dalam Gambar 2.7.

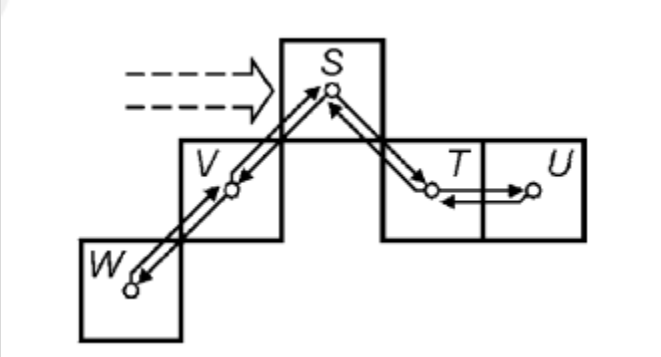


Gambar 2.7 *Pixel* Putih yang Mengelilingi Kontur Diberi Tanda Integer Negatif Ketika Kontur Internal Berhasil Di-*Trace*

Operasi penandaan piksel di sekitar piksel putih dengan sebuah integer negatif termasuk dalam prosedur *tracer*, yang kemudian disebut prosedur *contour tracing.*

2.6.1 Contour Tracing

Tujuan dari prosedur *contour tracing* adalah menemukan kontur eksternal maupun internal pada titik S pada Gambar 2.7. Pada titik S awalnya dieksekusi sebuah prosedur *tracer*. Jika S diindikasikan oleh *tracer* sebagai objek yang terisolasi, maka titik akhir dari *contour tracing* telah ditemukan. Jika terjadi sebaliknya, *tracer* akan memunculkan titik kontur yang mengikuti S, disebut titik T. Pengeksekusian *tracer* bisa dilanjutkan untuk menemukan titik kontur yang mengikuti T dan seterusnya, sampai memenuhi dua kondisi: (1) *Tracer* menghasilkan S dan (2) titik kontur yang mengikuti S adalah T. Prosedur tersebut dihentikan hanya ketika dua kondisi tersebut terjadi. Seperti yang ditunjukkan dalam Gambar 2.7, S adalah titik mula dan T adalah titik kontur setelah S. Jalur yang dilalui *tracer* adalah STUTSVWVS. Proses menapak komponen ditunjukkan dalam Gambar 2.7



Gambar 2.7 Proses Menapak Sebuah Komponen

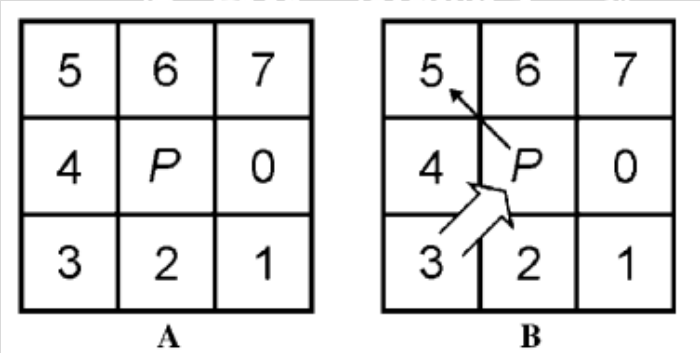
2.6.2 Tracer

Untuk titik kontur yang telah ditetapkan, P, tujuan dari *tracer* adalah untuk menemukan delapan titik di sekitar P yang mengikuti P. Posisi dari tiap sisi dari P

ditugaskan oleh sebuah indeks seperti yang ditunjukkan pada Gambar 2.8(A). Hasil yang didapatkan dalam sebuah arah jarum jam dari posisi awal ditentukan dengan cara sebagai berikut:

- Jika P adalah titik awal dari sebuah kontur eksternal, maka posisi awal diatur bernilai 7 (kanan atas) karena titik di atas P dikenal sebagai piksel putih dan titik berikutnya dalam arah jarum jam menunjuk pada posisi 7
- Jika P adalah titik awal dari sebuah kontur internal, posisi awal diatur bernilai 3 karena titik di bawah P dikenali sebagai titik piksel putih dan titik berikutnya menurut arah jarum jam adalah pada arah posisi 3 (kiri bawah)

Sebagai contoh jika posisi awal di atur bernilai 5 karena piksel pada posisi 4 seharusnya telah dilalui, ditunjukkan dalam Gambar 2.8(B). Pada umumnya ketika P bukan titik awal sebuah kontur, baik kontur internal ataupun eksternal, posisi awal pencarian diatur sebesar d+2, d adalah posisi dari titik kontur sebelumnya. Proses penomoran indeks di sekitar titik P ditunjukkan dalam Gambar 2.8.



Gambar 2.8. (A) Sekeliling TItik P Diberi Indeks 1 sampai 7 (B) Jika Titik Kontur Sebelumnya Pada Arah 3, Pencarian Titik Selanjutnya Pada Arah 5

Sekali posisi awal ditentukan, dihasilkan dalam arah jarum jam arah untuk mengetahui lokasi piksel hitam setelahnya. Piksel tersebut adalah piksel yang mengikuti P. Jika tidak terdapat piksel hitam yang ditemukan dalam satu siklus, P diidentifikasikan sebagai titik yang terisolasi (piksel tunggal).

BAB III

METODOLOGI

Dalam penyusunan skripsi ini, dirancang suatu program penghitung kendaraan satu Arah, sedangkan metode yag digunakan bersifat aplikatif, yakni merupakan perencanaan dan pembuatan sistem dengan langkah-langkah sebagai berikut :

3.1. Studi Literatur

Mempelajari literatur-literatur atau buku serta dokumen-dokumen yang berhubungan dengan bahasa pemrograman C++, pemrosesan citra digital dan pustaka OPENCV.

3.2. Analisis Kebutuhan

Analisis kebutuhan bertujuan untuk mendapatkan semua kebutuhan yang diperlukan atas sistem yang akan dibangun. Perangkat yang digunakan untuk menunjang pembuatan program penghitung kendaraan satu arah untuk arduino, antara lain:

a. Perangkat Keras :

1) Kamera Digital 2 Mega Pixel
2) Kabel USB
3) Perangkat komputer jinjing dengan spesifikasi:

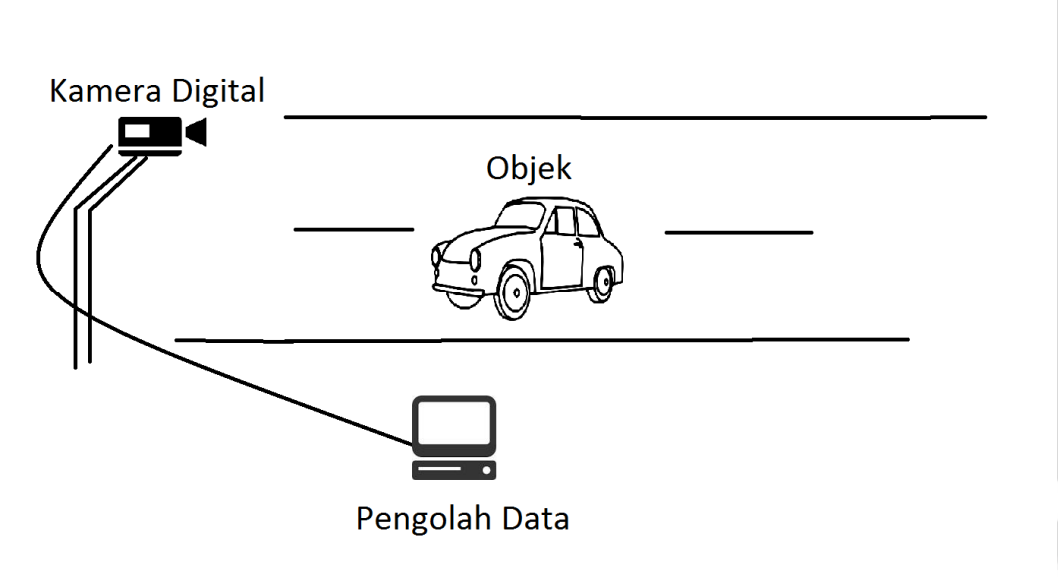
- CPU AMD Dual Core 1 GHz
- RAM 2 Gigabytes
- Hard Disk Internal 250 Gigabytes

b. Perangkat Lunak :

1) Sistem Operasi Windows 8
2) Microsoft Visual C++ 2010
3) Pustaka OPENCV 2.4.4
4) Pustaka CVBlob

3.3. Perancangan dan Implementasi Sistem

Skema gambar implementasi perangkat penghitung kendaraan ditunjukkan dalam Gambar 3.1.



Gambar 3.1 Implementasi Perangkat Penghitung Kendaraan

Cara kerja dari sistem tersebut dapat dijelaskan secara singkat sebagai berikut :

1) Kamera digital mengubah sinyal analog berupa intensitas cahaya menjadi sinyal digital

2) Sinyal digital yang berasal dari kamera ditransmisikan melalui kabel USB menuji mesin pengolah data
3) Dengan menggunakan perangkat komputer data berbentuk digital kemudian diolah dan didapatkan urutan gambar yang kemudian diolah dalam program yang akan dirancang

3.4 Pengujian Sistem

Pengujian pada skripsi ini dilakukan agar dapat menunjukkan bahwa sistem telah mampu bekerja sesuai dengan spesifikasi dari kebutuhan yang melandasinya. Serta menjelaskan langkah-langkah pengujian dari sistem yang telah dibuat dan membahas hasil pengujiannya.

3.5. Pengambilan Kesimpulan

Pengambilan kesimpulan dilakukan setelah semua tahapan perancangan, implementasi dan pengujian sistem telah selesai dilakukan. Kesimpulan diambil dari hasil data-data pengujian dan analisis terhadap sistem yang dibangun. Tahap terakhir dari penulisan adalah pemberian saran yang dimaksudkan untuk memperbaiki kesalahan-kesalahan yang terjadi dan menyempurnakan penulisan serta untuk memberikan pertimbangan atas pengembangan sistem selanjutnya.

BAB IV

PERANCANGAN DAN IMPLEMENTASI

Bab ini membahas tahapan-tahapan dalam perancangan Program Penghitung Jumlah Kendaraan Di Lintasan Jalan Raya Satu Arah Menggunakan Bahasa Pemrograman C++ Dengan Pustaka Opencv. Pembuatan secara bertahap dimaksudkan agar mudah dalam melakukan analisis pada setiap bagiannya maupun secara keseluruhan.

4.1 Perancangan Secara Umum

Perancangan sistem secara umum merupakan tahap awal sebagai acuan dalam perancangan sistem yang akan dibuat. Pada bagian ini akan dibahas mengenai garis besar dari masing-masing bagian dari sub-program dan proses apa yang akan terjadi dari tiap-tiap bagiannya.

Perancangan program terbagi menjadi empat bagian yang terdiri dari inisialisasi masukan video, *background subtraction*, pelacakan *blob* (*blobtracking)* dan algoritma penghitung jumlah kendaraan.

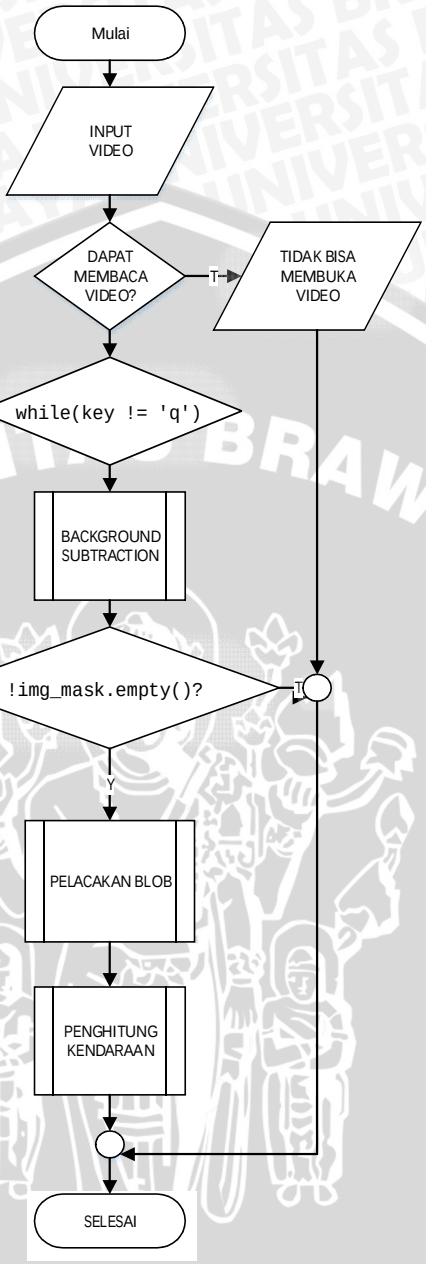
Untuk masukan berupa video didapatkan dari pengambilan *video* secara *real-time* melalui kamera digital. Masukan berupa video ini kemudian akan dikenali program sebagai kumpulan larik yang setiap lariknya berisi informasi dari setiap piksel dari frame dalam video tersebut.

Masukan berupa frame tersebut pada awalnya belum dapat dibedakan antara objek statis dalam video (*background)* dan objek dinamisnya(*foreground)*. Pemisahan antara *background* dan *foreground* diperlukan karena dalam program ini informasi dari *foreground* diperlukan untuk mengetahui berapa objek yang terdapat dalam *foreground.* Proses pemisahan ini dinamakan *background subtraction.*

Setelah *foreground* dapat dipisahkan dari *background,* pemindaian *(scanning)* dilakukan pada objek dilakukan untuk memisahkan setiap gumpalan (*blob)* yang terdapat dalam *foreground*. Setiap gumpalan tang berhasil dipisahkan

satu dengan yang lainnya akan dibandingkan dengan *blob* yang terdapat dalam *frame* berikutnya sehingga dapat diperoleh informasi tentang berapa besar, jalur manakah yang dilalui, dan berapa jumlah dari setiap *blob* yang berhasil terdeteksi. Rangkaian proses tersebut disebut dengan *blobtracking*.

Setelah melalui proses *blobtracking*, *blob* yang berisi informasi tentang posisinya dalam *frame* tersebut akan dihitung jumlahnya ketika melewati posisi tertentu yang telah ditetapkan. Proses tersebut dilakukan untuk mengaetahui berapa jumlah kendaraan yang akan dihitung. Perancangan sistem secara keseluruhan ditunjukkan dalam Gambar 4.1.

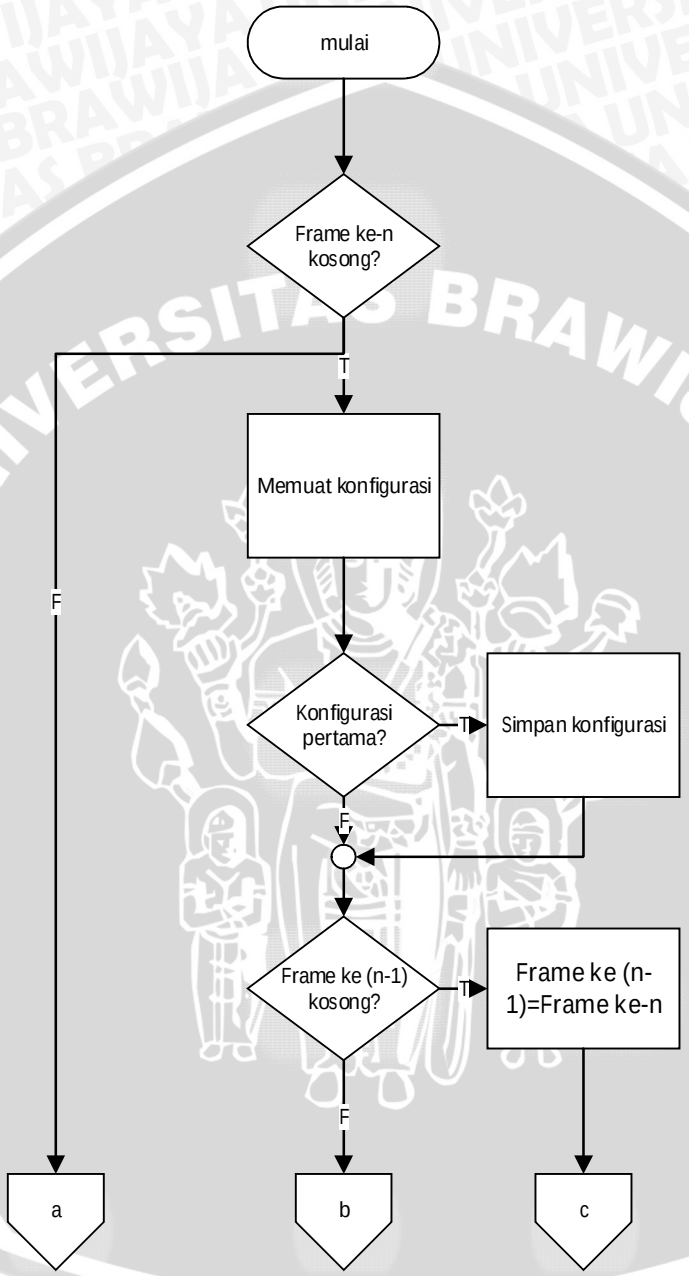


Gambar 4.1 Diagram Blok Program.

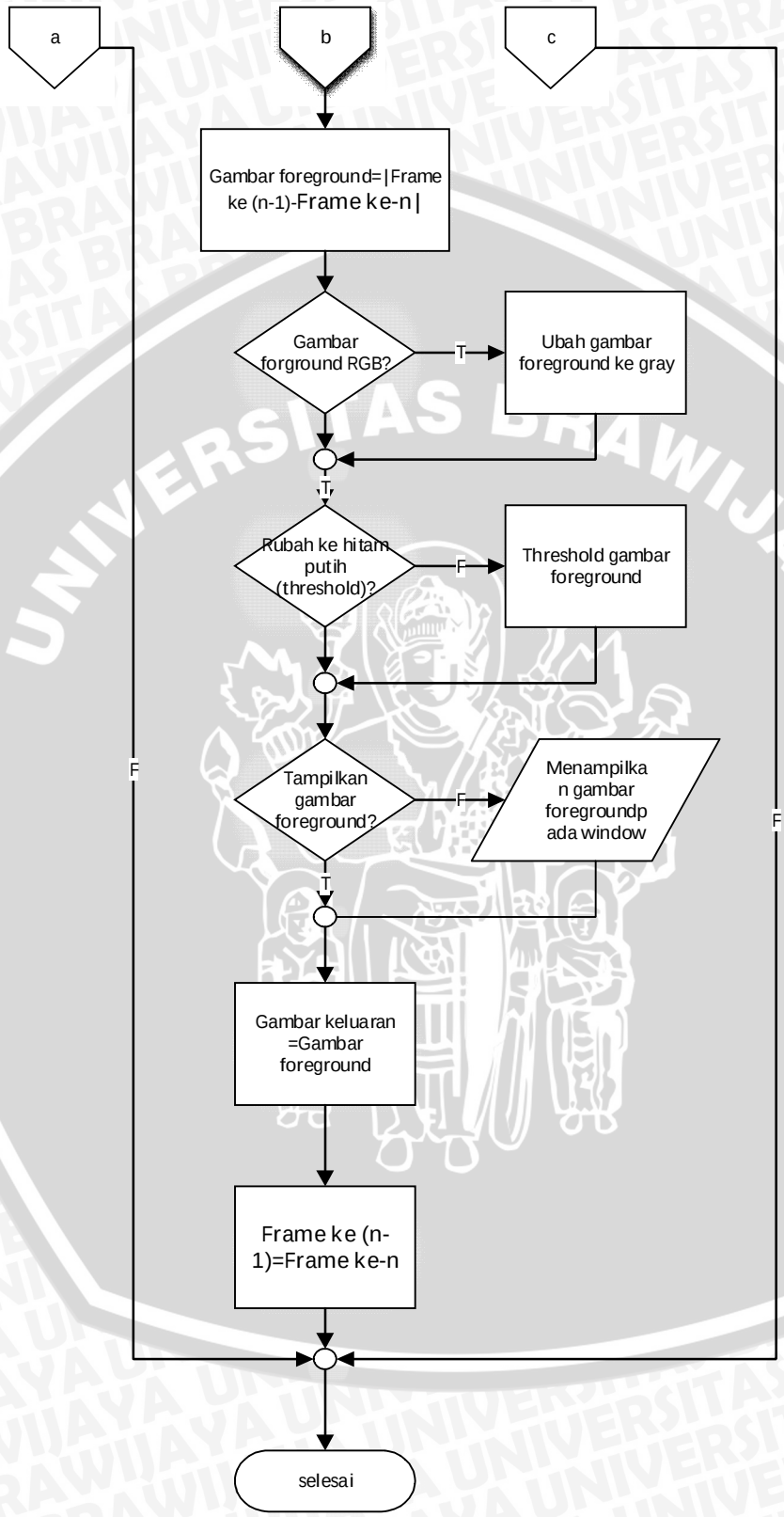
4.2 Perancangan *Backround substraction*

Dalam perancangan ini data masukan berasal dari video dari kamera yang berisi informasi biner yang memuat intensitas piksel yang ada di dalamnya. Sedangkan output dari *background subtraction* berupa gambar *foreground* yang

nantinya akan diproses dalam proses *blobtracking*. Diagram alir perancangan *background subtraction* ditunjukkan dalam Gambar 4.2 dan Gambar 4.3.



Gambar 4.2 Diagram Alir *Background Subtraction*


a
b
c
Gambar foreground=|Frame ke (n-1)-Frame ke-n|
Gambar forground RGB?
T
Ubah gambar foreground ke gray
T
Rubah ke hitam putih (threshold)?
F
Threshold gambar foreground
Tampilkan gambar foreground?
F
Menampilkan gambar foregroundpada window
T
F
F
Gambar keluaran =Gambar foreground
Frame ke (n-1)=Frame ke-n
selesai

Gambar 4.3. Diagram Alir *Background Subtraction* (Lanjutan)

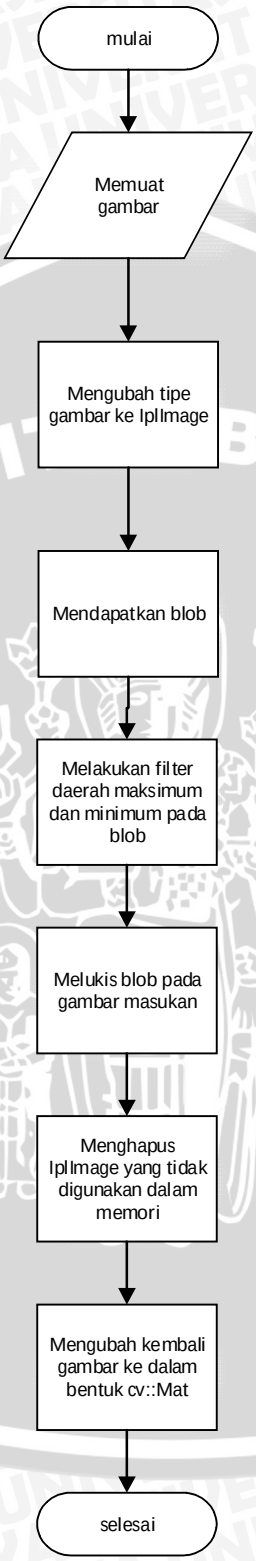
Diagram alir proses *background subtraction* dijelaskan sebagai berikut:

1) Di awal program dilakukan pengecekan *frame* masukan, jika tidak terdapat *frame masukan* maka keluar dari program.
2) Dilakukan pengecekan setelan konfigurasi *background subtraction,* jika variable *firstime* bernilai *true* maka konfigurasi tersebut disimpan dalam *FrameDifferenceBGS.xml*
3) Dilakukan pengecekan *frame* yang sudah tersimpan sebelum *frame* masukan, jika belum ada *frame* tersimpan maka *frame* masukan disalin ke *frame frame* yang sudah tersimpan.
4) Nilai pada *frame* masukan dan *frame* yang telah tersimpan dikurangkan dan nilai hasil pengurangan disimpan ke dalam gambar *foreground.*
5) Jika gambar *foreground* berupa gambar RGB, gambar tersebut dirubah menjadi gambar abu-abu (*grey)*
6) Gambar abu-abu kemudian ditubah menjadi gambar bernilai biner dengan merubah nilai piksel maksimum apabila nilai piksel abu-abu berada di atas nilai ambang (*threshold)* dan bernilai minimum apabila berada di bawah nilai ambang
7) Menampilkan gambar *foreground* pada window
8) Menyimpan gambar foreground pada gambar keluaran
9) Menyimpan gambar gambar *foreground* ke dalam *frame* masukan yang telah tersimpan

4.3 Perancangan *Blobtracking*

Pada perancangan ini pustaka *cvblob* digunakan untuk mempermudah programmer dalam memisahkan tiap-tiap gumpalan yang terdapat dalam suatu gambar dan kemudian melabelinya sehingga antara gumpalan yang satu dengan

gumpalan yang lainnya dapat dibedakan dengan identitas yang berbeda. Diagram alir perancangan *blobtracking* ditunjukkan dalam Gambar 4.4.


mulai
Memuat gambar
Mengubah tipe gambar ke IplImage
Mendapatkan blob
Melakukan filter daerah maksimum dan minimum pada blob
Melukis blob pada gambar masukan
Menghapus IplImage yang tidak digunakan dalam memori
Mengubah kembali gambar ke dalam bentuk cv::Mat
selesai

Gambar 4.4 Diagram Blok *Blobtracking*

Diagram alir proses *blobtracking* dijelaskan sebagai berikut:

1) Pada awal proses gambar didapatkan dari proses *background subtraction.*
2) Dilakukan proses morfologi terbuka pada gambar untuk memisahkan *blob* yang berdekatan dan menghilangkan *noise.*
3) Gambar tersebut kemudian diubah tipe datanya menjadi IplImage.
4) Setelah gambar bertipe IplImage, masing-masing blob didapatkan dan memiliki karakteristik panjang, lebar, titik pusat (*centroid)* dan nomor identifikasi yang dapat dibedakan satu dengan yang lainnya.
5) Blob kemudian difilter, Blob yang tidak memenuhi karakteristik yang diinginkan kemudian dihapus sehingga hanya menyisakan blob yang dibutuhkan.
6) Blob yang telah teridentifikasi dilukiskan ke dalam gambar masukan pada jendela *blobtrack.*
7) Gambar *blobtrack* yang memiliki tipe data *iplImage* dirubah kembali menjadi gambar bertipe data *std::map.*

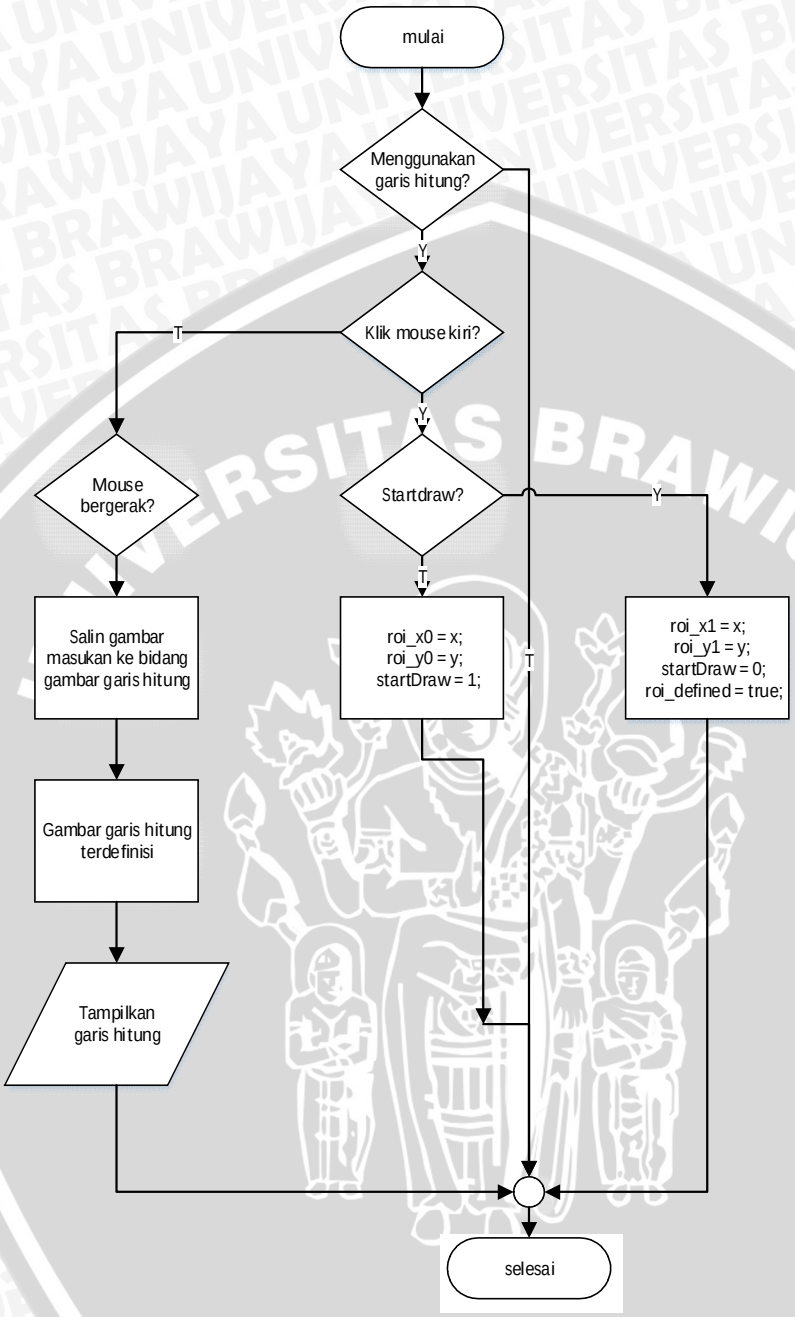
4.4 Penghitung Kendaraan

Dalam proses penghitungan kendaraan ini dibagi menjadi empat bagian yaitu pengaturan metode penghitungan, pengaturan garis hitung, proses penghitungan dan proses penandaan jalur (*track).*

4.4.1 Pengaturan Garis Hitung

Pengaturan garis hitung dilakukan untuk mendapatkan garis hitung yang paling sesuai dengan kondisi jalan yang akan diamati. Pengaturan garis hitung dilakukan dengan melakukan dua kali *mouse click* sehingga membentuk garis tegak lurus dengan arah objek yang akan dihitung. Penjelasan diagram alir proses penghitung kendaraan dijelaskan sebagai berikut:

1) Dilakukan pengecekan apakah program menggunakan garis hitung. Jika tidak, maka dilanjutkan dengan keluar dari *sub-program*.
2) Jika tombol *mouse* ditekan dilakukan pengecekan pada veriabel *startdraw*. Pada awal program *startdraw* diatur bernilai 0.
3) Titik di mana *mouse* pertama kali di *click* disimpan sebagai koordinat titik awal garis hitung. *Startdraw kemudian diatur bernilai 1*
4) Jika pengecakan *startdraw* kembali bernilai 1 maka pada titik *click* kedua disimpan sebagai titik koordinat akhir dari garis hitung.
5) Setelah *click* pertama, ketika kondisi *mouse* bergerak, garis hitung digambarkan melintang dari titik *click* pertama sampai kepada ujung koordinat *mouse* yang bergerak dan menjadi garis permanen setelah terjadi *click* yang kedua. Diagram alir perancangan garis hitung ditunjukkan dalam Gambar 4.5.



Gambar 4.5 Diagram Alir Pengaturan Garis Hitung

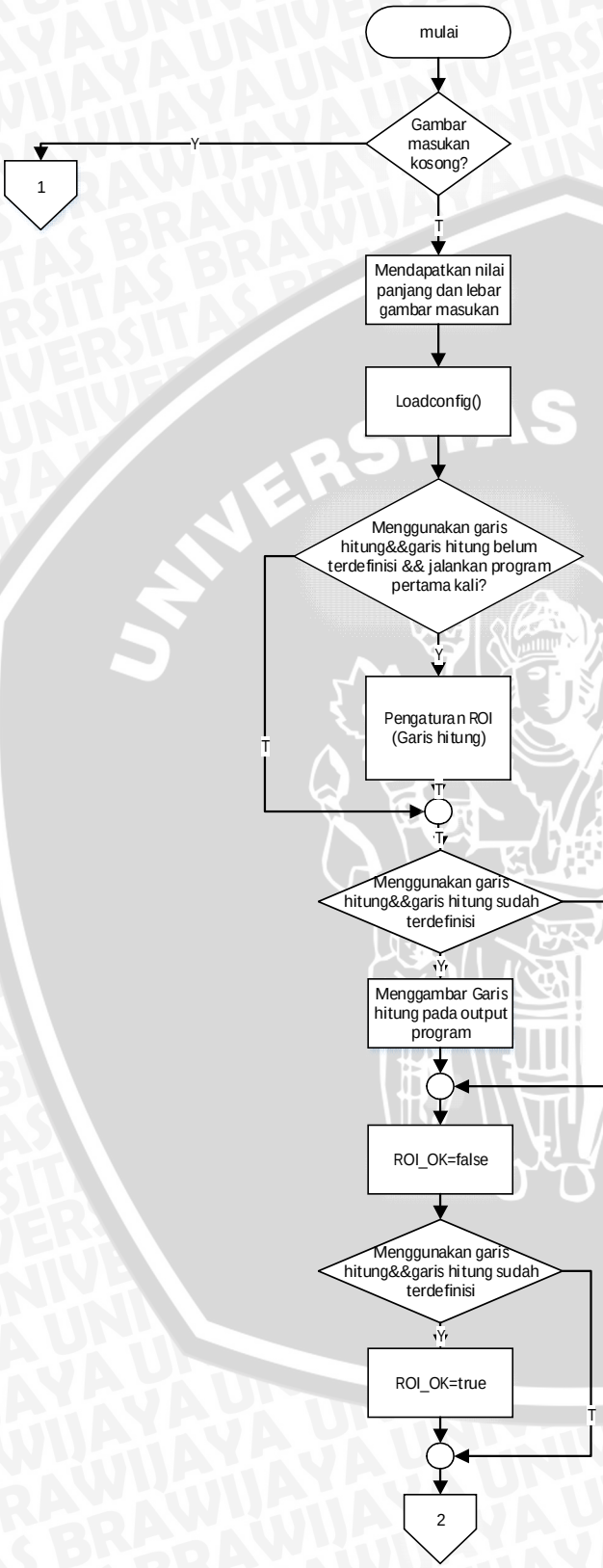
4.4.2 Perancangan Metode Penghitungan Kendaraan

Pada perancangan metode penghitungan kendaraan, *blob* hasil dari proses *blobtracking* dibandingkan posisi titik pusatnya dengan garis hitung yang telah ditetapkan. Setiap titik pusat melewati garis yang telah ditetapkan maka akan

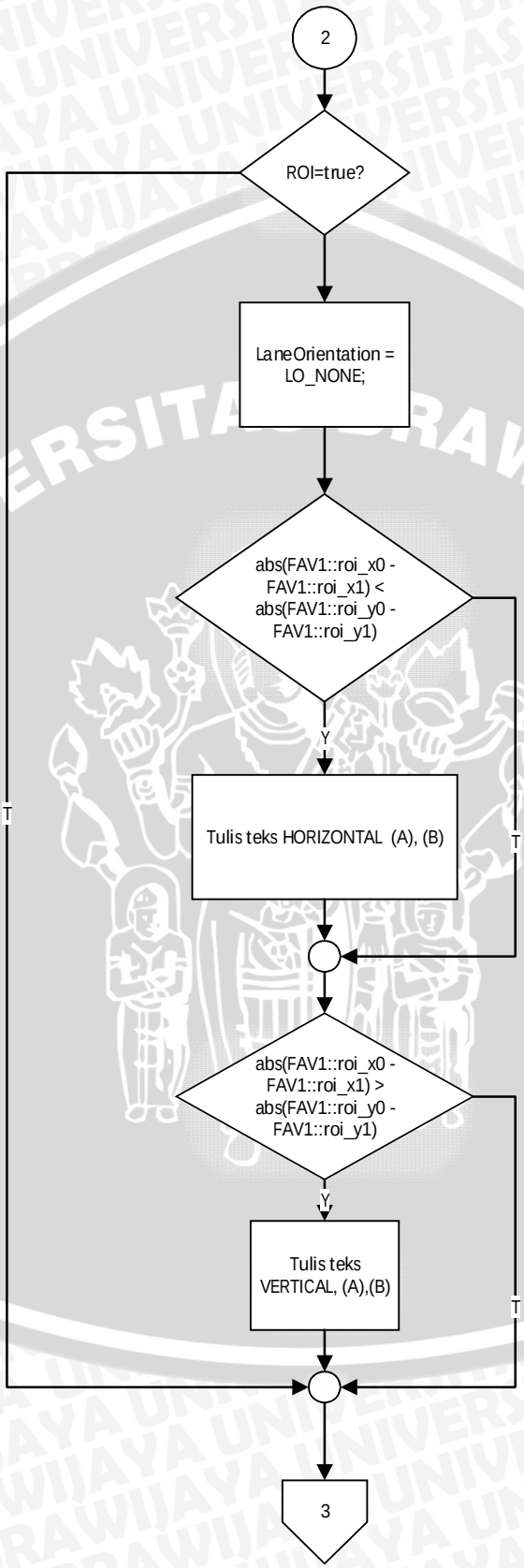
menambah jumlah kendaraan yang berhasil dihitung. Diagram alir proses perhitungan kendaraan ditunjukkan dalam Gambar 4.6.

Diagram alir proses penghitung kendaraan dijelaskan sebagai berikut:

1) Gambar masukan diperiksa apakah memiliki nilai data ataukah tidak/ kosong
2) Data masukan berupa gambar didapatkan nilai panjang dan lebarnya sebagai nilai yang digunakan untuk menentukan panjang dan lebar maksimum dalam area yang akan digunakan untuk menghitung kendaraan.
3) Apabila pengaturan menggunakan garis hitung, garis hitung belum terdefinisi dan program baru dijalankan pertama kali proses dilanjutkan dengan pengaturan garis hitung secara manual dengan melakukan *mouse click* pada kedua ujung garis yang akan ditentukan. Posisi dua titik ujung garis hitung tersebut akan disimpan dalam *vehiclecounting.xml.*
4) Posisi garis hitung kemudian akan ditampilkan dalam gambar keluaran ketika garis hitung telah terdefinisi. Setelah proses ini variable *ROI_OK* diberi nilai benar.
5) Langkah berikutnya adalah pengecekan apakah garis hitung sudah terdefinisi. Apabila terdefinisi dilakukan pengecekan apakah garis hitung yang terdefinisi berupa garis vertikal ataukah hotisontal
6) Garis didefinisikan horisontal apabila pengurangan absolut sumbu-X dua titik garis hitungnya lebih kecil daripada pengurangan sumbu-Ynya
7) Garis didefinisikan horisontal apabila pengurangan absolut sumbu-X dua titik garis hitungnya lebih besar daripada pengurangan sumbu-Ynya.
8) Untuk mempermudah analisis garis yang digunakan sebagai penanda, horisontal ataukah vertikal maka dituliskan apakah program menggunakan metode vertikal ataukah horisontal pada proses perhitungannya.


mulai
Gambar masukan kosong?
Y
1
T
Mendapatkan nilai panjang dan lebar gambar masukan
Loadconfig()
Menggunakan garis hitung&&garis hitung belum terdefinisi && jalankan program pertama kali?
Y
Pengaturan ROI (Garis hitung)
T
T
Menggunakan garis hitung&&garis hitung sudah terdefinisi
Y
Menggambar Garis hitung pada output program
T
ROI_OK=false
Menggunakan garis hitung&&garis hitung sudah terdefinisi
Y
ROI_OK=true
T
2

Gambar 4.6 Metode Penghitungan Kendaraan

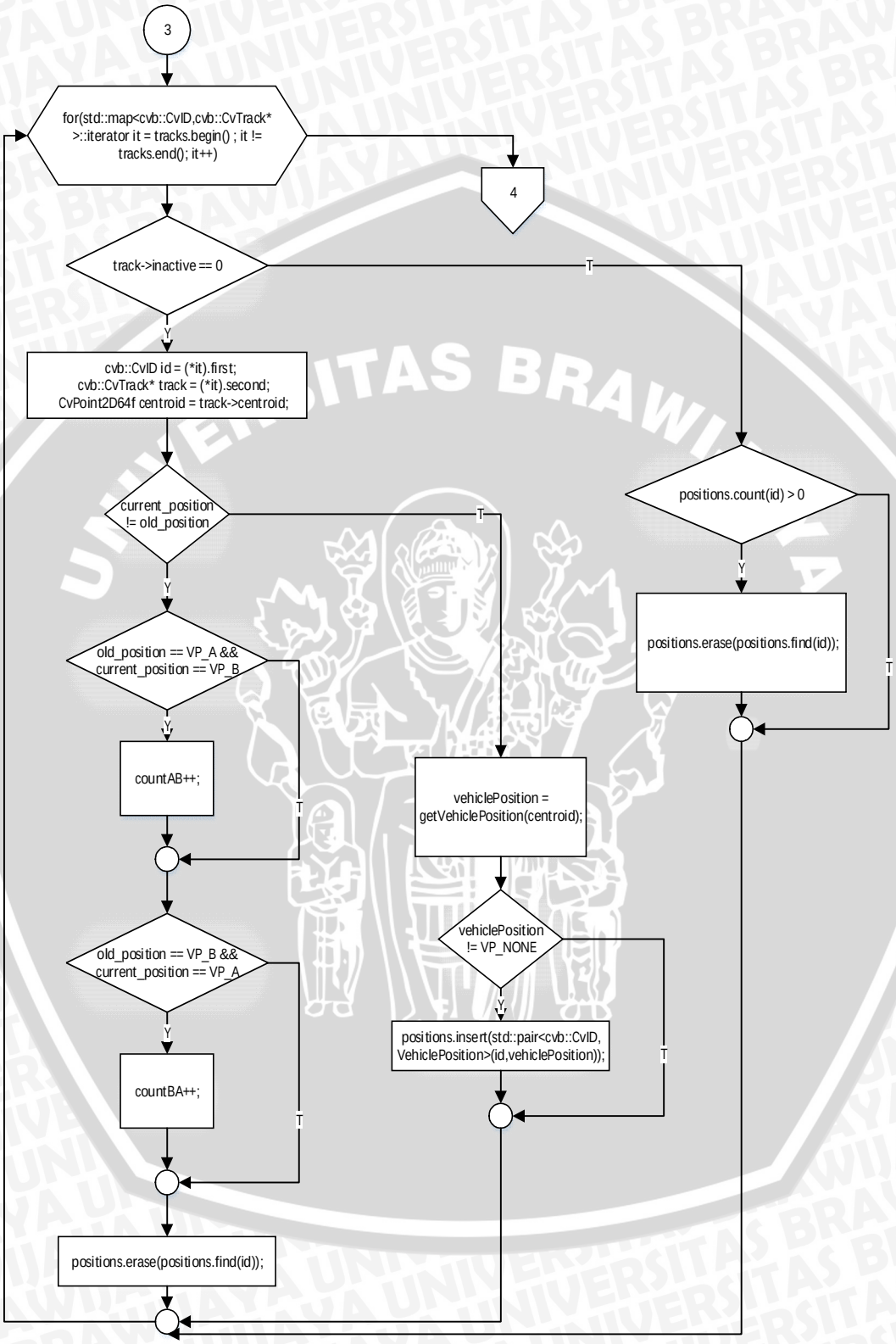

2
ROI=true?
LaneOrientation = LO_NONE;
abs(FAV1::roi_x0 - FAV1::roi_x1) < abs(FAV1::roi_y0 - FAV1::roi_y1)
Y
T
Tulis teks HORIZONTAL (A), (B)
T
abs(FAV1::roi_x0 - FAV1::roi_x1) > abs(FAV1::roi_y0 - FAV1::roi_y1)
Y
Tulis teks VERTICAL, (A),(B)
T
3

Gambar 4.7 Metode Penghitungan Kendaraan (Lanjutan)

4.4.3 Perancangan Penghitungan Kendaraan

Langkah berikutnya adalah dimulainya perhitungan kendaraan. Diagram alir proses penghitung kendaraan dijelaskan sebagai berikut:

1) Menerapkan proses iterasi pada *map* tracks yang berisi nomor identitas *blob* dan daftar *track* yang didapatkan dari proses *blobtracking.*
2) Kondisi *track* diperiksa apakah ada *frame* yang telah dalam kondisi *inactive.* Jika tidak terdapat *frame* dalam kondisi *inactive* maka *id* dan *tracks* hasil proses *blobtracking* disimpan dalam *map tracks.*
3) Dilakukan pengecekan kategori posisi yang dimiliki oleh *blob* dengan memasukkan nilai *centroid* pada *blob* tersebut ke dalam sub program *getVehicleposition. Blob* dikategorikan sebagai VP_A apabila terletak sebelum garis hitung(nilai *centroid*-nya lebih kecil daripada koordinat garis hitung dalam bidang), VP_B apabila terletak setelah garis hitung dan VP_NONE apabila bukan merupakanpada dua kategori sebelumnya.
4) Kategori posisi yang dimiliki *blob* dibandingkan dengan blob sebelumnya dengan *id* yang sama. Apabila kategori berubah dari VP_A menjadi VP_B maka nilai variable countAB ditambahkan 1. Apabila kategori berubah dari VP_B menjadi VP_A maka nilai variable countBA ditambahkan 1. CountAB dan countBA menandakan jumlah kendaraan yang berahis dihitung oleh program.
5) Untuk menghemat alokasi memori, anggota dari *map* yang tidak diperlukan dihapus dari memori. Diagram alir penghitungan kendaraan ditunjukkan dalam Gambar 4.7.


3
for(std::map<cvb::CvID,cvb::CvTrack*
>::iterator it = tracks.begin() ; it !=
tracks.end(); it++)
4
track->inactive == 0
T
Y
cvb::CvID id = (*it).first;
cvb::CvTrack* track = (*it).second;
CvPoint2D64f centroid = track->centroid;
positions.count(id) > 0
current_position
!= old_position
T
Y
Y
positions.erase(positions.find(id));
T
old_position == VP_A &&
current_position == VP_B
Y
countAB++;
T
vehiclePosition =
getVehiclePosition(centroid);
old_position == VP_B &&
current_position == VP_A
vehiclePosition
!= VP_NONE
Y
Y
countBA++;
positions.insert(std::pair<cvb::CvID,
VehiclePosition>(id,vehiclePosition));
T
T
positions.erase(positions.find(id));

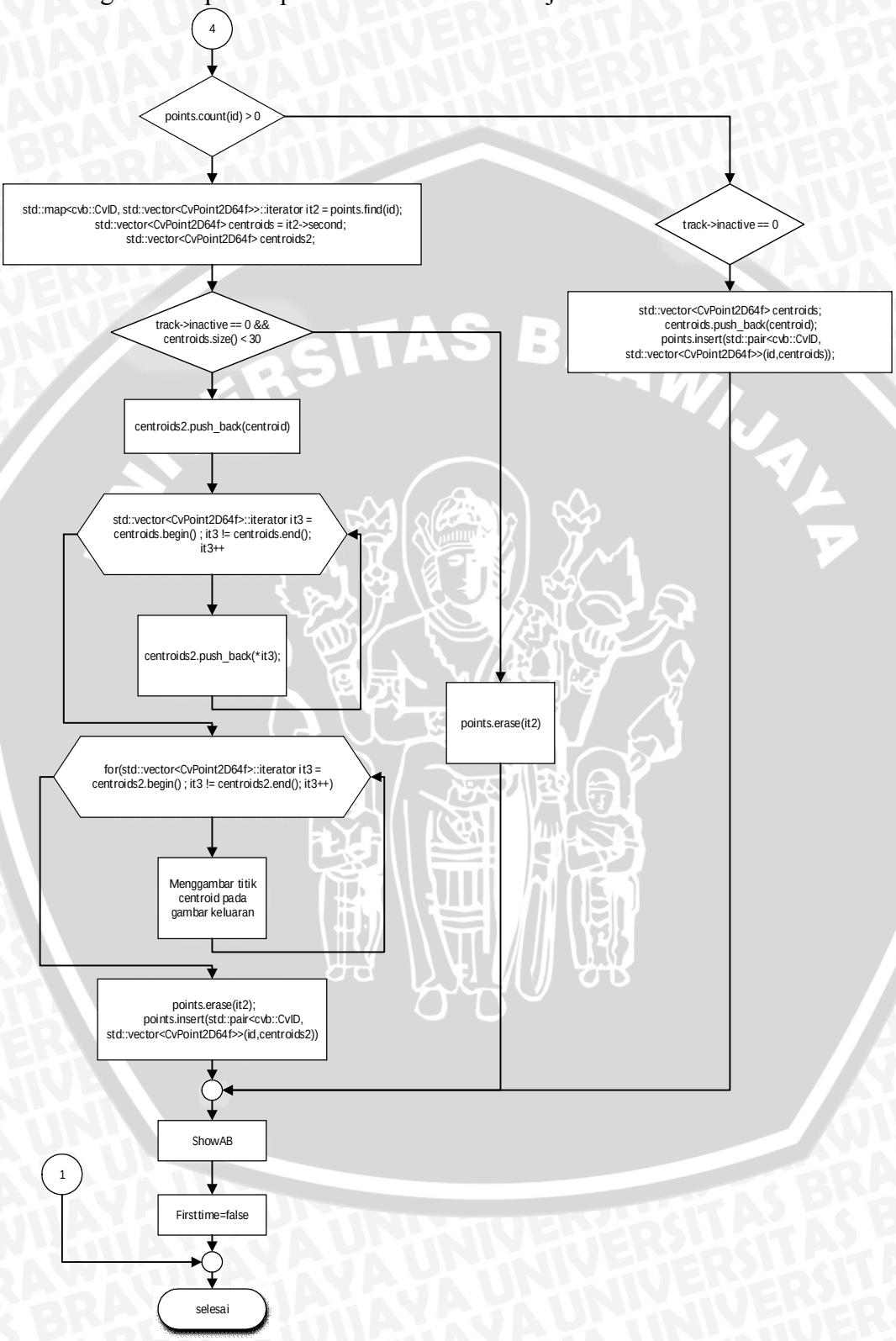
Gambar 4.8 Diagram Alir Penghitungan Kendaraan

4.4.4 Penandaan Lintasan

Untuk mengetahui proses pelacakan *blob* berjalan dengan baik, dilakukan proses penandaan jalur lintsasan yang dilewati oleh kendaraan yang diamati. Diagram alir proses penandaan lintasan dijelaskan sebagai berikut:

1) Melakukan proses iterasi untuk mengakses *id* dan *track* dari proses *blontracking*
2) Selama status *track* masih belum *inactive* dilakukan pengaksesan *id* dan *centroid* tiap-tiap objek yang berhasil dideteksi melalui proses *blobtracking*.
3) Selama objek dalam kondisi aktif dan jumlah *frame* yang terdapat objek tersebut di dalamnya kurang dari tiga puluh, letak *centroid* objek tersebut pada frame sebelumnya disalin pada *frame* berikutnya sehingga jumlah *centroid* pada objek dengan *id* tertentu sama dengan jumlah *frame* objek tersebut aktif.
4) Titik-titik *centroid* yang terdapat dalam proses sebelumnya digambarkan dalam *frame* berupa bulatan kecil menandai jalur yang dilewati objek yang berhasil dideteksi melalui proses *blobtracking.*
5) Saat objek dalam kondisi *inactive, id* dan *centroid* yang terdapat dalam *memory* dihapus untuk mengefektifkan ruang *memory* yang terpakai selama proses penandaan lintasan.
6) Proses berikutnya menampilkan letak objek yang melintas terdapat dalam area A atau area B. Area A menunjukkan objek belum melewati garis hitung dan area B menandakan objek telah melewati garis hitung.

Diagram alir proses penandaan lintasan ditunjukkan dalam Gambar 4.8.



Gambar 4.8 Diagram Alir Penandaan Lintasan

BAB V

PENGUJIAN DAN ANALISIS

5.1 Prosedur Operasional Program

Pada awal program dijalankan dilakukan pengaturan garis yang dilintasi objek yang akan dihitung jumlahnya. Jika garis yang ditetapkan berfungsi, maka garis hitung akan ditampilkan dalam jendela program. Pada awal proses ditampilkan gambar masukan dari sumber dan pesan "Tentukan Garis Hitung". Kemudian pengguna dapat melakukan pengaturan garis dengan melakukan klik pada mouse pada kedua titik ujung garis hitung. Tampilan awal proses pengaturan garis hitung ditunjukkan dalam Gambar 5.1.

Gambar 5.1 Tampilan Awal Penentuan Garis Hitung

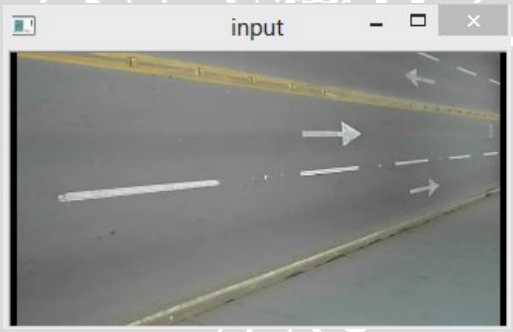
Pengaturan garis hitung ditunjukkan dalam Gambar 5.2.

Gambar 5.2 Pengaturan Garis Hitung

Jika proses penentuan garis hitung berhasil dan garis hitung terdefinisi, akan ditampilkan garis terdefinisi dalam jendela program. Tampilan program jika garis hitung terdefinisi ditunjukkan dalam Gambar 5.3.

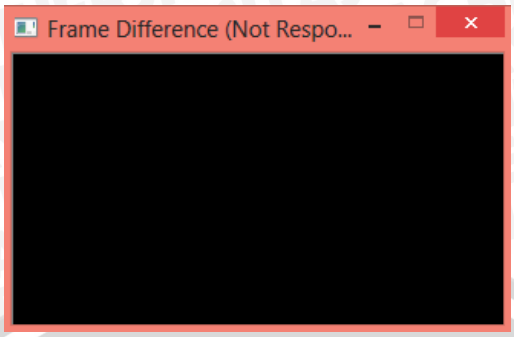
Gambar 5.3 Tampilan Program Jika Garis Hitung Terdeteksi

Berikutnya program akan mendeteksi objek dengan menampilkan beberapa jendela program antara lain Jendela input, *frame difference, Blob Tracking* dan Penghitung Kendaraan. Jendela *input* ditunjukkan dalam Gambar 5.4.



Gambar 5.4 Jendela *Input*

Jendela *Frame Difference* ditunjukkan dalam Gambar 5.5.



Gambar 5.5 Jendela *Frame Difference*

Jendela program *blobtracking* ditunjukkan dalam Gambar 5.6.

Gambar 5.6 *Blob Tracking*

Jendela program penghitung kendaraan ditunjukkan dalam Gambar 5.7.

Gambar 5.7 Penghitung Kendaraan

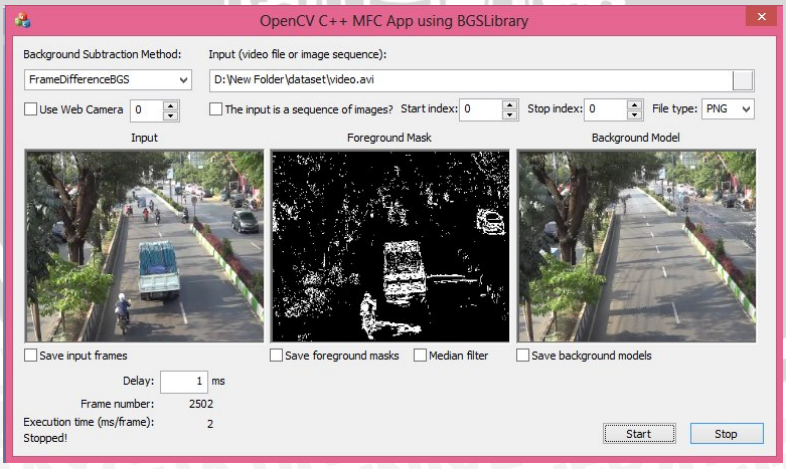
5.2 Pengujian

Pengujian pada skripsi ini terdiri dari tiga bagian meliputi pengujian kecepatan proses *background subtraction* dan pengujian *Blob tracking*. Pengujian

dilakukan di atas jembatan penyebrangan Rumah Sakit Syaiful Anwar Malang pada pukul 15.00 WIB. Pengujian *blobtracking* dan penghitung kendaraan dilakukan dengan menggunakan tiga sudut pengambilan kamera yang diletakkan di atas jembatan penyebrangan menghadap searah dengan arus jalan raya dengan sudut 45^0, 60^0, dan 90^0 dari garis yang tegak lurus menuju permukaan tanah.

5.2.1 Pengujian Kecepatan Proses *Background Subtraction*

Pengujian kecepatan proses *background Subtraction* dilakukan dengan membandingkan kecepatan dari sepuluh metode *Background Subtraction* untuk menentukan metode yang paling tepat digunakan. Metode yang bandingkan antara lain *Adaptive BG Learning, Eigen BG, Mean BGS, Gaussian average of wren, Fuzzi adaptive, Fuzzi Gaussian, Mixture off Gaussian, Simple Gaussian, Multi layer BGS, Frame difference.* Tampilan program pengujian kecepatan proses *background subtraction* ditunjukkan dalam Gambar 5.8.



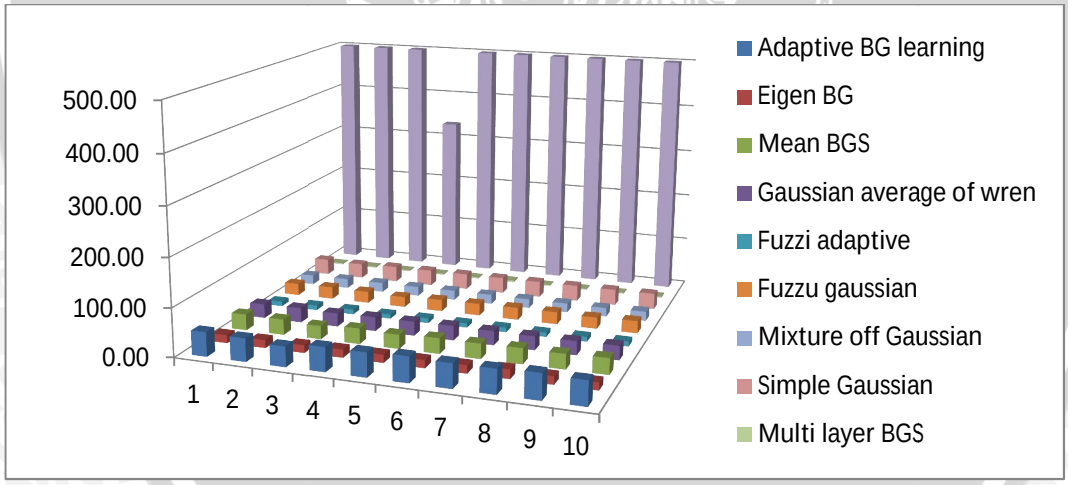
Gambar 5.8. Tampilan Program Penghitung Kecepatan Proses *Background Subtraction*

Hasil pengujian kecepatan proses *background subtraction* ditunjukkan dalam Tabel 5.1.

Tabel 5.1. Hasil Pengujian Kecepatan Proses *Background Subtraction*.

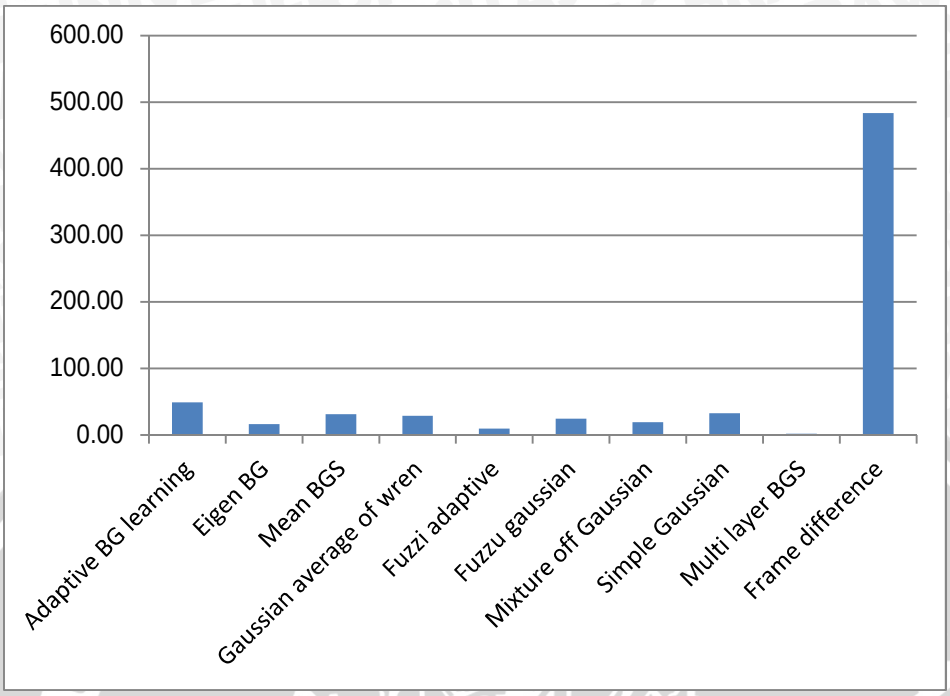
Background Subtraction	Kecepatan proses Frame ke- (Frame per detik)										Rata-rata
	1	2	3	4	5	6	7	8	9	10	
Adaptive BG learning	50.00	47.62	41.67	50.00	50.00	52.63	50.00	50.00	52.63	50.00	49.45
Eigen BG	16.67	16.39	16.13	16.95	16.67	16.39	16.67	19.23	16.67	15.87	16.76
Mean BGS	33.33	32.26	27.78	32.26	29.41	32.26	31.25	32.26	31.25	33.33	31.54
Gaussian average of wren	29.41	30.30	27.78	29.41	29.41	30.30	29.41	28.57	28.57	29.41	29.26
Fuzzi adaptive	9.80	9.71	10.00	9.17	10.00	9.09	9.43	8.85	9.90	10.00	9.60
Fuzzi gaussian	25.00	25.64	24.39	21.74	24.39	25.00	25.64	25.64	23.81	25.64	24.69
Mixture off Gaussian	20.00	20.00	20.00	19.61	20.00	20.00	19.23	18.87	18.18	19.61	19.55
Simple Gaussian	33.33	31.25	33.33	33.33	33.33	33.33	33.33	33.33	33.33	33.33	33.13
Multi layer BGS	2.22	2.33	2.11	2.33	2.34	3.42	2.10	2.30	2.14	2.33	2.36
Frame difference	500.00	500.00	500.00	333.33	500.00	500.00	500.00	500.00	500.00	500.00	483.33

Grafik hasil pengujian kecepatan proses *background subtraction* ditunjukkan dalam Gambar 5.9.



Gambar 5.9. Hasil Pengujian Kecepatan Proses *Background Subtraction* dalam Satuan *Frame* Per Detik

Grafik nilai kecepatan rata-rata proses *background subtraction* ditunjukkan dalam Gambar 5.10.



Gambar 5.10. Grafik Nilai Kecepatan Rata-Rata Proses *Background Subtraction* dalam *Satuan Frame* Per Detik

Bardasarkan hasil pengujian kecepatan proses *background subtraction,* hanya metode *frame difference* yang dapat dijalankan untuk menghitung kendaraan secara *real time*. Metode *background subtraction* lainnya akan berhenti beberapa saat setelah program dijalankan. Metode *background subtraction* lainnya hanya dapat digunakan untuk masukan video yang sudah direkam sebelumnya sehingga untuk pengujian *blobtracking* menggunakan metode *frame difference.*

5.2.2 Pengujian *Blobtracking* dan Penghitung Kendaraan

Dalam pengujian ini dihitung kendaraan yang melintas menggunakan program penghitung kendaraan dan dengan penghitungan manual. Hasil yang didapatkan dibandingkan untuk mendapatkan ketepatan program dalam menghitung jumlah kendaraan. Pengujian dilakukan dengan menggunakan tiga buah video. Video yang pertama adalah hasil pengambilan video dengan sudut 45^{o}, video kedua yang

digunakan adalah hasil pengambilan video dengan sudut 60^o dan video ketiga yang digunakan adalah hasil pengambilan video dengan sudut 90^o. Setiap video dilakukan tiga kali pengujian terhadap 100 Unit kendaraan yang melewati daerah pengujian. Pengujian *blobtracking* dikatakan berhasil apabila setiap kendaraan yang melintas dideteksi oleh program sebagai satu buah objek ketika melewati garis hitung. Kemungkinan lain yang akan terjadi adalah sebuah objek dideteksi menjadi lebih dari satu objek. Hasil pengujian *blobtracking* ditunjukkan dalam Tabel 5.2, Tabel 5.3 dan Tabel 5.4.

Tabel 5.2 Hasil Pengujian *Blobtracking* dengan Objek Terdeteksi Satu Kali

Pengujian	sudut (^{0})		
	45	60	90
1	70	80	74
2	77	76	75
3	77	80	76
Rata-rata (Unit)	75	79	75

Tabel 5.3 Hasil Pengujian *Blobtracking* dengan Objek Terdeteksi Dua Kali

Pengujian	sudut (^{0})		
	45	60	90
1	10	5	1
2	7	7	1
3	6	5	2
Rata-rata (Unit)	8	6	1

Tabel 5.4 Hasil Pengujian *Blobtracking* dengan Objek Tidak Terdeteksi

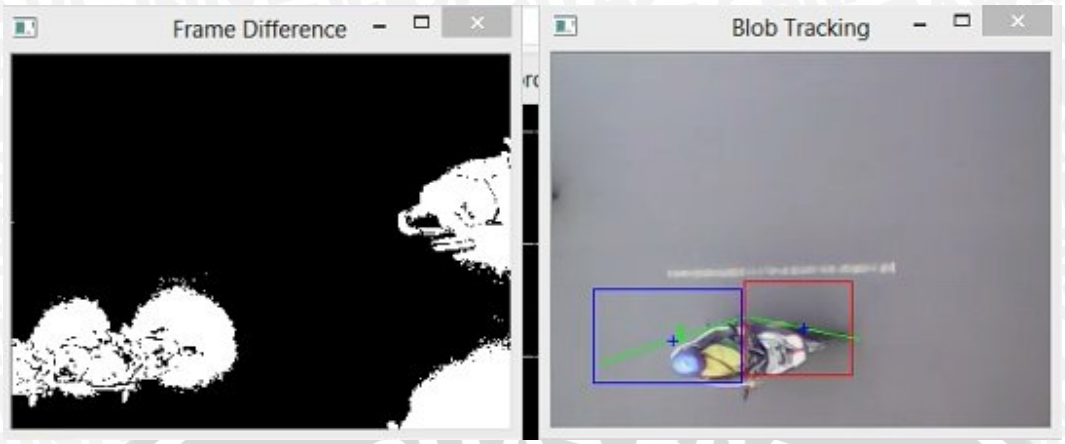
Pengujian	sudut (^{0})		
	45	60	90

1	10	10	24
2	9	10	23
3	11	10	20
Rata-rata (Unit)	10	10	22

Objek yang terdeteksi sebagai satu *blob* ditunjukkan dalam Gambar 5.11.

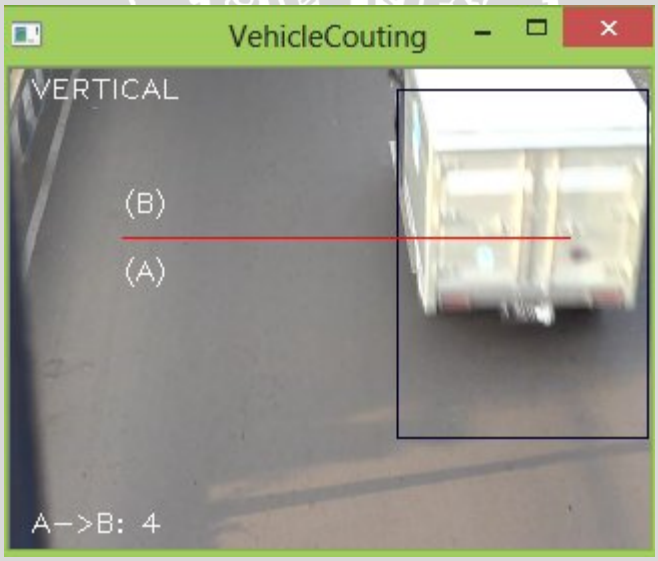
Gambar 5.11 Satu Objek yang Terdeteksi Sebagai Satu *Blob*

Objek yang terdeteksi sebagai lebih dari satu *blob* ditunjukkan dalam Gambar 5.12



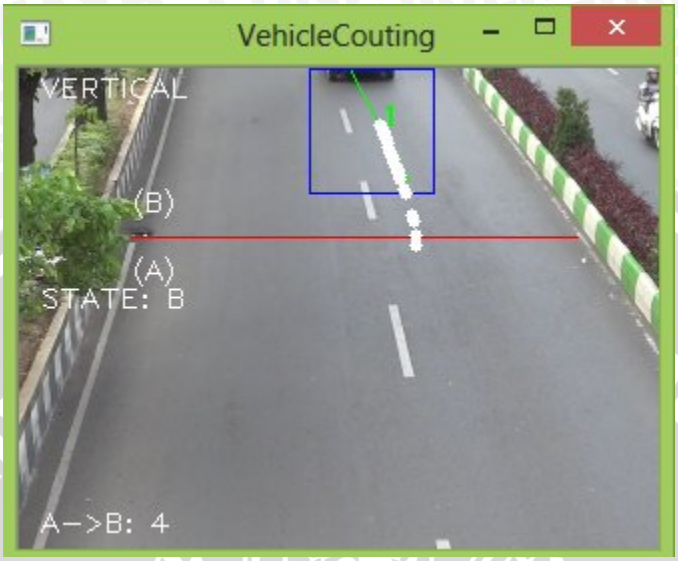
Gambar 5.12. Satu Objek yang Terdeteksi Sebagai Satu *Blob*

Pengujian program penghitung kendaraan dengan Sudut 45^{o} ditunjukkan dalam Gambar 5.13.



Gambar 5.13 Pengujian Program Penghitung Kendaraan Dengan Sudut 45^{o}

Pengujian program penghitung kendaraan dengan Sudut 45^{o} ditunjukkan dalam Gambar 5.14



Gambar 5.14 Pengujian Program Penghitung Kendaraan Dengan Sudut 60^{o}

Pengujian program penghitung kendaraan dengan sudut 90 ^{o} ditunjukkan dalam Gambar 5.15.

Gambar 5.15 Pengujian Program Penghitung Kendaraan Dengan Sudut 90^{o}

5.3 Analisis Sistem

Analisis sistem dilakukan berdasar pada bagian tinjauan pustaka. Analisis dilakukan terhadap parameter-parameter sistem yang digunakan dan hasil yang didapat dari proses pengujian.

5.3.1 Analisis Hasil Pengujian Kecepatan Proses *Background Subtraction*

Pengujian kecepatan pada sepuluh metode *background Subtraction* menunjukkan metode *frame difference* memiliki kecepatan mengolah *frame* paling tinggi. Hal tersebut dikarenakan tidak menggunakan jumlah *frame* yang banyak untuk menentukan objek statis dari suatu video. *Frame difference* menggunakan perhitungan matematis sederhana untuk melakukan proses pemisahan *background*. *Background* yang digunakan bersifat statis sehingga tidak memerlukan adaptasi terhadap perubahan posisi *background*, sedangkan metode lainnya menggunakan adaptasi perubahan *background* agar perubahan posisi kamera tidak berpengaruh signifikan terhadap perubahan bentuk *background*.

Proses adaptasi *background* memerlukan jumlah *frame* yang lebih banyak dan terus menerus sehingga harus menggunakan *memory* lebih besar daripada *background* yang tidak bersifat adaptif. Penggunaan *memory* yang terlalu besar menyebabkan program berhenti ketika melakukan proses *background subtraction* pada video secara *real time* sehingga lebih tepat digunakan untuk proses *background subtraction* pada video yang sudah direkam sebelumnya. Karena pada pengujian ini dilakukan secara *rea ltime,* maka proses pengujian *blobtracking* dilakukan dengan menggunakan metode *frame difference* agar program dapat dijalankan hingga data yang diperlukan tercukupi.

Background yang bersifat statis memiliki kelemahan, apabila terjadi pergeseran posisi kamera maka gambar *foreground* yang dihasilkan jauh berbeda dengan gambar *foreground* aslinya. Dalam pengujian ini kamera diupayakan bersifat statis supaya tidak mempengaruhi hasil dari pengolahan *background subtraction.*

.

5.3.2 Analisis *Blobtracking* dan Penghitungan Kendaraan

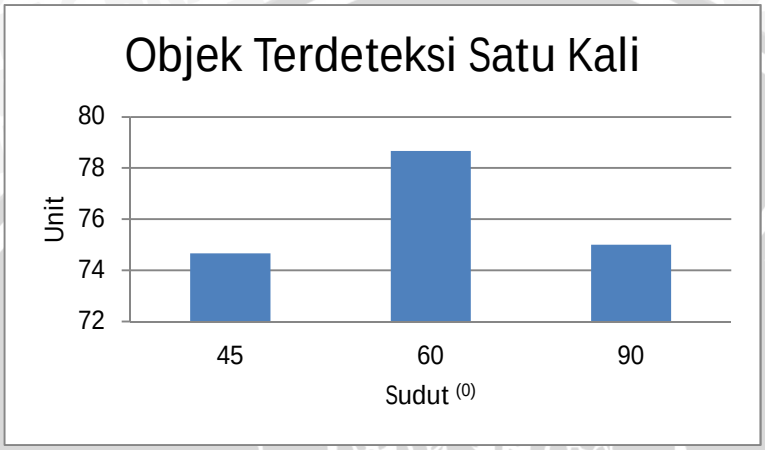
Dalam proses *blobtracking* terdapat satu objek yang dideteksi sebagai dua objek. Hal tersebut dapat terjadi karena hasil dari substraksi *background* yang membentuk dua gumpalan pada *foreground.* Dua gumpalan tersebut terpisah karena adanya proses erosi dan dilasi yang diterapkan pada objek, di mana tujuan awal dari dilakukannya erosi dan dilasi adalah untuk memisahkan objek yang satu dengan objek yang lainnya.

Keberhasilan program dalam menentukan *foreground* dipengaruhi juga dengan kecepatan objek di jalan raya yang diproses. Semakin cepat objek akan menghasilkan *foreground* yang semakin panjang dan akan terpisah menjadi dua objek jika terlalu cepat. Hal tersebut dapat terjadi karena *foreground* yang terbentuk menjadi pecah dan tidak menyatu sehingga dideteksi sebagai *blob* yang berbeda.

Pada kecepatan rendah *blob* dapat dengan mudah dideteksi secara tepat karena *foreground* yang dihasilkan utuh. Bentuk *foreground* yang utuh tidak akan bisa terpisah oleh proses erosi dan dilasi yang diterapkan pada objek dalam video yang diambil dalam sudut 45^0 memiliki nilai prosentase ketepatan hitung yang lebih besar. Hal tersebut dikarenakan luasan daerah yang tertangkap oleh kamera lebih luas, jumlah *frame* yang diproses lebih banyak sehingga lebih mudah dideteksi.

Besar sudut pengambilan gambar juga berpengaruh kepada besarnya *blob* yang didapatkan. Jika pengambilan gambar dilakukan secara tegak lurus maka objek akan terlihat lebih besar. Objek yang terlalu besar pada gambar tidak akan terdeteksi sebagai suatu *blob* karena ketika objek tersebut lebih besar daripada luasan *frame* maka tidak akan membentuk suatu objek.

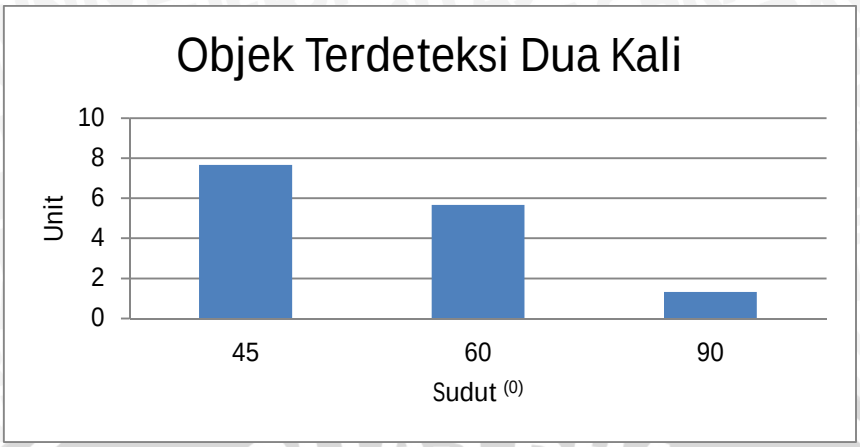
Grafik objek yang terdeteksi satu kali selama proses *blobtracking* ditunjukkan dalam Gambar 5.16.



Gambar 5.16 Grafik Objek yang Terdeteksi Satu Kali

Dalam Gambar 5.16 ditunjukkan proses *blobtracking* pada pengambilan sudut kamera 60^0 memiliki keberhasilan paling tinggi yaitu mampu mendeteksi 79 Unit kendaraan dari 100 Unit yang dideteksi dengan tingkat keberhasilan 79%. Pengambilan video pada sudut tersebut menghasilkan tampilan kendaraan berukuran tidak melebihi atau kurang dari ukuran *filter blob* yang ditetapkan program. Sehingga *blob* yang duhasilkan tidak dihapus oleh *filter blob*.

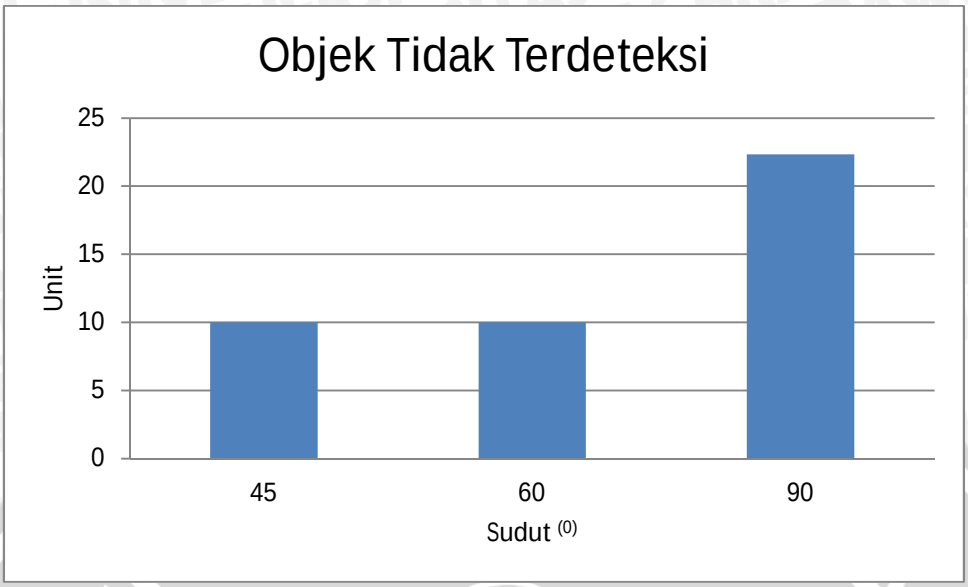
Grafik objek yang terdeteksi satu kali selama proses *blobtracking* ditunjukkan dalam Gambar 5.17.



Gambar 5.17 Grafik Objek yang Terdeteksi Dua Kali

Dalam Gambar 5.17 satu objek dideteksi sebagai dua objek paling banyak terdapat pada pengambilan sudut kamera 45^0. Hal tersebut dikarenakan sudut kamera 45^0 memiliki jarak pandang paling dekat dengan objek yang diproses sehingga kecepatan objek yang tertangkap kamera lebih tinggi daripada pengambilan sudut kamera 60^0 dan 90^0. Kecepatan objek yang tinggi menyebabkan *blob* yang dihasilkan tidak utuh atau terpisah menjadi dua bagian. *Blob* yang tidak utuh tersebut muncul pada objek yang berukuran lebih besar seperti mobil. Pada objek yang kecil seperti sepeda motor *blob* yang dihasilkan dapat dideteksi sebagai satu objek.

Grafik objek yang tidak terdeteksi selama proses *blobtracking* ditunjukkan dalam Gambar 5.18.



Gambar 5.18 Grafik Objek yang Tidak Terdeteksi

Pada pengambilan sudut 90^0 *blob* yang dihasilkan tidak besar karena kecepatan objek yang ditangkap kamera rendah disebabkan jarak kamera dengan permukaan jalan raya jauh sehingga objek yang terlihat berukuran kecil. Kecepatan yang tidak terlalu tinggi menyebabkan program paling berhasil mendeteksi satu kendaraan sebagai satu objek atau tidak terdeteksi sama sekali karena ukuran *blob* yang terlalu kecil.

BAB VI

KESIMPULAN DAN SARAN

6.1 KESIMPULAN

Kesimpulan yang dihasilkan dalam penelitian ini adalah:

1) Untuk melakukan proses *background subtraction* untuk pengujuan secara real time secara dapat menggunakan metode *frame difference* dengan kecepatan 483,33 *frame* per detik.
2) Untuk mendapatkan hasil objek yang dapat dideteksi melalui *blobtracking* perlu ditetapkan *filter blob* yang sesuai dengan kriteria objek yang akan dideteksi.
3) Untuk menggunakan program penghitung kendaraan harus memperhatikan kecepatan objek yang akan dihitung dan pengambilan sudut kamera yang tepat. Sudut optimal pengambilan gambar terletak pada sudut 60^0 dengan tingkat keberhasilan 79%

6.2 SARAN

Saran untuk memperoleh hasil penelitian yang lebih baik adalah:

1) Menggunakan perangkat *hardware* berupa kamera yang memiliki sensitifitas cahaya yang baik sehingga objek dapat terdeteksi secara akurat.
2) Menggunakan metode pemisahan latar belakang dan objek yang menghasilkan latar belakang dengan nilai *pixel* yang lebih stabil sehingga latar belakang yang didapatkan benar-benar bersifat statis.
3) Menggunakan *frame* berukuran sekecil mungkin dengan area yang memadahi untuk digunakan dalam proses penghitungan kendaraan sehingga menghemat ruang *memory* yang degunakan selama proses untuk mendapatkan proses penghitungan yang cepat dan akurat.
4) Menggunakan posisi yang lebih tinggi untuk mendapatkan area pandang kamera yang memuat semua objek yang akan dihitung.

DAFTAR PUSTAKA

Arymurthy, Aniati Murni. 1992. *Pengantar Pengolahan Citra.* Jakarta: PT. Elex Media Komputindo.

Bradsky, Gary. 2008. *Learning OpenCV.* California: O'Reilly Media.

Fairhurst, Michael and Smith, Stephen L. and Mitchell, John . 1995. "Automated Image-Analysis in Visuo-Motor Testing for the Specification of an Integrated Evaluation and Terapy Suport Toll for Rehabilitation". IEEE Transactions on Rehabilitation Engineering.

Fu Chang, Chun Jen Chen, and Chi Jen Lu. 2003. "A Linear Time Component-Labelling Algorithm Using Contour Tracing Technique". Taipei: Institute of Information Science, Academia Sinica.

Hinz, S. 2005. "Fast and Subpixel Precise Blob Detection and Attribution". IEEE International conference of Imagee Processing. Volume 3.

Kumar Jain, Anil. 1989. *Fundamentals of Digital Image Processing*. New Jarsey: Prentice Hall Inc.

Laganiere, Robert. 2009. *OpenCV 2 Computer Vision Application and Programmng Cookbook*. Birmingham: Pact Publishing.

Lindenberg, T. 1993. "Detecting Salient Blob-Like Structures and Their Scales with A Scale-Space Primal Sketch: A Method for Focus of Attention". International Journal of Computer Vision. Volume 11.

Low, Adrian. 1991. *Introductory Computer Vision and Image Processing Paperback.* Mcgraw Hill Book Co Ltd.

Rosenfeld, A and C.Y. Sher. 1998. "Detecting Images Primitives UsingFeature Pyramids" Information Sciences: An International Journal. Volume 107.

Serra, J. 1983. *Image Analysis and Mathematical Morphology*, New York: Academic Press.

Shapiro, Linda. 2001. *Computer Vision.* Washington: The University of Washington.

Yang Q. and B. Parvin. 2002. "Chef: Convex Hull of Elliptic Features for 3D Blob Detection".

Demo.cpp (1)

```
#include <iostream>
#include <cv.h>
#include <highgui.h>

#include "package_bgs/FrameDifferenceBGS.h"

#include "package_tracking/BlobTracking.h"

#include "package_analysis/VehicleCouting.h"

int main(int argc, char **argv)
{
       // input dari kamera
   CvCapture *capture = cvCaptureFromCAM(1);
         cvSetCaptureProperty(capture,CV_CAP_PROP_FRAME_WIDTH,320.0);
         cvSetCaptureProperty(capture,CV_CAP_PROP_FRAME_HEIGHT,178.0);
         cvNamedWindow("Camera Capture", CV_WINDOW_AUTOSIZE);
  if(!capture){
    std::cerr << "Video tidak bisa dibuka!" << std::endl;
    return 1;
  }

   int resize_factor = 100; // 50% dari gambar asal
  IplImage *frame_aux = cvQueryFrame(capture);
  IplImage *frame = cvCreateImage(cvSize((int)((frame_aux->width*resize_factor)/100) ,
(int)((frame_aux->height*resize_factor)/100)), frame_aux->depth, frame_aux-
>nChannels);

  /* alokasi memori untuk Background Subtraction */
  IBGS *bgs;

  bgs = new FrameDifferenceBGS;

  /* alokasi memori untuk Blob Tracking */
  cv::Mat img_blob;
  BlobTracking* blobTracking;
  blobTracking = new BlobTracking;

  /* alokasi memori untuk penghitung kendaraan */
  VehicleCouting* vehicleCouting;
  vehicleCouting = new VehicleCouting;

  int key = 0;
  while(key != 'q')
  {
    frame_aux = cvQueryFrame(capture);
    if(!frame_aux) break;

    cvResize(frame_aux, frame);

    cv::Mat img_input(frame);
    cv::imshow("input", img_input);

    // proses background subtraction
    cv::Mat img_mask;
```

Demo.cpp (2)

```
    bgs->process(img_input, img_mask);

    if(!img_mask.empty())
    {
      // blobtracking
      blobTracking->process(img_input, img_mask, img_blob);

      // penghitungan kendaraan
      vehicleCouting->setInput(img_blob);
      vehicleCouting->setTracks(blobTracking->getTracks());
      vehicleCouting->process();
    }

    key = cvWaitKey(1);
  }

  delete vehicleCouting;
  delete blobTracking;
  delete bgs;

  cvDestroyAllWindows();
  cvReleaseCapture(&capture);

  return 0;
}
```

Blobtracking.cpp (1)

```
#include <iostream>
#include <cv.h>
#include <highgui.h>

#include "package_bgs/FrameDifferenceBGS.h"

#include "package_tracking/BlobTracking.h"

#include "package_analysis/VehicleCouting.h"

int main(int argc, char **argv)
{
       // input dari kamera
   CvCapture *capture = cvCaptureFromCAM(1);
         cvSetCaptureProperty(capture,CV_CAP_PROP_FRAME_WIDTH,320.0);
         cvSetCaptureProperty(capture,CV_CAP_PROP_FRAME_HEIGHT,178.0);
         cvNamedWindow("Camera Capture", CV_WINDOW_AUTOSIZE);
  if(!capture){
    std::cerr << "Video tidak bisa dibuka!" << std::endl;
    return 1;
  }

   int resize_factor = 100; // 50% dari gambar asal
  IplImage *frame_aux = cvQueryFrame(capture);
  IplImage *frame = cvCreateImage(cvSize((int)((frame_aux->width*resize_factor)/100) ,
(int)((frame_aux->height*resize_factor)/100)), frame_aux->depth, frame_aux-
>nChannels);

  /* alokasi memori untuk Background Subtraction */
  IBGS *bgs;

  bgs = new FrameDifferenceBGS;

  /* alokasi memori untuk Blob Tracking */
  cv::Mat img_blob;
  BlobTracking* blobTracking;
  blobTracking = new BlobTracking;

  /* alokasi memori untuk penghitung kendaraan */
  VehicleCouting* vehicleCouting;
  vehicleCouting = new VehicleCouting;

  int key = 0;
  while(key != 'q')
  {
    frame_aux = cvQueryFrame(capture);
    if(!frame_aux) break;

    cvResize(frame_aux, frame);

    cv::Mat img_input(frame);
    cv::imshow("input", img_input);

    // proses background subtraction
    cv::Mat img_mask;
```

Blobtracking.h (2)

```
    bgs->process(img_input, img_mask);

    if(!img_mask.empty())
    {
      // blobtracking
      blobTracking->process(img_input, img_mask, img_blob);

      // penghitungan kendaraan
      vehicleCouting->setInput(img_blob);
      vehicleCouting->setTracks(blobTracking->getTracks());
      vehicleCouting->process();
    }

    key = cvWaitKey(1);
  }

  delete vehicleCouting;
  delete blobTracking;
  delete bgs;

  cvDestroyAllWindows();
  cvReleaseCapture(&capture);

  return 0;
}
```

FrameDifferenceBGS.cpp (1)

```
#include "FrameDifferenceBGS.h"

FrameDifferenceBGS::FrameDifferenceBGS() : firstTime(true), enableThreshold(true),
threshold(15), showOutput(true)
{
  std::cout << "FrameDifferenceBGS()" << std::endl;
}

FrameDifferenceBGS::~FrameDifferenceBGS()
{
  std::cout << "~FrameDifferenceBGS()" << std::endl;
}

void FrameDifferenceBGS::process(const cv::Mat &img_input, cv::Mat &img_output)
{
  if(img_input.empty())
    return;

  loadConfig();

  if(firstTime)
    saveConfig();

  if(img_input_prev.empty())
  {
    img_input.copyTo(img_input_prev);
    return;
  }

  cv::absdiff(img_input_prev, img_input, img_foreground);

  if(img_foreground.channels() == 3)
    cv::cvtColor(img_foreground, img_foreground, CV_BGR2GRAY);

  if(enableThreshold)
    cv::threshold(img_foreground, img_foreground, threshold, 255, cv::THRESH_BINARY);

  if(showOutput)
    cv::imshow("Frame Difference", img_foreground);

  img_foreground.copyTo(img_output);

  img_input.copyTo(img_input_prev);

  firstTime = false;
}

void FrameDifferenceBGS::saveConfig()
{
  CvFileStorage* fs = cvOpenFileStorage("./config/FrameDifferenceBGS.xml", 0,
CV_STORAGE_WRITE);

  cvWriteInt(fs, "enableThreshold", enableThreshold);
  cvWriteInt(fs, "threshold", threshold);
  cvWriteInt(fs, "showOutput", showOutput);
```

FrameDifferenceBGS.cpp (2)

```
  cvReleaseFileStorage(&fs);
}

void FrameDifferenceBGS::loadConfig()
{
  CvFileStorage* fs = cvOpenFileStorage("./config/FrameDifferenceBGS.xml", 0,
CV_STORAGE_READ);

  enableThreshold = cvReadIntByName(fs, 0, "enableThreshold", true);
  threshold = cvReadIntByName(fs, 0, "threshold", 15);
  showOutput = cvReadIntByName(fs, 0, "showOutput", true);

  cvReleaseFileStorage(&fs);
}
```

Vehiclecounting.cpp (1)

```
#include "VehicleCouting.h"

namespace FAV1
{
  IplImage* img_input1 = 0;
  IplImage* img_input2 = 0;
  int roi_x0 = 0;
  int roi_y0 = 0;
  int roi_x1 = 0;
  int roi_y1 = 0;
  int numOfRec = 0;
  int startDraw = 0;
  bool roi_defined = false;
  bool use_roi = true;
  void VehicleCouting_on_mouse(int evt, int x, int y, int flag, void* param)
  {
    if(!use_roi)
      return;

    if(evt == CV_EVENT_LBUTTONDOWN)
    {
      if(!startDraw)
      {
        roi_x0 = x;
        roi_y0 = y;
        startDraw = 1;
      }
      else
      {
        roi_x1 = x;
        roi_y1 = y;
        startDraw = 0;
        roi_defined = true;
      }
    }

    if(evt == CV_EVENT_MOUSEMOVE && startDraw)
    {
      //redraw ROI selection
      img_input2 = cvCloneImage(img_input1);
      cvLine(img_input2, cvPoint(roi_x0,roi_y0), cvPoint(x,y), CV_RGB(255,0,255));
      cvShowImage("VehicleCouting", img_input2);
      cvReleaseImage(&img_input2);
    }
  }
}

VehicleCouting::VehicleCouting(): firstTime(true), showOutput(true), key(0),
countAB(0), countBA(0), showAB(0)
{
  std::cout << "Penghitung Kendaraan()" << std::endl;
}

VehicleCouting::~VehicleCouting()
{
```

Vehichlecounting.cpp (2)

```
  std::cout << "~Penghitung Kendaraan()" << std::endl;
}

void VehicleCouting::setInput(const cv::Mat &i)
{
  //i.copyTo(img_input);
  img_input = i;
}

void VehicleCouting::setTracks(const cvb::CvTracks &t)
{
  tracks = t;
}

VehiclePosition VehicleCouting::getVehiclePosition(const CvPoint2D64f centroid)
{
  VehiclePosition vehiclePosition = VP_NONE;

  if(LaneOrientation == LO_HORIZONTAL)
  {
    if(centroid.x < FAV1::roi_x0)
    {
      cv::putText(img_input, "STATE: A", cv::Point(10,img_h/2),
cv::FONT_HERSHEY_PLAIN, 1, cv::Scalar(255,255,255));
      vehiclePosition = VP_A;
    }

    if(centroid.x > FAV1::roi_x0)
    {
      cv::putText(img_input, "STATE: B", cv::Point(10,img_h/2),
cv::FONT_HERSHEY_PLAIN, 1, cv::Scalar(255,255,255));
      vehiclePosition = VP_B;
    }
  }

  if(LaneOrientation == LO_VERTICAL)
  {
    if(centroid.y > FAV1::roi_y0)
    {
      cv::putText(img_input, "STATE: A", cv::Point(10,img_h/2),
cv::FONT_HERSHEY_PLAIN, 1, cv::Scalar(255,255,255));
      vehiclePosition = VP_A;
    }

    if(centroid.y < FAV1::roi_y0)
    {
      cv::putText(img_input, "STATE: B", cv::Point(10,img_h/2),
cv::FONT_HERSHEY_PLAIN, 1, cv::Scalar(255,255,255));
      vehiclePosition = VP_B;
    }
  }

  return vehiclePosition;
}
```

Vehiclecounting (3)

```
void VehicleCouting::process()
{
  if(img_input.empty())
    return;

  img_w = img_input.size().width;
  img_h = img_input.size().height;

  loadConfig();

  //--------------------------------------------------------------------------

  if(FAV1::use_roi == true && FAV1::roi_defined == false && firstTime == true)
  {
    do
    {
      cv::putText(img_input, "Tentukan Garis Hitung", cv::Point(10,15),
cv::FONT_HERSHEY_PLAIN, 1, cv::Scalar(0,0,255));
      cv::imshow("Penghitung Kendaraan", img_input);
      FAV1::img_input1 = new IplImage(img_input);
      cvSetMouseCallback("Penghitung Kendaraan", FAV1::VehicleCouting_on_mouse, NULL);
      key = cvWaitKey(0);
      delete FAV1::img_input1;

      if(FAV1::roi_defined)
      {
        std::cout << "Garis hitung terdefinisi (" << FAV1::roi_x0 << "," <<
FAV1::roi_y0 << "," << FAV1::roi_x1 << "," << FAV1::roi_y1 << ")" << std::endl;
        break;
      }
      else
        std::cout << "Garis Hitung tak terdefinisi!" << std::endl;
    }while(1);
  }

  if(FAV1::use_roi == true && FAV1::roi_defined == true)
    cv::line(img_input, cv::Point(FAV1::roi_x0,FAV1::roi_y0),
cv::Point(FAV1::roi_x1,FAV1::roi_y1), cv::Scalar(0,0,255));

  bool ROI_OK = false;

  if(FAV1::use_roi == true && FAV1::roi_defined == true)
    ROI_OK = true;

  if(ROI_OK)
  {
    LaneOrientation = LO_NONE;

    if(abs(FAV1::roi_x0 - FAV1::roi_x1) < abs(FAV1::roi_y0 - FAV1::roi_y1))
    {
      if(!firstTime)
        cv::putText(img_input, "HORIZONTAL", cv::Point(10,15), cv::FONT_HERSHEY_PLAIN,
1, cv::Scalar(255,255,255));
      LaneOrientation = LO_HORIZONTAL;
```

VehicleCounting.cpp (4)

```
      cv::putText(img_input, "(A)", cv::Point(FAV1::roi_x0-32,FAV1::roi_y0),
cv::FONT_HERSHEY_PLAIN, 1, cv::Scalar(255,255,255));
      cv::putText(img_input, "(B)", cv::Point(FAV1::roi_x0+12,FAV1::roi_y0),
cv::FONT_HERSHEY_PLAIN, 1, cv::Scalar(255,255,255));
    }

    if(abs(FAV1::roi_x0 - FAV1::roi_x1) > abs(FAV1::roi_y0 - FAV1::roi_y1))
    {
      if(!firstTime)
        cv::putText(img_input, "VERTICAL", cv::Point(10,15), cv::FONT_HERSHEY_PLAIN,
1, cv::Scalar(255,255,255));
      LaneOrientation = LO_VERTICAL;

      cv::putText(img_input, "(A)", cv::Point(FAV1::roi_x0,FAV1::roi_y0+22),
cv::FONT_HERSHEY_PLAIN, 1, cv::Scalar(255,255,255));
      cv::putText(img_input, "(B)", cv::Point(FAV1::roi_x0,FAV1::roi_y0-12),
cv::FONT_HERSHEY_PLAIN, 1, cv::Scalar(255,255,255));
    }
  }

  //--------------------------------------------------------------------------

  for(std::map<cvb::CvID,cvb::CvTrack*>::iterator it = tracks.begin() ; it !=
tracks.end(); it++)
  {
    std::cout << (*it).first << " => " << (*it).second << std::endl;
    cvb::CvID id = (*it).first;
    cvb::CvTrack* track = (*it).second;

    CvPoint2D64f centroid = track->centroid;

    std::cout << "------------------------------------------------------------------" <<
std::endl;
    std::cout << "0)id:" << id << " (" << centroid.x << "," << centroid.y << ")" <<
std::endl;
    std::cout << "track->active:" << track->active << std::endl;
    std::cout << "track->inactive:" << track->inactive << std::endl;
    std::cout << "track->lifetime:" << track->lifetime << std::endl;


    //--------------------------------------------------------------------------

    if(track->inactive == 0)
    {
      if(positions.count(id) > 0)
      {
        std::map<cvb::CvID, VehiclePosition>::iterator it2 = positions.find(id);
        VehiclePosition old_position = it2->second;

        VehiclePosition current_position = getVehiclePosition(centroid);

        if(current_position != old_position)
        {
          if(old_position == VP_A && current_position == VP_B)
            countAB++;
```

VehichleCounting.cpp (5)

```
          if(old_position == VP_B && current_position == VP_A)
            countBA++;

          positions.erase(positions.find(id));
        }
      }
      else
      {
        VehiclePosition vehiclePosition = getVehiclePosition(centroid);

        if(vehiclePosition != VP_NONE)
          positions.insert(std::pair<cvb::CvID, VehiclePosition>(id,vehiclePosition));
      }
    }
    else
    {
      if(positions.count(id) > 0)
        positions.erase(positions.find(id));
    }

    //--------------------------------------------------------------------------

    if(points.count(id) > 0)
    {
      std::map<cvb::CvID, std::vector<CvPoint2D64f>>::iterator it2 = points.find(id);
      std::vector<CvPoint2D64f> centroids = it2->second;

      std::vector<CvPoint2D64f> centroids2;
      if(track->inactive == 0 && centroids.size() < 30)
      {
        centroids2.push_back(centroid);

        for(std::vector<CvPoint2D64f>::iterator it3 = centroids.begin() ; it3 !=
centroids.end(); it3++)
        {
          centroids2.push_back(*it3);
        }

        for(std::vector<CvPoint2D64f>::iterator it3 = centroids2.begin() ; it3 !=
centroids2.end(); it3++)
        {
          cv::circle(img_input, cv::Point((*it3).x,(*it3).y), 3,
cv::Scalar(255,255,255), -1);
        }

        points.erase(it2);
        points.insert(std::pair<cvb::CvID, std::vector<CvPoint2D64f>>(id,centroids2));
      }
      else
      {
        points.erase(it2);
      }
    }
    else
    {
```

VehichleCounting.cpp (6)

```
      if(track->inactive == 0)
      {
        std::vector<CvPoint2D64f> centroids;
        centroids.push_back(centroid);
        points.insert(std::pair<cvb::CvID, std::vector<CvPoint2D64f>>(id,centroids));
      }
    }

    //cv::waitKey(0);
  }

  //--------------------------------------------------------------------------

  if(showAB == 0)
  {
    cv::putText(img_input, "A->B: " + boost::lexical_cast<std::string>(countAB),
cv::Point(10,img_h-20), cv::FONT_HERSHEY_PLAIN, 1, cv::Scalar(255,255,255));
    cv::putText(img_input, "B->A: " + boost::lexical_cast<std::string>(countBA),
cv::Point(10,img_h-8), cv::FONT_HERSHEY_PLAIN, 1, cv::Scalar(255,255,255));
  }

  if(showAB == 1)
    cv::putText(img_input, "A->B: " + boost::lexical_cast<std::string>(countAB),
cv::Point(10,img_h-8), cv::FONT_HERSHEY_PLAIN, 1, cv::Scalar(255,255,255));

  if(showAB == 2)
    cv::putText(img_input, "B->A: " + boost::lexical_cast<std::string>(countBA),
cv::Point(10,img_h-8), cv::FONT_HERSHEY_PLAIN, 1, cv::Scalar(255,255,255));

  if(showOutput)
    cv::imshow("Penghitung Kendaraan", img_input);

  if(firstTime)
    saveConfig();

  firstTime = false;
}

void VehicleCouting::saveConfig()
{
  CvFileStorage* fs = cvOpenFileStorage("./config/VehicleCouting.xml", 0,
CV_STORAGE_WRITE);

  cvWriteInt(fs, "showOutput", showOutput);
  cvWriteInt(fs, "showAB", showAB);

  cvWriteInt(fs, "fav1_use_roi", FAV1::use_roi);
  cvWriteInt(fs, "fav1_roi_defined", FAV1::roi_defined);
  cvWriteInt(fs, "fav1_roi_x0", FAV1::roi_x0);
  cvWriteInt(fs, "fav1_roi_y0", FAV1::roi_y0);
  cvWriteInt(fs, "fav1_roi_x1", FAV1::roi_x1);
  cvWriteInt(fs, "fav1_roi_y1", FAV1::roi_y1);

  cvReleaseFileStorage(&fs);
}
```

VehichleCounting.cpp (7)

```
void VehicleCouting::loadConfig()
{
  CvFileStorage* fs = cvOpenFileStorage("./config/VehicleCouting.xml", 0,
CV_STORAGE_READ);

  showOutput = cvReadIntByName(fs, 0, "showOutput", true);
  showAB = cvReadIntByName(fs, 0, "showAB", 1);

  FAV1::use_roi = cvReadIntByName(fs, 0, "fav1_use_roi", true);
  FAV1::roi_defined = cvReadIntByName(fs, 0, "fav1_roi_defined", false);
  FAV1::roi_x0 = cvReadIntByName(fs, 0, "fav1_roi_x0", 0);
  FAV1::roi_y0 = cvReadIntByName(fs, 0, "fav1_roi_y0", 0);
  FAV1::roi_x1 = cvReadIntByName(fs, 0, "fav1_roi_x1", 0);
  FAV1::roi_y1 = cvReadIntByName(fs, 0, "fav1_roi_y1", 0);

  cvReleaseFileStorage(&fs);
}
```

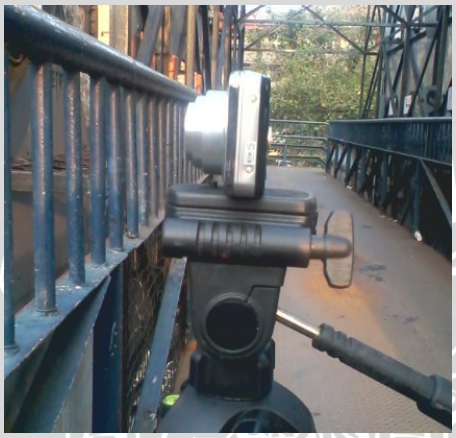
Lampiran Foto Pengujian

Gambar 1. Pengambilan Gambar Dengan Sudut 45^0 Tampak Dari Samping

Gambar 2. Pengambilan Gambar Dengan Sudut 45^0 Tampak Dari Belakang

Gambar 3. Pengambilan Gambar Dengan Sudut 60^0 Tampak Dari Samping

Gambar 4. Pengambilan Gambar Dengan Sudut 60^0 Tampak Dari Belakang



Gambar 5. Pengambilan Gambar Dengan Sudut 90^0 Tampak Dari Samping

Gambar 6. Pengambilan Gambar Dengan Sudut 90^0 Tampak Dari Samping

Gambar 7. Tempat Pengambilan Gambar di Jembatan Penyebrangan RS Syaiful Anwar Malang