**BAB III**

**PERANCANGAN SISTEM**

* 1. **DESKRIPSI UMUM**

Proyek akhir ini bertujuan untuk membuat sebuah sistem yang bisa membantu memantau kecepatan kendaraan menggunakan sensor kamera. Dalam proyek akhir ini, lokasi pemantauannya adalah di Tol Surabaya-Gresik. Tidak menutup kemungkinan sistem ini juga bisa digunakan di lokasi lain selain jalan tol.

Sistem ini terdiri dari dua bagian, pertama adalah sistem yang digunakan untuk mendeteksi kecepatan kendaraan. Hasil dari bagian ini berupa hasil kecepatan kendaraan dan gambar dari kendaraan apabila melebihi batas kecepatan yang ditentukan. Yang kedua, sistem laporan hasil pemantauan kecepatan kendaraan berupa data statistik pelanggar batas kecepatan seperti jumlah pelanggar dan kecepatan rata-rata pelanggar.

Selanjutnya dalam bab ini akan dijelaskan mengenai perancangan sistem beserta proses yang terjadi dalam pembuatan proyek akhir ini seperti pada gambar 3.1 dibawah ini.



**Gambar 3.1.** Perancangan Sistem

Penjelasannya adalah :

1. Pengambilan data video

Merupakan proses pengambilan video lalu-lintas kendaraan pada jalan tol Surabaya-Gresik dan sebaliknya.

1. Pemrograman dan *build* sistem menggunakan Delphi 7

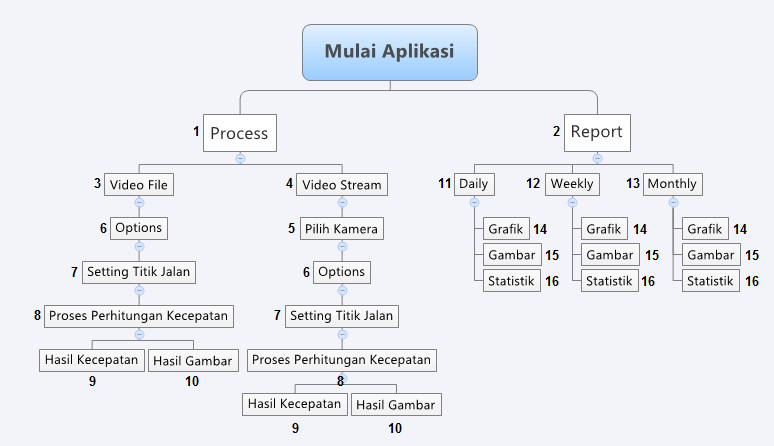
Sistem ini dibuat dengan menggunakan program Delphi 7

1. Penggunaan sistem oleh user

User bisa menggunakan sistem yang sudah dibuat

* 1. **RANCANGAN APLIKASI**

Aplikasi ini terdiri dari beberapa bagian yang saling berhubungan satu sama lain sesuai alur atau rancangan yang telah ditentukan. Rancangan tersebut dapat dijelaskan melalui diagram blok seperti gambar 3.2 di bawah ini.



**Gambar 3.2.** Rancangan Aplikasi

Aplikasi ini terdiri dari dua bagian, yaitu proses penghitungan kecepatan dan report. Berikut penjelasannya :

1. *Process*

Ini adalah bagian untuk melakukan pemantauan kecepatan kendaraan.

1. *Report*

Merupakan bagian untuk melihat laporan mengenai pelanggar batas kecepatan lalu-lintas.

1. *Video File*

Merupakan pemilihan video yang sudah direkam sebelumnya untuk digunakan sebagai sumber data kecepatan kendaraan yang akan dihitung kecepatannya.

1. *Video Stream*

Ini adalah penggunaan video secara *real time* sebagai sumber gambar kecepatan kendaraan menggunakan kamera yang sudah terintergasi dengan sistem.

1. Pilih Kamera

Opsi ini hanya ada jika menggunakan *video stream* sebagai sumber data. Berguna untuk memilih kamera yang akan digunakan.

1. *Options*

Pada bagian ini berisi *setting* untuk jenis *filtering* yang akan digunakan, batas kecepatan yang diijinkan, panjang jalan sebenarnya yang ada di video, tampilan *track* yang dideteksi dalam video dan lokasi penyimpanan hasil capture.

1. *Setting* Titik Jalan

Ini merupakan bagian dimana kita melakukan *setting* titik-titik mana yang akan dilakukan deteksi kecepatan kendaraan.

1. Proses Perhitungan Kecepatan

Disini dilakukan proses perhitungan kecepatan dari kendaraan yang ada pada sumber data.

1. Hasil Kecepatan

Merupakan hasil kecepatan kendaraan dari proses perhitungan kecepatan.

1. Hasil Gambar

Apabila hasil kecepatan melebihi batas kecepatan yang diijinkan, maka hasilnya juga berupa gambar kendaraan yang melebihi batas kecepatan.

1. *Daily*

Merupakan fitur untuk melihat laporan pelanggar lalu lintas dalam satu hari.

1. *Weekly*

Merupakan fitur untuk melihat laporan pelanggar lalu lintas dalam satu minggu.

1. *Monthly*

Merupakan fitur untuk melihat laporan pelanggar lalu lintas dalam satu bulan.

1. Grafik

Laporan dalam bentuk grafik jumlah pelanggar dan rata-rata kecepatannya.

1. Gambar

Laporan dalam bentuk gambar kendaraan pelanggar.

1. Statistik

Laporan berbentuk tabel yang berisi tanggal, lokasi jalan, jumlah pelanggar dan rata-rata kecepatan.

* 1. **IMPLEMENTASI SISTEM**
     1. **Tools Yang Digunakan**

Aplikasi ini dibuat menggunakan program Delphi 7, tanpa menggunakan library pengolahan citra seperti OpenCV, Matlab ataupun library lainnya.

Delphi adalah suatu bahasa pemograman (development language) yang digunakan untk merancang suatu aplikasi program. Delphi termasuk dalam pemrograman bahasa tingkat tinggi (high level lenguage). Maksud dari bahasa tingkat tinggi yaitu perintah-perintah programnya menggunakan bahasa yang mudah dipahami oleh manusia. Bahasa pemrograman Delphi disebut bahasa prosedural artinya mengikuti urutan tertentu. Delphi menggunakan lingkungan pemrograman visual.



**Gambar 3.3.** Program Delphi 7

* + 1. **Pengambilan Video**

Dalam pembuatan proyek akhir ini, video merupakan salah satu bagian penting karena berfungsi sebagai sumber data yang akan diolah. Gambar kendaraan yang akan dihitung kecepatannya berasal dari video ini. Dibawah ini adalah detail hasil video dan kamera yang digunakan untuk merekam lalu-lintas pada jalan tol Surabaya-Gresik dan sebaliknya.

**Tabel 3.1.** Detail Kamera dan Hasil Video

|  |  |
| --- | --- |
| **Nama** | Lenovo S820 |
| **Pixel** | 12 Megapixel |
| ***Frame* *Width*** | 1280 |
| ***Frame* *Height*** | 720 |
| ***Frame Rate*** | 30 *Frame/Second* |
| **Tanggal Pengambilan** | 6 & 8 Desember 2014 |
| **Lokasi Pengambilan** | Tol Surabaya - Gresik  Tol Gresik - Surabaya |

Proses pengambilan video dilakukan pada sebuah jembatan penyeberangan yang berada diatas jalan tol. Karena proyek akhir ini bertujuan untuk mengetahui para pelanggar batas kecepatan kendaraan, maka dibutuhkan video yang merekam kendaraan yang melaju dengan kecepatan tinggi. Video rekaman yang dibutuhkan merupakan rekaman dari atas jalan, sehingga perekaman bisa dilakukan dari jembatan penyebrangan. Hasil rekaman tidak akan sesuai kebutuhan jika di dapat dari jalan yang berada di dalam kota, karena lokasi jembatan penyebrangan biasanya berada diatas jalan yang ramai kendaraan. Jadi kendaraan yang melalui jalanan tersebut kecepatannya kurang maksimal. Sehingga muncul ide untuk melakukan pengambilan video kondisi pada jalan tol. Dimana kecepatan kendaraan di jalan tol bisa lebih maksimal daripada kecepatan kendaraan saat di jalan dalam kota.



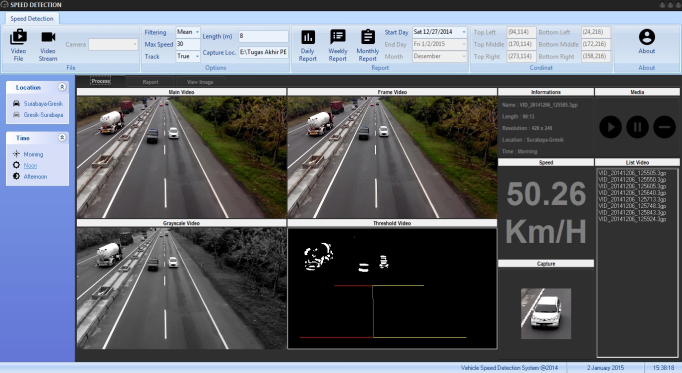
**Gambar 3.4.** Cara Pengambilan Video

* + 1. **Pembuatan Aplikasi**

Dalam pembuatan aplikasi ini ada beberapa tahapan yang harus dikerjakan, yaitu :

1. **Desain *Interface***

Sesuai dengan rancangan aplikasi pada gambar 3.2, maka harus kita buat tampilan yang sesuai dengan kebutuhan yang ada pada rancangan aplikasi. Bentuk tampilan dibuat agar mudah digunakan nantinya oleh calon pengguna.



**Gambar 3.5.** Desain *Interface* Aplikasi

1. **Implementasi *Grayscale Image***

Sebelum mengimplementasikan metode *optical flow* terhadap video rekaman, terlebih dahulu dilakukan perbuhan gambar asli menjadi abu-abu (*Grayscale*). Hal ini bertujuan untuk mengurangi nois yang mungkin akan terjadi. Caranya adalah dengan menggunakan rata-rata dari warna dalam sebuah pixel. Implementasi *grayscale* bisa dilihat dari gambar *pseudocode* dibawah ini.

procedure Grayscale(x,y);

begin

R = Nilai Red Pixel (x,y)

G = Nilai Green Pixel (x,y)

B = Nilai Blue Pixel (x,y)

Pixel(x,y) = (R + G + B) div 3

end;

**Gambar 3.6.** Prosedur Implementasi *Grayscale*

Dari prosedur diatas, hasil yang diperoleh adalah seperti pada gambar dibawah ini.



**Gambar 3.7.** Gambar Asli Sebelum Grayscale



**Gambar 3.7.** Gambar Hasil Grayscale

1. **Implementasi Program *Optical Flow***

Setelah melakukan *grayscale* dari gambar asli, tahap selanjutnya adalah mengimplementasikan metode *optical flow* untuk mengetahui obyek yang bergerak atau kendaraan. Prosesnya adalah dengan membandingkan setiap piksel pada sebuah sekuen gambar. Adanya perbedaan nilai pada sebuah piksel antara *frame* sekarang dengan *frame* sebelumnya menandakan adanya obyek bergerak pada gambar tersebut. *Pseudocode* bisa dilihat dari gambar dibawah ini.

procedure OpticalFlow;

begin

frameSekarang = Nilai Pixel (frame)

frameSebelum = Nilai Pixel (frame-1)

hasilOpticalFlow = frameSekarang - frameSebelum

end;

**Gambar 3.8.** Prosedur Implementasi *Optical Flow*

Pada tahap ini, hasilnya belum berupa gambar. Hanya berupa nilai yang menunukkan apakah ada atau tidaknya perbedaan antara *frame* sekarang dengan *frame* sebelumnya.

1. **Implementasi Program *Threshold***

Agar bentuk obyek yang bergerak bisa diketahui dengan baik, hasil dari *optical flow* akan ditampilkan dalam bentuk gambar hitam putih. Yang berwarna hitam adalah *background* sedangkan putih adalah obyek yang bergerak atau kendaraan. *Pseudocode* bisa dilihat dari gambar dibawah ini.

procedure Threshold;

begin

if hasilOpticalFlow < 60 then pixel(x,y) = 0 else

pixel(x,y) = 255;

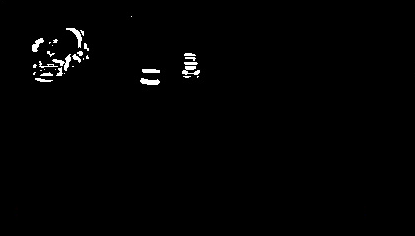
end;

**Gambar 3.9.** Prosedur Implementasi *Threshold*

Pada *pseudocode* tersebut, hasil merupakan nilai hasil perhitungan *optical flow*. Sedangkan angka 60 adalah batas *threshold* yang sudah ditentukan. Pada tahap ini hasilnya berupa gambar yang menunjukkan ada atau tidaknya obyek bergerak dalam sekuen gambar.



**Gambar 3.10.** Gambar Asli Sebelum *Threshold*



**Gambar 3.11.** Gambar Hasil *Threshold*

1. **Implementasi *Mean Filtering***

Hasil dari *thresholding* masih menghasilkan banyak noise. Dimana yang dianggap obyek bergerak bukan hanya kendaraan, melainkan juga ranting-ranting pohon yang bergoyang karena terkena angin. Diperlukan metode tambahan untuk bisa membedakan antara ranting pohon yang bergerak dengan mobil. Untuk itu digunakan *mean filtering*. *Pseudocode* implementasi dari metode ini bisa dilihat pada gambar dibawah ini.

procedure MeanFiltering(x,y);

begin

nilai\_1 = hasilOpticalFlow(x,y)

nilai\_2 = hasilOpticalFlow(x+1,y)

nilai\_3 = hasilOpticalFlow(x+1,y+1)

nilai\_4 = hasilOpticalFlow(x,y+1)

nilai\_5 = hasilOpticalFlow(x-1,+1y)

nilai\_6 = hasilOpticalFlow(x-1,y)

nilai\_7 = hasilOpticalFlow(x-1,y-1)

nilai\_8 = hasilOpticalFlow(x,y-1)

nilai\_9 = hasilOpticalFlow(x+1,y-1)

hasilMeanFiltering(x,y) = (nilai\_1+ nilai\_2+ nilai\_3+ nilai\_4+ nilai\_5+ nilai\_6+ nilai\_7+ nilai\_8+ nilai\_9) div 3;

end;

**Gambar 3.12.** Prosedur Implementasi *Mean Filtering*

Hasil dari *mean filtering* bisa dilihat pada bab selanjutnya, yaitu bab uji coba dan analisa untuk mengetahui perbedaan antara mean filtering dengan thresholding sebelumnya.

1. **Implementasi *Adaptive Thresholding***

Sama seperti dengan *mean filtering*, penggunaan *adaptive thresholding* juga bertujuan untuk meghilangkan noise dari hasil dari *thresholding*. Metode ini digunakan untuk membandingkan apakah hasil dari *mean filtering* sudah cukup untuk menghilangkan noise tersebut atau hasil *adaptive thresholding* yang lebih baik. *Pseudocode* implementasi dari metode ini bisa dilihat pada gambar dibawah ini.

procedure AdaptiveThresholding(x,y);

begin

hasilAdaptive(x,y) = hasilMeanFiltering(x,y) \*

( 1 + (0.34\*((deviasi(x,y)/128)-1)) )

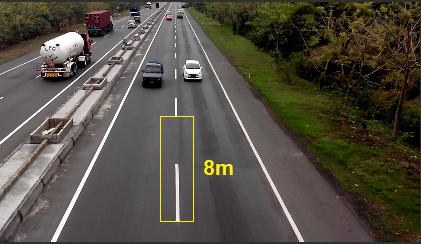
end;

**Gambar 3.13.** Prosedur Implementasi *Adaptive Thresholding*

Untuk hasil dari *apative thresholding* akan dibahas pada bab selanjutnya yaitu uji coba dan analisa. Untuk membandingkan metode mana yang lebih baik digunakan untuk menghilangkan noise dari hasil *thresholding*

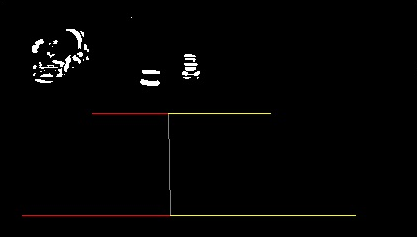
1. **Penghitungan Jarak Sebenarnya**

Penentuan jarak sebenarnya dalam video ditentukan dengan menggunakan panjang marka jalan sebagai acuan. Sebelum aplikasi bisa melakukan proses perhitungan kecepatan kendaraan dalam video, *user* harus menentukan dimana titik yang menjadi acuan panjang jarak sebenarnya. Untuk lebih jelasnya bisa melihat pada penjelasan dibawah ini.



**Gambar 3.14.** Penentuan Jarak Sebenarnya

Sebelumnya sudah dijelaskan bahwa jarak antara marka adalah 5m dan panjang marka tersebut adalah 3m. Jarak yang digunakan dalam proses ini adalah gabungan panjang keduanya, sehingga jarak yang bisa diketahui dalam video adalah sepanjang 8m. Setelah itu kita menentukan dimana titik kordinat 0m dan 8m. Hal ini berguna untuk mendapatkan panjang sebenarnya dalam satuan pixel.



**Gambar 3.15.** Penentuan Titik Kordinat Jarak

Setelah mendapatkan titik kordinat 0m dan 8m kita akan mengetahui berapa panjang pixel untuk memperoleh jarak 8m dalam video dengan menggunakan rumus jarak antara dua titik.

(8)

*Pseudocode* penghitungan jarak sebenarnya dalam video bisa dilihat pada gambar dibawah ini

procedure JarakSebenarnya(x1,y1,x2,y2);

begin

jarak := Sqrt(Sqr(Abs(x1-x2))+Sqr(Abs(y1-y2)));

end;

**Gambar 3.13.** Prosedur Implementasi *Adaptive Thresholding*

Contoh perhitungannya :

Kordinat (x1,y1) = 170,114

Kordinat (x2,y2) = 172,216

Jarak =

Jarak = pixel per 8 meter

1. **Penghitungan Kecepatan**
2. **Report**
3. **Programming**

Pada tahap ini adalah tahap dimana desain *interface* yang sudah disusun tadi di berikan program agar berjalan sesuai dengan yang diharapkan. Sesuai penjelasan sebelumnya, program dibuat dengan menggunakan Delphi 7.

Beberapa hal penting dalam pembuatan proyek akhir ini adalah bagaimana mengimplementasikan metode *optical flow*, *threshold*, *mean filtering*, *adaptive thresholding* dan *BLOB analysis* untuk mendeteksi obyek bergerak dalam video dimana dalam proyek akhir ini obyeknya adalah kendaraan roda 4 atau lebih. Selain itu, yang juga penting adalah bagaimana bisa menghitung kecepatan kendaraan dari hasil deteksi obyek yang dilakukan.

Fitur report bukan merupakan fitur utama dari proyek akhir ini, fitur ini hanya untuk memberikan kemudahan bagi user untuk melihat hasil pemantauan kecepatan kendaraan oleh program.

1. **Pengujian**

Pengujian dilakukan untuk mengetahui sejauh mana ketelitian program yang sudah dibuat apakah sudah sesuai dengan harapan atau tidak. Cara pengujian dan hasil pengujiaannya akan dijelaskan lebih lanjut pada bab selanjutnya.

**\*\*\* Halaman ini sengaja dikosongkan \*\*\***