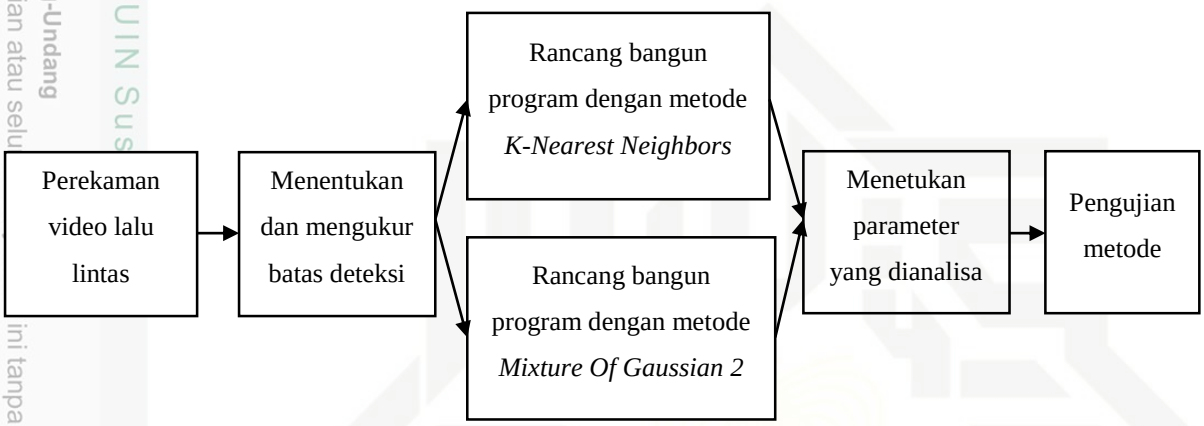
BAB III
METODOLOGI PENELITIAN

Pada penelitian ini metode yang digunakan adalah metode eksperimen. Penelitian ini dilakukan dalam beberapa tahap yang digambarkan dalam bentuk *flowchart*. Tahapan-tahapan penelitian ini dapat dilihat pada Gambar 3.1.



Gambar 3.1 Blok diagram penelitian

Langkah pertama dari penelitian perbandingan akurasi metode *mixture of gaussian 2* dengan *k-nearest neighbor* dalam mendeteksi kecepatan kendaraan berbasis video adalah perekaman video lalu lintas. Video yang direkam pada penelitian ini berlokasi di jalan Soedirman Pekanbaru, Riau. Video direkam dari jembatan penyebrangan dengan posisi kamera menghadap kebawah. Perekaman video dilakukan pada pukul 10:54 WIB dengan posisi bayangan hampir tegak lurus terhadap kendaraan. Video yang direkam memiliki empat skenario kecepatan dengan menggunakan sepeda motor. Skenario kecepatan yang direkam adalah kecepatan 20 km/jam, 40 km/jam, 50 km/jam dan 60 km/jam. Satu video yang direkam terdapat satu skenario kecepatan kendaraan.

Langkah kedua yaitu menentukan batas deteksi dan mengukur batas deteksi. Batas deteksi kendaraan dalam penelitian ini adalah sepanjang cat pembatas jalur yang terdapat dijalan Soedirman Pekabaru, Riau. Batas deteksi hanya mendeteksi satu dari tiga jalur yang ada di jalan Soedirman Pekanbaru, Riau. Panjang cat pembatas jalur tesebut adalah 2,98 meter, sedangkan panjang cat pembatas jalur yang terekam dalam video adalah 75 piksel.

Langkah ketiga yaitu rancangan bangun program deteksi kecepatan kendaraan dengan metode *mixture of gaussian 2* dan *k-nearest neighbor*. Tahap rancang bangun

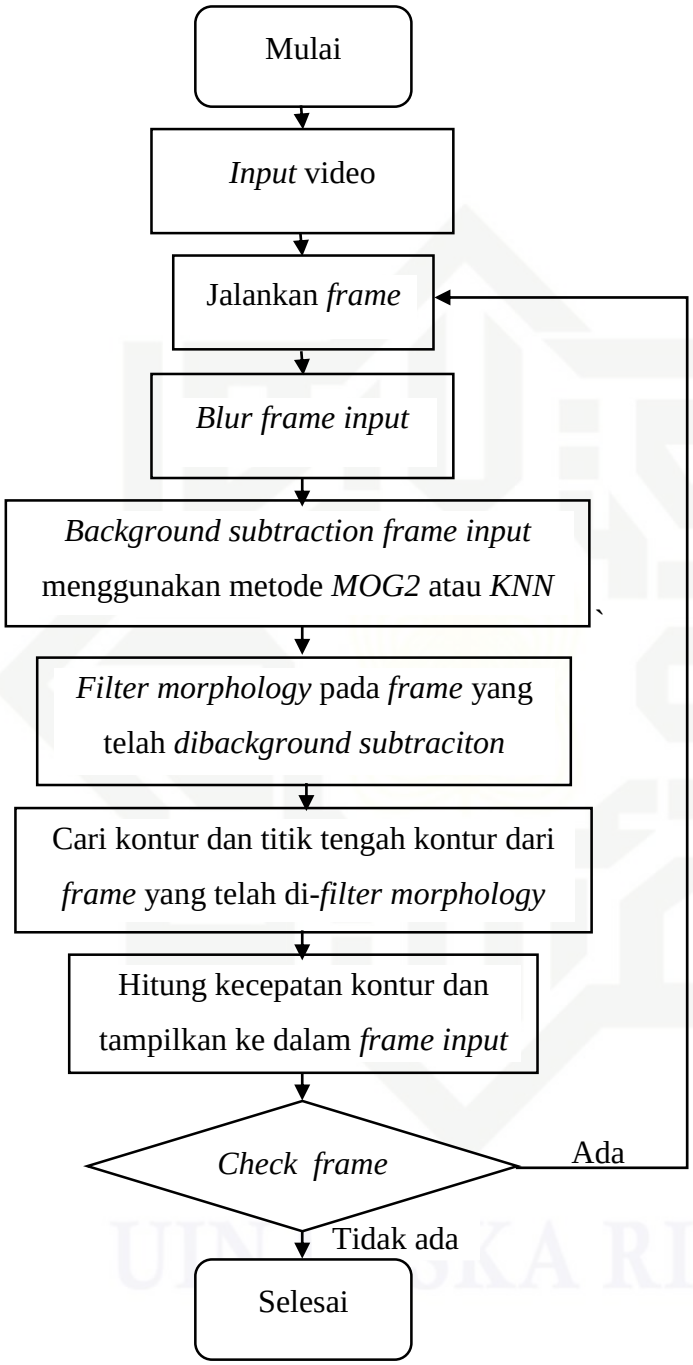
program deteksi kecepatan kendaraan menggunakan metode *mixture of gaussian 2* dan *k-nearest neighbor* dimulai dari membaca *input* video, kemudian setiap *frame* yang ada di video *input* dijalankan. Setiap *frame* video tersebut akan diproses untuk mendapatkan kecepatan yang terdeteksi pada setiap *frame*, proses pertama yaitu memanggil fungsi *blur* untuk menghilangkan *noise*. Proses kedua yaitu memanggil fungsi *mixture of gaussian 2* atau *k-nearest neighbor* untuk mendapatkan objek yang bergerak atau kendaraan dalam bentuk citra biner. Namun hasil citra biner yang didapat dari kedua metode tersebut masih terdapat *noise*, sehingga program akan melakukan *filter* pada citra biner. *Filter* yang digunakan untuk citra biner adalah *filter morphology open* dan *close*. Proses ketiga adalah mencari kontur dan menghitung titik tengah dari kontur untuk mendapatkan jarak perpindahan kontur setiap *frame*. Proses selanjutnya menghitung kecepatan kendaraan setiap *frame* berdasarkan persamaan kecepatan kendaraan yang tedapat dalam penelitian sadewo dan kawan-kawan pada tahun 2015. Setelah kecepatan kendaraan didapat, kecepatan ditampilkan diatas objek.

Langkah ketiga adalah menentukan parameter-parameter yang akan dianalisa. Sebelum program yang dirancang dan dibangun diuji, parameter diambil untuk dianalisa adalah kecepatan terdeteksi, waktu perpindahan *frame* dan jarak perpindahan titik tengah kontur. Parameter ini akan dianalisa untuk mendapatkan perbandingan akurasi metode mixture of gaussian 2 dengan k-nearest neighbor dalam mendeteksi kecepatan kendaraan.

Langkah terakhir adalah pengujian metode. Metode yang digunakan dalam program diuji dengan input video yang telah direkam pada langkah pertama. Empat video yang terdapat skenario kecepatan kendaraan diuji pada program menggunakan *mixture of gaussian 2* dan program menggunakan metode *k-nearest neighbor* yang telah dibangun untuk mendapatkan kecepatan kendaraan yang terdeteksi oleh program. Pengujian kedua program pada setiap video dilakukan dengan 10 - 12 kali percobaan, dengan mayoritas parameter dari 10-12 kali percobaan yang dijadikan parameter penelitian.

3.1 Alur kerja program *Mixture Of Gaussian 2* dan *program K-Nearest Neighbor*

Alur kerja program deteksi kecepatan kendaraan menggunakan metode *mixture of gaussiang 2* dan *k-nearest neighbor* dijelaskan dengan Gambar 3.2.



Gambar 3.2 *Flowchart* cara kerja program deteksi kecepatan kendaraan.

Alur kerja program pendeteksi kecepatan kendaraan berbasis video pada penelitian ini terdapat beberapa langkah. Pertama video sekenario kecepatan yang telah direkam di-*input* kedalam program. *Input video* dilakukan didalam program dengan melampirkan posisi video *input* dalam kode program. Selanjutnya program akan menjalankan setiap *frame* dari

video *input* berdasarkan urutan *frame*. Selanjutnya program mem-*blur frame input* untuk menghilangkan *noise*. Proses *blur frame input* ini dilakukan dengan memanggil fungsi *blur()*.Setelah *frame input* di-*blur*, selanjutnya *frame input* akan dibiner dengan metode *background subtraction*. *Background subtraction* merupakan proses membiner *frame* video untuk menetukan *foreground* dan *background*. Untuk *foreground* diberi nilai 0 dan *background* diberi nilai 1. Pembineran *frame input* dengan *background subtraction* dilakukan dalam dua tahap, tahap pertama dengan memanggil fungsi *createBackgroundSubtractor()*. Pada tahap ini metode akan diatur nilai *history*, nilai *thresholding* dan nilai biner rekayasa bayangan. Kemudian fungsi ini disimpan dalam sebuah variabel. Selanjutnya variabel tersebut akan di-*apply* pada *frame input*. Metode yang dipakai yaitu *MOG2* dan *KNN*, dimana setiap program memiliki satu metode *background subtraction*. Selanjutnya program melakukan *filter morphology*, proses *filter* ini dilakukan dengan memanggil fungsi *morphologyEx()*. *Filter morphology* berfungsi untuk menghilangkan *noise* pada *frame* biner. Pada program ini filter yang dipakai adalah *filter morphology openin*g dan *closing*. Pertama *noise* disekitar objek direduksi menggunakan *filter morphology opening*. Proses *filter morphology opening* dilakukan dengan memanggil fungsi *morphologyEx()* yang didalam fungsinya terdapat *syntax CV_MOP_OPEN*. Kemudian *noise* didalam *foreground* atau objek direduksi menggunakan *filter morphology closing*. Proses *filter morphology closing* dilakukan dengan memanggil fungsi *morphologyEx()* yang didalam fungsinya terdapat *syntax CV_MOP_CLOSE*. Selanjutnya menemukan kontur dalam setiap *frame* yang telah di-*filter* sebelumnya. Pencarian kontur dilakukan dengan menggunakan fungsi *findContours()*. Setelah kontur ditemukan, selanjutnya kontur diseleksi berdasarkan koordinat batas area deteksi. Langkah selanjutnya yaitu mencari koordinat titik tengah kontur yang telah melewati area deteksi. Setelah didapatkan nilai titik tengah, langkah selanjutnya melakukan menghitung kecepatan kontur dengan persamaan yang digunakan pada penelitian sadewo dan kawan-kawan tahun 2015. Setelah kecepatan kontur didapat, langkah selanjutnya konversi kecepatan kontur dalam satuan km/jam. Langkah selanjutnya memberi *bounding box* dan menampilkan kecepatan kendaraan diatas objek yang terdeteksi dalam setiap *frame input*. Memberi *bounding box* dilakukan dengan menggunakan fungsi *rectangle*() sedangkan menampilkan kecepatan didalm *frame input* dilakukan dengan memanggil fungsi *putText()*.

3.2 Perekaman video

Tahap pertama pada penelitian ini adalah perekaman video. Video yang di rekam berlokasi di jalan soedirman pekanbaru, Riau pada pukul 10:54 WIB. Perekaman video menggunakan kamera *handphone* dengan resolusi kamera 13 MP dengan ukuran *frame* video yang dihasilkan adalah 720 × 1280 piksel. Posisi bayangan pada pukul 10:54 WIB adalah tegak lurus terhadap benda. Perekaman video menggunakan skenario kecepatan yang dibuat menggunakan sepeda motor dengan kecepatan 20 Km/jam, 40 Km/jam, 50 Km/jam, 60 Km/jam yang direkam dari atas jembatan penyebrangan dijalan Jendral Soedirman, Pekanbaru, Riau dengan kamera menghadap diagonal kebawah. Empat skenario kecepatan kendaraan ini yang dijadikan parameter kecepatan sebenarnya.

3.3 Menentukan dan mengukur batas area deteksi

Langkah kedua dari penelitian ini adalah menentukan dan mengukur batas area deteksi program. Setelah video direkam, selanjutnya video di *play* untuk menentukan batas area deteksi. Batas areadeteksi yang dididapat setelah video hasil rekaman tersebut di *play* adalah sepanjang marka cat pembatas jalur. Marka cat pembatas jalur dijadikan batas deteksi dikarenakan penulis lebih mudah mengukur panjang sebenarnya marka cat pembatas jalan dan panjang marka cat pembatas jalur didalam video. Batas area deteksi pada penelitian ini dapat dilihat pada Gambar 3.3.

Gambar 3.3 Batas area deteksi program deteksi kecepatan kendaraan.

Langkah selanjutnya mengukur panjang area batas deteksi, panjang area deteksi ini menjadi panjang lintasan yang terdeteksi sistem, setelah diukur didapatkan panjang marka

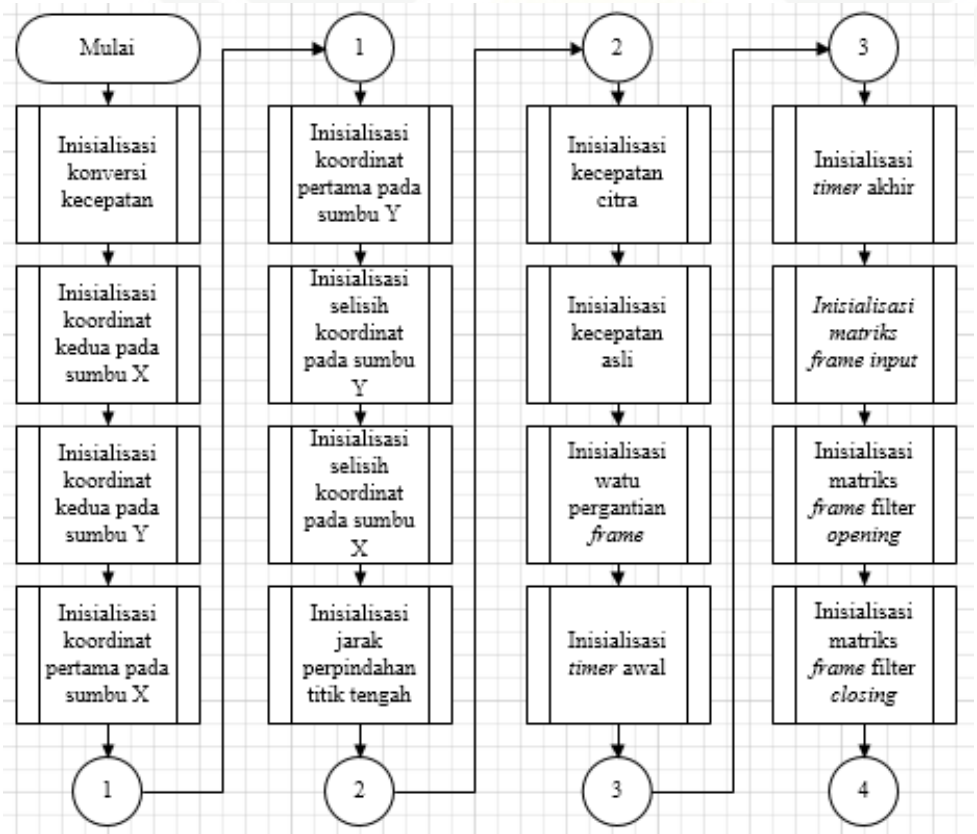
cat pembatas jalur tersebut adalah 2.98 meter. Parameter panjang area deteksi ini digunakan untuk mendapat kecepatan yang terdeteksi program.

3.4 Rancang bangun program deteksi kecepatan kendaraan

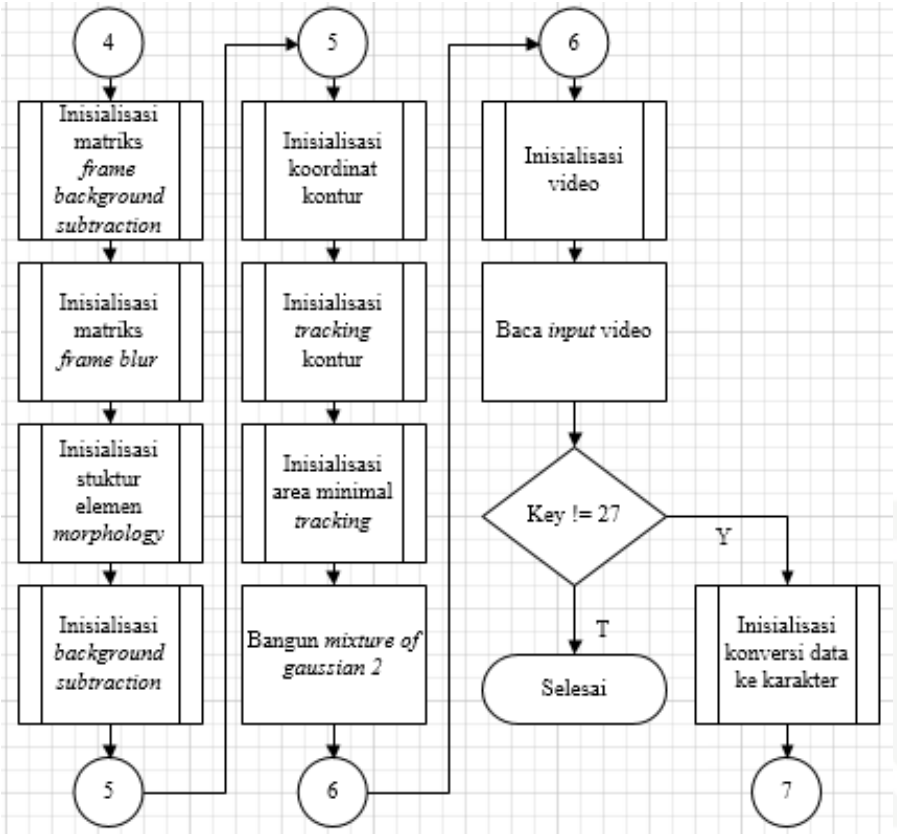
Tahap kedua dari penelitian ini adalah perancangan program deteksi kecepatan. Program yang akan dirangcang menggunakan dua metode yaitu metode *mixture of gaussian 2* dan metode *k-nearest neighbors*.

3.4.1 Rancangan bangun program deketsi kecepatan kendaraan menggunakan metode *Mixture Of Gaussian 2*

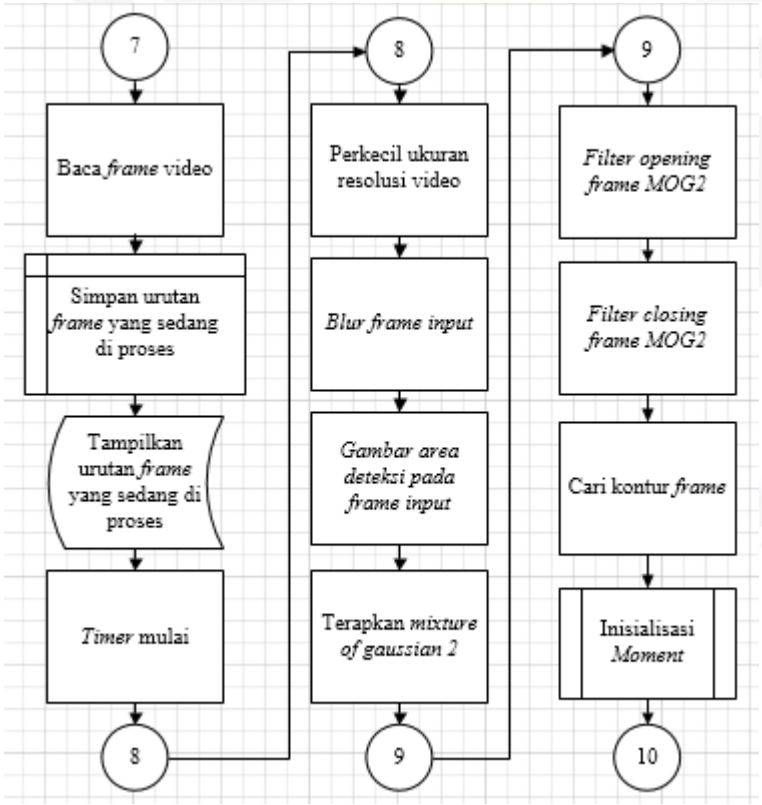
Perancangan program deteksi kecepatan kendaran menggunakan metode *mixture of gaussian 2* digambarkan dalam bentuk *flowchart* untuk memudahkan dalam menerjemahkan kedalam bahasa pemrograman C. *Flowchart* program perancangan program deteksi kecepatan kendaraan menggunakan metode *mixture of gaussian 2* dapat di lihat pada Gambar 3.4.



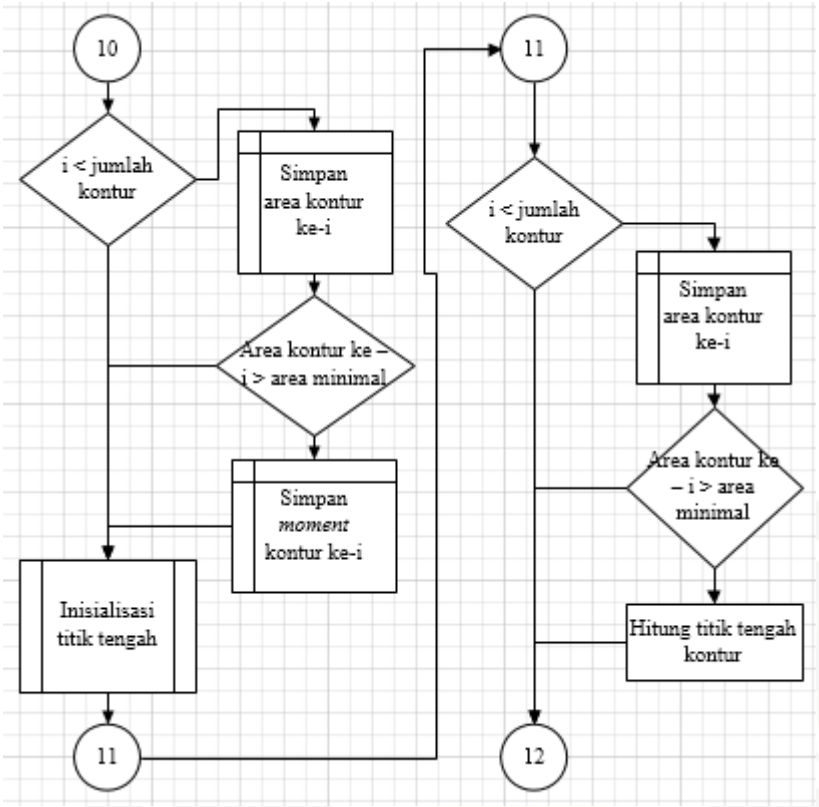
Gambar 3.4 *Flowchart* program dengan metode *MOG2*.



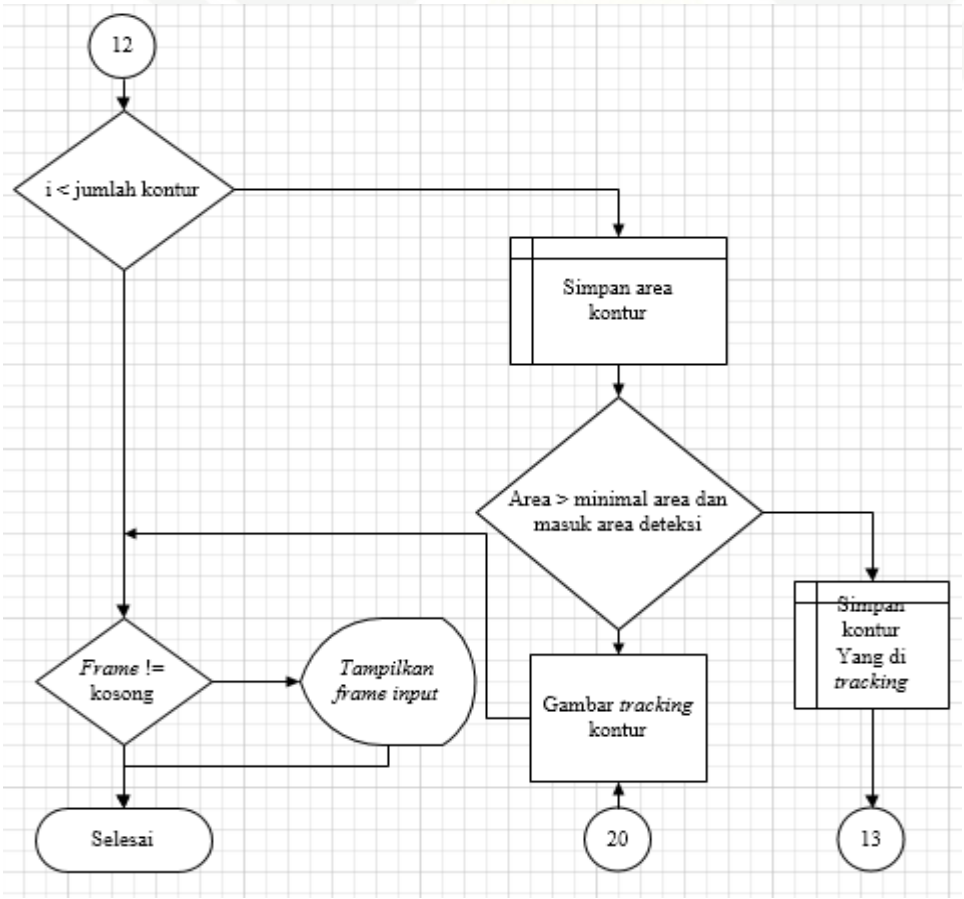
Gambar 3.5 *Flowchart* program dengan metode *MOG2* lanjutan.



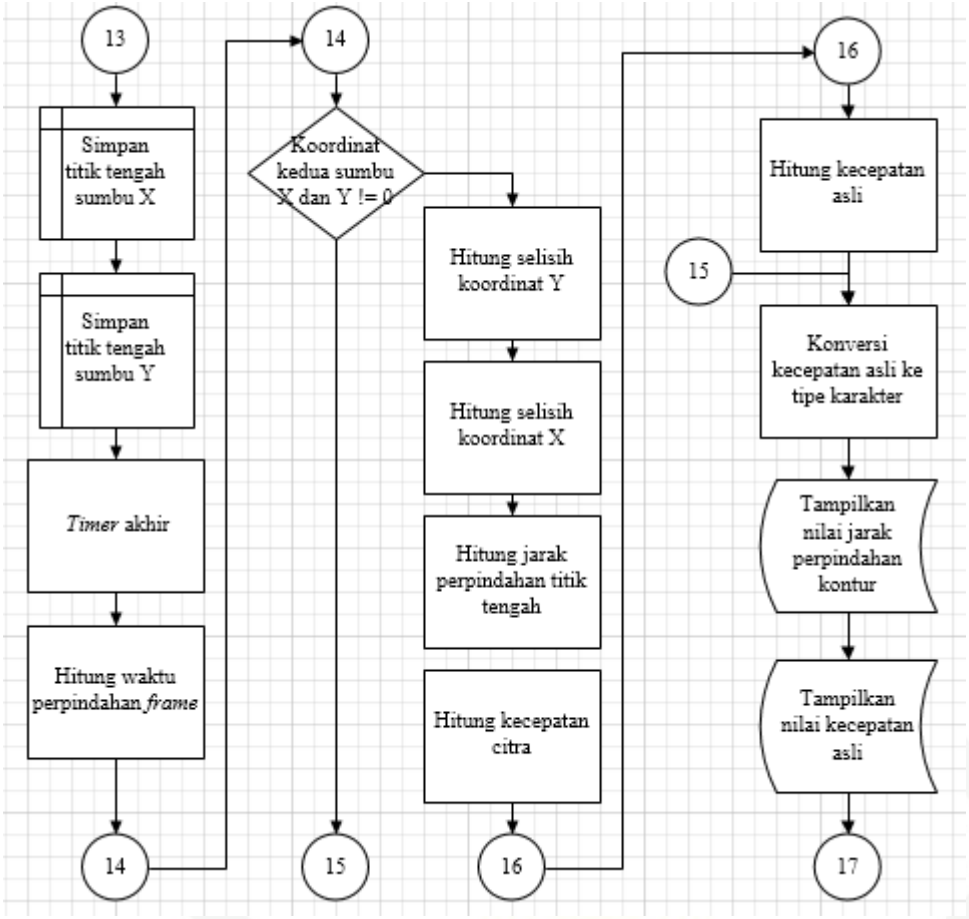
Gambar 3.6 *Flowchart* program dengan metode *MOG2* lanjutan.



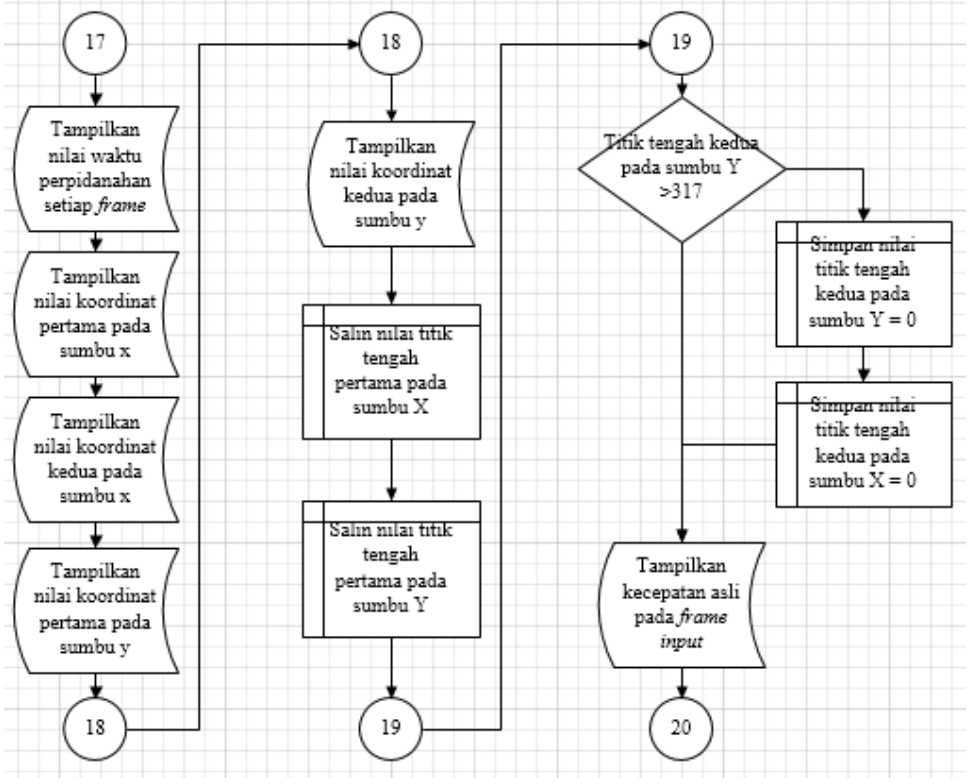
Gambar 3.7 *Flowchart* program dengan metode *MOG2* lanjutan.



Gambar 3.8 *Flowchart* program dengan metode *MOG2* lanjutan.



Gambar 3.9 *Flowchart* program dengan metode *MOG2* lanjutan.



Gambar 3.10 *Flowchart* program dengan metode *MOG2* lanjutan.

Pada program deteksi kecepatan kendaraan menggunakan metode *MOG2*, program dimulai dengan dengan inisialisai data-data yang diperlukan. Data pertama adalah inisialisasi konversi kecepatan. Inisialisasi konversi kecepatan di simpan kedalam variabel *mstokmh*. Kecepatan citra dengan satuan *m/s* dikonversi mejadi *Km/h*. karena hasil konversi kecepatan berupa bilangan desimal, maka tipe data yang digunakan adalah *float*. Kode program yang digunakan adalah :

```
……
float mstokmh = (1000.0/3600)
…..
```

Langkah berikutnya adalah inisialisasi koordinat kedua pada sumbu x. koordinat kedua pada sumbu x di inisialisasi dengan nama variabel *x2* dengan nilai awal nol (0) dengan tipe data *float*. Selanjutnya inisialisasi koordinat kedua pada sumbu y, inisialisasi koordinat kedua pada sumbu y di inisialisasi dengan nama variabel *y2* dengan nilai nol (0) dengan tipe data *float*. Kemudian inisialisasi koordinat pertama pada sumbu x dengan nama variabel *x1*, selanjutnya inisialisasi koordinat pertama pada sumbu y dengan nama variabel *y1*. Selanjutnya inisialisasi selisih koordinat pertama dengan kedua pada sumbu y dengan nama variabel *yt* dan inisialisasi selisih koordinat pertama dengan kooridnat kedua pada sumbu x dengan nama variabel *xt*. Selanjutnya inisialisasi jarak perpindah titik tengah objek dengan nama variabel *s*. Kemudian inisialisasi kecepatan citra dengan satuan m/s dengan nama variabel *vms* dan inisialisasi kecepatan yang telah di konversi kedalam satuan km/h dengan nama varibel *vasli*. Selanjutnya inisialisasi *timer* mulai menghitung waktu dengan nama variabel *awal*. Dan variabel *timer* untuk menghentikan waktu dengan nama variabel *akhir*, Serta inisialisasi waktu perpindahan *frame* dengan nama variabel *t* yang bernilai nol. Setiap variabel diatas di inisialisasi dengan tipe data *float*. Kode program inisialisasi secara berturut-turut adalah :

```
……
float x2 = 0;
float y2 = 0;
float x1, y1, yt, xt, s, vcitra, vasli, awal, akhir;
float t = 0;
…..
```

Langkah selanjutnya adalah insialisasi matriks untuk *frame input* pada video input. *Frame input* video di inisialisasi dengan nama matriks *src*. Selanjutnya inisialisasi matriks untuk *frame* yang telah di *filter morphology opening* dengan nama matriks *opening*. Kemudian inisialiasi matriks untuk *frame* yang telah di *filter morphology closing* dengan nama matriks *closing*. Selanjutnya inisialisasi matriks untuk *frame input* yang telah di *background subtraction* dengan nama matriks *bs*. Dan inisialisasi matriks untuk *frame input* yang telah di *blur* dengan nama matriks *blur1*. Inisialisasi matriks *frame* ini menggunakan tipe data *Mat* pada *library opencv* dimana citra atau *frame* pada video dikonversi kedalam sebuah bilangan *real* pada setiap baris dan kolom piksel *frame* atau citra dengan panjang baris dan kolom sesuai ukuran piksel frame. Sehingga bilangan tersebut dapat di proses. Kode program yang digunakan pada inisalisasi matriks untuk *frame* secara berturut-turut adalah :

```
……
Mat src, opening, closing, bs, blur1;
…..
```

Langkah selanjutnya adalah inisalisasi *element* pada *filter morphology*. Struktur *element* disimpan dalam tipe data *Mat*. *Mat* merupakan matriks yang disediakan *library opencv* dengan nama matriks *element*. Sturuktur *element morphology* yang digunakan adalah struktur *element rectanguler*, ukuran piksel yang akan di *filter* yaitu 5 x 5 piksel dan *filter* dimulai pada pada sumbu x = 0 dan y = 0. Kode program yang digunakan adalah :

```
……
mat element = getStructuringElement (MORP_RECT, Size(5, 5), point (0, 0));
…..
```

Langkah selanjutnya adalah inisialisasi *background subtraction*. Inisialisasi ini berfungsi menyimpan jenis metode *background subtraction* yang akan digunakan. Inisialisasi *background subtraction* di simpan dengan nama variabel *bsMOG2*. Kode program yang digunakan adalah :

```
……
Ptr<BackgroundSubtractor> bsMOG2;
…..
```

Langkah selanjutnya adalah inisialisasi koordinat kontur. Inisialisasi koordinat di simpan dalam sebuah variabel dengan nama *contur*. Kemudian vektor *output* kontur dengan dimensi 4x4 disimpan dengan variabel *hierarchy*. Kode program yang digunakan adalah :

```
……
vector<vector<point>> countours;
vector<Vec4i> heirarchy;
…..
```

Langkah berikutnya inisialisasi *tracking kontur*. Objek kontur di *tracking* dengan bentuk persegi panjang. Inisialisasi *tracking* kontur di simpan dengan nama *bounding_rect*. Kode program yang digunakan adalah :

```
……
Rect bounding_rect;
…..
```

Langkah selanjutnya adalah inisialisasi luas minimal pada area yang akan di *tracking*. Luas minimal area yang ditracking adalah 1300 piksel^2 dengan tipe data *integer*. Kode program yang digunakan adalah :

```
……
int min_area = 1300;
…..
```

Langkah berikutnya adalah membangun *mixture of gaussian 2*. *Mixture of gaussian 2* dibangun dengan memanggil fungsi *createBackgroundSubtratorMOG2()*, pengaturan nilai *history* adalah 300, nilai *threshold* adalah 32 dan 0 untuk tidak menampilkan rekayasa bayangan. Pengaturan ini di simpan kedalam variabel yang *bsMOG2* yang telah di inisialisasi sebelumnya. Kode program yang digunakan adalah :

```
……
bsMOG2 = createBackgroundSubtratorMOG2(300, 32, 0);
…..
```

Langkah berikutnya adalah inisialisasi video. Video di inisialisasi dengan nama variabel *capture*. Kemudian langkah selanjutnya adalah membaca video *input*. Video *input* dapat diambil dari kamera internal atau eksternal dari sebuah *pc* atau laptop dan dapat juga dari video yang disimpan pada media penyimpanan. Kode program beturut-turut yang digunakan adalah :

```
……
VideoCapture caputre;
Capture.open (“D:/60km.avi”);
…..
```

Langkah selanjutnya adalah pengulangan yang berfungsi untuk memproses setiap *frame* berdasarkan urutan *frame* dari video *input*. Dan proses akan dihentikan apabila *user* menekan tombol *esc* dengan nilai *ASCII* 27. Kode program yang digunakan adalah :

```
……
while (waitKey(1) != 27)
{
…..
}
```

Langkah selanjutnya inisialisasi konversi data ke dalam bentuk karakter. Data hasil kalkulasi kecepatan akan di konversi ke dalam tipe data karakter. Nama variabel untuk mengkonversi hasil kalkulasi kedalam bentuk karakter adalah *ss*. Hasil konversi data tersebut digunakan untuk menampilkan kecepatan objek yang terdeksi dalam setiap *frame*. Nama variabel yang akan menampilkan kecepatan disetiap *frame* adalah *tampilkan*. Kode program yang digunakan adalah :

```
……
stringstream ss;
string tampilkan;
…..
```

Langkah berikutnya adalah membaca setiap *frame* video *input* kedalam matriks *frame input*. Kemudian simpan informasi *frame* yang sedang diproses kedalam variabel *framesrkg* yang di inisialisasi dengan tipe data *integer*. Kemudian tampilkan *frame* yang sedang di proses dalam *command prompt*. Kode program yang digunakan secara berturut-turut adalah :

```
……
capture >> src;
int frameskrg = capture.get(CAP_PROP_POS_FRAME);
cout << "frame ke- "<< frameskrg << endl;
…..
```

Langkah berikutnya adalah memulai *timer*. *Timer* di mulai apabila *frame* akan di proses. Kemudian resolusi *frame* video akan di perkecil 5/2 lebih kecil dari *frame input*nya. Pengecilan ukuran *frame* video dilakukan dengan memanggil fungsi *resize()*. Pengecilan ukuran *frame* berfungsi untuk memperkecil waktu proses setiap *frame*. Kode program yang digunakan secara berturut-turut adalah :

```
……
awal = clock();
resize (src, src, Size (src.size().width/2.5, src.size().height/2.5));
…..
```

Langkah berikutnya mem-*blur frame input*. *Frame* dari video *input* di-*blur* dengan memanggil fungsi *blur()*. Tujuan *frame* di *blur* untuk menghilangkan *noise* sebelum *frame input* akan di *threshloding*. Proses *blur* ini mem-*blur* setiap piksel pada *frame input* dengan ukuran panjang 5 piksel dan lebar 5 piksel. *Output* dari proses *blur* ini akan disimpan dalam matriks *blur1*. Kode program yang digunakan adalah :

```
……
blur (src, blur1, size ( 5, 5));
…..
```

Langkah selanjutnya adalah menggambar area deteksi di dalam *frame input*. Hal ini memudahkan pengguna dalam melihat area yang terdeteksi oleh sistem. Fungsi *line()* dipanggil Untuk menggambar garis di-*frame* video. Fungsi ini akan menggambar garis dari titik-titik koordinat yang telah dilampirkan dalam program. Kode program yang digunakan adalah sebagai berikut :

```
……
ine(src, Point(91, 345), Point(106, 274), scalar (0, 255, 255), 2) ;
ine(src, Point(106, 274), Point(193, 278), scalar (0, 255, 255), 2) ;
ine(src, Point(91, 345), Point(197, 349), scalar (0, 255, 255), 2) ;
ine(src, Point(193, 278), Point(197, 349), scalar (0, 255, 255), 2) ;
…..
```

Langkah selanjutnya menerapkan *background subtraction MOG2*. Penerapan *MOG2* ini dilakukan dengan memanggil fungsi *apply()* dari variabel yang menyimpan *background subtraction*. Penerapan *MOG2* ini untuk mendeteksi objek yang bergerak dengan nilai biner dan *frame output* berwarna hitam putih dari *frame input*. Untuk benda bergerak dinyatakan dengan warna putih, sedangkan untuk latar belakang dinyatakan dengan warna hitam. Pengaturan metode *background subtraction* yang di simpan dalam variabel *bsMOG2* pada langkah sebelumnya di

terapkan pada *frame* yang telah di-*blur* pada langkah sebelumnya. Kemudian *output* dari proses *background subtraction* disimpan kedalam matriks dengan nama *bs*. Kode program yang digunakan adalah :

```
……
bsMOG2->apply(blur1,bs);
…..
```

Langkah selanjutnya adalah *filter morphology opening* pada *frame* yang telah di-*background subtraction MOG2*. *Filter morphology opening* berfungsi untuk menghilangkan *noise* di sekitar objek pada *frame* yang telah di *background subtraction MOG2*. *Filter*

morphology opening dilakukan dengan memanggil fungsi *morphologyEx()* yang terdapat *syntax CV_MOP_OPEN* dalam fungsi tersebut. *Frame* yang telah di *background subtraction MOG2* yang telah di-*filter* disalin kedalam matriks dengan nama *opening*. Langkah berikutnya adalah *filter morphology closing* pada *frame* yang telah di-*filter morphology opening*. *Filter morphology closing* ini dilakukan dengan memanggil fungsi *morphologyEx()* yang terdapat *CV_MOP_CLOSE* didalam fungsi tersebut. *Filter morphology closing* bertujuan untuk menghilangkan warna hitam didalam objek yang terdeteksi atau berwarna putih. *Frame morphology opening* yang telah di *filter morphology closing* di salin kedalam matriks *closing*. Struktur elemen pada kedua *filter morphology* teresebut menggunakan struktur elemen yang telah diinisialisasi pada langkah sebelumnya. Kode program yang digunakan secara berturut-turut adalah :

```
……
morphologyEx(bs, opening, CV_MOP_OPEN, element);
morphologyEx(opening, closing, CV_MOP_CLOSE, element);
…..
```

Langkah berikutnya adalah mencari kontur pada *frame* yang telah di *filter morphology* pada langkah sebelumnya. Setelah kontur di temukan kontur tersebut disimpan dalam variabel beranama *contours*. Pencarian kontur dilakukan dengan memanggil fungsi *findContours()*. Kode program yang digunakan adalah :

```
……
findContours (closing, opening, hierarchy, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE):
…..
```

Langkah berikutnya mencari *moment* dari kontur yang di dapat. Mencari *moment* ini berfungsi dalam menentukan titik tengah dari sebuah kontur yang ada di setiap *frame*. Langkah pertama mencari *moment* adalah inisialisasi *moment* dengan nama variabel *momentum*. Kemudian luas kontur di simpan dalam variabel *area*. *Moment* yang di cari harus di seleksi pada kontur yang melebihi area minimal yaitu 1300 piksel^2. *Moment* yang dicari dilakukan dengan memanggil fungsi *moments()*. *Moment* ini disimpan sama berdasarkan urutan kontur setiap *frame*. Kode program yang digunakan adalah :

```
……
vector<Moments> momentum(contours.size());
for(int i =0; i<contours.size(); i++);
{
        double area = contourArea(contours[i], 0);
if(area>min_area){
momentum[i] = moments(contours[i], false);
        }
}
…..
```

Langkah selanjutnya adalah mencari titik tengah berdasarkan *moment* setiap kontur yang telah di temukan. Pertama inisialisasi titik tengah kontur dengan nama variabel *center*. Selanjutnya simpan luas area setiap kontur dalam variabel *area*. Kemudian seleksi setiap kontur yang memiliki luas kontur lebih dari luas minimum. Kemudian kalkulasi titik tengah berdasarkan *moment* yang telah di temukan pada setiap kontur. Langkah berikutnya simpan titik tengah setiap kontur dalam variabel *center*. Kode program yang digunakan adalah :

```
……
vector<Point2f> center(contours.size());
for(int i =0; i<contours.size(); i++);
{
        double area = contourArea(contours[i], 0);
if(area>min_area){
        center[i] = Point2f ( momentum[i]. m10/ momentum[i]. m00, momentum[i]. m01/
momentum[i]. m00;
        }
}
```

Langkah berikutnya adalah memproses setiap kontur pada setiap *frame* video *input*. Setiap luas area kontur di simpan kedalam variabel *area*. Kemudian semua kontur yang terdapat dalam *frame* di seleksi dengan luas area kontur minimal 1300 piksel^2 dan melewati

batas deteksi dengan sumbu x antara 97 hingga 197 dan sumbu y antara 274 hingga 349. Kode program yang digunakan adalah :

```
……
for(int i =0; i<contours.size(); i++);
{
        double area = contourArea(contours[i], 0);
if(area>min_area && center[i].y > 274 && center[i].y < 349 && center[i].x > 97 && cenrer[i].x < 197){
…..
```

Selanjutnya kontur yang memenuhi seleksi akan di simpan ke dalam variabel *bounding_rect* untuk di *tracking* atau di *bounding box*. Selanjutnya sumbu x pada kontur yang memenuhi seleksi diatas akan di simpan pada variabel *x1*. Sedangkan sumbu y pada kontur yang memenuhi seleksi diatas akan disimpan dalam variabel *y1*. Kode program yang digunakan secara berturut-turut adalah :

```
……
        bounding_rect = boundingRect(counturs[i]);
        x1= center[i].x;
        y1= center[i].y;
…..
```

Selanjutnya menghentikan *timer*. *Timer* dihentikan dan disimpan kedalam variabel *akhir*. Kemudian waktu perpindahan ini akan dihitung dengan satuan detik. Kode program yang digunakan adalah :

```
……
        akhir = clock();
        t = (akhir – awal)/CLOCK_PER_SEC;
…..
```

Selanjutnya melakukan perhitungan kecepatan. Pada langkah ini titik tengah kedua pada sumbu x dan sumbu y ini akan diseleksi. Karna pada saat objek yang pertama kali melewati batas deteksi tidak memiliki titik tengah kedua pada sumbu x dan sumbu y

sehingga proses menghitung kecepatan di lewati. Kemudian langkah selanjutnya menghitung selisih kedua koordinat pada sumbu x dan sumbu y. Langkah selanjutnya mencari jarak perpindahan titik tengah sesuai dengan persamaan 2.1. Kode program yang digunakan secara berturut-turut adalah :

```
……
        if(x2!=0|| y2!=0)
{
        yt = y2 - y1;
        xt = x2 – x1;
        s = sqrt(pow(xt, 2) + pow(yt, 2));
…..
```

Langkah selanjutnya menghitung kecepatan kendaraan. Persamaan yang digunakan didapat dari penelitian sebelumnya (Sadewo dkk, 2015) tentang *Sistem Pengukur Kecepatan Kendaraan Berbasis Video* yaitu :

$$kecepatan\ asli = \frac{jarak\ perpindahan\ kontur\ x\ panjang\ lintasan\ asli}{waktu\ x\ panjang\ lintasan\ pada\ citra} \quad 3.1$$

Langkah selanjutnya menentukan panjang lintasan pada citra atau *frame*. Berdasarkan seleksi kontur pada langkah sebelumnya, setiap kontur bergerak dari sumbu y = 274 hingga y = 349. Sehingga panjang lintasan pada citra yang didapat adalah 75 piksel. Sedangkan panjang lintasan sebenarnya yang telah di ukur pada langkah sebelumnya adalah 2.98 meter. Setelah kecepatan didapat dalam satuan *m/s* , kecepatan ini dikonversi ke dalam satuan *km/h*. Langkah selanjutnya adalah menyimpan kecepatan yang telah diukur ke dalam variabel *ss* untuk di konversi dalam bentuk tipe data karakter. Langkah menyimpan hasil ke variabel *ss* ini merupakan langkah terakhir dalam menghitung kecepatan. Kode program yang digunakan secara berturut-turut adalah :

```
……
        vms =( s *2.98) / (t*75);
        vasli = vms / mstokmh;
        ss << vasli << km/h;
}
…..
```

Langkah selanjutnya adalah mengkonversi data yang disimpan dalam variabel *ss* menjadi bentuk tipe data karakter yaitu string. Kemudian langkah selanjutnya adalah menampilkan nilai koordinat pertama pada sumbu x dan y, menampilkan nilai koordinat kedua pada sumbu x dan y, menampilkan jarak perpindahan kontur, waktu proses *frame* dan kecepatan kendaraan atau objek yang terdeteksi dalam *command prompt*. Hali ini berguna untuk memudahkan proses analisa sistem deteksi kendaraan. Kode program yang digunakan secara berturut-turut adalah :

```
……
tampilkan = ss.str();
cout << " s = " << s <<"\t v= " << v << vasl <<"\t t = " << t << endl;
cout <<" x1 = " << x1 << "\t x2 = " << endl;
cout <<" y1 = " << y1 <<"\t y1 = " << endl;
…..
```

Langkah selanjutnya adalah menyalin nilai titik tengah pertama ke dalam variabel titik tengah kedua pada sumbu x dan sumbu y sebelum pergantian *frame*. Sebelum pergantian *frame* titik tengah kedua pada sumbu y di seleksi apabila akan melewati akhir dari batas deteksi kecepatan, maka nilai titik tengah kedua pada sumbu y akan di jadikan nol. Hal ini berguna untuk mengurangi besar *error* apabila ada kontur selanjutnya yang melewati batas deteksi. Kode program yang digunakan adalah :

```
……
        x2 = x1;
        y2 = y1;
        if (y2 >= 317)
        {
                y2 = 0;
                x2 = 0;
        }
…..
```

Langkah selanjutnya adalah menampilkan kecepatan kontur yang terdeteksi pada *frame input*. Kecepatan yang ditampilkan didalam *frame input* dilakukan dengan memanggil

fungsi *putText()*. Kecepatan yang telah diukur kemudian di konversi ke bentuk karakter sehingga dapat ditampilkan di *frame input*. Posisi kecepatan yang ditampilkan terletak di koordinat kontur yang terdekat dengan titik awal atau nol. Kode program yang digunakan adalah :

```
……
        putText (src, tampilkan.c_str(), cv::Point(bounding_rect.x, bounding_rect.y),
FONT_HERSHEY_SIMPLEX, 0.75, cv::Scalar(0, 0, 0) , 2);
        }
…..
```

Langkah selanjutnya adalah memberi *bounding box* dalam bentuk persegi panjang. Fungsi *rectangle()* digunakan untuk membuat *bounding box* atau *tracking area* pada objek atau kontur. Kontur yang telah memenuhi seleksi pada proses sebelumnya akan diberi tanda berbentuk persegi panjang berwarna hijau yang menandakan bahwa objek atau kendaraan yang terdeteksi. Kode program yang digunakan adalah sebagai berikut :

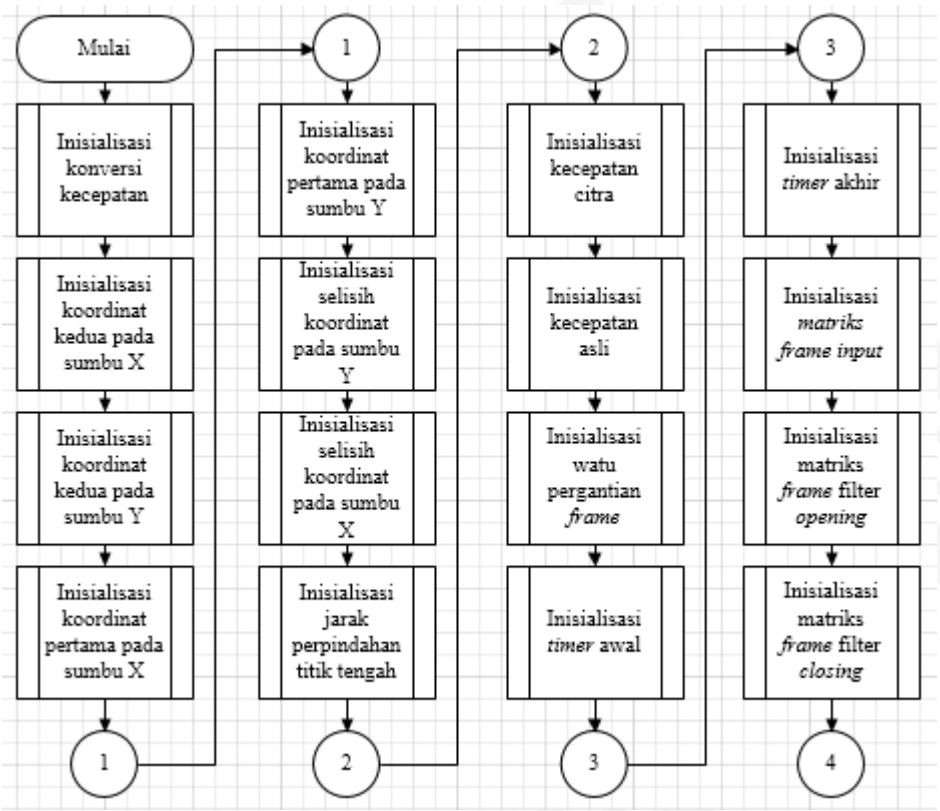
```
……
        rectangle (src, bounding_rect, Scalar (0, 255, 0), 2, 8, 0) ;
        }
…..
```

Langkah selanjutnya adalah menampilkan *frame input* yang telah diproses pada tahap sebelumnya. *Frame input* di-*check*, *kemudian frame input* ditampilkan dengan nama jendela *frame*. Kode program yang digunakan adalah :

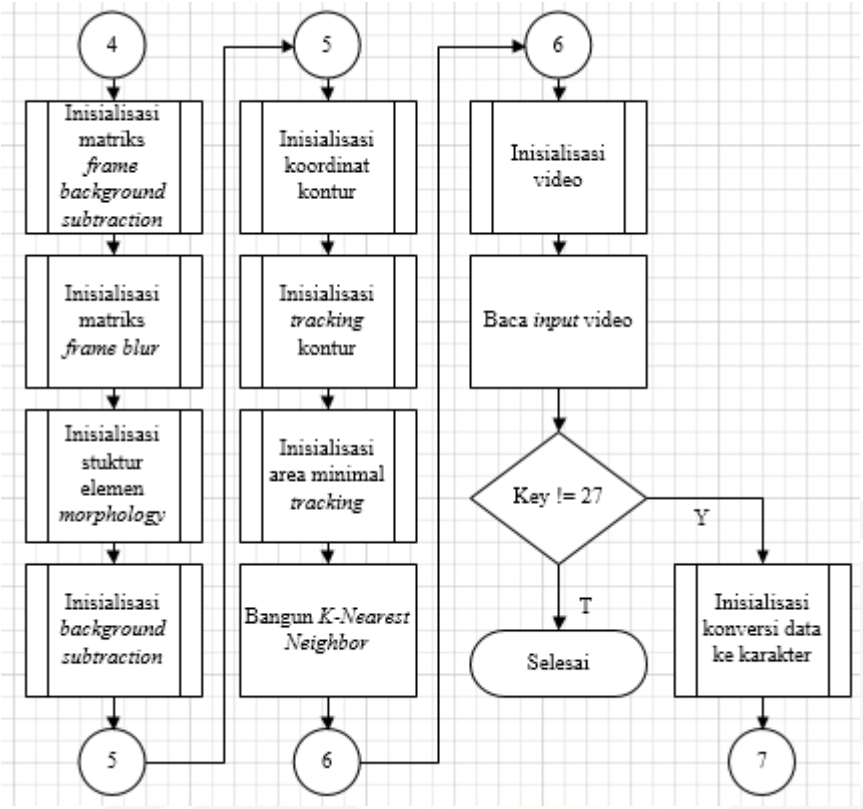
```
……
        if(!src.empty()){
                imshow("frame", src);
                }
        }
        return 0;
        }
…..
```

3.4.2 Rancang bangun program deketsi kecepatan kendaraan menggunakan metode *K-Nearest Neighbor*

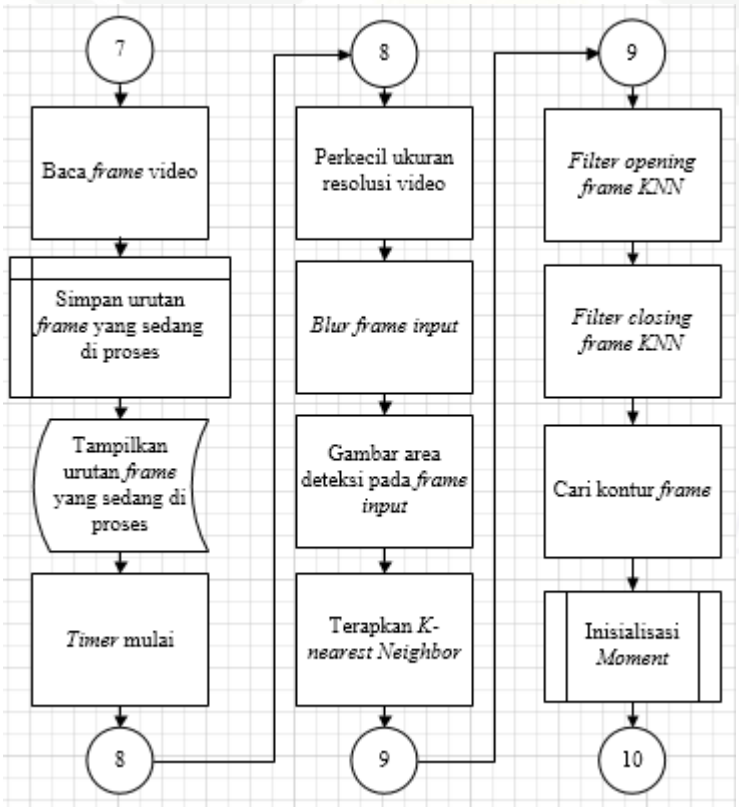
Perancangan program deteksi kecepatan kendaran menggunakan metode *K-Nearest Neighbor* digambarkan dalam bentuk *flowchart* untuk memudahkan dalam menerjemahkan kedalam bahasa pemrograman C. *Flowchart* program perancangan program deteksi kecepatan kendaraan menggunakan metode *K-Nearest Neighbor* dapat di lihat pada Gambar 3.11.



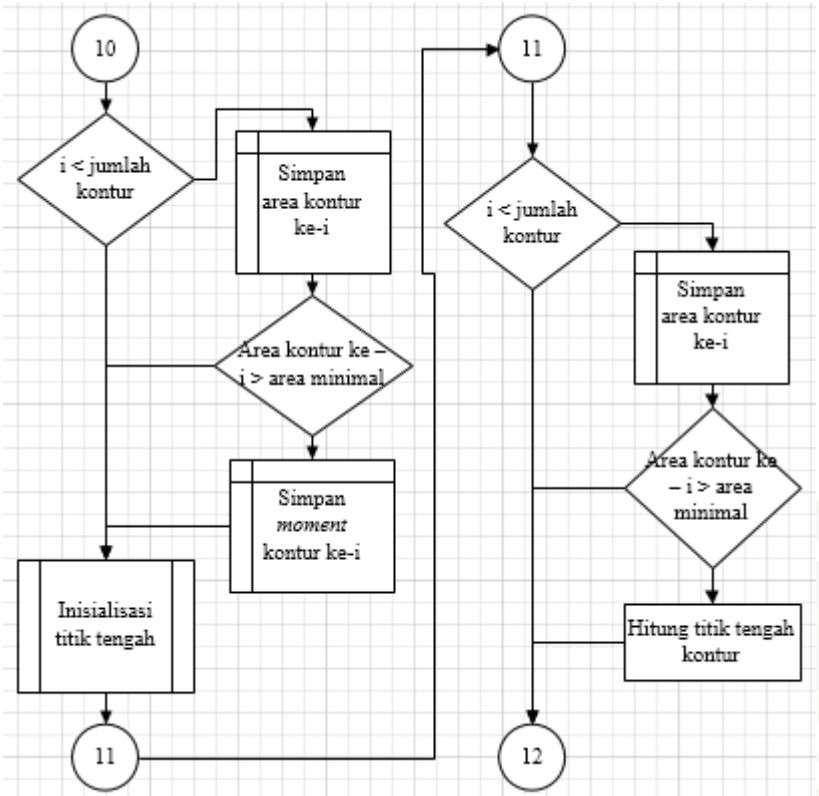
Gambar 3.11 *Flowchart* program dengan metode *KNN*.



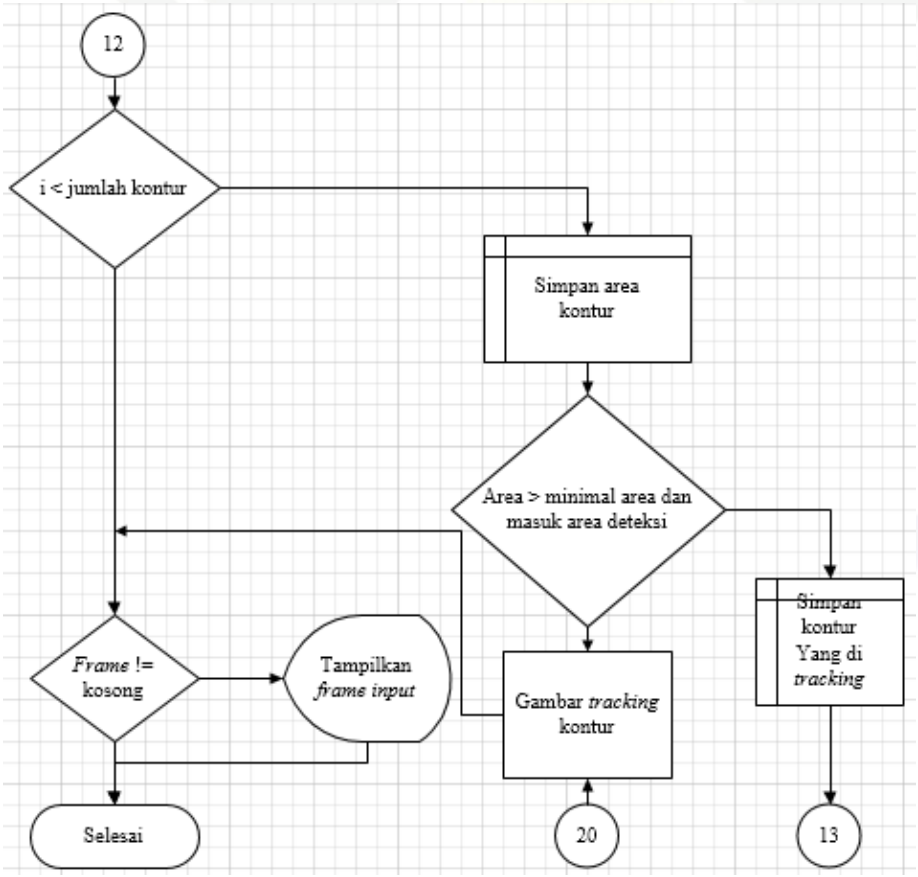
Gambar 3.12 *Flowchart* program dengan metode *KNN* lanjutan.



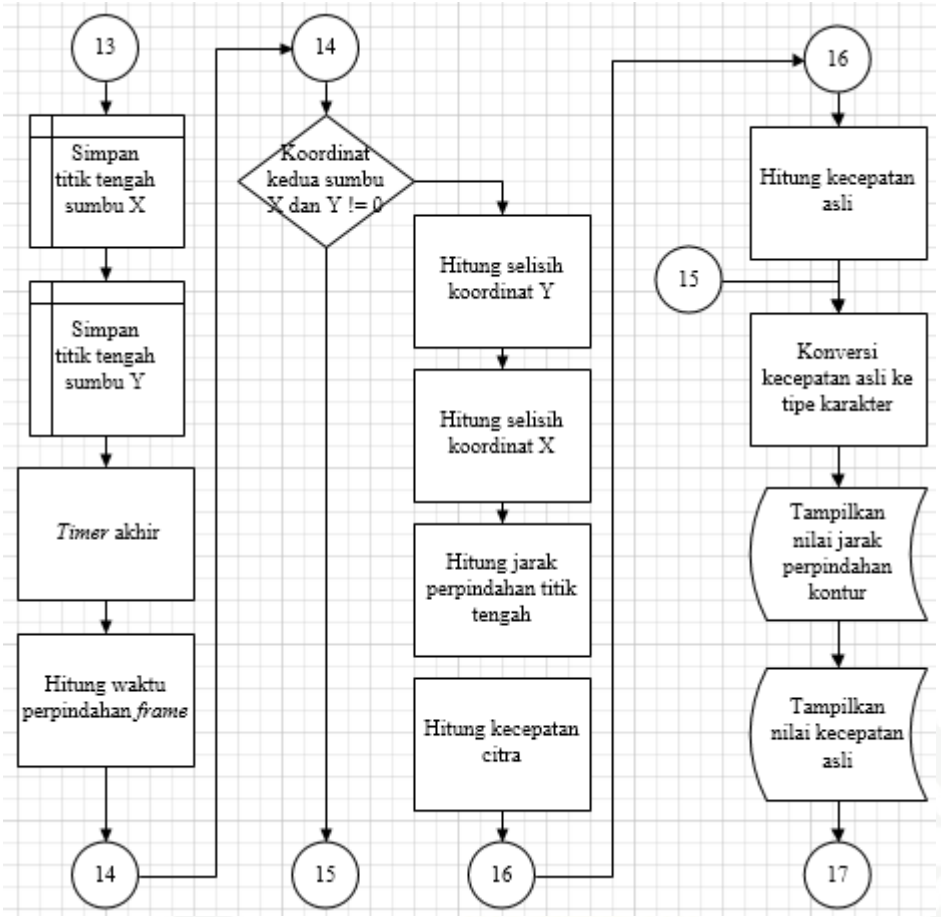
Gambar 3.13 *Flowchart* program dengan metode *KNN* lanjutan.



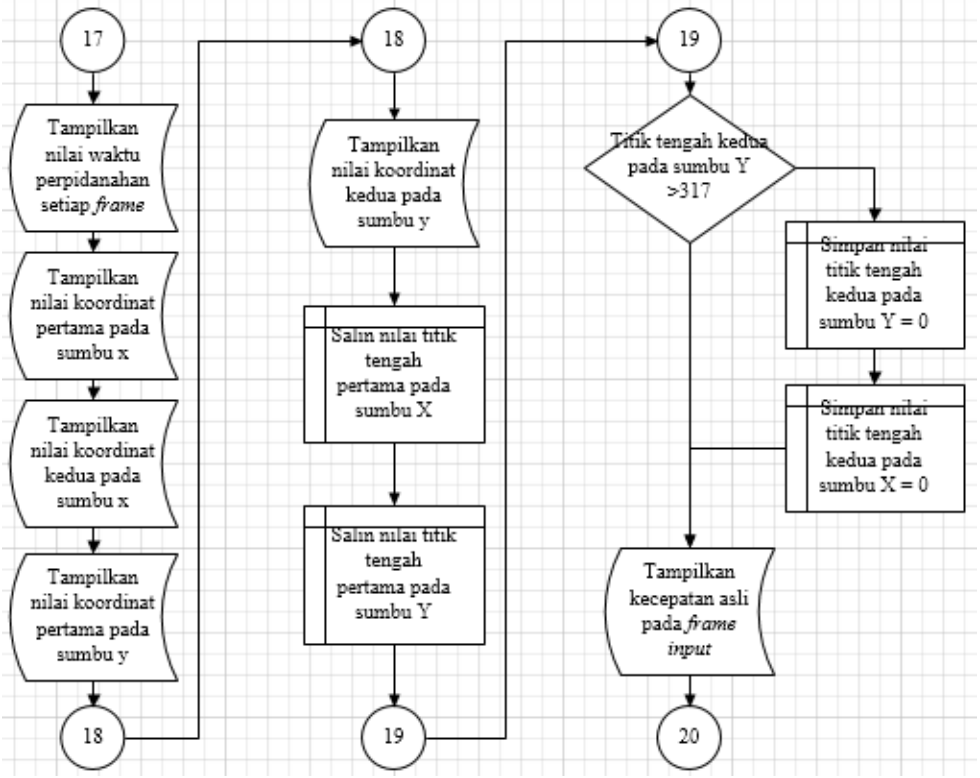
Gambar 3.14 *Flowchart* program dengan metode *KNN* lanjutan.



Gambar 3.15 *Flowchart* program dengan metode *KNN* lanjutan.



Gambar 3.16 *Flowchart* program dengan metode *KNN* lanjutan.



Gambar 3.17 *Flowchart* program dengan metode KNN lanjutan.

Pada program deteksi kecepatan kendaraan menggunakan metode *KNN*, program di mulai dengan dengan inisialisai data-data yang diperlukan. Data pertama adalah inisialisasi konversi kecepatan. Inisialisasi konversi kecepatan di simpan kedalam variabel *mstokmh*. Kecepatan citra dengan satuan *m/s* dikonversi mejadi *Km/h*. karena hasil konversi kecepatan berupa bilangan desimal, maka tipe data yang digunakan adalah *float*. Kode program yang digunakan adalah :

```
……
float mstokmh = (1000.0/3600)
…..
```

Langkah berikutnya adalah inisialisasi koordinat kedua pada sumbu x. koordinat kedua pada sumbu x di inisialisasi dengan nama variabel *x2* dengan nilai awal nol (0) dengan tipe data *float*. Selanjutnya inisialisasi koordinat kedua pada sumbu y, inisialisasi koordinat kedua pada sumbu y di inisialisasi dengan nama variabel *y2* dengan nilai nol (0) dengan tipe data *float*. Kemudian inisialisasi koordinat pertama pada sumbu x dengan nama variabel *x1*, selanjutnya inisialisasi koordinat pertama pada sumbu y dengan nama variabel *y1*. Selanjutnya inisialisasi selisih koordinat pertama dengan kedua pada sumbu y dengan nama variabel *yt* dan inisialisasi selisih koordinat pertama dengan kooridnat kedua pada sumbu x dengan nama variabel *xt*. Selanjutnya inisialisasi jarak perpindah titik tengah objek dengan nama variabel *s*. Kemudian inisialisasi kecepatan citra dengan satuan m/s dengan nama variabel *vms* dan inisialisasi kecepatan yang telah di konversi kedalam satuan km/h dengan nama varibel *vasli*. Selanjutnya inisialisasi *timer* mulai menghitung waktu dengan nama variabel *awal*. Dan variabel *timer* untuk menghentikan waktu dengan nama variabel *akhir*, Serta inisialisasi waktu perpindahan *frame* dengan nama variabel *t* yang bernilai nol. Setiap variabel diatas di inisialisasi dengan tipe data *float*. Kode program inisialisasi secara berturut-turut adalah :

```
……
float x2 = 0;
float y2 = 0;
float x1, y1, yt, xt, s, vcitra, vasli, awal, akhir;
float t = 0;
…..
```

Langkah selanjutnya adalah insialisasi matriks untuk *frame input* pada video input. *Frame input* video diinisialisasi dengan nama matriks *src*. Selanjutnya inisialisasi matriks untuk *frame* yang telah di-*filter morphology opening* dengan nama matriks *opening*. Kemudian inisialiasi matriks untuk *frame* yang telah di-*filter morphology closing* dengan nama matriks *closing*. Selanjutnya inisialisasi matriks untuk *frame input* yang telah di *background subtraction* dengan nama matriks *bs*. Dan inisialisasi matriks untuk *frame input* yang telah di *blur* dengan nama matriks *blur1*. Inisialisasi matriks *frame* ini menggunakan tipe data *Mat* pada *library opencv* dimana citra atau *frame* pada video dikonversi kedalam sebuah bilangan *real* pada setiap baris dan kolom piksel *frame* atau citra dengan panjang baris dan kolom sesuai ukuran piksel frame. Sehingga bilangan tersebut dapat di proses. Kode program yang digunakan pada inisalisasi matriks untuk *frame* secara berturut-turut adalah :

```
……
Mat src, opening, closing, bs, blur1;
…..
```

Langkah selanjutnya adalah inisalisasi *element* pada *filter morphology*. Struktur *element* disimpan dalam tipe data *Mat*. *Mat* merupakan matriks yang disediakan *library opencv* dengan nama matriks *element*. Sturuktur *element morphology* yang digunakan adalah struktur *element rectanguler*, ukuran piksel yang akan di *filter* yaitu 5 x 5 piksel dan *filter* dimulai pada pada sumbu x = 0 dan y = 0. Kode program yang digunakan adalah :

```
……
mat element = getStructuringElement (MORP_RECT, Size(5, 5), point (0, 0));
…..
```

Langkah selanjutnya adalah inisialisasi *background subtraction*. Inisialisasi ini berfungsi menyimpan jenis metode *background subtraction* yang akan digunakan. Inisialisasi *background subtraction* di simpan dengan nama variabel *bsKNN*. Kode program yang digunakan adalah :

```
……
Ptr<BackgroundSubtractor> bsKNN;
…..
```

Langkah selanjutnya adalah inisialisasi koordinat kontur. Inisialisasi koordinat di simpan dalam sebuah variabel dengan nama *contur*. Kemudian vektor *output* kontur dengan dimensi 4x4 disimpan dengan variabel *hierarchy*. Kode program yang digunakan adalah :

```
……
vector<vector<point>> countours;
vector<Vec4i> heirarchy;
…..
```

Langkah berikutnya inisialisasi *tracking kontur*. Objek kontur di *tracking* dengan bentuk persegi panjang. Inisialisasi *tracking* kontur di simpan dengan nama *bounding_rect*. Kode program yang digunakan adalah :

```
……
Rect bounding_rect;
…..
```

Langkah selanjutnya adalah inisialisasi luas minimal pada area yang akan di *tracking*. Luas minimal area yang ditracking adalah 1300 piksel^2 dengan tipe data *integer*. Kode program yang digunakan adalah :

```
……
int min_area = 1300;
…..
```

Langkah berikutnya adalah membangun *k-nearest neighbor*. *K-nearest neighbor* dibangun dengan memanggil fungsi *createBackgroundSubtratorKNN()*. Dengan nilai *history* adalah 300, nilai *threshold* adalah 32 dan tanpa rekayasa bayangan. Pengaturan ini

di simpan kedalam variabel yang *bsKNN* yang telah di inisialisasi sebelumnya. Kode program yang digunakan adalah :

```
……
bsKNN = createBackgroundSubtratorKNN(300, 32, 0);
…..
```

Langkah berikutnya adalah inisialisasi video. Video di inisialisasi dengan nama variabel *capture*. Kemudian langkah selanjutnya adalah membaca video *input*. Video *input* dapat diambil dari kamera internal atau eksternal dari sebuah *pc* atau laptop dan dapat juga dari video yang disimpan pada media penyimpanan. Kode program beturut-turut yang digunakan adalah :

```
……
VideoCapture caputre;
Capture.open (“D:/60km.avi”);
…..
```

Langkah selanjutnya adalah pengulangan yang berfungsi untuk memproses setiap *frame* dari video *input*. Dan proses akan dihentikan apabila *user* menekan tombol *esc* dengan nilai *ASCII* 27. Kode program yang digunakan adalah :

```
……
while (waitKey(1) != 27)
{
…..
}
```

Langkah selanjutnya inisialisasi konversi data ke dalam bentuk karakter. Data hasil kalkulasi kecepatan akan di konversi ke dalam tipe data karakter. Nama variabel untuk mengkonversi hasil kalkulasi ke dalam bentuk karakter adalah *ss*. Hasil konversi data tersebut digunakan untuk menampilkan kecepatan objek yang terdeksi dalam setiap *frame*. Nama variabel yang akan menampilkan kecepatan di setiap *frame* adalah *tampilkan*. Kode program yang digunakan adalah :

```
……
stringstream ss;
string tampilkan;
…..
```

Langkah berikutnya adalah membaca setiap *frame* video *input* kedalam matriks *frame input*. Kemudian simpan informasi *frame* yang sedang diproses kedalam variabel *framesrkg* yang di inisialisasi dengan tipe data *integer*. Kemudian tampilkan *frame* yang sedang di proses dalam *command prompt*. Kode program yang digunakan secara berturut-turut adalah :

```
……
capture >> src;
int frameskrg = capture.get(CAP_PROP_POS_FRAME);
cout << "frame ke- "<< frameskrg << endl;
…..
```

Langkah berikutnya adalah memulai *timer*. *Timer* di mulai apabila *frame* akan di proses. Kemudian resolusi *frame* video akan di perkecil 5/2 lebih kecil dari *frame input*nya. Pengecilan *frame* ini dilakukan dengan memanggil fungsi *resize*. Pengecilan *frame* bertujuan untuk memperkecil waktu proses setiap *frame*. Kode program yang digunakan secara berturut-turut adalah :

```
……
awal = clock();
resize (src, src, Size (src.size().width/2.5, src.size().height/2.5));
…..
```

Langkah berikutnya mem-*blur frame input*. Tujuan dari *blur* ini untuk menghilangkan *noise* sebelum *frame input* di *threshloding*. Proses *blur* dilakukan dengan memanggil fungsi *blur()*. *frame* akan di-*blur* dengan ukuran panjang 5 piksel dan lebar 5 piksel. *Output* dari proses *blur* ini akan di simpan dalam matriks *blur1*. Kode program yang digunakan adalah :

```
……
blur (src, blur1, size ( 5, 5));
…..
```


Langkah selanjutnya adalah menggambar area deteksi di dalam *frame input*. Hal ini memudahkan pengguna dalam melihat area yang terdeteksi oleh sistem. Proses menggambar batas atau area deteksi ini dengan memanggil fungsi *line()*. Fungsi ini menggambar dari titik ke titik yang telah beri kooridnat dalam fungsi tersebut. Kode program yang digunakan adalah sebagai berikut :

```
……
ine(src, Point(91, 345), Point(106, 274), scalar (0, 255, 255), 2) ;
ine(src, Point(106, 274), Point(193, 278), scalar (0, 255, 255), 2) ;
ine(src, Point(91, 345), Point(197, 349), scalar (0, 255, 255), 2) ;
ine(src, Point(193, 278), Point(197, 349), scalar (0, 255, 255), 2) ;
…..
```

Langkah selanjutnya menerapkan *background subtraction KNN*. Penerapan *KNN* dilakukan dengan memanggil fungsi apply() pada variable yang membangun background subtraction KNN. Tujuan dari fungsi *apply()* ini untuk mendeteksi objek yang bergerak dengan nilai biner dan *frame output* berwarna hitam putih pada *frame input*. Untuk benda bergerak dinyatakan dengan warna putih, sedangkan untuk latar belakang dinyatakan dengan warna hitam. Pengaturan metode *background subtraction* yang di simpan dalam variabel *bsKNN* pada langkah sebelumnya diterapkan pada *frame* yang telah di *blur* pada langkah sebelumnya. Kemudian *output* dari proses *background subtraction* disimpan kedalam matriks dengan nama *bs*. Kode program yang digunakan adalah :

```
……
bsKNN->apply(blur1,bs);
…..
```

Langkah selanjutnya adalah *filter morphology opening* pada *frame KNN*. *Filter morphology opening* berfungsi untuk menghilangkan *noise* disekitar objek pada citra biner. Proses *filter morphology opening* dilakukan dengan memanggil fungsi *morphologyEx()* yang terdapat *syntax CV_MOP_OPEN*. *Frame* yang telah di *background subtraction KNN* di-*filter* dengan *morphology opening* kemudan di salin kedalam matriks dengan nama *opening*. Langkah berikutnya adalah *filter morphology closing* pada *frame* yang telah di-*filter morphology opening* pada langkah sebelumnya. Proses *filter* morphology dilakukan dengan memanggil fungsi *morphologyEx()*yang terdapat *syntax CV_MOP_CLOSE*. *Filter morphology closing* bertujuan untuk menghilangkan warna hitam di dalam objek yang terdeteksi atau berwarna putih. *Frame morphology opening* yang telah di *filter morphology closing* disalin kedalam matriks *closing*. Struktur elemen pada kedua *filter morphology* teresebut menggunakan struktur elemen yang telah di inisialisasi pada langkah sebelumnya. Kode program yang digunakan secara berturut-turut adalah :

```
……
morphologyEx(bs, opening, CV_MOP_OPEN, element);
morphologyEx(opening, closing, CV_MOP_CLOSE, element);
…..
```

Langkah berikutnya adalah mencari kontur pada *frame* biner yang telah di *filter morphology* pada langkah sebelumnya. Pencarian kontur dilakukan dengan memanggil fungsi *findContours()*. Setelah kontur di temukan kontur tersebut disimpan dalam variabel beranama *contours*. Kode program yang digunakan adalah :

```
……
findContours (closing, opening, hierarchy, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE):
…..
```

Langkah berikutnya mencari *moment* dari kontur yang di dapat. Mencari *moment* ini berfungsi dalam menentukan titik tengah dari sebuah kontur yang ada di setiap *frame*. Langkah pertama mencari *moment* adalah inisialisasi *moment* dengan nama variabel *momentum*. Kemudian luas kontur di simpan dalam variabel *area*. *Moment* yang di cari harus di seleksi pada kontur yang melebihi area minimal yaitu 1300 piksel^2. Selanjutnya

moment akan dicari dengan memanggil fungsi *moments* dari setiap kontur. Urutan *moment* ini disimpan berdasarkan urutan kontur setiap *frame*. Kode program yang digunakan adalah :

```
……
vector<Moments> momentum(contours.size());
for(int i =0; i<contours.size(); i++);
{
        double area = contourArea(contours[i], 0);
if(area>min_area){
momentum[i] = moments(contours[i], false);
        }
}
…..
```

Langkah selanjutnya adalah mencari titik tengah berdasarkan *moment* setiap kontur yang telah ditemukan. Pertama inisialisasi titik tengah kontur dengan nama variabel *center*. Selanjutnya simpan luas area setiap kontur dalam variabel *area*. Kemudian seleksi setiap kontur yang memiliki luas kontur lebih dari luas minimum. Kemudian kalkulasi titik tengah berdasarkan *moment* yang telah di temukan pada setiap kontur. Untuk persamaan mencari titik tengah ini telah disediakan oleh pustaka OpenCV. Langkah berikutnya simpan titik tengah setiap kontur dalam variabel *center*. Kode program yang digunakan adalah :

```
……
vector<Point2f> center(contours.size());
for(int i =0; i<contours.size(); i++);
{
        double area = contourArea(contours[i], 0);
if(area>min_area){
        center[i] = Point2f ( momentum[i]. m10/ momentum[i]. m00, momentum[i]. m01/
momentum[i]. m00;
        }
}
```

Langkah berikutnya adalah memproses setiap kontur pada setiap *frame* video *input*. Setiap luas area kontur di simpan kedalam variabel *area*. Kemudian semua kontur yang terdapat dalam *frame* di seleksi dengan luas area kontur minimal 1300 piksel^2 dan melewati batas deteksi dengan sumbu x antara 97 hingga 197 dan sumbu y antara 274 hingga 349. Kode program yang digunakan adalah :

```
……
for(int i =0; i<contours.size(); i++);
{
        double area = contourArea(contours[i], 0);
if(area>min_area && center[i].y > 274 && center[i].y < 349 && center[i].x > 97 &&
cenrer[i].x < 197){
…..
```

Selanjutnya kontur yang memenuhi seleksi akan di simpan ke dalam variabel *bounding_rect* untuk di-*tracking* atau di-*bounding box*. Selanjutnya sumbu x pada kontur yang memenuhi seleksi diatas akan di simpan pada variabel *x1*. Sedangkan sumbu y pada kontur yang memenuhi seleksi diatas akan disimpan dalam variabel *y1*. Kode program yang digunakan secara berturut-turut adalah :

```
……
        bounding_rect = boundingRect(counturs[i]);
        x1= center[i].x;
        y1= center[i].y;
…..
```

Selanjutnya menghentikan *timer*. *Timer* dihentikan dan disimpan kedalam variabel *akhir*. Kemudian waktu perpindahan ini akan dihitung dengan satuan detik. Kode program yang digunakan adalah :

```
……
        akhir = clock();
        t = (akhir – awal)/CLOCK_PER_SEC;
…..
```

Selanjutnya melakukan perhitungan kecepatan. Pada langkah ini titik tengah kedua pada sumbu x dan sumbu y ini akan diseleksi. Karna pada saat objek yang pertama kali melewati batas deteksi tidak memiliki titik tengah kedua pada sumbu x dan sumbu y sehingga proses menghitung kecepatan di lewati. Kemudian langkah selanjutnya menghitung selisih kedua koordinat pada sumbu x dan sumbu y. Langkah selanjutnya mencari jarak perpindahan titik tengah sesuai dengan persamaan 2.1. Kode program yang digunakan secara berturut-turut adalah :

```
……
        if(x2!=0|| y2!=0)
{
        yt = y2 - y1;
        xt = x2 – x1;
        s = sqrt(pow(xt, 2) + pow(yt, 2));
…..
```

Langkah selanjutnya menghitung kecepatan kendaraan. Persamaan yang digunakan adalah persamaan 3.1. Langkah selanjutnya menentukan panjang lintasan pada citra atau *frame*. Berdasarkan seleksi kontur pada langkah sebelumnya, setiap kontur bergerak dari

sumbu y = 274 hingga y = 349. Sehingga panjang lintasan pada citra yang didapat adalah 75 piksel. Sedangkan panjang lintasan sebenarnya yang telah di ukur pada langkah sebelumnya adalah 2.98 meter. Setelah kecepatan didapat dalam satuan *m/s* , kecepatan ini dikonversi ke dalam satuan *km/h*. Langkah selanjutnya adalah menyimpan kecepatan yang sudah dihitung ke dalam variabel *ss* untuk di konversi dalam bentuk tipe data karakter. Langkah menyimpan hasil ke variabel *ss* ini merupakan langkah terakhir dalam menghitung kecepatan. Kode program yang digunakan secara berturut-turut adalah :

```
……
        vms =( s *2.98) / (t*69.95);
        vasli = vms / mstokmh;
        ss << vasli << km/h;
}
…..
```

Langkah selanjutnya adalah mengkonversi data yang disimpan dalam variabel *ss* menjadi bentuk tipe data karakter yaitu string. Kemudian langkah selanjutnya adalah menampilkan nilai koordinat pertama pada sumbu x dan y, menampilkan nilai koordinat kedua pada sumbu x dan y, menampilkan jarak perpindahan kontur, waktu proses *frame* dan kecepatan kendaraan atau objek yang terdeteksi dalam *command prompt*. Hali ini berguna untuk memudahkan proses analisa sistem deteksi kendaraan. Kode program yang digunakan secara berturut-turut adalah :

```
……
tampilkan = ss.str();
cout << " s = " << s <<"\t v= " << v << vasl <<"\t t = " << t << endl;
cout <<" x1 = " << x1 << "\t x2 = " << endl;
cout <<" y1 = " << y1 <<"\t y1 = " << endl;
…..
```

Langkah selanjutnya adalah menyalin nilai titik tengah pertama ke dalam variabel titik tengah kedua pada sumbu x dan sumbu y sebelum pergantian *frame*. Sebelum pergantian *frame* titik tengah kedua pada sumbu y di seleksi apabila akan melewati akhir

dari batas deteksi kecepatan, maka nilai titik tengah kedua pada sumbu y akan di jadikan nol. Hal ini berguna untuk mengurangi besar *error* apabila ada kontur selanjutnya yang melewati batas deteksi. Kode program yang digunakan adalah :

```
……
        x2 = x1;
        y2 = y1;
        if (y2 >= 317)
        {
                y2 = 0;
                x2 = 0;
        }
…..
```

Langkah selanjutnya adalah menampilkan kecepatan kontur yang terdeteksi pada *frame input*. Kecepatan yang telah dihitung kemudian dikonversi ke bentuk karakter sehingga dapat ditampilkan di-*frame input*. Proses menampilkan kecepatan pada *frame input* dilakukan dengan memanggil fungsi *putText()*. Posisi kecepatan yang ditampilkan terlektak di koordinat kontur yang terdekat dengan titik awal atau nol. Kode program yang digunakan adalah :

```
……
        putText (src, tampilkan.c_str(), cv::Point(bounding_rect.x, bounding_rect.y),
FONT_HERSHEY_SIMPLEX, 0.75, cv::Scalar(0, 0, 0) , 2);
        }
…..
```

Langkah selanjutnya adalah memberi *bounding box* dalam bentuk persegi panjang. Pemberian *bounding box* atau *tracking object* dilakukan dengan memanggil fungsi *rectangle()*. Kontur yang telah memenuhi seleksi pada proses sebelumnya akan diberi tanda berbentuk persegi panjang berwarna hijau yang menandakan bahwa objek atau kendaraan yang terdeteksi. Kode program yang digunakan adalah sebagai berikut :

```
……
        rectangle (src, bounding_rect, Scalar (0, 255, 0), 2, 8, 0) ;
        }
…..
```

Langkah selanjutnya adalah menampilkan *frame input*. *Frame input* di-*check*, apakah *frame input* dalam video masih ada atau tidak. Kemudian *frame input* yang telah diproses pada tahap sebelumnya ditampilan dengan nama jendela "frame". Kode program yang digunakan adalah :

```
……
        if(!src.empty()){
                imshow("frame", src);
                }
        }
        return 0;
        }
…..
```

3.4 Menentukan parameter-parameter yang dianalisa

Tahap selanjutnya adalah menetukan parameter-parameter yang dianalisa. Pada tahap ini parameter-parameter yang akan dianalisa akan ditentukan berdasarkan tujuan dari penelitian pada bab sebelumnya yaitu membandingkan kecepatan yang terdeteksi pada program dengan menggunakan metode *MOG2* dan metode *KNN* dengan kecepatan sebenarnya untuk mendapatkan akurasi dalam bentuk persenatase *error*. Parameter yang dianalisa adalah kecepatan kendaran yang terdeteksi, waktu perpindahan *frame*, dan perpindahan titik tengah kontur.

3.5 Pengujian Metode

Tahap terakhir adalah melakukan pengujian metode *mixture of gaussian 2* dan *k-nearest neighbor* pada program yang telah dirancang dan dibangun. Program tersebut diuji dengan empat video yang telah direkam pada tahap sebelumnya. Pengujian dilakukan sebanyak 10-12 kali pada setiap video menggunakan program dengan metode *MOG2* dan *KNN*. Terdapat empat video dengan skenario kecepatan kendaraan yang diuji yaitu 20

km/jam, 40 km/jam, 50 km/jam, 60 km/jam. Spesifikasi video yang diuji pada pernelitian ini dapat dilihat pada tabel 3.1.

Tabel 3.1 Spesifikasi video yang diuji.

No	Nama File	Skenario kecepatan kendaraan	Resolusi (width x Height)	Durasi	Ukuran
1	20km.avi	20 Km/jam	720 x 1280	4 detik	5.158 KB
2	40km.avi	40 Km/jam	720 x 1280	7 detik	9.173 KB
3	50km.avi	50 Km/jam	720 x 1280	7 detik	9.116 KB
4	60km.avi	60 Km/jam	720 x 1280	10 detik	12.925 KB

Pengujian kedua program deteksi kecepatan kendaraan dilakukan menggunakan komputer untuk mendapat kecepatan kendaraan yang terdeteksi sistem. Adapun spesifikasi komputer yang digunakan dapat dilihat pada tabel 3.2.

Tabel 3.2 Spesifikasi komputer yang digunakan.

No	Pheriperal	Keterangan
1	Memori	2048 MB RAM.
2	Prosesor	Intel(R) core(TM) 2 duo CPU.
3	Sistem operasi	Windows 7 ultimate 32 bit.
4	Kompiler	Microsoft Visual Studio express 2012
5	Pustaka (*library*)	OpenCV 3.0.0