BAB II
LANDASAN TEORI

2.1 Pengolahan Citra

Secara umum, pengolahan citra digital menunjukkan pada pemrosesan gambar 2 dimensi menggunakan komputer.Dalam konteks yang lebih luas, pengolahan citra mengacu pada pemrosesan setiap data 2 dimensi. Citra merpakan sebuah larik *(array)* yang nilai-nilai real maupun komplek yang direpresentasikan dengan deretan bit tertentu (dharma putra, 2010). Citra atau *image* adalahistilah lain untuk gambar sebagai salah satu komponen multimedia yang memegang peranan sangat penting sebagai bentuk informasi visual. Citra mempunyai karakteristik yang tidak dimiliki oleh teks, yaitu citra kaya akan informasi.

Citra atau gambar merupakan salah satu komponen dalam deteksi tepi yang bersifat 2D. Teknologi dasar untuk menciptakan dan menampilkan warna pada citra digital berdasarkan pada penelitian bahwa sebuah warna merupakan kombinasi dari tiga warna dasar, yaitu merah, hijau, dan biru (*Red, Green,Blue* - RGB).

Dalam pemrosesan citra terdapat sautu proses dengan menggunakan deteksi tepi, yang mana proses ini ditujukan untuk mendapatkan batas garis pada suatu tepian citra mereprentasikan objek-objek yang terkandung dalam citra tersebut, baik bentuk, ukurannya dan informasi tentang teksturnya (Putra Dedi, 2010).

Ada beberapa metode yang sering digunakan dalam deteksi tepi, seperti operator Robert, operator Prewitt, operator Sobel dan operator *Canny.*Deteksi tepi *Canny* merupakan operator deteksi tepi yang menggunakan beberapa tahap algoritma untuk mendeteksi tepian dalam gambar.Deteksi tepi *Canny* dikembangkan oleh John F. Canny pada tahun 1986. Canny juga menghasilkan teori komputasi deteksi tepiyang menjelaskan bagai mana teknik ini bekerja.Tujuan dariJ FC adalah untuk mengembangkan algoritma yang optimal,ada beberapa kireteria dari metode *Canny* adalah sebagai berikut (Edi Winarno, 2011):

a. Mendeteksi dengan baik (kriteria deteksi)

   Kemampuan untuk meletakkan dan menandai semua tepi yang ada sesuai dengan pemilihan parameter-parameter konvolusi yang dilakukan.Juga memberikan

fleksibilitas yang sangat tinggi dalam hal menentukan tingkat deteksi ketebalan tepi sesuai yang diinginkan.

b. Melokalisasi dengan baik (kriteria lokalisasi)

Metode *Canny* menghasilkan jarak yang minimum antara tepi yang dideteksi dengan tepi yang asli.

c. Respon yang jelas (kriteria respon)

Hanya ada satu respon untuk tiap tepi.Mudah dideteksi dan tidak menimbulkan kerancuan pada pengolahan citra selanjutnya.

Dalam deteksi tepi dengan menggunakan metode *Canny* ada beberapa tahap untuk menemukan tepian yang optimal adalah sebagai berikut (Canny, 2009).

1. *Noise reduction*

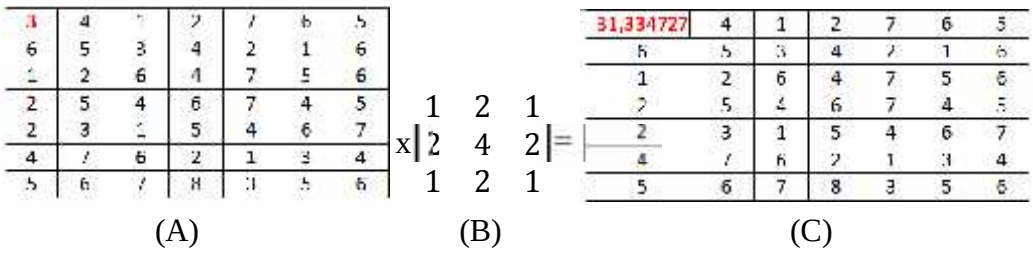
Dilakukannya *Noise reduction* karena pada deteksi tepi *Canny* rentan terhadap *noise*, penghilangan *noise* ini menggunakan *Gaussian* filterkurva, di mana citra *input* dikonvolusi dengan *Gaussian* filter.Hasil dari *Gaussian* filter ini citra akan tampak kabur namun pada citra tidak terdapat *noise* lagi.Berikut adalah contoh dari *Gaussian* filter3x3, digunakan untuk membuat gambar ke kanan, denganσ= 0,5 (Masoud Nosrati, dkk, 2013).

$$\frac{1}{16}\begin{vmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{vmatrix} \qquad 2.1$$

Untuk menyelesaikan konvolusi *gaussian* filter menggunakan rumus sebagai berikut:

$$sum = sum + coeffs[i+3] * temp.at<uchar>(y, x1) \qquad 2.2$$

Pada gambar 2.1 diperlihatkan proses konvolusi yang dilakukan oleh *Gaussian*, konvolusi dilakukan dengan caramencari nilai piksel pada piksel pertama kemudian hasil dari konvolusi disimpan pada matrik baru.



Gambar 2.1 A. Gambar *grayscale* B. Kernel *gaussian* C. Hasil konvolusi

Nilai 31,334727 pada citra hasil konvolusi diperoleh dengan perhitungan berikut.

Sum_1 = 0 + (0,00443x3) =0,013299

Sum_2 = 0,013299+(0,054006x3) =0,201915

Sum_3 = 0,201915+(0,242036x3) =1,331853

Sum_4 = 1,331853+(0,399043x3) =5,192688

Sum_5= 5,192688+(0,242036x3) =16,304172

Sum_6= 16,304172+(0,054006x3) =49,074534

Sum_7= 49,074534+(0,00443x3) =147,224631

$$\frac{0,013299+0,201915+1,331853+5,192688+16,304172+49,074534+147,224631}{7}=31,334727$$

Dengan demikian nilai 3 pada citra *grayscale* menjadi 31,334727 setelah dikonvolusi menggunakan *gaussian* dengan menggunakan kernel 3x3.

2. *Finding the intensity gradient of the image*

   Sebuah tepi dalam foto dapat menunjukkan dalam berbagai arah, sehingga algoritma *Canny* menggunakan empat filter untuk mendeteksi horisontal, vertikal dan diagonal tepi dalam gambar kabur. Seperti deteksi tepi (Roberts, Prewitt, Sobel) mengembalikan nilai turunan pertama dalam arah horizontal (G_x) dan arah vertikal (G_y).Dari gradien dan arah tepi dapat ditentukan (Masoud Nosrati, dkk, 2013).Dalam penelitian ini pendeteksi horisontal, vertikal dan diagonal tepi dalam gambar kabur menggunaka persaman yang digunakan dalam metode sobel.Operator Sobel terdiri dari matriks 3x3 masing-masing adalah $G_x$dan G_y. Matriks*mask* tersebut dirancang untuk memberikan respon secara maksimal terhadap tepi objek baik horizontal maupun vertikal. *Mask* dapat diaplikasikan secara terpisah terhadap *input* citra. Operator Sobel menggunakan kernel operator gradien 3 x 3, dengan koefisien yang telah ditentukan.$G_x$dan$G_y$dapat dinyatakan sebagai berikut (Munir, 2004):

$$G_x = \begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{matrix} \text{ dan } G_y = \begin{matrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{matrix} \quad (2.3)$$

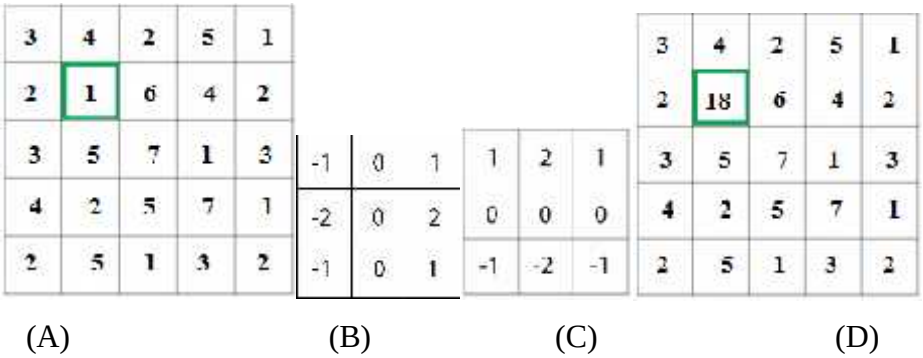
Kernel di atas dirancang untuk menyelesaikan permasalahan deteksi tepi baik secara vertikal maupun horizontal. Penggunaan kernel-kernel ini dapat digunakan bersamaan ataupun secara tepisah (Purnomo dan Muntasa, 2010). Untuk mendapatkan nilai maksimum dari operator Sobel, proses selanjutnya adalah dengan menghitung kekuatan tepi citra terhadap warna kecerahannya dengan cara mencari nilai*magnitude* yang dapat dihitung dengan persamaan sebagai berikut (Munir 2004):

$$M = \sqrt{G_x^2 + G_y^2} \quad (2.4)$$

Karena menghitung akar adalah persoalan rumit dan menghasilkan nilai real, maka dalam mencari kekuatan tepi*(magnitude)* dapat disederhanakan perhitungannya. Besarnya *magnitude* gradien dapat dihitung lebih cepat lagi dengan menggunakan persamaan sebagai berikut (Munir, 2004):

$$M = |G_x| + G_y \quad (2.5)$$

Pada Gambar 2.1 diperlihatkan deteksi tepi dengan operator Sobel. Operasi konvolusi bekerja dengan menggeser kernel piksel per piksel, yang hasilnya kemudian disimpan dalam matriks baru. Konvolusi pertama dilakukan terhadap piksel yang bernilai 1 (di titik pusat *mask*) (Munir, 2004).



Gambar 2.2. (A) Citra *grayscale*, (B) G_x, (C) G_y, (D) Hasil konvolusi

Nilai 18 pada citra hasil konvolusi diperoleh dengan perhitungan berikut (Munir, 2004):

$G_x = 3\ -1\ +\ 2\ -2\ +\ 3\ -1\ +\ 2\ 1\ +\ 6\ 2\ +\ 7\ (1) = 11$

$G_y = (3)\ 1\ +\ 4\ 2\ +\ 2\ 1\ +\ 3\ -1\ +\ 5\ -2\ +\ 7\ (-1) = -7$

$$M = \sqrt{G_x^2 + G_y^2} = \sqrt{(11)^2 + (-7)^2} \cong |G_x| + |G_y| = |11| + |-7| = 18$$

Dengan demikian,nilai 1 diubah menjadi nilai 18 pada citra keluaran.

3. *Non-maximum*

   Mengingat perkiraan gradien gambar, pencarian dilakukan untuk menentukan apakah besarnya gradien mengasumsikan maksimum lokal dalam arah gradien.Dalam beberapa implementasi, algoritma mengkategorikan arah gradien kontinu ke dalam satu set kecil arah diskrit, dan kemudian bergerak atas filter 3x3 output dari langkah sebelumnya (yaitu, kekuatan tepi dan arah gradien).Pada setiap pixel, ia menekan kekuatan tepi pixel pusat (dengan menetapkan nilainya ke 0) jika besarnya tidak lebih besar dari besarnya dua tetangga dalam arah gradien.Sebagai contoh(Masoud Nosrati, dkk, 2013).

   a. jika sudut gradien bulat adalah nol derajat (yaitu tepi berada di arah utara-selatan) titik akan dianggap berada di tepi jika gradien besarnya lebih besar dari besaran pada piksel di arah timur dan barat.
   b. jika sudut gradien bulat adalah 90 derajat (yaitu tepi berada dalam arah timur-barat) titik akan dianggap berada di tepi jika gradien besarnya lebih besar dari besaran pada piksel di bagian arah utara dan selatan,
   c. jika sudut gradien bulat adalah 135 derajat (yaitu tepi berada dalam arah timur laut-barat daya) titik akan dianggap berada di tepi jika gradien besarnya lebih besar dari besaran pada piksel dalam arah timur utara barat dan selatan,
   d. jika sudut gradien bulat adalah 45 derajat (yaitu tepi adalah ke arah barat-selatan timur utara) titik akan dianggap berada di tepi jika gradien besarnya lebih besar dari besaran pada piksel di barat utara timur dan selatan arah.

   Dalam implementasi yang lebih akurat, interpolasi linear digunakan antara dua piksel tetangga yang mengangkang arah gradien. Sebagai contoh, jika sudut gradien adalah antara 45 derajat dan 90 derajat interpolasi antara gradien di utara dan timur utara piksel akan memberikan satu nilai interpolasi, dan interpolasi antara selatan dan barat selatan piksel akan memberikan yang lain (menggunakan konvensi paragraf terakhir). Gradien besarnya pada pixel pusat harus lebih besar dari kedua

hal tersebut untuk itu untuk ditandai sebagai tepi.Perhatikan bahwa tanda arah tidak relevan, yaitu utara-selatan adalah sama selatan-utara dan seterusnya(Masoud Nosrati, dkk, 2013).

4. *Tracing edges through the image and hysteresis thresholding*

   Gradien intensitas besar mungkin lebih sesuai untuk dengan tepi dari gradien intensitas kecil.Hal ini tidak banyak dalam kasus untuk menentukan ambang batas di mana diberikan intensitas *switch* gradien dari sesuai dengan tepi menjadi tidak melakukannya.Oleh karena itu *Canny* menggunakan *thresholding* dengan *hysteresis*.*Thresholding* dengan *hysteresis* membutuhkan dua ambang yaitu tinggi dan rendah.Membuat asumsi bahwa tepi penting harus sepanjang kurva kontinu dalam gambar memungkinkan kita untuk mengikuti bagian samar garis tertentu dan untuk membuang piksel bising beberapa yang tidak merupakan garis tapi telah menghasilkan gradien besar.Oleh karena itu mulai dengan menerapkan ambang batas tinggi.Ini menandai keluar tepi dapat cukup yakin yang asli.Mulai dari ini, dengan menggunakan informasi yang diperoleh sebelumnya directional, tepi dapat ditelusuri melalui gambar. Sementara menelusuri tepi, menerapkan batas bawah, memungkinkan untuk melacak bagian samar tepi selama menemukan titik awal.Setelah proses ini selesai citra biner memilki setiap pixel ditandai baik sebagai pixel tepi atau pixel non-tepi.Dari output komplementer darilangkah tepi tracing, peta tepi biner diperoleh dengan cara ini juga dapat diperlakukan sebagai satu set kurva tepi, yang setelah diproses lebih lanjut dapat direpresentasikan sebagai poligon dalam domain citra (Masoud Nosrati, dkk, 2013).

5. *geometric formulation of the Canny edge detector*

   Tepiyang Sebuah pendekatan yang lebih halus untuk mendapatkan tepi dengan akurasi sub-pixel adalah dengan menggunakan pendekatan deteksi tepi diferensial, di mana kebutuhan penindasan non-maksimum diformulasikan dalam bentuk derivatif kedua dan ketiga-order dihitung dari ruang skala (Masoud Nosrati, dkk, 2013)..

6. *Variational-geometric formulation of the Haralick-Canny edge detector*

   Penjelasan variasional untuk bahan utama dari detektor tepi *Canny*, yaitu, menemukan nol penyeberangan dari turunan ke-2 sepanjang arah gradien, terbukti merupakan hasil dari sebuah meminimalkan *Kronrod Minkowski* fungsional

sekaligus memaksimalkan integral atas penyelarasan tepi dengan bidang gradien (Masoud Nosrati, dkk, 2013)..

Algoritma *Canny* berisi sejumlah parameter yang dapat disesuaikan, yang dapat mempengaruhi perhitungan waktu dan efektivitas algoritma (Masoud Nosrati, dkk, 2013)..

a. Ukuran dari *Gaussian* filter: filter *smoothing* digunakan pada tahap pertama langsung mempengaruhi hasil dari algoritma *Canny*.Filter yang lebih kecil menyebabkan kurang kabur, dan kecil kemungkinan terdeteksi, garis-garis tajam.Sebuah filter yang lebih besar menyebabkan lebih kabur, mengolesi keluar nilai pixel yang diberikan di wilayah yang lebih besar dari gambar.Jari-jari kabur lebih besar lebih berguna untuk mendeteksi besar, halus tepi - misalnya, tepi pelangi.
b. Ambang batas: penggunaan dua ambang batas dengan *hysteresis* memungkinkan fleksibilitas lebih dalam pendekatan single-threshold, tapi masalah umum pendekatan *thresholding* masih berlaku.Sebuah ambang batas yang ditetapkan terlalu tinggi dapat kehilangan informasi penting.Di sisi lain, sebuah ambang batas yang ditetapkan terlalu rendah akan mengidentifikasi informasi palsu yang tidak relevan (seperti *noise*) sama pentingnya.Sulit untuk memberikan batas generik yang bekerja dengan baik pada semua gambar.Belum ada pendekatan tentang pengujian masalah ini.

2.2*Computer Vision*

Compuer vision merupakan salah satu teknologi yang sangat erat kaitannya dengan pengolahan citra. *computer vision* sebenarnya mengikuti cara kerja sistem visual manusia *(human vision).* Sesungguhnya *human vision* sangat kompleks dalam melihat objek dengan indera penglihatan, kemudian objek yang dilihat diteruskan ke otak untuk diinterprasi sehingga manusia mengerti objek apa yang telah terlihat dalam pandangannya.

Sedangkan dalam *computer vision* sendiri dalam pengenalan banyak sekali para ilmuan yang mengartikan tentang *computer vision* tersebut. Seperti Ballard dan Brown, Boyle dan Tomas, Gonzalez dan Wintz. Ballard dan Browen mendefenisikan *computer vision* sebagai suatu kegiatan awal pengotomatisan suatu pemrosesan dan reseprentasi sebagai suatu persepsi visual dengan tahap-tahap tertentu (Fadlisyah, 2007).

Boyle dan Thomas memberi pengertian terhadap *computer vision* bahwa *computer vision* lebih dari hanya sekedar *image recognition.* Namun *computer vision* juga menghadirkan operasi *low level processing* sebagai suatu algoritma pengolahan citra yang dapat disebut *purely* yang kemudian mengategorikan citra tersebut kedalam *computer vision* (Fadlisyah, 2007).

Dari uraian tentang pengertian *computer vision* tersebut, maka dapat diambil suatu kesimpulan bahwa *computer vision* dapat didefenisikan sama dengan pengolahan citra yang dikaitkan dengan akuisis citra, pemrosesan, klafikasi, pengakuan, dan pencakupan keseluruhan, pengambilan keputusan yang diikuti dengan pengidentifikasian citra (Fadlisyah, 2007).

Beberapa deskripsi yang berkaitan dengan *computer vision* telah memaparkan seberapa jauh kemampuan visi komputer mendeteksi kemampuan visi manusia. Ada ada 4 langkah besar dalam *computer vision* yang meniru penglihatan manusia, yaitu akusisi citra, pengolahan citra, analisis citra, dan pemahaman citra (Suparman dan Marlan, 2007).

Ada beberapa perangkat lunak yang digunakan untuk visi komputer diantaranya Aforge.Net, VXL dan OpenCV. OpenCV adalah singkatan dari *Open Computer Vision*, yaitu *library open source* yang dikhususkan untuk melakukan pengolahan citra. *Library* ini ditulis dengan bahasa C dan C++, serta dapat dijalankan dengan Linux, Windows, dan Mac OS X. OpenCV dirancang untuk efisiensi komputasional dan dengan fokus yang kuat pada aplikasi *real-time*. Tujuannya adalah agar komputer mempunyai kemampuan yang mirip dengan cara pengolahan visual pada manusia. OpenCVmemiliki API (*Application Programming Interface*) untuk pengolahan tingkat tinggi maupun tingkat rendah. Pada OpenCV juga terdapat fungsi-fungsi siap pakai untuk me-*load*,menyimpan, serta mengakuisisi gambar dan video (Bradski dan Kaehler, 2008).

Tujuan OpenCV adalah untuk menyediakan infrastruktur visi komputer yang mudah digunakan yang membantu orang-orang dalam membangun aplikasi - aplikasi visi yang mutakhirdengan cepat. *Library* pada OpenCV berisi lebih dari 500 fungsi yang menjangkau berbagai area dalam permasalahan visi, meliputi inspeksi produk pabrik, pencitraan medis, keamanan, antarmuka pengguna, kalibrasi kamera, visi stereo, dan robotika (Bradski dan Kaehler, 2008).

Lisensi *open source* pada OpenCV telah distrukturisasi sehingga pengguna dapat membangun produk komersial menggunakan seluruh bagian pada OpenCV. OpenCV telah digunakan pada banyak aplikasi, produk, dan usaha-usaha penelitian. Aplikasi-aplikasi ini

meliputi penggabungan citra pada peta web dan satelit, *image scan alignment*, pengurangan *noise* pada citra medis, sistem keamanan dan pendeteksian gangguan, sistem pengawasan otomatis dan keamanan, sistem inspeksi pabrik, kalibrasi kamera, aplikasi militer, serta kendaraan udara tak berawak, kendaraan darat, dan kendaraan bawah air. OpenCV juga telah digunakan untuk pengenalan suara dan musik, dimana teknik pengenalan visi diaplikasikan pada citra spektogram suara (Bradski dan Kaehler, 2008).

Libraries OpenCV menyediakan banyak algoritma visi komputer dasar, dengan keuntungan bahwa fungsi-fungsi tersebut telah diuji dengan baik dan digunakan oleh para peneliti di seluruh dunia. *Libraries* OpenCV juga menyediakan sebuah modul untuk pendeteksian tepi objek yang menggunakan algoritma metode *Canny*.

Mat adalah salah satu *static method* dari *class additional core*. *Class mat* mewakili *array* 2D (dua dimensi) atau *array* multi-dimensi. *Mat* berfungsi untuk menyimpan matriks, gambar, histogram, fitur deskriptor, *voxel volume*, dan lain - lain. *Class mat* memiliki sintaks sebagai berikut (OpenCV, 2013):

```
Mat(int rows, int cols, int type)
```

Keterangan sintaks di atas dapat dilihat pada Tabel 2.1.

Tabel 2.1. Parameter *mat*

Parameter	Keterangan
Rows	Banyaknya baris dalam *array* 2D
Cols	Banyaknya kolom dalam *array* 2D
Type	Tipe *array*

Imread adalah salah satu*static method* dari *classadditional highgui*. *Imread* berfungsi untuk me-*load* citra dari *file* yang telah ditentukan dan mengembalikannya. Jika citra tidak dapat di-*load*, hal itu terjadi karena *file* yang hilang, perizinan yang tidak tepat, dan format yang didukung tidak valid. Imread memiliki sintaks sebagai berikut (OpenCV, 2013):

```
Mat imread(const string& filename, int flags )
```

Keterangan sintaks di atas dapat dilihat pada Tabel 2.2.

Tabel 2.2. Parameter *imread*

Parameter	Keterangan
filename	Nama *file* yang akan di-*load*
Flags	Menentukan jenis warna dari citra yang di-*load* (optional)

Imwrite adalah salah satu*static method* dari *class additional highgui*.*Imwrite* berfungsi untuk menyimpan citra ke *file* yang ditentukan. Format citra dipilih berdasarkan nama ekstensi *file* yang juga ditentukan. *Imwrite* memiliki sintaks sebagai berikut (OpenCV, 2013):

```
bool imwrite(const string& filename, Input img)
```

Keterangan sintaks di atas dapat dilihat pada Tabel 2.3.

Tabel 2.3. Parameter *imwrite*

Parameter	Keterangan
filename	Nama *file* yang akan disimpan
Input img	Citra yang akan disimpan

CvtColor adalah*static method* dari *class additional imgproc*. *Cvtcolor* berfungsi untuk mengkonversi citra dari satu ruang warna ke yang lain. *Cvtcolor* memiliki sintaks sebagai berikut (OpenCV, 2013):

```
void cvtColor ( Input src , Output dst , int code)
```

Keterangan sintaks di atas dapat dilihat pada Tabel 2.4.

Tabel 2.4. Parameter *cvtcolor*

Parameter	Keterangan
Src	Citra *input*
Dst	*Output* citra dengan ukuran yang sama dengan citra *input*
Code	Kode konversi ruang warna

Code pada sintaks di atas adalah parameter tambahan yang menunjukkan jenis transformasi yang akan dilakukan. Dalam hal ini menggunakan CV_BGR2GRAY yang berfungsi untuk mengubah citra dari BGR ke format *Grayscale*. OpenCV memiliki fungsi yang sangat bagus untuk melakukan hal ini, sintaks yang digunakan adalah (OpenCV, 2013):

```
cvtColor (src, gray_image, CV_BGR2GRAY);
```

gaussianBlurr adalah fungsi dari OpenCV yang bertugas untuk menghaluskan dan menghilangkan *noise*. *gaussianBlurr* memiliki sintaks sebagai berikut (OpenCV, 2013):

```
Void GaussianBlur(InputArray src, OutputArray dst, Size ksize, double sigmaX,
double sigmaY=0, int borderType=BORDER_DEFAULT )
```

Keterangan sintaks di atas dapat dilihat pada Tabel 2.5.

Tabel 2.5. Parameter *GaussianBlurr*.

Parameter	Keterangan
Src	Ganbar *input*
Dst	*Output* citra dengan ukuran yang sama dengan citra *input*
Ksize	Ukuran kernel *Gaussian*
sigmaX	GaussiankernelstandardeviasiarahX.
sigmaY	GaussiankernelstandardeviasiarahY
borderType	metode ekstrapolasipixel

NamedWindow berfungsi untuk membuat tampilan *window highgui*. *NamedWindow* memiliki sintaks sebagai berikut (OpenCV, 2013):

```
void namedWindow(const string& winname, int flags=WINDOW_AUTOSIZE )
```

Keterangan sintaks di atas dapat dilihat pada Tabel 2.6.

Tabel 2.6. Parameter*NamedWindow*.

Parameter	Keteranagn
Name	Namajendeladi jendelacaptionyang dapat digunakansebagaiidentifierwindow
Window-Normal	Jika ini diset, pengguna dapatmengubah ukuran jendela(tidak adakendala).

Imshow adalah perintah dari OpenCV untuk menampilkan citra asli, deteksi tepi, *grayscale.Imshow* memiliki sintaks sebagai berikut (OpenCV, 2013):

```
void imshow(const string& winname, InputArray mat)
```

Keterangan sintaks di atas dapat dilihat pada Tabel 2.7.

Tabel 2.7. Parameter*NamedWindow*.

Parameter	Keterangan
Winname	Nama dari *window*

createdTrackbar adalah perintah untuk menciptakan trackbar dan menempel kejendela yang ditentukan.*createdTrackbar* memiliki sintaks sebagai berikut (OpenCV, 2013):

```
int createTrackbar(const string& trackbarname, const string& winname, int*
value, int count, TrackbarCallback onChange=0, void* userdata=0)
```

Keterangan sintaks di atas dapat dilihat pada Tabel 2.8.

Tabel 2.8. Parameter*NamedWindow.*

Parameter	Keterangan
Trackbarname	Nama dari *trackbar*
Winname	Namadarijendelayang akan digunakansebagaiinduk daritrackbardibuat.
Value	PointerOpsionalvariabel integeryang nilainyamencerminkanposisislider. Setelahpenciptaan, posisisliderdidefinisikan olehvariabel ini.
Count	PosisiMaksimalslider. Posisiminimalselalu0
onChange	Pointerke fungsiuntuk dipanggilsetiap kalisliderperubahanposisi

Canny adalah perintah untuk menemukan tepi dalam gambar menggunakan algoritma *Canny*. *Canny* memiliki sintaks sebagai berikut (OpenCV, 2013):

```
void Canny(InputArray image, OutputArray edges, double threshold1, double
threshold2, int apertureSize= 3, bool L2gradient=false )
```

Keterangan sintaks di atas dapat dilihat pada Tabel 2.9.

Tabel 2.9. Parameter*Canny.*

Parameter	Keterangan
Image	tunggal-channel input gambar8-bit.
Edges	Peta keluaran tepi, memiliki ukuran dan jenis yang sama seperti gambar.
Threshold1	Ambang pertama untuk prosedur hysteresis
Threshold2	ambang bataskedua untuk prosedur histeresis.
apertureSize	Ukuran aperture untuk operator sobel.

findContours adalah perintah dari OpenCV untuk menemukan kontur dalam citra biner. *findContours* memiliki sintaks sebagai berikut (OpenCV, 2013):

void findContours(InputOutputArray image, OutputArrayOfArrays contours, OutputArray hierarchy, int mode, int method, Point offset=Point())

Keterangan sintaks di atas dapat dilihat pada Tabel 2.9.

Tabel 2.10. Parameter*findContours.*

Parameter	Keterangan
Image	Source,8-bittunggal-channelgambar
Countours	Terdeteksikontur. Setiapkonturdisimpansebagaivektorpoin
Hierarchy	Vektor *output* yangopsional, yang berisi informasi tentangtopologigambar
Mode	Moduspengambilankontur
CV_RETR_TREE	mengambilsemuakonturdanmerekonstruksihirarkipenuhbersarangkontur.
CV_CHAIN_APPROX_NONE	tokobenar-benarsemua titikkontur

drawContours adalah perintah menarik garis kontur atau kontur diisi. *drawContours* memiliki sintaks sebagai berikut (OpenCV, 2013):

void **drawContours**(InputOutputArray image, InputArrayOfArrays contours, int contourIdx, const Scalar& color, int thickness=1, int lineType=8, InputArray hierarchy=noArray(), int maxLevel=INT_MAX, Point offset=Point())

Keterangan sintaks di atas dapat dilihat pada Tabel 2.9.

Tabel 2.11. Parameter*drawContours.*

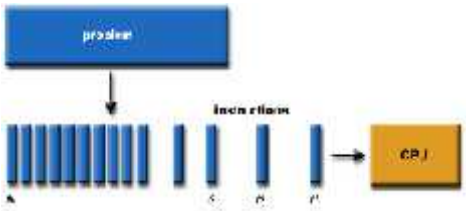
Parameter	Keterangan
Image	Gambar tujuan
Contours	Masukkan semua *contours*
Controldx	Parameter yang menunjukkan untuk menggambar *contours*.
Color	Warna dari *contours*
Thickness	Ketebalan gariskontur yang akan diambil.
lineType	Konektivitas garis
hierarchy	Informasi Opsional tentang hirarki.
Maxlevel	Tingkat maksimal untuk pengaturan kontur.
Offset	Parameter pergeseran kontur opsional

2.3 Pemrosesan Paralel

Pemrograman paralel adalah komputasi dua atau lebih *task* pada waktu bersamaan dengan tujuan untuk mempersingkat waktu penyelesaian *tasks* tersebut dengan cara mengoptimalkan *resource* pada sistem komputer yang ada untuk mencapai tujuan yang sama. Pemrosesan paralel dapat mempersingkat waktu eksekusi suatu program dengan cara membagi suatu program menjadi bagian-bagian yang lebih kecil yang dapat dikerjakan pada masing-masing prosesor secara bersamaan (Hidayat, 2006).

Tujuan dari penggunaan prosesor paralel adalah untuk mengatasi kendala kecepatan dan kapasitas memori, dengan asumsi bahwa sumber daya paralel sudah tersedia. Seperti komputasi pada prosesor tunggal, kinerja komputasi paralel dipengaruhi oleh teknik pemrograman, arsitektur, atau keduanya (Barney, 2009).

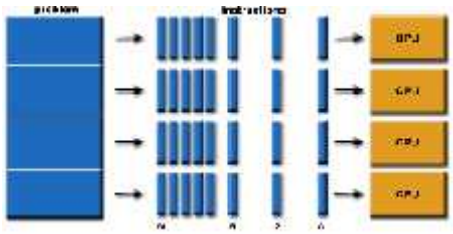
Kinerja prosesor pada dasarnya dilakukan dengan cara menghitung waktu sebelum dilakukan proses pekerjaan dimulai dan diakhiri saat proses pekerjaan selesai. Proses ini pada dasarnya membagi sejumlah *array* ke dalam *sub array* dimana setiap *sub array* dikerjakan oleh satu buah prosesor, secara berurutan dalam kurun waktu tertentu, seperti terlihat pada Gambar 2.3 (Barney, 2009).



Gambar 2.3. Penyelesaian Sebuah Masalah pada Komputasi Tunggal

Sumber : Barney, 2009

Proses paralel adalah proses yang dilakukan pada p buah prosesor, sehingga data yang ada dipecah dalam beberapa bagian, dimana setiap bagian data tersebut di serahkan ke prosesor masing-masing untuk diolah, seperti terlihat pada Gambar 2.4 (Barney, 2009).



Gambar 2.4. Penyelesaian Sebuah Masalah pada Komputasi Paralel

Sumber : Barney, 2009

Dari kedua gambar di atas, dapat disimpulkan bahwa kinerja komputasi paralel lebih efektif dan dapat menghemat waktu untuk pemrosesan data yang banyak daripada komputasi tunggal. Banyak parameter yang dapat digunakan untuk mengukur kinerja sistem paralel, diantaranya adalah waktu komputasi dan *speedup* komputasi paralel (Barney, 2009).

Dalam suatu sistem paralel biasanya *programmer* perlu mengukur kinerja dengan membandingkan waktu proses algoritma paralel dengan waktu proses sekuensial. Pemrograman paralel bertujuan untuk meningkatkan performa komputasi. Semakin banyak hal yang bisa dilakukan secara bersamaan, semakin banyak pekerjaan yang bisa diselesaikan. Batas bawah *speedup* adalah 1 dan batas atasnya adalah jumlah *core* yang digunakan (*p*). Performa dalam pemrograman paralel diukur dari berapa banyak peningkatan kecepatan (*Speed Up*) yang diperoleh dalam menggunakan teknik paralel. Dengan mendefinisikan t_s dan t_p sebagai waktu proses pemrograman serial dan pemrograman paralel, maka *speedup* dapat dihitung dengan persamaan (Wilkinson dan Allen, 2010):

$$\boldsymbol{SpeedUp} = \frac{\mathbf{T_{serial}}}{\mathbf{T_{paralel}}} \tag{2.6}$$

Dimana:

Tserial = waktu eksekusi pada komputasi serial.

Tparalel = waktu eksekusi pada komputasi paralel.

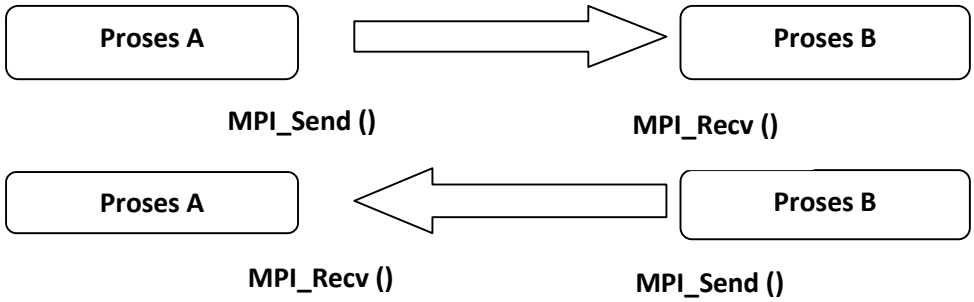
Speed up pada satu prosesor adalah sama dengan satu, dan *speed up* pada p prosesor bernilai $1 \leq S_p \leq p$. Secara ideal *speed up* meningkat sebanding dengan bertambahnya jumlah prosesor. Jadi jika digunakan p prosesor, *speed up* idealnya adalah p(Wilkinson dan Allen, 2010).

2.3.1 *Message Passing Interface* (MPI)

Ada banyak sekali bahasa pemrograman paralel yang diperkenalkan dan sebagian besar merupakan bahasa tingkat tinggi untuk menyederhanakan kompleksitaspemrograman paralel.*Message passing interface* (MPI) merupakan API yang umum digunakan untuk membuat aplikasi paralel(Kurniawan, 2010). Konsep dasar mekanisme komunikasi dari MPI adalah perpindahan data dari satu prosesor ke prosesor yang lainmenggunakan fungsi

dari MPI *run-time library,* dimana satu prosesor mengirim data dan yang lainnya menerima data tersebut(Kurniawan, 2010).

Hampir sebagian besar komunikasi yang terjadi pada MPI didasarkan pada komunikasi *point to point*, sehingga komunikasi ini sangatlah penting dan sebagai dasar untuk komunikasi pada MPI. Komunikasi *point to point* merupakan proses pertukaran data pada sepasang proses dimana satu proses sebagai pengirim dan satu proses lagi sebagai penerima, seperti terlihat pada Gambar 2.5 (Kurniawan, 2010).



Gambar 2.5 komunikasi *point to point*

Dari Gambar 2.5 terlihat bahwa proses A mengirim pesan ke proses B, kemudian proses B kembali mengirim pesan ke proses A. Kedua proses tersebut saling melakukan pertukaran pesan dengan saling mengirim dan menerima data diantara keduanya. Pada MPI ada beberapa fungsi yang disediakan untuk mengirim dan menerima data baik secara *blocking* maupun *nonblocking* (Kurniawan, 2010).

Sebelum mengimplementasikan MPI ke dalam suatu program, seorang *programmer* harus menentukan *header*, komunikator dan rutin manajemen lingkungan apa yang akan digunakan. *Header* diperlukan oleh program untuk memanggil fungsi yang ada pada *library* MPI, *header* yang digunakan dalam implementasi MPI adalah *#include <mpi.h>*. Hampir semua fungsi-fungsi atau *routine* menggunakan komunikator sebagai argumen. Salah satu komunikator yang paling sering digunakan adalah MPI_COMM_WORLD (Barney, 2013). MPI juga memiliki beberapa rutin manajemen lingkungan yang sering digunakan. Rutin manajemen lingkungan MPI tersebut antara lain (Kurniawan, 2010):

a. *MPI_Init,* digunakan untuk menginisialisasi eksekusi MPI. Kode program yang digunakan adalah *MPI_Init(&argc,&argv);*

b. *MPI_Comm_size*, digunakan untuk mengembalikan jumlah proses MPI dalam komunikator.Kode program yang digunakan adalah*MPI_Comm_Size (MPI_COMM_WORLD,&size);*

c. *MPI_Comm_rank*, digunakan untuk memanggil proses MPI dalam komunikator. *Rank* sering juga disebut *task ID* dan digunakan untuk menentukan *source* dan *destination* dari sebuah pesan. Kodeprogram yang digunakan adalah *MPI_Comm_rank (MPI_COMM_WORLD,&rank);*

d. *MPI_Wtime*, digunakan untuk menentukan waktu yang dipakai saat prosesor dipanggil (dalam detik). Kode program yang digunakan adalah *MPI_Wtime();*

e. *MPI_Finalize*, digunakan untuk mengakhiri proses eksekusi MPI. Kode program yang digunakan adalah *MPI_Finalize ();*

Operasi *Nonblocking send/receive* adalah proses yang akan mengembalikan suatu nilai meskipun *buffer* belum penuh dengan data yang akan dikirim/diterima, artinya *nonblocking send/receive* akan mengembalikan nilai walaupun data yang dikirim/diterima belum dieksekusi. Operasi *nonblocking* MPI dikenali dengan nama operasinya dimana dapat dikategorikan menjadi empat bagian yaitu (Kurniawan, 2010):

a. *Immediate,* disingkat dengan I atau i
b. *Buffer,* disingkat dengan B atau b
c. *Synchronous,* disingkat dengan S atau s
d. *Ready,* disingkat dengan R atau r

Pada operasi *non-blocking immediate* digunakan *MPI_Isend()* dan *MPI_Irecv,* karakter I menunjukkan operasi MPI dengan mode segera (*Immediate).* Operasi MPI_*Isend()* dan *MPI_Irecv* merupakan operasi untuk pengiriman dan penerimaan data secara *nonblocking* dan bersifat segera (*immediate*) (Kurniawan, 2010). Deklarasi kode *non-blocking* mode *immediate* seperti terlihat pada Tabel 2.12.

Tabel 2.12. Deklarasi kode *non-blocking* mode *immediate*

Fungsi MPI	**Format penulisan**
Send	int MPI_Isend(void* buf, int count, MPI_Datatype datatype, int dest,int tag, MPI_Comm comm, MPI_Request *request)
Recv	int MPI_Irecv(void* buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Request *request)

Sumber : Kurniawan (2010)

Tabel 2.12 menunjukan parameter-parameter yang digunakan dalam operasi *non-blocking*. Fungsi dan keterangan parameter-parameter tersebut dapat dilihat pada Tabel 2.13 dan Tabel 2.14.

Tabel 2.13. Parameter *non-blocking send immediate*

Parameter	Keterangan	I/O
Buf	*Buffer* data yang akan dikirim	IN
Count	Jumlah *buffer* data	IN
Datatype	Tipe data dari *buffer* data yang akan dikirim	IN
Dest	*Rank* tujuan	IN
Tag	*Message tag*	IN
Comm	*Communicator* yang digunakan	IN
Request	*Communicator request*	OUT

Sumber : Kurniawan, 2010

Tabel 2.14. Parameter *non-blocking recv immediate*

Parameter	Keterangan	I/O
Buf	*Buffer* data yang akan diterima	OUT
Count	Jumlah *buffer* data yang diterima	IN
Datatype	Tipe data dari *buffer* data yang diterima	IN
Source	Sumber *rank* yang akan ditunggu data yang masuk	IN
Tag	*Message tag*	IN
Comm	*Communicator* yang digunakan	IN
Request	*Communicator request*	OUT

Sumber : Kurniawan, 2010

2.3.2*Open Multi-Processing* (OpenMP)

OpenMP pertama kali dirilis pada tahun 1997 dan merupakan sebuah *Application Programming Interface* (API) untuk pemrograman paralel yang mendukung multi *processing shared memory* di C,C++ dan Fortran. OpenMP memiliki keuntungan mudah diterapkan pada arsitektur *multicore portable* untuk mengembangkan aplikasi paralel pada *platform* mulai dari *desktop* sampai super komputer.OpenMP terdiri dari satu set perintah *compiler*, rutin *library,* dan variabel lingkungan yang mempengaruhi *runtime* (Kiessling, 2009).

Direktif *compiler* muncul sebagai komentar dalam kode sumber dan diabaikan oleh *compiler* kecuali jika *programmer* memberitahu sebaliknya, biasanya dengan menetapkan bendera *compiler* yang sesuai. Direktif *compiler* OpenMP digunakan untuk berbagai tujuan sebagai berikut (OpenMP, 2013):

- Pemijahan wilayah paralel
- Membagi blok kode di antara *thread*
- Mendistribusikan iterasi *loop* di antara *thread*
- Serialisasi bagian kode
- Sinkronisasi kerja di antara *thread*

Direktif *compiler* memiliki sintaks sebagai berikut:

#pragma omp nama-direktif *clause* (argumen)

Keterangan sintaks di atas dapat dilihat pada Tabel 2.15.

Tabel 2.15. Sintaks dan keterangan format direktif C/C++ OpenMP

No.	Sintaks	Keterangan
1.	#pragma omp	Dibutuhkan untuk seluruh direktif C/C++ OpenMP
2.	Nama-direktif	Direktif OpenMP valid
3.	*Clause* (argumen)	Opsional

Sumber : OpenMP (2013)

Direktif *parallel for* adalah konsep paralel yang membentuk tim *threads* dan menetapkan bahwa iterasi *looping* akan didistribusikan untuk diksekusi oleh tim *threads*. Untuk menghitung waktu eksekusi pada pemrograman paralel OpenMP digunakan rutin pewaktuan. Rutin pewaktuan digunakan untuk mendukung pengaturan waktu. Fungsi rutin pewaktuan yang digunakan adalah *omp_get_wtime()* yang berfungsi untuk mengembalikan waktu berlalu dalam detik (OpenMP, 2013).