

**PENDETEKSIAN OBJEK YANG BERGERAK DENGAN METODE
CONTOUR MENGGUNAKAN SOFTWARE VISUAL STUDIO
DAN DETEKTOR WEBCAM**

TUGAS AKHIR

Diajukan sebagai Salah Satu Syarat

Untuk Memperoleh Gelar Sarjana di

Jurusan Teknik Elektro



Oleh :

Nama : Meezan Khalil Gibran

No. Mahasiswa : 05 524 019

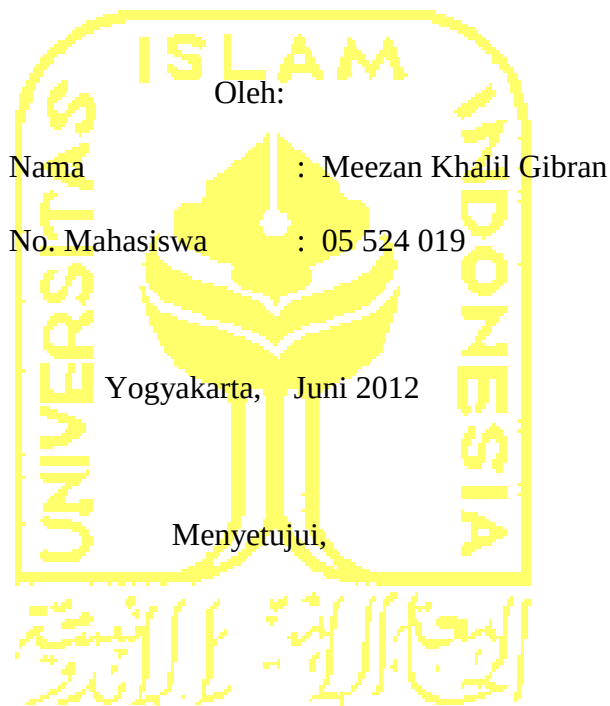
**JURUSAN TEKNIK ELEKTRO
FAKULTAS TEKNOLOGI INDUSTRI
UNIVERSITAS ISLAM INDONESIA
YOGYAKARTA**

2012

LEMBAR PENGESAHAN PEMBIMBING

**PENDETEKSIAN OBJEK YANG BERGERAK DENGAN METODE
CONTOUR MENGGUNAKAN SOFTWARE VISUAL STUDIO
DAN DETEKTOR WEBCAM**

TUGAS AKHIR



Oleh:

Nama : Meezan Khalil Gibran

No. Mahasiswa : 05 524 019

Yogyakarta, Juni 2012

Menyetujui,

Pembimbing I

A blue ink signature of Dwi Ana Ratna Wati, ST., M.Eng.

Dwi Ana Ratna Wati, ST., M.Eng.

Pembimbing II

A blue ink signature of Tito Yuwono S.T., M.Sc.

Tito Yuwono S.T., M.Sc.

LEMBAR PENGESAHAN PENGUJI

**PENDETEKSIAN OBJEK YANG BERGERAK DENGAN METODE
CONTOUR MENGGUNAKAN SOFTWARE VISUAL STUDIO
DAN DETEKTOR WEBCAM**

TUGAS AKHIR

Oleh:

Nama : Meezan Khalil Gibran

No. Mahasiswa : 05 524 019

**Telah Dipertahankan di Depan Sidang Penguji sebagai Salah Satu Syarat
untuk Memperoleh Gelar Sarjana Teknik Elektro**

Fakultas Teknologi Industri Universitas Islam Indonesia

Yogyakarta, 14 Juni 2012

.Tim Penguji,

Tito Yuwono, ST., M.Sc.

Ketua

.....

Wahyudi Budi Pramono, ST., M.Eng.

Anggota I

.....

Alvin Sahroni, ST., M.Eng.

Anggota II

.....

Mengetahui,

Ketua Jurusan Teknik Elektro
Universitas Islam Indonesia



Tito Yuwono, ST., M.Sc.

HALAMAN PERSEMBAHAN

Dengan segala kerendahan dan ketulusan hati,

Karya sederhana ini Kupersembahkan untuk:

Allah SWT Atas segala kemudahan yang Engkau berikan, sehingga satu

amanahmu mampu hamba selesaikan

Sholawat dan salam teruntuk Junjungan kita Nabi Muhammad SAW

Yang telah membawa manusia dari alam kegelapan ke dunia terang

benderang

dan yang selalu kunanti syafaatnya kelak di yaumul qyamah

allahuma amin....

Ayahanda dan Ibunda yang telah mengukir jiwa & ragaku hingga detik

ini

Temen_temenku:

"All Angkatan "TE 05", Yang telah hadir dalam hidupku dan

menggoreskan beraneka tulisan dan lukisan yang tertoreh dalam

catatan perjalananku.

Dan jasa para pendidik yang telah mengantarkanku

hingga hari ini

HALAMAN MOTTO

"Dia pula yang menurunkan kepadamu Kitab diantaranya ayat-ayat yang *mukhamat* jelas dan terang, inilah pokok-pokok Alquran. Yang lainnya *mutasyabihat* (yang susah ditafsirkan). Adapun orang-orang yang hatinya condong kepada kesesatan lebih suka kepada *mutasyabihat* karena ingin mencari fitnah dan takwilnya. Sedangkan yang mengetahui takwilnya hanyalah Allah. Orang-orang yang tinggi ilmunya akan selalu berkata, "Kami beriman kepada Alquran, semua ayatnya berasal dari Tuhan kami." Tiada yang memetik pengertian ini kecuali mereka yang berakal cerdas."

(Al Qur'an QS. Ali Imran : 7)

"Dialah yang mengkaruniakan hikmah kepada yang Ia kehendaki. Siapa pun yang mendapat hikmah, dia telah mendapatkan kebaikan yang berlimpah. Namun yang mampu mengambil peringatan ini hanya orang-orang yang berpikiran cerdas"

(Al Qur'an Al Baqarah : 269)

Manusia senantiasa berusaha dan berdo'a, untuk hasil yang diperoleh bertawakal dan berdzikirlah atas nama Allah SWT

KATA PENGANTAR

Assalaamu'alaikum Wr.Wb.

Syukur alhamdulillah tak henti - hentinya dipanjatkan atas kehadiran Allah SWT yang mana telah melimpahkan rahmat dan hidayah-Nya sehingga tugas akhir ini dapat diselesaikan. Shalawat dan salam semoga tercurah kepada Rasulullah Muhammad SAW beserta para keluarganya, sahabat dan para pengikutnya hingga akhir zaman.

Tugas Akhir berjudul “Pendeteksian Objek Yang Bergerak Dengan Metode *Contour* Menggunakan *Software Visual Studio* Dan Detektor *Webcam*” ini disusun sebagai salah satu syarat untuk memperoleh gelar Sarjana Jurusan Teknik Elektro, Fakultas Teknologi Industri, Universitas Islam Indonesia.

Pada kesempatan ini, ungkapan rasa terima kasih yang sebesar-besarnya diucapkan kepada berbagai pihak yang telah memberikan bantuan, bimbingan, dukungan, kerjasama, fasilitas dan kemudahan lainnya. Untuk itu, dengan ketulusan hati saya mengucapkan terima kasih yang sebesar-besarnya kepada :

1. Kedua orang tua yang telah memberikan semangat baik secara materil maupun moril, hingga tugas akhir ini dapat diselesaikan
2. Bapak Tito Yuwono, ST, MSc. selaku Ketua Jurusan Teknik Elektro Universitas Islam Indonesia.
3. Ibu Dwi Ana Ratna Wati, ST, M.Eng. selaku Pembimbing I Tugas Akhir yang telah meluangkan waktunya sampai terselesaikannya Tugas Akhir ini.

4. Bapak Tito Yuwono, ST., M.Sc. selaku Pembimbing II Tugas Akhir yang telah meluangkan waktunya sampai terselesaikannya Tugas Akhir ini.
5. Teman – teman yang selalu memberikan support dan semangat dalam menyelesaikan tugas akhir ini. Special thanks for : Babe, Ajay, Jogor, Mumun, Yusi, Ki buyut, kang Hasan, bang Ucup, bang Bayu, bang Eko, bung Andi, mas Pram, Sidik, Budi, & Oktavianna Dwi Pertiwi.
6. Dan banyak pihak lain yang tidak dapat kami sebutkan seluruhnya yang telah membantu dalam penyelesaian Tugas Akhir ini.

Penulis menyadari bahwa dalam laporan tugas akhir ini masih jauh dari sempurna, oleh karena itu penulis mengharapkan kritik dan saran membangun dari semua pihak demi kemajuan penulis di masa mendatang.

Harapan penulis laporan tugas akhir ini dapat membantu mengembangkan ilmu pengetahuan penulis pada khususnya dan pembaca pada umumnya.

Wassalaamu'alaikum Wr.Wb.

Yogyakarta, Juni 2012

Penulis

ABSTRAK

Pada aplikasinya metode yang digunakan dalam mendeteksi kontur objek bergerak adalah dengan menggunakan metode pendeteksian tepi. Pendeteksian tepi yang paling optimal digunakan saat ini yaitu dengan metode *Canny Edge Detection*. Pendeteksian tepi *Canny* ini dapat dilakukan dengan 4 langkah, yaitu *smoothing*, *gradient computation*, *non maximal suppression*, dan *hysteresis thresholding*. Sebelum dideteksi tepiannya citra RGB harus dikonversi terlebih dahulu menjadi citra *grayscale*. Aplikasi ini menggunakan Bahasa C++ sebagai bahasa pemrogramannya yang terdapat pada *Visual Studio 2005* yang diintegrasikan dengan OpenCV sebagai *library*-nya. Pengujian dilakukan dengan beberapa warna pada *background* dan perubahan intensitas cahaya, kemudian dilakukan analisa. Deteksi gerakan sangat berpengaruh terhadap perubahan baik warna *background* maupun intensitas cahaya. Hasil yang diperoleh dari aplikasi ini yaitu tampilan kontur suatu objek yang sedang bergerak dapat terdeteksi dengan baik di saat objek tersebut mengalami pergerakan.

Kata kunci : Pengolahan citra, Deteksi Tepi Canny, Kontur, *Visual Studio*, OpenCV *library*.

DAFTAR ISI

Halaman Judul	i
Lembar Pengesahan Pembimbing	ii
Lembar Pengesahan Penguji	iii
Halaman Persembahan	iv
Halaman Motto	v
Kata Pengantar	vi
Abstrak	viii
Daftar Isi	ix
Daftar Gambar	xiii
Daftar Tabel	xvi

BAB I : PENDAHULUAN

1.1 Latar belakang	1
1.2 Rumusan Masalah	2
1.3 Batasan Masalah	3
1.4 Tujuan Penelitian	3
1.5 Sistematika Penulisan	3

BAB II : TINJAUAN PUSTAKA

2.1 Tinjauan Pustaka	5
2.2 Pengolahan Citra	5
2.2.1 Piksel Dan Hubungan Antar Piksel	6
2.2.2 Citra Digital	7

2.2.3 Pengolahan Citra Digital	8
2.2.4 Elemen - Elemen Citra Digital	9
2.2.5 Segmentasi Citra	11
2.3 <i>Computer Vision</i>	12
2.3.1 Hubungannya Dengan Pengolahan Citra	13
2.3.2 <i>Motion Detection</i>	14
2.4 Citra <i>Grayscale</i>	15
2.5 Pendeteksian Tepi	18
2.5.1 Deteksi Tepi Unsur Pada Citra	18
2.5.2 Deteksi Tepi <i>Canny</i>	20
2.6 <i>Contour</i>	25
2.7 <i>Visual Studio 2005</i>	28
2.8 Bahasa Pemrograman C++	30
2.8.1 Sejarah Bahasa C++	31
2.8.2 Keunggulan Bahasa C++	32
2.9 <i>OpenCV Library</i>	33
2.9.1 Fitur <i>OpenCV</i>	33
2.9.2 <i>Setting OpenCV Library</i>	34
2.10 Sistem <i>Realtime</i>	36

BAB III : PERANCANGAN SISTEM

3.1 Perancangan Sistem	38
3.2 Proses Pengambilan Gambar	40
3.2.1 Proses Pembuatan <i>Window</i>	41

3.2.2 Penempatan Dan Tipe Gambar	42
3.3 Konversi <i>Gray Image</i>	42
3.4 Pendeteksian Tepi <i>Canny</i>	43
3.4.1 Penghalusan Citra (<i>Smoothing</i>)	44
3.4.2 Perhitungan Potensi Gradien Citra	46
3.4.3 <i>Non Maximum Suppression</i>	48
3.4.4 <i>Hysteresis Thresholding</i>	49
3.4.5 <i>Coding</i> Deteksi Tepi <i>Canny</i> Pada <i>Visual C++</i>	49
3.5 Proses Deteksi <i>Contour</i>	50
3.6 Proses <i>Motion Detection</i>	51
3.6.1 Proses Pengambilan Dan Pengolahan Gambar	52
3.6.2 Perhitungan Perbedaan <i>Image</i>	53
3.7 Citra <i>Output</i>	54
3.8 Analisis Kebutuhan Perangkat Keras	54
3.9 Analisis Kebutuhan Perangkat Lunak	55

BAB IV : PENGUJIAN, ANALISIS DAN PEMBAHASAN

4.1 <i>Background</i> Putih	57
4.1.1 Kondisi Cahaya Terang (Cahaya Matahari)	57
4.1.2 Kondisi Cahaya Agak Terang (Cahaya Lampu)	57
4.1.3 Kondisi Gelap (Tanpa Cahaya)	58
4.2 <i>Background</i> Hitam	58
4.2.1 Kondisi Cahaya Terang (Cahaya Matahari)	58
4.2.2 Kondisi Cahaya Agak Terang (Cahaya Lampu)	59

4.2.3 Kondisi Gelap (Tanpa Cahaya)	60
4.3 <i>Background</i> Merah	60
4.3.1 Kondisi Cahaya Terang (Cahaya Matahari)	60
4.3.2 Kondisi Cahaya Agak Terang (Cahaya Lampu)	61
4.3.3 Kondisi Gelap (Tanpa Cahaya)	61
4.4 <i>Background</i> Hijau	62
4.4.1 Kondisi Cahaya Terang (Cahaya Matahari)	62
4.4.2 Kondisi Cahaya Agak Terang (Cahaya Lampu)	63
4.4.3 Kondisi Gelap (Tanpa Cahaya)	63
4.5 <i>Background</i> Biru	64
4.5.1 Kondisi Cahaya Terang (Cahaya Matahari)	64
4.5.2 Kondisi Cahaya Agak Terang (Cahaya Lampu)	64
4.5.3 Kondisi Gelap (Tanpa Cahaya)	65
4.6 <i>Background</i> Warna Lain	66
4.6.1 Kondisi Cahaya Terang (Cahaya Matahari)	66
4.6.2 Kondisi Cahaya Agak Terang (Cahaya Lampu)	69
4.6.3 Kondisi Gelap (Tanpa Cahaya)	72

BAB V : PENUTUP

5.1 Kesimpulan	75
5.2 Saran	76

DAFTAR PUSTAKA

LAMPIRAN

DAFTAR GAMBAR

Gambar 2.1 Blok diagram Pengolahan Citra	6
Gambar 2.2 Proses <i>Motion Detection</i>	14
Gambar 2.3 Komposisi Warna RGB	16
Gambar 2.4 Ruang Warna RGB	16
Gambar 2.5 Hasil Proses Konversi RGB ke Bentuk <i>Grayscale</i>	18
Gambar 2.6 Model Tepi Satu Dimensi	18
Gambar 2.7 Matriks <i>Template</i> Pendeteksi Tepi	19
Gambar 2.8 Gambar Proses Konvolusi	19
Gambar 2.9 Contoh Hasil Deteksi Tepi <i>Prewitt</i>	20
Gambar 2.10 Citra Derau Dan Hasil Deteksi Tepi <i>Prewitt</i> -nya	20
Gambar 2.11 Blok Diagram Deteksi Tepi <i>Canny</i>	21
Gambar 2.12 Hasil Deteksi Tepi Pada Citra Terfilter	22
Gambar 2.13 <i>Convolution Mask Operator Sobel</i>	23
Gambar 2.14 <i>Edge Direction</i>	24
Gambar 2.15 Contoh Proses Deteksi Tepi <i>Canny</i>	25
Gambar 2.16 Contoh Tes Pencarian Kontur Pada Suatu Citra	26
Gambar 2.17 Bentuk Tipe <i>Contour</i>	27
Gambar 2.18 Tampilan Awal <i>Visual Studio 2005</i>	29
Gambar 2.19 Klik <i>Tools</i> Pilih <i>Options</i>	34
Gambar 2.20 Klik VC++ <i>Directories</i>	35
Gambar 2.21 Contoh <i>Webcam</i>	37

Gambar 3.1 Gambaran Umum Sistem	38
Gambar 3.2 Blok Diagram Sistem	39
Gambar 3.3 <i>Convolution Mask</i> Operator Sobel	47
Gambar 3.4 Proses <i>Non Maximum Suppression</i>	48
Gambar 3.5 Contoh Proses Deteksi Tepi <i>Canny</i>	50
Gambar 3.6 <i>Motion Detection</i>	51
Gambar 4.1 <i>Background</i> Putih Pada Kondisi Terang	57
Gambar 4.2 <i>Background</i> Putih Kondisi Cahaya Lampu	57
Gambar 4.3 <i>Background</i> Putih Pada Kondisi Gelap	58
Gambar 4.4 <i>Background</i> Hitam Pada Kondisi Terang	59
Gambar 4.5 <i>Background</i> Hitam Pada Kondisi Cahaya Lampu	59
Gambar 4.6 <i>Background</i> Hitam Pada Kondisi Gelap	60
Gambar 4.7 <i>Background</i> Merah Pada Kondisi Terang	60
Gambar 4.8 <i>Background</i> Merah Pada Kondisi Cahaya Lampu	61
Gambar 4.9 <i>Background</i> Merah Pada Kondisi Gelap	62
Gambar 4.10 <i>Background</i> Hijau Pada Kondisi Terang	62
Gambar 4.11 <i>Background</i> Hijau Pada Kondisi Cahaya Lampu	63
Gambar 4.12 <i>Background</i> Hijau Pada Kondisi Gelap	63
Gambar 4.13 <i>Background</i> Biru Pada Kondisi Terang	64
Gambar 4.14 <i>Background</i> Biru Pada Kondisi Cahaya Lampu	65
Gambar 4.15 <i>Background</i> Biru Pada Kondisi Gelap	65
Gambar 4.16 <i>Background</i> Kuning Pada Kondisi Terang	66
Gambar 4.17 <i>Background</i> Ungu Pada Kondisi Terang	66

Gambar 4.18 <i>Background Cyan</i> Pada Kondisi Terang	67
Gambar 4.19 <i>Background Merah Muda</i> Pada Kondisi Terang	67
Gambar 4.20 <i>Background Jingga</i> Pada Kondisi Terang	68
Gambar 4.21 <i>Background Kuning</i> Pada Kondisi Cahaya Lampu	69
Gambar 4.22 <i>Background Ungu</i> Pada Kondisi Cahaya Lampu	69
Gambar 4.23 <i>Background Cyan</i> Pada Kondisi Cahaya Lampu	70
Gambar 4.24 <i>Background Merah Muda</i> Pada Kondisi Cahaya Lampu	70
Gambar 4.25 <i>Background Jingga</i> Pada Kondisi Cahaya Lampu	71
Gambar 4.26 <i>Background Kuning</i> Pada Kondisi Gelap	72
Gambar 4.27 <i>Background Ungu</i> Pada Kondisi Gelap	72
Gambar 4.28 <i>Background Cyan</i> Pada Kondisi Gelap	73
Gambar 4.29 <i>Background Merah Muda</i> Pada Kondisi Gelap	73
Gambar 4.30 <i>Background Jingga</i> Pada Kondisi Gelap	74

DAFTAR TABEL

Tabel 3.1 Tabel Kernel Gaussian Filter Dengan $c = 1$	45
Tabel 3.2 Tabel Kernel Gaussian Filter Dengan $c = 99$	45

BAB I

PENDAHULUAN

1.1 Latar Belakang

Pendeteksian benda yang bergerak merupakan tingkat awal yang dapat di aplikasikan secara luas oleh komputer, seperti *video surveillance*, robotika, sistem otentikasi, *user interface* dengan gerak tangan, dan tahap pra-kompresi gambar MPEG4. Pengembangan software dari tingkat awal dalam sebuah aplikasi sangat penting karena sangat berpengaruh terhadap kinerja semua tingkat yang lebih tinggi dari berbagai aplikasi lanjutannya.

Deteksi gerak merupakan salah satu masalah yang dapat dipecahkan atau dipelajari menggunakan komputer. Ada dua jenis pendekatan yaitu pendekatan berbasis wilayah dan pendekatan berbasis batas. Pendekatan berbasis wilayah digunakan untuk mendeteksi gerak tanpa menggunakan model. Ada beberapa metode yang dapat digunakan pada pendekatan jenis ini, diantaranya adalah *background subtraction* dan *optical flow*. Masalah yang umum pada metode *background subtraction* ini adalah membutuhkan waktu yang lama untuk memperkirakan model latar belakang. Diperlukan waktu beberapa detik untuk estimasi model latar belakang karena kecepatan dari perubahan pencahayaan dan gerakan kecil di latar belakang sangat lambat. Metode *optical flow* juga memiliki masalah yang disebabkan perubahan pencahayaan.

Pada kasus pendekatan berbasis batas, banyak yang menggunakan pendeteksi tepi yang berbasis aliran optik, tingkat set, dan kontur aktif, seperti

metode *snakes*, *balloons*, dan *geodesic active contours*. Metode yang sering digunakan yaitu pendeteksian tepi yang berbasis aliran optik (*optical flow*). Salah satu keunggulan dari metode ini adalah tidak sensitif terhadap perubahan iluminasi. Namun, hasil dari metode ini tidak tepat bila gambar tidak diperhalus, karena menghitung kecepatan tanpa menggunakan informasi spasial piksel lingkungannya. Sebagai hasil ekstraksi pendeteksian tepi ini akan menghasilkan suatu *contour* objek yang bergerak tersebut.

Software yang akan digunakan pada penelitian ini yaitu Visual Studio 2005 dengan menggunakan bahasa C sebagai bahasa pemrograman. C merupakan sebuah bahasa pemrograman yang berorientasi pada objek yang dikembangkan oleh *Microsoft* dan menjadi salah satu bahasa pemrograman yang mendukung *.Net programming* melalui Visual Studio.

1.2 Rumusan Masalah

Dari latar belakang yang sudah dijelaskan di atas, maka dapat dibuat suatu rumusan masalah, yaitu sebagai berikut :

1. Bagaimana mengubah data dari suatu citra analog menjadi data biner dengan menggunakan perangkat komputer.
2. Bagaimana mengolah data biner tersebut untuk menghasilkan suatu tepian (*contour*).
3. Bagaimana mendeteksi objek yang sedang bergerak berdasarkan *contour* dari objek tersebut.

1.3 Batasan Masalah

Dari rumusan tersebut terdapat batasan-batasan masalah yang dapat dijadikan penelitian lanjutan. Batasan masalah yang diperoleh, yaitu :

1. Pendeteksi objek yang bergerak menggunakan *webcam*.
2. *Webcam* digunakan sebagai detektor yang dibatasi dalam satu ruangan.
3. *Contour construction* dijadikan sebagai metode dalam pendeteksi pergerakan objek.

1.4 Tujuan Penelitian

Tujuan dari penelitian tugas akhir ini adalah terciptanya suatu sistem yang dapat digunakan untuk mendeteksi pergerakan suatu objek berdasarkan *contour* dari suatu objek tersebut, dan dapat ditampilkan secara *real time* pada komputer.

1.5 Sistematika Penulisan

Sistematika penulisan dan pembahasan laporan tugas akhir ini dapat dijelaskan sebagai berikut :

BAB I PENDAHULUAN

Pada bab ini diuraikan tentang judul, latar belakang, rumusan masalah, batasan masalah, dan tujuan serta sistematika pembahasan dari tugas akhir ini sendiri.

BAB II TINJAUAN PUSTAKA

Pada bab ini diuraikan mengenai teori-teori pengolahan citra dan deteksi tepi (*edge detection/contour*) yang mendukung dalam pelaksanaan penyelesaian tugas akhir, serta *hardware* dan *software* yang digunakan pada tugas akhir.

BAB III PERANCANGAN SISTEM

Dalam bab ini diuraikan tentang perencanaan serta langkah-langkah dalam suatu sistem pendeteksian tepi (*contour*) objek yang sedang bergerak dengan menggunakan Bahasa C sebagai bahasa pemrogramannya pada Visual Studio 2005.

BAB IV PENGUJIAN, ANALISIS DAN PEMBAHASAN

Hasil dari perancangan yang telah dibuat, akan menghasilkan suatu *software* yang dapat digunakan untuk mendeteksi tepian objek yang sedang bergerak melalui proses *capture* secara langsung (*real time*) terhadap suatu objek yang bergerak dengan menggunakan *webcam*. Hasil dari pendeteksi tepian ini kemudian akan langsung ditampilkan pada komputer. Pada bab ini juga akan melaporkan hasil dari pengujian terhadap *software* tersebut untuk kemudian di analisa, sehingga dari *software* yang telah diselesaikan dapat diketahui seberapa jauh kebenaran yang dihasilkan dalam praktek bila dibandingkan dengan teori-teori penunjang yang ada.

BAB V PENUTUP

Bab ini berisikan tentang kesimpulan dari pengujian dalam pendeteksi tepian objek yang sedang bergerak berdasarkan pada *contour*-nya serta saran-saran yang mendukung agar penelitian yang telah dilakukan menjadi lebih baik.

BAB II

TINJAUAN PUSTAKA

2.1 Tinjauan Pustaka

Pendekatan yang cepat dan akurat untuk mendeteksi dan melacak obyek yang sedang bergerak adalah menggunakan metode ekstraksi tepi sebuah obyek menggunakan aliran optik dan detektor tepi menjadi sebuah garis dimana latar belakang *frame* sebelumnya dikurangi terlebih dahulu. Kontur obyek diperoleh dengan menggunakan metode *snakes* untuk mengelompokkan baris tersebut. Obyek yang dilacak akan terdeteksi dan setiap obyek yang dilacak memiliki sebuah daerah untuk menangani oklusi dan gangguan (Yokoyama, M., & Poggio, T., 2005).

Perancangan sistem untuk mendeteksi gerakan dapat menggunakan *webcam* sebagai media pemantauan. Jika *webcam* menangkap obyek yang ukurannya setara dengan ukuran manusia, alat ini akan mengaktifkan alarm dan merekam obyek tersebut (Firdausy, K., at all, 2008).

2.2 Pengolahan Citra

Sebuah citra diubah ke bentuk digital agar dapat disimpan dalam memori komputer atau media lain. Proses mengubah citra ke bentuk digital bisa dilakukan dengan beberapa perangkat, misalnya *scanner*, kamera digital, dan *handycam*. Ketika sebuah citra sudah diubah ke dalam bentuk digital (selanjutnya disebut citra digital), bermacam-macam proses pengolahan citra dapat diperlakukan terhadap citra tersebut.

Image processing atau sering disebut dengan pengolahan citra digital merupakan suatu proses dari gambar asli menjadi gambar lain yang sesuai dengan keinginan kita. Misal suatu gambar yang kita dapatkan terlalu gelap maka dengan *image processing* gambar tersebut bisa kita proses sehingga mendapat gambar yang jelas. Secara garis besar dapat diilustrasikan sebagai berikut :



Gambar 2.1. Blok Diagram Pengolahan Citra

2.2.1 Piksel Dan Hubungan Antar Piksel

Pada citra digital dapat dilihat titik-titik kecil berbentuk segiempat yang membentuk citra tersebut. Titik-titik tersebut merupakan satuan terkecil dari suatu citra digital yang disebut sebagai “*picture element*”, “*pixel*”, “*piksel*”, “*pel*”. Jumlah piksel per satuan panjang akan menentukan resolusi citra tersebut. Makin banyak piksel yang mewakili suatu citra, makin tinggi nilai resolusinya dan makin halus gambarnya. Pada sistem dengan tampilan citra digital yang dirancang dengan baik (beresolusi tinggi), titik-titik kecil tersebut tidak teramati oleh manusia yang melihat secara normal.

Piksel merupakan satuan komponen terkecil dari gambar yang menentukan ukuran dari suatu gambar. Resolusi citra menyatakan ukuran bit dari suatu citra dalam satuan piksel. Untuk citra biner, piksel gambar yang kecerahannya dibawah tingkat tertentu diwakili oleh “0” sedangkan diatasnya diwakili oleh “1”, dengan demikian semua citra didalam *memory* komputer dapat diwakili oleh logika “1”

dan “0”. Dengan demikian, jumlah *memory* yang dibutuhkan untuk menyimpan citra tersebut dapat dihitung. Sebagai contoh citra yang memiliki resolusi 256 x 256 piksel (jumlah total 65536), karena setiap piksel diwakili oleh “1” dan “0” dan komputer membutuhkan satu bit untuk menyimpan setiap piksel, sehingga dibutuhkan total bit sekitar 64 Kb.

2.2.2 Citra Digital

Istilah citra yang digunakan dalam bidang pengolahan citra dapat diartikan sebagai suatu fungsi kontinyu dari intensitas cahaya dalam bidang dua dimensi. Pemrosesan citra dengan komputer digital membutuhkan citra digital sebagai masukannya. Citra digital adalah citra kontinu yang diubah dalam bentuk diskrit, baik koordinat ruang maupun intensitas cahayanya. Citra digital dapat dinyatakan dalam matriks dua dimensi $f(x,y)$ dimana x dan y merupakan koordinat piksel dalam matriks dan ' f ' merupakan derajat intensitas piksel tersebut. Citra digital berbentuk matriks dengan ukuran $M \times N$ akan tersusun sebagai berikut :

$$f(x,y) = \begin{pmatrix} f(0,0) & f(0,1) & f(0,2) & \dots & f(0,N-1) \\ f(1,0) & f(1,1) & f(1,2) & \dots & f(1,N-1) \\ f(2,0) & f(2,1) & f(2,2) & \dots & f(2,N-1) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ f(M-1,0) & f(M-1,1) & f(M-1,2) & \dots & f(M-1,N-1) \end{pmatrix} \quad [2.1]$$

Suatu citra $f(x, y)$ dalam fungsi matematis dapat dituliskan sebagai berikut :

$$f(x,y) \begin{cases} 0 \leq x \leq M-1 \\ 0 \leq y \leq N-1 \\ 0 \leq f(x, y) \leq G-1 \end{cases} \quad [2.2]$$

Dimana : M = banyaknya garis pada *array* citra (lebar)
 N = banyaknya kolom pada *array* citra (tinggi)
 G = banyaknya skala keabuan (*gray level*)

Interval $0, G$ disebut skala keabuan (*grayscale*). Besar G tergantung pada proses digitalisasinya. Biasanya keabuan 0 (nol) menyatakan intensitas putih. Pada citra 8-bit, nilai G sama dengan $2^8 = 256$ warna (derajat keabuan).

2.2.3 Pengolahan Citra Digital

Citra ada dua macam: citra kontinu dan citra diskrit (digital). Citra kontinu dihasilkan dari sistem optik yang menerima sinyal analog, misalnya mata manusia dan kamera analog. Citra diskrit dihasilkan melalui proses digitalisasi terhadap citra kontinu. Beberapa sistem optik dilengkapi dengan fungsi digitalisasi sehingga menghasilkan citra diskrit, misalnya kamera digital dan *scanner*. Citra diskrit disebut juga citra digital. Komputer digital yang umum dipakai saat ini hanya dapat mengolah citra digital.

Digital Image Processing atau biasa disebut dengan pengolahan citra digital adalah salah satu bidang ilmu pengetahuan yang dapat dikatakan berkembang sangat pesat sejalan dengan berkembangnya kemajuan teknologi pada industri yang ada saat ini. Pengolahan citra digital merupakan suatu proses perubahan gambar asli menjadi gambar lain yang sesuai dengan keinginan kita tanpa meninggalkan tekstur yang ada pada gambar asli tersebut. Jika ingin mendefinisikannya lebih bebas lagi, pengolahan citra dapat didefinisikan sebagai pengolahan bentuk gambar/citra yang hasilnya dapat diterima secara baik oleh indera penglihatan. (Riyanto S., 2005).

Proses pengolahan citra digital dengan menggunakan komputer digital adalah suatu proses yang terlebih dahulu melakukan transformasi citra ke dalam bentuk besaran-besaran diskrit dari nilai tingkat keabuan atau derajat keabuan pada titik - titik elemen citra. Bentuk citra ini biasa disebut dengan citra digital. Elemen-elemen citra digital apabila citra tersebut ditampilkan didalam layar monitor akan menempati sebuah ruang yang biasa disebut dengan *dot* atau piksel (*picture elemen/pixel*).

2.2.4 Elemen - Elemen Citra Digital

Citra digital mengandung sejumlah elemen-elemen dasar yang dapat dimanipulasi dalam pengolahan citra dan dieksploitasi lebih lanjut pada *computer vision*. Elemen-elemen citra digital yang penting diantaranya adalah :

1. Kecerahan (*Brightness*)

Kecerahan merupakan intensitas cahaya, dimana pada sebuah piksel di dalam citra bukanlah intensitas riil, tetapi intensitas rata-rata dari suatu area yang melingkupinya. Sistem visual manusia mampu menyesuaikan dirinya dengan tingkat kecerahan (*brightness level*) mulai dari yang paling rendah sampai yang paling tinggi dengan jangkauan sebesar 10^{10} .

2. Kontras (*Contrast*)

Kontras menyatakan sebaran terang (*lightness*) dan gelap (*darkness*) di dalam sebuah gambar. Citra dengan kontras rendah dicirikan oleh sebagian besar komposisi citranya adalah terang atau sebagian besar gelap. Pada citra dengan kontras yang baik, komposisi gelap dan terang tersebar secara merata.

3. Kontur (*Contour*)

Kontur adalah keadaan yang ditimbulkan perubahan intensitas pada *pixel* yang bertetangga. Karena adanya perubahan intensitas inilah mata manusia mampu mendeteksi tepi-tepi (*edge*) objek di dalam citra.

4. Warna (*Colour*)

Warna adalah persepsi yang dirasakan oleh sistem visual manusia terhadap panjang gelombang (λ) yang berbeda. Warna yang diterima oleh mata merupakan hasil kombinasi cahaya dengan panjang gelombang berbeda. Penelitian memperlihatkan bahwa kombinasi warna yang memberikan rentang warna yang paling lebar adalah *red* (*R*), *green* (*G*), dan *blue* (*B*). Ketiga warna tersebut dinamakan warna pokok (*primaries*).

5. Bentuk (*Shape*)

Shape adalah merupakan properti intrinsik untuk sistem visual manusia (objek 3 dimensi). Manusia lebih sering mengasosiasikan objek dengan bentuknya daripada elemen lainnya (warna misalnya). Pada umumnya, citra yang dibentuk merupakan citra dwimatra (2 dimensi), sedangkan objek yang dilihat berbentuk trimatra (3 dimensi).

6. Tekstur (*Texture*)

Tekstur dicirikan sebagai distribusi spasial dari derajat keabuan di dalam sekumpulan *pixel* yang bertetangga. Jadi, tekstur tidak dapat didefinisikan sebagai sebuah *pixel*. Sistem visual manusia pada hakikatnya tidak menerima informasi citra secara independen pada setiap *pixel*, melainkan suatu citra dianggap sebagai satu kesatuan.

2.2.5 Segmentasi Citra

Segmentasi citra merupakan bagian dari proses pengolahan citra. Proses segmentasi citra ini lebih banyak merupakan suatu proses pra-pengolahan citra pada sistem pengenalan objek dalam citra. Segmentasi citra (*image segmentation*) mempunyai arti membagi suatu citra menjadi wilayah-wilayah yang homogen berdasarkan kriteria keserupaan yang tertentu antara tingkat keabuan suatu piksel dengan tingkat keabuan piksel-piksel tetangganya, kemudian hasil proses segmentasi ini akan digunakan untuk proses tingkat tinggi lebih lanjut yang dapat dilakukan terhadap suatu citra, misalnya proses klasifikasi citra dan proses identifikasi objek. Adapun dalam proses segmentasi citra itu sendiri terdapat beberapa algoritma, diantaranya : algoritma deteksi titik, deteksi garis, dan deteksi sisi (berdasarkan Operator *Robert* dan Operator *Sobel*). Algoritma segmentasi didasarkan pada dua buah karakteristik nilai derajat kecerahan citra, yaitu : *discontinuity* dan *similarity*.

Segmentasi bertujuan untuk memilih dan mengisolasi (memisahkan) suatu objek dari keseluruhan citra. Segmentasi terdiri dari *downsampling*, penapisan, dan deteksi tepian. Tahap *downsampling* merupakan proses untuk menurunkan jumlah piksel dan menghilangkan sebagian informasi dari citra. Dengan resolusi citra yang tetap, *downsampling* menghasilkan ukuran citra yang lebih kecil. Tahap segmentasi selanjutnya adalah penapisan dengan filter median, hal ini dilakukan untuk menghilangkan derau yang biasanya muncul pada frekuensi tinggi pada spektrum citra. Pada penapisan dengan filter median, *gray level* citra pada setiap piksel digantikan dengan nilai median dari *gray level* pada

piksel yang terdapat pada *window filter*. Tahap yang terakhir pada proses segmentasi yaitu deteksi tepian. Pendekatan algoritma *Canny* dilakukan berdasarkan konvolusi fungsi citra dengan operator Gaussian dan turunan-turunannya. Pendeteksi tepi ini dirancang untuk merepresentasikan sebuah tepian yang ideal, dengan ketebalan yang diinginkan.

2.3 *Computer Vision*

Terminologi lain yang berkaitan erat dengan pengolahan citra adalah *computer vision* atau *machine vision*. Pada hakikatnya, *computer vision* mencoba meniru cara kerja sistem visual manusia (*human vision*). *Human vision* sesungguhnya sangat kompleks. Manusia melihat objek dengan indera penglihatan (mata), lalu citra objek diteruskan ke otak untuk diinterpretasi sehingga manusia mengerti objek apa yang tampak dalam pandangan matanya. Hasil interpretasi ini mungkin digunakan untuk pengambilan keputusan (misalnya menghindari, jika melihat mobil melaju di depan).

Computer Vision adalah transformasi atau perubahan dari data-data yang dapat berupa gambar diam ataupun video kamera menjadi bentuk lain atau suatu representasi baru dan membantu dalam pengambilan keputusan. Semua bentuk transformasi yang dilakukan di arsipkan atau disimpan untuk tujuan tertentu. Input data dapat berasal dari informasi yang berhubungan dengan objek. Pada representasi barunya dimungkinkan berarti merubah warna dari suatu citra menjadi citra yang *grayscale* atau menghilangkan pergerakan kamera dari suatu citra berurutan.

2.3.1 Hubungannya Dengan Pengolahan Citra

Pengolahan citra dan pengenalan pola merupakan bagian dari *computer vision*. Pengolahan citra merupakan proses awal (*preprocessing*) pada *computer vision*, sedangkan pengenalan pola merupakan proses untuk menginterpretasi citra. Teknik-teknik di dalam pengenalan pola memainkan peranan penting dalam *computer vision* untuk mengenali objek. Jika dihubungkan dengan grafika komputer, maka *computer vision* merupakan kebalikannya.

Grafika komputer membentuk (sintesis) citra, sedangkan *computer vision* mengoraknya (analisis). Pada masa awal kedua bidang ini, tidak ada hubungan antara keduanya, tetapi beberapa tahun belakangan kedua bidang tersebut berkembang semakin dekat. *Computer vision* menggunakan representasi kurva dan permukaan dan beberapa teknik lain dari grafika komputer, sedangkan grafika komputer menggunakan teknik-teknik di dalam *computer vision* untuk memuat citra realistik (*virtual reality*).

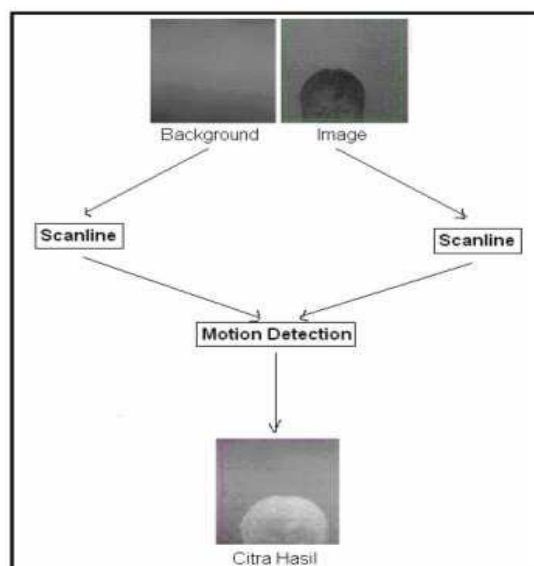
Proses-proses di dalam *computer vision* dapat digolongkan menjadi tiga aktivitas, yaitu :

1. Memperoleh atau mengakuisisi citra digital.
2. Melakukan teknik komputasi untuk memproses atau memodifikasi data citra (operasi - operasi pengolahan citra).
3. Menganalisis dan menginterpretasi citra dan menggunakan hasil pemrosesan untuk tujuan tertentu, misalnya memandu robot, mengontrol peralatan, memantau proses manufaktur dan lain sebagainya.

2.3.2 Motion Detection

Motion detection merupakan penelitian yang penting dalam keilmuan *computer vision*. Banyak metode *motion detection* yang telah ditemukan, satu diantaranya adalah dengan menghitung perbedaan nilai-nilai intensitas pada suatu piksel dari dua *frame* gambar yang diambil secara berturut-turut yang kemudian dilakukan proses *thresholding* untuk menentukan adanya perubahan objek atau tidak. Sekalipun metode ini sangat sederhana dalam proses implementasinya, tapi metode ini merupakan metode dasar dari proses *motion detection*. Hanya saja metode ini kurang efektif untuk menentukan pergerakan objek secara keseluruhan, terutama bagian dalam dari objek yang bergerak, akan tetapi secara umum metode ini sudah mampu mengidentifikasi adanya perubahan objek. Pengurangan nilai-nilai intensitas setiap piksel yang ada pada *background* terhadap suatu *image* baru merupakan metode yang paling populer untuk proses *motion detection*.

Langkah – langkah dari *motion detection* secara umum adalah :



Gambar 2.2. Proses *Motion Detection*

Pada gambar di atas, sebelah kiri atas merupakan gambar asli atau *current image* (*background*), sebelah kanan atas merupakan gambar dengan adanya objek atau *previous image*, sedangkan gambar dibawahnya merupakan hasil dari proses *motion detection*. Langkah-langkah *motion detection* adalah :

1. Memasukkan gambar pertama sebagai *background*.
2. Memasukkan gambar kedua sebagai gambar yang ada objeknya.
3. Gambar pada langkah kesatu dan kedua lakukan *scanline*, fungsinya adalah untuk menemukan apakah ada perubahan objek atau tidak ada perubahan. *Scanline* dilakukan dari kiri atas gambar sampai dengan kanan bawah gambar.
4. Lakukan perbandingan antara gambar *background* dengan gambar satunya lagi, hal ini dilakukan supaya bisa mengetahui adanya perubahan atau adanya perbedaan antara gambar kesatu dan gambar kedua. Pada langkah ini biasanya ada proses *thresholding* dan deteksi pinggiran jika diperlukan. Ini juga disebut proses *motion detection*.
5. Setelah melakukan proses diatas maka akan dihasilkan citra hasil *motion detection*.

2.4 Citra Grayscale

Proses awal yang banyak dilakukan dalam Pengolahan Citra adalah mengubah citra berwarna menjadi citra *grayscale*, hal ini digunakan untuk menyederhanakan model citra. Pada awalnya citra terdiri dari 3 layer matrik yaitu R-layer, G-layer dan B-layer. Perhatikan gambar di bawah ini :

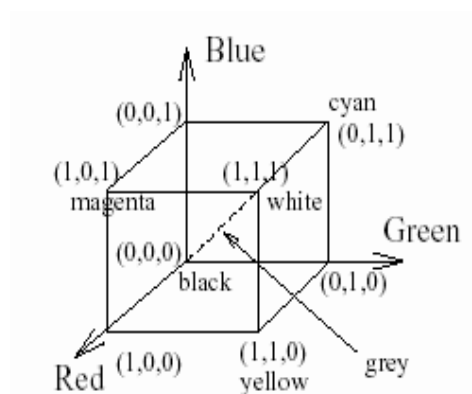


Gambar 2.3. Komposisi Warna RGB

Model warna RGB mendeskripsikan warna sebagai kombinasi positif dari 3 (tiga) warna dasar, yaitu merah (*red*), hijau (*green*), dan biru (*blue*) sehingga kombinasi warna-warna tersebut nantinya akan membentuk sebuah warna baru yang dimisalkan dengan C , dengan rumusan sebagai berikut,

$$C = rR, gG, bB \quad [2.3]$$

Jika skalar r , g , b diberi nilai antara 0 dan 1, maka semua definisi warna akan memiliki nilai yang berbeda-beda. Sebagai perumpamaan nilai-nilai tersebut berada dalam kubus seperti berikut :



Gambar 2.4. Ruang Warna RGB

Sehingga untuk melakukan proses-proses selanjutnya tetap diperhatikan tiga layer di atas. Bila setiap proses perhitungan dilakukan menggunakan tiga layer, berarti dilakukan tiga perhitungan yang sama. Sehingga konsep itu diubah dengan mengubah 3 layer di atas menjadi 1 layer matrik *grayscale* dan hasilnya adalah citra *grayscale*. Dalam citra ini tidak ada lagi warna, yang ada adalah derajat keabuan. Pengubahan citra RGB yang mempunyai matrik masing-masing R (*red*), G (*green*), dan B (*blue*) menjadi citra S (hasil citra *grayscale*), maka konversi dapat menggunakan persamaan berikut :

$$S(x,y) = 0.299R(x,y) + 0.587G(x,y) + 0.114B(x,y) \quad [2.4]$$

Dimana : S = nilai matrik skala keabuan

 R = nilai matrik warna merah

 G = nilai matrik warna hijau

 B = nilai matrik warna biru

Berikut algoritma untuk mengkonversi citra asli ke dalam bentuk citra *grayscale* :

1. Ambil nilai matrik warna merah, hijau, dan biru dari tiap piksel citra asli.
2. Masukkan nilai matrik warna merah, hijau, dan biru, kemudian lakukan perhitungan menggunakan persamaan 2.4.
3. Tampilkan hasil operasi langkah ke-2 dalam tiap satu piksel citra.
4. Hasil proses langkah ke-3 berupa *output* hasil konversi (citra *grayscale*).

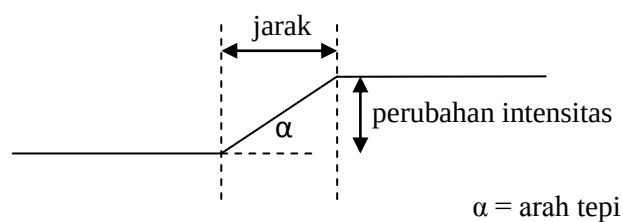
Hasil dari implementasi persamaan *grayscale* dapat dilihat pada gambar 2.5 yang menunjukkan perubahan citra warna menjadi citra yang hanya mempunyai derajat keabuan.



Gambar 2.5. Hasil Proses Konversi RGB ke Bentuk *Grayscale*

2.5 Pendeteksian Tepi

Tepi (*edge*) adalah perubahan nilai intensitas derajat keabuan yang mendadak (besar) dalam jarak yang singkat (Gambar 2.6). Perbedaan inilah yang menampakkan rincian pada gambar. Tepi biasanya terdapat pada batas antara dua daerah berbeda pada suatu citra. Tepi dapat diorientasikan dengan suatu arah, dan arah ini berbeda-beda tergantung pada perubahan intensitas.



Gambar 2.6. Model Tepi Satu Dimensi

2.5.1 Deteksi Tepi Unsur Pada Citra

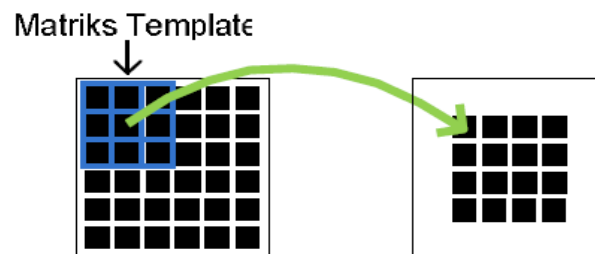
Keberadaan tepi unsur ditandai dengan tingginya perubahan nilai piksel atau kontras. Tepi unsur pada citra dideteksi dengan operator deteksi tepi diantaranya berupa matriks *template* berukuran tertentu, seperti pendeteksi tepi

Robert (terdiri dari matriks detektor tepi diagonal M^- dan M^+) serta pendeteksi tepi *Prewitt* dan *Sobel* (keduanya terdiri dari matriks template vertikal, M_x dan horizontal, M_y) seperti ditunjukkan gambar 2.7.

M^-	M^+	M_x	M_y	M_x	M_y																																												
<table><tr><td>1</td><td>0</td></tr><tr><td>0</td><td>-1</td></tr></table>	1	0	0	-1	<table><tr><td>0</td><td>1</td></tr><tr><td>-1</td><td>0</td></tr></table>	0	1	-1	0	<table><tr><td>1</td><td>0</td><td>-1</td></tr><tr><td>1</td><td>0</td><td>-1</td></tr><tr><td>1</td><td>0</td><td>-1</td></tr></table>	1	0	-1	1	0	-1	1	0	-1	<table><tr><td>1</td><td>1</td><td>1</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>-1</td><td>-1</td><td>-1</td></tr></table>	1	1	1	0	0	0	-1	-1	-1	<table><tr><td>1</td><td>0</td><td>-1</td></tr><tr><td>2</td><td>0</td><td>-2</td></tr><tr><td>1</td><td>0</td><td>-1</td></tr></table>	1	0	-1	2	0	-2	1	0	-1	<table><tr><td>1</td><td>2</td><td>1</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>-1</td><td>-2</td><td>-1</td></tr></table>	1	2	1	0	0	0	-1	-2	-1
1	0																																																
0	-1																																																
0	1																																																
-1	0																																																
1	0	-1																																															
1	0	-1																																															
1	0	-1																																															
1	1	1																																															
0	0	0																																															
-1	-1	-1																																															
1	0	-1																																															
2	0	-2																																															
1	0	-1																																															
1	2	1																																															
0	0	0																																															
-1	-2	-1																																															
Robert		Prewitt		Sobel																																													

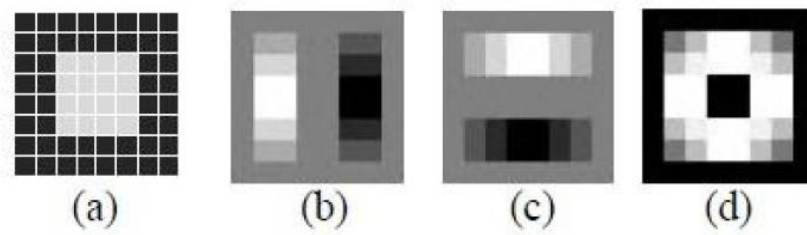
Gambar 2.7. Matriks *Template* Pendeteksi Tepi

Dalam mendeteksi tepi, dilakukan perkalian elementer antara matriks template pada gambar 2.7 di atas dengan kelompok piksel pada citra *input*. Pada hasil perkalian yang diperoleh, nilai setiap elemen dijumlahkan dan dijadikan nilai dari elemen matriks hasil konvolusi.



Gambar 2.8. Gambar Proses Konvolusi

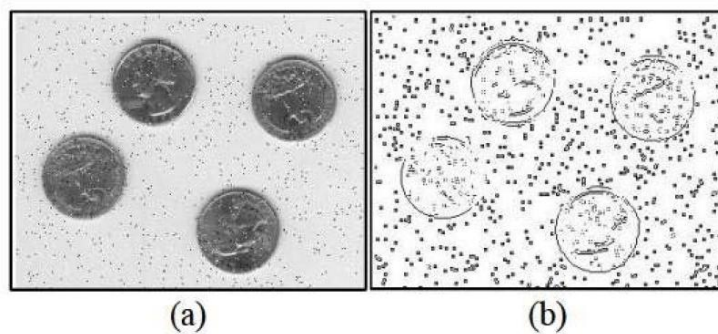
Deteksi tepi dengan metode *Prewitt* pada suatu citra biner berukuran 8x8 piksel (Gambar 2.9 (a)), akan menghasilkan citra konvolusi vertikal (M_y) (Gambar 2.9 (b)) dan citra konvolusi horizontal (M_x) (Gambar 2.9 (c)), citra hasil deteksi tepi keseluruhan (M) seperti ditunjukkan gambar 2.9 (d).



Gambar 2.9. Contoh Hasil Deteksi Tepi *Prewitt*

2.5.2 Deteksi Tepi *Canny*

Deteksi tepi dengan operator-operator di atas masih memiliki kelemahan, diantaranya pada satu perubahan nilai piksel akan diperoleh banyak piksel tepi seperti ditunjukkan pada gambar 2.9(d). Disamping itu, derau dan tekstur yang memiliki variasi nilai piksel yang cukup tinggi (tekstur kasar) pada suatu citra dapat juga terdeteksi sebagai tepi. Sebagai contoh, pada deteksi tepi dengan metode *Prewitt*, derau pada gambar 2.10(a), terdeteksi sebagai piksel tepi (gambar 2.10(b)), perhatikan gambar di bawah ini.

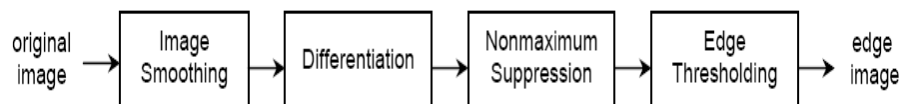


Gambar 2.10. Citra Dengan Derau Dan Hasil Deteksi Tepi *Prewitt*-nya

Upaya perbaikan terhadap hasil deteksi tepi telah banyak diteliti orang dengan mengembangkan berbagai metode, salah satu metode yang dikenal secara luas adalah deteksi tepi metode *Canny* yang memiliki kriteria sebagai berikut :

- Deteksi tepi optimal tanpa adanya kesalahan deteksi;
- Lokalisasi yang baik dengan jarak minimal antara tepi terdeteksi dengan posisi tepi sebenarnya;
- Respon tunggal terhadap tepi unsur.

Berdasarkan kriteria di atas, *Canny edge detection* merupakan salah satu pendeteksi tepi yang dikembangkan secara optimal. Algoritma *Canny edge detection* secara detail beroperasi sebagai berikut :



Gambar 2.11. Blok Diagram Deteksi Tepi *Canny*

Langkah-langkah dalam proses *Canny Edge Detection*, yaitu :

1. *Smoothing* dengan filter *Gaussian*

Langkah pertama yang harus dilakukan adalah melakukan *smoothing* pada citra dengan menggunakan filter *Gaussian*, yang berfungsi untuk menghilangkan *noise* pada citra sebelum dilakukan *edge detection*. Penggunaan filter *Gaussian* untuk melakukan *smoothing* dikarenakan kesederhanaan *mask*-nya. *Smoothing* menggunakan filter *Gaussian* ini dapat dilakukan dengan cara melakukan konvolusi yaitu menjalankan *mask* pada citra. *Mask* harus mempunyai ukuran lebih kecil daripada citra, semakin besar *mask Gaussian*, maka semakin tidak sensitif terhadap *noise*. *Mask Gaussian* ini sendiri dapat dirancang dengan menggunakan persamaan berikut :

$$g(x,y) = ce^{\frac{(x^2+y^2)}{2\sigma^2}} \quad [2.5]$$

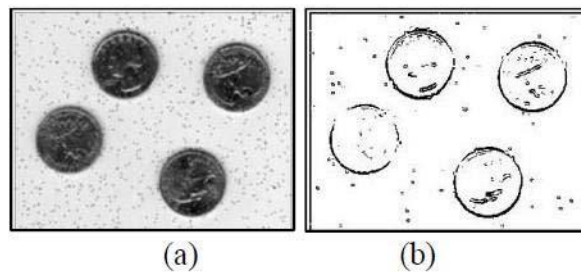
$$\sigma = 0.3 \left(\frac{(n-1)}{2} - 1 \right) + 0.8 \quad [2.6]$$

Dimana : c adalah konstanta normalisasi

σ adalah lebar fungsi *Gaussian*

n adalah ukuran kernel *Gaussian* (x,y)

Sebagai contoh, penerapan *filtering Gaussian* terhadap gambar 2.10(a) di atas akan menghasilkan citra terfilter yang ditunjukkan oleh gambar 2.12(a). Pada hasil deteksi tepi dengan metode *Prewitt* pada citra terfilter tersebut (gambar 2.12(b)), Nampak bahwa sebagian derau yang terdeteksi sebagai tepi berkurang.



Gambar 2.12. Hasil deteksi tepi pada citra terfilter

2. Penghitungan *magnitude* dari gradien yang didapat menggunakan operator Sobel

Setelah melakukan proses *smoothing* dan mengeliminasi *noise* yang ada pada citra, maka langkah selanjutnya adalah mencari gradien pada citra tersebut. Pencarian gradien dilakukan dengan cara melakukan konvolusi

pada citra, yaitu dengan menjalankan *mask* dari Sobel dalam dua arah, x untuk arah horizontal (G_x), dan y untuk arah vertikal (G_y).

-1	0	+1
-2	0	+2
-1	0	+1

G_x

+1	+2	+1
0	0	0
-1	-2	-1

G_y

Gambar 2.13. *Convolution Mask Operator Sobel*

Magnitude atau *edge strength* dari gradien-gradien yang didapatkan, dapat dihitung dengan persamaan berikut :

$$|G| = \sqrt{G_x^2 + G_y^2} \quad [2.7]$$

Dimana : G_x adalah gradien dari arah x

G_y adalah gradien dari arah y

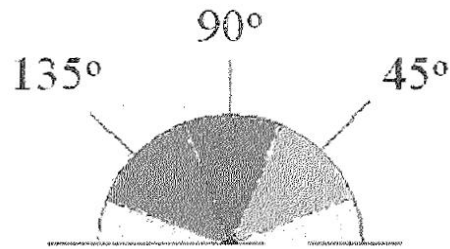
3. Pencarian *edge direction* (arah tepi)

Pencarian arah dari suatu *edge* sangat penting setelah gradien dari arah x dan arah y didapatkan. Pencarian *edge direction* dapat dihitung menggunakan persamaan berikut :

$$\theta = \tan^{-1} \frac{|G_y|}{|G_x|} \quad [2.8]$$

4. Melakukan normalisasi dari *edge direction*

Edge direction yang digunakan dalam penelitian ini ada 4, yaitu 0° , 45° , 90° , dan 135° .



Gambar 2.14. *Edge Direction*

Untuk *edge direction* yang jatuh dalam *range* $0^{\circ} - 22.5^{\circ}$ dan $157.5^{\circ} - 180^{\circ}$ akan dilakukan normalisasi menjadi 0° . Untuk *edge direction* yang jatuh dalam *range* $22.5^{\circ} - 67.5^{\circ}$ akan dilakukan normalisasi menjadi 45° . Untuk *edge direction* yang jatuh dalam *range* $67.5^{\circ} - 112.5^{\circ}$ akan dilakukan normalisasi menjadi 90° , dan untuk *edge direction* yang jatuh dalam *range* $112.5^{\circ} - 157.5^{\circ}$ akan dilakukan normalisasi menjadi 135° .

5. Melakukan proses *non-maximum suppression*

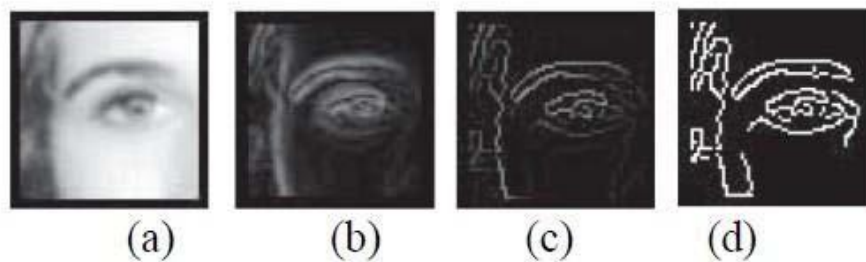
Setelah proses normalisasi dari *edge direction* selesai, kemudian dilakukan proses *non-maximum suppression*. Proses *non-maximum suppression* adalah melakukan penelusuran terhadap *region-region* gradien yang didapat dari setiap piksel, jika nilainya tidak maksimum akan dilakukan penurunan nilai pikselnya menjadi 0.

6. Melakukan pengurangan pada *array* gradien dengan menggunakan *Hysteresis Thresholding*

Proses histeresis digunakan untuk mengeliminasi *streakin* (garis-garis yang tebal), proses ini berguna untuk melakukan *thinning* pada citra, sehingga lebar tepi menjadi 1 piksel saja. Pada proses histeresis ini ada 2 buah nilai *threshold*, yaitu T1 (*threshold* nilai rendah) dan T2 (*threshold*

nilai tinggi). Setiap piksel tepi dengan nilai lebih kecil dari T_1 akan di-set menjadi 0 (bukan tepi), sebaliknya jika setiap piksel tepi dengan nilai lebih besar dari T_2 akan di-set menjadi 255 (termasuk tepi). Piksel yang *magnitude*-nya berada diantara T_1 dan T_2 akan di-set menjadi 0, kecuali jika ada jalan (*path*) dari titik tersebut yang memiliki *magnitude* di atas T_2 .

Hasil dari rangkaian proses deteksi tepi dengan metode *Canny* pada suatu citra, akan menghasilkan suatu citra biner yang terdiri dari piksel-piksel tepi tunggal, seperti pada gambar 2.15.



Gambar 2.15. Contoh Proses Deteksi Tepi *Canny*

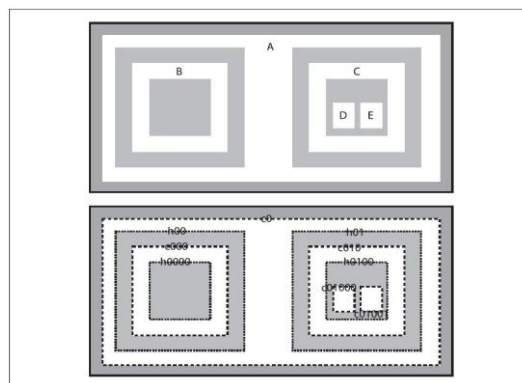
Keterangan : (a) Citra terfilter (c) Hasil *Non-maximal Suppression*
(b) Hasil deteksi tepi Sobel (d) Hasil *Hysteresis Thresholding*

2.6 *Contour*

Kontur (*contour*) adalah suatu kumpulan *point* atau titik yang didapatkan dari suatu komputasi atau perhitungan yang mewakili bentuk dari suatu batas objek pada suatu citra yang nantinya dihubungkan dengan garis dari setiap titik atau *point* yang telah didapat, sehingga menghasilkan kontur dari objek tersebut. Kontur biasa dipakai untuk melihat garis-garis batas dari suatu objek, mirip dengan tepi (*edge*) hanya saja kontur lebih detil, tegas dan jelas. Misalkan melakukan pencarian kontur terhadap suatu citra untuk melihat apakah ada kontur

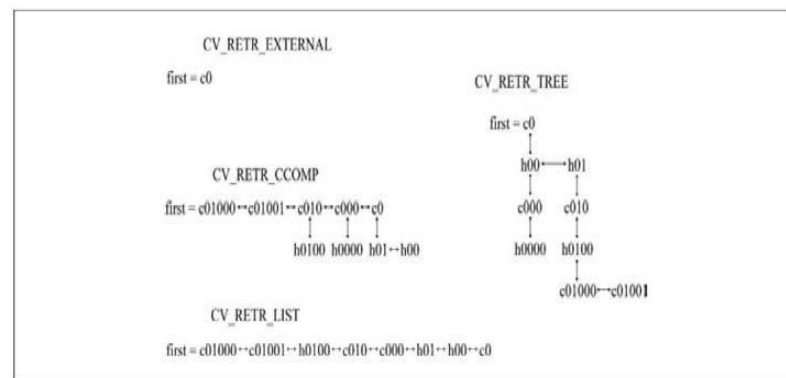
yang berbentuk, seperti sebuah mobil dalam suatu citra. Kontur didalam *openCV* membutuhkan suatu tempat memory khusus untuk menampung *sequence* yang berisikan titik atau *point-point* yang ditemukan. Maka di dalam *openCV* dibuat suatu struktur data yang merupakan *linkedlist* yang *sequential* tersendiri untuk menampung titik-titik kontur tersebut yang disebut dengan *sequence* atau disingkat **cvSeq** (*computer vision Sequential*).

Dalam pembuatan kontur diperlukan rutin untuk melakukan proses pencarian kontur. Citra yang dipakai adalah citra hasil konversi menjadi *grayscale* dan dikenakan *threshold* sehingga menghasilkan citra *bi-level*. Dalam gambar 2.16 dapat terlihat ilustrasi dari pencarian kontur. Terdapat gambar yang memiliki daerah putih dan abu-abu (dengan tingkat ketebalan yang berbeda) dan merupakan *background*, daerah putih diberi label A sampai E dengan *background* gelap. Di bagian gambar paling bawah terdapat hasil dari rutin pencarian kontur dan diberi label *cX* dan *hX* (*X* merupakan nilai biner iterator). Untuk label *cX* merupakan kontur dari setiap objek yang ada pada gambar sedangkan label *hX* merupakan *hole* atau lubang yang nantinya merupakan bagian yang dilingkari oleh kontur, yang didalam gambar kontur di perlihatkan sebagai garis yang putus-putus.



Gambar 2.16. Contoh Tes Pencarian Kontur Pada Suatu Citra

Dalam pencarian kontur ada 4 metode yang dapat digunakan dalam pengambilan kontur pada suatu citra yaitu *Retrieval External*, *Retrieval Components*, *Retrieval List*, dan *Retrieval Tree*.



Gambar 2.17. Bentuk Tipe *Contour*

1. *Retrieval External*

Pengambilan kontur yang didasarkan hanya pada kontur-kontur yang terlihat jelas dan berada di bagian luar objek yang ada dalam citra.

2. *Retrieval List*

Pengambilan seluruh kontur dan menyimpannya di dalam *list*. Tersimpan sebagai *linkedlist*.

3. *Retrieval Components*

Pengambilan seluruh kontur dan menyimpannya pada dua level hirarki dimana hirarki paling atas/ *top level boundaries* (kontur yang paling luar yang melingkari *hole*) adalah *external boundaries* dari komponen sedangkan level kedua adalah *hole boundaries*.

4. *Retrieval Tree*

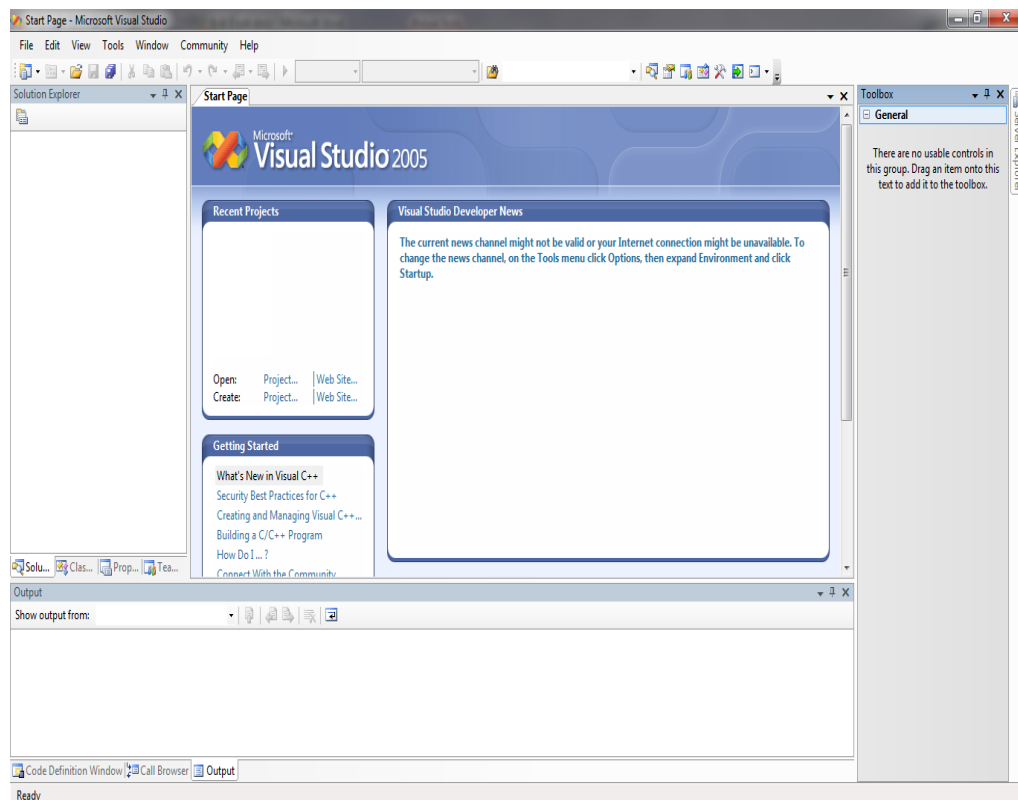
Pengambilan seluruh kontur dan merekonstruksikannya ke dalam hirarki (*full hierarchy*) menjadi kontur yang bersarang.

2.7 *Visual Studio 2005*

Microsoft Visual Studio merupakan sebuah perangkat lunak lengkap (*suite*) yang dapat digunakan untuk melakukan pengembangan aplikasi, baik itu aplikasi bisnis, aplikasi personal, ataupun komponen aplikasinya, dalam bentuk aplikasi *console*, aplikasi *Windows*, ataupun aplikasi *Web*. *Visual Studio* mencakup kompiler, SDK, *Integrated Development Environment* (IDE), dan dokumentasi (umumnya berupa *MSDN Library*). Kompiler yang dimasukkan ke dalam paket *Visual Studio* antara lain *Visual C++*, *Visual C#*, *Visual Basic*, *Visual Basic .NET*, *Visual InterDev*, *Visual J++*, *Visual J#*, *Visual FoxPro*, dan *Visual SourceSafe*.

Microsoft Visual Studio dapat digunakan untuk mengembangkan aplikasi dalam *native code* (dalam bentuk bahasa mesin yang berjalan di atas *Windows*) ataupun *managed code* (dalam bentuk *Microsoft Intermediate Language* di atas *.NET Framework*). Selain itu, *Visual Studio* juga dapat digunakan untuk mengembangkan aplikasi *Silverlight*, aplikasi *Windows Mobile* (yang berjalan di atas *.NET Compact Framework*).

Ketika pertama kali *Visual Studio 2005* diaktifkan, maka akan muncul tampilan seperti pada gambar berikut :



Gambar 2.18. Tampilan Awal *Visual Studio 2005*

Keterangan dari gambar di atas :

1. *Menubar*

Menubar berfungsi untuk memilih tugas-tugas tertentu, seperti memulai, membuka dan menyimpan *project*, mengompilasi *project* menjadi *file executable (exe)* dan lain-lain.

2. *Toolbar*

Toolbar memiliki fungsi yang sama seperti *menubar*, dan juga berfungsi seperti jalan pintas, karena lebih praktis dalam penggunaannya.

3. *Recent Project*

Jendela ini memuat proyek-proyek yang pernah dibuat dan disimpan, baik yang sudah selesai atau yang masih dalam tahap pengembangan.

4. *Getting Started*

Jendela ini memuat semacam tutorial untuk Visual C++ 2005.

5. *Visual Studio Developer News*

Pada jendela ini diperlihatkan berita-berita dari *Microsoft*. Biasanya berita tersebut adalah berita semacam *upgrade* baru, *patch*, *add-in*, atau berita lain mengenai *Microsoft* dan produknya.

6. *Solution Explorer*

Jendela ini baru aktif sewaktu ada proyek yang dibuka. *Solution* adalah semacam kumpulan *file* (biasanya berupa sebuah direktori) yang dibuat.

7. *Toolbox*

Jika ada proyek yang sedang aktif, maka *toolbox* akan berisi kontrol-kontrol yang dapat digunakan dalam pengembangan program aplikasi.

8. *Output*

Jendela ini menampilkan hasil kompilasi atau *build* dari proyek yang dibangun, dengan kata lain *output* yang diinginkan dari proses akhir suatu proyek dapat dilihat pada jendela ini.

2.8 Bahasa Pemrograman C++

C++ adalah modernisasi dan kelanjutan dari bahasa pemrograman C yang terkenal karena ketangguhannya di dunia pemrograman *windows*. Bahasa C++ biasanya digunakan untuk melakukan pemrograman yang menyangkut masalah *hardware* komputer atau sistem operasi dan banyak digunakan dalam bidang teknik atau pemrograman *game*. Bahasa C++ merupakan bahasa yang tangguh karena termasuk dalam kategori bahasa tingkat menengah, yaitu bahasa yang bisa

mengakses sistem operasi dan *hardware* secara langsung, seperti *assembly*, tetapi juga memiliki bahasa yang mudah dipahami. Hal itu tentu saja merupakan kombinasi yang sangat kuat dalam dunia pemrograman.

2.8.1 Sejarah Bahasa C++

Bahasa C adalah salah satu bahasa pemrograman komputer yang termasuk bahasa pemrograman tingkat menengah. Bahasa C dibuat pada tahun 1970-an untuk Sistem Operasi Unix oleh Bell Labs (Ken Thompson dan Dennis M. Ritchie) dan merupakan kelanjutan dari bahasa BCPL. Bahasa C merupakan salah satu bahasa pemrograman yang paling sering dipakai oleh pemrogram di seluruh dunia, karena C memperbolehkan pengakses memori secara manual.

C merupakan bahasa pemrograman terstruktur yang membagi program ke dalam sejumlah blok (sub program). Tujuannya adalah untuk memudahkan dalam pembuatan dan pengembangan program. Program yang ditulis dengan C mudah sekali dipindahkan dari satu jenis mesin ke mesin lain. Hal ini karena adanya standarisasi C yaitu ANSI (*American National Standards Institute*) yang menjadi acuan para pembuat *compiler* C. C++ diciptakan satu dekade setelah C. C++ diciptakan oleh Bjarne Stroustrup dari Bell Labs, AT&T pada tahun 1983. Pada awalnya C++ diberi nama “*A better C*”. Nama C++ sendiri diberikan oleh Rick Mascitti. Adapun tanda ++ berasal dari operator *increment* pada bahasa C.

Salah satu keistimewaan C++ adalah bahasa ini mendukung OOP (*Object Oriented Programming*). Tujuan utama pembuatan C++ adalah untuk meningkatkan produktivitas pemrogram dalam membuat aplikasi. Kebanyakan pakar setuju bahwa OOP dan C++ mampu mengurangi kompleksitas terutama

program yang terdiri dari 10.000 baris lebih, bahkan dapat meningkatkan produktivitas 2x lipat dari C, Pascal dan Basic.

2.8.2 Keunggulan Bahasa C++

Dibandingkan dengan bahasa pemrograman lain, C++ mempunyai beberapa kelebihan. Beberapa kelebihan bahasa C++ antara lain adalah:

1. Bahasa C++ tersedia hampir di semua jenis komputer.
2. Kode bahasa C++ dengan portabilitas dan fleksibilitas yang tinggi untuk semua jenis komputer.
3. Bahasa C++ hanya menyediakan sedikit kata-kata kunci (48 kata kunci).
4. Proses eksekusi program bahasa C++ lebih cepat. Dengan demikian, aplikasi yang dibuat dengan bahasa C++ akan menjadi aplikasi yang efisien dan kompetitif.
5. Dukungan pustaka fungsi dan kelas yang banyak sehingga memungkinkan pembuatan aplikasi makro.
6. C++ adalah bahasa yang terstruktur, dengan demikian akan lebih mendukung OOP.
7. Bahasa C++ termasuk bahasa tingkat menengah dan lebih dekat dengan bahasa mesin.
8. Kode program bersifat *reuseable*, sehingga dapat digunakan kembali pada *project* lain dengan hanya menggunakan *library* dan *file header*.
9. C++ dapat membuat aplikasi *graphic processor* berkualitas tinggi

2.9 *OpenCV Library*

OpenCV adalah suatu *library* gratis yang dikembangkan oleh *developer-developer Intel Corporation*. *Library* ini terdiri dari fungsi-fungsi *computer vision* dan API (*Application Programming Interface*) untuk *image processing*, baik *high level* maupun *low level* dan sebagai optimasi aplikasi *realtime*. OpenCV sangat disarankan untuk programmer yang akan berkecukupan pada bidang *computer vision*, karena *library* ini mampu menciptakan aplikasi yang handal, kuat dibidang *digital vision*, dan mempunyai kemampuan yang mirip dengan cara pengolahan visual pada manusia. Saat ini para *developer* dari *Intel Corporation* telah membuat berbagai macam versi, yaitu:

- *OpenCV* untuk bahasa pemrograman C/C++,
- *OpenCV* untuk bahasa pemrograman C# (masih dalam tahap pengembangan), dan
- *OpenCV* untuk bahasa pemrograman *Java*.

Untuk bahasa pemrograman C# dan *Java*, karena masih dalam tahap pengembangan, maka kita membutuhkan *library* lain sebagai pelengkap kekurangan yang ada.

2.9.1 *Fitur OpenCV*

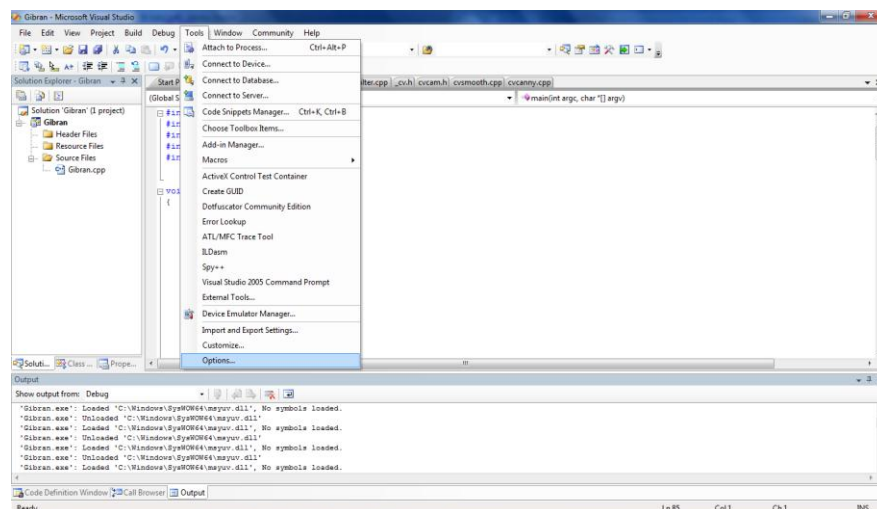
Berikut ini adalah fitur-fitur pada *library* OpenCV :

- Manipulasi data gambar (alokasi memori, melepaskan memori, *copy* gambar, *setting* serta konversi gambar)
- *Image/Video I/O* (bisa menggunakan kamera yang sudah didukung oleh *library* ini)

- Manipulasi matriks dan vector serta terdapat juga *routines linear algebra (products, solvers, eigenvalues, SVD)*
- *Image processing* dasar (*filtering, edge detection* (pendeteksian tepi), *sampling* dan *interpolasi*, konversi warna, operasi morfologi, histogram, *image pyramids*)
- Analisis struktural
- Kalibrasi kamera
- Pendeteksian gerak
- Pengenalan objek
- *Basic GUI* (*Display gambar/video, mouse/keyboard kontrol, scrollbar*)
- *Image Labelling* (*line, conic, polygon, text drawing*)

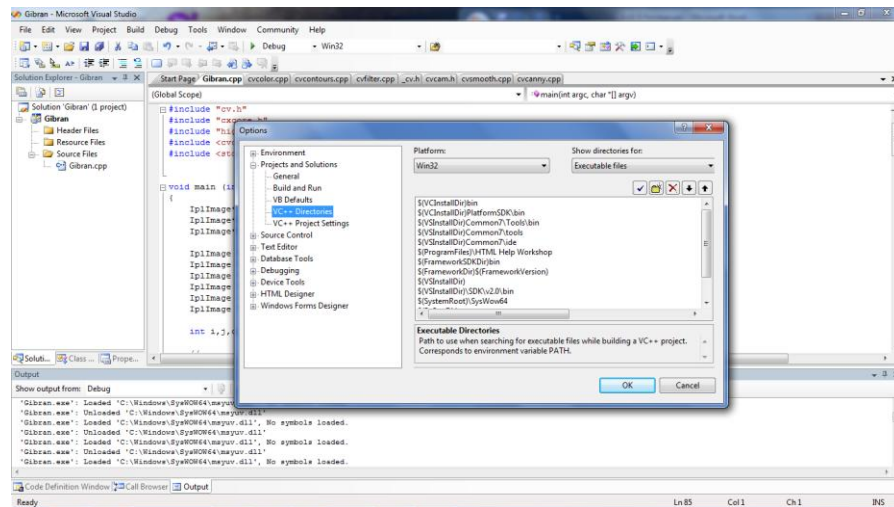
2.9.2 Setting OpenCV Library

Pada awalnya *software Visual Studio 2005* yang telah terpasang di dalam komputer, belum terhubung dengan *OpenCV library*. Langkah-langkah berikut merupakan *setting library OpenCV* pada *Visual Studio 2005* :



Gambar 2.19. Klik *Tools* pilih *Options*

Setelah itu akan muncul jendela berikut :



Gambar 2.20. Klik VC++ Directories

Setelah itu pilih *Include file* pada box “*Show directories for:*”, kemudian tambahkan alamat berikut :

C:\Program Files\OpenCV\cvaux\include

C:\Program Files\OpenCV\cxcore\include

C:\Program Files\OpenCV\cv\include

C:\Program Files\OpenCV\otherlibs\highgui

C:\Program Files\OpenCV\otherlibs\cvcam\include

C:\Program Files\OpenCV\ml\include

Ubah *Include files* menjadi *Library files*, kemudian tambahkan alamat berikut :

C:\Program Files\OpenCV\lib

Ubah *Library files* menjadi *Source files*, kemudian tambahkan alamat berikut :

C:\Program Files\OpenCV\cv\src

C:\Program Files\OpenCV\cvaux\src

C:\Program Files\OpenCV\cxcore\src

C:\Program Files\OpenCV\otherlibs\highgui

C:\Program Files\OpenCV\otherlibs\cvcam\src\windows

Setelah *setting* di atas dilakukan, kini *library OpenCV* telah dihubungkan dengan Visual Studio 2005. Langkah selanjutnya mengatur *library* tersebut ke dalam suatu *project C++* untuk menentukan *library* mana saja yang akan dibutuhkan saat *project* dijalankan. Langkah pertama, klik kanan pada *project files* yang telah dibuat sebelumnya, kemudian klik *Properties*. Pilih tab *Linker* dan kategori *Input*, kemudian tambahkan modul ***cv.lib cxcore.lib cvaux.lib highgui.lib ml.lib*** ke dalam *Additional Dependencies*. Kini *library OpenCV* dapat digunakan dalam *project* yang akan dikembangkan.

2.10 Sistem Real Time

Pengertian sistem *real time* secara umum merupakan suatu ukuran kinerja suatu sistem tertentu yang mana menyangkut batasan kinerja sistem dan tahapan perancangan sistem. Dalam sebuah sistem *real time* yang mana memiliki permasalahan waktu harus nyata, maksudnya harus memiliki persyaratan pewaktuan yang sesuai dengan keadaan nyata yang dijadikan parameter dalam sistem itu.

Untuk mendapatkan hasil penangkapan yang *real time*, secara umum media yang digunakan adalah kamera digital. Sebuah kamera digital mampu menjangkau hingga resolusi tertentu. Biasanya resolusi yang mampu dijangkau mencapai 320x460 sampai 768x1024 piksel. Kemampuan kamera menangkap gambar utuh (*frame*) setiap detiknya sangatlah signifikan untuk menghasilkan

video yang halus. Secara umum kamera dapat dikatakan *real time* jika mampu menangkap paling tidak 25 gambar utuh (*frame*) setiap detiknya. Ada dua jenis kamera yang bisa digunakan, tipe *Webcam* atau kamera *video* yang dihubungkan dengan *port* USB dan tipe VCR (*Video Camera Recorder*) atau kamera *video* digital yang dihubungkan dengan *port firewire*.



Gambar 2.21. Contoh *Webcam*

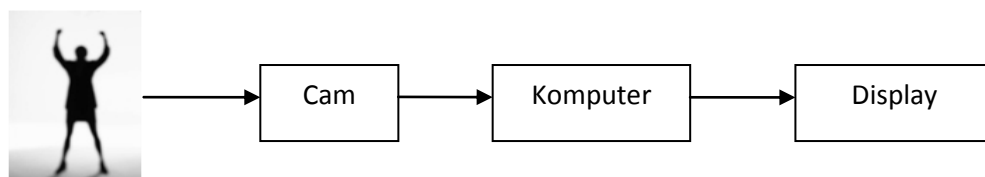
BAB III

PERANCANGAN SISTEM

Pada bab ini akan dijelaskan perencanaan dan pembuatan perangkat lunak secara keseluruhan. Perangkat lunak yang digunakan disini tidak lepas dari studi yang berkaitan dengan pengolahan citra dalam bidang komputer, seperti yang dijelaskan pada bab sebelumnya.

3.1 Perancangan Sistem

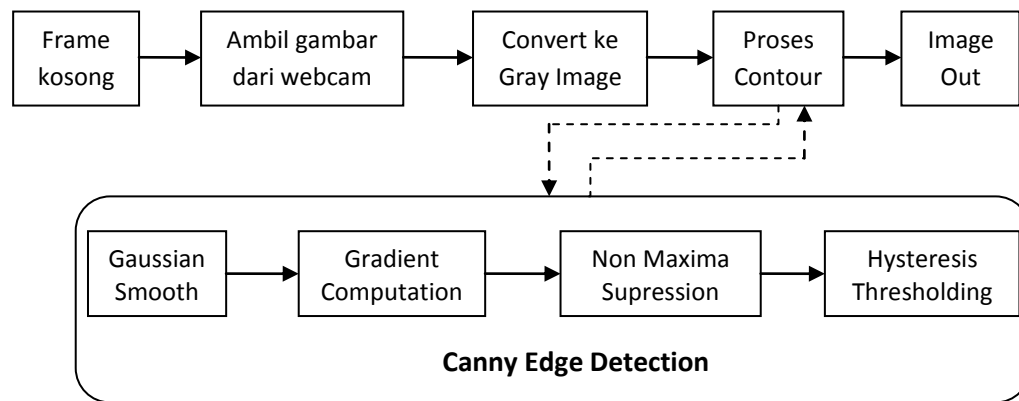
Pengolahan citra bertujuan memperbaiki kualitas citra agar mudah diinterpretasi oleh manusia dan mesin (dalam hal ini komputer). Teknik-teknik pengolahan citra mentransformasikan citra menjadi citra lain, seperti terlihat pada gambar 3.1. Sebelum sistem diimplementasikan, terlebih dahulu dibuat sistematika perancangan yang merepresentasikan keseluruhan sistem agar didapatkan hasil yang maksimal.



Gambar 3.1 Gambaran Umum Sistem

Suatu objek (benda) yang bergerak dapat disimpan dan ditampilkan (*display*) menggunakan komputer dalam bentuk data berupa citra, sehingga benda yang ditangkap oleh kamera (*cam*) dapat terlihat seperti aslinya. Citra tersebut dapat diolah menggunakan komputer agar menghasilkan citra baru yang dapat

digunakan untuk berbagai keperluan analisa. Proses pengolahan inilah yang disebut dengan pengolahan citra.



Gambar 3.2 Blok Diagram Sistem

Blok diagram di atas menegaskan kembali proses pengolahan citra di dalam komputer, dimana sebuah frame kosong diisi dengan data berupa citra yang ditangkap melalui sebuah kamera (*cam*). Citra tersebut pada awalnya merupakan citra berwarna (citra *Red Green Blue* (RGB)) yang kemudian di konversi menjadi citra *Grayscale* dengan 255 derajat keabuan untuk memudahkan dalam proses selanjutnya. Pembentukan *contour* dilakukan dengan salah satu metode deteksi tepi paling optimal saat ini, yaitu deteksi tepi canny. *Gaussian smooth* adalah salah satu filter untuk mereduksi *noise* pada gambar yang paling sering digunakan. *Gradien computation* adalah metode yang digunakan untuk menentukan tepi dari sebuah gambar. Beberapa operator yang digunakan untuk pendeteksian tepi obyek adalah operator Robert, Sobel dan Prewitt. *Non maxima supression* adalah algoritma untuk menjadikan tepi dari obyek menjadi satu *pixel*. *Hysteresis Thresholding* adalah algoritma yang menggunakan dua nilai batas, yaitu *thigh* dan *tlow*. Jika *pixel* yang sedang diproses nilainya lebih kecil dari *tlow*,

pixel tersebut diberi tanda sebagai bukan tepian dari suatu objek. Metode ini untuk menentukan piksel-piksel yang benar-benar tepi yang boleh ada, sedangkan yang lainnya harus dihapus.

Citra keluaran (*image out*) merupakan citra baru setelah diolah dengan metode di atas, kemudian langsung ditampilkan pada layar. Citra baru ini akan menampilkan objek-objek mana saja yang sedang bergerak dengan mengamati perubahan *contour* dari benda yang bergerak tersebut secara *realtime* (video).

3.2 Proses Pengambilan Gambar

Video merupakan kumpulan beberapa *frame* gambar yang ditangkap sebuah kamera dengan kecepatan sepersekian detik. Mata manusia tidak dapat menangkap *frame* dengan kecepatan yang tinggi ini, melainkan menangkapnya sebagai rangkaian yang kontinyu (*video*). Pada pembahasan ini akan dipaparkan bagaimana proses pengambilan gambar oleh sebuah kamera dan penyediaan *window* untuk menampilkannya. Pada penjelasan sebelumnya telah dijelaskan untuk proses komunikasi hingga didapatkan objek akan menggunakan *library* OpenCV yang telah tersedia secara gratis.

Modul-modul pada OpenCV dalam bentuk *Header File* antara lain :

- a. `#include "cv.h"` // fungsi-fungsi utama OpenCV,
- b. `#include "cxcore.h"` // dukungan struktur data dan aljabar linear,
- c. `#include <cvcam.h>` // fungsi-fungsi yang berkaitan dengan kamera,
- d. `#include "highgui.h"` // fungsi-fungsi yang menangani masalah GUI.

3.2.1 Proses Pembuatan *Window*

Window disini digunakan sebagai wadah untuk menempatkan gambar-gambar (*frame*) yang akan ditangkap oleh kamera agar dapat ditampilkan. Berikut fungsi untuk inisialisasi kamera dan *window* :

```
=====
cvNamedWindow("ASLI", CV_WINDOW_AUTOSIZE);..... (1)
cvNamedWindow("detection", CV_WINDOW_AUTOSIZE);..... (2)
cvNamedWindow("gray", CV_WINDOW_AUTOSIZE);..... (3)
cvNamedWindow("gaussian", CV_WINDOW_AUTOSIZE);..... (4)

CvCapture* cv_cap=cvCaptureFromCAM(-1);..... (5)
newImg = cvQueryFrame(cv_cap);..... (6)
=====
```

Rincian informasi dari coding di atas dijelaskan sebagai berikut :

1. Inisialisasi *window* “ASLI”, untuk menampilkan *frame* asli dari kamera.
2. Inisialisasi *window* “detection”, untuk menampilkan *frame* hasil contour (*output image*).
3. Inisialisasi *window* “gray”, untuk menampilkan gambar hasil konversi dari citra RGB menjadi *grayscale* (keabuan).
4. Inisialisasi *window* “gaussian”, untuk menampilkan gambar hasil dari proses penghalusan citra (*smooth*) menggunakan filter Gaussian.
5. Inisialisasi kamera, mengambil gambar dari kamera dan ditampung ke *cv_cap*.
6. Inisialisasi variabel *newImg*.

3.2.2 Penempatan Dan Tipe Gambar

Gambar-gambar (*frame*) yang telah ditangkap (*capture*) kamera diolah lalu ditempatkan pada *window*-nya. Berikut sebagian fungsi inisialisasi tipe dan penempatan gambar untuk keperluan pengolahan citra :

```
=====
IplImage*newImg = 0;..... (1)

gray1=cvCreateImage(cvSize(newImg->width,newImg->height),
IPL_DEPTH_8U,1);..... (2)

gray1->origin = newImg->origin;..... (3)
=====
```

Rincian informasi dari *coding* di atas dijelaskan sebagai berikut :

1. Dalam OpenCV, gambar disimpan dalam struktur *IplImage*, jadi pointer ditetapkan untuk gambar yang awalnya diatur dengan nol.
2. Pengalokasian memori untuk gambar dengan menggunakan *cvCreateImage* dalam *gray1*. Parameter pertama adalah ukuran gambar yang sama seperti gambar aslinya, yang kedua adalah kedalaman gambar, dan ketiga adalah jumlah saluran. Gambar disini memiliki kedalaman 8-bit dengan 1 saluran.
3. *Origin* dimaksudkan proses pengolahan dimulai dari piksel bagian atas-kiri dari citra.

3.3 Konversi Gray Image

Citra yang telah ditempatkan ke masing-masing *image header* selanjutnya diolah sesuai dengan nama *image header* yang diberikan, contohnya *image header* “gray1”, dimaksudkan untuk mengolah citra tersebut menjadi citra *grayscale* (skala keabuan). Citra yang ditangkap oleh kamera biasanya berformat RGB (Red

Green Blue). Citra warna tersebut dikonversi menjadi citra dengan skala keabuan untuk mempermudah proses pendeteksian tepinya. Persamaan di bawah ini dapat digunakan sebagai konversi dari citra warna menjadi citra skala keabuan :

$$Y = 0,299R + 0,587G + 0,114B \quad [3.1]$$

dimana Y adalah nilai kecerahan pada citra abu-abu, sedangkan R, G, dan B adalah nilai warna merah, hijau, dan biru yang masing-masing didiskritkan dalam skala 256.

Berikut fungsi inisialisasi untuk mengkonversi citra warna menjadi citra skala keabuan :

```
=====
cvCvtColor( newImg, gray1, CV_BGR2GRAY );
=====
```

Citra newImg dikonversi menjadi citra skala keabuan dan disimpan di gray1.

3.4 Pendeteksian Tepi *Canny*

Pada pembahasan sebelumnya telah dibahas algoritma pendeteksian tepi canny dioperasikan dalam 4 tahapan, yaitu :

1. Penghalusan Citra (*Smoothing*),
2. Perhitungan gradient citra,
3. *Non-maximum supression*,
4. *Hysteresis Thresholding*.

3.4.1 Penghalusan Citra (*Smoothing*)

Citra hasil konversi tersebut selanjutnya dilakukan penghalusan citra menggunakan *Gaussian filter* untuk mengurangi noise dan menghasilkan citra yang lebih halus. Fungsi di bawah ini digunakan untuk merancang kernel Gaussian (perhitungan lengkapnya dapat dilihat pada bab selanjutnya) :

$$G(x, y) = c e^{-\frac{(x^2 + y^2)}{2\sigma^2}} \quad [3.2]$$

$$\sigma = 0.3 \left(\frac{(n-1)}{2} - 1 \right) + 0.8 \quad [3.3]$$

dimana : c adalah konstanta normalisasi

σ (*sigma*) adalah lebar dari fungsi Gaussian

n adalah ukuran kernel gaussian (x, y)

Berikut inisialisasi untuk *smoothing* (penghalusan citra) dengan merancang kernel *gaussian filter* :

```
=====
cvSmooth(gray1, gaussian1, CV_GAUSSIAN, 7, 7);
=====
```

Citra gray1 dihaluskan menggunakan gaussian filter dengan ukuran kernel ($n = 7 \times 7$) dan dialokasikan dalam gaussian1.

Pada inisialisasi di atas, ukuran kernel yang digunakan untuk merancang kernel Gaussian ini adalah 7 ($n=7$). Nilai σ sendiri dapat diketahui dengan menggunakan persamaan 3.3 di atas, yaitu 1,4 ($\sigma = 1,4$). Kedua nilai ini kemudian dimasukkan ke dalam persamaan 3.2, dan menghasilkan matriks seperti pada tabel di bawah ini :

Tabel 3.1. Tabel kernel Gaussian filter dengan $c = 1$

(x,y)	-3	-2	-1	0	1	2	3
-3	0.0101	0.0362	0.0780	0.1006	0.0780	0.0362	0.0101
-2	0.0362	0.1299	0.2792	0.3604	0.2792	0.1299	0.0362
-1	0.0780	0.2792	0.6003	0.7748	0.6003	0.2792	0.0780
0	0.1006	0.3604	0.7748	1.000	0.7748	0.3604	0.1006
1	0.0780	0.2792	0.6003	0.7748	0.6003	0.2792	0.0780
2	0.0362	0.1299	0.2792	0.3604	0.2792	0.1299	0.0362
3	0.0101	0.0362	0.0780	0.1006	0.0780	0.0362	0.0101

Akan tetapi dalam penerapan suatu filter yang dibutuhkan adalah filter dengan nilai-nilai pembobotan yang bulat untuk memudahkan perhitungan dalam komputer, sebab intensitas piksel dalam citra digital diwakili oleh bilangan bulat (integer). Untuk itu, nilai pada salah satu sudut atau nilai pecahan terkecil dalam matriks yang terbentuk diambil dan digunakan untuk menentukan konstanta c sehingga nilai elemen matriks pada sudut tersebut berubah menjadi satu. Dari contoh di atas, didapatkan :

$$\frac{G(3,3)}{c} = e^{-\frac{(3^2 + 3^2)}{2(1.4)^2}} = 0.0101 \Rightarrow c = \frac{G(0,0)}{0.0101} = \frac{1.0000}{0.0101} = 99 \quad [3.4]$$

Sekarang tinggal mengalikan matriks tadi dengan konstanta c yang baru didapat, dan menghasilkan matriks baru yang dapat dilihat pada tabel berikut :

Tabel 3.2. Tabel kernel Gaussian filter dengan $c = 99$

(x,y)	-3	-2	-1	0	1	2	3
-3	1	4	8	10	8	4	1
-2	4	13	28	36	28	13	4
-1	8	28	60	77	60	28	8
0	10	36	77	99	77	36	10
1	8	28	60	77	60	28	8
2	4	13	28	36	28	13	4
3	1	4	8	10	8	4	1

Tabel di atas adalah hasil akhir dari pembentukan filter Gaussian yang merupakan jendela konvolusi yang akan dikonvolusikan pada citra. Hasil dari proses konvolusi ini dapat dipastikan bahwa nilainya menjadi sangat besar, jauh lebih besar dari 255 sebagai nilai intensitas tertinggi dalam citra dengan format data 8-bit, jadi masih harus dinormalkan kembali. Nilai piksel hasil perhitungan dapat dinormalkan dengan cara membaginya dengan jumlah semua elemen nilai pembobot pada filter. Dari tabel di atas, pembagiannya adalah :

$$\sum_{x=-3}^3 \sum_{y=-3}^3 G(x, y) = 1203 \quad [3.5]$$

Dengan demikian nilai baru hasil perhitungan menjadi :

$$H(x, y) = \frac{1}{1203} (f(x, y) * G(x, y)) \quad [3.6]$$

3.4.2 Perhitungan Potensi Gradien Citra

Gradien merupakan operator yang paling mendekati definisi dari sebuah tepi. Pada langkah menghitung potensi gradien citra ada dua buah informasi yang dibutuhkan yaitu kekuatan *edge* (*edge strength/magnitude*), dan arah *edge* (*edge direction/orientation*). Citra hasil gaussian di atas kemudian dikonvolusi dengan operator Sobel yang memanfaatkan dua buah *template edge* pada dua arah tegak lurus, yakni horizontal (Gx), dan vertikal (Gy). Kernel dari operator Sobel umumnya berukuran 3x3 *convolution mask*, satu untuk menaksir gradien di arah sumbu x (kolom), dan yang lainnya menaksir gradien di arah sumbu y (baris). Kernel yang digunakan pada operator sobel ini dapat dilihat pada gambar berikut :

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Gambar 3.3 Convolution Mask Operator Sobel

Citra yang telah dikonvolusi dengan dua operator di atas, kemudian dilakukan penghitungan gradiennya untuk menentukan kekuatan tepiannya. Besarnya gradien (kekuatan tepi) dapat ditentukan dengan menggunakan persamaan berikut:

$$|G| = \sqrt{Gx^2 + Gy^2} \quad [3.7]$$

Arah tepi (*edge orientation*) dapat ditentukan dengan menggunakan persamaan berikut :

$$\theta = \tan^{-1} \frac{|Gy|}{|Gx|} \quad [3.8]$$

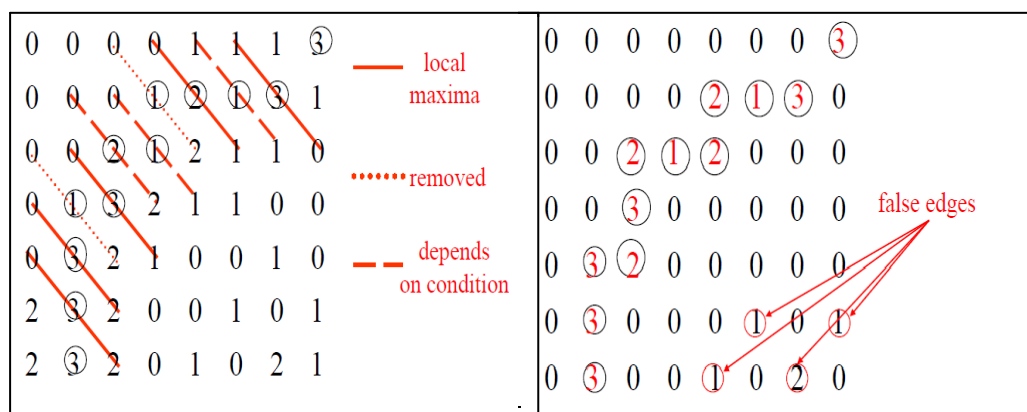
dimana : Gx = Gradient filter koordinat X

Gy = Gradient filter koordinat Y

Pada persamaan *edge orientation* diatas, akan terjadi kesalahan jika nilai dari gradien dalam arah x sama dengan nol, sehingga diperlukan pembatasan dalam kode jika hal ini terjadi. Pada saat nilai $Gx = 0$, arah tepi (θ) harus sama dengan 90^0 atau 0^0 , tergantung dari nilai gradien dalam arah y (Gy). Apabila Gy memiliki nilai sama dengan 0, arah tepi akan sama dengan 0^0 , dan jika tidak maka arah tepi akan sama dengan 90^0 .

3.4.3 Non Maximum Suppression

Proses *Non Maximum Suppression* adalah penelusuran terhadap region-region gradient yang didapat dari setiap piksel, jika nilainya tidak maksimum akan dilakukan penurunan nilai pikselnya menjadi 0. *Non maximum suppression* ini bertujuan membuang potensi gradient di suatu piksel dari kandidat *edge* jika piksel tersebut bukan merupakan maksimal lokal pada arah *edge* di posisi tersebut (disinilah arah gradient diperlukan). Gambaran umum dari proses ini dapat ditunjukkan pada gambar di bawah ini :



Gambar 3.4 Proses Non Maximum Suppression

Gambar di atas menunjukkan proses *non maximum suppression* ini sangat mirip dengan proses *thinning* (perampingan). Perbedaan dari kedua proses tersebut dapat dilihat dari jumlah *threshold* yang digunakan, jika proses *thinning* hanya menggunakan satu *threshold* saja, sedangkan *non maximal suppression* menggunakan dua *threshold* yang dikenal dengan *Hysteresis Thresholding*.

3.4.4 *Hysteresis Thresholding*

Proses *hysteresis* digunakan untuk mengeliminasi garis-garis tebal (*streaking*). Proses ini berguna untuk melakukan *thinning* pada citra, sehingga lebar tepi menjadi 1 pixel aja. Pada proses *hysteresis* ini ada dua buah nilai *threshold* yaitu T1 (*threshold* nilai rendah) dan T2 (*threshold* nilai tinggi). Setiap piksel tepi dengan nilai lebih kecil dari T1 akan di-set menjadi 0 (bukan tepi), sebaliknya jika setiap piksel tepi dengan nilai lebih besar dari T2 akan di-set menjadi 255 (termasuk tepi). Piksel yang *magnitude*-nya berada diantara T1 dan T2 akan di-set menjadi 0, kecuali jika ada jalan (*path*) dari titik tersebut yang memiliki *magnitude* di atas T2.

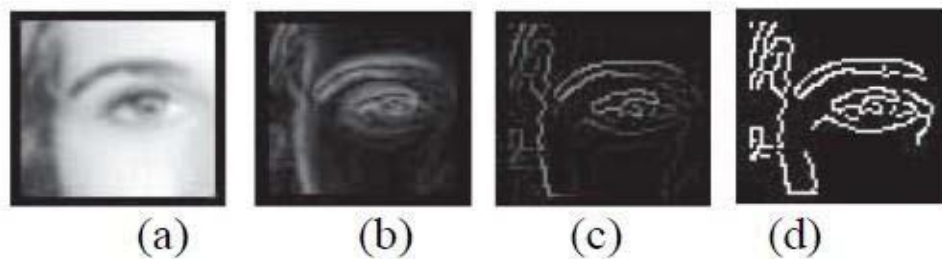
3.4.5 *Coding Deteksi Tepi Canny Pada Visual C++*

Coding untuk inisialisasi keseluruhan proses Canny ini terdapat pada salah satu *library* dari **OpenCV**. Berikut inisialisasi yang digunakan dalam proses deteksi tepi Canny :

```
=====
cvCanny(gaussian1, cannyImg, 50, 150, 3);
=====
```

Citra masukan *gaussian1* akan di deteksi tepiannya dengan menggunakan metode Canny, hasilnya disimpan menjadi *cannyImg*. Thresholding yang digunakan dalam proses ini ada dua, yaitu T1 = 50 (*low-thresholding*) dan T2 = 150 (*high-thresholding*). Parameter ketiga dari coding tersebut menunjukkan nilai *aperture size* yang digunakan dalam operator Sobel, dengan kata lain kernel Sobel yang digunakan berukuran 3x3.

Hasil dari rangkaian proses deteksi tepi dengan metode *Canny* pada suatu citra biner yang terdiri dari piksel-piksel tepi tunggal seperti terlihat pada gambar berikut :



Gambar 3.5 Contoh Proses Deteksi Tepi Canny

Keterangan : (a) Citra terfilter (c) Hasil *Non Maximal Suppression*
(b) Hasil deteksi tepi Sobel (d) Hasil *Hysteresis Thresholding*

3.5 Proses Deteksi *Contour*

Keluaran citra dari proses deteksi tepi menggunakan metode Canny di atas merupakan citra biner, dimana piksel yang bernilai 0 merupakan piksel bukan tepi dan 255 merupakan piksel tepi. Piksel-piksel pada citra tersebut banyak sekali ditemukan *gap* yang mengakibatkan tepian dari objek terputus. Hal ini terjadi kemungkinan dikarenakan perbedaan piksel yang tipis antara piksel *background* dengan *foreground*-nya, sehingga piksel yang seharusnya merupakan tepian dari objek tidak terdeteksi dengan baik oleh sistem. Kemungkinan kedua adalah adanya tepian dari bukan objek (objek lain dari *background*) yang terdeteksi oleh sistem sebagai tepi dari objek (*foreground*).

Sesuai dengan sifat *contour* yang tidak pernah bercabang dan selalu menutup bentuknya antara piksel tepi yang satu dengan yang lainnya, maka proses selanjutnya adalah proses pendeteksian *contour* dari citra *cannyImg* di atas. Berikut inisialisasi dalam proses mendeteksi *contour* :

```

=====
CvMemStorage * storage = cvCreateMemStorage(0); ..... (1)
CvSeq * contour = 0; ..... (2)
int mode = CV_RETR_CCOMP; ..... (3)

cvFindContours(cannyImg, storage, &contour, sizeof(CvContour),
mode, CV_CHAIN_APPROX_SIMPLE, cvPoint(0,0)); ..... (4)
=====

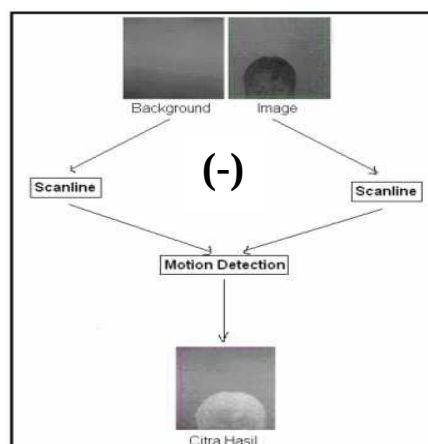
```

Berikut penjelasan dari coding di atas :

1. Penyimpanan memori sementara dialokasikan dalam **storage**.
2. Inisialisasi urutan *contour* tidak ditentukan (*default*).
3. *Mode* pengambilan *contour* dua sisi yaitu dari luar dan dalam.
4. **cannyImg** sebagai sumber citra, wadah *contour* yang telah diambil disimpan sementara di **storage**, prosesnya pertama kali diambil dari luar terlebih dahulu, ukuran *contour* yang diambil, mode pengambilan *contour*, metode pendekatan *contour* yang digunakan dengan menekan segmen secara horizontal, vertikal, dan diagonal (**CV_CHAIN_APPROX_SIMPLE**), titik *offset* yang menunjukkan proses *contour* dimulai dari titik (0,0).

3.6 Proses Motion Detection

Langkah – langkah dari *motion detection* secara umum adalah :



Gambar 3.6 Motion Detection

1. Memasukkan gambar pertama sebagai *background*.
2. Memasukkan gambar kedua sebagai gambar yang ada objeknya.
3. Melakukan *scanline* pada kedua gambar tersebut. *Scanline* dilakukan dari kiri atas gambar sampai dengan kanan bawah gambar.
4. Lakukan perbandingan antara gambar *background* dengan gambar satunya lagi untuk mengetahui adanya perubahan atau adanya perbedaan antara gambar kesatu dan gambar kedua.
5. Setelah melakukan proses diatas maka akan dihasilkan citra hasil *motion detection*. Objek yang tampak merupakan objek yang mengalami pergerakan.

3.6.1 Proses Pengambilan Dan Pengolahan Gambar

Citra yang telah diproses di atas merupakan *frame background* dari citra yang akan diambil pada langkah berikutnya. Citra yang diambil saat ini akan menunjukkan ada atau tidaknya objek yang bergerak dalam suatu ruangan. Sebelumnya citra ini akan diolah terlebih dahulu sama halnya dengan pengolahan pada citra sebelumnya. Berikut inisialisasi pengambilan dan pengolahan citra yang nantinya akan dibandingkan dengan *frame* sebelumnya untuk proses mendeteksi objek yang bergerak :

```
=====
for (;;) {
    newImg=cvQueryFrame(cv_cap);..... (1)
    if (!newImg) ..... (2)
    break; ..... (3)
    cvCvtColor(newImg,gray2, CV_BGR2GRAY);..... (4)
    cvSmooth(gray2, gaussian2, CV_GAUSSIAN, 7, 7);..... (5)
    cvCopy(sekarang,sebelum,0); ..... (6)
    cvCopy(gaussian2,sekarang,0); ..... (7)
}
=====
```

Berikut penjelasan *coding* di atas :

1. Pengambilan gambar dan inisialisasi variabel `newImg`.
2. Kondisi jika kamera tidak terdeteksi.
3. *Output* jika kondisi nomor 3 terpenuhi.
4. Konversi **newImg** citra RGB menjadi *grayscale* dan ditampung di **gray2**.
5. *Smoothing* **gray2** dengan kernel gaussian 7x7 ditampung di **gaussian2**.
6. Menyalin citra **sekarang** ke citra **sebelum**.
7. Menyalin citra **gaussian2** ke citra **sekarang**.

3.6.2 Perhitungan Perbedaan *Image*

Pada *frame* 1 dan *frame* 2 terdapat perbedaan jika terdapat objek yang sedang bergerak. Perbedaan ini dapat ditentukan dengan membandingkan kedua *frame* tersebut, sehingga citra yang terbentuk merupakan objek yang sedang bergerak saja. Salah satu metode yang digunakan untuk membedakan kedua *frame* tersebut adalah dengan menggunakan metode **cvAbsDiff** yang terdapat di dalam *library* OpenCV. Berikut inisialisasi dari fungsi tersebut :

```
=====
for (i=0; i<newImg->height;i++) for (j=0; j<newImg->width;j++)
(cannyImg->imageData + i*newImg->width)[j]= abs((sebelum-
>imageData + i*newImg->width)[j] - (sekarang->imageData +
i*newImg->width)[j]);
=====
```

Coding di atas dapat diperjelas dengan sebuah persamaan seperti berikut :

$$\mathbf{dst}(i)_c = |\mathbf{src1}(I)_c - \mathbf{src2}(I)_c| \quad [3.9]$$

dimana : $\mathbf{src1}(I)_c$ merupakan citra dari frame 1 (**sebelum**)

$\mathbf{src2}(I)_c$ merupakan citra dari frame 2 (**sekarang**)

$\mathbf{dst}(i)_c$ merupakan citra hasil *motion detection* (**cannyImg**)

3.7 Citra Output

Hasil akhir dari keseluruhan proses di atas berupa citra keluaran yang sifatnya real time (video). Berikut inisialisasi yang digunakan untuk menampilkan citra *output* dengan *window*-nya :

```
=====
cvShowImage ("ASLI", newImg); ..... (1)
cvDestroyWindow ("detection"); ..... (2)
cvReleaseCapture (&cv_cap); ..... (3)
cvReleaseImage( &cannyImg ); ..... (4)
cvReleaseMemStorage (&storage); ..... (5)
=====
```

Berikut penjelasan coding di atas :

1. Inisialisasi fungsi untuk menampilkan citra **newImg** dengan *window* “ASLI”. Berlaku juga untuk *window* dan *image header* lainnya.
2. Inisialisasi fungsi untuk menghancurkan *window* “**detection**”. Berlaku juga untuk *window* lainnya.
3. Inisialisasi fungsi untuk melepaskan struktur **cv_cap** yang dialokasikan oleh **cvCaptureFromCam**.
4. Inisialisasi untuk melepaskan *image header* dan gambar dari **cannyImg**.
5. Inisialisasi fungsi untuk melepaskan penyimpanan memori dalam **storage**.

3.8 Analisis Kebutuhan Perangkat Keras

Adapun kebutuhan perangkat keras yang digunakan untuk menjalankan sistem ini adalah :

1. *Dekstop* ASUS A42J Series.
2. *Processor* Intel Core i5-460M 2.53 GHz.

3. RAM 2 GB.
4. *Hard disk* 2.3 GB (untuk Visual Studio 2005), 60MB (untuk *library* OpenCV), dan 20 MB (untuk memori penyimpanan data sistem yang dirancang).
5. *Web-cam* VGA 640x480 piksel.

3.9 Analisis Kebutuhan Perangkat Lunak

Perangkat lunak yang dibutuhkan untuk menjalankan sistem pendeteksian objek yang bergerak dengan metode *contour* ini adalah :

1. Sistem operasi Windows 7 Home Premium 64-bit (6.1, Build 7600).
2. Visual C++ dalam paket Microsoft Visual Studio 2005.
3. OpenCV 2.1.

BAB IV

PENGUJIAN, ANALISIS, DAN PEMBAHASAN

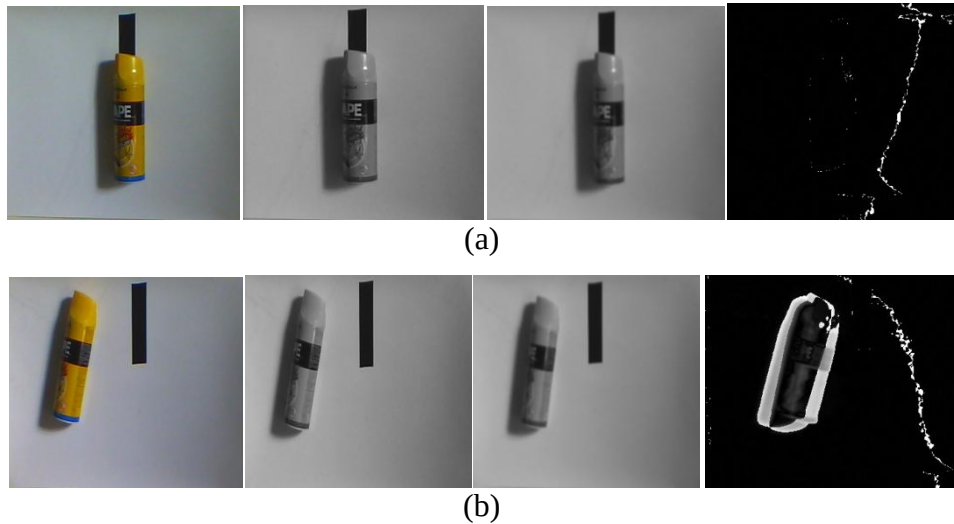
Pengujian *software* ini menggunakan beberapa parameter untuk mengetahui ada tidaknya perubahan yang terjadi. Parameter yang digunakan antara lain, warna *background*, intensitas cahaya, dan perbandingan antara benda yang diam dengan benda yang bergerak. Warna yang dipilih menjadi *background* terdiri dari 10 warna, sedangkan intensitas cahaya yang digunakan berdasarkan kondisi cahaya pada saat pengujian berlangsung. Kondisi cahaya terang pada saat siang hari (2.500 lux), kondisi agak terang menggunakan cahaya lampu kamar di malam hari (500 lux), dan kondisi gelap saat lampu kamar dimatikan (0,25 lux).

Tampilan citra pada pengujian ini akan menghasilkan 4 tampilan citra untuk masing-masing parameter yang disebut di atas. Keempat tampilan citra ini nantinya akan ditampilkan pada *window* yang berbeda-beda sesuai dengan proses pengolahan citra yang dilakukan, misalnya untuk citra yang langsung dari kamera akan ditampilkan dalam *window* '**asli**'. Untuk citra *grayscale* akan ditampilkan pada *window* '**gray**', dan citra yang terfilter dengan *Gaussian filter* akan ditampilkan pada *window* '**gaussian**'. Tampilan terakhir yakni tampilan '**detection**' yang merupakan citra output, dengan memperhatikan citra ini, maka dapat diketahui benda tersebut mengalami pergerakan atau tidak.

Pengujian akan dilakukan dua kali, yakni pada saat benda diam (a), dan saat benda digerakkan (b) pada masing-masing warna *background* dan intensitas cahaya yang akan diujikan. Berikut hasil yang diperoleh selama pengujian.

4.1 *Background* Putih

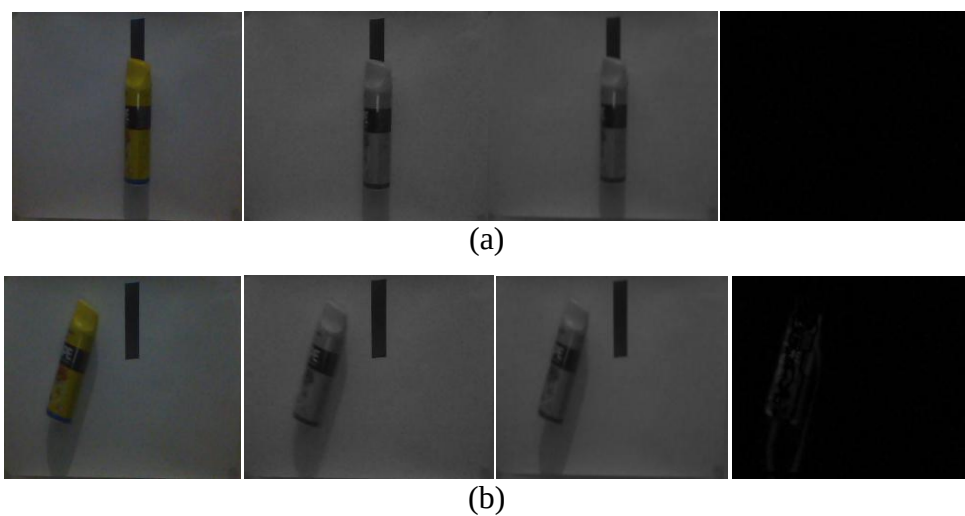
4.1.1 Kondisi Cahaya Terang (Cahaya Matahari)



Gambar 4.1. *Background* Putih Pada Kondisi Terang (a) Benda Diam;
(b) Benda Bergerak

Citra hasil yang diperoleh banyak mengandung derau. Hal ini dikarenakan warna putih sangat baik dalam memantulkan cahaya, bahkan bayangan dari objek tersebut dideteksi sebagai tepian.

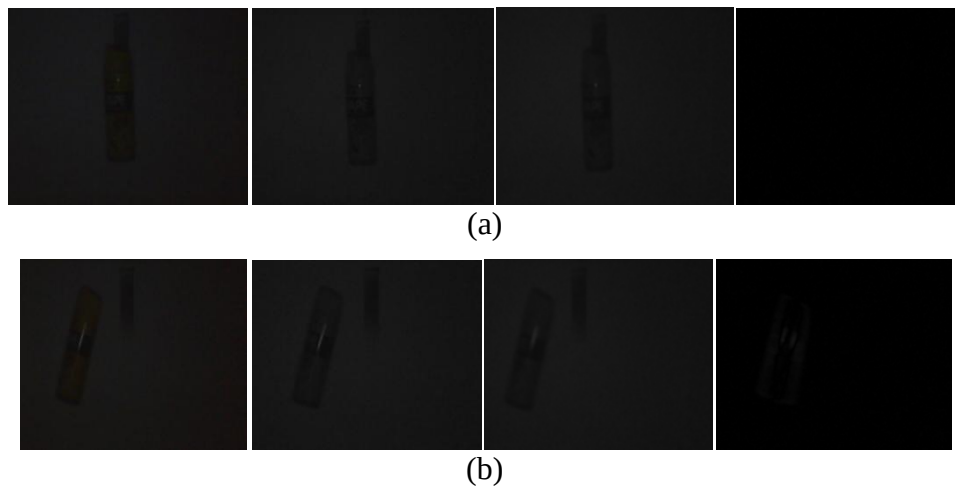
4.1.2 Kondisi Cahaya Agak Terang (Cahaya Lampu)



Gambar 4.2. *Background* Putih Pada Kondisi Cahaya Lampu (a) Benda Diam;
(b) Benda Bergerak

Citra **detection** pada gambar 4.2(a) memperlihatkan tidak adanya objek yang bergerak di depan kamera, sehingga citra yang dihasilkan menjadi gelap. Pada gambar 4.2(b) terlihat derau pada citra **detection** sudah jauh berkurang, hanya saja bayangan objek masih terdeteksi sebagai tepi.

4.1.3 Kondisi Gelap (Tanpa Cahaya)

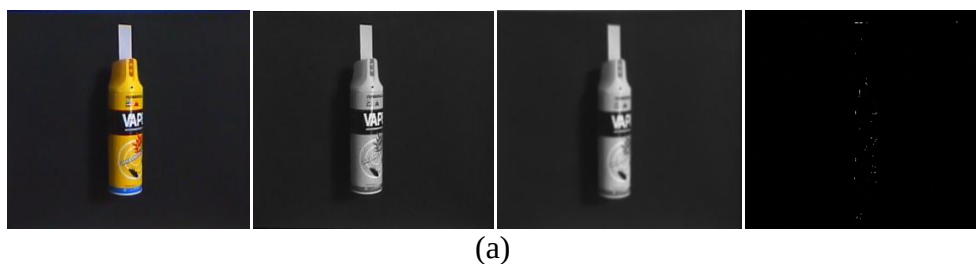


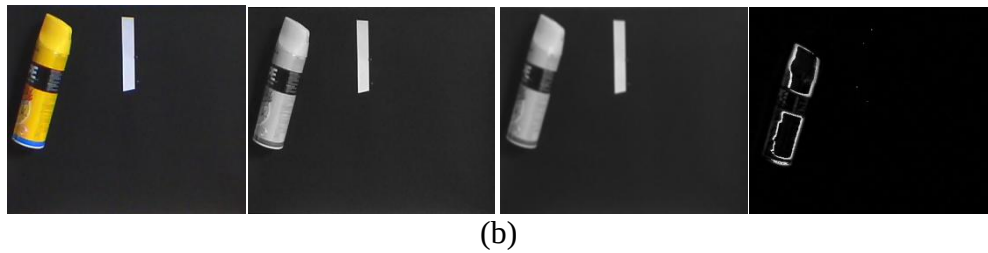
Gambar 4.3. *Background* Putih Pada Kondisi Gelap (a) Benda Diam; (b) Benda Bergerak

Citra **detection** yang ditampilkan pada gambar 4.3(b) dapat dikatakan sempurna. Tepian dari objek walaupun tipis namun dapat terlihat dengan jelas tanpa adanya derau.

4.2 *Background* Hitam

4.2.1 Kondisi Cahaya Terang (Cahaya Matahari)

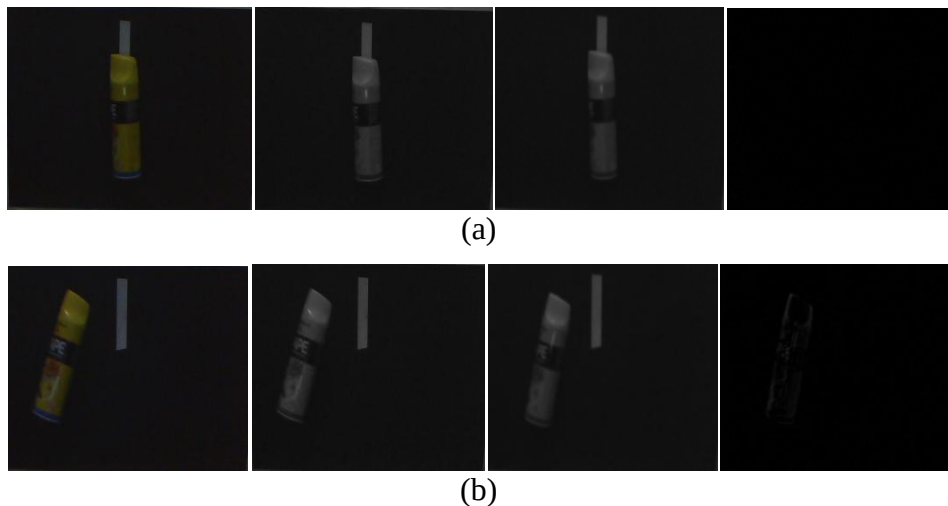




Gambar 4.4. *Background* Hitam Pada Kondisi Terang (a) Benda Diam;
(b) Benda Bergerak

Pada gambar di atas terlihat derau yang dihasilkan pada citra **detection** sangat sedikit. Hal ini dikarenakan warna hitam merupakan pemantul cahaya yang buruk. Pada pengujian dengan menggunakan intensitas cahaya yang terang (cahaya matahari), hasil citra **detection** yang dapat dikatakan mendekati sempurna, yakni saat *background* berwarna hitam. Hal ini dapat dilihat saat benda bergerak seperti gambar 4.4(b) di atas.

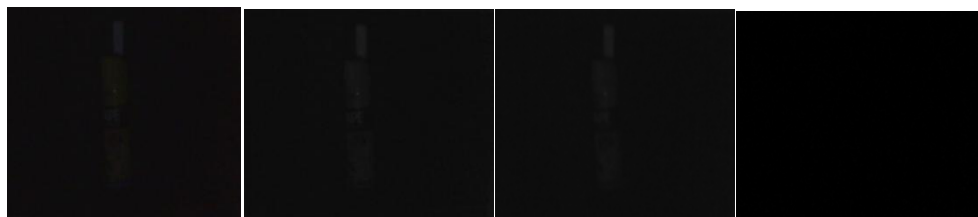
4.2.2 Kondisi Cahaya Agak Terang (Cahaya Lampu)



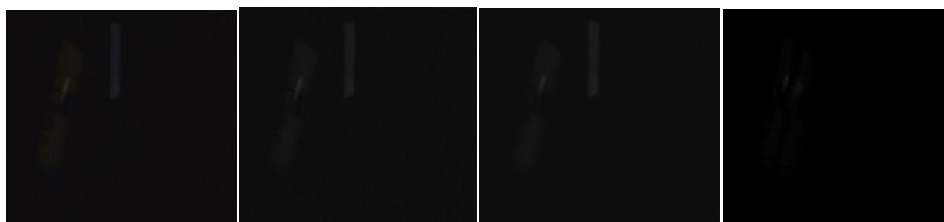
Gambar 4.5. *Background* Hitam Pada Kondisi Cahaya Lampu (a) Benda Diam;
(b) Benda Bergerak

Bahkan, di saat intensitas cahaya agak terang (cahaya lampu) citra **detection** yang dihasilkan dapat dikatakan mendekati sempurna (gambar 4.5(b)). Saat benda diam citra *output* yang dihasilkan tidak mengandung derau sedikitpun.

4.2.3 Kondisi Gelap (Tanpa Cahaya)



(a)



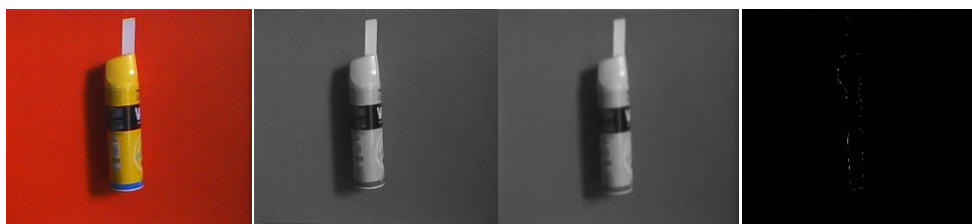
(b)

Gambar 4.6. *Background* Hitam Pada Kondisi Gelap (a) Benda Diam; (b) Benda Bergerak

Dari hasil citra ***detection*** di atas (gambar 4.6(b)), dapat dikatakan *background* hitam merupakan *background* yang paling buruk pada saat kondisi ruangan tanpa cahaya (gelap).

4.3 *Background* Merah

4.3.1 Kondisi Cahaya Terang (Cahaya Matahari)



(a)

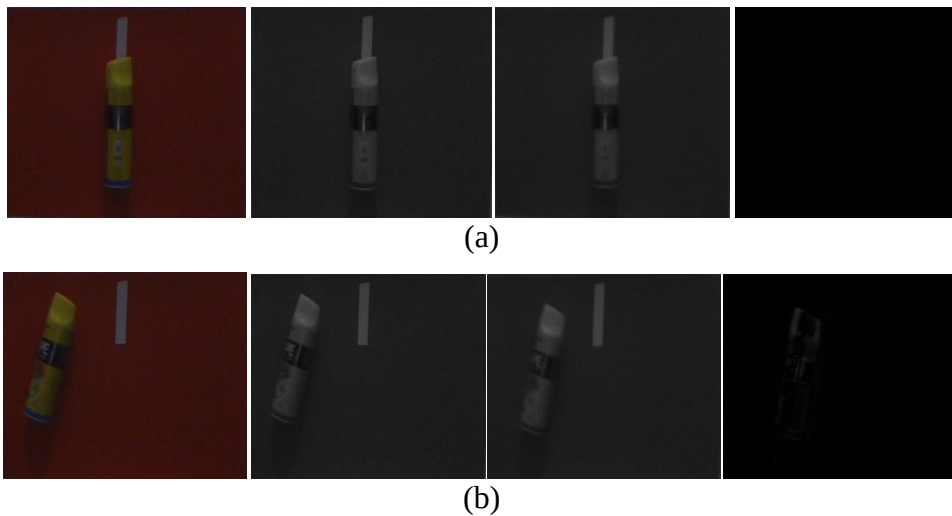


(b)

Gambar 4.7. *Background* Merah Pada Kondisi Terang (a) Benda Diam; (b) Benda Bergerak

Pada saat benda diam tampak derau yang diakibatkan oleh pantulan cahaya. Pada saat benda bergerak bayangan objek masih terdeteksi sebagai tepi. Namun, secara keseluruhan deteksi kontur pada *background* merah dengan pencahayaan matahari dapat dikatakan baik.

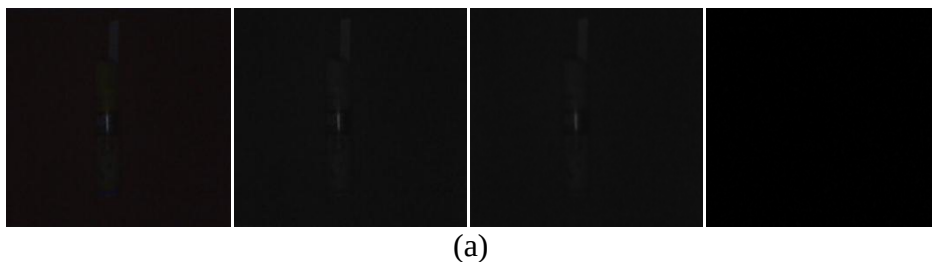
4.3.2 Kondisi Cahaya Agak Terang (Cahaya Lampu)

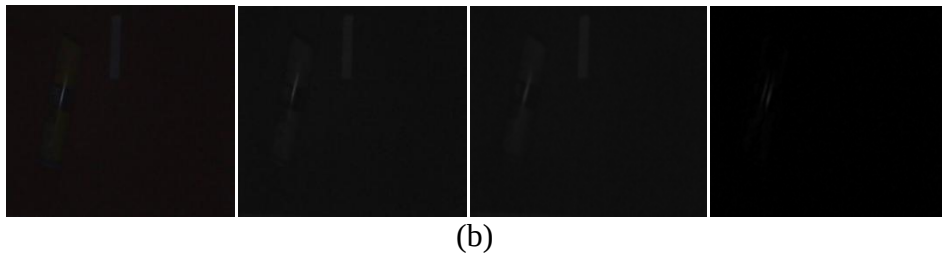


Gambar 4.8. *Background* Merah Pada Kondisi Cahaya Lampu (a) Benda Diam; (b) Benda Bergerak

Saat benda dalam keadaan diam, tampak tidak ada deteksi kontur yang ditampilkan pada *window detection* (gambar 4.8(a)). Pendeteksian kontur objek yang bergerak tampak terlihat dengan baik dan jelas tanpa adanya derau yang mengganggu (gambar 4.8(b)).

4.3.3 Kondisi Gelap (Tanpa Cahaya)



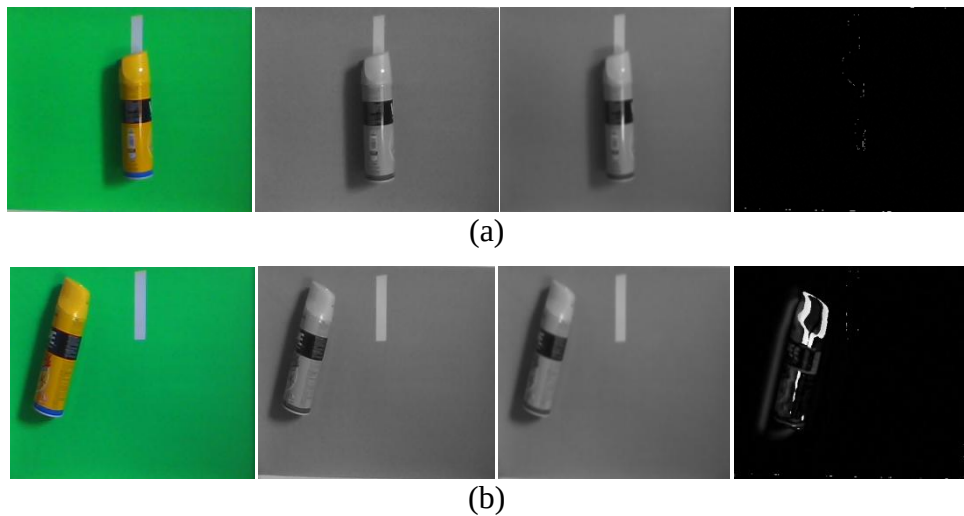


Gambar 4.9. *Background Merah* Pada Kondisi Gelap (a) Benda Diam;
(b) Benda Bergerak

Pendeteksian kontur baik objek yang diam dan objek yang bergerak pada *background merah* saat kondisi gelap sangat buruk sekali. Pada gambar 4.9(b) tampak sedikit sekali objek yang sedang bergerak yang terdeteksi.

4.4 *Background Hijau*

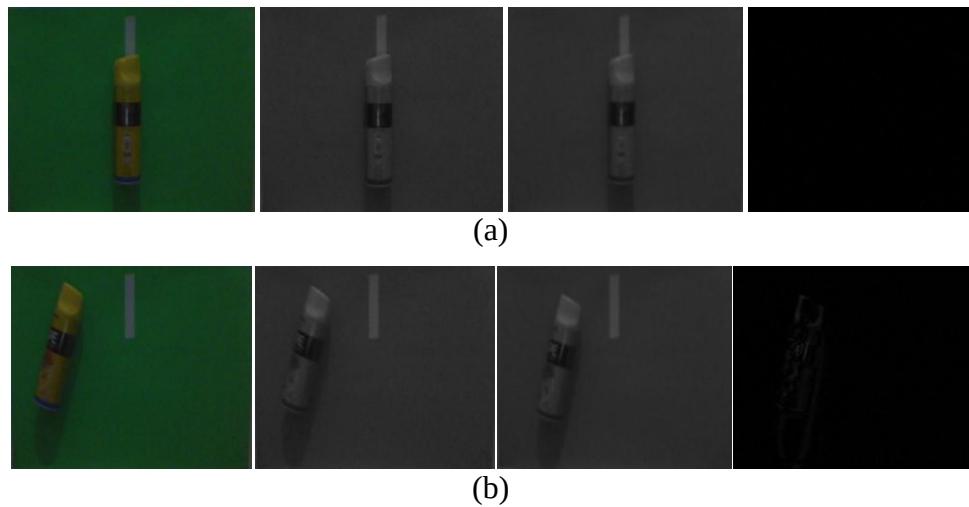
4.4.1 Kondisi Cahaya Terang (Cahaya Matahari)



Gambar 4.10. *Background Hijau* Pada Kondisi Terang (a) Benda Diam;
(b) Benda Bergerak

Pada benda diam terlihat adanya derau akibat pantulan cahaya terhadap objek (gambar 2.10(a)). Sedangkan saat benda digerakkan, bayangan yang terbentuk dari cahaya matahari di deteksi sebagai kontur (gambar 4.10(b)).

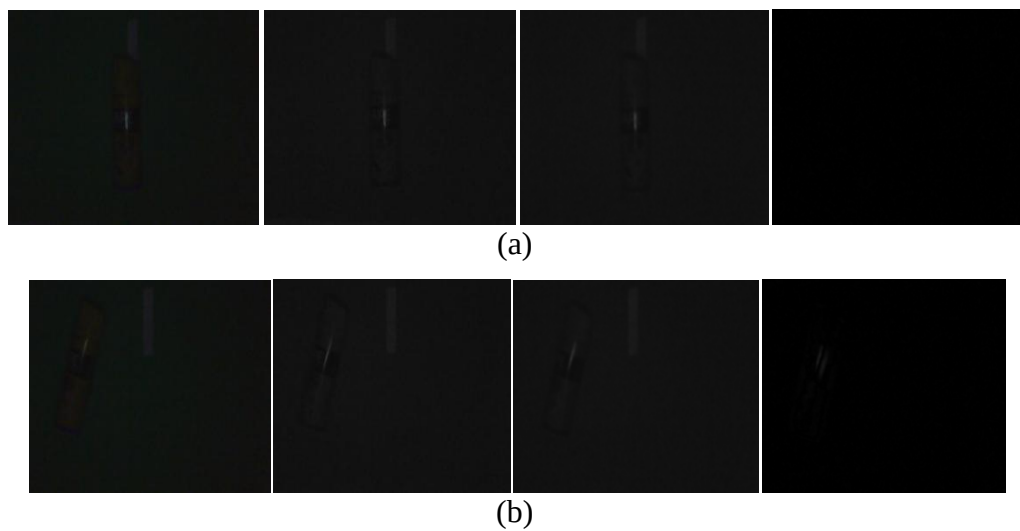
4.4.2 Kondisi Cahaya Agak Terang (Cahaya Lampu)



Gambar 4.11. *Background Hijau Pada Kondisi Cahaya Lampu* (a) Benda Diam; (b) Benda Bergerak

Hasil deteksi kontur yang diperoleh lebih baik dibandingkan dengan pencahayaan terang. Pada objek yang bergerak (gambar 4.11(b)), tampak sekali kontur yang terbentuk saat benda digerakkan.

4.4.3 Kondisi Gelap (Tanpa Lampu)

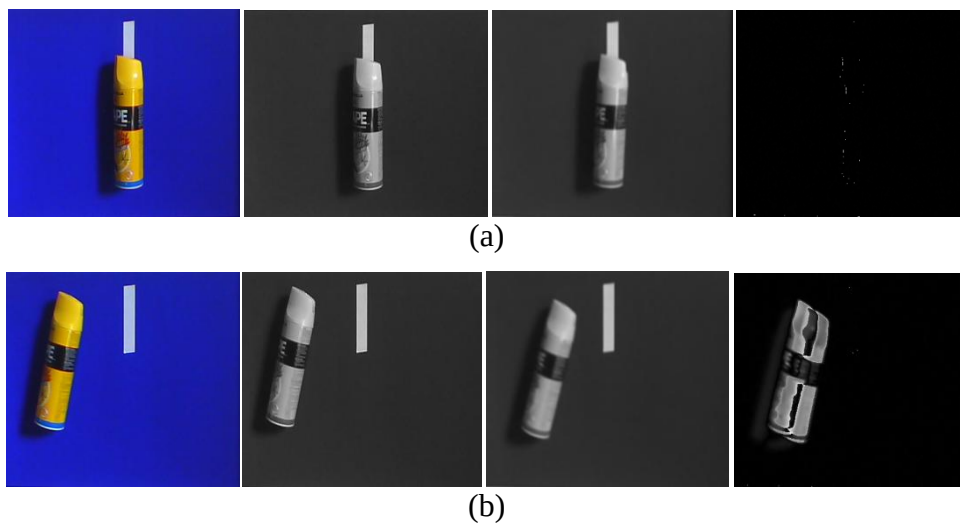


Gambar 4.12. *Background Hijau Pada Kondisi Gelap* (a) Benda Diam; (b) Benda Bergerak

Hasil pendeteksian kontur terhadap objek yang bergerak pada *background* hijau saat kondisi gelap sangat buruk sekali. Kontur dari objek yang bergerak hampir tidak terdeteksi sama sekali (gambar 4.12(b)).

4.5 *Background* Biru

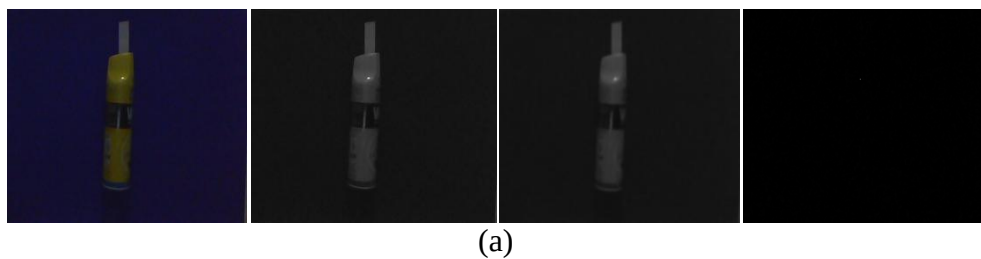
4.5.1 Kondisi Cahaya Terang (Cahaya Matahari)

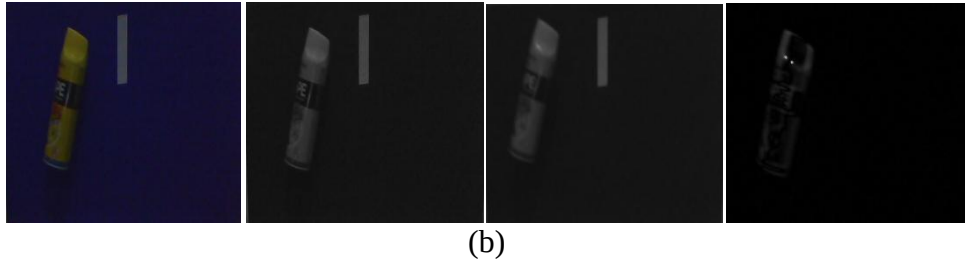


Gambar 4.13. *Background* Biru Pada Kondisi Terang (a) Benda Diam; (b) Benda Bergerak

Citra hasil deteksi sudah terdeteksi dengan baik saat benda bergerak (gambar 4.13(b)), hanya ada sedikit derau yang terdeteksi akibat pembiasan dan pantulan cahaya terhadap objek.

4.5.2 Kondisi Cahaya Agak Terang (Cahaya Lampu)



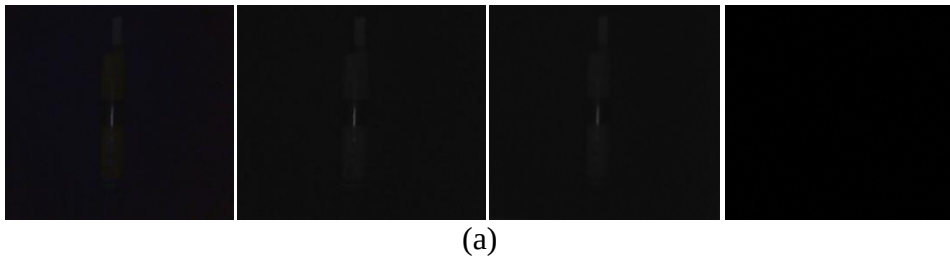


(b)

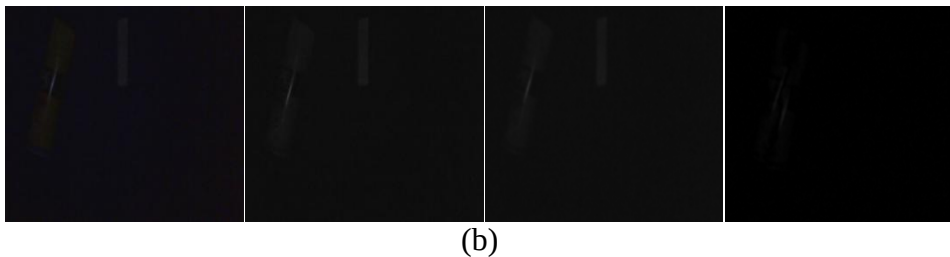
Gambar 4.14. *Background* Biru Pada Kondisi Cahaya Lampu (a) Benda Diam;
(b) Benda Bergerak

Hasil yang diperoleh lebih baik dibandingkan pada saat kondisi terang. Hal ini dapat dilihat dari gambar di atas. Kontur dari objek yang bergerak dapat terlihat dengan jelas (gambar 4.14(b)).

4.5.3 Kondisi Gelap (Tanpa Lampu)



(a)



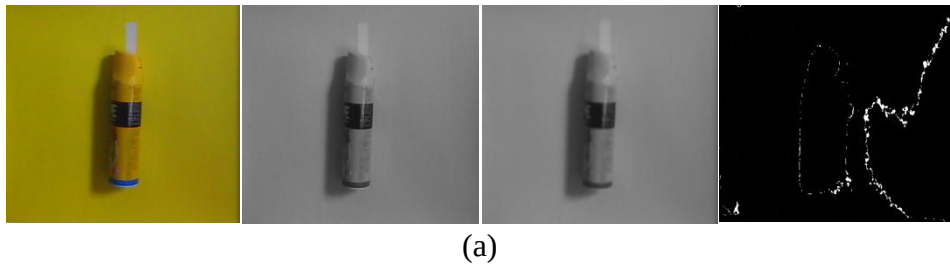
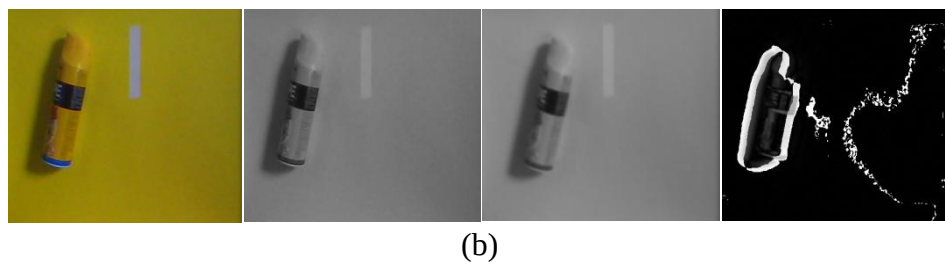
(b)

Gambar 4.15. *Background* Biru Pada Kondisi Gelap (a) Benda Diam;
(b) Benda Bergerak

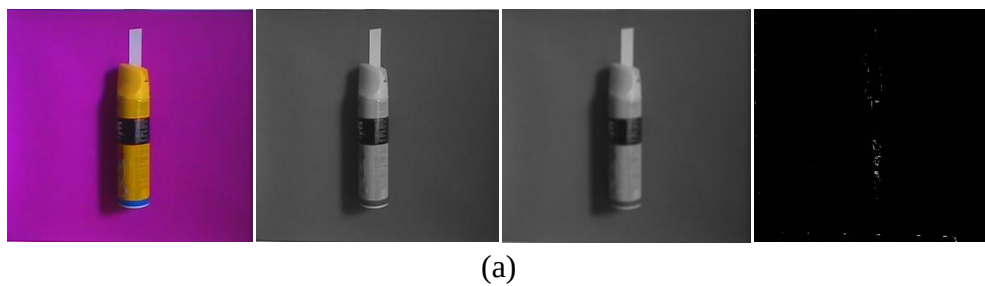
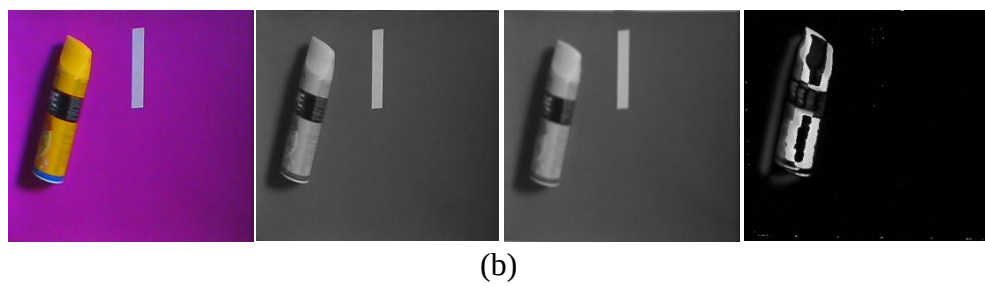
Saat kondisi gelap, hasil yang diperoleh kurang maksimal. Kontur hampir tidak terdeteksi sama sekali. Dari ketiga warna primer (RGB), *background* biru memiliki hasil yang sedikit lebih baik saat dalam kondisi gelap.

4.6 *Background* Warna Lain

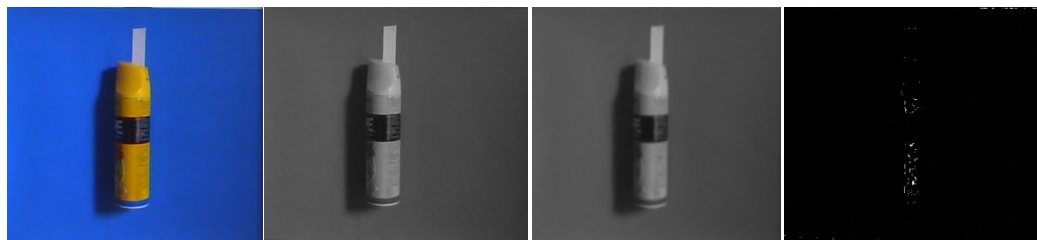
4.6.1 Kondisi Cahaya Terang (Cahaya Matahari)



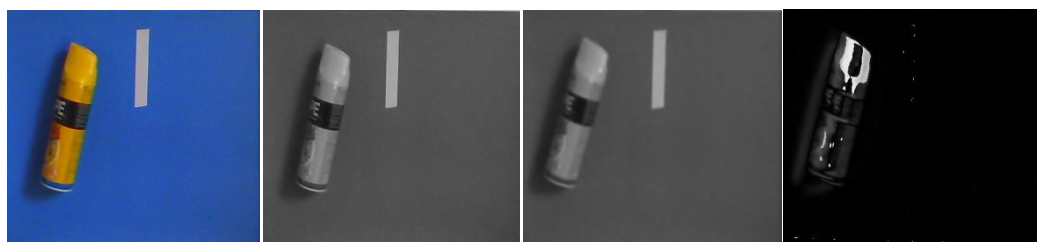
Gambar 4.16. *Background* Kuning Pada Kondisi Terang (a) Benda Diam; (b) Benda Bergerak



Gambar 4.17. *Background* Ungu Pada Kondisi Terang (a) Benda Diam; (b) Benda Bergerak

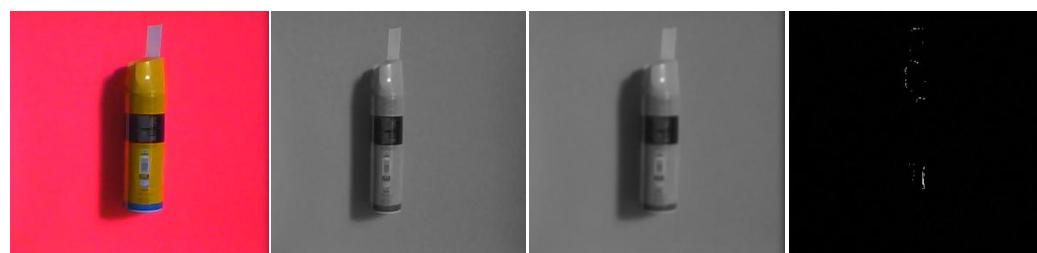


(a)

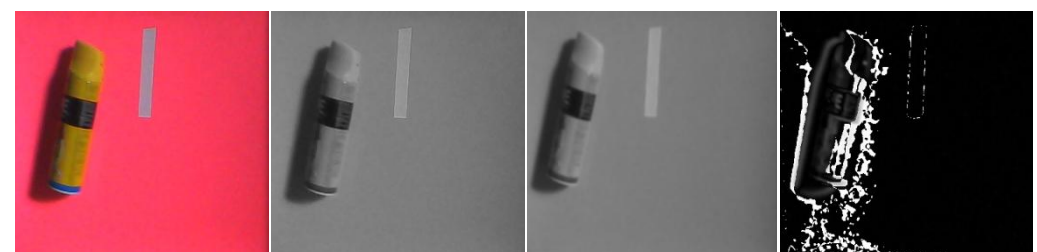


(b)

Gambar 4.18. *Background Cyan* Pada Kondisi Terang (a) Benda Diam;
(b) Benda Bergerak

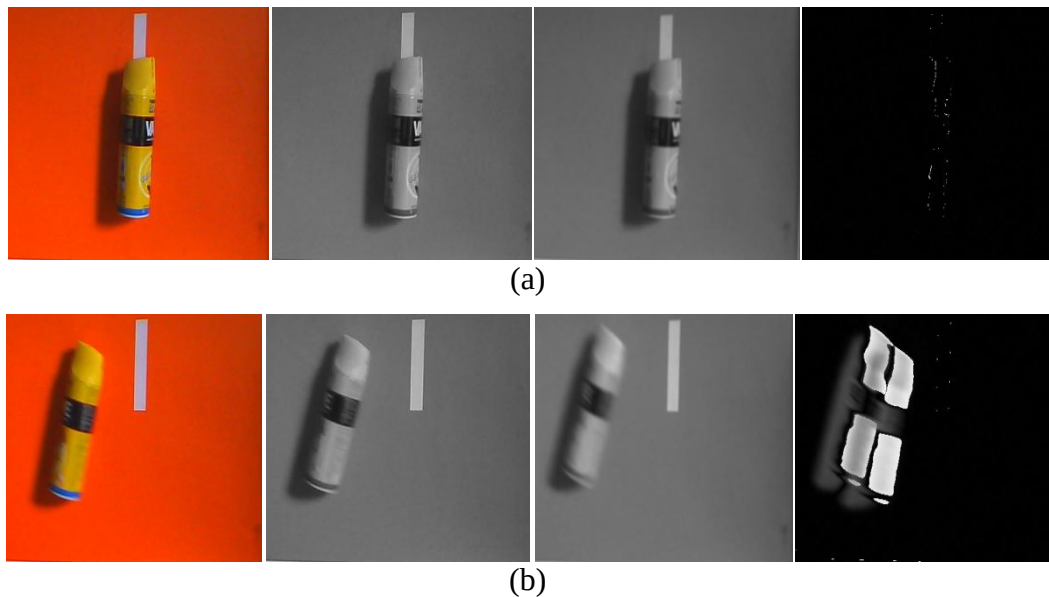


(a)



(b)

Gambar 4.19. *Background Merah Muda* Pada Kondisi Terang (a) Benda Diam;
(b) Benda Bergerak

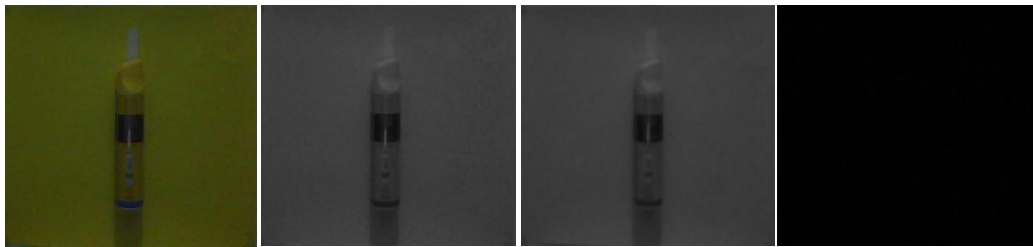


Gambar 4.20. *Background Jingga Pada Kondisi Terang* (a) Benda Diam;
(b) Benda Bergerak

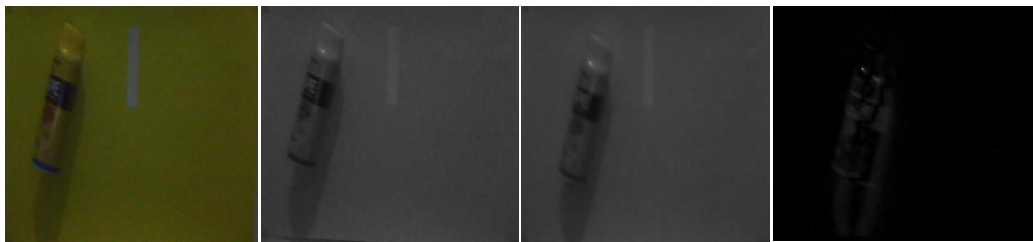
Pada pengujian di atas tampak bahwa cahaya terang (cahaya matahari) sangat mempengaruhi pendeteksian kontur yang dihasilkan. Selain cahaya matahari, faktor background ternyata memiliki pengaruh yang signifikan. Hal ini terlihat jelas pada saat *background* kuning atau merah muda, jika dibandingkan dengan *background* ungu atau cyan. *Background* yang dikatakan sebagai warna cerah seperti putih, kuning, merah muda, ataupun jingga, ternyata merupakan *background* yang buruk pada waktu kondisi terang, karena banyaknya derau yang dihasilkan. Sebaliknya, warna gelap seperti hitam, biru, ataupun merah, menghasilkan kontur yang dapat dikatakan cukup baik.

Derau ini dihasilkan karena pantulan cahaya terang tadi terhadap *background*-nya. Fungsi kamera sebagai penangkap objek sekaligus pengatur cahaya memiliki peran penting dalam pendeteksian kontur. Derau ini dapat di minimalkan apabila menggunakan kamera beresolusi tinggi.

4.6.2 Kondisi Cahaya Agak Terang (Cahaya Lampu)

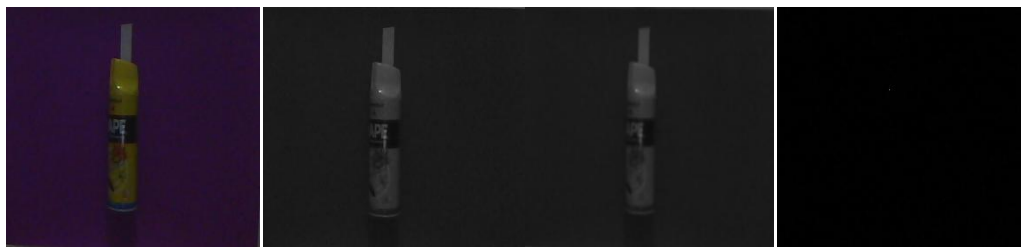


(a)

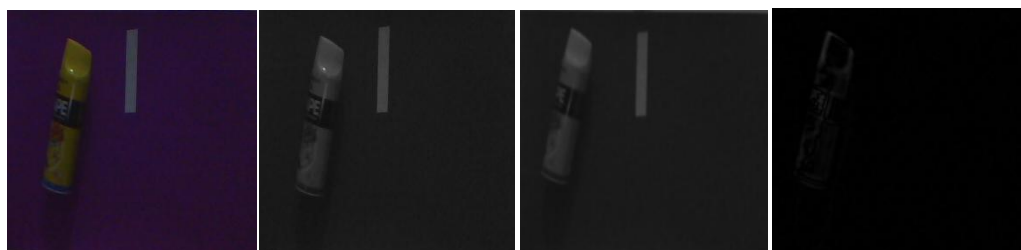


(b)

Gambar 4.21. *Background* Kuning Pada Kondisi Cahaya Lampu (a) Benda Diam; (b) Benda Bergerak

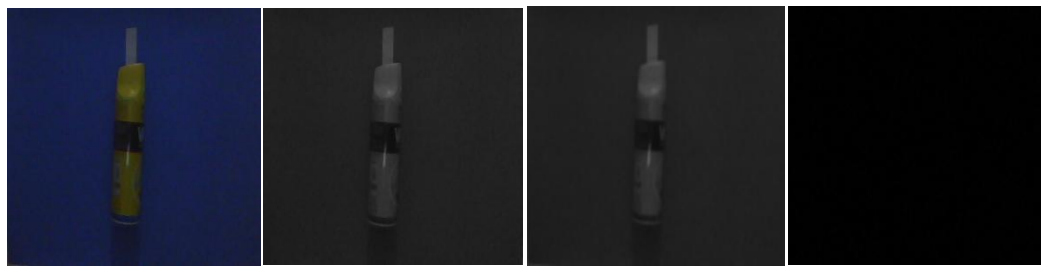


(a)



(b)

Gambar 4.22. *Background* Ungu Pada Kondisi Cahaya Lampu (a) Benda Diam; (b) Benda Bergerak

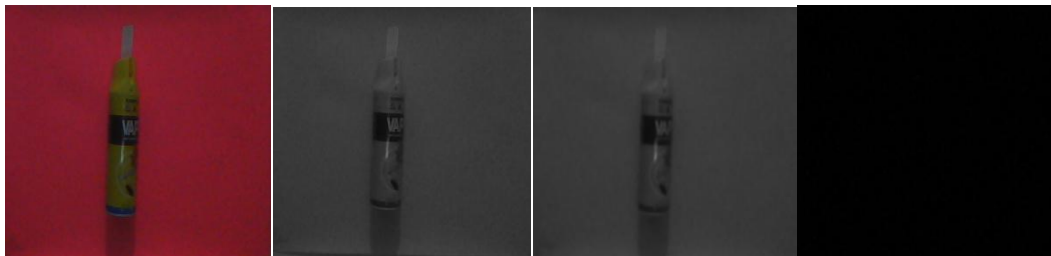


(a)

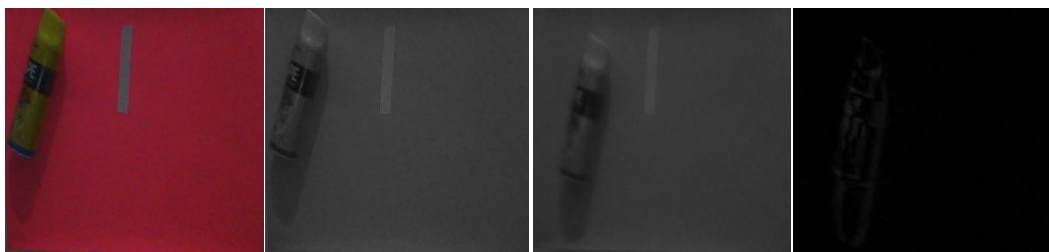


(b)

Gambar 4.23. *Background Cyan* Pada Kondisi Cahaya Lampu (a) Benda Diam; (b) Benda Bergerak

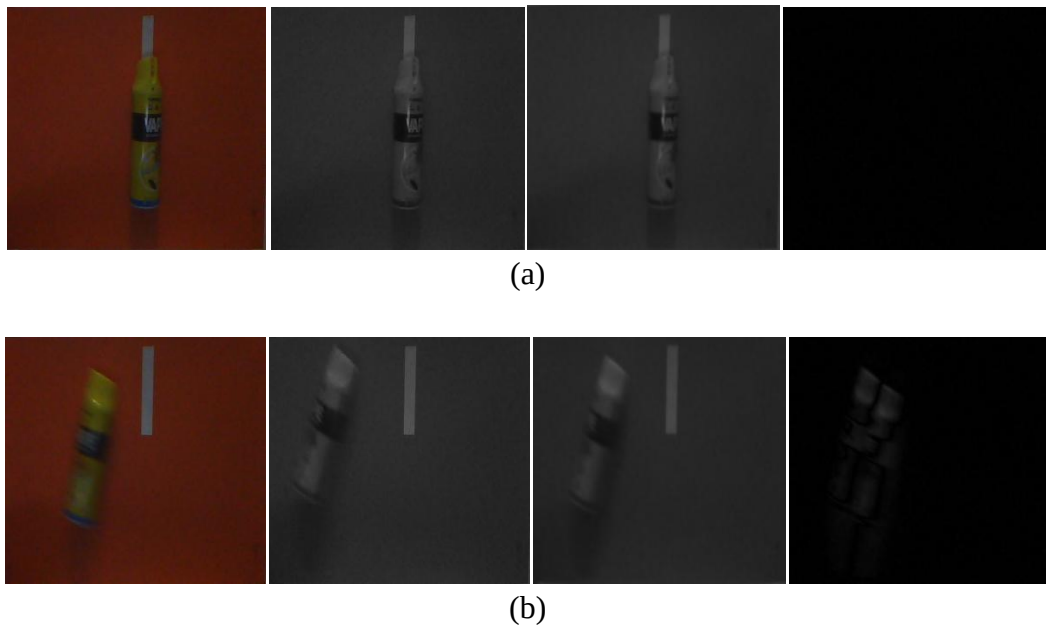


(a)



(b)

Gambar 4.24. *Background Merah Muda* Pada Kondisi Cahaya Lampu (a) Benda Diam; (b) Benda Bergerak

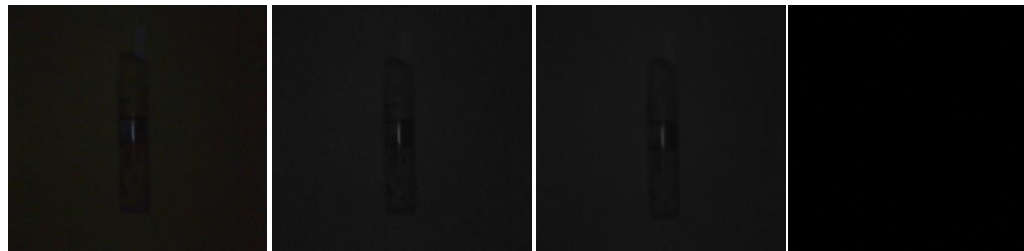


Gambar 4.25. *Background* Jingga Pada Kondisi Cahaya Lampu (a) Benda Diam; (b) Benda Bergerak

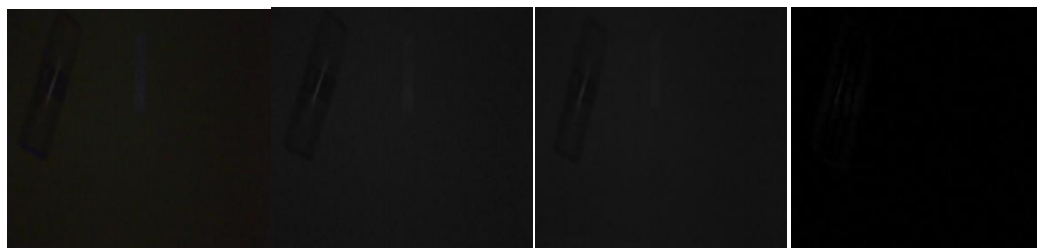
Pada pencahayaan dengan menggunakan lampu kamar ini dilakukan saat malam hari. Dan daya lampu yang digunakan, yaitu 20 Watt. Dengan menggunakan cahaya lampu (agak terang), dapat dilihat hasil deteksi kontur yang diperoleh jauh lebih baik dibandingkan dengan cahaya terang. Derau yang dihasilkan dari pantulan cahaya yang berlebihan tidak terlihat sama sekali. Bahkan, di saat benda diam, layar pendeteksian tidak menangkap gambar apapun (gelap). Walaupun ada beberapa *background* yang masih menangkap bayangan objek sebagai kontur, namun dapat disimpulkan bahwa dengan pencahayaan yang sesuai, hasil deteksi kontur yang diperoleh dapat maksimal.

Ada beberapa gambar yang menangkap *double* sebuah objek yang sedang bergerak yang mempengaruhi hasil pendeteksian kontur. Hal ini dapat dihindari jika menggunakan kamera dengan *frame rate* yang tinggi.

4.6.3 Kondisi Gelap (Tanpa Cahaya)

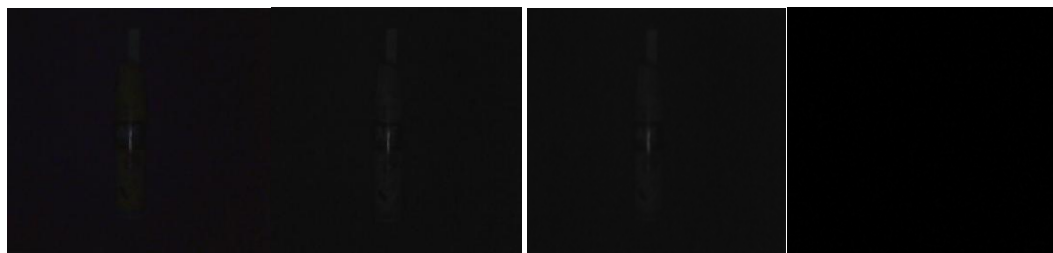


(a)

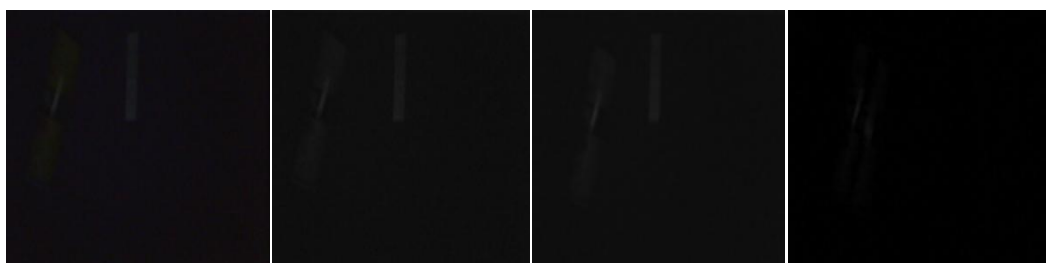


(b)

Gambar 4.26. *Background* Kuning Pada Kondisi Gelap (a) Benda Diam;
(b) Benda Bergerak



(a)

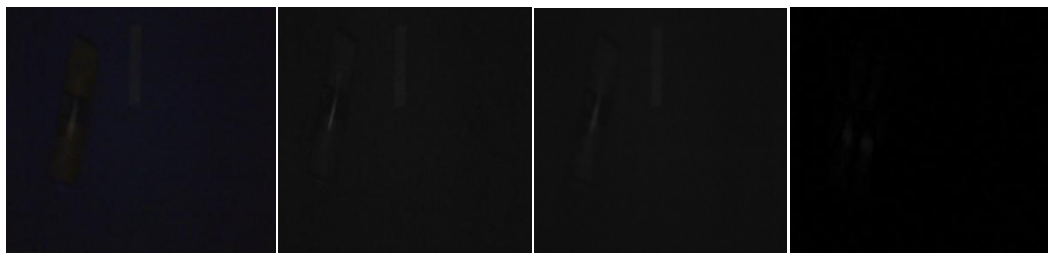


(b)

Gambar 4.27. *Background* Ungu Pada Kondisi Gelap (a) Benda Diam;
(b) Benda Bergerak

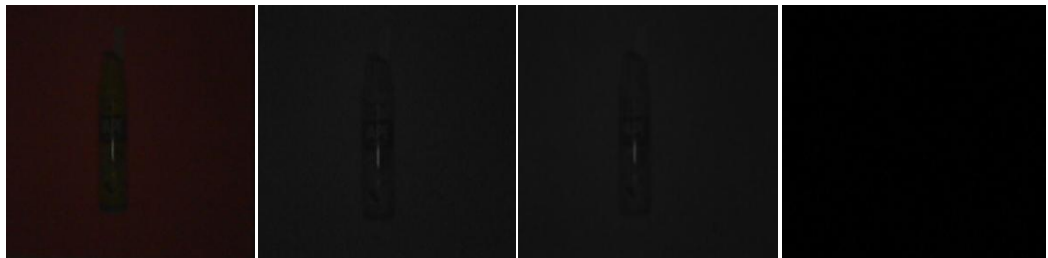


(a)

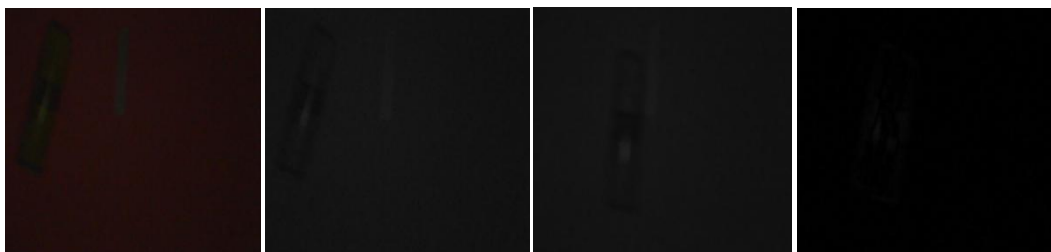


(b)

Gambar 4.28. *Background Cyan* Pada Kondisi Gelap (a) Benda Diam;
(b) Benda Bergerak

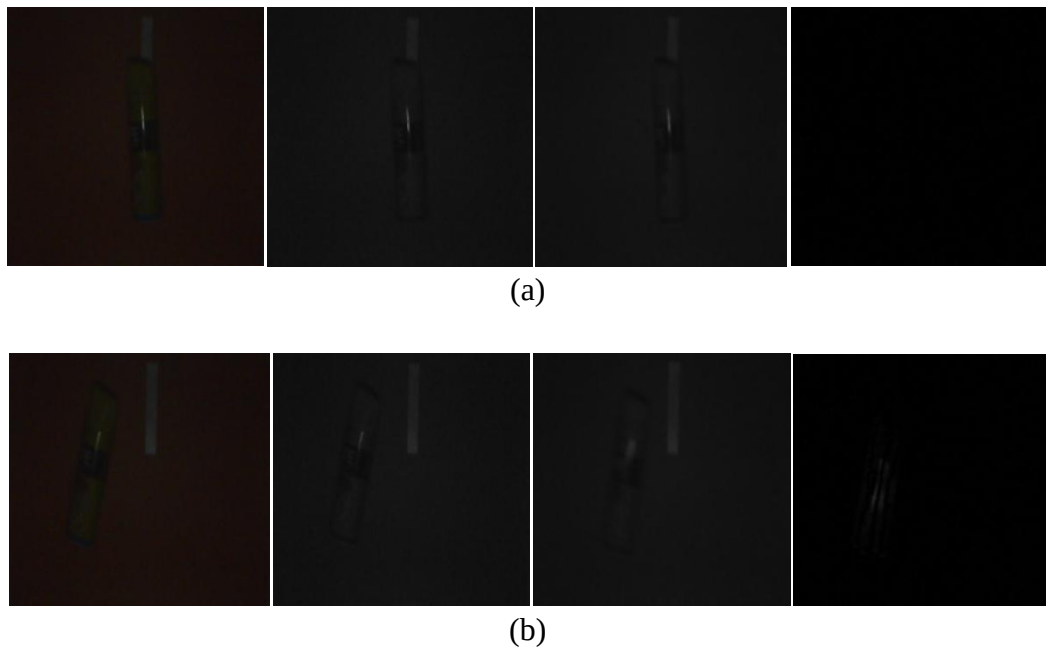


(a)



(b)

Gambar 4.29. *Background Merah Muda* Pada Kondisi Gelap (a) Benda Diam;
(b) Benda Bergerak



Gambar 4.30. *Background* Jingga Pada Kondisi Gelap (a) Benda Diam;
(b) Benda Bergerak

Hasil pendeteksian kontur pada objek yang bergerak di saat kondisi gelap menunjukkan hasil yang kurang memuaskan. Meskipun ada beberapa *background* yang mampu menangkap adanya gerakan, namun sebagian besar kontur yang terbentuk tidak sempurna atau tidak sesuai dengan kontur dari objek tersebut. Hal ini dikarenakan tidak adanya cahaya yang ditangkap oleh kamera, sehingga tepi dari objek menyatu dengan *background* yang tampak gelap.

Pendeteksian objek yang bergerak dengan metode kontur ini sangat dipengaruhi oleh intensitas cahaya dan resolusi kamera. Dengan cahaya yang agak terang (sebuah lampu), akan menghasilkan kontur yang lebih baik ketimbang pada saat kondisi gelap maupun terang. Resolusi kamera yang tinggi akan membantu dalam pengaturan cahaya yang diterima, dan mampu menangkap *frame* dengan kecepatan tinggi, sehingga citra yang dihasilkan tidak patah-patah.

BAB V

PENUTUP

5.1 Kesimpulan

Setelah melakukan berbagai macam pengujian maka dapat disimpulkan bahwa :

1. Pendeteksian kontur dilakukan berdasarkan ada atau tidaknya objek yang bergerak dalam suatu ruangan. Dimana pada saat tidak terjadi perubahan objek pada suatu *background* (tidak ada pergerakan), maka pendeteksian akan menampilkan layar hitam gelap. Namun, apabila terdapat objek yang bergerak pada suatu *background*, maka sistem akan mendeteksi objek tersebut berdasarkan kontur dari objek itu sendiri untuk kemudian ditampilkan secara *realtime*.
2. Warna *background* dan intensitas cahaya menjadi faktor pendukung dalam mendeteksi objek yang bergerak. Dengan intensitas cahaya yang kurang, kontur dari sebuah objek tampak tidak sempurna, dikarenakan cahaya yang dipantulkan oleh objek sangat kurang. Hal ini tentu berpengaruh pada saat *background* yang digunakan berwarna gelap (hitam, ungu, biru, dll). Sebaliknya, pada saat intensitas cahaya berlebih (terang), kontur yang dihasilkan dari objek yang bergerak mengandung banyak derau. Hal ini dikarenakan banyaknya pantulan cahaya yang diterima kamera, sehingga pantulan cahaya dari *background* dideteksi sebagai tepi (kontur). Derau akan banyak dihasilkan saat *background* yang digunakan berwarna cerah.

3. Secara keseluruhan proses pendeteksian objek yang bergerak dengan metode kontur ini sangat berpengaruh terhadap detektor yang digunakan untuk menangkap citra. Semakin tinggi resolusi kamera yang digunakan, semakin baik dalam pendeteksiannya, walau dengan intensitas cahaya yang kurang (gelap) maupun lebih (terang).

5.2 Saran

Masukan maupun saran untuk pengembangan lebih lanjut dari sistem ini sangat dibutuhkan, berikut beberapa masukan dan saran yang dapat dipertimbangkan :

1. Detektor yang digunakan dalam sistem ini menggunakan sebuah *webcam* yang disediakan pada sebuah *notebook*, tentunya kualitas yang dihasilkan masih standar, ke depan diharapkan dapat menggunakan sebuah kamera dengan resolusi yang cukup tinggi.
2. *Background* yang digunakan dalam pengujian menggunakan kertas karton berwarna, ke depan diharapkan *background* yang digunakan berupa tembok berwarna sehingga dapat langsung diterapkan dalam kehidupan sehari-hari dan hasilnya dapat dibandingkan.
3. Sistem ini hanya melakukan pendeteksian objek yang bergerak dengan menampilkan kontur dari objek tersebut (hasilnya berupa data visual), untuk ke depannya diharapkan adanya pemasangan alarm untuk menunjang data visual tersebut.
4. Pengembangan dalam hal sekuritas berbasis alarm dalam suatu gedung.

DAFTAR PUSTAKA

- Yokoyama, Masayuki., & Poggio, Tomaso., 2005. *A Contour-Based Moving Object Detection and Tracking*. Massachusetts Institute of Technology Cambridge, MA 02139.
- Firdausy, Kartika., Daryono., Yudhana, Anton., 2008. *Webcam Untuk Sistem Pemantauan Menggunakan Metode Deteksi Gerakan*. Skripsi, tidak diterbitkan. Yogyakarta : Teknik Elektro Universitas Ahmad Dahlan.
- Fahmi., 2007. *Perancangan Algoritma Pengolahan Citra Mata Menjadi Citra Polar Iris Sebagai Bentuk Antara Sistem Biometrik*. Skripsi, tidak diterbitkan. Medan : Fakultas Teknik Universitas Sumatera Utara.
- Rahardjo, Chita Ralina., Herdiyeni, Yeni., Ardiansyah, Firman., 2008. *Metode Cluster Self-Organizing Map Untuk Temu Kembali Citra*. Skripsi, tidak diterbitkan. Bogor : Fakultas Matematika dan Ilmu Pengetahuan Alam Institut Pertanian Bogor.
- Niam, Bahrn., 2008. *Analisi Deteksi Tepi Pada Citra Berdasarkan Perbaikan Kualitas Citra*. Skripsi, tidak diterbitkan. Semarang : Fakultas Teknik Universitas Diponegoro.
- CV. ANDI OFFSET. 2008. *Cepat Menguasai Visual Studio .Net 2008 Express*. Yogyakarta : pengarang.
- Ahmad, Usman., 2005. *Pengolahan Citra Digital & Teknik Pemrogramannya*. Yogyakarta : Graha Ilmu
- Sigit, Riyanto., 2005. *Step By Step Pengolahan Citra I*. Yogyakarta : ANDI

'LAMPIRAN'

Listing Program

PENDETEKSIAN OBJEK YANG BERGERAK DENGAN METODE CONTOUR MENGUNAKAN SOFTWARE VISUAL STUDIO DAN DETEKTOR WEBCAM

```
#include "cv.h"
#include "cxcore.h"
#include "highgui.h"
#include <cvcam.h>
#include <stdio.h>

void main (int argc, char* argv[])
{
    IplImage* newImg = 0;
    IplImage* image_out = 0;
    IplImage* cannyImg = 0;

    IplImage * gray1=0;
    IplImage * gray2=0;
    IplImage * gaussian1=0;
    IplImage * gaussian2=0;
    IplImage * sekarang=0;
    IplImage * sebelum=0;

    int i,j,c,n;

    //-----frame kosong-----
    cvNamedWindow("ASLI", CV_WINDOW_AUTOSIZE);
    cvNamedWindow("detection", CV_WINDOW_AUTOSIZE);
    cvNamedWindow("gray", CV_WINDOW_AUTOSIZE);
    cvNamedWindow("gaussian", CV_WINDOW_AUTOSIZE);

    //-----ambil gambar dari kamera dan simpan dinama newimage
    CvCapture* cv_cap=cvCaptureFromCAM(-1);
    newImg = cvQueryFrame(cv_cap);

    //tempat gambar dalam bentuk gray, gaussian, sekarang, sebelum dan cannyImg
    gray1 = cvCreateImage(cvSize(newImg->width, newImg->height), IPL_DEPTH_8U, 1);
    gray2 = cvCreateImage(cvSize(newImg->width, newImg->height), IPL_DEPTH_8U, 1);

    gaussian1 = cvCreateImage(cvSize(newImg->width, newImg->height), IPL_DEPTH_8U, 1);
    gaussian2 = cvCreateImage(cvSize(newImg->width, newImg->height), IPL_DEPTH_8U, 1);

    cannyImg = cvCreateImage(cvSize(newImg->width, newImg->height), IPL_DEPTH_8U, 1);
    sekarang = cvCreateImage(cvSize(newImg->width, newImg->height), IPL_DEPTH_8U, 1);
    sebelum = cvCreateImage(cvSize(newImg->width, newImg->height), IPL_DEPTH_8U, 1);

    gray1->origin = newImg->origin ;
    gray2->origin = newImg->origin ;
```

```

gaussian1->origin = newImg->origin ;
gaussian2->origin = newImg->origin ;
cannyImg->origin = newImg->origin ;

sekarang->origin = newImg->origin ;
sebelum->origin = newImg->origin ;

cvCvtColor( newImg, gray1, CV_BGR2GRAY ); //convert gambar menjadi grayscale

cvSmooth(gray1, gaussian1, CV_GAUSSIAN, 7, 7); //penghalusan citra dengan filter Gaussian

cvCanny(gaussian1, cannyImg, 50, 150, 3); //deteksi tepi dengan metode Canny
cvCopy(cannyImg, sekarang,0);

//-----proses contour-----
//parameters for the contour detection
CvMemStorage * storage = cvCreateMemStorage(0);
CvSeq * contour = 0;
int mode = CV_RETR_EXTERNAL;
mode = CV_RETR_CCOMP; //detect both outside and inside contour

//find the contour
cvFindContours (cannyImg, storage, &contour, sizeof(CvContour), mode, CV_CHAIN_
APPROX_SIMPLE, cvPoint(0,0));

//proses image
for(;;){
    newImg=cvQueryFrame(cv_cap);
    if(!newImg)
        break;
    cvCvtColor(newImg,gray2, CV_BGR2GRAY);
    cvSmooth(gray2, gaussian2, CV_GAUSSIAN, 7, 7);
    cvCopy(sekarang,sebelum,0);/--cany
    cvCopy(gaussian2,sekarang,0);/---gray

    //hitung beda image dengan rumus cvabsdiff
    //dst(i,j)= abs(srcA)-srcB
    for(i=0; i<newImg->height;i++) for(j=0; j<newImg->width;j++)
        (cannyImg->imageData + i*newImg->width)[j]= abs((sebelum->imageData + i*newImg->width)[j]
            - (sekarang->imageData + i*newImg->width)[j]);

    //tampilkan image out
    cvShowImage("ASLI",newImg);
    cvShowImage("detection", cannyImg);
    cvShowImage("gray", gray2);
    cvShowImage("gaussian", gaussian2);

```

```
c = cvWaitKey(1);
    if (c==27)
        break;
}
cvDestroyWindow( "ASLI" );
cvDestroyWindow( "detection" );

cvReleaseCapture(&cv_cap);
cvReleaseImage( &cannyImg );
cvReleaseMemStorage(&storage);
return 0;
//
}
```



