SKRIPSI

PEMBUATAN ALGORITMA FOTO PANORAMIC SECARA OTOMATIS MENGGUNAKAN KAMERA IP

Diajukan Untuk Memenuhi Persayaratn Dalam Mencapai Gelar Sarjan Strata Satu (S-1)
Teknik Geodesi

MUDZAKKIRO YAOMI **07.25.004**

JURUSAN TEKNIK GEODESI
FAKULTAS TEKNIK SIPIL DAN PERENCANAAN
INSTITUT TEKNOLOGI NASIONAL
MALANG

2012

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

LEMBAR PERSETUJUAN

Pembuatan Algoritma Foto Panoramic Secara Otomatis Menggunakan Kamera IP

Diajukan Sebagai Salah Satu Syarat Memperoleh Gelar Sarjana Teknik Geodesi S-1 Institut teknologi Nasional Malang

Disusun Oleh :

MUDZAKKIRO YAOMI

07.25.004

Menyetujui :

Dosen Pembimbing 1	Dosen Pembimbing 2
M. Edwin Tjahjadi,ST,M.Geom.Sc.,Ph.D	**Ir. Leo Pantimena, MSc.**

Mengetahui,

Ketua Jurusan Teknik Geodesi S-1

Ir. Agus Darpono, MT

LEMBAR PENGESAHAN

Pembuatan Algoritma Foto Panoramic Secara Otomatis Menggunakan Kamera IP

SKRIPSI

Telah dipertahankan di hadapan panitia penguji skripsi jenjang Strata-1 (S1) :

Pada Hari : Sabtu

Tanggal : 28 Juli 2012

Dan Diterima untuk memenuhi persyaratan guna memperoleh gelar Sarjana Teknik (ST)

Disusun Oleh :

MUDZAKKIRO YAOMI

07.25.004

Panitia Ujian Skripsi :

Ketua	Sekretaris
Ir. Agus Darpono MT	Silvester Sari Sai, ST. MT

Anggota Penguji :

Penguji I	Penguji II	Penguji III
Ir. Agus Darpono MT	M. Edwin Tjahjadi,ST,M.Geom.Sc.,Ph.D	Ir. M. Nurhadi MT

PEMBUATAN ALGORITMA FOTO PANORAMIC SECARA OTOMATIS MENGGUNAKAN KAMERA IP

Mudzakkiro Yaomi 07.25.004

Dosen Pembimbing I : M.Edwin Tjahjadi,ST,M.Geom.Sc.,Ph.D
Dosen Pembimbing II : Ir.Leo Pantimena, MSc

Abstraksi

Penelitian ini menyajikan suatu teknik untuk membangun mozaik foto panorama *(panoramic image mosaic)* yang dibentuk oleh sepasang foto stereo. Langkah awal yaitu dengan menentukan titik fitur pada tiap foto dengan *edge detector* dengan menggunakan metode yang dikemukakan oleh Harris dan Stephens. Lalu dengan menggunakan teknik *correlation*, titik fitur yang sudah terdeteksi akan di hubungkan antara foto kiri dan kanan. Pada proses ini, terdapat banyak kesalahan yang masih terjadi sehingga digunakan proses RANSAC untuk mengeliminasi kesalahan korelasi yang terjadi. Di dalam proses RANSAC terjadi perhitungan *homography* untuk membedakan korelasi yang benar dan salah sesuai dengan permintaan besaran nilai korelasi yang dianggap benar sehingga tidak di eliminasi. Terakhir, kedua foto tersebut akan di leburkan (*blending*) untuk dijadikan menjadi 1 buah citra foto panorama.

Kata kunci : *edge detector, correlation, RANSAC, blending*, panorama.

PERNYATAAN KEASLIAN SKRIPSI

Saya bertanda tangan di bawah ini :

Nama : Mudzakkiro Yaomi

NIM : 07.25.004

Program Studi : Teknik Geodesi S1

Fakultas : Teknik Sipil Dan Perencanaan

Menyatakan dengan sesungguhnya bahwa skripsi saya dengan judul :

"Pembuatan Algoritma Foto Panoramic Secara Otomatis Menggunakan Kamera IP"

Adalah hasil karya saya sendiri, bukan merupakan duplikat serta tidak mengutip atau menyadur dari hasil karya orang lain kecuali disebutkan sumbernya.

Malang, 23 September 2012

Yang membuat pernyataan

Mudzakkiro Yaomi
07.25.004

LEMBAR PERSEMBAHAN

Terima kasih ya Allah ya Robbi atas semua berkah dan rahmat-Mu. Selesai sudah tugas dan amanat keluarga saya untuk menyelesaikan program studi S1. Tak pernah hamba mengucapkan syukur dan puji hanya untuk-Mu.

Teruntuk kedua orang tua ku yang sudah berada di surga, tak pernah lupa anakmu ini mendoakan kalian selalu. Dan berharap agar kalian selalu memperhatikan anakmu dari jauh sana. Amin. Ibu, tunai sudah amanat yang engkau berikan untukku, dan aku selalu bersyukur atas kasih sayang yang selama ini sudah engkau berikan. Maafkan anakmu jikalau masih belum sempat membahagiakanmu. Semoga dengan selesainya amanat yang engkau berikan, engkau bisa tersenyum selalu. Amin

Teruntuk semua kakak-kakak ku, tak ada kata yang bisa mewakili betapa inginnya aku berterima kasih kepada kalian. Karena atas dukungan kalian, baik dalam moriil maupun dana, aku bisa sampai sekarang ini. Bersyukur selalu atas kalian, yang telah menjadi saudara kandung ku. Semoga Allah membalasnya dengan yang lebih indah dan baik kepada kalian. Insya Allah, Amin. Ka Eka, Bang Uwi, Bang Ica, Bang Oyi, Ka Lia. I love u all, always.

Semua dosen, yang telah mengajarkan saya dan memberikan ilmunya kepada saya. Terima kasih banyakkkkkkk pak!!!! Pak Edwin, Pak Hery, Pak Agus, Pak Sil dan dosen lainnya. You are the best lecture sir. Regrets for me can learn from you sir. Thank you very much.

Teman-teman seperjuangan selama kuliah ini, angkatan geo zero 7, jaya shiva jaya mahe. My best friend, idi, runi, icho,arya (ayo diselesaikan kuliah mu bro), teddy, gede. Without u guys, maybe I can't be like this. And all for geo zero 7, I wish success to all and someday we can meet and sharing bout our life. And laughing together again ☺.

Kakak-kakak tingkat selama di sini, terima kasih atas semua bimbingannya, sehingga saya yang bodoh ini, bisa sedikit demi sedikit belajar dan memahami baik tentang akademik dan tuntunannya dalam berorganisasi. Adik-adik tingkat, teruslah berjuang untuk menyelesaikan apa yang telah kalian mulai. Buatlah kedua orang tua kalian bahagia. So happy I am, can know u guys, you all ROCK!!!!

Masih banyak lagi yang ingin ku sebutkan, tapi huruf-huruf ini tak akan pernah cukup, kata-kata pun tak akan ada habisnya untuk menuliskannya dalam lembaran kertas ini.

See you, until we meet again ☺

KATA PENGANTAR

Puji Syukur penulis panjatkan kepada Tuhan Yang Maha Esa, karena atas berkat rahmat dan karunia-Nya penulis dapat menyelesaikan skripsi yang berjudul "**Pembuatan Algoritma Foto Panoramic Secara Otomatis Menggunakan Kamera IP**", dimana penulisan skripsi ini disusun untuk memenuhi salah satu syarat untuk meraih gelar Sarjana Teknik pada Jurusan Teknik Geodesi Fakultas Teknik Sipil dan Perencanaan Institut Teknologi Nasional Malang.

Penulisan ini tidak akan dapat terselesaikan tanpa bantuan dan dukungan berbagai pihak. Oleh karena itu, penenliti ingin mngucapkan terima kasih yang sebesar-besarnya kepada :

1. Bapak Ir. Agus Darpono, MT, selaku Ketua Jurusan Teknik Geodesi Institut Teknologi Nasional Malang.
2. Bapak M. Edwin Tjahjadi, ST, M.Geom.Sc., Ph.D, selaku Dosen Pembimbing I.
3. Bapak Ir.Leo Pantimena,MSc, selaku Dosen pembimbing II.
4. Segenap dosen, staff pengajar dan *recording* Jurusan Teknik Geodesi Fakultas Teknik Sipil dan Perencanaan Institut Teknologi Nasional Malang.
5. Kedua orang tua ku yang sudah tiada, dan seluruh kakak-kakak ku.
6. Semua teman-teman, *Team Rapid Mapping* dan Deliniasi Garis Pantai, Geodesi ITN 2005-2011, dalam kerjasama dan dukungannya.
7. Semua pihak yang telah membantu terlaksananya penelitian dan penulisan laporan ini, yang tidak dapat disebutkan satu persatu.

Penulis menyadari masih banyak kekurangan dalam penulisan laporan penelitian ini. Oleh karena itu, penulis mengharapkan kritik dan saran yang membangun dari pembaca., dan semoga laporan ini dapat berguna bagi pembacanya.

Malang, 23 September 201

Penulis

DAFTAR ISI

Halaman Judul

Lembar Persetujuan

Lembar Pengesahan

Abstraksi

Pernyataan Keaslian Skripsi

Lembar Persembahan

Kata Pengantar

Daftar Isi

Daftar Gambar

BAB I PENDAHULUAN **1**

I.1 Latar Belakang 1

I.2 Identifikasi Masalah 2

I.3 Tujuan Penelitian 2

I.4 Batasan Masalah 2

I.5 Manfaat Penelitian 3

BAB II DASAR TEORI **4**

II.1 *Corner Detector* 4

II.1.1 *Moravec Corner Detection* 5

II.1.2 *Harris Corner Detection* 6

II.1.2.1 Konvolusi 6

II.1.2.2 Prewitt 8

II.1.2.3 *Blur* 9

II.1.2.4 Operator Plessey 11

II.1.2.5 *Non-max Suppression* 12

II.1.2.6 *Thresholding* 12

II.2 *Image Matching* 15

II.2.1 Korelasi Silang 16

II.2.2 *Normalized Cross Correlation* 18

II.3 RANSAC 21

II.4 *Image Blending* 24

BAB III TAHAP PELAKSANAAN **29**

III.1 Persiapan 29
III.1.1 Materi Penelitian 29
III.1.2 Peralatan Penelitian 29
III.2 Langkah Penelitian 30
III.3 Penjelasan Tahap Penelitian 31
III.3.1 Input Data 33
III.3.1.1 Pelaksanaan Pengambilan Data Kamera IP 33
III.3.1.2 Ekstrak Video Kamera IP 40
III.3.2 *Harris Corner Detection* 44
III.3.3 *Correlation* 47
III.3.4 RANSAC 50
III.3.5 *Blending* 53
III.3.6 Citra Mosaik Foto 57
BAB IV HASIL PENELITIAN DAN PEMBAHASAN 60
IV.1 Hasil Penelitian 60
IV.2 Pembahasan 65
BAB V KESIMPULAN DAN SARAN 67
V.1 Kesimpulan 67
V.2 Saran 68
Daftar Pustaka
Lampiran

DAFTAR GAMBAR

Gambar 2.1 Implementasi Moravec Operator pada *Blok Test Image* 6

Gambar 2.2. Perhitungan konvolusi secara grafis 8

Gambar 2.3 Operator Prewitt 8

Gambar 2.4 Konvolusi 8

Gambar 2.6 Proses Gaussian *Blur* 11

Gambar 2.7 Sepasang citra dalam bentuk visual 19

Gambar 2.8 (a) proses korelasi 2 foto sebelum RANSAC 23

(b) hasil korelasi 2 foto yang telah di RANSAC 23

Gambar 2.9 Representasi proses blending antara dua buah foto 26

Gambar 3.1 desain *interface panoramik* 32

Gambar 3.2 *Window IP Device Setup* 34

Gambar 3.3 *Window Scan Camera* 35

Gambar 3.4 Kamera terdeteksi 35

Gambar 3.5 *Query* pada kamera 35

Gambar 3.6 *IP Device Setup* 36

Gambar 3.7 Mengubah Status Koneksi Kamera 36

Gambar 3.8 Memulai *Monitoring* Kamera 37

Gambar 3.9 Melihat Hasil *Monitoring* 37

Gambar 3.10 *GV-NVR system* untuk melihat dan menyimpan hasil *monitoring* 38

Gambar 3.11 *Save Video* 38

Gambar 3.12 *Window Save as Avi* 39

Gambar 3.13 *Window Save Avi File* 39

Gambar 3.14 *Window Merging List* 39

Gambar 3.15 Buka Program *Free Video to JPEG Converter* 40

Gambar 3.16 *Window Free Video to JPEG Converter* 40

Gambar 3.17 *Add Files* Video 41

Gambar 3.18 *Window Conversion Progress* 41

Gambar 3.19 Hasil Ekstraksi Video (*.Jpg) 42

Gambar 3.20 *resource* data *.*JPG* 42

Gambar 3.21 tampilan foto pada *interface* 44

Gambar 3.22 hasil proses *Harris Corner Detection* 46

Gambar 3.23 hasil proses korelasi 49
Gambar 3.24 hasil proses RANSAC 52
Gambar 3.25 (a) hasil *blend* 54
(b) *seam* yang masih terlihat 55
Gambar 3.26 (a) dialog penyimpanan 58
(b) hasil citra panoramic 59
Gambar 4.1 Tampilan program pembuatan foto panoramic 60

BAB I

PENDAHULUAN

1.1 Latar Belakang

Photogrametry adalah sebuah teknik untuk mendapatkan informasi mengenai posisi, ukuran dan bentuk dari sebuah objek dengan mengukur foto tersebut secara langsung *(K.B.Atkinson, 2001).* Secara umum *photogrametry* dikenal dengan *aerial photogrametry* (foto udara) dan *close range photogrametry.* Foto udara didapatkan dengan menempatkan kamera pada pesawat. Teknik *close range photogrametry* digunakan ketika jarak suatu objek dengan kamera kurang dari 100 meter *(K.B.Atkinson, 2001).*

Kekurangan dari penggunaan kamera adalah terbatasnya daya tangkap suatu objek atau luasan yang akan diukur, untuk itu perlu dibuat suatu mosaik foto. Mosaik foto terdiri dari foto-foto yang dijadikan satu menjadi sebuah foto dengan syarat memiliki pertampalan atau *overlapping* pada foto yang akan dijadikan mosaik. Salah satu yang juga termasuk dalam mosaik foto adalah foto panorama. Keuntungan foto panorama adalah dapat menampilkan informasi dan ciri-ciri suatu obyek yang lebih banyak daripada foto berukuran biasa. Selain itu foto panorama juga berpotensi untuk mengetahui ukuran 3D suatu pemandangan atau untuk mengestimasi suatu lokasi *(Fay Huang et al., 2008).*

Proses dalam membuat foto panorama secara otomatis dengan membuat sebuah program dapat mengefisiensikan waktu untuk melakukan proses tersebut. Banyaknya metode yang digunakan dalam pembentukan foto panorama yang akan berpengaruh pada mosaik foto panorama yang akan dihasilkan. Oleh karena itulah

penulis melakukan penelitian pembuatan mosaik foto panorama secara otomatis dengan memilih sebuah metode dalam setiap prosesnya. Secara garis besar metode – metode tersebut yaitu penentuan titik fitur pada kedua foto dengan menggunakan metode *Harris Corner Detection*, melakukan korelasi antara foto kiri dan kanan dari titik fitur yang telah didapatkan dengan metode *Image Matching*, mengeliminasi kesalahan yang terjadi pada saat melakukan korelasi dengan proses homography pada metode *Random Sample Consensus (RANSAC)*, serta penggabungan antara kedua foto menjadi 1 foto panoramik dengan metode *Blending*. Adapun semua metode yang akan digunakan pada penelitian ini mengacu pada *Accord.net* karena pada *Library* tersebut secara spesifik telah melakukan penggabungan 2 foto menjadi 1 foto panoramik.

1.2 Identifikasi Masalah

Membuat algoritma untuk melakukan proses foto panorama secara otomatis dari 2 (dua) buah foto yang bertampalan dengan menggunakan metode – metode yang terdapat pada *Library Accord.net (Souza. Cesar., 2009-2010).*

1.3 Tujuan Penelitian

Tujuan penelitian ini adalah membuat algoritma pembuatan foto panorama yang dapat dilakukan secara otomatis dari 2 (dua) foto dengan menggunakan *library* dari *Accord.Net.*

1.4 Batasan Masalah

Batasan masalah dari penelitian ini adalah pembuatan algoritma untuk melakukan proses foto panorama secara otomatis yang mengacu pada metode – metode yang digunakan pada *Library Accord.net* (*Souza. Cesar., 2009-2010).*

1.5 Manfaat Penelitian

Manfaat yang diharapkan dari penelitian ini adalah dapat mengolah foto panoramik secara otomatis dari foto - foto stereo agar dapat mempercepat proses pengolahan dalam membuat foto panorama.

BAB II

DASAR TEORI

Pembuatan algoritma untuk membentuk suatu panorama atau mosaik baru dengan menggabungkan beberapa foto *(image stitching)* sudah lama dan dikembangkan dalam hal yang sangat luas dalam berbagai bidang ilmu. Berbagai metode telah digunakan untuk mendapatkan algoritma *image stitching* yang diharapkan dapat membentuk panorama atau mosaik foto dengan kualitas yang baik.

Dalam pembuatan panorama tersebut, banyak metodologi yang telah di gunakan sesuai dengan kebutuhan dan keperluan dari mosaik tersebut. Tetapi hal utama yang harus diperhatikan adalah meregistrasikan titik ikat *(tie point)* pada foto yang bertampalan secara manual maupun otomatis.

Adapun beberapa metode yang dapat digunakan dalam membuat panorama secara otomatis yaitu *corner detector, correlation, Random Consensius Sample*, serta *blending*. Metode - metode tersebut akan dijelaskan secara satu persatu, namun secara garis besar pemilihan metode tersebut dikarenakan pada tiap – tiap metodenya mempunyai keunggulan yang dapat membantu dalam proses pembuatan algoritma panorama secara otomatis.

2.1 Corner Detector

Konsistensi penyaringan tepi gambar sangat penting untuk interpretasi citra 3D dengan menggunakan fitur pelacakan algoritma. Untuk memenuhi daerah gambar yang berisi fitur tekstur dan terisolasi, gabungan sudut dan tepi detektor berdasarkan

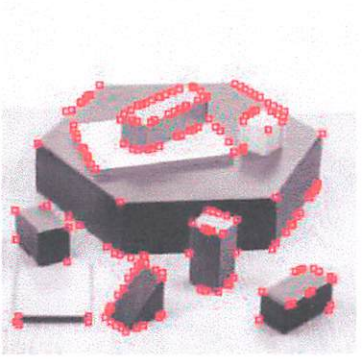
fungsi lokal korelasi otomatis dapat digunakan, dan ditampilkan untuk melakukan konsistensi yang baik pada gambar alam (Harris dan Stephens,1988).

Deteksi sudut *(Corner Detection)* atau terminologi yang lebih umum yaitu *Interest Point Detection*, adalah pendekatan yang digunakan dalam *Computer Vision* untuk mengekstrak jenis fitur tertentu dari sebuah gambar. Deteksi sudut sering digunakan untuk deteksi gerakan, pencocokan gambar, pelacakan, gambar mosaic, *panorama stitching*, pemodelan 3D dan pengenalan objek *(Wikipedia, 2011)*. Berikut ini akan dijelaskan beberapa deteksi sudut yang pernah ada antara lain *Moravec Corner Detection* serta *Harris Corner Detection*.

2.1.1 Moravec Corner Detection

Salah satu penemu pertama *Interest Point Detection* adalah Hans P. Moravec pada tahun 1977 dalam sebuah penelitian. Moravec mendefinisikan konsep *Interest Point Detection* pada gambar dan menyimpulkan *Interest Point Detection* dapat digunakan untuk menemukan area yang cocok di gambar yang berbeda.

Penemuan Moravec dianggap sebagai deteksi sudut karena mendefinisikan titik-titik yang menarik *(Interest Points)* sebagai titik dimana ada variasi intensitas yang besar dari berbagai arah. Yang patut di perhatikan bahwa Moravec secara spesifik tidak tertarik dalam menentukan sudut, tapi hanya wilayah-wilayah yang berbeda dalam sebuah gambar yang dapat digunakan untuk mendata bingkai gambar secara berurutan (Moravec,1977).



Gambar 2.1 : Implementasi Moravec Operator pada *Blok Test Image*

2.1.2 Harris Corner Detection

Harris Corner Detector dikembangkan oleh Chris Harris dan Mike Stephens pada tahun 1988 sebagai langkah pengolahan untuk membangun interpretasi dari robot yang didasarkan pada urutan gambar. Seperti Moravec, mereka membutuhkan metode untuk menyamai poin yang sesuai dalam bingkai gambar secara berurutan, namun lebih fokus dalam pelacakan baik untuk sudut dan tepi antara bingkai.

Harris dan Stephens memperbaiki detektor sudut yang digunakan Moravec dengan mempertimbangkan diferensial dari nilai sudut yang berhubungan dengan arah langsung. *Harris Corner Detector* menghitung matriks lokal rata-rata dari gradien gambar, dan kemudian menggabungkan nilai Eigen matriks untuk menghitung ukuran sudut, dimana nilai maksimum menunjukkan posisi sudut (Harris dan Stephens,1988). Berikut adalah langkah-langkahnya:

2.1.2.1 Konvolusi

Konvolusi adalah perkalian total dari dua buah fungsi f dan f yang

didefinisikan dengan:

$$f * h = \int_0^T f(t)h(T-t)dt \qquad (2.1)$$

Untuk fungsi f dan h yang berdimensi 2, maka konvolusi dua dimensi didefinisikan dengan:

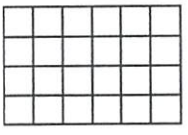
$$f * h = \int_0^{T_x} \int_0^{T_y} f(x,y)h(T_x - x, T_y - y)dxdy \qquad (2.2)$$

Konvolusi 2D inilah yang banyak digunakan pengolahan citra digital, sayangnya rumus diatas sangat sulit diimplementasikan menggunakan komputer, karena pada dasarnya komputer hanya bisa melakukan perhitungan pada data yang diskrit sehingga tidak dapat digunakan untuk menghitung intregral di atas. Konvolusi pada fungsi diskrit f(n,m) dan h(n,m) didefinisikan dengan:

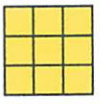
$$y(k1,k2) = \sum_{n-1}^{Tn} \sum_{m-1}^{Tm} f(k1+n, k2, m)h(n,m) \qquad (2.3)$$

Perhitungan konvolusi semacam ini dapat digambarkan dengan:

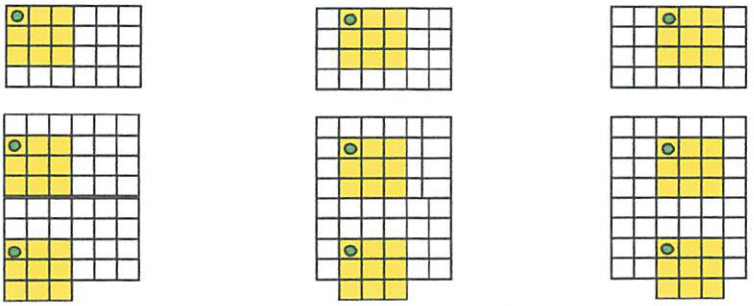
F(x,y)



h(x,y)



Bila ingin dihitung y = f * h, maka proses perhitungannya dapat dilakukan dengan:



Gambar 2.2. Perhitungan konvolusi secara grafis

Filter pada citra pada bidang spasial dapat dilakukan dengan menggunakan konvolusi dari citra (I) dan fungsi filternya (H), dan dituliskan dengan:

(2.4)

Dan dirumuskan dengan:

(2.5)

dimana m,n adalah ukuran dari fungsi filter dalam matrik

2.1.2.2 Prewitt

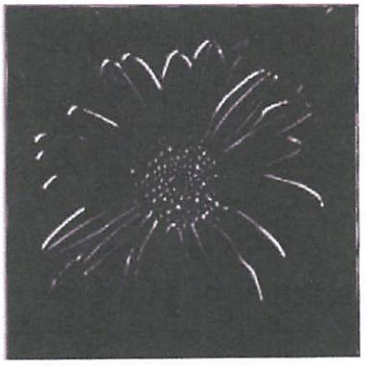
Konvolusi pada citra untuk mendeteksi tepi secara horizontal dan vertical, dengan menggunakan Operator Prewitt.

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Turunan-x Turunan-y

Gambar 2.3 Operator Prewitt

Dalam Corner Detection, diperlukan 3 buah Image turunan yaitu Turunan X, Turunan Y, Turunan X*Y



Gambar 2.4 Konvolusi

2.1.2.3 *Blur*

Proses *Blur* merupakan pengolahan citra agar suatu citra terlihat kabur. Prosesnya sana dengan proses konvolusi, namun dengan operator konvolusi atau kemel yang berbeda. Untuk proses *blur* ini menggunakan operator Gaussian, dimana operator Gaussian yang dianalogikan sebagai *window* 3x3 tersebut didapat dari perhitungan rumus:

$$G\,(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \qquad (2.6)$$

Dengan σ merupakan sigma yang merupakan variabel penting dalam penentuan tingkat *blur* citra. Dengan $\sigma=1$ maka akan didapat sebuah operator Gaussian seperti ini:

G(-1,-1)=0.0585498 G(0,-1)=0.0965324 G(1,-1)=0.0585498

G(-1,0)=0.0965324 G(0,0)=**0.159155** G(1,0)=0.0965324

G(-1,1)=0.0585498 G(0,1)=0.0965324 G(1,1)=0.0585498

Namun pada pendeteksian sudut kali ini digunakan σ bernilai 1.4.

Setelah operator Gaussian ditentukan, maka proses konvolusi dengan operator tersebut dapat dijalankan dengan menghitung nilai suatu piksel yang menjadi titik tengah dari suatu *window* 3x3 dengan menggunakan perhitungan konvolusi sebagai berikut:

$$I'(x,y) = \sum_{i=-n}^{n} \sum_{j=-m}^{m} G(i,j) I\,(x+i, y+j) \qquad (2.7)$$

Dimana I merupakan citra awal. G adalah fungsi Gaussian dan I' merupakan citra hasil.

Citra hasil deteksi tepi dari proses sebelumnya kemudian di-*Blur*. Proses *Blur* sendiri dimulai dari pengambilan citra degredasi dari langkah 1. Seandainya I adalah citra *(image)*, maka ∂I/∂x adalah turunan x dan ∂I/∂y adalah turunan y. dalam proses *blur* kita mengkalkulasikan ketiga variabel dibawah ini:

$$A = \left(\frac{\partial 1}{\partial x}\right)^2 \otimes w \qquad B = \left(\frac{\partial 1}{\partial y}\right)^2 \otimes w$$

$$C = \left(\frac{\partial 1.\partial 1}{\partial x.\partial y}\right) \otimes w \qquad (2.8)$$

Dimana *w* adalah perkalian matriks Gaussian. Tiga citra hasil konvolusi dengan menggunakan operator Prewitt pada proses sebelumnya akan di-*blur* dengan operator

Gaussian tersebut, sehingga akan tampak citra yang sedikit lebih buram.

Gambar 2.6 Proses Gaussian *Blur*

2.1.2.4 Operator Plessey

Untuk setiap titk pada citra dibangun matriks 2x2 *M* dan mengkalkulasikan operator Plessey.

$$M = \begin{bmatrix} A & C \\ C & B \end{bmatrix} \tag{2.9}$$

Dimana A adalah citra Turunan X yang telah di-*Blur*, B citra Turunan Y yang telah di-*Blur*, dan C adalah Turunan X*Y yang telah di-*Blur* dari proses Gaussian *Blur* sebelumnya.

Setelah itu baru nilai Plessey dari masing – masing piksel bisa didapatkan dengan menghitung melalui rumus berikut:

$$\boldsymbol{R(x,y) = \det(M) - k^{*}\,[tr(M)]^{2} \quad \text{dan k} = 0.04} \tag{2.10}$$

det(M) merupakan Determinan dari matriks M yaitu *(A*D)-(C*C)* dan *tr(M)* merupakan *Trace* dari matrix M yaitu *(A+B)*.

2.1.2.5 *Non-max suppression*

Proses Non Maximum Suppression yang mirip dengan proses *thinning* (perampingan) dilakukan untuk menentukan piksel tepi dengan posisi paling mendekati lokasi terjadinya perubahan nilai piksel diantara banyaknya piksel tepi yang terdeteksi. Dimana pada umumnya, perubahan nilai piksel berada pada pusat kumpulan piksel tepi [Nixon dan Aguado, 2002].

Bila nilai Plessey dari sebuah titik tertentu yang dihasilkan adalah nilai maksimum lokal dalam sebuah wilayah 3x3, maka dapat disimpulkan sementara bahwa titik ini adalah sebuah titik pojok.

2.1.2.6 *Thresholding*

Thresholding merupakan salah satu teknik segmentasi yang baik digunakan untuk citra dengan perbedaan nilai intensitas yang signifikan antara latar belakang dan objek utama. Dalam pelaksanaannya *Thresholding* membutuhkan suatu nilai yang digunakan sebagai nilai pembatas antara objek utama dengan latar belakang, dan nilai tersebut dinamakan dengan *threshold*.

Thresholding digunakan untuk mempartisi citra dengan mengatur nilai intensitas semua piksel yang lebih besar dari nilai *threshold T* sebagai latar depan dan yang lebih kecil dari nilai *threshold T* sebagai latar belakang. Biasanya pengaturan nilai *threshold* dilakukan berdasarkan histogram *grayscale* (Gonzales dan

Woods, 2002).

Jumlah titik-titik yang akan dikumpulkan dari hasil langkah 4 akan membengkak, dan tidak semua titik merupakan titik-titik yang penting. Dari semua titik yang ditemukan jumlah ini akan diturunkan dengan cara mengambil sebagian dari titik-titik yang memiliki nilai Plessey terbesar.

Dari penjelasan diatas, tanpa mengesampingkan turunannya, kita akan mengasumsikan citra 2 dimensi berbentuk *grayscale* yang akan digunakan. Sebagai contoh gambar akan diberikan identifikasi *I*. Dan dengan mempertimbangkan untuk mengambil sebuah patch gambar atas wilayah tersebut *(u,v)* dan bergeser dengan *(x,y)*. dengan menggunakan *sum of squared differences* antara dua patch, dinotasikan *S*, dapat dituliskan sebagai berikut :

$$S(x,y)=\sum_u \sum_v w(u,v)\big(I(u+x,v+y)-I(u,v)\big)^2 \qquad (2.11)$$

Dimana $I\ (u+x,\ v+y)$ dapat dicari dengan menggunakan deret *Taylor*. Misalkan I_x dan $_yI$ menjadi turunan parsial , sehingga dapat dituliskan :

$$I(u+x,v+y) \approx I(u,v)+I_x(u,v)x+I_y(u,v)y \qquad (2.12)$$

Dan akan menghasilkan pendekatan

$$S(x,y) \approx \sum_u \sum_v w\,(u,v)(I_x(u,v)x+I_y(u,v)y)^2 \qquad (2.13)$$

Yang dapat dituliskan dalam bentuk matriks :

$$S(x,y) \approx (x\ y)A\begin{pmatrix}x\\y\end{pmatrix} \tag{2.14}$$

Dimana A adalah tensor struktur yang digunakan dalam metode *Harris Corner Detector* adalah

$$A = \sum_u \sum_v w\,(u,v)\begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix} = \begin{bmatrix} \langle I_x^2 \rangle & \langle I_xI_y \rangle \\ \langle I_xI_y \rangle & \langle I_y^2 \rangle \end{bmatrix} \tag{2.15}$$

Matriks ini adalah matriks Harris, dan kurung sudut menunjukkan rata-rata (yaitu penjumlahan atas *(u, v)).))*. Jika jendela bundar (atau sirkuler jendela tertimbang, seperti Gaussian) yang digunakan, maka respon akan menggunakan isotropik.

Sebuah sudut (atau secara umum *interest point*) ditandai oleh variasi yang besar dari *S* ke segala arah dari vector *(x y)*. Dengan menganalisis nilai – nilai eigen dari *A*, karakterisasi ini dapat dinyatakan dalam cara berikut: *A* harus memiliki dua "besar" nilai eigen untuk *interest point*. Berdasarkan besaran nilai eigen (λ), kesimpulan berikut dapat dibuat berdasarkan argumen ini:

1. Jika $\lambda_1 \approx 0$ dan $\lambda_2 \approx 0$ maka piksel *(x, y)* tidak memiliki fitur yang menarik.
2. Jika $\lambda_1 \approx 0$ dan λ_2 memiliki beberapa nilai positif yang besar, maka tepi ditemukan.
3. Jika $\lambda\ \lambda_1$ dan $_2$ memiliki nilai positif yang besar, maka sudut ditemukan.

Dengan *Harris Corner Detector*, pencocokan antara gambar tepi pada piksel-piksel cocok untuk stereo, karena diketahui epipolar geometri kamera. Namun untuk kamera yang bergerak, dimana gerakan kamera tidak diketahui, masalah lubang lensa

mencegah kita melakukan eksplisit pencocokan tepi. Hal ini dapat diatasi dengan pemecahan gerakan sebelumnya, tetapi masih dihadapkan dengan pelacakan tiap sisi piksel gambar dan memperkirakan lokasi 3D dari, sebagai contoh, *Kalman Filtering*. Pendekatan ini tidak efektif diperbandingkan dengan tepi-tepi gambar ke dalam segmen, dan pelacakan segmen ini sebagai fitur.

2.2. Image Matching

Image matching merupakan suatu proses dalam fotogrametri digital untuk mengidentifikasikan dan mengukur titik konjugasi secara otomatis pada dua atau lebih foto yang saling *overlap*. Ada sejumlah metode *image matching* yang dapat dipakai untuk keperluan proses restuisi foto yang selama ini diketahui orang. Schenk (1999) menguraikan dengan rinci ketiga metode yang sejauh ini banyak digunakan. Ketiga metode yang dimaksud adalah *area-based, feature-based,* dan *symbolic.*

1. *Feature-based matching* merupakan hubungan antara *feature* entitas yang diekstrak dari *gray level* citra sebenarnya. *Feature* yang dimaksudkan antara lain titik, tepi/garis dan daerah. (*Shenck,* 1999).
2. *Area-based matching* merupakan metode yang digunakan dalam menentukan ketepatan objek (*image matching*) antar dua buah foto yang *overlap.* Gruen (1985) menyimpulkan bahwa, jika pertampalan dinilai cukup maka pada citra yang sama nilai tersebut dapat diuraikan dengan suatu transformasi antar objek dalam kedua citra. *Area-based matching* mendasarkan hubungan antara dua *image* menurut kesamaan *grey level.* Teknik yang sering digunakan pada metode

ini adalah teknik korelasi silang (*Cross Correlation*) dan *Least Square Matching* (LSM).

2.2.1 Korelasi Silang (*Cross Correlation)*

Prinsip teknik koreksi silang *cross correlation* adalah mencari pasangan titik piksel antara foto referensi/foto kiri dengan foto pasangan/foto kanan. Pada foto kiri ditentukan jendela sasaran yang memuat titik piksel yang akan dicari pasangannya pada foto kanan. Pada foto kanan ditentukan daerah selidik yang mempunyai ukuran lebih besar dari pada daerah sasaran.

Algoritma untuk korelasi silang (*cross correlation) :*

1. Identifikasi *template*

   1.1 *Template* merupakan kumpulan *gray value* (m,n) yang digunakan sebagai referensi atau acuan dalam pencarian titik konjugasi. *Template* biasanya berukuran ganjil yaitu 3 x 3, 5 x 5, 7 x 7, dan seterusnya

   1.2 Identifikasi *template* diawali dengan menentukan objek yang akan dicari, kemudian secara otomatis sistem komputer akan merekam nilai *gray value template* sesuai dengan ukuran yang telah ditentukan.

2. Identifikasi *search window*

   2.1 Untuk mengidentifikasi *search window*, maka bantuan operator dibutuhkan untuk mencari lokasi yang diperkirakan sama dengan lokasi pada *template*. Kemudian, *search window* ditentukan lebih besar dari ukuran template. Di dalam *search window*, otomatis akan terdapat *matching windows* yang berukuran sama dengan *template*.

3. *Cross Correlation*

Proses *Cross Correlation* dimulai dari pojok kiri atas pada *search window* kemudian *matching window* bergerak ke kanan dengan *increment* 1 piksel sepanjang baris dan kolom kemudian kearah bawah sebanyak satu kolom dan bergerak lagi kearah kanan. Ketika pancarian dilakukan sistem komputer mencatat nilai korelasi (ρ) antara *matching window* dan *template*. Dari nilai korelasi yang dicatat sistem komputer dapat ditentukan mirip tidaknya kedua matrik pada *template* dan *matching window*. Apabila makin besar nilai ρ maka makin mirip bentuk kedua objek tersebut atau dapat dikatakan kedua objek tersebut merupakan titik yang sama.

Nilai korelasi antara dua kelompok dari *gray value* dihitung berdasarkan rumus matematis pada persamaan berikut :

$$\rho = \frac{\text{covariance}_{TA/SA}}{\text{standard dev}_{TA} \,.\text{standard dev}_{SA}} \tag{2.16}$$

$$\rho = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m}(g_T(i,j)-\bar{g}_T)(g_S(i,j)-\bar{g}_S)}{\sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m}(g_T(i,j)-\bar{g}_T)^2 \sum_{i=1}^{n}\sum_{j=1}^{m}(g_S(i,j)-\bar{g}_S)^2}} \tag{2.17}$$

ρ = koefisien korelasi

g_T, g_S = nilai keabuan pada *template* dan *search window*

$\bar{g}_T, \bar{g}_S$ = *mean gray value*

n, m = baris dan kolom pada *template* dan *search window*

TA = *Target Area, the template*

SA = *Search Area, the matching window*

Dari nilai korelasi tersebut dapat ditentukan mirip tidaknya kedua matriks tersebut. Makin besar nilai ρ mendekati 1, makin mirip bentuk kedua objek tersebut atau dapat dikatakan kedua objek tersebut merupakan titik yang sama.

2.2.2 *Normalized Cross Correlation* (NCC)

Normalized Cross Correlation (r) merupakan salah satu pengukuran titik konjugasi yang digunakan dalam fotogrametri *(Martince, 2011)*. Prinsip *normalized cross correlation* yaitu mencari pasangan titik piksel antara citra pertama dengan citra pasangan/citra kedua. Pada citra pertama ditentukan jendela sasaran *(template)* yang memuat titik piksel yang akan dicari pasangannya pada citra kedua. Pada citra kedua ditentukan daerah selidik *(search window)* yang mempunyai ukuran lebih besar daripada daerah sasaran *(template)*. Pada daerah selidik *(search window)* dengan ukuran yang sama dengan jendela/daerah sasaran *(template)*. *Matching window* ini bergerak *(moving window)* dengan *increament* 1 piksel sepanjang baris dan kolom di daerah selidik *(search window)*. Dihitung nilai korelasi (r) antara *template* dan *matching window*. Nilai korelasi antara dua kelompok data *gray value* dihitung berdasarkan rumus matematis pada persamaan berikut (*Mitchell* dan *Pilgrim, 1987*; *Schenk, 1999: Wolf dan Dewit, 2000; Campbell et al., 2008*) :

$$S_x^2 = \sum x_i^2 - \frac{(\Sigma x_i)^2}{n} \tag{2.18a}$$

$$S_y^2 = \sum y_i^2 - \frac{(\Sigma y_i)^2}{n} \tag{2.18b}$$

$$S_{xy} = \sum(x_i y_i) - \frac{(\Sigma x_i)(\Sigma y_i)}{n} \tag{2.18c}$$

$$\beta = \frac{S_{xy}}{S_x^2} \tag{2.18d}$$

$$\alpha = -\beta \frac{\Sigma x_i}{n} \tag{2.18e}$$

(2.18f)

Dimana :

n : jumlah data (baris x kolom)

r : koefisien *normalized cross correlation*

σ_T, σ_S : standar deviasi pada *template* dan *search image*

σ_{TS} : kovarian nilai *gray value* pada *template* dan *search image*

: nilai keabuan *template* dan *search image*

: nilai rata – rata dari *template* dan *search image*

R,C : baris dan kolom dari *image patches*

Gambar 2.7 Sepasang citra dalam bentuk visual

Apabila pada foto pertama ditentukan sebuah objek sebagai titik acuan pencarian, maka komputer harus menentukan objek tersebut pada foto kedua dengan mengamati sekumpulan nilai *gray value* pada kedua foto. Dalam domain digital, citra tersebut dipresentasikan sebagai variasi nilai piksel yang membentuk dimensi m x n piksel atau m = n piksel. Apabila pada foto pertama ditentukan *template* yang berdimensi 5 x 5 disekeliling titik objek maka *template* ini berisi sekumpulan (25) nilai piksel disekeliling titik acuan. Pada foto kanan dibentuk *matching window* yang memiliki dimensi yang sama dengan *template*, dimana *matching window* ini terdapat dalam *search window*. Sampai tahap ini akan diperoleh nilai dua buah matrik (*template* dan *matching window*) dengan dimensi yang sama.

Penempatan *matching window* dimulai dari posisi ujung kiri atas dari *search window*. Kemudian *matching window* akan bergeser menelusuri citra kolom demi kolom kearah kanan sampai mencapai batas dari *search window*. Setelah itu, *matching window* akan bergeser ke bawah sebanyak satu baris dan kembali menelusuri sepanjang baris tersebut ke arah kiri, demikian seterusnya. Proses penelusuran dilakukan sampai ke seluruh *search window*. Untuk setiap tahap penelusuran, nilai *c* dihitung dan dicatat oleh sistem komputer.

Koefisien *normalized cross correlation (c)* memiliki nilai antara $-1 \leq c \geq 1$. Nilai +1 diindikasikan sebagai korelasi yang sempurna dan nilai -1 diperoleh ketika ada kecocokan dari citra positif dan negatif. Nilai koefisien yang mendekati nilai 0 diidentifikasikan sebagai nilai yang tidak memiliki korelasi *(mismatch)*.

Dalam kasus bingkai terpisah yang didapat menggunakan kamera, ada persyaratan tambahan yaitu citra poin yang didapat berasal dari titik 3D yang sama mungkin harus diekstraksi. Oleh karena itu, hanya maxima lokal dari fungsi respon sudut yang dapat dianggap sebagai fitur. Presisi sub-piksel dapat dicapai melalui aproksimasi kuadrat dari daerah maxima lokal. Untuk kasus ini yang cocok adalah dengan menggunakan Gaussian. Pencocokan ini biasanya dilakukan dengan membandingkan hal kecil dan jendela berpusat di sekitar fitur melalui SSD atau NCC.

2.3 RANSAC

Sebuah paradigma baru, *Random Sample Consensus (RANSAC)*, memperkenalkan model untuk eksperimen data. RANSAC mampu menafsirkan atau merapikan data yang berisi persentase yang signifikan dari kesalahan-kesalahan yang menyolok, dengan demikian RANSAC idealnya cocok untuk aplikasi dalam analisis citra otomatis di mana interpretasi didasarkan pada data yang disediakan oleh kesalahan-rawan fitur detektor. Bagian utamanya adalah mendeskripsikan aplikasi dari RANSAC terhadap *Location Determination Problem (LDP)*. Memberikan sebuah gambaran terhadap *landmark* yang sudah diketahui lokasinya, menentukan bahwa titik di dalam ruang dari setiap gambar dapat diperoleh. Sebagai tanggapan terhadap kebutuhan RANSAC, hasil baru yang diperoleh pada jumlah minimum terhadap *landmark* diperlukan untuk mendapatkan sebuah solusi, dan algoritma disajikan untuk komputasi solusi minimum *landmark* ini dalam bentuk tertutup. Hasil

ini memberikan dasar bagi sebuah sistem otomatis yang dapat memecahkan LDP dalam kondisi dan analisis yang sulit (Martin dan Robert,1978).

Random Sample Consensus (RANSAC) secara konsep akan melibatkan dua aktifitas yang berbeda, yaitu : pertama, menemukan kesesuaian yang terbaik antara data dan satu model yang tersedia; kedua, menghitung nilai yang terbaik untuk parameter bebas dari parameter pendekatan (*Fischler* dan *Bolles, 1981*). Konsep ini membentuk paradigma yang ada didalam metode RANSAC yang dikenal dengan paradigma RANSAC. Ada tiga parameter yang dibahas dari paradigma RANSAC, yaitu :

1. Menentukan batas toleransi kesalahan yang digunakan untuk mendapatkan suatu model yang cocok.
2. Menentukan nilai maksimum yang dapat ditetapkan atau dipakai untuk suatu percobaan.
3. Menurunkan batas ketepatan yang dapat diterima.

(a)

(b)

Gambar 2.8 (a) proses korelasi 2 foto sebelum RANSAC

(b) hasil korelasi 2 foto yang telah di RANSAC

Untuk sebagian besar, tahap analisis (tahap secara umum) terfokus pada interpretasi dari sebuah model data. Secara konseptual, interpretasi mencakup dua kegiatan yang berbeda :

1. Ada sebuah permasalahan untuk menemukan yang terbaik antara data dan satu model yang digunakan (masalah klasifikasi).
2. Ada sebuah permasalahan pada komputasi terhadap nilai terbaik untuk parameter bebas dari model yang digunakan (masalah estimasi parameter).

Dalam prakteknya, kedua masalah tersebut tidak independen karena seringkali solusi untuk parameter estimasi diperlukan untuk memecahkan masalah klasifikasi.

Teknik klasik untuk estimasi parameter, seperti *least square*, optimalisasi (tergantung pada fungsi objek secara spesifik) yang pas dari fungsi deskripsi (model) untuk semua data yang disajikan. Teknik ini tidak mempunyai mekanisme khusus untuk deteksi dan membuang kesalahan yang mencolok. Teknik tersebut merupakan teknik standar yang bergantung pada asumsi *(smoothing assumption)* yang membutuhkan deviasi maksimum berbagai datum dari model yang diasumsikan seperti fungsi langsung dari ukuran data, dan terlepas dari ukuran data yang terhimpun, maka selalu ada nilai yang baik untuk dihaluskan dari kesalahan yang ada.

Pada kebanyakan permasalahan praktek estimasi parameter, *smoothing assumption* tidak memiliki data yang berisi kompensasi kesalahan yang mencolok. Untuk menyelesaikan situasi tersebut, beberapa penelitian telah dilakukan. Teknik yang biasanya digunakan adalah beberapa variasi yang pertama kali menggunakan semua data untuk memperoleh model parameter, lalu menentukan lokasi datum yang terbaik dari kesepakatan dengan model yang digunakan. Berasumsi dari kesalahan tersebut, penghapusan, dan iterasi dalam proses ini sampai deviasi maksimum tidak lebih dari beberapa batas yang telah ditetapkan atau sampai tidak ada lagi data yang cukup untuk diproses.

2.4 Image Blending

Sebuah gambar gradient *(Image Gradient)* adalah perubahan yang terarah pada intensitas atau warna dalam gambar. *Image gradient* dapat digunakan untuk mengekstrak informasi dari gambar. Dalam perangkat lunak grafis untuk mengedit

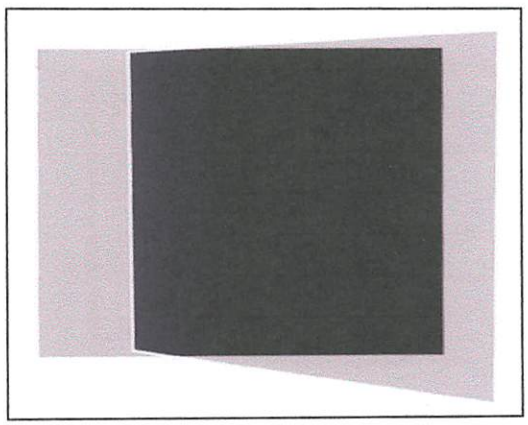
gambar dijital, istilah gradien digunakan untuk mencampur warna *(blend of color)* yang dapat dianggap sebagai gradasi dari nilai rendah ke tinggi, seperti penggunaan putih ke hitam dalam sebuah gambar. Sebutan lainnya yaitu perkembangan warna *(Color Progression)*.

Secara matematis, gradien dari fungsi dua variabel (fungsi intensitas citra) adalah pada setiap titik gambar vector 2D dengan komponen yang diberikan oleh derivatif dalam arah horizontal dan vertikal. Pada setiap titik gambar, titik-titik vektor gradien dalam arah meningkatkan kemungkinan intensitas terbesar, dan panjang dari vektor gradien sesuai dengan tingkat perubahan ke arah itu.

Karena fungsi intensitas dari gambar dijital hanya dikenal pada titik-titik diskrit, turunan dari fungsi intensitas berkelanjutan yang telah mendasari sampel pada titik-titik gambar. Dengan beberapa asumsi tambahan, turunan dari fungsi intensitas berkelanjutan dapat dihitung sebagai fungsi pada sampel intensitas, yaitu gambar dijital. Ternyata bahwa derivatif pada setiap titik tertentu adalah fungsi dari nilai intensitas di hampir semua titik gambar. Namun, perkiraan fungsi-fungsi derivatif dapat didefinisikan pada derajat yang lebih remdah atau lebih besar akurasinya.

Dalam beberapa kasus, seringkali garis tepi dari foto-foto yang digabungkan tidaklah berkesesuaian, sehingga perbedaan antara kedua foto terlihat sangat mencolok *(visible seam)*, adanya daerah yang kabur *(blur)* dan bayangan *(ghosting) (Szeliski, 2006)*. Untuk menghilangkan efek tersebut dibutuhkan sebuah metode *blending* (pencampuran) warna pada kedua foto.

Gradient blending bekerja dengan membentuk perubahan *gradual alpha channel* yang disambungkan pada pusat dari dua foto tersebut.



Gambar 2.9 Representasi proses blending antara dua buah foto

(Souza. Cesar., 2009-2010)

Gambar 2.9, pada bagian yang paling kiri menunjukkan foto sebelah kiri (foto pertama), sedangkan bagian yang paling kanan menunjukkan foto sebelah kanan (foto kedua). Bagian yang didalam (ditengah) menunjukkan daerah yang bertampalan, daerah yang akan dilakukan operasi *blending.*

Pada daerah yang bertampalan algoritma *blending* digunakan untuk menghitung besarnya kontribusi gambar pertama dan gambar kedua pada setiap piksel. Foto-foto dicampurkan dengan menggunakan rumus sebagai berikut *(Rankov et all, 2005;Porter and Duff, 1984)* :

$$N(x,y) = \alpha I(x,y) + (1-\alpha)\, C(x,y) \qquad (2.19)$$

Dimana :

N(x,y) = *Piksel foto yang baru*

α = *Nilai bobot*

C(x,y) = *Piksel foto pertama*

I(x,y) = *Piksel foto kedua*

Nilai bobot pada setiap piksel dihitung dengan menormalisasi terlebih dahulu daerah yang bertampalan. Proses normalisasi merupakan proses untuk merubah nilai intensitas jangkauan piksel yang memiliki nilai sangat beragam sehingga di dapatkan nilai maksimum 1 dan nilai minimum 0, dimana proses normalisai ini bersifat linear *(Wikipedia, 2011)*. Proses normalisasi akan menghasilkan daerah yang ber*gradient*, yang memiliki dimensi yang konstan. Maka, jika terdapat daerah yang bertampalan dari dua foto yang berbeda dikenai proses normalisasi, daerah tersebut akan memiliki karakteristik atau ciri-ciri yang sama pada lokasi spasial yang sama *(Wikipedia, 2011)*. Normalisasi $(\bar{d})$ dilakukan dengan membagi selisih nilai jarak antara piksel yang berada pada daerah yang bertampalan dan pusat dari dua foto tersebut *(p)* dengan selisih antara jarak piksel yang paling dekat dengan pusat foto *(N)*, yang dinyatakan dengan persamaan :

$$\bar{d} = \frac{p}{N} \tag{2.20}$$

Jarak *(d)* antara dua buah titik piksel *p(x,y)* dan *q(s,t)* dihitung dengan menggunakan rumus *(Gonzales. R. C., Woods. R. E, 2008)*,

$$d = [(x - s)^2 + (y - t)^2]^{\frac{1}{2}} \quad (2.21)$$

Dimana :

d : *Jarak antara dua buah piksel*

x : *Nilai koordinat titik pertama pada kolom piksel*

y : *Nilai koordinat titik pertama pada baris piksel*

s : *Nilai koordinat titik kedua pada kolom piksel*

t : *Nilai koordinat titik kedua pada baris piksel*

Dengan menghitung nilai jarak dari dua buah titik piksel, kita dapat mengukur tingkat kemiripan citra. Citra dengan nilai jarak yang lebih kecil dianggap memiliki tingkat kemiripan komposisi warna yang lebih tinggi atau lebih mirip dibandingkan dengan citra yang memiliki nilai jarak yang lebih besar *(Rahman, 2009)*. Selain dari itu piksel yang terdapat dekat dengan pusat dari citra memiliki bobot lebih berat/lebih besar dari bobot yang mendekati tepi gambar *(Szeliski, 2006)*.

BAB III

PELAKSANAAN PENELITIAN

Selain teori - eori yang berkaitan dengan penelitian, beberapa syarat juga harus dipenuhi untuk mer patkan tujuan yang maksimal, antara lain, peralatan dan bahan penelitian, pembuata diagram alir, dan pengolahan data. Dalam bab ini akan dibahas mengenai proses pela anaan penelitian mulai dari persiapan sampai pada pengolahan data yang diperoleh.

3.1. Persiapan

Sebelum mela kan sebuah penelitian diperlukan suatu persiapan yang matang guna kelancaran selan proses penelitian sampai penyajian hasil. Agar diperoleh hasil yang optimal maka ada beb pa hal yang harus dipersiapkan terlebih dahulu, yaitu :

3.1.1 Materi P elitian

Materi penelit , sebelumnya telah dijelaskan pada bab 2 yaitu dasar teori. Penelitian ini dilakuka nengacu pada dasar teori yang telah dijabarkan secara lengkap dan jelas.

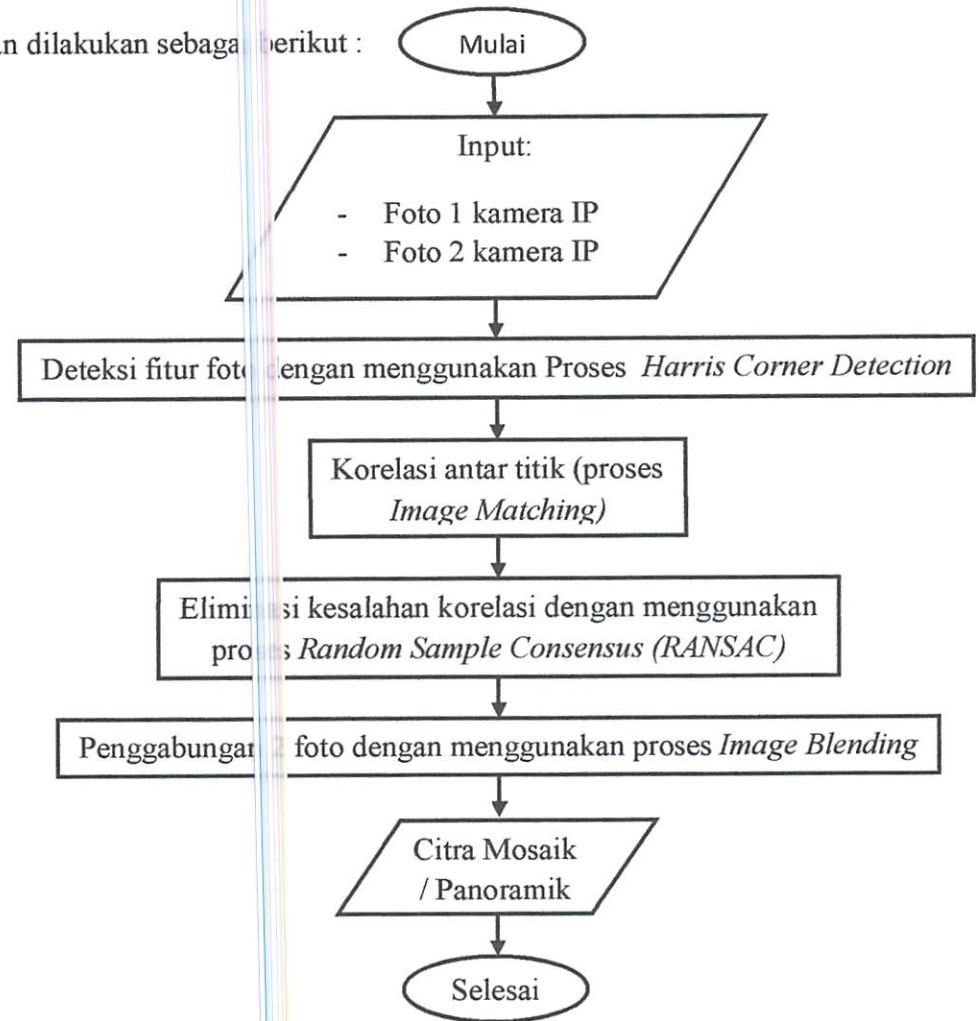
3.1.2 Peralatar enelitian

Adapun alat da ahan yang dibutuhkan dalam proses penelitian ini baik perangkat lunak (*software)* maupu perangkat keras (*hardware)* antara lain :

1. *Har* *are* terdiri dari
   - 1 nit laptop HP G42
2. *Soft* *re* yang digunakan adalah
   - M crosoft Word 2007
   - M crosoft Visual Studio 2010 aplikasi C#

3.2 Langkah Peneliti

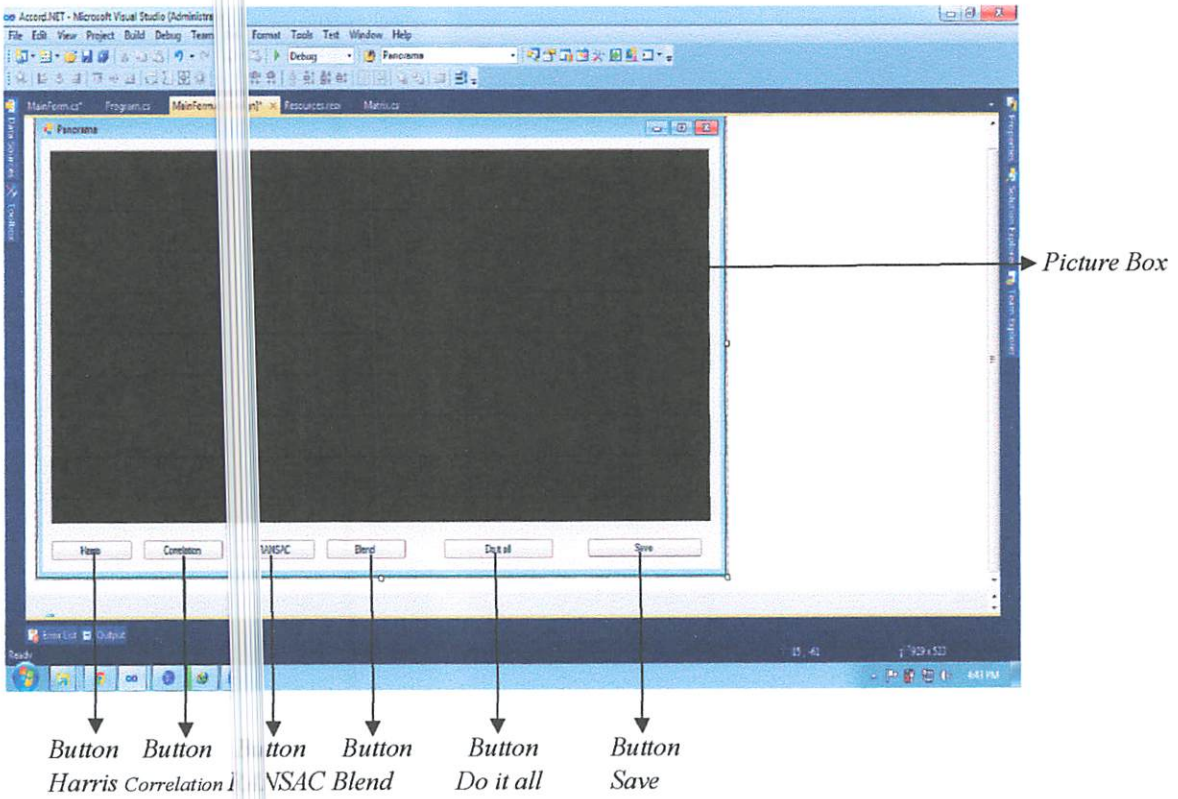
Dalam proses nelitian haruslah dibuat suatu kerangka pekerjaan yang sistematis agar mudah dipahar dan mempermudah dalam proses penelitian. Dengan acuan ***Accord.net*** dalam mel ukan penelitian ini, maka adapun langkah atau alur penelitian yang akan dilakukan sebaga erikut :



3.3 Penjelasan Tahapan Penelitian

Pembuatan algoritma ini menggunakan bahasa C# (dibaca "C *Sharp*") yang ada dalam perangkat lunak *Microsoft Visual Studio* 2010. C# adalah bahasa pemrograman yang dirancang untuk membangun berbagai aplikasi yang berjalan di *NET Framework*. Karena bahasa C# yang begitu sederhana, kuat, aman, dan berorientasi objek maka dimungkinkan untuk melakukan inovasi dalam pengembangan yang cepat sementara tetap mempertahankan ekspresi dan keanggunan dari bahasa C itu sendiri. C# dapat digunakan untuk membuat aplikasi pada *windows*, web XML, komponen distribusi, aplikasi *client-server*, aplikasi *database*, dan masih banyak lagi. Visual C# 2010 menyediakan sebuah editor kode maju, desain antar muka yang mudah digunakan, *debugger* yang terintegrasi, dan lain – lainnya yang dapat membuat lebih mudah untuk pengembangan aplikasi.

Dengan mengacu pada *Accord.net*, maka untuk membuat algoritma penelitian ini terlebih dahulu harus membuat sebuah desain *interface* pada aplikasi bahasa C#.



Gambar 3.1 desain *interface panoramik*

Pada desain *interface* yang terdapat pada gambar 3.1 ada beberapa fungsi *toolbox* yang digunakan antara lain *Button, Picture Box, Text Box*, dan lainnya. Semua fungsi yang ada di *toolbox* mempunyai kegunaan masing – masing. Pada pembuatan *interface* ini digunakan *Button* dan *PictureBox*. ***Button*** berfungsi untuk membuat perintah seperti OK, Cancel, dan lain sebagainya. Sedangkan ***PictureBox*** berfungsi untuk menampilkan gambar yang akan diproses.

Setelah membuat desain *interface* yang diinginkan, maka sesuai diagram alir di atas, dapat dijelaskan sebagai berikut:

3.3.1 Input Dat

3.3.1.1 Pelaksa an Pengambilan Data Kamera IP

Dalam pe ambilan data kamera IP terdapat langkah-langkah pelaksanaan pengambilan data a. Adapun langkah-langkah yang harus dilakukan untuk pengambilan data k era IP yaitu :

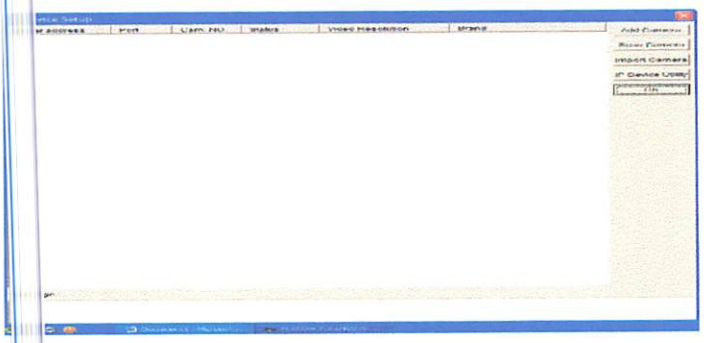
1. Persiapan, u k melakukan pekerjaan pengambilan data, harus dipersiapkan secara maksi l semua peralatan yang dibutuhkan dengan maksud tujuan untuk meminimalis kesalahan pada saat pengambilan data. Persiapan ini juga merupakan la kah melakukan rangkaian pada kamera IP, yaitu :
   a. Pasang ro *rack* pada mobil, lalu pasang dome di *Roof Rack* pada posisi yang telah atur.
   b. Masukkan mera IP ke dalam dome, lalu dibaut.
   c. Pasang ka power dan kabel LAN ke kamera IP.
   d. Sambungk kabel power kamera IP ke kabel roll, maka kamera IP telah menyala.
   e. 2 (dua) ka l LAN yang menyambung pada kamera IP disambungkan ke USB HUB -LINK. Dan label LAN yang 1 (satu) disambungkan ke laptop.
   f. Sambungk kabel power USB HUB D-LINK ke kabel roll.
   g. Untuk kab roll disambungkan pada inverter 700 Watt, dan inverter disambung n pada aki.
   h. Maka seles sudah persiapan untuk menggunakan kamera IP.

2. Integrasikan kamera IP pada laptop yang telah diinstal perangkat lunak *GV-NVR* untuk melakukan settingan pada kamera IP. Langkah untuk melakukan settingan pada kamera IP yaitu :

   a. Untuk memulai program *GV-NVR*, tekan *start GV-NVR GeoVision GV-NVR System.*

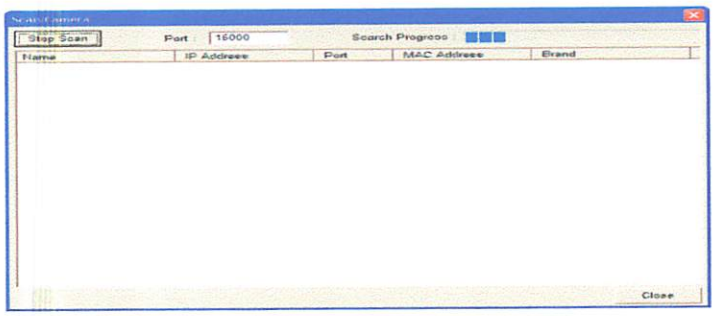
   b. Untuk penggunaan pertama kali, akan diminta untuk memasukkan ID dan *Password*. Untuk memudahkan masukkan ID : admin dan *Password* : admin. Lalu tekan OK.

   c. Akan muncul *window IP Device Setup* yang berfungsi untuk mencari kamera yang akan digunakan. Pilih *Scan Camera.*

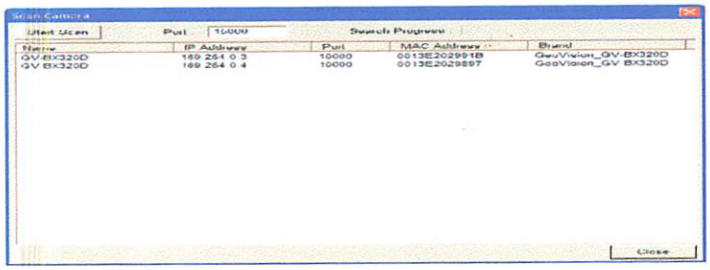


Gambar 3.2 *Window IP Device Setup*

   d. Lalu pada *window* selanjutnya tekan *Scan Camera* dan program ini akan secara otomatis mencari kamera yang sudah dihubungkan.

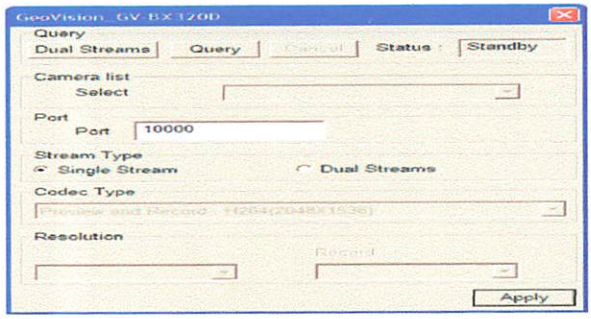


Gambar 3.3 *Window Scan Camera*



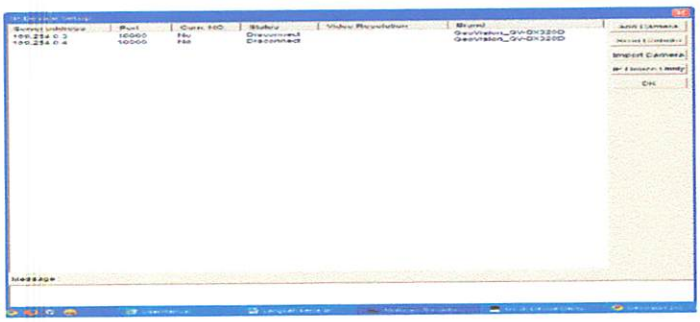
Gambar 3.4 Kamera terdeteksi

e. Klik 2x pada tiap kamera yang terdeteksi, maka akan otomatis melakukan *Query* pada masing-masing kamera.



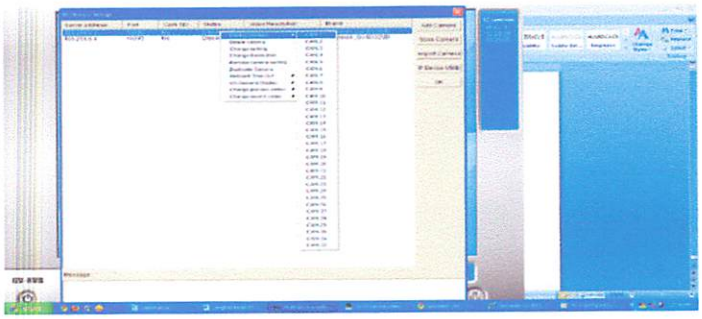
Gambar 3.5 *Query* pada kamera

f. Setelah itu tekan *Apply*, maka kamera akan muncul pada *IP Device Setup*.



Gambar 3.6 *IP Device Setup*

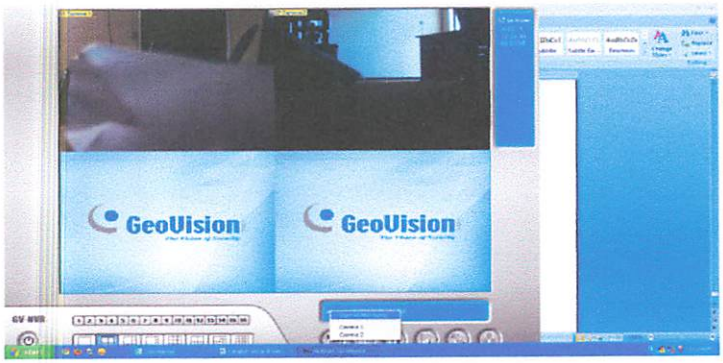
g. Klik kamera yang muncul, lalu klik kanan pada status *display* kamera pilih nomor kamera (CAM.1) untuk mengubah status kamera dari *disconnect* menjadi *connect*.



Gambar 3.7 Mengubah Status Koneksi Kamera

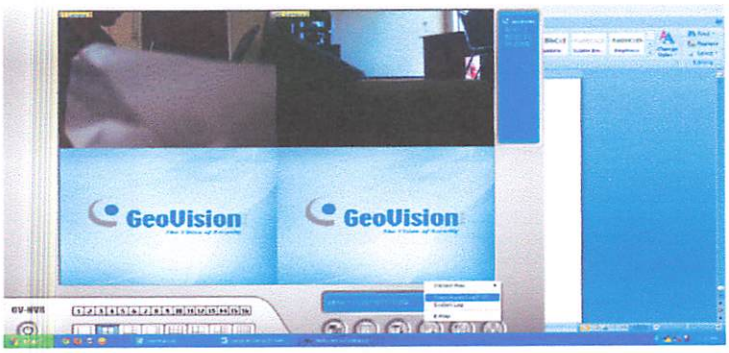
h. Lakukan pada tiap status kamera, jika semua sudah *connect* klik OK dan kamera sudah siap untuk melakukan *monitoring*.

3. Untuk memulai *monitoring*, klik *Monitor* lalu pilih kamera yang akan melakukan *monitoring*. Jika sudah selesai, klik kembali *monitoring* dan pilih *stop*.



Gambar 3.8 Memulai *Monitoring* Kamera

4. Untuk melihat hasil *monitoring*, klik pada *icon View Log*, lalu pilih *video/audio log*.

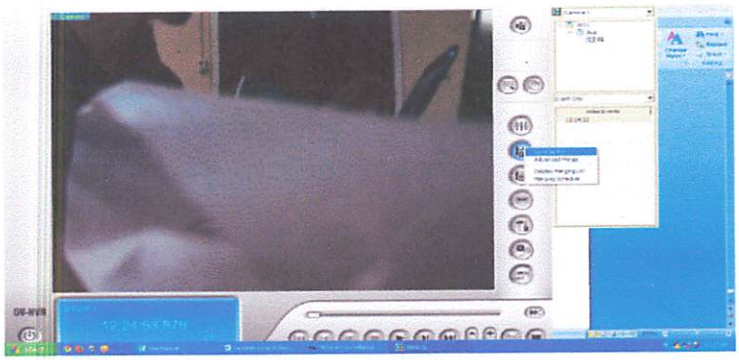


Gambar 3.9 Melihat Hasil *Monitoring*

5. Maka muncul program bagian dari *GV-NVR system* untuk melihat dan menyimpan hasil *monitoring*.

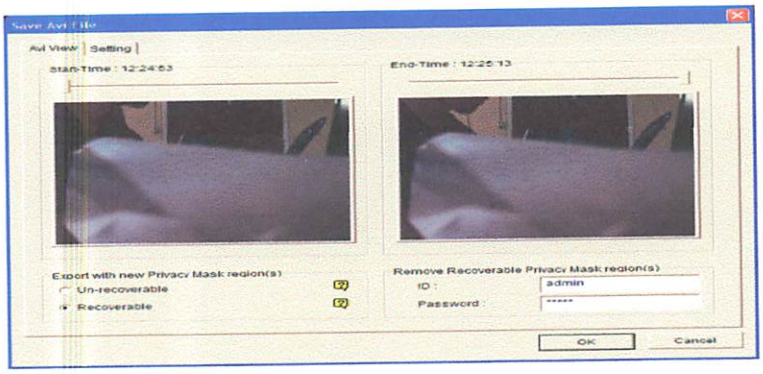
Gambar 3.10 *GV-NVR system* untuk melihat dan menyimpan hasil *monitoring*

6. Klik pada *icon Save Video*, pilih *Save as Avi.*

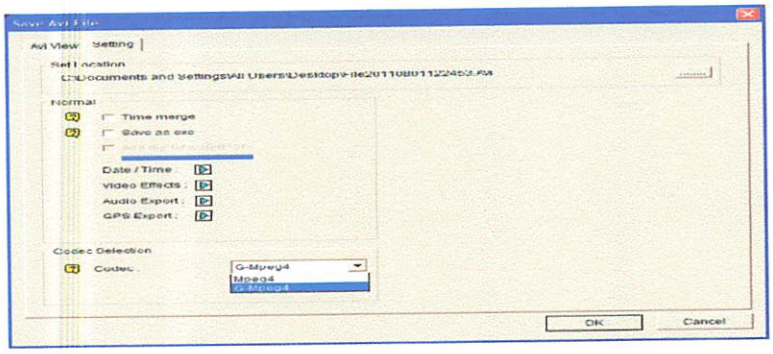


Gambar 3.11 *Save Video*

7. Pada *window Save avi File*, klik *tab setting* dan ganti *codec G-Mpeg4* menjadi *Mpeg4*. Ini bertujuan agar video Avi yang disimpan kompatibel dengan program ekstrasi yang digunakan pada penelitian kali ini.

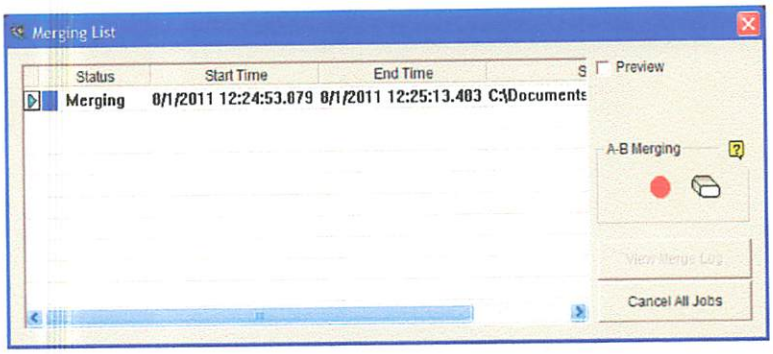


Gambar 3.12 *Window Save as Avi*



Gambar 3.13 *Window Save Avi File*

8. Klik OK jika sudah selesai maka akan muncul *window Merging List*, jika *Merging* sudah *Complete*, tutup *window* tersebut maka video hasil *monitoring* sudah berupa format Avi.



Gambar 3.14 *Window Merging List*

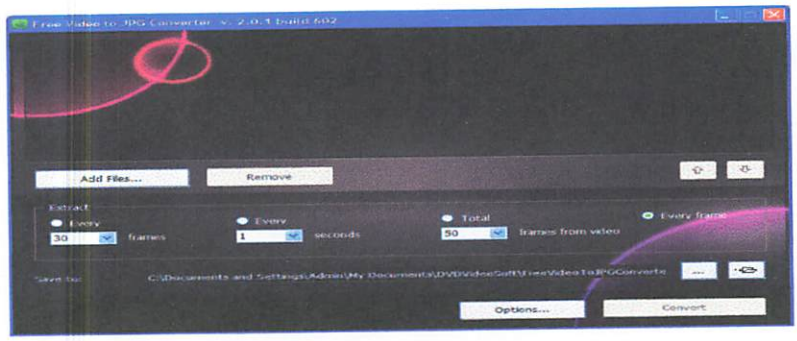
3.3.1.2. Ekstrak Video Kamera IP (*Converter Video to Jpg*)

Untuk ekstrasi video, program yang digunakan adalah *Free Video to JPEG Converter Version 2.0.1.602*. Program ini bisa didapatkan dengan cara mendownload di berbagai situs di internet. *install* program ini pada komputer atau laptop untuk menggunakannya. Langkah-langkah ekstraksi video sebagai berikut :

1. Buka program *Free Video to JPEG Converter.*

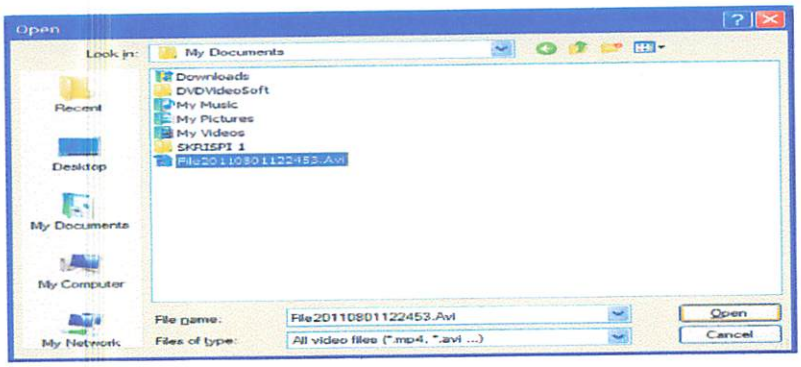
Gambar 3.15 Buka Program *Free Video to JPEG Converter*

2. Maka program *Free Video to JPEG Converter* dapat digunakan.



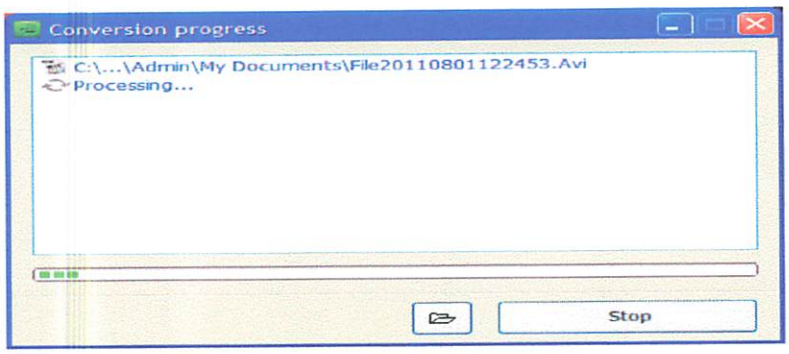
Gambar 3.16 *Window Free Video to JPEG*

3. Setelah itu klik *add files* untuk memasukkan file video yang ingin di ekstrak.



Gambar 3.17 *Add Files* Video

4. pilih video yang ingin di ekstrak setelah itu klik *Open*.
5. Setelah itu pilih hasil ekstraksi video yang diinginkan pada kolom *Extract*. Untuk kali ini dipilih hasil untuk *every frame*.
6. Jika sudah selesai langsung klik tombol *convert* untuk memulai ekstraksi data foto.

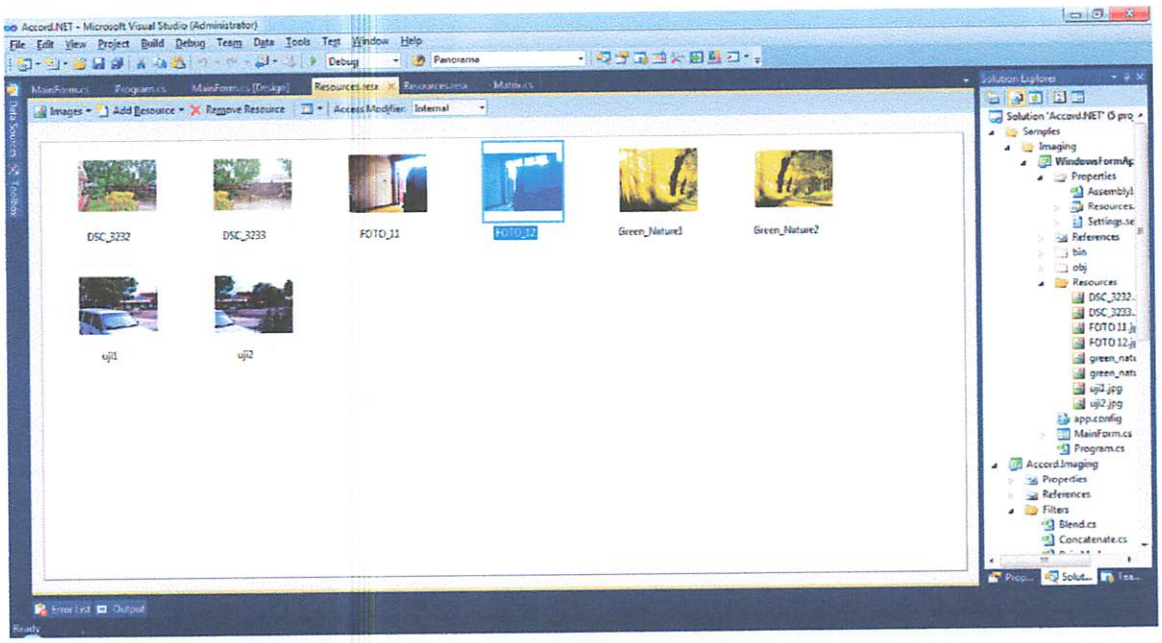


Gambar 3.18 *Window Conversion Progress*

7. Setelah *Conversion Progress* selesai, maka akan didapatkan hasil ekstraksi video berupa data foto (*.Jpg).

Gambar 3.19 Hasil Ekstraksi Video (*.Jpg)

Dalam prosesnya, untuk menginputkan data ke dalam algoritma menggunakan perintah *add resource* yang terdapat pada *solution explorer*. Pada penelitian ini, data **.JPG* yang digunakan adalah FOTO_11 dan FOTO_22.



Gambar 3.20 *resource* data **.JPG*

Adapun foto – foto yang digunakan mempunyai pertampalan dan paling tidak ada objek yang dapat di identifikasi. Pertampalannya sendiri dapat berupa kanan – kiri ataupun atas – bawah, namun tidak dapat menggunakan data yang mempunyai jarak atau skala yang terlalu jauh antara dua foto yang digunakan untuk melakukan proses panorama algoritma yang dibuat. Sehingga ada beberapa tahapan dalam proses pengambilan foto, antara lain:

1. Kamera IP diletakkan pada tripod atau tempat penyangga kamera yang lainnya.
2. Lakukan pemotretan yang pertama untuk foto sebelah kiri setelah itu gerakkan kamera sekitar 30 derajat ke kanan dan lakukan pemotretan yang kedua untuk foto yang kanan.
3. Hal yang sama berlaku untuk foto atas – bawah, hanya pergeseran kamera bukan ke kanan tetapi ke atas.
4. Yang patut dicatat adalah posisi kamera tetap atau dapat bergeser dari posisi awal dalam pengambilan foto pertama ke foto kedua, namun tidak boleh terjadi pergeseran terlalu jauh yang dapat menyebabkan jauhnya jarak antara foto kiri dan foto kanan karena bila kedua foto yang digunakan mempunyai jarak atau skala yang terlalu jauh, algoritma pada penelitian ini tidak akan bisa melakukan proses pembuatan sebuah panorama.

Setelah memasukkan foto yang akan digunakan untuk menyelesaikan penelitian ini, maka dalam algoritma yang dibuat harus dipanggil foto – foto tersebut agar muncul dalam *picture box* di desain *interface* yang telah dibuat. adapun algoritma yang dibuat seperti berikut.

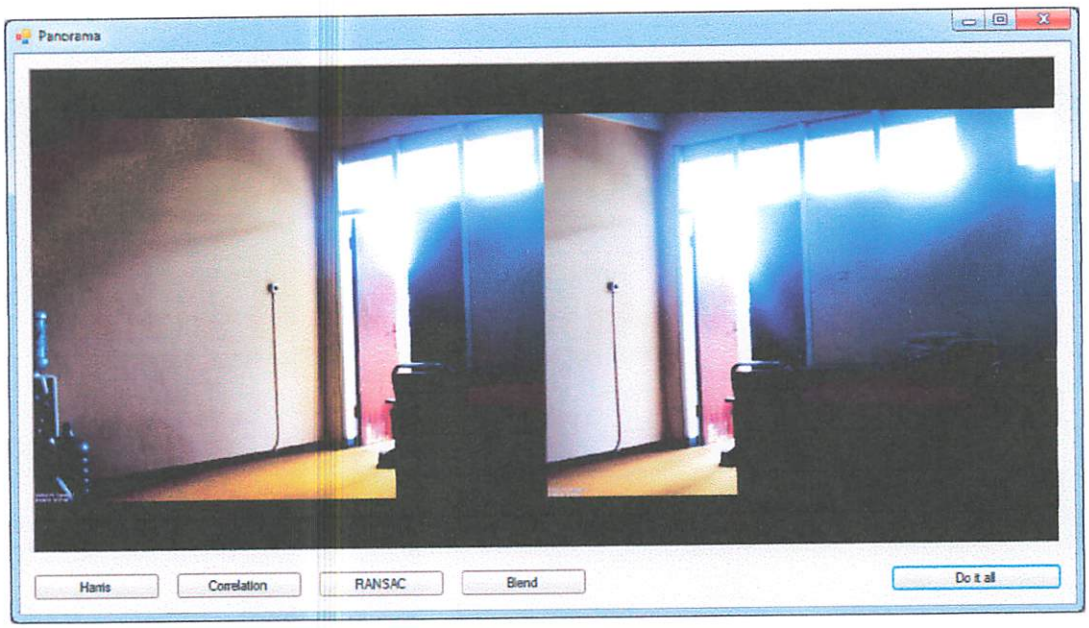
```
public partial class MainForm : Form
    {
        private Bitmap img1 = Panorama.Properties.Resources.FOTO_11;
        private Bitmap img2 = Panorama.Properties.Resources.FOTO_12;

        private IntPoint[] harrisPoints1;
        private IntPoint[] harrisPoints2;

        private IntPoint[] correlationPoints1;
        private IntPoint[] correlationPoints2;

        private MatrixH homography;
```

Sehingga apabila algoritma tersebut dijalankan, maka di *picture box* pada desain *interface* yang dibuat akan muncul foto – foto yang telah dipanggil sebagai data *resource*.



Gambar 3.21 tampilan foto pada *interface*

3.3.2 *Harris Corner Detection*

Untuk membuat algoritma *interest point detector*, proses ini dapat mengacu pada *Accord.Imaging.HarrisCornerDetector class*. Untuk membuat proses lebih cepat, dapat

digunakan ambang penekanan (*thresholding)* yang lebih tinggi untuk menekan beberapa poin. Pada proses pembuatan algoritma ini, akan digunakan nilai 1000. Pada tombol *button1*, ganti nama textnya menjadi Harris agar memudahkan dalam melakukan proses foto panoramik ini (gambar 3.1). Klik 2x pada *button* Harris tersebut Sehingga otomatis akan muncul *MainForm* sebagai tempat untuk memasukkan algoritma dari proses Harris. Algoritma yang terbentuk pada *main form* dapat di lihat sebagai berikut:

```
private void button1_Click(object sender, EventArgs e)
        {
            // Step 1: Detect feature points using Harris Corners Detector
            HarrisCornersDetector harris = new HarrisCornersDetector(0.04f, 1000f);
            harrisPoints1 = harris.ProcessImage(img1).ToArray();
            harrisPoints2 = harris.ProcessImage(img2).ToArray();

            // Show the marked points in the original images
            Bitmap img1mark = new PointsMarker(harrisPoints1).Apply(img1);
            Bitmap img2mark = new PointsMarker(harrisPoints2).Apply(img2);

            // Concatenate the two images together in a single image (just to show
on screen)
            Concatenate concatenate = new Concatenate(img1mark);
            pictureBox.Image = concatenate.Apply(img2mark);
        }
```

Dengan menggunakan nilai 1000 sebagai batasan antara objek utama dengan latar belakang atau bisa disebut sebagai *thresholding* (untuk penjelasan *thresholding* dapat dilihat pada bab 2 halaman 11) maka proses penentuan *interest point detector* akan bejalan lebih cepat.

Gambar 3.22 hasil proses *Harris Corner Detection*

Pada gambar diatas, dapat dilihat titik – titik yang terdapat dalam foto kiri dan kanan hasil proses *Harris Corner Detection* (sebagian terdapat dalam persegi yang telah ditandai). Adapun proses algoritma yang digunakan untuk mendapatkan hasil tersebut mengadopsi dari *Accord.Imaging.HarrisCornerDetector class*, sehingga dapat dibahasakan sebagai berikut, namun bahasa yang ditampilkan pada pembahasan ini hanya awalan, untuk lengkapnya akan dicantumkan pada lampiran.

```
namespace Accord.Imaging
{
    using System.Collections.Generic;
    using System.Drawing;
    using System.Drawing.Imaging;
    using AForge.Imaging;
    using AForge.Imaging.Filters;
    using AForge;

    public class HarrisCornersDetector : ICornersDetector
```

```
{
    private float k = 0.04f;
    private float threshold = 1000f;
    private double sigma = 1.4;
    private int r = 3;


    /// <summary>
    ///   Harris parameter k. Default value is 0.04.
    /// </summary>
    public float K
    {
        get { return k; }
        set { k = value; }
    }

    /// <summary>
    ///   Harris threshold. Default value is 1000.
    /// </summary>
    public float Threshold
    {
        get { return threshold; }
        set { threshold = value; }
    }

    /// <summary>
    ///   Gaussian smoothing sigma. Default value is 1.4.
    /// </summary>
    public double Sigma
    {
        get { return sigma; }
        set { sigma = value; }
    }

    /// <summary>
    ///   Non-maximum suppression window radius. Default value is 3.
    /// </summary>
    public int Suppression
    {
        get { return r; }
        set { r = value; }
    }
```

3.3.3 *Correlation*

Pada pembuatan algoritma korelasi, dapat dicapai dengan mengacu pada *Accord.Imaging.CorrelationMatching class*. Sama seperti proses Harris, ganti *text button2*

menjadi *correlation* (gambar 3.1) dan klik 2x pada *button* tersebut maka otomatis akan masuk ke *MainForm*. tuliskan seperti berikut ini:

```
private void btnCorrelation_Click(object sender, EventArgs e)
        {
            // Step 2: Match feature points using a correlation measure
            CorrelationMatching matcher = new CorrelationMatching(9);
            IntPoint[][] matches = matcher.Match(img1, img2, harrisPoints1,
harrisPoints2);

            // Get the two sets of points
            correlationPoints1 = matches[0];
            correlationPoints2 = matches[1];

            // Concatenate the two images in a single image (just to show on screen)
            Concatenate concat = new Concatenate(img1);
            Bitmap img3 = concat.Apply(img2);

            // Show the marked correlations in the concatenated image
            PairsMarker pairs = new PairsMarker(
                correlationPoints1, // Add image1's width to the X points to show
the markings correctly
                correlationPoints2.Apply(p => new IntPoint(p.X + img1.Width, p.Y)));

            pictureBox.Image = pairs.Apply(img3);
        }
```

Pada proses *correlation*, untuk mencari titik yang sama antara foto kiri dan kanan digunakan ukuran korelasi yaitu dengan menentukan 9 piksel di sekitar titik. Jadi hasil dari *interest point detector* berupa titik – titik di foto kiri dan kanan akan disinkronkan dengan proses korelasi ini.

Gambar 3.23 hasil proses korelasi

Adapun hasil dari algoritma yang digunakan pada proses korelasi sebagai berikut:

```
namespace Accord.Imaging
{
    using System;
    using System.Collections.Generic;
    using System.Drawing;
    using System.Drawing.Imaging;
    using Accord.Math;
    using AForge;
    using AForge.Imaging.Filters;

    public class CorrelationMatching
    {

        private int windowSize;
        private double dmax;

        /// <summary>
        ///   Gets or sets the maximum distance to consider
        ///   points as correlated.
        /// </summary>
        public double DistanceMax
        {
            get { return dmax; }
```

```
        set { dmax = value; }
    }

    /// <summary>
    ///   Gets or sets the size of the correlation window.
    /// </summary>
    public int WindowSize
    {
        get { return windowSize; }
        set { windowSize = value; }
    }



    /// <summary>
    ///   Constructs a new Correlation Matching algorithm.
    /// </summary>
    public CorrelationMatching(int windowSize)
        : this(windowSize, 0)
    {
    }

    /// <summary>
    ///   Constructs a new Correlation Matching algorithm.
    /// </summary>
    public CorrelationMatching(int windowSize, double maxDistance)
    {
        if (windowSize % 2 == 0)
            throw new ArgumentException("Window size should be odd",
"windowSize");

        this.windowSize = windowSize;
        this.dmax = maxDistance;
    }
```

3.3.4 RANSAC

Pembuatan algoritma proses RANSAC *(Random Sample Consensus)* akan mengacu pada algoritma *Accord.Imaging.RansacHomographyEstimator class* untuk memperkirakan matriks *homography* menggunakan RANSAC. *RansacHomographyEstimator* adalah kelas untuk memberikan kenyamanan perhitungan semua logika yang digunakan dalam sebuah objek *Accord.MachineLearning*. RANSAC dengan parameter yang dipasang matriks

homography. Untuk *button* sama seperti proses pada Harris dan *Correlation* (gambar 3.1). Adapun bahasa yang dapat dibuat untuk proses ini sebagai berikut:

```
private void btnRansac_Click(object sender, EventArgs e)
        {
            // Step 3: Create the homography matrix using a robust estimator
            RansacHomographyEstimator ransac = new RansacHomographyEstimator(0.001,
0.99);
            homography = ransac.Estimate(correlationPoints1, correlationPoints2);

            // Plot RANSAC results against correlation results
            IntPoint[] inliers1 = correlationPoints1.Submatrix(ransac.Inliers);
            IntPoint[] inliers2 = correlationPoints2.Submatrix(ransac.Inliers);

            // Concatenate the two images in a single image (just to show on screen)
            Concatenate concat = new Concatenate(img1);
            Bitmap img3 = concat.Apply(img2);

            // Show the marked correlations in the concatenated image
            PairsMarker pairs = new PairsMarker(
                inliers1, // Add image1's width to the X points to show the markings
correctly
                inliers2.Apply(p => new IntPoint(p.X + img1.Width, p.Y)));

            pictureBox.Image = pairs.Apply(img3);
        }
```

Proses korelasi hanya mencocokkan titik – titik yang ada pada foto kiri dan foto kanan tanpa menghiraukan kebenaran dari hasil korelasi yang sesuai pada kedua foto tersebut. Idealnya korelasi yang baik yaitu kecocokkan kedua foto tersebut pada bidang planar dan objek yang sama. Untuk meminimalisir atau menghilangkan kesalahan pada proses korelasi tersebut maka dibutuhkan proses RANSAC. Pada proses ini, dilakukan perhitungan homography sebagai elemen untuk meminimalisir kesalahan pada proses korelasi. RANSAC sendiri merupakan metode untuk mengestimasi parameter agar sesuai dengan model matematika dari point hasil korelasi yang mungkin tidak benar.

Gambar 3.24 hasil proses RANSAC

Dari gambar tersebut dapat dilihat bahwa dengan proses RANSAC, banyak garis – garis yang telah dihapus karena ketidaksesuaian korelasi pada proses korelasi itu sendiri. Algoritma yang dibuat dapat dilihat sebagai berikut:

```
namespace Accord.Imaging
{
    using System;
    using System.Drawing;
    using Accord.MachineLearning;
    using Accord.Math;
    using AForge;

public class RansacHomographyEstimator
    {
        private RANSAC<MatrixH> ransac;
        private int[] inliers;

        private PointF[] pointSet1;
        private PointF[] pointSet2;


        /// <summary>
```

```
        ///   Gets the RANSAC estimator used.
        /// </summary>
        public RANSAC<MatrixH> Ransac
        {
            get { return this.ransac; }
        }

        /// <summary>
        ///   Gets the final set of inliers detected by RANSAC.
        /// </summary>
        public int[] Inliers
        {
            get { return inliers; }
        }


        /// <summary>
        ///   Creates a new RANSAC homography estimator.
        /// </summary>
        /// <param name="threshold">Inlier threshold.</param>
        /// <param name="probability">Inlier probability.</param>
        public RansacHomographyEstimator(double threshold, double probability)
        {
            // Create a new RANSAC with the selected threshold
            ransac = new RANSAC<MatrixH>(4, threshold, probability);

            // Set RANSAC functions
            ransac.Fitting = homography;
            ransac.Degenerate = degenerate;
            ransac.Distances = distance;
        }
```

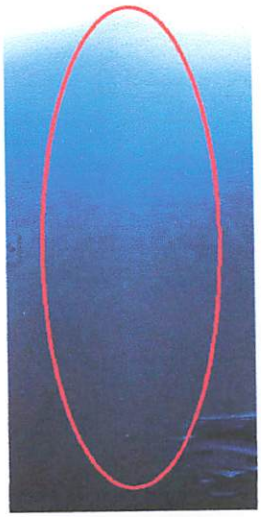
3.3.5 *Blending*

Langkah terakhir adalah untuk membuat campuran dari dua gambar. Untuk proses ini mengacu pada filter *Accord.Imaging.Blend*, yang diimplementasikan sebagai filter gambar *AForge.NET*. Proses *button* seperti langkah sebelumnya (gambar 3.1). Pembuatan algoritma dapat diuraikan sebagai berikut:

```
private void btnBlend_Click(object sender, EventArgs e)
        {
            // Step 4: Project and blend the second image using the homography
            Blend blend = new Blend(homography, img1);
            pictureBox.Image = blend.Apply(img2);
        }
```

Peleburan warna antara 2 (dua) foto menjadi sebuah foto yang saling bertampalan merupakan tujuan dari proses *image blending*. Seperti yang telah dijelakan pada bab 2 algoritma *gradient image blending* ini dipakai dengan maksud untuk meminimalisir atau bahkan menghilangkan masalah adanya perbedaan antara foto 1 dan foto 2 ketika sudah digabungkan (*visible seam*), daerah yang kabur *(blur)*, dan bayangan *(ghosting)*. Namun pada penelitian masih terdapat adanya perbedaan tersebut ditunjukkan oleh gambar berikut:

(a)



(b)

Gambar 3.25 (a) hasil *blend* (b) *seam* yang masih terlihat

Perbedaan antara foto yang digabungkan (*visible seam*) diakibatkan perbedaan intensitas cahaya, *blurring* yang terjadi karena kesalahan registrasi dan adanya bayangan (*ghosting*) karena pergerakan objek pada saat perekaman citra. Algoritma dari proses *blend* sendiri dapat dilihat sebagai berikut:

```
namespace Accord.Imaging.Filters
{
    using System.Collections.Generic;
    using System.Drawing;
    using System.Drawing.Imaging;
    using AForge.Imaging;
    using System;
    using Matrix = Accord.Math.Matrix;

public class Blend : AForge.Imaging.Filters.BaseTransformationFilter
    {
        private MatrixH homography;
        private Bitmap overlayImage;
        private Point offset;
        private Point center;
```

```
        private Size imageSize;
        private Color fillColor = Color.FromArgb(0, Color.Black);
        private Dictionary<PixelFormat, PixelFormat> formatTranslations = new
Dictionary<PixelFormat, PixelFormat>();



        /// <summary>
        /// Format translations dictionary.
        /// </summary>
        public override Dictionary<PixelFormat, PixelFormat> FormatTranslations
        {
            get { return formatTranslations; }
        }

        /// <summary>
        ///   Gets or sets the Homography matrix used to map a image passed to
        ///   the filter to the overlay image specified at filter creation.
        /// </summary>
        public MatrixH Homography
        {
            get { return homography; }
            set { homography = value; }
        }

        /// <summary>
        ///   Gets or sets the filling color used to fill blank spaces.
        /// </summary>
        /// <remarks>
        ///   The filling color will only be visible after the image is converted
        ///   to 24bpp. The alpha channel will be used internally by the filter.
        /// </remarks>
        public Color FillColor
        {
            get { return fillColor; }
            set { fillColor = value; }
        }

        /// <summary>
        ///   Constructs a new Blend filter.
        /// </summary>
        /// <param name="homography">The homography matrix mapping a second image to
the overlay image.</param>
        /// <param name="overlayImage">The overlay image (also called the
anchor).</param>
        public Blend(double[,] homography, Bitmap overlayImage)
            : this(new MatrixH(homography), overlayImage)
        {
        }

        /// <summary>
        ///   Constructs a new Blend filter.
        /// </summary>
```

```
        /// <param name="homography">The homography matrix mapping a second image to
the overlay image.</param>
        /// <param name="overlayImage">The overlay image (also called the
anchor).</param>
        public Blend(MatrixH homography, Bitmap overlayImage)
        {
            this.homography = homography;
            this.overlayImage = overlayImage;
            formatTranslations[PixelFormat.Format8bppIndexed] =
PixelFormat.Format32bppArgb;
            formatTranslations[PixelFormat.Format24bppRgb] =
PixelFormat.Format32bppArgb;
            formatTranslations[PixelFormat.Format32bppArgb] =
PixelFormat.Format32bppArgb;
        }
```

Semua tahapan penelitian yang telah dijelaskan tersebut merupakan isi dari tiap *Button* yang terdapat pada desain *interface* pada gambar 3.1. Untuk membuat citra foto yang ada menjadi sebuah panoramik secara otomatis, maka diperlukan sebuah *Button* yang berfungsi agar proses yang diinginkan berjalan secara otomatis. Sehingga dapat dibahasakan sebagai berikut:

```
private void btnDoItAll_Click(object sender, EventArgs e)
        {
            // Do it all
            btnHarris_Click(sender, e);
            btnCorrelation_Click(sender, e);
            btnRansac_Click(sender, e);
            btnBlend_Click(sender, e);
        }
```

3.3.6 Citra Mosaik Foto

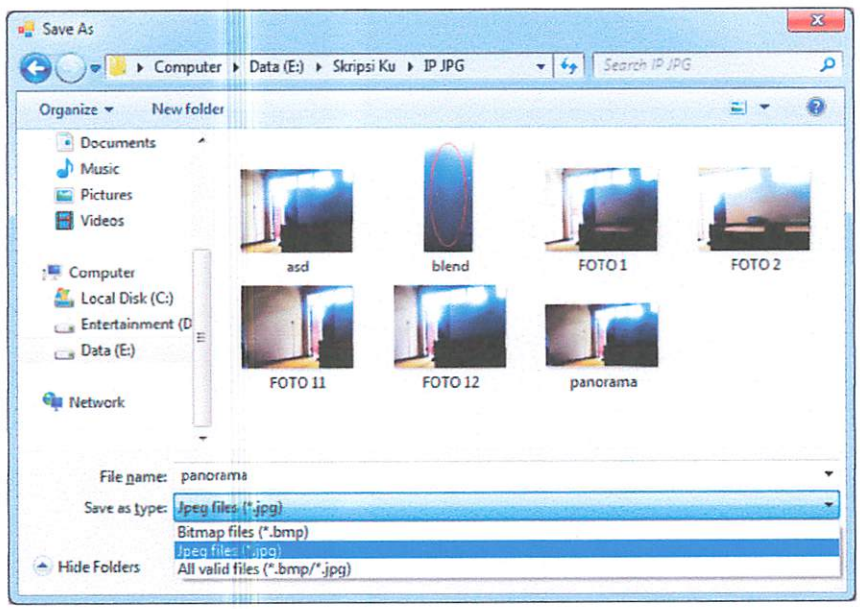
Setelah panorama yang diinginkan sudah tercipta dengan proses yang telah dijelaskan, maka dapat disimpan hasil citra mosaik tersebut ke dalam sebuah folder yang disiapkan. Untuk itu dibutuhkan suatu tombol *button save* untuk proses penyimpanan ini (gambar 3.1). Algoritma untuk proses *save* dapat dilihat sebagai berikut:

```
private void button6_Click(object sender, EventArgs e)
        {
            SaveFileDialog saveFileDialog = new SaveFileDialog();

            saveFileDialog.InitialDirectory = "c:\\";
            saveFileDialog.Filter = "Bitmap files (*.bmp)|*.bmp|Jpeg files
(*.jpg)|*.jpg|All valid files (*.bmp/*.jpg)|*.bmp/*.jpg";
            saveFileDialog.FilterIndex = 1;
            saveFileDialog.RestoreDirectory = true;

            if (DialogResult.OK == saveFileDialog.ShowDialog())
            {
                pictureBox.Image.Save(saveFileDialog.FileName);
            }
```

Pada proses ini hanya untuk menampilkan hasil algoritma yang dibuat agar dapat disimpan dalam berbagai format. Pada penelitian kali ini format yang dapat digunakan untuk penyimpanan adalah *.bmp dan *.jpg. Prosesnya sendiri menggunakan algoritma *save dialog file*.



(a)

(b)

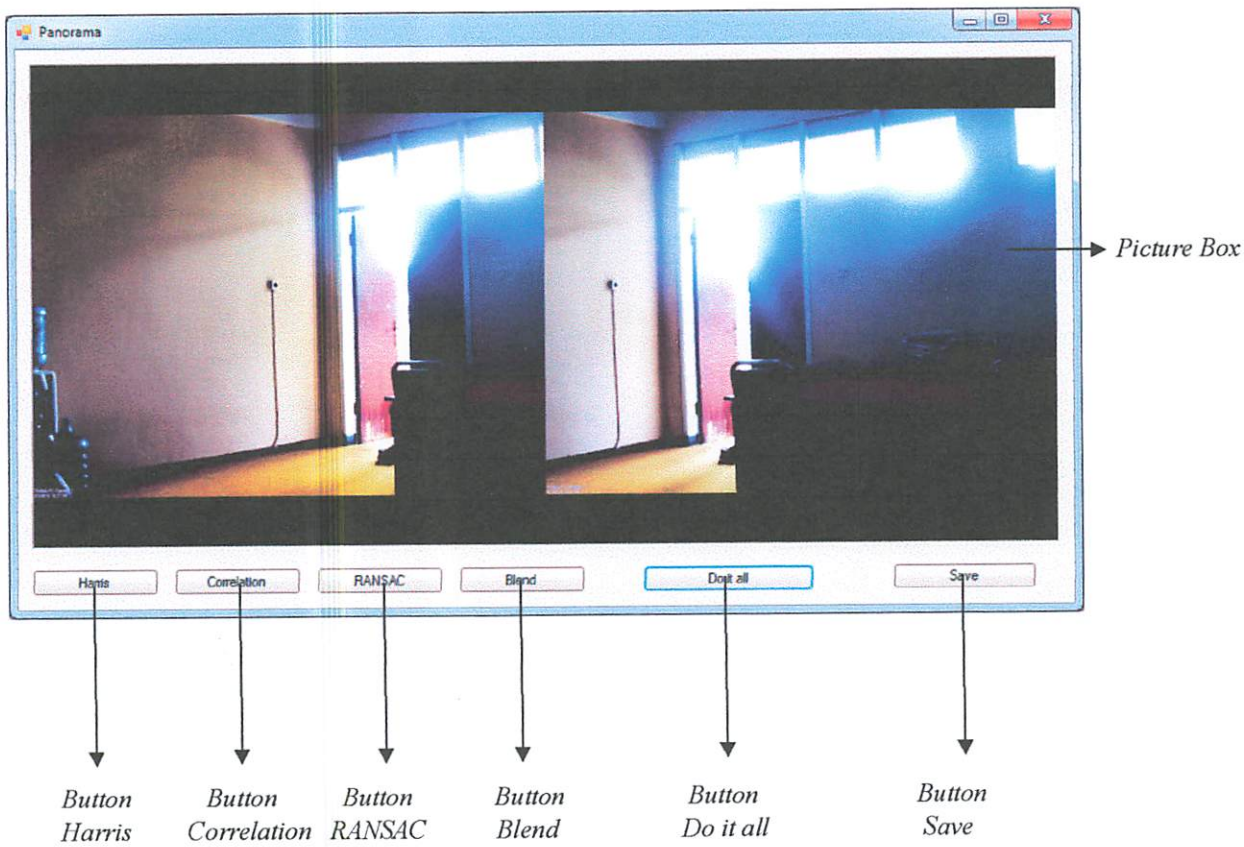
Gambar 3.26 (a) dialog penyimpanan (b) hasil citra panoramik

BAB IV

HASIL PENELITIAN DAN PEMBAHASAN

Proses analisa atau pembahasan dari sebuah hasil yang telah dicapai selama proses pelaksanaan penelitian akan mengetahui keberhasilan dari penelitian tersebut. Pada bab ini akan disampaikan mengenai hasil dan pembahasan dari pembuatan algoritma pembuatan foto panoramik secara otomatis.

4.1 Hasil Penelitian



Gambar 4.1 Tampilan program pembuatan foto panoramik

Hasil pembuatan algoritma yang telah dilakukan pada bab 3 akan menampilkan sebuah *interface* untuk proses pembuatan sebuah foto panoramik dari 2 (dua) buah foto yang digunakan. Ada beberapa tombol yang dipakai untuk melakukan penelitian ini antara lain sebagai berikut :

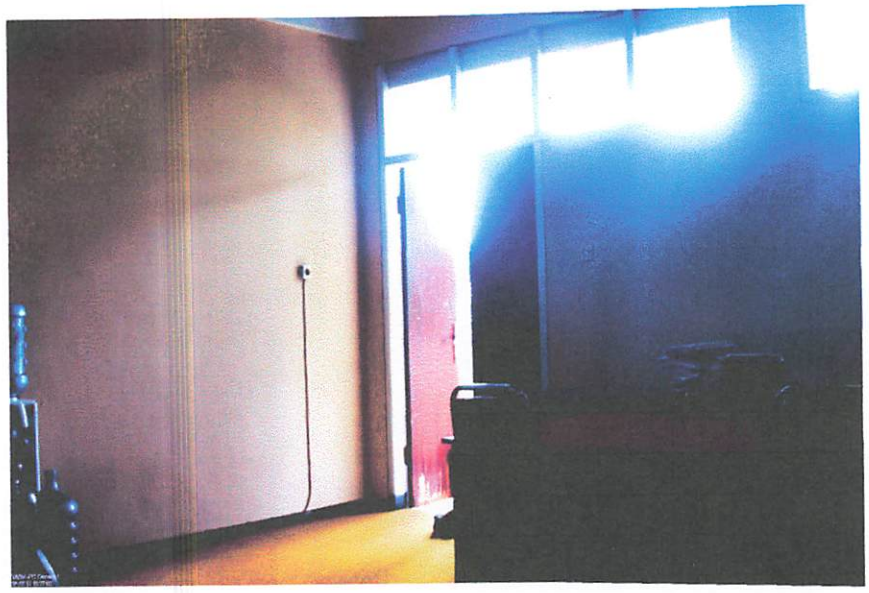
1. ***Button Harris*** : tombol ini digunakan untuk melakukan pendeteksian titik – titik pada foto kiri dan kanan sesuai dengan *thresholding* yang dipakai untuk melakukan proses pembuatan foto panoramik. Nilai *thresholding* yang dipakai untuk melakukan proses *Harris Corner Detection* ini adalah 1000.

2. ***Button Correlation*** : tombol ini berfungsi agar terjadi korelasi antara foto kiri serta foto kanan yang sudah terdeteksi pada saat proses *Harris*. Pada proses ini, korelasi terjadi dengan mencari *gray value* pada piksel yang terdapat pada foto kiri, kemudian disesuaikan dengan *gray value* pada foto

kanan. Proses kerja korelasi sendiri telah dijelaskan pada bab 2 sehingga untuk mengetahui prosesnya dapat dilihat pada halaman 15.

3. ***Button RANSAC*** : tombol yang berfungsi untuk mengeliminasi korelasi yang salah antara foto kiri dan kanan dengan menggunakan metode homography agar kedua foto dapat *blending* dengan baik. Proses ini dapat dilakukan secara berulang – ulang untuk mendapatkan korelasi yang paling benar.

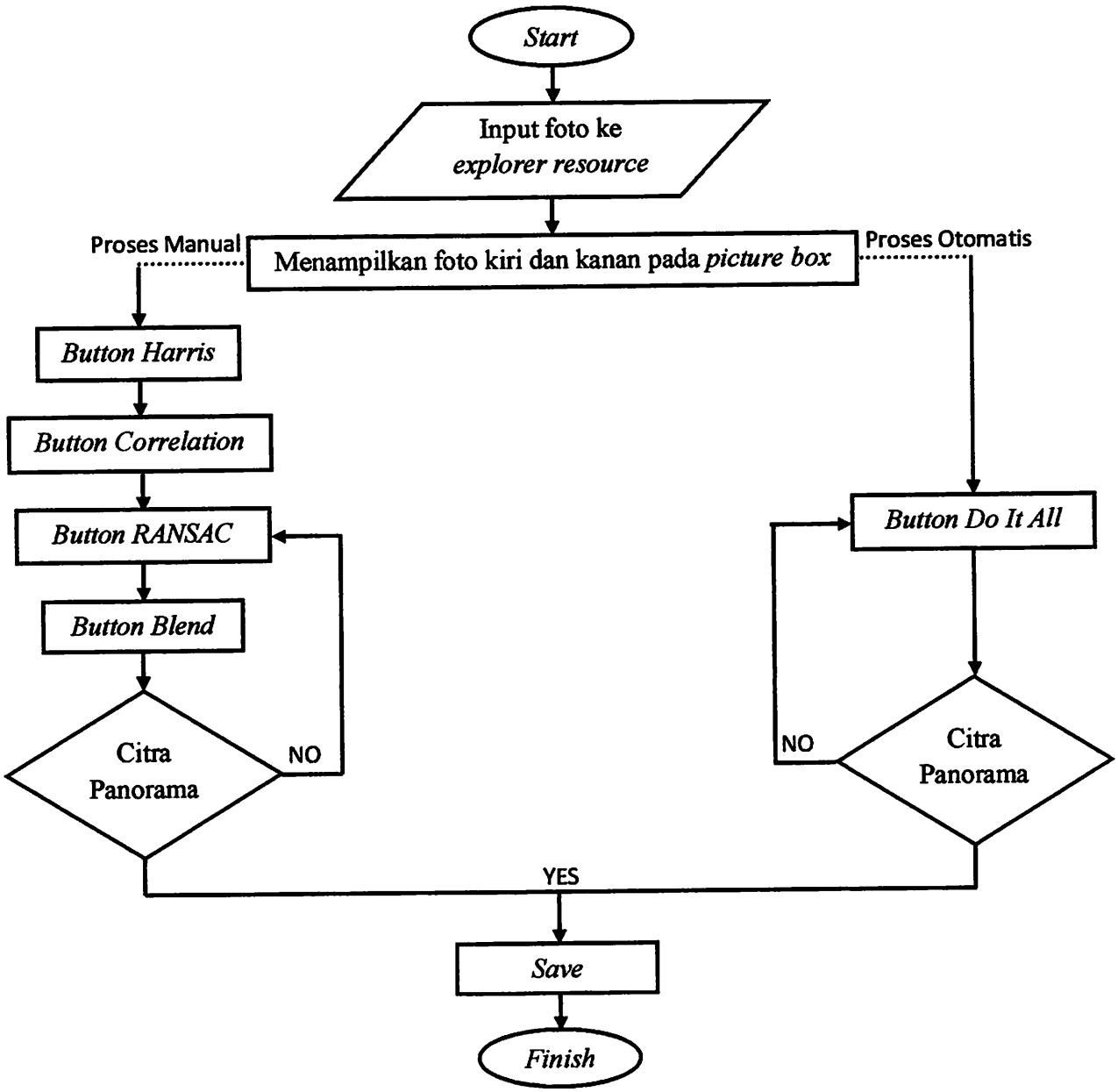
4. ***Button Blend*** : tombol yang berfungsi untuk menyatukan kedua foto antara foto kiri dan foto kanan menjadi sebuah (1) foto panoramik.

5. 

6. ***Button Do It All*** : tombol yang berfungsi untuk melakukan semua proses pembuatan foto panoramik secara otomatis.

7. ***Button Save*** : tombol untuk melakukan penyimpanan terhadap hasil panoramik yang telah melalui proses – proses diatas.

Dari hasil tersebut dapat dijelaskan secara bagan diagram sebagai berikut :



4.2 Pembahasan

Pembahasan yang bisa didapat dari pembuatan foto panoramik otomatis ini adalah sebagai berikut :

1. Algoritma yang di buat dapat membuat panoramik secara otomatis dengan hasil sebuah foto hasil pertampalan 2 buah foto.

2. Foto yang dihasilkan bagus atau tidaknya tergantung data foto yang digunakan, semakin sedikit bidang obyek yang nampak di foto maka semakin bagus pula hasil foto panoramik. Hal ini dikarenakan pada homography yang digunakan mendeteksi bidang planar satu persatu.

3. Pada saat proses RANSAC, dapat dipilih hasil RANSAC yang terbaik dengan melakukan proses tersebut secara berkali-kali.

4. Jika pada korelasi, jendela yang digunakan kurang dari 9 maka objek yang terkorelasi akan semakin banyak dan akan mengakibatkan kesalahan semakin besar pula.

5. Masih belum adanya *button open* untuk melakukan proses penginputan data pada algoritma. Sehingga untuk melakukan proses pemasukkan data pada *picture box* harus dengan mengganti perintah pada *main form* program dengan memanggil data yang ada pada *resource*.

6. Pada data foto yang digunakan mempunyai pertampalan yang baik antara foto kiri dan foto kanan baik pertampalan antara dua buah foto panorama yang

terbentuk dari kiri dan kanan atau atas dan bawah. Namun tidak dapat melakukan proses panorama yang mempunyai pertampalan foto berbeda jaraknya atau skala antara kiri dan kanan.

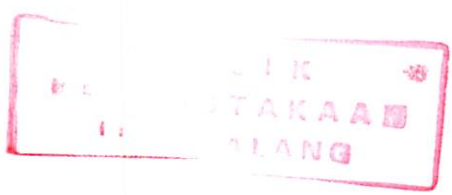
BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Kesimpulan yang dapat diambil dari penelitian yang dibuat adalah :

1. Membuat citra panoramik dapat dilakukan secara otomatis dengan membuat sebuah algoritma yang berdasarkan teori Harris dilanjutkan korelasi lalu RANSAC dan diakhiri dengan *blending.*
2. Pada pembuatan algoritma penelitian ini mengacu pada algoritma yang telah dibuat pada *Library Accord.net* karena pada *library* tersebut secara spesifik melakukan pengolahan citra panoramik dengan menggunakan metode yang sama pada penelitian ini.
3. Dalam membuat citra panoramik, perlu diperhatikan data-data yang digunakan agar hasil citra tersebut dapat digunakan untuk pandang yang lebih luas. Adapun maksud dari data-data yang digunakan yaitu foto yang digunakan sebagai data *resource* harus memiliki skala atau jarak dari pandang kamera yang tidak terlalu jauh. Algoritma ini dapat melakukan proses panoramik yang mempunyai pertampalan kiri dan kanan atau atas dan bawah yang mempunyai jarak tidak berbeda jauh antara dua foto yang dipakai.



5.2 Saran

Dalam proses penelitian ini, masih banyak menurut penulis kekurangan yang ada sehingga dapat disarankan beberapa hal sebagai berikut :

1. Dalam membuat algoritma harus disesuaikan fitur-fitur yang mendukung sesuai dengan versi yang terdapat dalam *Accord.Net* dan *AForge.Net* yang digunakan sebagai bahan acuan dalam membuat algoritma.
2. Harus memahami lebih dalam tentang membuat bahasa dalam C# Visual Studio 2010 agar memudahkan dalam mengaplikasikan algoritma yang akan dibuat.
3. Algoritma yang dibuat masih terbatas dalam 2 foto sehingga jika ada mahasiswa yang akan melanjutkan dapat mengembangkan algoritma dengan membuat lebih dari 2 foto untuk dijadikan sebuah foto panoramik.
4. Jika ingin melanjutkan penelitian ini, maka disarankan untuk lebih memvariasikan data yang dapat di olah dengan mengembangkan algoritma ini. Sehingga data yang digunakan tidak hanya terbatas pada kesamaan dua foto yang memiliki jarak tidak terlalu jauh. Namun bisa menghasilkan panorama dari dua buah foto yang memiliki skala berbeda.

Daftar Pustaka

Atkinson, K.B, 2001, ***Close Range Photogrametry and Machine Vision***, Whittles Publishing. Scotland, UK.

Bolles, R.C., Quam, L.H., Fischler. M.A., and Wolf, H.C. The SRI road expert: Image to database correspondence. In Proc. Image Understanding Workshop, Pittsburgh, Pennsylvania, Nov., 1981

Campbell, N.A, and Wu, X., 2008. "Cross Correlation For Sub-Pixel Matching". The International Archives of The Photogrammetry, Remote Sensing and Spasial Information Sciences, Vol. XXXVI. Part B7, Beijing 2008

Gonzalez, R. C. and Woods, R. E. 2008. ***Digital Image Processing***, *3rd ed.*, Prentice Hall, Upper Saddle River, NJ.

Huang, F. Klette, R. Scheibe, K. 2008. ***Panoramic Imaging Sensor-Line Cameras and Laser Range-Finder***. A John Wiley and Son Ltd. Inggris.

Mitchell, H,L., and Pilgrim, L.J., 1987. "Selection of an Image Matching Algorithm". Department Of Civil Engineering and Surveying. University of Newcastel.

Moravec, H.P.: Towards Automatic Visual Obstacle Avoidance. *Proc. 5thInternational Joint Conference on Artificial Intelligence*, pp. 584, 1977.

Martin, A.F., Robert, C.B.: Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartograph, Pittsburgh, Pennsylvania, Nov., 1978

Nixon, M.S., Aguado, A.S.: Feature Extraction and Image Processing. Non Maximum Suppression page 113-115, Oxford, Britain. 2002

Novianti. M, 2010, ***Studi Penentuan Titik Konjugasi Pada Foto Yang Bertampalan (Image Matching) Menggunakan Metode Area-Based Matching Pada Foto Stereo***, Jurusan Teknik Geodesi, Institut Teknologi Nasional, Malang.

Harris, C., Stephens, M.: A combined corner and edge detector. In: Proceedings, 4th Alvey Vision Conference. (1988) 147-151 Manchester.

Pollefeys, M.: Visual 3D Modeling from Images. 2002.

Porter.T and Duff. T, 1984, ***Compositing Digital Images***. Proceeding of ACM SIGGRAPH 84, ACM Computer Graphic., pp.253-259.

Rahman. Arif., 2009. ***Sistem Temu Balik Citra Menggunakan Jarak Histogram dalam Model Warna YIQ.*** Seminar Nasional Aplikasi Teknologi Informasi. Yogyakarta.

Rankov. V et all., 2005, ***An Algorithm for Image Stitching and Blending***. Proceeding of SPIE volume 570.

Schenk, T., 1999. ***Digital Photogrammetry,*** TerraScience, Ohio, USA.

Souza. Cesar., 2009-2010., http://www.codeproject.com/KB/recipes/automatic_panoramas.aspx (accord.net, *online access* 7 Juli 2011).

Szeliski. R., 2006. ***Image Alignment and Stitching : A Tutorial.*** Microsoft Research, USA.

Wikipedia Contributors. "RANSAC" *Wikipedia, The Free Encyclopedia.* Diakses pada tanggal 26 Oktober 2011.

Wikipedia Contributors, "Corner Detection" *Wikipedia, The Free Encyclopedia.* Diakses pada tanggal 27 Oktober 2011.

Wikipedia Contributors. "Image Gradient" *Wikipedia, The Free Encyclopedia.* Diakses pada tanggal 3 November 2011.

Wikipedia, 2011., ***"Normalization (Image Processing)"*** *Wikipedia, The Free Encyclopedia.* Diakses tanggal 13 Oktober 2011.

Wolf , P.R., and Dewitt, B. A., 2000. ***Element of Photogrammetry with Application in GIS 3rd***, McGraw-Hill Higher Education.pp 334-341.New York.

LAMPIRAN
ALGORITMA PROSES *HARRIS CORNER DETECTOR*

```
/ Accord Imaging Library
/ Accord.NET framework
/ http://www.crsouza.com
/
/ Copyright © César Souza, 2009-2010
/ cesarsouza at gmail.com
/

amespace Accord.Imaging

   using System.Collections.Generic;
   using System.Drawing;
   using System.Drawing.Imaging;
   using AForge.Imaging;
   using AForge.Imaging.Filters;
   using AForge;

   /// <summary>
   ///   Harris Corners Detector.
   /// </summary>
   /// <remarks>
   /// <para>This class implements the Harris corners detector.</para>
   /// <para>Sample usage:</para>
   /// <code>
   /// // create corners detector's instance
   /// HarrisCornersDetector hcd = new HarrisCornersDetector( );
   /// // process image searching for corners
   /// Point[] corners = hcd.ProcessImage( image );
   /// // process points
   /// foreach ( Point corner in corners )
   /// {
   ///     // ...
   /// }
   /// </code>
   ///
   /// <para>
   ///   References:
   ///   <list type="bullet">
   ///     <item><description>
   ///       P. D. Kovesi. MATLAB and Octave Functions for Computer Vision and Image
Processing.
   ///       School of Computer Science and Software Engineering, The University of
Western Australia.
   ///       Available in:
http://www.csse.uwa.edu.au/~pk/Research/MatlabFns/Spatial/harris.m</description></item>
   ///     <item><description>
   ///       C.G. Harris and M.J. Stephens. "A combined corner and edge detector",
   ///       Proceedings Fourth Alvey Vision Conference, Manchester.
   ///       pp 147-151, 1988.</description></item>
   ///     <item><description>
   ///       Alison Noble, "Descriptions of Image Surfaces", PhD thesis, Department
   ///       of Engineering Science, Oxford University 1989, p45.</description></item>
   ///   </list>
   /// </para>
```

```
/// </remarks>
///
/// <seealso cref="MoravecCornersDetector"/>
/// <seealso cref="SusanCornersDetector"/>
///
public class HarrisCornersDetector : ICornersDetector
{

    private float k = 0.04f;
    private float threshold = 1000f;
    private double sigma = 1.4;
    private int r = 3;


    /// <summary>
    ///   Harris parameter k. Default value is 0.04.
    /// </summary>
    public float K
    {
        get { return k; }
        set { k = value; }
    }

    /// <summary>
    ///   Harris threshold. Default value is 1000.
    /// </summary>
    public float Threshold
    {
        get { return threshold; }
        set { threshold = value; }
    }

    /// <summary>
    ///   Gaussian smoothing sigma. Default value is 1.4.
    /// </summary>
    public double Sigma
    {
        get { return sigma; }
        set { sigma = value; }
    }

    /// <summary>
    ///   Non-maximum suppression window radius. Default value is 3.
    /// </summary>
    public int Suppression
    {
        get { return r; }
        set { r = value; }
    }



    /// <summary>
    ///   Initializes a new instance of the <see cref="HarrisCornersDetector"/>
class.
    /// </summary>
    public HarrisCornersDetector()
    {
```

```
        }

        /// <summary>
        ///   Initializes a new instance of the <see cref="HarrisCornersDetector"/>
lass.
        /// </summary>
        public HarrisCornersDetector(float k)
            : this()
        {
            this.k = k;
        }

        /// <summary>
        ///   Initializes a new instance of the <see cref="HarrisCornersDetector"/>
lass.
        /// </summary>
        public HarrisCornersDetector(float k, float threshold)
            : this()
        {
            this.k = k;
            this.threshold = threshold;
        }

        /// <summary>
        ///   Initializes a new instance of the <see cref="HarrisCornersDetector"/>
lass.
        /// </summary>
        public HarrisCornersDetector(float k, float threshold, double sigma)
            : this()
        {
            this.k = k;
            this.threshold = threshold;
            this.sigma = sigma;
        }



        /// <summary>
        /// Process image looking for corners.
        /// </summary>
        ///
        /// <param name="image">Source image data to process.</param>
        ///
        /// <returns>Returns list of found corners (X-Y coordinates).</returns>
        ///
        /// <exception cref="UnsupportedImageFormatException">
        ///   The source image has incorrect pixel format.
        /// </exception>
        ///
        public List<IntPoint> ProcessImage(UnmanagedImage image)
        {
            // check image format
            if (
                (image.PixelFormat != PixelFormat.Format8bppIndexed) &&
                (image.PixelFormat != PixelFormat.Format24bppRgb) &&
                (image.PixelFormat != PixelFormat.Format32bppRgb) &&
                (image.PixelFormat != PixelFormat.Format32bppArgb)
                )
```

```
        {
            throw new UnsupportedImageFormatException("Unsupported pixel format of
he source image.");
        }

        // make sure we have grayscale image
        UnmanagedImage grayImage = null;

        if (image.PixelFormat == PixelFormat.Format8bppIndexed)
        {
            grayImage = image;
        }
        else
        {
            // create temporary grayscale image
            grayImage = Grayscale.CommonAlgorithms.BT709.Apply(image);
        }


        // get source image size
        int width = grayImage.Width;
        int height = grayImage.Height;
        int stride = grayImage.Stride;
        int offset = stride - width;



        // 1. Calculate partial differences
        UnmanagedImage diffx = UnmanagedImage.Create(width, height,
ixelFormat.Format8bppIndexed);
        UnmanagedImage diffy = UnmanagedImage.Create(width, height,
ixelFormat.Format8bppIndexed);
        UnmanagedImage diffxy = UnmanagedImage.Create(width, height,
ixelFormat.Format8bppIndexed);

        unsafe
        {
            // Compute dx and dy
            byte* src = (byte*)grayImage.ImageData.ToPointer();
            byte* dx = (byte*)diffx.ImageData.ToPointer();
            byte* dy = (byte*)diffy.ImageData.ToPointer();
            byte* dxy = (byte*)diffxy.ImageData.ToPointer();

            // for each line
            for (int y = 0; y < height; y++)
            {
                // for each pixel
                for (int x = 0; x < width; x++, src++, dx++, dy++)
                {
                    // TODO: Place those verifications
                    // outside the innermost loop
                    if (x == 0 || x == width  - 1 ||
                        y == 0 || y == height - 1)
                    {
                        *dx = *dy = 0; continue;
                    }

                    int h = -(src[-stride - 1] + src[-1] + src[stride - 1]) +
```

```
                    (src[-stride + 1] + src[+1] + src[stride + 1]);
            *dx = (byte)(h > 255 ? 255 : h < 0 ? 0 : h);

            int v = -(src[-stride - 1] + src[-stride] + src[-stride + 1]) +
                     (src[+stride - 1] + src[+stride] + src[+stride + 1]);
            *dy = (byte)(v > 255 ? 255 : v < 0 ? 0 : v);
        }
        src += offset;
        dx += offset;
        dy += offset;
    }


    // Compute dxy
    dx = (byte*)diffx.ImageData.ToPointer();
    dxy = (byte*)diffxy.ImageData.ToPointer();

    // for each line
    for (int y = 0; y < height; y++)
    {
        // for each pixel
        for (int x = 0; x < width; x++, dx++, dxy++)
        {
            if (x == 0 || x == width  - 1 ||
                y == 0 || y == height - 1)
            {
                *dxy = 0; continue;
            }

            int v = -(dx[-stride - 1] + dx[-stride] + dx[-stride + 1]) +
                     (dx[+stride - 1] + dx[+stride] + dx[+stride + 1]);
            *dxy = (byte)(v > 255 ? 255 : v < 0 ? 0 : v);
        }
        dx += offset;
        dxy += offset;
    }
}


// 2. Smooth the diff images
if (sigma > 0.0)
{
    GaussianBlur blur = new GaussianBlur(sigma);
    blur.ApplyInPlace(diffx);
    blur.ApplyInPlace(diffy);
    blur.ApplyInPlace(diffxy);
}


// 3. Compute Harris Corner Response
float[,] H = new float[height, width];

unsafe
{
    byte* ptrA = (byte*)diffx.ImageData.ToPointer();
    byte* ptrB = (byte*)diffy.ImageData.ToPointer();
    byte* ptrC = (byte*)diffxy.ImageData.ToPointer();
    float M, A, B, C;
```

```
    for (int y = 0; y < height; y++)
    {
        for (int x = 0; x < width; x++)
        {
            A = *(ptrA++);
            B = *(ptrB++);
            C = *(ptrC++);

            // Harris corner measure
            M = (A * B - C * C) - (k * ((A + B) * (A + B)));

            if (M > threshold)
                H[y, x] = M;
            else H[y, x] = 0;
        }

        ptrA += offset;
        ptrB += offset;
        ptrC += offset;
    }
}


// Free resources
diffx.Dispose();
diffy.Dispose();
diffxy.Dispose();

if (image.PixelFormat != PixelFormat.Format8bppIndexed)
    grayImage.Dispose();


// 4. Suppress non-maximum points
List<IntPoint> cornersList = new List<IntPoint>();

// for each row
for (int y = r, maxY = height - r; y < maxY; y++)
{
    // for each pixel
    for (int x = r, maxX = width - r; x < maxX; x++)
    {
        float currentValue = H[y, x];

        // for each windows' row
        for (int i = -r; (currentValue != 0) && (i <= r); i++)
        {
            // for each windows' pixel
            for (int j = -r; j <= r; j++)
            {
                if (H[y + i, x + j] > currentValue)
                {
                    currentValue = 0;
                    break;
                }
            }
        }
```

```
                // check if this point is really interesting
                if (currentValue != 0)
                {
                    cornersList.Add(new IntPoint(x, y));
                }
            }
        }

        return cornersList;
    }

    /// <summary>
    /// Process image looking for corners.
    /// </summary>
    ///
    /// <param name="imageData">Source image data to process.</param>
    ///
    /// <returns>Returns list of found corners (X-Y coordinates).</returns>
    ///
    /// <exception cref="UnsupportedImageFormatException">
    ///   The source image has incorrect pixel format.
    /// </exception>
    ///
    public List<IntPoint> ProcessImage(BitmapData imageData)
    {
        return ProcessImage(new UnmanagedImage(imageData));
    }

    /// <summary>
    /// Process image looking for corners.
    /// </summary>
    ///
    /// <param name="image">Source image data to process.</param>
    ///
    /// <returns>Returns list of found corners (X-Y coordinates).</returns>
    ///
    /// <exception cref="UnsupportedImageFormatException">
    ///   The source image has incorrect pixel format.
    /// </exception>
    ///
    public List<IntPoint> ProcessImage(Bitmap image)
    {
        // check image format
        if (
            (image.PixelFormat != PixelFormat.Format8bppIndexed) &&
            (image.PixelFormat != PixelFormat.Format24bppRgb) &&
            (image.PixelFormat != PixelFormat.Format32bppRgb) &&
            (image.PixelFormat != PixelFormat.Format32bppArgb)
            )
        {
            throw new UnsupportedImageFormatException("Unsupported pixel format of
the source");
        }

        // lock source image
        BitmapData imageData = image.LockBits(
            new Rectangle(0, 0, image.Width, image.Height),
```

```
            ImageLockMode.ReadOnly, image.PixelFormat);

        List<IntPoint> corners;

        try
        {
            // process the image
            corners = ProcessImage(new UnmanagedImage(imageData));
        }
        finally
        {
            // unlock image
            image.UnlockBits(imageData);
        }

        return corners;
    }

}
```

LAMPIRAN
ALGORITMA PROSES *CORRELATION*

```
/ Accord Imaging Library
/ Accord.NET framework
/ http://www.crsouza.com
/
/ Copyright © César Souza, 2009-2010
/ cesarsouza at gmail.com
/

amespace Accord.Imaging

   using System;
   using System.Collections.Generic;
   using System.Drawing;
   using System.Drawing.Imaging;
   using Accord.Math;
   using AForge;
   using AForge.Imaging.Filters;

   /// <summary>
   ///   Maximum cross-correlation feature point matching algorithm.
   /// </summary>
   /// <remarks>
   ///   <para>
   ///     This class matches feature points by using a maximum cross-correlation
easure.</para>
   ///   <para>
   ///     References:
   ///     <list type="bullet">
   ///       <item><description>
   ///         P. D. Kovesi. MATLAB and Octave Functions for Computer Vision and Image
rocessing.
   ///         School of Computer Science and Software Engineering, The University of
estern Australia.
   ///         Available in: <a
ref="http://www.csse.uwa.edu.au/~pk/Research/MatlabFns/Match/matchbycorrelation.m">
   ///
ttp://www.csse.uwa.edu.au/~pk/Research/MatlabFns/Match/matchbycorrelation.m</a>
   ///       </description></item>
   ///       <item><description>
   ///         <a
ref="http://www.instructor.com.br/unesp2006/premiados/PauloHenrique.pdf">
   ///         http://www.instructor.com.br/unesp2006/premiados/PauloHenrique.pdf</a>
   ///       </description></item>
   ///       <item><description>
   ///         <a href="http://siddhantahuja.wordpress.com/2010/04/11/correlation-based-
imilarity-measures-summary/">
   ///         http://siddhantahuja.wordpress.com/2010/04/11/correlation-based-
imilarity-measures-summary/</a>
   ///       </description></item>
   ///     </list></para>
   /// </remarks>
   ///
   /// <seealso cref="RansacHomographyEstimator"/>
   ///
```

```
public class CorrelationMatching
{

    private int windowSize;
    private double dmax;

    /// <summary>
    ///   Gets or sets the maximum distance to consider
    ///   points as correlated.
    /// </summary>
    public double DistanceMax
    {
        get { return dmax; }
        set { dmax = value; }
    }

    /// <summary>
    ///   Gets or sets the size of the correlation window.
    /// </summary>
    public int WindowSize
    {
        get { return windowSize; }
        set { windowSize = value; }
    }



    /// <summary>
    ///   Constructs a new Correlation Matching algorithm.
    /// </summary>
    public CorrelationMatching(int windowSize)
        : this(windowSize, 0)
    {
    }

    /// <summary>
    ///   Constructs a new Correlation Matching algorithm.
    /// </summary>
    public CorrelationMatching(int windowSize, double maxDistance)
    {
        if (windowSize % 2 == 0)
            throw new ArgumentException("Window size should be odd", "windowSize");

        this.windowSize = windowSize;
        this.dmax = maxDistance;
    }



    /// <summary>
    ///   Matches two sets of feature points computed from the given images.
    /// </summary>
    public IntPoint[][] Match(Bitmap image1, Bitmap image2,
        IntPoint[] points1, IntPoint[] points2)
    {
        // Make sure we are dealing with grayscale images.
        Bitmap grayImage1, grayImage2;
        if (image1.PixelFormat == PixelFormat.Format8bppIndexed)
```

```
        {
            grayImage1 = image1;
        }
        else
        {
            // create temporary grayscale image
            grayImage1 = Grayscale.CommonAlgorithms.BT709.Apply(image1);
        }

        if (image2.PixelFormat == PixelFormat.Format8bppIndexed)
        {
            grayImage2 = image2;
        }
        else
        {
            // create temporary grayscale image
            grayImage2 = Grayscale.CommonAlgorithms.BT709.Apply(image2);
        }


        // Generate correlation matrix
        double[,] correlationMatrix =
            computeCorrelationMatrix(grayImage1, points1, grayImage2, points2,
indowSize, dmax);


        // Free allocated resources
        if (image1.PixelFormat != PixelFormat.Format8bppIndexed)
            grayImage1.Dispose();

        if (image2.PixelFormat != PixelFormat.Format8bppIndexed)
            grayImage2.Dispose();


        // Select points with maximum correlation measures
        int[] colp2forp1; Matrix.Max(correlationMatrix, 1, out colp2forp1);
        int[] rowp1forp2; Matrix.Max(correlationMatrix, 0, out rowp1forp2);

        // Construct the lists of matched point indices
        int rows = correlationMatrix.GetLength(0);
        List<int> p1ind = new List<int>();
        List<int> p2ind = new List<int>();

        // For each point in the first set of points,
        for (int i = 0; i < rows; i++)
        {
            // Get the point j in the second set of points with which
            // this point i has a maximum correlation measure. (i->j)
            int j = colp2forp1[i];

            // Now, check if this point j in the second set also has
            // a maximum correlation measure with the point i. (j->i)
            if (rowp1forp2[j] == i)
            {
                // The points are consistent. Ensure they are valid.
                if (correlationMatrix[i, j] != Double.NegativeInfinity)
                {
                    // We have a corresponding pair (i,j)
```

```
                    p1ind.Add(i); p2ind.Add(j);
                }
            }
        }

        // Extract matched points from original arrays
        var m1 = points1.Submatrix(p1ind.ToArray());
        var m2 = points2.Submatrix(p2ind.ToArray());

        // Create matching point pairs
        return new IntPoint[][] { m1, m2 };
    }


    /// <summary>
    ///   Constructs the correlation matrix between selected points from two images.
    /// </summary>
    /// <remarks>
    ///   Rows correspond to points from the first image, columns correspond to
oints
    ///   in the second.
    /// </remarks>
    private static double[,] computeCorrelationMatrix(
        Bitmap image1, IntPoint[] points1,
        Bitmap image2, IntPoint[] points2,
        int windowSize, double maxDistance)
    {

        // Create the initial correlation matrix
        double[,] matrix = Matrix.Create(points1.Length, points2.Length,
ouble.NegativeInfinity);

        // Gather some information
        int width1 = image1.Width;
        int width2 = image2.Width;
        int height1 = image1.Height;
        int height2 = image2.Height;

        int r = (windowSize - 1) / 2; //  'radius' of correlation window
        double m = maxDistance * maxDistance; // maximum considered distance
        double[,] w1 = new double[windowSize, windowSize]; // first window
        double[,] w2 = new double[windowSize, windowSize]; // second window

        // Lock the images
        BitmapData bitmapData1 = image1.LockBits(new Rectangle(0, 0, width1,
eight1),
            ImageLockMode.ReadOnly, PixelFormat.Format8bppIndexed);
        BitmapData bitmapData2 = image2.LockBits(new Rectangle(0, 0, width2,
eight2),
            ImageLockMode.ReadOnly, PixelFormat.Format8bppIndexed);

        int stride1 = bitmapData1.Stride;
        int stride2 = bitmapData2.Stride;


        // We will ignore points at the edge
        int[] idx1 = Matrix.Find(points1,
            p => p.X >= r && p.X < width1 - r &&
```

```
            p.Y >= r && p.Y < height1 - r);

        int[] idx2 = Matrix.Find(points2,
            p => p.X >= r && p.X < width2 - r &&
                 p.Y >= r && p.Y < height2 - r);


        // For each index in the first set of points
        foreach (int n1 in idx1)
        {
            // Get the current point
            var p1 = points1[n1];

            unsafe // Create the first window for the current point
            {
                byte* src = (byte*)bitmapData1.Scan0 + (p1.X - r) + (p1.Y - r) *
tride1;


                for (int j = 0; j < windowSize; j++)
                {
                    for (int i = 0; i < windowSize; i++)
                        w1[i, j] = (byte)(*(src + i));
                    src += stride1;
                }
            }

            // Normalize the window
            double sum = 0;
            for (int i = 0; i < windowSize; i++)
                for (int j = 0; j < windowSize; j++)
                    sum += w1[i, j] * w1[i, j];
            sum = System.Math.Sqrt(sum);
            for (int i = 0; i < windowSize; i++)
                for (int j = 0; j < windowSize; j++)
                    w1[i, j] /= sum;


            // Identify the indices of points in p2 that we need to consider.
            int[] candidates;
            if (maxDistance == 0)
            {
                // We should consider all points
                candidates = idx2;
            }
            else
            {
                // We should consider points that are within
                //  distance maxDistance apart

                // Compute distances from the current point
                //  to all points in the second image.
                double[] distances = new double[idx2.Length];
                for (int i = 0; i < idx2.Length; i++)
                {
                    double dx = p1.X - points2[idx2[i]].X;
                    double dy = p1.Y - points2[idx2[i]].Y;
                    distances[i] = dx * dx + dy * dy;
                }
```

```
                candidates = idx2.Submatrix(Matrix.Find(distances, d => d < m));
            }


            // Calculate normalized correlation measure
            foreach (int n2 in candidates)
            {
                var p2 = points2[n2];

                unsafe // Generate window in 2nd image
                {
                    byte* src = (byte*)bitmapData2.Scan0 + (p2.X - r) + (p2.Y - r) *
tride2;


                    for (int j = 0; j < windowSize; j++)
                    {
                        for (int i = 0; i < windowSize; i++)
                            w2[i, j] = (byte)(*(src + i));
                        src += stride2;
                    }
                }

                double sum1 = 0, sum2 = 0;
                for (int i = 0; i < windowSize; i++)
                {
                    for (int j = 0; j < windowSize; j++)
                    {
                        sum1 += w1[i, j] * w2[i, j];
                        sum2 += w2[i, j] * w2[i, j];
                    }
                }

                matrix[n1, n2] = sum1 / System.Math.Sqrt(sum2);
            }
        }

        // Release the images
        image1.UnlockBits(bitmapData1);
        image2.UnlockBits(bitmapData2);

        return matrix;
    }

}
```

LAMPIRAN
ALGORITMA PROSES RANSAC

```
/ Accord Imaging Library
/ Accord.NET framework
/ http://www.crsouza.com
/
/ Copyright © César Souza, 2009-2010
/ cesarsouza at gmail.com
/

amespace Accord.Imaging

    using System;
    using System.Drawing;
    using Accord.MachineLearning;
    using Accord.Math;
    using AForge;

    /// <summary>
    ///   RANSAC Robust Homography Matrix Estimator.
    /// </summary>
    ///
    /// <remarks>
    /// <para>
    ///   Fitting a homography using RANSAC is pretty straightforward. Being a iterative
method,
    ///   in a single iteration a random sample of four correspondences is selected from
the
    ///   given correspondence points and a homography H is then computed from those
points.</para>
    /// <para>
    ///   The original points are then transformed using this homography and their
distances to
    ///   where those transforms should be is then computed and matching points can
classified
    ///   as inliers and non-matching points as outliers.</para>
    /// <para>
    ///   After a given number of iterations, the iteration which produced the largest
number
    ///   of inliers is then selected as the best estimation for H.</para>
    ///
    /// <para>
    ///   References:
    ///   <list type="bullet">
    ///     <item><description>
    ///       http://www.cs.ubc.ca/grads/resources/thesis/May09/Dubrofsky_Elan.pdf
</description></item>
    ///     <item><description>
    ///       http://www.cc.gatech.edu/classes/AY2005/cs4495_fall/assignment4.pdf
</description></item>
    ///   </list></para>
    /// </remarks>
    ///
    public class RansacHomographyEstimator
    {
        private RANSAC<MatrixH> ransac;
```

```
private int[] inliers;

private PointF[] pointSet1;
private PointF[] pointSet2;


/// <summary>
///   Gets the RANSAC estimator used.
/// </summary>
public RANSAC<MatrixH> Ransac
{
    get { return this.ransac; }
}

/// <summary>
///   Gets the final set of inliers detected by RANSAC.
/// </summary>
public int[] Inliers
{
    get { return inliers; }
}


/// <summary>
///   Creates a new RANSAC homography estimator.
/// </summary>
/// <param name="threshold">Inlier threshold.</param>
/// <param name="probability">Inlier probability.</param>
public RansacHomographyEstimator(double threshold, double probability)
{
    // Create a new RANSAC with the selected threshold
    ransac = new RANSAC<MatrixH>(4, threshold, probability);

    // Set RANSAC functions
    ransac.Fitting = homography;
    ransac.Degenerate = degenerate;
    ransac.Distances = distance;
}

/// <summary>
///   Matches two sets of points using RANSAC.
/// </summary>
/// <returns>The homography matrix matching x1 and x2.</returns>
public MatrixH Estimate(Point[] points1, Point[] points2)
{
    // Initial argument checkings
    if (points1.Length != points2.Length)
        throw new ArgumentException("The number of points should be equal.");

    if (points1.Length < 4)
        throw new ArgumentException("At least four points are required to fit an
homography");

    PointF[] p1 = new PointF[points1.Length];
    PointF[] p2 = new PointF[points2.Length];
    for (int i = 0; i < points1.Length; i++)
    {
        p1[i] = new PointF(points1[i].X, points1[i].Y);
```

```
            p2[i] = new PointF(points2[i].X, points2[i].Y);
        }

        return Estimate(p1, p2);
    }

    /// <summary>
    ///   Matches two sets of points using RANSAC.
    /// </summary>
    /// <returns>The homography matrix matching x1 and x2.</returns>
    public MatrixH Estimate(IntPoint[] points1, IntPoint[] points2)
    {
        // Initial argument checkings
        if (points1.Length != points2.Length)
            throw new ArgumentException("The number of points should be equal.");

        if (points1.Length < 4)
            throw new ArgumentException("At least four points are required to fit an
omography");

        PointF[] p1 = new PointF[points1.Length];
        PointF[] p2 = new PointF[points2.Length];
        for (int i = 0; i < points1.Length; i++)
        {
            p1[i] = new PointF(points1[i].X, points1[i].Y);
            p2[i] = new PointF(points2[i].X, points2[i].Y);
        }

        return Estimate(p1, p2);
    }

    /// <summary>
    ///   Matches two sets of points using RANSAC.
    /// </summary>
    /// <returns>The homography matrix matching x1 and x2.</returns>
    public MatrixH Estimate(PointF[] points1, PointF[] points2)
    {
        // Initial argument checkings
        if (points1.Length != points2.Length)
            throw new ArgumentException("The number of points should be equal.");

        if (points1.Length < 4)
            throw new ArgumentException("At least four points are required to fit an
homography");


        // Normalize each set of points so that the origin is
        //  at centroid and mean distance from origin is sqrt(2).
        MatrixH T1, T2;
        this.pointSet1 = Tools.Normalize(points1, out T1);
        this.pointSet2 = Tools.Normalize(points2, out T2);


        // Compute RANSAC and find the inlier points
        MatrixH H = ransac.Compute(points1.Length, out inliers);

        if (inliers == null || inliers.Length < 4)
```

```
            //throw new Exception("RANSAC could not find enough points to fit an
omography.");
            return null;


        // Compute the final homography considering all inliers
        H = homography(inliers);

        // Denormalise
        H = T2.Inverse() * (H * T1);

        return H;
    }

    /// <summary>
    ///   Estimates a homography with the given points.
    /// </summary>
    private MatrixH homography(int[] points)
    {
        // Retrieve the original points
        PointF[] x1 = this.pointSet1.Submatrix(points);
        PointF[] x2 = this.pointSet2.Submatrix(points);

        // Compute the homography
        return Tools.Homography(x1, x2);
    }

    /// <summary>
    ///   Compute inliers using the Symmetric Transfer Error,
    /// </summary>
    private int[] distance(MatrixH H, double t)
    {
        int n = pointSet1.Length;

        // Compute the projections (both directions)
        PointF[] p1 = H.TransformPoints(pointSet1);
        PointF[] p2 = H.Inverse().TransformPoints(pointSet2);

        // Compute the distances
        double[] d2 = new double[n];
        for (int i = 0; i < n; i++)
        {
            // Compute the distance as
            float ax = pointSet1[i].X - p2[i].X;
            float ay = pointSet1[i].Y - p2[i].Y;
            float bx = pointSet2[i].X - p1[i].X;
            float by = pointSet2[i].Y - p1[i].Y;
            d2[i] = (ax * ax) + (ay * ay) + (bx * bx) + (by * by);
        }

        // Find and return the inliers
        return Matrix.Find(d2, z => z < t);
    }

    /// <summary>
    ///   Checks if the selected points will result in a degenerate homography.
    /// </summary>
    private bool degenerate(int[] points)
```

```
    {
        PointF[] x1 = this.pointSet1.Submatrix(points);
        PointF[] x2 = this.pointSet2.Submatrix(points);

        // If any three of the four points in each set is colinear,
        //  the resulting homography matrix will be degenerate.

        return Tools.Colinear(x1[0], x1[1], x1[2]) ||
               Tools.Colinear(x1[0], x1[1], x1[3]) ||
               Tools.Colinear(x1[0], x1[2], x1[3]) ||
               Tools.Colinear(x1[1], x1[2], x1[3]) ||

               Tools.Colinear(x2[0], x2[1], x2[2]) ||
               Tools.Colinear(x2[0], x2[1], x2[3]) ||
               Tools.Colinear(x2[0], x2[2], x2[3]) ||
               Tools.Colinear(x2[1], x2[2], x2[3]);
    }

}
```

LAMPIRAN
ALGORITMA PROSES *POINT HOMOGRAPHY*

```
// Accord Imaging Library
// Accord.NET framework
// http://www.crsouza.com
//
// Copyright © César Souza, 2009-2010
// cesarsouza at gmail.com
//

namespace Accord.Imaging
{
    using System.Drawing;
    using System;

    /// <summary>
    ///   Represents an ordered pair of real x- and y-coordinates and scalar w that defines
    ///   a point in a two-dimensional plane using homogeneous coordinates.
    /// </summary>
    ///
    /// <remarks>
    /// <para>
    ///   In mathematics, homogeneous coordinates are a system of coordinates used in
    ///   projective geometry much as Cartesian coordinates are used in Euclidean
geometry.</para>
    /// <para>
    ///   They have the advantage that the coordinates of a point, even those at infinity,
    ///   can be represented using finite coordinates. Often formulas involving homogeneous
    ///   coordinates are simpler and more symmetric than their Cartesian counterparts.</para>
    /// <para>
    ///   Homogeneous coordinates have a range of applications, including computer graphics,
    ///   where they allow affine transformations and, in general, projective transformations
    ///   to be easily represented by a matrix.</para>
    ///
    /// <para>
    ///   References:
    ///   <list type="bullet">
    ///     <item><description>
    ///       http://alumnus.caltech.edu/~woody/docs/3dmatrix.html</description></item>
    ///     <item><description>
    ///       http://simply3d.wordpress.com/2009/05/29/homogeneous-
coordinates/</description></item>
    ///   </list></para>
    /// </remarks>
    ///
    public struct PointH
    {
        private float px, py, pw;

        /// <summary>
        ///   The first coordinate.
        /// </summary>
        public float X
        {
            get { return px; }
            set { px = value; }
        }

        /// <summary>
```

```
///   The second coordinate.
/// </summary>
public float Y
{
    get { return py; }
    set { py = value; }
}

/// <summary>
///   The inverse scaling factor for X and Y.
/// </summary>
public float W
{
    get { return pw; }
    set { pw = value; }
}

/// <summary>
///   Creates a new point.
/// </summary>
public PointH(float x, float y)
{
    px = x;
    py = y;
    pw = 1;
}

/// <summary>
///   Creates a new point.
/// </summary>
public PointH(float x, float y, float w)
{
    px = x;
    py = y;
    pw = w;
}

/// <summary>
///   Transforms a point using a projection matrix.
/// </summary>
public void Transform(float[,] matrix)
{
    px = matrix[0, 0] * px + matrix[0, 1] * py + matrix[0, 2] * pw;
    py = matrix[1, 0] * px + matrix[1, 1] * py + matrix[1, 2] * pw;
    pw = matrix[2, 0] * px + matrix[2, 1] * py + matrix[2, 2] * pw;
}

/// <summary>
///   Normalizes the point to have unit scale.
/// </summary>
public void Normalize()
{
    px = px / pw;
    py = py / pw;
    pw = 1;
}

/// <summary>
///   Gets whether this point is normalized (w = 1).
/// </summary>
public bool IsNormalized
```

```
{
    get { return pw == 1f; }
}

/// <summary>
///   Gets whether this point is at infinity (w = 0).
/// </summary>
public bool IsAtInfinity
{
    get { return pw == 0f; }
}

/// <summary>
///   Gets whether this point is at the origin.
/// </summary>
public bool IsEmpty
{
    get { return px == 0 && py == 0; }
}

/// <summary>
///   Converts the point to a array representation.
/// </summary>
public double[] ToArray()
{
    return new double[] { px, py, pw };
}

/// <summary>
///   Multiplication by scalar.
/// </summary>
public static PointH operator *(PointH a, float b)
{
    return new PointH(b * a.X, b * a.Y, b * a.W);
}

/// <summary>
///   Multiplication by scalar.
/// </summary>
public static PointH operator *(float b, PointH a)
{
    return a * b;
}

/// <summary>
///   Subtraction.
/// </summary>
public static PointH operator -(PointH a, PointH b)
{
    return new PointH(a.X - b.X, a.Y - b.Y, a.W - b.W);
}

/// <summary>
///   Addition.
/// </summary>
public static PointH operator +(PointH a, PointH b)
{
    return new PointH(a.X + b.X, a.Y + b.Y, a.W + b.W);
}

/// <summary>
```

```
///   Equality
/// </summary>
public static bool operator ==(PointH a, PointH b)
{
    return (a.px / a.pw == b.px / b.pw && a.py / a.pw == b.py / b.pw);
}

/// <summary>
///   Inequality
/// </summary>
public static bool operator !=(PointH a, PointH b)
{
    return (a.px / a.pw != b.px / b.pw || a.py / a.pw != b.py / b.pw);
}

/// <summary>
///   PointF Conversion
/// </summary>
public static implicit operator PointF(PointH a)
{
    return new PointF((float)(a.px / a.pw), (float)(a.py / a.pw));
}

/// <summary>
///   Converts to a Integer point by computing the ceiling of the point coordinates.
/// </summary>
public static Point Ceiling(PointH point)
{
    return new Point(
        (int)System.Math.Ceiling(point.px / point.pw),
        (int)System.Math.Ceiling(point.py / point.pw));
}

/// <summary>
///   Converts to a Integer point by rounding the point coordinates.
/// </summary>
public static Point Round(PointH point)
{
    return new Point(
        (int)System.Math.Round(point.px / point.pw),
        (int)System.Math.Round(point.py / point.pw));
}

/// <summary>
///   Converts to a Integer point by truncating the point coordinates.
/// </summary>
public static Point Truncate(PointH point)
{
    return new Point(
        (int)System.Math.Truncate(point.px / point.pw),
        (int)System.Math.Truncate(point.py / point.pw));
}

/// <summary>
///   Compares two objects for equality.
/// </summary>
public override bool Equals(object obj)
{
    if (obj is PointH)
    {
        PointH p = (PointH)obj;
```

```
            if (px / pw == p.px / p.pw &&
                py / pw == p.py / p.pw)
                return true;
        }

        return false;
    }

    /// <summary>
    ///   Returns the hash code for this instance.
    /// </summary>
    public override int GetHashCode()
    {
        return px.GetHashCode() ^ py.GetHashCode() ^ pw.GetHashCode();
    }



    /// <summary>
    ///   Returns the empty point.
    /// </summary>
    public static readonly PointH Empty = new PointH(0, 0, 1);
}
```

LAMPIRAN
ALGORITMA PROSES *MATRIX HOMOGRAPHY*

```
' Accord Imaging Library
' Accord.NET framework
' http://www.crsouza.com
'
' Copyright © César Souza, 2009-2010
' cesarsouza at gmail.com
'

namespace Accord.Imaging

  using System;
  using System.Drawing;

  /// <summary>
  ///   Encapsulates a 3-by-3 general transformation matrix that represents
  ///   a (possibly) non-linear transform.
  /// </summary>
  /// <remarks>
  /// <para>
  ///   Linear transformations are not the only ones that can be represented by
  ///   matrices. Using homogeneous coordinates, both affine transformations and
  ///   perspective projections on R^n can be represented as linear transformations
  ///   on R^n+1 (that is, n+1-dimensional real projective space).</para>
  /// <para>
  ///   The general transformation matrix has 8 degrees of freedom, as the last element
s just a scale parameter.</para>
  /// </remarks>
  ///
  [Serializable]
  public class MatrixH
  {

      private float[] elements;

      /// <summary>
      ///   Creates a new projective matrix.
      /// </summary>
      public MatrixH()
      {
          // Start as the identity matrix
          this.elements = new float[] { 1, 0, 0, 0, 1, 0, 0, 0 };
      }

      /// <summary>
      ///   Creates a new projective matrix.
      /// </summary>
      public MatrixH(float m11, float m12, float m13,
                     float m21, float m22, float m23,
                     float m31, float m32)
      {
          this.elements = new float[8];
          this.elements[0] = m11; this.elements[1] = m12; this.elements[2] = m13;
          this.elements[3] = m21; this.elements[4] = m22; this.elements[5] = m23;
          this.elements[6] = m31; this.elements[7] = m32;
```

```
}

/// <summary>
///   Creates a new projective matrix.
/// </summary>
public MatrixH(float m11, float m12, float m13,
               float m21, float m22, float m23,
               float m31, float m32, float m33)
    : this(m11, m12, m13, m21, m22, m23, m31, m32)
{
    for (int i = 0; i < 8; i++)
        elements[i] /= m33;
}

/// <summary>
///   Creates a new projective matrix.
/// </summary>
public MatrixH(double[,] H)
{
    this.elements = new float[8];
    for (int i = 0, k = 0; i < 3; i++)
        for (int j = 0; j < 3 && k < 8; j++, k++)
            this.elements[k] = (float)(H[i, j] / H[2, 2]);
}

/// <summary>
///   Gets the elements of this matrix.
/// </summary>
public float[] Elements
{
    get { return elements; }
}

/// <summary>
///   Gets the offset x
/// </summary>
public float OffsetX
{
    get { return elements[2]; }
}

/// <summary>
///   Gets the offset y
/// </summary>
public float OffsetY
{
    get { return elements[5]; }
}

/// <summary>
///   Gets whether this matrix is invertible.
/// </summary>
public bool IsInvertible
{
    get
    {
        float det = elements[0] * (elements[4] - elements[5] * elements[7])
            - elements[1] * (elements[3] - elements[5] * elements[6])
```

```
                + elements[2] * (elements[3] * elements[7] - elements[4] *
ements[6]);

            return det > 0;
        }
    }

    /// <summary>
    ///   Gets whether this is an Affine transformation matrix.
    /// </summary>
    public bool IsAffine
    {
        get { return (elements[6] == 0 && elements[7] == 0); }
    }

    /// <summary>
    ///   Gets whether this is the identity transformation.
    /// </summary>
    public bool IsIdentity
    {
        get
        {
            return
                elements[0] == 1 && elements[1] == 0 && elements[2] == 0 &&
                elements[3] == 0 && elements[4] == 1 && elements[5] == 0 &&
                elements[6] == 0 && elements[7] == 0;
        }
    }

    /// <summary>
    ///   Resets this matrix to be the identity.
    /// </summary>
    public void Reset()
    {
        elements[0] = 1; elements[1] = 0; elements[2] = 0;
        elements[3] = 0; elements[4] = 1; elements[5] = 0;
        elements[6] = 0; elements[7] = 0;
    }

    /// <summary>
    ///   Returns the inverse matrix, if this matrix is invertible.
    /// </summary>
    public MatrixH Inverse()
    {
        //    m = 1 / [a(ei-fh) - b(di-fg) + c(dh-eg)]
        //
        //                  (ei-fh)   (ch-bi)   (bf-ce)
        //  inv(A) =   m  x (fg-di)   (ai-cg)   (cd-af)
        //                  (dh-eg)   (bg-ah)   (ae-bd)
        //

        float a = this.elements[0], b = this.elements[1], c = this.elements[2];
        float d = this.elements[3], e = this.elements[4], f = this.elements[5];
        float g = this.elements[6], h = this.elements[7];

        float m = 1f / (a * (e - f * h) - b * (d - f * g) + c * (d * h - e * g));
        float na = m * (e - f * h);
        float nb = m * (c * h - b);
```

```
            float nc = m * (b * f - c * e);
            float nd = m * (f * g - d);
            float ne = m * (a - c * g);
            float nf = m * (c * d - a * f);
            float ng = m * (d * h - e * g);
            float nh = m * (b * g - a * h);
            float nj = m * (a * e - b * d);

            return new MatrixH(na, nb, nc, nd, ne, nf, ng, nh, nj);
        }

        /// <summary>
        ///   Transforms the given points using this transformation matrix.
        /// </summary>
        public PointH[] TransformPoints(params PointH[] points)
        {
            PointH[] r = new PointH[points.Length];

            for (int j = 0; j < points.Length; j++)
            {
                r[j].X = elements[0] * points[j].X + elements[1] * points[j].Y +
lements[2] * points[j].W;
                r[j].Y = elements[3] * points[j].X + elements[4] * points[j].Y +
lements[5] * points[j].W;
                r[j].W = elements[6] * points[j].X + elements[7] * points[j].Y +
oints[j].W;
            }

            return r;
        }

        /// <summary>
        ///   Transforms the given points using this transformation matrix.
        /// </summary>
        public PointF[] TransformPoints(params PointF[] points)
        {
            PointF[] r = new PointF[points.Length];

            for (int j = 0; j < points.Length; j++)
            {
                float w = elements[6] * points[j].X + elements[7] * points[j].Y + 1f;
                r[j].X = (elements[0] * points[j].X + elements[1] * points[j].Y +
elements[2]) / w;
                r[j].Y = (elements[3] * points[j].X + elements[4] * points[j].Y +
elements[5]) / w;
            }

            return r;
        }

        /// <summary>
        ///   Multiplies this matrix, returning a new matrix as result.
        /// </summary>
        public MatrixH Multiply(MatrixH matrix)
        {
            float na = elements[0] * matrix.elements[0] + elements[1] *
matrix.elements[3] + elements[2] * matrix.elements[6];
```

```
            float nb = elements[0] * matrix.elements[1] + elements[1] *
atrix.elements[4] + elements[2] * matrix.elements[7];
            float nc = elements[0] * matrix.elements[2] + elements[1] *
atrix.elements[5] + elements[2];

            float nd = elements[3] * matrix.elements[0] + elements[4] *
atrix.elements[3] + elements[5] * matrix.elements[6];
            float ne = elements[3] * matrix.elements[1] + elements[4] *
atrix.elements[4] + elements[5] * matrix.elements[7];
            float nf = elements[3] * matrix.elements[2] + elements[4] *
atrix.elements[5] + elements[5];

            float ng = elements[6] * matrix.elements[0] + elements[7] *
atrix.elements[3] + matrix.elements[6];
            float nh = elements[6] * matrix.elements[1] + elements[7] *
atrix.elements[4] + matrix.elements[7];
            float ni = elements[6] * matrix.elements[2] + elements[7] *
atrix.elements[5] + 1f;

            return new MatrixH(na, nb, nc, nd, ne, nf, ng, nh, ni);
        }

        /// <summary>
        ///   Compares two objects for equality.
        /// </summary>
        public override bool Equals(object obj)
        {
            if (obj is MatrixH)
            {
                MatrixH m = obj as MatrixH;
                return this == m;
            }
            else
            {
                return false;
            }
        }

        /// <summary>
        ///   Returns the hash code for this instance.
        /// </summary>
        public override int GetHashCode()
        {
            return elements.GetHashCode();
        }

        /// <summary>
        ///   Double[,] conversion.
        /// </summary>
        public static explicit operator double[,](MatrixH matrix)
        {
            return new double[,]
            {
                { matrix.elements[0], matrix.elements[1], matrix.elements[2] },
                { matrix.elements[3], matrix.elements[4], matrix.elements[5] },
                { matrix.elements[6], matrix.elements[7], 1.0 },
            };
        }
```

```
    /// <summary>
    ///   Single[,] conversion.
    /// </summary>
    public static explicit operator float[,](MatrixH matrix)
    {
        return new float[,]
        {
            { matrix.elements[0], matrix.elements[1], matrix.elements[2] },
            { matrix.elements[3], matrix.elements[4], matrix.elements[5] },
            { matrix.elements[6], matrix.elements[7], 1.0f },
        };
    }

    /// <summary>
    ///   Matrix multiplication.
    /// </summary>
    public static MatrixH operator *(MatrixH matrix1, MatrixH matrix2)
    {
        return matrix1.Multiply(matrix2);
    }

    /// <summary>
    ///   Equality
    /// </summary>
    public static bool operator ==(MatrixH a, MatrixH b)
    {
        for (int i = 0; i < 8; i++)
            if (a.elements[i] != b.elements[i])
                return false;

        return true;
    }

    /// <summary>
    ///   Inequality
    /// </summary>
    public static bool operator !=(MatrixH a, MatrixH b)
    {
        for (int i = 0; i < 8; i++)
            if (a.elements[i] == b.elements[i])
                return true;

        return false;
    }

}
```

LAMPIRAN

ALGORITMA PROSES *IMAGE BLENDING*

```
/ Accord Imaging Library
/ Accord.NET framework
/ http://www.crsouza.com
/
/ Copyright © César Souza, 2009-2010
/ cesarsouza at gmail.com
/

amespace Accord.Imaging.Filters

   using System.Collections.Generic;
   using System.Drawing;
   using System.Diagnostics;
   using System.Drawing.Imaging;
   using AForge.Imaging;
   using System;
   using Matrix = Accord.Math.Matrix;


   /// <summary>
   ///   Linear Gradient Blending filter.
   /// </summary>
   /// <remarks>
   /// <para>
   ///   The blending filter is able to blend two images using a
       homography matrix.
   ///   A linear alpha gradient is used to smooth out differences between
       the two
   ///   images, effectively blending them in two images. The gradient is
       computed
   ///   considering the distance between the centers of the two
       images.</para>
   /// <para>
   ///   The first image should be passed at the moment of creation of the
       Blending
   ///   filter as the overlay image. A second image may be projected on
       top of the
   ///   overlay image by calling the Apply method and passing the second
       image as
   ///   argument.</para>
   /// <para>
   ///   Currently the filter always produces 32bpp images, disregarding
       the format
   ///   of source images. The alpha layer is used as an intermediate mask
       in the
   ///   blending process.</para>
   /// </remarks>
   public class Blend : AForge.Imaging.Filters.BaseTransformationFilter
   {
       private MatrixH homography;
       private Bitmap overlayImage;
       private Point offset;
```

```
private Point center;
private Size imageSize;
private Color fillColor = Color.FromArgb(0, Color.Black);
private Dictionary<PixelFormat, PixelFormat> formatTranslations = new
Dictionary<PixelFormat, PixelFormat>();



/// <summary>
/// Format translations dictionary.
/// </summary>
public override Dictionary<PixelFormat, PixelFormat> FormatTranslations
{
    get { return formatTranslations; }
}

/// <summary>
///   Gets or sets the Homography matrix used to map a image passed   to
///    the filter to the overlay image specified at filter creation.
/// </summary>
public MatrixH Homography
{
    get { return homography; }
    set { homography = value; }
}

/// <summary>
///   Gets or sets the filling color used to fill blank spaces.
/// </summary>
/// <remarks>
///   The filling color will only be visible after the image is converted
///    to 24bpp. The alpha channel will be used internally by the filter.
/// </remarks>
public Color FillColor
{
    get { return fillColor; }
    set { fillColor = value; }
}

/// <summary>
///   Constructs a new Blend filter.
/// </summary>
/// <param name="homography">The homography matrix mapping a second image to the
     overlay image.</param>
/// <param name="overlayImage">The overlay image (also called the
     anchor).</param>
public Blend(double[,] homography, Bitmap overlayImage)
    : this(new MatrixH(homography), overlayImage)
{
}

/// <summary>
///   Constructs a new Blend filter.
/// </summary>
/// <param name="homography">The homography matrix mapping a second image to the
    overlay image.</param>
/// <param name="overlayImage">The overlay image (also called the
     anchor).</param>
```

```
public Blend(MatrixH homography, Bitmap overlayImage)
{
    this.homography = homography;
    this.overlayImage = overlayImage;
    formatTranslations[PixelFormat.Format8bppIndexed] =
    PixelFormat.Format32bppArgb;
    formatTranslations[PixelFormat.Format24bppRgb] = PixelFormat.Format32bppArgb;
    formatTranslations[PixelFormat.Format32bppArgb] =
    PixelFormat.Format32bppArgb;
}

/// <summary>
///   Computes the new image size.
/// </summary>
protected override Size CalculateNewImageSize(UnmanagedImage sourceData)
{
    // Calculate source size
    float w = sourceData.Width;
    float h = sourceData.Height;

    // Get the four corners and the center of the image
    PointF[] corners =
    {
        new PointF(0, 0),
        new PointF(w, 0),
        new PointF(0, h),
        new PointF(w, h),
        new PointF(w / 2f, h / 2f)
    };

    // Project those points
    corners = homography.Inverse().TransformPoints(corners);

    // Recalculate image size
    float[] px = { corners[0].X, corners[1].X, corners[2].X, corners[3].X };
    float[] py = { corners[0].Y, corners[1].Y, corners[2].Y, corners[3].Y };

    float maxX = Matrix.Max(px);
    float minX = Matrix.Min(px);
    float newWidth = Math.Max(maxX, overlayImage.Width) - Math.Min(0, minX);

    float maxY = Accord.Math.Matrix.Max(py);
    float minY = Accord.Math.Matrix.Min(py);
    float newHeight = Math.Max(maxY, overlayImage.Height) - Math.Min(0, minY);


    // Store overlay image size
    this.imageSize = new Size((int)Math.Round(maxX-minX), (int)Math.Round(maxY-
    minY));

// Store image center
this.center = Point.Round(corners[4]);

// Calculate and store image offset
int offsetX = 0, offsetY = 0;
if (minX < 0) offsetX = (int)Math.Round(minX);
if (minY < 0) offsetY = (int)Math.Round(minY);
```

```
is.offset = new Point(offsetX, offsetY);

 Return the final image size
eturn new Size((int)Math.Ceiling(newWidth), (int)Math.Ceiling(newHeight));



ummary>
 Process the image filter.
/summary>
ted override void ProcessFilter(UnmanagedImage sourceData, UnmanagedImage
estinationData)


/ Locks the overlay image
itmapData overlayData = overlayImage.LockBits(new Rectangle(0, 0, overlayImage.Width,
verlayImage.Height),ImageLockMode.ReadOnly, overlayImage.PixelFormat);


/ get source image size
nt width = sourceData.Width;
nt height = sourceData.Height;

/ get destination image size
nt newWidth = destinationData.Width;
nt newHeight = destinationData.Height;

nt srcPixelSize = System.Drawing.Image.GetPixelFormatSize(sourceData.PixelFormat) / 8;
nt orgPixelSize = System.Drawing.Image.GetPixelFormatSize(overlayData.PixelFormat) / 8;

nt srcStride = sourceData.Stride;
nt dstOffset = destinationData.Stride - newWidth * 4; // destination always 32bpp argb


/ Get center of first image
oint center1 = new Point((int)(overlayImage.Width / 2.0),(int)(overlayImage.Height /
.0));

/ Get center of second image
oint center2 = this.center;



/ Compute maximum center distances
louble dmax1 = Math.Min(
    distance(center1.X, center1.Y, center1.X + overlayImage.Width / 2.0, center1.Y),
distance(center1.X, center1.Y, center1.X, center1.X + overlayImage.Height / 2.0));

louble dmax2 = Math.Min(
    distance(center2.X, center2.Y, center2.X + imageSize.Width / 2.0, center2.Y),
distance(center2.X, center2.Y, center2.X, center2.Y + imageSize.Height / 2.0));


louble dmax = -System.Math.Abs(dmax2 - dmax1);

/ fill values
yte fillR = fillColor.R;
```

```
te fillG = fillColor.G;
te fillB = fillColor.B;
te fillA = 0;//fillColor.A;

 Retrieve homography matrix as float array
oat[,] H = (float[,])homography;


 do the job
safe

  byte* org = (byte*)overlayData.Scan0.ToPointer();
  byte* src = (byte*)sourceData.ImageData.ToPointer();
  byte* dst = (byte*)destinationData.ImageData.ToPointer();

  // destination pixel's coordinate relative to image center
  double cx, cy;

  // destination pixel's homogenous coordinate
  double hx, hy, hw;

  // source pixel's coordinates
  int ox, oy;



  // Copy the overlay image
  for (int y = 0; y < newHeight; y++)
  {
      for (int x = 0; x < newWidth; x++, dst += 4)
      {
          ox = (int)(x + offset.X);
          oy = (int)(y + offset.Y);

          // validate source pixel's coordinates
            if ((ox < 0) || (oy < 0) || (ox >= overlayData.Width) || (oy >=
            overlayData.Height))
          {
              // fill destination image with filler
              dst[0] = fillB;
              dst[1] = fillG;
              dst[2] = fillR;
              dst[3] = fillA;
          }
          else
          {
              int c = oy * overlayData.Stride + ox * orgPixelSize;

              // fill destination image with pixel from original image
              dst[0] = org[c];

              if (orgPixelSize >= 3)
              {
                  // 24/32 bpp
                  dst[1] = org[c + 1];
                  dst[2] = org[c + 2];
                  dst[3] = (orgPixelSize == 4)
                      ? org[c + 3] // 32 bpp
```

```
                    : (byte)255; // 24 bpp
            }
            else
            {
                // 8 bpp
                dst[1] = org[c];
                dst[2] = org[c];
                dst[3] = 255;
            }
        }
    }
    dst += dstOffset;
}

org = (byte*)overlayData.Scan0.ToPointer();
src = (byte*)sourceData.ImageData.ToPointer();
dst = (byte*)destinationData.ImageData.ToPointer();

// Project and blend the second image
for (int y = 0; y < newHeight; y++)
{
    for (int x = 0; x < newWidth; x++, dst += 4)
    {
        cx = x + offset.X;
        cy = y + offset.Y;

        // projection using homogenous coordinates
        hw = H[2, 0] * cx + H[2, 1] * cy + H[2, 2];
        hx = (H[0, 0] * cx + H[0, 1] * cy + H[0, 2]) / hw;
        hy = (H[1, 0] * cx + H[1, 1] * cy + H[1, 2]) / hw;

        // coordinate of the nearest point
        ox = (int)(hx);
        oy = (int)(hy);

        // validate source pixel's coordinates
        if ((ox >= 0) && (oy >= 0) && (ox < width) && (oy < height))
        {
            int c = oy * srcStride + ox * srcPixelSize;

            // fill destination image with pixel from source image
            if (dst[3] > 0)
            {
                // there is a pixel from the other image here, blend
                double d1 = distance(ox, oy, center1.X, center1.Y);
                double d2 = distance(ox, oy, center2.X, center2.Y);
                 double f = Accord.Math.Tools.Scale(0, dmax, 0, 1, d1 - d2);

                if (f < 0) f = 0;
                if (f > 1) f = 1;
                double f2 = (1.0 - f);

                dst[0] = (byte)(src[c] * f2 + dst[0] * f);

                if (srcPixelSize >= 3)
                {
                    // 24/32 bpp
                    dst[1] = (byte)(src[c + 1] * f2 + dst[1] * f);
```

```
                dst[2] = (byte)(src[c + 2] * f2 + dst[2] * f);
                dst[3] = (srcPixelSize == 4)
                    ? (byte)(src[c + 3] * f2 + dst[3] * f) // 32 bpp
                    : (byte)255;                           // 24 bpp
            }
            else
            {
                // 8 bpp
                dst[1] = (byte)(src[c] * f2 + dst[1] * f);
                dst[2] = (byte)(src[c] * f2 + dst[2] * f);
                dst[3] = (byte)255;
            }
        }
        else
        {
            // just copy the source into the destination image
            dst[0] = (byte)src[c];

            if (srcPixelSize >= 3)
            {
                // 24/32bpp
                dst[1] = (byte)src[c + 1];
                dst[2] = (byte)src[c + 2];
                dst[3] = (srcPixelSize == 4)
                    ? (byte)src[c + 3] // 32bpp
                    : (byte)255;       // 24bpp
            }
            else
            {
                // 8bpp
                dst[1] = (byte)src[c];
                dst[2] = (byte)src[c];
                dst[3] = (byte)255;
            }
                    }
                }
            }
            dst += dstOffset;

        }
    }

    overlayImage.UnlockBits(overlayData);
}

/// <summary>
///   Computes a distance metric used to compute the blending mask
/// </summary>
private static double distance(double x1, double y1, double x2, double y2)
{
    // Euclidean distance
    double u = (x1 - x2);
    double v = (y1 - y2);
    return Math.Sqrt(u * u + v * v);
}
}
}
```