OPTIMASI DETEKSI *MARKER* PADA NYARTOOLKIT MENGGUNAKAN METODE RANSAC

SKRIPSI

Laboratorium Komputasi Cerdas Dan Visualisasi

Untuk memenuhi sebagian persyaratan memperoleh gelar Sarjana Komputer

Disusun oleh :

ANISA AINI ARIFIN

NIM 0910683017

PROGRAM STUDI INFORMATIKA/ILMU KOMPUTER

PROGRAM TEKNOLOGI INFORMASI DAN ILMU KOMPUTER

UNIVERSITAS BRAWIJAYA

MALANG

2013

LEMBAR PERSETUJUAN

OPTIMASI DETEKSI *MARKER* PADA NYARTOOLKIT MENGGUNAKAN METODE RANSAC

SKRIPSI

Laboratorium Komputasi Cerdas Dan Visualisasi

Untuk memenuhi sebagian persyaratan memperoleh gelar Sarjana Komputer

Disusun Oleh:

ANISA AINI ARIFIN

NIM 0910683017

Menyetujui :

Dosen Pembimbing I	**Dosen Pembimbing II**
Budi Darma Setiawan S.Kom, M.Cs.	**Wibisono Sukmo Wardhono, ST, MT.**
NIP. 841015 06 11 0090	NIK. 820404 06 1 1 0091

LEMBAR PENGESAHAN

OPTIMASI DETEKSI *MARKER* PADA NYARTOOLKIT MENGGUNAKAN METODE RANSAC

SKRIPSI

Laboratorium Komputasi Cerdas Dan Visualisasi

Untuk memenuhi sebagian persyaratan memperoleh gelar Sarjana Komputer

Disusun oleh:

ANISA AINI ARIFIN

NIM 0910683017

Skripsi ini telah diuji dan dinyatakan lulus pada

tanggal 6 Desember 2013

Penguji I	**Penguji II**
Lailil Muflikhah, S.Kom., M.Sc.	**Novanto Yudistira, S.Kom., M.Sc.**
NIP. 19741113 200501 2 001	**NIK. 831110 16 1 1 0425**

Penguji III

Barlian Henryranu P., ST., MT.
NIK. 821024061 10254

Mengetahui
Ketua Program Studi Informatika/Ilmu Komputer

Drs. Marji, M.T.
NIP.19670801 199203 1 001

KATA PENGANTAR

Dengan nama Allah SWT Yang Maha Pengasih dan Penyayang. Segala puji bagi Allah SWT karena atas rahmat dan hidayahNya-lah penulis dapat menyelesaikan Tugas Akhir yag berjudul "Optimasi Deteksi *Marker* Pada NyARToolKit Menggunakan Metode RANSAC". Shalawat serta salam atas junjungan besar kita Nabi Muhammad S.A.W. beserta keluarga dan para sahabat sekalian. Tugas Akhir ini disusun untuk memenuhi sebagian persyaratan memperoleh gelar Sarjana Komputer di Program Studi Informatika/Ilmu Komputer Program Teknologi Informasi dan Ilmu Komputer (PTIIK) Universitas Brawijaya Malang.

Saya menyadari bahwa Tugas Akhir ini dapat terselesaikan atas bantuan, petunjuk, dan bimbingan dari berbagai pihak yang telah membantu proses penyelesaiannya. Melalui kesempatan ini, penulis ingin menyampaikan rasa hormat dan terima kasih penulis yang sebesar-besarnya kepada semua pihak yang telah memberikan bantuan bantuan baik lahir maupun batin selama penulisan tugas akhir ini. Oleh karena itu, pada kesempatan ini penulis ingin menyampaikan rasa hormat dan terima kasih penulis kepada :

1. Kedua Orang Tua penulis, adik tersayang, beserta seluruh keluarga besar atas segala nasihat, perhatian, dan kesabarannya dalam membesarkan dan mendidik penulis, serta yang senantiasa tiada henti-hentinya memberikan do'a demi terselesaikannya tugas akhir ini.
2. Bapak Budi Darma Setiawan S.Kom, M.Cs. dan Bapak Wibisono Sukmo Wardhono, ST, MT. selaku dosen pembimbing tugas akhir penulis yang telah memberikan bimbingan dan arahan untuk kesempurnaan penulisan Tugas Akhir.
3. Bapak Eriq Muhammad A. J., Mas Aninditya N. dan Mas Franciscus P. atas bantuan penggunaan peralatan dan peminjaman Laboraturium Game dan Laboraturium Komputer Dasar selama penulis mengerjakan Tugas Akhir.
4. Seluruh Dosen dan Karyawan Informatika/Ilmu Komputer PTIIK Universitas Brawijaya atas kesediaan membagi ilmunya kepada penulis.
5. Mas Adam Hendra B., Mas Febri Abdullah, Hanifa Vidya R., Pricillia Carolina K., Mas Kurnia Prima P., Ika Kusumaning P., Mas Ari Hernawan, Mas Yogi

Kurniawan, dan Mas Yohannes Kinskij atas kesediaan waktunya untuk membantu penulis pengerjaan aplikasi dan laporan Tugas Akhir.

6. Sahabat-sahabat GMT09; Austin Buya Oryza, Mamluatul Hani'ah, Nina Amalia Dewi, Hendro Pramudyo Saputro, Rizal Setya Perdana, Meidi Dian Laksana Putra, dan Putu Arya Kurnia Y., Ikram Baharsyah, Yugo Yudansha L., dan Danial M. Zaki atas dukungan dan semangat yang diberikan.
7. Teman-teman seperjuangan; Milani W., Mbak Sufia Adha P., Mbak Prima Arfianda P., Astri Tika P., Deppy Rangga M., Aulia Meitika K., Ervin Yohanes, Arianty A., dan Fransischa Tika S. atas dukungan dan semangat yang diberikan.
8. Sahabat-sahabatku Angkatan 2009 Teknik Informatika Universitas Brawijaya, terimakasih atas segala bantuan yang diberikan selama menempuh studi di Informatika/Ilmu Komputer PTIIK Universitas Brawijaya.
9. Martin Hirzer, Ryo Iizuka, Biben Nurbani Hasan, Muhamad Wirabuana, dan Muhammad Akbar Sidiq atas kesediaannya berbagi ilmu dan memberikan saran demi kelancaran penulis dalam menyelesaikan aplikasi Tugas Akhir ini.
10. Sahabat-sahabatku OP SMAN 3 Malang atas doa, dukungan dan semangat yang selalu diberikan selama penulis menempuh studi di Informatika/Ilmu Komputer PTIIK Universitas Brawijaya.
11. Semua pihak yang tidak dapat penulis sebutkan satu per satu yang terlibat baik secara langsung maupun yang tidak langsung demi terselesaikan-nya skripsi ini.

Hanya doa yang bisa penulis berikan semoga Allah SWT memberikan pahala serta balasan kebaikan yang berlipat. Penulis menyadari bahwa tugas akhir ini masih banyak kekurangan dan masih jauh dari sempurna. Untuk itu, saran dan kritik yang membangun sangat penulis harapkan.Semoga Tugas Akhir ini dapat memberikan manfaat bagi pembaca sekaligus dapat menjadi bahan acuan untuk penelitian selanjutnya.

Malang, Desember 2013

Penulis

ABSTRAK

Anisa Aini Arifin. 2013.: Optimasi Deteksi *Marker* Pada NyARToolKit Menggunakan Metode RANSAC. Skripsi Program Studi Informatika/Ilmu Komputer, Program Teknologi Informasi dan Ilmu Komputer, Universitas Brawijaya. Dosen Pembimbing: Budi Darma Setiawan, S.Kom., M.Cs., Wibisono Sukmo Wardhono, S.T., M.T.

Pembacaan *marker* pada aplikasi berbasis *Augmented reality* (AR) menggunakan pustaka NyARToolKit 4.0.3 masih kurang optimal. Proses deteksi *marker* pada pustaka tersebut dilakukan dengan menemukan empat garis lurus dari *marker* yang saling berhubungan, sehingga aplikasi tidak dapat mendeteksi *marker* apabila *marker* tidak ideal. Untuk mengatasi kondisi tersebut, dibutuhkan metode untuk mengoptimalkan kinerja aplikasi AR dalam membaca *marker* yang tidak ideal seperti metode RANSAC. *Random Sample Consensus* (RANSAC) adalah metode iterasi untuk memperkirakan parameter dari model matematika dari sekumpulan data yang terdiri atas *inliers* dan *outliers*. Ketidakidealan *marker* yang digunakan hanya melingkupi *marker* dengan tinta luntur, *marker* lecek, dan *marker* robek. Langkah pertama adalah menemukan *pixel* tepian, kemudian menghubungkan tepian tersebut menjadi segmen-segmen kecil menggunakan metode RANSAC. Setelah itu, menggabungkan segmen menjadi garis dan menemukan semua titik potong dari semua garis yang saling tegak lurus. Langkah terakhir, menemukan dua pasang titik potong dengan jarak terjauh, maka akan didapatkan empat garis terluar *marker* yang merupakan langkah penting dalam mendeteksi *marker*. Pada penelitian ini terdapat tiga variabel yang diujikan, yaitu kondisi maker, posisi penangkap citra, dan kemiringan *marker*. Keseluruhan hasil pengujian performa ini menunjukkan bahwa penambahan metode RANSAC pada aplikasi berbasis AR dapat memperbaiki dan meningkatkan performa aplikasi dalam mendeteksi *marker* yang tidak ideal secara memuaskan. Performa aplikasi rata-rata meningkat sebesar 50% dalam mendeteksi sepuluh kondisi *marker*, 40% dalam mendeteksi *marker* dengan tiga posisi penangkap citra, dan 38% dalam mendeteksi *marker* dengan lima kemiringan *marker*.

Kata Kunci: *Augmented reality*, Deteksi *Marker*, RANSAC

ABSTRACT

Anisa Aini Arifin. 2013.: ***Marker Detection Optimization for*** **NyARToolKit** ***Using*** **RANSAC.** ***Thesis Informatics/Computer Science Program, Informatic Technology and Computer Science Program, University of Brawijaya. Advisor*****: Budi Darma Setiawan, S.Kom., M.Cs., Wibisono Sukmo Wardhono, S.T., M.T.**

Marker detection in application based on augmented reality (AR) *using* NyARToolKit 4.0.3 *libraries is still less optimal. Marker detection in this library is done by finding four straight connected lines of the marker, that makes the application cannot detect non ideal marker. To overcome this condition, we need a method to optimize* AR *application performance in detecting non ideal marker such as* RANSAC. *Random Sample Consensus* (RANSAC) *is an iterative method to estimate parameters of a mathematical model from a set of data that consist of inliers and outliers. Kind of non ideal marker that used in this research cover only marker with faded marker, crumpled marker, and torn marker. First step is finding edges pixel, then linking those edges into small segment using* RANSAC. *After that, merging the segments into lines and finding all points of intersection of the perpendicular lines. Finally, find two pairs of corner point that have the longest distance and then get the outer line of the marker which is the important step of marker detection. In this research, three variabels are tested such as marker conditions, position of image capture, and the slope of marker. The entire performance test results showed that the addition of* RANSAC *into* AR *application can repair and improve performance of application to detect non ideal marker satisfactorily. Average application performance increase 50% while detecting ten marker conditions, increase 40% while detecting three positions of image capture, and increase 38% while detecting five slopes of marker.*

Keywords*: Augmented reality, Marker Detection,* RANSAC

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I
PENDAHULUAN

1.1 Latar Belakang

Augmented reality (AR) merupakan mekanisme penggabungan objek maya dua dimensi maupun tiga dimensi ke dalam sebuah lingkungan nyata tiga dimensi kemudian memproyeksikan obyek maya tersebut dalam waktu-nyata. Pada teknologi AR, pengguna dapat melihat lingkungan nyata yang ada di sekelilingnya dengan penambahan obyek maya yang dihasilkan oleh komputer [ART-97].

Kebanyakan aplikasi yang berbasis AR masih menggunakan marker fisik yang dicetak di atas kertas. Prinsip kerja AR secara umum adalah kombinasi dari komputer yang terhubung dengan kamera, *marker*, dan aplikasi pendukungnya. Keberhasilan aplikasi mendeteksi letak dan posisi *marker* mempunyai peran yang sangat penting dan mempengaruhi inti dari aplikasi.

Penggunaan *marker* pada pengguna tingkat akhir rentan terhadap kerusakan dikarenakan kesalahan dalam penyimpanan. Dengan demikian dibutuhkan sebuah aplikasi AR yang dapat mendeteksi *marker* yang rusak atau tidak ideal. Sehingga pengguna tetap dapat memanfaatkan *marker* yang tidak ideal tersebut.

Saat ini mekanisme pembacaan *marker* pada aplikasi berbasis AR menggunakan pustaka NyARToolKit 4.0.3. Pustaka ini memiliki API yang sudah berbentuk Class pustaka, mendukung format gambar yang banyak, proses *fitting* yang lebih cepat daripada pustaka lainya dan memiliki sistem *tracking* yang bagus.

Proses deteksi *marker* pada pustaka NyARToolKit 4.0.3 dilakukan dengan cara menemukan empat garis lurus dari *marker* yang saling berhubungan, sehingga kondisi ini membuat sistem tidak berjalan dengan baik apabila *marker* tidak pada keadaan yang ideal [HUD-12]. Selain *marker* yang tidak ideal, kegagalan pendeteksian *marker* juga dipengaruhi oleh kondisi lingkungan dan kualitas perangkat yang tidak ideal.

Pada tahun 2008, Martin Hirzer merilis hasil penelitiannya yang berjudul, *Marker Detection for Augmented reality Applications*. Penelitian tersebut membahas konsep aplikasi AR yang dapat membaca *marker* dalam kondisi lingkungan yang tidak ideal (*marker* tertutup sebagian dan kondisi pencahayaan yang tidak ideal). Aplikasi tersebut dikembangkan dengan pendekatan berbasis menghubungkan tepian

yang saling berhubungan. Setiap *pixel* tepian dicari menggunakan pendeteksi tepi yang dihubungkan dengan menggunakan metode RANSAC [MHZ-08].

Random Sample Consensus (RANSAC) adalah metode iterasi untuk memperkirakan parameter dari model matematika dari sekumpulan data yang terdiri atas *inliers*, yaitu data yang distribusinya dapat dijelaskan oleh beberapa set parameter model, dan *outliers* yang merupakan data yang tidak sesuai model. Algoritma ini memberikan hasil yang optimal dengan meningkatkan peluang sebanyak jumlah iterasi yang dilakukan [RFD-08].

Penelitian ini menekankan pada penerapan metode RANSAC dalam mendeteksi *marker* tidak ideal dengan menggunakan pendekatan menghubungkan tepian yang saling berhubungan. Langkah pertama adalah menemukan *pixel* tepian, kemudian menghubungkan tepian tersebut menjadi segmen-segmen kecil menggunakan metode RANSAC. Setelah itu, dilakukan penggabungan segmen menjadi garis dan menemukan semua titik potong dari semua garis yang saling tegak lurus. Langkah terakhir, menemukan dua pasang titik potong dengan jarak terjauh, maka akan didapatkan empat garis terluar *marker* yang merupakan langkah penting dalam mendeteksi *marker*. Dengan melakukan deteksi tepi terlebih dahulu sebelum mendeteksi garis dari *marker*, maka diharapkan aplikasi ini akan lebih optimal dalam mendeteksi *marker* yang tidak ideal.

1.2 Rumusan Masalah

Berdasarkan uraian latar belakang di atas, maka dapat dirumuskan permasalahan pada tugas akhir ini yaitu:

1. Bagaimana merancang peningkatan kinerja AR terhadap pengenalan *marker* tidak ideal yang diberikan dengan metode RANSAC?
2. Bagaimana mengimplementasikan algoritma metode RANSAC pada pustaka NyARToolKit 4.0.3?
3. Bagaimana pengaruh metode RANSAC terhadap peningkatan performa pendeteksian *marker*?

1.3 Tujuan

Tujuan penulisan tugas akhir ini adalah merancang peningkatan kinerja AR terhadap *marker* yang tidak ideal yang diberikan dengan metode RANSAC,

mengimplementasikan algoritma metode RANSAC pada pustaka NyARToolKit 4.0.3, dan mengetahui pengaruh metode RANSAC terhadap peningkatan performa *marker*.

1.4 Batasan Masalah

Agar permasalahan yang dirumuskan dapat lebih fokus, maka penelitian tugas akhir ini dibatasi dalam hal:

1. Materi yang diteliti hanya difokuskan pada tahap pendeteksian area segi empat pada *marker*.
2. Ketidakidealan *marker* hanya melingkupi kerusakan karena tinta pada *marker*, rusak terkena air (tinta luntur), kertas *marker* tidak licin (lecek), dan kertas *marker* hilang sebagian (robek).
3. Pustaka AR yang dipakai dalam penelitian ini adalah NyARToolKitCS 4.0.3
4. Nilai *frame rate* yang digunakan pada pengujian sebesar 30 fps.
5. Dalam pengujian hanya menggunakan satu pola *marker* persegi hitam-putih berukuran 3 inchi x 3 inchi (1 inchi = 2.54 cm). *Marker* dicetak ke dalam kertas 70 gram berukuran 21 cm x 14.85 cm.
6. Pada saat melakukan pendeteksian dan pengujian hanya terdapat satu buah *marker* dengan latar belakang putih.

1.5 Manfaat

Penelitian ini diharapkan dapat bermanfaat untuk berbagai pihak. Manfaat dari penelitian ini adalah sebagai berikut:

1. Umum

   Memberikan salah satu solusi dalam memecahkan permasalahan deteksi *marker* yang kurang optimal menghadapi ketidakidealan *marker*.
2. Khusus
   a. Bagi penulis:
      - Mengaplikasikan ilmu AR, Pengolahan Citra Digital, dan *Computer Vision* yang didapat selama mengikuti perkuliahan di Teknik Informatika Universitas Brawijaya.
      - Mendapatkan pemahaman tentang perancangan dan pengembangan aplikasi AR.

b. Bagi pembaca:

- Mendapat wawasan tentang pengembangan aplikasi AR.
- Mendapat wawasan mengenai penerapan metode RANSAC untuk aplikasi AR.

1.6 Sistematika Penulisan

Sistematika pembahasan ditunjukkan untuk memberikan gambaran dan uraian dari penulisan skripsi ini. Secara garis besar yang meliputi beberapa bab, sebagai berikut :

Bab I Pendahuluan

Menguraikan mengenai latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, dan sistematika pembahasan, serta jadwal penelitian.

Bab II Dasar Teori

Menguraikan tentang dasar teori dan referensi yang mendasari pengembangan perangkat lunak AR, NyARToolKit, *Grayscale*, *Thresholding*, Deteksi Tepi, Deteksi *Marker*, dan metode RANSAC.

Bab III Metodologi dan Perancangan

Menguraikan tentang metode dan langkah kerja yang dilakukan dalam penulisan tugas akhir yang terdiri atas studi literatur, perancangan perangkat lunak, implementasi perangkat lunak, pengujian dan analisis, pengambilan kesimpulan dan saran serta membahas perancangan aplikasi yang dibuat.

Bab IV Implementasi

Membahas implementasi aplikasi yang dibangun, meliputi pembuatan aplikasi.

Bab V Pengujian dan Analisis

Memuat analisis hasil perancangan aplikasi, hasil analisis output aplikasi, dan pembahasan terjadinya kegagalan (apabila terjadi kegagalan).

Bab VI Penutup

Memuat kesimpulan yang diperoleh dari pembuatan dan pengujian perangkat lunak yang dikembangkan dalam tugas akhir ini serta saran–saran untuk pengembangan lebih lanjut.

BAB II
KAJIAN PUSTAKA DAN DASAR TEORI

2.1 Kajian Pustaka

Pada penelitian yang dilakukan oleh Martin Hirzer tahun 2008 yang berjudul *Marker Detection for Augmented reality Applications*, membahas mengenai deteksi *marker* dengan menggunakan deteksi garis berdasarkan pendekatan berbasis menghubungkan tepian yang saling berhubungan. Penelitian tersebut dilakukan untuk dapat mendeteksi *marker* pada lingkungan yang tidak ideal (*marker* tertutup sebagian dan kondisi pencahayaan yang tidak ideal). Deteksi ini berawal dari menemukan *pixel* tepian kemudian dihubungkan menjadi segmen, setelah itu dikelompokkan sehingga menjadi segi empat (*quadrangle*). Terdapat tiga langkah utama dalam pengerjaan penelitian tersebut. Pertama segmen garis yang telah ditemukan dengan mendeteksi tepian yang diambil secara *sampling* dihubungkan menggunakan RANSAC. Kemudian segmen garis tersebut digabungkan untuk membentuk garis yang lebih panjang. Langkah selanjutnya, semua garis yang terdeteksi diperpanjang berdasarkan informasi gradiennya, sehingga mendapatkan garis dengan panjang maksimal. Setelah itu, garis-garis tersebut digabungkan menjadi segi empat [MHZ-08].

Pada penelitian "Optimasi Deteksi *Marker* Pada NyARToolKit Menggunakan Metode RANSAC" akan dibahas penerapan metode RANSAC dalam mendeteksi *marker* tidak ideal dengan menggunakan pendekatan menghubungkan tepian yang saling berhubungan. Dengan melakukan deteksi tepi terlebih dahulu sebelum mendeteksi garis dari *marker*, maka diharapkan aplikasi ini akan lebih optimal dalam mendeteksi *marker* yang tidak ideal.

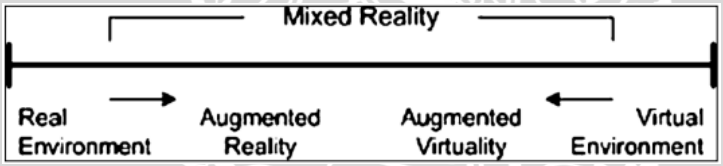
2.2 *Augmented reality* (AR)

Istilah AR muncul pertama kali pada tahun 1990 ketika Tom Caudell dan David Mizell mendesain suatu *head-set display* yang disebut HUDset (*Heads-Up, see through, head-mounted Display*). Para peneliti mengenai AR, salah satunya adalah, Ronald T. Azuma mendefinisikan *augmented reality* sebagai penggabungan benda nyata dan objek maya di lingkungan nyata, berjalan secara interaktif dalam waktu-nyata, dan terdapat integrasi antar objek dalam tiga dimensi, yaitu objek maya

terintegrasi dalam dunia nyata [ART-97]. AR merupakan salah satu teknologi yang menggunakan teknik *computer vision* dalam menentukan kesesuaian antara citra dan dunia nyata.

Selain menambahkan objek maya dalam lingkungan nyata, AR juga berpotensi menghilangkan benda-benda yang sudah ada. Menambah sebuah lapisan gambar maya dimungkinkan untuk menghilangkan atau menyembunyikan lingkungan nyata dari pandangan pengguna. Misalnya, untuk menyembunyikan sebuah meja dalam lingkungan nyata, perlu digambarkan lapisan representasi tembok dan lantai kosong yang diletakkan di atas gambar meja nyata, sehingga menutupi meja nyata dari pandangan pengguna.

Paul Milgram and Fumio Kishino merumuskan kerangka kemungkinan penggabungan dan peleburan dunia nyata dan dunia maya yang disebut *Milgram's Reality-Virtuality Continuum* pada tahun 1994. Sisi yang paling kiri adalah lingkungan nyata yang hanya berisi benda nyata, dan sisi paling kanan adalah lingkungan maya yang berisi objek maya [MIL-94].



Gambar 2.1 *Milgram's Reality-Virtuality Continuum*
Sumber: [MIL-94]

Dalam Gambar 2.1 AR, yang lebih dekat ke sisi kiri, lingkungan bersifat nyata dan benda bersifat maya, sementara dalam *augmented virtuality*, yang lebih dekat ke sisi kanan, lingkungan bersifat maya dan benda bersifat nyata. AR dan *augmented virtuality* digabungkan menjadi *mixed reality*.

2.3 NyARToolKit

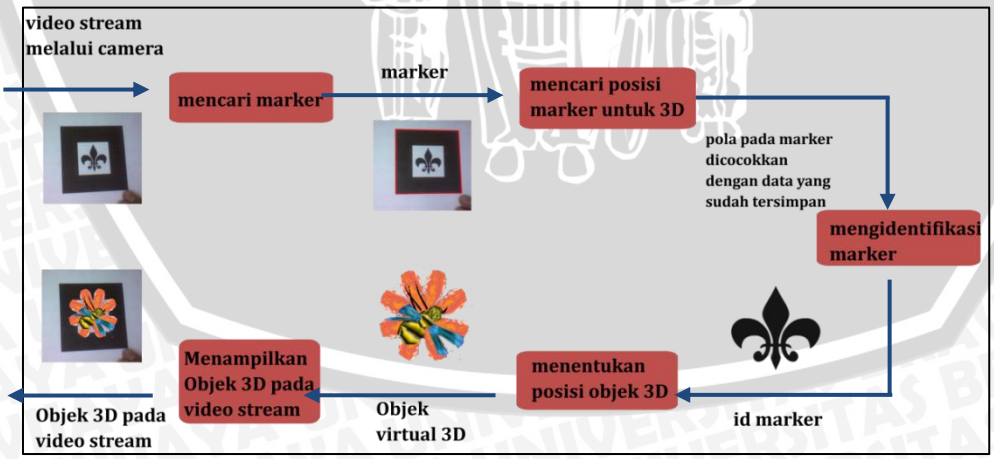
ARToolKit merupakan kumpulan pustaka AR yang didesain untuk dapat menghubungkan pengguna dengan aplikasi. ARToolKit didistribusikan dalam bentuk *source code* dan proses *compile* dilakukan pada sistem operasi dan *platform* yang spesifik. Sistem operasi membutuhkan *development environment* khusus dan *free environment* telah tersedia untuk berbagai platform.

ARToolKit dikembangkan oleh Dr. Hirokazu Kato dari Universitas Osaka Jepang dan Mark Billinghurst dari *Human Interface Technology Laboratory* (HIT

Lab). ARToolKit banyak digunakan untuk mengembangkan aplikasi AR. *Marker* pada ARToolkit merupakan gambar yang terdiri atas *border outline* dan *pattern image* seperti terlihat pada Gambar 2.2.

Gambar 2.2 Contoh *Marker*
Sumber: [HUD-12]

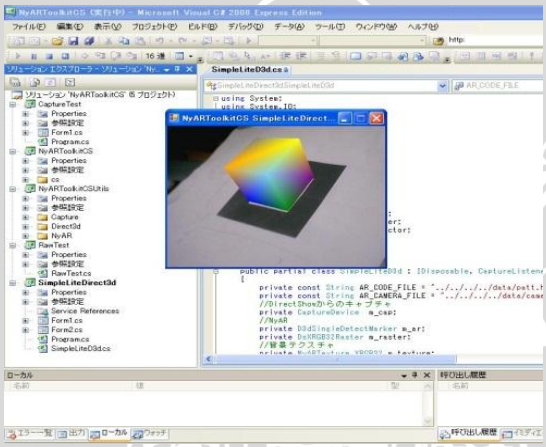
ARToolKit memberikan fungsi yang sama dan dapat beroperasi pada *multiple platform*, namun proses instalasinya berbeda tergantung pada sistem operasi yang digunakan. Sistem operasi dan *platform* harus memenuhi beberapa persyaratan sebelum dapat melakukan instalasi. Perangkat keras yang dimiliki harus memiliki kemampuan video *stream* dan memiliki alokasi CPU yang cukup untuk menangani video *processing* dan *display*. Selain itu juga terdapat beberapa *basic software dependency* untuk dapat menghindari kesalahan pada *compiler* dan *linker*. Paket utama ARToolKit bersifat *cross-platform* seperti OpenGL, GLUT dan lain-lain yang saling berelasi dengan video pustaka untuk komputer seperti DirectShow, V4L, dan Quicktime [HUD-12]. Komponen yang diperlukan ARToolkit adalah: development environment, DSVideoLib-0.0.8b-win32, GLUT, DirectX Runtime, Video input device, dan OpenVRML-0.14.3-win32.



Gambar 2.3 Bagan proses kerja ARToolKit
Sumber: [HUD-12]

Secara umum prinsip kerja ARToolKit adalah sebagai berikut [HUD-12] (Gambar 2.3):

1. Kamera menangkap gambar dari dunia nyata secara *live* dan mengirimkannya ke komputer.
2. Perangkat lunak dalam komputer akan mencari semua yang berbentuk segi empat pada masing-masing *frame* video.
3. Jika semua segi empat telah ditemukan, komputer akan memproses secara matematis posisi relatif dari kamera ke kotak hitam.
4. Apabila posisi kamera diketahui, maka model objek 3D akan digambarkan pada posisi yang sama.
5. Model obyek 3D akan ditambahan pada dunia nyata.
6. Sehingga ketika pengguna melihat pada tampilan, akan tampak bahwa objek 3D akan terlihat menempel di atas *marker* persegi tersebut seperti muncul pada dunia nyata.



Gambar 2.4 Screenshot NyARToolKit For C#
Sumber: [NYA-12]

NyARToolKit merupakan *augmented reality* toolkit turunan dari ARToolKit yang dikembangkan oleh Nyatla, sebuah pengembang perangkat lunak dari Jepang. NyARToolkit telah mendukung berbagai *platform* seperti Java, C#, Android dan C/C++. NyARToolKit dilengkapi dengan seluruh kemampuan ARToolKit serta memiliki beberapa kelebihan yang menjadikan NyARToolKit sangat powerfull [NYA-12].

Beberapa Fitur NyARToolkit:

a. API dari NyARToolKit sudah berbentuk Class pustaka.
b. Mendukung format gambar yang banyak.
c. Proses fitting yang lebih cepat daripada ARToolkit biasa.
d. Fungsi pelabelan yang cepat dan mendukung pelabelan RLE.

e. NyARToolKit dapat mengenali pola ID *Marker*.

f. NyARToolkit memiliki sistem *tracking* yang bagus.

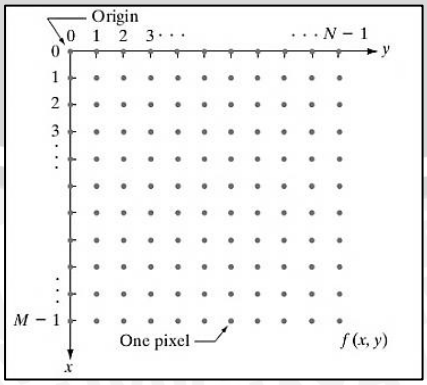
2.4 Computer Vision

Computer vision (sering disebut juga dengan *Machine Vision*) adalah penyimpulan (deduksi) otomatis akan struktur atau properti dari dunia tiga dimensi dari satu atau lebih citra dua dimensi dunia tersebut dan pengenalan objek-objek dengan bantuan properti-properti ini, atau secara singkat yaitu proses mengenali objek tertentu dari suatu citra.

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

Computer vision merupakan proses otomatis yang mengintegrasikan sejumlah besar proses untuk persepsi visual, seperti akuisisi citra, pengolahan citra, klasifikasi, pengenalan (*recognition*), dan membuat keputusan. *Computer vision* terdiri atas teknik-teknik untuk mengestimasi ciri-ciri objek di dalam citra, pengukuran ciri yang berkaitan dengan geometri objek, dan menginterpretasi informasi geometri tersebut. Pada bagian awal *computer vision* memerlukan beberapa teknik pengolahan citra digital. Khususnya pada bagian *preprocessing* seperti *grayscale*, *thresholding*, dan deteksi tepi.

2.5 Pengolahan Citra Digital



Gambar 2.5 Koordinat citra digital
Sumber : [GWS-07]

Secara umum, pengolahan citra digital menunjuk pada pemrosesan citra dua dimensi yang menggunakan media komputer. Dalam konteks yang lebih luas, pengolahan citra digital mengacu pada pemrosesan setiap data dua dimensi. Citra digital merupakan sebuah larik (*array*) yang berisi nilai-nilai real maupun kompleks yang direpresentasikan dengan deretan bit tertentu. Gambar 2.1 menunjukkan posisi koordinat citra digital.

Suatu citra dapat didefinisikan sebagai fungsi $f(x,y)$ berukuran M baris dan N kolom, dengan x dan y adalah koordinat spasial, dan amplitudo f di titik koordinat (x,y) dinamakan intensitas atau tingkat keabuan dari citra pada titik tersebut. Apabila nilai x, y dan nilai amplitudo f secara keseluruhan berhingga (*finite*) dan bernilai diskrit maka dapat dikatakan bahwa citra tersebut adalah citra digital [GWS-07].

Nilai pada suatu irisan antara baris dan kolom (pada posisi x,y) disebut dengan *picture elements*, *image elements*, *pels*, atau *pixels*. Citra digital dapat ditulis kedalam bentuk matrik berikut ini:

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) \dots \dots \dots f(0,N-1) \\ f(1,0) & f(1,1) \dots \dots \dots f(1,N-1) \\ . & . \quad . \\ . & . \quad . \\ . & . \quad . \\ f(M-1,0) & f(M-1,1) \dots \dots \dots f(M-1,N-1) \end{bmatrix} \quad \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots (2.1)$$

Meskipun sebuah citra memiliki banyak informasi, namun seringkali citra yang kita miliki mengalami penurunan mutu (*degradasi*), misalnya mengandung cacat atau derau (*noise*), warnanya terlalu kontras, kurang tajam, kabur (*blurring*), dan sebagainya. Tentu saja citra semacam ini menjadi lebih sulit diinterpretasi karena informasi yang disampaikan oleh citra tersebut menjadi berkurang. Agar citra yang mengalami gangguan mudah diinterpretasi (baik oleh manusia maupun mesin), maka citra tersebut perlu dimanipulasi menjadi citra lain yang kualitasnya lebih baik.

Pengolahan citra adalah pemrosesan citra menjadi citra yang kualitasnya lebih baik sehingga mudah diinterpretasi oleh manusia atau mesin (dalam hal ini komputer). Teknik-teknik pengolahan citra mentransformasikan citra menjadi citra lain. Jadi, masukannya adalah citra dan keluarannya juga citra, namun citra keluaran mempunyai kualitas lebih baik daripada citra masukan.

2.5.1 Grayscale

Citra *Grayscale*, yaitu citra yang nilai *pixel*-nya merepresentasikan derajat keabuan atau intensitas warna putih. Nilai intensitas paling rendah merepresentasikan warna hitam dan nilai intensitas paling tinggi merepresentasikan warna putih. Pada umumnya citra *Grayscale* memiliki kedalaman *pixel* 8 bit (256 derajat keabuan), tetapi ada juga citra *Grayscale* yang kedalaman *pixel*-nya bukan 8 bit, misalnya 16 bit untuk penggunaan yang memerlukan ketelitian tinggi.

Citra *Grayscale* merupakan citra satu kanal, dimana citra $f(x,y)$ merupakan fungsi tingkat keabuan dari hitam ke putih, x menyatakan variabel kolom atau posisi *pixel* di garis jelajah dan y menyatakan variabel kolom atau posisi *pixel* di garis jelajah. Intensitas f dari citra hitam putih pada titik (x,y) disebut derajat keabuan (*grey level*), yang dalam hal ini derajat keabuannya bergerak dari hitam ke putih. Derajat keabuan memiliki rentang nilai dari I_{min} sampai I_{max}, atau $I_{min} <$ f $< I_{max}$, selang (I_{min}, I_{max}) disebut skala keabuan.

Biasanya selang (I_{min}, I_{max}) sering digeser untuk alasan-alasan praktis menjadi selang $[0, L]$, yang dalam hal ini nilai intensitas 0 menyatakan hitam, nilai intensitas L menyatakan putih, sedangkan nilai intensitas antara 0 sampai L bergeser dari hitam ke putih. Sebagai contoh citra *grayscale* dengan 256 level artinya mempunyai skala keabuan dari 0 sampai 255 atau $[0,255]$, yang dalam hal ini intensitas 0 menyatakan hitam, intensitas 255 menyatakan putih, dan nilai antara 0 sampai 255 menyatakan warna keabuan yang terletak antara hitam dan putih.

Untuk mengubah citra berwarna, yang mempunyai nilai matriks masing-masing r, g, dan b, menjadi citra *grayscale* dengan nilai y maka konversi dapat dilakukan dengan mengambil rata-rata dari nilai r, g dan b sehingga dapat dituliskan menjadi:

$$y = (r + g + b)/3 \quad \text{.....(2.2)}$$

Atau,

$$\boldsymbol{y = 0,3\, r + 0,59\, g + 0,11\, b} \quad \text{.....(2.3)}$$

2.5.2 *Thresholding*

Thresholding citra adalah suatu metode yang digunakan untuk mengubah citra *grayscale* menjadi citra biner, citra digital yang hanya memiliki dua nilai *pixel* yaitu hitam dan putih, sehingga objek yang diinginkan terpisah dari latar belakangnya

[JKS-95]. *Thresholding* merupakan metode paling sederhana, dimana setiap objek atau *region* citra dibedakan berdasarkan penyerapan cahaya atau reflektifitas konstan pada permukaannya. Suatu nilai *threshold* (nilai konstan *brightness*) dapat ditentukan untuk membedakan objek dengan latar belakangnya.

Tujuan *thresholding* adalah memisahkan *pixel* yang mempunyai nilai keabuan (*gray value*) lebih tinggi dengan yang lebih rendah berdasarkan nilai *threshold* (t) yang sudah ditentukan. Misalnya *pixel* yang nilai keabuannya lebih tinggi dari nilai t diberi nilai biner 1 sedangkan *pixel* dengan nilai lebih rendah diberi nilai biner 0. Gambar 2.6 merupakan beberapa hasil proses pengolahan citra digital.

Gambar 2.6 (a) Citra berwarna, (b) Citra *Grayscale*, (c) Citra hasil *thresholding* dengan t=50, dan (d) Citra hasil *thresholding* dengan t=100

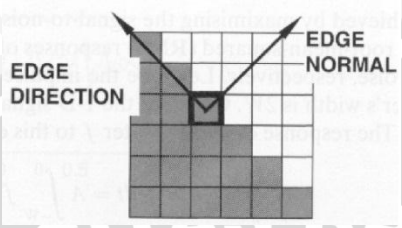
2.5.3 Deteksi Tepi

Deteksi tepi (*Edge Detection*) pada suatu citra adalah suatu proses yang menghasilkan tepi-tepi dari obyek-obyek citra, tujuannya adalah untuk menandai bagian yang menjadi detail citra dan untuk memperbaiki detail dari citra yang kabur, yang terjadi karena kesalahan atau adanya efek dari proses akuisisi citra. Suatu titik (*x*, *y*) dikatakan sebagai tepi (*edge*) dari suatu citra bila membatasi dua wilayah citra homogen yang memiliki tingkat kecerahan yang berbeda [PII-93].

Beberapa pendeteksi tepi cenderung menggunkan intensitas gradien dalam perhitungannya. Perubahan intensitas yang besar dalam jarak yang singkat dipandang sebagai fungsi yang memiliki kemiringan yang besar. Kemiringan ini dilakukan dengan menghitung turunan pertama (gradien).

Gradien menghitung bagaimana intensitas sebuah citra berubah. Sebagai sebuah vektor, gradien membawa dua informasi. Magnitud dari gradien memberi

tahu mengenai seberapa cepat citra berubah dan memberikan nilai perubahan sebuah *pixel* dengan *pixel* tetangganya (kekuatan dari sebuah tepi). Sedangkan orientasi dari gradien menjelaskan arah dari perubahan yang sangat besar, dimana arahnya selalu tegak lurus terhadap arah tepian tersebut (arah tepian dirotasi sebesar -90° dari arah gradien) (Gambar 2.7) [JKS-95].



Gambar 2.7 *Edge Descriptor*
Sumber: [COR-07]

Pada penelitian ini akan digunakan deteksi tepi menggunakan operator sobel. Operator ini merupakan operator diferensiasi diskrit dengan menggunakan pendekatan gradien dari suatu fungsi intensitas citra. Operator ini menggunakan dua buah kernel atau *mask* 3x3 yang diproses dengan citra asli (A), baik secara horisontal (gx) (Persamaan 2.4) maupun secara vertikal (gy) (Persamaan 2.5). Gradien merupakan sebuah vector sehingga memiliki magnitud (g) (persamaan 2.6) dan orientasi (θ) (Persamaan 2.7) yang dapat digunakan untuk menentukan tepian. Gambar 2.8 merupakan contoh hasil deteksi tepi.

$$gx_{i,j} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \otimes \begin{bmatrix} A_{i-1,j-1} & A_{i,j-1} & A_{+1i,j-1} \\ A_{i-1,j} & A_{i,j} & A_{i+1,j} \\ A_{i-1,j+1} & A_{i,j+1} & A_{i+1,j+1} \end{bmatrix} \quad \text{.....(2.4)}$$

$$gy_{i,j} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} A_{i-1,j-1} & A_{i,j-1} & A_{+1i,j-1} \\ A_{i-1,j} & A_{i,j} & A_{i+1,j} \\ A_{i-1,j+1} & A_{i,j+1} & A_{i+1,j+1} \end{bmatrix} \quad \text{.....(2.5)}$$

$$g = \sqrt{(gx^2 + gy^2)} \quad \text{.....(2.6)}$$

$$\theta = atan\left(\frac{gy}{gx}\right) \quad \text{.....(2.7)}$$

Keterangan:

- Operator ⊗ merupakan operator *masking*.

Gambar 2.8 Contoh hasil deteksi tepi

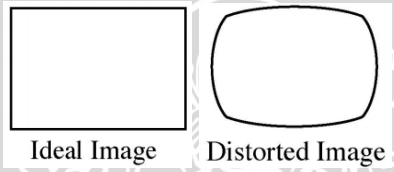
2.6 Deteksi *Marker*

Proses deteksi *marker* pada pustaka AR secara garis besar sebagai berikut [KOT-09]:

a. Mengambil Video dari Kamera

Langkah awal yang harus dilakukan adalah mendapatkan masukan video dari sebuah kamera, seperti yang ditunjukkan Gambar 2.9. Video yang di-*streaming* secara waktu-nyata ini akan diolah oleh sistem untuk dianalisis *frame* per *frame*.

Gambar 2.9 Mengambil citra dari kamera
Sumber: [KOT-09]



Gambar 2.10 Perbandingan antara citra yang ideal dengan citra yang disebabkan oleh faktor distorsi
Sumber: [HUP-12]

Sebelum kamera digunakan, kamera harus dikalibrasi terlebih dahulu. Kalibrasi kamera merupakan bagian yang sangat penting dalam proses pengambilan masukan video. Hal ini disebabkan oleh distorsi pada lensa kamera dimana setiap kamera berbeda karakteristiknya (Gambar 2.10). Tujuan dari kalibrasi kamera adalah untuk menghitung tingkat distorsi dari sebuah lensa kamera yang digunakan agar citra yang dihasilkan mendekati citra ideal.

b. Binerisasi citra masukan (*Thresholding*)

Langkah pertama pada aplikasi pengenalan citra yang terletak pada deteksi tepi adalah men-*threshold* sumber citra atau disebut juga binerisasi. *Thresholding* mengkonversi citra *grayscale* (Gambar 2.11) ke citra biner (Gambar 2.12) sehingga memudahkan untuk komputasi. Sebuah citra biner dibuat dengan mengubah *pixel* yang lebih cerah daripada nilai *threshold* ke suatu warna, dan *pixel* yang lebih gelap daripada nilai *threshold* ke suatu warna lainnya.

Nilai *threshold* berada pada angka 0-255 dan melalui beberapa percobaan *trial-and-error*, *threshold* pada aplikasi ini bernilai 150. Fungsi dari proses ini adalah

untuk membantu sistem agar dapat mengenali bentuk segi empat dan pola pada *marker* pada citra yang diterima. Nilai *threshold* dapat diubah dan disesuaikan dengan kondisi cahaya disekitar *marker* untuk tetap membuat *marker* terlihat sebagai segi empat. Karena ketika cahaya di sekitar *marker* berkurang ataupun berlebih pada saat proses *thresholding*, sistem tidak dapat mendeteksi *marker*.

Gambar 2.11 Citra *Grayscale*
Sumber: [KOT-09]

Gambar 2.12 Citra biner
Sumber: [KOT-09]

c. Deteksi Area Segi Empat

Langkah selanjutnya mencari area yang kemudian ditandai sebagai segi empat dari *marker* (*marker outline*). Proses ini merupakan fokus materi dalam penelitian ini. Setelah citra mengalami *preprocessing* (*thresholding*), sistem akan mencari bagian yang memiliki bentuk segi empat dan menandainya. Sistem juga akan menghilangkan area yang tidak berbentuk segi empat sehingga yang akan ditampilkan pada layar hanyalah area yang memiliki bentuk segi empat (Gambar 2.13).

Gambar 2.13 Mencari area segi empat (*Marker* Outline Detection)
Sumber: [KOT-09]

Pada pustaka NyARToolKit 4.0.3, sistem akan melakukan pencarian area berbentuk segi empat dari citra hasil *preprocessing* dengan menghitung nilai kesalahan (*error*) menggunakan rumus *Euclidean Distance* terhadap koordinat *pixel* (persamaan 2.8). Apabila suatu objek yang memiliki empat garis lurus yang saling berhubungan yang dianggap *marker* memiliki nilai error kurang dari *threshold* yang diberikan, maka objek tersebut akan ditentukan sebagai *marker*. Metode ini cocok untuk mencari *marker* yang ideal, namun kurang mampu mendeteksi *marker-marker* yang tidak ideal. Sehingga dibutuhkan aplikasi AR yang dapat mendeteksi *marker* tersebut.

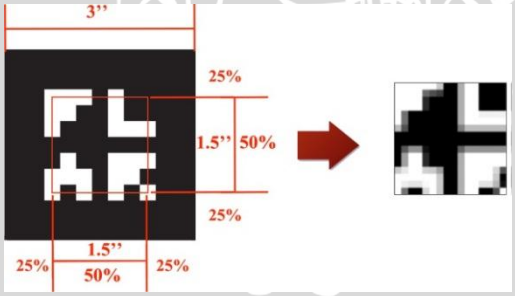
$$err = \frac{1}{4}\sum_{i=1,2,3,4}\{(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2\} \quad (2.8)$$

Untuk mengatasi masalah di atas, proses menemukan empat garis lurus yang saling berhubungan akan digantikan dengan metode pendeteksian garis berdasarkan data tepian hasil *preprocessing* yang akan dilakukan konsensus berdasar nilai magnitud dan orientasinya, kemudian dihubungkan menggunakan metode RANSAC.

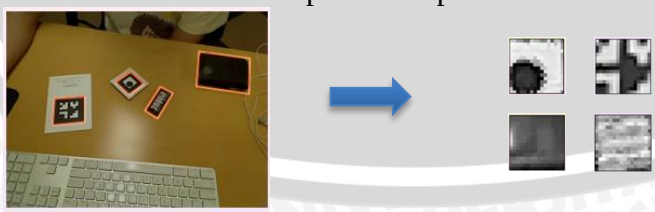
d. Pencocokan Pola

Setelah semua area segi empat ditandai, sistem akan menganalisis citra yang berada di dalam segi empat dan membandingkan polanya dengan sekumpulan pola yang telah ditentukan (pencocokan pola). Sistem mengekstrak pola di dalam segi empat menggunakan transformasi *homography* dan memberikan sebuah nilai '*confidence'* kepada setiap pola yang cocok. Jika kecocokannya di atas nilai yang telah ditentukan maka polanya dinyatakan cocok atau ditemukan. Spesifikasi pola *marker* (Gambar 2.14):

- Harus berupa persegi.
- Hanya 50% dari tengah area yang digunakan untuk proses pencocokan pola.
- Pola *marker* secara *default* -nya adalah 16 x 16 titik.
- Ukuran pola bisa lebih besar, tapi membutuhkan waktu yang lebih lama untuk diproses. Proses pencocokan pola *marker* dapat dilihat pada Gambar 2.15.



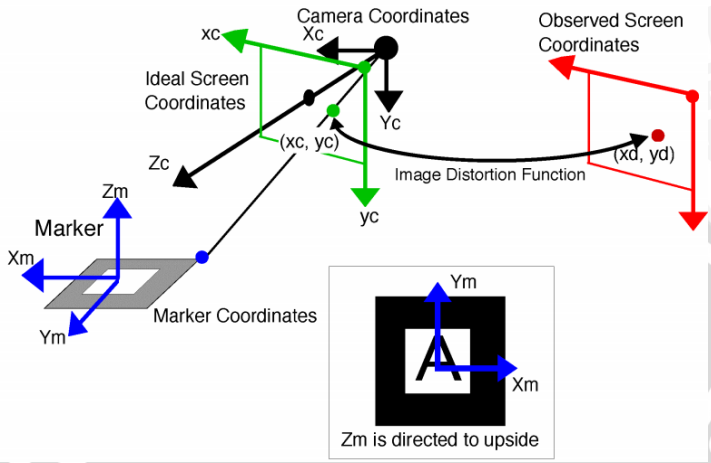
Gambar 2.14 Spesifikasi pola *marker*



Gambar 2.15 Pencocokan pola *marker*
Sumber: [KOT-09]

e. Menghitung Transformasi Matriks

Transformasi matriks dihitung dari titik-titik segi empat *marker* yang dideteksi. Matriks tersebut digunakan untuk proses *render* objek 3 dimensi. *Marker* dengan bentuk persegi digunakan sebagai basis koordinat dalam layar (Gambar 2.16).



Gambar 2.16 Hubungan antara koordinat *marker* dengan koordinat kamera
Sumber: [HUP-12]

f. Me-*render* Objek 3 dimensi

Sistem menggunakan transformasi matriks yang dikalkulasikan pada langkah sebelumnya dan menampilkan objek yang sesuai dengan sebuah pustaka 3 dimensi, seperti yang ditunjukkan pada Gambar 2.17. Sistem menyertakan kelas pendukung yang mengkonversikan transformasi matriks ke setiap kelas matriks pustaka internal 3 dimensi tersebut.

Gambar 2.17 Render objek 3 dimensi
Sumber: [KOT-09]

2.7 RANSAC

Pada sebuah eksperimen yang melibatkan puluhan, ratusan atau bahkan ribuan data, seringkali dihadapkan pada sejumlah data yang memiliki relasi linier. Dari data-data tersebut, akan menjelaskan relasi linier antar data dengan menurunkan sebuah persamaan linier melalui teknik regresi atau *least-square*. Apabila data-data tersebut memiliki relasi linier yang cukup sempurna, model persamaan yang diturunkan pun akan mengandung kesalahan yang cukup kecil. Namun bila data-data tersebut memiliki relasi linier yang tidak terlalu baik (dengan

berbagai macam noise atau *outliers*), model persamaan yang dihasilkan akan berubah dan bukan tidak mungkin memiliki tingkat akurasi yang cukup rendah [WSR-13].

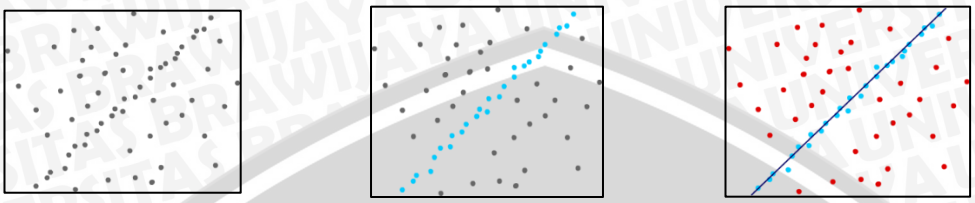
Pada aplikasi AR, proses pendeteksian *marker* menjadi suatu proses yang memegang peranan penting. Aplikasi tidak akan mencocokkan pola, menghitung transformasi matriks, maupun me-*render* objek virtual, sebelum *marker* terdeteksi. Pendeteksian *marker* diawali dengan mencari tepi dari *marker* tersebut. Sehingga, pendeteksi yang digunakan pada penelitian ini harus dapat menemukan tepi *marker* secara akurat begitu juga dengan perpotongannya, sudut dari *marker*, juga harus diketahui. Selain itu, karena deteksi *marker* ini digunakan pada aplikasi AR, yang berjalan dalam waktu-nyata, maka yang dibutuhkan adalah pendeteksi tepi yang sangat cepat.

Random Sample Consensus (RANSAC) pertama kali diperkenalkan oleh Fischler dan Bolles pada tahun 1981 sebagai metode iteratif untuk estimasi parameter dari model matematika dari serangkaian data yang diamati yang berisi *outliers*. Ini adalah algoritma untuk menemukan kelompok *inliers*, yaitu poin yang dapat dipasangkan ke dalam garis. Metode ini adalah algoritma non-deterministik, dalam pengertian bahwa ia hasil yang masuk akal hanya dengan menggunakan peluang kepastian tertentu, dimana besar peluang akan meningkat sebanyak jumlah iterasi yang dilakukan [RFD-08].

Secara garis besar, algoritma ini memiliki prosedur perhitungan sebagai berikut:

a. Asumsikan titik data sejumlah n, yakni $X = \{x_1, x_2, \ldots\ldots, x_n\}$, dimana sebuah model relasi linier dari data tersebut dibuat dengan menggunakan setidaknya m titik, dimana $m \leq n$. Untuk menghasilkan sebuah garis sempurna setidaknya dibutuhkan 2 titik.
b. Mulai *counter* iterasi $k = 1$
c. Pilih secara acak m buah data dari X, kemudian hitung sebuah model linier
d. Untuk sebuah nilai toleransi ε dari model yang dihasilkan, tentukan seberapa banyak elemen X yang berada dalam nilai toleransi tersebut. Jika jumlah dari elemen-elemen ini melebihi sebuah nilai ambang batas t, maka hitung kembali model menggunakan titik-titik konsensus, kemudian hentikan algoritma.

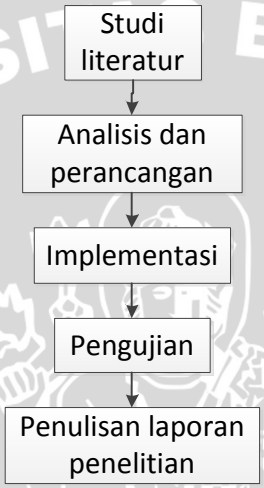
e. Naikan nilai counter menjadi $k = k + 1$. Jika$k < K$, untuk sebuah nilai K yang ditentukan di awal perhitungan, maka ulangi langkah c. Apabila terjadi selainnya, maka terima hasil perhitungan model linier dengan jumlah titik-titik konsensus terbesar. Ilustrasi metode RANSAC dapat dilihat pada Gambar 2.18.



Gambar 2.18 Ilustrasi RANSAC

BAB III
METODOLOGI DAN PERANCANGAN

Pada bab ini akan dibahas mengenai langkah-langkah yang akan dilakukan dalam pengerjaan tugas akhir, yaitu studi literatur, penyusunan dasar teori, analisis dan perancangan, implementasi dan pengujian dari aplikasi perangkat lunak yang akan dibuat, hingga penulisan laporan. Kesimpulan dan saran disertakan sebagai catatan atas aplikasi dan kemungkinan arah pengembangan aplikasi selanjutnya. Gambar 3.1 merupakan blok diagram runtutan pengerjaan penelitian ini:



Gambar 3.1 Blok Diagram Runtutan Pengerjaan Penelitian

3.1 Studi Literatur

Studi literatur mempelajari mengenai penjelasan dasar teori yang digunakan untuk menunjang penulisan skripsi. Teori-teori pendukung tersebut diperoleh dari buku, jurnal, e-book, dan dokumentasi projek. Semakin banyak studi literatur yang dilakukan akan mendukung penulis dalam membangun dasar teori dan aplikasi sistem. Penyusunan dasar teori dilakukan setelah mendapatkan referensi yang tepat untuk mendukung penulisan penelitian ini. Teori-teori pendukung tersebut meliputi:

a. *Augmented reality* (AR)
b. NyARToolKit
c. *Grayscale*
d. *Thresholding*
e. Deteksi Tepi
f. Deteksi *marker*
g. *Random Sample Consensus* (RANSAC)

3.2 Analisis Kebutuhan

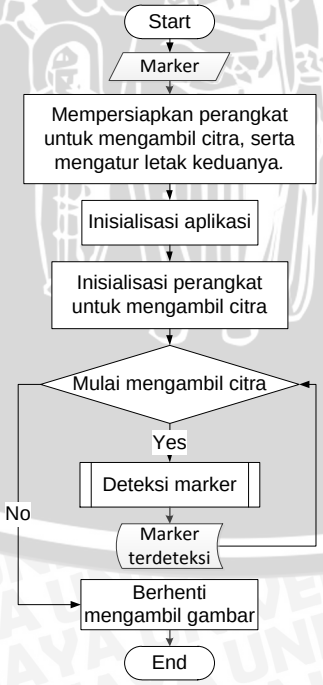
Analisis kebutuhan bertujuan untuk menganalisis dan mendapatkan semua kebutuhan yang diperlukan. Analisis kebutuhan disesuaikan dengan lokasi dan variabel penelitian, menentukan kebutuhan data yang akan digunakan, dan mempersiapkan alat dan bahan penelitian.

Seluruh proses yang dilakukan dibuat menggunakan perangkat lunak untuk memecahkan masalah matematis dimana komputer yang digunakan memiliki spesifikasi sebagai berikut :

a. Performa : Intel® Core™ i5-2320 CPU @ 3.00GHz (4 CPUs), ~3.0GHz
b. Sistem operasi : Windows® 7 Ultimate 32-bit (6.1, Build 7600)
c. VGA card : AMD Radeon HD 6450
d. Kamera : Logitech® HD Webcam C525
e. Aplikasi dibangun dengan Microsoft® Visual Studio® 2008

3.3 Perancangan Sistem

Perancangan sistem dibangun berdasarkan hasil pengambilan data dan analisis kebutuhan yang telah dilakukan. Pada bagian ini akan dipaparkan mengenai langkah-langkah yang akan dijalankan untuk melakukan pendeteksian *marker*.

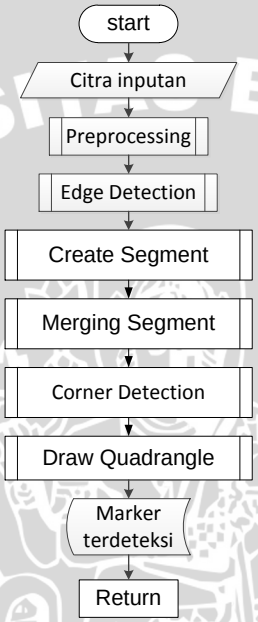


Gambar 3.2 Diagram Alir Sistem Utama

Sistem utama dimulai ketika *marker* diarahkan ke aplikasi yang sedang berjalan. Pertama-tama, pengguna mempersiapkan *marker* dan perangkat untuk

menangkap citra serta mengatur letak keduanya. Kemudian setelah aplikasi dijalankan, pengguna akan diperlihatkan kotak dialog yang berisi pemilihan perangkat untuk menangkap citra. Setelah memilih, pengguna akan diperlihatkan tampilan utama dan aplikasi akan mulai menangkap citra dan berusaha mendeteksi *marker*. Saat tampilan program ditutup maka proses pengambilan citra akan berhenti dan aplikasi akan berhenti. Diagram alir sistem utama dapat dilihat pada Gambar 3.2.

3.3.1 Deteksi *Marker*



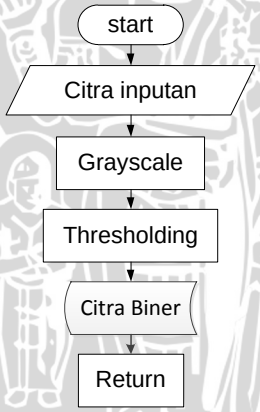
Gambar 3.3 Diagram Alir Proses Deteksi *Marker*

Dalam penelitian ini sebagian besar mengaplikasikan langkah-langkah pada penelitian yang sudah dilakukan oleh Martin Hizer, namun dilakukan beberapa modifikasi dalam prosesnya. Penelitian yang sudah dilakukan sebelumnya memiliki beberapa langkah dalam mendeteksi *marker* termasuk tiga langkah utama yang sudah disebutkan pada bab sebelumnya. Langkah pertama menemukan data tepian secara *sampling* kemudian menghubungkannya menggunakan metode RANSAC sehingga menjadi segmen-segmen kecil. Setelah itu menggabungkan segmen-segmen tersebut agar menjadi garis yang lebih panjang. Kemudian memperpanjang garis-garis tersebut berdasarkan orientasi gradien untuk mendapatkan panjang garis maksimal. Langkah selanjutnya, mencari titik sudut dari *marker* dengan memotongkan garis-garis yang ada dengan memperhatikan orientasi gradiennya. Ketika semua garis sudah ditemukan, maka ditemukanlah segi empat tersebut.

Namun, pada penelitian ini, modifikasi dilakukan setelah proses menggabungkan segmen. Proses ini mengaplikasikan metode RANSAC. Kemudian untuk proses memperpanjang segmen dan mencari titik sudut *marker* digantikan dengan menemukan 16 titik potong dari delapan garis yang terbentuk dari sebuah *marker*. Kemudian mencari dua pasang titik dengan jarak terjauh, maka akan terbentuk segi empat yang diinginkan. Secara singkat langkah-langkah deteksi *marker* pada penelitian ini dapat dilihat pada Gambar 3.3.

3.3.1.1 *Preprocessing*

Pada tahap *preprocessing*, citra masukan akan dijadikan citra biner dengan melakukan dua tahap. Pertama, citra masukan akan diproses menjadi citra *grayscale* dengan menggunakan proses yang sudah dijelaskan pada bab II. Kemudian citra *grayscale* tersebut akan di-*threshold* untuk menjadikannya citra biner. Nilai *threshold* yang digunakan adalah statis. Hal ini memungkinkan untuk membuat hasil biner tidak maksimal apabila tingkat kecerahan citra masukan awal terlalu tinggi. Diagram alir proses *preprocessing* dapat dilihat pada Gambar 3.4.



Gambar 3.4 Diagram Alir Proses *Preprocessing*

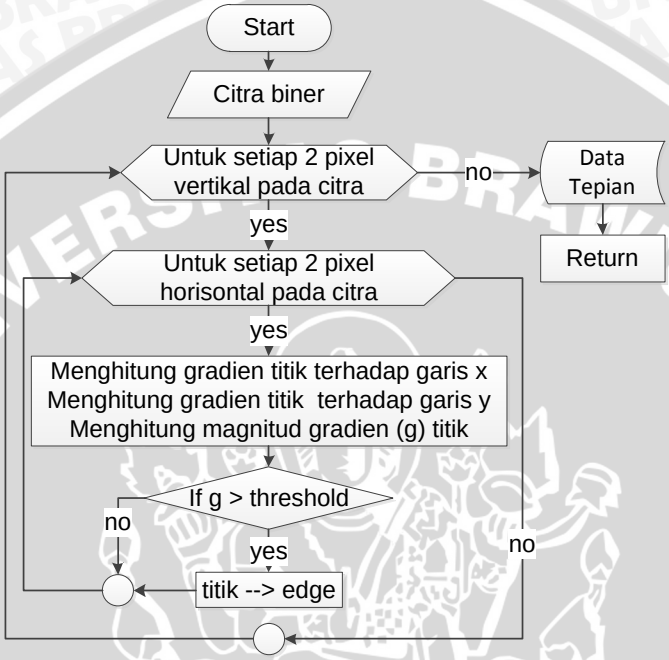
3.3.1.2 Deteksi Area Segi Empat

a. Deteksi Tepi

Untuk dapat membentuk garis dengan metode RANSAC diperlukan kumpulan titik. Sehingga pada tahap pertama dilakukan proses deteksi tepi terlebih dahulu. Untuk meningkatkan kecepatan pendeteksian garis, maka pada proses ini citra hasil *preprocessing* dilakukan pengambilan data secara *sampling*. Data citra diseleksi setiap dua *pixel* secara horisontal dan vertikal [CJZ-96]. Proses deteksi tepi pada

penelitian ini menggunakan metode Sobel dengan mempertimbangkan penelitian sebelumnya [MHZ-08] [KOT-12].

Proses untuk menentukan data tepian yaitu dengan mengambil nilai magnitud gradien masing-masing *pixel* untuk dilakukan seleksi terhadap *threshold* yang diberikan. Apabila sesuai, maka akan ditemukan *pixel-pixel* tepian. Diagram alir proses deteksi tepi dapat dilihat pada Gambar 3.5.



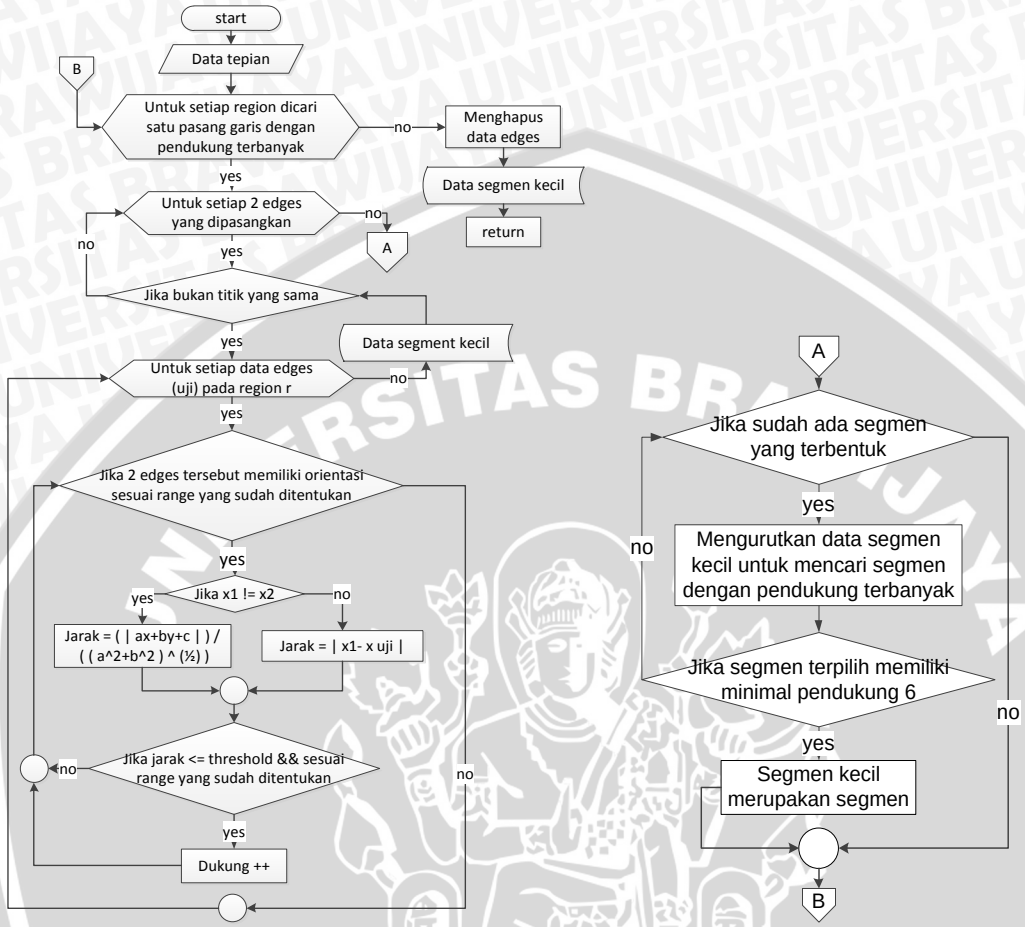
Gambar 3.5 Diagram Alir Proses Deteksi Tepi

b. Membuat Segmen

Pada proses ini menggunakan metode RANSAC untuk membuat segmen. Data yang digunakan merupakan semua data sampling tepian yang sudah ditemukan sebelumnya. Pertama, citra dibagi menjadi beberapa area yang lebih kecil atau region yang ukurannya sama besar yang diproses oleh RANSAC secara berurutan. Untuk menghipotesis sebuah segmen, RANSAC menggambil dua titik yang memiliki orientasi sama untuk dihitung jumlah konsensus atau jumlah pendukungnya.

Pada penelitian sebelumnya [MHZ-08] [KOT-12], RANSAC hanya menggunakan data yang jumlahnya terbatas pada perulangan dengan batas yang sudah ditentukan. Sehingga tidak semua data *sampling* digunakan. Dengan mempertimbangkan tingkat efisiensi dan efektivitas untuk mendapatkan dua titik dengan panjang maksimum dan pendukung maksimum, maka pada penelitian ini

semua data yang sudah ditemukan pada langkah sebelumnya digunakan secara keseluruhan. Diagram alir proses membuat segmen dapat dilihat pada Gambar 3.6.



Gambar 3.6 Diagram Alir Proses Membuat Segmen

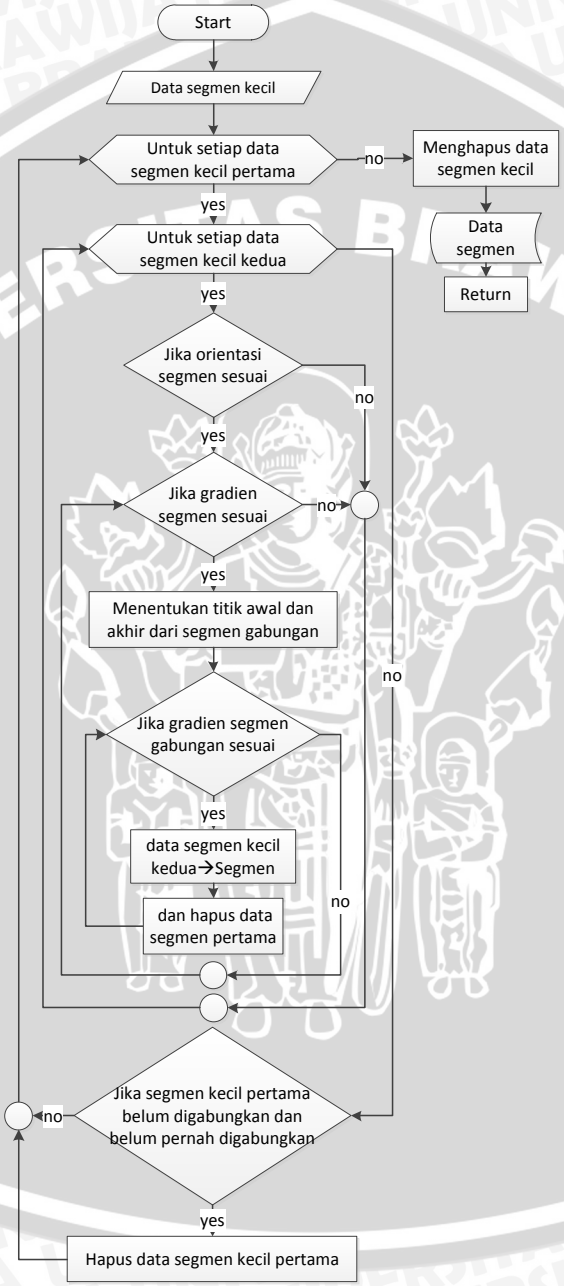
c. Menggabungkan Segmen

Data segmen dari proses sebelumnya, dilakukan penggabungan berdasarkan threshold orientasi masing-masing segmen kecil sehingga membentuk segmen yang lebih panjang. Syarat terjadinya penggabungan segmen adalah dua garis yang digabungkan dan hasil segmen gabungan harus memiliki gradien yang sesuai dengan threshold yang diberikan. Apabila sesuai maka akan dicari titik awal dan titik akhir dari kedua segmen yang digabungkan menggunakan rumus Manhattan Distance seperti berikut:

$$d_M = |x_1 - x_2| + |y_1 - y_2| \quad \text{.....(3.2)}$$

Kemudian mengecek kembali gradien dari segmen hasil penggabungan. Apabila sesuai dengan threshold yang diberikan, maka proses dilanjutkan dengan mengganti data segmen kecil kedua dengan data segmen terbaru dan menghapus data

segmen kecil pertama. Kemudian mengulang untuk mencari segmen kecil pertama yang baru. Apabila segmen ini tidak digabungkankan dan belum pernah digabungkan sebelumnya maka segmen ini akan dihapus dan mengulang untuk mencari segmen kecil yang lainnya. Diagram alir proses menggabungkan segmen dapat dilihat pada Gambar 3.7.

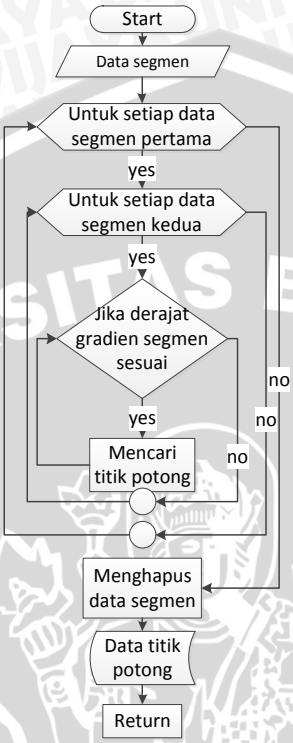


Gambar 3.7 Diagram Alir Proses Menggabungkan Segmen

d. Mendeteksi Titik Sudut

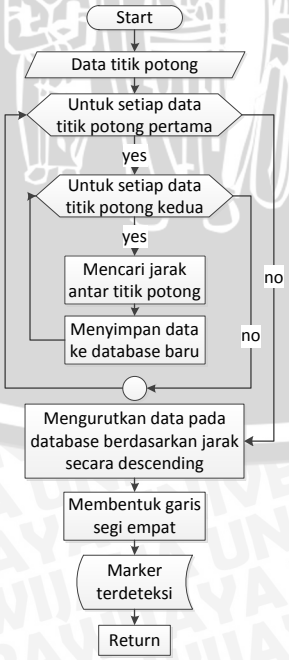
Proses ini akan memotongkan semua segmen yang terbentuk dari hasil proses sebelumnya. Apabila segmen yang terbentuk berjumlah delapan buah, maka titik

potong yang terjadi adalah 16 buah. Syarat berpotongannya dua segmen harus sesuai dengan *threshold* yang diberikan dan tidak boleh memotongkan dua garis yang sejajar. Diagram alir proses mendeteksi titik sudut dapat dilihat pada Gambar 3.8 berikut ini :



Gambar 3.8 Diagram Alir Proses Mendeteksi Titik Sudut

e. Membentuk Garis Segi Empat (Menemukan *Marker)*



Gambar 3. 9 Diagram Alir Proses Membentuk Garis Segi Empat

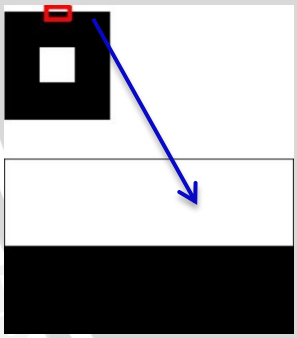
Proses terakhir dari rangkaian deteksi area segi empat adalah membentuk area tersebut. Proses ini akan mencari dua pasang titik potong dari proses sebelumnya yang paling panjang. Pada kondisi normal, jarak terpanjang dari sebuah segi empat adalah garis diagonalnya, sehingga area segi empat dapat ditemukan (menemukan *marker*). Pencarian jarak terpanjang antar titik sudut yang sudah ditemukan pada proses sebelumnya menggunakan rumus *Manhattan Distance* seperti pada Persamaan 3.2. Diagram alir proses membentuk area segi empat dapat dilihat pada Gambar 3. 9.

3.4 Perhitungan Manual

Pada bab ini akan menampilkan contoh perhitungan manual untuk sistem deteksi *marker* menggunakan metode RANSAC.

3.4.1 Proses Deteksi Tepi

Seperti yang sudah dijelaskan pada bab sebelumnya, proses ini menggunakan metode Sobel dalam mendeteksi tepian. Untuk menemukan *pixel* yang merupakan tepian maka digunakan magnitud *pixel* yang dibandingkan dengan *threshold* yang diberikan. Sebagai contoh, Gambar 3.10 merupakan contoh citra yang sudah dilakukan *preprocessing* dengan ukuran 6x6 *pixel*:



0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
255	255	255	255	255	255
255	255	255	255	255	255
255	255	255	255	255	255

Gambar 3.10 Contoh Citra Hasil *Preprocessing*

1. Mencari gradien terhadap sumbu x (gx) dengan menggunakan Persamaan 2.4. Berikut contoh perhitungan:

$$gx_{i,j} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \otimes \begin{bmatrix} A_{i-1,j-1} & A_{i,j-1} & A_{i+1,j-1} \\ A_{i-1,j} & A_{i,j} & A_{i+1,j} \\ A_{i-1,j+1} & A_{i,j+1} & A_{i+1,j+1} \end{bmatrix}$$

$$gx_{2,3} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \otimes \begin{bmatrix} A_{1,2} & A_{2,2} & A_{3,2} \\ A_{1,3} & A_{2,3} & A_{3,3} \\ A_{1,4} & A_{2,4} & A_{3,4} \end{bmatrix}$$

$$gx_{2,3} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \otimes \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 255 & 255 & 255 \end{bmatrix}$$

$$gx_{2,3} = ((-1) * 0 + (-2) * 0 + (-1) * 0 + 0 * 0 + 0 * 0 + 0 * 0 + 1 * 255 + 2 * 255 + 1 * 255)$$

$$gx_{2,3} = 1020$$

2. Mencari gradien terhadap sumbu y (gy) menggunakan Persamaan 2.5. Berikut perhitungannya:

$$gy_{i,j} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} A_{i-1,j-1} & A_{i,j-1} & A_{i+1,j-1} \\ A_{i-1,j} & A_{i,j} & A_{i+1,j} \\ A_{i-1,j+1} & A_{i,j+1} & A_{i+1,j+1} \end{bmatrix}$$

$$gy_{2,3} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} A_{1,2} & A_{2,2} & A_{3,2} \\ A_{1,3} & A_{2,3} & A_{3,3} \\ A_{1,4} & A_{2,4} & A_{3,4} \end{bmatrix}$$

$$gy_{2,3} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 255 & 255 & 255 \end{bmatrix}$$

$$gy_{2,3} = ((-1) * 0 + 0 * 0 + 1 * 0 + (-2) * 0 + 0 * 0 + 2 * 0 + (-1) * 255 + 0 * 255 + 1 * 255)$$

$$gy_{2,3} = 0$$

3. Mencari magnitud titik (g) menggunakan Persamaan 2.6, sedangkan untuk mencari orientasi titik (θ) menggunakan Persamaan 2.7. Berikut contoh perhitungan magnitud pada *pixel* (2, 3) :

$$g_{2,3} = \sqrt{(gx_{2,3}{}^2 + gy_{2,3}{}^2)}$$

$$g_{2,3} = \sqrt{(1020^2 + 0^2)}$$

$$g_{2,3} = 1020$$

4. Perhitungan untuk orientasi titik (θ) pada *pixel* (2, 3)

$$\theta_{2,3} = atan\left(\frac{gy_{2,3}}{gx_{2,3}}\right) \qquad \theta_{2,3} = 180$$

$$\theta_{2,3} = \text{atan}\left(\frac{0}{1020}\right)$$

Untuk *pixel* sebelum dan selanjutnya akan dilakukan perhitungan dengan langkah yang sama. Maka didapatkan:

Gradien terhadap sumbu x (gx) dan gradien terhadap sumbu y (gy):

gx:

*	*	*	*	*	*
*	0	0	0	0	*
*	1020	1020	1020	1020	*
*	1020	1020	1020	1020	*
*	0	0	0	0	*
*	*	*	*	*	*

gy:

*	*	*	*	*	*
*	0	0	0	0	*
*	0	0	0	0	*
*	0	0	0	0	*
*	0	0	0	0	*
*	*	*	*	*	*

Serta menghasilkan magnitud (g) dan orientasi (θ) pada masing-masing *pixel*:

g:

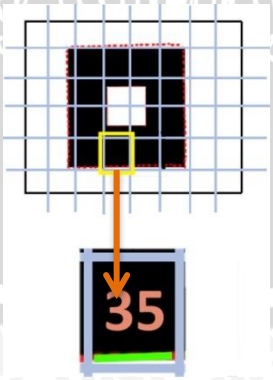
*	*	*	*	*	*
*	0	0	0	0	*
*	1020	1020	1020	1020	*
*	1020	1020	1020	1020	*
*	0	0	0	0	*
*	*	*	*	*	*

θ:

*	*	*	*	*	*
*	*	*	*	*	*
*	180	180	180	180	*
*	180	180	180	180	*
*	*	*	*	*	*
*	*	*	*	*	*

Orientasi *pixel* tepian, orientasi segmen garis, serta garis akhirnya nanti, bisa memiliki nilai mulai 0° sampai 360°, bergantung pada nilai intesitas citra. Sehingga *pixel* tepian yang berada pada tepi putih-hitam dan yang berada pada tepi hitam-putih yang memiliki arah yang sama, orientasinya berbeda sekitar 180°. Berdasarkan hal ini, nilai gradien dan orientasi *pixel* yang didapatkan dari proses pendeteksian tepi ini dapat digunakan untuk mencari nilai orientasi garis tertentu. Proses pencarian orientasi garis ini dilakukan dengan cara menarik garis *scanline* yang tegak lurus terhadap sebuah *pixel* acuan yang nilai orientasi *pixel*-nya telah didapatkan sebelumnya dari proses deteksi tepi tersebut [MHZ-08].

3.4.2 Proses Membuat Segmen



Gambar 3.11 Contoh Hasil Deteksi Tepi yang akan Dilakukan Pembuatan Segmen

Proses selanjutnya adalah menghubungkan semua data *sampling* dari proses sebelumnya dengan menggunakan metode RANSAC. Seperti yang sudah dijelaskan

pada bab sebelumnya, proses ini diawali dengan membagi citra menjadi beberapa region dengan ukuran yang sama. Selanjutnya mengambil data per region, kemudian mencari sepasang titik yang memiliki jarak maksimum dan jumlah pendukung maksimum sesuai *threshold* yang diberikan. Sebagai contoh pada Gambar 3.11, region 35 yang akan dilakukan pembuatan segmen.

Tabel 3.1 Contoh Tabel yang Berisi Tepian pada Region yang Sama

edges	x	y	t	reg
300	120	195	0	35
301	121	196	0	35
302	122	195	0	35
303	123	195	0	35

Dimisalkan jumlah data tepian pada region 35 adalah 4 buah. Kemudian dilakukan perhitungan jarak dan jumlah pendukung terhadap semua titik sesuai dengan model yang sudah ditentukan sebelumnya. Berikut contoh perhitungannya.

1. Memeriksa orientasi calon titik pendukung.
2. Apabila orientasi sesuai, maka menghitung jumlah pendukung untuk setiap dua titik yang dipasangkan pada region 35. Sebagai contoh titik yang dipasangkan adalah A(120, 195) dan B(121, 196). Pendukung ditentukan berdasar jarak (d) titik C(122, 195) terhadap pasang garis yang dianggap segmen. Berikut perhitungannya:

$$d = \frac{|ax_0 + by_0 + c|}{\sqrt{a^2 + b^2}}$$

$$d_c = \frac{\left|(-(y_b - y_a))x_c + (x_a - x_b)y_c + \left(((-y_a) * (x_a - x_b)) + (x_a * (y_b - y_a))\right)\right|}{\sqrt{(-(y_b - y_a))^2 + (x_a - x_b)^2}}$$

$$d_c = \frac{\left|(-(196 - 195)) * 122 + ((120 - 121)) * 195 + \left(((-195) * (120 - 121)) + (120 * (196 - 195))\right)\right|}{\sqrt{(-(196 - 195))^2 + (120 - 121)^2}}$$

$$d_c = \frac{2}{1.414214}$$

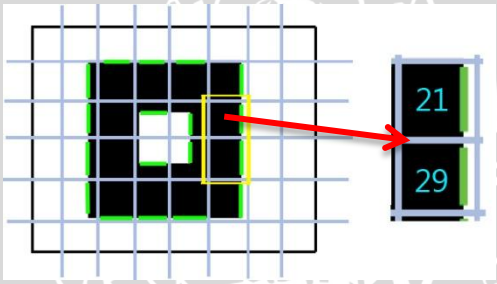
$$d_c = 1.414214$$

Jarak yang dihasilkan akan dihitung dan dibandingkan dengan *threshold* yang diberikan, apabila sesuai maka pendukung dari segmen AB akan ditambahkan 1.

3. Setelah semua jarak titik dihitung terhadap semua pasang titik yang dianggap segmen, langkah selanjutnya adalah mengurutkan data berdasarkan jumlah pendukung dan panjang segmen yang terbentuk.
4. Data yang berada pada urutan teratas akan mewakili region tersebut apabila jumlah pendukung sesuai dengan *threshold* yang diberikan.

3.4.3 Proses Menggabungkan Segmen

Proses selanjutnya akan membuktikan bahwa metode RANSAC berjalan dengan baik. Proses ini akan menggabungkan semua segmen yang terbentuk menjadi segmen yang lebih panjang dengan orientasi yang sesuai. Proses akan mengawali dengan mengambil satu segmen, kemudian mencari segmen lain dengan orientasi yang sesuai. Sebagai contoh, pada Gambar 3.12 segmen pada region 21 akan digabungkan dengan segmen pada region 29.



Gambar 3.12 Contoh Hasil Pembuatan Segmen yang akan Digabungkan

1. Pertama, melakukan pengecekan orientasi titik pada semua ujung segmen.
2. Apabila sesuai, kemudian mengecek gradien kedua segmen.
3. Apabila sesuai maka akan dicari pasang titik dengan jarak terjauh dari keempat ujung segmen menggunakan rumus *Manhattan Distance* seperti pada Persamaan 3.2. Misalkan mencari pasangan titik terjauh dari segmen pada region 21 yaitu A(238,81) dan B(237,118) dan segmen pada region 29 yaitu C(238,122) dan D(238,158).

$d = |x_1 - x_2| + |y_1 - y_2|$ $\quad$ $d_{AC} = |238 - 238| + |81 - 122|$

$d_{AC} = |x_A - x_C| + |y_A - y_C|$ $\quad$ $d_{AC} = 41$

Semua titik dihitung jaraknya seperti perhitungan di atas, maka dihasilkan jarak,

$d_{AC} = 41$ $\quad$ $d_{BC} = 5$

$d_{AD} = 77$ $\quad$ $d_{BD} = 41$

Dari perhitungan ini maka didapatkan bahwa segmen yang baru merupakan segmen AD dengan panjang 77 *pixel*.

4. Kemudian melakukan pengecekan ulang terhadap gradien garis yang sudah dilakukan penggabungan. Apabila sesuai dengan *threshold* yang diberikan, maka segmen ini akan menjadi acuan untuk menemukan segmen dengan orientasi yang sama selanjutnya.

3.4.4 Proses Mendeteksi Titik Sudut

Setelah semua segmen ditemukan, langkah selanjutnya adalah mendeteksi titik sudut dari *marker*. Proses ini diawali dengan mengambil salah satu segmen kemudian mencari segmen lain yang tidak dianggap sejajar sesuai dengan *threshold* yang diberikan. Untuk menemukan titik potong, menggunakan rumus yang merupakan turunan dari :

$$y = m * x + c$$

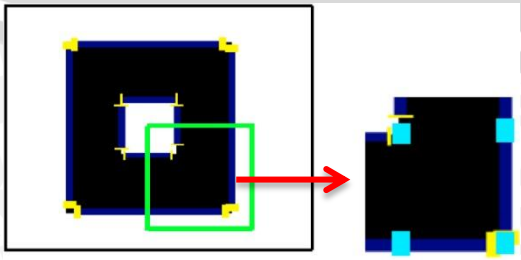
Ketika dua garis berpotongan maka kedua garis tersebut akan bertemu pada satu titik (x, y) yang sama. Maka:

$$y' = y'$$
$$m_1 * x' + c_1 = m_2 * x' + c_2$$
$$(m_1 * x' - m_2 * x') = c_2 - c_1$$
$$x' * (m_1 - m_2) = c_2 - c_1$$
$$x' = \frac{c_2 - c_1}{m_1 - m_2}$$

Kemudian mensubtitusikan nilai x pada salah satu persamaan, maka

$$y' = m_1 * x' + c_1$$

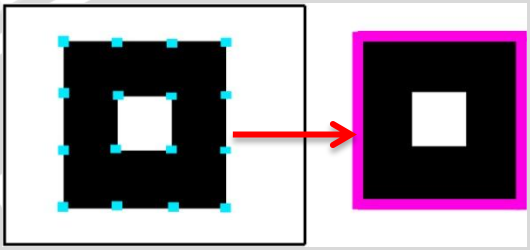
Maka diketahui bahwa kedua garis tersebut akan berpotongan pada titik (x', y'). Apabila pada proses sebelumnya ditemukan delapan garis, maka akan ditemukan 16 titik potong untuk segmen-segmen yang tidak dianggap sejajar. Contoh hasil penggabungan segmen yang akan dideteksi titik sudutnya dapat dilihat pada Gambar 3.13.



Gambar 3.13 Contoh Hasil Penggabungan segmen yang akan Dideteksi Titik Sudutnya

3.4.5 Proses Membentuk Garis Segi Empat (Menemukan *Marker)*

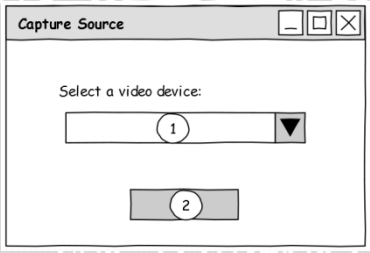
Pada proses ini akan dihitung semua jarak antar titik potong yang sudah ditemukan pada proses sebelumnya. Kemudian data akan diurutkan berdasarkan jarak yang diperoleh dari perhitungan jarak terhadap semua titik menggunakan rumus *Mahattan Distance* seperti pada Persamaan 3.2. Dua pasang titik pada baris teratas akan dianggap sebagai empat sudut *marker*. Apabila langkah ini selesai, maka *marker* telah ditemukan. Cotoh hasil deteksi *marker* dapat dilihat pada Gambar 3.14.



Gambar 3.14 Contoh Hasil Deteksi Titik Potong dan Hasil *Marker* yang Sudah Terdeteksi

3.5 Rancangan Antarmuka

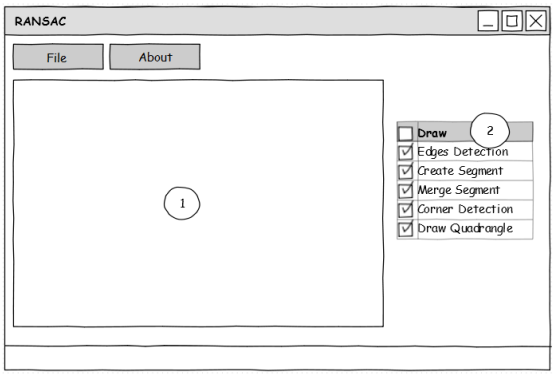
Implementasi aplikasi akan dilakukan dengan bahasa pemrograman C# dan pembuatan rancangan antar-muka menggunakan aplikasi yang bernama *Pencil*. Gambar 3.15 merupakan perancangan antarmuka awal aplikasi untuk memilih alat penangkap citra yang akan digunakan dalam aplikasi ini.



Gambar 3.15 Rancangan Tampilan Awal Aplikasi

Pada tampilan awal aplikasi akan menampilkan kotak dialog. Berisi sebuah *ComboBox*(1) yang akan meminta pengguna untuk memilih salah satu perangkat yang akan digunakan untuk menangkap citra dan sebuah *button*(2) untuk mengeksekusi pilihan. Sedangkan pada Gambar 3.16 merupakan perancangan antarmuka utama aplikasi.

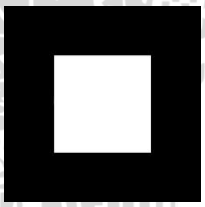
Pada tampilan utama aplikasi terdapat 1 *box* gambar dan 1 *Checklistbox*. *Box*(1) digunakan untuk menampilkan terjadinya proses AR dan *Checklistbox*(2) digunakan untuk memberikan pilihan pengguna untuk menampilkan langkah-langkah dalam mendeteksi *marker*.



Gambar 3.16 Rancangan Tampilan Utama Aplikasi

3.6 Skenario Pengujian Dan Analisis

Setelah tahapan implementasi algoritma selesai, maka tahap selanjutnya adalah melakukan pengujian terhadap sistem untuk melihat hasilnya. Pengujian dimaksudkan untuk menguji kelayakan sistem pendeteksi *marker* dan mengetahui apakah sistem yang telah dibuat bekerja dengan baik sesuai spesifikasi sistem yang telah ditentukan. Terdapat tiga variabel yang akan diujikan pada penelitian ini, yaitu: kondisi *maker*, posisi penangkap citra, dan kemiringan *marker*. Berikut merupakan salah satu *marker* yang akan digunakan dalam pengujian dapat dilihat pada Gambar 3.17.



Gambar 3.17 *Marker* yang Dipergunakan dalam Pengujian

Untuk melihat pengaruh metode RANSAC terhadap peningkatan performa pendeteksian *marker*, maka pada pengujian ini anak membandingkan aplikasi AR yang menggunakan dan tidak menggunakan metode RANSAC dengan menggunakan pustaka NyARToolKit 4.0.3. Pengujian akan dilakukan terhadap tiga jenis *marker* tidak ideal, melingkupi *marker* luntur, *marker* lecek, dan *marker* robek, dan satu *marker* ideal. Semua pengujian akan dilakukan di ruangan tertutup pada siang hari sekitar pukul 11.00-14.00. Kondisi ini diperlukan untuk membuat kondisi standar pengujian.

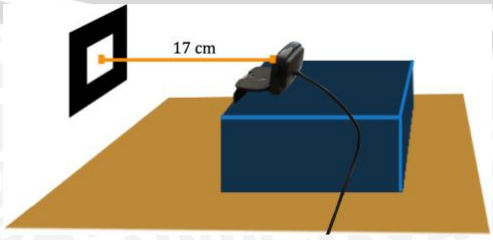
3.6.1 Pengujian Performa

Pengujian performa dilakukan untuk mengetahui bagaimana performa metode RANSAC pada aplikasi AR dalam melakukan pendeteksian *marker*. Pengujian performa aplikasi AR terhadap empat kondisi *marker* seperti yang sudah dijelaskan sebelumnya terdiri atas tiga tahap.

Pengujian performa 1 akan menguji performa metode RANSAC dalam mendeteksi beberapa macam kondisi *marker*. Masing-masing *marker* yang tidak ideal akan dihitung kerusakannya menggunakan aplikasi yang telah dibuat sebelumnya dengan membandingkan *marker* yang tidak ideal tersebut dengan *marker* yang ideal. Persentase kerusakan *marker* memiliki nilai yang berbeda dengan kondisi yang berbeda-beda. Setelah itu dilakukan pendeteksian *marker* yang tidak ideal tersebut menggunakan aplikasi AR tanpa dan yang sudah ditambahkan metode RANSAC. Hasil pengujian ini akan dianalisis untuk melihat peningkatan performa aplikasi setelah ditambahkan metode RANSAC. Pada pengujian ini, penangkap citra dan *marker* dipisahkan dengan jarak yang tetap yaitu 17 cm. Perancangan tabel pengujian dapat dilihat pada Tabel 3.2 dan perancangan lingkungan pengujian performa 1 dapat dilihat pada Gambar 3.18.

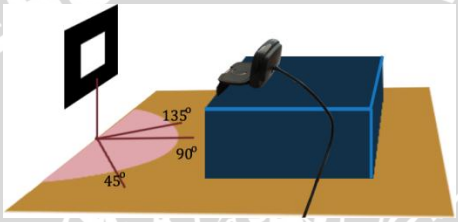
Tabel 3.2 Contoh Tabel Pengujian Performa 1

Kondisi *Marker*	**Kerusakan (%)**	**Hasil Deteksi NyARToolKit**	
		Tanpa RANSAC	**Dengan RANSAC**
Ideal			
Tinta luntur			
Tinta luntur			
Tinta luntur			
Lecek			
Lecek			
Lecek			
Robek			
Robek			
Robek			
Keterangan:			



Gambar 3.18 Contoh Perancangan Pengujian Performa 1

Pengujian performa berikutnya bertujuan untuk membandingkan aplikasi AR yang menggunakan dan yang tidak menggunakan metode RANSAC berdasarkan posisi penangkap citra. Sepuluh video pendek untuk masing-masing aplikasi akan digunakan dalam pengujian ini. Masing-masing video akan menampilkan satu *marker* yang berbeda pada setiap video dengan tiga perlakuan berbeda. Penangkapan citra dilakukan secara berurutan dari tiga sudut yang berbeda yaitu; 45°, 90°, dan 135°. *Marker* akan ditata sedemikian rupa sehingga akan terlihat pada semua *frame*. Kemudian akan dihitung persentase *marker* yang terdeteksi dengan baik. Perancangan lingkungan pengujian performa 2 dapat dilihat pada Gambar 3.19 dan perancangan tabel pengujian dapat dilihat pada Tabel 3.3.



Gambar 3.19 Contoh Perancangan Pengujian Performa 2

Tabel 3.3 Contoh Tabel Pengujian Performa 2

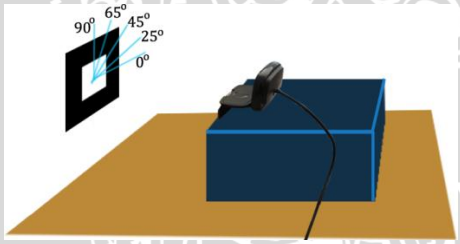
Kondisi *Marker*	**Kerusakan (%)**	**Hasil Deteksi NyARToolKit**					
		Tanpa RANSAC			**Dengan RANSAC**		
		45°	**90°**	**135°**	**45°**	**90°**	**135°**
Ideal							
Tinta luntur							
Tinta luntur							
Tinta luntur							
Lecek							
Lecek							
Lecek							
Robek							
Robek							
Robek							
Keterangan:							

Pengujian performa ketiga bertujuan untuk membandingkan aplikasi AR yang menggunakan metode RANSAC dan yang tidak menggunakan berdasarkan kemiringan *marker*. Sepuluh video pendek untuk masing-masing aplikasi akan digunakan dalam pengujian ini. Masing-masing video akan menampilkan satu *marker* yang berbeda pada setiap video dengan lima perlakuan berbeda. Penangkapan citra hanya dilakukan dari sudut 90° dan menggunakan beberapa macam bentuk kemiringan *marker* terhadap sumbu-y yaitu: 0°, 25°, 45°, 65°, dan

90°. *Marker* akan ditata sedemikian rupa sehingga akan terlihat pada semua *frame*. Kemudian akan dihitung persentase *marker* yang terdeteksi dengan baik. Perancangan tabel pengujian dapat dilihat pada Tabel 3.4 dan perancangan lingkungan pengujian performa 3 dapat dilihat pada Gambar 3.20.

Tabel 3.4 Contoh Tabel Pengujian Performa 3

Kondisi *Marker*	**Kerusakan (%)**	**Hasil Deteksi NyARToolKit**									
		Tanpa RANSAC					**Dengan RANSAC**				
		0°	**25°**	**45°**	**65°**	**90°**	**0°**	**25°**	**45°**	**65°**	**90°**
Ideal											
Tinta luntur											
Tinta luntur											
Tinta luntur											
Lecek											
Lecek											
Lecek											
Robek											
Robek											
Robek											
Keterangan:											



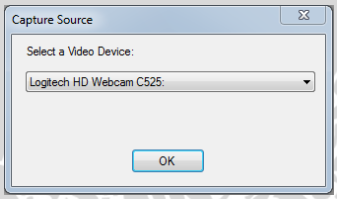
Gambar 3.20 Contoh Perancangan Pengujian Performa 3

BAB IV
IMPLEMENTASI

Pada bab ini akan dibahas implementasi berdasarkan metodologi dan perancangan yang telah dibahas pada bab sebelumnya.

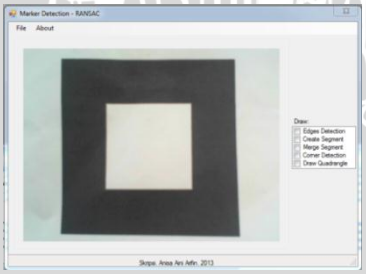
4.1 Implementasi Perangkat Lunak

Tahapan implementasi perangkat lunak merupakan tahap implementasi berdasarkan perancangan dan analisis kebutuhan yang telah dijelaskan sebelumnya. Tahapan ini akan menghasilkan suatu aplikasi sebagai media representatif terhadap hasil dari metode RANSAC. Berikut adalah penjelasan mengenai implementasi antarmuka aplikasi secara umum.



Gambar 4.1 Tampilan Awal Aplikasi

Gambar 4.1 merupakan tampilan yang muncul pertama kali ketika aplikasi dijalankan. Pada jendela ini, pengguna akan diberikan beberapa pilihan alat penangkap citra yang sudah terhubung pada PC. Setelah memilih alat penangkap citra yang diinginkan, pengguna akan diberikan tampilan utama aplikasi seperti pada Gambar 4.2.

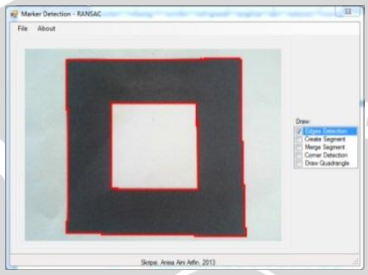


Gambar 4.2 Tampilan Utama Aplikasi

Pada jendela ini, akan ditampilkan sebuah kotak yang berisi video hasil proses AR. Pada bagian kanan jendela, terdapat *checklistbox* yang dapat dimanfaatkan oleh pengguna untuk memilih proses yang akan ditampilkan pada video. Aplikasi ini hanya terbatas untuk mendeteksi satu buah *marker* saja.

4.1.1 Implementasi Proses Deteksi Tepi

Tahap awal dari deteksi *marker* adalah deteksi tepi. Proses deteksi tepi dilakukan dengan menggunakan beberapa *pixel* dari citra berukuran 320 x 240 *pixel*. Setiap dua *pixel* data citra diseleksi secara vertikal dan horisontal untuk mendapatkan data tepian. Proses ini terbukti membuat aplikasi berjalan cukup cepat dan memberikan hasil yang cukup optimal. Berikut merupakan hasil implementasi dari proses deteksi tepi.



Gambar 4.3 Hasil Implementasi Proses Deteksi Tepi

4.1.2 Implementasi Proses Membuat Segmen

Langkah berikutnya, metode RANSAC digunakan untuk menemukan segmen dari kumpulan tepian yang sudah didapatkan dari proses deteksi tepi. Implementasi dari proses ini pada aplikasi adalah membuat sebuah *method*, yaitu createSegment yang merupakan bagian dari form.cs. Seperti yang sudah dibahas pada bab III bahwa data yang digunakan dalam proses ini adalah semua data yang ditemukan pada proses sebelumnya.

```
1   public void createSegment(){
2      for (int r = 0; r < 48; r++)
3      {
4      var query3 = from row1 in edges.AsEnumerable()
   where (row1.Field<int>("reg") == r) select row1;
5      foreach (DataRow row1 in query3){
6           random.ImportRow(row1);
7        }
8       ……
9      }
10  }
```

Gambar 4.4 Potongan Kode *Method* createSegment untuk Mengambil Data Per-Region

Pada baris ketiga pada potongan kode Gambar 4.4, proses ini diawali dengan mengambil data per region. Pembagian region sudah dilakukan pada proses sebelumnya dengan membagi data citra hasil proses deteksi tepi berdasarkan region sebesar 40x40 *pixel*. Sehingga didapatkan 48 buah region yaitu region 0 - 47. Kemudian pada baris 4-6, untuk setiap data tepian pada tabel edge diambil per-region dan menyimpan data tersebut pada tabel sementara yaitu random untuk diproses.

```
1   ……
2     for (int s = 0; s < random.Rows.Count - 1; s++)
3     {
4        for (int q = s + 1; q < random.Rows.Count; q++)
5        {
6            DataRow row = random.Rows[s];
7            b1[0] = Convert.ToInt32(row[0]);
8            b1[1] = Convert.ToInt32(row[1]);
9            b1[2] = Convert.ToInt32(row[2]);
10           b1[3] = Convert.ToInt32(row[3]);
11           b1[4] = Convert.ToInt32(row[4]);
12           DataRow row1 = random.Rows[q];
13           b2[0] = Convert.ToInt32(row1[0]);
14           b2[1] = Convert.ToInt32(row1[1]);
15           b2[2] = Convert.ToInt32(row1[2]);
16           b2[3] = Convert.ToInt32(row1[3]);
17           b2[4] = Convert.ToInt32(row1[4]);
18           ……
19           for (int f = 0; f < random.Rows.Count; f++){
20               DataRow row2 = random.Rows[f];
21               d[0] = Convert.ToInt32(row2[0]);
22               d[1] = Convert.ToInt32(row2[1]);
23               d[2] = Convert.ToInt32(row2[2]);
24               d[3] = Convert.ToInt32(row2[3]);
25               d[4] = Convert.ToInt32(row2[4]);
26               ……
27           }
28           ……
29       }
30    }
31  ……
```

Gambar 4.5 Potongan Kode *Method* createSegment untuk Mengambil Data yang akan Diproses

Pada baris ke 6 – 17 Gambar 4.5, *array* b1 dan b2 merupakan sepasang titik yang memiliki orientasi yang sesuai yang dianggap segmen untuk dicari jumlah pendukungnya. Sedangkan pada baris ke-21, *array* d adalah semua titik di dalam region selain pasangan titik yang dianggap segmen. Titik ini harus memiliki orientasi yang sesuai dengan b1 dan b2, dan akan dihitung jaraknya terhadap segmen b1 - b2 menggunakan Persamaan 3.1. Apabila jarak yang dihasilkan sesuai maka pendukung dari b1 - b2 akan ditambahkan 1. Semua data akan disimpan pada tabel sementara yang lainnya yang bernama kecil.

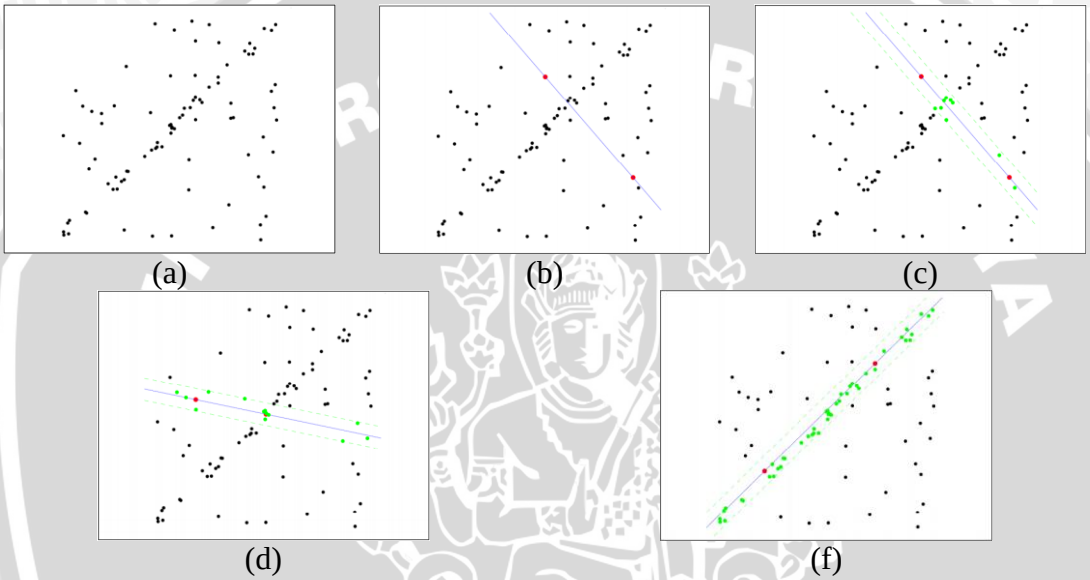
```
1   ……
2   string strSort = "dkg DESC, jarak DESC";
3   DataView dtview = new DataView(kecil);
4   dtview.Sort = strSort;
5   DataTabel dtsorted = dtview.ToTabel();
6   ……
7   if (Convert.ToInt32(dtsorted.Rows[0][7]) >= 6){
8           segment.Rows.Add(new object[] { Convert.ToInt32(dtsorted.Rows[0][0]),
    Convert.ToInt32(dtsorted.Rows[0][1]), Convert.ToInt32(dtsorted.Rows[0][2]),
    Convert.ToInt32(dtsorted.Rows[0][3]), Convert.ToInt32(dtsorted.Rows[0][4]),
    Convert.ToInt32(dtsorted.Rows[0][5]), degree, r,
    Convert.ToInt32(dtsorted.Rows[0][7]), coupled });
9       }
10  ……
```

Gambar 4.6 Potongan Kode *Method* createSegment untuk Menentukan Segmen

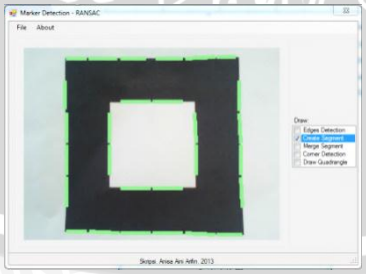
Pada baris ke 2 - 5 Gambar 4.6, setelah semua titik diproses satu-persatu, maka tabel kecil akan diurutkan berdasarkan jumlah pendukung secara *descending*

dan jarak secara *descending* antar titik yang dianggap segmen ke dalam tabel dtsorted. Pada baris ke 7, apabila baris dengan urutan teratas pada tabel dtsorted memiliki jumlah pendukung sesuai dengan *threshold* yang diberikan maka data tersebut akan disimpan kedalam tabel baru yang bernama segment.

Gambar 4.7 merupakan ilustrasi proses pembuatan segmen menggunakan metode RANSAC. Sedangkan Gambar 4.8 merupakan hasil implementasi dari proses membuat segmen. Setiap segmen yang terbentuk mewakili masing-masing region dan memiliki orientasi sesuai dengan tepian yang membentuknya.



Gambar 4.7 (a) Kumpulan data tepian, (b) menghipotesis dua data tepian secara random dan mengecek orientasi keduanya, (c) Menghitung jumlah titik yang mendukung kedua tepian yang dihipotesis, (d) Mengulangi penghipotesisan dan penghitungan jumlah konsensus tepian hingga semua data tepian diproses, dan (e) Mencari pasangan tepian yang memiliki jarak terjauh dan jumlah konsensus atau pendukung terbanyak.



Gambar 4.8 Hasil Implementasi Proses Membuat Segmen

4.1.3 Implementasi Proses Menggabungkan Segmen

Implementasi dari proses ini pada aplikasi adalah membuat sebuah *method*, yaitu mergeSegment yang merupakan bagian dari form.cs.

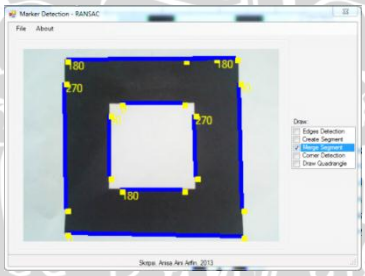
```
1   public void mergeSegment()
2   {
3     …….
4     string strSort1 = "reg ASC";
5     DataView dtview1 = new DataView(segment);
6     dtview1.Sort = strSort1;
7     segmentSortedbyRegion = dtview1.ToTable();
8     for (int s = 0; s < segmentSortedbyRegion.Rows.Count - 1; s++)
9     {
10       for (int q = s + 1; q < segmentSortedbyRegion.Rows.Count; q++)
11       {
12            DataRow row5 = segmentSortedbyRegion.Rows[s];
13            ky[0] = Convert.ToDouble(row5[0]);
14            ky[1] = Convert.ToDouble(row5[1]);
15            ky[2] = Convert.ToDouble(row5[2]);
16            ky[3] = Convert.ToDouble(row5[3]);
17            ky[4] = Convert.ToDouble(row5[4]);
18            ky[5] = Convert.ToDouble(row5[5]);
19            ky[6] = Convert.ToDouble(row5[6]);
20            ky[7] = Convert.ToDouble(row5[7]);
21            ky[8] = Convert.ToDouble(row5[8]);
22            ky[9] = Convert.ToDouble(row5[9]);
23            DataRow row8 = segmentSortedbyRegion.Rows[q];
24            yu[0] = Convert.ToDouble(row8[0]);
25            yu[1] = Convert.ToDouble(row8[1]);
26            yu[2] = Convert.ToDouble(row8[2]);
27            yu[3] = Convert.ToDouble(row8[3]);
28            yu[4] = Convert.ToDouble(row8[4]);
29            yu[5] = Convert.ToDouble(row8[5]);
30            yu[6] = Convert.ToDouble(row8[6]);
31            yu[7] = Convert.ToDouble(row8[7]);
32            yu[8] = Convert.ToDouble(row8[8]);
33            yu[9] = Convert.ToDouble(row8[9]);
34            if ((Math.Abs(ky[2] - yu[2]) <=thresholdOr) && (Math.Abs(ky[5] -
    yu[5]) <= thresholdOr))
35            {
36              if (Math.Abs(ky[6] - yu[6]) <= thresholdGrad)
37              {
38                  ……
                  double shadowGrad = Math.Atan2((yu[id2 + 1] - ky[id1 + 1]),
39  (yu[id2] - ky[id1])) * 180 / Math.PI;
40                if (shadowGrad - ky[6] <= (gradien))
41                {
42                  segmentSortedbyRegion.Rows[q][0] = ky[id1];
43                  segmentSortedbyRegion.Rows[q][1] = ky[id1 + 1];
44                  segmentSortedbyRegion.Rows[q][2] = ky[id1 + 2];
45                  segmentSortedbyRegion.Rows[q][3] = yu[id2];
46                  segmentSortedbyRegion.Rows[q][4] = yu[id2 + 1];
47                  segmentSortedbyRegion.Rows[q][5] = yu[id2 + 2];
48                  segmentSortedbyRegion.Rows[q][6]= shadowOrientation;
49                  segmentSortedbyRegion.Rows[q][7] = panjang;
50                  segmentSortedbyRegion.Rows[q][8] = theta;
51                  segmentSortedbyRegion.Rows[q][9] = 1;
52                  doesnotmerge = false;
53                  segmentSortedbyRegion.Rows[s].Delete();
54                  s--;
55                  ……
56                }
57              }
58              ……
59            }
60       }
         if(doesnotmerge == true &&
61  Convert.ToInt32(segmentSortedbyRegion.Rows[s][9]) != 1)
62       {
63          segmentSortedbyRegion.Rows[s].Delete();
64       }
65     }
66     ……
67  }
```

Gambar 4.9 Potongan Kode *Method* mergeSegment untuk Menggabungkan Segmen

Pada Gambar 4.9 baris 4-7, proses diawali dengan menggabungkan segmen pertama yaitu *array* ky dan segmen kedua yaitu *array* yu dengan mengurutkan data dari tabel segment yang merupakan hasil proses sebelumnya berdasarkan region secara *ascending* ke dalam tabel segmentSortedbyRegion. Kemudian pada baris 8-33 mengambil dua segmen kecil untuk dilakukan pengecekan orientasi dari keempat titik pembentuk dua segmen tersebut pada baris 34 dan pengecekan gradien pada baris 36. Apabila sesuai dengan *threshold* yang diberikan maka akan dicari jarak titik terjauh dari kedua segmen untuk menentukan kedua ujung segmen baru seperti perhitungan pada subbab 3.4.3 menggunakan rumus *Mahattan Distance*.

Setelah ditentukan segmen yang baru maka akan dihitung gradien segmen tersebut (shadowGrad). Apabila nilai shadowGrad sesuai dengan *threshold* yang diberikan, maka data segmen yang baru akan menggantikan data segmen yu dan segmen ky akan dihapus. Namun apabila data segmen ky tidak cocok dengan segmen yu, atau belum pernah dipasangkan sebelumnya, maka segmen ky akan dihapus dan proses akan dilanjutkan dengan segmen ky yang baru.

Gambar 4.10 merupakan hasil implementasi dari proses menggabungkan segmen. Garis yang dihasilkan tidak memiliki panjang maksimal karena batas garis ditentukan oleh segmen garis yang dihasilkan oleh proses sebelumnya.



Gambar 4.10 Hasil Implementasi Proses Menggabungkan Segmen

4.1.4 Implementasi Proses Mendeteksi Titik Sudut

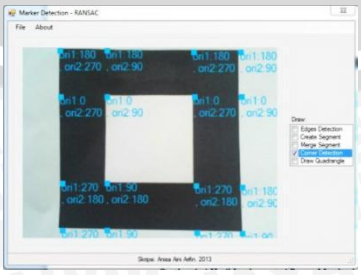
Implementasi dari proses ini pada aplikasi adalah membuat sebuah *method*, yaitu cornerDetection yang merupakan bagian dari form.cs.

```
1  public void cornerDetection(Bitmap biner)
2  {
3      ……
4      for (int i = 0; i < segmentSortedbyRegion.Rows.Count - 1; i++)
5      {
6          for (int j = i + 1; j < segmentSortedbyRegion.Rows.Count; j++)
7          {
8              DataRow row5 = segmentSortedbyRegion.Rows[i];
9              ch[0] = Convert.ToDouble(row5[0]);
10             ch[1] = Convert.ToDouble(row5[1]);
```

```
11              ch[2] = Convert.ToDouble(row5[2]);
12              ch[3] = Convert.ToDouble(row5[3]);
13              ch[4] = Convert.ToDouble(row5[4]);
14              ch[5] = Convert.ToDouble(row5[5]);
15              ch[6] = Convert.ToDouble(row5[6]);
16              ch[7] = Convert.ToDouble(row5[7]);
17              ch[8] = Convert.ToDouble(row5[8]);
18              ch[9] = Convert.ToDouble(row5[9]);
19              DataRow row6 = segmentSortedbyRegion.Rows[j];
20              ho[0] = Convert.ToDouble(row6[0]);
21              ho[1] = Convert.ToDouble(row6[1]);
22              ho[2] = Convert.ToDouble(row6[2]);
23              ho[3] = Convert.ToDouble(row6[3]);
24              ho[4] = Convert.ToDouble(row6[4]);
25              ho[5] = Convert.ToDouble(row6[5]);
26              ho[6] = Convert.ToDouble(row6[6]);
27              ho[7] = Convert.ToDouble(row6[7]);
28              ho[8] = Convert.ToDouble(row6[8]);
29              ho[9] = Convert.ToDouble(row6[9]);
30              if (Math.Abs(ch[6] - ho[6]) > thresholdGrad)
31              {
32                  ……
33                  quadrangle.Rows.Add(new object[] { ch[0], ch[1],  ch[3],
    ch[4], ch[8], ho[0], ho[1], ho[3], ho[4], ho[8],  titikPotong[0],
    titikPotong[1] });
34              }
35          }
36      }
37      ……..
38  }
```

Gambar 4.11 Potongan Kode *Method* conrnerDetection untuk Mendeteksi Titik Sudut *Marker*

Pada Gambar 4.11, baris ke 8-29 merupakan langkah pertama dalam mendeteksi titik sudut pada *marker*, yaitu mengambil data segmen ch dan segmen ho dari tabel segmentSortedbyRegion, hasil proses penggabungan segmen. Kedua segmen ini harus sesuai gradiennya dengan nilai *threshold* yang diberikan, dan tidak boleh sejajar. Apabila sesuai maka akan dicari titik potong dari kedua segmen tersebut berdasarkan persamaan garis sesuai perhitungan pada subbab 3.4.4. Jumlah titik yang dihasilkan bergantung dengan jumlah garis yang dihasilkan pada proses menggabungkan segmen. Hasil proses ini akan disimpan pada tabel quadrangle. Hasil implementasi dari proses mendeteksi titik sudut dapat dilihat pada Gambar 4.12.



Gambar 4.12 Hasil Implementasi Proses Mendeteksi Titik Sudut

4.1.5 Implementasi Proses Membentuk Garis Segi Empat (Menemukan *Marker*)

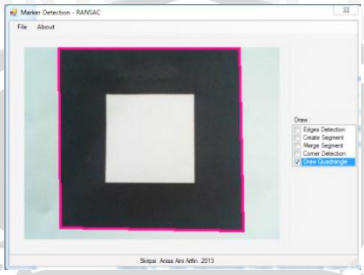
Implementasi dari proses ini pada aplikasi adalah membuat sebuah *method*, yaitu drawQuadrangle yang merupakan bagian dari form.cs.

```
1  public void drawQuadrangle()
2  {
3     for (int u = 0; u < quadrangle.Rows.Count - 1; u++)
4     {
5        for (int v = u + 1; v < quadrangle.Rows.Count; v++)
6        {
7            DataRow row5 = quadrangle.Rows[u];
8            hy[0] = Convert.ToDouble(row5[0]);
9            hy[1] = Convert.ToDouble(row5[1]);
10           hy[2] = Convert.ToDouble(row5[2]);
11           hy[3] = Convert.ToDouble(row5[3]);
12           hy[4] = Convert.ToDouble(row5[4]);
13           hy[5] = Convert.ToDouble(row5[5]);
14           hy[6] = Convert.ToDouble(row5[6]);
15           hy[7] = Convert.ToDouble(row5[7]);
16           hy[8] = Convert.ToDouble(row5[8]);
17           hy[9] = Convert.ToDouble(row5[9]);
18           hy[10] = Convert.ToDouble(row5[10]);
19           hy[11] = Convert.ToDouble(row5[11]);
20           DataRow row6 = quadrangle.Rows[v];
21           un[0] = Convert.ToDouble(row6[0]);
22           un[1] = Convert.ToDouble(row6[1]);
23           un[2] = Convert.ToDouble(row6[2]);
24           un[3] = Convert.ToDouble(row6[3]);
25           un[4] = Convert.ToDouble(row6[4]);
26           un[5] = Convert.ToDouble(row6[5]);
27           un[6] = Convert.ToDouble(row6[6]);
28           un[7] = Convert.ToDouble(row6[7]);
29           un[8] = Convert.ToDouble(row6[8]);
30           un[9] = Convert.ToDouble(row6[9]);
31           un[10] = Convert.ToDouble(row6[10]);
32           un[11] = Convert.ToDouble(row6[11]);
33           ……
             quadrangleCornerFinal.Rows.Add(new object[] { hy[10],
hy[11], un[10], un[11], jaraktitik });
35       }
36    }
37    string strSort2 = "jarak DESC";
38    DataView dtview2 = new DataView(quadrangleCornerFinal);
39    dtview2.Sort = strSort2;
40    CornerFinal = dtview2.ToTable();
41    ……
42 }
```

Gambar 4.13 Potongan Kode *Method* drawQuadrangle untuk Membentuk Segi Empat dari *Marker*

Proses terakhir dalam mendeteksi *marker* adalah membentuk pola segi empat. Prinsip terbentuknya segi empat adalah dengan mencari dua pasang titik sudut dengan jarak terjauh. Pada Gambar 4.13, proses ini diawali dengan mengombinasikan dua data titik yaitu *array* hy dan *array* un dari semua data hasil proses mendeteksi titik sudut secara berurutan. Kemudian menghitung jarak kedua titik tersebut menggunakan rumus *Manhattan Distance* seperti perhitungan pada subbab 3.4.3. Hasil semua pehitungan akan dimasukkan pada tabel quadrangleCornerFinal. Kemudian data tersebut akan diurutkan berdasarkan jarak secara *descending* untuk

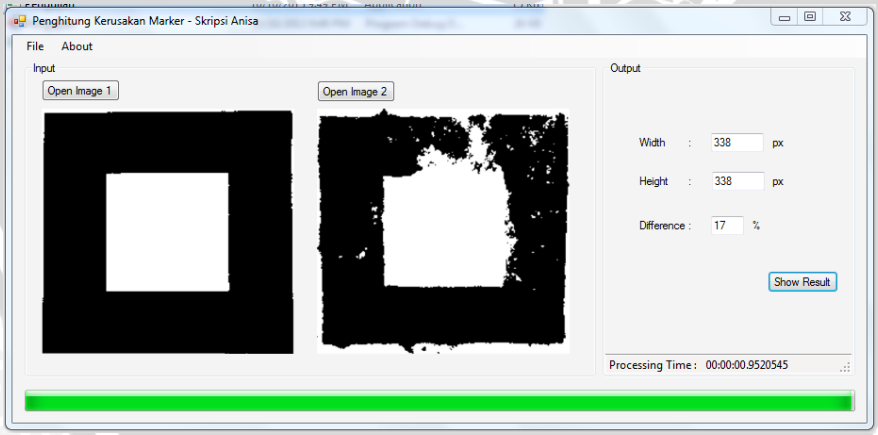
mendapatkan dua pasang titik yang jaraknya paling jauh ke dalam tabel CornerFinal. Pada kondisi normal, dua pasang titik yang memiliki jarak paling panjang dari sebuah bangun segi empat adalah diagonalnya. Sehingga, dua baris teratas pada tabel CornerFinal akan dianggap sebagai ujung-ujung *marker*. Hasil implementasi dari proses membentuk garis segi empat dapat dilihat pada Gambar 4.14.



Gambar 4.14 Hasil Implementasi Proses Membentuk Garis Segi Empat (Menemukan *Marker*)

4.2 Program Untuk Menghitung Persentase Kerusakan *Marker*

Program ini akan membandingkan setiap satu *pixel* pada citra *marker* acuan dengan citra *marker* yang tidak ideal yang telah dibinerisasi. Apabila perbedaan keduanya sesuai dengan *threshold* yang diberikan maka perbedaan mereka akan dihitung. Program ini bekerja sesuai data citra yang diberikan, sehingga persentase kerusakan sangat bergantung pada kondisi saat citra ditangkap (*capture*).

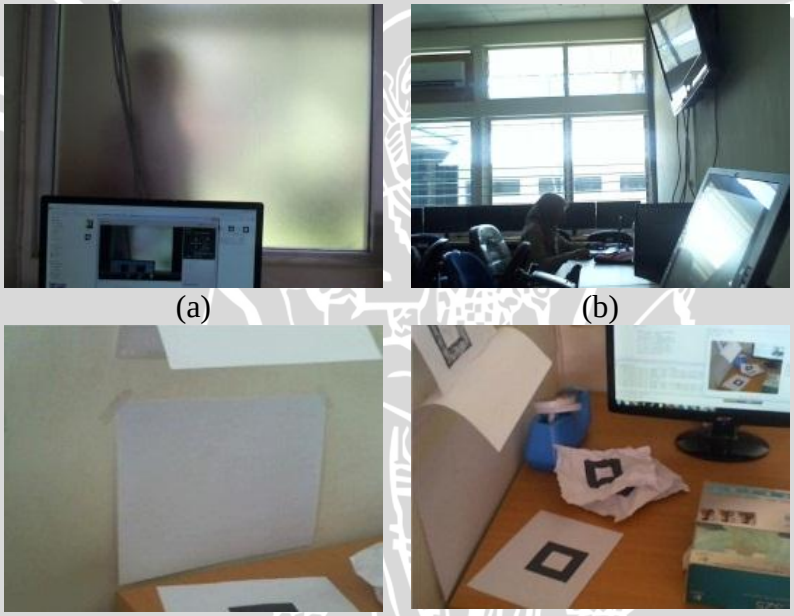


Gambar 4.15 Tampilan Program Penghitung Kerusakan *Marker*

BAB V
PENGUJIAN DAN ANALISIS

5.1 Pengujian

Sesuai dengan perancangan uji coba pada bab III, pengujian performa dilakukan dengan membandingkan aplikasi yang menggunakan dan yang tidak menggunakan metode RANSAC. Pengujian dilakukan pada tanggal 9 Oktober 2013 pukul 11.30 - 14.00 di Laboraturioum Game PTIIK. Sumber cahaya terdapat pada sebelah kanan dan dominan pada sebelah kiri tempat pengujian. Kondisi ini menjadi standar dalam seluruh rangkaian pengujian performa yang dilakukan pada penelitian ini (Gambar 5.1).



(a) (b)

(c) (d)

Gambar 5.1 Kondisi Lingkungan Pengujian (a) Sebelah Kanan, (b) Sebelah Kiri, (c) Tempat Meletakkan *Marker*, dan (d) Tempat Pengujian.

Pada penelitian ini, *marker* akan dianggap telah terbaca apabila sisi terluar dari *marker* pernah ditandai dengan garis segi empat yang mengelilingi *marker*, minimal satu kali oleh aplikasi. Sebelum melakukan pengujian, terlebih dahulu *marker* yang akan diuji dihitung persentase kerusakannya dengan program yang sudah dijelaskan sebelumnya.

Tabel 5.1 Persentase Kerusakan *Marker* Uji

Kondisi *Marker*	Kerusakan (%)	Kondisi *Marker*	Kerusakan (%)
Ideal 	0	Lecek 	15
Tinta luntur 	10	Lecek 	29
Tinta luntur 	17	Robek 	25
Tinta luntur 	7	Robek 	11
Lecek 	21	Robek 	16

Hasil pengujian performa 1 yang digunakan untuk menguji metode RANSAC dalam mendeteksi beberapa macam persentase kerusakan *marker* dapat dilihat pada Tabel 5.1. Pengujian ini dilakukan untuk melihat peningkatan performa aplikasi AR sebelum dan setelah ditambahkan metode RANSAC dalam mendeteksi *marker* yang tidak ideal.

Tabel 5.2 Hasil Pengujian Performa 1

Kondisi *Marker*	Kerusakan (%)	Hasil Deteksi NyARToolKit	
		Tanpa RANSAC	Dengan RANSAC
Ideal	0	1	1
Tinta luntur	10	1	1
Tinta luntur	17	0	1
Tinta luntur	7	1	1
Lecek	21	0	1
Lecek	15	0	1
Lecek	29	0	1
Robek	25	0	1
Robek	11	1	1
Robek	16	1	1
Keterangan:		(1): terdeteksi	(0): tidak terdeteksi

Pada di atas dapat dilihat bahwa aplikasi yang menggunakan metode RANSAC dapat mendeteksi semua *marker* dengan baik. Sedangkan pada aplikasi

yang tidak menggunakan metode RANSAC hanya 50% *marker* (5 dari 10 kondisi *marker* yang diberikan) yang dapat dideteksi. Kegagalan ini disebabkan oleh tidak ditemukannya empat titik sudut *marker* dengan sempurna oleh aplikasi.

Hasil dari pengujian performa 2 yang bertujuan untuk membandingkan aplikasi AR yang menggunakan dan yang tidak menggunakan metode RANSAC berdasarkan posisi penangkap citra dapat dilihat padaTabel 5.3. Pada pengujian ini, *marker* ditata sedemikian rupa sehingga terlihat pada semua *frame* tanpa memperhatikan jarak penangkap citra dan *marker*.

Tabel 5.3 Hasil Pengujian Performa 2

Kondisi *Marker*	**Kerusakan (%)**	**Hasil Deteksi NyARToolKit**					
		Tanpa RANSAC			**Dengan RANSAC**		
		45°	**90°**	**135°**	**45°**	**90°**	**135°**
Ideal	0	1	1	1	1	1	1
Tinta luntur	10	0	1	1	1	1	1
Tinta luntur	17	0	0	0	1	1	1
Tinta luntur	7	1	1	1	1	1	1
Lecek	21	0	0	0	1	1	0
Lecek	15	0	1	0	1	1	0
Lecek	29	0	0	0	1	1	0
Robek	25	0	0	0	0	1	0
Robek	11	1	1	1	1	1	1
Robek	16	0	1	0	1	1	1
Keterangan:		(1): terdeteksi			(0): tidak terdeteksi		

Pada Tabel 5.3 dapat dilihat bahwa dari 30 kali percobaan untuk setiap aplikasi AR, didapatkan bahwa 83,33% *marker* dapat dideteksi oleh aplikasi yang menggunakan metode RANSAC, hanya 25 dari 30 kondisi kombinasi 10 kondisi *marker* dan 3 posisi penangkap citra. Sedangkan pada aplikasi yang tidak menggunakan metode RANSAC, hanya sekitar 43,33% *marker* yang dapat dideteksi, hanya 13 dari 30 kondisi kombinasi 10 kondisi *marker* dan 3 posisi penangkap citra. Hal ini dimungkinkan karena berpidahnya posisi penangkap citra akan mempengaruhi luasan wilayah *marker* yang ditangkap tidak sempurna karena ketidakidealan *marker* tersebut.

Untuk hasil dari pengujian performa 3 yang bertujuan untuk membandingkan aplikasi AR yang menggunakan dan yang tidak menggunakan metode RANSAC berdasarkan kemiringan *marker* dapat dilihat pada Tabel 5.4. Pada pengujian ini, *marker* ditata sedemikian rupa sehingga terlihat pada semua *frame* tanpa memperhatikan jarak penangkap citra dan *marker*.

Tabel 5.4 Hasil Pengujian Performa 3

Kondisi *Marker*	Kerusakan (%)	Hasil Deteksi NyARToolKit									
		Tanpa RANSAC					Dengan RANSAC				
		0°	25°	45°	65°	90°	0°	25°	45°	65°	90°
Ideal	0	1	1	1	1	1	1	1	1	1	1
Tinta luntur	10	1	1	1	1	1	1	1	1	1	1
Tinta luntur	17	0	0	0	0	0	1	1	1	1	1
Tinta luntur	7	1	1	1	1	1	1	1	1	1	1
Lecek	21	0	0	1	0	0	1	1	1	1	1
Lecek	15	1	1	1	1	1	1	1	1	1	1
Lecek	29	0	0	0	0	0	1	0	0	0	1
Robek	25	0	0	0	0	0	1	1	1	1	1
Robek	11	1	1	1	1	1	1	1	1	1	1
Robek	16	1	0	0	0	1	1	1	1	1	1
Keterangan:		(1): terdeteksi				(0): tidak terdeteksi					

Pada Tabel 5.4 dapat dilihat bahwa dari 50 percobaan yang telah dilakukan, aplikasi yang menggunakan metode RANSAC berhasil mendeteksi 94% *marker* yang diberikan, hanya 47 dari 50 kondisi kombinasi 10 kondisi *marker* dan 5 kemiringan *marker*. Sedangkan aplikasi yang tidak menggunakan metode RANSAC hanya mampu mendeteksi 56% saja, hanya 28 dari 50 kondisi kombinasi 10 kondisi *marker* dan 5 kemiringan *marker*. Aplikasi dengan menggunakan RANSAC masih kurang berhasil mendeteksi *marker* yang lecek dengan persentase kerusakan yang cukup tinggi apabila posisi *marker* tidak pada posisi tegak. Sedangkan aplikasi tanpa metode RANSAC tidak dapat mendeteksi *marker* yang memiliki sisi-sisi tidak lurus sempurna dan salah satu sudut atau lebih yang hilang atau tidak sempurna seperti yang dapat diihat pada Tabel 5.1.

5.2 Analisis

Berdasarkan tabel hasil pengujian performa 1, dapat dilihat bahwa aplikasi AR yang menggunakan metode RANSAC berhasil mengoptimalkan aplikasi dalam mendeteksi *marker* dengan sepuluh kondisi *marker*, rata-rata sebesar 50%. Sedangkan dari tabel hasil pengujian performa 2, didapatkan bahwa metode RANSAC berhasil mengoptimalkan aplikasi dalam mendeteksi *marker* dengan tiga macam posisi penangkap citra, rata-rata sebesar 40%. Terakhir, dari tabel hasil pengujian performa 3, metode RANSAC berhasil mengoptimalkan aplikasi dalam mendeteksi *marker* dengan lima macam kemiringan *marker*, rata-rata sebesar 38%.

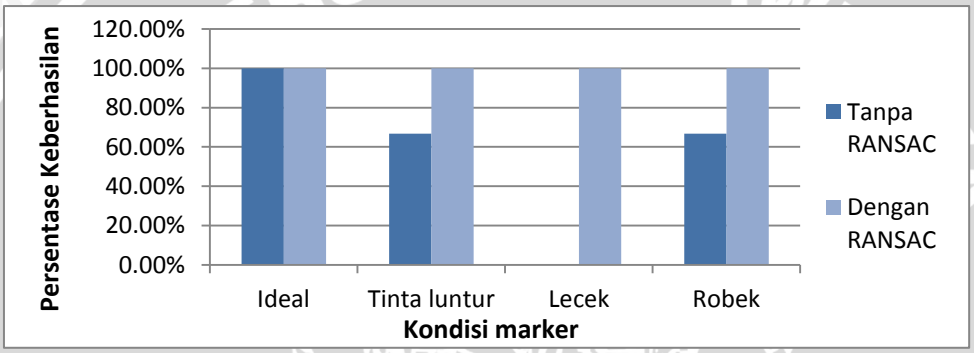
Dari hasil yang didapatkan dari pengujian performa dapat dihitung persentase keberhasilan aplikasi dalam mendeteksi *marker* per kondisi *marker*. Persentase

didapat dengan membagi jumlah keberhasilan dengan jumlah percobaan terhadap masing-masing kondisi *marker*. Tabel 5.5 merangkum persentase keberhasilan pengujian performa 1.

Tabel 5.5 Persentase Keberhasilan Pengujian Performa 1

Kondisi *Marker*	Persentase Keberhasilan	
	Tanpa RANSAC	Dengan RANSAC
Ideal	100.00%	100.00%
Tinta luntur	66.67%	100.00%
Lecek	0.00%	100.00%
Robek	66.67%	100.00%

Untuk mempermudah dalam pembacaan Tabel 5.5, berikut adalah representasi grafik persentase keberhasilan pengujian performa 1.



Gambar 5.2 Grafik Persentase Keberhasilan Pengujian Performa 1

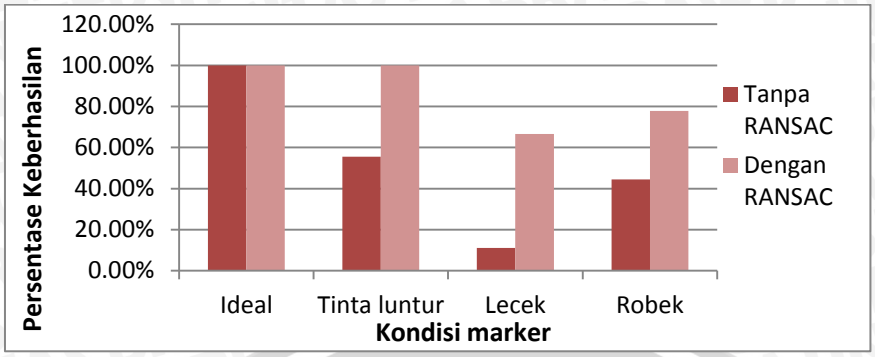
Dari grafik pada Gambar 5.2 dapat dilihat bahwa aplikasi yang menggunakan metode RANSAC berhasil mendeteksi semua *marker* dengan sempurna yaitu 100%. Sedangkan pada aplikasi yang tidak menggunakan metode RANSAC, hanya berhasil mendeteksi *marker* ideal dengan baik yaitu sebesar 100%, namun sama sekali tidak dapat mendeteksi *marker* dengan kondisi lecek. Sedangkan pendeteksian terhadap *marker* dengan tinta luntur dan *marker*robek hanya berhasil mendeteksi 66.67% saja.

Selanjutnya, untuk merangkum persentase keberhasilan pengujian performa 2 dapat dilihat pada Tabel 5.6.

Tabel 5.6 Persentase Keberhasilan Pengujian Performa 2

Kondisi *Marker*	Persentase Keberhasilan	
	Tanpa RANSAC	Dengan RANSAC
Ideal	100.00%	100.00%
Tinta luntur	55.56%	100.00%
Lecek	11.11%	66.67%
Robek	44.44%	77.78%

Untuk mempermudah dalam pembacaan Tabel 5.6, berikut adalah representasi grafik persentase keberhasilan pengujian performa 2.



Gambar 5.3 Grafik Persentase Keberhasilan Pengujian Performa 2

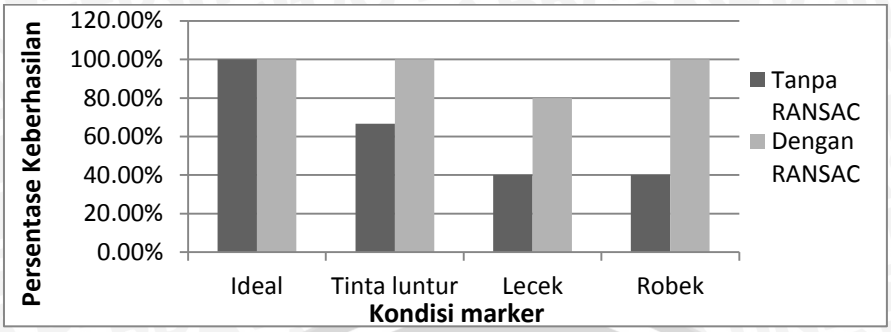
Dari Gambar 5.3 dapat dilihat bahwa aplikasi yang menggunakan metode RANSAC hampir selalu lebih baik atau sama dengan aplikasi yang tidak menggunakan metode RANSAC dalam mendeteksi *marker*. Pada pengujian ini pun *marker* ideal berhasil dideteksi dengan baik oleh kedua aplikasi, dan *marker* lecek tidak terlalu sempurna dideteksi oleh kedua aplikasi. Untuk aplikasi dengan metode RANSAC hanya berhasil mendeteksi *marker* lecek sebesar 66.67% dengan beberapa perubahan posisi penangkap citra. Sedangkan aplikasi tanpa metode RANSAC hanya berhasil 11.11% saja. Pada kondisi *marker* dengan tinta yang luntur, aplikasi dengan metode RANSAC berhasil 100% mendeteksi *marker* dengan beberapa posisi penangkap citra, sedangkan aplikasi tanpa metode RANSAC hanya berhasil 55.56% mendeteksi *marker*. Untuk kondisi *marker* terakhir, yaitu *marker* robek, aplikasi dengan RANSAC berhasil mendeteksi *marker* lebih baik daripada aplikasi tanpa metode RANSAC yaitu sebesar 77.78%, sedangkan yang tanpa RANSAC hanya sebesar 44.44%.

Berikutnya, untuk persentase keberhasilan pengujian performa 3 akan dirangkum pada Tabel 5.7.

Tabel 5.7 Persentase Keberhasilan Pengujian Performa 3

Kondisi *Marker*	**Persentase Keberhasilan**	
	Tanpa RANSAC	**Dengan RANSAC**
Ideal	100.00%	100.00%
Tinta luntur	66.67%	100.00%
Lecek	40.00%	80.00%
Robek	40.00%	100.00%

Gambar 5.4 berikut akan mempermudah pembacaan Tabel 5.7 dalam bentuk representasi grafik.



Gambar 5.4 Grafik Persentase Keberhasilan Pengujian Performa 3

Dari grafik di atas dapat dilihat bahwa aplikasi yang menggunakan metode RANSAC lebih baik dalam mendeteksi *marker* yang posisinya dimiringkan daripada aplikasi yang tidak menggunakan metode RANSAC. Aplikasi yang menggunakan metode RANSAC dapat mendeteksi 100% semuaposisi kemiringan *marker* selain *marker* yang lecek yang hanya dapat mendeteksi 80% posisi *marker*. Sedangkan aplikasi yang tidak menggunakan metode RANSAC hanya dapat mendeteksi 100% posisi *marker* ideal yang dimiringkan saja. Sedangkan untuk *marker* dengan tinta yang luntur aplikasi tanpa metode RANSAC hanya mampu mendeteksi 66.67% posisi kemiringan *marker*. Dan untuk dua kondisi *marker* terakhir, *marker* lecek dan *marker* robek, aplikasi AR tanpa metode RANSAC hanya dapat mendeteksi *marker* sebesar 40% posisi kemiringan *marker*.

Dari hasil pengujian performa 1, 2, dan 3, dapat dilihat bahwa penambahan metode RANSAC dapat memperbaiki dan meningkatkan performa aplikasi dalam mendeteksi *marker* yang tidak ideal lebih baik daripada aplikasi mendeteksi *marker* yang menggunakan pustaka NyARToolKit 4.0.3. Karena pendeteksian garis berdasarkan hasil konsensus dari kumpulan data tepian yang memiliki nilai magnitud dan orientasi yang hampir sama kemudian dihubungkan dengan metode RANSAC lebih baik dalam mendeteksi garis *marker* yang tidak sempurna pada *marker* tidak ideal daripada mencari empat garis lurus yang saling berhubungan dari *marker* seperti yang dilakukan pada aplikasi yang menggunakan pustaka NyARToolKit 4.0.3.

BAB VI
PENUTUP

6.1 Kesimpulan

Berdasarkan penelitian yang telah dilakukan, kesimpulan yang diperoleh dari penelitian ini, antara lain:

1. Untuk mengetahui peningkatan performa aplikasi AR dalam mendeteksi *marker* yang tidak ideal menggunakan metode RANSAC, maka penulis merancang dan membangun sebuah aplikasi yang mengimplementasikan metode tersebut. Perancangan aplikasi direpresentasikan dalam bentuk beberapa diagram alir.
2. Metode RANSAC telah berhasil diimplementasikan pada pustaka NyARToolKit 4.0.3 dengan memodifikasi tahapan *preprocessing* hingga proses mendeteksi *marker*.
3. Penambahan metode RANSAC pada aplikasi AR telah mampu meningkatkan performa aplikasi dalam mendeteksi *marker* yang tidak ideal. Hal ini dikarenakan mendeteksi *marker* dengan mencari tepian terlebih dahulu lebih baik dalam menemukan *marker* yang tidak ideal daripada langsung melakukan pencarian terhadap empat garis lurus yang saling berhubungan dari *marker*. Hasil pengujian performa aplikasi AR yang telah dilakukan, menunjukkan bahwa performa aplikasi rata-rata meningkat sebesar 50% dalam mendeteksi sepuluh kondisi *marker*, meningkat 40% dalam mendeteksi *marker* dengan tiga posisi penangkap citra, dan meningkat 38% dalam mendeteksi *marker* dengan lima kemiringan *marker*.

6.2 Saran

Beberapa saran untuk pengembangan lebih lanjut yang dapat diberikan penulis antara lain:

1. Untuk mendapatkan hasil yang lebih sempurna perlu adanya perbaikan proses *preprocessing* sehingga stabil untuk kondisi cahaya yang ekstrim.
2. Perlu adanya penyempurnaan proses pembuatan garis segi empat (menemukan *marker*) sehingga lebih stabil dalam mendeteksi *marker* tidak ideal dengan tingkat kemiringan tertentu.
3. Perlu adanya pengembangan lebih lanjut untuk mendeteksi *marker* tidak ideal lebih dari satu.
4. Hasil dari penelitian ini dapat dilanjutkan pada tahapan mengenali pola *marker* dan me-*render* objek 3D.

DAFTAR PUSTAKA

[ART-97] Azuma, Ronald T. 1997,"A Survey of *Augmented reality*", Presence: Teleoperators and Virtual Environments. vol. 6, no. 4, Aug.1997, pp.355-385.

[CJZ-96] Clarke, J. C., Zisserman, A. 1996, "Detection and Tracking of Independent Motion", Department of Engineering Science, University of Oxford, UK.

[MHZ-08] Hirzer, Martin. 2008, "*Marker* Detection For *Augmented reality* Applications", Inst. for Computer Graphics and Vision Graz University of Technology, Austria.

[RFD-08] Zulaini, Marco. 2008, "RANSAC for Dummies", Free Software Foundation.

[KOT-12] Hasan, B. N., WIrabuana, M., Sidiq, M. A.. 2012, "Pendeteksian Posisi *Marker* Pada Teknologi *Augmented reality*", Politeknik Negeri Bandung.

[MIL-94] Milgram, Paul. 1994, " A Taxonomy of Mixed Reality Virtual Diplays", IEICE Paper.

[NYA-12] Nyatla. 2012, NyARToolKit for C#, Akses dari http://nyatla.jp/nyartoolkit/wiki/index.php?NyARToolkit for C#. Tanggal akses 19 November 2012.

[HUD-12] Human Interface Technology Lab (HITLab). 1999, ARToolKit Documentation, Akses dari http://www.hitl.washington.edu/artoolkit/documentation/. Tanggal akses 3 November 2012.

[KOT-09] Koyama, Tomohiko. 2009, "Introduction to FLARToolKit",Makalah disajikan dalam *Future Innovation Technology Creativity Design & Technology Festival*, Toronto.

[HUP-12] Human Interface Technology Lab (HITLab). 1999, ARToolKit Paper. Akses dari http://www.hitl.washington.edu/artoolkit/Papers/. Tanggal akses 12 Oktober 2012.

[JKS-95] Jain, R., Kasturi, R., Schunck, B. G., 1995, "Machine Vision",

McGraw-Hill, Inc., ISBN 0-07-032018-7.

[GWS-07] Gonzales, R.C., Woods, R. E.. 2007, ”Digital Image Processing Third Edition”, dalam *Image Sampling and Quantization*, Prentice-Hall.

[PII-93] Pitas, Ioannis. 1993, “Digital Image Processing Algorithms”, Prentice-Hall.

[WSR-13] Wibirawa, Sunu., Teknik Optimasi Data Berbasis *Iterative Model Fitting*. RANSAC Paper. Akses dari http://academia.edu/Download/ Tanggal Akses 14 Januari 2013.

[COR-07] Collins, R. 2007, “*Introduction to Computer Vision*”, Fall 2007 Lecture Notes. CSE/EE486 Computer Vision I. CSE Department, Penn State University.

[HDC-13] Huttenlocher, D. “*RANSAC and some HOUGH transform*”. RANSAC Presentation. Cornell University. Akses dari http://cronos.rutgers.edu/~meer/TEACHTOO/RANSACPLUS-5.pdf Tanggal Akses 12 Desember 2013.