TESIS – EE185401

RANCANG BANGUN SISTEM PEMETAAN RUANG TERTUTUP MENGGUNAKAN KAMERA RGB-D

DANIEL KRISTIANTO HARYONO
07111650040002

DOSEN PEMBIMBING
Dr. Ir. Djoko Purwanto, M.Eng.
Dr. Ir. Hendra Kusuma, M.Eng.Sc.

PROGRAM MAGISTER
BIDANG KEAHLIAN TEKNIK ELEKTRONIKA
DEPARTEMEN TEKNIK ELEKTRO
FAKULTAS TEKNOLOGI ELEKTRO
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2019

TESIS – EE185401

RANCANG BANGUN SISTEM PEMETAAN RUANG TERTUTUP MENGGUNAKAN KAMERA RGB-D

DANIEL KRISTIANTO HARYONO
07111650040002

DOSEN PEMBIMBING
Dr. Ir. Djoko Purwanto, M.Eng.
Dr. Ir. Hendra Kusuma, M.Eng.Sc.

PROGRAM MAGISTER
BIDANG KEAHLIAN TEKNIK ELEKTRONIKA
DEPARTEMEN TEKNIK ELEKTRO
FAKULTAS TEKNOLOGI ELEKTRO
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2019

LEMBAR PENGESAHAN

Tesis disusun untuk memenuhi salah satu syarat memperoleh gelar
Magister Teknik (M.T.)
di
Institut Teknologi Sepuluh Nopember

oleh:

Daniel Kristianto Haryono
NRP. 07111650040002

Tanggal Ujian : 20 Desember 2018
Periode Wisuda : Maret 2019

Disetujui oleh:

1. Dr. Ir. Djoko Purwanto, M.Eng. (Pembimbing I)
   NIP: 196512111990021002
2. Dr. Ir. Hendra Kusuma, M.Eng.Sc. (Pembimbing II)
   NIP: 196409021989031003
3. Dr. Muhammad Rivai, S.T., M.T. (Penguji)
   NIP: 196904261994031003
4. Dr. Ir. Totok Mujiono, M.IKom. (Penguji)
   NIP: 196504221989031001
5. Muhammad Attamimi, B.Eng., M.Eng., Ph.D. (Penguji)
   NIP: 1985201711039

Dekan Fakultas Teknologi Elektro

Dr. Tri Arief Sardjono, S.T., M.T.
NIP. 197002121995121001

Halaman ini sengaja dikosongkan

PERNYATAAN KEASLIAN TESIS

Dengan ini saya menyatakan bahwa isi keseluruhan Tesis saya dengan judul "**RANCANG BANGUN SISTEM PEMETAAN RUANG TERTUTUP MENGGUNAKAN KAMERA RGB-D**" adalah benar-benar hasil karya intelektual mandiri, diselesaikan tanpa menggunakan bahan-bahan yang tidak diijinkan dan bukan merupakan karya pihak lain yang saya akui sebagai karya sendiri.

Semua referensi yang dikutip maupun dirujuk telah ditulis secara lengkap terdapat pada daftar pustaka. Apabila ternyata pernyataan ini tidak benar, saya bersedia menerima sanksi sesuai peraturan yang berlaku.

Surabaya, 18 Januari 2019

Daniel Kristianto Haryono
NRP. 07111650040002

Halaman ini sengaja dikosongkan

RANCANG BANGUN SISTEM PEMETAAN RUANG TERTUTUP MENGGUNAKAN KAMERA RGB-D

Nama mahasiswa : Daniel Kristianto Haryono
NRP : 07111650040002
Pembimbing : 1. Dr. Ir. Djoko Purwanto, M.Eng.
2. Dr. Ir. Hendra Kusuma, M.Eng.Sc.

ABSTRAK

Pemetaan dipandang sebagai salah satu hal yang menjadi kunci utama untuk membuat sebuah robot agar sepenuhnya bersifat otomatis. Apabila sebuah robot memiliki fitur pemetaan, maka robot tersebut dapat melakukan perencanaan rute dari suatu lokasi ke lokasi lainnya pada area kerja yang belum ditentukan sebelumnya, dimana hal tersebut merupakan kondisi yang umumnya ditemukan pada aplikasi nyata. Penelitian ini berfokus kepada penerapan kamera RGB-D untuk keperluan pembuatan peta pada ruang tertutup. Jenis kamera RGB-D yang digunakan pada penelitian ini adalah Kinect Xbox 360.

Sistem ini memiliki tiga bagian utama yaitu RGB-D *registration*, *camera pose estimation*, serta *laser scan based depth measurement*. Bagian RGB-D *registration* digunakan agar pada setiap piksel RGB terdapat nilai *depth* yang sesuai. Hal tersebut diperlukan karena letak dari sensor RGB dan *depth* pada Kinect tidak sama. Luaran dari proses tersebut akan digunakan oleh *camera pose estimation*. *Camera pose estimation* digunakan untuk mengestimasi *heading* (paralel terhadap lantai) serta pergerakan maju Kinect. Estimasi *heading* didapatkan dengan cara menghitung transformasi fitur-fitur *corner* yang didapat dari sekuen citra RGB, sedangkan estimasi pergerakan maju didapatkan dari perubahan informasi *depth* Kinect. Bagian *laser scan based depth measurement* digunakan untuk melakukan konversi informasi citra *depth* Kinect ke dalam representasi *laser scan* (LiDAR) sehingga dapat digunakan untuk proses pembuatan peta.

Pengujian dilakukan pada ruangan dengan dimensi 10.81 x 15.15 m. Evaluasi dilakukan dengan cara membandingkan luas area sebenarnya dengan luas yang diukur dari peta yang dibuat. Perbandingan yang dilakukan meliputi dua hal yaitu *uncovered area* dan *excessed area*. Dari hasil percobaan, algoritma FAST menghasilkan total *error* (*uncovered area* ditambah *excessed area*) sebesar 8.27 m^2, GFtT sebesar 8.18 m^2, dan Harris sebesar 8.54 m^2, terhadap luasan area aktual sebesar 27.873 m^2. Berdasarkan hasil percobaan tersebut, penggunaan algoritma GFtT memberikan performa yang lebih baik, yaitu nilai total *error* yang paling kecil, apabila dibandingkan dengan FAST maupun Harris.

Kata kunci: pemetaan, RGB-D, *pose estimation*, Kinect

Halaman ini sengaja dikosongkan

DESIGN AND IMPLEMENTATION OF CLOSED SPACE MAPPING SYSTEM UTILIZING RGB-D CAMERA

By : Daniel Kristianto Haryono
Student Identity Number : 07111650040002
Supervisor(s) : 1. Dr. Ir. Djoko Purwanto, M.Eng.
2. Dr. Ir. Hendra Kusuma, M.Eng.Sc.

ABSTRACT

Mapping is seen as one of the main objectives in-order to make a fully automatic robot. If a robot has mapping features, the robot can do route planning from one location to another in a undetermined work area, which is a condition commonly found in real applications. This research focuses on applying RGB-D in closed space mapping system. The RGB-D camera which has been used is Kinect Xbox 360.

This system has three main parts, namely RGB-D registration, camera pose estimation, and laser scan based depth measurement. The RGB-D registration section is used so that in each RGB pixel there is an appropriate depth value. This is necessary because the location of the RGB and depth sensor on the Kinect is different. The output of the process will be used by camera pose estimation. Camera pose estimation is used to estimate headings (parallel to the floor) and the forward movement of Kinect. The heading estimation is obtained by calculating the transformation of corner features obtained from the RGB image sequence, while the forward movement estimation is obtained from changes in Kinect depth information. The laser scan based depth measurement section is used to convert depth Kinect image information into laser scan representation (LiDAR) so that it can be used for the map making process.

Tests has been carried out in rooms with dimensions of 10.81 x 15.15 m. Evaluation is done by comparing the actual area with the area measured from the map made. Comparisons has been done considering two things, namely uncovered area and excessed area. From the results of the experiment, FAST algorithm gave a total error (uncovered area plus excessed area) of 8.18 m^2, GFtT a total error of 8.18 m^2, and Harris a total error of 8.54 m^2 on the actual area of 27.873 m^2. Based on the result, which is smaller total error, GFtT gave a better performance compared to FAST and Harris.

Keywords: mapping, RGB-D, pose estimation, Kinect

Halaman ini sengaja dikosongkan

KATA PENGANTAR

Puji syukur kepada Tuhan, yang selalu memberikan arahan-Nya, tesis ini dapat diselesaikan. Tesis berjudul "RANCANG BANGUN SISTEM PEMETAAN RUANG TERTUTUP MENGGUNAKAN KAMERA RGB-D" ini disusun untuk memenuhi sebagian persyaratan memperoleh gelar Magister Teknik (M.T.) pada jurusan Teknik Elektronika, fakultas Teknologi Elektro, Institut Teknologi Sepuluh Nopember.

Penulis menyadari bahwa dalam penyusunan tesis ini tidak terlepas dari bantuan berbagai pihak. Oleh karena itu, dengan segala ketulusan dan kerendahan hati penulis menyampaikan terima kasih kepada:

1. Ayah, Ibu, Kakak, dan seluruh keluarga yang telah memberikan dukungan serta motivasi untuk penulis dalam menempuh studi pascasarjana.
2. Bapak Dr. Ir. Djoko Purwanto, M.Eng. selaku dosen pembimbing I yang telah sabar, banyak memberikan saran, bantuan, bimbingan, serta motivasi kepada penulis untuk menyelesaikan penelitian.
3. Bapak Dr. Ir. Hendra Kusuma, M.Eng.Sc. selaku dosen pembimbing II yang telah sabar, banyak memberikan saran, bantuan, bimbingan, serta motivasi kepada penulis untuk menyelesaikan penelitian.
4. Bapak Dr. Achmad Arifin, S.T., M.Eng. selaku dosen wali penulis dan dosen pengajar jurusan Teknik Elektro yang telah banyak memberikan teladan-teladan.
5. Bapak Dr. Tri Arief Sardjono, S.T., M.T. selaku pengajar dan dekan fakultas Teknologi Elektro.
6. Bapak Dr. Muhammad Rivai, S.T., M.T. selaku pengajar dan koordinator program pascasarjana jurusan Teknik Elektro.
7. Bapak Dr. Ronny Mardiyanto, S.T., M.T. selaku dosen pengajar jurusan Teknik Elektro.
8. Bapak Dr. Astria Nur Irfansyah, S.T., M.Eng. selaku dosen pengajar jurusan Teknik Elektro.

9. Bapak Muhammad Atamimi, B.Eng., M.Eng., Ph.D. selaku dosen pengajar jurusan Teknik Elektro.
10. Bapak Dr. Ir. Totok Mujiono, M.IKom. selaku dosen pengajar jurusan Teknik Elektro.
11. Kepada Koordinator Pascasarjana Teknik Elektronika, koordinator pascasarjana jurusan Teknik Elektro dan ketua jurusan Teknik Elektro serta karyawan pascasarjana Teknik Elektro yang telah membantu penulis dalam segala urusan administrasi selama menempuh kuliah di ITS.
12. Citta Anindya selaku rekan yang selalu siap membantu dan memberikan semangat kepada penulis dalam menyelesaikan tugas kuliah dan tesis.
13. Rekan-rekan di Laboratorium B203, yang telah memberikan banyak saran serta menjadi teman bicara yang baik.
14. Rekan-rekan pascasarjana jurusan Teknik Elektro angkatan 2016 telah banyak membantu penulis dalam menyelesaikan tugas selama masa kuliah dan tesis ini.

Seluruh Civitas Akademisi Jurusan Teknik Elektro Fakultas Teknologi Industri Institut Teknologi Sepuluh Nopember Surabaya., atas segala bantuan yang telah diberikan.

Surabaya, 18 Januari 2019
Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

Halaman ini sengaja dikosongkan

BAB 1
PENDAHULUAN

1.1 Latar Belakang

Selama beberapa dekade terakhir, pengembangan teknologi robotika menarik banyak minat dari para peneliti di seluruh dunia. Hal tersebut ditandai dengan banyaknya gagasan-gagasan penemuan yang berkontribusi pada aplikasi robot di dunia nyata. Saat ini robot *mobile* dapat bekerja secara otomatis, dimana beberapa tahun silam masih diperlukan interaksi dan perintah secara langsung dari pengguna. Robot *mobile* memiliki rentang aplikasi yang sangat beragam, mulai dari dunia medis [1], militer [2], industri [3], hingga hiburan [4]. Salah satu contoh aplikasi nyata robot *mobile* telah diusulkan dan dikembangkan oleh Purwanto, Rivai, dan Soebhakti [5]. Pada contoh-contoh aplikasi yang telah disebutkan, robot *mobile* diharuskan untuk memiliki kemampuan navigasi yang kompleks tanpa perintah dari pengguna. Agar dapat melakukan navigasi serta merencanakan rute perjalanan yang optimal dari satu lokasi ke lokasi yang lain, maka robot memerlukan peta dari area kerjanya, di mana hal tersebut umumnya tidak tersedia. Maka dari itu, agar robot dapat bekerja secara otomatis akan dibutuhkan kemampuan untuk membuat peta.

Pemetaan dipandang sebagai salah satu hal yang menjadi kunci utama untuk menjadikan sebuah robot sepenuhnya bersifat otomatis [6]. Proses kerja pemetaan secara umum memanfaatkan dua buah jenis sensor, yaitu sensor jarak yang umumnya memanfaatkan sinar laser, serta *odometer* (umumnya berbasis *encoder*) yang akan dipasang pada roda robot. Sensor jarak akan digunakan untuk mengetahui jarak robot ke dinding atau halangan-halangan lainnya, sedangkan *odometer* akan digunakan untuk memperkirakan lokasi robot. Adapun penggunaan *encoder* pada roda robot memiliki aspek mekanis yang banyak, sehingga akan banyak pula hal-hal mekanis yang berpotensi untuk menjadi *noise*. Beberapa contoh batasan mekanis yang dimaksudkan adalah seperti [7]:

1. Teknik tersebut hanya dapat digunakan pada robot beroda,
2. Bentuk roda yang tidak simetris,

3. Terjadinya selip, lintasan yang tidak rata, dll.,
4. *Error* yang terjadi pada pengukuran *jarak* akan bersifat akumulatif.

Salah satu penelitian mengenai pemetaan sebelumnya telah dilakukan [8]. Penelitian tersebut menggunakan Kinect (kamera RGB-D) dan *encoder* yang dipasang pada roda robot. Adapun Kinect digunakan untuk mengetahui jarak robot dari halangan-halangan yang ada sehingga dapat dibuat sebuah peta, sedangkan estimasi pergerakan robot didapatkan melalui data dari *encoder*. Pada penelitian ini akan diusulkan penerapan Kinect yang sekaligus digunakan untuk mengestimasi pergerakan robot, sehingga menggantikan peran dari *encoder*. Dengan teknik ini diharapkan proses pemetaan terbebas dari batasan-batasan mekanik yang telah disebutkan sebelumnya.

1.2 Rumusan Masalah

Berdasarkan dari latar belakang yang telah disampaikan, maka penulis membuat perumusan masalah yaitu bagaimana membuat sistem pemetaan dengan memanfaatkan kamera RGB-D pada ruang tertutup untuk menjadi solusi bagi batasan-batasan penerapan *encoder*.

1.3 Tujuan

Tujuan diadakannya penelitian ini adalah untuk memberikan solusi dari batasan-batasan mekanis yang timbul pada penerapan odometri berbasis *encoder*, untuk keperluan pemetaan.

1.4 Batasan Masalah

Batasan masalah pada penelitian ini adalah bahwa area kerja memiliki fitur *corner* lebih dari 5 poin, pencahayaan ruang yang baik, serta jarak untuk setiap citra *depth* yang ditangkap dapat ditentukan.

.

1.5 Kontribusi

Manfaat kedepan yang diharapkan oleh penulis dari penelitian ini adalah, teknik pemetaan ruang tertutup menggunakan kamera RGB-D dapat digunakan

untuk menggantikan peran dari *encoder* pada sistem pemetaan. Dengan demikian, kelemahan-kelemahan pemetaan berbasis *encoder* yang sebelumnya telah disebutkan dapat diatasi. Hilangnya batasan bahwa jenis robot haruslah beroda juga akan membuka kesempatan pengembangan penelitian lainnya seperti penerapan teknik pemetaan pada robot berkaki, robot *aerial*, dll.

1.6 Metodologi Penelitian

Pada bagian ini akan disampaikan blok utama sistem secara umum, yaitu sistem pemetaan para ruang tertutup memanfaatkan kamera RGB-D. Blok tersebut kemudian diuraikan menjadi bagian-bagian sederhana yang berfungsi sebagai komponen-komponen penyusun utama.

Halaman ini sengaja dikosongkan

BAB 2
KAJIAN PUSTAKA

2.1 Kajian Penelitian Terkait

Beberapa penelitian mengenai sistem lokalisasi dan pemetaan telah dilakukan menggunakan berbagai metode, di mana salah satunya adalah memanfaatkan sensor jarak infra merah [9]. Penggunaan sensor jarak yang memanfaatkan infra merah, sonar, ataupun laser dapat memberikan hasil yang cukup baik. Namun apabila fitur-fitur yang didapatkan oleh sensor jarak dibandingkan dengan fitur-fitur yang ada pada sebuah citra, maka jumlah fitur-fitur yang dapat dideteksi pada sebuah citra akan jauh lebih banyak. Dengan memanfaatkan citra, selain jarak dengan obyek, indentifikasi obyek pun dapat dilakukan. Selain itu, pada permasalahan pemetaan dan lokalisasi, semakin banyak fitur yang dapat dideteksi akan membuat proses estimasi posisi kamera menjadi lebih akurat [10].

Selain penelitian yang telah disebutkan, telah dilakukan pula penelitian yang memanfaatkan sensor *Inertial Measurement Unit* (IMU) serta *encoder* [11]. *Encoder* pada penelitian tersebut dipasang pada gulungan kabel yang kemudian kabelnya akan ditarik oleh robot pada saat berjalan menelusuri pipa. Dapat ditarik kesimpulan bahwa *encoder* tidak berkaitan langsung dengan aspek mekanik dari robot. Apabila terdapat selip pada roda robot, panjang kabel yang ditarik pun tidak bertambah. Apabila roda robot tidak simetris, hal tersebut juga tidak berpengaruh pada pembacaan jarak tempuh robot oleh *encoder*. Lain halnya dengan penggunan *encoder*, yang umumnya dipasang pada roda robot. Apabila terjadi selip, maka robot akan dianggap bergerak, namun kenyataannya adalah sebaliknya.

Berdasarkan hal-hal di atas, maka saat ini fokus penelitian sistem lokalisasi dan pemetaan banyak beralih pada penerapan metode odometri visual. Beberapa penelitian yang telah dilakukan adalah seperti penggunaan kamera monokuler [12] dan kamera stereo [13]. Penelitian ini bertujuan untuk mengembangkan metode odometri visual yang telah digunakan pada penelitian-penelitian sebelumnya, yaitu memanfaatkan kamera RGB-D. Dengan menggunakan kamera stereo, bisa

didapatkan informasi *depth* melalui proses *triangulation*. Namun karena sifat dari kamera stereo adalah pasif, untuk mendapatkan informasi *depth* diperlukan tekstur visual pada citra yang ditangkap. Lain halnya dengan kamera RGB-D yang bersifat aktif. Kamera RGB-D menggunakan cahaya laser terstruktur untuk proses perhitungan *depth*, sehingga pembacaan informasi *depth* tidak tergantung oleh tekstur visual.

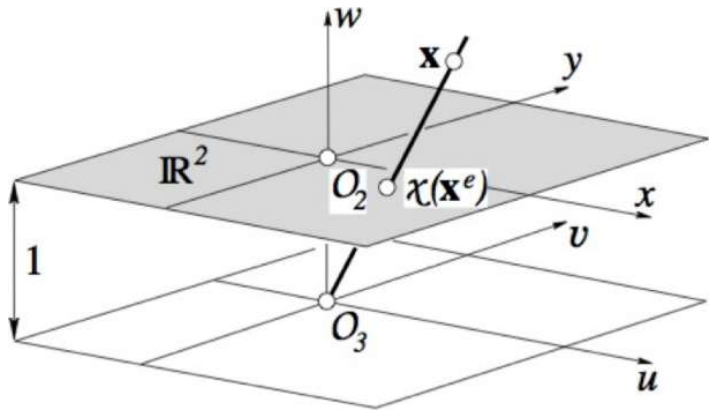
2.2 Koordinat Homogen

Homogeneous coordinates atau dalam Bahasa Indonesia disebut sebagai koordinat homogen merupakan sistem koordinat yang digunakan dalam permasalahan *projective geometry*. Dua alasan mengapa koordinat homogen digunakan adalah [14]:

1. Suatu poin pada titik tak terbatas (*infinity*) dapat dituliskan pada koordinat homogen,
2. Transformasi *affine* dan transformasi *projective* dapat diwakili oleh sebuah matrik saja.

Representasi dari sebuah titik **x** akan bersifat homogen (sama) apabila **x** dan $\lambda\mathbf{x}$ merepresentasikan titik yang sama untuk $\lambda \neq 0$. Hal tersebut dapat dirumuskan oleh (2.1).

$$\mathbf{x}_{\mathrm{H}} = \begin{bmatrix} u \\ v \\ \lambda \end{bmatrix} = \begin{bmatrix} u/\lambda \\ v/\lambda \\ 1 \end{bmatrix} \rightarrow \mathbf{x}_C = \begin{bmatrix} u/\lambda \\ v/\lambda \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} \tag{2.1}$$



Gambar 2.1 Relasi Koordinat Homogen dan Koordinat Cartesian [14]

Di mana u, v, dan λ merupakan koordinat homogen dari suatu titik **x**, sedangkan x dan y merupakan koordinat Cartesian-nya. Persamaan (2.1) dapat diilustrasikan oleh Gambar 2.1. Hal yang sama juga berlaku untuk sebuah titik pada bidang 3D menggunakan (2.2).

$$\mathbf{x}_H = \begin{bmatrix} u \\ v \\ w \\ \lambda \end{bmatrix} = \begin{bmatrix} u/\lambda \\ v/\lambda \\ w/\lambda \\ 1 \end{bmatrix} \rightarrow \mathbf{x}_C = \begin{bmatrix} u/\lambda \\ v/\lambda \\ w/\lambda \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{2.2}$$

2.3 Transformasi Geometri pada Koordinat Homogen

Pada koordinat Cartesian, persamaan transformasi *projective* dapat dituliskan sebagai (2.3), di mana A, B, C, D, E, F, a, b, dan c merupakan konstanta-konstanta untuk transformasi *projective*, x dan y merupakan koordinat Cartesian awal dari sebuah titik, sedangkan x' dan y' merupakan koordinat Cartesian hasil transformasinya.

$$\mathbf{x}'_C = \begin{bmatrix} x' \\ y' \end{bmatrix} = \left(\begin{bmatrix} A & B \\ D & E \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} C \\ F \end{bmatrix} \right) \frac{1}{\left(\begin{bmatrix} a & b \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + c \right)} \tag{2.3}$$

Apabila koordinat Cartesian suatu titik pada bidang 2D diubah ke dalam sistem koordinat homogen, maka (2.3) dapat diubah menjadi (2.4).

$$\mathbf{x}_C = \begin{bmatrix} x \\ y \end{bmatrix} \rightarrow \mathbf{x}_H = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\mathbf{x}'_H = \begin{bmatrix} x'\lambda' \\ y'\lambda' \\ \lambda' \end{bmatrix} = \begin{bmatrix} A & B & C \\ D & E & F \\ a & b & c \end{bmatrix} \mathbf{x}_H = \begin{bmatrix} A & B & C \\ D & E & F \\ a & b & c \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} Ax + By + C \\ Dx + Ey + F \\ ax + by + c \end{bmatrix} \tag{2.4}$$

Hasil dari (2.4) merupakan koordinat homogen, dan dapat diubah ke sistem koordinat Cartesian dengan cara membagi hasil tersebut dengan λ'.

$$\mathbf{x}'_H = \begin{bmatrix} x'\lambda' \\ y'\lambda' \\ \lambda' \end{bmatrix} = \begin{bmatrix} Ax + By + C \\ Dx + Ey + F \\ ax + by + c \end{bmatrix} = \begin{bmatrix} \frac{Ax+By+C}{ax+by+c} \\ \frac{Dx+Ey}{ax+by+c} \\ 1 \end{bmatrix} \rightarrow \mathbf{x}'_C = \begin{bmatrix} \frac{Ax+By+C}{ax+by+c} \\ \frac{Dx+Ey+F}{ax+b} \end{bmatrix} \tag{2.5}$$

Melalui persamaan (2.4), proyeksi *affine* dan *projective* dapat direpresentasikan dengan menggunakan 1 matrik saja sehingga relatif menyederhanakan baris kode program serta mempercepat waktu komputasi.

2.4 Matrik *Essential*

Pada konsep *stereo vision* orientasi relatif antara dua buah kamera diketahui sebelumnya, sehingga dapat ditentukan matrik transformasi antara citra dari kamera kiri dan kanan. Lain halnya pada *mono vision*, di mana orientasi pose relatif terhadap pose awal kamera tidak diketahui. Persamaan dari dua buah pose *mono vision* dapat dituliskan sebagai (2.6) [15].

$$\lambda \mathbf{p}_H = \mathbf{A}[\mathbf{I}|\mathbf{0}]\mathbf{P}_H = \mathbf{A}\mathbf{P}_H, \text{ untuk pose pertama kamera}$$
$$\lambda' \mathbf{p}'_H = \mathbf{A}'[\mathbf{R}|\mathbf{t}]\mathbf{P}_H, \text{ untuk pose kedua kamera} \quad (2.6)$$

dengan:

- λ: faktor scaling koordinat homogen
- $\mathbf{p}_H$: koordinat homogen dari titik pada citra (2D)
- $\mathbf{P}_H$: koordinat homogen dari titik pada dunia nyata (3D)
- $\mathbf{R}$: matrik rotasi
- $\mathbf{t}$: matrik translasi
- $\mathbf{A}$: matrik intrinsik kamera 1
- $\mathbf{A}'$: matrik intrinsik kamera 2

adapun pada penelitian ini pengambilan citra akan dilakukan menggunakan kamera yang sama sehingga

$$\mathbf{A}' = \mathbf{A}$$

jika $\mathbf{x}$ dan $\mathbf{x}'$ merupakan $\mathbf{p}_H$ dan $\mathbf{p}'_H$ yang ternormalisasi

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{p}_H = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \text{ dan } \mathbf{x}' = \mathbf{A}^{-1}\mathbf{p}'_H = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$$

maka (2.6) dapat ditulis kembali sebagai berikut

$$\lambda \mathbf{x} = \mathbf{P}_H, \text{ untuk posisi pertama kamera}$$
$$\lambda' \mathbf{x}' = [\mathbf{R}|\mathbf{t}]\mathbf{P}_H = \mathbf{R}\mathbf{P}_H + \mathbf{t}$$
$$\lambda' \mathbf{x}' = [\mathbf{R}|\mathbf{t}]\mathbf{P}_H = \lambda \mathbf{R}\mathbf{x} + \mathbf{t}, \text{ untuk posisi kedua kamera} \quad (2.7)$$

Untuk menghilangkan $\mathbf{t}$ pada sisi kanan (2.7), maka kedua sisi dapat dikalikan dengan matrik *skew-symmetric* $[\mathbf{t}]_\times$ di mana

$$[\mathbf{t}]_\times = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix}$$

sehingga

$$\lambda' [\mathbf{t}]_\times \mathbf{x}' = \lambda ([\mathbf{t}]_\times \mathbf{R})\mathbf{x} \quad (2.8)$$

dan jika (2.8) dikalikan dengan transpose dari $\mathbf{x}'$, maka dapat ditulis

$$\lambda' \mathbf{x}'^{\mathrm{T}} ([\mathbf{t}]_\times \mathbf{x}') = \lambda \mathbf{x}'^{\mathrm{T}} ([\mathbf{t}]_\times \mathbf{R})\mathbf{x} \quad (2.9)$$

Karena $\mathbf{x}'^{\mathrm{T}} ([\mathbf{t}]_\times \mathbf{x}') = 0$, maka $\mathbf{x}'^{\mathrm{T}} ([\mathbf{t}]_\times \mathbf{R})\mathbf{x} = 0$, sehingga dapat dituliskan

$$\mathbf{x}'^{\mathrm{T}} \mathbf{E} \mathbf{x} = \mathbf{0} \quad (2.10)$$

di mana $\mathbf{E} = [\mathbf{t}]_\times \mathbf{R}$ disebut sebagai matrik *essential*. Dengan tujuan untuk menemukan nilai $\mathbf{E}$ dari korespondensi antara titik $\mathbf{x}$ dan $\mathbf{x}'$, (2.10) dapat diturunkan sebagai berikut

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} \begin{bmatrix} E_{11} & E_{12} & E_{13} \\ E_{21} & E_{22} & E_{23} \\ E_{31} & E_{32} & E_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \mathbf{0}$$

sehingga

$$[xx'E_{11} + yx'E_{12} + x'E_{13} + xy'E_{21} + yy'E_{22} + y'E_{23} + xE_{31} + yE_{32} + E_{33}] = \mathbf{0}$$

$$[xx' \quad yx' \quad x' \quad xy' \quad yy' \quad y' \quad x \quad y \quad 1] \begin{bmatrix} E_{11} \\ E_{12} \\ E_{13} \\ E_{21} \\ E_{22} \\ E_{23} \\ E_{31} \\ E_{32} \\ E_{33} \end{bmatrix} = \mathbf{0}$$

$$\mathbf{D}.\mathbf{E} = \mathbf{0} \tag{2.11}$$

di mana (2.11) merupakan persamaan homogen. Apabila jumlah korespondensi titik $\geq$ 8, maka nilai dari **E** dapat ditentukan menggunakan *Singular Value Decomposition* (SVD) [15].

Karena **E** merupakan hasil perkalian matrik *skew-symmetric* $[\mathbf{t}]_\times$ dengan matrik **R**, maka transformasi rotasi dan translasi bisa didapatkan melalui dekomposisi matrik **E**. Proses dekomposisi matrik **E** dapat dilakukan menggunakan SVD namun dengan memperhitungkan batasan khusus yaitu dua *singular value*-nya bernilai sama, dan *singular value* ke tiga bernilai nol [16].

2.5 *Singular Value Decomposition*

SVD merupakan faktorisasi sebuah matrik **A** ke dalam bentuk $\mathbf{U\Sigma V}^{\mathrm{T}}$, di mana **U** merupakan matrik orthogonal $m \times m$, **Σ** merupakan matrik diagonal $m \times n$ yang memiliki angka positif untuk masing-masing elemen diagonalnya, dan **V** merupakan matrik orthogonal $n \times n$ [17].

$$\mathbf{A} = \mathbf{U\Sigma V}^{\mathrm{T}} \tag{2.12}$$

di mana

$$\mathbf{U}^{\mathrm{T}}\mathbf{U} = \mathbf{V}^{\mathrm{T}}\mathbf{V} = \mathbf{I}$$

Kolom dari matrik **U** merupakan *eigenvector orthonormal* dari $\mathbf{AA}^{\mathrm{T}}$, sedangkan kolom dari matrik **V** merupakan *eigenvector orthonormal* dari $\mathbf{A}^{\mathrm{T}}\mathbf{A}$, sedangkan **Σ** merupakan matrik diagonal yang berisi nilai akar dari *eigenvalue* **U** atau **V** dengan urutan *descending*.

SVD memiliki banyak aplikasi di mana salah satunya adalah untuk menyelesaikan sistem persamaan homogen seperti (2.11). Dengan menggunakan (2.12), nilai **E** dari dapat ditentukan sebagai *eigenvector* dari **D** dengan nilai *eigenvalue* terkecil dari matrik **V**.

2.6 Estimasi Matrik Essential

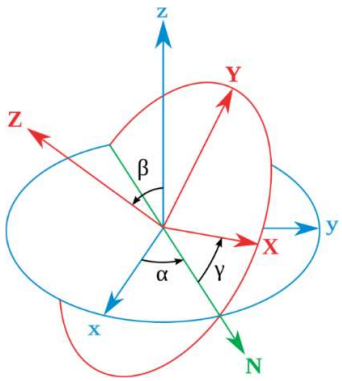
Pada penelitian ini, proses estimasi matrik *essential* akan dilakukan menggunakan MATLAB. Pada MATLAB terdapat fungsi `estimateEssentialMatrix` [18], di mana proses estimasi memerlukan jumlah korespondensi titik ≥ 5. Fungsi tersebut menggunakan algoritma yang diusulkan oleh Nister [19].

2.7 Matrik R dan t dari Matrik Essential

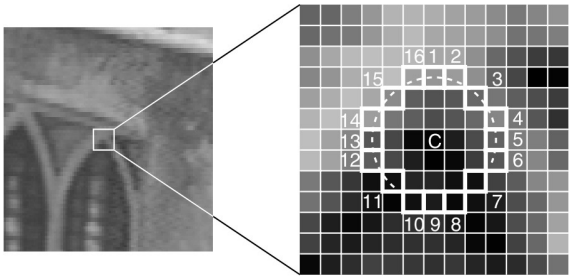
Proses untuk mendapatkan matrik **R** dan **t** dapat dilakukan menggunakan SVD dengan beberapa batasan khusus. Pada penelitian ini, proses untuk mendapatkan matrik **R** dan **t** dari matrik *essential* akan dilakukan menggunakan salah satu fungsi MATLAB yaitu `relativeCameraPose` [18].

2.8 Representasi Euler untuk R

Matrik rotasi **R** dapat direpresentasikan ke dalam bentuk Euler, di mana representasi Euler biasa digunakan untuk menggambarkan orientasi suatu obyek terhadap suatu sistem koordinat acuan. Ilustrasi dari sudut Euler terdapat pada Gambar 2.2 di bawah.



Gambar 2.2 Ilustrasi Sudut Euler [21]



Gambar 2.3 Ilustrasi Pendeteksian fitur *corner* FAST [22]

Proses konversi **R** ke dalam bentuk Euler dapat dilakukan mengikuti langkah-langkah yang dideskripsikan oleh Mallick pada artikelnya [20]. Adapun pada penelitian ini proses konversi akan dilakukan menggunakan fungsi MATLAB yaitu `rotm2eul` [18], di mana fungsi ini memberikan *output* yang sama.

2.9 Fitur *Corner* FAST

Sebuah *corner* dapat didefinisikan sebagai titik sambungan dari dua buah *edge* yang berasal dari arah yang berbeda, sedangkan *edge* sendiri merupakan perubahan drastis dari kecerahan pada sebuah citra. Fitur *corner* merupakan fitur yang tahan terhadap translasi, rotasi, serta iluminasi. Fitur *corner* banyak dimanfaatkan pada aplikasi-aplikasi seperti *motion tracking*, *image stitching*, *stereo vision*, dll.

FAST merupakan kependekan dari *Features from Accelerated Segment Test*. FAST digagas untuk mengatasi permasalahan di mana proses algoritma pendeteksian fitur *corner* yang ada pada saat itu tidak cukup cepat untuk diaplikasikan secara *real-time*. Adapun algoritma pendeteksian fitur *corner* FAST dapat diringkas sebagai berikut [22]:

1. Pilih sebuah piksel **p** pada citra *grayscale* yang akan dideteksi apakah merupakan sebuah *corner* atau tidak. Intensitas dari **p** akan disebut sebagai $I_{\mathbf{p}}$,
2. Tentukan sebuah nilai batas t,
3. Seperti yang diilustrasikan pada Gambar 2.3, 16 piksel yang mengelilingi **p** akan diperiksa,

4. Piksel **p** akan dinyatakan sebagai *corner* apabila terdapat **n** piksel kontinu pada 16 piksel yang mengelilinginya, yang semuanya memiliki intensitas lebih terang dari $I_{\mathbf{p}} + t$ atau intensitas lebih gelap dari $I_{\mathbf{p}} - t$. Nilai dari **n** yang dipilih adalah 12.

Sebelum poin 4 dilakukan, *high speed test* untuk **p** dapat dilakukan untuk mengabaikannya jika tidak memiliki kemungkinan untuk dinyatakan sebagai *corner*. *High speed test* dilakukan dengan cara mengamati 4 buah piksel yaitu 1, 9, 5, 13. Apabila piksel 1 dan 9 memiliki intensitas lebih terang dari $I_{\mathbf{p}} + t$ atau lebih gelap dari $I_{\mathbf{p}} - t$, cek piksel 5 dan 13. Apabila **p** merupakan *corner*, maka setidaknya 3 piksel akan memenuhi kriteria tersebut. Pada penelitian ini, proses pendeteksian fitur corner FAST akan dilakukan menggunakan fungsi MATLAB yaitu `detectFASTFeatures` [18].

2.10 *Optical Flow*

Pada penelitian ini, teknik *optical flow* Lucas-Kanade akan digunakan untuk mencari korespondensi fitur *corner* antara citra $n-1$ dengan citra n. Penjelasan sederhana dari cara kerja teknik ini adalah sebagai berikut. Seperti pada Gambar 2.3, asumsikan bahwa kita melihat suatu *scene* melalui sebuah lubang kotak. Nilai intensitas awal, yaitu a, yang kita amati melalui lubang kotak tersebut akan bervariasi. Jika nilai intensitas naik menjadi b, maka dapat diasumsikan terdapat pergerakan ke kiri dan ke atas.

Apabila penambahan intensitas suatu piksel pada sumbu x adalah $I_x(x, y)$ dan pada sumbu y adalah $I_y(x, y)$, total penambahan intensitas dari pergerakan u pada sumbu x dan pergerakan v pada sumbu y adalah $I_t(x, y)$, yang dapat dituliskan sebagai (2.13) [23].

$$I_x(x, y) \times u + I_y(x, y) \times v = (b - a) = -I_t(x, y) \tag{2.13}$$

Pada penerapannya, estimasi pergerakan u dan v akan sulit dilakukan melalui pengamatan sebuah piksel saja. Maka dari itu sebuah piksel akan diamati bersama dengan tetangganya, sebagai contoh matrik $3x3$ dengan piksel yang

hendak diamati pergerakannya berada di tengah matrik, sehingga (2.13) dapat dituliskan sebagai (2.14).

$$\begin{bmatrix} I_x(x-1,y-1) & I_y(x-1,y-1) \\ I_x(x-1,y) & I_y(x-1,y) \\ I_x(x-1,y+1) & I_y(x-1,y+1) \\ I_x(x,y-1) & I_y(x,y-1) \\ I_x(x,y) & I_y(x,y) \\ I_x(x,y+1) & I_y(x,y+1) \\ I_x(x+1,y-1) & I_y(x+1,y-1) \\ I_x(x+1,y) & I_y(x+1,y) \\ I_x(x+1,y+1) & I_y(x+1,y+1) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -I_t(x-1,y-1) \\ -I_t(x-1,y) \\ -I_t(x-1,y+1) \\ -I_t(x,y-1) \\ -I_t(x,y) \\ -I_t(x,y+1) \\ -I_t(x+1,y-1) \\ -I_t(x+1,y) \\ -I_t(x+1,y+1) \end{bmatrix}$$

$$\mathbf{S}\begin{bmatrix} u \\ v \end{bmatrix} = \mathbf{t} \tag{2.14}$$

karena (2.14) merupakan *overdetermined system*, maka pendekatan nilai u dan v dapat dicari menggunakan metode *ordinary least squares* sehingga dapat dituliskan kembali menjadi (2.15).

$$\mathbf{S}^{\mathrm{T}}\mathbf{S}\begin{bmatrix} u \\ v \end{bmatrix} = \mathbf{S}^{\mathrm{T}}\mathbf{t} \rightarrow \begin{bmatrix} u \\ v \end{bmatrix} = (\mathbf{S}^{\mathrm{T}}\mathbf{S})^{-1}\mathbf{S}^{\mathrm{T}}\mathbf{t} \tag{2.15}$$

Pada penelitian ini, optical flow Lucas-Kanade akan dilakukan menggunakan *object* MATLAB yaitu `vision.PointTracker` [18].

2.11 Kinect *for* Xbox 360

Kinect *for* Xbox 360 (selanjutnya akan disebut sebagai Kinect) merupakan modul *motion sensing* yang dibuat oleh Microsoft sebagai perangkat *input game console* Xbox. Modul ini memiliki dua buah sensor utama, yaitu sensor untuk citra berwarna dan sensor untuk menerima pantulan cahaya IR. Kinect yang diluncurkan oleh Microsoft pada tahun 2010 mendapat banyak perhatian tidak hanya pada kalangan *gamers* namun juga pada kalangan akademisi. Bagi para *gamers*, Kinect menyediakan metode *input* yang baru untuk bermain *game* yaitu *Natural Interaction* (NI). Bagi kalangan akademisi, Kinect merupakan suatu produk yang menarik karena pada saat itu modul kamera dengan fungsi serupa memiliki harga yang cukup tinggi. Kinect memiliki tiga komponen utama, yaitu proyektor IR,

sensor CMOS untuk citra RGB, dan sensor CMOS untuk citra *depth*. Ilustrasi dari Kinect terdapat pada Gambar 2.4 (a).

Secara sederhana, Kinect bekerja dengan prinsip yang hampir sama dengan kamera *Time-of-Flight* (ToF). Proyektor IR Kinect akan memancarkan cahaya, yang kemudian akan dipantulkan oleh obyek sehingga cahaya tersebut diterima kembali oleh sensor. Dengan mengetahui bahwa kecepatan cahaya adalah $\sim 3 \times 10^8$ $^m/_s$, maka jarak obyek dari Kinect dapat diestimasi.

Hal tersebut akan mudah dilakukan apabila hanya satu titik saja yang hendak diketahui nilai *depth*-nya. Namun pada aplikasinya, Kinect dapat mengetahui nilai *depth* untuk masing-masing piksel RGB. Hal tersebut dapat dilakukan karena proyektor IR Kinect memancarkan titik-titik berpola tertentu (*light-coded*). Pola tersebut ditangkap oleh sensor CMOS untuk kemudian dikenali, sehingga dapat dilakukan korespondensi antara pancaran proyektor IR dengan hasil yang ditangkap oleh sensor CMOS. Setelah korespondensi didapatkan, maka dapat dilakukan proses triangulation untuk mendapatkan informasi *depth* [25]. Contoh Proyeksi IR oleh Kinect terdapat pada Gambar 2.4 (b).

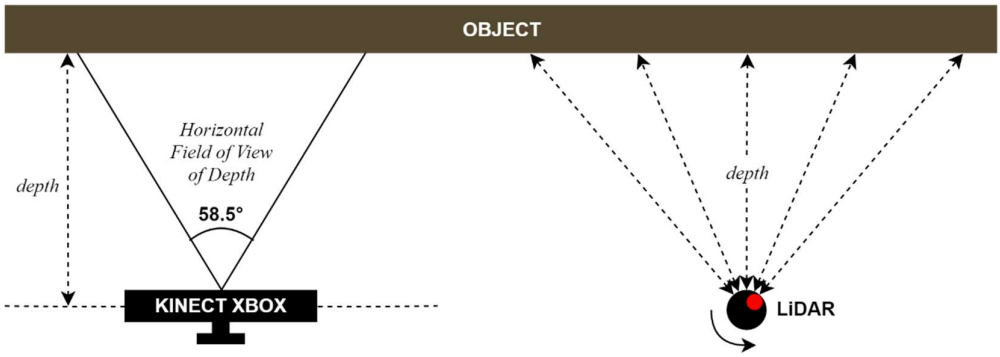
2.12 Pengukuran *Depth* pada Kinect

Hasil pengukuran jarak oleh Kinect disebut juga sebagai *depth*, di mana *depth* berbeda dengan hasil pengukuran jarak oleh LiDAR. Depth merupakan jarak yang diukur paralel dari tubuh Kinect, sedangkan jarak dari LiDAR merupakan jarak melingkari *transceiver* sinar *laser*. Ilustrasi perbedaan *depth* dari Kinect dengan jarak dari LiDAR terdapat pada Gambar 2.5. Untuk keperluan pembuatan peta yang akan dilakukan menggunakan fungsi MATLAB, maka perlu dilakukan konversi data *depth* dari Kinect ke dalam bentuk jarak LiDAR.

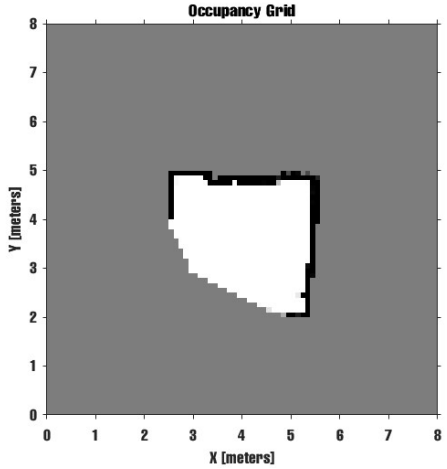
Gambar 2.4 Ilustrasi Sederhana Konsep *Optical Flow* [23]

(a) (b)

Gambar 2.5 (a) Kinect *for* Xbox 360 [24]; (b) Proyeksi sinar IR oleh Kinect [24]



Gambar 2.6 Perbedaan Pengukuran *Depth* pada Kinect dan Jarak pada LiDAR



Gambar 2.7 Ilustrasi hasil *occupancy grid map*

Dengan mengetaui *field of view* dari sensor *depth* Kinect [26], serta resolusi citra *depth* yang ditangkap, maka dapat diketahui nilai derajat untuk masing-masing piksel citra *depth* terhadap *field of view*. Dengan demikian proses konversi format *depth* Kinect ke format *depth laser scan* dapat dilakukan.

2.13 *Occupancy Grid Map*

Occupancy grid map merupakan algoritma yang digunakan untuk membuat peta berdasarkan dari pengukuran sensor jarak. Teknik ini memiliki asumsi bahwa kondisi lingkungan yang akan dipetakan adalah statis. Peta yang akan dibuat merupakan kumpulan dari *grid-grid* dengan ukuran tertentu (resolusi), di mana masing-masing grid akan memiliki nilai probabilitas apakah dalam keadaan *occupied* (ditempati) dengan warna hitam atau *free* (kosong) dengan warna putih. Warna abu-abu merupakan daerah pada peta yang masih belum dapat dipastikan apakah dalam keadaan *occupied* atau *free*.

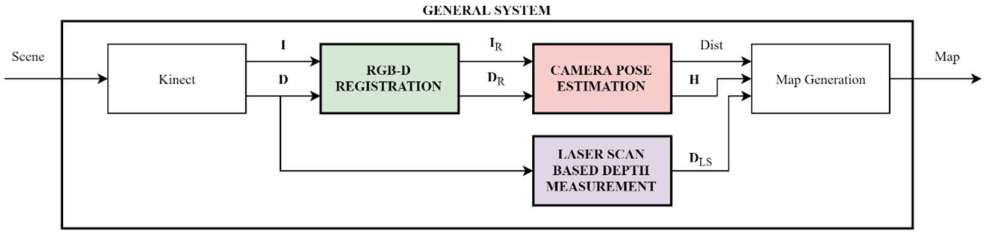
Pada penelitian ini proses pembuatan *occupancy grid map* dilakukan menggunakan bantuan fungsi dari MATLAB di mana hasil tersebut dapat diilustrasikan pada Gambar 2.7.

Halaman ini sengaja dikosongkan

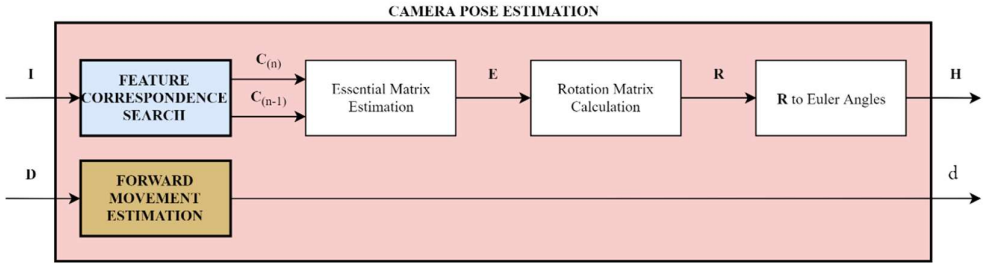
BAB 3
METODOLOGI PENELITIAN

Sistem pada penelitian ini membutuhkan tiga hal utama yakni, kemampuan untuk mengestimasi sudut *heading* dari kamera, kemampuan untuk mengestimasi pergerakan maju kamera, serta kemampuan untuk membuat peta dari sekuen citra RGB dan *depth*. Diagram blok dari sistem yang dirancang secara umum diilustrasikan oleh Gambar 3.1.

Proses estimasi sudut *heading* dan pergerakan maju kamera dirangkum dalam satu blok yaitu *camera pose estimation*. Pada blok tersebut sudut *heading* diestimasi melalui tiga tahap yaitu, mendeteksi fitur *corner* FAST ($\mathbf{C}_{(n-1)}$) pada citra RGB ke-$(n-1)$, mencari korespondensi fitur *corner* FAST ($\mathbf{C}_{(n)}$) pada citra RGB ke-n, kemudian mengestimasi matrik transformasi antara $\mathbf{C}_{(n-1)}$ dan $\mathbf{C}_{(n)}$. Sedangkan perkiraan pergerakan maju kamera akan diestimasi berdasarkan selisih nilai *depth* pada saat $(n-1)$ dengan (n).



Gambar 3.1 Diagram Blok Umum



Gambar 3.2 Diagram Blok *Camera Pose Estimation*

Setelah sudut *heading* dan pergerakan kamera diestimasi, maka bagian berikutnya adalah pembuatan peta. Adapun proses pembuatan peta dilakukan

dengan bantuan fungsi MATLAB, yaitu `insertRay`, di mana salah parameter *input*-nya merupakan data *depth* dalam format *laser scan*, sehingga diperlukan proses konversi pada citra *depth* dari Kinect agar memiliki format yang sesuai.

3.1 RGB-D *Registration*

Karena sensor RGB dan *depth* pada Kinect terletak pada posisi yang berbeda, maka terdapat perbedaan (pergeseran) pada hasil *capture* citra RGB dan citra *depth*. Proses RGB-D *registration* adalah proses transformasi citra RGB ke *depth* maupun sebaliknya. Proses tersebut diawali dengan proses kalibrasi untuk masing-masing sensor, sehingga jika parameter keduanya didapatkan, maka transformasi citra pun dapat dilakukan. Hasil dari proses ini adalah pada masing-masing piksel di citra RGB terdapat nilai *depth* yang sesuai. Proses RGB-D *registration* dilakukan dengan bantuan fungsi MATLAB, yaitu `pcfromkinect`. Fungsi tersebut tidak melakukan proses kalibrasi, karena MATLAB menggunakan parameter sensor yang telah ditentukan pada saat proses produksi Kinect.

3.2 *Camera Pose Estimation*

Detil dari bagian *camera pose estimation* diilustrasikan oleh Gambar 3.2. Blok tersebut membutuhkan dua buah *input* yaitu citra RGB dan citra *depth* dari Kinect. *Output* dari *camera pose estimation* ada dua, yaitu jarak (d) yang ditempuh dari saat ($n-1$) sampai (n), serta sebuah vektor **H** di mana elemen-elemennya merupakan sudut *heading* dari kamera (*yaw, pitch, roll*). Kedua *output* tersebut akan digunakan sebagai parameter *input* fungsi MATLAB yang digunakan untuk menggambar peta ruangan. Adapun *output* inisial dari blok ini adalah 0, baik untuk **H**, maupun untuk d, karena dibutuhkan minimal dua citra di mana salah satu citra akan digunakan sebagai referensi perhitungan sudut *heading* serta jarak tempuh.

3.2.1 *Feature Correspondence Search*

Detil dari bagian *feature correspondence search* diilustrasikan oleh Gambar 3.3. Blok tersebut membutuhkan satu buah *input* yaitu citra RGB dari Kinect. Sebelum pengolahan dilakukan, akan dilakukan konversi *color space* citra

ke *grayscale*. Hal tersebut karena untuk pendeteksian fitur *corner* informasi warna pada citra tidak dibutuhkan.

Korespondensi fitur dilakukan pada dua buah citra. Maka dari itu citra ke-$(n-1)$ akan dianalisa terlebih dahulu untuk ditemukan fitur-fitur *corner*-nya. Proses penentuan fitur corner dilakukan menggunakan algoritma FAST.

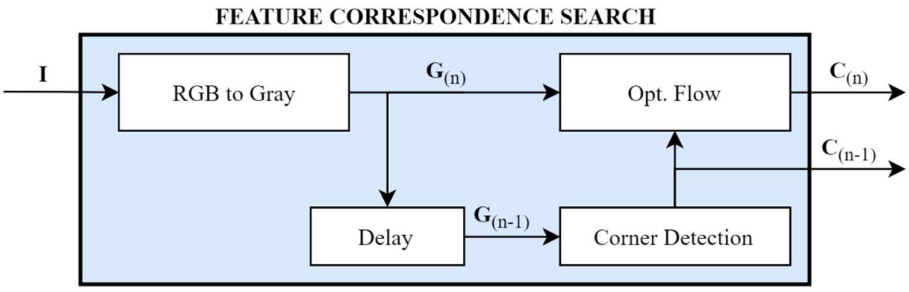
Pada citra ke-(n), pendeteksian fitur *corner* akan dilakukan menggunakan teknik *optical flow* Lucas-Kanade. Teknik tersebut memanfaatkan letak dari fitur *corner* yang sebelumnya sudah ditentukan pada citra ke-$(n-1)$. Teknik ini akan memberikan performa yang lebih cepat apabila dibandingkan dengan teknik *brute-force*. *Output* dari blok ini ada dua yaitu koordinat fitur corner pada saat $(n-1)$ dan koordinat pada saat (n).

Setelah koordinat fitur *corner* untuk kedua citra ditemukan, langkah berikut yang akan dilakukan adalah melakukan estimasi matrik *essential*. Seperti yang telah dibahas sebelumnya, bahwa matrik *essential* memiliki komponen rotasi dan translasi. Sudur *heading* dari kamera dapat diestimasi melalui komponen rotasi tersebut. Adapun komponen translasi tidak digunakan pada penelitian ini karena pada sistem *mono vision*, translasi hanya bersifat *up to a scale* sehingga tidak menggambarkan jarak yang ditempuh kamera sebenarnya.

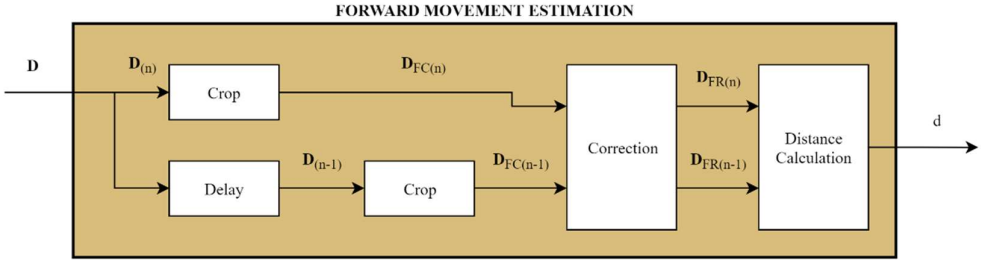
3.2.2 *Forward Movement Estimation*

Detil dari bagian *forward movement estimation* diilustrasikan oleh Gambar 3.4. Blok tersebut membutuhkan satu buah *input* yaitu citra *depth* dari Kinect. Pada blok ini tidak semua data dari citra *depth* akan digunakan, namun hanya bagian tengah citra saja. Alasan dari pemilihan area tersebut adalah kecenderungan bahwa bagian tepi akan menangkap dinding ruangan. Apabila Kinect bergerak paralel terhadap dinding, maka nilai *depth* pada bagian tersebut akan cenderung sama, sehingga hal tersebut mempengaruhi pengukuran jarak tempuh.

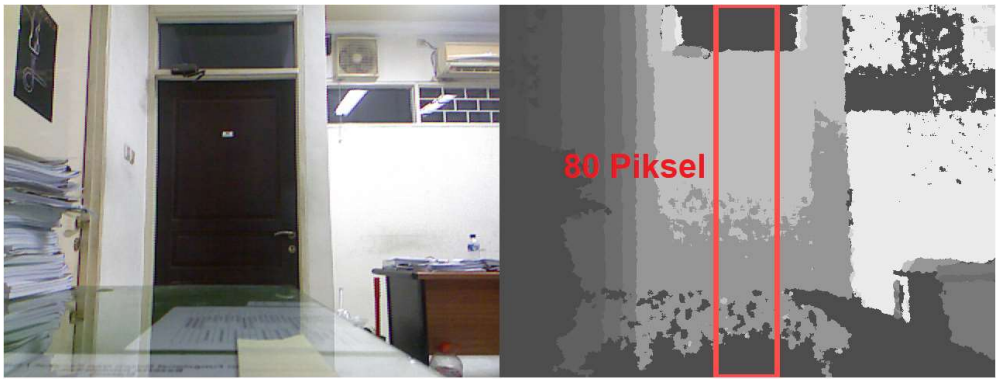
Pada blok ini dibutuhkan dua buah citra *depth*, di mana salah satu citra akan digunakan sebagai referensi pengukuran. Masing-masing citra akan di-*crop* kemudian dikurangkan sehingga jarak tempuh kamera dapat diketahui. Ilustrasi area *depth* yang akan digunakan diilustrasikan pada Gambar 3.5.



Gambar 3.3 Diagram Blok *Feature Correspondence Search*



Gambar 3.4 Diagram Blok *Forward Movement Estimation*



Gambar 3.5 Area Proses *Cropping* pada Blok *Forward Movement Estimation*

Total *distance* yang dilalui robot dapat dirumuskan secara sederhana sebagai (3.1) berikut

$$Total\ Distance = \sum_{n=2}^{m} \left(\mathbf{D}_{FR(n-1)} - \mathbf{D}_{FR(n)} \Big/ k_{(n)} \right) \qquad (3.1)$$

dengan:

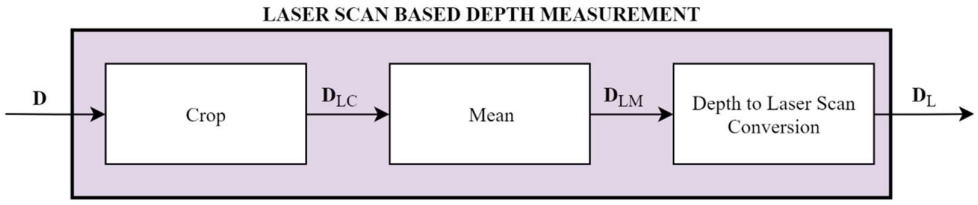
- $\mathbf{D}_{FR}$: citra *depth* proses *cropping* yang piksel bernilai nan-nya dibuat 0
- k: jumlah piksel yang nilainya tidak 0 atau nan

3.3 *Laser Scan Based Depth Measurement*

Blok ini digunakan untuk menyesuaikan data *depth* dari Kinect agar dapat digunakan sebagai parameter fungsi menggambar peta pada MATLAB. *Input* untuk blok ini adalah citra *depth* dari Kinect. Sama seperti pada blok *forward movement estimation*, tidak keseluruhan data citra *depth* akan digunakan, maka dari itu akan dilakukan proses *cropping*. Area *cropping* pada penelitian ini ditentukan memiliki tinggi 8% dari tinggi resolusi citra *depth*. Hasil dari *cropping* tersebut kemudian akan dirata-rata sehingga menjadi sebuah vektor baris $\mathbf{D}_{LM}$. Pada proses *depth to laser scan conversion*, $\mathbf{D}_{LM}$ akan dikalikan secara *element-wise* dengan $\sec(\mathbf{V})$, di mana $\mathbf{V}$ merupakan vektor baris dengan 640 elemen yang bernilai -29.25 sampai 29.25. Nilai tersebut diambil dari nilai *field of view* horisontal *depth* Kinect, yaitu 58.5° yang kemudian dibagi 2. Diagram blok *laser scan based depth measurement* diilustrasikan pada Gambar 3.6.

Konversi citra *depth* dari Kinect ke dalam representasi *laser scan* dapat dirumuskan secara sederhana sebagai (3.2) berikut

$$\mathbf{D}_L = \mathbf{D}_{LM(n-1)} \sec(\mathbf{V}) \tag{3.2}$$



Gambar 3.6 Diagram Blok *Laser Scan Based Depth Measurement*

Halaman ini sengaja dikosongkan

BAB 4
HASIL DAN PEMBAHASAN

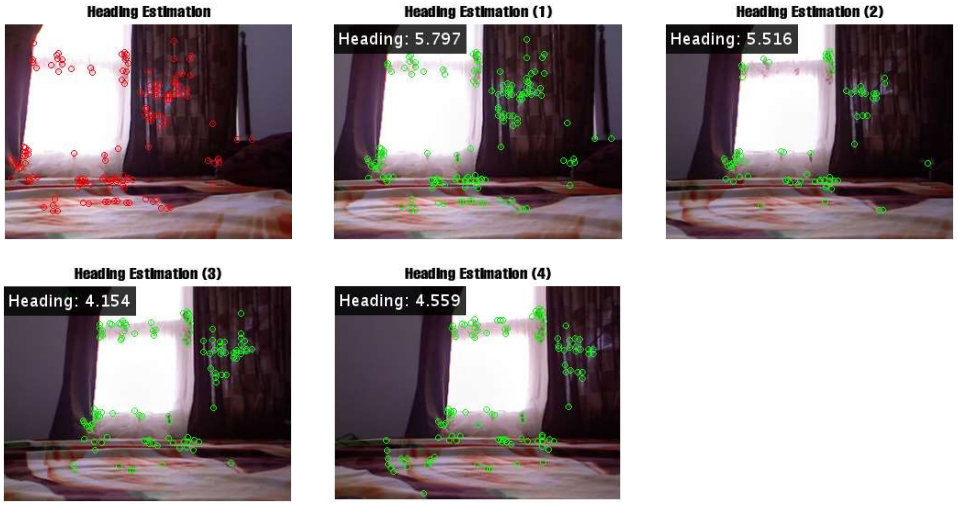
Berdasarkan penjelasan sebelumnya, akan dilakukan uji coba pada sistem yang dikembangkan mulai dari bagian-bagian kecil yang menyusun blok diagram pada Gambar 3.1, hingga hasil uji coba pemetaan. Uji coba akan dilakukan dengan cara menggerakkan Kinect secara manual.

4.1 Uji Coba Estimasi *Heading*

Pertama-tama, uji coba akan dilakukan pada 5 buah citra, di mana pada masing-masing citra yang ditangkap oleh Kinect dilakukan perputaran sebesar 5°. Uji coba berikutnya adalah menggunakan hasil rekam *video* oleh Kinect, di mana Kinect akan diputar sebesar 90° selama durasi tersebut. Uji coba berikutnya adalah menggunakan hasil rekam *video* oleh Kinect, di mana Kinect diletakkan di atas robot yang kemudian didorong menyusuri lorong dengan heading 0°.

4.1.1 Estimasi *Heading* pada 5 Buah Citra

Hasil dari pengujian estimasi *heading* terhadap 5 buah citra terdapat pada Gambar 4.1.



Gambar 4.1 *Screenshot* Uji Coba Estimasi *Heading* pada 5 Buah Citra

Tabel 4.1 Hasil Uji Coba pendeteksian *Heading* pada 5 Buah Citra

Perputaran	FAST	GFtT	Harris
5°	5.797°	4.210°	5.795°
5°	5.516°	5.463°	5.554°
5°	4.154°	4.108°	4.236°
5°	4.559°	4.919°	4.854°
Sum Absolute Error	2.6°	1.8093°	2.259°

Proses uji coba dilakukan menggunakan beberapa algoritma pendeteksi fitur *corner*, yaitu FAST, *Good Features to Track* (GFtT), dan Harris. Hasil dari masing-masing algoritma dirangkum pada Tabel 4.1. Berdasarkan uji coba didapatkan hasil *sum absolute error* dari FAST adalah 2.6°, GFtT adalah 1.8093°, dan Harris adalah 2.259°.

4.1.2 Estimasi *Heading* pada *Video* 1

Screenshot hasil dari pengujian estimasi *heading* pada hasil rekaman *video* 90° oleh Kinect tedapat pada Gambar 4.2. Hasil akhir, yaitu pada *frame* ke 301 merupakan akumulasi dari total estimasi *heading*. Pengujian ini dilakukan beberapa algoritma pendeteksi fitur *corner*, yaitu FAST, *Good Features to Track* (GFtT), dan Harris. Hasil dari masing-masing algoritma dirangkum pada Tabel 4.2. Berdasarkan uji coba didapatkan hasil *absolute sum of error* dari FAST adalah 0.776°, GFtT adalah 0.861°, dan Harris adalah 2.149°.

4.1.3 Estimasi *Heading* pada *Video* 2

Screenshot hasil dari pengujian estimasi *heading* pada hasil rekaman *video* saat Kinect berjalan pada lorong dengan sudut heading 0° terdapat pada Gambar 4.3. Hasil akhir, yaitu pada *frame* ke-451 merupakan akumulasi dari total estimasi *heading*. Pengujian ini dilakukan beberapa algoritma pendeteksi fitur *corner*, yaitu FAST, *Good Features to Track* (GFtT), dan Harris. Hasil dari masing-masing algoritma dirangkum pada Tabel 4.3. Berdasarkan uji coba didapatkan hasil *absolute sum of error* dari FAST adalah -1.374°, GFtT adalah 1.868°, dan Harris adalah 1.78°.

Tabel 4.2 Hasil Uji Coba pendeteksian *Heading* pada *Video* 1

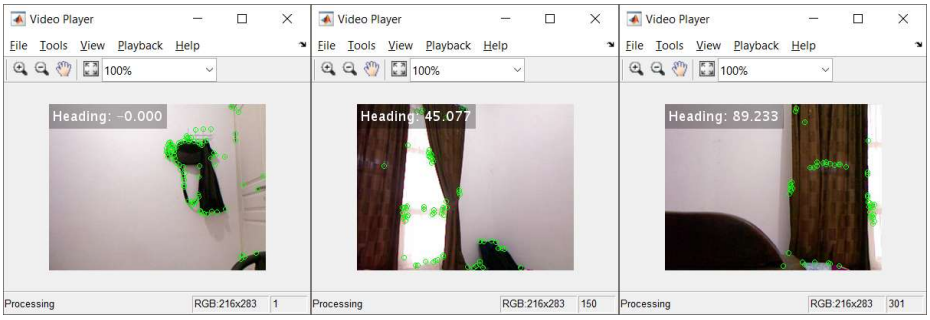
Perputaran	FAST	GFtT	Harris
90°	89.224°	90.861°	87.851°
Absolute Error	0.776°	0.861°	2.149°

Tabel 4.3 Hasil Uji Coba pendeteksian *Heading* pada *Video* 2

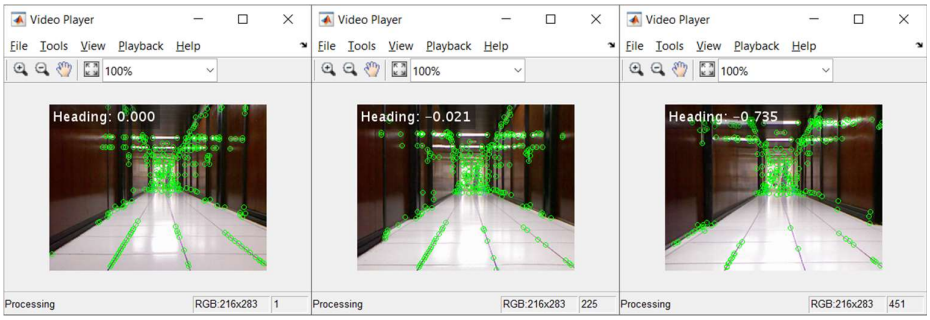
Perputaran	FAST	GFtT	Harris
0°	-1.374°	1.868°	1.78°
Absolute Error	1.374°	1.868°	1.78°

4.2 Uji Coba *Forward Movement Estimation*

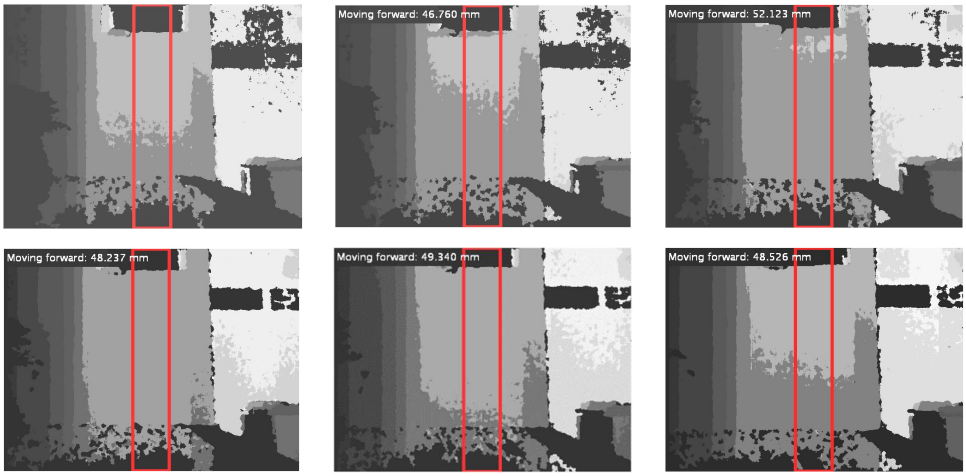
Uji coba *forward movement estimation* akan dilakukan pada 6 citra, di mana citra pertama berjarak 5 cm dari citra kedua, citra kedua berjarak 5 cm dari citra ketiga, dst. Setelah uji coba tersebut, akan dilakukan pula uji coba pada hasil rekaman *video* di mana kamera akan digerakkan maju sepanjang 30 cm.



Gambar 4.2 *Screenshot* Uji Coba Estimasi *Heading* pada *Video* 1



Gambar 4.3 *Screenshot* Uji Coba Estimasi *Heading* pada *Video* 2



Gambar 4.4 *Screenshot* Uji Coba *Forward Movement Estimation* pada 6 Buah Citra *Depth*

Tabel 4.4 Hasil Uji Coba *Forward Movement Estimation* pada 6 Buah Citra *Depth*

Gerakan Maju	**Estimasi**	***Error***
50 mm	46.760 mm	3.24 mm
50 mm	52.123 mm	-2.123 mm
50 mm	48.237 mm	1.763 mm
50 mm	49.34 mm	0.66 mm
50 mm	48.526 mm	1.474 mm
Absolute Sum of Error		5.014 mm

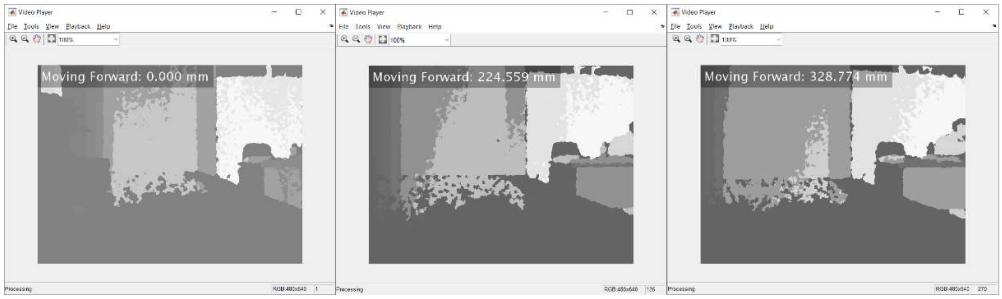
4.2.1 *Forward Movement Estimation* pada 6 Buah Citra *Depth*

Hasil dari pengujian *forward movement estimation* pada 6 buah citra terdapat pada Gambar 4.4. Hasil pengujian tersebut dirangkum ke dalam Tabel 4.4.

4.2.2 *Forward Movement Estimation* pada *Video*

Hasil dari pengujian *forward movement estimation* pada *video* terdapat pada Gambar 4.5. Hasil akhir, yaitu pada *frame* ke-271 merupakan akumulasi dari *forward movement estimation*. Meskipun metode ini dapat memberikan hasil yang cukup baik (*error* untuk masing-masing pengukuran kurang dari 5 mm) pada pengujian sebelumnya, metode ini tidak sesuai apabila digunakan pada *video*. Hal tersebut dikarenakan *error* pengukuran terakumulasi. Hasil yang diberikan oleh metode ini berbeda ~69 cm dari jarak yang sebenarnya yaitu ~30 cm.

Gambar 4.5 Hasil Uji Coba Pertama *Forward Movement Estimation* pada *Video*



Gambar 4.6 Hasil Uji Coba Kedua *Forward Movement Estimation* pada *Video*

Maka dari itu, pada pergerakan maju, akan diambil citra *depth* awal yang akan digunakan sebagai referensi. Citra *depth* referensi tersebut akan digunakan selama kamera bergerak maju. Jarak tempuh dari kamera dapat diestimasi dengan cara mengurangkan citra *depth* referensi dengan citra *depth* yang ditangkap berikutnya. Persamaan (3.1) akhirnya disesuaikan menjadi (4.1)

$$Total\ Distance = \mathbf{D}_{FR(ref)} - \mathbf{D}_{FR(n)} \tag{4.1}$$

Hasil proses uji coba terdapat pada Gambar 4.6. Dengan cara tersebut, hasil yang diberikan oleh metode ini berbeda ~2.8 cm dari jarak yang sebenarnya yaitu ~30 cm.

4.3 Uji Coba *Laser Scan Based Depth Measurement*

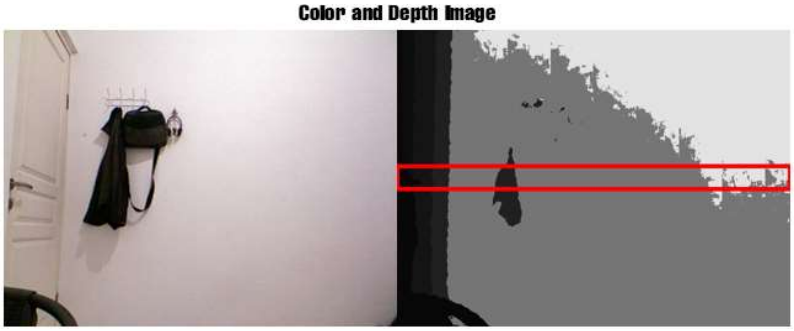
Pengujian dilakukan dengan cara mengambil citra *depth* yang kemudian akan dikonversi agar representasinya sesuai dengan *output* dari *laser scan* (LiDAR). Hasil dari konversi tersebut akan digunakan sebagai *input* fungsi

MATLAB yaitu `insertRay`. Hasil dari uji coba ini terdapat pada Gambar 4.7 dan Gambar 4.8.

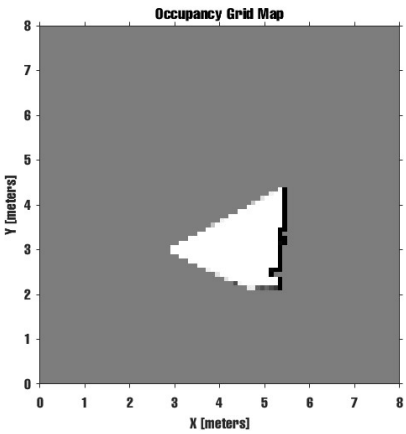
4.4 Uji Coba Pemetaan Ruangan 1

Berdasarkan hasil uji coba yang dilakukan sebelumnya, dilakukan uji coba pemetaan ruangan. Uji coba dilakukan menggunakan 3 buah algoritma pendeteksi corner yang berbeda, yaitu FAST, GFtT, dan Harris.

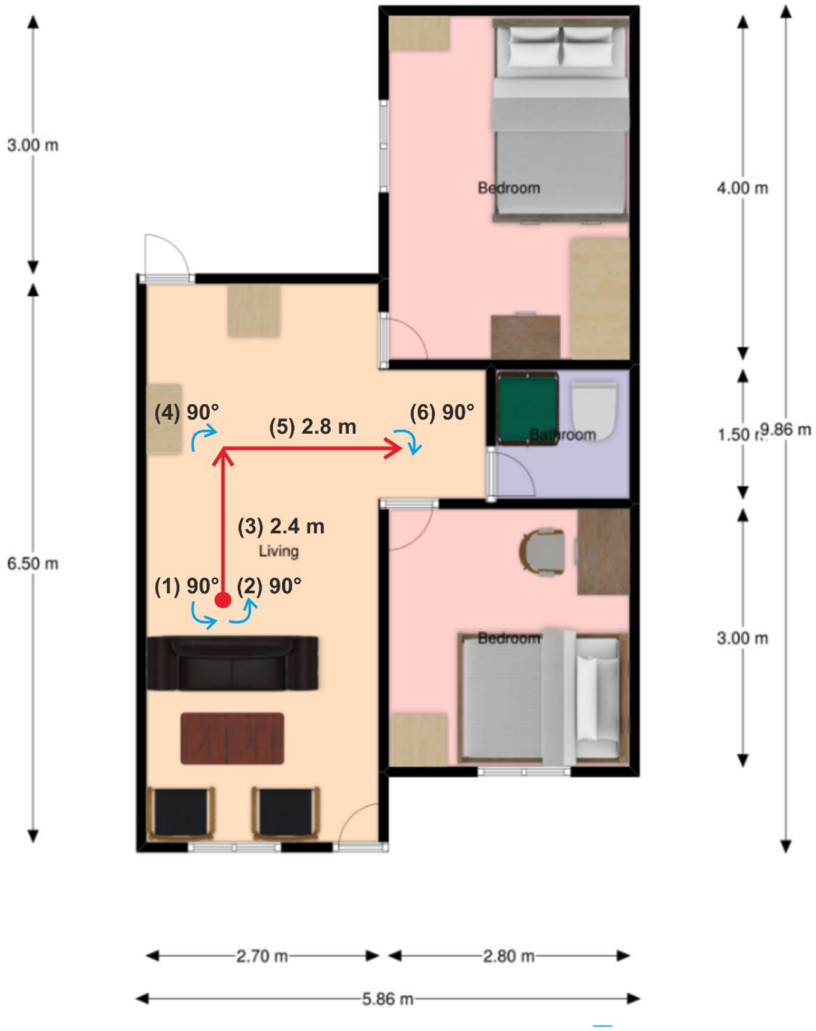
Proses uji coba sistem dilakukan dengan menggerakkan Kinect dengan jalur yang telah ditentukan sesuai Gambar 4.9. Hasil pemetaan jika fitur pendeteksi *corner* yang digunakan adalah FAST terdapat pada Gambar 4.10. Hasil pemetaan jika fitur pendeteksi *corner* yang digunakan adalah GFtT terdapat pada Gambar 4.11. Sedangkan hasil pemetaan jika fitur pendeteksi *corner* yang digunakan adalah Harris terdapat pada Gambar 4.12.



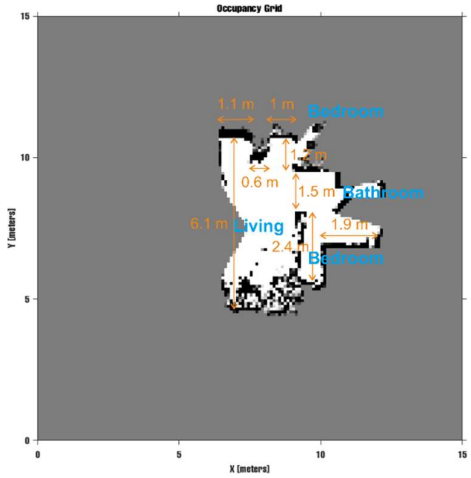
Gambar 4.7 *Screenshot* Citra *Depth* untuk Uji Coba *Laser Scan Based Depth Measurement*



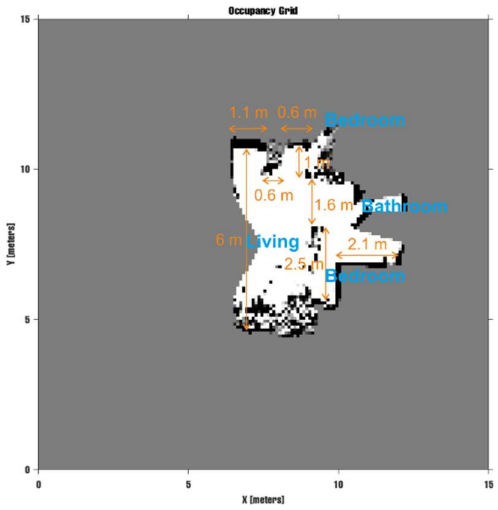
Gambar 4.8 Hasil Penggambaran *Depth* pada *Occupancy Grid Map*



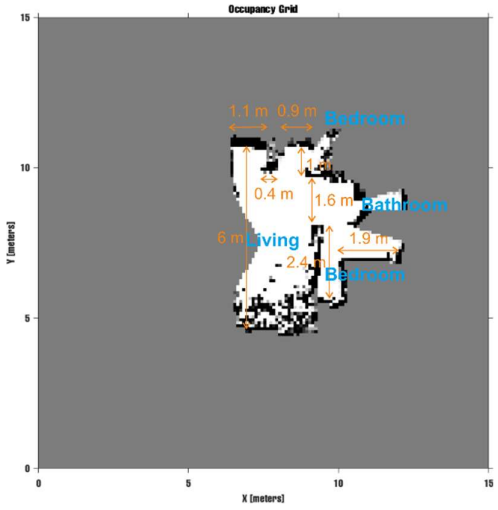
Gambar 4.9 Denah Ruangan 1 dan Jalur yang ditempuh Kinect



Gambar 4.10 Hasil Pemetaan Ruangan 1 dengan Algoritma FAST

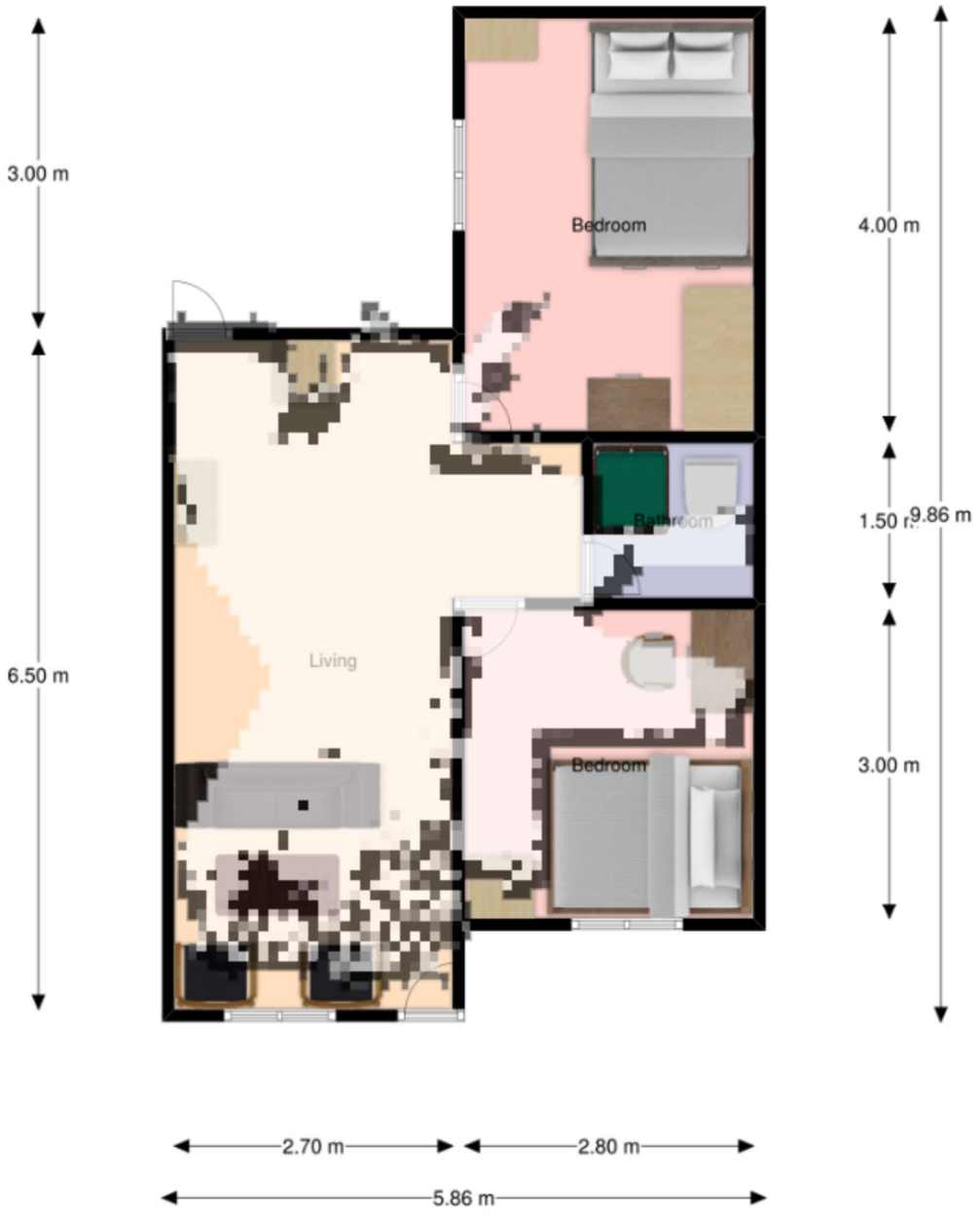


Gambar 4.11 Hasil Pemetaan Ruangan 1 dengan Algoritma GFtT

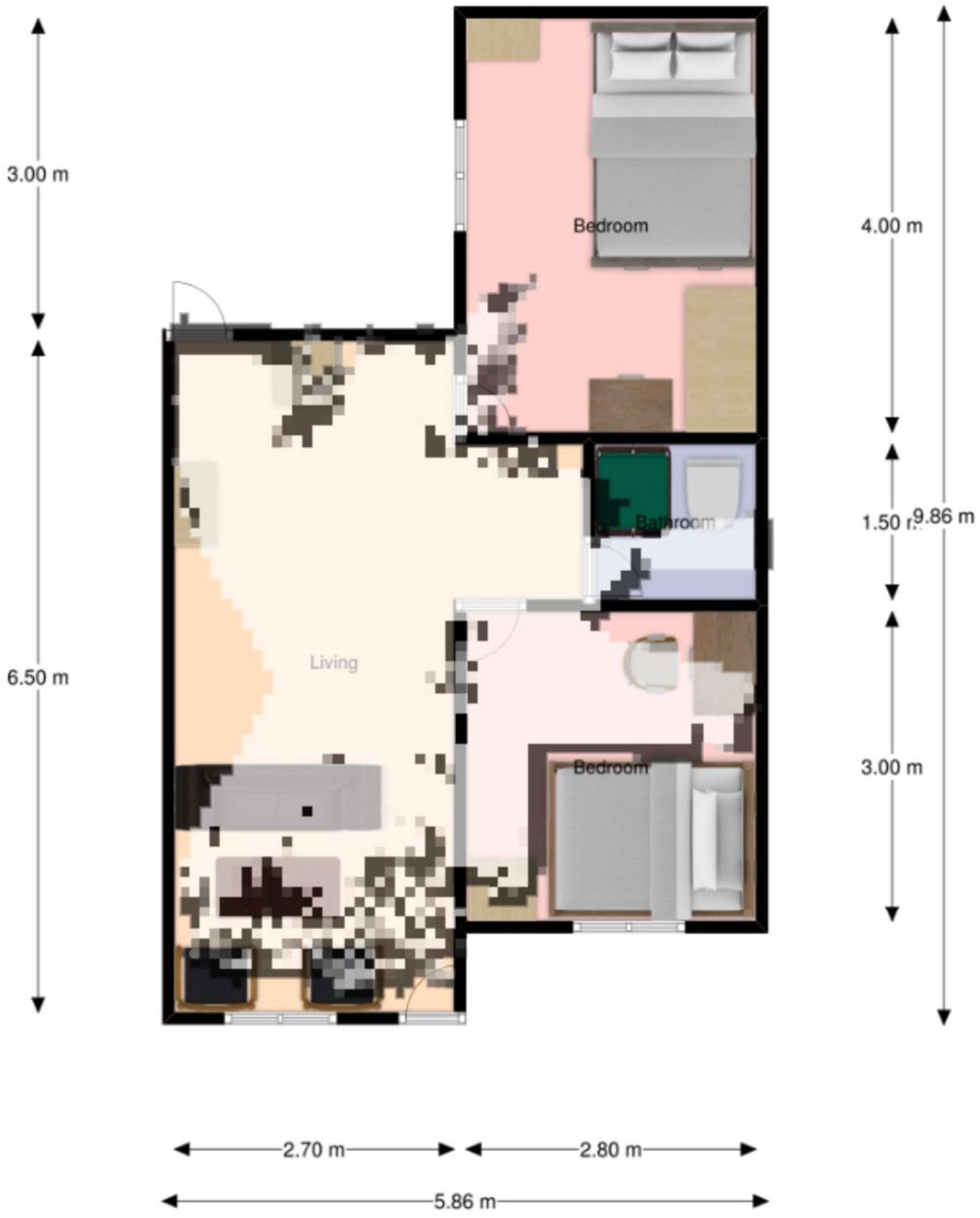


Gambar 4.12 Hasil Pemetaan Ruangan 1 dengan Algoritma Harris

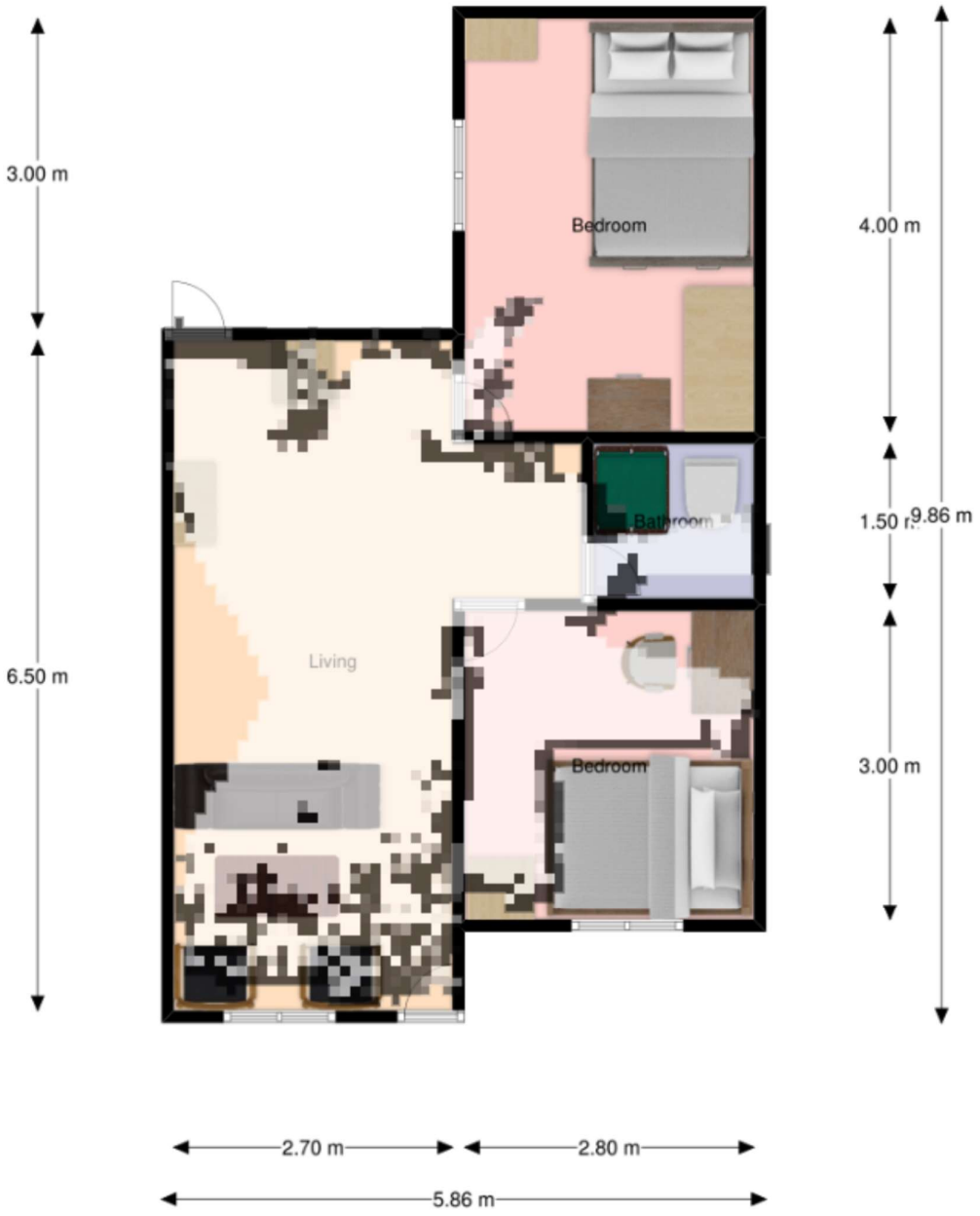
Perbandingan antara denah ruangan 1 dengan hasil pemetaan terdapat pada Gambar 4.13, Gambar 4.14, dan Gambar 4.15. Dari hasil-hasil peta yang telah didapatkan, akan dilakukan perbandingan antara jarak yang terukur oleh Kinect pada beberapa tempat dengan jarak yang sebenarnya. Hasil dari perbandingan tersebut dirangkum pada Tabel 4.5. Dengan skenario uji coba tersebut, hasil pemetaan menggunakan algoritma FAST sebagai pendeteksi fitur *corner* memberikan nilai *absolute error* yang paling kecil, yaitu 0.3 m dari pengukuran sebenarnya. Sedangkan algoritma GFtT dan Harris masing-masing memberikan nilai *absolute error* 0.6 m dan 0.8 m.



Gambar 4.13 Perbandingan Hasil Pemetaan memanfaatkan Algoritma FAST dengan Denah Ruangan 1



Gambar 4.14 Perbandingan Hasil Pemetaan memanfaatkan Algoritma GFtT dengan Denah Ruangan 1



Gambar 4.15 Perbandingan Hasil Pemetaan memanfaatkan Algoritma Harris dengan Denah Ruangan 1

4.5 Uji Coba Pemetaan Ruangan 2

Setelah uji coba pemetaan ruangan yang pertama, uji coba pemetaan dilakukan pada ruangan kedua dengan ukuran yang lebih besar dengan ukuran 10.3 x 15 m. Uji coba ini juga dilakukan menggunakan 3 buah algoritma pendeteksi *corner* yang

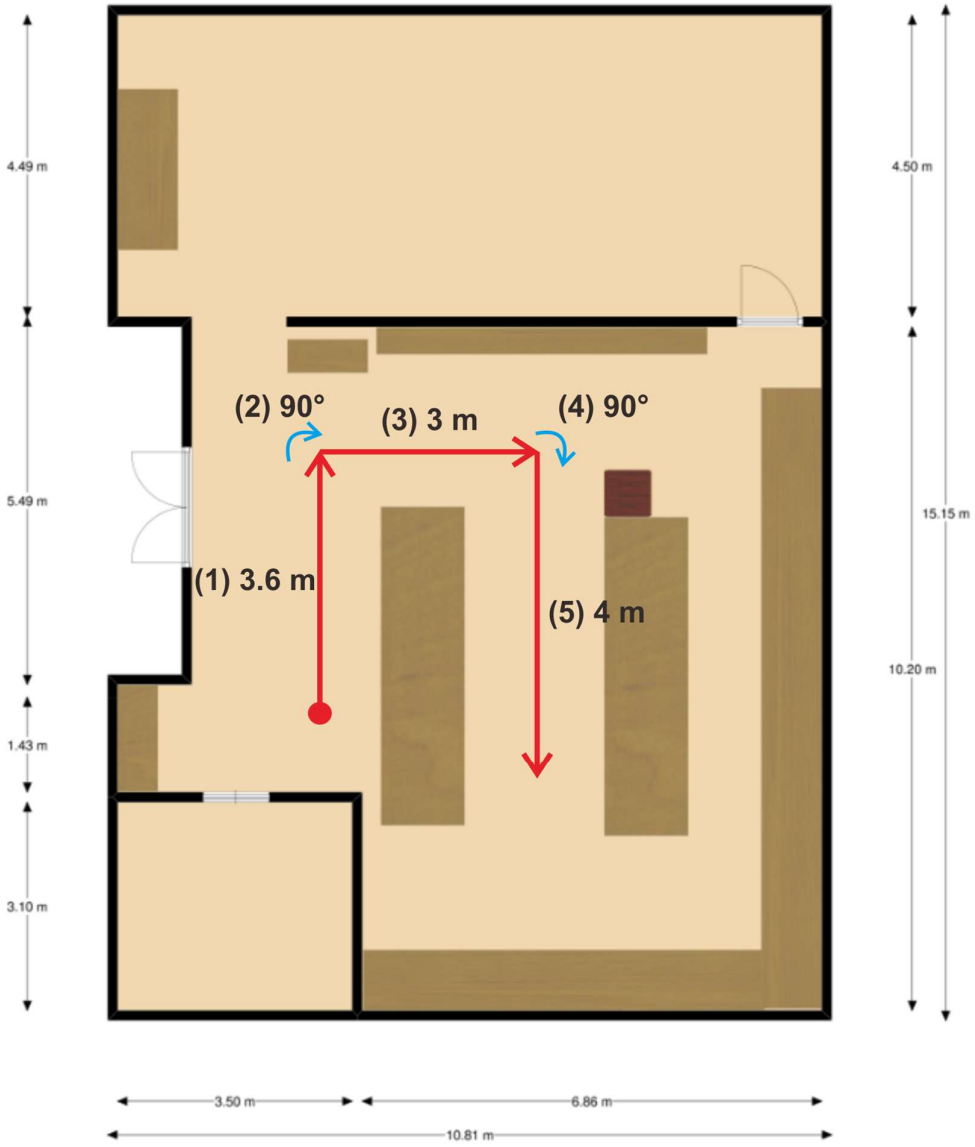
berbeda, yaitu FAST, GFtT, dan Harris. *Setting* dari uji coba ini diilustrasikan oleh Gambar 4.16 dan 4.17.

Tabel 4.5 Hasil Uji Coba Pemetaan Ruangan 1

	Jarak Sebenarnya	**FAST**	**GFtT**	**Harris**
	6.5 m	6.1 m	6 m	6 m
	1 m	1.1 m	1.1 m	1.1 m
	0.6 m	0.6 m	0.6 m	0.4 m
	1.1 m	1 m	0.6 m	0.9 m
	1 m	1.2 m	1 m	1 m
	1.5 m	1.5 m	1.6 m	1.6 m
	2.4 m	2.4 m	2.5 m	2.4 m
	2 m	1.9 m	2.1 m	1.9 m
Jumlah	16.1 m	15.8 m	15.5 m	15.3 m
Absolute Error		0.3 m	0.6 m	0.8 m

Gambar 4.16 Foto ruangan 2 yang hendak dipetakan

Gambar 4.17 Peletakan Kinect dan *laptop* pada *frame* AIV



Gambar 4.18 Denah Ruangan 2 dan Jalur yang ditempuh Kinect

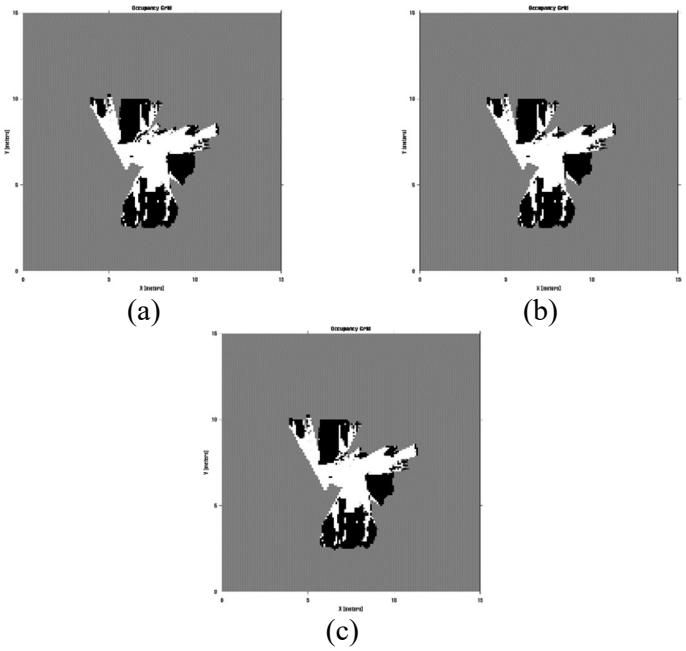
Proses uji coba sistem dilakukan dengan menggerakkan Kinect dengan jalur yang telah ditentukan sesuai Gambar 4.18. Hasil pemetaan dengan fitur pendeteksi *corner* FAST, GFtT, dan Harris terdapat pada Gambar 4.19.

Dari hasil uji coba yang dilakukan, metode ini tidak sesuai apabila digunakan pada ruangan dengan dimensi besar. Hal tersebut dikarenakan batas pengukuran *depth* oleh Kinect adalah sampai ~4 m. Karena dimensi ruangan serta rute yang digunakan tidak dapat memenuhi kriteria tersebut, maka pengukuran jarak tempuh Kinect mengalami kegagalan sehingga peta yang dihasilkan pun tidak

baik. Apabila jalur yang ditempuh oleh Kinect diperpendek seperti pada Gambar 4.20, dengan tujuan agar Kinect mendapatkan informasi *depth*, maka hasil dari pemetaan adalah seperti yang terdapat pada Gambar 4.21.

Berdasarkan hasil yang didapat sebelumnya, pada percobaan berikutnya, informasi pergerakan maju Kinect tidak didapatkan melalui informasi *depth*, melainkan melalui pengamatan secara manual. Hal ini dikarenakan metode yang digunakan tidak sesuai jika ruangan terlalu besar. *Frame* AIV akan didorong maju dengan kecepatan tertentu selama beberapa detik. Dengan demikian, pergerakan maju Kinect dapat diperkirakan. Rute yang digunakan adalah sama dengan yang diilustrasikan pada Gambar 4.18. Hasil dari percobaan ini terdapat pada Gambar 4.22.

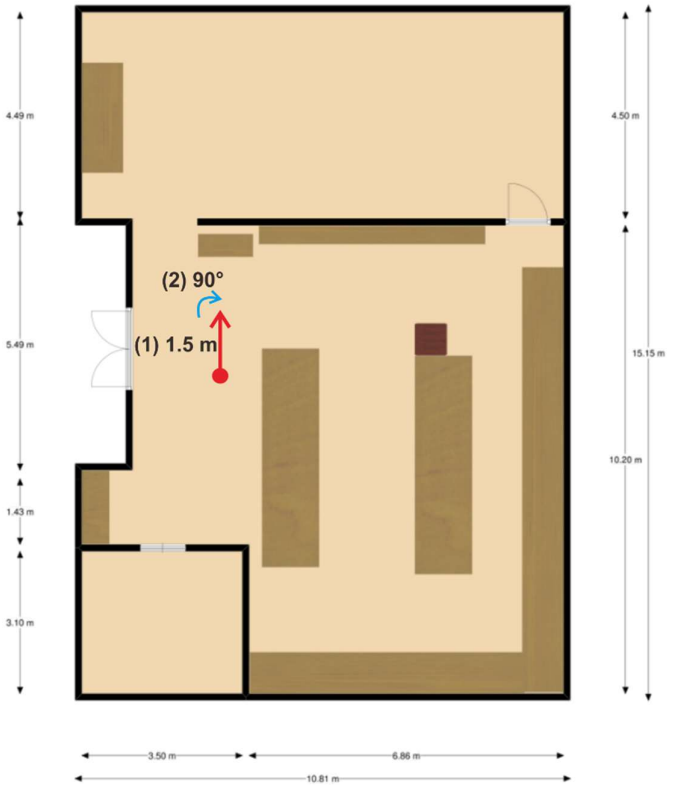
Perbandingan antara denah ruangan 2 dengan hasil pemetaan terdapat pada Gambar 4.23, Gambar 4.24, dan Gambar 4.25. Berdasarkan hasil tersebut, masih terdapat pergeseran lokasi *obstacle* antara hasil pemetaan dengan denah ruangan sebenarnya.



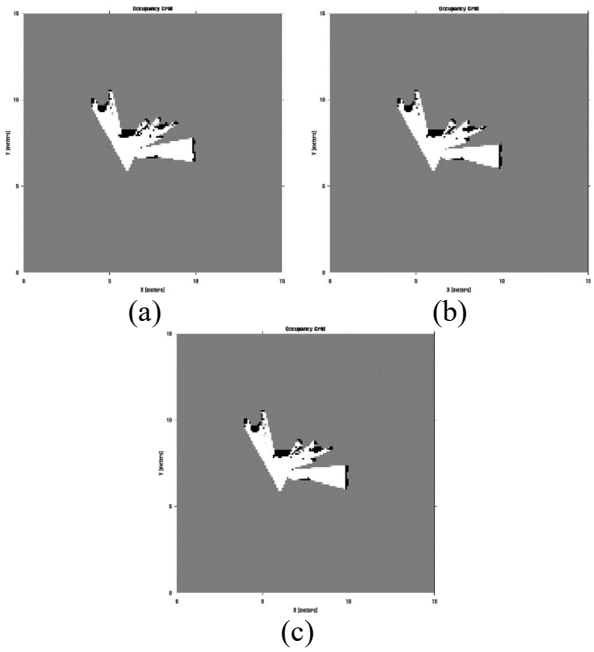
(a) (b)

(c)

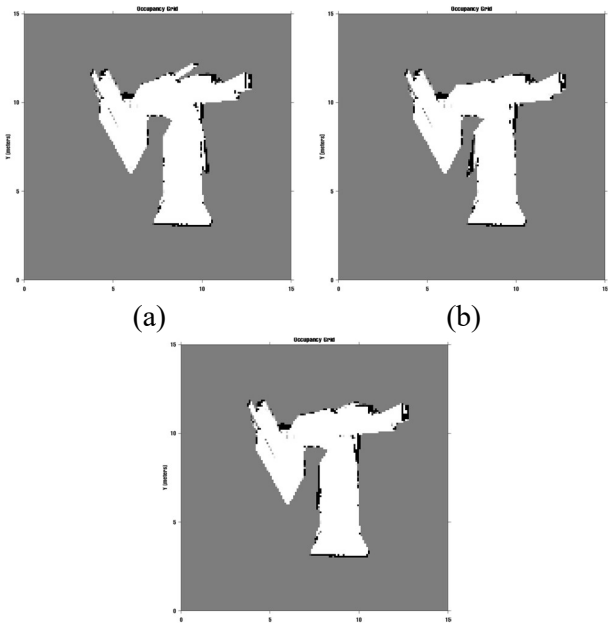
Gambar 4.19 Hasil Pemetaan Ruangan 2 dengan Algoritma (a) FAST, (b) GFtT, (c) Harris



Gambar 4.20 Denah Ruangan 2 dan Jalur Tempuh Kinect yang diperpendek



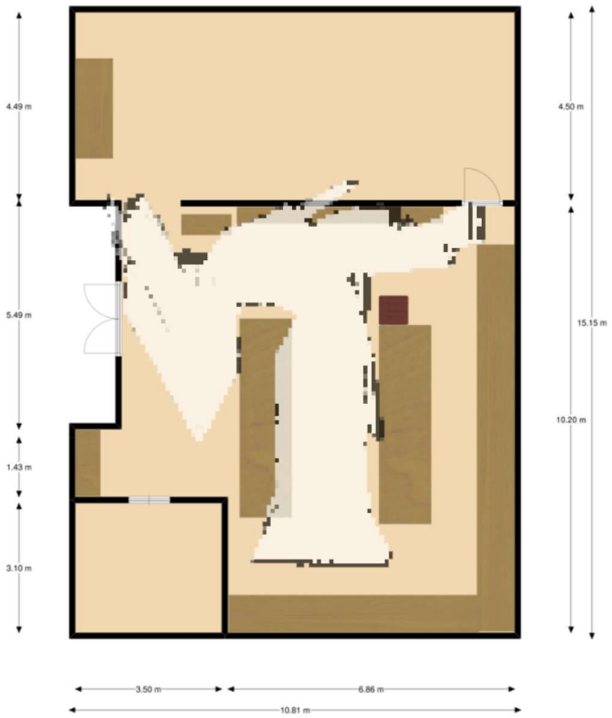
Gambar 4.21 Hasil Pemetaan Ruangan 2 dengan Algoritma (a) FAST, (b) GFtT, (c) Harris jika Jalur Tempuh Kinect diperpendek



(a) (b)

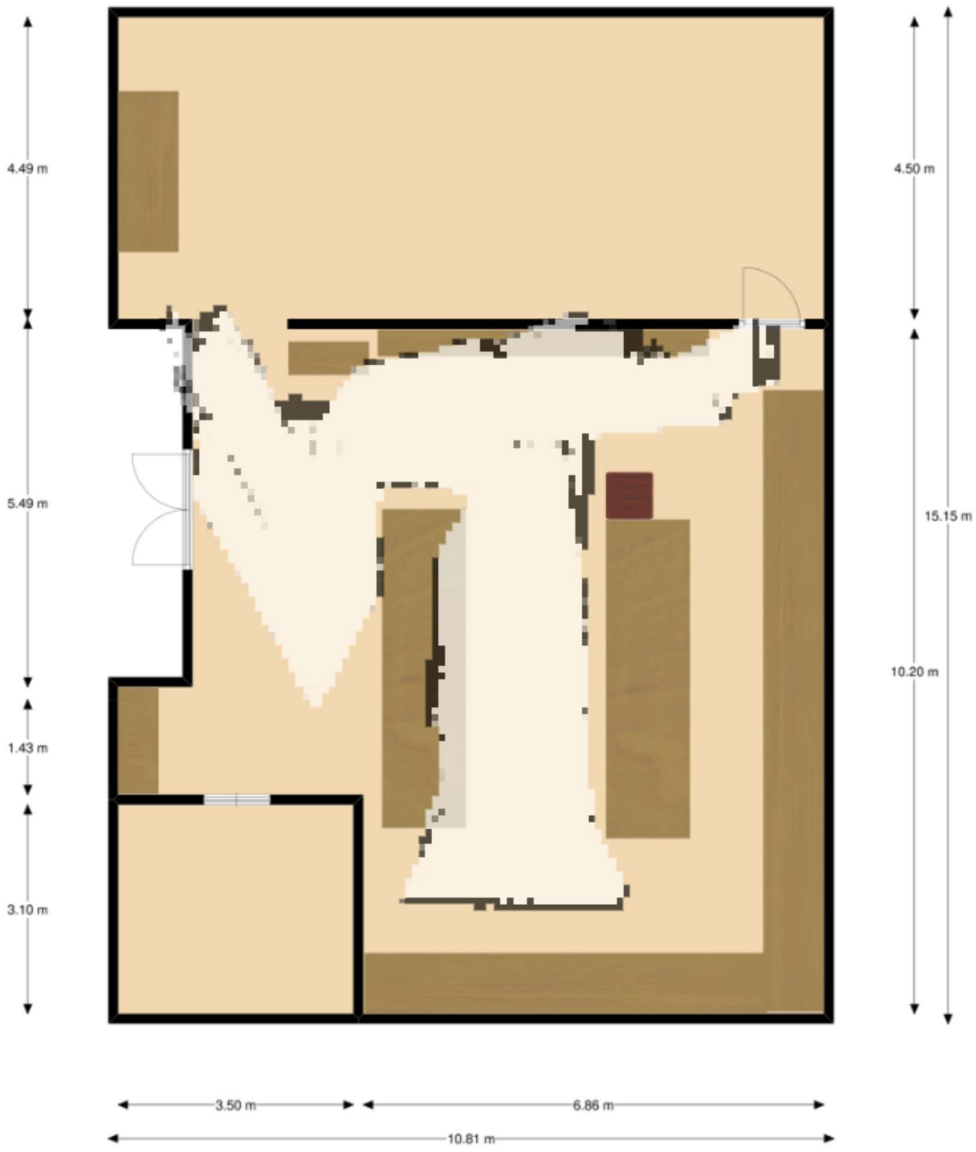
(c)

Gambar 4.22 Hasil Pemetaan Ruangan 2 menggunakan Informasi Jarak Tempuh secara Manual dengan Algoritma (a) FAST, (b) GFtT, (c) Harris



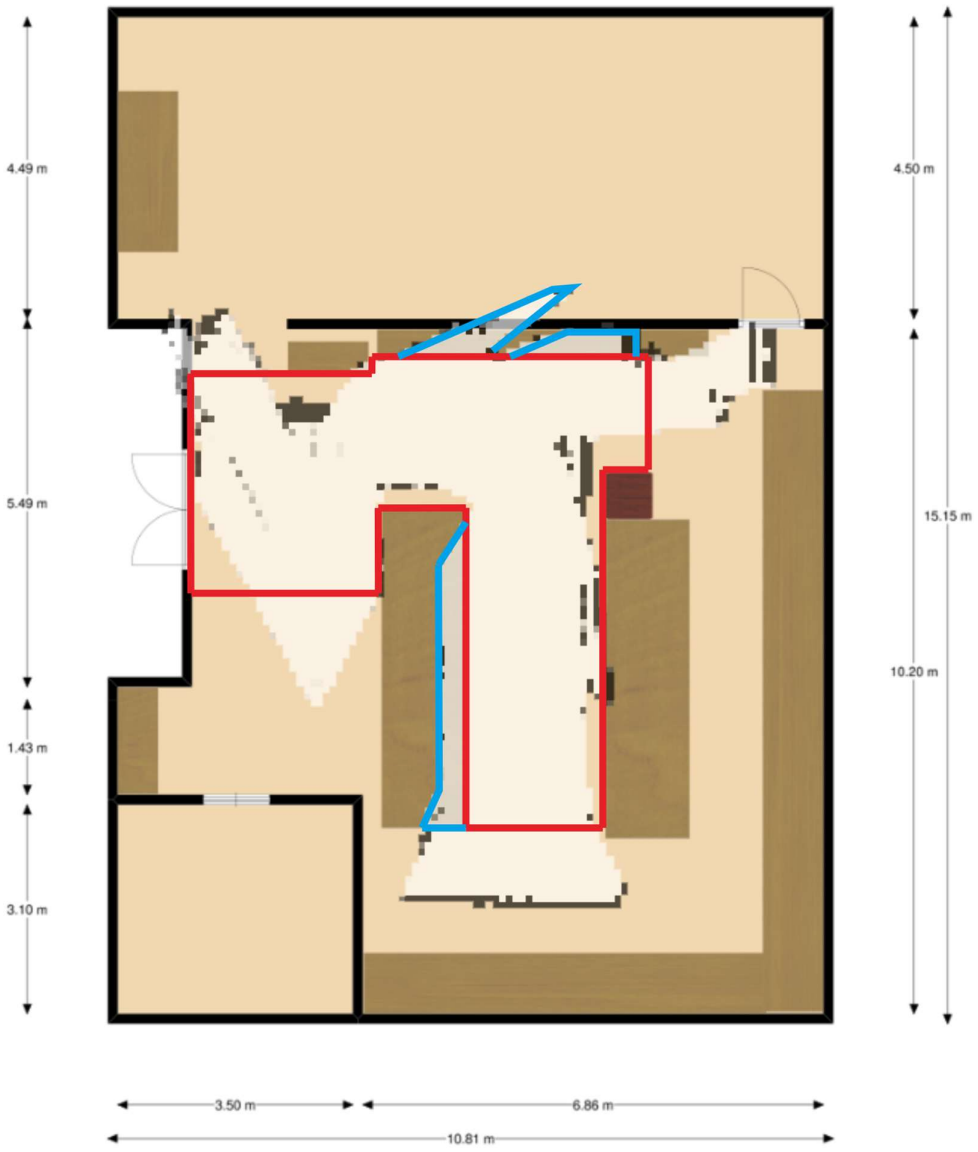
Gambar 4.23 Perbandingan Hasil Pemetaan memanfaatkan Algoritma FAST dengan Denah Ruangan 2

Gambar 4.24 Perbandingan Hasil Pemetaan memanfaatkan Algoritma GFtT dengan Denah Ruangan 2

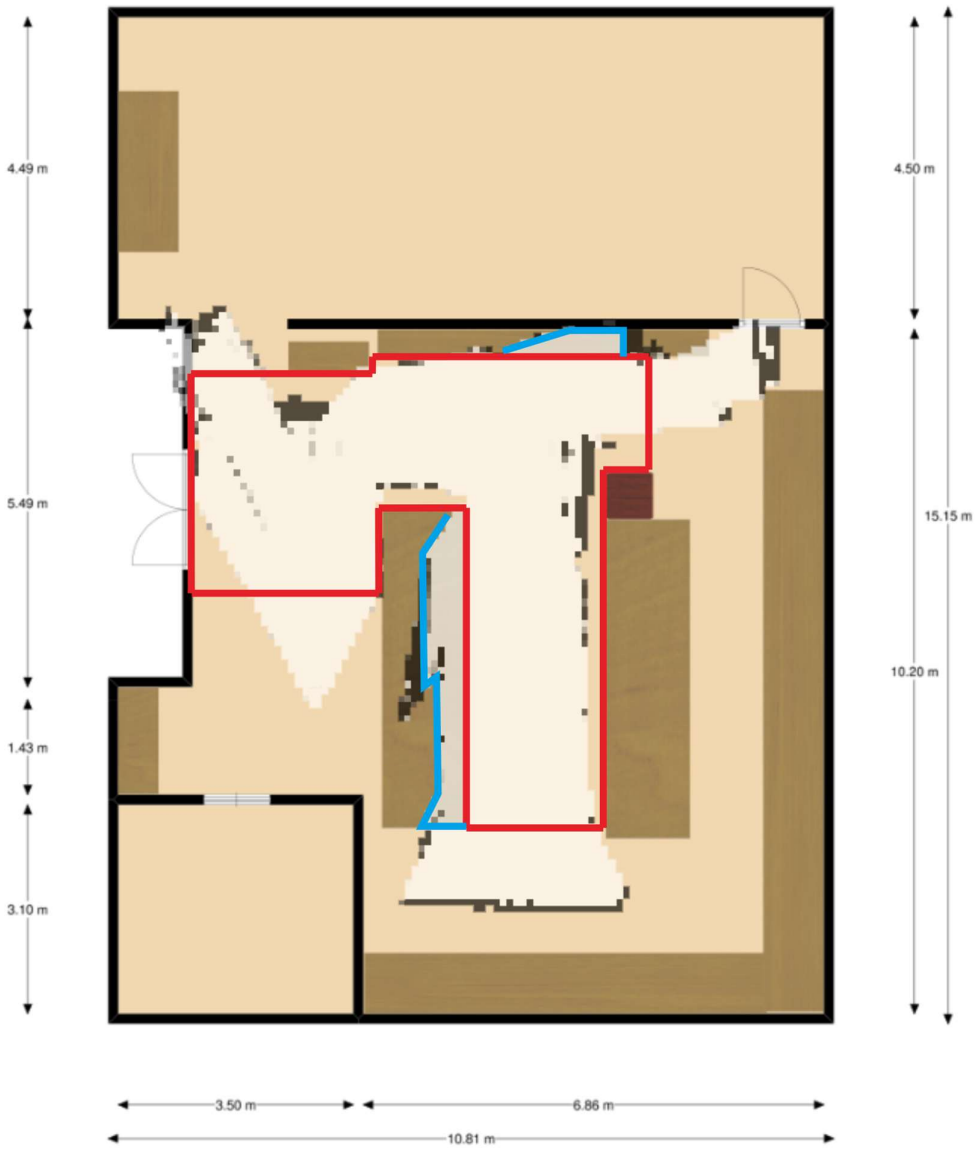


Gambar 4.25 Perbandingan Hasil Pemetaan memanfaatkan Algoritma Harris dengan Denah Ruangan 2

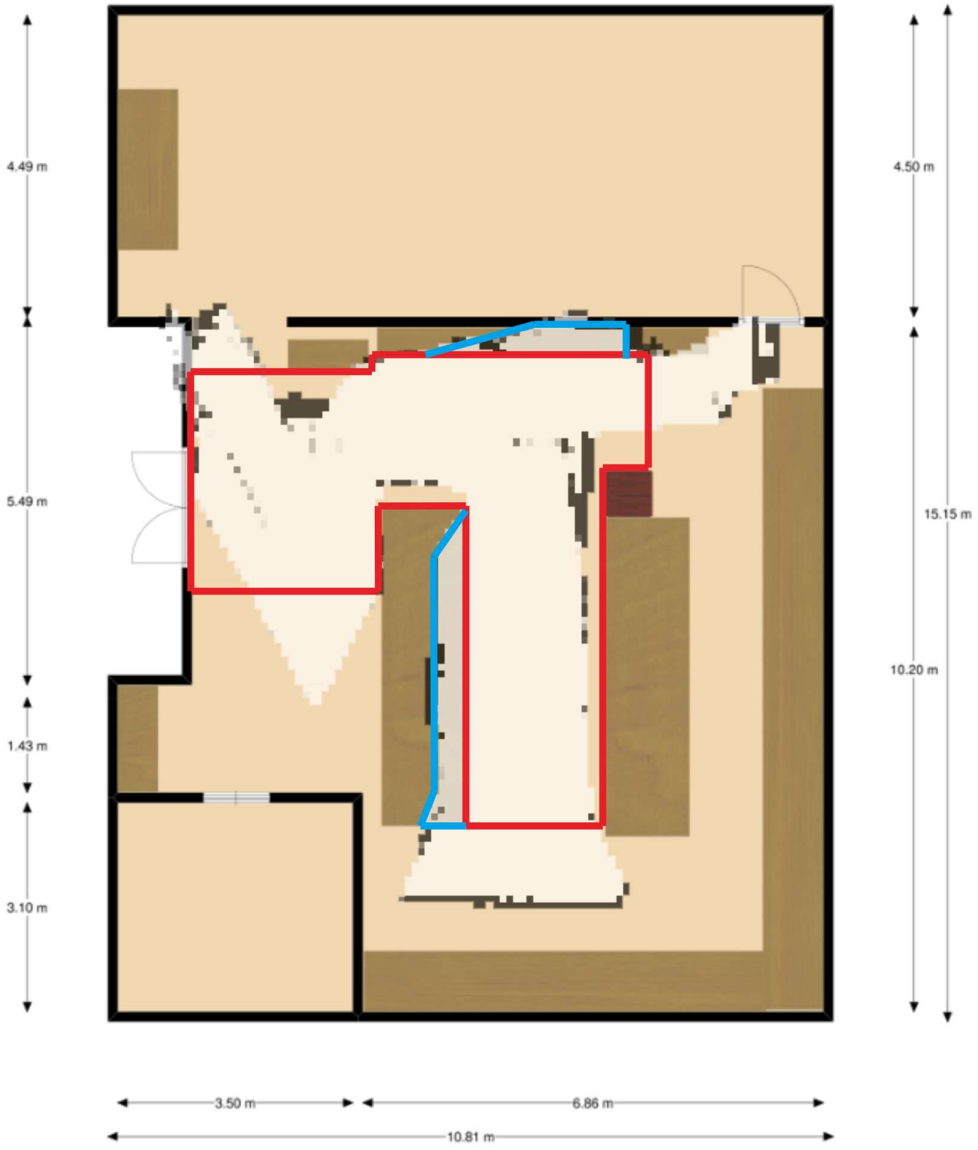
Akurasi masing-masing percobaan diukur berdasarkan perbandingan antara luas sebenarnya dengan luas yang diukur melalui peta yang dibuat. Adapun area pada ruangan yang akan digunakan diilustrasikan pada Gambar 4.26, Gambar 4.27, dan Gambar 4.28. Perbandingan yang dilakukan meliputi dua hal yaitu *uncovered area* (nilai mutlak dari luas area dalam garis merah dikurangi luas area dalam garis merah yang berwarna putih) dan *excessed area* (luas area dalam garis biru). Hasil perbandingan dirangkum pada Tabel 4.6 dan Tabel 4.7.



Gambar 4.26 Area Pengukuran pada Denah Ruangan 2 dan Hasil Pemetaan memanfaatkan Algoritma FAST



Gambar 4.27 Area Pengukuran pada Denah Ruangan 2 dan Hasil Pemetaan memanfaatkan Algoritma GFtT



Gambar 4.28 Area Pengukuran pada Denah Ruangan 2 dan Hasil Pemetaan memanfaatkan Algoritma Harris

Tabel 4.56 *Uncovered Area* pada Pemetaan Ruangan 2

Luas	**Aktual**	**FAST**	**GFtT**	**Harris**
	27.873 m²	23.963 m²	23.953 m²	23.713 m²
Uncovered Area	-	3.91 m²	3.92 m²	4.16 m²

Tabel 4.7 *Excessed Area* Pemetaan Ruangan 2

	FAST	**GFtT**	**Harris**
Excessed Area	4.36 m²	4.26 m²	4.38 m²

Berdasarkan hasil percobaan, pemetaan menggunakan algoritma FAST memberikan *error uncovered area* sebesar 3.91 m², GFtT 3.92 m², Harris 4.16 m², terhadap luasan area aktual sebesar 27.873 m². Sedangkan untuk *excessed area*, algoritma FAST memberikan *error* sebesar 4.36 m², GFtT 4.26 m², dan Harris 4.38 m². Apabila kedua error tersebut dijumlahkan, maka algoritma FAST memiliki *error* 8.27 m², GFtT memiliki *error* 8.18 m², sedangkan Harris memiliki *error* 8.54 m². Berdasarkan hasil tersebut, dapat disimpulkan bahwa penggunakan algoritma GFtT memberikan performa yang lebih baik dibandingkan dengan FAST ataupun Harris.

BAB 5
KESIMPULAN

5.1 Kesimpulan

Berdasarkan hasil dari uji coba, perkiraan *heading* pada *video* dengan putaran sebesar 90° memberikan *sum abosolute error* sebesar 0.776° untuk FAST, 0.861° untuk GFtT, 2.149° untuk Harris. Berdasarkan hasil dari uji coba, perkiraan jarak tempuh maju didapatkan hasil *sum absolute error* 69 cm dengan cara mengakumulasi selisih citra *depth* ke $(n-1)$ dengan citra *depth* ke n. Hasil yang diberikan lebih baik jika perkiraan gerakan maju dilakukan dengan cara mengambil sebuah citra *depth* sebagai referensi, yang kemudian akan dikurangkan dengan citra depth yang didapatkan setiap waktu. Dengan cara tersebut, didapatkan *sum absolute error* sebesar 2.8 cm.

Uji coba pembuatan peta telah dilakukan pada ruangan dengan dimensi 10.81 x 15.15 m. Pada ruangan tersebut pengukuran jarak tempuh maju oleh Kinect tidak dapat dilakukan, sehingga informasi tersebut diberikan secara manual pada program, yaitu berdasarkan lama waktu rekaman dan jarak aktual yang ditempuh. Akurasi masing-masing percobaan diukur berdasarkan perbandingan antara luas sebenarnya dengan luas yang diukur melalui peta yang dibuat. Perbandingan yang dilakukan meliputi dua hal yaitu *uncovered area* dan *excessed area*. Berdasarkan evaluasi tersebut, didapati bahwa kinerja pemetaan lebih baik pada algoritma GFtT, yaitu dengan total *error* (*uncovered area* ditambah *excessed area*) sebesar 8.18 m², di mana FAST memberikan total error sebesar 8.27 m², dan Harris sebesar 8.54 m².

5.2 Saran

Setelah melakukan penelitian dan berbagai uji coba dalam menyusun tesis ini, peniliti ingin memberikan beberapa saran sebagai berikut:

1. Bedasarkan masukan dari Bapak Muhammad Attamimi, B.Eng., M.Eng., Ph.D., karena kamera RGB-D memiliki informasi *depth*, maka dimungkinkan untuk membuat *point cloud* dari sebuah *scene*. Apabila *point cloud* ke $(n-1)$ dan *point cloud* ke n didapatkan, maka dapat digunakan

metode *Iterative Closest Point* (ICP) untuk memperkirakan pergerakan baik rotasi maupun translasi dari kamera.

2. Proses perkiraan *heading* pada penelitian ini adalah berbasis fitur *corner*, sehingga adanya pencahayaan ruangan yang baik, serta banyaknya dekorasi ruangan sangat berperan penting. Algoritma ICP dapat bekerja hanya dengan informasi *depth*, sehingga tidak terganggu dengan hal-hal yang tersebut. Adapun jika kamera RGB-D tidak mendapatkan informasi *depth*, maka pergerakan kamera tidak dapat diperkirakan. Maka dari itu mungkin penggabungan antara metode ICP dan fitur *corner* dapat dilakukan.

DAFTAR PUSTAKA

[1] L. Landers, C. F. Timoney, and R. A. Felder, "Medical Mobile Robotics: An Industry Update," *J. Lab. Autom.*, vol. 5, no. 3, pp. 26–29, 2000.

[2] J. Khurshid and H. Bing-rong, "A Glimpse from Today and Tomorrow," *Control*, no. December, pp. 771–777, 2004.

[3] A. Grau, M. Indri, L. Lo Bello, and T. Sauter, "Industrial robotics in factory automation: From the early stage to the Internet of Things," *Proc. IECON 2017 - 43rd Annu. Conf. IEEE Ind. Electron. Soc.*, vol. 2017–Janua, pp. 6159–6164, 2017.

[4] P. Afonso, J. Azevedo, C. Cardeira, B. Cunha, P. Lima, and V. Santos, "Challenges and Solutions in an Autonomous Driving Mobile Robot Competition," *Proc. Control. - 7th Port. Conf. Autom. Control*, no. December 2013, 2006.

[5] D. Purwanto, M. Rivai, and H. Soebhakti, "Vision-based multi-point sensing for corridor navigation of Autonomous Indoor Vehicle," *ICECOS 2017 - Proceeding 2017 Int. Conf. Electr. Eng. Comput. Sci. Sustain. Cult. Herit. Towar. Smart Environ. Better Futur.*, pp. 67–70, 2017.

[6] F. Fang, X. Ma, and X. Dai, "A multi-sensor fusion SLAM approach for mobile robots," *Mechatronics Autom. 2005 ...*, no. 2002, pp. 1837–1841, 2005.

[7] K. Yousif, A. Bab-Hadiashar, and R. Hoseinnezhad, "An Overview to Visual Odometry and Visual SLAM: Applications to Mobile Robotics," *Intell. Ind. Syst.*, vol. 1, no. 4, pp. 289–311, 2015.

[8] R. Mardiyanto, J. Anggoro, and F. Budiman, "2D map creator for robot navigation by utilizing Kinect and rotary encoder," *2015 Int. Semin. Intell. Technol. Its Appl. ISITIA 2015 - Proceeding*, pp. 81–84, 2015.

[9] A. A. Mane, M. N. Parihar, S. P. Jadhav, and B. B. Digey, "Robotics based simultaneous localization and mapping of an unknown environment using Kalman Filtering," *NUiCONE 2015 - 5th Nirma Univ. Int. Conf. Eng.*, 2016.

[10] L. Kneip, M. Chli, and R. Siegwart, "Robust Real-Time Visual Odometry with a Single Camera and an IMU," *Procedings Br. Mach. Vis. Conf. 2011*, p. 16.1-16.11, 2011.

[11] A. C. Murtra and J. M. Mirats Tur, "IMU and cable encoder data fusion for in-pipe mobile robot localization," *IEEE Conf. Technol. Pract. Robot Appl. TePRA*, 2013.

[12] Davison, "Real-time simultaneous localisation and mapping with a single camera," *Proc. Ninth IEEE Int. Conf. Comput. Vis.*, pp. 1403–1410 vol.2, 2003.

[13] I. Mahon, S. B. Williams, O. Pizarro, and M. Johnson-Roberson, “Efficient view-based SLAM using visual loop closures,” *IEEE Trans. Robot.*, vol. 24, no. 5, pp. 1002–1014, 2008.

[14] C. Stachniss, “Robot Mapping A Short Introduction to Homogeneous Coordinates.” .

[15] S. R. and R. Illah, *Introduction to Autonomous Mobile Robots*, 2nd ed. Cambridge, 2014.

[16] R. Hartley and Z. Andrew, *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2004.

[17] K. Baker, “Singular Value Decomposition Tutorial.” pp. 1–24, 2011.

[18] “MATLAB Documentation.” [Online]. Available: https://www.mathworks.com/help/matlab/. [Accessed: 19-Dec-2018].

[19] D. Nistér, “An efficient solution to the five-point relative pose problem,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 6, pp. 756–770, 2004.

[20] M. Satya, “Rotation Matrix To Euler Angles | Learn OpenCV,” 2016. [Online]. Available: https://www.learnopencv.com/rotation-matrix-to-euler-angles/. [Accessed: 16-Dec-2018].

[21] “File:Eulerangles.svg - Wikipedia.” [Online]. Available: https://en.wikipedia.org/wiki/File:Eulerangles.svg. [Accessed: 19-Dec-2018].

[22] E. Rosten and T. Drummond, “Fusing points and lines for high performance tracking,” *Proc. IEEE Int. Conf. Comput. Vis.*, vol. II, pp. 1508–1515, 2005.

[23] R. Rojas, “Lucas-Kanade in a Nutshell,” *Freie Universit at Berlinn, Dept. of Computer Science, Tech. Rep*. pp. 121–130, 1981.

[24] J. Welsh and R. Stewart, “Computer Vision Final Project: Kinect.” [Online]. Available: http://www.cs.virginia.edu/~jfw3x/cs4501/. [Accessed: 16-Dec-2018].

[25] C. D. Mutto and P. Zanuttigh, *Time-of-Flight Cameras and Microsoft Kinect TM*. 2013.

[26] R. Smeenk, “Kinect V1 and Kinect V2 fields of view compared - Roland Smeenk,” 2014. [Online]. Available: https://smeenk.com/kinect-field-of-view-comparison/. [Accessed: 19-Dec-2018].

LAMPIRAN

1. **Fungsi pendeteksi *corner***

```
function corn = detCorners(img, alg)

if alg == 1
   corn = detectFASTFeatures(img,...
      'MinQuality', 0.04,...
      'MinContrast', 0.025);
elseif alg == 2
   corn = detectMinEigenFeatures(img,...
      'MinQuality', 0.03);
elseif alg == 3
   corn = detectHarrisFeatures(img,...
      'MinQuality', 0.0005);
end

end
```

2. **Fungsi estimasi *heading***

```
function [rotDeg, relLoc, valid] = estRotOptFlow(oldMatchedPoints,...
   newMatchedPoints,...
   cameraParams)

[E, epipolarInliers] = estimateEssentialMatrix(oldMatchedPoints,...
   newMatchedPoints,...
   cameraParams,...
   'Confidence', 99.99,...
   'MaxNumTrials', 500,...
   'MaxDistance', 3.2);

inlierPoints1 = oldMatchedPoints(epipolarInliers, :);
inlierPoints2 = newMatchedPoints(epipolarInliers, :);

[orient, relLoc, valid] = relativeCameraPose(E, cameraParams,...
   inlierPoints1, inlierPoints2);
rotDeg = rad2deg(rotm2eul(orient));

end
```

3. **Fungsi konversi citra *depth* ke dalam bentuk *laser scan***

```
function lsOut = lsDepth(dimg, r1, r2, dWidth, lengthUnit,...
   lsVertAnglesSec)

dimgCropped = dimg(r1:r2, 1:dWidth);
```

```
meanDimgCropped = mean(dimgCropped) / lengthUnit;
lsOut = times(meanDimgCropped, lsVertAnglesSec);

end
```

4. **Fungsi inisialisasi konversi citra *depth* ke dalam bentuk *laser scan***

```
function [r1, r2, dWidth, lengthUnit, lsVertAngles, lsVertAnglesSec] =...
   lsDepthInit(lsParams)

% lsParameters(1) = lengthUnit
% lsParameters(2) = depthWidth
% lsParameters(3) = depthHeight
% lsParameters(4) = depthVertViewAngle
% lsParameters(5) = percentage
% lsParameters(6) = verticalAlignment

dWidth = lsParams(2);
lengthUnit = lsParams(1);

dRoiHeight = lsParams(3) * lsParams(5) / 100;
lsVertAngles = linspace(deg2rad(...
   -lsParams(4) / 2), deg2rad(...
   lsParams(4) / 2), lsParams(2));
lsVertAnglesSec = sec(lsVertAngles);

if lsParams(6) == 1
   r1 = 1;
   r2 = dRoiHeight;
elseif lsParams(6) == 2
   r1 = (lsParams(3) / 2) - (dRoiHeight / 2);
   r2 = (lsParams(3) / 2) + (dRoiHeight / 2);
elseif lsParams(6) == 3
   r1 = lsParams(3) - (dRoiHeight - 1);
   r2 = lsParams(3);

end
```

5. **Fungsi pemetaan saat gerak maju**

```
function [mapOut, posXOut, posYOut, thetaOut] = mapFwd(vC, vD,
posX, posY, theta, way, lsParams, mapIn)

[~, ~, ~, iter] = size(vC);

[r1, r2, dWidth, lengthUnit, lsVertAngles, lsVertAnglesSec] = ...
   lsDepthInit(lsParams);
```

```
% oldD = vD(:, :, 250);
oldD = vD(:, :, 1);
coldD = oldD(:, 281:360);

% lsOut = lsDepth(vD(:, :, 1), r1, r2, dWidth, lengthUnit,...
%         lsVertAnglesSec);
% insertRay(mapIn, [posX, posY, 0], lsOut, lsVertAngles,
mapIn.Resolution);

% oldPose = [posX, posY, 0];

% for i = 250:1:iter
for i = 2:1:iter
%     lsOut = lsDepth(vD(:, :, i) * 0, r1, r2, dWidth, lengthUnit,...
%         lsVertAnglesSec);
%     insertRay(mapIn, oldPose, lsOut, lsVertAngles, mapIn.Resolution);
    newD = vD(:, :, i);
    cnewD = newD(:, 281:360);

    %correction holds the binary values 0 and 1 corresponding to each
pixel. If pixel depth %is >0 then the binary value is 1 otherwise it is 0
    correction = (coldD > 0);

    %correction holds the binary values 0 and 1 corresponding to each pixel
in both the %images imda and imdb. If pixel depth is >0 in both the
images then the binary value is 1
    %otherwise 0
    correction = correction & (cnewD > 0);

    %type conversion from logical array to integer array
    correction = uint16(correction);

    %imda_new and imdb_new hold the actual depth value of only those
pixels whose depth in %both the images imd1 and imd2 is >0. The binary
value is 1 iff the corresponding entry %in the correction matrix is also 1
    coldDC = coldD .* correction;
    cnewDC = cnewD .* correction;

    %subtract the depth value to get the distance traversed by the bot in the
forward %direction
    depth_diff = coldDC - cnewDC;

    sumb = sum(sum(correction, 1), 2);
    suma = sum(sum(depth_diff, 1), 2);
```

```
    if sumb ~= 0
        translation = suma / sumb;
        if translation < 10.0
            translation = 0;
        end
    else
        translation = 0;
    end

    if theta == 90
        pose = [posX, posY + (translation / 1000), deg2rad(theta)];
        disp(translation);
    elseif theta == 0
        pose = [posX + (translation / 1000), posY, deg2rad(theta)];
    elseif theta == -90
        pose = [posX, posY - (translation / 1000), deg2rad(theta)];
    end
    lsOut = lsDepth(vD(:, :, i), r1, r2, dWidth, lengthUnit,...
        lsVertAnglesSec);
    insertRay(mapIn, pose, lsOut, lsVertAngles, mapIn.Resolution);

    oldPose = pose;
    %     oldD = newD;
    disp(i);
end

if theta == 90
    posXOut = posX;
    posYOut = posY + (translation / 1000);
elseif theta == 0
    posXOut = posX + (translation / 1000);
    posYOut = posY;
elseif theta == -90
    posXOut = posX;
    posYOut = posY - (translation / 1000);
else
    posXOut = posX - (translation / 1000);
    posYOut = posY;
end

thetaOut = 0;

mapOut = mapIn;
```

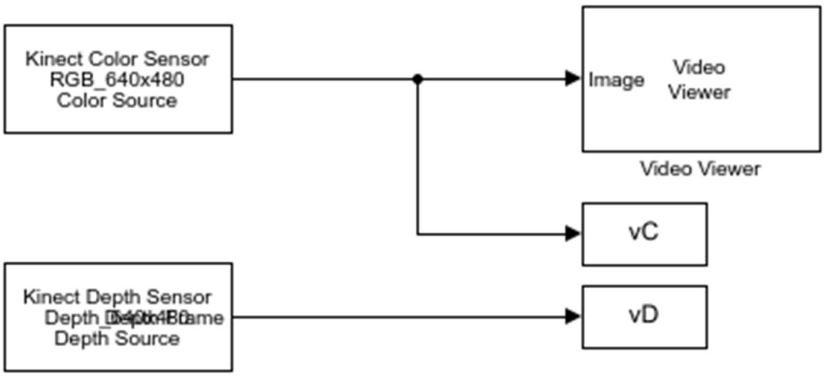
```
end
```

6. **Fungsi pemetaan saat gerak berputar**

```
function [mapOut, posXOut, posYOut, thetaOut] = mapRot(vC, vD, posX,
posY, theta, dDevice, camParams, ...
   lsParams, mapIn, fac)

[~, ~, ~, iter] = size(vC);

tracker = vision.PointTracker('NumPyramidLevels', 11,...
   'MaxBidirectionalError', 1,...
   'BlockSize', [11, 11],...
   'MaxIterations', 50);

[r1, r2, dWidth, lengthUnit, lsVertAngles, lsVertAnglesSec] = ...
   lsDepthInit(lsParams);

roi = [45, 35, 640 - 75, 480 - 50];

oldImg = []; oldCorn = []; oldCornLoc = []; oldDimg = [];
img = []; corn = []; cornLoc = []; dimg = [];
sumRot = [0, 0, 0];
getNewCorn = false;

for i = 1:1:iter
   if mod(i, 20) == 0 || i == 1 || getNewCorn
      getNewCorn = false;
      if i > 1
         oldImg = vC(:, :, :, i - 1);
         oldDimg = vD(:, :, i - 1);
      else
         oldImg = vC(:, :, :, i);
         oldDimg = vD(:, :, i);
      end

      [oldImg, oldDimg] = regis(dDevice, oldImg, oldDimg);
      oldImg = undCrop(oldImg, roi, camParams);
%        oldImg = histeq(oldImg);
      oldDimg = undCrop(oldDimg, roi, camParams);

      oldCorn = detCorners(rgb2gray(oldImg), 1);
      oldCornLoc = oldCorn.Location;
      release(tracker);
      initialize(tracker, oldCornLoc, oldImg);
   end
```

```
    img = vC(:, :, :, i);
    dimg = vD(:, :, i);
    [img, dimg] = regis(dDevice, img, dimg);
    img = undCrop(img, roi, camParams);
%     img = histeq(img);
    dimg = undCrop(dimg, roi, camParams);

    [cornLoc, matchedId] = step(tracker, img);
    oldMatchedCorn = oldCornLoc(matchedId, :);
    matchedCorn = cornLoc(matchedId, :);
    [rotDeg, relLoc, valid] = estRotOptFlow(oldMatchedCorn,...
        matchedCorn, camParams);

    % If matched points that are being tracked by optical flow is too few,
    % detect new corners.
    totalMatchedPoints = size(matchedCorn);
    if totalMatchedPoints(1) < 30
        getNewCorn = true;
    end

    % Since the frames were taken when the robot was moving smoothly,
then
    % any estimation of rotation that exceed +/- 5 degree can be ignored.
    if ~any(abs(rotDeg) > 5)
        sumRot = sumRot + rotDeg * fac
        oldCornLoc = cornLoc;
        oldDimg = dimg;
    end

    pose = [posX, posY, deg2rad(theta) + deg2rad(sumRot(2))];
    lsOut = lsDepth(vD(:, :, i), r1, r2, dWidth, lengthUnit,...
        lsVertAnglesSec);
    insertRay(mapIn, pose, lsOut, lsVertAngles, mapIn.Resolution);
    disp(i);
end

posXOut = posX;
posYOut = posY;
thetaOut = deg2rad(sumRot(2));
mapOut = mapIn;

end
```

7. *Project* Simulink untuk merekam *frame* dari Kinect



Halaman ini sengaja dikosongkan