Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR - EE 184801

NAVIGASI MOBILE ROBOT DARAT MENGGUNAKAN ODOMETRI VISUAL BERBASIS CITRA STEREO

Ilham Wicaksono
NRP 07111540000063

Dosen Pembimbing
Mochammad Sahal, ST., M.Sc.
Ir. Rusdianto Effendi A.K., MT.

DEPARTEMEN TEKNIK ELEKTRO
Fakultas Teknologi Elektro
Institut Teknologi Sepuluh Nopember
Surabaya 2019

[Halaman ini sengaja dikosongkan]

TUGAS AKHIR - EE 184801

NAVIGASI MOBILE ROBOT DARAT MENGGUNAKAN ODOMETRI VISUAL BERBASIS CITRA STEREO

Ilham Wicaksono
NRP 07111540000063

Dosen Pembimbing
Mochammad Sahal ST., M.Sc.
Ir. Rusdianto Effendi A.K., MT.

DEPARTEMEN TEKNIK ELEKTRO
Fakultas Teknologi Elektro
Institut Teknologi Sepuluh Nopember
Surabaya 2019

[Halaman ini sengaja dikosongkan]

FINAL PROJECT - EE 184801

GROUND MOBILE ROBOT NAVIGATION USING STEREO VISION BASED VISUAL ODOMETRY

Ilham Wicaksono
NRP 07111540000063

Supervisor
Mochammad Sahal ST., M.Sc.
Ir. Rusdianto Effendi A.K., MT.

DEPARTMENT OF ELECTRICAL ENGINEERING
Faculty of Electrical Technology
Institut Teknologi Sepuluh Nopember
Surabaya 2019

[Halaman ini sengaja dikosongkan]

PERNYATAAN KEASLIAN
TUGAS AKHIR

Dengan ini saya menyatakan bahwa isi sebagian maupun keseluruhan Tugas Akhir saya dengan judul

"Navigasi Mobile Robot Darat Menggunakan Odometri Visual Berbasis Citra Stereo"

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 pada daftar pustaka. Apabila ternyata pernyataan ini tidak benar, saya bersedia menerima sanksi sesuai peraturan yang berlaku.

Surabaya, 3 Juli 2019

Ilham Wicaksono
Nrp 07111540000063

[Halaman ini sengaja dikosongkan]

NAVIGASI MOBILE ROBOT DARAT MENGGUNAKAN ODOMETRI VISUAL BERBASIS CITRA STEREO

TUGAS AKHIR

Diajukan Guna Memenuhi Sebagian Persyaratan
Untuk Memperoleh Gelar Sarjana Teknik
Pada
Bidang Studi Teknik Sistem Pengaturan
Departemen Teknik Elektro
Institut Teknologi Sepuluh Nopember

Menyetujui

Dosen Pembimbing I,	Dosen Pembimbing II,
Mochammad Sahal, S.T., M.Sc.	Ir. Rusdhianto Effendi A. K., MT.
NIP. 197011191998021002	NIP. 195704241985021001

DEPARTEMEN TEKNIK ELEKTRO

SURABAYA
JULI 2019

[Halaman ini sengaja dikosongkan]

NAVIGASI MOBILE ROBOT DARAT MENGGUNAKAN ODOMETRI VISUAL BERBASIS CITRA STEREO

Ilham Wicaksono
07111540000063

Dosen Pembimbing I: Mochammad Sahal ST., M.Sc.
Dosen Pembimbing II: Ir. Rusdianto Effendi A K., MT.

ABSTRAK

Navigasi adalah bagian terpenting pada sistem mobile robot untuk melaksanakan tugasnya. Salah satu solusi dalam menyelesaikan masalah navigasi adalah dengan menggunakan citra 3d. Sensor aktif yang dapat menghasilkan 3d, contohnya LiDAR, memiliki sistem yang rumit dan sangat mahal. Di lain pihak, dengan menggunakan sebuah divais kamera stereo, citra 3d dapat dihasilkan, dengan biaya yang jauh lebih murah. Dengan menggunakan proses visual odometry, informasi dari kamera 3d dapat diolah dan menghasilkan estimasi posisi dari robot. Hal ini dilakukan dengan memanfaatkan algoritma deteksi fitur dan traking fitur. Dari pergerakan fitur-fitur ini, dapat diestimasi pergerakan dan posisi akhir robot. Hasil dari pengujian sistem ini menunjukkan bahwa proses *stereo vision* berbasis dua *webcam*, dan diproses dengan prosesor i5-5200U 2.20 GHz, memungkinkan untuk digunakan pada penggunaan waktu riil dengan refresh rate hasil rektifikasi dan disparity bernilai 29 FPS. Deteksi fitur dan triangulasi fitur pun menunjukkan hasil yang baik dengan rata-rata 61 *inlier* terdeteksi dan *error* triangulasi fitur hanya 0.06 Meter. Namun hasil pengujian *visual odometry* masih menunjukkan *error* pada setiap pengujiannya yaitu 1.20 Meter, 2.21 Meter, dan 2.04 Meter.

Kata Kunci: Navigasi, *Visual Odometry*, *Mobile Robot*, Citra Stereo, Kamera Stereo.

[Halaman ini sengaja dikosongkan]

GROUND MOBILE ROBOT NAVIGATION USING STEREO VISION BASED VISUAL ODOMETRY

Ilham Wicaksono
07111540000063

Supervisor I: Mochammad Sahal ST., M.Sc.
Supervisor II: Ir. Rusdianto Effendi A K., MT.

ABSTRACT

Navigation is the most fundamental part on the mobile robot's system to complete its tasks. One solution to solve navigation problem is to use 3D vision. Active sensor that could produce 3D information, e.g. LiDAR, have a complex system and not affordable. On the other hand, using stereo camera device, 3D vision could also produced, with affordable price. Usin the visual odometry process, 3D camera information can be processed and produce the estimation position of a robot. This could be reach using feature detection and feature tracking algorithm. From these features motions, the motion and the position of the robot could be estimated. Results of the test conducted shows that stereo vision process based on two webcam, and processed on i5-5200U, 2.20 GHz processor, could be used on real time application with the refresh rate of rectified image and disparity map stable on 29 FPS. Feature detection and feature triangulation also show a good result with average of 61 inlier features detected and triangulation only has 0.06 meter error. However, the visual odometry tests conducted show error of 1.20 Meter, 2.21 Meter, and 2.04 Meter on each of the tests conducted.

Keywords: *Navigation, Visual Odometry, Mobile Robot, Stereo Vision, Stereo Camera.*

[Halaman ini sengaja dikosongkan]

KATA PENGANTAR

Puji syukur penulis panjatkan kehadirat Allah SWT, berkat rahmat dan karunia-Nya, penulis dapat menyelesaikan laporan Tugas Akhir berjudul "**Navigasi Mobile Robot Darat Menggunakan Odometri Visual Berbasis Citra Stereo**" untuk memenuhi syarat kelulusan pada Bidang Studi Teknik Sistem Pengaturan, Departemen Teknik Elektro, Fakultas Teknologi Elektro, Institut Teknologi Sepuluh Nopember, Surabaya.

Tugas Akhir ini tidak mungkin terselesaikan tanpa adanya izin dari Allah SWT. Penulis mengucapkan terima kasih pula kepada kedua orang tua penulis yang dengan dorongan motivasi moral, bantuan material, dan doa yang tiada henti dipanjatkan. Terima kasih kepada Bapak Mochammad Sahal ST., M.Sc dan Bapak Rusdianto Effendi AK MT. selaku dosen pembimbing atas masukan, arahan dan ilmu yang disalurkan kepada penulis dalam menyelesaikan buku laporan ini. Terima kasih juga penulis sampaikan kepada dosen-dosen Bidang Studi Teknik Sistem Pengaturan atas ilmu-ilmu yang disalurkan, baik melalui perkuliahan maupun non-perkuliahan. Terakhir penulis juga mengucapkan terima kasih kepada keluarga Tim Robotika ITS, baik dosen pembimbing, senior penulis dan junior penulis, serta keluarga Laboratorium Sistem dan Sibernetik AJ204 beserta teman-teman lain yang telah membantu serta berjuang bersama-sama penulis.

Penulis berharap buku laporan ini dapat memberikan manfaat kepada pembaca, baik secara langsung maupun tak langsung. Laporan ini memang masih jauh dari sempurna, oleh karenanya penulis mengharapkan kritik dan saran yang membangun dari pembaca.

Surabaya, 26 Mei 2019

Penulis

[Halaman ini sengaja dikosongkan]

DAFTAR ISI

[Halaman ini sengaja dikosongkan]

DAFTAR GAMBAR

[Halaman ini sengaja dikosongkan]

DAFTAR TABEL

[Halaman ini sengaja dikosongkan]

BAB 1
PENDAHULUAN

1.1. LATAR BELAKANG

Penggunaan *mobile robot* pada sektor industri meningkat akhir-akhir ini. *Mobile robot* didefinisikan sebagai sebuah robot, yaitu alat dengan mekanisme otomatis dan intelegensi sederhana, namun memiliki kemampuan khusus untuk berpindah tempat dari titik A ke titik B. Hal ini kontras dengan sebuah *Static robot*, yaitu robot yang tidak dapat atau hanya memiliki kemampuan sedikit untuk berpindah tempat. *Mobile robot* memiliki banyak varisai, sesuai lingkungan dimana dia bergerak. Dalam penggunaan darat, biasa dipakai robot beroda atau AGV (*Autonomous Ground Vehicle*) atau robot berkaki (*legged robot*). Pada penggunaan diatas permukaan air biasa digunakan sebuah kapal yang memiliki otonomi biasa disebut USV (*Unmanned Surface Vehicle).* Pada penggunaan udara biasa digunakan UAV (*Unmanned Aerial Vehicle*) dalam bentuk pesawat *fixed wing* ataupun *multirotor* VTOL (*Vertical Take-Off & Landing).*

Untuk robot beroda sudah umum menggunakan *wheel odometry + gyroscope* dimana navigasi menggunakan perhitungan putaran roda dibantu *gyroscope* untuk sudut heading-nya. Hal ini menghasilkan apa yang dinamakan navigasi lokal, yaitu estimasi posisi robot relatif terhadapat posisi awalnya (bukan relatif suatu titik global). Untuk robot jenis lain yang tidak beroda biasa digunakan GNSS dan INS, navigational INS yang dapat digunakan untuk *position estimation* sangatlah mahal, menjadikan GNSS satu-satunya sistem navigasi yang mumpuni untuk waktu lama dengan harga terjangkau. *Wheel odometry + gyroscope* memiliki kelemahan pada selip roda pada penggunaan di permukaan sembarang. Hal ini dapat terjadi pada penggunaan navigasi yang sangat lama, dan penggunaan robot pada permukaan yang tidak sesuai dengan karakteristik roda dan suspensi. Sedangkan GNSS membutuhkan koneksi yang mumpuni kepada satelit penyedia layanan GNSS, hal ini akan bermasalah pada daerah yang tertutup, dalam ruangan, disekitar gedung-gedung tinggi, ataupun di daerah yang tertutup lainnya.

Untuk menyelesaikan permasalahan diatas, akhir-akhir ini digunakan teknologi LiDAR untuk mendapatkan citra 3d yang

penggunaan untuk perhitungan *odometry* menggunakan metode *Visual Odometry*, atau metode SLAM (*Simultaneous Localization And Mapping*) dan variasinya. Permasalahan yang dihadapi dari sensor LiDAR adalah harga yang tersedia di pasaran untuk LiDAR yang mumpuni sangatlah mahal dan rawan kerusakan karena sistem mekanisnya yang bergerak terus menerus. Selain itu penggunaan LiDAR pun rawan distorsi cahaya-cahaya di sekitar lingkungan robot.

Alternatif lain untuk mendapatkan citra 3 dimensi adalah menggunakan metode *stereo-vision*, penggunaan dua kamera yang telah di kalibrasi, lalu mendapatkan citra 3 dimensi dari *disparity map*. Cara kerjanya mirip dengan cara kerja mata manusia. Dengan harga kamera yang saat ini sudah sangat murah, sistem ini berpotensi untuk menggantikan sensor LiDAR untuk keperluan navigasi dan sensing pada mobile robot.

1.2. PERUMUSAN MASALAH

Berikut ini adalah rumusan masalah pada tugas akhir ini:

1. Bagaimana mendapatkan citra dari dua kamera webcam, dan mengolahnya menjadi informasi *disparity map* menggunakan platform Robot Operating System dan library OpenCV.
2. Bagaimana mendapatkan data navigasi dari data *disparity map* tersebut menggunakan teknik *visual odometry* pada platform Robot Operating System dan library OpenCV.

1.3. BATASAN MASALAH

Batasan-batasan masalah dari tugas akhir ini adalah sebagai berikut ini:

- Pergerakan *mobile robot* berkarakter *non-holonomic*, terbatas pada satu sumbu translasi yang searah dengan hadapnya kamera, dan satu sumbu rotasi yang tegak lurus kearah atas kamera.
- Pergerakan *mobile robot* pada kedua sumbu geraknya yang telah disebutkan dilakukan relatif terpisah, tidak bersamaan.
- Nilai rotasi dari IMU dianggap tanpa *drift* dan kesalahan lain pada pengujian jalan.
- Terdapat benda-benda yang berbeda dan dapat dideteksi di depan kamera.

- Mayoritas benda-benda di depan kamera adalah benda-benda yang tidak bergerak terhadap satu sama lain.
- Jarak pandang kamera lebih dari 4 meter.

1.4. TUJUAN TUGAS AKHIR DAN MANFAAT

Tujuan yang ingin dicapai pada tugas akhir ini adalah bagaimana mendapatkan *disparity map* dari citra 2 kamera, lalu mendeteksi dan mencocokkan fitur-fitur dari frame-frame nya, seteleh itu mengestimasi perubahan posisi robot dari perubahan posisi fitur-fitur ini, serta mengimplementasikannya menjadi program siap pakai pada platform *Robot Operating System*.

Manfaat dari tugas akhir ini adalah mahasiswa dapat berkontribusi membuat modul perangkat lunak dan set perangkat keras yang siap digunakan untuk navigasi pada mobile robot yang menggunakan *Robot Operating System*.

1.5. METODOLOGI

Penelitian ini akan dilakukan melalui beberapa tahap, yaitu dijelaskan sebagai berikut:

1.5.1. Desain Penelitian

Pada penelitian ini digunakan *true experimental designs*. Ini berarti desain penelitian berupaya untuk mengontrol setiap variabel dan mengamati efek ketika variabel dimanipulasi. Penelitian ini memiliki subjek berupa *mock mobile robot* atau dapat dikatakan sebuah wahana yang secara bentuk menyerupai *mobile robot* beroda, namun tidak memiliki sistem yang menjadikannya sebuah *mobile robot*, dan objek berupa sistem *stereo vision* yang masing-masing didalamya memiliki beberapa variabel.

Hipotesis (prediksi) diformulasikan terlebih dahulu sebelum melakukan eksperimen untuk menentukan variabel-variabel apa saja yang digunakan untuk pengujian dan bagaimana variabel tersebut diukur. Variabel dalam penelitian ini meliputi kamera 1, kamera 2, dan komponen-komponen perangkat lunak di dalam sistem yang akan dirancang terlebih dahulu, yang dapat dijadikan variabel untuk menguji hipotesis.

1.5.2. Tipe Penelitian

Tugas akhir ini memiliki tipe *applied* dan *empirical*. *Applied* karena hasil dari tugas akhir ini dapat diaplikasikan langsung kepada masalah yang terjadi di dunia nyata, dan berupa sebuah sistem yang memiliki bentuk nyata, bukan simulasi. Dan *empirical* karena semua berbasiskan data yang diukur yang divalidasi dengan teori yang ada.

1.5.3. Pendekatan Penelitian

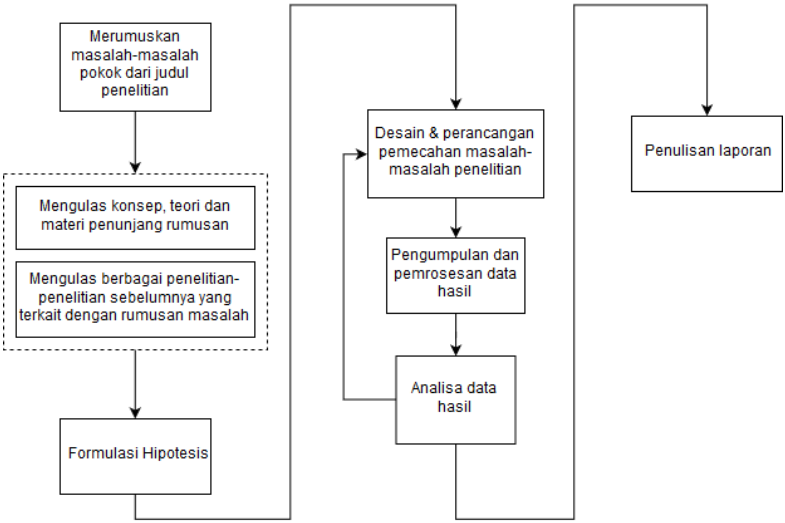
Pendekatan penelitian ini adalah pendekatan kuantitatif. Pendekatan kuantitatif melibatkan pembuatan data yang bersumber dari perhitungan beberapa sub pendekatan yang lain diantaranya *inferential, experimental,* dan *simulation*.

1.5.4. Teknik Penelitian

Teknik penelitian tuga akhir ini menggunakan teknik laboratorium dengan menggunakan perangkat kamera stereo, PC, serta mikrokontroller.

1.5.5. Proses Penelitian

Diagram proses penelitian dapat digambarkan pada Gambar 1-1



Gambar 1-1 Proses penelitian tugas akhir

1.6. SISTEMATIKA PENELITIAN

Dalam buku tugas akhir ini, dipaparkan mengenai studi literatur, perancangan alat, dan pengujian alat serta hasil yang didapatkan dalam lima bab:

1. Pendahuluan

   Dalam bab ini diuraikan mengenai latar belakang, perumusan masalah, tujuan penelitian, metodologi penelitian, sistematika penulisan, serta relevansi dan penelitian
2. Teori Penunjang

   Dalam bab ini diuraikan segala teori dan hasil dari studi literatur sebagai dasar perancangan sistem dan alat
3. Perancangan Sistem

   Dalam bab ini diuraikan mengenai tahap-tahap perancangan sistem dan desain perangkat keras *stereo visual odometry*
4. Pengujian dan Analisis

   Dalam bab ini diuraikan hasil dari pengujian sistem yang telah dirancang serta analisa dari hasil pengujian terhadap keluaran yang diinginkan
5. Penutup

   Dalam bab ini dipaparkan mengenai kesimpulan penelitian tugas akhir ini dan saran-saran yang dapat digunakan untuk pengembangan *stereo visual odometry* lebih lanjut

1.7. RELEVANSI DAN MANFAAT PENELITIAN

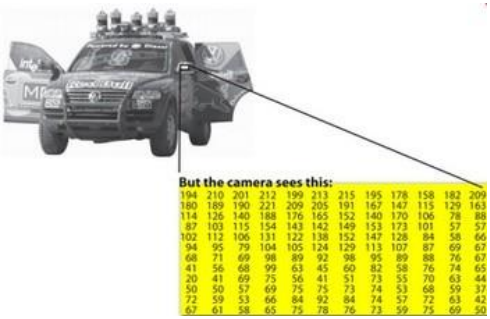
Citra stereo dari dua buah kamera yang telah diproses menjadi citra 3 dimensi dapat digunakan untuk mengestimasi posisi translasi dan rotasi dari robot. Desain hardware, akuisisi data, dan pengolahan citra ini menentukan kualitas estimasi posisi dari *ground truth*.

[Halaman ini sengaja dikosongkan]

BAB 2
DASAR TEORI

2.1. PENGOLAHAN GAMBAR DIGITAL & KAMERA

Untuk mendapatkan citra digital dari dunia nyata kita dapat melakukannya dengan berbagai cara seperti dengan kamera digital atau dengan scanner. Namun sebenarnya saat melakukan proses ini pada divais digital kita, yang dilakukan adalah menyimpan nilai numerik dari setiap titik pada gambar yang diterima.



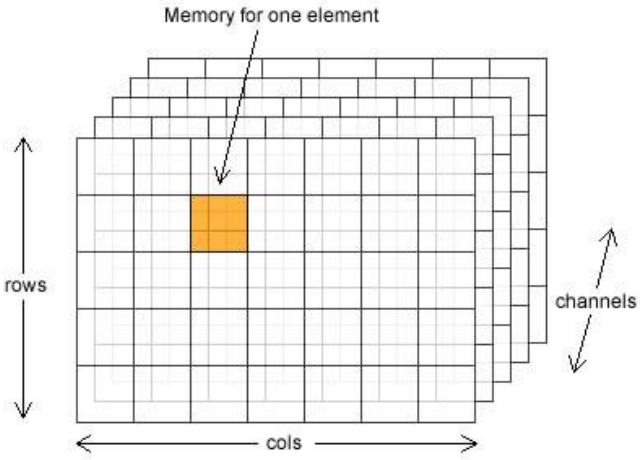
Gambar 2-1 Contoh nilai titik-titik pixel dari bagian gambar *grayscale*

Contohnya pada gambar diatas, gambar spion mobil dari gambar mobil diatas tidak lain adalah sebuah matriks yang berisi nilai intensitas dari titik-titik pixel. Cara untuk mendapatkan gambar digital bermacam-macam, namun pada akhirnya matriks ini lah yang didapat sebagai gambar digital. Matriks ini pula yang akan diolah, diproses, dan dimanipulasi pada bidang *computer vision* [1].

2.1.1. Gambar digital & warna

Dengan gambar digital tidak lain adalah sebuah matriks dengan setiap elemennya berisi nilai numerik dari setiap pixel gambar, pengolahan gambar tidak lain adalah pengolahan data matriks.

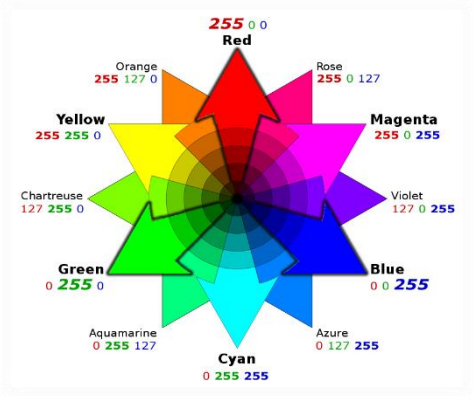
Representasi koordinat pixel dari gambar diperlihatkan pada gambar Gambar 2-2.



Gambar 2-2 Representasi gambar digital sebagai matriks 3 dimensi

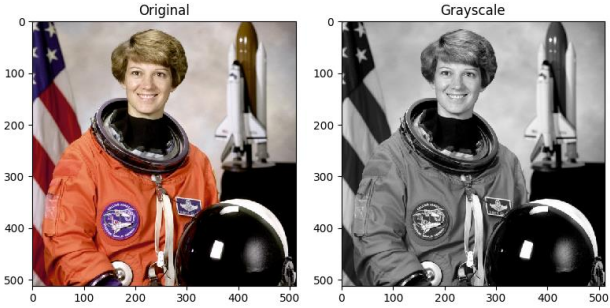
Dalam bidang computer vision, matriks gambar ini direpresentasikan sebagai matriks 2 dimensi ruang + 1 dimensi *channel*. Titik (0,0) pada matriks ini terletak pada pixel dengan posisi paling atas sebelah kiri, dengan koordinat x mengarah ke kanan (sebagai kolom) dan koordinat y mengarah kebawah (sebagai baris). 1 dimensi channel ini merepresentasikan data warna per pixel. Besarnya channel tergantung tipe warna gambar yang digunakan.

Salah satu tipe warna yang biasa digunakan adalah BGR/RGB. Tiap channel dari gambar dengan tipe warna ini merepresentasikan derajat warna biru, hijau, dan merah pada pixel tersebut. Kombinasi dari tiga warna ini akan diterjemahkan menjadi sebuah warna baru.



Gambar 2-3 Contoh kombinasi warna dari sistem BGR

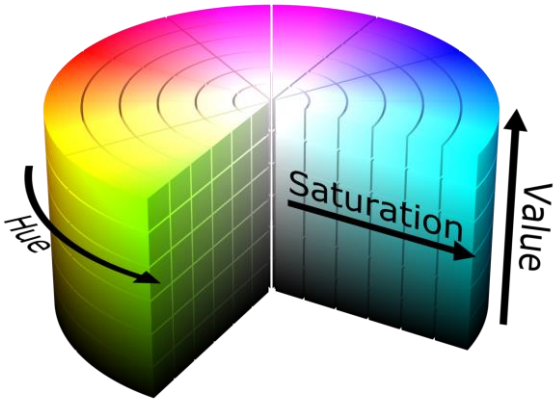
Tipe warna lain yang sering digunakan adalah 1 channel derajat keabu-abuan atau *grayscale*. Dalam sistem ini setiap pixel hanya memiliki satu channel yang merepresentasikan intensitas warna abu-abu pixel tersebut. Nilai dari intensitas ini dari 0 (hitam) sampai 255 (putih).



Gambar 2-4 Contoh representasi grayscale dari gambar berwarna

Adapula mode warna HSV (Hue, Saturation, Value) yang juga merepresentasikan warna namun dengan sistem yang berbeda dari BGR.

Pada mode ini terdapat 3 channel yang merepresentasikan Hue, yaitu warna pixel, Saturation, yaitu derajat keterangan pixel, dan Value derajat kegelapan pixel.

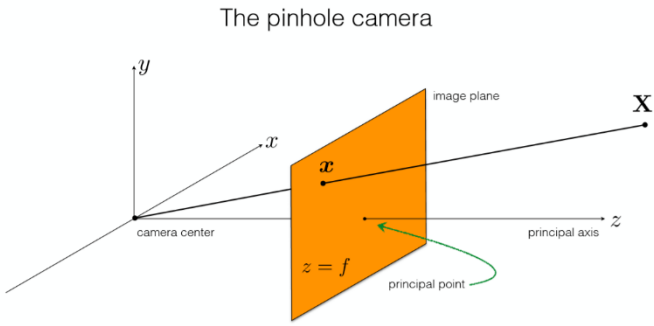


Gambar 2-5 Sistem warna HSV

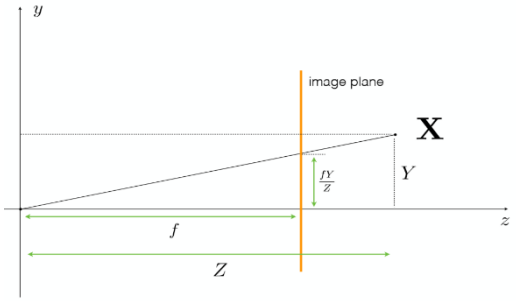
2.1.2. Model Kamera

Pada dasarnya sebuah kamera bekerja dengan konsep memetakan antara dunia 3d (ruang objek) dan gambar 2d. Pada [2] dijelaskan bahwa berbagai model kamera dapat dimodelkan yang menggambarkan sifat kamera tersebut dalam sebuah matriks yang menggambarkan pemetaan dari kamera tersebut. Model dasar kamera yang biasa dipakai adalah *pinhole camera model.*

Model kamera pinhole diilustrasikan pada gambar Gambar 2-6. Pusat proyeksi pada kamera dianggap berada pada titik origin dari sebuah sistem koordinat euclidean. Pada posisi $z = f$ terdapat sebuah bidang gambar atau bidang titik api (*focal plane*). Pada model kamera *pinhole*, sebuah titik $\mathrm{X} = (\mathrm{x}, \mathrm{y}, z)^T$ dalam ruang koordinat ini, dipetakan pada titik pada bidang gambar dimana sebuah garis yang menghubungkan origin/pusat proyeksi kepada titik X melewati/menembus bidang gambar. Dari deskripsi ini dapat dihitung posisi suatu titik $(\mathrm{x}, \mathrm{y}, \mathrm{z})^T$ dipetakan pada titik $\left(\frac{f\mathrm{x}}{\mathrm{z}}, \frac{f\mathrm{y}}{\mathrm{z}}, f\right)^T$.



Gambar 2-6 Ilustrasi model kamera *pinhole*



Gambar 2-7 Proyeksi titik pada bidang gambar

Perhitungan ini menghasilkan yang disebut matriks karakteristik kamera. Matriks ini pada dasarnya memetakan titik X koordinat euclidean kepada titik x dalam bidang gambar. Dengan menambahkan parameter offset *principal point*, yaitu bilamana titik origin koordinat tidak berada pada principal point, matriks ini ditulis sebagai matriks K.

$$K = \begin{bmatrix} f_x & 0 & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \quad \textbf{(2-1)}$$

Dimana,

- f_x, f_y adalah focal length dari kamera
- p_x, p_y adalah offset *principal point* kamera

Dengan matriks K ini, maka proyeksi titik X kepada titik x dalam bidang gambar adalah:

$$x = K \cdot X \quad \textbf{(2-2)}$$

Pada [2] matriks K juga disebut sebagai matriks kalibrasi kamera.

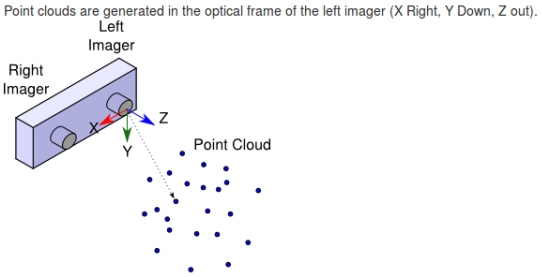
2.1.3. Optical Frame Kamera

Pada pengolahan citra komputer, digunakan sebuah sistem koordinat yang sudah disepakati untuk mendeskripsikan posisi suatu titik terhadap kamera. Untuk citra 3d, biasa digunakan sebuah sistem koordinat yang disebut *optical frame*. Hal ini digambarkan pada gambar Gambar 2-8.

Sistem koordinat ini digambarkan dengan sumbu x mengarah ke kanan, sumbu y ke bawah, dan sumbu z ke depan kamera. Origin dari sistem koordinat ini terletak pada *optical center* dari kamera. Sistem ini biasa digunakan untuk mendeskripsikan posisi titik-titik pada dunia 3d, biasa disebut *pointcloud*, terhadap kamera/sensor yang digunakan.

2.2. STEREO VISION & CITRA 3 DIMENSI

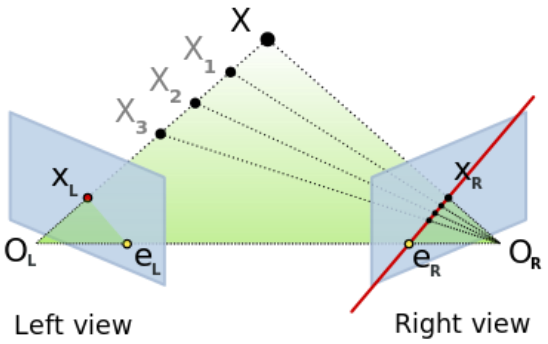
Stereo Vision secara harafiah dapat diartikan sebagai pengelihatan solid [3]. Pada tugas akhir ini stereo vision dimaknai sebagai proses pengolahan citra dari dua kamera, yang berbeda posisi dan sudut pandangnya, untuk mendapatkan informasi citra geometris 3 dimensi.



Gambar 2-8 Ilustrasi optical frame dari suatu kamera stereo

2.2.1. Geometri Epipolar

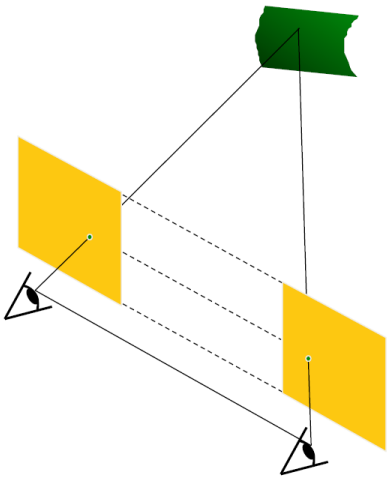
Untuk merekonstruksi informasi 3 dimensi dari dua citra 2 dimensi, kita harus tahu dimana suatu titik pada kamera a berada pada kamera b. Hal ini dijelaskan dalam konsep geometri epipolar pada [3] sebagai berikut.



Gambar 2-9 Geometri Epipolar

Karena kedua kamera memiliki *optical center* yang berbeda, kedua kamera memiliki sebuah titik yang tepat berada pada *optical center*-nya masing-masing. Pada kamera kiri garis O_L-X merupakan

sebuah titik karena tepat segaris pada *optical center* kamera kiri. Namun kamera kanan melihat ini sebagai garis x_R-e_R pada bidang gambarnya. Secara simetris begitu pula garis O_R-X merupakan titik pada kamera kanan, namun terlihat sebagai garis x_L-e_L oleh kamera kiri pada bidang-gambar kamera kiri (Gambar 2-9).



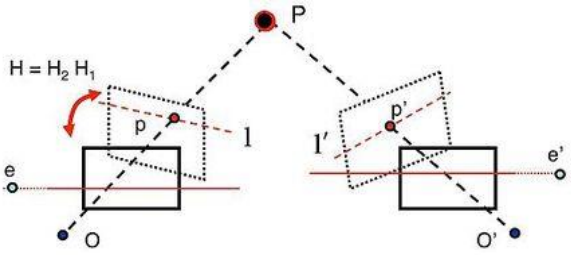
Gambar 2-10 Geometri Epipolar Hasil Rektifikasi

Geometri epipolar ini dijelaskan pada [3] dapat di sederhanakan bila bidang gambar dari kedua kamera sejajar satu dengan yang lain dan dengan *baseline*. Dengan hal ini maka epipolar line-nya akan parallel dengan garis O_R-O_L. Ini berarti untuk setiap titik pada satu gambar, titik yang sesuai pada gambar yang lain dapat ditemukan hanya dengan mencari sepanjang sumbu horizontalnya (Gambar 2-10).

Maka setelah disederhanakan posisi kedua kamera, kedua gambar memiliki karakteristik:

- Semua *epipolar line* parallel dengan sumbu horizontal.
- Titik yang berkorespondensi pada kedua kamera memiliki posisi vertikal yang sama.

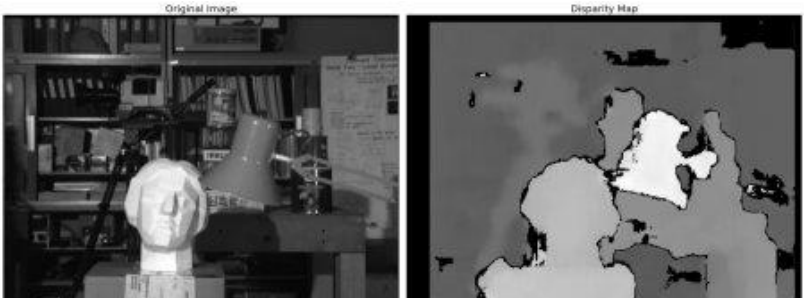
Memposisikan kedua kamera dengan sejajar sempurna untuk menyederhanakan geometri epipolar sangatlah sulit. Maka, pada tiap kamera dikenakan proses *rectifying* [3]. Proses ini adalah proses transformasi dari gambar yang digunakan untuk memproyeksikan pada satu bidang gambar yang sama. Proses transformasi rektifikasi gambar tiap kamera ini dijelaskan pada [4].



Gambar 2-11 Rektifikasi Citra Stereo

2.2.2. *Disparity map*

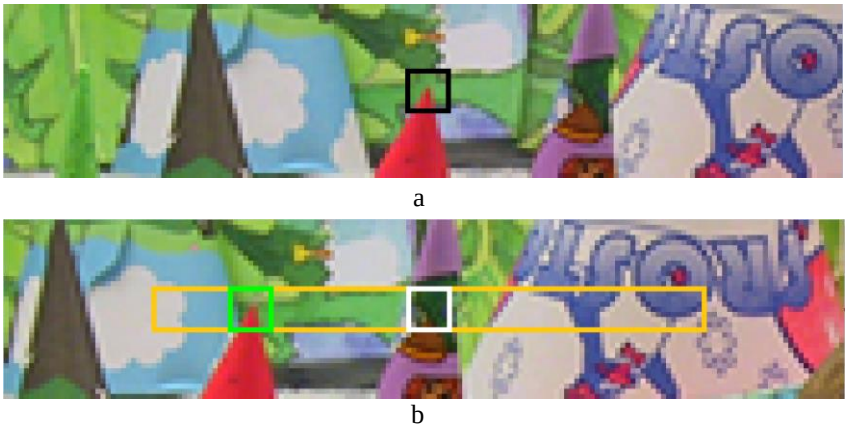
Disparity map adalah sebuah informasi perbedaan posisi dari pixel-pixel yang berkoresponden, dari satu pasangan gambar stereo. Untuk mendapatkan *disparity map*, dijelaskan pada [5] untuk menjalankan proses *stereo matching*. *Stereo matching* adalah proses menemukan satu pixel pada gambar yang sesuai, pada satu pixel di gambar lainnya.



Gambar 2-12 Contoh *disparity map* dari sebuah gambar

Salah satu algoritma yang dapat digunakan untuk *stereo matching* adalah *block matching*. Proses ini mengambil satu blok di sekitar *point* yang ingin dicari *matching point* nya, lalu mencari blok yang memiliki skor kecocokan yang tertinggi sepanjang garis epipolar pada gambar yang lain. Dengan mengasumsikan bahwa kedua gambar sudah terektifikasi maka *matching point* dari dua gambar hanya berbeda pada sumbu horizontalnya saja.

Contohnya adalah pada Gambar 2-13 kita mengambil suatu blok pada gambar kamera kiri yang ingin dicari matching *block* nya pada gambar kanan. Maka prosesnya adalah dengan menghitung nilai SAD (*Sum of absolute differences)* sepanjang *epipolar line* pada gambar kanan.



a

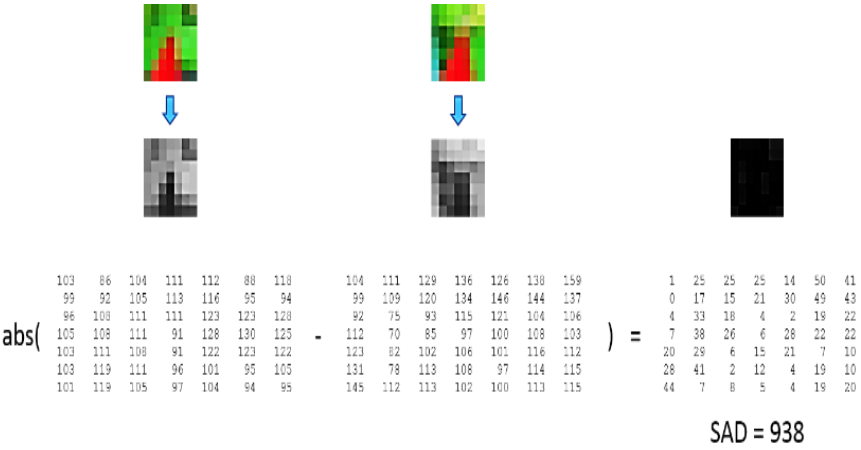
b

Gambar 2-13 Blok yang ingin dicari pasangannya (a) dan pencarian pasangannya (b).

Perhitungan SAD oleh [5] adalah:

$$SAD = \sum_{(i,j)\in W} |I_1(i,j) - I_2(x+i, y+j)| \qquad \textbf{(2-3)}$$

Proses perhitungan SAD adalah dengan merubah block yang akan dihitung menjadi *grayscale* terlebih dahulu, lalu kita kurangkan satu dengan lainnya untuk setiap pixel pada kedua block tersebut dan kita ambil perbedaan absolut dari keduanya per pixelnya. Lalu kita jumlahkan semua nilai perbedaan ini dan didapatkan sebuah nilai yang dengan kasar menilai kesamaan dari kedua blok yang dihitung. Makin rendah skornya maka makin mirip kedua blok tersebut.



Gambar 2-14 Perhitungan Sum Of Absolute Difference dari dua blok gambar

Dengan memilih blok dengan skor SAD paling rendah, disparity nya pada [3] dituliskan sebagai:

$$\delta = u_1 - u_2 \qquad \textbf{(2-4)}$$

Dengan,

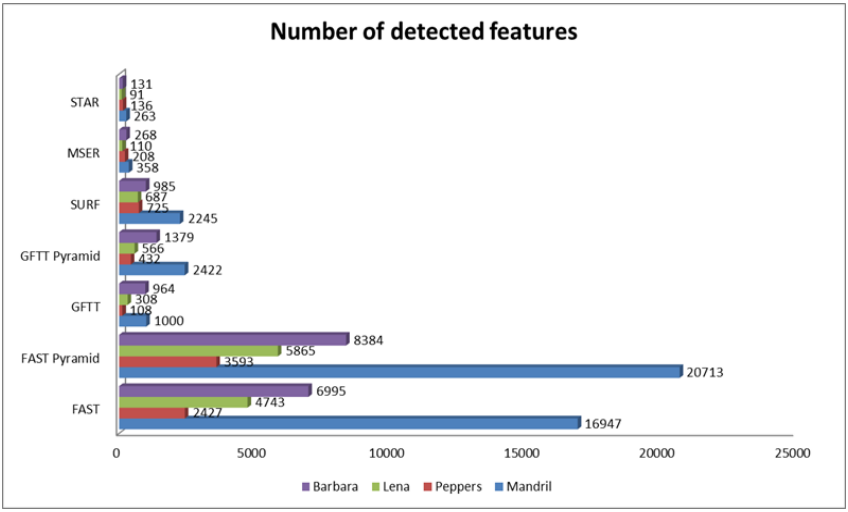
- δ adalah nilai disparity dari suatu pixel
- u_1 adalah posisi sumbu x suatu pixel pada gambar kiri
- u_2 adalah posisi sumbu x suatu pixel yang sesuai pada gambar kanan

2.3. FEATURE DETECTION

2.3.1. Pengertian deteksi fitur secara umum.

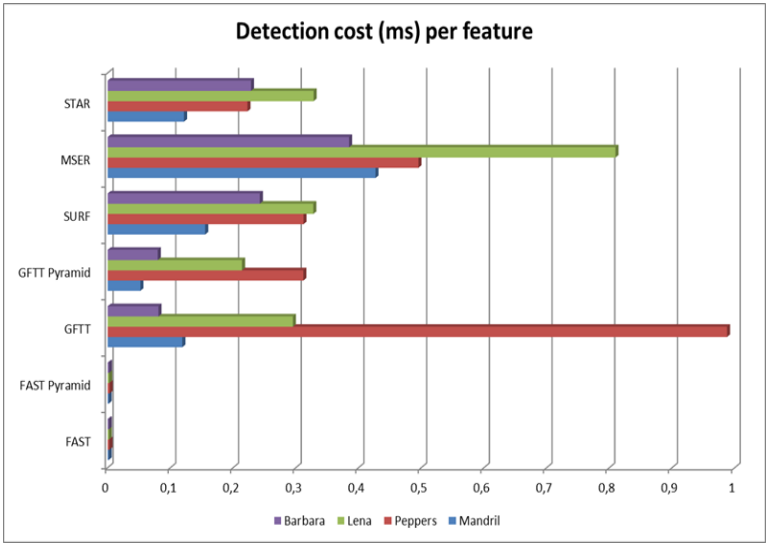
Feature detection pada aplikasi citra komputer berupa berbagai algoritma pendeteksi titik yang mencolok dari titik lainnya. *Feature detection* yang dibutuhkan pada aplikasi topik tugas akhir ini adalah sebuah detektor *corner* atau sudut. *Corner detection* dapat dilakukan dengan berbagai algortima yang sudah di temukan berbagai peneliti sebelumnya. Beberapa algoritma ini adalah Harris Corner Detection, Shi-Tomasi Corner Detector & Good Feature to Track (GFTT), SIFT, SURF, dan FAST. Berdasarkan [6] dan [7] FAST memiliki kelebihan daripada algoritma lain pada kasus penggunaan waktu riil.

Pada [6] FAST teruji dapat mendeteksi fitur dengan jumlah lebih banyak daripada algoritma lain. Hal ini di uji pada 4 jenis gambar yang sudah sering digunakan pada dunia citra komputer yaitu Barbara, Lena, Peppers, dan Mandril. Hal ini ditunjukkan pada Gambar 2-15.



Gambar 2-15 Perbandingan banyak fitur terdeteksi dari berbagai algortima deteksi fitur

Selain itu pada [6] juga diuji waktu deteksi berbagai algoritma deteksi fitur. Dibanding algoritma lain, FAST jauh lebih unggul dalam hal waktu proses untuk mendapatkan per fitur. Hal ini sangat penting pada penggunaan yang membutuhkan pemrosesan pada waktu riil. Ditunjukkan pada Gambar 2-16.

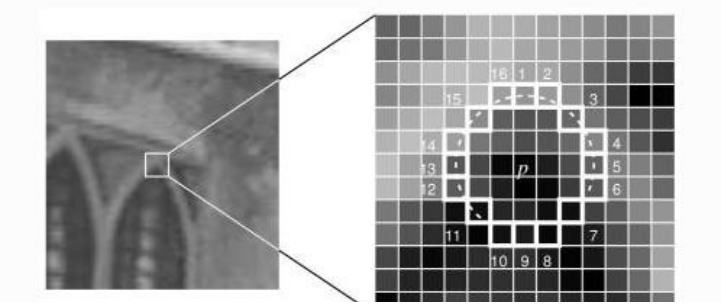


Gambar 2-16 Perbandingan waktu deteksi per fitur dari berbagai algoritma deteksi fitur

Gambar 2-17 Gambar untuk pengujian deteksi fitur pada **[6]**.
Dari kiri atas searah jarum jam: Mandril, Barbara, Peppers, Lena

2.3.2. Algoritma deteksi fitur FAST

Algoritma FAST secara lengkap di jelaskan pada [7]. FAST adalah singkatan dari *Features from Accelerated Segment Test*. FAST dilakukan pada gambar grayscale karena memerlukan intensitas pixel dengan 1 *channel*. Proses yang dilakukan adalah dengan membuat lingkaraan sebesar 16 *pixels* disekitar kandidat sudut p. Nyatakan intensitasnya sebagai I_P. *Pixel* p dinyatakan sebagai sebuah fitur sudut jika terdapat terdapat n pixel (pembuat algoritma ini menggunakan n=12) lebih terang dari $I_p + t$ atau lebih gelap dari $I_p - t$, dengan t adalah



Gambar 2-18 Contoh *corner* terdeteksi oleh FAST

threshold intensitas, pada kumpulan *pixel* didalam lingkaran (16 *pixel*) yang telah dibuat.

Beberapa masalah timbul dari metode ini dan oleh [7] diselesaikan dengan pembuatan model dengan *machine learning* untuk penyelesaiannya. *Machine learning* yang dibuat adalah dengan mendeteksi fitur pada beberapa set gambar dengan algoritma FAST yang telah ada. Setiap titik fitur disimpan beserta 16 pixel disekitarnya menjadi vektor fitur. Setiap pixel pada jendala 16 pixel ini akan memenuhi 3 keadaan:

$$S_{p\to x} = \begin{cases} d, & & I_{p\to x} \leq I_p - t & (darker) \\ s, & I_p - t < I_{p\to x} < I_p + t & (similar) \\ b, & I_p + t \leq I_{p\to x} & (brighter) \end{cases} \quad \textbf{(2-5)}$$

Berdasarkan keadaan ini, oleh [7] fitur p dimasukkan kedalam 3 *subset* Pd, Ps, Pb. Definisikan variabel Kp untuk menentukan apakah p adalah *interest point*. Terakhir dilakukan minimisasi entropi dengan algoritma ID3. Hal ini dilakukan rekursif kepada semua subset sampai entropi mencapai nol.

Metode ini beberapa kali lebih cepat dari metode *corner detection* lain yang ada. Namun penggunaannya dinilai tidak *robust* untuk citra dengan noise tinggi, dan bergantung pada *threshold*.

2.3.3. Triangulasi fitur menjadi koordinat 3 dimensi

Setelah didapatkan fitur-fitur dalam koordinat pixel 2 dimensi, dilakukan triangulasi atau reprojeksi fitur-fitur tersebut menjadi koordinat 3 dimensi sesuai sistem koordinat *world*. Hal ini dilakukan dengan memanfaatkan informasi karakteristik kamera, dan informasi *disparity* yang sudah didapatkan sebelumnya.

Proses ini diformulasikan pada [8] untuk setiap titik *pixel* x, y dan nilai *disparity* δ:

$$\begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} = Q * \begin{bmatrix} x \\ y \\ \delta \\ 1 \end{bmatrix} \quad \textbf{(2-6)}$$

$$\_3dImage(x,y) = (\frac{X}{W}, \frac{Y}{W}, \frac{Z}{W}) \quad \textbf{(2-7)}$$

Dengan,

Q adalah matriks transformasi perspektif kamera, 4 x 4.

$\_3dImage(x,y)$ adalah koordinat 3d.

Sedangkan matriks transformasi perspektif kamera Q dituliskan sebagai:

$$Q = \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 0 & -f \\ 0 & 0 & -1/T_x & 0 \end{bmatrix} \quad \textbf{(2-8)}$$

Dengan,

- c_x, c_y adalah offset *principal point* dari kamera kiri
- f adalah focal lenght dari kamera kiri
- T_x adalah jarak kamera kanan terhadap kamera kiri dalam meter

Koordinat 3d yang didapat adalah dari titik pada koordinat gambar (x,y), dalam hal ini koordinat 3d ini merepresentasikan sebuah fitur pada sistem koordinat *world.*

2.4. FEATURE TRACKING

Setelah didapatkan fitur pada *frame* F_a pada waktu a dilakukan pencarian fitur koresponden, atau dapat dikatakan posisi terkini, dari fitur yang telah ditemukan sebelumnya pada F_b, dengan waktu $b > a$. Hal ini dapat dilakukan dengan algoritma *feature tracking*.

2.4.1. Kanade-Lucas-Tomasi Optical Flow

Pada tugas akhir ini digunakan algoritma feature tracker Kanade Lucas Tomasi Optical Flow atau biasa disingkat KLT, diperkenalkan pada [9]. Algoritma ini akan mencari fitur koresponden pada *frame* F_b dari fitur-fitur yang sudah dideteksi pada F_a. Secara umum pada paper [9] sebuah sekuens gambar dapat dituliskan dalam sebuah fungsi dengan 3 variabel $I(x, y, t)$ dengan x, y variabel ruang dan t variabel waktu. Dapat dikatakan dalam sebuah sekuens gambar, akan ada pola yang bergerak dalam *stream* gambar. Maka fungsi sekuens gambar harus memenuhi:

$$I(x, y, t + \tau) = I(x - \xi, y - \eta, t) \quad \textbf{(2-9)}$$

Dimana gerakan $\boldsymbol{d} = (\xi, \eta)$ disebut sebagai perpindahan (*displacement*) dari titik pada $\boldsymbol{x} = (x, y)$ antara waktu t dan $t + \tau$. KLT tidak melakukan tracking dari satu pixel fitur, namun menggunakan *windows of pixels*, selanjutnya disebut jendela fitur.

Jendela fitur ini adalah kumpulan *pixel* disekitar sebuah fitur sudut, biasa disebut juga sebagai *descriptor* dari sebuah fitur. Proses *tracking* jendela fitur ini dijelaskan dengan mendefinisikan $J(x) = I(x, y, t + \tau)$ dan $I(x - \boldsymbol{d}) = I(x - \xi, y - \eta, t)$, dimana dengan menghilangkan variabel waktu untuk menyederhanakan formula, model gambar lokal dapat ditulis:

$$J(x) = I(x - \boldsymbol{d}) + n(x) \quad \textbf{(2-10)}$$

Dimana n adalah noise. Maka vektor perpindahan **d** dipilih nilai yang mana meminimalisir error residu yang didefinisikan dengan integral pada jendela W:

$$\epsilon = \int_W [I(x - \boldsymbol{d}) - J(x)]^2 w \, dx \quad \textbf{(2-11)}$$

Dalam formulasi ini w adalah bobot. Pada kasus paling sederhana bobot dapat berupa 1, alternatifnya bobot dapat berupa fungsi yang menyerupai Gaussian, yang menegaskan bagian pusat dari jendela. Teknik ini membutuhkan sebuah karakteristik jendela yang baik

digunakan untuk proses tracking. Pada Kanade-Lucas-Tomasi hal ini jendela yang baik dijelaskan lebih lanjut pada [10].

2.5. INLIER DETECTION

Proses ini dilakukan untuk memilah dari fitur-fitur yang sudah terdeteksi hanya fitur-fitur yang diam (pada dunia nyata 3d) terhadap satu sama lain. Hal ini dilakukan untuk membuang fitur-fitur yang terdeteksi pada objek-objek yang bergerak (bukan objek lingkungan sekitar). *Inlier detection* pada tugas akhir ini berdasarkan algoritma pada paper [11] dengan beberapa modifikasi pada *feature matching* yang sudah digantikan dengan KLT *feature tracker*:

1. Untuk setiap fitur $f_a \in F_a$, cari koresponden fitur yang sesuai pada fitur $f_b \in F_b$. Pada tugas akhir ini digunakan KLT *optical flow* untuk menyederhanakan pencarian fitur koresponden.
2. Nyatakan (f_a, f_b) sebagai pasangan-cocok dan $Match$ sebagai himpunan semua pasangan-cocok.
3. Dapatkan matriks konsistensi M dari semua kombinasi pasangan, dari pasangan-cocok pada $Match$, menggunakan *rigidity-constraint* sederhana. Dengan ketentuan:

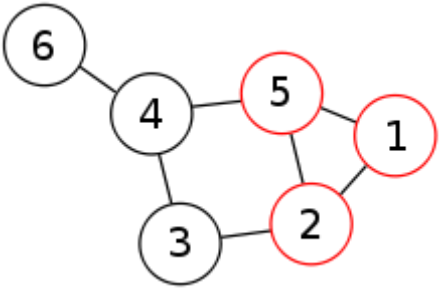
$$|w_a - w'_a| - |w_b - w'_b| < \delta \qquad \textbf{(2-12)}$$

Dengan w adalah koordinat *world* (3d) dari fitur, dan δ adalah *threshold*. w_a adalah suatu fitur 3d dari fitur 2d $i \in F_a$ dan w'_a suatu fitur 3d dari fitur 2d $j \in F_a$, dimana $i \neq j$, dan w_b dan w'_b adalah korespondensi kedua fitur tersebut pada F_b. Maka dapat dituliskan matriks konsistensi M sebagai:

$$M_{i,j} = \begin{cases} 1, & jika\ jarak\ 3d\ fitur\ i\ dan\ j\ konsisten \\ 0, & jika\ tidak\ konsisten \end{cases} \qquad \textbf{(2-13)}$$

Matriks konsistensi M ini pada dasarnya adalah sebuah matriks *adjacency* dari suatu *graph*, yang menyatakan konsistensi jarak antara satu fitur dengan fitur lain pada dua frame yang berbeda waktu.

4. Dari matriks M, tentukan set terbesar dari pasangan-cocok yang *mutually consistent*. Untuk mencari set terbesar dari matriks *adjacency* adalah sebuah permsalahan *Maximum Clique Problem* dangan matriks *M* sebagai *adjacency matrix*. Sebuah *clique* adalah *subset* dari *graph* yang mana hanya mengandung *nodes* yang terkoneksi antara satu dengan lainnya.



Gambar 2-19 Sebuah *Graph, node* 1,2,5 sebagai sebuah maximum clique

Dilakukan sebuah algoritma *greedy heuristic* yang memberikan kita *clique* yang mendekati pada solusi optimal:

1. Nyatakan Q sebagai *clique*, yang akan berisi *node* yang konsisten satu dengan lainnya.
2. Pilih satu *node* dengan *degree* maksimum (*degree* paling banyak). Masukkan *node* ini pada himpunan *clique* Q.
3. Dari *node* yang sudah ada pada *clique* Q, cari *subset node* yang terkoneksi pada semua *node* yang terdapat pada *clique*. Nyatakan *subset* ini dalam $\boldsymbol{v}$.
4. Dari $\boldsymbol{v}$ cari satu *node* dengan koneksi terbanyak pada *node* lain dalam $\boldsymbol{v}$. Tambahkan *node* tersebut pada *clique*. Ulangi step 3 sampai tidak ada lagi *node* yang dapat ditambahkan.

2.6. MOTION ESTIMATION

Untuk menemukan estimasi pergerakan robot, dilakukan perhitungan perpindahan fitur 3d dari *inlier* yang sudah ditemukan pada Q. Untuk mendapatkan estimasi perpindahan robot dari waktu a kepada

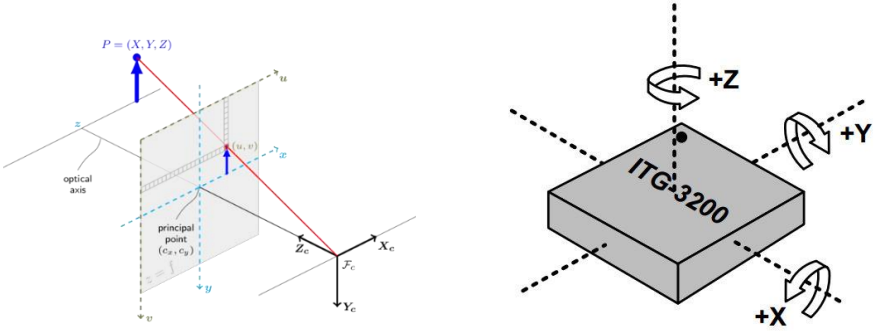
waktu b, perpindahan fitur 3d harus memperhitungkan rotasi untuk mendapatkan estimasi translasi yang valid. Pada tugas akhir ini digunakan sebuah IMU yang sudah siap pakai untuk mendapatkan sudut setiap waktu.

Sudut-sudut dari *gyroscope* ini dinyatakan dengan,

$$\theta = sudut\ pitch, rotasi\ sumbu\ y\ gyroscope$$
$$\phi = sudut\ roll, rotasi\ sumbu\ x\ gyroscope$$
$$\psi = sudut\ yaw, rotasi\ sumbu\ z\ gyroscope$$

dengan rotasi diasumsikan sebagai posisi sudut sumbu dari awal IMU aktif.

Untuk mensinergikan antara IMU dengan kamera, perlu adanya penyesuaian pada sumbu-sumbu IMU. Hal ini karena pada dunia *computer vision*, sudah disepakati sebuah *frame* yang disebut *optical frame* yang berbeda dari *frame* wahana yang umum digunakan. Hal ini membutuhkan penyesuaian rotasi pada IMU disesuaikan pada sumbu *optical frame*, sesuai dengan posisi perangkat keras *gyroscope* pada divais kamera stereo.



Gambar 2-20 Sistem koordinat kamera (kiri) dan IMU (kanan)

Dengan menyesuaikan perbedaan sumbu antara IMU dengan *optical frame* kamera. Maka perpindahan translasi dari waktu a sampai waktu b, dengan N –buah fitur:

$$tvec = \frac{\sum_n (R_{ba} \cdot f_{an} - f_{bn})}{N} \quad \textbf{(2-14)}$$

dengan R_{ba} adalah matriks rotasi dari beda sudut waktu b dan a IMU (dengan $b > a$) yang sudah disesuaikan dengan sumbu *optical frame.* Dengan kebutuhan robot yang hanya memiliki 3 DOF (translasi x, y dan rotasi sumbu z), maka dengan menyesuaikan kepada sistem koordinat kamera, dimana sumbu rotasi heading robot berada pada sumbu y kamera, didapat R_{ba} sebagai:

$$R_{ba} = \begin{bmatrix} \cos(\psi_{ba}) & 0 & sin(\psi_{ba}) \\ 0 & 1 & 0 \\ -sin(\psi_{ba}) & 0 & cos(\psi_{ba}) \end{bmatrix} \quad \textbf{(2-15)}$$

Setelah didapatkan perpindahan dari waktu a dan b, maka dapat dihitung dengan sum rekursif untuk mendapatkan posisi terbaru robot, relatif dari posisi awalnya dengan persamaan:

$$pos_{current} = pos_{previous} + R \cdot tvec_{current} \quad \textbf{(2-16)}$$

dengan R adalah matriks rotasi IMU dari awal waktu yang sudah disesuaikan dengan sumbu *optical frame*. Dengan penggunaan pada Maka R adalah:

$$R = \begin{bmatrix} \cos(\psi) & 0 & sin(\psi) \\ 0 & 1 & 0 \\ -sin(\psi) & 0 & cos(\psi) \end{bmatrix} \quad \textbf{(2-17)}$$

2.7. OPENCV

OpenCV adalah singkatan dari Open Source Computer Vision. OpenCV adalah *library* yang dapat digunakan pada bahasa C++, Python, Matlab, Ruby dan bahasa pemrograman lain. OpenCV juga kompatibel pada *operating system* (OS) yang banyak digunakan seperti Windows,

MacOS, dan berbagai distro Linux salah satunya ubuntu. Karena sifatnya yang *open source* membuat library ini mempercepat proses pengembangan teknologi *computer vision* untuk semua kalangan.

OpenCV diklaim memiliki 2500 fungsi yang berhubungan dengan fungsi-fungsi *computer vision*. Hal ini seperti akuisisi data baik gambar, video, atau hasil tangkapan kamera langsung, filter citra, identifikasi objek berdasarkan warna atau bentuk, *feature detection*, *object tracking*, dan berbagai fungsi lain.

OpenCV bersifat *open source* yang berarti bersifat terbuka dan dapat digunakan oleh siapapun selama mematuhi peraturan lisensi dari OpenCV. Lisensi OpenCV bersifat *BSD License (Berkeley Software Distribution License)* yaitu lisensi untuk software gratis yang berisi 3 klausa yang wajib dipatuhi semua penggunanya.

2.8. ROBOT OPERATING SYSTEM

Robot Operating System atau biasa disingkat ROS adalah sebuah *middleware* sistem robotik, yang berisi berbagai koleksi *software framework* untuk pengembangan perangakat lunak robotik. ROS bukanlah sebuah sistem operasi seperti Windows, MacOS, Android atau sistem operasi lain yang kita kenal, namun sebuah perangkat lunak yang menyediakan berbagai *service* yang didesain untuk *heterogenous computer cluster* seperti: abstraksi perangkat keras, pengaturan divais level rendah, implementasi fungsionalitas yang sering dipakai (pada robot), penyampaian pesan antar proses, dan menejemen *package*.

Software pada ROS dapat dibagi menjadi tiga grup: *tools* untuk mendistribusikan perangkat lunak berbasis ROS, implementasi ROS *client library* seperti roscpp, rospy, roslisp, dan yang terakhir *package* yang berisi kode yang berhubungan dengan aplikasi fungsional robot.

Client libraries ROS utamanya diperuntukkan untuk sistem UNIX. Untuk hal ini Ubuntu Linux terdaftar sebagai *"supported"* sedangkan varian lain seperti Fedora Linux, macOS, serta Windows terdaftar sebagai *"experimental"*.

2.8.1. ROS Package

Perangkat lunak pada ROS terorganisir di dalam *packages*. Didalam *packages* tersebut dapat berisi ROS *Node*, *library* independen, dataset, file konfigurasi, dan hal lain yang mendukung modul tersebut. Tujuan dari *packages* adalah untuk menyediakan fungsionalitas yang berguna ini dalam sifat yang "mudah-dikonsumsi" dengan tujuan menjadikan perangkat linak yang dapat dengan mudah digunakan kembali (*reuse)* oleh komunitas pengguna ROS.

2.8.2. ROS Node

Sebuah *node* dalam ROS adalah sebuah proses yang melakukan suatu komputasi. berbagai *node* ini terkombinasi bersama menjadi suatu *graph* dan berkomunikasi dari satu *node* ke *node* lain menggunakan aliran *topics*, *services*, dan *parameter server*. *Node* ini diperuntukkan untuk beroperasi pada level yang sangat sempit. Sebagai contoh sebuah sistem kendali robot biasanya terdiri dari banyak *node*: satu *node* mengendalikan *laser range finder*, satu *node* mengendalikan motor, satu *node* lokalisasi, satu *node* untuk perencanaan jalur (*path planning)*, satu *node* untuk *monitoring*, dan seterusnya.

Penggunaan *node* ini memberi manfaat antara lain: toleransi kegagalan yang terisolasi pada satu fungsionalitas tertentu pada individu *node*, kompleksitas dari kode berkurang dibanding sebuah sistem *monolithic*, *reusability* serta *maintainability* tinggi dengan penyederhanaan berdasarkan fungsionalitas-fungsionalitas tersebut.

Selain itu sebuah node dapat dikembangkan menjadi sebuah *nodelet*, yaitu sebuah node yang dapat dipanggil dari dalam kode program lainnya menggunakan modul *nodelet manager*.

2.8.3. ROS Topic

Topic adalah komponen yang berfungsi sebagai media pertukaran pesan (disebut *message* dalam sistem ROS) antar *node* yang dinamai sesuai isi dari pesan tersebut. Sistemnya menggunakan *anonymous publish/subscribe semantics*. *Nodes* yang membutuhkan informasi dari suatu *topic* akan melakukan *subscribe* dan *node* yang menyediakan

informasi akan melakukan *publish*. Jumlah *node* yang melakukan *subscribe* ataupun *publish* pada suatu *topic* tidak terbatas jumlahnya.

Setiap *topic* memiliki tipe datanya masing-masing sesuai tipe ROS *message* yang di bawanya. Kita dapat membuat tipe *message* sendiri sesuai kebutuhan kita, atau menggunakan tipe *message* yang disediakan oleh ROS *client library*. ROS *topic* juga mendukung komunikasi via TCP/IP atau UDP.

2.9. KAMERA STEREO BERBASIS USB WEBCAM

Sebagai input dari sistem *visual odometry* berbasis citra stereo ini dibutuhkan sebuah hardware *stereo camera*. *Stereo camera* yang digunakan adalah dua buah webcam USB logitech C270 yang dirakit menjadi satu *stereo camera*.

Kamera USB webcam adalah jenis kamera yang dapat digunakan pada PC ataupun laptop dengan mudah karena sifatnya yang plug & play. Memiliki interface berupa kabel USB memudahkan penggunaan pada PC ataupun laptop modern. Kamera ini dijuluki webcam karena penggunaan utamanya yang diperuntukkan pada penggunaan video chatting dan live video stream via internet.

Untuk membuat stereo kamera dibutuhkan dua buah USB webcam yang dirakit menjadi satu. Sebisa mungkin kedua kamera tersebut sejajar atau parallel secara horizontal. Pada tugas akhir ini digunakan papan akrilik yang dibentuk menjadi sebuah black box yang menjadi rangka untuk kedua USB webcam tersebut.

Sedangkan spesifikasi dari kamera logitech C270 yang digunakan tercantum pada [12] adalah:

Connection Type	: Corded USB
USB TypeHigh	: Speed USB 2.0
Microphone	: Built-in, Noise Supression
Lens and Sensor Type	: Plastic
Focus Type	: Fixed
Field of View (FOV)	: 60°
Focal Length	: 4.0 mm
Optical Resolution (True)	: 1280 x 960 1.2MP
Frame Rate (max)	: 30fps @ 640x480

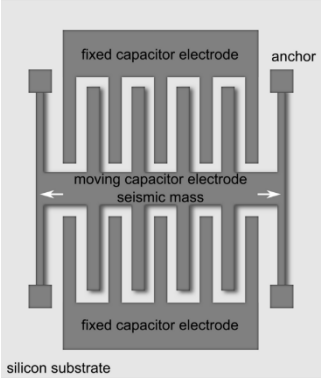
2.10. INERTIAL MEASUREMENT UNIT (IMU)

Inertial Measurement Unit (IMU) adalah sebuah sensor yang mengukur suatu gaya spesifik, laju perpindahan sudut, dan gaya magnetik pada beberapa alat. Sensor ini biasanya paling tidak berisi dua buah sensor yaitu accelerometer dan *gyroscope*, pada kasus lain ditambahkan sensor magnetometer. IMU memiliki banyak variasi dari sensornya berupa sensor mekanis, dapat juga berupa laser ring, dan varian yang banyak terdapat di pasaran adalah tipe MEMS atau Micro-Electro-Mechanical Sensor.

Penggunaan paling umum dari IMU adalah dalam mengukur posisi sudut dari suatu wahana. IMU dengan dua sensor, *gyroscope* dan accelerometer, memiliki 6 DOF pembacaan besaran yang dapat diproses untuk menghasilkan pembacaan rotasi sudut dari 3 sumbu wahana. Penggunaan lain dari IMU yaitu untuk mengestimasi translasi dari wahana namun hal ini membutuhkan kualitas IMU khusus penggunaan navigasi.

2.10.1. Accelerometer

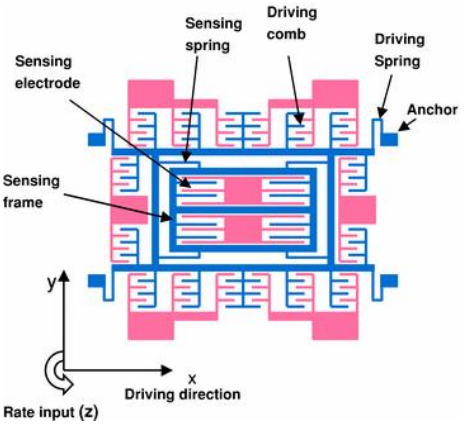
Sensor accelerometer membaca percepatan linear yang terjadi pada sensor. Nilai dari yang terbaca merupakan gabungan dari nilai sebenarnya, bias, dan juga noise, maka dibutuhkan pengolahan lebih lanjut untuk mendapatkan nilai yang benar dari percepatan linear yang terjadi.



Gambar 2-21 Struktur accelerometer MEMS

2.10.2. Gyroscope

Sensor ini akan mengukur kecepatan sudut yang terjadi terhadap sensor. Pembacaannya pun harus dilalui dengan proses pengolahan untuk mendapatkan nilai kecepatan sudut yang sebenarnya, karena terdapat noise dan bias dalam *gyroscope*.



Gambar 2-22 Struktur Gyroscope MEMS

IMU yang digunakan pada tugas akhir ini adalah sebuah paket modul IMU siap pakai yang berisi:

- Arduino Nano
- MPU6080 GY-87
- Firmware dengan output posisi rotasi 3 DOF

Penggunaan paket modul siap pakai ini dilakukan untuk mengkerucutkan masalah pada algoritma pengolahan citra.

2.11. MOBILE ROBOT DARAT

Mobile robot adalah tipe robot yang memiliki karakteristik berbeda dari robot statik seperti lengan industri, dimana *mobile robot* memiliki karakteristik bergerak bebas dari satu tempat ke tempat lain. Jenis-jenisnya pun bermacam-macam sesuai dengan caranya bergerak.

Mobile robot darat dapat berbentuk robot beroda, ataupun jenis robot berkaki. Hal ini disesuaikan dengan kebutuhan penggunaan robot,

dengan medan yang dilalui, lingkungan sekitar, dan fungsional dari robot sendiri.

Hal yang membedakan dari mobile robot darat dan mobile robot lain (UAV, USV) adalah keterbatasan degree of freedom pada sistem navigasinya. Untuk mobile robot beroda yang digunakan hanya di dalam ruangan, dengan permukaan yang relatif rata, hanya perlu digunakan 3 DOF navigasi yaitu X, Y, dan Sudut Heading. Hal ini yang akan menjadi sebuah parameter pengujian sistem *visual odometry* yang penulis rancang.

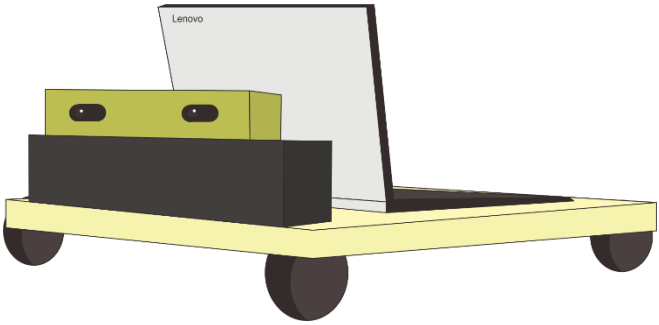
Gambar 2-23 Contoh *mobile robot* darat

Untuk mengkerucutkan permasalahan tugas akhir ini pada proses *visual odometry*, hanya akan digunakan sebuah *mock* mobile robot, yaitu sebuah wahana yang memiliki karakteristik mirip dengan karakteristik mobile robot beroda, hanya saja tidak memiliki sistem yang menjadikannya sebuah robot. Hal ini dinilai penulis sudah cukup untuk menguji kelayakan, dan hipotesis-hipotesis yang dibangun pada sistem *visual odometry* yang akan dirancang.

[Halaman ini sengaja dikosongkan]

BAB 3
PERANCANGAN SISTEM

Dalam bab ini akan dibahas mengenai perancangan sistem navigasi atau lebih tepatnya estimasi posisi lokal sebuah *mobile robot* darat menggunakan *visual odometry* berbasis citra stereo. Perancangan sistem ini akan meliputi perancangan perangkat keras kamera stereo, lalu perancangan perangkat lunak yang meliputi pembuatan ROS *package* khusus yang berisi berbagai *node* yaitu *node* driver kamera stereo, rektifikasi gambar, *dan block matching*/*stereo processing*, lalu *node stereo visual odometry*, dan terakhir *node* penerimaan data IMU. Dalam bab ini akan dijelaskan cara kerja berbagai komponen sistem navigasinya per blok bagiannya.



Gambar 3-1 Ilustrasi sistem mock mobile robot dengan sistem stereo vision yang dirancang

Tugas Akhir ini akan lebih membahas pada blok perancangan sistem *vision* atau perancangan sebuah ROS Package dari akuisisi data dari kamera stereo, lalu diproses sampai menghasilkan suatu data estimasi posisi lokal robot. Untuk menghasilkan suatu ROS *Package visual*

odometry tersebut hal pertama yang harus dirancang adalah *node* akuisisi data dari kamera stereo. Setelahnya kita lakukan kalibrasi kamera dan kita rektifikasi sesuai karakteristik kedua kamera yang kita dapatkan pada proses kalibrasi kamera. Data gambar yang telah di rektifikasi ini siap untuk kita olah lebih lanjut menjadi *disparity map*. Setelah kita dapatkan *disparity map*, kita dapat lakukan proses perhitungan algoritma *visual odometry* untuk mendapatkan estimasi posisi kamera relatif terhadap posisi awalnya.

3.1. PERANGKAT KERAS KAMERA STEREO USB

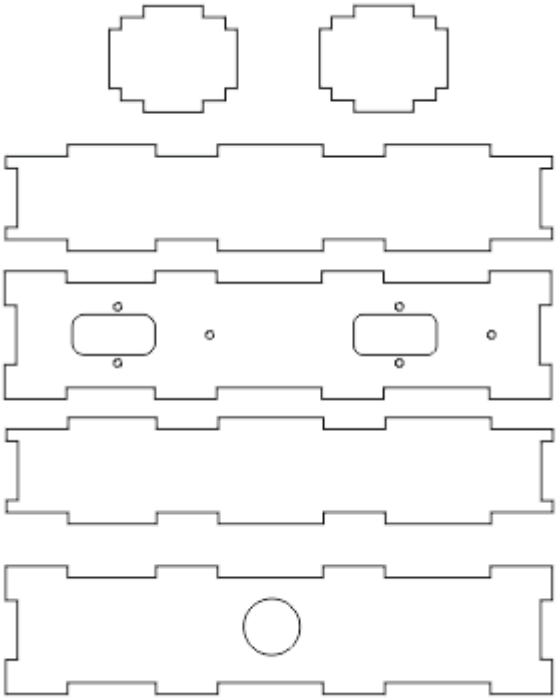
Sebagai divais akuisisi data atau informasi citra stereo, dirancang sebuah kamera stereo yang berbasis dua buah USB webcam logitech c270. Untuk menggabungkan dua buah kamera ini dibuat sebuah *enclosure* dari papan akrilik dengan tebal 0.5 cm. Posisi kedua kamera sedemikian rupa di desain untuk sebisa mungkin hanya berbeda posisi pada sumbu horizontalnya.

Gambar 3-2 USB *Webcam* Logitech c270

Untuk menghasilkan *enclosure* kamera stereo ini digunakan software CorelDraw X7 untuk mendesain potongan-potongan akrilik yang akan dirakit menjadi suatu *encolsure*. Bagian-bagiannya didesain sedemikian rupa agar mampu mewadahi kedua kamera dengan baik. Dalam hal ini, didesain agar jarak antara lensa kamera sekitar 118 mm pada sumbu horizontalnya, dan sebisa mungkin tidak memiliki beda

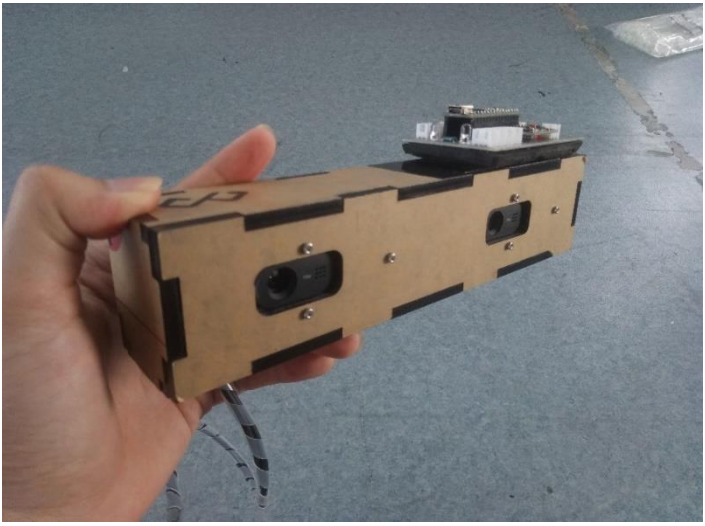
posisi pada sumbu vertikalnya. Selain itu didesain pula lubang pada bagian belakan *enclosure* untuk kabel kamera.

Desain dari *enclosure* ini mengikuti kaidah stereo kamera model *Horizontal Pair*. Desain *enclosure* ditunjukkan Gambar 3-3.



Gambar 3-3 Desain *enclosure* untuk dua buah *webcam* c270

Dimensi dari *enclosure* kamera stereo ini adalah: 220mm x 52mm x 45mm.



Gambar 3-4 Kamera stereo setelah dirakit

3.2. MOCK MOBILE ROBOT

Seperti telah disampaikan pada bab sebelumnya, untuk keperluan pengujian sistem nantinya, dibutuhkan sebuah model *mock mobile robot* darat yang dipergunakan untuk menguji sistem yang dirancang pada lingkungan *mobile robot* darat, dan pengujian karakteristik gerak sesuai dengan bagaiman semestinya sebuah *mobile robot* darat beroda berjalan.

Model *mock mobile robot* beroda ini didesain dengan dua roda fixed wheel pada bagian belakang, dan dua *swivel wheel* pada bagian depan. Hal ini mengacu pada desain sebuah *mobile robot* beroda non holonomic. Sedangkan pada bagian atas, didesain penempatan kamera yang memungkinkan untuk menghasilkan gambar yang sesuai dengan fungsional *visual odometry*. Maka dari itu penempatan kamera memiliki ketinggian sekitar 283 mm dari permukaan tanah, hal ini untuk membuat

gambar yang tertangkap kamera tidak mayoritas berupa gambar permukaan, namun juga menangkap banyak obyek didepan kamera.

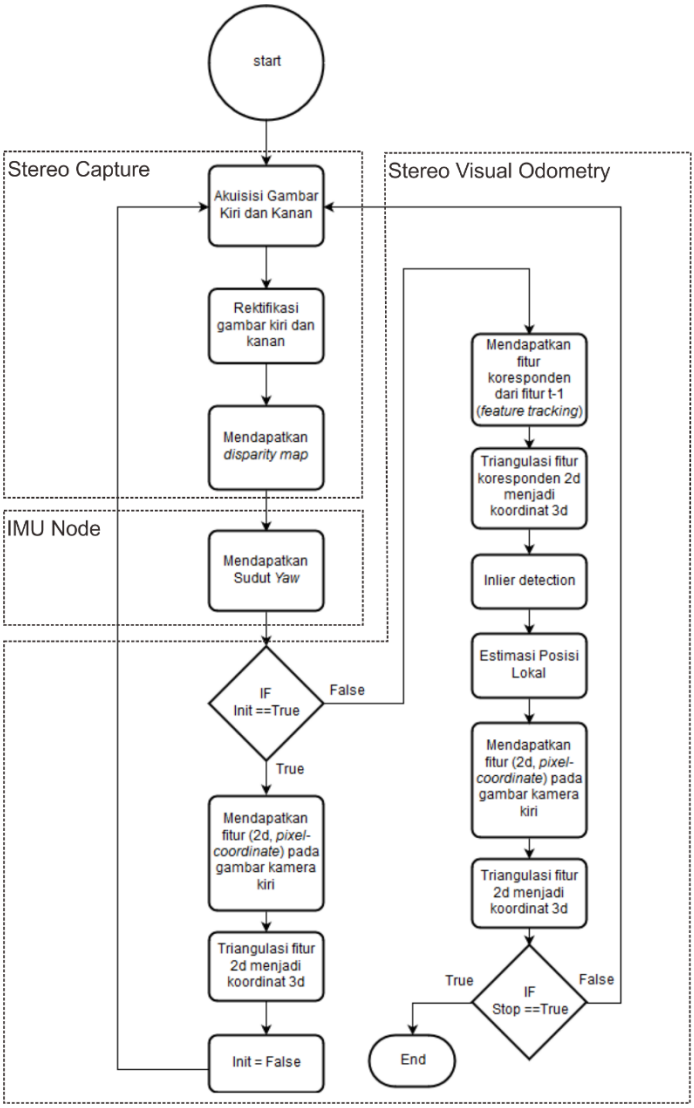
Gambar 3-5 Mock Mobile Robot dengan kamera stereo dan laptop yang digunakan

3.3. SISTEM PERANGKAT LUNAK KESULURUHAN

Perangkat lunak yang harus dibuat dibagi menjadi beberapa bagian sesuai dengan tahapan proses dari awal akuisisi data dari divais kamera sampai menghasilkan estimasi posisi dengan algoritma stereo visual odometry. Proses secara keseluruhan dari sistem perangkat lunak yang harus dibuat secara umum adalah:

1. Akuisisi data kamera stereo.
2. Rektifikasi gambar kiri dan kanan.
3. Mendapatkan *Disparity map*.
4. Mendapatkan rotasi *Yaw* dari Gyroscope IMU.
5. Deteksi fitur 2d dari gambar kiri pada frame waktu t-1.
6. Triangulasi fitur 2d waktu t-1 menjadi koordinat 3d.
7. *Visual Odometry*

Secara keseluruhan, alur kerja perangkat lunak dapat deskripsikan dengan flowchart pada Gambar 3-6:



Gambar 3-6 *Flowchart* proses yang dijalankan sistem yang dirancang

Agar sistem bersifat *modular* sesuai dengan sistem ROS, sistam perangkat lunak ini dibagi menjadi 3 ROS *node*: stereo capture, imu *node*, dan stereo visual odometry. Pembagian proses per-*node* dari flowchart proses sistem sebelumnya, ditunjukkan pada Gambar 3-6.

3.4. NODE DRIVER KAMERA

3.4.1. Akuisisi data kamera stereo

Untuk akuisisi data dari divais kamera stereo, dibutuhkan sebuah *software driver* untuk mengambil data gambar dari kamera stereo. Maka dari itu penulis membuat sebuah ROS *node* khusus sebagai akuisisi data citra dari perangkat lunak. Perangkat lunak yang dibuat menggunakan bahasa C++ dan library OpenCV.

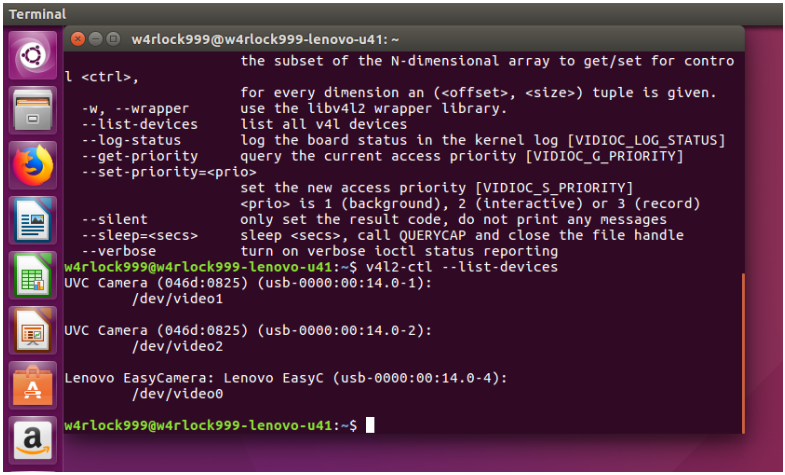
Untuk membuat *driver* ini harus diketahui sebelumnya *index* kedua kamera yang sesuai. Index ini tergantung pada urutan terkoneksinya sebuah divais kamera dengan PC yang digunakan. Pada kasus ini disepakati bahwa divais kamera kiri harus dikoneksikan terlebih dahulu sebelum divais kamera kanan. Hasilnya adalah kamera kiri memiliki indeks /dev/video1 dan kamera kanan memiliki indeks /dev/video2, sementara /dev/video0 sudah terpakai oleh *webcam* bawaan laptop. Ditunjukkan pada Gambar 3-7.

3.4.1.1. Perancangan Class Akuisisi Data dari Kamera Stereo.

Agar pengambilan gambar dapat bersamaan antara kamera kanan dan kiri maka tidak bisa kita akuisisi secara *serial programming*. Hal ini tidak dapat dilakukan karena *latency* dari program akuisisi data gambar terbilang lama karena besarnya data, maka jika dilakukan *serial programming* data kamera kiri, yang diakuisisi terlebih dahulu, akan tampak telat dibanding data kamera kanan, yang relatif lebih baru.

Untuk mengatasi hal ini harus dilakukan *parallel programming.* Teknik ini dapat dilakukan salah satunya menggunakan teknik *multi threading.* Multi threading adalah teknik pemrograman dengan menjalankan banyak proses secara bersamaan. Satu proses ini biasa disebut satu *thread*. Untuk kasus akuisisi data kamera stereo ini dibuat

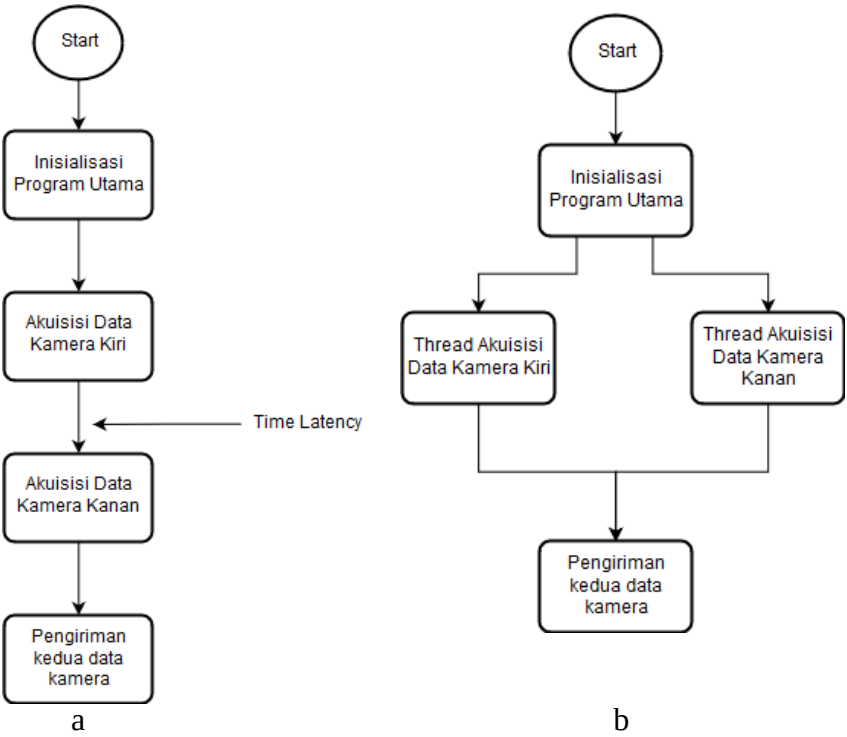
dua *thread* yang mengakuisisi data kamera kiri dan kamera kanan secara bersamaan. Ditunjukkan pada Gambar 3-8.



Gambar 3-7 Tampilan daftar divais kamera pada laptop

Karena itu dibuat sebuah *class* khusus untuk mengambil gambar secara parallel dengan nama `StereoCaptureClass` dalam file stereoCaptureClass.cpp dan stereoCaptureClass.hpp. Untuk mengambil gambar dari kamera, OpenCV menyediakan *class* `VideoCapture`. Untuk memulai mengambil gambar kamera kiri diinisialisasi sebuah *instance* `VideoCapture` dengan nama captureLeft dan memasukkan parameter *index* kamera kiri.

Untuk mengurangi beban pemrosesan gambar dan juga agar fps yang didapat stabil, perlu dilakukan pengaturan pada *software* Video For Linux Control atau v4l2-ctl. Pengaturan yang dilakukan adalah dengan mengatur parameter power_line_frequency, exposure_auto_priority, exposure_auto, dan backlight_compensation pada nilai 1,0,3,1 secara berurutan.



Gambar 3-8 Proses akuisisi gambar secara *seri* (a) dan *parallel* (b)

3.4.1.2. Main Program dari Node Driver Kamera Stereo

Setelah membuat *class* khusus akuisisi gambar kamera kiri dan kanan, dibuat sebuah program utama untuk menggunakan fungsi-fungsi dari *class* yang dibuat, dan mengkonversi data gambar sesuai sistem ROS. Program utama ini merupakan sebuah ROS *node* yang diberi nama Stereo Capture, yang bertugas sebagai *node driver* kamera. Diagram alur main program ini ditunjukkan pada Gambar 2-13. Secara keseluruhan tugas dari program utama *node driver* kamera ini adalah:

- mengakuisisi gambar menggunakan *class* `StereoCaptureClass`
- mengkonversi gambar menjadi format ROS Image Message
- mem-*publish message* ini dalam ROS *topic* kepada ROS.

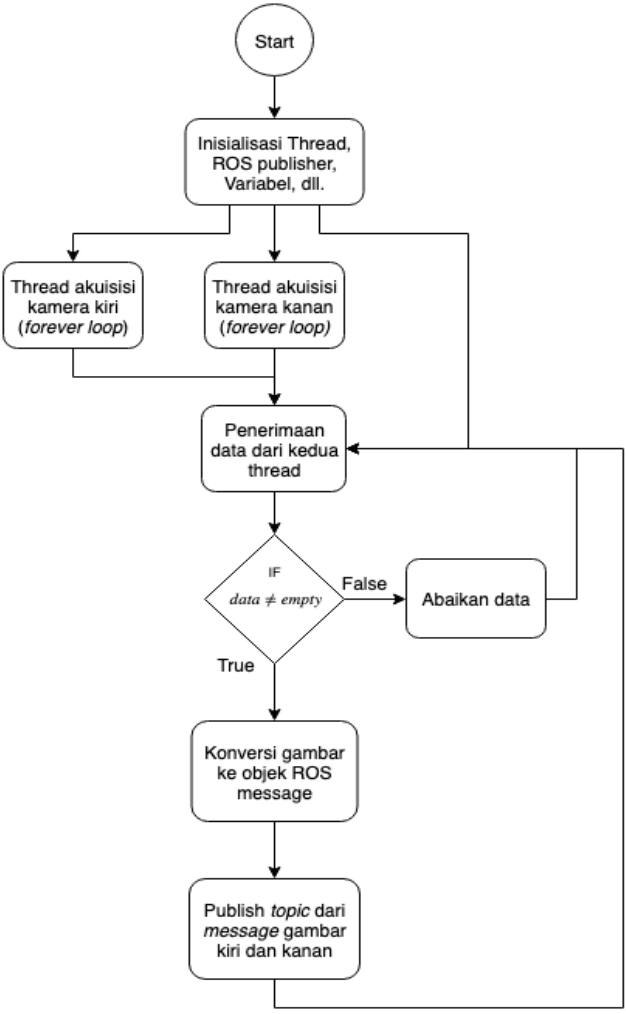
Untuk mengambil gambar dari class `StereoCaptureClass`, dipanggil *method* `getLeftRightFrames` dengan parameter pointer pada `Mat` dari `cameraLeft_image` dan `cameraRight_image`. Jika didapat frame kanan dan kiri bukan frame kosong (*empty*) proses ini dilanjutkan dengan proses selanjutnya, namun jika terdapat frame kosong dari kedua data yang diterima maka data diabaikan dan dilakukan penerimaan data sampai kuda frame tidak kosong.

Sebelum me-*publish* gambar, `cameraLeft_image` dan `cameraRight_image` harus diberi *timestamp* terlebih dahulu dengan `ros::Time::now()` yang menandakan timestamp waktu sekarang, baru dapat melakukan *publish* dengan perintah `pubCamLeft.publish` dan `pubCamRight.publish`, memanfaatkan method `toImageMsg()` pada `cv_bridge::CvImagePtr` untuk mengubah format dari *class* tersebut menjadi sebuah *message*. Hasil gambar yang didapat dari *node driver* ini ditunjukkan pada gambar Gambar 3-10.

3.4.2. Kalibrasi kamera

Kalibrasi kamera dilakukan untuk mengetahui karakteristik-karakteristik dari perangkat keras dan informasi gambar yang kita dapat dari kamera stereo. Kalibrasi kamera ini akan menghasilkan apa yang disebut parameter intrinsik dan parameter ekstrinsik kamera. Parameter intrinsik dari kamera merepresentasikan *focal length* dan *optical center* dari kamera. Sedangkan parameter ekstrinsik merepresentasikan posisi pada koordinat 3 dimensi.

Untuk melakukan kalibrasi kamera ini digunakan sebuah package yang disediakan ROS yang khusus digunakan untuk mengkalibrasi kamera stereo. Package ini ditulis dengan memanfaatkan fungsi kalibrasi kamera yang telah tersedia pada *library* OpenCV. *Package* ini memanfaatkan sebuah *checkerboard* yang diketahui jumlah kotaknya dan ukuran masing-masing kotaknya. *Checkerboard* inilah yang akan menjadi *calibration rig* untuk mendapatkan informasi karakteristik kamera dari informasi ukuran dan jumlah *Checker* yang digunakan.



Gambar 3-9 *Flowchart* proses pada *node* driver kamera Stereo Capture

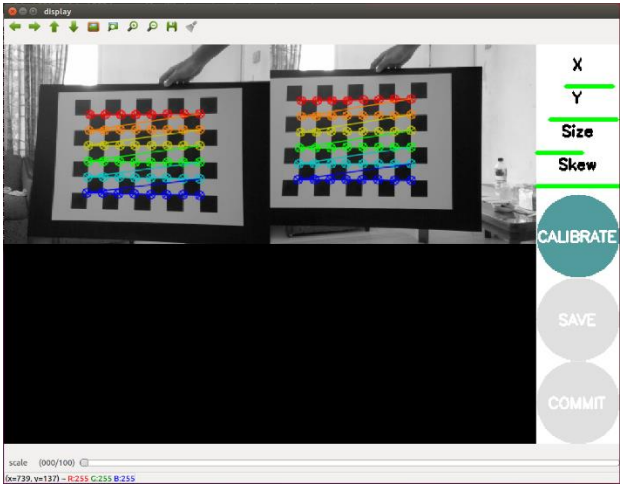
Gambar 3-10 Tampilan gambar kamera kiri dan kanan

Package ini akan mengambil gambar-gambar yang dibutuhkan untuk keperluan kalibrasi kamera diatas. Kita hanya perlu memberikan *topic* gambar kamera kiri ("camera/left/image_raw") dan *topic* gambar kamera kanan ("camera/right/image_raw"). Checkerboard yang digunakan pada tugas akhir ini berukuran 8x6 (dihitung berdasarkan *inner corner*) dan setiap kotak berukuran 5,1 cm x 5,1 cm. Untuk memanggil program kalibrasi kamera, dilakukan pemanggilan lewat terminal ubuntu dengan perintah:

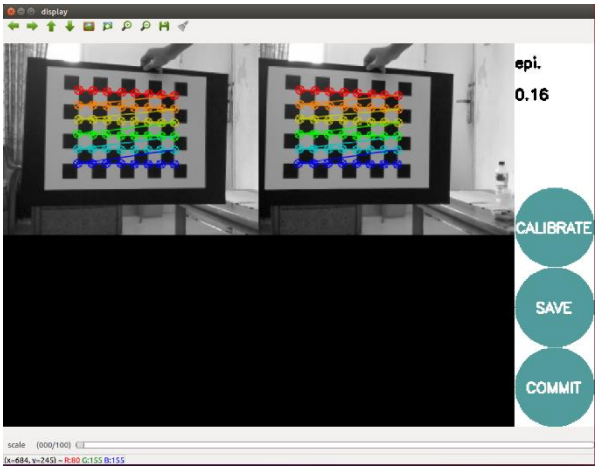
```
rosrun camera_calibration cameracalibrator.py --size 8x6 --
square 0.051 right:=/camera/right/image_raw
left:=/camera/left/image_raw
```

Untuk mendapatkan hasil kalibrasi yang baik diperlukan gambar objek checkerboard ini pada posisi-posisi tertentu: dekat-jauh (*scaling*), atas-bawah (sumbu y), kiri-kanan (sumbu x), dan posisi condong pada dua sumbunya (*skew*). Jika gambar yang didapat sudah mencukupi jumlahnya untuk dilakukan kalibrasi, dapat di *click* tombol kalibrasi.

Gambar 3-11 Tampilan program kalibrasi kamera sebelum mengambil gambar



Gambar 3-12 Tampilan program kalibrasi kamera setelah gambar yang didapat mencukupi untuk kalibrasi



Gambar 3-13 Gambar setelah kalibrasi sukses dilakukan.

Dari proses kalibrasi kamera didapatkan karakteristik berupa matriks intrinsik, matriks distorsi, matriks rotasi, dan vektor translasi dari kamera kiri dan kamera kanan. Setelahnya package ini membuat suatu file yang mendeskripsikan semua karakteristik tersebut dalam sebuah file dengan ekstensi .yaml. Dalam file ini tertulis informasi:

1. Lebar dan tinggi gambar
2. Nama kamera
3. Matriks karakteristik kamera
4. Model distorsi lensa
5. Koefisien distorsi
6. Matriks rektifikasi
7. Matriks proyeksi

Informasi yang didapat pada file left.yaml untuk kamera kiri dan right.yaml untuk kamera kanan (terlampir) ini berfungsi untuk merektifikasi kedua gambar agar menjadi sebidang dan menyederhanakan perhitungan geometri epipolar-nya. Dari proses tersebut didapatkan matriks karekteristik kamera kiri dan kanan berikut:

$$K_l = \begin{bmatrix} 417.33 & 0 & 153.93 \\ 0 & 417.37 & 130.68 \\ 0 & 0 & 1 \end{bmatrix}$$

$$K_r = \begin{bmatrix} 415.00 & 0 & 142.93 \\ 0 & 415.63 & 105.99 \\ 0 & 0 & 1 \end{bmatrix}$$

3.4.3. *Publish* Topik Camera Info

Setelah melakukan kalibrasi kamera stereo dan mendapatkan file *camera info* untuk masing-masing kamera kiri dan kanan, kita harus me-*publish* dalam bentuk message camera_info dalam suatu *topic* yang dikehendaki.

Hal ini dilakukan dengan menambahkan fungsi pada program *node* driver kamera yang telah dibuat sebelumnya. Pertama-tama yaitu dengan membuat dua objek `std::string` yang berisi alamat file camera info berada. Lalu dengan membuat objek `camera_info_manager::CameraInfoManager` dari library C++ ROS dan memberikan objek std::string alamat *camera info* yang telah kita buat, kita dapat mendapatkan *camera info* pada objek `sensor_msgs::cameraInfo` sebagai *message* yang akan kita *publish*.

Setelah membuat dua objek `sensor_msgs::cameraInfo` untuk kedua kamera dan memberikan data *camera info* yang sesuai, dapat mempublishnya dengan method `publish()` dari objek `ros::publisher pubCamLeftInfo` untuk kamera kiri dan `pubCamRightInfo` untuk kamera kanan yang telah dibuat sebelumnya.

3.4.4. Rektifikasi gambar kiri dan kanan

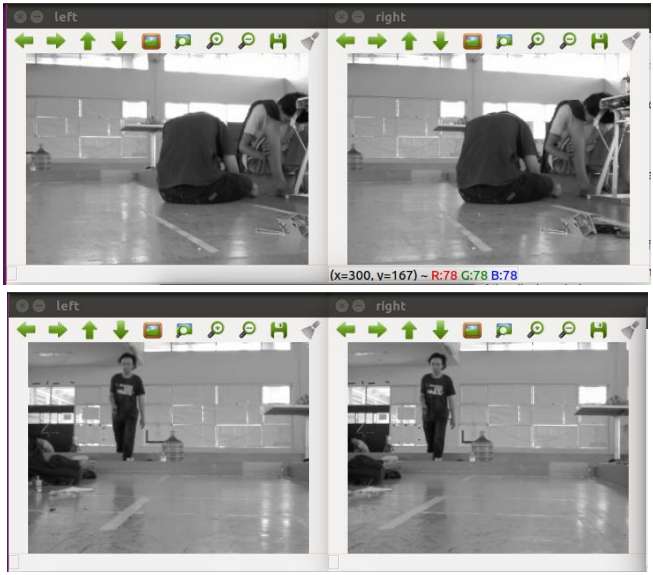
Sebelum mendapatkan data *disparity*, gambar yang didapat dari kamera harus direktifikasi terlebih dahulu agar seolah-olah kedua kamera berada pada satu bidang epipolar yang hanya berbeda posisi horizontalnya. Rektifikasi ini memanfaatkan topik *camera info* yang di *publish* dari *node driver* kamera.

Pada ROS *client library* implementasi dari rektifikasi dengan OpenCV sudah disediakan agar sederhana dan mudah digunakan dalam

bentuk ROS *nodelet*. Yang perlu dilakukan adalah memanggil *nodelet* tersebut dan memasukkan parameter dan *remapping* yang diinginkan.

Untuk memanggil *node*let ini, dibuat sebuah fungsi yang diberi nama `loadImageProc` yang akan dipangil pada masing-masing gambar kiri dan kanan. Fungsi ini akan melakukan *remapping* atau penyesuaian nama-nama *topic* di dalam *node*let yang akan dipanggil, baik *topic* yang di *publish* atau di *subscribe* oleh *node*let, sesuai dengan nama *topic* yang dibuat *node* driver kamera dan kebutuhan *node-node* selanjutnya. *Node*let yang dipanggil adalah image_proc/debayer, dan dua image_proc/rectify. Dua *node*let image_proc/rectify ini difungsikan untuk gambar berwarna dan gambar monokrom.

Hasil dari proses ini, yaitu gambar yang telah terrektifikasi, ditunjukkan pada gambar Gambar 3-14.



Gambar 3-14 Hasil gambar setelah proses rektifikasi

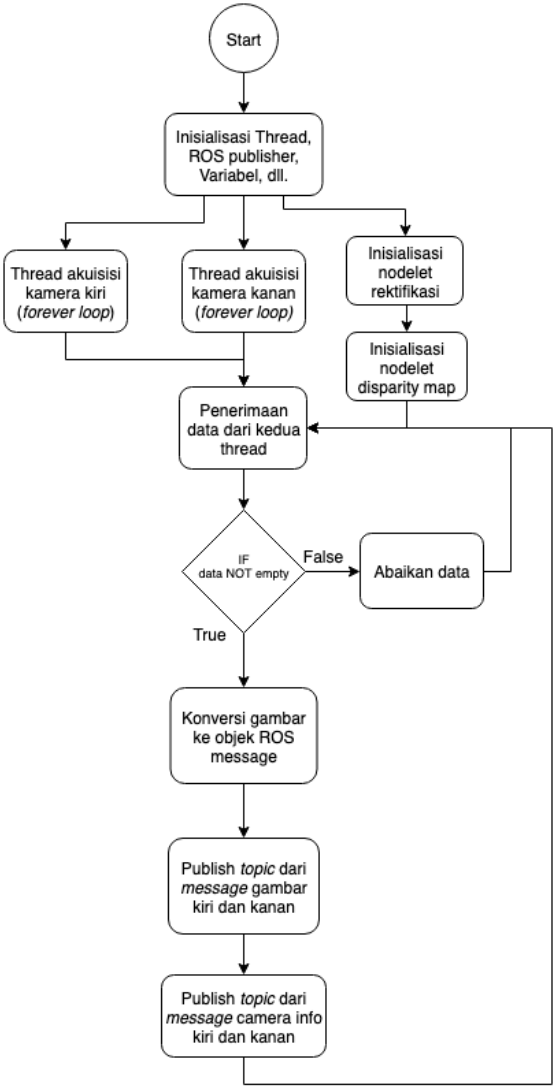
3.4.5. Mendapatkan *Disparity map*

Untuk mendapatkan *disparity map*, ROS menyediakan *nodelet* yang dapat dipanggil dan akan menghasilkan *message* gambar *disparity map*. *Nodelet* ini dipanggil pada perangkat lunak driver kamera dengan menambahkan perintah untuk memanggil *nodelet* ini dan melakukan *remapping* atau penyesuaian nama-nama *topic* di dalam *node*let yang akan dipanggil, disesuaikan dengan nama-nama *topic* yang telah dan akan dipakai. *Nodelet* yan dipanggil adalah stereo_proc/disparity.

Sama seperti inisialisasi *nodelet* rektifikasi, *remapping* dilakukan pada awal inisialisasi, dengan menyesuaikan nama-nama *topic* yang dibutuhkan *nodelet*. Pada inisialisasi ini, 4 *topic* yang di-*subscribe* dari *nodelet disparity map* adalah *topic* gambar yang telah terrektifikasi, yaitu camera/left/image_rect dan camera/right/image_rect, hasil proses *nodelet* rektifikasi, dan *topic* camera info dari kedua kamera yaitu camera/left/camera_info dan camera/right/camera_info.

Hasil dari proses ini adalah sebuah *topic* yang berisi gambar *disparity map*, ditunjukkan pada gambar dan Gambar 3-15.

Gambar 3-15 Hasil *disparity map* dan gambar asalnya



Gambar 3-16 *Flowchart* proses pada *node* driver kamera Stereo Capture terkini

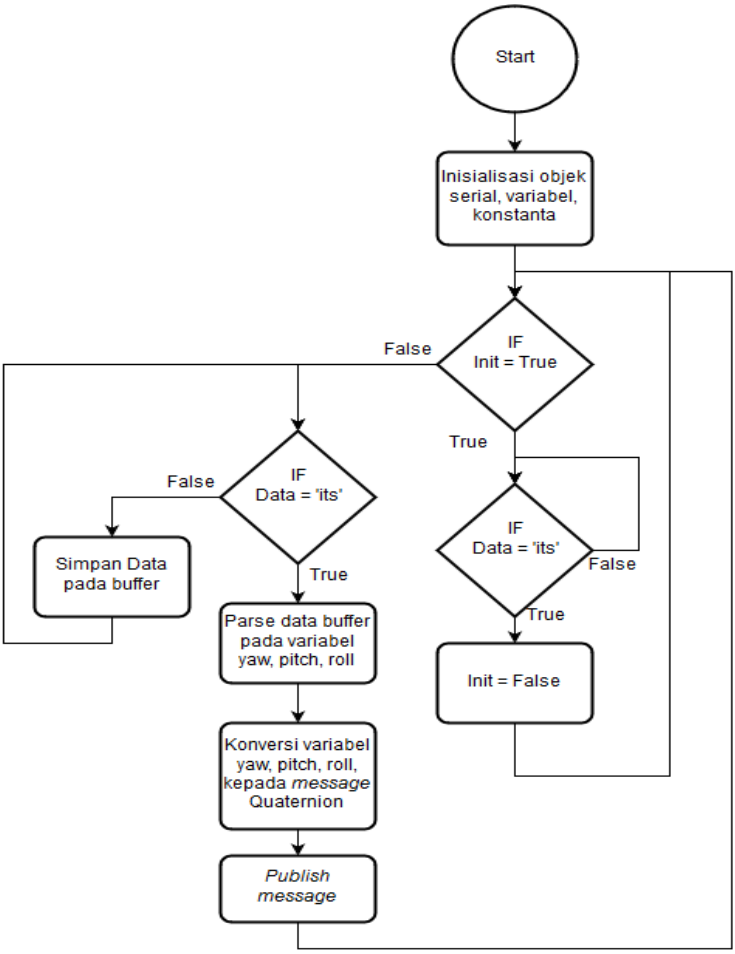
Dengan penambahan inisialisasi *nodelet* ini, maka terjadi perubahan pada alur proses program *node* driver kamera secara keseluruhan, ditunjukkan pada Gambar 3-16. Dengan ini pula, program *node* driver secara keseluruhan selesai dirancang.

3.5. NODE PENERIMAAN DATA IMU

IMU yang digunakan adalah sebuah modul *black box* sudah terprogram firmware IMU dan sudah mengeluarkan data rotasi 3 DOF, dengan baudrate 115200 via komunikasi serial. Untuk mengambil data dan menggunakannya pada sistem ROS, dibuat sebuah *node* khusus untuk akuisisi data IMU dan *publish* data ini pada sistem ROS. Program ini menggunakan bahasa python dan modul rospy serta beberapa modul python lain yang dibutuhkan untuk komunikasi serial. Proses kerja *node* ini digambarkan dalam gambar Gambar 3-17.

Pada python disediakan sebuah modul serial yang dibutuhkan untuk komunikasi dengan arduino IMU. Untuk menggunakan modul ini, dibuat sebuah objek serial, dengan memasukkan parameter ID divais pada komputer (/dev/ttyUSB0) dan baudrate yang sesuai (115200). Setiap data rotasi memiliki header khusus untuk mengetahuai awalan data yang diterima, pada kasus ini headernya adalah karakter ‘its’. Dengan *method* `read_until` kita dapat melakukan *parsing* data IMU pada variabel yang sesuai.

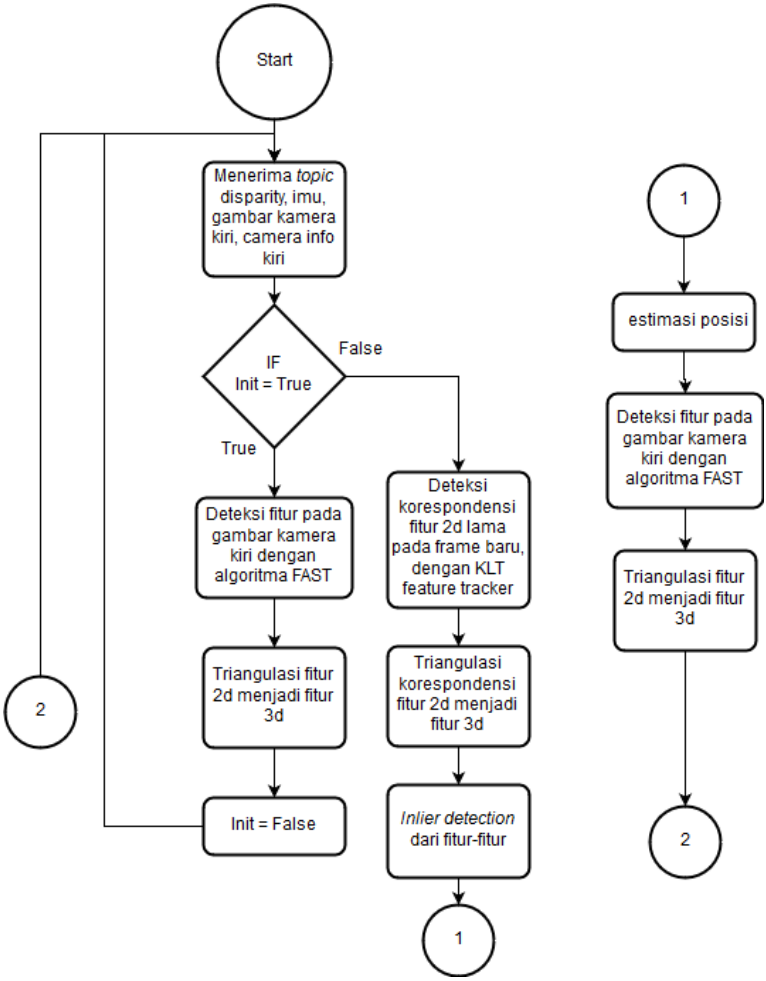
Untuk melakukan *publish* data pada sistem ROS, kita dapat memasukkan data yang diterima pada tipe *message* Quaternion dari modul `geometry_msg` yang disediakan ROS Client Library. Sebelum melakukan *publish*, perlu dilakukan konversi ke quaternion karena data rotasi yang diterima adalah rotasi Tait-Bryan dengan urutan data Roll, Pitch, dan Yaw. Konversi ini dapat dilakukan dengan modul `tf_conversion` yang disediakan ROS Client Library menggunakan method `transformations.quaternion_from_euler`. Setelah didapat objek rotasi dalam quaternion kita dapat melakukan *publish* kepada sistem ROS. Kode program *node* imu terlampir.



Gambar 3-17 *Flowchart* proses penerimaan data IMU.

3.6. NODE VISUAL ODOMETRY

Untuk melakukan proses visual odometry, dibuat sebuah *node* khusus yang bekerja untuk menjalankan proses visual odometry. Proses kerja *node* ini digambarkan dalam bagan:



Gambar 3-18 *Flowchart* proses *visual odometry* pada *node* Stereo Visual Odometry

3.6.1. Subscribe topic-topic

Untuk mendapatkan informasi disparity yang telah didapat dan di *publish* sebelumnya, *node* visual odometry ini melakukan *subscribe* kepada *topic* terkait dengan memasukkan parameter nama *topic* yaitu "camera/disparity" dan nama fungsi callback yang dipanggil setiap menerima data baru yaitu `disparityCallback` pada deklarasi objeknya.

Untuk mendapatkan informasi rotasi dari *node* IMU, harus dilakukan *subscribe* pada *topic Quaternion* yang telah di *publish* sebelumnya dengan juga memasukkan parameter nama *topic* yaitu "imu_quat" dan fungsi callback yang dipanggil setiap menerima data imu, yaitu `imuCallback`. Setelah itu dilakukan *parsing* kedalam suatu variabel / objek agar dapat digunakan pada program *node visual odometry* dengan juga mengkonversi kedalam rotasi euler dengan `getRPY()`.

Untuk mendapatkan gambar kamera kiri, dilakukan *subscribe* pada *topic* gambar kiri yang telah di *publish* sebelumnya dengan memasukkan parameter nama *topic* "camera/left/image_rect_color", dan fungsi callback `camLeftCallback`.

Karena proses mayoritas *visual odometry* dilakukan dengan memproses gambar kamera kiri dan dengan tambahan informasi *disparity* dan rotasi yang telah didapatkan sebelumnya, maka proses utama dilakukan pada fungsi `camLeftCallback` ini.

3.6.2. Deteksi fitur dengan algoritma FAST

Setiap mendapatkan gambar baru, untuk mendeteksi fitur-fitur yang ada, digunakan algoritma deteksi fitur pada gambar tersebut. Untuk algoritma FAST, pada OpenCV sudah disediakan *class* yang dapat digunakan untuk melakukannya yaitu *class* `cv::FastFeatureDetector`.

Untuk membuat persebaran fitur merata, dilakukan *bucketing* dalam melakukan deteksi fitur. Hal ini adalah dengan membagi gambar menjadi beberapa bagian persegi dengan *i* baris dan *j* kolom (Gambar 3-19) dan membatasi deteksi fitur per bagian gambar (per *bucket*) sebanyak *k* fitur. Dalam tugas akhir ini digunakan 2 baris *bucket* dan 5 baris *bucket*. Proses ini ditunjukkan pada Gambar 3-20.

Setiap mendapatkan fitur per *bucket,* fitur disimpan sementara pada `keypointsBucket`, dipilih beberapa fitur yang terbaik dengan

fungsi `retainBest()`, dan menambahkannya pada variabel `keypointsCurrent.`

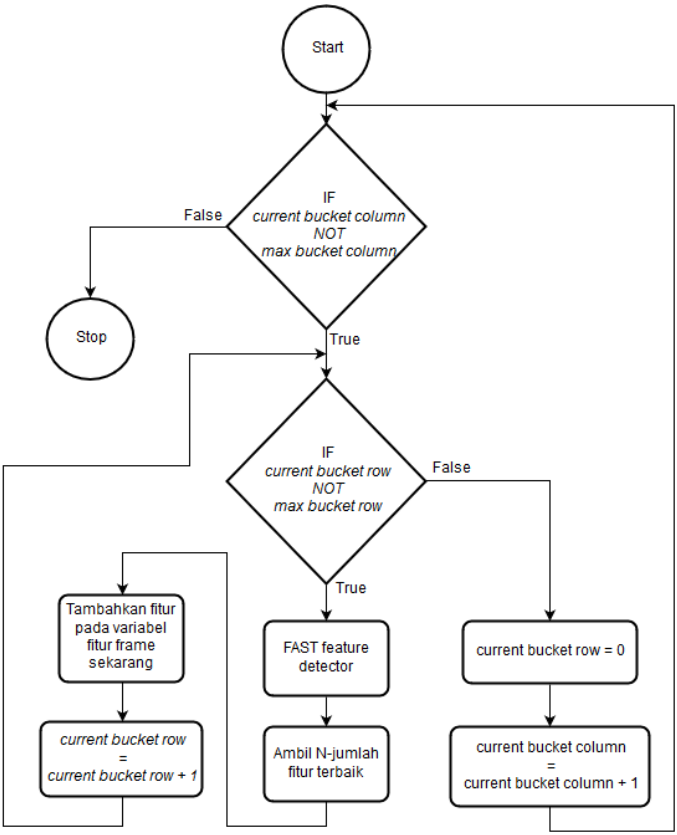
Gambar 3-19 Bucketing dengan membagi gambar menjadi 2 baris dan 5 kolom

3.6.3. Triangulasi fitur 3 dimensi

Setelah didapatkan fitur 2 dimensi, dilakukan triangulasi dengan fungsi yang telah disediakan ROS untuk mendapatkan koordinat 3 dimensi untuk setiap fitur. fungsi yang digunakan adalah `projectDisparityTo3d` dengan parameter titik fitur berupa variabel keypoint, informasi disparity dari fitur tersebut, dan variabel untuk menyimpan hasil perhitungan triangulasi berupa objek `Point3d`. Hal ini dilakukan untuk setiap fitur yang di dapat pada proses sebelumnya, dan disimpan pada variabel `keypoint3dCurrent.`

Untuk setiap fitur yang didapat dari proses sebelumnya, dipanggil *method*:

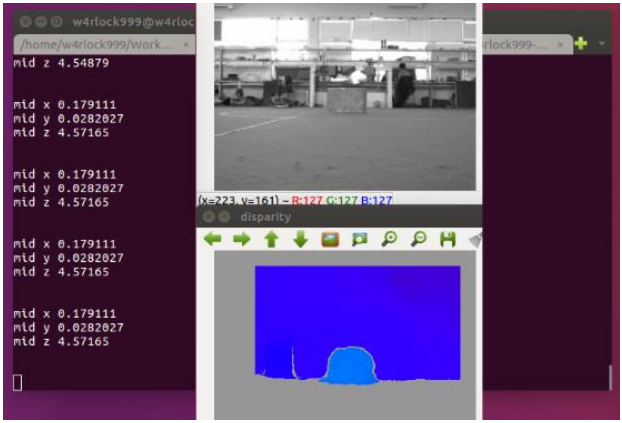
```
cameraModel.projectDisparityTo3d(keypointsCurrentVP[i],
    disparity.at<float>(keypointsCurrentVP[i].y,
    keypointsCurrentVP[i].x),
    keypoint3dbuff);

keypoint3dCurrent.push_back(keypoint3dbuff);
```



Gambar 3-20 *Flowchart* proses deteksi fitur dengan *bucketing*

Dimana `keypointsCurrentVP[i]` adalah keypoint suatu fitur, `disparity` adalah data disparity terbaru yang diterima oleh program, dan menggunakan method `at<float>(y,x)` didapatkan informasi disparity pada koordinat pixel x,y, dan `keypoint3dbuff` adalah variabel sementara untuk menyimpan hasil triangulasi satu fitur ini yang berisi informasi 3d dari fitur pada dunia nyata (X,Y,Z). Setelah didapatkan triangulasi satu fitur, hasil ini disimpan pada objek vektor yang setiap elemennya adalah objek `Point3d` yang menyimpan 3 informasi

koordinat dunia. Triangulasi ini dilakukan dengan for loop untuk semua elemen dari variabel `keypointCurrentVP.`



Gambar 3-21 Tampilan hasil triangulasi titik pada koordinat 3d

3.6.4. Tracking fitur

Setelah didapat frame gambar baru dari kamera kiri, dilakukan tracking fitur yang telah didapat pada frame gambar sebelumnya. Hal ini dilakukan dengan algoritma Kanade-Lucas-Tomasi (KLT) Feature Tracker. Fungsi yang digunakan adalah `calcflowpyrlk` yang disediakan oleh OpenCV.

Fungsi yang dipanggil adalah:

```
calcOpticalFlowPyrLK(prevFrame, currentFrame,
   keypointsPrevVP_Temp, keypointsCurrentVP_Temp, status,
   err);
```

Parameter yang dipanggil untuk menjalankan fungsi KLT pada OpenCV adalah `prevFrame` yaitu gambar sebelumnya, `currentFrame` yaitu gambar terbaru pada saat ini, `keypointsPrevVP_Temp` adalah fitur-fitur pada frame sebelumnya yang ingin dicari korespondennya pada frame sekarang, `keypointsCurrentVP_Temp` yaitu objek untuk menyimpan koresponden fitur sebelumnya dimana objek ini berupa

vektor yang urutan elemen-elemennya sesuai dengan `keypointsPrevVP_Temp` dan berisi koordinat pada frame terbaru jika ditemukan korespondennya.

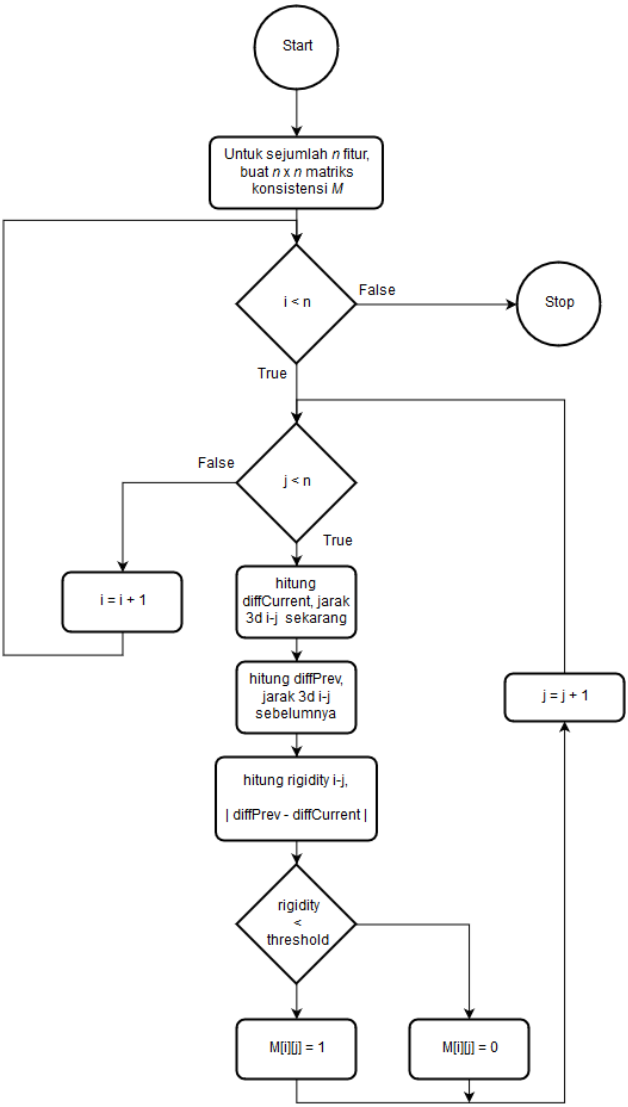
Pada prakteknya setelah ditemukan koresponden fitur pada gambar terbaru, dilakukan triangulasi lagi pada koresponden fitur 2d yang ditemukan untuk mencari koordinat-koordinatnya dalam 3d, dengan proses yang sama seperti proses yang sudah dijelaskan sebelumnya, hasil dari triangulasi ini disimpan pada objek `keypoint3dCurrent.`

3.6.5. Deteksi inlier fitur sesuai rigidity constraint

Deteksi *inlier* dilakukan untuk menghilangkan fitur-fitur yang tidak sesuai dengan kaidah *rigidity constraint*. Hal ini dilakukan dengan harapan bahwa set fitur terdeteksi yang terbanyak adalah fitur-fitur yang bersifat statis satu sama lain, bukan fitur bergerak, dan merupakan fitur dari benda-benda yang tidak bergerak (bagian dari lingkungan sekitar yang statis).

Pertama tama dibuat sebuah matriks konsistensi *M* yang berukuran *n* x *n* dimana n adalah jumlah fitur yang ditemukan korespondennya pada tahap sebelumnya. Matriks ini dapat dikatakan adalah sebuah matriks *adjacency* dari sebuah *graph* yang menyatakan konsistensi jarak antar satu fitur dengan fitur lain. Elemen matriks *M* ini akan bernilai 1 jika kedua fitur memiliki jarak yang konsisten pada selang waktu yang terjadi, dan bernilai 0 jika jaraknya tidak konsisten pada selang waktu yang terjadi. Proses pembuatan matriks M ini diperlihatkan pada Gambar 3-22. Setelahnya dilakukan pemecahan *maximum clique* dari suatu *graph* dengan matriks *M* sebagai matriks *adjacency* .

Tahap pertama pemecahan masalah *maximum clique* pada tugas akhir ini adalah menginisialisasi *clique* dengan sebuah *node* dengan *degree* paling besar/banyak. Proses ini dilakukan dengan kode program berikut:



Gambar 3-22 Proses pembuatan matriks konsistensi/*adjacency* ***M***

```
for(int i = 0; i < static_cast<int>(keypointsCurrentVP.size());
i++){
    for(int j = 0; j <
static_cast<int>(keypointsCurrentVP.size()); j++){

        if( M[i][j] == 1){
            degreeCurrNode++;
        }
    }

    // initialize node with max degree as candidate
    if(degreeCurrNode > degreeMax){
        degreeMax = degreeCurrNode;
        initialNodeToClique = i;
    }

    degreeCurrNode = 0;
}
clique.push_back( initialNodeToClique );
```

Tahap kedua adalah melakukan pencarian *node-node* yang potensial sebagai anggota baru clique, dilakukan dengan kode program berikut:

```
for(int i = 0; i < static_cast<int>(keypointsCurrentVP.size());
i++){

    int currNodePositive = 0;
    for(int j = 0; j < static_cast<int>( clique.size()); j++ ){

        if( M[i][clique.at(j)] )
            currNodePositive++;
    }

    if(currNodePositive == static_cast<int>( clique.size())){
        if(std::find(clique.begin(), clique.end(), i) ==
clique.end()){
```

```
            potentialNodes.push_back( i );
        }
    }
}
```

Dilakukan pengecekan berapa jumlah elemen matriks M pada indeks i yang konsisten dengan semua *clique* yang telah didapat pada perintah:

```
if( M[i][clique.at(j)] )
    currNodePositive++;
```

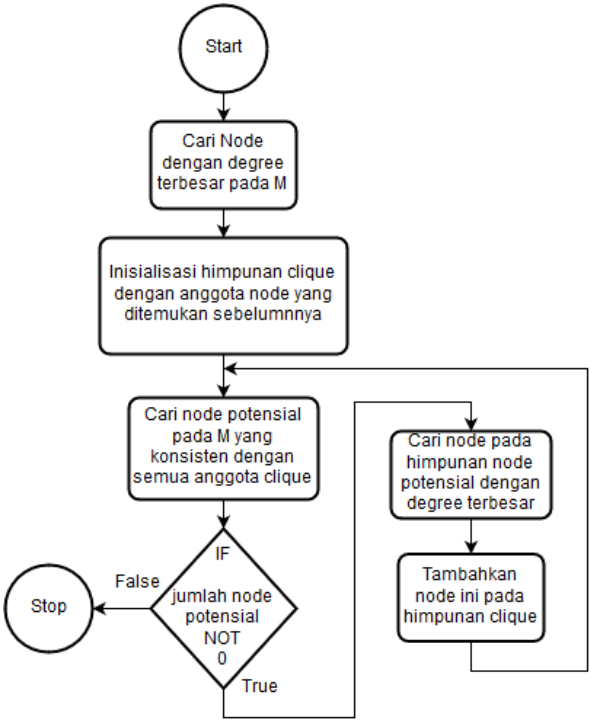
Jumlah anggota elemen M pada indeks i yang konsisten ditunjukkan pada variabel `currNodePositive`, variabel ini akan dibandingkan dengan jumlah *node* pada *clique* sekarang dan apabila nilainya sama maka *node* indeks i ini dimasukkan pada variabel `potentialNodes` karena konsisten dengan semua *node* anggota `clique.`

Tahap ketiga adalah mendapatkan satu *node* dengan konsistensi paling besar/banyak antara satu *node* dengan *node* lain pada variabel *`potentialNodes`*, dan menambahkannya pada variabel `clique`. Proses ini dilakukan pada kode program berikut:

```
for(int i = 0; i < static_cast<int>(potential Nodes.size()); i++){

    int degreeCurr NodePotential = 0;
    for(int j = 0; j < static_cast<int>(potential Nodes.size());
j++){

        if( M[potential Nodes.at(i)][potential Nodes.at(j)] ){
            degreeCurr NodePotential++;
        }
    }
    if( degreeCurr NodePotential > degreeMaxPotential){
        degreeMaxPotential=degreeCurr NodePotential;
        node ToClique = potential Nodes.at(i);
    }
```

```
}
clique.push_back( nodeToClique );
```

Proses ini dilakukan dengan **mengulangi tahap kedua dan tahap ketiga** sampai tidak ada `potentialNode` yang ditemukan konsisten dengan semua anggota `clique` untuk ditambahkan. Algoritma secara keseluruhan untuk masalah pencarian *maximum clique* ini ditunjukkan pada gambar Gambar 3-23.



Gambar 3-23 *Flowchart* proses pencarian *maximum clique*

3.6.6. Estimasi posisi lokal relatif terhadap posisi awal

Estimasi perubahan posisi robot setiap iterasinya didapatkan dari perubahan posisi fitur-fitur inlier selama waktu t-1 dan t, dengan mengikutkan informasi rotasi selama waktu t-1 dan t pada perhitungan.

Pertama-tama adalah mendapatkan matriks rotasional dari informasi rotasi berupa sudut euler yang telah didapatkan. Hal ini dilakukan dengan fungsi `euler2rot(eulerTwist)`, dimana `eulerTwist` adalah rotasi yang terjadi selama waktu frame sebelumnya dan frame sekarang. `eulerTwist` didapatkan dengan perintah:

```
eulerTwist.at<double>(0,0) =   -(pitchPrev - pitchCurrent);
eulerTwist.at<double>(1,0) =   -(yawPrev   - yawCurrent);
eulerTwist.at<double>(2,0) =    (rollPrev  - rollCurrent);
```

dengan `pitchCurrent`, `yawCurrent`, `rollCurrent` adalah variabel yang menyimpan rotasi dari awal sampai sekarang, dan `pitchPrev`, `yawPrev`, `rollPrev` adalah data sebelumnya.

Dalam `euler2rot` dibuat matriks rotasional yang akan digunakan pada perhitungan perpindahan posisi dari frame sebelumnya kepada frame sekarang. Matriks rotasional ini disimpan pada objek `cv::Mat rotationMatrixTwist`.

Untuk menghitung perpindahan posisi saat ini, dilakukan penjumlahan dari semua perpindahan fitur pada koordinat 3d, dan merata-rata nilai ini dengan jumlah fitur yang ada. Hasil perhitungan ini disimpan pada variabel `avgDisplacement`. Setelah itu dilakukan penjumlahan rekursif untuk mendapatkan posisi robot relatif terhadap posisi awalnya. Hal ini dilakukan dengan melakukan rotasi terlebih dahulu pada `avgDisplacement` sesuai dengan rotasi yang terjadi.

Pertama-tama dibuat matriks rotasional untuk melakukan rotasi pada avgDisplacement dengan kode program:

```
eulerPose.at<double>(0,0) = - pitchCurrent;
eulerPose.at<double>(1,0) = - yawCurrent;
eulerPose.at<double>(2,0) =   rollCurrent;

cv::Mat rotationMatrixPose = euler2rot(eulerPose);
```

dimana eulerPose adalah variabel rotasi euler dari awal alat dinyalakan sampai sekarang. Dengan menggunakan perintah `euler2rot(eulerPose)` didapatkan matriks rotasional yang digunakan untuk merotasi `avgDisplacement` sebelum dilakukan sum rekursif.

Setelah didapatkan matriks rotational, dicari nilai `avgDisplacement` yang sudah terotasi oleh `rotationMatrixPose` dengan melakukan operasi *dot product* pada keduanya. Variabel ini disimpan pada variabel `avgDisplacement_rotated`. Setelah didapatkan variabel ini, dapat dicari posisi robot dari posisi awal dengan perintah:

```
displacement = displacement + avgDisplacement_rotated;
```

`displacement` adalah variabel posisi robot relatif dari posisi awal robot.

3.7. PERANCANGAN PENGUJIAN SISTEM

Dalam system yang dirancang, diperlukan beberapa pengujian dala tahapan-tahapan perancangannya untuk menguji hasil dari sub-sistem dan komponen-komponen yang membangunnya, dan juga menguji hasil keluaran sistem secara keseluruhan. Pengujian-pengujian yang dibutuhkan adalah:

3.7.1. Pengujian frame rate hasil rektifikasi & disparity

Pengujian ini dilakukan untuk mengetahui frame rate yaitu berapa jumlah frame per detik yang didapatkan dari gambar hasil rektifikasi dan *disparity map*. Hal ini akan menjadi catatan untuk mengetahui tingkat kemampuan sistem memproses gambar, dan menguji apakah sistem layak digunakan pada penggunaan waktu riil.

3.7.2. Pengujian drift sudut yaw IMU

Pengujian ini dilakukan untuk mengetahui drift sudut yaw dari IMU. Hal ini dilakukan untuk menguji apakah drift dari device yang digunakan sama seperti spesifikasi dari IMU tersebut. Selain itu hal ini juga menguji apakah divais ini memenuhi spesifikasi untuk digunakan pada sistem ini dan digunakan pada pengujian berikutnya.

3.7.3. Pengujian deteksi fitur

Dalam pengujian ini akan diuji berapa jumlah fitur yang didapat dalam beberapa percobaan. Akan dinilai apakah algoritma yang digunakan sudah menghasilkan keluaran yang diharapkan, dan apakah penggunaan teknik *bucketing* sudah mencukupi untuk digunakan pada penggunaannya sebagai dasar proses *visual odometry*.

Variabel yang akan diuji adalah fitur total terdeteksi dan inlier fitur yang didapat. Pengujian akan dilakukan dengan menggerakkan robot pada suatu lintasan arena robot. Selama robot bergerak akan dihitung kedua variabel tersebut dan setelahnya akan dilakukan pengolahan data yang didapat.

3.7.4. Pengujian triangulasi fitur

Dalam pengujian ini akan diuji apakah teknik triangulasi yang diimplementasikan sudah menghasilkan perhitungan yang mencukupi dengan *ground truth* yang ada. Akan diuji hasil triangulasi dengan bebarapa *ground truth* yang berbeda jaraknya terhadap kamera, apakah hasil triangulasi sudah cukup dibandingkan *ground truth* yang diketahui. Dengan menguji hasil triangulasi ini dapat diketahui tingkat keakurasian proses triangulasi dan sekaligus menguji hasil proses pencarian *disparity map.*

3.7.5. Pengujian hasil estimasi posisi

Pengujian hasil estimasi posisi ini dilakukan untuk menguji seluruh sistem dengan membandingkan hasil estimasi posisi robot saat melakukan gerakan dengan *ground truth* atau posisi asli terukur, relatif dari posisi awal. Variabel yang diuji adalah koordinat x,y, dan sudut heading.

[Halaman ini sengaja dikosongkan]

BAB 4
PENGUJIAN DAN ANALISA SISTEM

Pada bab ini, hasil pengujian dari rancangan yang telah dibuat pada Bab 3 dipaparkan. Dari hasil pengujian yang didapatkan dilakukan analisa terhadapnya. Pengujian dan analisa ini bertujuan untuk mengukur tingkat ketercapaian dari sistem yang dirancang.

4.1. Pengujian Frame Rate

Pada pengujian ini akan dilakukan pengujian frame rate dari:

- Gambar hasil rektifiksi
- *Disparity map*

Pengujian ini akan menentukan berapa frame yang dihasilkan pada satu detik. Hal ini akan dilakukan dengan mengukur frame rate dari waktu ke waktu dan akan dihitung berapa frame rate rata-rata yang didapat dari masing-masing gambar.

Hasil dari pengujian ini akan menentukan seberapa kemampuan dari sistem yang dirancang untuk menghasilkan informasi yang dibutuhkan untuk tahap-tahap berikutnya. Selain itu frame rate dari kedua informasi ini juga menentukan seberapa kemampuan sistem yang dirancang untuk digunakan pada keadaan waktu-riil.

4.1.1. Pengujian frame rate hasil rektifikasi

Pengujian pertama dilakukan pada gambar hasil rektifikasi. Karena hasil rektifikasi di *publish* pada sistem ROS, dapat digunakan perintah `rostopic hz /topic_name` pada terminal ubuntu untuk mengetahui berapa refresh rate dari *topic* tersebut. Dengan memasukkan nama *topic* dari gambar hasil rektifikasi, dapat didapatkan refresh rate setiap detiknya dari gambar hasil rektifikasi.

Hal ini dilakukan selama 3 menit dan dilakukan pengambilan data setiap interval 15 detik. Setelah dilakukan pengambilan data, dihitung rata-rata refresh rate dari gambar hasil rektifikasi.

Tabel 4-1 Hasil pengambilan data refresh rate gambar hasil rektifikasi

No	Refresh Rate (hz)
1	29.79
2	29.78
3	29.78
4	29.78
5	29.78
6	29.78
7	29.78
8	29.78
9	29.78
10	29.78
11	29.78
12	29.78
Avg	**29.78**

4.1.2. Pengujian frame rate *disparity map*

Pengujian kedua dilakukan pada *disparity map* yang dihasilkan sistem. Metode yang digunakan sama seperti pengujian sebelumnya yaitu dengan mengambil data dengan interval 15 detik selama 3 menit.

Tabel 4-2 Hasil pengambilan data refresh rate *disparity map*

No	Refresh Rate (hz)
1	29.79
2	29.78
3	29.79
4	29.8
5	29.75
6	29.77
7	29.75
8	29.76
9	29.76
10	29.71
11	29.6
12	29.56
Avg	**29.735**

4.1.3. Analisa data

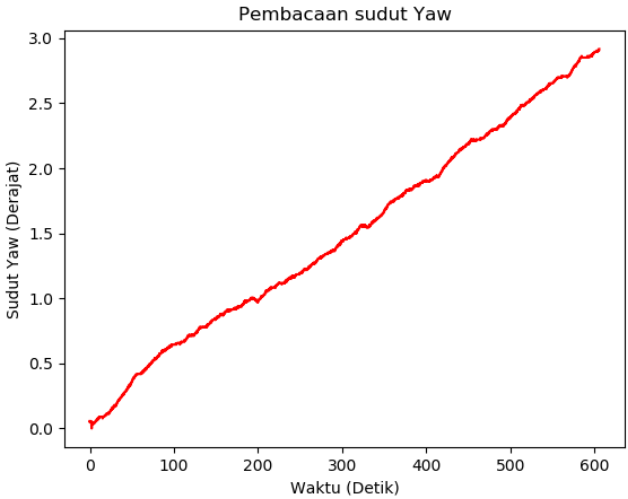
Didapatkan rata-rata refresh rate pada hasil gambar rektifikasi didapatkan 29.78, dengan begitu dapat dihitung bahwa rata-rata waktu pemrosesan rektifikasi gambar adalah 0.033 detik. Sedangkan pada data *disparity map* didapatkan refresh rate rata-rata 29.735, dengan waktu pemrosesan rata-rata didapatkan 0.033 detik.

Dengan hasil ini penulis simpulkan bahwa performa dari pemrosesan kedua data ini dapat digunakan pada penggunaan waktu-riil. Hal ini karena mengingat spesifikasi kamera yang digunakan memiliki FPS maksimal dengan nilai 30 FPS, maka hanya kehilangan 1 *Frame* per detiknya.

4.2. Pengujian Drift Sudut Yaw IMU

Pengujian ini dilakukan dengan merekam sudut yaw yang terbaca dari IMU pada keadaan diam selama 10 menit. Dengan keadaan diam maka dapat diasumsikan sudut yang seharusnya terbaca selalu menunjukkan 0 derajat. Dengan pengujian ini maka akan terlihat berapa drift sudut yaw dari divais IMU yang digunakan.

Didapatkan hasil pada Gambar 4-1.



Gambar 4-1 Pembacaan sudut yaw

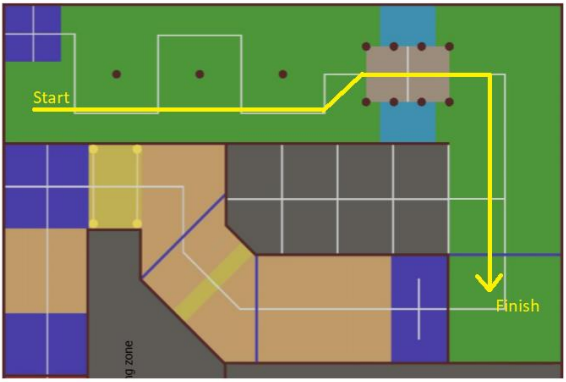
Didapatkan hasil dengan pembacaan pada detik ke 600 sebesar 2.9 derajat dan tidak menyimpang dari spesifikasi divais. Disimpulkan pula bahwa divais dapat digunakan untuk pengujian lebih lanjut pada estimasi posisi robot, karena pengujian kedepannya tidak lebih dari 2 menit.

4.3. Pengujian Deteksi Fitur

Pengujian ini akan dilakukan dengan menghitung rata-rata dari:

- Total fitur yang terdeteksi
- *Inlier* yang didapatkan

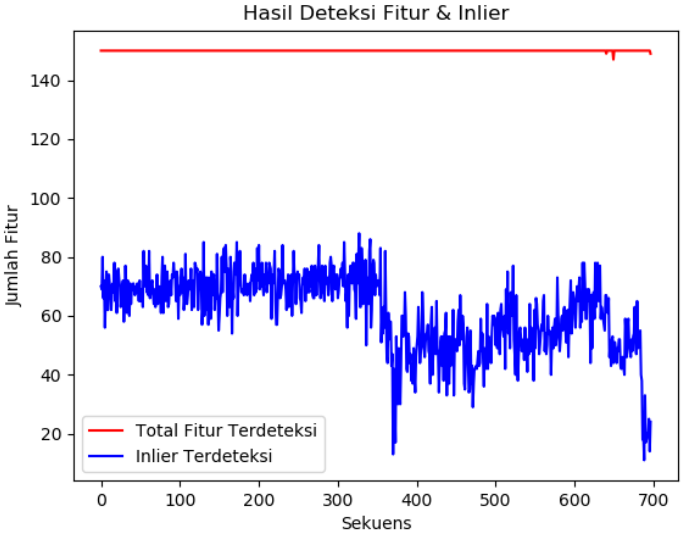
Pengujian ini dilakukan dengan menggerakkan robot pada suatu lintasan (Gambar 4-2) dan merekam semua variabel diatas selama robot berjalan. Setelah didapatkan data, dilakukan perhitungan rata-rata dari ketiga variabel tersebut selama robot bergerak. Dengan total fitur terdeteksi adalah total seluruh fitur pada bucket, dan *inlier* adalah total fitur yang konsisten satu dengan lainnya.



Gambar 4-2 Lintasan pengujian deteksi fitur

Pengujian ini akan menghasilkan karakteristik deteksi fitur pada sistem yang dirancang. Hasil pengujian ini akan menentukan apakah sistem yang dirancang sudah menghasilkan hasil yang sesuai dalam mendeteksi fitur dan memilih *inlier* fitur.

Didapatkan hasil deteksi dengan jumlah total fitur dan jumlah *inlier* terdeteksi pada Gambar 4-3.



Gambar 4-3 Total fitur dan Inlier terdeteksi

Didapatkan hasil pengujian dengan rata-rata total fitur terdeteksi sejumlah 149.99, dengan rata-rata inlier terdeteksi dengan jumlah 61.35. Dengan hasil ini dinilai jumlah fitur yang terdeteksi pada lintasan memenuhi untuk lintasan digunakan pada pengujian estimasi posisi selanjutnya.

4.4. Pengujian Triangulasi Fitur

Pada bagian ini dilakukan pengujian dengan melakukan perbandingan antara estimasi jarak 3d hasil perhitungan sistem dari suatu benda di depan kamera dibandingkan dengan jarak sesungguhnya yang terukur dari kamera. Pada bagian ini dilakukan pengukuran dalam satu sumbu Z dari *optical frame* kamera.

Pengujian dilakukan dengan meletakkan blok *styrofoam* pada jarak-jarak tertentu yang diketahui di depan kamera dan melakukan triangulasi untuk mendapatkan estimasi jaraknya. Dari dua data jarak ini,

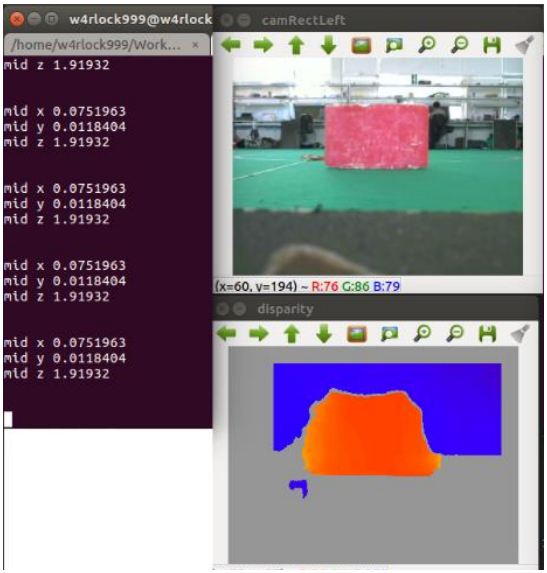
dibandingkanlah untuk mengetahui tingkat keakurasian sistem yang dirancang dalam melakukan estimasi jarak fitur.

Tabel 4-3 Jarak sebenarnya dan jarak hasil estimasi triangulasi dalam meter

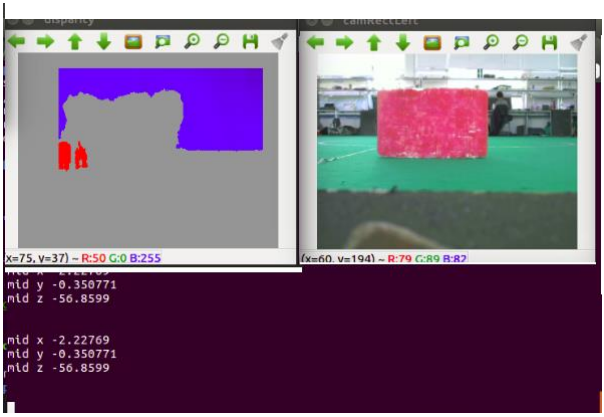
No	Jarak Sebenarnya (m)	Jarak Hasil Estimasi Triangulasi (m)
1	1.5	-56
2	1.8	1.8
3	2.1	2.12
4	2.4	2.4
5	2.7	2.73
6	3	2.98
7	3.3	3.3
8	3.6	3.64
9	3.9	3.95
10	4.2	4.15
11	4.5	4.51
12	4.8	4.87
13	5.1	5.12
14	5.4	5.22
15	5.7	5.58
16	6	6.08
17	6.3	6.39
18	6.6	6.63
19	6.9	6.93
20	7.2	7.29

Pada data ke-1 hasil estimasi sangat jauh dari hasil pengukuran. Hal ini dikarenakan objek benda terlalu dekat dengan kamera. Hal ini mengakibatkan data *disparity* tidak muncul karena nilai disparity yang terlalu besar.

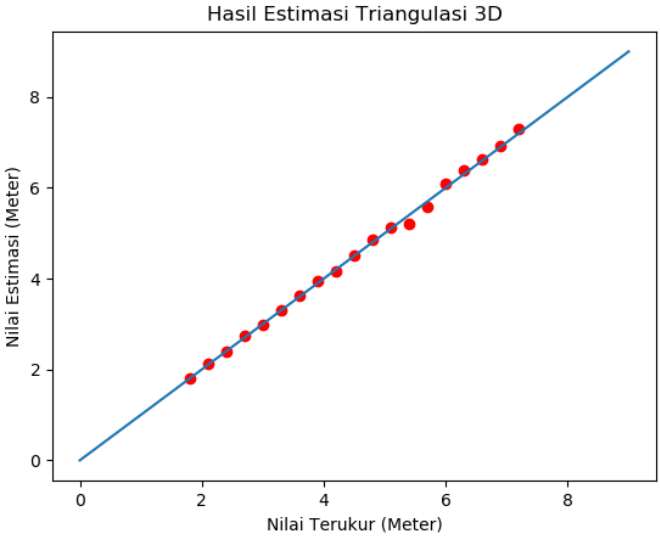
Pembatasan nilai *disparity* dilakukan sebelumnya untuk membatasi proses pencarian *matching point* pada tahap pencarian *disparity map*, sehingga pencarian *matching point* tidak memakan waktu relatif lebih lama, efeknya *disparity* yang bernilai besar tidak terdeteksi.



Gambar 4-4 Tampilan hasil triangulasi titik



Gambar 4-5 Pengujian data ke-1, disparity objek tidak terdeteksi

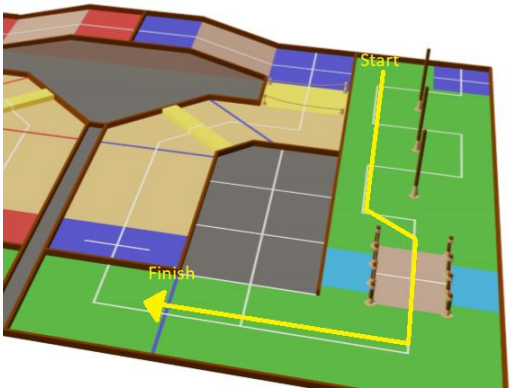


Gambar 4-6 Hubungan nilai terukur dengan nilai estimasi triangulasi tanpa data ke-1

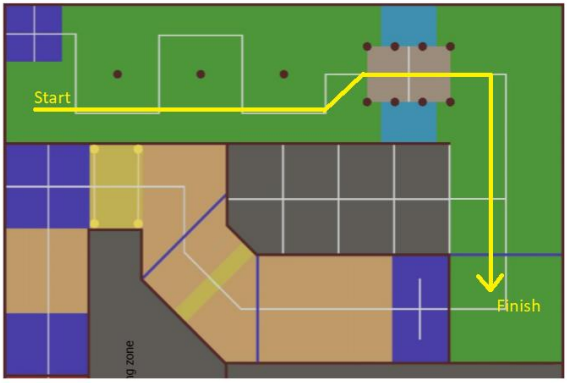
Dengan menganggap bahwa data ke-1 sebagai pengecualian, maka didapatkan **RMSE sebesar 0.066 meter** untuk hasil estimasi jarak benda terhadap kamera dengan proses triangulasi. Dengan hasil ini, penulis menganggap hasil triangulasi fitur pada sistem yang dirancang memiliki tingkat akurasi yang baik.

4.5. Pengujian Estimasi Posisi Lokal Robot

Pada pengujian ini dilakukan perbandingan estimasi posisi robot, relatif terhadap posisi awalnya dibandingkan dengan jarak yang terukur pada lintasan yang dilalui (*ground truth*). Pengujian ini dilakukan pada arena Kontes Robot ABU Indonesia milik Tim KRAI Institut Teknologi Sepuluh Nopember, yang bertempat di Ruang Workshop, Gedung Pusat Robotika ITS. Pengujian gerakan yang dilakukan ditunjukkan pada gambar Gambar 4-7.



a



b

Gambar 4-7 Lintasan pengujian pada arena robot secara isometris (a) dan tampak atas (b)

Pengujian ini dilakukan dengan mengamati trajektori hasil estimasi dari sistem yang telah dirancang pada titik-titik pada lintasan yang dilewati yang telah diketahui jarak/posisi nya relatif dari posisi awal robot. Pengujian dilakukan 3 kali untuk menguji konsistensi estimasi posisi dari sistem. *Mock mobile robot* yang telah dibuat didorong oleh penulis pada lintasan yang sudah ditentukan dengan bersama itu merekam

data hasil estimasi posisi setiap saat. Juga dilakukan input saat robot melewati titik-titik yang akan menjadi titik pengujian.

Tabel 4-4 Hasil pengujian ke-1 estimasi posisi robot.

	Hasil Terukur		**Hasil Estimasi**	
No	**X**	**Y**	**X**	**Y**
1	1	0	0.85	0
2	2	0	1.83	0.06
3	3	0	2.66	0.1
4	4	0	3.43	0.12
5	5	0	4.7	0.12
6	6	0.5	5.8	0.65
7	7	0.5	7.6	0.67
8	8	0.5	9.4	0.68
9	8	0	9.88	0.04
10	8	-1	9.9	-1
11	8	-2	9.9	-2.18
12	8	-3	10.08	-3.8

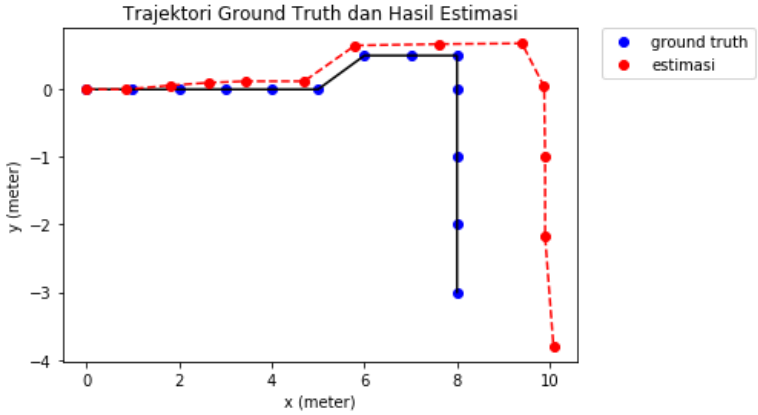
Tabel 4-5 Hasil pengujian ke-2 estimasi posisi robot.

	Hasil Terukur		**Hasil Estimasi**	
No	**X**	**Y**	**X**	**Y**
1	1	0	1.5	-0.1
2	2	0	2.18	-0.15
3	3	0	3.3	-0.15
4	4	0	4.7	-0.15
5	5	0	6	0
6	6	0.5	6.6	0.25
7	7	0.5	9.6	0.25
8	8	0.5	11.42	0.25
9	8	0	11.2	-1.2
10	8	-1	11.2	-2.32
11	8	-2	11.1	-4.1
12	8	-3	11.1	-5.3

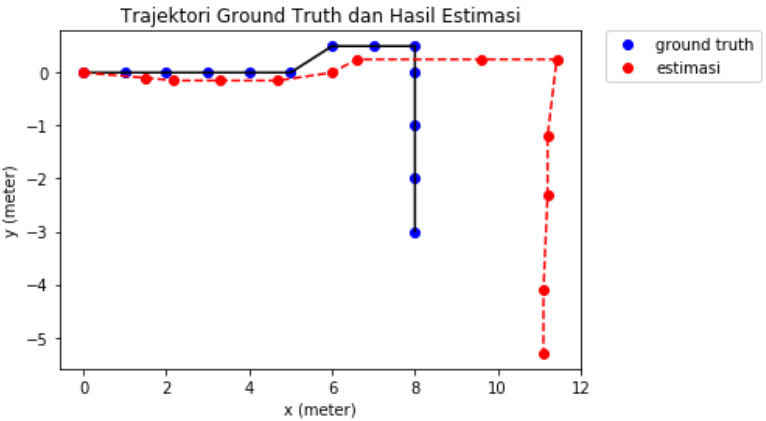
Tabel 4-6 Hasil pengujian ke-3 estimasi posisi robot.

	Hasil Terukur		Hasil Estimasi	
No	**X**	**Y**	**X**	**Y**
1	1	0	0.98	0.12
2	2	0	2.6	0
3	3	0	3.3	0.15
4	4	0	4.5	0.2
5	5	0	5.9	0.12
6	6	0.5	7.58	0.57
7	7	0.5	9.09	0.6
8	8	0.5	10.59	0.54
9	8	0	11.03	0.075
10	8	-1	11.03	-1.05
11	8	-2	11.02	-2.39
12	8	-3	11	-3.8

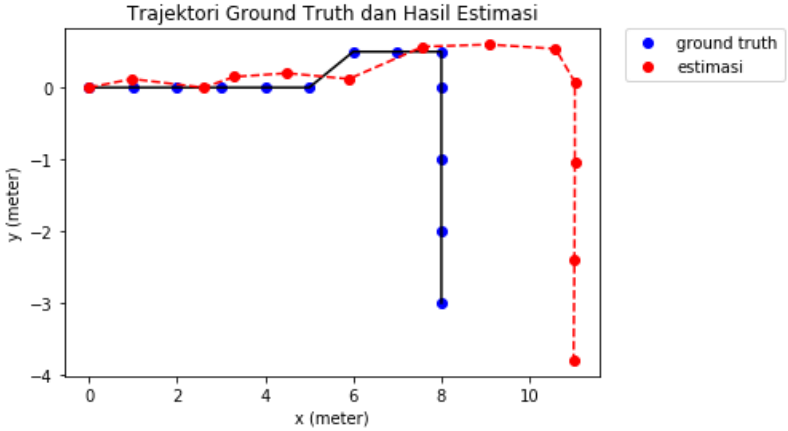
Hasil dari 3 kali pengujian ditunjukkan pada 3 gambar grafik berikut,



Gambar 4-8 Pengujian estimasi posisi 1



Gambar 4-9 Pengujian estimasi posisi 2



Gambar 4-10 Pengujian estimasi posisi 3

Dari ketiga pengujian ini, didapatkan hasil RMSE**:**

No	**x**	**y**	**d**
1	**1.22 m**	**0.25 m**	**1.2 m**
2	**2.24 m**	**1.04 m**	**2.21 m**
3	**2.07 m**	**0.27 m**	**2.04 m**

Terlihat walaupun saat diam divais kamera stereo dapat mengestimasi fitur dengan akurasi yang baik (pada pengujian sebelumnya), saat digunakan dengan bergerak divais stereo memiliki penurunan akurasi.

Terlihat pada titik ke 7 pada setiap pengujian, estimasi mulai melenceng jauh pada sumbu x. Juga pada sumbu y mulai titik 12-13. Hal ini karena pandangan robot pada titik 7 dan titik 12 tidak ada fitur-fitur dekat yang terdeteksi, fitur yang ada berada jauh didepan robot dan mengurangi akurasi triangulasi.

Selain itu ketidak konsistenan terjadi pada ketiga pengujian ini. Hal ini dianalisa dikarenakan sinkronisasi kamera kanan dan kiri yang tidak sempurna, yang mengakibatkan kesalahan perhitungan *disparity*. Saat bergerak data pada kamera kiri dan kanan tidak sinkron sepersekian detik, maka algoritma *feature matching* pada disparity akan menghasilkan kesalahan. Karena hal ini, mengakibatkan perhitungan estimasi posisi pun juga mengalami kesalahan.

[Halaman ini sengaja dikosongkan]

BAB 5
KESIMPULAN DAN SARAN

5.1. Kesimpulan

Dari hasil simulasi dan analisa pada bab 4, penulis mengambil beberapa kesimpulan sebagai berikut:

1. Hasil gambar rektifikasi dan *disparity map* menunjukkan refresh rate yang cukup untuk dilakukan proses dalam waktu riil. Hal ini terlihat pada refresh rate gambar rektifikasi yang menghasilkan 29.78 frame per detik, dan *disparity map* yang menghasilkan 29.735 frame per detik.
2. Deteksi fitur dan deteksi inlier pada saat robot bergerak disimpulkan cukup untuk dilakukan pengolahan fitur dengan keadaan robot yang bergerak. Hal ini ditunjukkan dengan dihasilkannya rata-rata 149.99 fitur dan 61.35 *inlier* pada pengujian.
3. Hasil triangulasi fitur dalam koordinat 3d disimpulkan memiliki akurasi yang cukup baik, dengan hanya memiliki error RMSE sebesar 0.066 Meter pada pengujian yang dilakukan.
4. Hasil estimasi posisi robot disimpulkan masih belum stabil. Ditunjukkan pada perbedaan antara posisi terukur dan posisi estimasi yang memiliki error RMSE 1.2 Meter, 2.21 Meter dan 2.04 Meter, serta hasil setiap pengujian yang tidak konsisten.

5.2. Saran

Saran yang dapat disampaikan oleh penulis untuk kelanjutan topik tugas akhir ini adalah bahwa sinkronisasi gambar kiri dan kanan perlu diteliti lebih lanjut. Hal ini adalah karena pada tugas akhir ini belum meneliti mendalam kepada sinkronisasi gambar dari kedua kamera. Harapannya jika gambar kiri dan kanan dapat tersinkronisasi dengan sempurna, maka hasil *disparity map* yang dihasilkan akan lebih stabil dan hasil estimasi posisi saat bergerak akan lebih baik, stabil, dan akurat.

Poin lain yang perlu diteliti adalah efek resolusi atau ukuran gambar terhadap akurasi triangulasi fitur 3d. Karena proses triangulasi pada tugas akhir ini sangat bergantung pada nilai *disparity* maka jika resolusi gambar dapat ditingkatkan kemungkinan akurasi hasil triangulasi akan naik

karena resolusi *disparity* yang juga naik. Hal ini akan berefek pada keadaaan dimana fitur yang terdeteksi berjarak relatif jauh dari kamera.

Terakhir adalah diperlukannya riset lebih lanjut untuk mengestimasi rotasi menggunakan perpindahan fitur-fitur yang terdeteksi. Hal ini memerlukan sinkronisasi kamera yang mumpuni, hasil *disparity* yang sangat stabil pada saat robot bergerak, serta resolusi *disparity* yang lebih tinggi.

DAFTAR PUSTAKA

[1] "Mat - The Basic Image Container — OpenCV 2.4.13.7 documentation." [Online]. Available: https://docs.opencv.org/2.4/doc/tutorials/core/mat_the_basic_image_container/mat_the_basic_image_container.html#matthebasicimage container. [Accessed: 20-May-2019].

[2] Richard Hartley and Andrew Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, 2004.

[3] Nicholas Ayache, *Artificial Vision for Mobile Robots*. Massachusetts: MIT Press, 1991.

[4] Huihuang Su and Bingwei He, "Stereo rectification of calibrated image pairs based on geometric transformation," *I.J. Modern Education and Computer Science*, 2011.

[5] pardo-beainy *et al.*, "Disparity map generation from the use of rectified images," *Symposium of Signals, Images and Artificial Vision*, pp. 1–6, 2013.

[6] Eugene Khvedchenya, "Comparison of the OpenCV's feature detection algorithms," *Comparison of the OpenCV's feature detection algorithms*, 04-Jan-2011. .

[7] Edward Rosten and Tom Drummond, "Machine Learning for High-Speed Corner Detection," *ECCV 2006: 9th European Conference on Computer Vision 2006*, Jul. 2006.

[8] "Camera Calibration and 3D Reconstruction — OpenCV 2.4.13.7 documentation." [Online]. Available: https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html. [Accessed: 16-May-2019].

[9] Carlo Tomasi and Takeo Kanade, "Detection and Tracking of Point Features," Apr. 1991.

[10] Jianbo Shi and Carlo Tomasi, "Good Features to Track," *IEEE Conference on COmputer Vision and Pattern Recognition*, 1994.

[11] Andrew Howard, "Real-Time Stereo Visual Odometry for Autonomous Ground Vehicles," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3946–3952, 2008.

[12] "Logitech HD Webcam C270 Technical Specifications." [Online]. Available: https://support.logitech.com/en_us/article/17556. [Accessed: 23-May-2019].

[Halaman ini sengaja dikosongkan]

LAMPIRAN

Lampiran 1: Main Program Node Stereo Capture

```
int main(int argc, char **argv)
{

    ros::init(argc, argv, "stereoCapture");

    ros::NodeHandle NH;
    ros::MultiThreadedSpinner MTS;


    image_transport::ImageTransport IT(NH);
    image_transport::Publisher pubCamLeft =
IT.advertise("camera/left/image_raw",1);
    image_transport::Publisher pubCamRight =
IT.advertise("camera/right/image_raw",1);

    ros::Publisher pubCamLeftInfo =
NH.advertise<sensor_msgs::CameraInfo>("camera/left/camera_info",1)
;
    ros::Publisher pubCamRightInfo =
NH.advertise<sensor_msgs::CameraInfo>("camera/right/camera_info",1
);

    StereoCaptureClass capture(30,1,2);

    cv_bridge::CvImagePtr cameraLeft_image;
    cv_bridge::CvImagePtr cameraRight_image;

    cameraLeft_image = boost::make_shared< cv_bridge::CvImage >();
    cameraRight_image = boost::make_shared< cv_bridge::CvImage
>();

    cameraLeft_image->encoding =
sensor_msgs::image_encodings::BGR8;
```

```
    cameraRight_image->encoding =
sensor_msgs::image_encodings::BGR8;

    sensor_msgs::ImagePtr cameraLeft_msg;
    sensor_msgs::ImagePtr cameraRight_msg;

    Mat frame1;
    Mat frame2;

    //camera info
    const std::string camLeftURL =
"file:///home/w4rlock999/Workspace/ros_itsrobocon/src/stereo_nav_p
kg/config/calib1/calibrationdata/left.yaml";
    const std::string camRightURL =
"file:///home/w4rlock999/Workspace/ros_itsrobocon/src/stereo_nav_p
kg/config/calib1/calibrationdata/right.yaml";

    camera_info_manager::CameraInfoManager camLeftInfo(NH,
"camera/left", camLeftURL);
    camera_info_manager::CameraInfoManager camRightInfo(NH,
"camera/right", camRightURL);

    sensor_msgs::CameraInfo camLeftInfoMsgs;
    sensor_msgs::CameraInfo camRightInfoMsgs;

    std_msgs::Header leftStamp;
    std_msgs::Header rightStamp;

    XmlRpc::XmlRpcValue shared_params;
    nodelet::Loader manager(false);
    nodelet::M_string remappings;
    nodelet::V_string my_argv;

    loadImgProc(manager, "left", shared_params, my_argv);
    loadImgProc(manager, "right", shared_params, my_argv);

    //=======================
```

```
    //=======================
    //   disparity nodelet
    //=======================
    //=======================

    std::string left_image_topic    =
ros::names::resolve("camera/left/image_rect");
    std::string left_info_topic     =
ros::names::resolve("camera/left/camera_info");
    std::string right_image_topic   =
ros::names::resolve("camera/right/image_rect");
    std::string right_info_topic    =
ros::names::resolve("camera/right/camera_info");
    std::string disparity_topic     =
ros::names::resolve("camera/disparity");

    remappings["left/image_rect"]   = left_image_topic;
    remappings["left/camera_info"]  = left_info_topic;
    remappings["right/image_rect"]  = right_image_topic;
    remappings["right/camera_info"] = right_info_topic;
    remappings["disparity"]         = disparity_topic;

    bool approx_sync;
    if(NH.getParam("approximate_sync", approx_sync))
        shared_params["approximate_sync"] =
XmlRpc::XmlRpcValue(approx_sync);

    std::string disparity_name = ros::this_node::getName();
    manager.load(disparity_name, "stereo_image_proc/disparity",
remappings, my_argv);

    while(ros::ok()){

        capture.getLeftRightFrames(cameraLeft_image->image,
cameraRight_image->image);
```

```
        if(!cameraLeft_image->image.empty() && !cameraRight_image-
>image.empty() ){

            camLeftInfoMsgs = camLeftInfo.getCameraInfo();
            leftStamp.stamp = ros::Time::now();
            cameraLeft_image->header.stamp = leftStamp.stamp;
            camLeftInfoMsgs.header.stamp = leftStamp.stamp;

            camRightInfoMsgs = camRightInfo.getCameraInfo();
            rightStamp.stamp = ros::Time::now();
            cameraRight_image->header.stamp = rightStamp.stamp;
            camRightInfoMsgs.header.stamp = rightStamp.stamp;

            pubCamLeft.publish(cameraLeft_image->toImageMsg());
            pubCamRight.publish(cameraRight_image->toImageMsg());

            pubCamLeftInfo.publish(camLeftInfoMsgs);
            pubCamRightInfo.publish(camRightInfoMsgs);

        }

    ros::spinOnce();
    }
    // MT.spin();

    return 0;
}
```

Lampiran 2: Program IMU Node

```
import rospy
import serial
import time
import struct
from std_msgs.msg import String
# import geometry_msgs
from geometry_msgs.msg import Pose, Quaternion
```

```
import tf_conversions


device = '/dev/ttyUSB0'
ser_to_imu = serial.Serial(device, 115200)
init = True

yaw = 0
pitch = 0
roll = 0

imuMsg = Quaternion()

degToRad = 0.0174533

def talker():
    pub = rospy.Publisher('chatter', String, queue_size=10)
    imuPub = rospy.Publisher('imu_quat', Quaternion, queue_size=5)
    rospy.init_node('imu node', anonymous=True)

    rate = rospy.Rate(10) # 10hz
    hello_str = "hello world, yaw: " + str(-yaw[0]) +"  "+ str(-
pitch[0]) + "  "+ str(roll[0])
    rospy.loginfo(hello_str)
    pub.publish(hello_str)
    imuPub.publish(imuMsg)


while 1:
    if init == True:
        ser_to_imu.read_until('its')
        print(' "its" sequence found, init complete, starting data
receiving ')
        init = False
    else:
        raw=ser_to_imu.read_until('its')
        yawTemp = raw[0]+raw[1]+raw[2]+raw[3]
```

```
        pitchTemp = raw[4]+raw[5]+raw[6]+raw[7]
        rollTemp = raw[8]+raw[9]+raw[10]+raw[11]
        yaw = struct.unpack('<f',yawTemp)
        pitch = struct.unpack('<f',pitchTemp)
        roll = struct.unpack('<f',rollTemp)

        # yaw = -yaw
        # pitch = -pitch
        # print(yaw[0])
        # print(pitch[0])
        # print(roll[0])

        # imuMsg.orientation =
geometry_msgs.msg.Quaternion(*tf_conversions.transformations.quate
rnion_from_euler(roll, pitch, yaw))
        imuMsg =
Quaternion(*tf_conversions.transformations.quaternion_from_euler(r
oll[0] * degToRad, (-pitch[0]) * degToRad, (-yaw[0]) * degToRad ))

        try:
            talker()
        except rospy.ROSInterruptException:
            pass
```

Lampiran 3: Program Inlier Detection

```
// ====================================
// ====================================
// 1. Create M consistency/adjacency matrix
// ====================================
// ====================================

int M[maxBucketFeatures * hBucket * vBucket][maxBucketFeatures *
hBucket * vBucket];

for(int i = 0; i < static_cast<int>(keypointsCurrentVP.size());
i++){
```

```
    for(int j = 0; j <
static_cast<int>(keypointsCurrentVP.size()); j++){

        float diffCurrent[3];
        float diffPrev[3];
        float rigidity[3];

        if(j != i){

            // only calculate if J > I,
            // M concistency matrix is diagonally symmetrical
            // is actually adjacency matrix
            if(j > i){
                // calculate current differences between point i
and point j on all axes
                diffCurrent[0]  = keypoint3dCurrent[i].x -
keypoint3dCurrent[j].x;
                diffCurrent[1]  = keypoint3dCurrent[i].y -
keypoint3dCurrent[j].y;
                diffCurrent[2]  = keypoint3dCurrent[i].z -
keypoint3dCurrent[j].z;

                // calculate previous differences between point i
and point j on all axes
                diffPrev[0]     = keypoint3dPrev[i].x -
keypoint3dPrev[j].x;
                diffPrev[1]     = keypoint3dPrev[i].y -
keypoint3dPrev[j].y;
                diffPrev[2]     = keypoint3dPrev[i].z -
keypoint3dPrev[j].z;

                // calculate rigidity for x,y,z coordinate on
current and prev absolute differences.
                // into --> 0 if perfectly fulfill rigidity
constraint
                for(int k = 0; k < 3; k++){
```

```
                    rigidity[k] = abs(diffPrev[k] -
diffCurrent[k]);

                    // if(rigidity[k] == 0.0){
                    //     std::cout<< "found zero on index " << k
<<"\n" ;
                    //     std::cout<< "previious difference is
"<<diffPrev[k]<<"\n";
                    //     std::cout<< "current difference is "<<
diffCurrent[k]<<"\n";
                    //     std::cout<< "disparity is " <<
disparity.at<float>(floor(keypointsCurrentVP[i].y),floor(keypoints
CurrentVP[i].x)) <<"\n";

                    // }

                    // std::cout <<k<<" : "<< rigidity[k] << "\n";
                }

                //fill M consistency matrix using rigidity
constraint
                // 1 if < threshold, 0 if > threshold
                if( rigidity[0] < 0.08 && rigidity[1] < 0.08 &&
rigidity[2] < 0.08){


if(disparity.at<float>(floor(keypointsCurrentVP[j].y),floor(keypoi
ntsCurrentVP[j].x)) <= 0 ){
                        M[i][j] = 0;
                        M[j][i] = 0;
                    }else{
                        M[i][j] = 1;
                        M[j][i] = 1;
                    }


                }else{
```

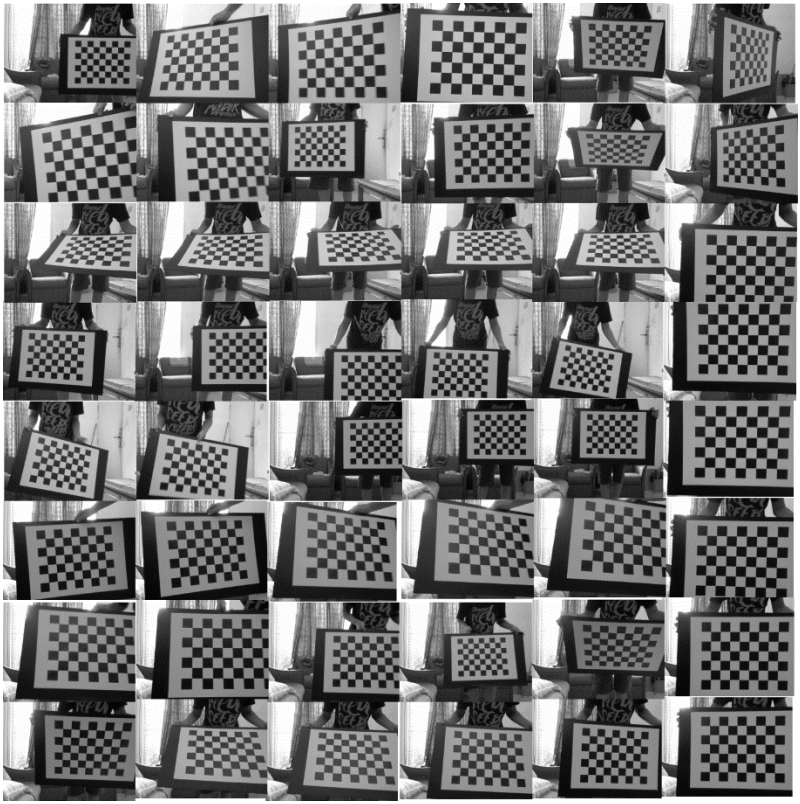
```
                    M[i][j] = 0;
                    M[j][i] = 0;
                }

            }
        }else{

            M[i][j] = 1;
        }
    }
}

// DISPLAY M MATRIX BELOW
// ========================

// std::cout << "\n" ;
// for(int i = 0; i < static_cast<int>(keypointsCurrentVP.size());
i++){

//     for(int j = 0; j <
static_cast<int>(keypointsCurrentVP.size()); j++){

//         std::cout << M[i][j] << " ";

//     }

//     std::cout << "\n";
// }


// ====================================
// ====================================
// 2. Initialize Clique with vertex/node with
// most degree
// ====================================
// ====================================
```

```
int degreeMax = 0;
int degreeCurrNode = 0;
std::vector<int> clique;
int initialNodeToClique;

for(int i = 0; i < static_cast<int>(keypointsCurrentVP.size());
i++){
    for(int j = 0; j <
static_cast<int>(keypointsCurrentVP.size()); j++){

        if( M[i][j] == 1){
            degreeCurrNode++;
        }
    }

    // initialize node with max degree as candidate
    if(degreeCurrNode > degreeMax){
        degreeMax = degreeCurrNode;
        initialNodeToClique = i;
    }

    degreeCurrNode = 0;
}

clique.push_back( initialNodeToClique );


while(1){

// ====================================
// ====================================
// 3. find potential node
// ====================================
// ====================================

    std::vector<int> potentialNodes;
```

```
    for(int i = 0; i <
static_cast<int>(keypointsCurrentVP.size()); i++){

        int currNodePositive = 0;
        for(int j = 0; j < static_cast<int>( clique.size()); j++
){

            if( M[i][clique.at(j)] )
                currNodePositive++;

        }

        if(currNodePositive == static_cast<int>( clique.size())){
            if(std::find(clique.begin(), clique.end(), i) ==
clique.end()){
                potentialNodes.push_back( i );
            }
        }
    }

    // std::cout<< "\n potentialnodes size: " <<
potentialNodes.size();

    // break the step 3-4 loop if no more inlier found
    if( static_cast<int>(potentialNodes.size() ) == 0 )
        break;

    // ====================================
    // ====================================
    // 4. update Clique from potential node
    // with most degree
    // ====================================
    // ====================================

    int degreeMaxPotential = 0;
    int nodeToClique;
```

```
    for(int i = 0; i < static_cast<int>(potentialNodes.size());
i++){

        int degreeCurrNodePotential = 0;
        for(int j = 0; j <
static_cast<int>(potentialNodes.size()); j++){

            if( M[potentialNodes.at(i)][potentialNodes.at(j)] ){
                degreeCurrNodePotential++;
            }
        }

        if( degreeCurrNodePotential > degreeMaxPotential){
            degreeMaxPotential=degreeCurrNodePotential;
            nodeToClique = potentialNodes.at(i);
        }

    }


    clique.push_back( nodeToClique );
}
```

Lampiran 4: Dataset Kalibrasi Kamera



Lampiran 5: File camera info hasil kalibrasi

Left.yaml:

image_width: 320
image_height: 240
camera_name: camera/left
camera_matrix:
  rows: 3
  cols: 3
  data: [417.336789, 0.000000, 153.932515, 0.000000, 417.370175, 130.681490, 0.000000, 0.000000, 1.000000]
distortion_model: plumb_bob
distortion_coefficients:
  rows: 1
  cols: 5
  data: [0.062901, 0.265136, -0.005361, -0.007217, 0.000000]
rectification_matrix:
  rows: 3
  cols: 3
  data: [0.999962, 0.008398, 0.002275, -0.008420, 0.999915, 0.009998, -0.002191, -0.010017, 0.999947]
projection_matrix:
  rows: 3
  cols: 4
  data: [463.776846, 0.000000, 141.829867, 0.000000, 0.000000, 463.776846, 117.138942, 0.000000, 0.000000, 0.000000, 1.000000, 0.000000]

Right.yaml:
image_width: 320
image_height: 240
camera_name: camera/right
camera_matrix:
  rows: 3

  cols: 3
  data: [415.002949, 0.000000, 142.939003, 0.000000, 415.637530, 105.996447, 0.000000, 0.000000, 1.000000]
distortion_model: plumb_bob
distortion_coefficients:
  rows: 1
  cols: 5
  data: [0.053452, 0.416603, -0.012894, -0.010673, 0.000000]
rectification_matrix:
  rows: 3
  cols: 3
  data: [0.999731, -0.004324, 0.022808, 0.004552, 0.999940, -0.009956, -0.022763, 0.010057, 0.999690]
projection_matrix:
  rows: 3
  cols: 4
  data: [463.776846, 0.000000, 141.829867, -56.859879, 0.000000, 463.776846, 117.138942, 0.000000, 0.000000, 0.000000, 1.000000, 0.000000]

[Halaman ini sengaja dikosongkan]

RIWAYAT HIDUP PENULIS

Ilham Wicaksono lahir pada tanggal 15 Januari 1997 di Jakarta. Setelah pada tahun 2000 pindah ke kota Solo, penulis menghabiskan masa sekolahnya di kota Solo. Merupakan anak tunggal dari pasangan Bachtiar Subandryo dan Amah Setiowati. Penulis menyelesaikan sekolah dasar di SD Muhammadiyah 1 Surakarta pada tahun 2009, sekolah menengah pertama di SMPN 9 Surakarta pada tahun 2012, dan sekolah menengah atas di SMAN 1 Surakarta pada tahun 2015. Memulai studinya di Institut Teknologi Sepuluh Nopember di program S1 Teknik Elektro pada tahun 2015. Penulis sangat meminati dunia robotika dimana selama kuliah menghabiskan waktunya berusaha berkontribusi pada tim ITS-Robocon untuk mewakili ITS pada Kontes Robot ABU Indonesia dari tahun 2016 sampai 2018. Pada tahun 2018 setelah menjuarai Kontes Robot ABU Indonesia tingkat nasional, penulis bersama tim RIVER ITS-Robocon berkesempatan mewakili Indonesia pada *ABU Asia-Pacific Robot Contest 2018* di Vietnam.

Email:
wicaksono.ilham99@gmail.com

[Halaman ini Sengaja dikosongkan]