# BAB II

# TINJAUAN PUSTAKA

## 2.1. Sistem Penentu Lokasi pada Kendaraan Otonom

Sistem penentu lokasi adalah implementasi dari algoritma untuk mengestimasi dimana lokasi dari kendaraan otonom berada. Tanpa sistem penentu lokasi, kendaraan tidak dapat mencapai tujuan dan tidak mengetahui kemana kendaraan tersebut menuju. Kendaraan otonom harus dapat berpindah dari satu tempat ke tempat yang lain, sehingga kendaraan otonom harus mengetahui lokasinya secara global. Selain itu, kendaraan otonom juga harus mengetahui lokasinya dengan lingkungan di sekitarnya agar kendaraan tersebut mengetahui kapan waktu untuk berhenti, mengubah arah, maupun berbelok. Sistem penentu lokasi kendaraan otonom dapat dibagi menjadi sistem penentu lokasi global, sistem penentu lokasi relatif, dan lokalisasi dan pemetaan secara simultan (SLAM) [3].

### 2.1.1. Sistem Penentu Lokasi Global

Sistem penentu lokasi secara global sangat penting dilakukan oleh kendaraan otonom agar dapat berpindah dari suatu tempat ke tempat lainnya. Navigasi dari kendaraan otonom dapat terbagi menjadi dua jenis, yaitu sistem navigasi satelit dan navigasi berdasarkan *landmark* atau tanda yang mencolok pada suatu lingkungan (*landmark based*) [3].

Pada sistem *landmark based,* apabila suatu tanda yang mencolok pada suatu lingkungan diketahui, lokasi dari kendaraan otonom dapat diketahui dengan mengukur posisi kendaraan dengan tanda tersebut. Tantangan terbesar dari sistem *landmark based* ini adalah diperlukan keakuratan dalam pendeteksian tanda tersebut.

Sistem navigasi satelit adalah sebuah sistem yang dapat memberikan posisi geospasial dari satelit yang dapat memberikan lokasi spesifik kendaraan otonom pada bumi. *Global Navigation Satellite System* (GNSS) adalah konstelasi satelit yang menyediakan layanan pemosisian dan navigasi tersebut. GNSS merupakan

satelit yang dapat memancarkan sinyal dan menangkap semua sinyal balik yang dipancarkan oleh *receiver* serta melakukan *pinpoint* lokasi dari *receiver* dimanapun di dunia [4]. Salah satu jenis GNSS yang sering digunakan adalah GPS (*Global Positioning System*). GPS mengirim sinyal yang mengandung informasi mengenai posisi dan waktu transmisi dari sinyal. Faktor-faktor yang menyebabkan kesalahan hasil pengukuran GPS:

a. Muatan ion pada *ionosphere* dapat menunda transmisi sinyal
b. Daerah sekeliling *receiver* juga mempengaruhi transmisi sinyal, contoh gedung dan pepohonan dapat menyebabkan pemantulan dari sinyal dari gps sehingga meningkatkan jarak yang tempuh oleh sinyal untuk mencapai receiver
c. Kesalahan pada proses sinkronisasi waktu.

### 2.1.2. Sistem Penentuan Lokasi Relatif

Untuk mengestimasi posisi relatif dari kendaraan otonom, terdapat tiga metode yang dapat digunakan, yaitu *Dead Reckoning*, navigasi inersia, dan odometri visual (VO). *Dead Reckoning* adalah metode perhitungan posisi kendaraan otonom pada saat ini dengan menggunakan posisi sebelumnya yang sudah ditentukan. Sensor yang dibutuhkan untuk sistem penentu lokasi ini relatif lebih murah dibanding sensor sistem penentu lokasi lainnya dan bebas dari interferensi sinyal. Namun, metode ini agak sulit digunakan pada lingkungan yang dinamis dan membutuhkan informasi posisi awal yang akurat [3].

Navigasi inersia adalah metode yang dapat digunakan untuk menghitung posisi secara relatif terhadap beberapa poin referensi dengan bantuan beberapa sensor seperti *gyroscope* dan *accelerometer* [5]. Jantung dari sistem navigasi inersia adalah IMU (*Inertial Measurement Unit*) yang terdiri atas tiga *gyroscope* ortogonal dan tiga *accelerometer* ortogonal. IMU akan mengukur pergerakan dengan 6 DOF (*degree of freedom*) atau derajat kebebasan.

Odometri visual adalah metode yang sedang sering dikembangkan dengan semakin canggihnya kemampuan komputasi dan *computer vision.* Odometri visual adalah proses penentuan posisi dan orientasi suatu robot/kendaraan berdasarkan perubahan dari gambar-gambar yang ditangkap oleh kamera.

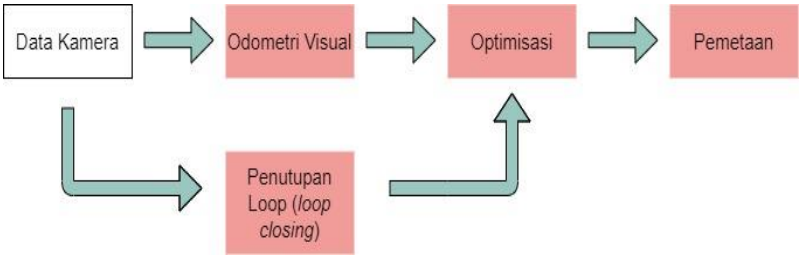
### 2.1.3. Lokalisasi dan Pemetaan Secara Simultan (SLAM)

Lokalisasi dan Pemetaan Secara Simultan (SLAM) adalah proses mengkonstruksi atau memperbarui peta dari lingkungan yang tidak diketahui sebelumnya sembari tetap melacak lokasi dari kendaraan otonom tersebut. SLAM sudah banyak digunakan pada robot, UAV, maupun kendaraan otonom. Teknologi SLAM terus dikembangkan sehingga terdapat SLAM yang optimal.

SLAM dapat dibagi berdasarkan jenis sensor yang digunakan, yaitu LiDAR SLAM dan Visual SLAM (vSLAM). LiDAR SLAM adalah metode SLAM dengan 3D LiDAR sebagai sensornya, sedangkan vSLAM menggunakan kamera sebagai sensornya. LiDAR SLAM dan vSLAM memiliki kelebihan dan kekurangan masing-masing. Kelebihan dari LiDAR adalah keakuratannya yang lebih tinggi dibanding vSLAM pada lingkungan statis. Namun, harga sensor LiDAR kurang terjangkau dibandingkan harga sensor kamera dan terkadang *point cloud* yang dihasilkan oleh sensor LiDAR tidak cukup efisien juga [2]. *Point cloud* adalah poin-poin yang dihasilkan oleh sensor LiDAR yang merepresentasikan objek atau fitur yang ada di sekitarnya. *Point cloud* LiDAR tidak dapat mendeteksi gambar dua dimensi. Walaupun vSLAM menghasilkan penentuan lokasi dan pemetaan yang cukup baik dan dapat mendetekasi gambar dua dimensi, vSLAM masih memiliki kekurangan, seperti penyimpangan faktor skala apabila menggunakan kamera monokuler, inisialisasi kedalaman yang tertunda apabila menggunakan kamera stereo, dan sulitnya penggunaan kamera RGB-D pada luar ruangan [2].

## 2.2. Visual SLAM (vSLAM)

Visual SLAM (vSLAM) adalah metode SLAM dengan menggunakan kamera untuk mengambil gambaran dari lingkungan sekitar. Sensor yang digunakan dapat berupa kamera stereo, kamera RGB-D, dan multi kamera. Kerangka kerja dari vSLAM terlampir pada Gambar 2.1.

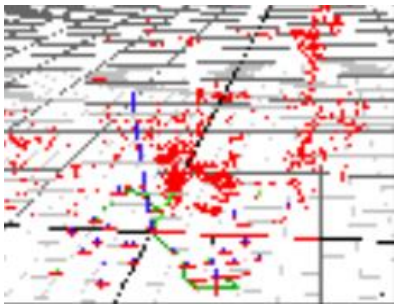



**Gambar 2.1.** Skema kerangka kerja dari Visual SLAM (vSLAM).

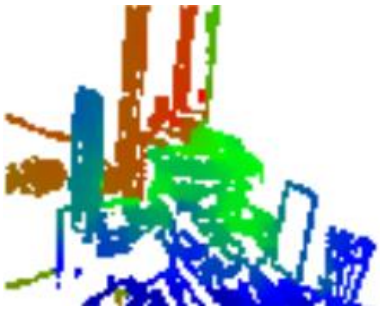
Dari skema kerangka kerja vSLAM tersebut, tahapan-tahapan vSLAM seperti odometri visual, optimisasi, penutupan *loop*, dan pemetaan akan dijelaskan lebih lanjut.

Odometri visual dari algoritma SLAM dilakukan untuk mengestimasi pergerakan kamera dan penampilan dari peta lokal menurut urutan *frame* gambar dan pasangan kedalaman gambar per-piksel [6]. Berdasarkan perbedaan cara penggunaan informasi gambar, teknik odometri visual dapat dibagi menjadi dua cara, yaitu metode yang berdasarkan fitur (*feature-based*) dan metode langsung (*direct*).

*Feature-based* vSLAM adalah metode pendeteksian fitur pada gambar, kemudian dilakukan pencocokan fitur tersebut seiring dengan waktu. Berdasarkan hubungan fitur pada tiap *frame*, pergerakan relatif antar *frame* dapat dihitung [7]. *Direct* vSLAM adalah metode dengan menggunakan keseluruhan informasi gambar tanpa abstraksi dengan detektor dan deskriptor fitur [8]. Konsistensi fotometrik digunakan sebagai pengukuran kesalahan dengan metode ini. Konsistensi fotometrik adalah fungsi skalar yang mengukur kesesuaian visual dari rekonstruksi tiga dimensi. Teknik *feature-based* vSLAM dan *direct* vSLAM terlampir pada Gambar 2.2.



(a)



(b)

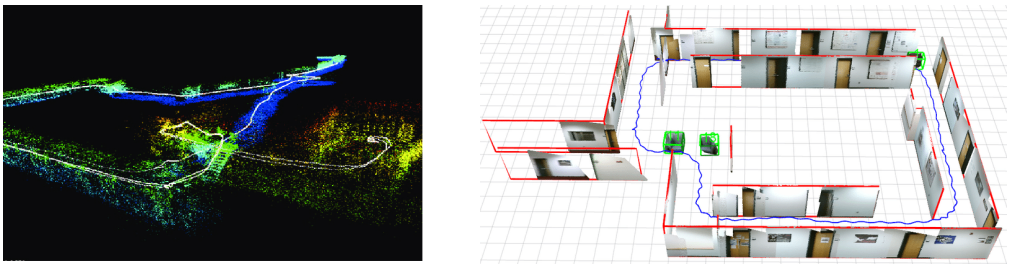
**Gambar 2.2.** (a) Teknik *fetaure-based* vSLAM, (b) Teknik *direct* vSLAM. Sumber: https://vision.in.tum.de/research/vslam/lsdslam

Dari Gambar 2.2, terlihat bahwa teknik *feature-based* vSLAM menghasilkan poin-poin fitur yang lebih sedikit karena yang diambil hanya fitur-fitur tertentu saja. Untuk *direct* vSLAM poin-poin fitur lebih banyak, sebab keseluruhan informasi gambar diambil tanpa abstraksi.

Apabila odometri visual adalah *front-end* nya, optimisasi adalah *back-end* nya. Optimisasi dilakukan untuk menyelesaikan masalah dari interferensi *noise* yang terdapat dari sensor. Informasi dari kamera yang sudah melalui tahapan odometri visual dan informasi yang diperoleh dari tahapan penutupan *loop* (*loop shaping*) dioptimasi untuk menghasilkan trajektori dan peta global yang konsisten. Metode optimasi dapat dibagi menjadi dua kelas, yaitu metode berdasarkan filter (*filter-based*) seperti *Extended Kalman Filter* (EKF) dan metode berdasarkan optimisasi grafik (*graph-based*). Optimisasi *graph-based* dapat mengatasi kekurangan dari optimisasi *filter-based* seperti kesalahan pada linearisasi dan asumsi distribusi Gaussian dari *noise* [9].

Tahapan penutupan *loop* (*loop shaping*) mendeteksi apabila kamera melewati tempat yang sama dan mengambil data yang sama seperti sebelumnya. Tujuan dari *loop shaping* adalah untuk mengatasi penyimpangan yang terakumulasi pada trajektori kendaraan otonom dari waktu ke waktu, sehingga saat kendaraan otonom mengunjungi lagi tempat yang sama, ketidakpastian dan kesalahan SLAM dapat diatasi.

Selanjutnya, tahapan pemetaan adalah proses mengkonstruksi deskripsi lingkungan. Tipe-tipe dari hasil pemetaan dapat terbagi menjadi peta tersebar (*sparse map*) dan peta padat (*dense map*). Pada *sparse map,* beberapa tanda yang mencolok dari suatu lingkungan (*landmark*) dipilih dan dibentuk menjadi suatu peta [10]. Sedangkan, pada *dense map,* seluruh detail yang terlihat dideskripsikan [11]. Tipe peta *sparse map* dan *dense map* terlampir pada Gambar 2.3.



(a) (b)

**Gambar 2.3.** (a) Tipe peta *sparse map.*
Sumber: Oskiper, Taragay & Samarasekara, Supun & Kumar, Rakesh. (2017). CamSLAM: Vision aided inertial tracking and mapping framework for large scale AR applications;
(b) Tipe peta *dense map.*
Sumber: Yang, Shichao & Scherer, Sebastian. (2018). Monocular Object and Plane SLAM in Structured Environments.

Berbagai metode vSLAM telah dikembangkan dengan perbedaan-perbedaan dari segi metode odometri visual, tipe pemetaan, *loop closing* dan optimisasinya. Pada Tabel 2.1 terlampir beberapa jenis metode vSLAM yang telah dikembangkan beserta karakteristiknya.

**Tabel 2.1.** Metode vSLAM beserta karakteristiknya.

| SLAM | Metode Odometri Visual | Tipe Pemetaan | *Loop Closing* | Optimisasi |
|---|---|---|---|---|
| MonoSLAM | *Feature* | *Sparse* | Tidak | Filter |
| ORB-SLAM | *Feature* | *Sparse* | Ya | *Bundle adjustment (Graph)* |
| PTAM (*Parallel Tracking and Mapping)* | *Feature* | *Sparse* | Tidak | *Bundle adjustment (Graph)* |
| RTAB-Map (*Real-Time Appearance-Based Map*) | *Feature* | *Dense* | Ya | *Graph* |
| DTAM (*Dense Tracking and Mapping*) | *Direct* | *Dense* | Tidak | *Graph* |

| | | | | |
|---|---|---|---|---|
| LSD-SLAM (*Large-Scale Direct* SLAM) | *Direct* | *Semi-Dense* | Ya | *Pose graph (Graph)* |

MonoSLAM diusulkan oleh Davison et al. [12] sebagai metode vSLAM dengan optimisasi *filter based.* ORB-SLAM diusulkan dan diterapkan oleh Mur-Artal [13] dan terbukti dapat diaplikasikan di dalam maupun di luar ruangan. ORB-SLAM sendiri adalah perbaikan dari PTAM (*Parallel Tracking and Mapping)*. PTAM adalah metode yang diajukan pada tahun 2007 oleh Klein et al. [14]. Metode RTAB-Map RTAB-Map (*Real-Time Appearance-Based Map*) yang diajukan oleh Labbé et al. [15] memiliki deteksi penutupan *loop* dan menggunakan pendekatan manajemen memori.

DTAM (*Dense Tracking and Mapping*) diajukan oleh Newcombe et al. [16] sebagai metode vSLAM dengan odometri visual *direct* sepenuhnya yang menghasilkan estimasi gerak yang lebih halus dibanding dengan PTAM dengan penerangan yang konstan. LSD-SLAM (*Large-Scale Direct* SLAM) adalah metode yang diusulkan oleh Engel et al. [17] dengan metode odometri visual *direct* dan tipe peta *semi-dense*.

### *2.3. Depth Camera* untuk Implementasi vSLAM

*Depth camera* adalah salah satu sensor yang umum digunakan untuk vSLAM. *Depth camera* digunakan karena kemampuannya untuk menghasilkan citra dengan piksel yang memiliki nilai numerik yang terkait dengan "kedalaman" atau jarak dari kamera.

Terdapat beberapa tipe *depth camera*, di antaranya adalah *stereo depth camera* dan *time-of-flight* (ToF) *depth camera* [18]. Pada *stereo depth camera*, terdapat dua sensor kamera yang diletakkan saling berdekatan. Dua citra dari masing-masing dua sensor tersebut diambil dan dibandingkan. Karena kedua jarak sensor diketahui, maka perbandingan tersebut memberikan informasi kedalaman. Sedangkan, *time-of-flight* (ToF) *depth camera* mengukur perbedaan fasa antara sinyal IR (*infrared*) yang diemisikan dan direfleksikan. *Stereo depth camera* dan *time-of-flight* (ToF) *depth camera* dapat digunakan untuk mengaplikasikan vSLAM. Pada [19], Engel et al. membuat algoritma LSD-SLAM untuk *stereo depth camera* dan pada [20],

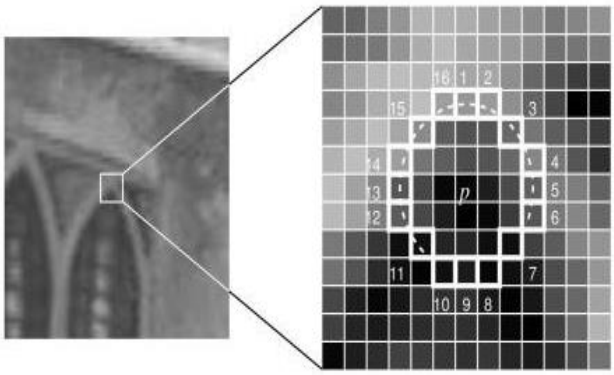
Paz et al. mengaplikasikan vSLAM dengan kamera stereo pada lingkungan terbuka dan tertutup yang luas. Salah satu penelitian vSLAM dengan ToF *depth camera* dilakukan oleh Hochdorfer et al. [21].

Selain itu, juga terdapat *depth camera* yang dapat bekerja dengan kamera RGB, yaitu kamera RGB-D. Pada kamera RGB-D, gambar RGB yang konvensional dapat ditambah dengan informasi kedalaman per piksel. Penelitian vSLAM dengan kamera RGB-D dilakukan oleh Kerl et al. [22] untuk membuat SLAM yang memiliki hasil pemetaan yang *dense* dengan kamera RGB-D dan Yang et al. [23] yang mencoba mengaplikasikan vSLAM dengan lebih dari satu RGB-D kamera.

## 2.4. ORB-SLAM2 (*Oriented FAST and Rotated BRIEF Simultaneous Localization and Mapping*)

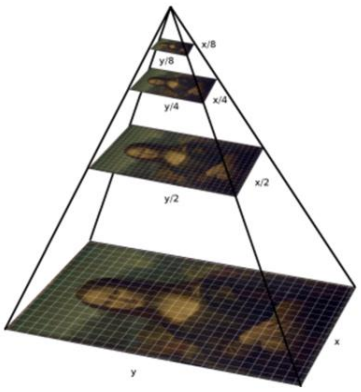
ORB-SLAM2 adalah sebuah *library* vSLAM yang dapat digunakan untuk kamera monokular, stereo, maupun kamera RGB-D yang menghitung trajektori kamera dan rekonstruksi peta *sparse* 3D. ORB [24] digunakan untuk mendeteksi *keypoint* yang penting dari *frame* yang ditangkap dan membuat deskripsi dari *frame* yang ditangkap. ORB terdiri dari detektor *keypoint* FAST (*Features from Accelerated and Segments Test*) dan deskriptor BRIEF (*Binary Robust Independent Elementary Feature*).

FAST adalah metode untuk menemukan *keypoint* atau piksel sudut. ORB mendeteksi *keypoint* pada gambar melalui intensitasnya. Apabila terdapat N piksel pada radius tertentu dari titik p yang jumlahnya lebih besar dari N atau jika terdapat N piksel pada radius tertentu dari titik p yang jumlahnya lebih besar dari N, maka kandidat titik p dianggap sebagai *keypoint* seperti pada ilustrasi Gambar 2.4.




**Gambar 2.4.** Visualisasi pendeteksian *keypoint* pada gambar.
Sumber: https://medium.com/data-breach/introduction-to-orb-oriented-fast-and-rotated-brief-4220e8ec40cf

Kemudian, ORB akan membuat *image pyramid* [25] yang merupakan kumpulan dari representasi resolusi berbeda dari satu gambar. Visualisasi dari *image pyramid* dapat dilihat pada Gambar 2.5.




**Gambar 2.5.** *Image Pyramid* yang diolah ORB.
Sumber: https://medium.com/data-breach/introduction-to-orb-oriented-fast-and-rotated-brief-4220e8ec40cf

Kemudian ORB akan mendeteksi *keypoint* dari masing-masing representasi. Setelah itu, ORB memberikan orientasi pada setiap *keypoint* bergantung pada perubahan intensitas disekitar *keypoint*. Orientasi ditentukan pada tiap potongan.

Sentroid intensitas digunakan untuk mendeteksi perubahan intensitas. Momen dari daerah sekitar *keypoint* dapat didefisinikan sebagai

$$m_{pq} = \sum_{x,y} x^p y^q I(\mathrm{x}, \mathrm{y}) \quad \textbf{(2.1)}$$

dengan $x$ dan $y$ sebagai koordinat dari piksel, p dan q adalah orde dari momen untuk piksel, dan $I(\mathrm{x},\mathrm{y})$ adalah intensitas. Kemudian, sentroid didapatkan dari

$$C = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}}\right) \quad \textbf{(2.2)}$$

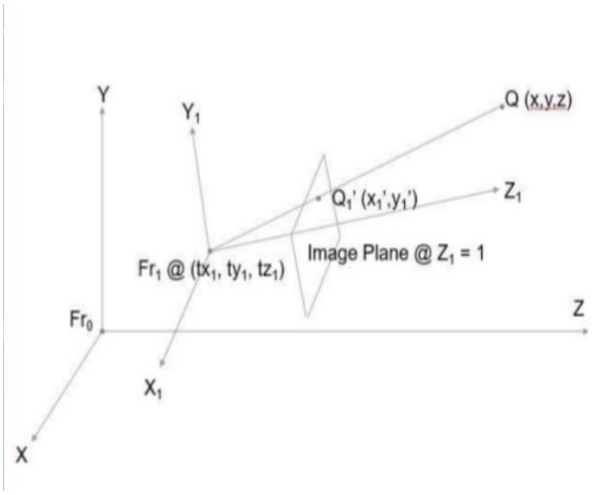
dengan $m_{00}$, $m_{10}$, $m_{01}$ sebagai momen. Kemudian, orientasi dapat ditentukan dengan

$$\theta = \mathrm{atan2}(m_{01,} m_{10}) \quad \textbf{(2.3)}$$

Selanjutnya, ORB akan melakukan BRIEF atau *binary robust independent elementary feature*. BRIEF digunakan untuk merepresentasikan objek dari *keypoint* yang telah didapatkan dengan mengkonversikan semua *keypoint* yang telah didapatkan ke dalam bentuk vektor biner. Metode BRIEF diawali dengan meminimumkan *noise* menggunakan *Gaussian kernel*. Setelah itu, BRIEF akan memilih satu pasang piksel di sekitar *keypoint*. Piksel pertama dipilih disekitar *keypoint* dengan standar deviasi tertentu. Sedangkan piksel kedua dipilih dari sekitar piksel pertama dengan standar deviasi sama dengan dua. Jika piksel pertama lebih terang dari piksel kedua maka nilainya satu, sebaliknya jika piksel pertama lebih gelap dari pada piksel kedua maka nilainya adalah dua. Kemudian dapat dinyatakan dengan matriks 2xn. Dengan n adalah berapa kali proses ini dilakukan pada lokasi ($x_n$,$y_n$).

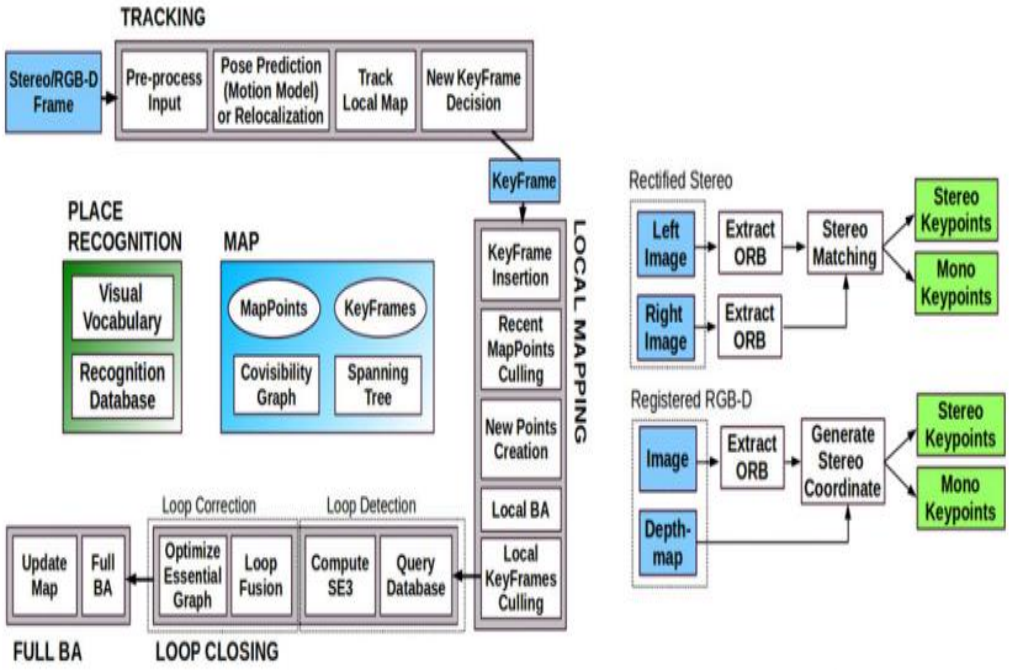
$$\mathrm{S} = \begin{pmatrix} x_1, & \dots, & x_n \\ y_1, & \dots, & y_n \end{pmatrix} \quad \textbf{(2.4)}$$

Pada ORB-SLAM2 proses optimisasi dilakukan dengan metode *bundle adjustment* [26]. *Bundle adjustment* (BA) adalah metode optimisasi yang umum digunakan untuk algoritma rekonstruksi 3D yang bersifat *feature based*. *Bundle adjustment* dapat menyempurnakan koordinat 3D yang mendeskripsikan geometri lingkungan, parameter gerakan relatif, dan karakteristik optik kamera yang digunakan untuk memperoleh citra menurut kriteria optimalitas yang melibatkan proyeksi citra yang sesuai dari semua titik. BA mengoptimisasi dengan memproyeksikan koordinat 3D dari peta (koordinat global) ke koordinat 2D gambar kamera. Kemudian dilakukan perbandingan hasil proyeksi dengan fitur yang terdeteksi. Koordinat global dan koordinat 2D dari gambar kamera ditunjukkan pada Gambar 2.6.




**Gambar 2.6.** Ilustrasi hubungan koordinat global 3D dan koordinat gambar 2D kamera.
Sumber: https://www.slideshare.net/embeddedvision/fundamentals-of-monocular-slam-a-presentation-from-cadence

ORB-SLAM2 terdiri dari tiga tahap [27] antara lain *tracking*, *local mapping*, dan *loop closing*. Tahapan dari ORB-SLAM2 dapat dilihat pada Gambar 2.7.




**Gambar 2.7.** Mekanisme metode ORB-SLAM2.
Sumber: Mur-Artal, Raul & Tardos, Juan. (2016). ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras.

Dari mekanisme metode ORB-SLAM2 tersebut, tahapan-tahapan ORB-SLAM2 akan dijelaskan lebih lanjut.

Pada proses *tracking*, masukan *frame* dari kamera diolah agar dihasilkan *keypoint* dan *keyframe* baru. Proses *tracking* diawali dengan proses ekstraksi ORB. Pada kamera stereo, proses FAST dan BRIEF untuk mengekstraksi ORB dilakukan pada hasil citra dari masing-masing kamera kiri dan kamera kanan. Kemudian, setiap ORB kiri dicari kecocokannya dengan citra kanan. Setelah itu didapatkan *keypoint* stereo yang didefinisikan dengan (uL, vL, $uR$), dimana (uL, vL) adalah koordinat dari citra kiri dan $uR$ adalah koordinat horizontal dari citra kanan.

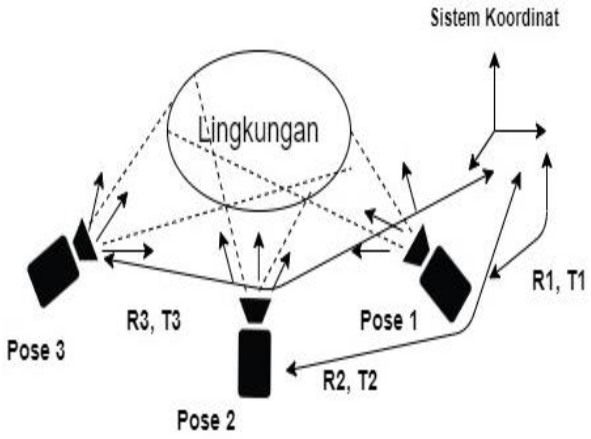
Untuk kamera RGB-D, fitur ORB diekstrak hanya pada citra RGB. Kemudian, untuk mendapatkan koordinat seperti stereo, setiap koordinat pada citra (uL, vL) dikonversikan kedalamannya (d) ke bentuk *virtual right coordinate* menggunakan persamaan berikut

$$uR = uL - \frac{f_x b}{d} \quad \textbf{(2.5)}$$

$f_x$ dan b bergantung pada jenis kamera yang digunakan. $f_x$ adalah panjang horizontal focal kamera sedangkan b adalah *baseline* diantara proyektor cahaya dan *infrared* dari kamera.

Apabila (uL, vL, $uR$) adalah *keypoint* stereo, (uL, vL) saja adalah *keypoint* monokular. *Keypoint* monokular adalah *keypoint* dimana kamera stereo kiri tidak dapat cocok dengan kamera stereo kanan dan *keypoint* apabila kamera RGB-D memiliki nilai kedalaman yang buruk.

Setelah itu, dilakukan estimasi pose kamera. Pose kamera adalah posisi dan orientasi kamera selama kamera melakukan translasi dan rotasi pada lingkungan seperti yang divisualisasikan pada Gambar 2.8. Apabila *tracking* dapat dilakukan dengan baik sebelumnya, digunakan model kecepatan yang konstan untuk memprediksi pose kamera dan mencari poin-poin dari *frame* sebelumnya. Jika terjadi *track loss*, digunakan *database* rekognisi DBoW2 untuk mencari kandidat *keyframe* untuk relokalisasi global. *Frame* dikonversi menjadi *bag of words* dan dicari kandidat *keyframe*-nya. *Pose* kamera dicari dengan iterasi RANSAC (*Random Sample Consensus*) dan algoritma PnP (*Perspective-n-Point*). Algoritma PnP adalah algoritma untuk mengestimasi pose dari kamera apabila terdapat beberapa set poin 3D dan projeksi 2D pada citra yang berhubungan.




**Gambar 2.8.** Visualisasi pergerakan kamera pada lingkungan

Setelah dilakukan estimasi pose kamera, dilakukan optimisasi pose kamera dengan *motion-only bundle adjustment* (*motion-only* BA). Pada *motion-only* BA, orientasi dan posisi kamera dioptimisasi untuk mengurangi kesalahan reprojeksi antara poin dalam koordinat asli dengan *keypoint.* Persamaan dari *motion-only* BA adalah

$$\{R,t\} = \underset{R,\,t}{argmin} \sum_{i \in x} \rho(\left\| x_{(.)}^{i} - \pi_{(.)}(RX^{i} + t) \right\|_{\Sigma}^{2}) \tag{2.6}$$

dimana $R$ adalah orientasi kamera, $t$ adalah posisi kamera, $\rho$ adalah *robust Huber cost function*, Σ adalah matriks kovarians yang berhubungan dengan skala dari *keypoint*, $X^{i}$ adalah poin 3D pada koordinat global, $x_{(.)}^{i}$ adalah koordinat *keypoint* dan $\pi_{(.)}$ adalah fungsi projeksi. Fungsi projeksi untuk kamera stereo adalah

$$\pi_s\left(\begin{bmatrix} X \\ Y \\ Z \end{bmatrix}\right) = \begin{bmatrix} f_x \frac{X}{Z} + c_x \\ f_y \frac{Y}{Z} + c_y \\ f_x \frac{X-b}{Z} + c_x \end{bmatrix} \tag{2.7}$$

dimana $(f_x, f_y)$ adalah panjang focal kamera dan $(c_x, c_y)$ adalah titik tengah proyeksi kamera. Sedangkan untuk persamaan fungsi huber, ditunjukkan pada persamaan 2.8.

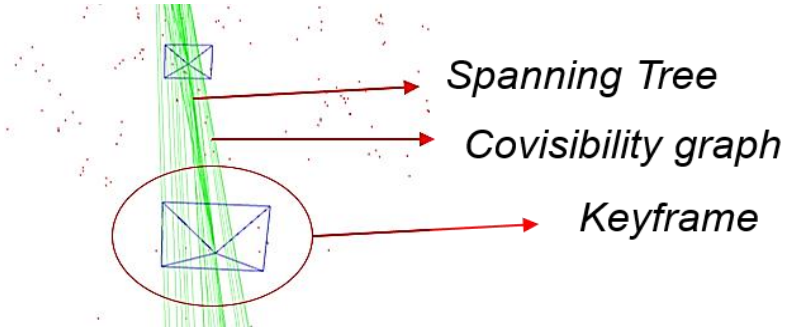
$$\rho\big(y_h, f(x_h)\big) = \begin{cases} \frac{1}{2}\big(y_h - f(x_h)\big)^2, & for\ |y_h - f(x_h)| \le \varepsilon, \\ \varepsilon|y_h - f(x_h)| - \frac{1}{2}\varepsilon^2 & otherwise \end{cases} \tag{2.8}$$

Dengan $y_h$ adalah nilai referensi, $f(x_h)$ adalah hasil estimasi, dan $\varepsilon$ adalah hiperparameter yang dipilih.

Setelah pose kamera dioptimisasi, peta dapat diproyeksikan pada *frame* dan dilakukan pencarian korespondensi terhadap peta yang sebelumnya sudah ada. Kemudian, dilakukan penentuan, apakah *frame* tersebut dapat dianggap sebagai *keyframe* baru atau tidak. Syarat dari *frame* agar layak menjadi *keyframe:*

a. Lebih dari 20 *frames* sudah dilalui dari pemasukan *keyframe* terakhir.
b. Terdeteksi paling sedikit 50 poin pada *frame*.
c. Terdapat kurang dari 90% poin yang sama dengan *keyframe* sebelumnya.

Kemudian, masuk ke proses *local mapping* seperti yang terdapat pada Gambar 2.7. Apabila *frame* pada tahap *tracking* layak dianggap sebagai *keyframe* baru, maka *keyframe* dimasukkan pada peta. Pada Gambar 2.9 terdapat visualisasi peta dari ORB-SLAM2.




**Gambar 2.9.** Visualisasi peta ORB-SLAM2.

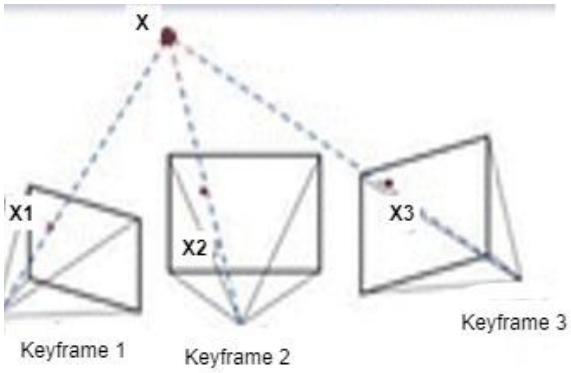
Kotak-kotak berwarna biru pada gambar adalah *keyframe*. Antar *keyframe* dihubungkan dengan grafik kovisibilitas dan *spanning tree.* Grafik kovisibilitas menghubungkan antar *keyframe-keyframe* yang dihasilkan. Apabila *keyframe* baru sedang dibuat, informasi dari *keyframe-keyframe* lama yang terhubung dengan grafik kovisibilitas digunakan untuk membantu pembuatan poin-poin baru. *Spanning tree* adalah garis yang menghubungkan antar *keyframe* yang paling banyak memiliki poin observasi yang sama. Pada tahap ini, representasi *bag of words* dari *keyframe* untuk *database* rekognisi juga dibuat.

Setelah itu, dilakukan pengurangan poin-poin yang tidak diperlukan pada *keyframe* baru agar poin-poin yang terdapat pada peta seluruhnya valid. Agar tidak dihilangkan, poin harus memenuhi kondisi:

a. Poin ditemukan di lebih dari 25% *frame*.

b. Poin tersebut sudah ditemukan paling sedikit pada 3 *keyframes* yang lain.

Kemudian, poin-poin pada peta dibuat dengan triangulasi ORB dengan *keyframe-keyframe* yang terkoneksi dengan *covisibility graph*. Pada Gambar 2.10 terdapat visualisasi proses triangulasi ORB untuk membentuk poin peta ORB-SLAM2.




**Gambar 2.10.** Visualisasi proses triangulasi ORB untuk pembentukan poin peta.

Setelah itu, dilakukan optimisasi *local bundle adjustment* (*local BA*). *Local* BA mengoptimasi *keyframe* dan seluruh poin $P_L$ terdapat pada *keyframe* $K_L$ yang sedang terdeteksi/terlihat. Persamaan dari *local* BA adalah

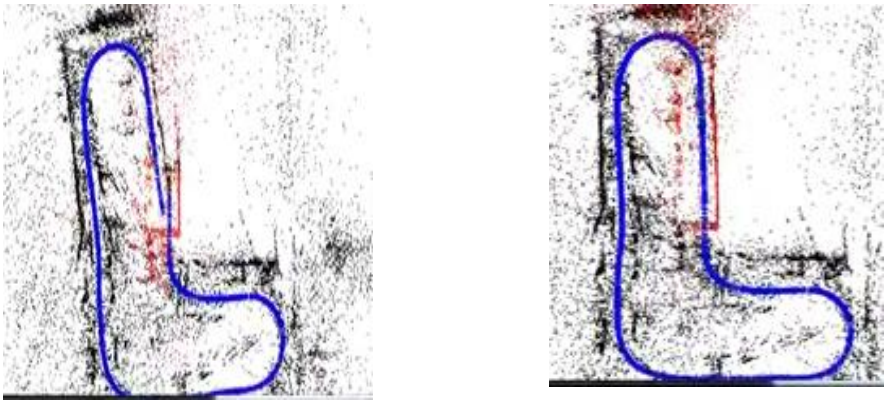
$$\{X^i, R_l, t_l | i \in P_L, l \in K_L\} = \underset{X^i, R_l, t_l}{argmin} \sum_{k \in K_L \cup K_F} \sum_{j \in x_k} \rho\left(E_{kj}\right) \quad \textbf{(2.9)}$$

$$E_{kj} = \left\| x_{(.)}^j - \pi_{(.)}\left(R_k X^j + t_k\right) \right\|_{\Sigma}^2 \quad \textbf{(2.10)}$$

dimana $K_F$ adalah *keyframe* lainnya (bukan $K_L$), dan $x_k$ adalah set kecocokan antara poin $P_L$ dan poin pada *keyframe k.*

Setelah proses optimisasi, dilakukan pengurangan *keyframe* yang dianggap redundan, dimana 90% dari poin-poinnya sudah ada pada paling sedikit tiga *keyframe.*

Apabila ORB-SLAM2 mendeteksi *keyframe* yang pernah dilewati sebelumnya, dilakukan *loop closing* (penutupan *loop*). Visualisasi dari *loop closing* terdapat pada Gambar 2.11.



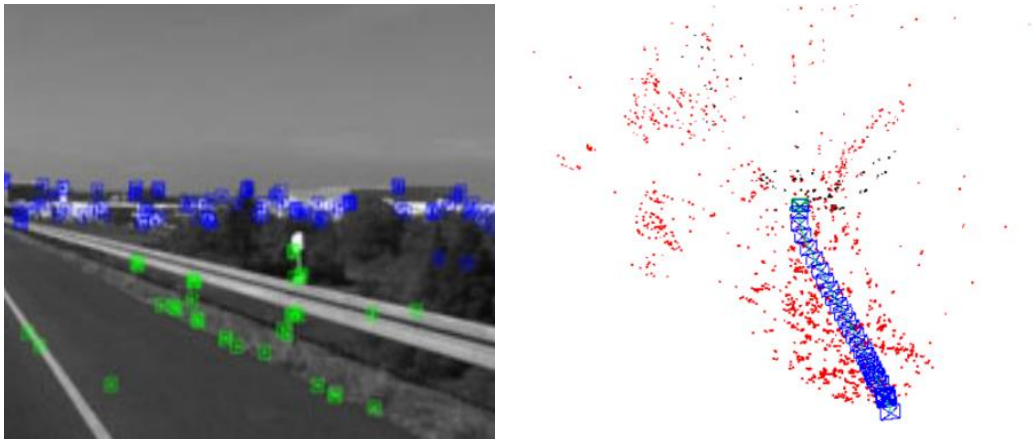
(a) (b)

**Gambar 2.11.** (a) sebelum *loop closing;* (b) setelah *loop closing.*
Sumber: https://youtu.be/OfrPAUyp1_s

*Loop closing* terdiri dari dua tahap [28] yaitu pertama, pengenalan lokasi untuk mengetahui apakah lokasi tersebut pernah dilewati sebelumnya dan kedua dilakukan koreksi terhadap *loop* tersebut. Pendeteksian dilakukan dengan *database* rekognisi. Persamaan antara *bag of words keyframe* dengan *keyframe* di sekitarnya dicari sebelum akhirnya dicari *keyframe* kandidat *loop* pada *database* rekognisi. Setelah kandidat *loop* ditemukan, kecocokan antara ORB pada poin *keyframe* saat ini dan *keyframe* kandidat *loop* dicari. Dilakukan iterasi RANSAC untuk mencari *similarity transformation* dengan metode Horn. Jika dianggap memiliki banyak kesamaan, *loop* dapat ditutup.

Apabila dianggap sebagai penutup *loop*, dilakukan penggabungan poin peta, *covisibility graph,* dan *keyframe*. *Pose graph optimization* dilakukan untuk mengurangi *error* akibat *loop closing* secara efektif. Apabila tahap pendeteksian *loop* dan koreksi *loop* dilakukan secara bertahap, maka akan berat untuk komputasi pada perangkat keras. Untuk mengatasi hal ini, kedua tahap tersebut dilakukan pada platform yang berbeda sehingga memungkinkan sistem untuk mendeteksi *loop* dan membentuk peta secara bersamaan. Setelah dilakukan *loop* closing, *keyframes* dan poin-poin peta kembali dioptimisasi. Optimisasi dilakukan dengan *full bundle adjustment* (*full* BA). *Full* BA adalah kasus spesifik dari *local* BA, dimana seluruh *keyframe* dan poin pada peta dioptimisasi, kecuali di *keyframe* awal yang dianggap tetap.

Pada Gambar 2.12 diperlihatkan visualisasi *keypoint* saat *tracking* dan peta yang dihasilkan oleh ORB-SLAM2.



(a) (b)

**Gambar 2.12.** (a) Visualisasi *keypoint* pada ORB-SLAM2; (b) Visualisasi peta ORB-SLAM2.
Sumber: Mur-Artal, Raul & Montiel, J. & Tardos, Juan. (2015). ORB-SLAM: a versatile and accurate monocular SLAM system.

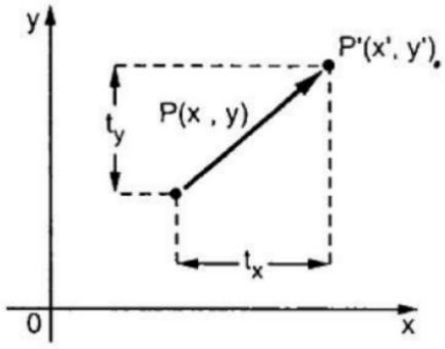
Pada Gambar 2.12.a terdapat dua jenis warna *keypoint*, yaitu poin hijau dan biru. Poin yang berwarna hijau memiliki kedalaman kurang dari 40 kali *baseline* kamera, sedangkan poin yang berwarna biru menandakan poin yang sangat jauh. Selain itu, poin biru juga menandakan bahwa poin tersebut dapat digunakan untuk mengestimasi orientasi namun memiliki informasi untuk translasi dan *scaling* yang lemah.

Pada Gambar 2.12.b, terdapat visualisasi hasil pemetaan ORB-SLAM2. Terdapat dua jenis poin yang berbeda warna, yaitu merah dan hitam. Poin berwarna merah adalah poin terbaru (*local map points*) sedangkan poin hitam adalah poin lama.

## 2.5. Transformasi Koordinat

Transformasi adalah fungsi yang memetakan suatu vektor berdimensi *n* ke vektor berdimensi *n* lain. Transformasi koordinat yang dapat dilakukan adalah translasi, pengubahan skala (*scaling*) uniform, pengubahan skala (*scaling*) non-uniform, dan rotasi [29]. Transformasi koordinat dapat digunakan untuk transformasi koordinat SLAM ke koordinat global karena sistem koordinatnya berbeda.

Translasi adalah proses pergeseran sistem koordinat dengan menambahkan koordinat translasi $(t_x, t_y)$ pada koordinat awal untuk mendapatkan koordinat yang baru. Translasi adalah proses transformasi yang paling sederhana. Translasi dapat digunakan untuk menggeser titik awal koordinat suatu sistem, contohnya menggeser titik awal koordinat SLAM agar sesuai dengan koordinat global. Pada Gambar 2.13 terdapat visualisasi translasi suatu sistem koordinat.

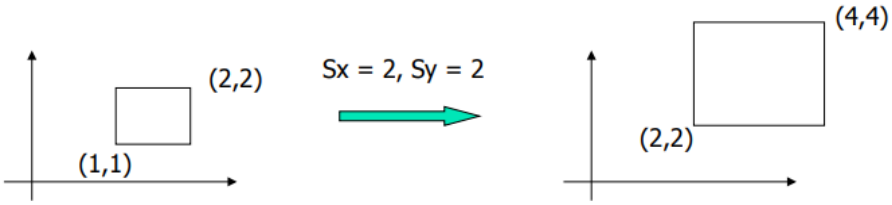



**Gambar 2.13.** Visualisasi translasi suatu sistem koordinat.
Sumber: https://www.hebergementwebs.com/infographic-tutorial/2d-transformation

Untuk menyesuaikan satuan pada koordinat SLAM agar dalam satuan meter (m) dapat dilakukan pengubahan skala koordinat. Matriks pengubahan skala dari suatu sistem koordinat dapat dideskripsikan dengan persamaan berikut

$$\mathrm{S}(sx, sy) = \begin{bmatrix} sx & 0 \\ 0 & sy \end{bmatrix} \quad \textbf{(2.11)}$$

dimana $sx$ adalah skala pengubahan pada arah X dan $sy$ adalah skala pengubahan pada arah Y. Jika $sx = sy$ , maka proses pengubahan skala disebut sebagai pengubahan skala uniform. Sebaliknya, apabila $sx \neq sy$, maka proses pengubahan skala disebut sebagai pengubahan skala non-uniform. Pada Gambar 2.14 terdapat visualisasi pengubahan skala dari suatu sistem koordinat.

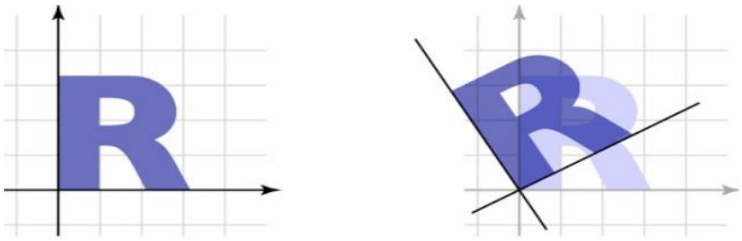



**Gambar 2.14.** Visualisasi pengubahan skala suatu sistem koordinat.
Sumber: https://web.cse.ohio-state.edu/~shen.94/681/Site/Slides_files/transformation_review.pdf

Untuk memutar koordinat SLAM agar sesuai dengan koordinat global dapat dilakukan rotasi. Rotasi adalah proses pemutaran sistem koordinat terhadap titik awal sebesar suatu sudut Θ. Persamaan dari matriks rotasi adalah

$$\mathrm{R}(\Theta) = \begin{bmatrix} cos\,\Theta & -sin\,\Theta \\ sin\,\Theta & cos\,\Theta \end{bmatrix} \quad \textbf{(2.12)}$$

Ruang rotasi 2D disebut dengan *SO*(2) dan ruang rotasi 3D disebut juga dengan *SO*(3). Pada Gambar 2.15 terdapat visualisasi dari rotasi suatu sistem koordinat dengan origin (0,0) sebagai pusat rotasinya.



**Gambar 2.15.** Visualisasi rotasi sebuah sistem koordinat.
Sumber: C. Cs and S. Marschner. (2004). 2D Geometric Transformations.

### 2.6. Sistem Kontrol pada Kendaraan Otonom

Pada kendaraan otonom, sistem kontrol digunakan untuk mengontrol pergerakan dari kendaraan otonom, seperti posisi, kecepatan, dan orientasi. Agar pergerakan dapat dikontrol, diperlukan informasi posisi, kecepatan, dan orientasi secara *real time* yang dapat diperoleh dengan SLAM. Kemudian, pengontrol akan berusaha untuk menyesuaikan posisi, kecepatan, orientasi kendaraan agar sesuai dengan yang diinginkan.

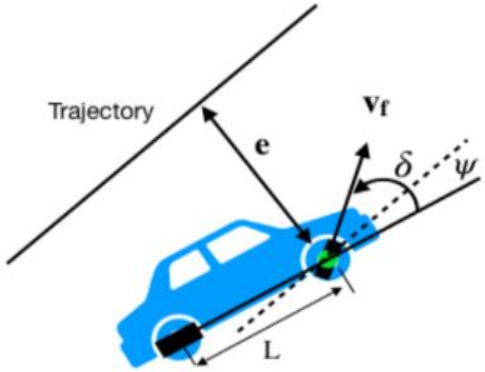
Sistem kontrol pada kendaraan otonom dapat dibagi menjadi dua, yaitu sistem kontrol longitudinal dan sistem kontrol lateral. Sistem kontrol longitudinal adalah

sistem kontrol yang mengatur kecepatan pada kendaraan menggunakan gas dan rem agar kendaraan tetap berada di kecepatan yang diinginkan. Sistem kontrol lateral adalah sistem kontrol yang mengatur agar kendaraan otonom dapat mengikuti jalur yang sesuai. Sistem kontrol lateral mendefinisikan kesalahan posisi kendaraan relatif terhadap jalur yang diinginkan, kemudian nilai kesalahan tersebut diminimasi.

Terdapat dua tipe dari desain kontrol yang dapat digunakan pada kendaraan otonom [30], yaitu pengontrol geometrik dan pengontrol dinamik. Pengontrol geometrik adalah pengontrol yang mengaplikasikan teknik geomterik diferensial pada sistem kontrol. Contoh dari pengontrol geometrik adalah Pure Pursuit dan Pengontrol Stanley. Pengontrol dinamik adalah metode yang menggunakan prediksi model untuk merancang hasil kontrol yang baik untuk sistem yang bervariasi terhadap waktu. Contoh dari pengontrol dinamik adalah *Model Predictive Control* (MPC), *sliding mode*, dan *feedback linearization*.

## 2.7. Pengontrol Stanley

Pengontrol Stanley adalah salah satu jenis pengontrol lateral pada kendaraan otonom [31]. Pengontrol Stanley dikembangkan oleh tim *Grand Challenge Defense Advanced Research Projects Agency* (DARPA) dari Stanford University. Pengontrol Stanley menggunakan poros roda depan sebagai poin referensinya. *Heading error* dan *cross-track error* diperhatikan. *Cross-track error* adalah jarak antara titik terdekat pada jalur dengan poros roda depan kendaraan. Pada Gambar 2.16, terdapat hubungan parameter-parameter dari Pengontrol Stanley




**Gambar 2.16.** Hubungan parameter-parameter dari Pengontrol Stanley.
Sumber: https://dingyan89.medium.com/three-methods-of-vehicle-lateral-control-pure-pursuit-stanley-and-mpc-db8cc1d32081

Dari Gambar 2.16, $\psi(t)$ adalah sudut antara trajektori tujuan dan tujuan kendaraan. Kemudian, $\delta$ adalah sudut kemudi.

Terdapat tiga hukum kontrol dari Pengontrol Stanley [32]. Untuk meminimasi *heading error* relatif terhadap jalur, sudut kemudi diatur sama dengan sudut antara trajektori tujuan dan tujuan kendaraan seperti pada persamaan 2.13.

$$\delta(t) = \psi(t) \tag{2.13}$$

Kemudian, untuk meminimasi *cross-track error*, ditambahkan kontrol proporsional yang penguatannya diskalakan dengan *inverse* (kebalikan) dari kecepatan maju. Kemudian kontrol dilewatkan melalui fungsi *inverse* tan yang memetakan sinyal kontrol proporsional ke kisaran sudut $-\pi$ ke $\pi$. Persamaan untuk mengoreksi sudut kemudi terdapat pada persamaan 2.14.

$$\delta(t) = \tan^{-1}\left(\frac{ke(t)}{v_f(t)}\right) \tag{2.14}$$

Dan terakhir, sudut kemudi dijaga agar tetap dalam sudut kemudi minimum ($\delta$min) dan sudut kemudi maksimum ($\delta$max) yang biasanya simetris di sekitar 0.

Dari ketiga hukum ini, didapatkan persamaan sudut kemudi dari kendaraan pada persamaan 2.15.

$$\delta(t) = \psi(t) + \tan^{-1}\left(\frac{ke(t)}{v_f(t)}\right), \delta(t) \in [\delta\min, \delta\max] \tag{2.15}$$

Untuk *cross-track error* yang bernilai cukup besar, berlaku

$$\tan^{-1}\left(\frac{ke(t)}{v_f(t)}\right) \approx \frac{\pi}{2} \quad \textbf{(2.16)}$$

Hal ini karena pada kasus *cross-track error* yang besar, nilai $ke(t)/v_f(t)$ akan semakin besar sehingga mendekati $\pi/2$. Nilai besar ini akan memaksimalkan pengemudian dan kendaraan akan berbelok pada jalur. Efek dari hal ini adalah untuk meningkatkan *heading error* pada arah yang berlawanan, sehingga kemudi akan bernilai 0 apabila *heading error* mencapai $-\pi/2$. Kemudian kendaraan kembali berjalan pada jalur hingga *cross-track error* berkurang. *Heading* kembali mengoreksi kesejajaran kendaraan dengan jalur dan kendaraan kembali melakukan *tracking* pada jalur secara lebih dekat.

## 2.8. *Robot Operating System* (ROS)

Untuk dapat mengimplementasikan SLAM secara *real time* pada sistem otonom, *Robot Operating System* (ROS) diperlukan. ROS adalah *meta-operating system* untuk robot yang bersifat *open-source* [33]. ROS menyediakan layanan-layanan seperti abstraksi sistem perangkat keras, kontrol perangkat tingkat rendah, penyampaian *message* (pesan) antar proses, dan manajemen *package.* Alat dan *library* untuk membangun, menulis, dan menjalankan kode di lebih dari satu komputer juga tersedia. ROS bersifat *real time* dan dapat menghubungkan berbagai sistem yang diperlukan untuk mengaplikasikan SLAM pada kendaraan otonom, seperti untuk menerima masukan sensor, menjalankan algoritma, dan menghubungkan hasil estimasi SLAM ke pengontrol.

ROS menunjang berbagai bahasa pemrograman, seperti C++, Python, Java, dan lain-lain. Walaupun bahasa pemrograman yang digunakan berbeda-beda, ROS memungkinkan setiap program dapat berkoordinasi satu sama lain. Dalam pemrosesan data, ROS menyediakan fitur sebagai berikut [34].

- *Node*

  *Node* adalah proses yang melakukan komputasi pada setiap program. Setiap *node* dapat melakukan komunikasi apabila masing-masing *node* terhubung dengan *master*.
- *Master*

ROS *master* menyediakan registrasi nama dan pencarian pada seluruh grafik komputasi. ROS *master* memungkinkan *node* untuk saling menemukan satu sama lain, bertukar pesan (*messages*), atau memanggil layanan (*services*).

- *Messages*

  Dalam berkomunikasi dengan satu sama lain, *nodes* menyampaikan *message*. *Message* adalah struktur data yang disampaikan. Data dapat disampaikan dalam tipe *integer, float, booelan,* dan lain-lain. Tipe *message* untuk komunikasi data sudah cukup banyak.

- *Topics*

  *Node* mengirimkan *message* dengan menerbitkan (*publish*) *message* tersebut ke dalam sebuah *topic. Topic* adalah nama yang digunakan untuk mengidentifikasi konten dari *message. Node* yang tertarik atau membutuhkan sebuah data tertentu harus berlangganan (*subscribe*) dengan *topic* tertentu.

- *Bags*

  *Bags* adalah format untuk penyimpanan dan pemutaran ulang data ROS *message. Bags* adalah mekanisme yang cukup penting dalam penyimpanan data, seperti data sensor yang sulit untuk diambil namun dibutuhkan dalam pengembangan dan percobaan algoritma.

## 2.9. Metriks Evaluasi

Untuk mengetahui performansi SLAM, SLAM biasa dibandingkan dengan *ground truth* seperti GNSS. Metriks evaluasi seperti *Root Mean Square Error* (RMSE) dan *Mean Absolute Error* (MAE) dapat digunakan untuk mengevaluasi SLAM.

### 2.9.1. *Root Mean Square Error* (RMSE)

*Root Mean Square Error* (RMSE) adalah standar deviasi dari residual (error prediksi). Residual adalah ukuran dari seberapa jauh titik data dari garis regresi. RMSE adalah ukuran seberapa menyebarnya residual tersebut. Formula dari RMSE dapat dinyatakan dalam persamaan

$$\mathrm{RMSE} = \sqrt{\frac{\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}{n}} \qquad \textbf{(2.17)}$$

Dimana $n$ adalah banyak data, $y_j$ adalah nilai obeservasi ke-j, dan $\hat{y}_j$ adalah nilai prediksi ke-j

### 2.9.2. *Mean Absolute Error (MAE)*

*Mean Absolute Error* (MAE) adalah metrik yang mengukur rata-rata kesalahan dari prediksi tanpa memperhatikan arah dari kesalahan tersebut (menuju positif atau negatif). MAE adalah perbedaan absolut dari nilai prediksi dan aktual dimana seluruh nilai memiliki bobot yang sama. Formula dari MAE dapat dinyatakan dalam persamaan.

$$\text{MAE} = \frac{1}{n}\sum_{j=1}^{n}\left|y_j - \hat{y}_j\right| \quad \textbf{(2.18)}$$

Dimana $n$ adalah banyak data, $y_j$ adalah nilai obeservasi ke-j, dan $\hat{y}_j$ adalah nilai prediksi ke-j.