

Tugas 7 Sistem Robot Otonom: Path Tracking Wheeled Mobile Robot P3DX Pada Coppeliasim

Nama: Falah Amru Dikasmara
NRP: 5022211041
Mata Kuliah: Sistem Robot Otonom

1 Pendahuluan

1.1 Latar Belakang

Dalam sistem robotika bergerak, salah satu konsep kunci yang selalu muncul adalah **lintasan (path)** yang harus diikuti robot. Secara umum, *path* dapat dipahami sebagai sekumpulan titik atau kurva di ruang konfigurasi (seringnya di bidang $x-y$) yang merepresentasikan *jalur ideal* yang diinginkan perancang sistem. Path ini dapat berupa garis lurus sederhana, kurva halus, maupun lintasan kompleks yang menghindari rintangan dan menghubungkan berbagai titik tujuan.

Fungsi path pada robotika sangat beragam, antara lain:

- sebagai **referensi geometris** yang menentukan di mana robot seharusnya berada pada setiap fase pergerakannya,
- sebagai **panduan navigasi** agar robot dapat berpindah dari titik awal ke titik tujuan tanpa menyimpang terlalu jauh dari jalur yang aman,
- sebagai dasar bagi **perencanaan gerak** yang mempertimbangkan keterbatasan kinematika dan dinamika robot.

Setelah path didefinisikan, masalah berikutnya adalah membuat robot *benar-benar* bergerak mengikuti path tersebut. Tugas ini dikenal sebagai **path tracking** atau **path following**. Secara konseptual, *path tracking* adalah proses mengatur gerak robot (posisi dan orientasi) agar tetap berada dekat dengan path yang diinginkan, tanpa mensyaratkan robot harus berada pada titik tertentu dari path pada waktu tertentu (berbeda dengan *trajectory tracking* yang biasanya mencakup dimensi waktu eksplisit). Fungsi path tracking pada robotika antara lain:

- menjaga robot agar selalu berada di sekitar jalur aman yang telah direncanakan,
- memastikan pergerakan robot halus dan terkontrol, meskipun terdapat gangguan atau ketidakpastian model,
- menjadi blok dasar bagi aplikasi yang lebih kompleks seperti navigasi otonom, *docking*, dan *coverage*.

Berbagai strategi path tracking telah dikembangkan dalam literatur robotika, mulai dari pendekatan *geometris* murni (misalnya *Pure Pursuit*, *Stanley Controller*) hingga pendekatan berbasis kontrol optimal atau kontrol nonlinier. Secara umum, sebagian besar metode path tracking melibatkan langkah-langkah berikut:

1. mendefinisikan **titik referensi** di sepanjang path yang harus “dikejar” robot (misalnya titik terdekat pada path, atau titik di depan robot sejauh *look-ahead distance*),
2. menghitung **galat posisi dan orientasi** robot terhadap titik referensi tersebut di dalam suatu *frame* yang sesuai (seringnya *body frame* robot),
3. mengubah galat-galat tersebut menjadi perintah kecepatan linear dan sudut yang konsisten dengan kinematika robot (misalnya kinematika *differential-drive*).

Pada tugas ini, path telah disediakan langsung di lingkungan simulasi **CoppeliaSim** dalam bentuk objek /Path. Robot yang digunakan adalah *wheeled mobile robot* Pioneer P3DX yang dimodelkan sebagai robot *differential-drive* bergerak di bidang. Path ini berisi serangkaian titik (x_i, y_i) di *world frame* yang membentuk jalur yang harus diikuti oleh robot. Di sisi lain, robot memiliki pose (x_b, y_b, yaw_b) terhadap *world frame* yang dapat dibaca secara langsung melalui *ZMQ Remote API*.

Strategi path tracking yang diimplementasikan pada simulasi ini dapat diringkas sebagai berikut:

- menentukan **titik look-ahead** di depan robot sejauh jarak tertentu (*look-ahead distance*),

- mencari **titik path yang paling dekat** dengan titik look-ahead tersebut (aproksimasi projeksi tegak lurus path),
- memetakan titik path referensi ini ke *frame robot* menggunakan matriks transformasi homogen 4×4 ,
- mendefinisikan sinyal galat pose (misalnya galat jarak sepanjang sumbu depan robot dan galat heading) di dalam *frame robot*,
- merancang loop kontrol yang mengubah galat-galat ini menjadi kecepatan linear v dan kecepatan sudut ω , lalu ke kecepatan roda kanan–kiri.

Dalam konfigurasi simulasi yang digunakan, path dari objek */Path* direpresentasikan sebagai sekumpulan titik dalam *world frame*, sedangkan pose robot diukur sebagai posisi dan orientasi terhadap *world frame*. Program Python kemudian:

- membaca data path dan pose robot,
- menyusun transformasi koordinat antara *world frame* dan *body frame*,
- menghitung galat path tracking di *frame robot*,
- dan mengirimkan perintah kecepatan ke roda P3DX melalui `setJointTargetVelocity`.

Dengan demikian, simulasi ini mengilustrasikan bagaimana **kONSEP PATH**, **TRANSFORMASI KOORDINAT**, dan **SINYAL GALAT PATH TRACKING** dapat dikombinasikan untuk menghasilkan perilaku robot yang mengikuti lintasan yang telah ditentukan.

1.2 Tujuan

Berdasarkan latar belakang tersebut, tujuan dari eksperimen ini adalah:

1. Menjelaskan dan mempraktikkan konsep **path** dan **path tracking** pada robot *differential-drive* (P3DX) dalam lingkungan simulasi CoppeliaSim.
2. Mengimplementasikan **transformasi koordinat** berbasis matriks homogen 4×4 untuk mengubah titik referensi path dari *world frame* ke *frame robot*:

$${}^b p_{\text{ref}} = {}^b T_w {}^w p_{\text{ref}}.$$

3. Mendefinisikan beberapa **sinyal galat path** dalam *frame robot*, seperti:
 - galat jarak sepanjang sumbu depan robot (e_d),
 - galat heading ke titik path referensi (e_h),
 - (opsional) galat orientasi akhir terhadap orientasi path atau objek pendukung (e_o).
4. Merancang **loop kontrol path tracking** yang mengubah galat-galat tersebut menjadi perintah kecepatan linear v dan kecepatan sudut ω , kemudian ke kecepatan roda kanan–kiri, sehingga robot mampu mengikuti path yang telah ditentukan.
5. Menyediakan **visualisasi dan data eksperimen** (lintasan dan galat terhadap waktu) sebagai sarana analisis dan pembelajaran mengenai perilaku sistem kontrol path tracking.

1.3 Fitur Proyek

Untuk mencapai tujuan di atas, program simulasi yang dibuat memiliki serangkaian fitur dan tugas utama sebagai berikut:

1. Inisialisasi dan akuisisi data dari simulator

- Membuka koneksi ke CoppeliaSim melalui *ZMQ Remote API*.
- Mengambil *handle* robot P3DX, roda kanan–kiri, objek */Path*, serta beberapa *marker* bantu (misalnya indikator titik look-ahead dan titik path yang sedang dikejar).
- Memulai simulasi dan mengatur mode eksekusi (kontinu/*stepping*).
- Membaca data path dari objek */Path* dalam bentuk tabel titik (x_i, y_i) di *world frame*.
- Secara berkala membaca:

- posisi dan orientasi robot (${}^w p_b$, yaw_b),
- (opsional) pose objek lain seperti `/Disc` yang dapat digunakan sebagai referensi orientasi.

2. Transformasi koordinat dan perhitungan galat path

- Menyusun matriks transformasi ${}^w T_b$ dari posisi dan yaw robot (roll dan pitch diabaikan untuk gerak planar).
- Menghitung invers ${}^b T_w = ({}^w T_b)^{-1}$ dan memetakan titik path referensi ke *frame robot*: ${}^b p_{\text{ref}} = {}^b T_w {}^w p_{\text{ref}}$.
- Menghitung sinyal galat path:
 - $e_d = x_b$: komponen posisi referensi sepanjang sumbu depan robot,
 - $e_h = \arctan 2(y_b, x_b)$: galat heading (arah garis pandang ke titik path referensi),
 - (opsional) e_o : galat orientasi antara yaw robot dan orientasi referensi (misalnya orientasi objek atau arah path lokal).
- Menghitung jarak Euclidean antara robot dan path sebagai ukuran kedekatan terhadap lintasan.

3. Pemilihan titik referensi path dan strategi look-ahead

- Menentukan **titik look-ahead** di depan robot sejauh jarak tertentu (*look-ahead distance*) di *world frame*.
- Menghitung jarak semua titik path terhadap titik look-ahead dan memilih **titik path terdekat** sebagai *titik referensi* yang harus dikejar.
- Menampilkan titik look-ahead dan titik referensi path di CoppeliaSim untuk mempermudah visualisasi proses path tracking.

4. Perancangan kontrol dan aktuasi roda

- Menghitung kecepatan linear:

$$v = k_{p,\text{lin}} e_d$$

- Menghitung kecepatan sudut:

$$\omega = k_{p,\text{ang}} e_h + k_{p,\text{ori}} e_o$$

dengan $k_{p,\text{ang}}$ dan $k_{p,\text{ori}}$ yang dapat bergantung pada jarak robot ke path (misalnya melalui fungsi eksponensial mode = $\exp(-d/d_{\text{sw}})$).

- Mengonversi (v, ω) ke kecepatan linear roda kanan–kiri (v_R, v_L) menggunakan kinematika *differential-drive*.
- Mengubah (v_R, v_L) menjadi kecepatan sudut roda (ω_R, ω_L) berdasarkan jari-jari roda, kemudian mengirimkan nilai tersebut ke CoppeliaSim melalui `setJointTargetVelocity`.

5. Pencatatan data dan visualisasi

- Menyimpan riwayat:

- lintasan robot $(x_w(t), y_w(t), \text{yaw}(t))$,
- posisi titik path referensi dalam *frame robot* ${}^b p_{\text{ref}}(t)$,
- sinyal galat $e_d(t)$, $e_h(t)$, dan (opsional) $e_o(t)$,
- waktu simulasi t .

- Menampilkan:

- lintasan robot pada bidang $x_w - y_w$ dengan penanda titik awal dan akhir,
- grafik $e_d(t)$ untuk melihat reduksi jarak ke path,
- grafik $e_h(t)$ (dalam derajat) untuk melihat proses penyelarasannya terhadap path.

1.4 Software yang Dibutuhkan dan Fungsinya

Beberapa perangkat lunak yang digunakan dalam proyek ini adalah:

- Python:** Bahasa pemrograman utama untuk mengimplementasikan loop kontrol, transformasi koordinat, pengolahan data, dan visualisasi.
- CoppeliaSim Edu:** Lingkungan simulasi robotik yang menyediakan model Pioneer P3DX, objek `/Path`, dinamika, serta *ground-truth* pose setiap objek terhadap *world frame*.

3. **ZMQ Remote API (Python client)**: Antarmuka komunikasi antara Python dan CoppeliaSim untuk:
 - membaca pose robot dan data path,
 - mengirim perintah kecepatan joint,
 - mengontrol start/stop simulasi.
4. **NumPy**: Pustaka komputasi numerik yang digunakan untuk operasi vektor/matriks, termasuk pembentukan matriks rotasi dan matriks transformasi homogen 4×4 , serta invers matriks.
5. **Matplotlib**: Pustaka visualisasi untuk menggambar lintasan, kurva galat terhadap waktu, dan menyimpan hasil plot sebagai berkas gambar (misalnya format `.svg`).

1.5 Tantangan

Dalam perancangan dan implementasi eksperimen path tracking ini, beberapa tantangan yang muncul antara lain:

1. Menyusun **alur program** yang terstruktur mulai dari inisialisasi koneksi, pembacaan data path dan pose, perhitungan transformasi dan galat, penerapan kontrol, hingga penghentian simulasi dan visualisasi hasil.
2. Mengimplementasikan **transformasi koordinat** secara tepat, termasuk penyusunan dan invers matriks transformasi homogen, sehingga pemetaan ${}^w p_{\text{ref}} \mapsto {}^b p_{\text{ref}}$ konsisten dengan konvensi frame yang digunakan.
3. Mendefinisikan **sinyal galat path** yang intuitif dan informatif untuk keperluan kontrol (memisahkan kontribusi jarak sepanjang sumbu depan, heading ke path, dan orientasi akhir jika diperlukan).
4. Merancang **mekanisme pemilihan titik referensi** (misalnya berbasis look-ahead) yang stabil dan tidak menyebabkan respons robot berosilasi atau menyimpang jauh dari path.
5. Mengimplementasikan **mode switching** yang halus antara fokus kontrol posisi dan orientasi (jika digunakan), agar robot tetap stabil di sekitar path.
6. Menangani **normalisasi sudut** (*wrap-to-π*) untuk mencegah diskontinuitas sudut yang dapat mengganggu perhitungan galat dan aksi kontrol.
7. Melakukan **tuning parameter kontrol** (gain linear dan angular) untuk mendapatkan kompromi yang baik antara kecepatan respon, kehalusan gerak, dan kestabilan sistem.

1.6 Kemampuan yang Dipelajari

Melalui pengerjaan proyek ini, beberapa kemampuan yang diharapkan dapat dikuasai antara lain:

1. Memahami konsep **path** dan **path tracking** pada robot bergerak planar serta hubungan antar-frame (world, body).
2. Menggunakan CoppeliaSim dan **ZMQ Remote API** untuk:
 - membangun lingkungan simulasi dengan path yang telah didefinisikan,
 - membaca pose robot dan data path, serta mengirim perintah kecepatan roda secara terprogram.
3. Menerapkan **transformasi koordinat** menggunakan matriks homogen 4×4 untuk memetakan titik referensi path ke *frame robot*.
4. Merancang dan mengimplementasikan **kontrol path tracking** berbasis sinyal galat posisi dan orientasi pada robot *differential-drive*.
5. Menghasilkan dan menginterpretasikan **visualisasi data** berupa lintasan dan kurva galat terhadap waktu sebagai alat analisis dan dokumentasi eksperimen.
6. (Opsional) Mengelola kode dan dokumentasi proyek menggunakan **Git/GitHub** sehingga eksperimen mudah direplikasi dan dikembangkan lebih lanjut.

2 Dasar Teori

2.1 Python

Python adalah bahasa tingkat tinggi yang mudah dibaca, portabel, dan memiliki ekosistem pustaka yang kaya (mis. NumPy, Matplotlib). Berjalan melalui interpreter lintas OS, Python memungkinkan pengembang fokus pada logika aplikasi tanpa mengurus detail rendah (memori, register, instruksi mesin), sehingga pengembangan lebih cepat dan ringkas.

2.2 CoppeliaSim

CoppeliaSim merupakan simulator robotika dengan mesin fisika untuk memodelkan robot, sensor, aktuator, dan lingkungan virtual. Platform ini banyak dipakai untuk riset dan pendidikan karena aman untuk uji algoritma sebelum implementasi nyata, serta mendukung berbagai API (Python, C++, MATLAB, ROS) untuk integrasi yang fleksibel.

2.3 Application Programming Interface (API)

API adalah antarmuka standar (aturan/fungsi/protokol) yang memungkinkan aplikasi berkomunikasi dengan sistem lain. Dengan API, pengembang dapat memakai layanan/fungsi yang sudah ada tanpa memahami detail internalnya, sehingga interoperabilitas meningkat dan proses pengembangan menjadi lebih cepat dan modular.

2.4 ZMQ Remote API

ZMQ Remote API pada CoppeliaSim (berbasis ZeroMQ) memungkinkan program eksternal mengontrol simulasi secara *message-based* dan asinkron. Melalui klien Python/C++/MATLAB, pengguna dapat menjalankan/menjeda simulasi, mengakses objek, membaca sensor, dan menggerakkan robot—dengan pemisahan yang jelas antara logika kendali di luar simulator dan lingkungan virtual di dalamnya.

2.5 NumPy

NumPy adalah pustaka Python untuk komputasi numerik. Inti NumPy adalah tipe data `ndarray`, yaitu larik (array) berdimensi- n dengan operasi vektor/matriks yang efisien. Beberapa hal penting:

- **Array** dibuat dengan `np.array([...], dtype=float)`.
- **Matriks identitas** dengan `np.eye(n)`.
- **Perkalian matriks** memakai operator `@` (bukan `*`).
- **Transpos** dengan `A.T`, dan blok-subarray dengan `A[:3, :3]`.

NumPy memudahkan kita membangun matriks rotasi, translasi, menggabungkannya menjadi matriks transformasi homogen 4×4 , lalu mengalikan dengan vektor titik.

2.6 Path, Trajectory, dan Path Tracking

Dalam robotika bergerak, penting untuk membedakan antara **path** dan **trajectory**:

- **Path** adalah lintasan *geometris* yang didefinisikan di ruang konfigurasi (misalnya bidang $x-y$), tanpa memuat informasi waktu eksplisit. Path dapat berupa rangkaian titik (x_i, y_i) atau kurva halus yang menghubungkan titik awal dan titik tujuan.
- **Trajectory** adalah lintasan yang *diparameterkan terhadap waktu*, misalnya $(x(t), y(t), \theta(t))$, sehingga selain bentuk geometris juga ditentukan *kapan* robot berada di setiap titik path.

Berkaitan dengan itu, terdapat dua tugas utama:

- **Path tracking / path following:** mengendalikan robot agar tetap berada di sekitar path yang diinginkan, tanpa mensyaratkan waktu tertentu untuk setiap titik path. Fokus utamanya adalah *konsistensi geometris* terhadap path.
- **Trajectory tracking:** mengendalikan robot agar mengikuti lintasan sekaligus jadwal waktu yang ditentukan, sehingga kesalahan dilihat pada domain posisi dan waktu.

Dalam proyek ini, yang diimplementasikan adalah **path tracking**. Objek /Path di CoppeliaSim menyediakan sekumpulan titik (x_i, y_i) di *world frame* yang membentuk lintasan ideal. Robot harus bergerak sedemikian rupa sehingga posisinya selalu dekat dengan path tersebut, dengan strategi:

1. Menentukan **titik look-ahead** di depan robot sejauh jarak tertentu (*look-ahead distance*) berdasarkan pose robot saat ini.
2. Mencari **titik path terdekat** terhadap titik look-ahead tersebut (aproksimasi proyeksi tegak lurus path).
3. Menggunakan titik path terpilih sebagai **titik referensi** yang harus dikejar oleh robot.
4. Mendefinisikan galat posisi dan heading robot terhadap titik referensi ini dalam *frame robot*, lalu mengubah galat tersebut menjadi perintah kecepatan v dan ω .

Strategi ini mirip dengan pendekatan *Pure Pursuit* secara intuitif, di mana robot selalu “mengejar” titik di depan sepanjang lintasan. Dengan cara ini, robot cenderung mengikuti bentuk path secara halus tanpa harus mengunjungi setiap titik path secara eksplisit.

2.7 Odometri Robot Mobile

Odometri merupakan metode untuk memperkirakan posisi dan orientasi (*pose*) robot berdasarkan informasi gerakan roda. Prinsip dasar odometri adalah mengintegrasikan kecepatan linier dan kecepatan sudut dari robot terhadap waktu, sehingga dapat diperoleh estimasi lintasan.

Pada robot *differential-drive*, odometri dihitung dari kecepatan sudut roda kanan (ω_r) dan kiri (ω_l) dengan jari-jari roda R serta setengah jarak antar roda L_{half} :

$$\begin{aligned} v_r &= \omega_r R, & v_l &= \omega_l R \\ v &= \frac{v_r + v_l}{2}, & \omega &= \frac{v_r - v_l}{2L_{\text{half}}} \end{aligned}$$

dengan v adalah kecepatan translasi robot dan ω adalah kecepatan rotasi (yaw rate). Pose robot (x, y, θ) kemudian diperbarui dengan integrasi:

$$\begin{aligned} x(t + \Delta t) &= x(t) + v \cos \theta \Delta t, \\ y(t + \Delta t) &= y(t) + v \sin \theta \Delta t, \\ \theta(t + \Delta t) &= \theta(t) + \omega \Delta t. \end{aligned}$$

Pada proyek ini, pose robot tidak dihitung melalui odometri, melainkan dibaca langsung dari CoppeliaSim sebagai *ground-truth*. Namun, hubungan kinematika differential-drive yang sama tetap digunakan untuk:

- menghubungkan kecepatan roda kanan–kiri dengan kecepatan tubuh robot (v, ω) ,
- membalik hubungan tersebut untuk mengonversi sinyal kontrol (v, ω) menjadi perintah kecepatan roda dalam bentuk kecepatan sudut joint.

2.8 Pose, Frame Koordinat, dan Matriks Transformasi Homogen

2.8.1 Pose dan Frame Koordinat

Dalam robotika mobile, posisi dan orientasi sebuah benda kaku relatif terhadap suatu acuan dinyatakan dalam bentuk **pose**. Untuk gerak planar (2D), pose robot relatif terhadap *world frame* $\{W\}$ sering ditulis sebagai:

$${}^w p_b = \begin{bmatrix} x_b \\ y_b \\ \theta_b \end{bmatrix},$$

dengan:

- x_b, y_b : posisi pusat robot di bidang lantai,
- θ_b : orientasi (yaw) robot relatif terhadap sumbu x_w .

Untuk menangani rotasi dan translasi secara seragam, digunakan konsep **frame koordinat** dan **transformasi koordinat** antar-frame. Setiap entitas (world, robot, objek, path lokal) diberi frame sendiri ($\{W\}, \{B\}, \{P\}$, dll.), dan hubungan antar-frame dinyatakan dengan transformasi homogen.

Dalam konteks path tracking, yang sering digunakan adalah:

- $\{W\}$: *world frame*, acuan global,
- $\{B\}$: *body frame* yang melekat pada robot (sumbu x_b ke depan robot),
- $\{P\}$: frame lokal di sepanjang path (misalnya di titik referensi) jika ingin membahas orientasi path.

2.8.2 Matriks Transformasi Homogen

Transformasi dari frame $\{B\}$ ke frame $\{W\}$ dapat ditulis sebagai

$${}^w T_b = \begin{bmatrix} R & t \\ 0 & 0 & 1 \end{bmatrix},$$

dengan:

- $R \in R^{3 \times 3}$: matriks rotasi, menyatakan orientasi $\{B\}$ relatif terhadap $\{W\}$,
- $t \in R^{3 \times 1}$: vektor translasi, menyatakan posisi asal $\{B\}$ dalam koordinat $\{W\}$.

Untuk gerak planar, rotasi efektif hanya pada sumbu z , tetapi representasi R dan t tetap memakai bentuk 3D agar konsisten dengan konvensi umum dalam robotika dan dengan API CoppeliaSim.

2.8.3 Matriks Rotasi dari Sudut Euler (Roll, Pitch, Yaw)

Salah satu cara umum untuk membangun matriks rotasi adalah dari sudut Euler:

$$\alpha = \text{roll}, \quad \beta = \text{pitch}, \quad \gamma = \text{yaw}.$$

Rotasi elementer didefinisikan sebagai:

$$\begin{aligned} R_x(\alpha) &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha \\ 0 & \sin \alpha & \cos \alpha \end{bmatrix}, \\ R_y(\beta) &= \begin{bmatrix} \cos \beta & 0 & \sin \beta \\ 0 & 1 & 0 \\ -\sin \beta & 0 & \cos \beta \end{bmatrix}, \\ R_z(\gamma) &= \begin{bmatrix} \cos \gamma & -\sin \gamma & 0 \\ \sin \gamma & \cos \gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}. \end{aligned}$$

Dengan suatu konvensi urutan (misalnya rotasi roll–pitch–yaw), rotasi total dapat ditulis sebagai

$$R = R_x(\alpha) R_y(\beta) R_z(\gamma).$$

Untuk kasus robot mobile planar, biasanya $\alpha = 0$ dan $\beta = 0$, sehingga orientasi hanya bergantung pada yaw $\gamma = \theta$.

2.8.4 Transformasi Titik antar-Frame

Misalkan posisi titik referensi path dalam frame $\{W\}$ dinyatakan sebagai vektor homogen:

$${}^w p_{\text{ref}} = \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}.$$

Posisi titik yang sama dalam frame $\{B\}$, yaitu ${}^b p_{\text{ref}}$, terkait dengan:

$${}^w p_{\text{ref}} = {}^w T_b {}^b p_{\text{ref}} \quad \Rightarrow \quad {}^b p_{\text{ref}} = ({}^w T_b)^{-1} {}^w p_{\text{ref}}.$$

Karena transformasi homogen memiliki bentuk

$${}^w T_b = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix},$$

maka inversnya diberikan oleh:

$$({}^w T_b)^{-1} = \begin{bmatrix} R^\top & -R^\top t \\ 0 & 1 \end{bmatrix}.$$

Rumus ini penting untuk memetakan titik referensi path dari *world frame* ke *body frame* robot, yang kemudian digunakan untuk mendefinisikan galat posisi dalam koordinat robot.

2.9 Definisi Galat Posisi dan Orientasi dalam Path Tracking

Dalam tugas *path tracking*, robot diharapkan mengatur **posisi** dan (jika diinginkan) **orientasinya** sehingga posisi tengah badan robot $\{B\}$ selalu berada dekat dengan path, dan orientasi robot mengikuti arah lokal path atau orientasi referensi lain (misalnya orientasi objek /Disc).

Misalkan posisi titik path referensi dalam frame robot adalah:

$${}^b p_{\text{ref}} = \begin{bmatrix} x_{pb} \\ y_{pb} \\ z_{pb} \\ 1 \end{bmatrix}.$$

Lalu yaw robot dan yaw referensi (misalnya dari path tangent atau disk) masing-masing adalah θ_b dan θ_{ref} .

2.9.1 Error Posisi Sepanjang Sumbu Depan Robot

Untuk mendekatkan robot ke titik referensi path sepanjang arah “maju” robot, komponen posisi titik referensi pada sumbu x_b (sumbu depan robot) dapat didefinisikan sebagai **error posisi**:

$$e_d = x_{pb}.$$

- Jika $e_d > 0$, titik referensi path berada di depan robot (robot perlu maju).
- Jika $e_d < 0$, titik referensi berada di belakang robot (robot perlu mundur atau memutar badan).

2.9.2 Error Heading (Arah ke Titik Path Referensi)

Error heading mendeskripsikan sudut arah titik path referensi terhadap sumbu depan robot, dalam frame robot:

$$e_h = \arctan 2(y_{pb}, x_{pb}).$$

Makna:

- $e_h = 0$: titik referensi segaris dengan sumbu depan robot,
- $e_h > 0$: titik referensi berada di sisi kiri robot,
- $e_h < 0$: titik referensi berada di sisi kanan robot.

Galat ini digunakan untuk memutar robot agar “menghadap” ke arah lintasan.

2.9.3 Error Orientasi Akhir

Jika orientasi robot juga ingin diselaraskan dengan orientasi path lokal atau objek referensi, dapat didefinisikan:

$$e_o = \text{wrapToPi}(\theta_{\text{ref}} - \theta_b),$$

yaitu selisih yaw referensi dan yaw robot yang dinormalisasi ke interval $(-\pi, \pi]$ sehingga merepresentasikan perbedaan sudut terkecil (*shortest angle*).

2.10 Normalisasi Sudut wrapToPi

Dalam perhitungan sudut, sering muncul masalah **diskontinuitas** karena periodisitas 2π . Misalnya, sudut 179° dan -179° secara geometris sangat berdekatan, tetapi selisihnya secara numerik tampak besar jika tidak dinormalisasi.

Normalisasi sudut ke rentang $(-\pi, \pi]$ dapat dilakukan, salah satunya, dengan:

$$\text{wrapToPi}(\phi) = 2(\sin \phi, \cos \phi).$$

Karena $2(\sin \phi, \cos \phi)$ selalu menghasilkan nilai di $(-\pi, \pi]$, maka $\text{wrapToPi}(\phi)$ memberikan representasi sudut yang konsisten dan kontinu, sehingga selisih sudut dapat dianalisis tanpa lompatan numerik yang besar. Pada proyek ini, normalisasi sudut penting untuk menjaga perhitungan e_h dan e_o tetap stabil.

2.11 Mode Switching Berbasis Fungsi Eksponensial

2.11.1 Motivasi Mode Switching

Dalam kontrol pose dan path tracking, terdapat dua fokus utama:

1. **Kontrol posisi:** membawa robot mendekati path (mengurangi e_d).
2. **Kontrol orientasi:** menyelaraskan arah robot dengan orientasi referensi (mengurangi e_o).

Secara intuitif:

- Ketika robot masih jauh dari path atau target, lebih penting mengarahkan robot agar menuju path (komponen heading e_h lebih dominan).
- Ketika robot sudah dekat path, fokus dapat dialihkan ke penyesuaian orientasi akhir agar orientasi robot sejajar dengan arah path atau objek referensi.

Untuk melakukan perpindahan fokus secara halus, dapat digunakan fungsi **mode switching** yang bergantung pada jarak robot terhadap path.

2.11.2 Definisi Fungsi Mode

Misalkan jarak Euclidean di bidang antara robot dan titik referensi path adalah:

$$d = \sqrt{(x_{\text{ref}} - x_b)^2 + (y_{\text{ref}} - y_b)^2}.$$

Salah satu fungsi *mode* yang halus dan monoton adalah:

$$\text{mode}(d) = \exp\left(-\frac{d}{d_{\text{sw}}}\right),$$

dengan d_{sw} adalah jarak karakteristik (*switching distance*).

Sifat-sifat:

- $d \gg d_{\text{sw}} \Rightarrow \text{mode} \approx 0$: robot jauh, fokus ke heading (e_h).
- $d \approx 0 \Rightarrow \text{mode} \approx 1$: robot dekat, fokus ke orientasi akhir (e_o).
- $\text{mode}(d)$ menurun secara halus seiring d bertambah.

2.11.3 Pengaruh Mode terhadap Gain Kontrol

Fungsi mode dapat digunakan untuk **menginterpolasi** gain kontrol, misalnya:

$$k_{p,\text{ang}}(d) = k_{\text{ang},\text{max}} [1 - \text{mode}(d)], \quad k_{p,\text{ori}}(d) = k_{\text{ori},\text{max}} \text{mode}(d).$$

Secara kualitatif:

- Saat jauh (mode ≈ 0):
 $k_{p,\text{ang}} \approx k_{\text{ang},\text{max}}, \quad k_{p,\text{ori}} \approx 0$,
sehingga komponen heading e_h dominan.
- Saat dekat (mode ≈ 1):
 $k_{p,\text{ang}} \approx 0, \quad k_{p,\text{ori}} \approx k_{\text{ori},\text{max}}$,
sehingga komponen orientasi akhir e_o dominan.
- Gain linear $k_{p,\text{lin}}$ dapat dipilih konstan atau dibuat fungsi jarak (misalnya diperkecil saat dekat) untuk menghindari gerak yang terlalu agresif di sekitar path.

2.12 Hukum Kontrol Kecepatan dan Kinematika Differential-Drive

2.12.1 Hukum Kontrol v dan ω

Dalam kerangka kontrol proporsional sederhana, kecepatan linear dan kecepatan sudut robot dapat didefinisikan sebagai:

$$v = k_{p,\text{lin}} e_d,$$

$$\omega = k_{p,\text{ang}} e_h + k_{p,\text{ori}} e_o.$$

Interpretasi:

- v mengurangi jarak sepanjang sumbu depan robot terhadap titik referensi path.
- ω terdiri dari dua kontribusi:
 - satu untuk mengarahkan robot ke titik path (e_h),
 - satu untuk menyelaraskan orientasi akhir (e_o) bila digunakan.

Dengan memilih $k_{p,\text{lin}}, k_{p,\text{ang}}, k_{p,\text{ori}}$ yang sesuai (serta fungsi mode), dapat diperoleh gerak yang stabil dan responsif mengikuti path.

2.12.2 Kinematika Differential-Drive: Turunan Rumus

Konfigurasi Geometris Robot Anggap sebuah robot dengan dua roda penggerak:

- roda kanan bergerak dengan kecepatan linear v_R ,
- roda kiri bergerak dengan kecepatan linear v_L ,
- jarak antar roda (lebar jejak roda) adalah $2r_b$,
- titik tengah antara kedua roda didefinisikan sebagai **pusat badan** robot.

Asumsi dasar kinematika:

- roda tidak mengalami slip (murni bergulir),
- robot bergerak di bidang (gerak planar),
- pusat badan robot bergerak dengan kecepatan linear v ,
- robot berotasi dengan kecepatan sudut yaw ω terhadap sumbu vertikal.

Gerak pada Lintasan Melingkar (Instantaneous Center of Curvature) Secara umum, jika kedua roda berputar dengan kecepatan yang mungkin *berbeda*, robot bergerak sepanjang suatu busur lingkaran. Pada saat tertentu, gerak robot dapat dimodelkan sebagai rotasi mengelilingi suatu titik khusus yang disebut **Instantaneous Center of Curvature (ICC)**.

Misalkan:

- jarak dari ICC ke pusat badan robot adalah R ,
- maka jarak dari ICC ke roda kanan adalah $R + r_b$,
- dan jarak dari ICC ke roda kiri adalah $R - r_b$.

Jika robot berotasi dengan kecepatan sudut ω terhadap ICC, maka:

$$v = \omega R, \quad v_R = \omega(R + r_b), \quad v_L = \omega(R - r_b).$$

Dari sini terlihat:

- ketika $R \rightarrow \infty$, lintasan mendekati garis lurus dan $v_R \approx v_L$,
- ketika $R = 0$, pusat badan robot tepat di ICC dan robot hanya berputar di tempat.

Derivasi Rumus Kecepatan Linear Pusat Badan v Kita mulai dari dua persamaan:

$$v_R = \omega(R + r_b), \quad v_L = \omega(R - r_b).$$

Dengan menjumlahkan:

$$v_R + v_L = \omega(R + r_b) + \omega(R - r_b) = \omega(2R) = 2\omega R.$$

Di sisi lain, kecepatan linear pusat badan adalah:

$$v = \omega R.$$

Sehingga:

$$v_R + v_L = 2v \Rightarrow v = \frac{v_R + v_L}{2}.$$

Interpretasi:

- kecepatan linear pusat badan robot adalah **rata-rata** kecepatan linear kedua roda,
- jika $v_R = v_L$, maka v sama dengan kecepatan masing-masing roda dan robot bergerak lurus tanpa berputar.

Derivasi Rumus Kecepatan Sudut ω Masih dari

$$v_R = \omega(R + r_b), \quad v_L = \omega(R - r_b),$$

kita kurangkan:

$$v_R - v_L = \omega(R + r_b) - \omega(R - r_b) = \omega(2r_b).$$

Maka:

$$v_R - v_L = 2r_b \omega \Rightarrow \omega = \frac{v_R - v_L}{2r_b}.$$

Interpretasi:

- ω ditentukan oleh **seberapa besar perbedaan** kecepatan kedua roda,
- jika $v_R = v_L$, maka $\omega = 0$: robot tidak berputar, hanya translasi,
- jika $v_R = -v_L$, maka $v = 0$ dan robot berputar di tempat:

$$\omega = \frac{v_R - (-v_R)}{2r_b} = \frac{2v_R}{2r_b} = \frac{v_R}{r_b}.$$

Bentuk Sistem Linier dan Matriks Kinematika Dua hubungan dasar yang sudah diperoleh dapat ditulis sebagai:

$$v = \frac{v_R + v_L}{2}, \quad \omega = \frac{v_R - v_L}{2r_b}.$$

Jika kita kalikan kedua persamaan dengan 2, diperoleh sistem:

$$\begin{cases} v_R + v_L = 2v, \\ v_R - v_L = 2r_b \omega. \end{cases}$$

Dari sini, kita dapat menyelesaikan untuk v_R dan v_L . Menjumlahkan:

$$(v_R + v_L) + (v_R - v_L) = 2v + 2r_b \omega \Rightarrow 2v_R = 2v + 2r_b \omega \Rightarrow v_R = v + r_b \omega.$$

Mengurangkan:

$$(v_R + v_L) - (v_R - v_L) = 2v - 2r_b \omega \Rightarrow 2v_L = 2v - 2r_b \omega \Rightarrow v_L = v - r_b \omega.$$

Dalam bentuk matriks:

$$\begin{bmatrix} v_R \\ v_L \end{bmatrix} = \begin{bmatrix} 1 & r_b \\ 1 & -r_b \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}.$$

2.12.3 Kecepatan Linear Roda dan Kecepatan Sudut Roda

Jika jari-jari roda adalah r_w , maka berlaku:

$$v_R = r_w \omega_R, \quad v_L = r_w \omega_L,$$

dengan ω_R, ω_L kecepatan sudut roda kanan-kiri (rad/s). Dari sini diperoleh:

$$\omega_R = \frac{v_R}{r_w}, \quad \omega_L = \frac{v_L}{r_w}.$$

Hubungan-hubungan ini adalah dasar untuk mengonversi sinyal kontrol (v, ω) menjadi perintah fisik ke aktuator (motor roda) dalam bentuk kecepatan sudut roda, yang kemudian diimplementasikan di CoppeliaSim melalui perintah `setJointTargetVelocity`.

3 Alur Kerja Program Simulasi Path Tracking

Program menjalankan *pipeline* berikut: *konfigurasi* → *koneksi simulator* → *loop kontrol path tracking (posisi + orientasi)* → *pencatatan data* → *terminasi* → *visualisasi hasil*.

1. Konfigurasi Awal

- Menetapkan path objek scene di CoppeliaSim: `/PioneerP3DX`, `/rightMotor`, `/leftMotor`, `/Disc` (sebagai referensi orientasi), objek lintasan `/Path`, serta dua marker visual `/LH` (look-ahead point) dan `/Perp` (titik path yang sedang dikejar).
- Menentukan **parameter geometris** robot:
 - jari-jari roda r_w (mis. `rw = 0.195/2`),
 - *half-wheelbase* r_b (mis. `rb = 0.381/2`).
- Menentukan **parameter kontrol**:
 - `look_dist`: jarak *look-ahead* di depan robot (mis. 0,5 m),
 - `d`: jarak karakteristik untuk *mode switching* (mis. 0,05 m),
 - `kp_lin`: gain proporsional untuk kecepatan linear,
 - `kp_ang` dan `kp_ori` yang nantinya diturunkan dari fungsi `mode`.
- Menentukan **durasi eksperimen** (mis. 60 detik) dan inisialisasi buffer data:
 - `d_xyaw` untuk menyimpan (x_w, y_w, yaw) ,
 - `d_t` untuk menyimpan waktu,
 - `dat_disc2rob` untuk menyimpan posisi titik referensi path dalam frame robot,
 - `dat_errors` untuk menyimpan galat (e_d, e_h, e_o) .

2. Koneksi ke CoppeliaSim dan Pembacaan Path

- Membuat klien ZMQ: `client = RemoteAPIClient()` dan mengambil modul `sim`.
- Menonaktifkan mode *stepping* eksplisit: `sim.setStepping(False)` sehingga simulasi berjalan kontinu.
- Memanggil `sim.startSimulation()` untuk memulai simulasi.
- Mengambil *handle* objek:
 - base robot: `/PioneerP3DX`,
 - roda kanan-kiri: `/rightMotor`, `/leftMotor`,
 - objek orientasi: `/Disc`,
 - objek lintasan: `/Path`,
 - marker visual: `/LH` dan `/Perp`.
- Membaca data path sekali di awal:
 - Mengambil buffer data path: `pathBuf = sim.getBufferProperty(path_Handle, 'customData.PATH')`.
 - Men-decode buffer menjadi tabel *double*: `pathData = sim.unpackDoubleTable(pathBuf)`.
 - Membentuk array NumPy: `np_path = np.array(pathData).reshape(-1, 7)`, sehingga setiap baris berisi $(x_i, y_i, z_i, qxi, qyi, qzi, qw_i)$ di *world frame*.
 - Hanya kolom (x_i, y_i) yang digunakan untuk path tracking.

3. **Loop Kontrol Path Tracking (Posisi + Orientasi)** Loop utama berjalan selama durasi eksperimen (mis. $t \leq 60$ detik) dan pada setiap iterasi melakukan langkah-langkah berikut:

(a) *Hitung waktu simulasi relatif*:

$$t_{\text{now}} = t_{\text{real}} - t_{\text{start}},$$

untuk menyatakan waktu sejak eksperimen dimulai di Python.

(b) *Baca pose robot dan disk di world frame*:

- posisi robot: `bod_pos_xyz = (xb, yb, zb) = getObjectPosition(P3DX, world)`,
- orientasi robot: `bod_pos_abg = (αb, βb, γb)`, dengan yaw robot $\theta_b = \gamma_b$,
- posisi disk: `disc_pos_xyz = (xd, yd, zd)`,
- orientasi disk: `disc_pos_abg = (αd, βd, γd)`, dengan yaw disk $\theta_d = \gamma_d$.

(c) *Hitung titik look-ahead di world frame*:

- Ditetapkan jarak look-ahead: `look_dist`.
- Dengan yaw robot θ_b , titik look-ahead adalah:

$$p_{\text{LA}} = \begin{bmatrix} x_b + \text{look_dist} \cos \theta_b \\ y_b + \text{look_dist} \sin \theta_b \end{bmatrix}.$$

• Titik ini divisualisasikan di CoppeliaSim dengan memindahkan objek /LH ke koordinat $(x_{\text{LA}}, y_{\text{LA}}, z_{\text{marker}})$.

(d) *Pilih titik path referensi terdekat terhadap look-ahead*:

- Ambil hanya koordinat path: `np_path_xy = [xi, yi]`.
- Hitung posisi relatif semua titik path terhadap titik look-ahead:

$$r_i = \begin{bmatrix} x_i - x_{\text{LA}} \\ y_i - y_{\text{LA}} \end{bmatrix}.$$

Secara numerik: `np_path_rel = np_path_xy - look_ahead_pt`.

- Hitung jarak Euclidean masing-masing titik:

$$d_i = \|r_i\|_2,$$

dengan NumPy: `path_sse = np.linalg.norm(np_path_rel, axis=1)`.

- Cari indeks titik path dengan jarak minimum:

$$\text{pendic_idx} = \arg \min_i d_i.$$

- Titik path pada indeks ini menjadi **titik referensi path** yang harus dikejar.
- Titik referensi divisualisasikan dengan memindahkan objek /Perp ke koordinat $(x_{\text{ref}}, y_{\text{ref}}, z_{\text{marker}})$.

(e) *Bentuk vektor homogen posisi titik referensi di world*:

$$w_{\text{pref}} = \begin{bmatrix} x_{\text{ref}} \\ y_{\text{ref}} \\ 0 \\ 1 \end{bmatrix}.$$

(f) *Bentuk matriks transformasi world → body*:

- Susun transformasi ${}^wT_b = T(\alpha_b, \beta_b, \theta_b, x_b, y_b, 0)$ dengan fungsi `transformMat`.
- Ambil inversnya untuk memperoleh ${}^bT_w = ({}^wT_b)^{-1}$.

(g) *Transformasi titik path referensi ke frame robot*:

$${}^b_{\text{pref}} = {}^bT_w w_{\text{pref}} = \begin{bmatrix} x_{pb} \\ y_{pb} \\ z_{pb} \\ 1 \end{bmatrix}.$$

Komponen x_{pb} dan y_{pb} digunakan sebagai basis perhitungan galat.

(h) *Hitung galat posisi & orientasi*:

i. **Error posisi sepanjang sumbu depan:**

$$e_d = x_{pb},$$

sehingga titik path diharap berada di depan robot ($e_d > 0$) dan menuju nol seiring robot mendekati lintasan.

ii. **Error heading (arah ke titik path):**

$$e_h = \arctan 2(y_{pb}, x_{pb}),$$

dengan interpretasi: $e_h = 0$ berarti titik referensi segaris dengan sumbu depan robot, sedangkan tanda e_h menunjukkan sisi kiri/kanan.

iii. **Error orientasi akhir (yaw) terhadap disk:**

$$e_o = \text{wrapToPi}(\theta_d - \theta_b),$$

yaitu perbedaan yaw disk dan yaw robot yang dinormalisasi ke rentang $(-\pi, \pi]$.

iv. **Jarak Euclidean robot–titik referensi path di body frame:**

$$\text{abs_d} = \sqrt{x_{pb}^2 + y_{pb}^2},$$

digunakan sebagai indikator kedekatan robot terhadap titik path yang sedang dikejar.

(i) *Hitung mode switching posisi–orientasi:*

- Definisikan

$$\text{mode} = \exp\left(-\frac{\text{abs_d}}{d}\right),$$

dengan d sebagai jarak karakteristik.

- Saat robot jauh dari titik referensi: $\text{abs_d} \gg d \Rightarrow \text{mode} \approx 0$, kontrol heading e_h lebih dominan.
- Saat robot dekat: $\text{abs_d} \approx 0 \Rightarrow \text{mode} \approx 1$, kontrol orientasi akhir e_o lebih dominan.
- Gain kontrol diatur sebagai:

$$k_{p,\text{lin}} = 1.2 \quad (\text{konstan}),$$

$$k_{p,\text{ang}} = 0.8(1 - \text{mode}), \quad k_{p,\text{ori}} = 8.0 \text{ mode}.$$

(j) *Hitung kecepatan linear dan sudut robot:*

$$v = k_{p,\text{lin}} e_d,$$

$$\omega = k_{p,\text{ang}} e_h + k_{p,\text{ori}} e_o.$$

Nilai v menggerakkan robot maju/mundur menuju path, sedangkan ω memutar robot untuk menegarah ke path dan menyelaraskan orientasi akhir.

(k) *Kinematika differential-drive → kecepatan roda:*

- Hubungan kinematika (dalam bentuk matriks):

$$\begin{bmatrix} v_R \\ v_L \end{bmatrix} = \begin{bmatrix} 1 & r_b \\ 1 & -r_b \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix},$$

sehingga diperoleh v_R dan v_L (m/s).

- Konversi ke kecepatan sudut roda:

$$\omega_R = \frac{v_R}{r_w}, \quad \omega_L = \frac{v_L}{r_w},$$

kemudian nilai inilah yang dikirim ke joint motor.

(l) *Aktuasi pada joint roda:*

- Mengirim perintah kecepatan sudut roda ke CoppeliaSim:

```
setJointTargetVelocity(rightMotor, ω_R),      setJointTargetVelocity(leftMotor, ω_L).
```

- Robot kemudian bergerak mengikuti kombinasi translasi–rotasi yang diinduksi oleh v dan ω .

(m) *Pencatatan data dan logging:*

- Menyimpan:
 - lintasan pusat robot di world: $(x_b(t), y_b(t), \theta_b(t))$ ke `d_xyaw`,
 - posisi titik referensi path dalam frame robot ${}^b p_{\text{ref}}(t)$ ke `dat_disc2rob`,
 - galat $e_d(t), e_h(t), e_o(t)$ ke `dat_errors`,
 - waktu t_{now} ke `d_t`.
- Mengirim `log` ke status bar CoppeliaSim berisi ringkasan nilai e_d, e_h , serta nilai `mode` dan waktu.

4. Terminasi Loop

- Setelah waktu eksperimen melewati batas (mis. $t_{\text{now}} > 60$ s), loop utama dihentikan.
- Kecepatan roda diset menjadi nol: `setJointTargetVelocity(wR, 0.0)`, `setJointTargetVelocity(wL, 0.0)`.
- (Opsional) Simulasi dapat dihentikan secara eksplisit dengan `sim.stopSimulation()`.
- Data yang terkumpul dikonversi menjadi larik NumPy: `dat_xyaw`, `dat_t`, `dat_disc2rob`, `dat_errors`.
- Sudut yaw lintasan robot dinormalisasi ke $(-\pi, \pi]$ untuk konsistensi:

$$\theta_b(t) \leftarrow 2(\sin \theta_b(t), \cos \theta_b(t)).$$

5. Visualisasi Hasil Eksperimen

(a) **Plot lintasan $x-y$ robot di world frame:**

- menampilkan kurva $(x_b(t), y_b(t))$ dengan penanda titik awal dan akhir,
- memberi label sumbu x_w dan y_w , grid, dan legenda,
- disimpan sebagai berkas `.svg` untuk dokumentasi.

(b) **Plot galat jarak $e_d(t)$:**

- menampilkan grafik e_d terhadap waktu t untuk melihat bagaimana komponen titik path sepanjang sumbu depan robot berkurang seiring waktu.

(c) **Plot galat heading $e_h(t)$:**

- menampilkan $e_h(t)$ (dikonversi ke derajat) terhadap waktu untuk mengamati proses penyelarasan arah robot terhadap path.

6. Normalisasi Sudut dan Konsistensi Data

- Sepanjang perhitungan, operasi pada sudut (terutama selisih yaw) menggunakan bentuk normalisasi:

$$\text{wrapToPi}(\phi) = 2(\sin \phi, \cos \phi),$$

agar sudut selalu berada pada rentang $(-\pi, \pi]$ dan menghindari lompatan numerik di sekitar $\pm\pi$.

- Normalisasi ini menjaga bentuk kurva galat sudut $(e_h(t), e_o(t))$ tetap halus dan mudah dianalisis, serta menghindari ambiguitas sudut akibat periodisitas 2π .

4 Tutorial Penggunaan Program dan Analisis Output Program

4.1 Prasyarat dan Instalasi

1. Python 3.x telah terpasang dan dapat diakses dari terminal (mis. `python --version` tidak menghasilkan error).
2. Pasang pustaka Python yang dibutuhkan:


```
pip install numpy matplotlib coppeliasim-zmqremoteapi-client
```
3. CoppeliaSim (Edu/Pro) terpasang dan dapat dijalankan di PC.
4. Program *path tracking* (misalnya `p3dx_path_tracking.py`) tersimpan sebagai satu berkas Python di suatu folder kerja.

4.2 Penyiapan Scene CoppeliaSim

1. Buka CoppeliaSim, lalu muat scene yang berisi:
 - robot /PioneerP3DX,
 - objek lintasan /Path (berisi titik-titik path di world),
 - dua objek bantu (marker) untuk visualisasi:
 - /LH sebagai marker *look-ahead point*,
 - /Perp sebagai marker titik path yang sedang dikejar.
 - (Opsional) objek /Disc sebagai referensi orientasi tambahan, jika ingin sekaligus melihat perilaku *orientation tracking*.
2. Pastikan hierarki/penamaan objek sesuai dengan program:
 - Base robot: /PioneerP3DX
 - Joint roda kanan: /rightMotor
 - Joint roda kiri: /leftMotor
 - Objek path: /Path
 - Marker look-ahead: /LH
 - Marker titik path referensi: /Perp
 - (Opsional) Objek orientasi: /Disc
3. Pastikan objek /Path sudah memiliki data lintasan (misalnya dibuat melalui *Path editing* di CoppeliaSim dan disimpan dalam `customData.PATH` atau properti serupa yang dibaca oleh program Python).
4. Atur posisi awal robot dan bentuk path sesuai skenario eksperimen (misalnya path berbentuk kurva di depan robot, dengan robot sedikit menyimpang dari lintasan pada awal simulasi).
5. Biarkan *simulation time step* pada nilai default atau sesuaikan sesuai kebutuhan (umumnya nilai default sudah cukup untuk eksperimen path tracking sederhana).

4.3 Konfigurasi Program

Sebelum menjalankan program, pengguna dapat menyesuaikan beberapa parameter pada bagian atas berkas Python (bagian *parameter robot & kontrol*), misalnya:

- **Parameter geometris robot:**
 - `rw`: jari-jari roda r_w (meter),
 - `rb`: *half-wheelbase* r_b (meter),
- Nilai ini biasanya diambil dari spesifikasi robot P3DX (atau mendekati) dan sebaiknya konsisten dengan model di scene CoppeliaSim.
- **Parameter look-ahead:**
 - `look_dist`: jarak *look-ahead* (meter), yaitu jarak di depan robot (sepanjang sumbu x_b) tempat titik look-ahead diletakkan. Nilai yang terlalu kecil dapat menyebabkan gerak berosilasi, sedangkan terlalu besar dapat membuat robot memotong tikungan.
- **Parameter switching jarak:**
 - `d`: jarak karakteristik (meter) untuk fungsi *mode switching* antara fokus kontrol heading dan orientasi akhir. Semakin kecil `d`, semakin tajam peralihan kontrol orientasi ketika robot mendekati titik referensi path.
- **Gain kontrol:**
 - `kp_lin`: gain proporsional untuk kecepatan linear v (mengurangi e_d),
 - `kp_ang`: basis gain untuk heading e_h (akan dikalikan $(1 - \text{mode})$),
 - `kp_ori`: basis gain untuk orientasi akhir e_o (akan dikalikan `mode`).

Nilai-nilai ini dapat di-tuning agar respon robot tidak terlalu lambat namun tetap stabil (tidak berosilasi berlebihan di sekitar path).

- **Durasi eksperimen:**

- Diatur pada kondisi penghentian loop, misalnya `if t_now > 60: break`, yang berarti program akan menjalankan kontrol selama ± 60 detik.

4.4 Menjalankan Program

1. Jalankan CoppeliaSim dan muat scene yang telah disiapkan. **Tidak perlu menekan tombol Play;** program Python akan memulai simulasi secara otomatis melalui `sim.startSimulation()`.
2. Buka terminal atau VS Code di folder tempat berkas Python disimpan.
3. Jalankan program, misalnya:

```
python p3dx_path_tracking.py
```
4. Program akan:
 - membuat koneksi ke CoppeliaSim via ZMQ Remote API,
 - memulai simulasi,
 - membaca data path dari objek `/Path`,
 - secara periodik membaca pose robot (dan disk bila digunakan),
 - menghitung titik look-ahead dan titik path referensi terdekat,
 - menghitung galat path tracking (e_d, e_h) serta (opsional) galat orientasi (e_o),
 - mengirim perintah kecepatan roda sehingga robot bergerak mengikuti path.
5. Setelah durasi eksperimen tercapai (mis. 60 detik) program:
 - menghentikan perintah gerak (kecepatan roda diset nol),
 - mengonversi data yang terekam ke dalam larik NumPy,
 - menampilkan jendela grafik hasil eksperimen,
 - menyimpan sebagian grafik ke berkas (misalnya format `.svg` atau `.png`) untuk dokumentasi laporan.

4.5 Output yang Dihasilkan

Setelah loop kontrol selesai, program menampilkan beberapa **jendela grafik** yang meringkas perilaku sistem selama eksperimen path tracking. Program menghasilkan tiga jendela utama:

1. **Jendela 1: Lintasan $x-y$ robot di world frame**

- Menampilkan lintasan pusat badan robot dalam koordinat dunia: $(x_w(t), y_w(t))$.
- Titik awal lintasan ditandai dengan marker (mis. lingkaran merah), sedangkan titik akhir dengan marker lain (mis. silang hijau).
- Grafik dilengkapi grid, label sumbu (x_w dan y_w), dan legenda.

Berdasarkan Gambar 1, pengguna dapat mengamati bagaimana robot bergerak dari posisi awal menuju dan mengikuti lintasan yang didefinisikan oleh objek `/Path`. Semakin mirip lintasan robot dengan lintasan path, semakin baik kualitas path tracking yang dicapai.

2. **Jendela 2: Kurva galat jarak $e_d(t)$**

- Menampilkan galat posisi sepanjang sumbu depan robot:

$$e_d(t) = x_{pb}(t),$$

yaitu komponen titik path referensi dalam frame robot pada sumbu x_b (arah maju).

- Sumbu horizontal menyatakan waktu t , dan sumbu vertikal menyatakan nilai e_d (meter).

Berdasarkan Gambar 2, nilai $e_d(t)$ yang bernilai mendekati nol menunjukkan bahwa loop kontrol berhasil mengurangi jarak antara robot dan titik path di sepanjang sumbu depan robot.

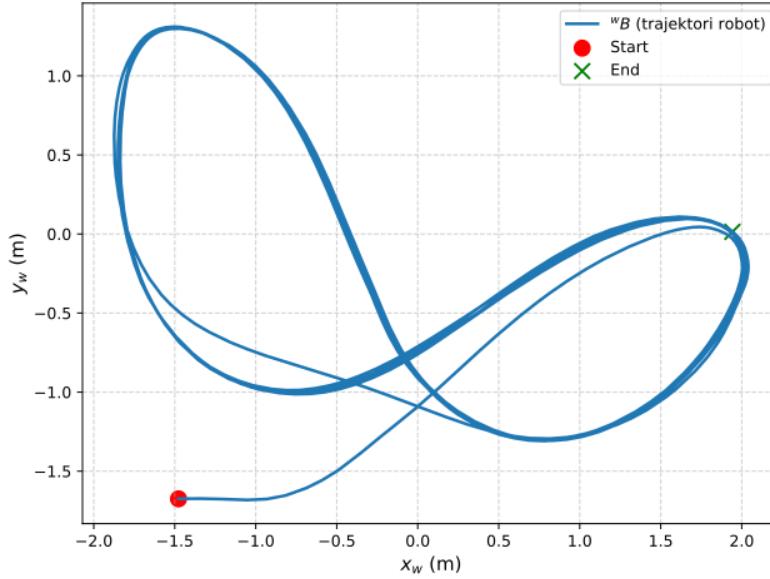


Figure 1: Lintasan robot P3DX pada bidang x_w-y_w selama proses path tracking

3. Jendela 3: Kurva galat heading $e_h(t)$

- Menampilkan galat heading:

$$e_h(t) = \arctan 2(y_{pb}(t), x_{pb}(t)),$$

yang biasanya dikonversi ke derajat untuk memudahkan interpretasi.

- Sumbu horizontal menyatakan waktu t , sedangkan sumbu vertikal menyatakan e_h (derajat).

Berdasarkan Gambar 3, nilai $e_h(t)$ yang bergerak disekitar nol seiring waktu menunjukkan bahwa loop kontrol berusaha secara bertahap memutar badan sehingga sumbu depan robot segaris dengan arah menuju titik path referensi. Pada fase awal, ketika robot masih jauh dari path, kontrol lebih menekankan pada pengurangan e_h . Ketika robot semakin dekat, kontribusi orientasi akhir (melalui e_o dan faktor `mode`) dapat menjadi lebih dominan jika diaktifkan.

Apabila lintasan robot mendekati path, serta $e_d(t)$ dan $e_h(t)$ menunjukkan trend konvergen menuju nol, maka dapat dikatakan konfigurasi scene dan parameter kontrol yang digunakan sudah tepat. Sebaliknya, jika grafik menunjukkan osilasi besar, divergensi, atau ketidakstabilan, maka diperlukan penyesuaian pada parameter kontrol (`kp_lin`, `kp_ang`, `kp_ori`), parameter `look_dist`, atau bahkan bentuk path dan posisi awal robot.

5 Kesimpulan

Berdasarkan perancangan, implementasi, dan pengujian program simulasi *path tracking* pada robot P3DX di CoppeliaSim, dapat disimpulkan bahwa:

1. Tujuan utama path tracking terhadap lintasan referensi tercapai.

Dengan memanfaatkan objek /Path sebagai lintasan referensi di *world frame*, program berhasil menggerakkan robot P3DX untuk mengikuti bentuk path secara umum. Hal ini terlihat dari lintasan (x_w, y_w) robot yang berada di sekitar kurva path dan dari galat posisi $e_d(t)$ serta galat heading $e_h(t)$ yang cenderung menurun seiring waktu. Dengan demikian, tujuan utama untuk mendemonstrasikan *path tracking* pada robot *differential-drive* telah tercapai.

2. Transformasi koordinat dan pemetaan titik path ke frame robot berjalan dengan baik.

Penggunaan matriks transformasi homogen wT_b dan inversnya bT_w untuk memetakan titik path referensi dari *world frame* ke *frame robot* terbukti bekerja dengan baik. Representasi titik path dalam bentuk vektor homogen, kemudian dikonversi menjadi koordinat (x_{pb}, y_{pb}) di frame robot, memungkinkan definisi galat e_d dan e_h dilakukan langsung dalam koordinat tubuh robot. Hal ini menjawab tujuan untuk mempraktikkan konsep *frame* dan transformasi koordinat dalam konteks kontrol lintasan.

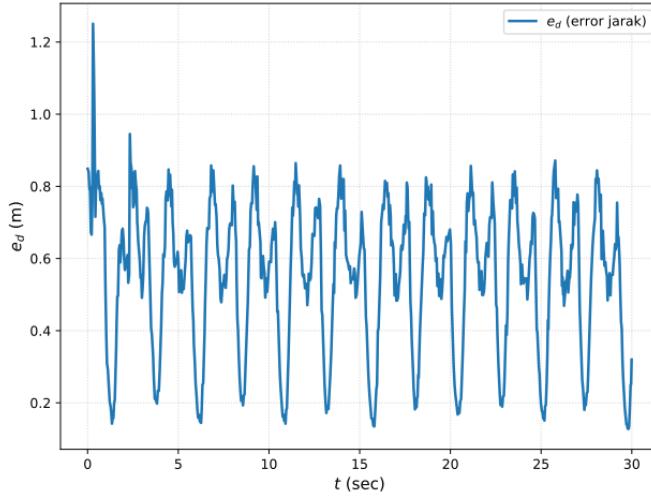


Figure 2: Galat jarak $e_d(t)$ antara robot dan titik path referensi sepanjang sumbu depan robot

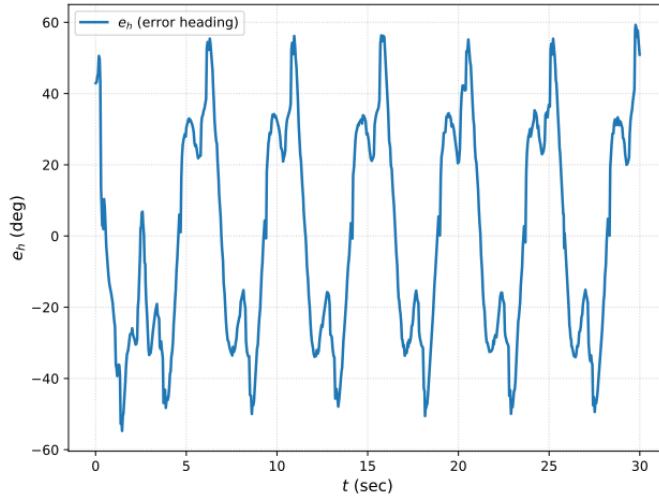


Figure 3: Galat heading $e_h(t)$ antara arah depan robot dan arah ke titik path referensi

3. Penggunaan metode look-ahead dalam pemilihan titik path referensi dapat menghasilkan path tracking yang cukup baik.

Pendekatan *look-ahead*, yaitu memilih titik di depan robot sejauh `look_dist` lalu mencari titik path terdekat terhadap titik tersebut, terbukti menghasilkan perilaku *path following* yang halus dan intuitif. Robot tidak sekadar mengejar satu titik statis, tetapi terus-menerus mengarahkan dirinya ke titik path yang berada di depan, sehingga lintasan yang terbentuk menyerupai path ideal. Dengan demikian, tujuan untuk menguji strategi path tracking berbasis *look-ahead point* dapat dikatakan tercapai.

4. Kontrol proporsional (v, ω) dengan blending heading–orientasi memberikan tracking yang stabil.

Kontrol proporsional :

$$v = k_{p,\text{lin}} e_d, \quad \omega = k_{p,\text{ang}} e_h + k_{p,\text{ori}} e_o$$

yang dikombinasikan dengan fungsi `mode` berbasis jarak menghasilkan gerak kombinasi translasi–rotasi yang halus, dimana robot maju ketika titik path berada di depan dan berputar ke arah path ketika terdapat galat heading, serta (bila diaktifkan) menyelaraskan yaw terhadap referensi orientasi. Dengan *tuning* parameter yang tepat, sistem menunjukkan respon yang stabil, sehingga tujuan untuk mengimplementasikan kontrol path tracking sederhana pada robot *differential-drive* dapat dicapai.

- Link Github :

<https://github.com/amrufal/path-track-p3dx-coppeliasim>

- Linkedin :

<https://www.linkedin.com/in/falah-amru-9a192a385/>