Tugas 6 Sistem Robot Otonom: Tracking Posisi dan Orientasi 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, tugas fundamental yang hampir selalu muncul adalah **mengetahui** dan **mengendalikan** pose robot terhadap lingkungan. Pada kasus gerak planar, pose cukup direpresentasikan dengan tiga besaran (x, y, yaw), yang menyatakan posisi dan orientasi robot terhadap sebuah *frame* acuan (misalnya *world frame*). Informasi pose ini dibutuhkan untuk berbagai tugas, seperti navigasi, pelacakan lintasan, *docking*, hingga berinteraksi dengan objek tertentu di lingkungan.

Dalam banyak skenario praktis, robot tidak hanya perlu "tahu di mana dirinya berada", tetapi juga harus mampu **menyesuaikan (menyelaraskan) posisi dan orientasinya terhadap objek tertentu** yang dianggap penting. Contohnya, robot harus berhenti tepat di depan objek dengan jarak tertentu, sekaligus menghadap ke arah yang sama dengan orientasi objek. Secara matematis, hal ini berarti membuat **frame robot** berimpit (atau setidaknya sejajar) dengan **frame objek** target.

Untuk mendeskripsikan hubungan antara robot, objek, dan lingkungan, digunakan konsep **transformasi koordinat** antarbeberapa frame: *world frame*, *body frame* (frame yang melekat pada robot), dan *object frame*. Transformasi ini dinyatakan dengan matriks homogen 4×4 sehingga rotasi dan translasi dapat dikomposisikan dengan cara yang seragam dan sistematis. Dengan transformasi tersebut, posisi objek dapat diubah dari koordinat dunia (${}^{w}p_d$) menjadi koordinat robot (${}^{b}p_d$), sehingga galat posisi dan orientasi dapat dihitung langsung dalam *frame robot*.

Pada tugas ini, simulasi dilakukan pada lingkungan (scene) yang dibuat menggunakan **CoppeliaSim** dengan model robot *wheeled mobile robot* Pioneer P3DX dan sebuah objek disk (`/Disc`) yang menjadi target. Baik robot maupun disk memiliki frame masing-masing yang pose-nya terhadap *world frame* dapat dibaca melalui *ZMQ Remote API*. Tujuan dari tugas simulasi ini antara lain adalah untuk :

- menggerakkan robot P3DX menuju objek target,
- menyelaraskan orientasi badan robot dengan orientasi objek,
- sehingga pada akhir simulasi, **frame robot mendekati frame objek** baik dari sisi posisi maupun orientasi.

Untuk mencapai berbagai tujuan simulasi diatas, dibutuhkan program untuk mengimplementasikan suatu loop kontrol yang bekerja sebagai berikut:

1. membaca pose robot dan objek terhadap *world frame*,
2. menyusun matriks transformasi ${}^{w}T_b$ lalu menghitung posisi objek dalam *frame robot* sebagai ${}^{b}p_d$,
3. mendefinisikan beberapa sinyal galat:
   - **error posisi** e_d (komponen objek di sumbu depan robot),
   - **error heading** e_h (arah garis pandang ke objek di frame robot),
   - **error orientasi akhir** e_o (selisih yaw objek dan yaw robot),
4. mengubah galat-galat ini menjadi kecepatan linear v dan kecepatan sudut ω robot,
5. mengonversi (v, ω) menjadi kecepatan roda kanan–kiri untuk robot *differential-drive*.

Dengan demikian, simulasi ini dapat mengilustrasikan bagaimana **transformasi koordinat** dan **sinyal galat pose** digunakan untuk mewujudkan *pose alignment* antara robot dan objek target.

1.2 Tujuan

Berdasarkan latar belakang tersebut, tujuan dari eksperimen ini adalah:

1. Menjelaskan dan mempraktikkan konsep **orientation and position tracking** pada robot *differential-drive* (P3DX) dalam lingkungan simulasi CoppeliaSim.
2. Mengimplementasikan **transformasi koordinat** berbasis matriks homogen 4×4 untuk mengubah pose objek dari *world frame* ke *frame robot* (${}^{b}p_d = {}^{b}T_w \, {}^{w}p_d$).
3. Mendapatkan beberapa **sinyal galat pose** yang digunakan sebagai input bagi loop kontrol:
   - galat jarak ke objek (e_d),
   - galat heading (e_h),
   - galat orientasi akhir (e_o).
4. Merancang **loop kontrol** yang mengubah galat-galat tersebut menjadi perintah kecepatan linear v dan kecepatan sudut ω, kemudian ke kecepatan roda kanan–kiri, sehingga robot mampu **menyelaraskan frame-nya dengan frame objek**.
5. Menyediakan **visualisasi dan data eksperimen** (lintasan dan galat terhadap waktu) sebagai sarana analisis dan pembelajaran mengenai perilaku sistem kontrol pose.

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 Disc.
   - Memulai simulasi dan mengatur mode eksekusi (kontinu/*stepping*).
   - Secara berkala membaca:
     - posisi dan orientasi robot (${}^{w}p_b$, yaw robot),
     - posisi dan orientasi objek (${}^{w}p_d$, yaw objek).
2. **Transformasi koordinat dan perhitungan galat**
   - Menyusun matriks transformasi ${}^{w}T_b$ dari posisi dan yaw robot (roll dan pitch diabaikan untuk gerak 2D).
   - Menghitung invers ${}^{b}T_w = ({}^{w}T_b)^{-1}$ dan memetakan posisi objek ke *frame robot*: ${}^{b}p_d = {}^{b}T_w \, {}^{w}p_d$.
   - Menghitung sinyal galat:
     - e_d: komponen x_b dari ${}^{b}p_d$ (jarak sepanjang sumbu depan robot),
     - $e_h = \arctan 2(y_b, x_b)$: galat heading ke objek,
     - $e_o = \text{wrapToPi}(\text{yaw}_{\text{disk}} - \text{yaw}_{\text{robot}})$: galat orientasi akhir.
   - Menghitung jarak Euclidean antara robot–objek di *world frame* sebagai ukuran kedekatan posisi.
3. **Mode switching antara kontrol posisi dan orientasi**
   - Menggunakan fungsi eksponensial mode $= \exp(-d/d_{\text{sw}})$ untuk menentukan mode atau fokus kontrol:
     - ketika robot masih jauh ($d \gg d_{\text{sw}}$): prioritas pada pengaturan heading (e_h),
     - ketika robot sudah dekat target ($d \approx 0$): prioritas pada pengaturan orientasi akhir (e_o).
   - Menyesuaikan gain kontrol heading dan orientasi ($k_{p,\text{ang}}$, $k_{p,\text{ori}}$) berdasarkan nilai mode (fungsi eksponensial).
4. **Perancangan kontrol dan aktuasi roda**
   - Menghitung kecepatan linear:
   $$v = k_{p,\text{lin}} \, e_d$$
   - Menghitung kecepatan sudut:
   $$\omega = k_{p,\text{ang}} \, e_h + k_{p,\text{ori}} \, e_o$$

- 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 objek dalam *frame robot* ${}^{b}p_d(t)$,
     - 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 objek,
     - grafik $e_h(t)$ (dalam derajat) untuk melihat proses penyelarasan heading.

1.4 Software yang Dibutuhkan dan Fungsinya

Beberapa perangkat lunak yang digunakan dalam proyek ini adalah:

1. **Python**: Bahasa pemrograman utama untuk mengimplementasikan loop kontrol, transformasi koordinat, pengolahan data, dan visualisasi.
2. **CoppeliaSim Edu**: Lingkungan simulasi robotik yang menyediakan model Pioneer P3DX, objek Disc, 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 dan kecepatan roda,
   - 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 *pose alignment* ini, beberapa tantangan yang muncul antara lain:

1. Menyusun **alur program** yang terstruktur mulai dari inisialisasi koneksi, akuisisi data, 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 ${}^{w}p_d \mapsto {}^{b}p_d$ konsisten dengan konvensi frame yang digunakan.
3. Mendefinisikan **sinyal galat pose** yang intuitif dan informatif untuk keperluan kontrol (memisahkan kontribusi posisi sepanjang sumbu depan, heading, dan orientasi akhir).
4. Merancang **mekanisme mode switching** yang halus antara kontrol posisi dan kontrol orientasi, agar robot tidak berosilasi dan respons tetap stabil.
5. Menangani **normalisasi sudut** (*wrap-to-*π) untuk mencegah diskontinuitas sudut yang dapat mengganggu perhitungan galat dan aksi kontrol.
6. Melakukan **tuning parameter kontrol** (gain linear dan angular) untuk mendapatkan kompromi yang baik antara kecepatan respon dan kestabilan gerak.

1.6 Kemampuan yang Dipelajari

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

1. Memahami konsep **pose** robot dan hubungan antar-frame (world, body, object) dalam konteks robotik bergerak planar.
2. Menggunakan CoppeliaSim dan **ZMQ Remote API** untuk:
   - membangun lingkungan simulasi sederhana,
   - membaca pose dan mengirim perintah kecepatan roda secara terprogram.
3. Menerapkan **transformasi koordinat** menggunakan matriks homogen 4×4 untuk memetakan posisi objek ke *frame robot*.
4. Merancang dan mengimplementasikan **kontrol pose** 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 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}:

$$v_r = \omega_r R, \qquad v_l = \omega_l R$$

$$v = \frac{v_r + v_l}{2}, \qquad \omega = \frac{v_r - v_l}{2L_{\text{half}}}$$

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

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

2.7 Pose, Frame Koordinat, dan Matriks Transformasi Homogen

2.7.1 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, sensor) diberi frame sendiri ($\{W\}, \{B\}, \{D\}$, dll.), dan hubungan antar-frame dinyatakan dengan transformasi homogen.

2.7.2 Matriks Transformasi Homogen

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

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

dengan:

- $R \in R^{3\times3}$: matriks rotasi, menyatakan orientasi $\{B\}$ relatif terhadap $\{W\}$,
- $t \in R^{3\times1}$: 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.

2.7.3 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:

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix},$$

$$R_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix},$$

$$R_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

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 $\gamma = \theta$.

2.7.4 Transformasi Titik antar-Frame

Misalkan pose sebuah titik (misalnya posisi objek) dalam frame $\{W\}$ dinyatakan sebagai vektor homogen:

$${}^{w}p = \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}.$$

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

$${}^{w}p = {}^{w}T_b\, {}^{b}p \quad \Rightarrow \quad {}^{b}p = ({}^{w}T_b)^{-1}\, {}^{w}p.$$

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 dari world ke body frame, yang kemudian digunakan untuk mendefinisikan galat posisi dalam koordinat robot.

2.8 Definisi Galat Posisi dan Orientasi dalam Pose Alignment

Pada tugas *pose alignment*, robot diharapkan mengatur **posisi** dan **orientasinya** sehingga frame robot $\{B\}$ mendekati frame objek $\{D\}$. Secara umum, hal ini memerlukan definisi galat di domain yang tepat, biasanya dalam *frame robot*.

Misalkan posisi objek dalam frame robot adalah:

$${}^{b}p_d = \begin{bmatrix} x_{db} \\ y_{db} \\ z_{db} \\ 1 \end{bmatrix}.$$

Lalu yaw robot dan yaw objek masing-masing adalah θ_b dan θ_d.

2.8.1 Error Posisi Sepanjang Sumbu Depan Robot

Untuk mendekatkan robot ke objek sepanjang arah "maju" robot, komponen posisi objek pada sumbu x_b (sumbu depan robot) dapat didefinisikan sebagai **error posisi**:

$$e_d = x_{db}.$$

- Jika $e_d > 0$, objek berada di depan robot (robot perlu maju).
- Jika $e_d < 0$, objek berada di belakang robot (robot perlu mundur atau memutar badan terlebih dahulu).

2.8.2 Error Heading (Arah ke Objek)

Error heading mendeskripsikan sudut arah objek terhadap sumbu depan robot, dalam frame robot:

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

Makna:

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

Galat ini digunakan untuk memutar robot agar "menghadap" objek.

2.8.3 Error Orientasi Akhir

Untuk menyelaraskan frame, orientasi akhir yang diinginkan adalah $\theta_b = \theta_d$. Galat orientasi dapat didefinisikan sebagai:

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

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

2.9 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 $2(y, x)$ 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.

2.10 Mode Switching Berbasis Fungsi Eksponensial

2.10.1 Motivasi Mode Switching

Dalam kontrol pose, terdapat dua fokus utama:

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

Secara intuitif:

- Ketika robot masih jauh dari target, lebih penting mengarahkan robot agar menuju objek (kontrol heading dominan).
- Ketika robot sudah dekat, fokus dialihkan ke penyesuaian orientasi akhir.

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

2.10.2 Definisi Fungsi Mode

Misalkan jarak Euclidean di bidang antara robot dan objek adalah:

$$d = \sqrt{(x_d - x_b)^2 + (y_d - y_b)^2}.$$

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

2.10.3 Pengaruh Mode terhadap Gain Kontrol

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

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

Secara kualitatif:

- Saat jauh (mode ≈ 0):
$$k_{p,\text{ang}} \approx k_{\text{ang,max}}, \quad k_{p,\text{ori}} \approx 0,$$
sehingga komponen heading e_h dominan.
- Saat dekat (mode ≈ 1):
$$k_{p,\text{ang}} \approx 0, \quad k_{p,\text{ori}} \approx k_{\text{ori,max}},$$
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 target.

2.11 Hukum Kontrol Kecepatan dan Kinematika Differential-Drive

2.11.1 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.
- ω terdiri dari dua kontribusi:
  - satu untuk mengarahkan robot ke objek (heading),
  - satu untuk menyelaraskan orientasi akhir.

Dengan memilih $k_{p,\text{lin}}, k_{p,\text{ang}}, k_{p,\text{ori}}$ yang sesuai, dapat diperoleh gerak yang stabil dan responsif.

2.11.2 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:

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

Dari sini terlihat:

- ketika $R \to \infty$, 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:

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

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:

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

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

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

kita kurangkan:

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

Maka:

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

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:

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

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:

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

2.11.3 Kecepatan Linear Roda dan Kecepatan Sudut Roda

Jika jari-jari roda adalah r_w, maka berlaku:

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

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

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

Hubungan-hubungan ini adalah dasar untuk mengonversi sinyal kontrol (v, ω) menjadi perintah fisik ke aktuator (motor roda) dalam bentuk kecepatan sudut roda.

3 Alur Kerja Program Simulasi Orientation dan Position Tracking

Program menjalankan *pipeline* berikut: *konfigurasi → koneksi simulator → loop kontrol orientasi & posisi → pencatatan data → terminasi → visualisasi hasil*.

1. **Konfigurasi Awal**
   - Menetapkan path objek scene di CoppeliaSim: `/PioneerP3DX`, `/rightMotor`, `/leftMotor`, `/Disc`, dan (opsional) sensor ultrasonik.
   - 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**:
     - `d_sw`: jarak karakteristik untuk *switching posisi–orientasi*,
     - `kp_lin_base`: gain dasar kecepatan linear,
     - `kp_ang_max`: gain maksimum untuk kontrol heading (positional orientation),
     - `kp_ori_max`: gain maksimum untuk orientasi akhir (orientation tracking).
   - Menentukan **durasi eksperimen** (mis. 30 detik) dan inisialisasi buffer data (`d_xyyaw`, `d_t`, `dat_disc2rob`, `dat_errors`).
2. **Koneksi ke CoppeliaSim**
   - 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 target: `/Disc`.

3. **Loop Kontrol Orientation & Position Tracking** Loop utama berjalan selama durasi eksperimen (mis. $t \leq 30$ 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 objek di world frame*:
   - posisi robot: `bod_pos_xyz` $= (x_b, y_b, z_b) =$ `getObjectPosition(P3DX, world)`,
   - orientasi robot: `bod_pos_abg` $= (\alpha_b, \beta_b, \gamma_b)$ dengan yaw robot $\theta_b = \gamma_b$,
   - posisi objek: `disc_pos_xyz` $= (x_d, y_d, z_d)$,
   - orientasi objek: `disc_pos_abg` $= (\alpha_d, \beta_d, \gamma_d)$ dengan yaw objek $\theta_d = \gamma_d$.

   (c) *Bentuk vektor homogen posisi objek di world*:
   $${}^{w}p_d = \begin{bmatrix} x_d \\ y_d \\ z_d \\ 1 \end{bmatrix}.$$

   (d) *Bentuk matriks transformasi world → body*:
   - Susun transformasi ${}^{w}T_b = T(\alpha_b, \beta_b, \gamma_b, x_b, y_b, z_b)$ dengan fungsi `transformMat`.
   - Ambil inversnya untuk memperoleh ${}^{b}T_w = ({}^{w}T_b)^{-1}$.

   (e) *Transformasi posisi objek ke frame robot*:
   $${}^{b}p_d = {}^{b}T_w \, {}^{w}p_d = \begin{bmatrix} x_{db} \\ y_{db} \\ z_{db} \\ 1 \end{bmatrix}.$$
   Di sini x_{db} dan y_{db} adalah koordinat objek dalam *frame robot* (body frame).

   (f) *Hitung galat posisi & orientasi*:

   i. **Error posisi sepanjang sumbu depan**:
   $$e_d = x_{db},$$
   sehingga objek diharap berada pada $x_{db} \to 0$ di depan robot.

   ii. **Error heading (arah ke objek)**:
   $$e_h = \arctan 2(y_{db}, x_{db}),$$
   dengan interpretasi: $e_h = 0$ berarti objek tepat di depan; tanda e_h menunjukkan sisi kiri/kanan.

   iii. **Error orientasi akhir (yaw)**:
   $$e_o = \text{wrapToPi}(\theta_d - \theta_b),$$
   yaitu perbedaan yaw objek dan yaw robot yang dinormalisasi ke rentang $(-\pi, \pi]$.

   iv. **Jarak Euclidean robot–objek di world**:
   $$d = \sqrt{(x_d - x_b)^2 + (y_d - y_b)^2},$$
   digunakan sebagai indikator kedekatan posisi.

   (g) *Hitung mode switching posisi–orientasi*:
   - Definisikan
   $$\text{mode} = \exp\left(-\frac{d}{d_{\text{sw}}}\right),$$
   sehingga:

  - $d \gg d_{\text{sw}} \Rightarrow$ mode ≈ 0 (fokus heading),
  - $d \approx 0 \Rightarrow$ mode ≈ 1 (fokus orientasi akhir).
- Tentukan gain kontrol:

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

$$k_{p,\text{ang}} = k_{\text{ang,max}}(1 - \text{mode}), \qquad k_{p,\text{ori}} = k_{\text{ori,max}}\,\text{mode}.$$

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

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

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

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

- Hubungan kinematika:

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

  sehingga diperoleh v_R dan v_L (m/s).
- Konversi ke kecepatan sudut roda:

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

  kemudian dapat diberikan *limit* jika diperlukan.

(j) *Aktuasi pada joint roda*:

- Mengirim perintah kecepatan sudut roda ke CoppeliaSim:

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

- Robot kemudian bergerak sesuai kombinasi translasi–rotasi yang ditentukan v dan ω.

(k) *Pencatatan data dan logging*:

- Menyimpan:
  - lintasan pusat robot di world: $(x_b(t), y_b(t), \theta_b(t))$,
  - posisi objek dalam frame robot ${}^{b}p_d(t)$,
  - galat $e_d(t), e_h(t), e_o(t)$,
  - waktu t_{now}.
- Mengirim *log* ke konsol CoppeliaSim berisi ringkasan nilai galat, jarak d, dan nilai `mode`.

4. **Terminasi Loop**

- Setelah waktu eksperimen melewati batas (mis. $t_{\text{now}} > 30$ 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 untuk dianalisis dan divisualisasikan.
- 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.

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

- menampilkan grafik e_d terhadap waktu t untuk melihat bagaimana jarak objek di sepanjang sumbu depan robot berkurang.

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

- menampilkan $e_h(t)$ dalam derajat terhadap waktu untuk mengamati proses penyelarasan arah robot terhadap objek.

(d) (Opsional) **Plot galat orientasi akhir** $e_o(t)$:

- dapat ditambahkan untuk menunjukkan bagaimana yaw robot mendekati yaw objek seiring waktu.

6. **Normalisasi Sudut dan Konsistensi Data**

- Sepanjang perhitungan, operasi pada sudut (terutama selisih yaw) menggunakan fungsi `wrapToPi`:

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

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.

4 Tutorial Penggunaan Program dan Analisis Output Program

4.1 Prasyarat dan Instalasi

1. **Python 3.x** telah terpasang dan dapat diakses dari terminal.
2. Pasang pustaka Python yang dibutuhkan:

```
pip install numpy matplotlib coppeliasim-zmqremoteapi-client
```

3. **CoppeliaSim (Edu/Pro)** terpasang dan dapat dijalankan.
4. Program *orientation and position tracking* tersimpan sebagai satu berkas Python.

4.2 Penyiapan Scene CoppeliaSim

1. Buka CoppeliaSim, muat scene yang berisi:
   - robot `/PioneerP3DX`,
   - objek target (misalnya disk) `/Disc`.
2. Pastikan hierarki/penamaan objek sesuai dengan program:
   - Base robot: `/PioneerP3DX`
   - Joint roda kanan: `/rightMotor`
   - Joint roda kiri: `/leftMotor`
   - Objek target: `/Disc`
3. Letakkan `/Disc` pada posisi dan orientasi yang diinginkan (misalnya beberapa meter di depan robot, dengan yaw tertentu).
4. Atur *simulation time step* sesuai kebutuhan (default scene CoppeliaSim biasanya sudah cukup untuk eksperimen ini).

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),

  yang biasanya diambil dari spesifikasi robot atau diukur dari scene.
- **Parameter switching jarak**:
  - `d_sw`: jarak karakteristik (meter) untuk *mode switching* antara fokus kontrol posisi dan orientasi akhir. Semakin kecil `d_sw`, semakin cepat kontrol berpindah fokus ke orientasi ketika robot mendekati objek.

- **Gain kontrol**:
  - `kp_lin_base`: gain dasar untuk kecepatan linear v,
  - `kp_ang_max`: gain maksimum untuk komponen heading e_h,
  - `kp_ori_max`: gain maksimum untuk komponen orientasi akhir e_o.

  Nilai-nilai ini dapat di-*tuning* agar respon robot tidak terlalu lambat namun tetap stabil.

- **Durasi eksperimen**:
  - Diatur pada pengecekan waktu, misalnya `if t_now > 30: break`, yang berarti program akan menjalankan kontrol selama ±30 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.
2. Buka terminal atau VS Code di folder tempat berkas Python disimpan.
3. Jalankan program, misalnya:

   ```
   python p3dx_pose_align.py
   ```

4. Program akan:
   - membuat koneksi ke CoppeliaSim,
   - memulai simulasi,
   - secara periodik membaca pose robot dan disk,
   - menghitung galat posisi–orientasi,
   - mengirim perintah kecepatan roda sehingga robot bergerak mendekati dan menyelaraskan diri dengan objek.
5. Setelah durasi eksperimen tercapai (mis. 30 detik) program:
   - menghentikan perintah gerak (kecepatan roda diset nol),
   - mengonversi data yang terekam ke dalam larik,
   - menampilkan jendela grafik hasil eksperimen.

4.5 Output yang Dihasilkan

Setelah loop kontrol selesai, program menampilkan beberapa **jendela grafik** yang meringkas perilaku sistem selama eksperimen. Sebagai contoh, program dasar ini 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 sekitar posisi objek `/Disc`. Bentuk lintasan menunjukkan kombinasi gerak maju dan berbelok yang dihasilkan oleh hukum kontrol. Kedekatan titik akhir lintasan terhadap posisi objek menandakan keberhasilan robot untuk mencapai target secara posisi.

2. **Jendela 2: Kurva galat jarak $e_d(t)$**
   - Menampilkan galat posisi sepanjang sumbu depan robot:

     $$e_d(t) = x_{db}(t)$$

     dalam satuan meter.

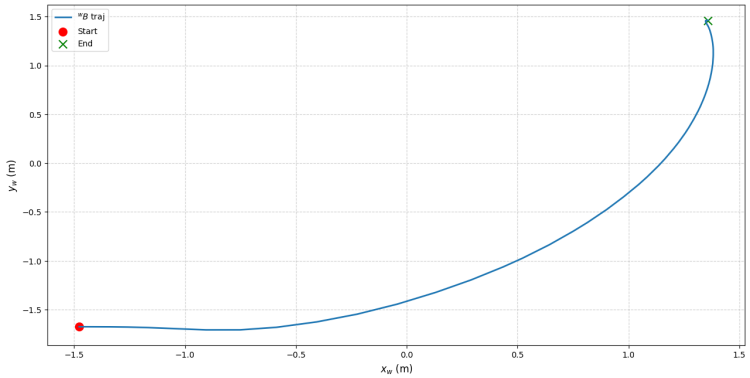

Figure 1: Lintasan robot P3DX pada bidang x_w–y_w selama proses pose alignment

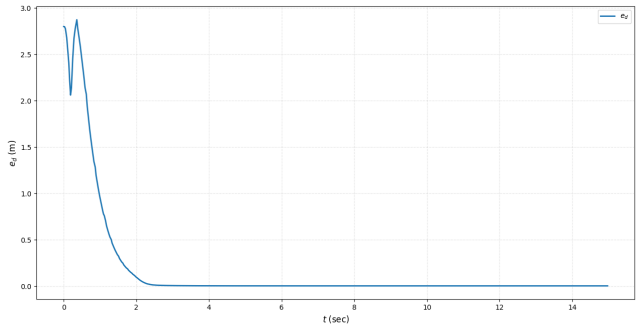

Figure 2: Galat jarak $e_d(t)$ antara robot dan objek sepanjang sumbu depan robot

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

Berdasarkan Gambar 2, nilai $e_d(t)$ yang **semakin mendekati nol** seiring waktu menunjukkan bahwa robot berhasil mengurangi jarak ke objek di sepanjang sumbu depan. Jika kurva berosilasi atau tidak konvergen, hal ini dapat menjadi indikasi perlunya penyesuaian parameter kontrol (`kp_lin_base` atau `d_sw`).

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

- Menampilkan galat heading:

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

  yang biasanya ditampilkan dalam derajat untuk memudahkan interpretasi.
- Sumbu horizontal menyatakan waktu t, sedangkan sumbu vertikal menyatakan e_h (derajat).

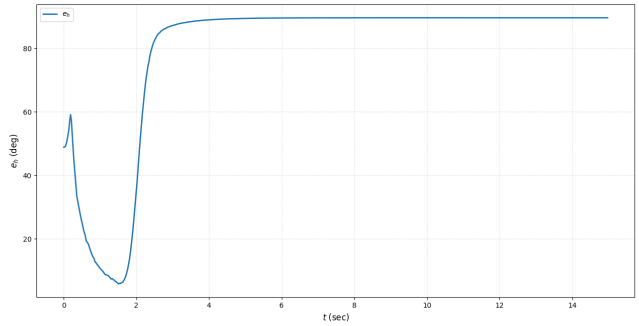

Figure 3: Galat heading $e_h(t)$ antara arah depan robot dan arah ke objek

Berdasarkan Gambar 3, nilai $e_h(t)$ yang bergerak menuju nol menunjukkan bahwa robot secara bertahap memutar badan sehingga sumbu depan robot segaris dengan arah menuju objek. Pada fase awal, ketika robot masih jauh, kontrol lebih menekankan pada pengurangan e_h. Ketika robot semakin dekat, peran e_o (galat orientasi akhir) dapat diperbesar (jika divisualisasikan) untuk menyelaraskan yaw robot dengan yaw objek.

4.6 Ringkasan Interpretasi Output

Secara umum, dari ketiga jendela grafik di atas, pengguna dapat menarik beberapa kesimpulan berikut:

- **Lintasan** x–y (Gambar 1) menunjukkan jalur fisik yang ditempuh robot menuju objek.
- **Kurva** $e_d(t)$ (Gambar 2) mengindikasikan keberhasilan robot dalam mengurangi jarak ke objek di sepanjang sumbu depan robot (kontrol posisi).
- **Kurva** $e_h(t)$ (Gambar 3) menggambarkan proses penyelarasan arah badan robot terhadap arah objek (kontrol heading).

Jika ketiga grafik menunjukkan konvergensi (lintasan mendekati objek, $e_d(t)$ dan $e_h(t)$ mendekati nol), maka program *orientation and position tracking* dapat dikatakan bekerja dengan baik untuk konfigurasi scene dan parameter kontrol yang digunakan.

5 Kesimpulan

Berdasarkan perancangan, implementasi, dan pengujian program simulasi *orientation and position tracking* (pose alignment) pada robot P3DX di CoppeliaSim, dapat disimpulkan bahwa:

1. **Tujuan pose alignment antara frame robot dan objek tercapai.**
   Dengan masukan berupa sinyal galat posisi e_d (jarak sepanjang sumbu depan), galat heading e_h (arah ke objek dalam frame robot), serta galat orientasi akhir e_o (selisih yaw objek dan robot), implementasi loop kontrol pada program mampu menggerakkan robot dari posisi awalnya menuju objek `/Disc` sekaligus memutar badan sehingga orientasi robot mendekati orientasi objek. Hal ini terlihat dari lintasan robot yang berakhir di sekitar posisi objek dan dari kurva galat yang cenderung konvergen menuju nol.

2. **Transformasi koordinat berbasis matriks homogen telah berhasil diterapkan.**
   Penggunaan matriks transformasi homogen ${}^{w}T_{b}$ dan inversnya ${}^{b}T_{w}$ untuk memetakan posisi objek dari *world frame* ke *frame robot* terbukti efektif dan konsisten. Representasi titik dalam bentuk vektor homogen memudahkan komposisi rotasi dan translasi, serta menjadi landasan utama untuk mendefinisikan galat dalam koordinat robot, sesuai dengan tujuan untuk mempraktikkan transformasi koordinat dalam konteks kontrol pose.

3. **Kontrol sederhana berbasis P-controller mampu menghasilkan perilaku tracking yang intuitif.**
   Kontrol proporsional berupa :

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

   terbukti cukup untuk menghasilkan gerak kombinasi translasi–rotasi yang baik, dimana robot maju ketika objek berada di depan dan berputar ke arah objek ketika terdapat galat heading atau orientasi. Dengan *tuning* gain yang tepat, sistem menunjukkan respon yang stabil dan memenuhi tujuan untuk mengimplementasikan kontrol pose differential-drive secara praktis.

4. **Mekanisme mode switching eksponensial efektif mengatur fokus kontrol.**
   Fungsi mode

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

   yang menginterpolasi kontribusi kontrol heading dan orientasi akhir memungkinkan sistem berpindah fokus secara *halus*: ketika jauh, kontrol lebih menekankan perbaikan heading (e_h), dan ketika sudah dekat, kontrol lebih menekankan penyelarasan orientasi akhir (e_o). Dengan demikian, tujuan untuk menunjukkan dan memanfaatkan *mode switching* dalam kontrol pose telah tercapai.

5. **Pipeline simulasi dari akuisisi data hingga visualisasi berjalan sesuai rancangan.**
   Seluruh *pipeline* program—mulai dari konfigurasi parameter, koneksi ke CoppeliaSim, pembacaan pose robot dan objek, perhitungan transformasi koordinat dan galat, implementasi kontrol, hingga pencatatan dan visualisasi data—berjalan sesuai alur yang direncanakan. Lintasan x–y, serta kurva $e_d(t)$ dan $e_h(t)$ memberikan gambaran yang jelas tentang proses konvergensi pose, sehingga tujuan edukatif proyek (memahami hubungan antara kinematika, transformasi koordinat, kontrol, dan visualisasi) dapat tercapai.

- Link Github :

  https://github.com/amrufal/orientation-and-position-tracking-p3dx

- Linkedin :

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