Tugas 5 Sistem Robot Otonom: Eksperimen Transformasi Koordinat Pada Data Odometri Wheeled Mobile Robot P3DX

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

1 Pendahuluan

1.1 Latar Belakang

Odometri adalah metode estimasi pose robot (x, y, yaw) dengan mengintegrasikan kecepatan roda dari waktu ke waktu. Untuk robot differential-drive, kecepatan linear dan angular dihitung dari kecepatan sudut roda kiri–kanan:

$$v = \tfrac{1}{2}(v_r + v_l), \qquad \omega = \frac{v_r - v_l}{2\,L_{\text{half}}}, \quad v_{r,l} = R\,\omega_{r,l}.$$

Pose diperbarui diskrit per langkah waktu Δt:

$$x \leftarrow x + v\cos\theta\,\Delta t, \quad y \leftarrow y + v\sin\theta\,\Delta t, \quad \theta \leftarrow \theta + \omega\,\Delta t.$$

Odometri berlatensi rendah dan sederhana, namun rentan drift (slip, kesalahan parameter R, L_{half}, dan akumulasi numerik). Data yang diperoleh pada eksperimen ini meliputi lintasan $\{x(t), y(t), \text{yaw}(t)\}$ *(ODO)* dan *ground-truth* (GT) dari simulator untuk pembanding.

Karena robot bekerja pada beberapa *frame* (world, base, sensor), data pose perlu diolah dengan **transformasi koordinat** berbasis matriks homogen 4×4 agar rotasi dan translasi dapat dikomposisikan secara sistematis. Hal penting yang ditekankan adalah **non-komutativitas**: Rotasi → Translasi menghasilkan lintasan berbeda dibanding Translasi → Rotasi.

Proyek/eksperimen ini dirancang sebagai **sarana edukasi** untuk menunjukkan bagaimana data odometri diolah dengan transformasi koordinat dan mengapa urutan operasi memengaruhi hasil.

1.2 Tujuan

1. Mengimplementasikan odometri *differential-drive* untuk mengestimasi pose (x, y, yaw) dan membandingkannya dengan *ground-truth* (GT) yang telah disejajarkan pada frame awal.
2. Menerapkan **transformasi koordinat** berbasis matriks homogen 4×4 pada lintasan odometri untuk tiga skenario edukatif: (E1) rotasi sebesar θ, (E2) Rotasi → Translasi, dan (E3) Translasi → Rotasi.
3. **Menjelaskan dan membuktikan non-komutativitas** rotasi dan translasi dengan **membandingkan** hasil lintasan E2 (R→T) vs E3 (T→R) baik secara visual maupun melalui metrik kuantitatif.
4. Menyajikan **visualisasi multi-jendela** (ODO vs GT; kurva galat; XY tiap eksperimen dengan overlay ODO; overlay seluruh lintasan) sebagai sarana edukasi.
5. Menyediakan **konfigurasi yang mudah** (parameter `ROT_DEG`, `TX`, `TY`, dan mode *stepping*) agar eksperimen bersifat praktis.

1.3 Fitur Proyek

Pada proyek simulasi metode odometri ini, program diharapkan dapat:

1. **Membaca parameter geometri secara otomatis** dari scene simulasi, seperti *jari-jari roda* dan *jarak pusat roda ke pusat robot* (half-track), serta *time step* simulasi.
2. **Mencatat data** posisi dan pose *ground-truth* (GT) robot selama simulasi berlangsung.
3. **Melakukan proses odometri** (integrasi kecepatan roda) dan **mencatat hasil perkiraan** posisi serta pose (ODO) selama simulasi berlangsung.
4. **Menampilkan visualisasi multi-jendela** saat simulasi dihentikan:

- **Jendela 1**: ODO vs GT — grafik $x(t)$, $y(t)$, yaw(t), dan lintasan x–y.
- **Jendela 2**: Kurva galat $e_x(t)$, $e_y(t)$, $e_{\text{yaw}}(t)$.
- **Jendela 3**: Lintasan hasil *eksperimen transformasi* (E1/E2/E3) yang dioverlay dengan lintasan ODO asli.
- **Jendela 4**: Overlay seluruh lintasan (GT, ODO, E1–E3) pada satu grafik.

5. **Menerapkan transformasi koordinat** berbasis matriks homogen 4×4 pada lintasan ODO dengan tiga skenario edukatif:
   - **E1**: Rotasi sebesar θ (mis. 90°).
   - **E2**: *Rotasi → Translasi.*
   - **E3**: *Translasi → Rotasi.*

   Skenario ini memperlihatkan **non-komutativitas** urutan operasi (R→T ≠ T→R).
6. **Konfigurasi mudah** melalui parameter `ROT_DEG` (sudut), `TX`, `TY` (translasi), dan mode *stepping*.

1.4 Software yang Dibutuhkan dan Fungsinya

Adapun beberapa software yang dibutuhkan untuk membuat proyek ini antara lain:

1. **Python**: Bahasa pemrograman dan interpreter utama.
2. **CoppeliaSim Edu**: Perangkat lunak simulasi yang menyediakan model robot (P3DX), dinamika, dan *ground-truth* pose.
3. **ZMQ Remote API (Python client)**: Antarmuka untuk membaca informasi (handle objek, waktu simulasi, kecepatan joint) dan memberi perintah pada CoppeliaSim dari Python.
4. **NumPy**: Komputasi numerik (vektor/matriks), termasuk pembentukan dan komposisi matriks homogen 4×4.
5. **Matplotlib**: Visualisasi data untuk kurva waktu dan lintasan x–y pada beberapa jendela.

1.5 Tantangan

Dalam pengerjaan proyek ini, terdapat beberapa tantangan yang dihadapi, antara lain:

1. Menentukan **algoritma** dan **urutan proses** yang tepat agar program memenuhi tujuan proyek (akuisisi data, odometri, transformasi, visualisasi).
2. Memilih dan menggunakan **perintah API CoppeliaSim** yang tepat (pembacaan handle, waktu simulasi, kecepatan joint).
3. **Penjajaran frame**: menyelaraskan GT ke frame awal robot agar perbandingan ODO vs GT adil.
4. **Stabilitas sudut**: normalisasi yaw ke rentang $[-\pi, \pi)$ untuk menghindari diskontinuitas.
5. **Non-komutativitas transformasi**: memastikan implementasi Rotasi→Translasi vs Translasi→Rotasi sesuai dengan yang diharapkan dari eksperimen.

1.6 Kemampuan yang Dipelajari

1. Penggunaan CoppeliaSim untuk membangun **lingkungan simulasi** sederhana (robot P3DX dan arena).
2. Pemrograman Python dan **ZMQ Remote API** untuk **membaca data** simulasi dan **mengontrol** jalannya simulasi.
3. **Implementasi odometri** differential-drive serta evaluasi terhadap *ground-truth*.
4. **Transformasi koordinat** dengan matriks homogen 4×4 (rotasi, translasi, komposisi) dan analisis **urutan operasi** (R→T vs T→R).
5. **Visualisasi multi-jendela** untuk diagnosis, analisis galat, dan dokumentasi hasil eksperimen.
6. (Opsional) **Pengelolaan kode** dan dokumentasi proyek menggunakan Git/GitHub.

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 Ground Truth dan Proyeksi ke Kerangka Lokal

Dalam simulasi CoppeliaSim, *ground truth* (GT) adalah posisi dan orientasi aktual robot di dalam kerangka dunia (world frame). Agar dapat dibandingkan langsung dengan odometri (yang selalu diasumsikan mulai dari $(0, 0, 0)$), maka pose GT perlu diproyeksikan ke kerangka lokal awal robot. Transformasi ini dilakukan dengan translasi dan rotasi menggunakan matriks rotasi $R(-\theta_0)$, di mana θ_0 adalah orientasi awal robot:

$$\begin{bmatrix} x_{gt} \\ y_{gt} \end{bmatrix} = \begin{bmatrix} \cos\theta_0 & \sin\theta_0 \\ -\sin\theta_0 & \cos\theta_0 \end{bmatrix} \begin{bmatrix} x_w - x_0 \\ y_w - y_0 \end{bmatrix},$$

$$\theta_{gt} = \theta_w - \theta_0.$$

2.8 Normalisasi Sudut

Karena operasi integrasi dan pengurangan sudut dapat menghasilkan nilai di luar rentang $[-\pi, \pi)$, maka digunakan fungsi pembungkus sudut:

$$\theta_{\text{wrap}} = \text{atan2}(\sin\theta, \cos\theta).$$

Dengan cara ini, nilai orientasi robot selalu konsisten dalam rentang $[-180°, 180°)$.

2.9 Vektor dan Matriks

Posisi titik di ruang tiga dimensi dinyatakan sebagai vektor kolom

$$\mathbf{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}.$$

Matriks adalah susunan bilangan berbentuk persegi panjang yang dapat mewakili transformasi linier. Jika A adalah matriks yang sesuai dimensinya, maka hasil transformasi adalah $A\,\mathbf{p}$.

2.10 Kerangka Acuan, Orientasi, dan Pose

Kerangka acuan (frame) adalah sistem koordinat ortonormal (sumbu x, y, z saling tegak lurus) dengan sebuah titik asal. **Pose** suatu frame U relatif terhadap frame B terdiri dari:

1. **Orientasi**—dinyatakan oleh matriks rotasi 3×3 (kolom-kolomnya ortonormal dan determinannya $+1$).
2. **Posisi**—dinyatakan oleh vektor translasi $\mathbf{t} = \begin{bmatrix} t_x & t_y & t_z \end{bmatrix}^\top$.

Jika $\mathbf{p}$ adalah koordinat titik pada frame U, maka koordinat titik yang sama pada frame B adalah

$$\mathbf{p}_B = R\,\mathbf{p}_U + \mathbf{t},$$

dengan R matriks rotasi (orientasi U relatif terhadap B) dan $\mathbf{t}$ vektor translasi asal U relatif terhadap B.

2.11 Translasi

Translasi memindahkan setiap titik sejauh vektor tertentu. Jika $\mathbf{t} = \begin{bmatrix} t_x & t_y & t_z \end{bmatrix}^\top$, maka

$$\mathbf{p}' = \mathbf{p} + \mathbf{t}.$$

2.12 Matriks Rotasi (3D)

Rotasi mengubah *orientasi* vektor/titik terhadap sumbu tertentu. Tiga matriks rotasi elementer (konvensi tangan kanan) adalah:

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}, \quad R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}, \quad R_z(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

2.13 Koordinat Homogen

Agar *rotasi dan translasi* dapat digabungkan dalam **satu** operasi, kita menambahkan komponen keempat w:

$$\tilde{\mathbf{p}} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

Dengan bentuk ini, baik rotasi maupun translasi bisa diterapkan sebagai satu perkalian matriks 4×4 terhadap vektor homogen $\tilde{\mathbf{p}}$.

2.14 Matriks Transformasi Homogen

Gabungan rotasi R dan translasi $\mathbf{t}$ ditulis sebagai:

$$T = \begin{bmatrix} R & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \quad \text{(ukuran } 4 \times 4\text{)}.$$

Jika titik ditulis homogen $\tilde{\mathbf{p}} = \begin{bmatrix} x & y & z & 1 \end{bmatrix}^\top$, maka hasil transformasinya

$$\tilde{\mathbf{p}}' = T\,\tilde{\mathbf{p}} \quad \Longrightarrow \quad \mathbf{p}' = R\,\mathbf{p} + \mathbf{t}.$$

3 Alur Kerja Program Simulasi Odometri dan Transformasi Koordinat

Program menjalankan pipeline berikut: *konfigurasi → koneksi simulator → akuisisi odometri & GT → perhitungan galat → eksperimen transformasi (E1–E3) → visualisasi multi-jendela.*

1. **Konfigurasi Awal**
   - Menetapkan path objek scene (`/PioneerP3DX`, `/rightMotor`, `/leftMotor`, dll.).
   - Menentukan mode *stepping* dan parameter eksperimen: sudut rotasi `ROT_DEG`, translasi `TX`, `TY`.
2. **Koneksi ke CoppeliaSim**
   - Membuat klien ZMQ: `RemoteAPIClient()` dan mengambil modul `sim`.
   - Mengaktifkan *stepping* (`sim.setStepping(True)`) lalu `sim.startSimulation()`.
3. **Baca Parameter Geometri & Waktu Scene**
   - Mengambil *handle* base dan joint roda.
   - Menghitung **half-track** $L_{\text{half}} = \frac{1}{2}|y_R - y_L|$ dari posisi joint pada frame base.
   - Mengambil **jari-jari roda** R dari geometri silinder roda (*diameter*/2).
   - Menyimpan **time step** scene untuk referensi.
4. **Loop Akuisisi Odometri & GT (hingga *stop*)**
   (a) *Kontrol status simulasi*: bila *paused* ⇒ tunggu; bila *running* ⇒ lanjut. Pada mode *stepping* dilakukan `sim.step()` per iterasi.
   (b) *Baca kecepatan joint* ω_r, ω_l (rad/s), lalu

$$v_r = R\omega_r, \quad v_l = R\omega_l, \qquad v = \tfrac{1}{2}(v_r + v_l), \quad \omega = \frac{v_r - v_l}{2L_{\text{half}}}.$$

   (c) *Integrasi odometri* untuk keadaan (x_o, y_o, θ_o):

$$x_o \leftarrow x_o + v\cos\theta_o\,\Delta t, \quad y_o \leftarrow y_o + v\sin\theta_o\,\Delta t, \quad \theta_o \leftarrow \text{wrap}(\theta_o + \omega\,\Delta t).$$

   (d) *Baca GT dunia* (x_w, y_w, θ_w), lalu **proyeksikan ke frame awal** $(x_{gt}, y_{gt}, \theta_{gt})$ agar sebanding dengan ODO:

$$\begin{bmatrix} x_{gt} \\ y_{gt} \end{bmatrix} = \begin{bmatrix} \cos\theta_0 & \sin\theta_0 \\ -\sin\theta_0 & \cos\theta_0 \end{bmatrix} \begin{bmatrix} x_w - x_0 \\ y_w - y_0 \end{bmatrix}, \qquad \theta_{gt} = \text{wrap}(\theta_w - \theta_0).$$

(e) *Hitung galat* setiap sampel: $e_x = x_o - x_{gt}$, $e_y = y_o - y_{gt}$, $e_\theta = \text{wrap}(\theta_o - \theta_{gt})$.

(f) *Simpan histori* waktu, ODO, GT, dan galat ke dalam larik untuk visualisasi akhir.

5. **Terminasi Loop**
   - Ketika status simulasi menjadi `simulation_stopped`, loop dihentikan dan `sim.stopSimulation()` dipanggil secara aman.
6. **Eksperimen Transformasi Lintasan (E1–E3)**
   - Bentuk vektor homogen tiap titik ODO: $\tilde{\mathbf{p}} = [x\ y\ 0\ 1]^\top$.
   - **E1 (Rotasi saja)**: $\tilde{\mathbf{p}}' = R_z(\texttt{ROT\_DEG})\,\tilde{\mathbf{p}}$.
   - **E2 (Rotasi → Translasi)**: hitung E1 lalu geser $(+\texttt{TX}, +\texttt{TY})$ pada (x', y').
   - **E3 (Translasi → Rotasi)**: geser $(x + \texttt{TX}, y + \texttt{TY})$ lalu kalikan $R_z(\texttt{ROT\_DEG})$.
   - Catatan: rotasi & translasi **tidak komutatif**, sehingga E2 ≠ E3.
7. **Visualisasi Multi-Jendela**

(a) **Jendela 1 (ODO vs GT)**: $x(t)$, $y(t)$, yaw(t), dan lintasan x–y (ODO vs GT).

(b) **Jendela 2 (Error)**: kurva $e_x(t)$, $e_y(t)$, $e_{\text{yaw}}(t)$.

(c) **Jendela 3 (Eksperimen per-XY)**: tiga subplot (E1, E2, E3) masing-masing di-*overlay* dengan lintasan ODO asli.

(d) **Jendela 4 (Overlay semua lintasan)**: ODO, GT, E1, E2, E3 pada satu grafik untuk perbandingan global.

8. **Normalisasi Sudut**
   - Setiap operasi pada sudut menggunakan `wrap_pi`: atan2$(\sin\theta, \cos\theta)$ agar $\theta \in [-\pi, \pi)$ dan menghindari lompatan $\pm\pi$.

4 Tutorial Penggunaan Program dan Analisis Output Program

4.1 Prasyarat dan Instalasi

1. **Python 3.x** terpasang.
2. Pasang pustaka yang dibutuhkan:

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

3. **CoppeliaSim (Edu/Pro)** terpasang dan dapat dijalankan.

4.2 Penyiapan Scene CoppeliaSim

1. Buka CoppeliaSim, muat scene yang berisi robot `PioneerP3DX`.
2. Pastikan hierarki/penamaan objek sesuai dengan program:
   - Base robot: `/PioneerP3DX`
   - Joint kanan: `/rightMotor`
   - Joint kiri: `/leftMotor`
   - Bentuk roda kanan: `/rightMotor/rightWheel` (tipe *primitive cylinder*)
3. Atur *simulation time step* sesuai kebutuhan (default scene juga bisa).

4.3 Konfigurasi Program

Simpan berkas program (satu file) lalu atur parameter di bagian *konfigurasi*:

- `ROT_DEG`: sudut rotasi eksperimen (derajat), mis. `90.0`.
- `TX`, `TY`: komponen translasi untuk eksperimen (meter), mis. `2.0`, `3.0`.
- `STEPPING`: `True` (*Python mengayuh simulasi* per langkah) atau `False`.

4.4 Menjalankan Program

1. Jalankan CoppeliaSim (tidak perlu menekan *Play*; program akan memulai simulasi).
2. Dari terminal/VS Code, jalankan program.
3. Biarkan simulasi berjalan.
4. **Hentikan simulasi** dari CoppeliaSim (*Stop*). Ketika status *stopped*, program akan otomatis menampilkan jendela-jendela grafik.

4.5 Output yang Dihasilkan

Setelah simulasi dihentikan, program menampilkan **empat jendela**:

1. **Jendela 1 (ODO vs GT)**: empat subplot
   - $x(t)$: kurva posisi x hasil odometri (ODO) dan *ground-truth* (GT).
   - $y(t)$: kurva posisi y ODO dan GT.
   - yaw(t): orientasi (derajat) ODO dan GT.
   - Lintasan x–y: grafik jejak ODO dan GT pada bidang.

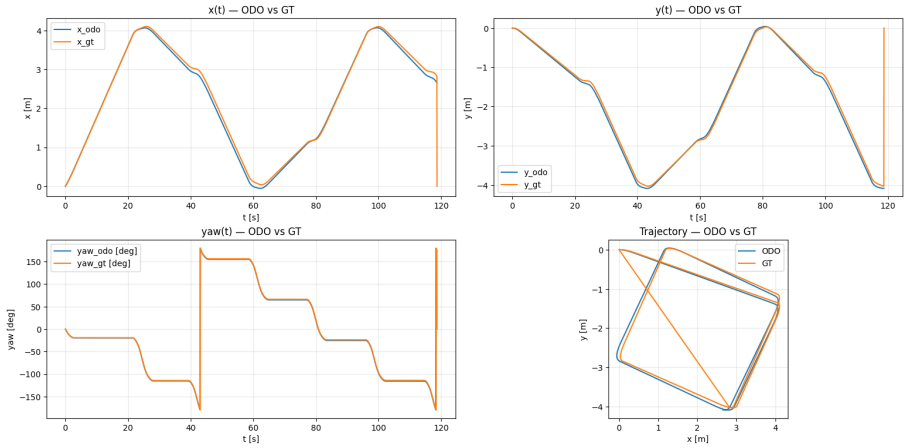

Figure 1: Odometri vs Ground Truth

Berdasarkan figure 1, kedekatan kurva ODO terhadap GT menandakan metode odometri menghasilkan perkiraan lintasan robot yang baik. Selisih visual menunjukkan drift/penyimpangan.

2. **Jendela 2 (Error)**: tiga subplot
   - $e_x(t) = x_{\text{odo}} - x_{\text{gt}}$ (meter).
   - $e_y(t) = y_{\text{odo}} - y_{\text{gt}}$ (meter).
   - $e_{\text{yaw}}(t) = \text{yaw}_{\text{odo}} - \text{yaw}_{\text{gt}}$ (derajat).

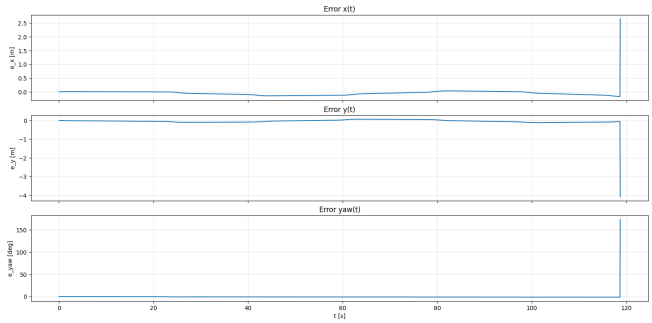

Figure 2: Odometri vs Ground Truth

Berdasarkan figure 2, nilai mendekati nol menandakan odometri memiliki akurasi yang baik.

3. **Jendela 3 (Eksperimen per-XY)**: tiga subplot
    - **E1**: Rotasi `ROT_DEG` pada lintasan ODO, dioverlay dengan ODO asli (garis putus).
    - **E2**: **Rotasi** → **Translasi** (+`TX`, +`TY`).
    - **E3**: **Translasi** → **Rotasi** (+`TX`, +`TY`).

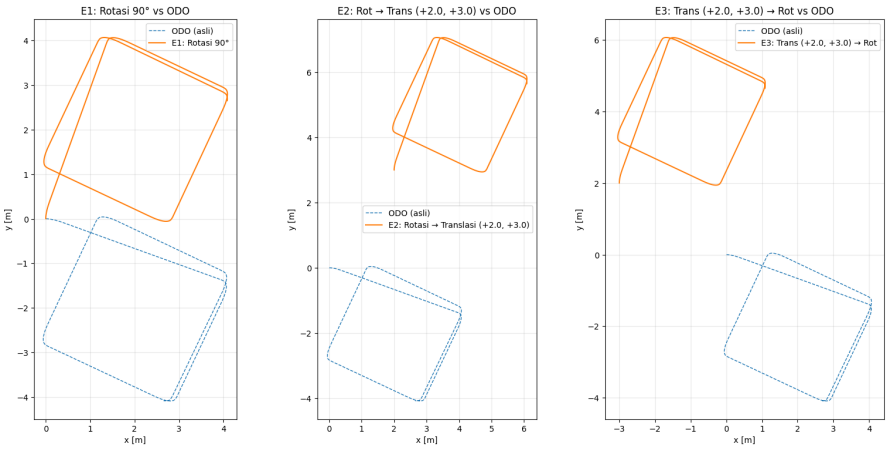

Figure 3: Odometri vs Ground Truth

Berdasarkan figure 3, bentuk lintasan pada eksperimen 2 (E2) dan eksperimen 3 (E3) berbeda karena setiap eksperimen memiliki urutan operasi rotasi–translasi yang berbeda. Hal ini menunjukkan bahwa operasi translasi dan rotasi **tidak bersifat komutatif**.

4. **Jendela 4 (Overlay semua lintasan)**:
    - Satu grafik yang menumpuk ODO, GT, E1, E2, E3.

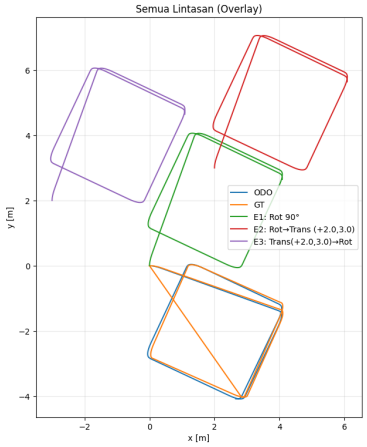

Figure 4: Odometri vs Ground Truth

Berdasarkan figure 4, pengguna dapat mengamati pengaruh urutan operasi transformasi koordinat pada tiap eksperimen terhadap data/bentuk lintasan odometri dalam grafik yang sama.

5 Kesimpulan

Berdasarkan implementasi dan pengujian yang telah dilakukan, dapat disimpulkan bahwa:

1. Program berhasil terhubung ke CoppeliaSim melalui ZMQ Remote API, **membaca parameter fisik** (jari–jari roda, half–track, dan *time step*) langsung dari scene, menjalankan **odometri differential–drive** selaras waktu simulasi, serta mengambil **ground truth** (GT) dan memproyeksikannya ke kerangka lokal awal sebagai perbandingan.

2. Mekanisme **visualisasi multi-jendela** berjalan baik: kurva $x(t), y(t), \text{yaw}(t)$ dan lintasan x–y (ODO vs GT) menunjukkan kedekatan estimasi terhadap GT.

3. Lintasan pada Eksperimen 2 dan 3 berbeda karena urutan operasi yang berbeda, hal ini sesuai dengan penjelasan berikut :

$$T(\mathbf{t}) = \begin{bmatrix} I & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix}, \quad \tilde{\mathbf{p}} = \begin{bmatrix} x & y & z & 1 \end{bmatrix}^\top, \quad \tilde{\mathbf{p}}' = \begin{bmatrix} R & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \tilde{\mathbf{p}}.$$

Dua urutan yang diuji menghasilkan:

$$\underbrace{T(\mathbf{t})\,R}_{\text{Rotasi}\to\text{Translasi}} \quad \tilde{\mathbf{p}} = R\,\tilde{\mathbf{p}} + \mathbf{t}, \qquad \underbrace{R\,T(\mathbf{t})}_{\text{Translasi}\to\text{Rotasi}} \quad \tilde{\mathbf{p}} = R\,\tilde{\mathbf{p}} + \underbrace{R\,\mathbf{t}}_{\text{translasi ikut berotasi}}.$$

Perbedaannya *independen titik*:

$$\Delta = \big(R\,T(\mathbf{t}) - T(\mathbf{t})\,R\big)\,\tilde{\mathbf{p}} = (R - I)\mathbf{t} \neq \mathbf{0} \quad \text{(umumnya)}.$$

- **Interpretasi kerangka acuan:** pada Rotasi→Translasi, translasi $\mathbf{t}$ diterapkan pada *frame awal* setelah rotasi. Sedangkan pada Translasi→Rotasi, translasi awalnya diterapkan pada *frame awal*, lalu operasi rotasi membuat vektor translasi ikut berputar, menyebabkan translasi menjadi bernilai $R\mathbf{t}$ dengan frame awal sebagai kerangka acuan.
- **Konsekuensi geometris:** kedua hasil eksperimen adalah lintasan odometri asli yang sama–sama telah diputar oleh R (sehingga bentuk intrinsik/shape tetap), tetapi **letaknya berbeda** karena offset translasi berbeda ($\mathbf{t}$ vs $R\mathbf{t}$). Inilah sebabnya lintasan E2 dan E3 tampak berbeda pada plot bidang xy.
- **Kapan komutatif?** $R\,T(\mathbf{t}) = T(\mathbf{t})\,R \Leftrightarrow R\mathbf{t} = \mathbf{t}$, yang hanya terjadi pada kasus khusus: $\mathbf{t} = \mathbf{0}$, atau $R = I$ (rotasi nol), atau (3D) $\mathbf{t}$ sejajar sumbu tetap rotasi R.
- **Contoh 2D:** untuk $R = R_z(90°)$ dan $\mathbf{t} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$,

$$R\mathbf{t} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \end{bmatrix} = \begin{bmatrix} -3 \\ 2 \end{bmatrix}, \quad \Delta = (R - I)\mathbf{t} = \begin{bmatrix} -5 \\ -1 \end{bmatrix} \neq \mathbf{0},$$

sehingga E2 (R→T) dan E3 (T→R) **berbeda letak** pada bidang xy.

- Link Github :

  https://github.com/amrufal/transformasi_koordinat_odometri

- Linkedin :

  https://www.linkedin.com/posts/falah-amru-9a192a385_program-ini-mensimulasikan-odometri-robot-acti
  utm_source=share&utm_medium=member_desktop&rcm=ACoAAF7_5wkBHEm5eAEQ0al5sPz8ty6cgqgTe08