

AAE 590LGM: Lie Group Methods
Problem Set 3

Travis Hastreiter

2 March, 2026

Contents

P1) Normal Subgroups and Quotient Groups	3
1.a) Definition and Intuition:	3
1.b) A Matrix Group Example:	3
1.c) First Isomorphism Theorem (Preview):	3
P2) SE(2) and the Semi-Direct Product	4
2.a) Matrix Representation:	4
P3) SE(2) Lie Algebra, Exp/Log, and Adjoint	5
3.a) Lie Algebra Structure:	5
3.b) Exponential Map:	5
3.c) Logarithm Map:	5
3.d) Adjoint Representation Ad_X :	6
3.e) Lie Bracket and ad_ξ :	6
Implementation and Unit Tests	7
P4) SE(2) Kinematics Simulation	11
4.a)	11
4.b)	11

Code Listing

1	SE(2) and $\mathfrak{se}(2)$ Function Implementations	7
2	Unit Tests for SE(2) Using Pytest	9
3	SE(2) Integrator Implementation	11
4	SE(2) Integrator Test	12

P1) Normal Subgroups and Quotient Groups

Before diving into $SE(2)$, let's understand why normal subgroups matter.

1.a) Definition and Intuition:

Question:

A subgroup $N \leq G$ is **normal** (written $N \trianglelefteq G$) if $gNg^{-1} = N$ for all $g \in G$.

- Explain why every subgroup of an abelian group is automatically normal.
- For non-abelian groups, conjugation can "twist" a subgroup. Give geometric intuition: if H is a set of translations and g is a rotation, what does gHg^{-1} represent?

Solution:

1.b) A Matrix Group Example:

Question:

Consider the determinant map $\det : GL(2, \mathbb{R}) \rightarrow \mathbb{R}^*$ (where $\mathbb{R}^* = \mathbb{R} \setminus \{0\}$ under multiplication).

- Verify \det is a homomorphism: $\det(AB) = \det(A)\det(B)$.
- What is $\ker(\det)$? (This group has a name—what is it?)
- What information about a matrix is "forgotten" when we apply \det ? What's preserved?

Solution:

1.c) First Isomorphism Theorem (Preview):

Question:

For a homomorphism $\phi : G \rightarrow H$:

- **Kernel:** $\ker(\phi) = \{g \in G : \phi(g) = e_H\}$ (elements mapped to identity)
- **Image (Range):** $\text{im}(\phi) = \{\phi(g) : g \in G\}$ (elements that ϕ "hits")

First Isomorphism Theorem: $G/\ker(\phi) \cong \text{im}(\phi)$.

In words: quotienting by what ϕ "kills" gives you what ϕ "sees."

- For $SE(2)$: if we define $\pi : SE(2) \rightarrow SO(2)$ by $\pi(\mathbf{t}, R) = R$, what is $\ker(\pi)$?
- What does the First Isomorphism Theorem tell us about $SE(2)/\ker(\pi)$?

Solution:

P2) SE(2) and the Semi-Direct Product

The group SE(2) of planar rigid motions is our first example of a **semi-direct product**. This structure—where one subgroup “twists” another—is fundamental to robotics.

The Translation Group $\mathbb{T}(2)$: The set of 2D translations forms a group $\mathbb{T}(2)$ under composition. As a matrix Lie group:

$$\mathbb{T}(2) = \left\{ \begin{bmatrix} I & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} : \mathbf{t} \in \mathbb{R}^2 \right\} \cong (\mathbb{R}^2, +)$$

This group is abelian (translations commute) and isomorphic to \mathbb{R}^2 with vector addition. Its Lie algebra is $\mathfrak{t}(2) \cong \mathbb{R}^2$.

2.a) Matrix Representation:

Question:

Write $X \in \text{SE}((2))$ as a 3×3 matrix:

$$X = \begin{bmatrix} R & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}, \quad R \in \text{SO}(2), \mathbf{t} \in \mathbb{R}^2$$

Tuple notation: We can also write $X = (\mathbf{t}, R)$ as a compact tuple. The correspondence is:

$$(\mathbf{t}, R) \longleftrightarrow \begin{bmatrix} R & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}$$

We order as (\mathbf{t}, R) (translation first) to match the semi-direct product $\mathbb{T}(2) \rtimes \text{SO}(2)$ convention. Both notations represent the same rigid motion: rotate by R , then translate by \mathbf{t} (in the world frame).

- Derive the composition $X_1 X_2$ and inverse X^{-1} formulas using block matrix multiplication. You don’t need to expand 2×2 blocks—leave products like $R_1 R_2$ as is.
- Express your results in tuple form: $(\mathbf{t}_1, R_1) \cdot (\mathbf{t}_2, R_2) = (?, ?)$ and $(\mathbf{t}, R)^{-1} = (?, ?)$.
- Implement `se2_compose(X1, X2)` and `se2_inverse(X)`.

Check your work: $(\mathbf{t}_1, R_1) \cdot (\mathbf{t}_2, R_2) = (\mathbf{t}_1 + R_1 \mathbf{t}_2, R_1 R_2)$ and $(\mathbf{t}, R)^{-1} = (-R^T \mathbf{t}, R^T)$.

Solution:

Code snippet 1 shows the implementation of `se2_compose(X1, X2)` and `se2_inverse(X)`

P3) SE(2) Lie Algebra, Exp/Log, and Adjoint

The Lie algebra $\mathfrak{se}(2)$ consists of twists. We use Translation-first ordering $\xi = (v_x \ v_y \ \omega)^T \in \mathbb{R}^3$

3.a) Lie Algebra Structure:

Question:

With $\xi = (v_x, v_y, \omega)^T = (\mathbf{v}^T, \omega)^T$, the wedge map is:

$$\xi^\wedge = \begin{bmatrix} \omega^\wedge & \mathbf{v} \\ \mathbf{0}^T & 0 \end{bmatrix} = \begin{bmatrix} 0 & -\omega & v_x \\ \omega & 0 & v_y \\ 0 & 0 & 0 \end{bmatrix}$$

- Verify this is block upper-triangular (rotation block in upper-left, translation in upper-right).
- Implement `se2_wedge(xi)` and `se2_vee(Xi)`.

Solution:

Code snippet 1 shows the implementation of `se2_wedge(xi)` and `se2_vee(Xi)`

3.b) Exponential Map:

Question:

The closed-form expression is:

$$\text{Exp}((\xi)^\wedge) = \begin{bmatrix} R(\omega) & V(\omega)\mathbf{v} \\ \mathbf{0}^T & 1 \end{bmatrix}$$

$$\text{where } V(\omega) = \frac{\sin \omega}{\omega} I + \frac{1 - \cos \omega}{\omega} \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}.$$

- Derive by hand using the matrix exponential power series (show key steps).
- Implement `se2_exp(xi)` with Taylor series for $\omega \approx 0$ (use 5 terms).

Solution:

Code snippet 1 shows the implementation of `se2_exp(xi)`

3.c) Logarithm Map:

Question:

The inverse of $V(\omega)$ is:

$$V(\omega)^{-1} = \frac{\omega}{2} \cot \frac{\omega}{2} I + \frac{\omega}{2} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

- Implement `se2_log(X)`. Use Taylor series for $\omega \approx 0$ (use 5 terms).
- Verify numerically: $\text{Exp}((\text{Log}((X)))^\wedge) = X$ for random $X \in \text{SE}((2))$.

Solution:

Code snippet 1 shows the implementation of `se2_log(X)`.

Code snippet 2 shows the implementation of the unit test verifying $\text{Exp}((\text{Log}((X)))^\wedge) = X$ for random $X \in \text{SE}((2))$ named `test_se2_exp_log()` and Fig 1 shows the test passing.

3.d) Adjoint Representation Ad_X :

Question:

The adjoint $\text{Ad}_X : \mathfrak{se}(2) \rightarrow \mathfrak{se}(2)$ is defined by $(\text{Ad}_X \xi)^\wedge = X \xi^\wedge X^{-1}$. *Note: You don't need to multiply out 2×2 blocks explicitly. Showing block form (e.g., $R \omega^\wedge R^T$, $R \mathbf{v}$) is sufficient.*

Translation-first ordering $\xi = (v_x, v_y, \omega)^T$:

- Compute $X \xi^\wedge X^{-1}$ explicitly for $X = (\mathbf{t}, R)$.
- Extract the 3×3 matrix Ad_X such that $\text{Ad}_X \xi$ gives the transformed twist.

You should get: $\text{Ad}_X = \begin{bmatrix} R & \mathbf{t}^\odot \\ \mathbf{0}^T & 1 \end{bmatrix}$ where $\mathbf{t}^\odot = \begin{bmatrix} t_y \\ -t_x \end{bmatrix} = -J\mathbf{t}$ with $J = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$.

Rotation-first ordering $\xi = (\omega, v_x, v_y)^T$:

- Repeat the derivation with this ordering. The wedge map produces the same 3×3 matrix, but extracting Ad_X requires matching to the new vector ordering.
- You should get: $\text{Ad}_X = \begin{bmatrix} 1 & \mathbf{0}^T \\ \mathbf{t}^\odot & R \end{bmatrix}$ (block *lower*-triangular).

Verification:

- Verify numerically: $\text{Ad}_{X_1 X_2} = \text{Ad}_{X_1} \text{Ad}_{X_2}$ for random $X_1, X_2 \in \text{SE}(()2)$.
- Verify numerically: $\text{Ad}_X^{-1} = \text{Ad}_{X^{-1}}$ (the adjoint respects inverses).
- **Connection to Problem 2:** Confirm that R acts on velocity components, as you predicted from the normal subgroup structure.

Solution:

Code snippet 2 shows the implementation of the unit test verifying $\text{Ad}_{X_1 X_2} = \text{Ad}_{X_1} \text{Ad}_{X_2}$ for random $X_1, X_2 \in \text{SE}(()2)$ for random $X \in \text{SE}(()2)$ named `test_se2_Ad_composition` and Fig 1 shows the test passing. Code snippet 2 shows the implementation of the unit test verifying $\text{Ad}_X^{-1} = \text{Ad}_{X^{-1}}$ for random $X \in \text{SE}(()2)$ named `test_se2_Ad_inv()` and Fig 1 shows the test passing.

3.e) Lie Bracket and ad_ξ :

Question:

The Lie bracket on vectors is defined by $[\xi_1, \xi_2]^\wedge := \xi_1^\wedge \xi_2^\wedge - \xi_2^\wedge \xi_1^\wedge$ (i.e., compute the matrix commutator, then apply vee). The small adjoint ad_ξ is the 3×3 matrix such that $[\xi_1, \xi_2] = \text{ad}_{\xi_1} \xi_2$.

Translation-first ordering $\xi = (v_x, v_y, \omega)^T$:

- Compute $[\xi_1^\wedge, \xi_2^\wedge]$ for $\xi_i = (v_{ix}, v_{iy}, \omega_i)^T$. Extract the result as a vector.
- Derive by hand the 3×3 matrix ad_ξ .

You should get: $\text{ad}_\xi = \begin{bmatrix} \omega^\wedge & \mathbf{v}^\odot \\ \mathbf{0}^T & 0 \end{bmatrix}$ where $\mathbf{v}^\odot = \begin{bmatrix} v_y \\ -v_x \end{bmatrix} = -J\mathbf{v}$.

Solution:

Implementation and Unit Tests

Code Snippet 1: SE(2) and $\mathfrak{se}(2)$ Function Implementations

```

1 import numpy as np
2 from Code.S02.S02_maps import so2_wedge, so2_vee, so2_exp, so2_log
3
4 def se2_compose(X1, X2):
5     """
6     Composition of elements of SE(2)
7
8     :param X1: element of SE(2)
9     :param X2: element of SE(2)
10    """
11    X1_t = X1[0:2, 2].reshape((2, 1))
12    X1_R = X1[0:2, 0:2]
13
14    X2_t = X2[0:2, 2].reshape((2, 1))
15    X2_R = X2[0:2, 0:2]
16
17    X12_R = X1_R @ X2_R
18    X12_t = X1_R @ X2_t + X1_t
19
20    X12 = np.block([[X12_R, X12_t],
21                    [np.zeros((1, 2)), 1]])
22
23    return X12
24
25 def se2_inverse(X):
26     """
27     Inverse of SE(2) element
28
29     :param X: element of SE(2)
30     """
31    X_t = X[0:2, 2].reshape((2, 1))
32    X_R = X[0:2, 0:2]
33
34    X_inv_t = -X_R.T @ X_t
35    X_inv_R = X_R.T
36
37    X_inv = np.block([[X_inv_R, X_inv_t],
38                      [np.zeros((1, 2)), 1]])
39
40    return X_inv
41
42 def se2_wedge(xi):
43     """
44     Maps from se2 real number parametrization to se2 Lie algebra
45
46     :param xi: twist vector (v_x, v_y, omega) in R3
47     """
48    xiwedge = np.block([[so2_wedge(xi[2]), xi[0:2].reshape((2, 1))],
49                        [np.zeros((1, 2)), 0]])
50
51    return xiwedge
52
53 def se2_vee(xiwedge):
54     """
55     Maps from se2 Lie algebra to its real number parametrization
56
57     :param xiwedge: 3x3 se2 Lie algebra element
58     """
59
60    xi = np.block([xiwedge[[0, 1], [2, 2]], so2_vee(xiwedge[0:2, 0:2])]);
61
62    return xi
63
64 def se2_exp(xi):

```



```

65 """
66 Maps from parametrization of se2 Lie algebra to SE2 Lie group
67
68 :param xi: twist vector (v_x, v_y, omega) in R3
69 """
70 # Extract elements
71 v = xi[0:2]
72 omega = xi[2]
73
74 # S02 exponential map
75 R = so2_exp(omega)
76
77 # Translation-orientation coupling
78 if abs(omega) > 1e-8:
79     V = np.sin(omega) / omega * np.eye(2) + (1 - np.cos(omega)) / omega * np.array([[0,
80 -1], [1, 0]])
81 else:
82     V = ((1 - omega ** 2 / 6 + omega ** 4 / 120) * np.eye(2)
83           + (omega / 2 - omega ** 3 / 24) * np.array([[0, -1], [1, 0]]))
84
85 X = np.block([[R,      V @ v.reshape((2, 1))],
86               [np.zeros((1, 2)),      1]])
87
88 return X
89
90 def se2_log(xiwedge):
91     """
92     Maps from SE2 Lie group to the parametrization of its Lie algebra se2
93
94     :param xiwedge: 3x3 se2 Lie algebra element
95     """
96     omega = so2_log(xiwedge[0:2, 0:2])
97
98     if abs(omega) > 1e-8:
99         V_inv = omega / 2 * (np.cos(omega / 2) / np.sin(omega / 2) * np.eye(2) + np.array
100            ([[0, 1], [-1, 0]]))
101     else:
102         V_inv = 1 / 2 * ((2 - omega ** 2 / 6 - omega ** 4 / 360 - 2 * omega ** 6 / 30240 +
103            omega ** 8 / 604800) * np.eye(2) + omega * np.array([[0, -1], [1, 0]]))
104
105     v = V_inv @ xiwedge[0:2, 2]
106
107     xi = np.vstack((v.reshape((2, 1)), np.reshape(omega, (1,1))))
108
109     return xi
110
111 def se2_Ad(X):
112     """
113     Maps from se2 to se2 defined by (Ad_X xi)**wedge = X xi**wedge X**-1
114     Shifts tangent spaces
115
116     :param X: SE2 Lie group element
117     """
118     Ad = np.copy(X)
119     Ad[0:2, 2] = -np.array([[0, -1], [1, 0]]) @ X[0:2, 2]
120
121     return Ad
122
123 def se2_ad(xi):
124     """
125     Derivative of Adjoint at identity
126
127     :param xi: se(2) Lie algebra element parametrization
128     """
129     ad = se2_wedge(xi)
130     ad[0:2, 2] = -np.array([[0, -1], [1, 0]]) @ ad[0:2, 2]
131
132     return ad

```

Code Snippet 2: Unit Tests for SE(2) Using Pytest

```

1 import pytest as pt
2 import numpy as np
3 from Code.SE2.SE2_maps import se2_compose, se2_inverse, se2_exp, se2_log, se2_ee, se2_wedge
  , se2_Ad, se2_ad
4
5 def test_se2_exp_log():
6     """
7     Test that exp(log(X)) = X for random SE2 elements
8     """
9     n_samples = 100
10
11     rng = np.random.default_rng()
12     xi_random = np.reshape(rng.uniform(-10, 10, 3 * n_samples), (3, n_samples))
13
14     checks = np.zeros([n_samples, 1])
15     for i in range(n_samples):
16         X_random = se2_exp(xi_random[:, i])
17
18         X_log = se2_log(X_random)
19         X_tilde = se2_exp(X_log)
20
21         checks[i] = np.all(X_tilde == pt.approx(X_random))
22
23     assert np.all(checks)
24
25 def test_se2_Ad():
26     """
27     Test that Ad_X(xi) = (X xi^wedge X^-1)^vee for random SE2, se2 elements
28     """
29     n_samples = 100
30
31     rng = np.random.default_rng()
32     xi_X_random = np.reshape(rng.uniform(-10, 10, 3 * n_samples), (3, n_samples))
33     xi_random = np.reshape(rng.uniform(-10, 10, 3 * n_samples), (3, n_samples))
34
35     checks = np.zeros([n_samples, 1])
36     for i in range(n_samples):
37         X_random = se2_exp(xi_X_random[:, i])
38
39         xi_prime = se2_ee(X_random @ se2_wedge(xi_random[:, i]) @ se2_inverse(X_random))
40         xi_prime_ad = se2_Ad(X_random) @ xi_random[:, i]
41
42         checks[i] = np.all(xi_prime_ad == pt.approx(xi_prime))
43
44     assert np.all(checks)
45
46 def test_se2_Ad_composition():
47     """
48     Test that Ad_{X_1 X_2} = Ad_{X_1}Ad_{X_2} for random SE2 elements
49     """
50     n_samples = 100
51
52     rng = np.random.default_rng()
53     xi1_random = np.reshape(rng.uniform(-10, 10, 3 * n_samples), (3, n_samples))
54     xi2_random = np.reshape(rng.uniform(-10, 10, 3 * n_samples), (3, n_samples))
55
56     checks = np.zeros([n_samples, 1])
57     for i in range(n_samples):
58         X1_random = se2_exp(xi1_random[:, i])
59         X2_random = se2_exp(xi2_random[:, i])
60
61         Ad_X1X2 = se2_Ad(se2_compose(X1_random, X2_random))
62         Ad_X1_Ad_X2 = se2_Ad(X1_random) @ se2_Ad(X2_random)
63
64         checks[i] = np.all(Ad_X1X2 == pt.approx(Ad_X1_Ad_X2))
65

```

```

66     assert np.all(checks)
67
68 def test_se2_Ad_inv():
69     """
70     Test that (Ad_X)^-1 = Ad_{X^-1} for random SE2 elements
71     """
72     n_samples = 100
73
74     rng = np.random.default_rng()
75     xi_random = np.reshape(rng.uniform(-10, 10, 3 * n_samples), (3, n_samples))
76
77     checks = np.zeros([n_samples, 1])
78     for i in range(n_samples):
79         X_random = se2_exp(xi_random[:, i])
80
81         Ad_X_inv = np.linalg.inv(se2_Ad(X_random))
82         Ad_Xinv = se2_Ad(se2_inverse(X_random))
83
84         a = se2_Ad(X_random) @ Ad_X_inv
85         b = se2_Ad(X_random) @ Ad_Xinv
86
87         checks[i] = np.all(Ad_X_inv == pt.approx(Ad_Xinv))
88
89     assert np.all(checks)
90
91 def test_se2_ad():
92     """
93     Test that ad_{xi_1}(xi_2) = [xi_1, xi_2] for random se2 elements
94     """
95     n_samples = 100
96
97     rng = np.random.default_rng()
98     xi1_random = np.reshape(rng.uniform(-10, 10, 3 * n_samples), (3, n_samples))
99     xi2_random = np.reshape(rng.uniform(-10, 10, 3 * n_samples), (3, n_samples))
100
101     checks = np.zeros([n_samples, 1])
102     for i in range(n_samples):
103         Lie_bracket = se2_vee(se2_wedge(xi1_random[:, i]) @ se2_wedge(xi2_random[:, i])
104                               - se2_wedge(xi2_random[:, i]) @ se2_wedge(xi1_random[:, i]))
105         Lie_bracket_ad = se2_ad(xi1_random[:, i]) @ xi2_random[:, i]
106
107         checks[i] = np.all(Lie_bracket_ad == pt.approx(Lie_bracket))
108
109     assert np.all(checks)

```

Figure 1: Pytest unit test output showing tests passing

```

===== test session starts =====
platform win32 -- Python 3.10.11, pytest-9.0.2, pluggy-1.6.0
rootdir: C:\Users\thatf\OneDrive\Documents\Purdue Classes\AAE 590LGM\Lie Group Methods
collected 5 items

Tests\SE2\test_SE2.py ..... [100%]

===== 5 passed in 0.56s =====

```

P4) SE(2) Kinematics Simulation

The kinematic equation $\dot{X} = X\xi^\wedge$ describes rigid body motion on SE(2)

4.a)

Question:

Unicycle Model: A unicycle has forward velocity v and yaw rate ω . In the body frame, there is no lateral motion ($v_y = 0$) so $v_x = v$.

- Write the body-frame twist for the unicycle
- Expand the KE to derive the ODEs
- For constant $v, \omega > 0$: What shape is the trajectory? What is the radius?

Solution:

The body frame twist is

$$\xi = \begin{pmatrix} v_x \\ v_y \\ \omega \end{pmatrix} = \begin{pmatrix} v \\ 0 \\ \omega \end{pmatrix}$$

Deriving the equations of motion

$$\begin{aligned} \dot{X} = X\xi^\wedge &= \begin{pmatrix} \cos(\theta) & -\sin(\theta) & x \\ \sin(\theta) & \cos(\theta) & y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & -\omega & v \\ \omega & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} -\omega \sin(\theta) & -\omega \cos(\theta) & v \cos(\theta) \\ \omega \cos(\theta) & -\omega \sin(\theta) & v \sin(\theta) \\ 0 & 0 & 0 \end{pmatrix} \\ \implies \dot{x} &= v \cos(\theta), \dot{y} = v \sin(\theta), \dot{\cos}(\theta) = -\omega \sin(\theta), \dot{\sin}(\theta) = \omega \cos(\theta) \\ \dot{\cos}(\theta) &= -\sin(\theta)\dot{\theta} = -\omega \sin(\theta) \implies \dot{\theta} = \omega \\ \dot{\sin}(\theta) &= \cos(\theta)\dot{\theta} = \omega \cos(\theta) \implies \dot{\theta} = \omega \end{aligned}$$

$$\boxed{\dot{x} = v \cos(\theta), \dot{y} = v \sin(\theta), \dot{\theta} = \omega}$$

For constant $v, \omega > 0$ the trajectory follows a circle with a radius found by computing the following integral

$$r = \int_0^{\frac{\pi}{2\omega}} v \cos(\omega t) dt = \frac{v}{\omega} \int_0^{\frac{\pi}{2}} \cos(\theta) d\theta = \frac{v}{\omega}$$

The trajectory is a circle with radius $\frac{v}{\omega}$

4.b)

Question:

Solution:

Code Snippet 3: SE(2) Integrator Implementation

```
1 import numpy as np
2 from Code.SE2.SE2_maps import se2_compose, se2_exp, se2_wedge
3
4 def se2_euler_integrator(X_k, xi_k, dt):
5     """
6     Integrate using first order Euler
```

```

7
8     :param X_k: group element at t_k
9     :param xi_k: twist at t_k
10    :param dt: timestep
11    """
12    X_kp1 = se2_compose(X_k, np.eye(3) + dt * se2_wedge(xi_k))
13
14    return X_kp1
15
16 def se2_Lie_group_integrator(X_k, xi_k, dt):
17     """
18     Integrate using exponential map (assuming twist is constant across timestep)
19
20     :param X_k: group element at t_k
21     :param xi_k: twist at t_k
22     :param dt: timestep
23     """
24     X_kp1 = se2_compose(X_k, se2_exp(dt * xi_k))
25
26     return X_kp1

```

Code Snippet 4: SE(2) Integrator Test

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from Code.SE2.SE2_maps import se2_compose, se2_exp, se2_log, se2_wedge, se2_ee, se2_inverse
4 from Code.SE2.SE2_integrators import se2_euler_integrator, se2_Lie_group_integrator
5
6 ## Unicycle sim
7
8 # Initialize
9 X_0 = np.eye(3)
10 v = 1 # [m / s]
11 omega = 0.5 # [rad / s]
12 xi_k = np.array([[v], [0], [omega]])
13 dt = 0.1 # [s]
14 tf = 20
15 time = np.arange(0, tf + dt, dt)
16
17 # Simulation loop
18 X_k_euler = X_0
19 X_k_Lie = X_0
20
21 positions_euler = np.zeros([2, time.size])
22 positions_Lie = np.zeros([2, time.size])
23
24 error_euler = np.zeros_like(time)
25 error_Lie = np.zeros_like(time)
26
27 for k in range(time.size - 1):
28     X_k_euler = se2_euler_integrator(X_k_euler, xi_k, dt)
29     X_k_Lie = se2_Lie_group_integrator(X_k_Lie, xi_k, dt)
30
31     positions_euler[:, k + 1] = X_k_euler[0:2, 2]
32     positions_Lie[:, k + 1] = X_k_Lie[0:2, 2]
33
34     R_euler = X_k_euler[0:2, 0:2]
35     R_Lie = X_k_Lie[0:2, 0:2]
36
37     error_euler[k + 1] = np.linalg.norm(R_euler.T @ R_euler - np.eye(2), "fro")
38     error_Lie[k + 1] = np.linalg.norm(R_Lie.T @ R_Lie - np.eye(2), "fro")
39
40 # Plot
41 # Trajectories
42 plt.plot(positions_euler[0, :], positions_euler[1, :], label = "Euler")
43 plt.plot(positions_Lie[0, :], positions_Lie[1, :], label = "Lie Group")
44 plt.xlabel("X Position [m]")
45 plt.ylabel("Y Position [m]")

```

```

46 plt.title("Unicycle Trajectories vs Integrator")
47 plt.legend()
48 plt.grid(True)
49 plt.gca().set_aspect('equal', adjustable='box')
50 plt.savefig("./HW2/Parts/Q4/AAE590LGM_HW2_Q4_traj.png")
51 plt.show()
52
53 # Error
54 plt.semilogy(time, error_euler, "o", label = "Euler")
55 plt.semilogy(time, error_Lie, "o", label = "Lie Group")
56 plt.xlabel("Time [s]")
57 plt.ylabel("Rotation Error")
58 plt.title("Rotation Error vs Time for Both Integrators")
59 plt.legend()
60 plt.grid(True)
61 plt.savefig("./HW2/Parts/Q4/AAE590LGM_HW2_Q4_error.png")
62 plt.show()
63
64 # Compare with Exp map
65 X_f = se2_compose(X_0, se2_exp(tf * xi_k))
66 print(np.linalg.norm(X_f - X_k_Lie, "fro"))

```

Figure 2: Trajectories of both integrators for unicycle

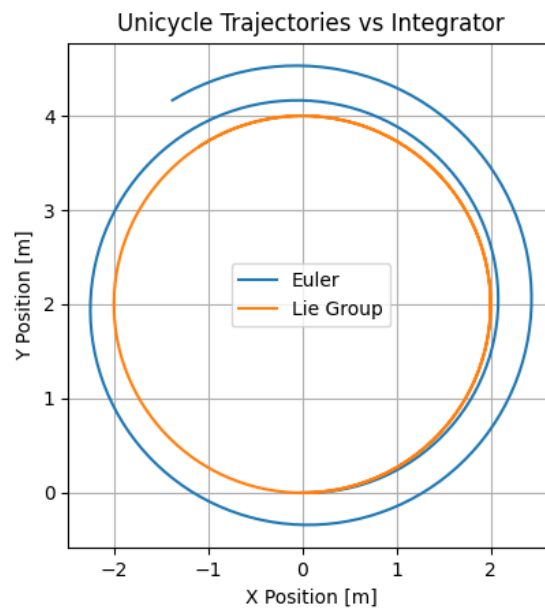
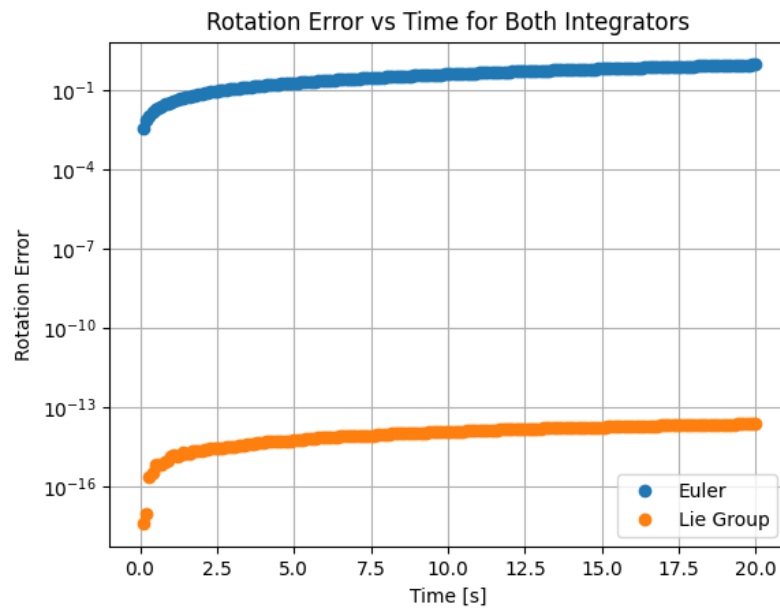


Figure 3: Error over time for both integrators



The error between the exact propagation using the exponential map and the Lie group integrator at the final time is $1.8289049996250573e-14$. This means that within numerical precision they are the same which makes sense given that the Lie group integrator uses the exponential map.