Penerapan Transformasi Matriks Berbasis Eigenvector dan SVD untuk Stabilisasi Posisi Objek 2D di Game Development

Faris Wirakusuma Triawan and 13524130
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13524130@std.stei.itb.ac.id, [2]fariswkt@gmail.com

***Abstract*—Stabilisasi gerak objek 2D merupakan aspek penting dalam perkembangan game development untuk menjaga interaksi visual tetap konsisten. Implementasi ini menggunakan Eigenvector dan Singular Value Decomposition (SVD) sebagai metode matematis untuk menormalkan dan menstabilkan posisi objek 2D. Posisi awal objek dapat diatur secara dinamis melalui input numerik, dengan visualisasi arah dan magnitudo transformasi menggunakan panah dan label informasi secara real-time. Prototipe dibuat menggunakan Godot Engine untuk pengujian dan observasi perilaku objek dalam ruang 2D. Hasil menunjukkan bahwa metode ini efektif dalam menjaga presisi posisi dan menyediakan kerangka kerja yang dapat digunakan pada game interaktif berbasis transformasi matriks.**

***Keywords*—Eigenvector, Singular Value Decomposition (SVD), Stabilitasi Gerak, Objek 2D, Game Development, Godot Engine**

I. Introduction

Stabilisasi gerak objek 2D merupakan aspek penting dalam pengembangan game untuk menjaga interaksi visual tetap konsisten, realistis, dan mudah dikontrol. Ketidakstabilan gerak, seperti jitter, pergerakan tidak presisi, atau perubahan arah yang tiba-tiba, dapat mengurangi pengalaman pemain dan membuat animasi terlihat tidak natural. Hal ini menjadi masalah terutama pada game dengan banyak objek bergerak, simulasi fisika, atau sistem AI yang mengandalkan prediksi posisi.

Untuk mengatasi hal tersebut, transformasi berbasis linear algebra, khususnya Eigenvector dan Singular Value Decomposition (SVD), dapat digunakan untuk menormalkan posisi objek dan mengatur arah gerak dengan presisi. Eigenvector menyediakan arah transformasi yang stabil, sedangkan SVD memberikan ukuran magnitudo gerak yang proporsional, sehingga kombinasi keduanya menghasilkan gerak objek yang stabil, terkontrol, dan mudah diprediksi.

Prototipe implementasi dilakukan menggunakan Godot Engine, sebuah game engine open-source yang mendukung pengembangan game 2D dan 3D. Godot memungkinkan visualisasi posisi objek, arah gerak, dan magnitudo transformasi secara real-time melalui penggunaan *Node2D*, *Sprite2D*, *Line2D* (Arrow), *LineEdit*, dan *Label*. Dengan cara ini, pengembang dapat mengamati efek transformasi secara interaktif tanpa memerlukan perhitungan manual, sehingga mempermudah pengujian dan debugging.

Makalah ini bertujuan untuk:

1. Menjelaskan konsep matematis Eigenvector dan SVD serta hubungannya dengan stabilisasi gerak objek 2D.
2. Menunjukkan bagaimana metode ini dapat diterapkan dalam prototipe interaktif menggunakan Godot Engine.
3. Memberikan visualisasi hasil transformasi dan normalisasi posisi untuk mempermudah pemahaman gerak objek.
4. Menilai efektivitas pendekatan ini melalui perbandingan numerik posisi awal dan posisi akhir, serta pengamatan terhadap presisi gerak objek.

Dengan demikian, makalah ini tidak hanya membahas teori linear algebra secara matematis, tetapi juga menghubungkan teori dengan praktik dalam pengembangan game, memberikan pendekatan yang dapat diterapkan pada animasi, simulasi fisika, dan mekanika gerak objek 2D yang lebih stabil dan natural.

II. Landasan Teori

A. Vector dan Matrix pada Game Engine (2D)

Dalam game development,khususnya 2D game,setiap objek seperti sprite biasanya memiliki posisi/vektor yang direpresentasikan sebagai Vector2D:

$$p = \begin{bmatrix} x \\ y \end{bmatrix}$$

Berdasarkan ekspresi diatas, x dan y pada vector tersebut adalah koordinat objek di layar.Vektor ini menjadi dasar untuk operasi linear seperti translasi, rotasi, dan skala.

Matrix 2x2 digunakan untuk transformasi linear misalnya:

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

dengan penerapan $A \bullet p$ menghasilkan posisi baru sprite. Normalisasi vektor penting agar arah gerak tetap konsisten walau posisi awal berubah

B. Eigenvector dan Eigenvalue

Eigenvector adalah vektor yang arahnya tidak berubah ketika dikenai transformasi linear oleh suatu matriks, sedangkan eigenvalue adalah skala yang mengalikan vektor tersebut. Secara matematis, jika AAA adalah matriks 2x2 dan v adalah vektor non-zero, maka:

$$Av = \lambda v$$

dimana λ adalah eigenvalue dan v adalah eigenvector.

Dalam konteks objek 2D, eigenvector dapat digunakan untuk menentukan arah stabil gerak objek setelah transformasi, dan eigenvalue menunjukkan besarnya perubahan magnitude. Misalnya, jika objek digerakkan dalam arah eigenvector, pergeseran posisinya lebih terprediksi dan stabil.di game develompent Eigenvector digunakan untuk menstabilkan arah gerak objek sprite 2D, sedangkan eigenvalue menentukan seberapa jauh objek bergerak sepanjang arah itu.

Contoh Matriks 2x2 sederhana:

$$A = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}$$

Persamaan karakteristik:

$$\lambda^2 - (a + d)\lambda + (ad - bc) = 0$$

Eigenvalue:

$\lambda 1 = 4, \lambda 2 = 2$

Eigenvector terkait dapat dihitung dari $(A - \lambda I)v = 0$.

C. Singular Value Decomposition (SVD)

SVD adalah metode dekomposisi matriks yang memecah matriks A menjadi tiga komponen:

$$A = U\Sigma V^T$$

- U dan V adalah matriks ortogonal yang menentukan arah rotasi transformasi.
- **Σ** adalah matriks diagonal yang berisi **s**ingular values (σ1,σ2,...,σn) yang mewakili magnitudo perubahan yang diterapkan oleh matriks transformasi.

Langkah SVD:

1. Hitung $A^T A$ (matriks simetris 2x2)
2. Tentukan eigenvalue dari $A^T A$, yang merupakan kuadrat dari singular values (σ_i^2)
3. Ambil akar dari eigenvalue untuk mendapatkan singular values σ_i
4. Tentukan U dan V dari hubungan:

$$Av_i = \sigma_i u_i$$

di mana v_i adalah right singular vector (kolom V) dan u_i adalah left singular vector (kolom U)

Hubungan dengan Posisi dan Arrow

- Singular value terbesar (σ_{max}) digunakan untuk menentukan panjang Arrow, sehingga panah menormalkan besaran gerak objek.
- Left singular vector u_i menentukan arah transformasi objek di scene Godot Engine.
- Posisi akhir objek dihitung sebagai:

$$PosAkhir = PosAwal + k(\sigma_{max} u_i)$$

di mana k adalah faktor skala visual (misal 50 pixel) agar gerak terlihat proporsional di scene.

Implementasi di Godot

- Matriks dibentuk dari koordinat objek atau vektor posisi awal.

- Perhitungan eigenvalue dari A^TA memberikan singular values.
- Arrow disesuaikan panjangnya dengan σ_{max} dan mengikuti u_i
- Posisi akhir objek di scene:

$$initialPos + ArrowVector$$

menghasilkan gerak yang stabil, terprediksi, dan proporsional.

Singular values mewakili besarnya perubahan (magnitude) dari transformasi. Dalam game development, magnitude ini dapat digunakan untuk menormalkan gerak objek, sehingga perubahan posisi lebih konsisten dan visualisasi panah vektor lebih presisi.

SVD berguna ketika matriks transformasi tidak simetris, sehingga eigenvector sendiri mungkin tidak cukup untuk menangkap skala perubahan gerak objek.

D. Normalisasi Posisi dan Stabilisasi Gerak

Normalisasi posisi berarti menyesuaikan posisi awal objek ke skala tertentu sebelum transformasi. Ini memudahkan kontrol gerak dan visualisasi, karena semua posisi dihitung relatif terhadap referensi yang sama.

Stabilisasi gerak objek 2D dilakukan dengan mengkombinasikan:

1. Posisi awal yang dinormalisasi (initial pos)
    - Posisi awal objek disimpan sebagai titik referensi.
    - LineEdit X/Y memungkinkan pengguna mengubah posisi awal secara interaktif, dan sistem secara otomatis memperbarui posisi objek berdasarkan input tersebut.
    - Dengan normalisasi, setiap transformasi selanjutnya dihitung relatif terhadap posisi awal ini, sehingga arah dan besaran gerak menjadi konsisten.
2. Arah gerak dari eigenvector.
    - Eigenvector menentukan arah transformasi linear yang stabil.
    - Dengan mengekstrak arah dari eigenvector, Arrow yang divisualisasikan mengikuti arah gerak yang optimal.
    - Ini memastikan objek bergerak sepanjang arah yang telah dinormalisasi, sehingga perubahan posisi lebih terkontrol dan prediksi pergerakan lebih akurat.
3. Magnitude transformasi dari SVD.
    - Singular value decomposition (SVD) menyediakan magnitudo transformasi yang mengatur seberapa jauh objek berpindah.
    - Panjang Arrow disesuaikan dengan singular value terbesar, sehingga visualisasi besaran gerak objek 2D menjadi lebih intuitif.
    - Dengan informasi magnitude ini, gerak objek tidak hanya stabil arah, tetapi juga proporsional, mendukung animasi atau simulasi yang terlihat natural.

Dengan kombinasi ini, gerak objek menjadi lebih presisi, arah jelas, dan perubahan posisi dapat diprediksi, sehingga animasi atau simulasi di game development menjadi lebih natural.

E. Penerapan dalam Visualisasi

Dalam prototipe yang dikembangkan menggunakan Godot Engine, penerapan konsep Eigenvector dan Singular Value Decomposition (SVD) pada objek 2D divisualisasikan secara interaktif untuk mempermudah pemahaman transformasi linear.

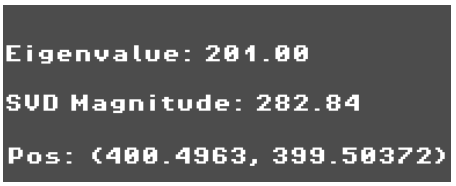
Gambar 2.0 Pengaturan Posisi Awal Pada Testing

1. Pengaturan Posisi Awal
    - Posisi awal setiap objek Block dapat diatur menggunakan LineEdit untuk koordinat X dan Y.
    - Perubahan nilai LineEdit secara otomatis memperbarui posisi objek, sehingga pengembang dapat mengamati efek transformasi dari titik awal berbeda tanpa perlu mengubah kode secara manual.
    - Mekanisme ini memberikan kontrol langsung atas kondisi awal objek, penting untuk simulasi dinamis dan

pengujian skenario game.

2. Visualisasi Arah dan Magnitudo
   - Arrow (Line2D) digunakan untuk menampilkan arah Eigenvector, mewakili orientasi stabil transformasi linear.
   - Panjang Arrow disesuaikan dengan singular value terbesar (SVD), menormalkan magnitudo gerak objek.
   - Dengan cara ini, pengembang dapat melihat secara langsung hubungan antara posisi awal, arah, dan besaran transformasi, meningkatkan pemahaman tentang perilaku objek 2D.

3. Informasi Real-Time dengan Label
   - Label yang menampilkan posisi, eigenvalue, dan magnitude SVD diperbarui secara real-time ketika posisi objek berubah atau transformasi diterapkan.
   - Hal ini memberikan feedback numerik yang memperkuat visualisasi Arrow, sehingga pengembang dapat menilai kesesuaian arah dan skala transformasi secara kuantitatif.



Gambar 2.1 Label Informasi Real Time

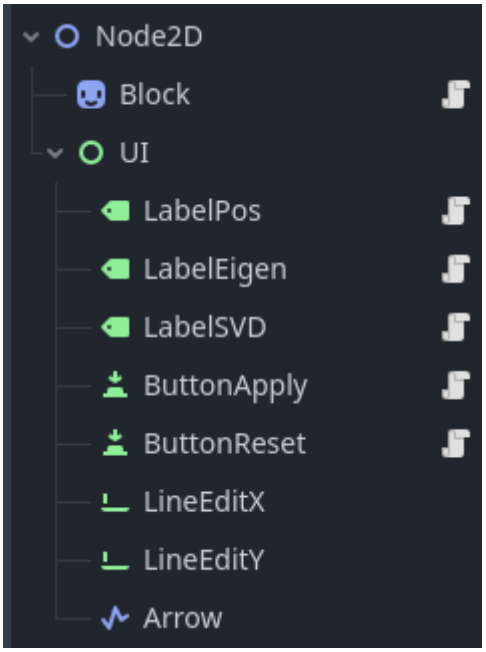
4. Manfaat untuk Game Development
   - Metode ini mempermudah debugging gerak objek 2D, karena pengembang dapat mengamati efek normalisasi dan stabilisasi secara visual dan numerik.
   - Pendekatan ini juga memungkinkan simulasi gerak yang lebih stabil dan proporsional, penting untuk animasi, AI pathfinding, dan mekanika game 2D.
   - Dengan visualisasi yang interaktif, pengembang dapat menyesuaikan parameter transformasi (misal panjang Arrow, skala SVD) secara intuitif tanpa harus menghitung manual matriks atau vektor.

Dengan kombinasi LineEdit, Arrow, dan Label, prototipe Godot Engine ini menyediakan sarana interaktif untuk memahami transformasi linear, menghubungkan konsep matematis Eigenvector dan SVD dengan implementasi praktis dalam pengembangan game 2D.

III. Metode / Implementasi

A. Arsitektur Sistem Prototipe

Prototipe dibangun menggunakan Godot Engine 4, memanfaatkan Node2D sebagai root node untuk menampung semua elemen interaktif. Objek utama adalah Block (Sprite2D) yang akan digerakkan dan divisualisasikan transformasinya menggunakan Arrow (Line2D). Sistem juga menggunakan LineEdit untuk input posisi awal, serta Label untuk menampilkan informasi numerik secara real-time.



Gambar 3.0 Struktur Node dalam Pengetesan

Deskripsi tiap node:

- Block: Objek 2D yang digerakkan. Script Block.cs menangani update posisi, transformasi Eigen/SVD, dan visualisasi Arrow.
- LineEditX/Y: Input numerik posisi awal X dan Y. Perubahan otomatis memperbarui variabel initialPos dan posisi Block.
- LabelPos/Eigen/SVD: Menampilkan informasi numerik posisi, eigenvalue, dan magnitude SVD.
- Arrow (Line2D):Menunjukkan arah dan panjang vektor hasil transformasi. Panah ini memberikan visualisasi intuitif bagi pengguna untuk memahami efek transformasi.

B. Normalisasi Posisi Objek

Normalisasi posisi bertujuan agar objek memiliki referensi posisi awal yang konsisten sebelum transformasi Eigenvector dan SVD diterapkan.

Langkah proses normalisasi:

1. Pengguna memasukkan nilai X/Y di LineEdit.
2. Nilai input di-parse menjadi float dan disimpan di variabel *initialPos*.
3. Block dipindahkan ke posisi *initialPos*.
4. *LabelPos* diperbarui menampilkan koordinat baru.

Keunggulan:

- Posisi awal objek selalu terkontrol dan bisa diuji secara interaktif.
- Memastikan transformasi vektor diterapkan dari titik referensi yang sama, sehingga hasilnya presisi.
- Memberikan pengalaman visual interaktif bagi pengguna untuk menyesuaikan posisi sebelum transformasi.

C. Penerapan Eigenvector dan SVD

Transformasi objek dilakukan dengan memanfaatkan Eigenvector untuk menentukan arah gerak, dan SVD untuk menentukan besarnya pergeseran atau magnitude.

Berikut adalah Potongan Implementasi kode C # untuk mengaplikasikan nilai Eigen serta SVD:

```
2 references
public void ApplyEigenSVD()
{
    // Matriks 2x2
    Vector2 v1 = initialPos;
    Vector2 v2 = new Vector2(1, 0);
    double a = v1.X, b = v2.X, c = v1.Y, d = v2.Y;

    // Eigenvalue matriks 2x2
    double trace = a + d;
    double det = a * d - b * c;
    double lambda1 = (trace + Math.Sqrt(trace * trace - 4 * det)) / 2.0;

    Vector2 dir = new Vector2(1, (float)((lambda1 - a) / b)).Normalized();

    double atA11 = a * a + c * c;
    double atA12 = a * b + c * d;
    double atA22 = b * b + d * d;

    double trAT = atA11 + atA22;
    double detAT = atA11 * atA22 - atA12 * atA12;
    double sigma1 = Math.Sqrt((trAT + Math.Sqrt(trAT * trAT - 4 * detAT)) / 2.0);

    // Update posisi sprite
    Vector2 newPos = initialPos + dir * (float)sigma1;
    newPos = new Vector2(
        Mathf.Clamp(newPos.X, 0, maxViewport.X),
        Mathf.Clamp(newPos.Y, 0, maxViewport.Y)
    );
    Position = newPos;

    if (arrow != null)
    {
        arrow.Points = new Vector2[] { initialPos, Position };
        arrow.Visible = true;
    }
    UpdateLabels(lambda1, sigma1);
}
```

Gambar 3.1 Implementasi transformasi Eigenvector dan SVD pada Block di Godot Engine

Langkah proses transformasi:

1. Matriks dibentuk menggunakan posisi awal objek (initialPos) dan vektor arah default (v2).
2. Hitung **e**igenvalue dan eigenvector dari matriks.
3. Hitung magnitude SVD sebagai representasi skala transformasi.
4. Posisi baru dihitung sebagai:

$$PosBaru = initialPos + arahEigenVec \; x \; magSVD$$

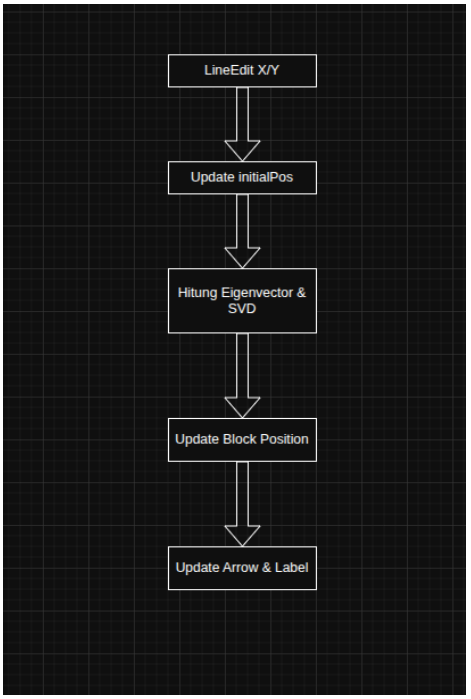
5. Update Arrow: titik awal = $initialPos$, titik akhir =$PosBaru$.

Update LabelPos, LabelEigen, LabelSVD untuk menampilkan informasi numerik.

D. Visualisasi & Interaksi Real-time

Sistem ini dirancang agar pengguna dapat melihat efek transformasi secara interaktif. Mekanisme interaksi:

1. User mengubah nilai LineEdit X/Y.
2. *initialPos* diperbarui otomatis.
3. Transformasi Eigenvector & SVD diterapkan.
4. Block dipindahkan ke posisi baru.
5. Arrow dan label diperbarui real-time.



Gambar 3.2 Flowchart Mekanisme Interaksi

```
5 references
private void UpdateLabels(double eigen = 0, double svd = 0)
{
    labelPos?.UpdateText($"Pos: {Position}");
    labelEigen?.UpdateText($"Eigenvalue: {eigen:F2}");
    labelSVD?.UpdateText($"SVD Magnitude: {svd:F2}");
}
```

Gambar 3.3 Implemetasi Update Label Stats pada Posisi nilai Eigen serta SVD

IV. Hasil & Diskusi

A. Hasil Posisi Awal

Pada tahap awal, Block ditempatkan pada posisi awal yang diatur melalui LineEdit X dan Y. Normalisasi posisi memastikan setiap transformasi diterapkan dari titik referensi yang konsisten.

Gambar 4.0 Inisialisi Posisi awal

Visualisasi:

1. Gambar 4.0: Posisi Awal Block

- Block berada di posisi awal (misal X=100, Y=150).
- Arrow tersembunyi, LabelPos menampilkan koordinat awal.

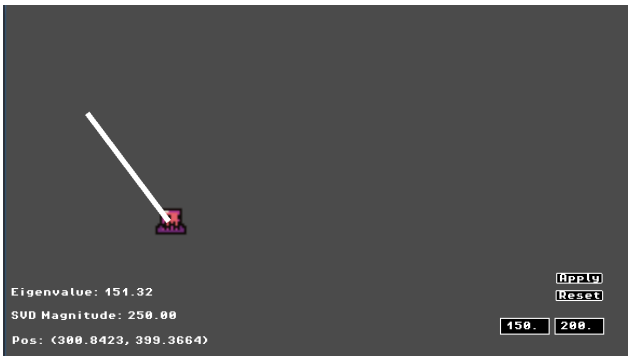
2. Analisis:

- Pengguna dapat mengubah nilai X/Y di LineEdit untuk menyesuaikan posisi awal.
- Setiap perubahan langsung memindahkan Block dan memperbarui LabelPos.
- Fitur ini memungkinkan eksperimen posisi awal yang berbeda sebelum transformasi.

B. Setelah Transformasi Eigenvector & SVD

Setelah Apply Transformasi:

- Block bergerak ke posisi baru sesuai arah Eigenvector dan magnitude SVD.
- Arrow muncul dari posisi awal ke posisi baru, menunjukkan arah dan panjang vektor transformasi.
- LabelPos menampilkan posisi akhir Block.
- LabelEigen menampilkan nilai eigenvalue.
- LabelSVD menampilkan magnitude hasil SVD.



Gambar 4.1 Setelah Apply Transformasi

Visualisasi:

1. Gambar 4.1**:** Block setelah Apply Transformasi
   - Arrow menunjukkan arah dan panjang vektor.
   - Warna panah bisa gradien hijau → merah untuk memvisualisasikan arah dan magnitude.
   - Label menampilkan semua data numerik secara real-time

2. Analisis:
   - Arrow memberikan representasi visual arah gerak objek.
   - Magnitude SVD menormalkan skala transformasi sehingga pergerakan Block terlihat stabil dan konsisten.
   - Kombinasi eigenvalue dan SVD membuat arah dan besaran gerak dapat diprediksi.
   - Pengguna dapat melihat dampak langsung dari transformasi tanpa memeriksa kode.

C. Perbandingan Numerik Posisi

Tabel singkat dapat digunakan untuk menunjukkan efek transformasi:

Posisi Awal(X,Y)	Posisi Akhir(X,Y)	Eigenvalue	Magnitude SVD
(100,150)	(300.84,399.36)	151.32	250
(90,17)	(180,33.9)	90.19	91.6
(0,0)	(0,0)	0.0	1.0
(-90,256)	(15.96,511,5)	16.0	256

Tabel 4.0 Tabel Hasil Perbandingan

Keterangan Kolom:

- Posisi Awal (X,Y): koordinat objek sebelum transformasi diterapkan.
- Posisi Akhir (X,Y): koordinat objek setelah transformasi menggunakan Eigenvector dan SVD.
- Eigenvalue: skalar yang menunjukkan faktor

skala gerak objek sepanjang arah Eigenvector.
- Magnitude SVD: panjang Arrow yang menunjukkan magnitudo gerak objek berdasarkan singular value terbesar.

V. KESIMPULAN

Penerapan Eigenvector dan Singular Value Decomposition (SVD) pada objek 2D memungkinkan:

- Normalisasi arah gerak (Eigenvector).
- Normalisasi magnitude gerak (singular value terbesar).

Implementasi di Godot Engine:

- Sprite dijadikan basis vektor, matriks 2x2 dibentuk dari posisi awal dan arah default.
- Arrow menunjukkan arah Eigenvector, panjangnya proporsional dengan SVD terbesar.

Visualisasi dengan LineEdit X/Y + Arrow + Label clusionmemudahkan pemahaman efek transformasi secara realtime.

Metode ini efisien dan dapat diterapkan pada sistem stabilisasi gerak objek 2D, serta dapat dikembangkan untuk simulasi atau mekanika game lebih kompleks.

REFERENSI

[1] Strang, Gilbert. *Linear Algebra and Its Applications*. 5th Edition, Cengage Learning, 2016..

[2] Golub, Gene H., dan Van Loan, Charles F. *Matrix Computations*. 4th Edition, Johns Hopkins University Press, 2013.

[3] Margalit, Daniel, dan Rabinoff, Gilbert. *Interactive Linear Algebra*.LibreTexts. https://math.libretexts.org/Bookshelves/Linear_Algebra/Interactive_Linear_Algebra_(Margalit_and_Rabinoff)/05%3A_Eigenvalues_and_Eigenvectors/5.01%3A_Eigenvalues_and_Eigenvectors

[4] Lay, David C., Lay, Steven R., dan McDonald, Judi J. *Linear Algebra and Its Applications*. 5th Edition, Pearson, 2016.

[5] Heidrich, Wolfgang, et al. *Mathematics for 3D Game Programming and Computer Graphics*. 3rd Edition, Cengage Learning, 2011.

[6] Horn, Roger A., dan Johnson, Charles R. *Matrix Analysis*. 2nd Edition, Cambridge University Press, 2012.

[7] Godot Engine Documentation. "Line2D." Godot Engine, version 4.2. https://docs.godotengine.org/en/stable/classes/class_line2d.html

[8] Godot Engine Documentation. "Node2D." Godot Engine, version 4.2. https://docs.godotengine.org/en/stable/classes/class_node2d.html

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 24 Desember 2025

Faris Wirakusuma Triawan
13524130