

5 Serbestlik Dereceli (5-DOF) Robot Kol Simülasyonu

Language / Dil: [TR Türkçe](#) | [GB English](#)

İçindekiler

- Proje Genel Bakış
- Kinematik Yapı ve Eklem Konfigürasyonu
- İleri Kinematik (Forward Kinematics)
- Ters Kinematik (Inverse Kinematics)
- Jacobian Matrisi ve Hız Kinematiği
- Trajektori Planlama
- 3D Görselleştirme
- Kullanım Kılavuzu
- Teknik Detaylar

1. Proje Genel Bakış

Bu proje, 5 serbestlik dereceli (5-DOF) bir robot kolun gerçek zamanlı simülasyonunu OpenGL ve C# kullanarak gerçekleştirmektedir. Proje, robotik alanındaki temel kinematik kavramları içermektedir:

- İleri Kinematik (FK):** Eklem açılarından uç efektör pozisyonunu hesaplama
- Ters Kinematik (IK):** Hedef pozisyondan eklem açılarını hesaplama
- Jacobian Matrisi:** Eklem hızları ile uç efektör hızı arasındaki ilişki
- Trajektori Planlama:** Yumuşak hareket geçişleri
- 3D Görselleştirme:** Gerçek zamanlı robot simülasyonu

Teknoloji Yığını

- Platform:** .NET Framework 4.7.2
- Grafik Kütüphanesi:** OpenTK 3.3.3 (OpenGL)
- Dil:** C#
- IDE:** Visual Studio / Visual Studio Code

2. Kinematik Yapı ve Eklem Konfigürasyonu

2.1 Eklem Tanımları

Robot kol 5 döner (revolute) eklemden oluşmaktadır:

Eklem	İsim	Dönüş Ekseni	Açı Değişkeni	Hareket Sınırıları	Açıklama
J1	Taba	Y ekseni	θ_1	Sınırsız	Taban

Eklem	İsim	Dönüş Eksenleri	Açı Değişkeni	Hareket Sınırları	Açıklama
	n (Base)				rotasyonu (Yaw)
J2	Omuz z (Shoulder)	X eksenleri	θ_2	$[-90^\circ, +90^\circ]$	Öne/arkaya eğilme (Pitch)
J3	Dirsek k (Elbow w)	X eksenleri	θ_3	$[-150^\circ, +150^\circ]$	Kol büükülmesi (Pitch)
J4	Bilek t Pitch)	X eksenleri	θ_4	$[-90^\circ, +90^\circ]$	Bilek yukarı/aşağı (Pitch)
J5	Bilek Roll (Wrist t Roll)	Y eksenleri	θ_5	Sınırsız	Uç efektör dönüşü (Yaw)

2.2 Link Uzunlukları

Robot kolun fiziksel parametreleri:

$$\begin{aligned}
 L_1 &= 1.0 \text{ birim} && (\text{Taban yüksekliği}) \\
 L_2 &= 0.8 \text{ birim} && (\text{Omuz-Dirsek arası}) \\
 L_3 &= 0.6 \text{ birim} && (\text{Dirsek-Bilek arası}) \\
 L_4 &= 0.5 \text{ birim} && (\text{Bilek-Uç arası})
 \end{aligned}$$

Toplam Maksimum Erişim: $L_1 + L_2 + L_3 + L_4 = 2.9$ birim

2.3 Koordinat Sistemi

Proje standart robotik koordinat sistemini kullanır: - **X eksenleri**: Kırmızı - Yatay (sağ/sol) - **Y eksenleri**: Yeşil - Dikey (yükari/aşağı) - **Z eksenleri**: Mavi - Derinlik (ileri/geri)

Sağ el kuralı (right-hand rule) uygulanır.

3. İleri Kinematik (Forward Kinematics)

3.1 Matematiksel Formülatasyon

İleri kinematik, eklem açıları vektörü $\mathbf{q} = [\theta_1, \theta_2, \theta_3, \theta_4, \theta_5]^T$ verildiğinde, uç efektörün (end-effector) pozisyonunu $\mathbf{p} = [x, y, z]^T$ hesaplar.

Denavit-Hartenberg (DH) Yaklaşımı Yerine Doğrudan Matris Çarpımı

Bu implementasyon, DH parametreleri yerine doğrudan homojen transformasyon matrislerini kullanır:

$$T_0^5 = T_0^1 \cdot T_1^2 \cdot T_2^3 \cdot T_3^4 \cdot T_4^5$$

Her eklem için transformasyon:

Eklem 1 (Taban - Y eksenini dönüsü):

$$T_0^1 = \text{Rot}(Y, \theta_1) \cdot \text{Trans}(0, L_1, 0)$$

Eklem 2 (Omuz - X eksenini dönüsü):

$$T_1^2 = \text{Rot}(X, \theta_2) \cdot \text{Trans}(0, L_2, 0)$$

Eklem 3 (Dirsek - X eksenini dönüsü):

$$T_2^3 = \text{Rot}(X, \theta_3) \cdot \text{Trans}(0, L_3, 0)$$

Eklem 4 (Bilek Pitch - X eksenini dönüsü):

$$T_3^4 = \text{Rot}(X, \theta_4) \cdot \text{Trans}(0, L_4, 0)$$

Eklem 5 (Bilek Roll - Y eksenini dönüsü):

$$T_4^5 = \text{Rot}(Y, \theta_5)$$

3.2 Homojen Transformasyon Matrisleri

Y eksenini etrafında dönüş:

$$R^y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

X eksenini etrafında dönüş:

$$R^x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Translasyon matrisi:

$$\text{Trans}(t) = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3.3 Kod Implementasyonu

```

Vector3 ComputeForwardKinematics(float θ₁, float θ₂, float θ₃, float θ₄,
float θ₅)
{
    // Derece -> Radyan dönüşümü
    float t1 = DegreesToRadians(θ₁);
    float t2 = DegreesToRadians(θ₂);
    float t3 = DegreesToRadians(θ₃);
    float t4 = DegreesToRadians(θ₄);
    float t5 = DegreesToRadians(θ₅);

    Matrix4 T = Identity;

    // Eklem transformasyonlarını sırayla uygula
    T *= RotationY(t1) * Translation(0, L₁, 0);
    T *= RotationX(t2) * Translation(0, L₂, 0);
    T *= RotationX(t3) * Translation(0, L₃, 0);
    T *= RotationX(t4) * Translation(0, L₄, 0);
    T *= RotationY(t5);

    // Orijin noktasını transformasyon matrisi ile dönüştür
    return TransformPosition(Vector3.Zero, T);
}

```

3.4 Hesaplama Karmaşıklığı

- Zaman Karmaşıklığı:** O(n) - n eklem sayısı (burada n=5)
 - Uzay Karmaşıklığı:** O(1) - Sabit matris boyutu
-

4. Ters Kinematik (Inverse Kinematics)

4.1 Problem Tanımı

Ters kinematik problemi: Verilen hedef pozisyon $\mathbf{p}_{\text{tar}} = [x_t, y_t, z_t]^T$ için, üç efektörün bu pozisyonala ulaşmasını sağlayacak eklem açıları $\mathbf{q} = [\theta_1, \theta_2, \theta_3, \theta_4, \theta_5]^T$ vektörünü bulmak.

Bu problem genellikle: - **Çok çözümeli** (multiple solutions) - **Çözümsüz** (hedef erişim alanı dışında) - **Analitik çözümü zor** (5+ DOF için)

4.2 Geometrik Analitik Çözüm Yaklaşımı

Bu implementasyon, 5-DOF problemi 3-DOF problemine indirgeyen geometrik bir yaklaşım kullanır.

Adım 1: Taban Açısı (ϑ_1) Hesabı

Hedef pozisyonla üstten bakıldığındırda (XZ düzlemini):

$$\theta_1 = \text{atan2}(x_t, z_t)$$

Hedefin tabana yatay uzaklığı:

$$r = \sqrt{(x_t^2 + z_t^2)}$$

Adım 2: Düzlemsel 2-Link IK Problemi

Taban döndürüldükten sonra, problem 2D'ye indirgenmiş olur. Şimdi (r, y) düzleminde çalışızıız.

Basitleştirme: Son iki linki birleştirerek 3-DOF sistemini 2-DOF'a indirge:

$$L_2' = L_2 = 0.8 \text{ (Üst kol)}$$

$$L_3' = L_3 + L_4 = 0.6 + 0.5 = 1.1 \text{ (Alt kol toplam)}$$

Omuz ekleminden hedef pozisyonuna mesafe:

$$y' = y_t - L_1 \text{ (taban yüksekliği çıkarılır)}$$

$$d = \sqrt{(r^2 + y'^2)}$$

Erişilebilirlik kontrolü:

```
if (d > L2' + L3') → Hedef erişilemez, return false
```

Adım 3: Kosinüs Teoremi ile Dirsek Açısı

Üçgen geometrisinden kosinüs teoremi:

$$\cos(\theta_3) = (d^2 - L_2'^2 - L_3'^2) / (2 \cdot L_2' \cdot L_3')$$

Dirsek açısı:

$$\theta_3 = \text{acos}(\cos(\theta_3))$$

Not: $\cos(\theta_3)$ değeri $[-1, 1]$ aralığında kısıtlanmalıdır (sayısal hata önleme).

Adım 4: Omuz Açısı (ϑ_2) Hesabı

İki bileşen açısının toplamı:

Hedefin açısal yönelimi:

$$\phi = \text{atan2}(y', r)$$

İç açı düzeltmesi:

$$\psi = \text{atan2}(L_3' \cdot \sin(\theta_3), L_2' + L_3' \cdot \cos(\theta_3))$$

Nihai omuz açısı:

$$\theta_2 = \pi/2 - (\phi + \psi)$$

Not: $\pi/2$ terimi, robotun dikey başlangıç duruşundan kaynaklanır.

Adım 5: Bilek Açılarının Dağıtımı

Hesaplanan toplam dirsek bükülmesini θ_3 ve θ_4 arasında dağıt:

$$\begin{aligned}\theta_3_{\text{final}} &= \theta_3_{\text{total}} \times 0.6 \quad (\text{Dirsek \%60}) \\ \theta_4_{\text{final}} &= \theta_3_{\text{total}} \times 0.4 \quad (\text{Bilek \%40})\end{aligned}$$

Son eklem sabit tutulur:

$$\theta_5 = 0$$

4.3 Kod Implementasyonu

```
bool SolveIK(Vector3 target, out float  $\theta_1$ , out float  $\theta_2$ ,
             out float  $\theta_3$ , out float  $\theta_4$ , out float  $\theta_5$ )
{
    // Adım 1: Taban açısı
    float t1 = atan2(target.X, target.Z);
    float r = sqrt(target.X2 + target.Z2);

    // Adım 2: 2D problem
    float y = target.Y - L1;
    float d = sqrt(r2 + y2);

    float l1 = L2;
    float l2 = L3 + L4;

    // Erişilebilirlik kontrolü
    if (d > l1 + l2) return false;

    // Adım 3: Kosinüs teoremi
    float cosQ3 = (d2 - l12 - l22) / (2·l1·l2);
    cosQ3 = Clamp(cosQ3, -1, 1);
    float q3 = acos(cosQ3);

    // Adım 4: Omuz açısı
    float phi = atan2(y, r);
    float psi = atan2(l2·sin(q3), l1 + l2·cos(q3));
    float q2 =  $\pi/2$  - (phi + psi);

    // Adım 5: Açıları dağıt
     $\theta_1$  = RadiansToDegrees(t1);
     $\theta_2$  = RadiansToDegrees(q2);

    float totalElbow = RadiansToDegrees(q3);
     $\theta_3$  = totalElbow × 0.6;
     $\theta_4$  = totalElbow × 0.4;
     $\theta_5$  = 0;
```

```

    return true;
}

```

4.4 Çözüm Özellikleri

- **Tip:** Geometrik analitik çözüm
 - **Hız:** O(1) - Sabit zaman
 - **Avantajlar:**
 - Çok hızlı
 - Deterministik
 - Kapalı form çözüm
 - **Dezavantajlar:**
 - Tek çözüm sağlar (birden fazla olası konfigürasyon varsa)
 -
 - 5. eklem sabit ($\theta_5 = 0$)
 - Eklem limitlerini tam kontrol etmez
-

5. Jacobian Matrisi ve Hız Kinematiği

5.1 Jacobian Matrisi Teorisi

Jacobian matrisi $J(\mathbf{q})$, eklem hızları $\dot{\mathbf{q}}$ ile uç efektör hızı $\dot{\mathbf{x}}$ arasındaki ilişkisi tanımlar:

$$\dot{\mathbf{x}} = J(\mathbf{q}) \cdot \dot{\mathbf{q}}$$

Burada: - $\dot{\mathbf{x}} = [\dot{x}, \dot{y}, \dot{z}]^T \in \mathbb{R}^3$: Kartezyen uzayda lineer hız - $\dot{\mathbf{q}} = [\dot{\theta}_1, \dot{\theta}_2, \dot{\theta}_3, \dot{\theta}_4, \dot{\theta}_5]^T \in \mathbb{R}^5$: Eklem açısal hızları - $J(\mathbf{q}) \in \mathbb{R}^{3 \times 5}$: Jacobian matrisi (3 satır × 5 sütun)

5.2 Jacobian Matris Yapısı

$$J = \begin{bmatrix} \frac{\partial x}{\partial \theta_1} & \frac{\partial x}{\partial \theta_2} & \frac{\partial x}{\partial \theta_3} & \frac{\partial x}{\partial \theta_4} & \frac{\partial x}{\partial \theta_5} \\ \frac{\partial y}{\partial \theta_1} & \frac{\partial y}{\partial \theta_2} & \frac{\partial y}{\partial \theta_3} & \frac{\partial y}{\partial \theta_4} & \frac{\partial y}{\partial \theta_5} \\ \frac{\partial z}{\partial \theta_1} & \frac{\partial z}{\partial \theta_2} & \frac{\partial z}{\partial \theta_3} & \frac{\partial z}{\partial \theta_4} & \frac{\partial z}{\partial \theta_5} \end{bmatrix}$$

Her sütun j , j 'inci eklemin uç efektör pozisyonu üzerindeki etkisini gösterir.

5.3 Nümerik Türev ile Jacobian Hesaplama

Analitik türev karmaşık olduğu için, nümerik yaklaşım kullanılır (sonlu farklar yöntemi):

Merkezi Fark Yaklaşımı:

$$\frac{\partial f}{\partial x} \approx [f(x + h) - f(x)] / h$$

Her eklem için:

$$J[:, i] = [FK(\theta_1, \dots, \theta_{i+h}, \dots, \theta_5) - FK(\theta_1, \dots, \theta_i, \dots, \theta_5)] / h$$

Burada: - $h = 0.1^\circ$ (türev adım boyutu) - $FK(\cdot)$ = İleri kinematik fonksiyonu

5.4 Kod Implementasyonu

```
float[,] ComputeJacobian(float θ1, float θ2, float θ3, float θ4, float θ5)
{
    float[,] J = new float[3, 5]; // 3x5 matris
    float h = 0.1°; // Türev adımı

    Vector3 current = FK(θ1, θ2, θ3, θ4, θ5);

    // Her sütun için (her eklem)
    for (int i = 0; i < 5; i++)
    {
        // i'inci açıyı h kadar artır
        Vector3 perturbed = FK_with_perturbation(i, h);

        // Nümerik türev
        Vector3 column = (perturbed - current) / Radians(h);

        J[0, i] = column.X;
        J[1, i] = column.Y;
        J[2, i] = column.Z;
    }

    return J;
}
```

5.5 Hız Kinematiği Hesaplama

Verilen eklem hızları için üç efektör hızını hesapla:

```
Vector3 ComputeEndEffectorVelocity(float[,] J, float[] q_dot)
{
    // v = J · ̇q
    Vector3 v = Vector3.Zero;

    for (int row = 0; row < 3; row++) // X, Y, Z bileşenleri
    {
        for (int col = 0; col < 5; col++) // 5 eklem
        {
            v[row] += J[row, col] * q_dot[col];
        }
    }

    return v;
}
```

Hız büyüklüğü:

$$|v| = \sqrt{v_x^2 + v_y^2 + v_z^2}$$

5.6 Jacobian'ın Önemi

Jacobian matrisi robot kontrolünde kritik öneme sahiptir:

1. **Hız Kontrolü:** Kartezyen uzayda hız hedefleri → Eklem hızları
2. **Kuvvet Kontrolü:** Eklem torkları → Uç efektör kuvveti
3. **Singülerite Analizi:** $\det(J \cdot J^T) \approx 0$ → Tekil konfigürasyon
4. **Ters Kinematik Optimizasyonu:** Nümerik IK çözümlerinde
5. **Yörünge Takibi:** Gerçek zamanlı yol kontrolü

5.7 Hesaplama Karmaşıklığı

- **Zaman:** $O(n^2)$ - n eklem sayısı (5 FK çağrıları × her biri $O(n)$)
 - **Uzay:** $O(n)$ - $3 \times n$ matris depolama
-

6. Trajektori Planlama

6.1 Trajektori Planlama Nedir?

Trajektori planlama, robotun başlangıç konfigürasyonundan hedef konfigürasyona **yumuşak** ve **sürekli** bir şekilde geçişini sağlar. Aniden atlama yerine, zaman içinde kademeli değişim gerçekleştirilir.

6.2 Lineer Interpolasyon (LERP)

En basit trajektori planlama yöntemi lineer interpolasyondur:

Matematiksel Formül:

$$q(t) = q_{\text{start}} + (q_{\text{target}} - q_{\text{start}}) \cdot u(t)$$

Burada: - **q(t):** t anındaki eklem açıları - **q_start:** Başlangıç açıları - **q_target:** Hedef açıları - **u(t):** Normalizasyon parametresi $[0, 1]$

Zaman normalizasyonu:

$$u(t) = t / T_{\text{duration}}$$

$$\begin{aligned} u(t) &\in [0, 1] \quad \text{where:} \\ u(0) &= 0 \quad \rightarrow \text{Başlangıç} \\ u(T) &= 1 \quad \rightarrow \text{Bitiş} \end{aligned}$$

6.3 Eklem Uzayı vs Kartezyen Uzay

Bu implementasyon: Eklem Uzayı (Joint Space)

Avantajları: - Her eklem bağımsız interpolasyon - Eklem limitleri kolay kontrol - Hesaplama basit ve hızlı - Singüleritelerden etkilenmez

Dezavantajları: - Uç efektör düz çizgi takip etmez - Kartezyen uzayda öngörülemeden yörünge

Alternatif: Kartezyen Uzay - Uç efktör düz yol izler - Her adımda IK çözümü gereklidir - Hesaplama yoğun - Singüleritelerle karşılaşabilir

6.4 Kod Implementasyonu

```
// LERP Fonksiyonu
float Lerp(float a, float b, float t)
{
    return a + (b - a) * t;
}

// Trajektori Güncellemesi (Her Frame)
void UpdateTrajectory(float deltaTime)
{
    if (!trajectoryActive) return;

    // Zamanı ilerlet
    trajectoryTime += deltaTime;

    // Normalizasyon parametresi
    float u = trajectoryTime / trajectoryDuration;

    // Bitiş kontrolü
    if (u >= 1.0f)
    {
        u = 1.0f;
        trajectoryActive = false; // Trajektori tamamlandı
    }

    // Her eklem için LERP
    θ₁ = Lerp(θ₁_start, θ₁_target, u);
    θ₂ = Lerp(θ₂_start, θ₂_target, u);
    θ₃ = Lerp(θ₃_start, θ₃_target, u);
    θ₄ = Lerp(θ₄_start, θ₄_target, u);
    θ₅ = Lerp(θ₅_start, θ₅_target, u);
}
```

6.5 Gelişmiş Trajektori Profilleri

Lineer interpolasyon ani hız değişimine neden olur. Daha yumuşak profiller:

S-Eğrisi (Sigmoid):

$$u_{\text{smooth}}(t) = 3u^2 - 2u^3$$

Kosinüs Profili:

$$u_{\text{smooth}}(t) = (1 - \cos(\pi u)) / 2$$

5. Derece Polinom:

$$u_{\text{smooth}}(t) = 10u^3 - 15u^4 + 6u^5$$

Bu profiller: - Başlangıç ve bitişte **sıfır hız** garanti eder - Sürekli ivme sağlar - Mekanik sistemler için daha uygun

6.6 Zaman Parametreleri

Proje varsayılan değerleri:

```
T_duration = 2.0 saniye (Toplam hareket süresi)  
FPS = 60 (Frame/saniye)  
Toplam Frame = 120 (2.0 × 60)
```

Her frame'de açı değişimi:

```
Δθ = (θ_target - θ_start) / 120
```

7. 3D Görselleştirme

7.1 OpenGL Render Pipeline

Proje OpenTK (OpenGL) kullanarak gerçek zamanlı 3D render yapar.

Render Adımları:

1. **Clear Buffer:** Ekranı ve derinlik tamponunu temizle
2. **Projection Matrix:** Perspektif kamera ayarla
3. **View Matrix:** Kamera pozisyonu ve yönelimi
4. **Model Matrix:** Her obje için transformasyon
5. **Draw Calls:** Geometrik primitifler çiz
6. **Swap Buffer:** Çift tamponlama (double buffering)

7.2 Kamera Sistemi

Perspektif Projeksiyon:

```
Projection = CreatePerspective(  
    FOV = 45°,           // Görüş alanı  
    Aspect = W/H,        // En-boy oranı  
    Near = 0.1,          // Yakın düzlem  
    Far = 100.0          // Uzak düzlem  
)
```

Sferik Kamera Hareketi:

Kamera pozisyonu polar koordinatlarda:

```
x = distance · cos(angleX) · cos(angleY)  
y = distance · sin(angleX)  
z = distance · cos(angleX) · sin(angleY)
```

Parametreler: - **distance**: Merkeze uzaklık [2, 20] - **angleX**: Dikey açı [-89°, +89°] - **angleY**: Yatay açı [Sınırsız]

View Matrix:

```
ViewMatrix = LookAt(  
    eye = cameraPosition,      // Kamera konumu  
    target = (0, 0, 0),        // Baktığı noktası (orijin)  
    up = (0, 1, 0)            // Yukarı yön vektörü  
)
```

7.3 Işıklandırma Modeli (Phong Lighting)

OpenGL sabit fonksiyon hattı (fixed pipeline) ışıklandırması:

Işık Bileşenleri:

```
I_total = I_ambient + I_diffuse + I_specular
```

Ambient (Ortam Işığım):

```
I_ambient = K_a · L_a  
K_a = 0.3  (Malzeme albedo)
```

Diffuse (Yayınık Işık):

```
I_diffuse = K_d · L_d · max(N · L, 0)  
K_d = 1.0  
L_d = (1, 1, 1)  (Beyaz ışık)
```

Işık Pozisyonu:

```
Light_position = (5, 10, 10, 1)
```

7.4 Geometri Çizimi

Silindir Primitifi:

Robot kolları silindir olarak modellenir (küp yerine daha gerçekçi):

```
void DrawCylinder(float radius, float height, int segments)  
{  
    // Yan yüzey (QuadStrip)  
    for (int i = 0; i <= segments; i++)  
    {  
        float angle = 2π · i / segments;  
        float x = radius · cos(angle);  
        float z = radius · sin(angle);  
  
        // Normal vektör (ışıklandırma için)  
        Normal(x, 0, z);  
  
        // Üst ve alt vertex
```

```

        Vertex(x, height, z);
        Vertex(x, 0, z);
    }

    // Üst ve alt kapaklı
    DrawDisk(radius, 0);      // Alt kapak
    DrawDisk(radius, height); // Üst kapak
}

```

Boyutlar: - Link radius: 0.1 birim - Joint radius: 0.18 birim - Segment sayısı: 24 (pürüzsüz silindir)

7.5 Hiyerarşik Transformasyon

Robot çizimi ağaç yapısında (scene graph):

```

World
└ Base (θ1)
  └ Link1 (L1)
    └ Shoulder (θ2)
      └ Link2 (L2)
        └ Elbow (θ3)
          └ Link3 (L3)
            └ Wrist (θ4)
              └ Link4 (L4)
                └ WristRoll (θ5)
                  └ Gripper

```

OpenGL Matris Yığını (Matrix Stack):

```

PushMatrix();           // Mevcut matrisi kaydet
Rotate(θ);             // Dönüş uygula
Translate(0, L, 0);   // İleri git
DrawLink();             // Link çiz
DrawChild();            // Alt eleman çiz
PopMatrix();            // Önceki matrise dön

```

Bu yöntem: - Ebeveyn transformasyonları alt elemanlara otomatik yayılır - Lokal koordinat sistemleri kullanılır - Kod temiz ve modüler olur

7.6 Renk Şeması

Eleman	Renk	RGB
Linkler	Turuncu	(1.0, 0.5, 0.0)
Eklemler	Gri	(0.6, 0.6, 0.6)
Gripper Taban	Koyu Gri	(0.3, 0.3, 0.3)
Gripper Parmaklar	Açık Gri	(0.8, 0.8, 0.8)
Zemin	Koyu Gri	(0.2, 0.2, 0.2)
X Ekseni	Kırmızı	(1, 0, 0)

Eleman	Renk	RGB
Y Ekseni	Yeşil	(0, 1, 0)
Z Ekseni	Mavi	(0, 0, 1)

7.7 Gripper Animasyonu

Gripper 3 parmaktan oluşur (120° açılarda):

$$\text{Parmak pozisyonu} = 0.05 + (0.16 - 0.05) \cdot t$$

$$t = (\text{gap} - \text{gap_min}) / (\text{gap_max} - \text{gap_min}) \quad [0, 1]$$

- **gap_min** = 0.03 (Kapalı)
 - **gap_max** = 0.25 (Açık)
-

8. Kullanım Kılavuzu

8.1 Kurulum

Gereksinimler:

.NET Framework 4.7.2 Developer Pack
Visual Studio 2019/2022 veya VS Code
OpenTK 3.3.3 (NuGet ile otomatik)

Derleme:

```
cd RobotArm5DOF
dotnet build
```

Çalıştırma:

```
dotnet run
# veya
.\RobotArm5DOF\bin\Debug\RobotArm5DOF.exe
```

8.2 Klavye Kontrolleri

Eklemler Kontrolleri

Tuş	Fonksiyon	Hareket
Q / A	Taban (θ_1)	Sola/Sağ'a dönüş
W / S	Omuz (θ_2)	Öne/Arkaya eğilme
E / D	Dirsek (θ_3)	Kol bükülme
R / F	Bilek Pitch (θ_4)	Yukarı/Aşağı
T / G	Bilek Roll (θ_5)	Dönüş

Hareket hızı: $60^\circ/\text{saniye}$

Gripper Kontrolü

Tuş Fonksiyon

X Gripper Açı

Z Gripper Kapat

Hareket hızı: 0.6 birim/saniye

Kamera Kontrolleri

Tuş Fonksiyon

← / → Yatay dönüş (angleY)

↑ / ↓ Dikey dönüş (angleX)

Page Up Yakınlaş (Zoom In)

Page Down Uzaklaş (Zoom Out)

Kamera limitleri: - Dikey: [-89°, +89°] - Uzaklık: [2, 20] birim

Ters Kinematik

Tuş Fonksiyon Açıklama

I IK Anında Hedef pozisyonanında git

Y IK Yumuşak Hedef pozisyonanın trajektori ile git (2 saniye)

Hedef pozisyon: (1.2, 1.5, 0.4)

Genel

Tuş Fonksiyon

ESC Çıkış

8.3 Pencere Başlığı Bilgileri

Program başlığında gerçek zamanlı bilgiler gösterilir:

5DOF Robot | Aci:(45,30,-20,15,0) | EE:(1.23,2.45,0.67) | |Vee|:0.34

- **Aci:** Mevcut eklem açıları $[\theta_1, \theta_2, \theta_3, \theta_4, \theta_5]$
 - **EE:** Uç efektör pozisyonu $[x, y, z]$
 - **|Vee|:** Uç efektör hız büyüklüğü (varsayılan eklem hızları için)
-

9. Teknik Detaylar

9.1 Performans Özellikleri

Hesaplama Yükü (Her Frame):

İşlem	Karmaşıklık	Çağrı Sayısı	Süre (yaklaşık)
Forward Kinematics	O(5)	7	< 0.1 ms

İşlem	Karmaşıklık	Çağrı Sayısı	Süre (yaklaşık)
Jacobian	$O(25)$	1	< 0.5 ms
Rendering	$O(N)$	1	1-2 ms
Toplam	< 3 ms		

Frame Rate: 60 FPS (16.67 ms/frame) → Yeterli marj

9.2 Sayısal Stabilite

Potansiyel Problemler:

1. Ters Kinematik:

- Acos domain hatası: $\cos(\theta) \in [-1, 1]$ kontrolü
- Çözüm: Clamp(value, -1, 1)

2. Jacobian Hesaplama:

- Sıfıra bölme: h değeri çok küçük olmamalı
- Çözüm: $h = 0.1^\circ$ (yeterince büyük)

3. Gimbal Lock:

- X ekseni dönüşleri art arda → Potansiyel tekil nokta
- Çözüm: Eklem limitlerle kontrol

Sayısal Hassasiyet: - Float (32-bit): ~7 ondalık basamak - Açı çözünürlüğü: 0.01° (yeterli)

9.3 Koordinat Sistemi Uyumu

Kritik: DrawRobot() ve ComputeForwardKinematics() fonksiyonları **birebir** aynı transformasyon sırasını kullanmalıdır.

Aksi halde: - Görsel ve hesaplanan pozisyon uyumsuz olur - IK çözümü hatalı hedeflere gider - Jacobian matrisi yanlış hesaplanır

Doğrulama Yöntemi: Herhangi bir eklem konfigürasyonu için:

```
visual_position ≈ FK( $\theta_1, \theta_2, \theta_3, \theta_4, \theta_5$ )
```

9.4 Eklem Limitleri

```
// Fiziksel limitler (derece cinsinden)
 $\theta_2 \in [-90, +90]$  // Omuz
 $\theta_3 \in [-150, +150]$  // Dirsek
 $\theta_4 \in [-90, +90]$  // Bilek
```

Neden gereklidir: - Fiziksel robotlarda mekanik limitler vardır - Self-collision (kendi kendine çarpma) önler - Gerçekçi hareket aralığı

Limit Kontrolü:

```
 $\theta_2 = \text{Clamp}(\theta_2, -90, 90);$ 
```

9.5 Singülerite (Tekil Noktalar)

Singülerite Nedir?

Robot belirli konfigürasyonlarda serbestlik kaybeder. Jacobian matrisinin determinantı sıfır olur.

Bu Sistemde Potansiyel Singüleriteler:

1. Taban Singüleritesi:

- Hedef orijin üzerinde ($r = 0$)
- θ_1 belirsiz hale gelir
- Çözüm: Minimal r eşiği

2. Kol Tam Uzandığında:

- $d = L_2 + L_3 + L_4$
- $\theta_3 \approx 0$ (Kol düz)
- Küçük hareket büyük eklem değişimi gerektirir

3. Kol Tam Büküldüğünde:

- $\theta_3 \approx \pm 150^\circ$
- Bilek ve dirsek üst üste

Singülerite Kontrolü:

$$\det(J \cdot J^T) < \varepsilon \rightarrow \text{Singülerite yakın}$$

Burada ε küçük eşik değeridir (örn. 0.001).

10. Matematiksel Referanslar

10.1 Kullanılan Formüller Özeti

Kavram	Formül	Bölüm
Forward Kinematics	$\mathbf{p} = FK(\mathbf{q}) = T_0^5 \cdot [0,0,0,1]^T$	3
Inverse Kinematics	$\theta_1 = \text{atan2}(x, z)$	4
Kosinüs Teoremi	$\cos(C) = (a^2 + b^2 - c^2) / (2ab)$	4
Jacobian	$J[i,j] = \partial p_i / \partial \theta_j$	5
Nümerik Türev	$f'(x) \approx [f(x+h) - f(x)] / h$	5
Hız Kinematiği	$\mathbf{v} = J(\mathbf{q}) \cdot \dot{\mathbf{q}}$	5
LERP	$\mathbf{q}(t) = \mathbf{q}_0 + (\mathbf{q}_1 - \mathbf{q}_0) \cdot t$	6
Sferik Kamera	$x = d \cdot \cos(\varphi) \cdot \cos(\theta)$	7

10.2 Notasyon Tablosu

Sembol	Anlamı	Birim
θ_i	i'inci eklem açısı	derece veya radyan
\mathbf{q}	Eklem açıları vektörü $[\theta_1, \dots, \theta_5]^T$	-

Symbol	Anlamı	Birim
p	Kartezyen pozisyon $[x,y,z]^T$	birim
L_i	i'inci link uzunluğu	birim
J	Jacobian matrisi (3×5)	birim/radyan
v	Lineer hız vektörü	birim/saniye
\dot{q}	Eklem açısal hızları	radyan/saniye
T	Homojen transformasyon matrisi (4×4)	-

10.3 Koordinat Sistemleri

Dünya Koordinatları (World Frame): - Orijin: Robot tabanı - Y ekseni: Yukarı

(gravitasyon tersi) **Eklem Koordinatları (Joint Frame):** - Her eklem lokal koordinat sistemine sahip - Dönüş ekseni: X veya Y

Uç Efektör Koordinatları (End-Effector Frame): - Son eklem merkezinde - Oryantasyon θ_5 ile belirlenir

11. Kaynakça ve İleri Okuma

11.1 Robotik Temel Kitaplar

1. **Craig, J.J.** (2005). *Introduction to Robotics: Mechanics and Control* (3rd ed.). Pearson Education.
 - Kinematik ve dinamik temel referans kitabı
 - Forward/Inverse Kinematics ve Jacobian detaylı anlatım
2. **Spong, M.W., Hutchinson, S., & Vidyasagar, M.** (2020). *Robot Modeling and Control* (2nd ed.). Wiley.
 - DH parametreleri ve transformasyonlar
 - Robot kontrol algoritmaları
3. **Lynch, K.M., & Park, F.C.** (2017). *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press.
 - Modern yaklaşımlar ve screw theory
 - [Ücretsiz online: modernrobotics.org](http://modernrobotics.org)
4. **Siciliano, B., Sciavicco, L., Villani, L., & Oriolo, G.** (2010). *Robotics: Modelling, Planning and Control*. Springer.
 - İleri seviye kontrol teorisi
 - Yörünge planlama ve manipülasyon
5. **Corke, P.I.** (2017). *Robotics, Vision and Control: Fundamental Algorithms in MATLAB* (2nd ed.). Springer.
 - Pratik MATLAB implementasyonları
 - Robot kütüphanesi örnekleri

11.2 Bilgisayar Grafikleri

6. Shirley, P., & Marschner, S. (2009). *Fundamentals of Computer Graphics* (3rd ed.). AK Peters/CRC Press.
 - Transformasyon matrisleri ve render pipeline
 - 3D grafik matematiği
7. Shreiner, D., et al. (2013). *OpenGL Programming Guide* (8th ed.). Addison-Wesley.
 - OpenGL API referansı (Red Book)
 - Shader programlama temelleri

11.3 Matematiksel Temeller

8. Strang, G. (2016). *Introduction to Linear Algebra* (5th ed.). Wellesley-Cambridge Press.
 - Matris işlemleri ve lineer dönüşümler
 - MIT OpenCourseWare dersleriyle destekli
9. Press, W.H., et al. (2007). *Numerical Recipes: The Art of Scientific Computing* (3rd ed.). Cambridge University Press.
 - Nümerik türev ve optimizasyon
 - Pratik algoritma implementasyonları

11.4 C# ve .NET Programlama

10. Albahari, J., & Albahari, B. (2021). *C# 10 in a Nutshell*. O'Reilly Media.
 - C# dil özellikleri ve .NET framework
 - LINQ ve async programlama

11.5 Online Kaynaklar

11. **OpenTK Documentation**
 - OpenTK API tam referansı
 12. **Learn OpenGL**
 - Modern OpenGL comprehensive tutorial
 13. **Robot Academy - QUT**
 - Peter Corke'un video ders serisi
 - Ücretsiz robotik eğitimi
 14. **Stanford CS223A - Introduction to Robotics**
 - Prof. Oussama Khatib video dersleri
 - Kinematik ve dinamik detaylı anlatım
 15. **IEEE Robotics and Automation Society**
 - Akademik makaleler ve konferanslar
 - Robotik araştırma trendleri
-

Ekler

A. Kod Yapisı

```
RobotArm5DOF/
└── Program.cs          (Ana kod - 617 satır)
    ├── Kinematic Variables (θ1-θ5, L1-L4)
    ├── OnLoad()           (OpenGL başlatma)
    ├── OnUpdateFrame()    (Fizik ve input)
    ├── OnRenderFrame()    (Render döngüsü)
    ├── DrawRobot()         (Robot çizimi)
    ├── DrawCylinder()     (Geometri)
    ├── ComputeForwardKinematics()
    ├── SolveIK()           (Ters kinematik)
    ├── ComputeJacobian()   (Jacobian matrisi)
    └── Lerp()               (Trajektori)
── RobotArm5DOF.csproj
── packages.config
── README.md             (Bu dosya)
```

B. Hızlı Başvuru Kartı

Eklem Kontrol:

Q/A: Taban (θ₁) W/S: Omuz (θ₂) E/D: Dirsek (θ₃)
R/F: Bilek (θ₄) T/G: Roll (θ₅) X/Z: Gripper

Kamera:

Oklar: Döndür PgUp/PgDn: Zoom ESC: Çıkış

IK:

I: Anında git Y: Yumuşak git

5 Degrees of Freedom (5-DOF) Robot Arm Simulation

Language / Dil: [TR](#) Türkçe | [GB](#) English

Table of Contents

1. Project Overview
2. Kinematic Structure and Joint Configuration
3. Forward Kinematics
4. Inverse Kinematics
5. Jacobian Matrix and Velocity Kinematics
6. Trajectory Planning

-
- 7. [3D Visualization](#)
 - 8. [User Guide](#)
 - 9. [Technical Details](#)
-

1. Project Overview

This project implements a real-time simulation of a 5 degrees of freedom (5-DOF) robot arm using OpenGL and C#. The project encompasses fundamental kinematic concepts in robotics:

- **Forward Kinematics (FK):** Computing end-effector position from joint angles
- **Inverse Kinematics (IK):** Computing joint angles from target position
- **Jacobian Matrix:** Relationship between joint velocities and end-effector velocity
- **Trajectory Planning:** Smooth motion transitions
- **3D Visualization:** Real-time robot simulation

Technology Stack

- **Platform:** .NET Framework 4.7.2
 - **Graphics Library:** OpenTK 3.3.3 (OpenGL)
 - **Language:** C#
 - **IDE:** Visual Studio / Visual Studio Code
-

2. Kinematic Structure and Joint Configuration

2.1 Joint Definitions

The robot arm consists of 5 revolute joints:

Joint	Name	Rotation Axis	Angle Variable	Motion Limits	Description
J1	Base	Y axis	θ_1	Unlimited	Base rotation (Yaw)
J2	Shoulder	X axis	θ_2	$[-90^\circ, +90^\circ]$	Forward/backward tilt (Pitch)
J3	Elbow	X axis	θ_3	$[-150^\circ, +150^\circ]$	Arm bending (Pitch)
J4	Wrist Pitch	X axis	θ_4	$[-90^\circ, +90^\circ]$	Wrist up/down (Pitch)
J5	Wrist Roll	Y axis	θ_5	Unlimited	End-effector rotation (Yaw)

2.2 Link Lengths

Physical parameters of the robot arm:

$L_1 = 1.0$ unit (Base height)
 $L_2 = 0.8$ unit (Shoulder-Elbow)
 $L_3 = 0.6$ unit (Elbow-Wrist)
 $L_4 = 0.5$ unit (Wrist-End)

Total Maximum Reach: $L_1 + L_2 + L_3 + L_4 = 2.9$ units

2.3 Coordinate System

The project uses the standard robotics coordinate system: - **X axis:** Red - Horizontal (left/right) - **Y axis:** Green - Vertical (up/down) - **Z axis:** Blue - Depth (forward/backward)

Right-hand rule is applied.

3. Forward Kinematics

3.1 Mathematical Formulation

Forward kinematics computes the end-effector position $\mathbf{p} = [x, y, z]^T$ given joint angles vector $\mathbf{q} = [\theta_1, \theta_2, \theta_3, \theta_4, \theta_5]^T$.

Direct Matrix Multiplication Instead of Denavit-Hartenberg (DH) Approach

This implementation uses homogeneous transformation matrices directly instead of DH parameters:

$$T_0^5 = T_0^1 \cdot T_1^2 \cdot T_2^3 \cdot T_3^4 \cdot T_4^5$$

Transformation for each joint:

Joint 1 (Base - Y axis rotation):

$$T_0^1 = \text{Rot}(Y, \theta_1) \cdot \text{Trans}(0, L_1, 0)$$

Joint 2 (Shoulder - X axis rotation):

$$T_1^2 = \text{Rot}(X, \theta_2) \cdot \text{Trans}(0, L_2, 0)$$

Joint 3 (Elbow - X axis rotation):

$$T_2^3 = \text{Rot}(X, \theta_3) \cdot \text{Trans}(0, L_3, 0)$$

Joint 4 (Wrist Pitch - X axis rotation):

$$T_3^4 = \text{Rot}(X, \theta_4) \cdot \text{Trans}(0, L_4, 0)$$

Joint 5 (Wrist Roll - Y axis rotation):

$$T^{45} = \text{Rot}(Y, \theta_5)$$

3.2 Homogeneous Transformation Matrices

Rotation around Y axis:

$$R^y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation around X axis:

$$R^x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Translation matrix:

$$\text{Trans}(t) = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3.3 Code Implementation

```
Vector3 ComputeForwardKinematics(float theta1, float theta2, float theta3, float theta4,
float theta5)
{
    // Degree -> Radian conversion
    float t1 = DegreesToRadians(theta1);
    float t2 = DegreesToRadians(theta2);
    float t3 = DegreesToRadians(theta3);
    float t4 = DegreesToRadians(theta4);
    float t5 = DegreesToRadians(theta5);

    Matrix4 T = Identity;

    // Apply joint transformations in sequence
    T *= RotationY(t1) * Translation(0, L1, 0);
    T *= RotationX(t2) * Translation(0, L2, 0);
    T *= RotationX(t3) * Translation(0, L3, 0);
    T *= RotationX(t4) * Translation(0, L4, 0);
    T *= RotationY(t5);

    // Transform origin point with transformation matrix
    return TransformPosition(Vector3.Zero, T);
}
```

3.4 Computational Complexity

- **Time Complexity:** O(n) - n is number of joints (here n=5)

- **Space Complexity:** O(1) - Constant matrix size
-

4. Inverse Kinematics

4.1 Problem Definition

The inverse kinematics problem: Given target position $\mathbf{p}_{\text{tar}} = [x_t, y_t, z_t]^T$, find the joint angles vector $\mathbf{q} = [\theta_1, \theta_2, \theta_3, \theta_4, \theta_5]^T$ that brings the end-effector to this position.

This problem is typically: - **Multiple solutions** (multiple possible configurations) - **No solution** (target outside reachable workspace) - **Difficult analytical solution** (for 5+ DOF)

4.2 Geometric Analytical Solution Approach

This implementation uses a geometric approach that reduces the 5-DOF problem to a 3-DOF problem.

Step 1: Base Angle (ϑ_1) Calculation

Looking at target position from top (XZ plane):

$$\theta_1 = \text{atan2}(x_t, z_t)$$

Horizontal distance to target from base:

$$r = \sqrt{(x_t^2 + z_t^2)}$$

Step 2: Planar 2-Link IK Problem

After rotating the base, the problem is reduced to 2D. Now we work in (r, y) plane.

Simplification: Combine last two links to reduce 3-DOF system to 2-DOF:

$$L_2' = L_2 = 0.8 \text{ (Upper arm)}$$

$$L_3' = L_3 + L_4 = 0.6 + 0.5 = 1.1 \text{ (Total lower arm)}$$

Distance from shoulder joint to target position:

$$y' = y_t - L_1 \text{ (base height subtracted)}$$

$$d = \sqrt{(r^2 + y'^2)}$$

Reachability check:

```
if (d > L2' + L3') → Target unreachable, return false
```

Step 3: Elbow Angle Using Law of Cosines

From triangle geometry using law of cosines:

$$\cos(\theta_3) = (d^2 - L2'^2 - L3'^2) / (2 \cdot L2' \cdot L3')$$

Elbow angle:

$$\theta_3 = \arccos(\cos(\theta_3))$$

Note: $\cos(\theta_3)$ value must be clamped to [-1, 1] (numerical error prevention).

Step 4: Shoulder Angle (ϑ_2) Calculation

Sum of two component angles:

Angular orientation of target:

$$\phi = \arctan2(y', r)$$

Internal angle correction:

$$\psi = \arctan2(L_3' \cdot \sin(\theta_3), L_2' + L_3' \cdot \cos(\theta_3))$$

Final shoulder angle:

$$\theta_2 = \pi/2 - (\phi + \psi)$$

Note: The $\pi/2$ term comes from the robot's vertical starting posture.

Step 5: Distribution of Wrist Angles

Distribute calculated total elbow bending between θ_3 and θ_4 :

$$\begin{aligned}\theta_3_{\text{final}} &= \theta_3_{\text{total}} \times 0.6 \quad (\text{Elbow } 60\%) \\ \theta_4_{\text{final}} &= \theta_3_{\text{total}} \times 0.4 \quad (\text{Wrist } 40\%)\end{aligned}$$

Last joint kept fixed:

$$\theta_5 = 0$$

4.3 Code Implementation

```
bool SolveIK(Vector3 target, out float theta1, out float theta2,
             out float theta3, out float theta4, out float theta5)
{
    // Step 1: Base angle
    float t1 = atan2(target.X, target.Z);
    float r = sqrt(target.X2 + target.Z2);

    // Step 2: 2D problem
    float y = target.Y - L1;
    float d = sqrt(r2 + y2);

    float l1 = L2;
    float l2 = L3 + L4;

    // Reachability check
    if (d > l1 + l2) return false;

    // Step 3: Law of cosines
    float cosQ3 = (d2 - l12 - l22) / (2 * l1 * l2);
```

```

cosQ3 = Clamp(cosQ3, -1, 1);
float q3 = acos(cosQ3);

// Step 4: Shoulder angle
float phi = atan2(y, r);
float psi = atan2(12·sin(q3), 11 + 12·cos(q3));
float q2 = π/2 - (phi + psi);

// Step 5: Distribute angles
θ1 = RadiansToDegrees(t1);
θ2 = RadiansToDegrees(q2);

float totalElbow = RadiansToDegrees(q3);
θ3 = totalElbow × 0.6;
θ4 = totalElbow × 0.4;
θ5 = 0;

return true;
}

```

4.4 Solution Properties

- **Type:** Geometric analytical solution
 - **Speed:** O(1) - Constant time
 - **Advantages:**
 - Very fast
 - Deterministic
 - Closed-form solution
 - **Disadvantages:**
 - Provides single solution (if multiple configurations possible)
 - 5th joint fixed ($θ_5 = 0$)
 - Does not fully check joint limits
-

5. Jacobian Matrix and Velocity Kinematics

5.1 Jacobian Matrix Theory

The Jacobian matrix $J(\mathbf{q})$ defines the relationship between joint velocities $\dot{\mathbf{q}}$ and end-effector velocity $\dot{\mathbf{x}}$:

$$\dot{\mathbf{x}} = J(\mathbf{q}) \cdot \dot{\mathbf{q}}$$

Where: - $\dot{\mathbf{x}} = [\dot{x}, \dot{y}, \dot{z}]^T \in \mathbb{R}^3$: Linear velocity in Cartesian space - $\dot{\mathbf{q}} = [\dot{\theta}_1, \dot{\theta}_2, \dot{\theta}_3, \dot{\theta}_4, \dot{\theta}_5]^T \in \mathbb{R}^5$: Joint angular velocities - $J(\mathbf{q}) \in \mathbb{R}^{3 \times 5}$: Jacobian matrix (3 rows × 5 columns)

5.2 Jacobian Matrix Structure

$$J = \begin{bmatrix} \frac{\partial x}{\partial \theta_1} & \frac{\partial x}{\partial \theta_2} & \frac{\partial x}{\partial \theta_3} & \frac{\partial x}{\partial \theta_4} & \frac{\partial x}{\partial \theta_5} \\ \frac{\partial y}{\partial \theta_1} & \frac{\partial y}{\partial \theta_2} & \frac{\partial y}{\partial \theta_3} & \frac{\partial y}{\partial \theta_4} & \frac{\partial y}{\partial \theta_5} \\ \frac{\partial z}{\partial \theta_1} & \frac{\partial z}{\partial \theta_2} & \frac{\partial z}{\partial \theta_3} & \frac{\partial z}{\partial \theta_4} & \frac{\partial z}{\partial \theta_5} \end{bmatrix}$$

Each column j shows the effect of the j th joint on end-effector position.

5.3 Numerical Jacobian Calculation Using Finite Differences

Since analytical derivatives are complex, numerical approximation is used (finite difference method):

Central Difference Approximation:

$$\frac{\partial f}{\partial x} \approx [f(x + h) - f(x)] / h$$

For each joint:

$$J[:, i] = [FK(\theta_1, \dots, \theta_i + h, \dots, \theta_5) - FK(\theta_1, \dots, \theta_i, \dots, \theta_5)] / h$$

Where: - $h = 0.1^\circ$ (derivative step size) - **FK()** = Forward kinematics function

5.4 Code Implementation

```
float[,] ComputeJacobian(float theta1, float theta2, float theta3, float theta4, float theta5)
{
    float[,] J = new float[3, 5]; // 3x5 matrix
    float h = 0.1°; // Derivative step

    Vector3 current = FK(theta1, theta2, theta3, theta4, theta5);

    // For each column (each joint)
    for (int i = 0; i < 5; i++)
    {
        // Perturb ith angle by h
        Vector3 perturbed = FK_with_perturbation(i, h);

        // Numerical derivative
        Vector3 column = (perturbed - current) / Radians(h);

        J[0, i] = column.X;
        J[1, i] = column.Y;
        J[2, i] = column.Z;
    }

    return J;
}
```

5.5 Velocity Kinematics Calculation

Calculate end-effector velocity for given joint velocities:

```

Vector3 ComputeEndEffectorVelocity(float[,] J, float[] q_dot)
{
    // v = J * q̇
    Vector3 v = Vector3.Zero;

    for (int row = 0; row < 3; row++)           // X, Y, Z components
    {
        for (int col = 0; col < 5; col++) // 5 joints
        {
            v[row] += J[row, col] * q_dot[col];
        }
    }

    return v;
}

```

Velocity magnitude:

$$|v| = \sqrt{v_x^2 + v_y^2 + v_z^2}$$

5.6 Importance of Jacobian

The Jacobian matrix is critical in robot control:

1. **Velocity Control:** Cartesian space velocity targets → Joint velocities
2. **Force Control:** Joint torques → End-effector force
3. **Singularity Analysis:** $\det(J \cdot J^T) \approx 0$ → Singular configuration
4. **Inverse Kinematics Optimization:** In numerical IK solutions
5. **Trajectory Tracking:** Real-time path control

5.7 Computational Complexity

- **Time:** $O(n^2)$ - n number of joints (5 FK calls × each $O(n)$)
 - **Space:** $O(n)$ - $3 \times n$ matrix storage
-

6. Trajectory Planning

6.1 What is Trajectory Planning?

Trajectory planning ensures the robot transitions from start configuration to target configuration in a **smooth** and **continuous** manner. Instead of sudden jumps, gradual change occurs over time.

6.2 Linear Interpolation (LERP)

The simplest trajectory planning method is linear interpolation:

Mathematical Formula:

$$q(t) = q_{\text{start}} + (q_{\text{target}} - q_{\text{start}}) \cdot u(t)$$

Where: - $\mathbf{q}(t)$: Joint angles at time t - $\mathbf{q}_{\text{start}}$: Starting angles - $\mathbf{q}_{\text{target}}$: Target angles - $u(t)$: Normalization parameter [0, 1]

Time normalization:

$$u(t) = t / T_{\text{duration}}$$

$$u(t) \in [0, 1] \quad \text{where:}$$

$$\begin{aligned} u(0) &= 0 & \rightarrow \text{Start} \\ u(T) &= 1 & \rightarrow \text{End} \end{aligned}$$

6.3 Joint Space vs Cartesian Space

This implementation: Joint Space

Advantages: - Independent interpolation for each joint - Easy joint limit control - Simple and fast computation - Unaffected by singularities

Disadvantages: - End-effector doesn't follow straight line - Unpredictable trajectory in Cartesian space

Alternative: Cartesian Space - End-effector follows straight path - Requires IK solution at each step - Computationally intensive - May encounter singularities

6.4 Code Implementation

```
// LERP Function
float Lerp(float a, float b, float t)
{
    return a + (b - a) * t;
}

// Trajectory Update (Every Frame)
void UpdateTrajectory(float deltaTime)
{
    if (!trajectoryActive) return;

    // Advance time
    trajectoryTime += deltaTime;

    // Normalization parameter
    float u = trajectoryTime / trajectoryDuration;

    // End check
    if (u >= 1.0f)
    {
        u = 1.0f;
        trajectoryActive = false; // Trajectory complete
    }

    // LERP for each joint
```

```

θ1 = Lerp(θ1_start, θ1_target, u);
θ2 = Lerp(θ2_start, θ2_target, u);
θ3 = Lerp(θ3_start, θ3_target, u);
θ4 = Lerp(θ4_start, θ4_target, u);
θ5 = Lerp(θ5_start, θ5_target, u);
}

```

6.5 Advanced Trajectory Profiles

Linear interpolation causes sudden velocity changes. Smoother profiles:

S-Curve (Sigmoid):

$$u_{smooth}(t) = 3u^2 - 2u^3$$

Cosine Profile:

$$u_{smooth}(t) = (1 - \cos(\pi u)) / 2$$

5th Order Polynomial:

$$u_{smooth}(t) = 10u^3 - 15u^4 + 6u^5$$

These profiles:
- Guarantee **zero velocity** at start and end
- Provide continuous acceleration
- More suitable for mechanical systems

6.6 Time Parameters

Project default values:

```

T_duration = 2.0 seconds (Total motion duration)
FPS = 60                (Frames/second)
Total Frames = 120       (2.0 × 60)

```

Angle change per frame:

$$\Delta\theta = (\theta_{target} - \theta_{start}) / 120$$

7. 3D Visualization

7.1 OpenGL Render Pipeline

The project performs real-time 3D rendering using OpenTK (OpenGL).

Render Steps:

1. **Clear Buffer:** Clear screen and depth buffer
2. **Projection Matrix:** Set up perspective camera
3. **View Matrix:** Camera position and orientation
4. **Model Matrix:** Transformation for each object
5. **Draw Calls:** Draw geometric primitives

6. **Swap Buffer:** Double buffering

7.2 Camera System

Perspective Projection:

```
Projection = CreatePerspective(  
    FOV = 45°,           // Field of view  
    Aspect = W/H,        // Aspect ratio  
    Near = 0.1,          // Near plane  
    Far = 100.0          // Far plane  
)
```

Spherical Camera Movement:

Camera position in polar coordinates:

```
x = distance · cos(angleX) · cos(angleY)  
y = distance · sin(angleX)  
z = distance · cos(angleX) · sin(angleY)
```

Parameters: - **distance**: Distance to center [2, 20] - **angleX**: Vertical angle [-89°, +89°] - **angleY**: Horizontal angle [Unlimited]

View Matrix:

```
ViewMatrix = LookAt(  
    eye = cameraPosition,      // Camera location  
    target = (0, 0, 0),        // Look-at point (origin)  
    up = (0, 1, 0)            // Up direction vector  
)
```

7.3 Lighting Model (Phong Lighting)

OpenGL fixed function pipeline lighting:

Light Components:

```
I_total = I_ambient + I_diffuse + I_specular
```

Ambient (Ambient Light):

```
I_ambient = K_a · L_a  
K_a = 0.3  (Material albedo)
```

Diffuse (Diffuse Light):

```
I_diffuse = K_d · L_d · max(N · L, 0)  
K_d = 1.0  
L_d = (1, 1, 1)  (White light)
```

Light Position:

```
Light_position = (5, 10, 10, 1)
```

7.4 Geometry Drawing

Cylinder Primitive:

Robot arms are modeled as cylinders (more realistic than cubes):

```
void DrawCylinder(float radius, float height, int segments)
{
    // Side surface (QuadStrip)
    for (int i = 0; i <= segments; i++)
    {
        float angle = 2π · i / segments;
        float x = radius · cos(angle);
        float z = radius · sin(angle);

        // Normal vector (for Lighting)
        Normal(x, 0, z);

        // Top and bottom vertex
        Vertex(x, height, z);
        Vertex(x, 0, z);
    }

    // Top and bottom caps
    DrawDisk(radius, 0);      // Bottom cap
    DrawDisk(radius, height); // Top cap
}
```

Dimensions: - Link radius: 0.1 unit - Joint radius: 0.18 unit - Segment count: 24 (smooth cylinder)

7.5 Hierarchical Transformation

Robot drawing in tree structure (scene graph):

```
World
└ Base (θ1)
  └ Link1 (L1)
    └ Shoulder (θ2)
      └ Link2 (L2)
        └ Elbow (θ3)
          └ Link3 (L3)
            └ Wrist (θ4)
              └ Link4 (L4)
                └ WristRoll (θ5)
                  └ Gripper
```

OpenGL Matrix Stack:

```
PushMatrix();           // Save current matrix
Rotate(θ);             // Apply rotation
```

```

Translate(0, L, 0);    // Move forward
DrawLink();           // Draw Link
DrawChild();          // Draw child element
PopMatrix();          // Restore previous matrix

```

This method: - Parent transformations automatically propagate to children - Uses local coordinate systems - Clean and modular code

7.6 Color Scheme

Element	Color	RGB
Links	Orange	(1.0, 0.5, 0.0)
Joints	Gray	(0.6, 0.6, 0.6)
Gripper Base	Dark Gray	(0.3, 0.3, 0.3)
Gripper Fingers	Light Gray	(0.8, 0.8, 0.8)
Ground	Dark Gray	(0.2, 0.2, 0.2)
X Axis	Red	(1, 0, 0)
Y Axis	Green	(0, 1, 0)
Z Axis	Blue	(0, 0, 1)

7.7 Gripper Animation

Gripper consists of 3 fingers (120° apart):

Finger position = $0.05 + (0.16 - 0.05) \cdot t$

$t = (\text{gap} - \text{gap_min}) / (\text{gap_max} - \text{gap_min}) [0, 1]$

- **gap_min** = 0.03 (Closed)
 - **gap_max** = 0.25 (Open)
-

8. User Guide

8.1 Installation

Requirements:

.NET Framework 4.7.2 Developer Pack
Visual Studio 2019/2022 or VS Code
OpenTK 3.3.3 (automatic via NuGet)

Build:

```
cd RobotArm5DOF
dotnet build
```

Run:

```
dotnet run
# or
.\RobotArm5DOF\bin\Debug\RobotArm5DOF.exe
```

8.2 Keyboard Controls

Joint Controls

Key	Function	Motion
Q / A	Base (θ_1)	Left/Right rotation
W / S	Shoulder (θ_2)	Forward/Backward tilt
E / D	Elbow (θ_3)	Arm bending
R / F	Wrist Pitch (θ_4)	Up/Down
T / G	Wrist Roll (θ_5)	Rotation

Motion speed: 60°/second

Gripper Control

Key	Function
X	Open Gripper
Z	Close Gripper

Motion speed: 0.6 units/second

Camera Controls

Key	Function
← / →	Horizontal rotation (angleY)
↑ / ↓	Vertical rotation (angleX)
Page Up	Zoom In
Page Down	Zoom Out

Camera limits: - Vertical: [-89°, +89°] - Distance: [2, 20] units

Inverse Kinematics

Key	Function	Description
I	IK Instant	Go to target position instantly
Y	IK Smooth	Go to target position with trajectory (2 seconds)

Target position: (1.2, 1.5, 0.4)

General

Key	Function
ESC	Exit

8.3 Window Title Information

Real-time information is displayed in the program title:

5DOF Robot | Ang:(45,30,-20,15,0) | EE:(1.23,2.45,0.67) | |Vee|:0.34

- **Ang:** Current joint angles $[\theta_1, \theta_2, \theta_3, \theta_4, \theta_5]$
 - **EE:** End-effector position [x, y, z]
 - **|Vee|:** End-effector velocity magnitude (for default joint velocities)
-

9. Technical Details

9.1 Performance Characteristics

Computational Load (Per Frame):

Operation	Complexity	Call Count	Time (approx)
Forward Kinematics	$O(5)$	7	< 0.1 ms
Jacobian	$O(25)$	1	< 0.5 ms
Rendering	$O(N)$	1	1-2 ms
Total			< 3 ms

Frame Rate: 60 FPS (16.67 ms/frame) → Sufficient margin

9.2 Numerical Stability

Potential Problems:

1. **Inverse Kinematics:**
 - Acos domain error: $\cos(\theta) \in [-1, 1]$ check
 - Solution: Clamp(value, -1, 1)
2. **Jacobian Calculation:**
 - Division by zero: h value should not be too small
 - Solution: $h = 0.1^\circ$ (sufficiently large)
3. **Gimbal Lock:**
 - Consecutive X axis rotations → Potential singularity
 - Solution: Control with joint limits

Numerical Precision: - Float (32-bit): ~7 decimal places - Angle resolution: 0.01° (sufficient)

9.3 Coordinate System Consistency

Critical: DrawRobot() and ComputeForwardKinematics() functions must use **exactly** the same transformation sequence.

Otherwise: - Visual and calculated positions will be inconsistent - IK solution goes to wrong targets - Jacobian matrix calculated incorrectly

Verification Method: For any joint configuration:

```
visual_position ≈ FK(θ1, θ2, θ3, θ4, θ5)
```

9.4 Joint Limits

```
// Physical Limits (in degrees)
θ2 ∈ [-90, +90] // Shoulder
θ3 ∈ [-150, +150] // Elbow
θ4 ∈ [-90, +90] // Wrist
```

Why necessary: - Physical robots have mechanical limits - Prevents self-collision - Realistic motion range

Limit Check:

```
θ2 = Clamp(θ2, -90, 90);
```

9.5 Singularities

What is Singularity?

The robot loses degrees of freedom in certain configurations. The Jacobian matrix determinant becomes zero.

Potential Singularities in This System:

1. **Base Singularity:**

- Target above origin ($r = 0$)
- θ_1 becomes undefined
- Solution: Minimal r threshold

2. **Arm Fully Extended:**

- $d = L_2 + L_3 + L_4$
- $\theta_3 \approx 0$ (Arm straight)
- Small motion requires large joint changes

3. **Arm Fully Bent:**

- $\theta_3 \approx \pm 150^\circ$
- Wrist and elbow overlap

Singularity Check:

```
det(J · JT) < ε → Singularity near
```

Where ϵ is small threshold value (e.g. 0.001).

10. Mathematical References

10.1 Summary of Formulas Used

Concept	Formula	Section
Forward Kinematics	$\mathbf{p} = \text{FK}(\mathbf{q}) = T_0^5 \cdot [0,0,0,1]^T$	3
Inverse Kinematics	$\theta_1 = \text{atan2}(x, z)$	4
Law of Cosines	$\cos(C) = (a^2 + b^2 - c^2) / (2ab)$	4
Jacobian	$J[i,j] = \partial p_i / \partial \theta_j$	5
Numerical Derivative	$f'(x) \approx [f(x+h) - f(x)] / h$	5
Velocity Kinematics	$\mathbf{v} = J(\mathbf{q}) \cdot \dot{\mathbf{q}}$	5
LERP	$q(t) = q_0 + (q_1 - q_0) \cdot t$	6
Spherical Camera	$x = d \cdot \cos(\varphi) \cdot \cos(\theta)$	7

10.2 Notation Table

Symbol	Meaning	Unit
θ_i	i th joint angle	degree or radian
\mathbf{q}	Joint angles vector $[\theta_1, \dots, \theta_5]^T$	-
\mathbf{p}	Cartesian position $[x, y, z]^T$	unit
L_i	i th link length	unit
J	Jacobian matrix (3×5)	unit/radian
\mathbf{v}	Linear velocity vector	unit/second
$\dot{\mathbf{q}}$	Joint angular velocities	radian/second
T	Homogeneous transformation matrix (4×4)	-

10.3 Coordinate Systems

World Coordinates (World Frame): - Origin: Robot base - Y axis: Up (anti-gravity)

Joint Coordinates (Joint Frame): - Each joint has its local coordinate system - Rotation axis: X or Y

End-Effector Coordinates (End-Effector Frame): - At last joint center - Orientation determined by θ_5

11. References and Further Reading

11.1 Fundamental Robotics Books

1. **Craig, J.J.** (2005). *Introduction to Robotics: Mechanics and Control* (3rd ed.). Pearson Education.
 - Fundamental reference for kinematics and dynamics
 - Detailed explanation of Forward/Inverse Kinematics and Jacobian

2. **Spong, M.W., Hutchinson, S., & Vidyasagar, M.** (2020). *Robot Modeling and Control* (2nd ed.). Wiley.
 - DH parameters and transformations
 - Robot control algorithms
3. **Lynch, K.M., & Park, F.C.** (2017). *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press.
 - Modern approaches and screw theory
 - [Free online: modernrobotics.org](http://modernrobotics.org)
4. **Siciliano, B., Sciavicco, L., Villani, L., & Oriolo, G.** (2010). *Robotics: Modelling, Planning and Control*. Springer.
 - Advanced control theory
 - Trajectory planning and manipulation
5. **Corke, P.I.** (2017). *Robotics, Vision and Control: Fundamental Algorithms in MATLAB* (2nd ed.). Springer.
 - Practical MATLAB implementations
 - Robotics library examples

11.2 Computer Graphics

6. **Shirley, P., & Marschner, S.** (2009). *Fundamentals of Computer Graphics* (3rd ed.). AK Peters/CRC Press.
 - Transformation matrices and render pipeline
 - 3D graphics mathematics
7. **Shreiner, D., et al.** (2013). *OpenGL Programming Guide* (8th ed.). Addison-Wesley.
 - OpenGL API reference (Red Book)
 - Shader programming fundamentals

11.3 Mathematical Foundations

8. **Strang, G.** (2016). *Introduction to Linear Algebra* (5th ed.). Wellesley-Cambridge Press.
 - Matrix operations and linear transformations
 - Supported by MIT OpenCourseWare lectures
9. **Press, W.H., et al.** (2007). *Numerical Recipes: The Art of Scientific Computing* (3rd ed.). Cambridge University Press.
 - Numerical derivatives and optimization
 - Practical algorithm implementations

11.4 C# and .NET Programming

10. **Albahari, J., & Bahari, B.** (2021). *C# 10 in a Nutshell*. O'Reilly Media.
 - C# language features and .NET framework
 - LINQ and async programming

11.5 Online Resources

11. [OpenTK Documentation](http://opentk.com)

- Complete OpenTK API reference
12. [Learn OpenGL](#)
- Comprehensive modern OpenGL tutorial
13. [Robot Academy - QUT](#)
- Peter Corke's video lecture series
 - Free robotics education
14. [Stanford CS223A - Introduction to Robotics](#)
- Prof. Oussama Khatib video lectures
 - Detailed explanation of kinematics and dynamics
15. [IEEE Robotics and Automation Society](#)
- Academic papers and conferences
 - Robotics research trends
-

Appendices

A. Code Structure

```
RobotArm5DOF/
├── Program.cs           (Main code - 617 lines)
│   ├── Kinematic Variables ( $\theta_1$ - $\theta_5$ ,  $L_1$ - $L_4$ )
│   ├── OnLoad()           (OpenGL initialization)
│   ├── OnUpdateFrame()    (Physics and input)
│   ├── OnRenderFrame()    (Render loop)
│   ├── DrawRobot()         (Robot drawing)
│   ├── DrawCylinder()     (Geometry)
│   ├── ComputeForwardKinematics()
│   ├── SolveIK()           (Inverse kinematics)
│   ├── ComputeJacobian()   (Jacobian matrix)
│   └── Lerp()              (Trajectory)
├── RobotArm5DOF.csproj
└── packages.config
    README.md             (This file)
```

B. Quick Reference Card

Joint Control:

Q/A: Base (θ_1)	W/S: Shoulder (θ_2)	E/D: Elbow (θ_3)
R/F: Wrist (θ_4)	T/G: Roll (θ_5)	X/Z: Gripper

Camera:

Arrows: Rotate	PgUp/PgDn: Zoom	ESC: Exit
----------------	-----------------	-----------

IK:

I: Instant go	Y: Smooth go
---------------	--------------