

5 Serbestlik Dereceli (5-DOF) Robot Kol Simülasyonu

İçindekiler

1. Proje Genel Bakış
 2. Kinematik Yapı ve Eklem Konfigürasyonu
 3. İleri Kinematik (Forward Kinematics)
 4. Ters Kinematik (Inverse Kinematics)
 5. Jacobian Matrisi ve Hız Kinematığı
 6. Trajektori Planlama
 7. 3D Görselleştirme
 8. Kullanım Kılavuzu
 9. 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 Sınırları	Açıklama
J1	Taban (Base)	Y ekseni	θ_1	Sınırsız	Taban rotasyonu (Yaw)
J2	Omu z (Shoulder)	X ekseni	θ_2	$[-90^\circ, +90^\circ]$	Öne/arkaya eğilme (Pitch)
J3	Dirsek (Elbow)	X ekseni	θ_3	$[-150^\circ, +150^\circ]$	Kol bükülmesi (Pitch)
J4	Bilek (Wrist Pitch)	X ekseni	θ_4	$[-90^\circ, +90^\circ]$	Bilek yukarı/aşağı (Pitch)
J5	Bilek Roll (Wrist Roll)	Y ekseni	θ_5	Sınırsız	Uç efektör dönüşü (Yaw)

2.2 Link Uzunlukları

Robot kolun fiziksel parametreleri:

$L_1 = 1.0$ birim (Taban yüksekliği)
 $L_2 = 0.8$ birim (Omuz-Dirsek arası)
 $L_3 = 0.6$ birim (Dirsek-Bilek arası)
 $L_4 = 0.5$ birim (Bilek-Uç arası)

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 eksen:** Kırmızı - Yatay (sağ/sol) - **Y eksen:** Yeşil - Dikey (yukarı/aşağı) - **Z eksen:** Mavi - Derinlik (ileri/geri)

Sağ el kuralı (right-hand rule) uygulanır.

3. İleri Kinematik (Forward Kinematics)

3.1 Matematiksel Formülasyon

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 eksen dönüşü):

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

Eklem 2 (Omuz - X eksen dönüşü):

$$T^1_2 = \text{Rot}(X, \theta_2) \cdot \text{Trans}(\theta, L_2, \theta)$$

Eklem 3 (Dirsek - X eksen dönüşü):

$$T^2_3 = \text{Rot}(X, \theta_3) \cdot \text{Trans}(\theta, L_3, \theta)$$

Eklem 4 (Bilek Pitch - X eksenî dönüşü):

$$T^{34} = \text{Rot}(X, \theta_4) \cdot \text{Trans}(0, L_4, 0)$$

Eklem 5 (Bilek Roll - Y eksenî dönüşü):

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

3.2 Homojen Transformasyon Matrisleri

Y eksenî 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 eksenî 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:

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

3.3 Kod İmplementasyonu

```
Vector3 ComputeForwardKinematics(float  $\theta_1$ , float  $\theta_2$ , float  $\theta_3$ , float  $\theta_4$ ,  
float  $\theta_5$ )
```

```
{
```

```
    // Derece -> Radyan dönüşümü
```

```
    float t1 = DegreesToRadians( $\theta_1$ );
```

```
    float t2 = DegreesToRadians( $\theta_2$ );
```

```
    float t3 = DegreesToRadians( $\theta_3$ );
```

```
    float t4 = DegreesToRadians( $\theta_4$ );
```

```
    float t5 = DegreesToRadians( $\theta_5$ );
```

```
    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 $\mathbf{p}_{tar} = [x_t, y_t, z_t]^T$ için, uç efektörün bu pozisyona ulaşmasını sağlayacak eklem açıları $\mathbf{q} = [\theta_1, \theta_2, \theta_3, \theta_4, \theta_5]^T$ vektörünü bulmak.

Bu problem genellikle: - **Çok çözümlü** (multiple solutions) - **Çözumsuz** (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 pozisyona üstten bakıldığında (XZ düzlemi):

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

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 çalışırız.

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 pozisyona mesafe:

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

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

Erişilebilirlik kontrolü:

if $(d > L_2' + L_3')$ → 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:

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

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

Adım 4: Omuz Açısı (θ_2) Hesabı

İki bileşen açının 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:

$$\theta_{3_final} = \theta_{3_total} \times 0.6 \quad (\text{Dirsek } \%60)$$

$$\theta_{4_final} = \theta_{3_total} \times 0.4 \quad (\text{Bilek } \%40)$$

Son eklem sabit tutulur:

$$\theta_5 = 0$$

4.3 Kod İmplementasyonu

```
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 = π/2 - (phi + psi);

// Adım 5: Açılar dağıt
θ1 = RadiansToDegrees(t1);
θ2 = RadiansToDegrees(q2);

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

5.1 Jacobian Matrisi Teorisi

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

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

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 - $\mathbf{J}(\mathbf{q}) \in \mathbb{R}^{3 \times 5}$: Jacobian matrisi (3 satır \times 5 sütun)

5.2 Jacobian Matris Yapısı

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

Her sütun j, j'inci eklemün 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:

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

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

5.4 Kod İmplementasyonu

```
float[,] ComputeJacobian(float  $\theta_1$ , float  $\theta_2$ , float  $\theta_3$ , float  $\theta_4$ , float  $\theta_5$ )
{
    float[,] J = new float[3, 5]; // 3x5 matris
    float h = 0.1°; // Türev adımı

    Vector3 current = FK( $\theta_1$ ,  $\theta_2$ ,  $\theta_3$ ,  $\theta_4$ ,  $\theta_5$ );

    // 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 uç efektör hızını hesapla:

```
Vector3 ComputeEndEffectorVelocity(float[, ] J, float[] q_dot)
{
    //  $v = J \cdot \dot{q}$ 
    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. **Singülarite Analizi:** $\det(J \cdot J^T) \approx 0 \rightarrow$ 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 çağrısı \times 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 İnterpolasyon (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:

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

$u(t) \in [0, 1]$ where:

$u(0) = 0 \rightarrow$ Başlangıç

$u(T) = 1 \rightarrow$ Bitiş

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 - Singülaritelerden etkilenmez

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

Alternatif: Kartezyen Uzay - Uç efektör düz yol izler - Her adımda IK çözümü gerekir - Hesaplama yoğun - Singülaritelerle 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:

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

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 nokta (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 Işığı):

```
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 Yığıcı (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 Eksen	Kırmızı	(1, 0, 0)
Y Eksen	Yeşil	(0, 1, 0)
Z Eksen	Mavi	(0, 0, 1)

7.7 Gripper Animasyonu

Gripper 3 parmağtan oluşur (120° açılarda):

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

Eklem Kontrolleri

Tuş	Fonksiyon	Hareket
Q / A	Taban (θ_1)	Sola/Sağa 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°/saniye

Gripper Kontrolü

Tuş	Fonksiyon
X	Gripper Aç
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: $[-89^\circ, +89^\circ]$ - Uzaklık: $[2, 20]$ birim

Ters Kinematik

Tuş	Fonksiyon	Açıklama
I	IK Anında	Hedef pozisyona anında git
Y	IK Yumuşak	Hedef pozisyona 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 Yüğü (Her Frame):

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

Potansiyel Problemler:

- Ters Kinematik:**
 - Acos domain hatası: $\cos(\theta) \in [-1, 1]$ kontrolü
 - Çözüm: $\text{Clamp}(\text{value}, -1, 1)$
- Jacobian Hesaplama:**

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

3. Gimbal Lock:

- X eksenini dönüşleri art arda \rightarrow Potansiyel tekil nokta
- Çözüm: Eklem limitlerle kontrol

Sayısal Hassasiyet: - Float (32-bit): ~ 7 ondalık basamak - Açık 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:

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

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 gerekli: - 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 Singülarite (Tekil Noktalar)

Singülarite Nedir?

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

Bu Sistemde Potansiyel Singülariteler:

1. Taban Singülaritesi:

- 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

Singülarite Kontrolü:

$\det(J \cdot J^T) < \epsilon \rightarrow$ Singülarite yakın

Burada ϵ küçük eşik değeridir (örn. 0.001).

10. Matematiksel Referanslar

10.1 Kullanılan Formüller Özeti

Kavram	Formül	Bölüm
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
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 Kinematığı	$\mathbf{v} = J(\mathbf{q}) \cdot \dot{\mathbf{q}}$	5
LERP	$q(t) = q_0 + (q_1 - q_0) \cdot t$	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 $[\theta_1, \dots, \theta_5]^T$	-
\mathbf{p}	Kartezyen pozisyon $[x, y, z]^T$	birim
L_i	i'inci link uzunluğu	birim
\mathbf{J}	Jacobian matrisi (3x5)	birim/radyan
\mathbf{v}	Lineer hız vektörü	birim/saniye
$\dot{\mathbf{q}}$	Eklem açısal hızları	radyan/saniye
T	Homojen transformasyon matrisi (4x4)	-

10.3 Koordinat Sistemleri

Dünya Koordinatları (World Frame): - Orijin: Robot tabanı - Y eksen: Yukarı (gravitasyon tersi) **Eklem Koordinatları (Joint Frame):** - Her eklem lokal koordinat sistemine sahip - Dönüş eksen: 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 Yapısı

RobotArm5DOF /

— Program.cs	(Ana kod - 617 satır)
— Kinematic Variables	(θ_1 - θ_5 , L_1 - L_4)
— 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 (θ_1)	W/S: Omuz (θ_2)	E/D: Dirsek (θ_3)
R/F: Bilek (θ_4)	T/G: Roll (θ_5)	X/Z: Gripper

Kamera:

Oklar: Döndür

PgUp/PgDn: Zoom

ESC: Çıkış

IK:

I: Anında git

Y: Yumuşak git