

10. Mellékletek

```
using JetBrains.Annotations;
using System.Collections;
using System.Collections.Generic;
using System.Threading;
using UnityEngine;
using UnityEngine.Animations;
using UnityEngine.Internal;

public class Viewercontrol : MonoBehaviour
{
    //Kamera mozgatási sebességek illetve az egérrel történő mozgás
    //érzékenységeinek megadása
    public float sebesseg = 750f;
    public float egererzekenység = 750f;
    public float fel_le_sebesseg = 750f;

    //Alap koordináták megadása
    //A szimuláció elindításakor ez lesz a kamera kiinduló pozíciója illetve
    //a "Kamera visszaállítása" gomb is ide állítja vissza a kamerát
    public float Alap_x = 1500;
    public float Alap_y = 1000;
    public float Alap_z = -500;
    public float Alap_pitch = 24;
    public float Alap_yaw = -90;

    //kezdeti nézőpont RPY szögeinek megadása
    //A szimuláció elindításakor ez lesz a kamera kiinduló elfordulása
    //illetve a "Kamera visszaállítása" gomb is ide állítja vissza a kamerát
    float pitch = 24;
    float yaw = -90;

    //"Kamera visszaállítása" gomb megnyomására kamera alaphelyzetbe
    //állítás
    public void Cam_reset()
    {
        transform.position = new Vector3(Alap_x, Alap_y, Alap_z);
        yaw = Alap_yaw;
        pitch = Alap_pitch;
    }

    //Egér zárt vagy feloldott állapotát tároló változó
    bool egerzarva = false;

    // Az update metódus minden egyes frame frissítésakor meghívásra kerül
    void Update()
    {
        //WASD-gombokkal translációs mozgások a térben
        transform.Translate(Vector3.right * Input.GetAxis("Horizontal") *
            sebesseg * Time.deltaTime);
        transform.Translate(Vector3.forward * Input.GetAxis("Vertical") *
            sebesseg * Time.deltaTime);

        //space vagy shift nyomva van-e
```

```
float fel = 0;
if (Input.GetKey(KeyCode.Space))
    fel = 1f;
if (Input.GetKey(KeyCode.LeftShift))
    fel = -1f;
//fel-le mozgás amennyiben a space vagy a shift gombok nyomva vannak
transform.position += Vector3.up * fel * fel_le_sebesseg *
    Time.deltaTime;

//Ha a jobb egérgomb le van nyomva akkor forgás a térben
if (Input.GetMouseButton(1))
{
    //Ha az egér nincs az ablakba zárva akkor oda zárása illetve
    láthatatlanná tétele
    if (!egerzarva)
    {
        Cursor.lockState = CursorLockMode.Locked;
        Cursor.visible = false;
        egerzarva = true;
    }
    //kamera yaw és pitch szögeinek beállítása az egér
    elmozdulásának megfelelően
    yaw += Input.GetAxis("Mouse X") * egererzekenyseg *
        Time.deltaTime;
    pitch -= Input.GetAxis("Mouse Y") * egererzekenyseg *
        Time.deltaTime;
}
else if(egerzarva) //Ha az egér az ablakhoz van zárva de már nincs
erre szükség akkor feloldása és láthatóvá tétele
{
    Cursor.lockState = CursorLockMode.None;
    Cursor.visible = true;
    egerzarva = false;
}

//Új megváltozott yaw és pitch szögek átadása a kamerának
transform.eulerAngles = new Vector3(pitch, yaw, 0f);

//pitch maximum és minimum értékek tartása, hogy ne lehessen fejfel
lefelé fordulni a kamerával
if (pitch > 90)
    pitch = 90;
if (pitch < -90)
    pitch = -90;
}
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class SetJointAngles : MonoBehaviour
{
    //InputFieldek meghívása a UI-ról, hogy be lehessen olvasni belőlük a
    kívánt csuklókoordinátákat
    public InputField J1input_text;
    public InputField J2input_text;
    public InputField J3input_text;
    public InputField J4input_text;
    public InputField J5input_text;
    public InputField J6input_text;

    //Joint GameObjectek meghívása, hogy lehessen transzformációkat
    végrehajtani rajtuk
    public GameObject Joint1;
    public GameObject Joint2;
    public GameObject Joint3;
    public GameObject Joint4;
    public GameObject Joint5;
    public GameObject Joint6;

    //A JointSet metódus a beállít gombra kattintáskor kerül meghívásra
    //A robotot pozícióba állítja a UI-ról beolvasott csuklókoordináták
    alapján
    public void JointSet()
    {
        {
            //Minden szegmens esetében először meghatározásra kerül az
            elforgatás tengelye
            //Másodszor létrehozásra kerül egy quaternion AngleAxis
            transzformáció, ami egy tengely körüli megadott szöggel való
            elforgatás
            //Utána végrehajtódik a szegmensen az összes előtte lévő
            szegmens transzformációja majd a sajátja is

            //1-es csukló:
            Vector3 Joint1_tengely = new Vector3(0, 1, 0);
            Quaternion Joint1_forgat = Quaternion.AngleAxis(-float.Parse
            (J1input_text.text), Joint1_tengely);
            Joint1.transform.rotation = Joint1_forgat;

            //2-es csukló:
            Vector3 Joint2_tengely = new Vector3(1, 0, 0);
            Quaternion Joint2_forgat = Quaternion.AngleAxis(float.Parse
            (J2input_text.text), Joint2_tengely);
            Joint2.transform.rotation = Joint1_forgat * Joint2_forgat;

            //3-as csukló:
            Vector3 Joint3_tengely = new Vector3(1, 0, 0);
```

...Data\Local\Temp\Rar\$DIa14468.4383\SetJointAngles.cs	2
---	---

```

Quaternion Joint3_forgat = Quaternion.AngleAxis(float.Parse
(J3input_text.text) - 90, Joint3_tengely);
Joint3.transform.rotation = Joint1_forgat * Joint2_forgat *
Joint3_forgat;

//4-es csukló:
Vector3 Joint4_tengely = new Vector3(0, 0, 1);
Quaternion Joint4_forgat = Quaternion.AngleAxis(-float.Parse
(J4input_text.text), Joint4_tengely);
Joint4.transform.rotation = Joint1_forgat * Joint2_forgat *
Joint3_forgat * Joint4_forgat;

//5-ös csukló:
Vector3 Joint5_tengely = new Vector3(1, 0, 0);
Quaternion Joint5_forgat = Quaternion.AngleAxis(float.Parse
(J5input_text.text), Joint5_tengely);
Joint5.transform.rotation = Joint1_forgat * Joint2_forgat *
Joint3_forgat * Joint4_forgat * Joint5_forgat;

//6-os csukló:
Vector3 Joint6_tengely = new Vector4(0, 0, 1);
Quaternion Joint6_forgat = Quaternion.AngleAxis(-float.Parse
(J6input_text.text), Joint6_tengely);
Joint6.transform.rotation = Joint1_forgat * Joint2_forgat *
Joint3_forgat * Joint4_forgat * Joint5_forgat * Joint6_forgat;
    }
}
}

```

```
using System.Collections;
using System.Collections.Generic;
using System.IO;
using Unity.VisualScripting;
using UnityEditor;
using UnityEngine;
using UnityEngine.UI;
using System;
using MathNet.Numerics.LinearAlgebra;

public class DenavitHartenberg : MonoBehaviour
{
    //Inputfieldek meghívása UI-ról, a szerszámkoordináták kiírásához és a
    //csuklókoordináták beolvasásához
    public InputField jointText_q1;
    public InputField jointText_q2;
    public InputField jointText_q3;
    public InputField jointText_q4;
    public InputField jointText_q5;
    public InputField jointText_q6;
    public InputField szamolt_szerszam_x;
    public InputField szamolt_szerszam_y;
    public InputField szamolt_szerszam_z;
    public InputField szamolt_szerszam_rx;
    public InputField szamolt_szerszam_ry;
    public InputField szamolt_szerszam_rz;
    public InputField szamolt_szerszam_qx;
    public InputField szamolt_szerszam_qy;
    public InputField szamolt_szerszam_qz;
    public InputField szamolt_szerszam_qw;

    //a kiszámolt szerszámkoordinátákat vizualizáló piros gömb objektum
    //meghívása
    public GameObject szamolt_szerszam;
    public GameObject geometriai_szerszam;

    //DH transzformációs mátrix előállítása a DH paramétereiből
    public Matrix<float> Matrix_eloallitas(float[] h)
    {
        //DH paraméterek kiszedése egyéni változókba
        float theta1 = h[0];
        float d1 = h[1];
        float a1 = h[2];
        float alfa1 = h[3];

        //Szögek szinuszának és koszinuszának kiszámolása
        float cos_theta1 = Mathf.Cos(theta1 * Mathf.PI / 180);
        float sin_theta1 = Mathf.Sin(theta1 * Mathf.PI / 180);
        float cos_alfa1 = Mathf.Cos(alfa1 * Mathf.PI / 180);
        float sin_alfa1 = Mathf.Sin(alfa1 * Mathf.PI / 180);

        //Denavit-Hartenberg transzformációs mátrix létrehozása az általános
        //képlettel a D-H paramétereiből
```

```

Matrix<float> A = Matrix<float>.Build.DenseOfArray(new float[,]
{
    {cos_theta1,-
      sin_theta1*cos_alfa1,sin_theta1*sin_alfa1,a1*cos_theta1},
    {sin_theta1,cos_theta1*cos_alfa1,-
      cos_theta1*sin_alfa1,a1*sin_theta1},
    {0,sin_alfa1,cos_alfa1,d1},
    {0,0,0,1}
});

return A;
}

//A robotmozog metódus abban az esetben fut le ha a felhasználó a
//Beállít gombra kattint
public void RobotMozog()
{
    //Szegmensek méreteinek megadása változóba
    float L0 = 350;
    float L1 = 95;
    float L2 = 245;
    float L3 = 135;
    float L4 = 270;
    float L5 = 85;

    //A csuklókoordináták beolvasása a UI-ról meghívott InputFieldekből
    //változóba
    float theta1 = float.Parse(jointText_q1.text);
    float theta2 = float.Parse(jointText_q2.text);
    float theta3 = float.Parse(jointText_q3.text);
    float theta4 = float.Parse(jointText_q4.text);
    float theta5 = float.Parse(jointText_q5.text);
    float theta6 = float.Parse(jointText_q6.text);

    //Itt kerülnek megadásra a robot DH paraméterei, ezeket akkor kell
    //változtatni ha másik robot kerül a szimulációba
    //Ezekből kerül kiszámolásra a D-H transzformációs mátrix
    float[] h0_1 = new float[4] { theta1, L0, L1, -90 };
    float[] h1_2 = new float[4] { -90 + theta2, 0, L2, 0 };
    float[] h2_3 = new float[4] { -90 + theta3, 0, L3, -90 };
    float[] h3_4 = new float[4] { theta4, L4, 0, 90 };
    float[] h4_5 = new float[4] { theta5, 0, 0, -90 };
    float[] h5_6 = new float[4] { 180+theta6, L5, 0, 0 };

    //Minden szegmensből a következőbe transzformáló mátrix létrehozása
    Matrix<float> H0_1 = Matrix_eloallitas(h0_1);
    Matrix<float> H1_2 = Matrix_eloallitas(h1_2);
    Matrix<float> H2_3 = Matrix_eloallitas(h2_3);
    Matrix<float> H3_4 = Matrix_eloallitas(h3_4);
    Matrix<float> H4_5 = Matrix_eloallitas(h4_5);
    Matrix<float> H5_6 = Matrix_eloallitas(h5_6);

    //A szegmensek közötti transzformációs mátrixok összeszorzása

```

```
    megfelelő sorrendben, hogy megkapjuk a H0_6 mátrixot
    Matrix<float> H0_2 = H0_1 * H1_2;
    Matrix<float> H0_3 = H0_2 * H2_3;
    Matrix<float> H0_4 = H0_3 * H3_4;
    Matrix<float> H0_5 = H0_4 * H4_5;
    Matrix<float> H0_6 = H0_5 * H5_6;

    //Roll-Pitch-Yaw szögek kiszámolása (Euler szögek)
    //Ez a D-H transzformációs mátrix megfelelő elemeiből történik
    float rot_z = Mathf.Rad2Deg * Mathf.Atan2(H0_6[1, 0], H0_6[0, 0]);
    float rot_y = Mathf.Rad2Deg * Mathf.Asin(-H0_6[2, 0]);
    float rot_x = Mathf.Rad2Deg * Mathf.Atan2(H0_6[2, 1], H0_6[2, 2]);

    //A kiszámolt szerszám koordináták átadása a GameObjektnak, hogy
    abba a pozícióba legyen mozgatva
    Vector3 szerszam_krsz = new Vector3((float)Math.Round(H0_6[0,3], 3),
    (float)Math.Round(-H0_6[1,3],3), (float)Math.Round(H0_6[2,3],3));
    szamolt_szerszam.transform.localPosition = szerszam_krsz;

    //A kiszámolt szerszám koordináták illetve orientáció 3 tizedes
    jegyre történő kerekítése
    //Majd átadása a UI-ra, hogy ott megjelenítésre kerüljenek
    szamolt_szerszam_x.text = Math.Round(H0_6[0, 3],3).ToString();
    szamolt_szerszam_y.text = Math.Round(-H0_6[1, 3],3).ToString();
    szamolt_szerszam_z.text = Math.Round(H0_6[2, 3],3).ToString();
    szamolt_szerszam_rz.text = Math.Round(rot_z, 3).ToString();
    szamolt_szerszam_ry.text = Math.Round(rot_y, 3).ToString();
    szamolt_szerszam_rx.text = Math.Round(rot_x, 3).ToString();
}
}
```

```
using System.Collections;
using System.Collections.Generic;
using System;
using UnityEngine;
using UnityEngine.UI;
using Unity.VisualScripting;
using MathNet.Numerics.LinearAlgebra;

public class Inverzgeometry1 : MonoBehaviour
{
    //Az InputFieldek meghívása a UI-ról
    //Így bolvasható a kívánt szerszámkoordináta és orientáció illetve
    //kiírásra kerülhet a szükséges csuklókoordináta
    public InputField input_x;
    public InputField input_y;
    public InputField input_z;
    public InputField input_rx;
    public InputField input_ry;
    public InputField input_rz;

    public InputField input_q1;
    public InputField input_q2;
    public InputField input_q3;
    public InputField input_q4;
    public InputField input_q5;
    public InputField input_q6;

    public InputField output_x;
    public InputField output_y;
    public InputField output_z;
    public InputField output_rx;
    public InputField output_ry;
    public InputField output_rz;

    //Meghívásra kerül a legördülő menü a későbbi konfiguráció választáshoz
    public Dropdown ConfSelect;

    //Létrehozásra kerül egy "JointAngles" nevű struktúra
    //Ez 6db float értéket képes tárolni amik az értékük beállításakor csak
    //akkor őrzik meg a kívánt értéket
    //ha az az adott csukló tartományán belüli szögérték, amennyiben nem az,
    //úgy NAN értéket vesznek fel
    struct JointAngles
    {
        //Itt kerül létrehozásra a 6 float változó
        float q1;
        float q2;
        float q3;
        float q4;
        float q5;
        float q6;

        //Mindegyik q-s változóhoz egy azonos számú Q változó létrehozása
    }
}
```

```
//Ezek a Q változók felelnek azért, hogy get utasítás esetén ↗  
    visszaadják a q értékét  
//set esetén pedig ha a beállítani kívánt szögérték a csukló fizikai ↗  
    határain belül van,  
//akkor azt a szögértéket adja a q-nak, ha pedig a határokon kívül ↗  
    van akkor NAN értéket tárol el  
public float Q1  
{  
    get { return q1; }  
    set  
    {  
        if (value > -170 * Mathf.Deg2Rad && value <= 170 * ↗  
            Mathf.Deg2Rad)  
        {  
            q1 = value;  
        }  
        else  
        {  
            q1 = float.NaN;  
        }  
    }  
}  
  
public float Q2  
{  
    get { return q2; }  
    set  
    {  
        if (value > -90 * Mathf.Deg2Rad && value <= 135 * ↗  
            Mathf.Deg2Rad)  
        {  
            q2 = value;  
        }  
        else  
        {  
            q2 = float.NaN;  
        }  
    }  
}  
  
public float Q3  
{  
    get { return q3; }  
    set  
    {  
        if (value > -20 * Mathf.Deg2Rad && value <= 171 * ↗  
            Mathf.Deg2Rad)  
        {  
            q3 = value;  
        }  
        else  
        {  
            q3 = float.NaN;  
        }  
    }  
}
```

```
    }  
    }  
}  
  
public float Q4  
{  
    get { return q4; }  
    set  
    {  
        if (value > -160 * Mathf.Deg2Rad && value <= 160 *  
            Mathf.Deg2Rad) ➤  
        {  
            q4 = value;  
        }  
        else  
        {  
            q4 = float.NaN;  
        }  
    }  
}  
  
public float Q5  
{  
    get { return q5; }  
    set  
    {  
        if (value > -120 * Mathf.Deg2Rad && value <= 120 *  
            Mathf.Deg2Rad) ➤  
        {  
            q5 = value;  
        }  
        else  
        {  
            q5 = float.NaN;  
        }  
    }  
}  
  
public float Q6  
{  
    get { return q6; }  
    set  
    {  
        if (value >= -360 * Mathf.Deg2Rad && value <= 360 *  
            Mathf.Deg2Rad) ➤  
        {  
            q6 = value;  
        }  
        else  
        {  
            q6 = float.NaN;  
        }  
    }  
}
```

```

    }

    //Egy string változó létrehozása ami az adott struktúra elem által
    //tárolt csuklókoordinátákat
    //egy jól átlátható stringgké alakítja, ez a konzolon való
    //megjelenítéshez kell
    public string Display()
    {
        return $"q1:{q1 * Mathf.Rad2Deg}, q2:{q2 * Mathf.Rad2Deg}, q3:
        {q3 * Mathf.Rad2Deg}, q4:{q4 * Mathf.Rad2Deg}, q5:{q5 *
        Mathf.Rad2Deg}, q6:{q6 * Mathf.Rad2Deg}";
    }
}

//DH transzformációs mátrix előállítás a DH paramétereiből
public Matrix<float> Matrix_eloallitas(float[] h)
{
    //DH paraméterek kiszedése egyéni változókba
    float theta1 = h[0];
    float d1 = h[1];
    float a1 = h[2];
    float alfa1 = h[3];

    //Szögek szinuszának és koszinuszának kiszámolása
    float cos_theta1 = Mathf.Cos(theta1 * Mathf.PI / 180);
    float sin_theta1 = Mathf.Sin(theta1 * Mathf.PI / 180);
    float cos_alfa1 = Mathf.Cos(alfa1 * Mathf.PI / 180);
    float sin_alfa1 = Mathf.Sin(alfa1 * Mathf.PI / 180);

    //Denavit-Hartenberg transzformációs mátrix létrehozása az általános
    //képlettel a D-H paramétereiből
    Matrix<float> A = Matrix<float>.Build.DenseOfArray(new float[, ]
    {
        {cos_theta1, -
        sin_theta1*cos_alfa1, sin_theta1*sin_alfa1, a1*cos_theta1},
        {sin_theta1, cos_theta1*cos_alfa1, -
        cos_theta1*sin_alfa1, a1*sin_theta1},
        {0, sin_alfa1, cos_alfa1, d1},
        {0, 0, 0, 1}
    });
    return A;
}

//Cos függvény metódus
//Ez a metódus a kapott szögérték koszinuszát adja vissza
//Erre azért van szükség, hogy a cos műveletet lehessen c-nek rövidíteni
//és így rövidebbek és átláthatóbbak legyenek a kódsorok
public float c(float szog)
{
    return Mathf.Cos(szog * Mathf.Deg2Rad);
}

```

```

//Sin függvény metódus
//Ez a metódus a kapott szögérték szinuszát adja vissza
//Erre azért van szükség, hogy a sin műveletet lehessen s-nek rövidíteni
//és így rövidebbek és átláthatóbbak legyenek a kódsorok
public float s(float szog)
{
    return Mathf.Sin(szog * Mathf.Deg2Rad);
}

//Configurations nevű lista típusú változó létrehozása a különböző
//konfigurációk tárolására
List<JointAngles> Configurations = new List<JointAngles>();

//A beallit metódus a szimulációban a beállít gombra kattintáskor fut le
//Ebben a metódusban kerülnek elvégzésre az inverz geometriai számítások
public void beallit()
{
    //A kívánt szerszámkoordináta és orientáció beolvasása a UI-ról
    //változókba
    float x = float.Parse(input_x.text);
    float y = float.Parse(input_y.text);
    float z = float.Parse(input_z.text);
    float rx = float.Parse(input_rx.text);
    float ry = float.Parse(input_ry.text);
    float rz = float.Parse(input_rz.text);

    //Konfigurációkat tároló lista kiürítése, hogy beleírásra
    //kerülhessenek az új konfigurációk
    Configurations.Clear();
    //Az UI-n a legödrülő menü elemeinek eltávolítása
    ConfSelect.ClearOptions();

    //Rotációs mátrix előállítása a kívánt Roll Pitch Yaw szögekből
    Matrix<float> R_rpy = Matrix<float>.Build.DenseOfArray(new float[, ]
    {
        {c(rz)*c(ry), (-s(rz)*c(rx))+c(rz)*s(ry)*s(rx), s(rz)*s(ry)*c(rx)},
        {s(rz)*c(ry), c(rz)*c(rx)+s(rz)*s(ry)*s(rx), (-c(rz)*s(ry)*c(rx))+s(rz)*s(ry)*c(rx)},
        {-s(ry), c(ry)*s(rx), c(ry)*c(rx)}
    });

    //Hatodik szegmens hosszának eltárolása egy változóban (a robot
    //változása esetén módosítani kell)
    float d6 = 85f;

    //Kéztőpont pozíció számítás (Kinematic Decoupling)
    float[] P_c = new float[3] { x - d6 * R_rpy[0, 2], y - d6 * R_rpy[1, 2], z - d6 * R_rpy[2, 2] };

    //Egy jointAngles nevű JointAngles struktúra létrehozása
    JointAngles jointAngles = new JointAngles();

```

```

//Egy for ciklus futtatása ami az inverz kinematikai számításokban az előjel váltásért felel
for (int i = 1; i >= -1; i -= 2)
{
    //Első csuklókoordináta számítása egy egyszerű szögfüggvénnyel
    jointAngles.Q1 = Mathf.Atan2(i * P_c[1], i * P_c[0]);
    if (jointAngles.Q1 == float.NaN) continue;

    //Egy for ciklus futtatása ami az inverz kinematikai számításokban az előjel váltásért felel
    for (int w = 1; w >= -1; w -= 2)
    {
        //Harmadik csuklókoordináta számítása
        float R = Mathf.Sqrt(Mathf.Pow(270, 2) + Mathf.Pow(135, 2));
        float k = Mathf.Sqrt(Mathf.Pow(P_c[0], 2) + Mathf.Pow(P_c[1], 2));
        float alfa = Mathf.Atan2(135, 270);
        if (k >= 95)
        {
            float D = (Mathf.Pow(k - 95, 2) + Mathf.Pow(P_c[2] - 350, 2) - Mathf.Pow(245, 2) - Mathf.Pow(R, 2)) /
                (2 * 245 * R);
            jointAngles.Q3 = Mathf.Atan2(w * Mathf.Sqrt(1 - Mathf.Pow(D, 2)), D) + alfa;
        }
        else
        {
            float D = (Mathf.Pow(-k + 95, 2) + Mathf.Pow(P_c[2] - 350, 2) - Mathf.Pow(245, 2) - Mathf.Pow(R, 2)) /
                (2 * 245 * R);
            jointAngles.Q3 = -(Mathf.Atan2(w * Mathf.Sqrt(1 - Mathf.Pow(D, 2)), D) - alfa);
        }
        if (jointAngles.Q3 == float.NaN) continue;

        //Második csuklókoordináta számítása
        jointAngles.Q2 = Mathf.PI / 2 - Mathf.Atan2(R * Mathf.Sin(jointAngles.Q3 - alfa),
            R * Mathf.Cos(jointAngles.Q3 - alfa) + 245) - Mathf.Atan2(P_c[2] - 350, k - 95);
        if (jointAngles.Q2 == float.NaN) continue;

        //Szegmensek méreteinek megadása
        float L0 = 350;
        float L1 = 95;
        float L2 = 245;
        float L3 = 135;

        //Itt kerülnek megadásra a robot DH paraméterei, ezeket

```

```

    akkor kell változtatni ha másik robot kerül a szimulációba
    //Ezekből kerül kiszámolásra a D-H transzformációs mátrix
    float[] h0_1 = new float[4] { jointAngles.Q1 *
        Mathf.Rad2Deg, L0, L1, -90 };
    float[] h1_2 = new float[4] { -90 + jointAngles.Q2 *
        Mathf.Rad2Deg, 0, L2, 0 };
    float[] h2_3 = new float[4] { -90 + jointAngles.Q3 *
        Mathf.Rad2Deg, 0, L3, -90 };

    //Minden szegmensből a következőbe transzformáló mátrix
    létrehozása
    Matrix<float> H0_1 = Matrix_eloallitas(h0_1);
    Matrix<float> H1_2 = Matrix_eloallitas(h1_2);
    Matrix<float> H2_3 = Matrix_eloallitas(h2_3);

    //Transzformációs mátrixok összeszorozása megfellelő
    sorrendben, hogy megkapjuk a H0_3 mátrixot
    Matrix<float> H0_2 = H0_1 * H1_2;
    Matrix<float> H0_3 = H0_2 * H2_3;

    //Transzponálás elvégzése majd R_rpy mátrix szorzása az
    inverz H0_3 mátrixal
    Matrix<float> R0_3 = H0_3.SubMatrix(0, 3, 0, 3);
    Matrix<float> R0_3_T = R0_3.Transpose();
    Matrix<float> R3_6 = R0_3_T * R_rpy;

    //4, 5 é 6 csukló csuklózóváltozóinak kiszámolása
    for (int o = 1; o >= -1; o -= 2)
    {
        jointAngles.Q5 = Mathf.Atan2(o * Mathf.Sqrt(1 -
            Mathf.Pow(R3_6[2, 2], 2)), R3_6[2, 2]);
        if (jointAngles.Q5 == float.NaN) continue;
        jointAngles.Q4 = (Mathf.Atan2(-o * R3_6[1, 2], o * -R3_6
            [0, 2])); // + Mathf.PI);
        if (jointAngles.Q4 == float.NaN) continue;
        jointAngles.Q6 = Mathf.Atan2(o * R3_6[2, 1], o * (-R3_6
            [2, 0]));
        if (jointAngles.Q6 == float.NaN) continue;

        //A kiszámolt konfiguráció hozzáadása a konfigurációs
        listához
        Configurations.Add(jointAngles);
    }
}

//Elérhető konfigurációk hozzáadása a konfigurációs listából a
konfiguráció választó legördülő listához
List<String> options = new List<String>();
for (int hh = 1; hh <= Configurations.Count; hh++)
{
    options.Add($"Option {hh}");
}

```

```
        Debug.Log($"Option {hh}: {Configurations[hh - 1].Display()}");
    }
    ConfSelect.AddOptions(options);

    //Kezdetben mindig a 0-s konfiguráció kiválasztása és a csuklókoordináta kiírása a UI-ra
    input_q1.text = (Math.Round(Configurations[0].Q1 * Mathf.Rad2Deg, 2)).ToString();
    input_q2.text = (Math.Round(Configurations[0].Q2 * Mathf.Rad2Deg, 2)).ToString();
    input_q3.text = (Math.Round(Configurations[0].Q3 * Mathf.Rad2Deg, 2)).ToString();
    input_q4.text = (Math.Round(Configurations[0].Q4 * Mathf.Rad2Deg, 2)).ToString();
    input_q5.text = (Math.Round(Configurations[0].Q5 * Mathf.Rad2Deg, 2)).ToString();
    input_q6.text = (Math.Round(Configurations[0].Q6 * Mathf.Rad2Deg, 2)).ToString();
}

//Konfiguráció választás a legördülő lista alapján majd a konfigurációhoz tartozó csuklókoordináta kiírása a UI-ra
//Ez a metódus akkor kerül futtatásra amikor változik a legördülő menüben a kiválasztott elem
public void DropDownChanged()
{
    input_q1.text = (Math.Round(Configurations[ConfSelect.value].Q1 * Mathf.Rad2Deg, 2)).ToString();
    input_q2.text = (Math.Round(Configurations[ConfSelect.value].Q2 * Mathf.Rad2Deg, 2)).ToString();
    input_q3.text = (Math.Round(Configurations[ConfSelect.value].Q3 * Mathf.Rad2Deg, 2)).ToString();
    input_q4.text = (Math.Round(Configurations[ConfSelect.value].Q4 * Mathf.Rad2Deg, 2)).ToString();
    input_q5.text = (Math.Round(Configurations[ConfSelect.value].Q5 * Mathf.Rad2Deg, 2)).ToString();
    input_q6.text = (Math.Round(Configurations[ConfSelect.value].Q6 * Mathf.Rad2Deg, 2)).ToString();
}

//A masolas metódus a szimulációba a másolás gombra kattintáskor kerül meghívásra
//Ez átmásolja az aktuális szerszámkoordinátát és orientációt az inverz geometria kívánt szerszámkoordináta és orientáció részébe
public void masolas()
{
    input_x.text = output_x.text;
    input_y.text = output_y.text;
    input_z.text = output_z.text;
    input_rx.text = output_rx.text;
    input_ry.text = output_ry.text;
    input_rz.text = output_rz.text;
```

}
}