Az inverz kinematika

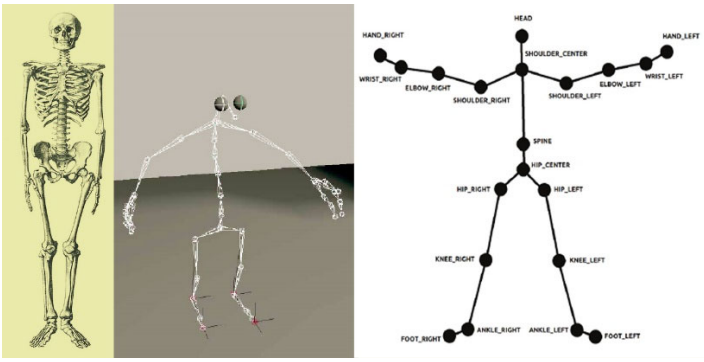
I. rész

Az analitikus megoldás

Az inverz kinematika egy animációs technika, amely összetett testek mozgatását tűzi ki célul. Az összetett testeket *csont/ízület-rendszerek*kel valósítjuk meg. A csont/ízület-rendszerek az emberi test anatómiáját utánozzák. A csontváz a test alakját adja meg és védi azt a külső behatásokkal szemben. A gerincesek szervezetében a csontok erős rostokból állnak, amelyek közé lerakódik a kalcium, így a vasbeton keménységével vetekedő szerv jön létre. A test csontjait az ízületek kapcsolják össze. A rugalmas ízületek a mozgás, mozgatás képességével ruházzák fel a csontvázat.

Ha jellemezni szeretnénk az emberi csontvázat, azt mondhatnánk, hogy mintegy 70 *szabadságfokú*. A szabadságfok (DOF – Degree Of Freedom) egy anyagi rendszer állapotának egyértelmű meghatározásához szükséges, egymástól független mennyiségek száma. Például a síkmozgásnak két szabadságfoka van: két egymásra merőleges irányú szabad elmozdulás.

Az emberi csontváz analógiájára építjük fel a mozgatni kívánt bábunk, alakunk, karakterünk, robotunk csont/ízület-rendszerét is azzal az előnnyel, hogy ez már akármilyen fantázia szülte csontváz is lehet. A csontok merevek, alapszabályuk, hogy hosszuk nem változhat, nem nyúlhatnak meg, nem húzódhatnak össze, az ízületek pedig összekötik a csontokat mozgásteret, pontosabban forgatási teret biztosítva számukra. Ha egy csont nem írhat le egy teljes kört az ízület körül (360°-ot), akkor úgynevezett *kényszer*eket, megkötéseket vezetünk be, pontosan leírva ezáltal a mozgástartományt, szögtartományt.

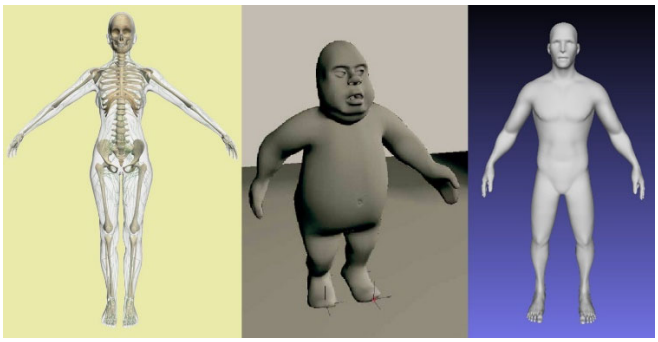


a) *b)* *c)*

1. ábra

Csontvázak: a) ember, b) számítógépes karakter, c) a Kinekt belső csontváza

Ha megterveztük a csontvázunkat, fel kell ezt öltöztetnünk, ellátnunk bőrrel, és máris kész van a mozgatni, animálni kívánt karakterünk. A csontvázak felöltöztetése, amit *skinning*-nek is nevezünk, a számítógépes animáció másik nagy problémája.



2. ábra
Csontvázak felöltöztetése

A csont/ízület rendszerek, de például a robotkarok mozgatását is többnyire *előremutató* vagy *inverz kinematika* segítségével oldjuk meg.

A módszerek ismertetése előtt ismerjünk meg egy pár fontos fogalmat.

*Mechanizmus*nak nevezzük az egymással mozgásbeli kényszerkapcsolatban álló merev testekből (csontokból) felépített mozgó szerkezetet.

A kapcsolódó merev testeket (csontokat) a mechanizmus *tag*jainak nevezzük.

A mechanizmusok általában hierarchikus rendszerek, vagyis a tagok között fölé- és alárendeltségi kapcsolatok vannak, sőt a mellérendeltség is létezhet. A hierarchikus szintek szülő–gyermek (apa–fiú) kapcsolatokban nyilvánulnak meg.

Az ízületeket *csukló*knak is nevezzük.

*Kinematikus lánc*nak nevezzük az egymás után szerelt tagokat (csontokat), vagyis az egyik csont végéből indul ki a másik csont, a két csontot ízület köti össze. Egy mechanizmus több kinematikus láncból is állhat. A kinematikus láncok lehetnek zártak és nyíltak. A zárt lánc tagjai zárt sokszöget alkotnak.

A mechanizmus tagjai közül az egyik általában rögzített, ezt *állvány*nak nevezzük.

Ha több kinematikus lánc egy ízületben összekapcsolódik, akkor azt az ízületet *alap*nak vagy *origó*nak nevezzük.

Végszerv, *végberendezés* vagy *effektor*, *end-effektor* a kinematikus lánc szabad vége, az utolsó mozgatható tag (csont), amely többnyire munkavégzésre szolgál.

A mechanizmus helyzetét meghatározó, egymástól független elmozdulások és elfordulások száma a mechanizmus *szabadságfok*a.

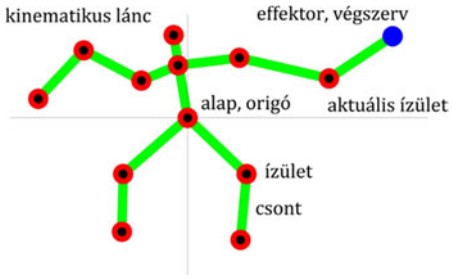
A függetlenül állítható paraméterek összességét *állapot*nak nevezzük.

Számítógépes animáció esetén a csontok az ízületek körül elfordulnak, de általános esetben, például egyes robotoknál is vannak olyan ízületek (csuklók), amelyeknél el is mozdulhatnak a csontok (tagok). Így általános esetben azt mondjuk, hogy egy kinematikus láncban egy tag a csukló tengelye körül a megelőző taghoz képest elfordulhat (rotációs csukló), vagy a csukló tengelye mentén a megelőző taghoz képest elmozdulhat (transzlációs csukló).

*Aktuális ízület*nek nevezzük azt az ízületet, amely körül éppen mozog egy csont.

A mozgatásnak, mozgásnak (animálásnak) mindig valami célja van, azaz a rendszer valamelyik pontját szeretnénk egy előírt pontig vezérelni.

A fent bevezetett fogalmak segítségével könnyen meg tudjuk fogalmazni a mozgatás, animálás technikáit.



3. ábra
A kinematikus lánc elemei

Az *előremutató*, *előremenő* vagy „*forward" kinematiká*nak az a lényege, hogy a kinematikus lánc tagjait az origótól mozgatjuk egyesével a végszerv felé, tehát közvetlenül az állapotváltozók terében dolgozunk, egyenként beállítunk minden csontot, ízületet. Ez által nagyon precíz mozgás valósítható meg, mindamellett a folyamat nehézkes és sok számítást vagy tapasztalatot igényel.

A mutató ujjunkat (végszerv) úgy tudjuk kinyújtani, hogy először elmozdítjuk a vállunkat (ízület), körülötte elforgatjuk a felkarcsontunkat, elmozdul a könyök (ízület), körülötte elforgatjuk az alkarcsontunkat, elmozdul a csukló (ízület), körülötte elfordulnak a kézcsontok, közöttük a kettes kézközépcsont is, amelyhez egy ízülettel kötődik a mutatóujj felső ujjperc csontja, ez is elfordul, ehhez egy ízülettel kötődik a középső ujjperccsont, ez is elfordul, ehhez szintén egy ízülettel kötődik a mutatóujj alsó ujjperc csontja, amely szintén elfordul, megvalósítva így a mutatóujj teljes elmozdulását.

Ez az előremenő kinematika nem használható akkor, ha a mechanizmus strukturális összefüggése erősen nemlineáris. Ekkor hiába interpolálunk egyenletesen az állapottérben, a végszerv vadul kalimpálni fog.

Gondoljunk arra, ha például nem azonos magasságú lépcsőfokokon kell felmenjünk egy lépcsőn, hol magasabbra, hol alacsonyabbra kell nagyon precízen lépnünk. Az emberi agy (amely előremenő kinematikával vezérli a lábfejeinket) összezavarja a végtagok mozgatását, nagyon sokszor megbotlunk, helytelenül lépünk.

Az ehhez hasonló nehéz eseteknél jelent megoldást az *inverz kinematika*, amely nem az állapotot, hanem a kritikus végszerv helyzetét interpolálja, majd az állapotot a végszerv interpolált helyzetéből számítja vissza.

Például az előbb említett lépcsős esetben elegendő a lábfejeket szépen felhelyezni a lépcsőfokokra, a „program" (sajnos az emberi agy nem inverz kinematikával működik) pedig kiszámítja pontosan, hogy milyen helyzetbe kell kerüljön a boka (ízület), mennyivel kell elforduljon ehhez a sípcsont, milyen helyzetbe kerül a térd (ízület), mennyivel kell ehhez elforduljon a combcsont stb., tehát lentről felfelé, inverz módon számítunk ki mindent.

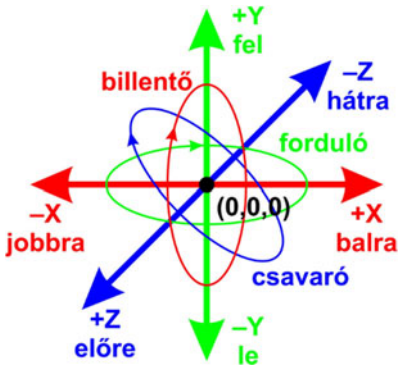
Ha matematikailag szeretnénk megfogalmazni a problémát, akkor legyen S az állapot, E pedig a végszerv helyzete, amely a pillanatnyi pozícióval (x, y, z) koordináták, valamint az orientációval (ψ, θ, ϕ) adható meg.

Könnyen igazolható, hogy egy geometriai objektum tetszőleges térbeli helyzetbe hozásához három, egymás után következő forgatás szükséges, amelyeket az úgynevezett *Euler-szögek* (ψ, θ, ϕ) írnak le.

Jelen tanulmányban a ψ (pszí) Z-tengely körüli, a θ (théta) X-tengely körüli, a φ (fí) pedig Y-tengely körüli forgatást jelent. Sajnos a szögek megadási sorrendjében és a tengelyek jelölésében, amelyek között a szögeket mérik, soha nem alakult ki egységes gyakorlat.

Az Euler-szögek megfelelnek a térben előforduló *csavaró (roll)* – *forduló (yaw)* – *billentő (pitch)* fordulásoknak, amelyek leginkább egy repülőgép mozgásával szemléltethetők.

Az X-tengely körüli forgást *billentő*nek, az Y-tengely körüli forgást *forduló*nak, a Z-tengely körülit pedig *csavaró*nak nevezzük. A térben ezen kívül hat kitüntetett irány van: *fel* (up), *le* (down), *jobbra* (right), *balra* (left), *előre* (forward), *hátra* (back).



4. ábra
Kitüntetett irányok és forgatások a térben

Nos, visszatérve a matematikai feladathoz, a kinematika azt jelenti, léteznie kell legalább egy, az E végszerv helyzetét az S állapotból kifejező strukturális függvénynek, amely csak a rendszer felépítésétől és geometriájától függ, vagyis:

$$E = F(S).$$

Az inverz kinematika esetében a végszerv pályáját tervezzük meg, vagyis az S állapot a strukturális függvény inverzével állítható elő az E végszerv helyzetéből:

$$S = F^{-1}(E).$$

Az invertálás azonban több problémát is felvet. Az f nemlineáris, az inverz függvény kiszámítása nem triviális, másrészt nem egy-egy értelmű: több állapothoz is tartozhat ugyanaz a végszerv-helyzet.

Gondoljunk bele, hogy hányféleképpen érinthetünk meg az ujjbegyünkkel egy falon lévő pontot…

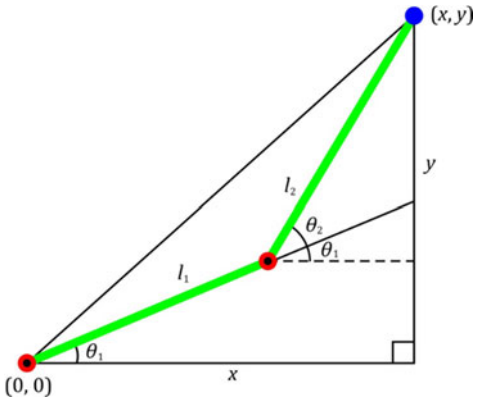
A feladat pontos matematikai megoldása (ezt nevezzük *analitikus megoldás*nak) sajnos csak két csont esetében létezik, kettőnél több csontra nagyon elbonyolódik a rendszer.

Fogalmazzuk meg a feladatot két csontra és szintén az egyszerűség kedvéért csak 2D-ben, tehát a pontokat két koordináta (x, y) segítségével írjuk le.

Feladat:

Adott két csont. Az első egyik vége az origóban $(0, 0)$ *található, a másik végéből pedig a második csont indul. Mekkora szögekkel kell elforgatnunk a két csontot ahhoz, hogy a második csont szabad vége (végszerv) egy adott* (x, y) *pozícióba kerüljön?*

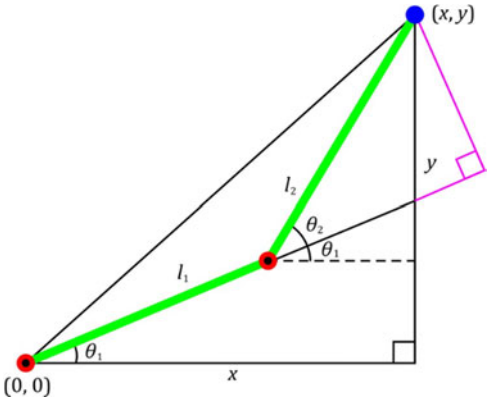
A feladathoz az 5. ábrát készíthetjük el.



5. ábra
Az inverz kinematika feladata

Meg kell határozzuk tehát a θ_1 és θ_2 szögeket. Ismert az l_1, l_2, valamint az (x, y).

A szögek meghatározása a 6. ábrán látható háromszögekből történik.



6. ábra
A szögek meghatározása

Technikai lépések, számítgatások sorozataként, a matematikai bizonyítás felírása nélkül fogadjuk most el, hogy a végeredmény a következő:

$$\cos\theta_2 = \frac{x^2+y^2-l_1^2-l_2^2}{2l_1l_2},$$

valamint:

$$\tan\theta_1 = \frac{y\cdot(l_1+l_2\cdot\cos\theta_2)-x\cdot l_2\cdot\sin\theta_2}{x\cdot(l_1+l_2\cdot\cos\theta_2)+y\cdot l_2\cdot\sin\theta_2}.$$

Innen is látszik, hogy három vagy több csont esetére nagyon bonyolulttá válnak a képletek.

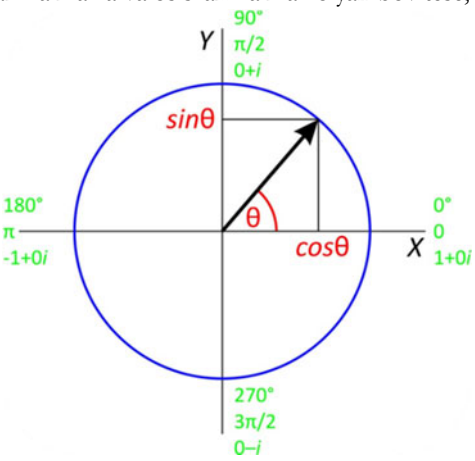
Ha a feladatot informatikus szemmel nézzük, a következőket tudjuk megállapítani (maradjunk szintén a 2D egyszerűbb helyzetben):

A forgatásokat, a rendszer állapotát nagyon jó, le tudjuk írni komplex számokkal. Matematikából tudjuk, hogy a komplex számhalmaz a valós számhalmaz olyan bővítése, melyben elvégezhető a negatív számból való négyzetgyökvonás.

Imaginárius (képzetes) egységnek az egyik olyan komplex számot nevezzük, amelynek a négyzete -1. Ennek jele i.

A $z = x + iy$ komplex számot trigonometrikus alakban így írhatunk fel: $z = r\cdot(\cos\theta + i\cdot\sin\theta)$, ahol $x = r\cdot\cos\theta$, $y = r\cdot\sin\theta$, valamint $r = \sqrt{x^2+y^2}$.

A θ szög meghatározására informatikában nem az arctg függvényt használjuk, hanem a szingularitások kiküszöbölése miatt az arctg2 függvényt használjuk.



7. ábra. *Egy komplex szám*

Az arctg2 függvény az arkusztangens (arctg) egyfajta általánosítása: alkalmas arra, hogy egy síkvektor y és x koordinátáiból – ügyelve a szokásoshoz képest fordított sorrendre – kiszámítsuk a vektor irányszögét (azaz az X-tengellyel bezárt szögét), nulla és 2π (vagy $-\pi$ és π) között.

Az arctg2 függvény minden valós (y, x) értékpárra értelmezve van, kivéve a $(0,0)$-t, mivel a nullvektor irányszöge definiálatlan. A gépi megvalósítások általában nullát adnak vissza ebben az esetben.

$$\mathbf{arctg2}(y,x) = \begin{cases} \mathbf{arctg}(y/x), & \text{ha } x \geq |y| \\ \pi/2 - \mathbf{arctg}(x/y), & \text{ha } y \geq |x| \\ \pi + \mathbf{arctg}(y/x), & \text{ha } x \leq -y \leq 0 \\ -\pi + \mathbf{arctg}(y/x), & \text{ha } x \leq y < 0 \\ -\pi/2 - \mathbf{arctg}(x/y), & \text{ha } y \leq -|x| \end{cases}$$

8. ábra

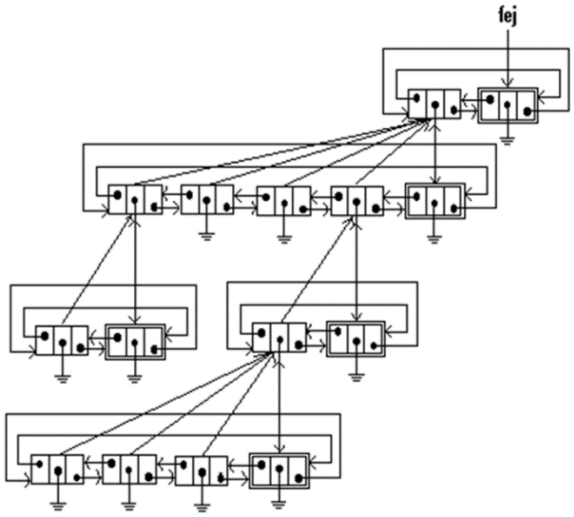
Az arctg2 függvény értelmezése

A komplex számokat C++-ban a következő osztállyal tudjuk például megvalósítani:

```
class Complex
{
private:
  double x;
  double y;
public:
  Complex(double x, double y);
  void SetComplex(double r, double theta);
  double Re();
  double Im();
};
```

A csont/ízület rendszereket leginkább egy objektumfával (rekurzív adatszerkezet) tudjuk megvalósítani, amely segítségével felépítjük az apa–fiú kapcsolatokat, és amilyen műveletet elvégzünk az apán, ugyanazt a műveletet elvégezzük az összes fián is.

A fent bemutatott `Complex` osztályon kívül a következő osztályokra lesz még szükségünk:



9. ábra. *Objektumfa*

```
class Point
{
public:
  double x;
  double y;
  Point() {}
  Point(double x, double y)
  {
```

```
      this->x = x; this->y = y;
  }
};

class Joint
{
private:
  Point point;
public:
  Joint() {}
  Joint(double x, double y) : point(x, y) {}
  void SetJoint(double x, double y) { point.x = x;
point.y = y; }
  double X() { return point.x; }
  double Y() { return point.y; }
  void Draw();
};

class Bone
{
private:
  Joint joint;
  Complex orientation;
  double length;
  bool endeffector;
  int id;
public:
  Bone(double x, double y, double l, double theta,
    int id, bool endeff);
  Bone() {}
  void Draw();
  void GoTo(double x, double y);
  void SetBone(double x, double y, double l, double
theta,
    int id, bool endeff);
  void SetAngle(double theta) {
orientation.SetComplex(length,
    theta); }
  double Re() { return orientation.Re(); }
  double Im() { return orientation.Im(); }
  double X() { return joint.X(); }
  double Y() { return joint.Y(); }
  double Length() { return length; }
  int Id() { return id; }
  bool IsEndEffector() { return endeffector; }
};

class BoneSystem
{
```

```
private:
  Bone parent;
  Bone child;
public:
  BoneSystem(double x, double y, double l1, double
l2,
    double theta, double phi);
  void Draw();
  void GoTo(double x, double y);
};
```

Itt számunkra a `GoTo` metódus érdekes, hisz ez oldja meg az inverz kinematika analitikus feladatát:

```
void BoneSystem::GoTo(double x, double y)
{
  GLint viewport[4];
  GLdouble modelview[16];
  GLdouble projection[16];
  GLfloat winX, winY, winZ;
  GLdouble posX, posY, posZ;
  glGetDoublev(GL_MODELVIEW_MATRIX, modelview);
  glGetDoublev(GL_PROJECTION_MATRIX, projection);
  glGetIntegerv(GL_VIEWPORT, viewport);
  winX = (float)x;
  winY = (float)viewport[3] - (float)y;
  glReadPixels(x, int(winY), 1, 1,
GL_DEPTH_COMPONENT,
    GL_FLOAT, &winZ);
  gluUnProject(winX, winY, winZ, modelview, project-
ion,
    viewport, &posX, &posY, &posZ);
  double theta1, theta2;
  posX -= parent.X();
  posY -= parent.Y();
  theta2 = acos((posX * posX + posY * posY -
parent.Length() *
         parent.Length() - child.Length() *
   child.Length())/
        (2 * parent.Length() * child.Length()));
  theta1 = atan2(posY * (parent.Length() +
 child.Length() *
          cos(theta2)) - posX * (child.Length() *
    sin(theta2)),
        (posX * (parent.Length() + child.Length() *
        cos(theta2)) + posY *
 (child.Length()*sin(theta2))));
  if ((!isnan(theta1)) && (!isnan(theta2)))
```