

TDK-dolgozat

Lehóczy Zoltán, Urbán Csaba

Intézmény: Óbudai Egyetem

Kar: Neumann János Informatikai Kar

Tanszék: Informatikai Rendszerek Intézet

Szerzők:

- Lehóczky Zoltán (3. évfolyam)
- Urbán Csaba (3. évfolyam)

Témavezető: Dr. Molnár András - egyetemi docens

A dolgozat címe: Tőzsdei brókerek kézjeleinek felismerése Kinect szenzor segítségével

Tartalom

Rendszerterv.....	4
Irodalomkutatás	6
A tőzsdei kézjelek	6
Vétel, eladás	6
Ár	6
Mennyiség	8
Egyéb jelzések.....	9
Kapcsolódó szoftverprojektek	9
Kinect for Windows SDK	9
Candescent NUI	9
Kinect Toolbox	10
Kinect SDK Dynamic Time Warping Gesture Recognition	10
Encog Machine Learning Framework	10
Aforge.NET.....	10
Egyéb	11
Összegzés és kiindulási tervek	11
Megvalósítás.....	12
A szoftver szerkezete.....	12
Kéz- és ujjdetektálás.....	12
Kinect Skeleton modell	12
Kéz- és ujjfelismerés a Candescent NUI keretrendszer segítségével	13
Kézadatok feldolgozása	14
Kézjel-felismerés	16
Tanítási mód	18
Felismerési mód	19
A felismerés vizsgált módszerei.....	19
Felhasználói felület.....	20
Hardver- és szoftverigény.....	25
Eredmények.....	25
Továbbfejlesztési lehetőségek	27
Összegzés.....	28
Irodalomjegyzék	29

Rendszerterv

Az általunk megvalósítandó szoftver célja, hogy a felhasználó által mutogatott tőzsdei parketten használt kézjeleket egy Kinect szenzor segítségével detektálja, majd feldolgozza úgy, hogy a felhasználó valós időben látja az adott kézjelet és a hozzá tartozó parancs (pl. ötszáz értékpapír eladása négy egységért) szöveges megfelelőjét.

A kezdeti információszerzés során megtudtuk, hogy a tőzsdei kézjelek kultúránként eltérőek lehetnek, ugyanis ezek nem egyszerre alakultak ki. A tendencia azt mutatja, hogy kézjelek fejlődése az egyszerűsödés irányába mutat, a szabályok nagy része azonban megegyezik a 19. század végén kidolgozott alapelvekkel [1].

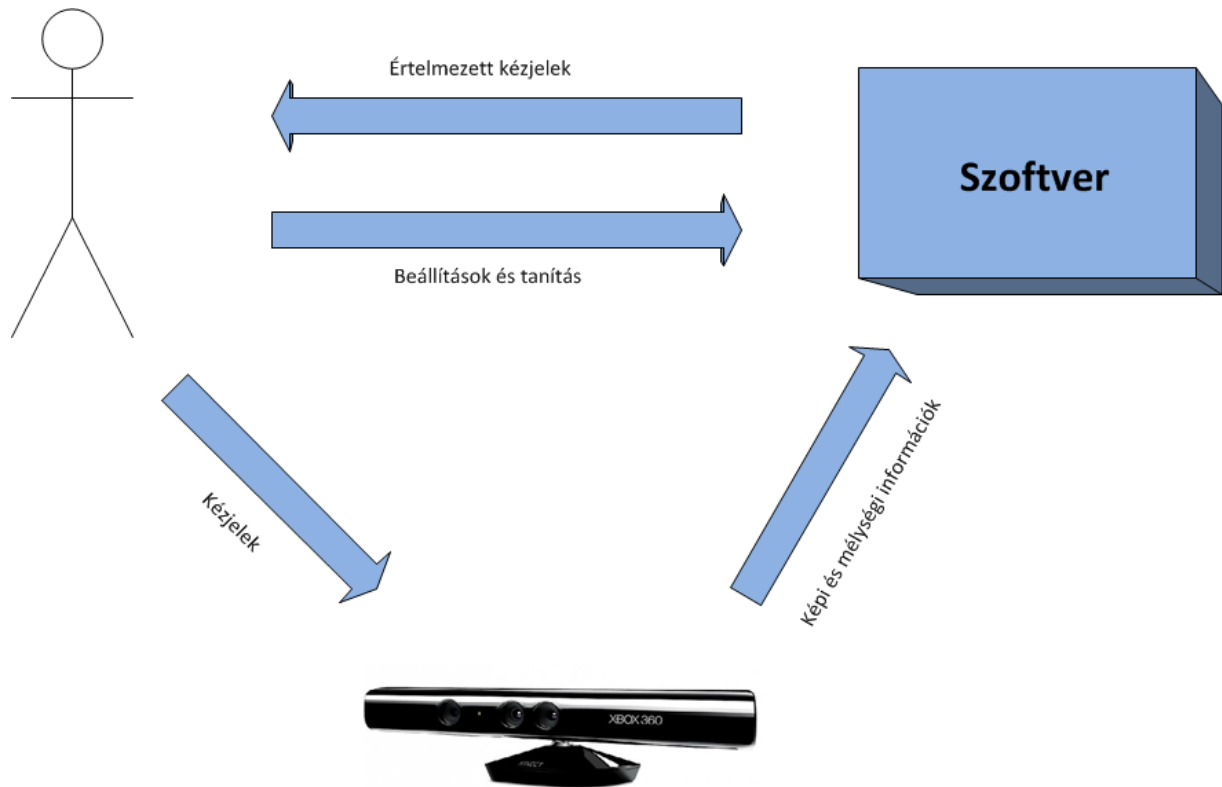
A kézjelek detektálásához azért választottuk a Kinect szenzort, mert a képi információ mellett (amit egy átlagos webkamera is szolgáltat) megkapjuk a pixelekhez rendelt távolsági adatokat is. Ez a többletinformáció jelentősen javítja a jelek felismerhetőségét.

Kézjelek (és karjelzések) Kinect szenzorral való felismerésével többen foglalkoznak (lásd: [Irodalomkutatás](#)), illetve különböző keretrendszerek is segítik a fejlesztőket: vannak kifejezetten kézjelekre szakosodott és a humán interakció szélesebb tárházát támogató rendszerek is. Célunk, hogy – figyelembe véve a rendelkezésre álló rövid időt – a létező megoldásokra maximálisan építsünk, azokat szükség szerint továbbfejlesszük és finomítsuk.

A szoftvert előzetesen meg fogjuk tanítani a kézjelek felismerésére. Ennek mikéntjét a későbbiekben fogjuk kialakítani.

A szoftver működésének magas szintű leírása, tanítás után:

- A felhasználó a számítógép képernyője és a Kinect szenzor előtt áll. A szoftver élőképet mutat a Kinect kamerájából.
- A felhasználó tetszőleges időközönként, akár folyamatosan is, kézjeleket mutat.
- A szoftver a jeleket valós időben feldolgozza és a felismert jelek szöveges megfelelőjét kiírja.



1. ábra: A rendszer elvárt működése

Irodalomkutatás

Mivel a használt beviteli eszköz a Kinect szenzor, adott, hogy a fejlesztésnél is leginkább ezt alkalmazzó, lehetőleg a mi általunk is használandó C# nyelven, .NET keretrendszerben íródott megoldásokat kutassuk fel.

A tőzsdei kézjelek

Vétel, eladás¹

BUY



SELL



2. ábra: a „vétel” és „eladás” kézjelei jól megkülönböztethetők

Ár

ONE



TWO



THREE



FOUR



¹ Az itt látható illusztrációk mind a Chicago Mercantile Exchange történetében használt kézjeleket ábrázolják és a CME által kiadott információs kiadványból származnak [1].

FIVE



SIX



SEVEN



EIGHT



NINE



EVEN



3. ábra: a számok kézjelei viszonylag jól elkülönülnek, bár gondot okozhat, hogy az 1-4-et a 6-9-től csak a kéz iránya különbözteti meg

A számok kézjelei viszonylag jól elkülönülnek, bár gondot okozhat, hogy az 1-4-et a 6-9-től csak a kéz iránya különbözteti meg. Fontos tudnivaló, hogy az ár esetében a jelzett szám csak az ár utolsó számjegyére vonatkozott.

Mennyiség

ONE



TEN



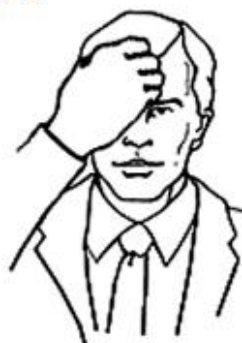
SEVEN



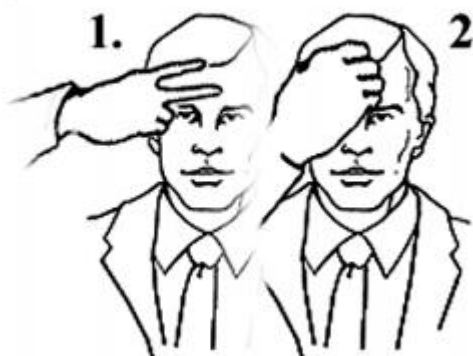
NINETY



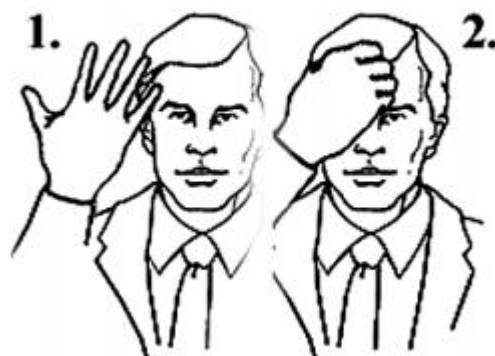
ONE HUNDRED



FIVE HUNDRED



SEVEN HUNDRED



4. ábra: a mennyiségek kézjelei

A mennyiségek kézjelei alapvetően ugyanazok, mint az árhoz használtak, viszont szorzójuk függ a kéz archoz viszonyított helyzetétől:

- Az egyszeres szorzóval jelzett mennyiségeknél (azaz 1-10) a szám mutatásakor a mutató kézzel meg kell érinteni az állat.
- Tízszeres szorzóhoz (azaz 10-100) meg kell érinteni a homlokot.
- Százás szorzóhoz először homlokhoz érintve mutatni kell a számot, majd ököllel meg kell érinteni a homlokot.

Egyéb jelzések



5. ábra: feltartott hüvelykujj mutatja, hogy a tranzakció létrejött („OK”)



6. ábra: a kéz végigmozgatása a torok előtt jelentése: mégse, a tranzakció visszavonása

Mindezek a kézjelek nem tartalmazzák az összes valaha használatos tőzsdei kézjeleket, viszont a szoftver megvalósításakor ezekre a fontosabb és általánosabb jelzésekre koncentrálunk.

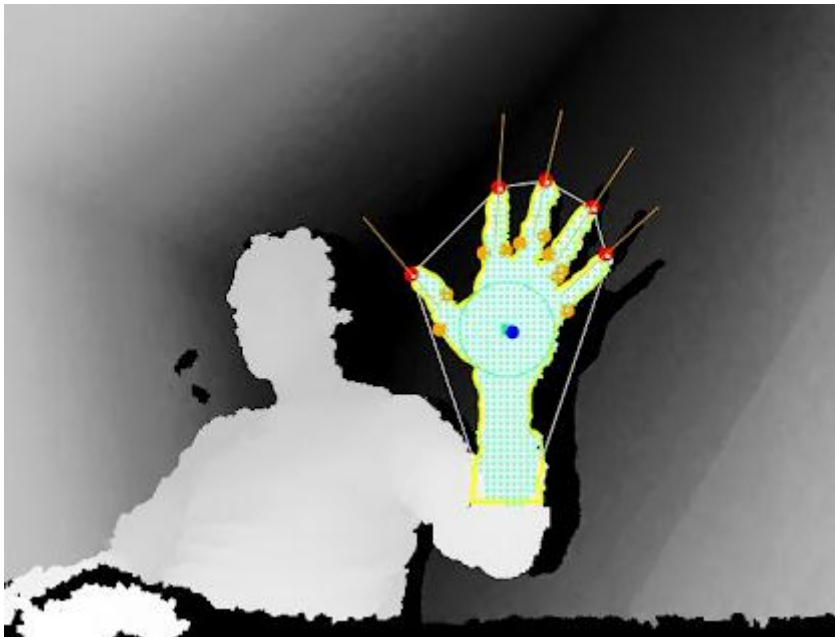
Kapcsolódó szoftverprojektek

Kinect for Windows SDK

A hivatalos SDK Kinect fejlesztéshez mindenképp szükséges, mivel a Kinect API-kat tartalmazza (valamint rendelkezik sok példaprogrammal is) [2].

Candescent NUI

A Candescent NUI kifejezetten kézjelek felismerésére, ujjak követésére is használható keretrendszer. Úgy gondoljuk, hogy ez a projekt nagy segítség lesz számunkra ugyanis ebben már megvalósítottak egy ujjak detektálására és követésére használható funkciót [3].



7. ábra: Képernyőkép a Candescent NUI példaprogramjából

A fenti képernyőkép a program ujjfelismerő képességeit demonstráló példaprogramból származik. Ezen jól látható, hogy az ujjbegyek, az ujjtövek illetve a tenyér középpontja kerül felismerésre. Lehetőséget látunk abban, hogy ezen pontok egymáshoz viszonyított helyzetéből tudjunk kézjeleket felismerni. Természetesen figyelembe kell vennünk azt is, hogy a kézjelek, amiket a szoftvernek fel kell ismernie, nem mindig szabályosak. Ennek tudatában fogunk alkalmazni olyan gépi tanuló algoritmust, melynek segítségével megoldható a nem teljesen szabályos, torz kézjelek bizonyos tűréshatárokon belüli felismerése is.

A keretrendszer API-ja végső soron térbeli pontok halmazaként adja vissza a kéz különböző pontjainak (ujjak, tenyér pontjai) helyzetét.

A Candescent NUI opcionálisan használhatja az OpenNI keretrendszert a Kinect SDK helyett [4].

Kinect Toolbox

A Kinect Toolbox keretrendszer kifejezetten gesztusok és kézpozíció felismerésére is használható [5].

Kinect SDK Dynamic Time Warping Gesture Recognition

A [6] cím alatt elérhető project szintén keretrendszer. Elsősorban teljes testmozdulatok felismerésére szánták. E tekintetben hasonló célból használhatnánk fel, mint a Kinect Toolbox-ot. Ugyanakkor az utóbbival ellentétben nem rendelkezik kielégítő dokumentációval, elegendő példaprogrammal és aktivitással.

Encog Machine Learning Framework

Az Encog teljes körű gépi tanulást használó szoftverek létrehozását segítő érett, aktívan fejlesztett keretrendszer, neurális hálózatokat is támogat [7].

Aforge.NET

Az Aforge.NET egy C#-ban megvalósított neurális hálózatok létrehozását segítő keretrendszer. A fejlesztése és felhasználói bázisa nem aktív és kevés szolgáltatással rendelkezik [8].

Egyéb

Áttekintettük még az alábbi keretrendszereket is:

- **Fast Artificial Neural Network Library:** Ez egy nyílt forrású C nyelven implementált neurális hálózat könyvtár. Előnye, hogy könnyen kezelhető és jól tanítható adatszerkezeteket használ, jól dokumentált, gyors, ugyanakkor .NET szoftverekben korlátozottan használható [9].
- **Openi NI:** Szintén egy nyílt forrású keretrendszer. A fejlesztéséért egy non-profit szervezet felelős mely azért jött létre, hogy az ember – gép kommunikációt segítő eszközök és szoftverek fejlesztését segítse. Szintén jól dokumentált és sok példaprogramot tartalmaz [10].
- **OpenCV:** Nyílt forrású, gépi látásra és gépi tanulásra kifejlesztett szoftverkönyvtár. Több mint 2500 jól optimalizált beépített algoritmust tartalmaz [11].

Összegzés és kiindulási tervek

Terveink szerint a szoftver megvalósításánál a következőket fogjuk használni:

- Kinect SDK mindenképp szükséges
- Candescent NUI és Kinect Toolbox a karok helyzetének és a kéz jellemzőinek felismeréséhez
- Encog Machine Learning Framework a keretrendszerek által adott adatok neurális hálózattal való felismertetéséhez

Az október 8-án megjelenő új Kinect SDK némileg módosíthatja még elképzeléseinket, ugyanis az tartalmazhat olyan szolgáltatásokat, amiket most még szükséges lenne nekünk kifejleszteni.

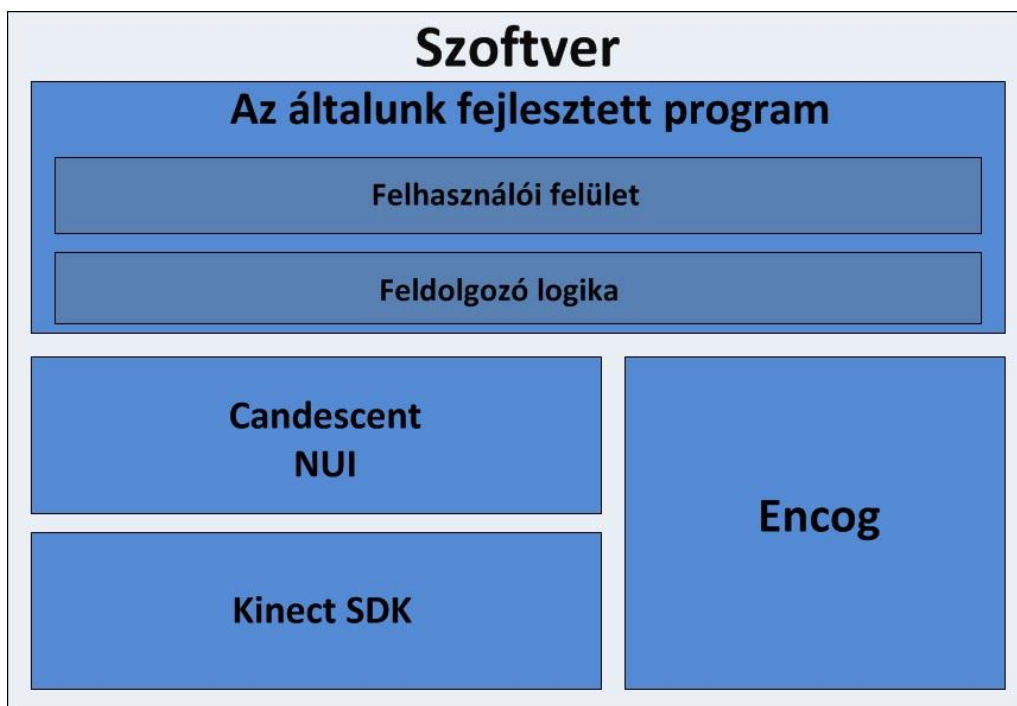
Megvalósítás

A következő rész bemutatja a blokkdiagram alapján a szoftver működési elvét és a tanítási mechanizmust, tehát azt a folyamatot, ami alapján a Kinect szenzorból érkező adatokból előáll a felismert kézjel.

A szoftver szerkezete

A szoftver elkészítésénél ügyeltünk a modern szoftverfejlesztési megoldások alkalmazására és bevált minták követésére. A programban például elkülönülnek az implementációk és az interfészek, ezáltal az egyes részek egyébként csak lazán csatolódnak egymáshoz.

Kezdetektől fogva az volt a cél, hogy a felhasználói interakciót biztosító programrész teljesen el legyen választva a kézjel-feldolgozási logikától azzal a céllal, hogy ez utóbbi majd – amikor már nem csak a demonstráció a cél – más programokban is könnyen felhasználható legyen.



8. ábra: A szoftver blokkdiagramja

A Kinect Toolbox használata végül fölöslegesnek bizonyult.

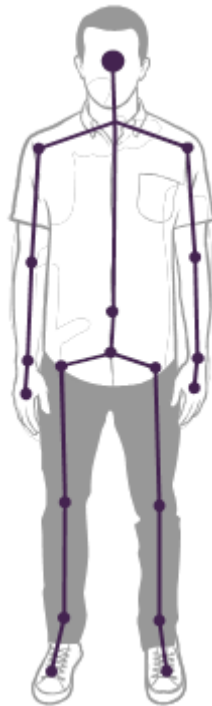
Kéz- és ujjdetektálás

A program működésének alapja a felhasználó által mutatott kezek (amiből elviekben kettőnél több is lehet több játékos esetén; a szoftver erre az esetre fel van készítve, de jelenleg csak két kéz adatait használja) megjelenésének érzékelése, majd azok paramétereinek, mint a tenyérközep és az ujjak helyzete, megállapítása.

Kinect Skeleton modell

Az általunk felhasznált Kinect SDK alapvetően nem szolgáltat információkat a kézfejről. Az általa nyújtott legrészletesebb, feldolgozott, a játékos testhelyzetét leíró adat az úgynevezett Skeleton váz.

Ebben a vázban kitüntetett pontként szerepel a lábfej, a térd, a csípő, a tenyér, a csukló, a könyök, a váll illetve a fej.



9. ábra: A Kinect SDK által előállított Skeleton [14]

Kéz- és ujjfelismerés a Candescent NUI keretrendszer segítségével

Mivel számunkra a kézfej és azon belül is az ujjak detektálása a cél, ezért sokkal részletesebb leírásra van szükségünk. Így erre a célra a már említett Candescent NUI Kinect szenzorhoz fejlesztett kézfelismerő-keretrendszert használtuk.

A Candescent NUI beépített funkciója, hogy a kezek, tenyérközéppontok és ujjak pozícióját, valamint a kéz kontúráját meghatározza. A keretrendszertől a HandCollection osztályban kapjuk meg a detektált kézfejek számát illetve azt a HandData típusú elemekből álló listát, ami már számunkra megfelelő információkat tartalmaz.

A HandCollection osztály:

```
public class HandCollection
{
    public HandCollection();
    public HandCollection(IList<HandData> data);

    public int Count { get; }
    public IList<HandData> Hands { get; }
    public bool HandsDetected { get; }
    public bool IsEmpty { get; }
}
```

A HandData osztály:

```
public class HandData : IHand, ILocatable
{
    public HandData(int id, Shape.Shape shape, Palm palm,
        IList<FingerPoint> fingerPoints);

    public Contour Contour { get; }
    public ConvexHull ConvexHull { get; }
    public int FingerCount { get; }
    public IList<FingerPoint> FingerPoints { get; }
    public IEnumerable<IFinger> Fingers { get; }
    public bool HasContour { get; }
    public bool HasFingers { get; }
    public bool HasPalmPoint { get; }
    public int Id { get; }
    public Point Location { get; }
    public IList<FingerPoint> NewlyDetectedFingerPoints { get; set; }
    public double PalmDistance { get; }
    public Point? PalmPoint { get; }
    public float PalmX { get; }
    public float PalmY { get; }
    public Volume Volume { get; }
}
```

A program a kézjel-felismerés során ebből az osztályból az ujjakra illetve a tenyérre vonatkozó adatokat használja fel és alakítja ki azt az adatstruktúrát, ami lehetővé teszi a különböző kézjelek tárolását és tanítását.

Kézzatok feldolgozása

Első lépésként szükséges, hogy a Candescent által adott adatokból olyan tisztított adathalmazt hozzunk létre, amit a későbbiek során jól tudunk kézjel-felismerésre használni. Ennek érdekében kerestünk jó leírószámokat, a gépi tanuló algoritmusoknak ugyanis számokra van szükségük.

A feldolgozó logikában a kezek leírását egy HandDescriptor nevű osztály valósítja meg, melyben egy bool tömböt (AreaDescriptor), egy lebegőpontos skalár számot (ScalarProductDescriptor) valamint egy lebegőpontos számokból álló tömböt (AngleDescriptor) tárol. Ez utóbbiban radiánban mért szögeket tárol el.

```
public interface IHandDescriptor
{
    bool[,] AreaDescriptor { get; }
    float ScalarProductDescriptor { get; }
    float[] AngleDescriptor { get; }
}
```

A Kinecttől érkező adatok feldolgozása a HandDataProcessor osztályban történik, ez állítja elő az IHandDescriptor objektumokat is (IProcessedHandData-ba csomagolva).

```
public interface IHandDataProcessor
{
    event HandProcessedHandler HandProcessed;

    void HandDetectedHandler(HandCollection handCollection);
    IProcessedHandData Process(HandCollection handCollection);
}
```

Ahogy a fenti kódrészlet is mutatja, az osztály Process metódusa kapja meg a Candescent által szolgáltatott HandCollection osztály egy példányát, melyből a kéz adatainak feldolgozása megtörténik. Ennek eredménye a kezet leíró adatok halmaza:

- AreaDescriptor: a kéz kontúrjai által körbezárt terület
- ScalarProductDescriptor: az ujjvektorok skaláris szorzata
- AngleDescriptor: az ujjvektorok függőlegessel bezárt szöge

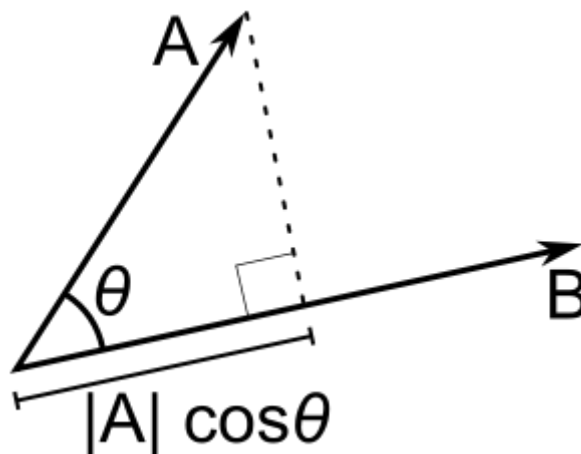
Az **AreaDescriptor** a Candescent által megadott kézkontúrok alapján készít egy feketén kitöltött kézképet. Ezután ezt lekicsinyíti 30*30 pixeles méretre, annak érdekében, hogy a felhasználó kézméretétől függetlenül ugyanakkora leírókép álljon elő. Ezt a monokróm képet egy egyszerű kétdimenziós boolean tömbben (igaz értékek jelölik azokat a pontokat, ami belül esik a kéz kontúrján) tároljuk.

A **ScalarProductDescriptor** mérőszám kiszámításának alapvetően négy fázisa van:

1. Az első fázisban a kéz „átlagolt” középpontjának a kiszámítása történik. Ez a Candescent által szolgáltatott kézközéppontot és tenyérközéppontot veszi, majd (mivel ezek zajosak és pontatlanak lehetnek) kiszámítja a két pont közötti pontot.
2. A második fázisban a már kiszámított átlagolt középpont és az ujjak végpontjait összekötő vektorok kiszámítása történik meg.
3. A harmadik fázis a kiszámított vektorok normalizálását végzi el.
4. Végül a vektorok skaláris szorzatát állítjuk elő.

Maga a mérőszám tíz ilyen mérés számtani átlagából áll össze.

Az **AngleDescriptor** tömb a kiszámított vektorok által bezárt szögeket tartalmazza radiánban mérve. A 10. ábra [12] szemlélteti a vektorok által bezárt szöget a 11. ábra [12] pedig a számításhoz szükséges képletet.



10. ábra: Két vektor által bezárt szög

$$\theta = \arccos \left(\frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \|\mathbf{b}\|} \right).$$

11. ábra: Két vektor által bezárt szög kiszámítása

A fejlesztés során kiderült, hogy a keretrendszer által adott adatok egyéb szűrés és az alapértelmezett beállítások változtatása nélkül túl zajosak ahhoz, hogy a további lépésekhez használhatóak legyenek. Ennek javítására a következő megoldásokat alkalmaztuk:

- Beállítottuk, hogy a Candescent 10-10 képkocka alapján döntsön egy új ujj megjelenéséről és egy ujj eltűnéséről.
- Alkalmaztunk egy egyszerű algoritmust a nyilvánvalóan hibásan felismert ujjak szűrésére: amennyiben egy kézhez ötnél több ujj tartozik, a program eldobja a legtávolabbi ujjak adatait (ilyen hibása felismert ujjak jelenhetnek meg például egy buggyos ujjú pulóveren).
- A ScalarProductDescriptor számítása tíz (de konfigurálható mennyiségű) mérési adat átlagolásával történik.

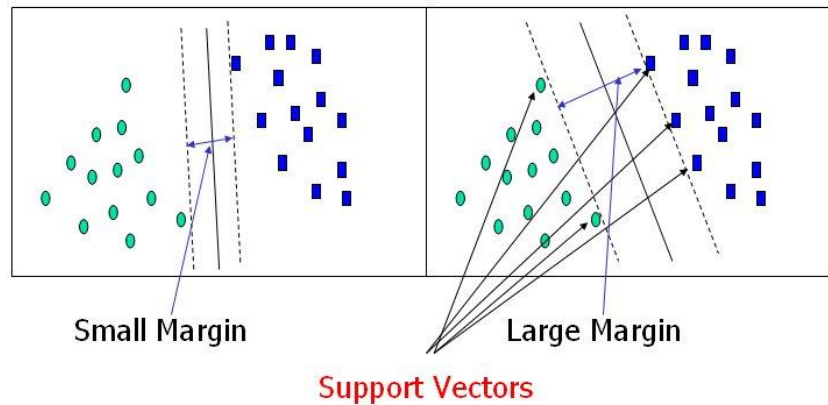
Kézzel-felismerés

Az Encog gépi tanulás keretrendszer biztosítja a program számára a tanulási és felismerési folyamatban felhasznált Support Vector Machine algoritmust.

Azért választottuk a Support Vector Machine-t a rendelkezésre álló gépi tanulás algoritmusok közül, mert az esetünkben megoldandó feladat osztályozási feladat, amire az SVM szolgál.

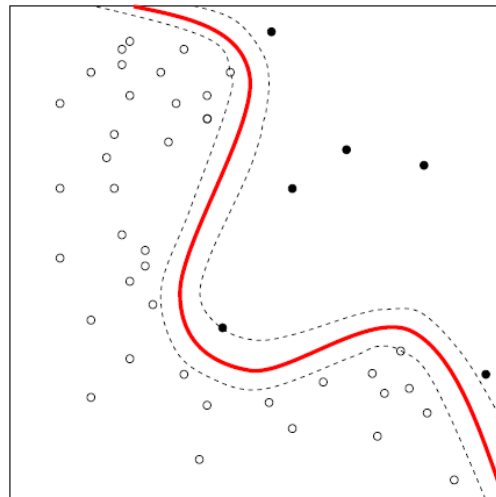
A következő összefoglaló az SVM algoritmusokról a [13] és [15] forrásanyagok alapján készült el.

Az SVM típusú algoritmusok célja egy olyan hipersík keresése, mellyel a két különböző osztályba tartozó elemeket jól el tudja különíteni. A keresés során az elsődleges szempont az, hogy a hipersík egy olyan - az elválasztó síkkal párhuzamos hipersíkokkal meghatározott - térrésszel rendelkezzen, ami a lehető legnagyobb és nem tartalmaz tanító mintákat (12. ábra). Ezt a térrészt margónak nevezzük. A legnagyobb szélességű margó kiválasztása azért fontos, mert az így választott elválasztó sík jól általánosít és az osztályozást pontosabbá teszi.



12. ábra: SVM margó

Az SVM algoritmusok alapvetően kétfajta osztályozási sémát támogatnak: lineáris és nem-lineáris klasszifikáció. Míg a 12. ábra egy lineáris SVM-et ábrázol addig a 13. ábra egy nem-lineáris SVM-et szemléltet.



13. ábra: Nem-lineáris SVM

Lineárisan szeparálható esetben (12. ábra) a

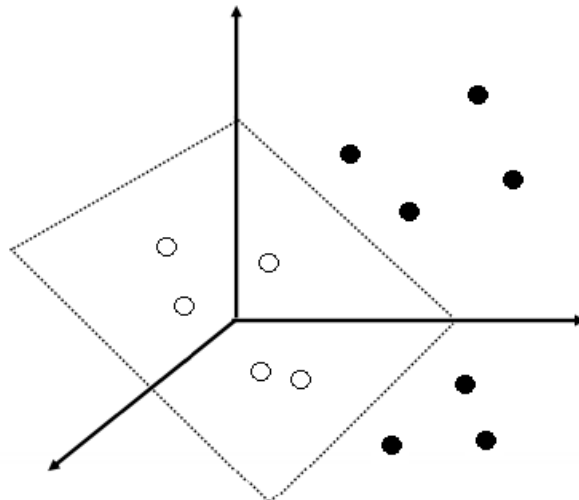
$$g(x) = W * x + b \quad (1)$$

függvény (ahol W az adott sík normálvektora, x egy pont a tanítóhalmazból és $*$ ebben az esetben skaláris szorzat) egy síkot határoz meg a tanítóhalmazban ami alapján a szeparáció megoldható. Ha a tanítóhalmaz nem lineárisan szeparálható a (1) függvény által meghatározott sík rossz klasszifikációt ad eredményül.

Ebben az esetben az SVM algoritmusok által használt ún. kernel függvények közül nem lineáris függvényt kell alkalmaznunk. A nem lineáris kernel függvény a tanítóminták által alkotott síkot egy olyan térbe képezi le melyben már megoldható a lineáris szeparáció:

$$\omega: x \rightarrow \varphi(x) \quad (2)$$

Az ω leképezés után keletkező teret a 14. ábra szemlélteti.



14. ábra: A leképezés után keletkező tér

Az ω transzformációban feltüntetett φ függvénynek az Encog keretrendszer által támogatott függvények közül a RadialBasisFunction - t választottuk ugyanis a tesztelések során ezzel a függvénnyel értük el a legjobb felismerési eredményeket.

Az SVM tanítása során a kézleíró számokat és a hozzájuk rendelt kézjel nevét használjuk fel.

Tanítási mód

A tanítás első lépése a kezletíró adatok rögzítése (azaz „felvesszük” egy mutatott kézjel leíró adatait). A rögzítés után ezek az adatok fájllokba kerülnek, és a későbbiekben innen kerülnek betöltésre a tanításhoz.

A kimentett adatok egy részlete látszik alább egy példafájl tartalmából, mely az „öt” kézjel mintáját tartalmazza:

```
{ "SignalName": "Five", "Measurements": [ [ { "AreaDescriptor": [ false, false, false, false, ... ], "ScalarProductDescriptor": 2.052339, "AngleDescriptor": [ 2.5929662560881011, 2.2401523226000957, 1.937718715592802, 1.3948944966822636, -2.5463832759441694 ] }, { "AreaDescriptor": [ false, true, true, false, ... ], "ScalarProductDescriptor": 2.048311, "AngleDescriptor": [ 2.6276726656215179, 2.2557770258321783, 1.9136611317535752, 1.43370410682545, -2.4857851140786686 ] }, { "AreaDescriptor": [ true, true, true, false, ... ], "ScalarProductDescriptor": 2.10487723, "AngleDescriptor": [ 2.6474764561220647, 2.2368166842040145, 1.9326545164959661, 1.4570740569721266, -2.5409924938565935 ] } ] ] }
```

Felismerési mód

A felismerési folyamat alatt az SVM a kapott adatokat osztályozza (itt osztály alatt az SVM klasszifikáló kimenetét értjük, ami egy egész szám). Mivel a tanítás során már kiderült, hogy az SVM mely kézjelhez mely osztályt rendeli, a felismerés során a bemeneti adatokra kapott osztály-érték hozzárendelhető egy kézjelhez.

A programot jelenleg a következő alapvető kézjelek felismerésére tanítottuk meg (ez a lista azonban további tanítással bővíthető):

- Vétel, eladás
- Számok egytől ötig
- OK

A felismerés vizsgált módszerei

A program fejlesztése során fokozatosan sikerült felfedeznünk a kifejlesztett módszerek gyengeségeit és ennek megfelelően javítottunk a felismerés hatékonyságán. Tapasztalataink a következők voltak:

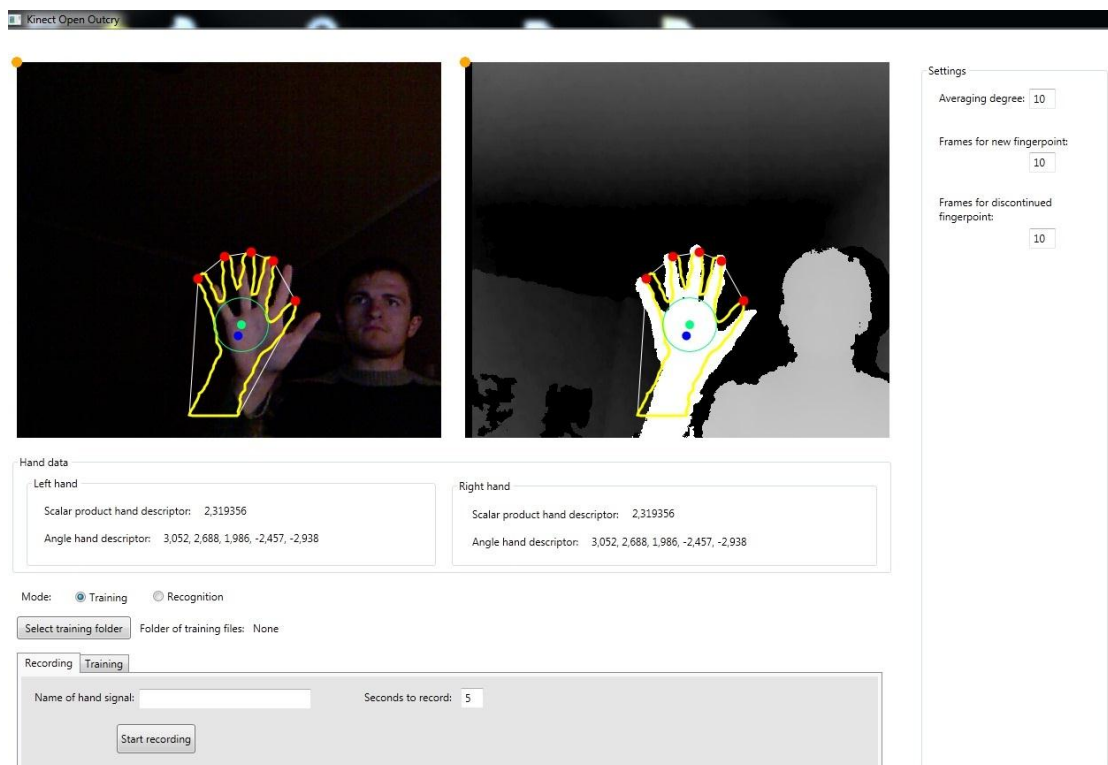
- A ScalarProductDescriptor bár valóban egy, a kézjelre jellemző számot állít elő, ez, mivel távolról sem képez egy-egyértelmű megfeleltetést az ujjak és a kézjel között (mivel a skaláris szorzat eredménye nem csak egyféleképp állhat elő), önmagában nem használható, de más módszerek pontosságát javíthatja.
- Az AngleDescriptor már egy viszonylag jó adatsor. Az ujjak pozícióját jól leírja, így már használható felismerésre.
Hátránya, hogy mivel csak kezenként legfeljebb öt adatból áll, a gépi tanuló algoritmusok könnyen elsiklanak a kezek közti különbségek között. Az általunk használt SVM sokszor nem volt képes a használt ujjak számában is eltérő jeleket emiatt felismerni.
További probléma, hogy ez a megoldás az ujjvégek felismerését igényli, ami csak akkor működik jól, ha a felhasználó nagyon odafigyel mutatáskor.
- Az AreaDescriptor minden tekintetben az eddigi legmegfelelőbb módszer. Szélesebb tűréshatárok között, mégis pontosabban lehet vele SVM-mel mintát felismertetni.

A végső implementációban csak az AreaDescriptor monokróm képét használtuk fel.

Az SVM konfigurálásához kipróbáltunk az Encog többféle kernel függvényét (lineáris, radiál alapú, polinomiális, szigmoid), de végül a radiál alapú bizonyult a leghatásosabbnak.

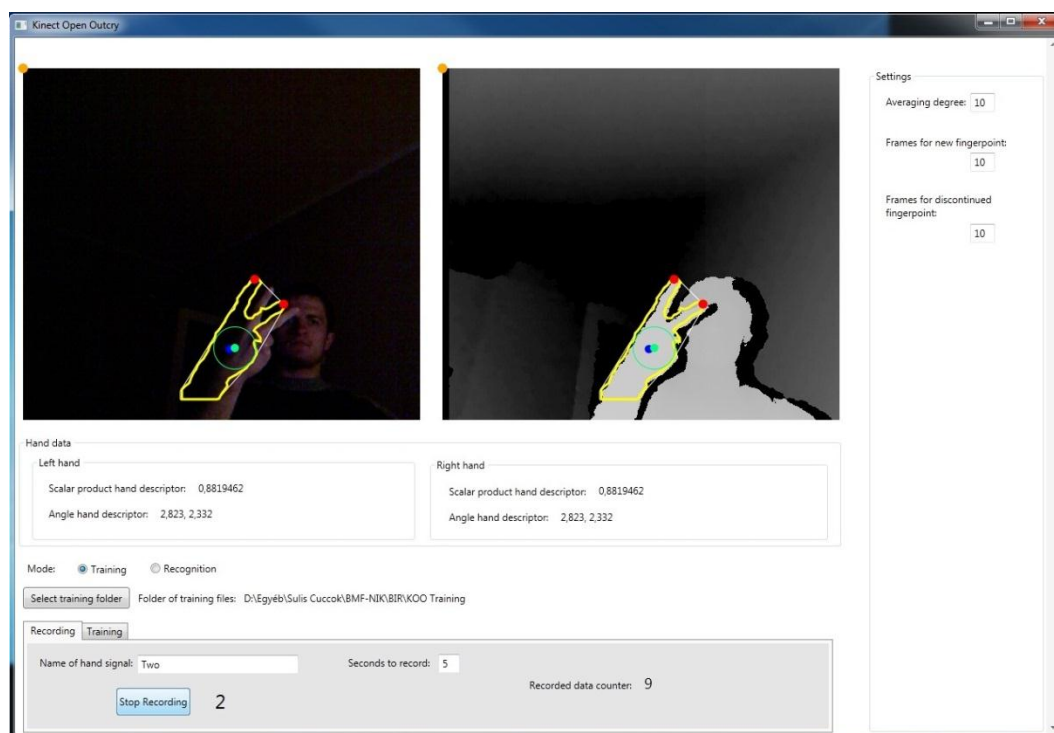
Felhasználói felület

A felhasználói felületen minden esetben megjelenik a Kinecttől érkező RGB és mélységi kép, valamint az ujjak detektálása is látható.

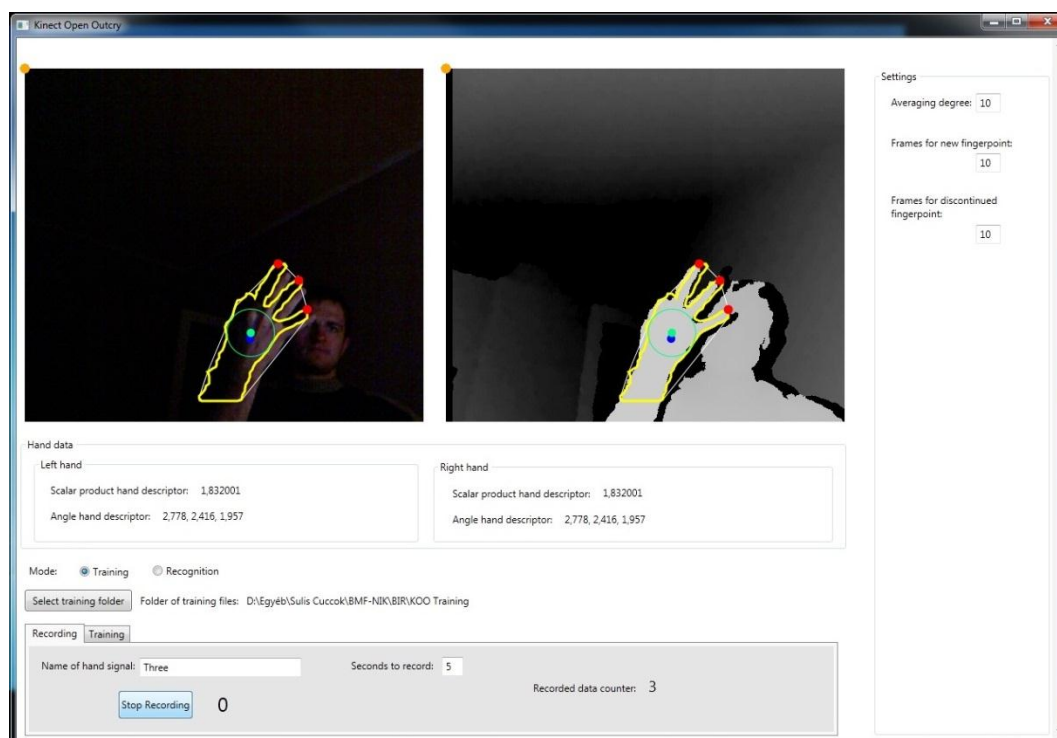


15. ábra: Az RGB és mélységi kép, valamint a felhasználó felület egyéb részei (módválasztó, beállítások)

A kézjelek rögzítése során a felhasználónak „Training” módba kell lépnie, meg kell adnia egy mappát, amibe a tanító fájlok kerülnek, a „Recording” fülön meg kell adnia a mutatott kézjel nevét, végül pedig egy időtartamot, amely a felvétel hosszát határozza meg.

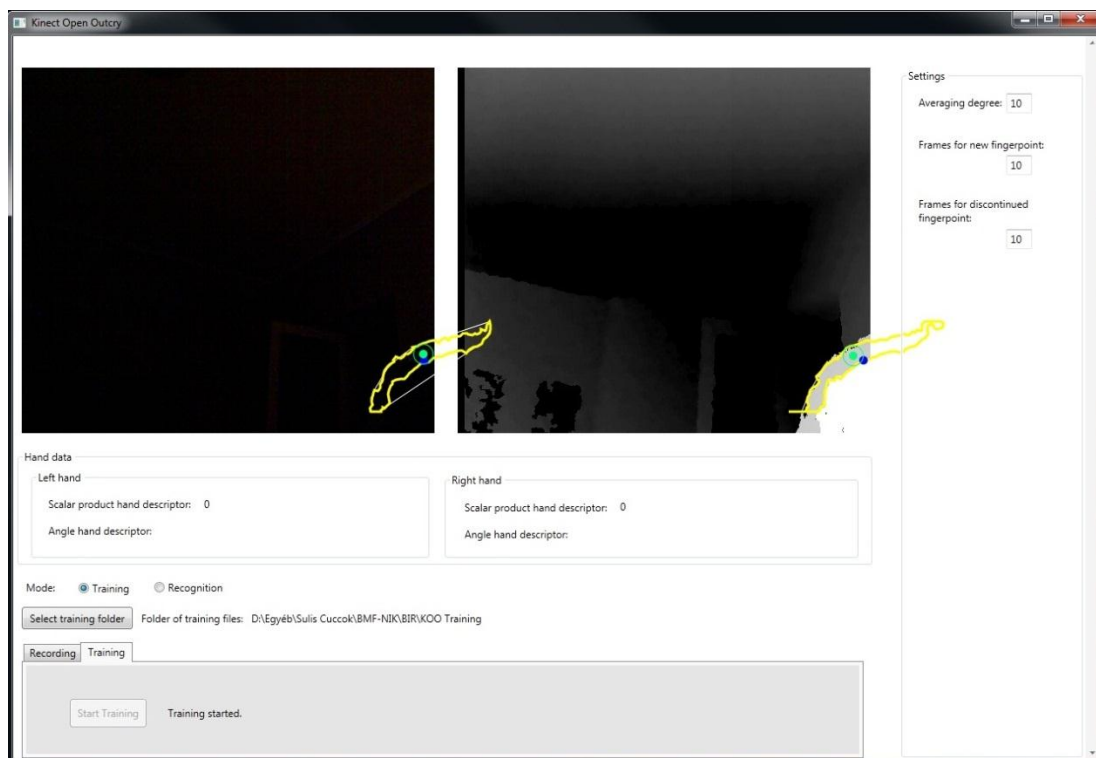


16. ábra: A 2-es kézjel rögzítése
(a "Recording data counter" mutatja a rögzített adatok számát)

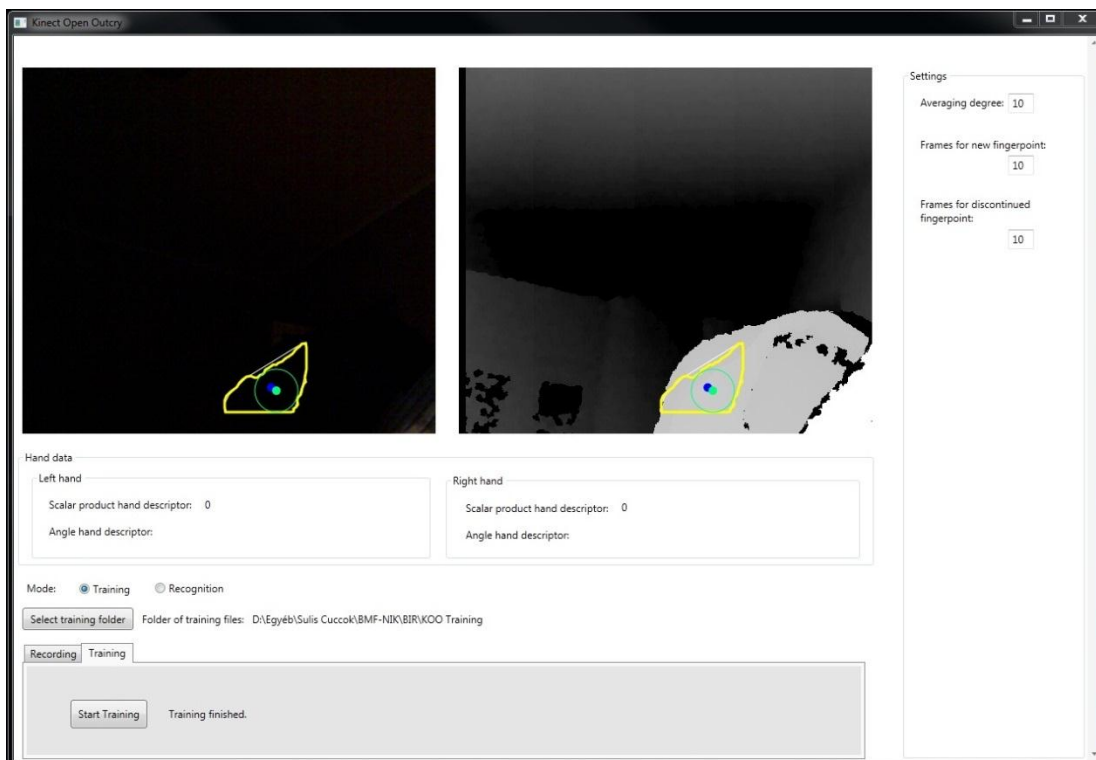


17. ábra: A 3-as kézjel rögzítése

A tanítás során „Training” módban meg kell adni azt a mappát, amiben a tanítófájlok találhatóak, majd a „Training” fülön a „Start Training” gombra való kattintás után a „Training started” felirat jelzi a tanítás kezdetét és „Training finished” a befejeztét.

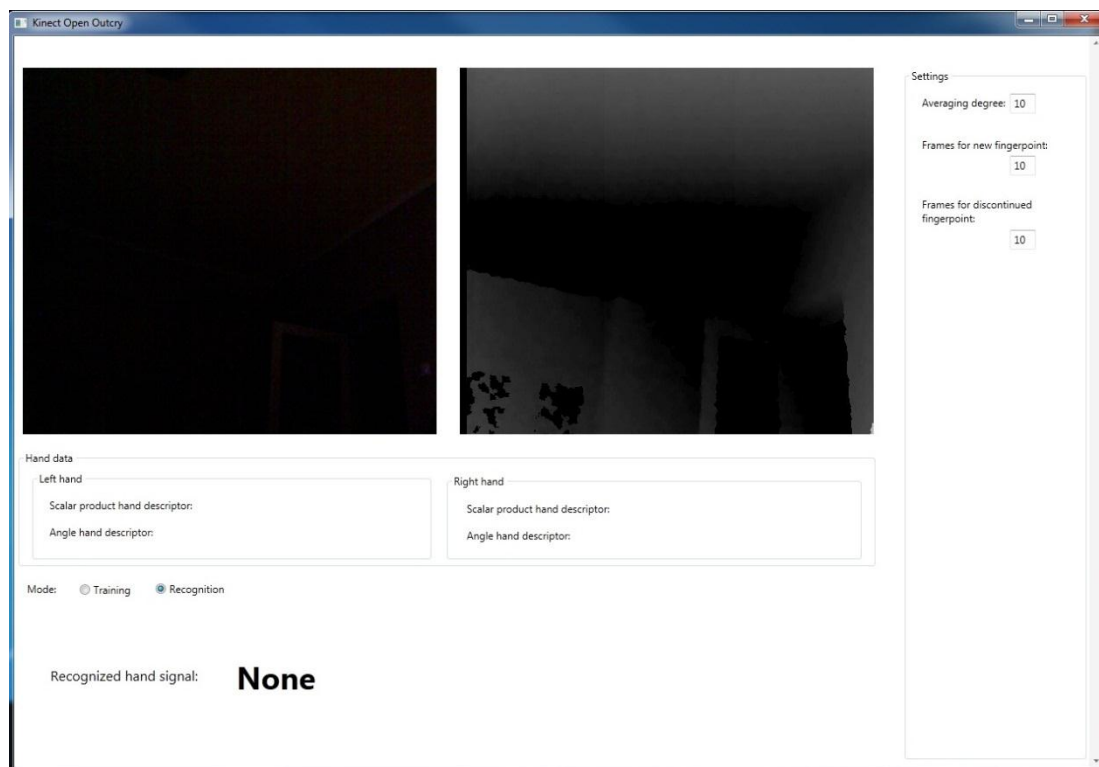


18. ábra: A tanulás megkezdése
(tanulás közben mindegy, mit „lát” a szenzor)

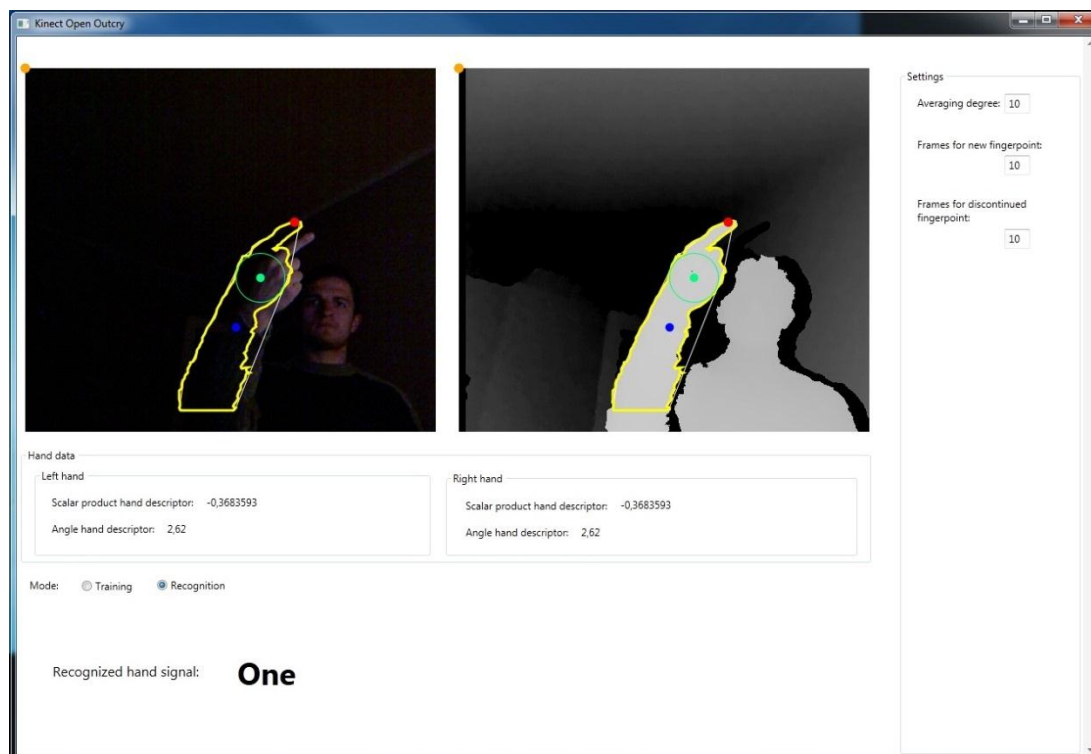


19. ábra: A tanulás vége

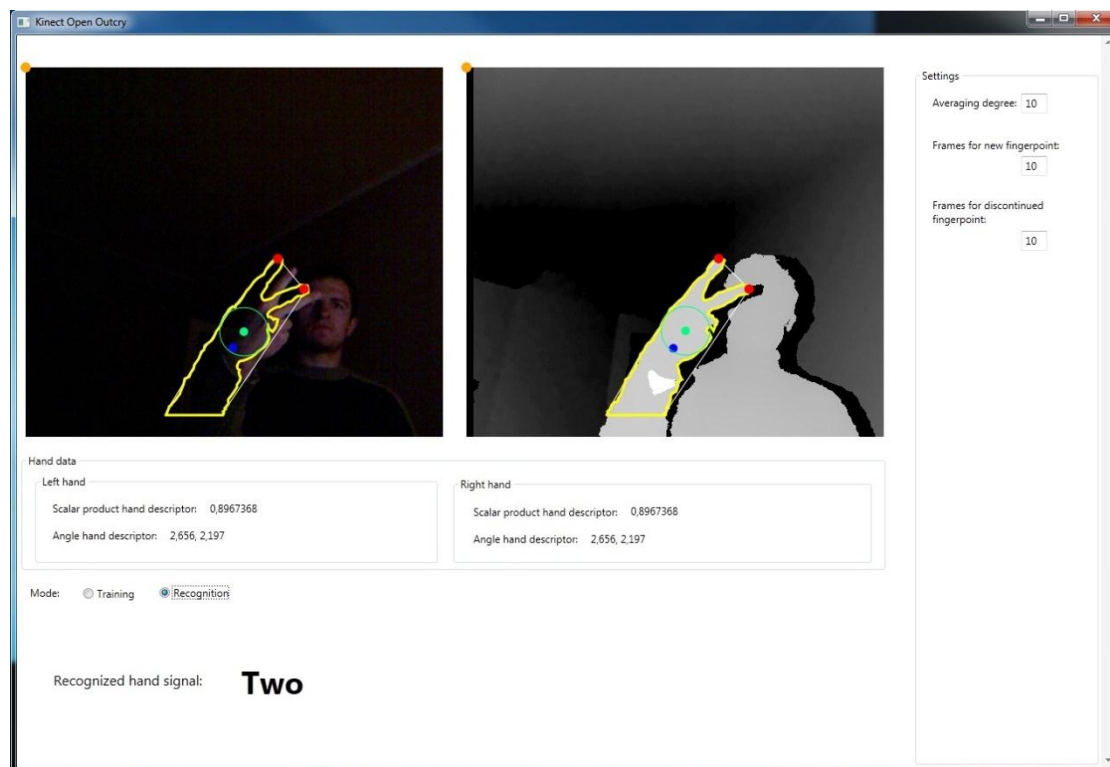
A tanítás után már képes a program a megtanított kézjelek felismerésére. Ehhez a felhasználónak „Recognition” módba kell váltania.



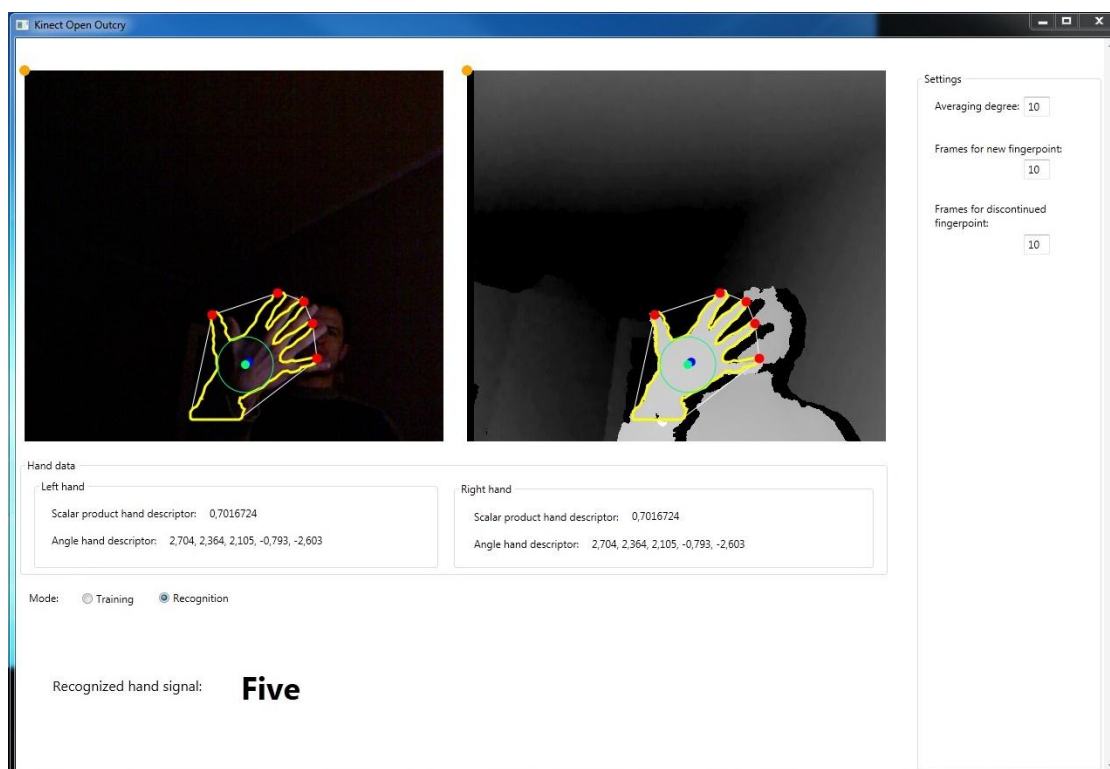
20. ábra: Nincs kézjel



21. ábra: 1-es kézjel felismerése



22. ábra: 2-es kézjel felismerése



23. ábra: 5-ös kézjel felismerése

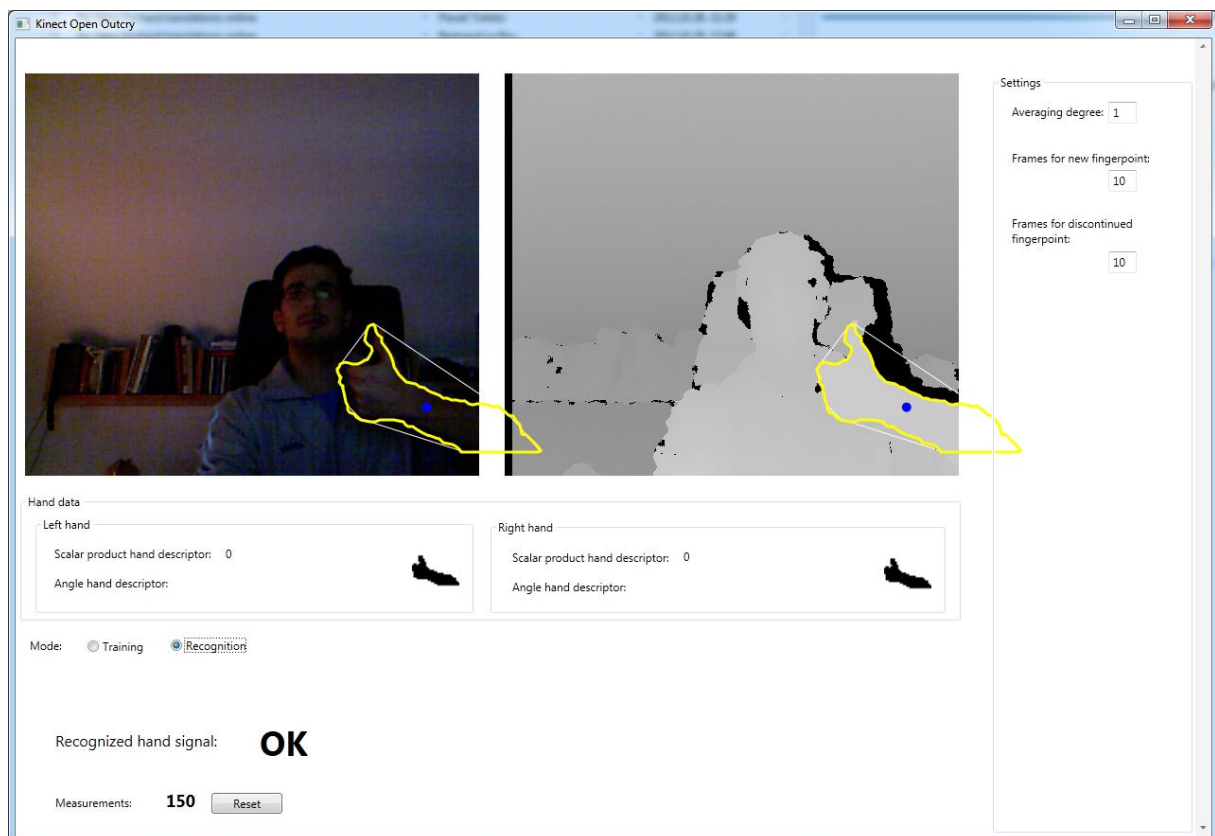
Hardver- és szoftverigény

A szoftver működéséhez természetesen szükséges egy csatlakoztatott és telepített driverekkel ellátott Kinect szenzor. Ezen kívül tapasztalataink szerint a program gördülékeny futtatásához kétféle, 2,5Ghz-es CPU-val és 1GB szabad RAM-mal rendelkező számítógépre van szükség, de a kézjel-felismerés változatlanul valós időben működik 2Ghz-es órajelű processzorral is (ekkor azonban a vizualizáció sokat késik).

A bemutatott szoftver .NET keretrendszeren futó program, így – legalábbis elviekben – minden olyan platformon képes futni, ahol a .NET és a Kinect támogatás implementálva van.

Eredmények

Az e dolgozat elkészültének pillanatában aktuális implementáció az alábbi paraméterekkel bír (a számok csak viszonyítási alapot képeznek). A teszthardver egy 4 darab 3,2Ghz-es maggal ellátott Intel i7-es processzorral rendelkező asztali számítógép volt.



24. ábra: a program legfrissebb állapota. Ez már csak az AreaDescriptort használja. Az videóképek alatti kis fekete képek a felismert kezek élőképei.

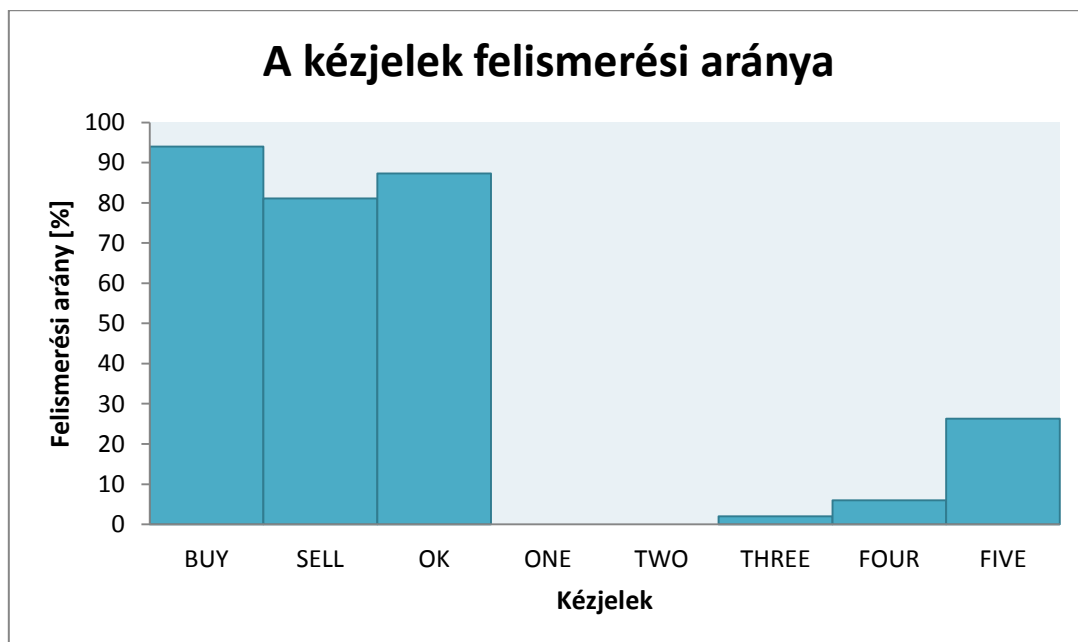
- A tanítóminta felvételekor alapértelmezésként hat másodpercig kell egy kézjelet mutatni. Ez mintegy 180 mérésnyi kézzeljel jelent (amennyiben van felismert kéz, a kézzeljelek a képi adatokkal együtt másodpercenként harmincszor érkeznek).
- Tapasztalataink szerint a vizsgált alapvető minták (a már említett vétel, eladás, OK, számok egytől ötig) esetében a fenti mérési gyakorisággal legtöbbször egyetlen tanítási felvétel is elegendő a későbbi eredményes felismeréshez.

- Az alapvető kézjelek viszonylag jó felismeréshez szükséges mennyiségben felvett tanítómintáival (azaz néhány jelhez több mint egy mérés tartozik) maga az SVM tanítása a teszthardveren mintegy három percet vesz igénybe.
- A kézjel-felismerés valós idejű, azaz a Kinect adatfolyamának beérkezésével szinkronban (másodpercenként harmincszor) történik.
- A szoftver felismerés közben átlagosan 35%-al terhelte le a teszthardver processzorát. Ez az érték tanítás közben kicsivel kisebb, illetve kézmutatás nélkül 20%-ra esik vissza. A memóriahasználat egyszer sem lépte túl a 250MB-ot.

A következő táblázatokban összefoglaltuk az alapvető kézjelek felismerési hatékonyságát, mely adatok csak tájékoztató jellegűek (a felismerési hatékonyság függ a felhasználó „gyakorlottságától” is).

Kézjelek	BUY	SELL	OK	ONE	TWO	THREE	FOUR	FIVE
Tesztesetek száma [db]	67	74	71	39	30	50	54	57
Helyes felismerés [db]	63	60	62	0	0	1	8	15
Felismerési arány [%]	94	81	87	0	0	2	6	26

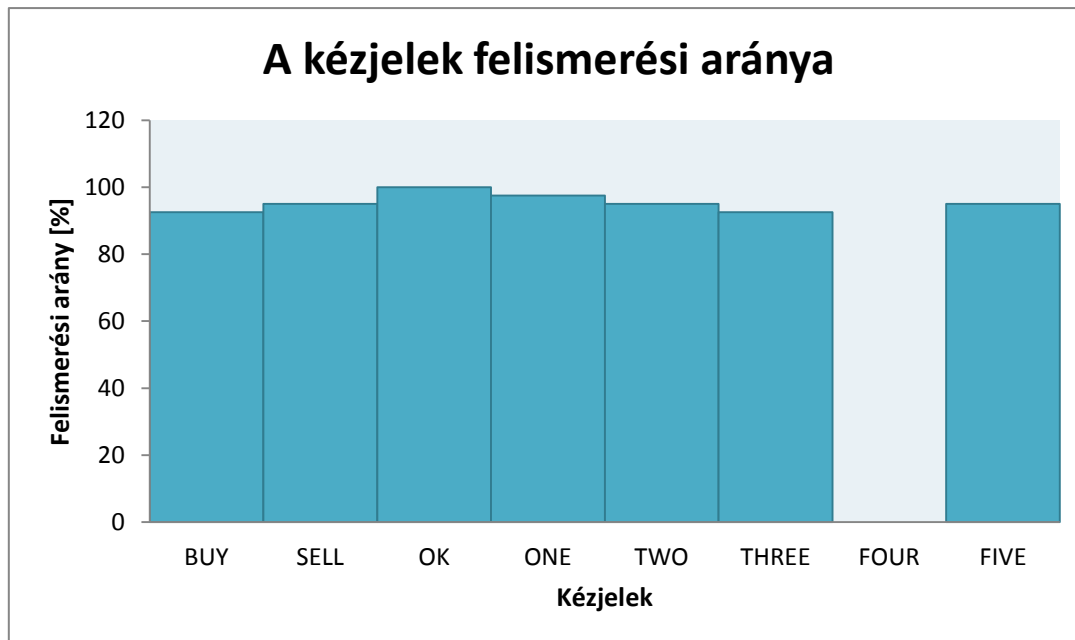
1. táblázat: Felismerési eredmények az AreaDescriptor használata előtt



1. diagram: Az 1. táblázathoz tartozó diagram

Kézjelek	BUY	SELL	OK	ONE	TWO	THREE	FOUR	FIVE
Tesztesetek száma [db]	40	40	40	40	40	40	40	40
Helyes felismerés [db]	37	38	40	39	38	37	0	38
Felismerési arány [%]	92,5	95	100	97,5	95	92,5	0	95

2. táblázat: Felismerési eredmények az AreaDescriptor használatával



2. diagram: A 2. táblázathoz tartozó diagram

Az adatokból is kitűnik, hogy az AreaDescriptor leíró tömb alkalmazásával már használható felismerési hatékonyságot tudtunk elérni.

Az SVM sajátosságaiból adódik, hogy ha egy jelnél „hibázik”, akkor azt a jelet újabb mintákkal tanítva jobban „megtanulja”, közben viszont más jelek felismerése romolhat, ami újabb tanítást igényel. Összességében azonban a megfelelő tanítóminták mennyiségének növelésével a felismerési hatékonyság (célzottan) javítható. Az itt látható eredmények összesen mintegy kétezer mérés (tehát kézzelként 1-2 hat másodperces felvétel) tanítómintaként való felhasználásával születtek.

Továbbfejlesztési lehetőségek

A legfontosabb továbbfejlesztési lehetőség a kézjel-felismerés biztonságának javítása és a felismert kézjelek számának növelése. Ennek kiindulási alapja a tanítóminták nagyarányú bővítése (úgy, hogy a mintákat több ember vegye fel a személyes sajátosságok kiküszöbölése érdekében). Mindezzel elérhető, hogy a felismerési algoritmus gyenge pontjait megtaláljuk.

További fejlesztési lehetőség még, hogy a szoftver bonyolultabb, dinamikus kézjelek, gesztusok felismerésére is képes legyen.

Összegzés

A fejlesztési célokat sikerült elérnünk: elkészült egy kézjelek felismerésére alkalmas, programozói ismeretek nélkül, felhasználói felületről is tanítható, Kinect szenzort használó szoftver.

A szoftver rövid időn belül történő lefejlesztése lehetetlen lett volna a fejlett és szabadon elérhető, nyílt forráskódú Candescent NUI és Encog keretrendszer nélkül. Ezek segítségével vált lehetségessé, hogy nekünk csak a szűken vett feladat megoldására kelljen koncentrálnunk az alacsonyabb szintű problémákkal való foglalkozás helyett.

Irodalomjegyzék

- [1] The Art of Hand Signals, Chicago Mercantile Exchange:
<http://www.oxfordfutures.com/docs/Handsignals.pdf> (2012. október 5.)
- [2] <http://www.microsoft.com/en-us/kinectforwindows/develop/developer-downloads.aspx> (2012. szeptember 29.)
- [3] <http://candescentnui.codeplex.com> (2012. október 20.)
- [4] <http://openni.org/> (2012. szeptember 29.)
- [5] <http://kinecttoolbox.codeplex.com/> (2012. szeptember 29.)
- [6] <http://kinectdtw.codeplex.com/> (2012. szeptember 29.)
- [7] <http://www.heatonresearch.com/encog> (2012. szeptember 29.)
- [8] <http://www.aforgenet.com/framework> (2012. szeptember 29.)
- [9] <http://leenissen.dk/fann/wp/> (2012. szeptember 29.)
- [10] <http://www.openni.org/> (2012. szeptember 29.)
- [11] <http://opencv.org> (2012. szeptember 29.)
- [12] http://en.wikipedia.org/wiki/Dot_product (2012. november 3.)
- [13] <http://www.dtrek.com/svm.htm> (2012. november 3.)
- [14] <http://msdn.microsoft.com> (2012. november 3.)
- [15] <http://research.microsoft.com/pubs/67119/svmtutorial.pdf> (2012. november 3.)