Debreceni Egyetem
Informatikai Kar

Dokumentumok számítógépes feldolgozása

Nagy András

programtervező matematikus

Diplomamunka

Témavezető

Dr. Fazekas Attila

egyetemi docens

Debrecen

2008

TARTALOMJEGYZÉK

I. rész

ELŐSZÓ

Egyetemi éveim során ismerkedtem meg a képfeldolgozással. A kötelező feladatokon túl is foglalkoztam vele a Debreceni Képfeldolgozó Csoport keretében. Karakterfelismerést, bábu- és mozgásdetektálást tanulmányoztam mélyebben.

Karakterfelismerést a Debreceni Egyetem Tehetséggondozó Programjának keretében vizsgáltam. Egy Walsh-transzformáción alapuló nyomtatott szövegek felismerésére képes szoftvert készítettem. A projekt lezárulta után továbbra is foglalkoztatott a téma. Kerestem a lehetőséget, hogyan fejlődhetek tovább a témakörben.

A diplomamunkám választott témája lehetővé teszi, hogy a tanulmányaim során megszerzett ismereteket hasznosíthassam. Azért választottam a címben szereplő témát, mert nem csak képfeldogozási, hanem más tárgyak alkalmazása során szerzett gyakorlati tudásom bemutatására is alkalmasnak ítéltem meg.

Napjainkban megkérdőjelezhetetlen szerepe van a számítógépeknek, amelyek csak digitális adatokon képesek műveleteket elvégezni. Az emberiség történelme során felhalmozódott emlékek túlnyomórészt írásos és nyomtatott anyagok. Megjelent az igény ezen információk számítógéppel történő feldolgozására. A való világban analóg képekkel találkozunk, ezeket kell digitalizálnunk egy céleszközzel, ami lehet egy fényképezőgép vagy egy szkenner. A digitális képből információt kinyerni nem egyszerű, de már számos célszoftver született erre a feladatra. Ezen szoftverek többségének karakterfelismerésre specializálódott. Nehéz olyat találni viszont, amelyik kitöltött űrlapok feldolgozását végzi.

A diplomamunka keretében a célom egy olyan szoftver elkészítése volt, amely űrlapok kitöltésének automatikus feldolgozását végzi, majd az eredményt elemzi és megfelelő kimeneti formátumban továbbítja. Ezeket ismert algoritmusok és technológiák segítségével igyekeztem megvalósítani, ezért döntöttem a .NET keretrendszer és a C# nyelv mellett. Törekedtem arra, hogy a feldolgozás lépései a lehető legáltalánosabban történjenek. Nem építettem be a kódba a tesztelt űrlapok feldolgozásához szükséges speciális ismereteket. Az űrlapok tulajdonságait külső, XML struktúrájú konfigurációs fájlokban tárolom. A tervem az volt, hogy egy a szoftverről és a háttérben lévő algoritmusokról is megfelelő ismeretekkel rendelkező személy képes legyen a konfigurációs fájlokat tetszőleges űrlap feldolgozásához elkészíteni.

Ezúton szeretnék köszönetet mondani témavezetőmnek, Fazekas Attilának, aki észrevételeivel és tanácsaival segítette a munkámat.

Kiemelt köszönettel tartozom Szüleimnek, akik tanulmányaim során végig mellettem álltak és mindenben támogattak.

Debrecen, 2008. április 20.

Szerző

II. rész

BEVEZETÉS

Az ókori kézírásos könyvek óta, a középkori nyomtatott könyveken át a mai napig a könyv a legelterjedtebb információhordozó. A papírlapokon szereplő jelek tartalmazzák számunkra az információt. A számítógépek megjelenésével igény jelentkezett ezen információk feldolgozására.

A karakterfelismerés kézzel vagy géppel írott, illetve nyomtatott szöveget tartalmazó képek átalakítása szerkeszthető formátumra. Az így átalakított szöveg tetszőlegesen feldolgozható számítógépes programokkal. Az OCR[1] korábban tükrökön és lencséken alapuló optikai technikákon alapult, ma már a digitális képfeldolgozás az alapja. A következő bekezdésekben bemutatom az OCR történelmét.

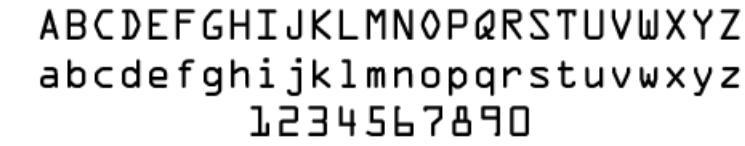
Az első karakterfelismerésre vonatkozó szabadalom az osztrák Gustav Tauschek nevéhez fűződik 1929-ből. Tauschek számtalan szabadalmat nyújtott be a korabeli számológépek és lyukkártyafeldolgozók témakörében. Készített egy olyan mechanikus gépet, amely sablonokat illesztett az egyes karakterekre és erős megvilágításnak tette ki őket. Elhelyezett egy fénydetektort oly módon, hogy sablon és a karakter pontos illeszkedése esetén nem jutott el fény a detektorba.

1950-ben David H. Shephard nyomtatott szövegek gépi formára alakításán kezdett dolgozni az Amerikai Nemzetbiztonsági Hivatalnak[2]. Kollégájával, Harvey Cook Jr.-ral egy év alatt elkészítették Gismo-t. Gismo egy olyan gép volt, amely egy szabványos írógép 23 betűjét képes volt felismerni. Újabb egy év elteltével mind a 26 karaktert képes volt felismerni.

1955-ben a Shephard által alapított IMR[3] készítette az első olyan gépet, amely bankkártyák lenyomatain szereplő kártyaszámok felismerését végezte.

1965 körül a Readers Digest és az RCA közösen egy OCR dokumentumolvasó építésébe kezdett, amely hirdetési kuponok sorozatszámának digitalizálását végezte. Az olvasó percenként 1500 dokumentumot is képes volt feldolgozni. Speciális, OCR-A betűtípust használtak a nyomtatványokon.

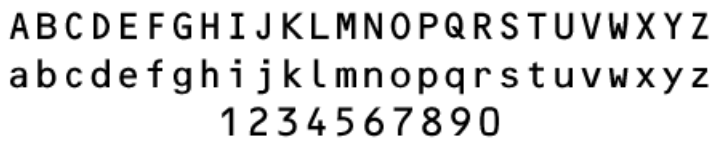
Az OCR-A betűtípust az Amerikai Szabványügyi Hivatal[4] számítógépes feldolgozásra fejlesztette ki 1968-ban. Lekerekített nyolcszögletű alakzatokból áll, ahogy ez 0.1 ábrán is látható. A betűtípus egyszerű kinézete miatt géppel könnyen feldolgozható, de emberi szemmel nehezen olvasható. Az ANSI szerzői jogai miatt kevéssbé elterjedt.



0.1. ábra. Az OCR-A betűtípus karakterkészlete

[1] Optical Character Recognition, optikai karakterfelismerés
[2] National Security Agency, NSA
[3] Intelligent Machines Research Corporation
[4] American National Standards Institute, ANSI

Az OCR-B betűtípust az európai szabványokhoz igazodva alkotta meg Adrian Frutiger szintén 1968-ban. Ez a betűtípus az optikai felismerés határait feszegeti, de emberi szemmel már könnyebben olvasható. Az eredeti OCR-B betűtípus kerek vonalvégekből, míg a második verziója már egyenes vonalvégekből állt. A karakterkészlete megtekinthető a 0.2 ábrán.



0.2. ábra. Az OCR-B betűtípus karakterkészlete

Az első igazán megbízható rendszer - amely Jacob Rabinow technológiáján alapul - 1965-ben került bevezetésre. Speciális, ultraibolya fény alatt látható tintával jelölt címek szerinti rendezésre használta az Egyesült Államok postája[5]. Ezt később átvette a brit és a kanadai posta is.

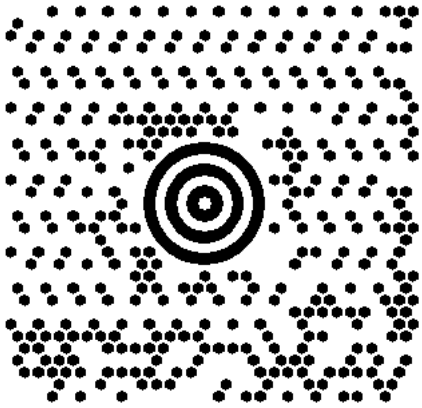
1974-ben Ray Kurzweil egy olyan vakoknak szánt olvasógép megalkotását kezdte el, amely egy számítógéppel felismertett szöveget olvasott fel. Ehhez szükség volt a Kurzweil Computer Products által kifejlesztett első, több betűtípust is felismerő karakterfelismerő rendszerre és egy szöveget beszéddé alakító beszédszintetizátorra. Az olvasógép[6] 1976-ban készült el, ami csak 64000 bájt memóriával rendelkezett. Az első olvasógépeket iskolákban és könyvtárakban helyezték el.

Nagyon pontos eredményt ad a mágneses tinta alapú karakterfelismerés. Az MICR[7] karakterek akkor is olvashatók maradnak, ha felülírták vagy felülpecsételték őket. A felismerés azon alapul hogy az egyes betűformák alakjai különböző hullámalakot jelenítenek meg az olvasó fejen. Ezen hullámalakok egyedisége biztosítja az egymástól jól elkülöníthetőséget és a pontos felismerési hatékonyságot.

Optikai jelfelismerés[8] során a lap adott területének tükröződése a feldolgozás alapja. Egy fénysugárral megvilágítva a szkenner képes megkülönböztetni a jelölt és a jelöletlen területeket. A karakterfelismeréstől (OCR) az különbözteti meg alapvetően, hogy nincs szükség felismerő egységre. A pontos beolvasáshoz nagy kontrasztú képre és könnyen felismerhető alakzatokra van szükség. A gyakorlatban feleletválasztós tesztek, lottószelvények, vonalkódok és kétdimenziós vonalkódok felismerésére használható. A 0.3 ábrán a Maxicode rendszerrel készített vonalkód látható.

Napjainkban a gépírásos szöveg felismerése már nem okoz problémát az OCR-programoknak. A digitális táblák, PDA-k és Tablet PC-k megjelenésével egyre nagyobb igény fogalmazódott meg a kézírásos szövegek felismerésére. Új

[5] US Postal Service
[6] Kurzweil Reading Machine
[7] Magnetic Ink Character Recognition
[8] OMR, Optical Mark Recognition



0.3. ábra. A UPS által használt Maxicode rendszerrel készített 2D-s vonalkód, amely a *Maxicode* adatot tartalmazza

módszerek kidolgozására van szükség, mivel a korábban alkalmazott algoritmusok már nem alkalmazhatók folyóírás felismerésére. Különböző személyek írásmódjainak sem szabad problémát okoznia, ahogy az ember is képes korábban nem ismert kézírást elolvasni. Egyes mai PC-s és PDA-s operációs rendszerek beépítve tartalmaznak kézírásfelismerést, mint például a Windows XP Tablet Edition, a Windows Vista és a Windows CE is.

A karakterfelismerőket osztályozhatjuk a következő szempontok szerint:

- OCR pontosság
- Oldal felépítésének helyreállítási pontossága
- Többfajta felismerési technológia
- Sebesség
- Nyelvek támogatása
- Felhasználói interfész

Mindenképpen meg kell említeni a Recognitát, mint kora egyik legsikeresebb magyar szoftverét. A magyar Recognita részvénytársaságot 1989-ben alapították külföldi társtulajdonossal. A 90-es évek elején Németországban és az Egyesült Államokban is terjeszkedni kezdett. Ebben az időben a cég fő terméke - a RecognitaPlus - Európában piacvezető, Európán kívül is az élmezőnyben volt. A szkennergyártókkal való szövetségkötésekben viszont a cég rendre alulmaradt, a független teszteken mégis a legjobb helyezést érte el többször is. 1996-ban az egyik legnagyobb versenytárs, a Caere vette meg a Recognitát. 2000-ben a Caere-t és vele a Recognitát is felvásárolta a Xerox leányvállalata, a ScanSoft. Az OmniPage karakterfelismerő program a Recognita, a ScanSoft és a Caere

motorjainak ötvözéséből állt elő. A konkrét feladatra, mindig az aktuálisan legjobb motort választja ki. A cég nyitni kezdett a nyelvi technológiák és a beszédfelismerés irányába is, így a vállalat nevét Nuance-Recognitára változtatták.

A ma legelterjedtebb OCR programokat hasonlítom össze néhány jellemzőjük alapján, elsősorban a társaiktól való eltérésüket emelem ki.

Az Adobe Acrobat 8 Professional beépített helyesírás-ellenőrzővel rendelkezik. Képes kötegelésre és a szöveg szerkesztésére, de nem képes foltmentesítésre és nem is tanítható.

Az OmniPage 16 Professional képes többszálú feldolgozára. Jól automatizálható könyvtár, vonalkód, postafiók figyelésére és ismétlődő műveletek végrehajtására.

A ReadIris 11 Corporate képes névjegykártya- és kézírásfelismerésre, de nem integrálható más alkalmazásokba.

Az ABBYY FineReader 8 Corporate beépített helyesírás-ellenőrzővel és szövegszerkesztővel rendelkezik. 179 nyelvet támogat, de nem tanítható.

Egyiküknek sem okoz problémát a varázslók használata, vonalkódok felismerés, webcímek felismerése, a különböző bemeneti illetve kimeneti formátumok és a Windows Vista alatti működés sem.

Elterjedsége és egyszerűsége miatt az XML[9] formátumot használom az feldolgozáshoz szükséges objektumok fájlban történő tárolására. A .NET keretrendszer összetett XML-kezelő függvénykönyvtárral rendelkezik. A feldolgozáshoz szükséges adatstruktúrák rendelkeznek XML-ből felépülő konstruktorral és XML-be kiíró metódussal is. Az adatstruktúrákon belül az adattagokra való hivatkozást XPath útvonalakkal végzem. A felismerés eredményeképpen kapott XML fájl struktúráltsága ellenére emberi szemmel nem jól átlátható. Ezért egy XSLT[10] stíluslap segítségével HTML[11] fájl készíthető belőle, ami elterjedtsége révén bármely böngészővel megtekinthető. A szemantikai elemzés eredménye is egy XML fájl, amiből szintén készíthetünk HTML-t egy megfelelő XSLT fájl segítségével.

[9] Extensible Markup Language
[10] Extensible Stylesheet Language Transformations
[11] HyperText Markup Language

1. A FELDOLGOZÁS ÁLTALÁNOS FOLYAMATA

Az űrlap feldolgozási folyamata számos lépésből áll. Csak annyit követelünk meg tőle, hogy szerepeljenek rajta vastag függőleges és vízszintes vonalak, amelyek a forgatást és az illesztést segítik. A következő felsorolásban az űrlapok feldolgozása esetén használt műveletek találhatók.

- A beolvasott kép előkészítése során különböző képfeldolgozási műveleteket kell rajta végrehajtanunk a minél pontosabb konfiguráció és felismerés érdekében. Az űrlap mezőinek helye pontosan meghatározott, ezért az űrlap beolvasása során előforduló elforgatást és eltolást korrigálnunk kell.
  - A kép beolvasása történhet fájlból illetve szkennerről.
  - Megkeressük a kép hasznos információt tartalmazó középső részét a későbbi gyorsabb feldolgozás érdekében.
  - A bináris képeken egyszerűbben és gyorsabban végezhetünk alapvető képműveleteket, ezért szürkeskálássá, majd binárissá konvertáljuk a képet.
  - Lokális maximum szűrést végzünk a zaj és a vékony vonalak eltüntetése érdekében.
  - Az eredeti kép vastag vonalait tartalmazó képet kaptunk, amelyre megpróbálunk egy egyenest illeszteni. A talált egyenes feltételezhetően eredetileg függőleges vagy vízszintes volt.
  - A talált egyenes elforgatási szögéből kiszámolható, hogy hány fokkal és milyen irányba kell a képet elforgatnunk az eredeti függőleges és vízszintes vonalak helyreállásához.
  - Végezzük el a forgatást az eredeti színes képen.
  - Hajtsuk végre ismét a szürkeskálássá, majd a binárissá konvertálást és a lokális maximum szűrést, így ismét csak a vastag vonalak szerepelnek, de ezek már pontosan függőlegesek és vízszintesek.
  - A konfigurációkor megadtuk ezen vastag vonalak egymáshoz viszonyított távolságarányait, ezek segítségével meghatározhatjuk az általánosan megadott vonalak konkrét helyzetét a képen.
- Az előkészített képen a konkrét vonalak és a konfigurációban szereplő adatok segítségével képesek vagyunk pontosan tájékozódni. Az űrlapon szereplő mezőket egyesével felismerjük.

  - A mezőt az oldalainak lokalizálása után kivágjuk a képből.
  - A mező szélének előre meghatározott részét az illesztési pontatlanságok miatt elhagyhatjuk.
  - Amennyiben olyan jeleket kell felismernünk, ahol különösen fontos a jelek alakja (pl. számjegyek), leszűkíthetjük a mező széleit pontosan a benne szereplő jelre.
  - A mezőt Walsh-transzformáció segítségével leképezzük vektorra.
  - A leképezett vektorhoz közel eső vektort keresünk a betanított vektorok között.

- Már nem tartozik a felismeréshez, de szorosan követi az elemzés. A felismerés eredményét különböző szabályok alapján elemezhetjük.
  - Szintaktikai elemzés esetén nincs ismeretünk arról, hogy létezne az űrlapnak helyes kitöltése. Csak annyit tudunk, hogy a lehetséges kitöltések megfelelnek bizonyos szabályoknak. Ezek az összetett szabályok elsőrendű logikán alapuló, rekurzív szabályokból épülnek fel. Például az ötöslottó szelvény feladásakor ellenőrzésre kerül, hogy pontosan 5 mezőt jelöltünk-e be a 90-es mezőcsoportokban.
  - Szemantikai elemzéssel szigorúbb ellenőrzést végezhetünk, mert ismert az űrlap helyes kitöltése. Az űrlapot kérdésekre osztjuk fel, amelyekhez pontosan egy helyes válasz tartozik. Amennyiben több válasz is kielégíti a kérdést, ezeket *logikai vagy* művelettel foghatjuk össze egy válasszá. A lehetséges válaszok itt is elsőrendű logikán alapuló, rekurzív szabályokból épülnek fel. Tesztkérdésekből álló dolgozat javítása a megfelelő példa ide.

- A felismerés és az elemzés eredménye is egy XML fájl, amelyet emberi szemmel áttekinteni nem egyszerű feladat. XSLT stíluslappal az XML-t tetszőlegesen formázott HTML fájllá transzformálhatjuk, amit már jól ismerünk.

2. XML HASZNÁLATA

Az XML általános célú leírónyelv. Egy XML-dokumentum tartalma különböző elemekre bontható. Minden elem a dokumentum egy logikai része, amely további elemeket is tartalmazhat. Amelyik elem tartalmazza az összes többi elemet, azt gyökérelemnek nevezzük. Az elemek által felépített hierarchikus szerkezetet dokumentumfának nevezzük. Minden elemnek van neve, továbbá lehet egy vagy több attribútuma.

Az XPath nyelv szorosan kötődik az XML-hez, a dokumentumfán belüli pozícionálásra szolgáló nyelv. A fa részeit csomópontoknak nevezi. Egy XPath kifejezés különböző kritériumoknak megfelelő csomópontokat jelölhet ki.

Mivel többféle XML dokumentum-típus létezik, ezért szükség van egy eszközre, amely a típusok közti transzformációra szolgál. Az XSLT egy olyan XML alapú nyelv, amely XML-dokumentumok más XML-dokumentumokba vagy emberi szemmel is átlátható fájlformátumba történő transzformációra használható. A transzformációhoz a forrás XML fájl mellett szükség van egy XSLT stíluslapra is. Magát a transzformációt egy XSLT feldolgozó végzi el, ami előállítja a kimeneti fájlt.

III. rész

FELDOLGOZÁS

3. KÉP ELŐKÉSZÍTÉSE

A szkennelés során előálló képen különböző előkészítő műveleteketet kell a felismerés előtt végrehajtani. Ha a szelvényünk mérete a teljes képhez viszonyítva kicsi, akkor teljesen felesleges a nagy képpel tovább dolgozni, elengendő a kivágott hasznos rész is. A szkennelt szelvény oldalai - tapasztalatom alapján - sosem párhuzamosak a képernyő széleivel. Ezért a precíz felismeréshez szükséges, hogy a szelvényt pontosan az eredeti pozíciójába forgassuk vissza, a kezdetben vízszintes vonalak újra vízszintesek legyenek. A forgatási szög megállapításához egyenest illesztünk a színes képből készített bináris képre, majd a megállapított meredekség ismeretében visszaforgatjuk a képet az eredeti pozíciójába. Végül a stuktúra konfigurálása során megadott illesztővonalak és mezők helyzetére vonatkozó információk alapján meghatározzuk a mezők helyét.

3.1. Definíciók

Vegye fel a Bi halmaz a bináris színkomponens által ábrázolható értékeket.

$$Bi = \{0, 1\}$$

Vegye fel a Gr halmaz a szürke színkomponens által ábrázolható értékeket.

$$Gr = \{0, 1, ..., 255\}$$

Vegye fel az R, a G és a B halmaz a vörös, a zöld és a kék színkomponensek által ábrázolható értékeket.

$$R = \{0, 1, ..., 255\}$$

$$G = \{0, 1, ..., 255\}$$

$$B = \{0, 1, ..., 255\}$$

Jelölje a C halmaz a színes képpontok által felvehető értékeket.

$$C = R \times G \times B$$

A $\mathbb{Z}^n$ halmazt n-dimenziós digitális síknak, elemeit pontoknak nevezzük.

Az n-dimenziós digitális sík nemüres részhalmazát n-dimenziós digitális halmaznak nevezzük.

Az n-dimenziós X digitális halmaz felett értelmezett $f : X \to \{0, 1, ..., m-1\}$ ($m \in \mathbb{N}$) függvényt m-szintű digitális képnek nevezzük.

Az $f : X \to Bi$ függvényt bináris képnek nevezzük.

Az $f : X \to Gr$ függvényt szürkeskálás képnek nevezzük.

Az $f : X \to C$ függvényt színes képnek nevezzük.

3.2. Hasznos rész kivágása

A hasznos rész kivágásának célja a szkennelt képről a felesleges tartalmak eltávolítása. A továbbiakban feldolgozandó kép mérete így csökken, a képműveletek gyorsabban végrehajtódnak. A cél a teljes képen belüli lehető legkisebb olyan téglalap meghatározása, amelyik tartalmazza a szkennelt szelvényt.

Tapasztalatom alapján a szkennelt kép keretéhez közel sötét képpontok jelennek meg. Rögzítsük azt a szkennelt kép keretétől való távolságot, amelybe az összes ilyen sötét képpont beleesik. Vizsgálataim szerint 50 pixel már elegendő. Határozzuk meg az a téglalapot, amely oldalai a teljes kép oldalaitól az előző lépésben rögzített távolságra vannak, így a sötét képpontok ezen téglalapon kívül maradnak.

Határozzuk meg azt a lépésközt, amivel az egyes oldalakat befelé fogjuk tolni. Kis lépésköz esetén a szűkítés pontosabb lesz, de tovább tart. Nagy lépésköz esetén hamar végbemegy a szűkítés, de csak durván közelíti meg az ideális eredményt. Gyakorlatban 20 pixel már megfelelőnek tekinthető.

Rögzítsük azt a világosságértéket, amelyiknél sötétebb képpontokat, már a szelvény részének tekintjük. A 150-es érték nálam megfelelően működött. Felhívom a figyelmet arra, hogy ez *nem* a két szintre vágás korlátja.

A csökkentendő téglalap minden oldalára hajtsuk végig a következő műveleteket: Vizsgáljuk meg az oldal minden képpontját. Ha minden képpont világosabb az előző lépésben rögzített világosságértéknél, akkor toljuk el az oldalt a vele szemközti oldal irányába az előzőleg rögzített lépésközzel és hajtsuk végre az ellenőrzést újra. Amennyiben találunk a rögzítettnél sötétebb képpontot, akkor a vizsgált oldalnak már van közös pontja a szelvénnyel, tehát túl közel van, így az előző lépésben vizsgált vonal lesz a végleges. Ha az első alkalommal vizsgált oldaltól kellene visszalépnünk, akkor a csökkentett kép oldala az eredeti oldal marad, mert nem tudtunk csökkenteni.

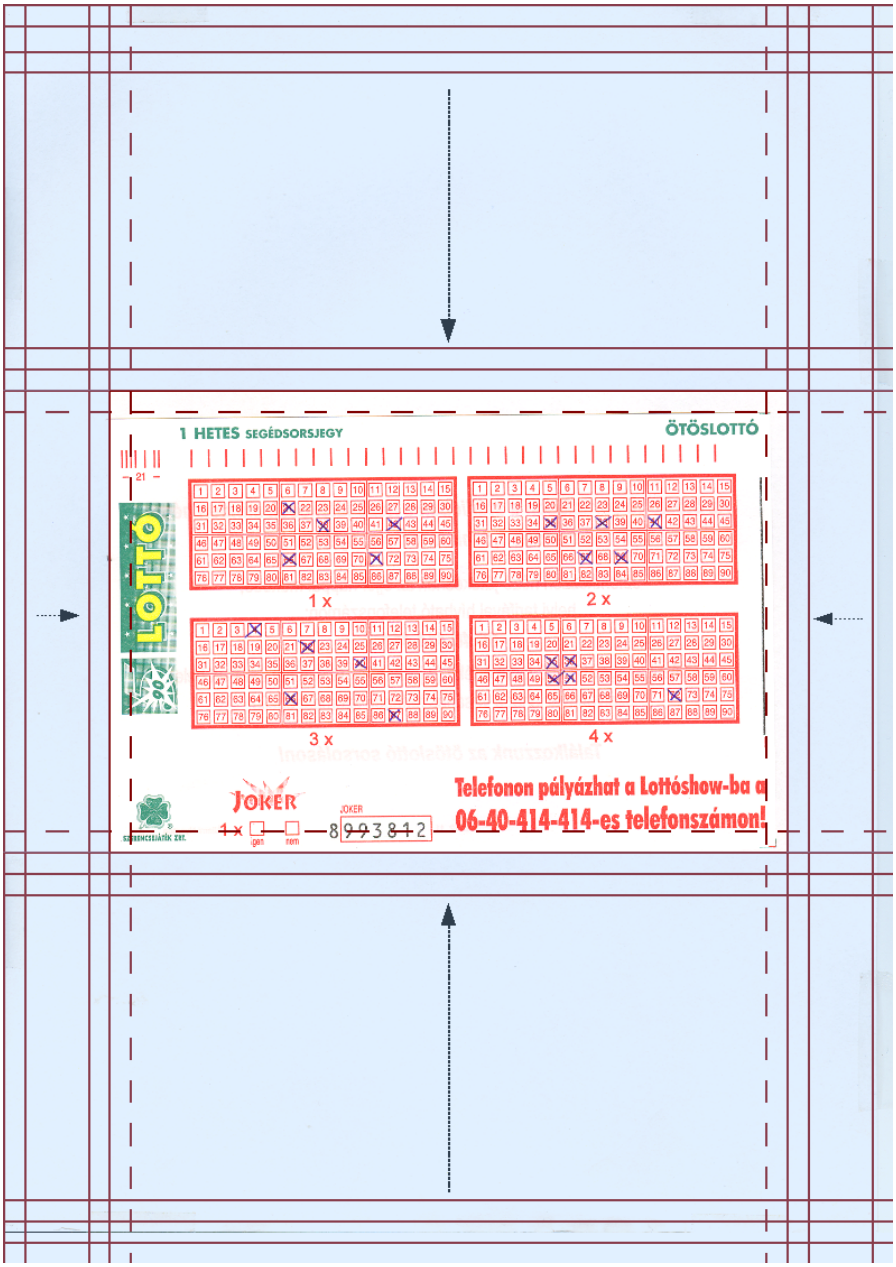
A szűkítés lépéseire egy példa a 3.1 ábrán található meg. Az eredeti képből levágott terület világoskékkel, a szelvénnyel közös pontokat tartalmazó vonalak szaggatottal vannak jelölve.

3.3. Szürkeskálássá konvertálás

Színes, RGB színcsatornákkal megadott képet kaptunk a szkennelés során, a hasznos rész kivágása során ez a tulajdonsága nem változik meg. A karakterfelismeréshez fekete-fehér képre van szükség, ehhez első lépésben szürkeskálássá alakítjuk a színes képet.

Feldolgozzuk a színes kép minden képpontját a következő módon: Kiszámítjuk a színes képpont R, G és B komponenseihez tartozó világosságértékeinek számtani közepének egészrészét. Ez lesz a szürkeskálás kép azonos pozíciójában található képpontjának világosságértéke.

Színes képre a 3.2 ábrán, szürkeskálás képre a 3.3 ábrán találhatunk példát.



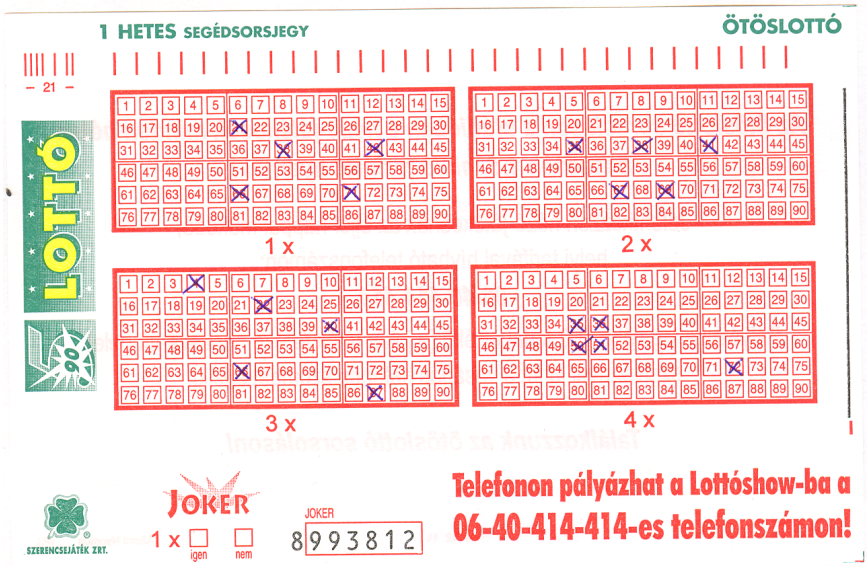
3.1. ábra. A téglalap oldalainak eltolása

A *szurkeskalassa_alakitas* függvény minden egyes színes képpontot szürke képponttá alakít át. Az f függvény színes képet jelöl.

$$szurkeskalassa\_alakitas : X \to Gr$$

Legyen $f(x) = c$, ahol $c = (r, g, b)$, $c \in C = R \times G \times B$, $r \in R$, $g \in G$, $b \in B$, $x \in X$.

$$szurkeskalassa\_alakitas(x) = \left[\frac{r+g+b}{3}\right]$$

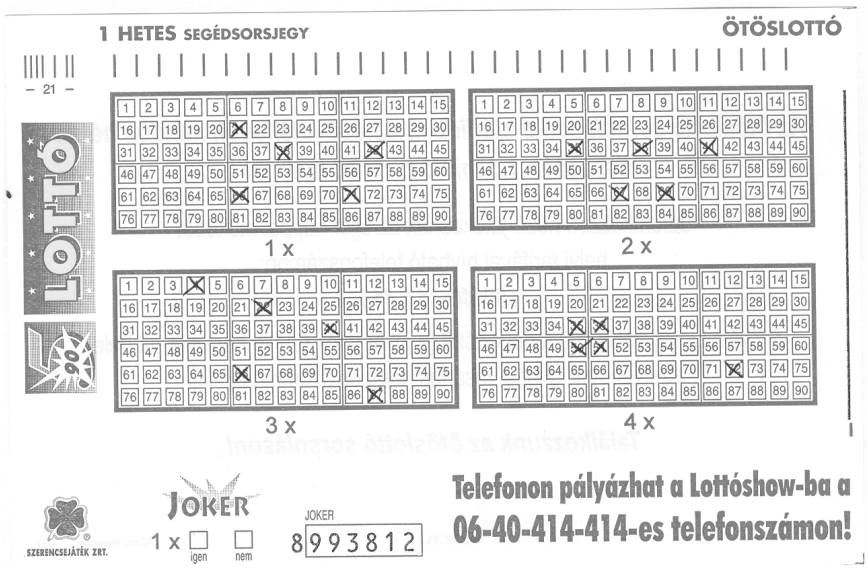


3.2. ábra. Színes kép

3.4. Binárissá alakítás

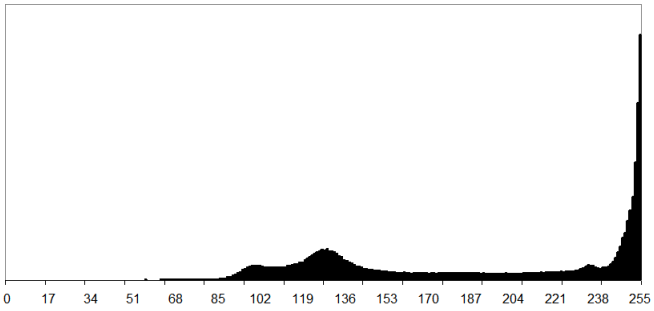
A binárissá alakításhoz, másnéven a két szintre vágáshoz meg kell határoznunk egy küszöböt, amelynél sötétebb képpontok feketévé, míg a világosabbak fehérré válnak a bináris képen.

Első lépésben készítsük el a szürkeskálás kép hisztogramját. A hisztogram egy területmérő függvény, amely az egyes világosságértékekhez hozzárendeli, hogy hány darab képpont szerepel belőle a képen. A 3.3 ábra hisztogramjának grafikonja található meg a 3.4 ábrán. Jól látható, hogy a képpontok egy világosabb és egy sötétebb képpont körül csoportosulnak, a két *hegy* között kevesebb képpont szerepel. A bal oldali hegycsúcs megfelel a toll és a nyomtatott szöveg képpontjainak, míg a jobb oldali a lap fehér része. A feladatunk ezen két csoport különválasztása. Többféle algoritmust is próbáltam implementálni



3.3. ábra. Szürkeskálás kép

a küszöb automatikus meghatározására, de különböző űrlapokon tesztelve már nem adtak pontos eredményt. Mivel egy konkrét űrlaptípus szkennelése esetén a fényviszonyok állandóak, ezért a konfigurációkori küszöbmeghatározás mellett döntöttem. Ezen érték meghatározását egy csúszka segítségével tehetjük meg. A 3.4 ábrán szereplő hisztogramnál a küszöböt 200-nak választottam.



3.4. ábra. Szürkeskálás kép hisztogramja

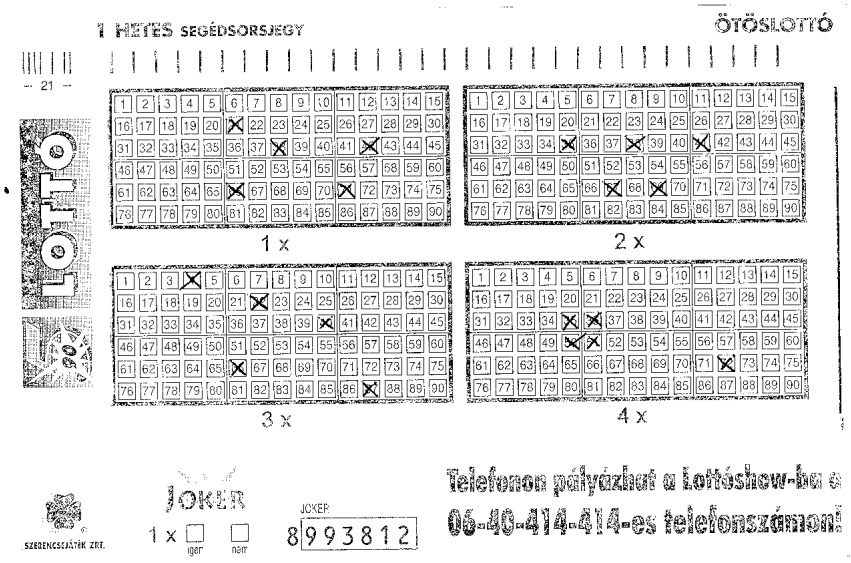
Két szintre vágásra nincs tökéletesen meghatározható, minden esetben egyformán viselkedő univerzális küszöb. A bináris képpel való további terveinktől függ, hogy sötétebb-világosabb vagy zajosabb-kevésbé zajos képre van szüksé-

günk. Fontos viszont, hogy adott képtípusra következetesen ugyanazt a küszöböt alkalmazzuk. A 3.5, a 3.6 és a 3.7 ábrákon különböző küszöbök melletti két szintre vágott képe található a 3.3 ábrán látható szürkeskálás képnek.

A *binarissa_alakitas* függvény minden egyes szürkeskálás képpontot egy küszöb ismeretében bináris képponttá alakít át. Az f függvény szürkeskálás képet jelöl.

$$binarissa\_alakitas : X \times \mathbb{N} \rightarrow Bi$$

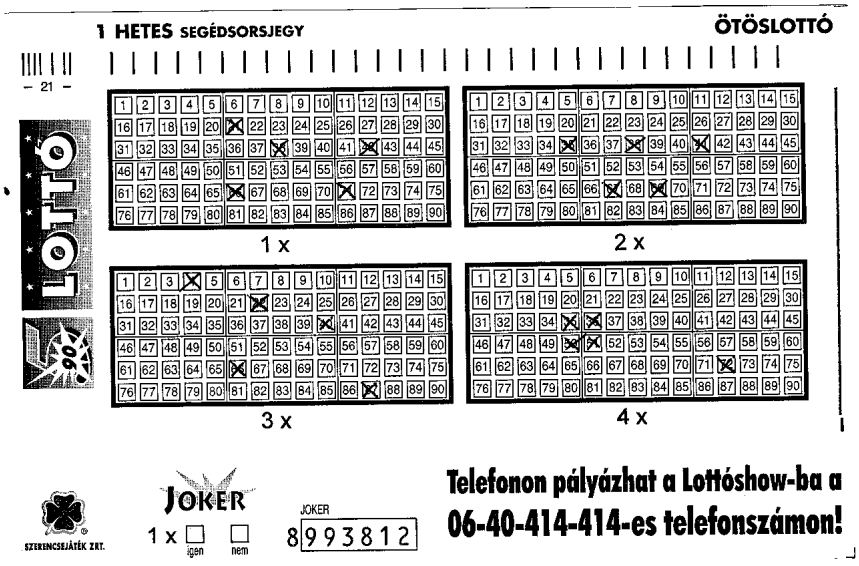
$$binarissa\_alakitas(x,k) = \begin{cases} 0, & \text{ha } f(x) < k \\ 1, & \text{ha } f(x) \geq k \end{cases}$$



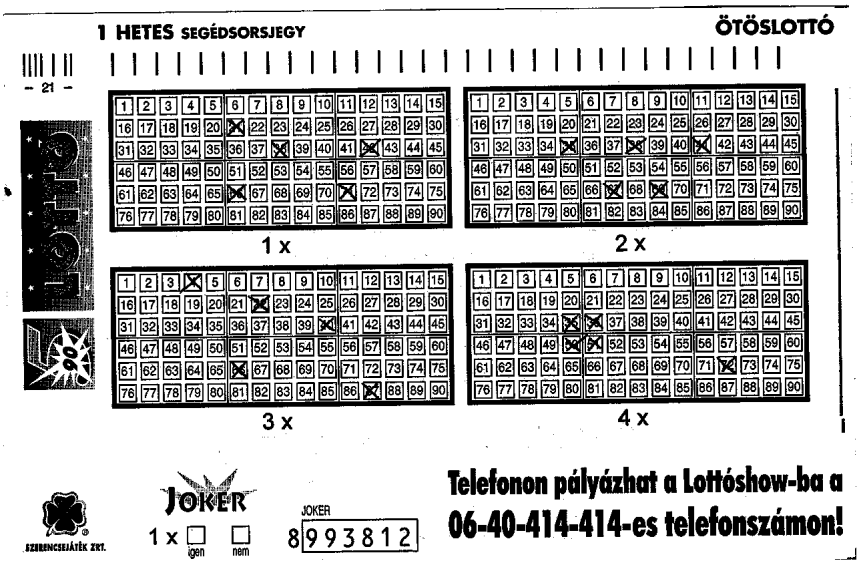
3.5. ábra. Két szintre vágás 130-as korláttal

3.5. Lokális maximum szűrés

A kapott bináris képen a célunk egy egyenes megtalálása. Tapasztalatom szerint a kép ehhez még túl részletes, túl sok felesleges információt tartalmaz, elegendő a vastagabb vonalak helyzete, ezért lokális maximum szűrést alkalmazunk. Minden egyes képpontnak megvizsgáljuk a 8-szomszédait. A képpont szűrt értéke a 8-szomszédok világosságértékei közül a legnagyobb értéke lesz. Ha nem létezik mind a 8 darab 8-szomszéd, akkor csak a létezőket vizsgáljuk. A szűrt képen is csak fekete és fehér (0 és 255 világosságértékű) képpontok szerepelhetnek. A szűrés hatására a vonalak elvékonyodnak. Ez annak köszönhető, hogy a szűrt képen csak ott találunk fekete képpontot, ahol az eredeti képen csupa fekete



3.6. ábra. Két szintre vágás 200-as korláttal



3.7. ábra. Két szintre vágás 240-es korláttal

képpont volt körülötte. Ha csak egy fehér is van a 8-szomszédai között, a maximumszámítás miatt a szűrés eredménye fehér képpont lesz.

A 3.8 ábrán lokális maximum szűrést végrehajtva a 3.9 ábrát kapjuk.

A $8szomszed$ függvény minden 2-dimenziós, koordinátáival adott képpontpár esetén megállapítja, hogy 8-szomszédok-e:

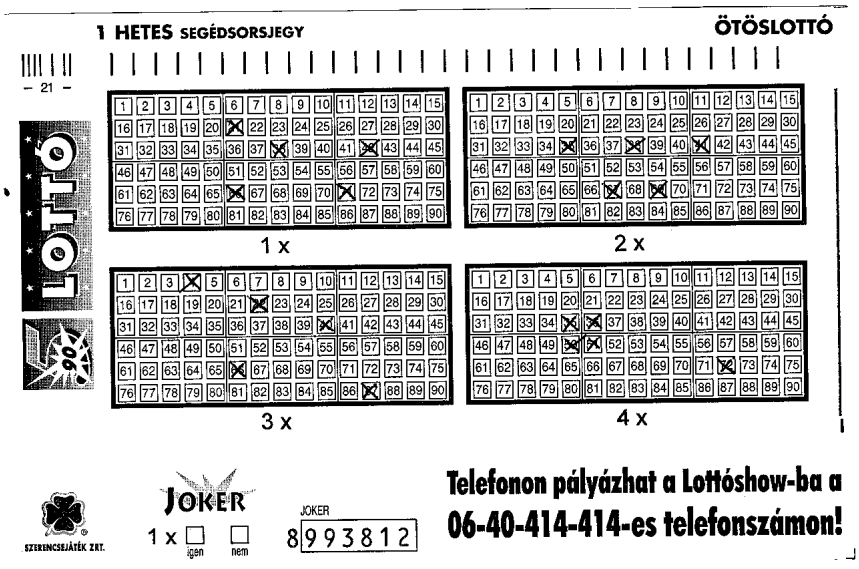
$$8szomszed : X \times X \to \{igaz, hamis\}\,.$$

$$8szomszed(P_{(x_0,y_0)}, Q_{(x_1,y_1)}) = \begin{cases} igaz, & \text{ha } |x_0 - x_1| \leq 1 \text{ és } |y_0 - y_1| \leq 1, \\ hamis, & \text{egyébként.} \end{cases}$$

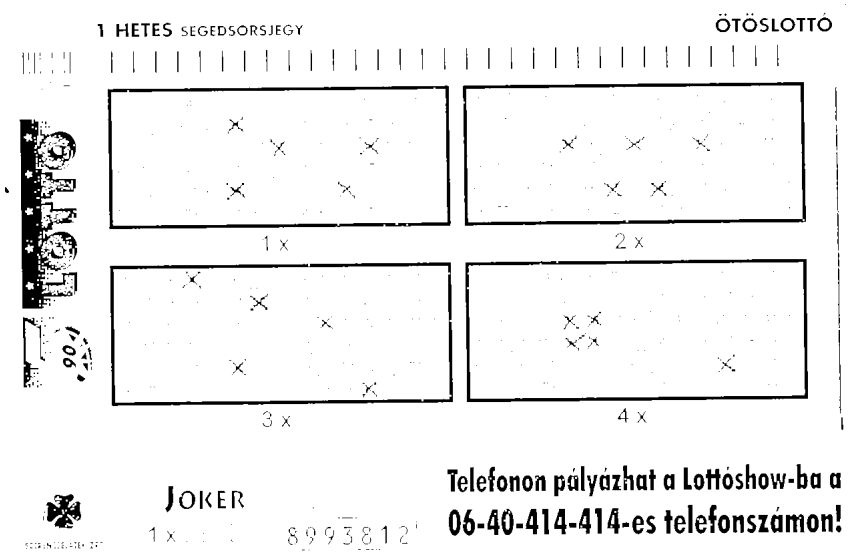
A $lokalis\_maximum\_szurt$ függvény minden képponthoz hozzárendeli a lokális maximum szűrés utáni bináris értékét. Az f függvény bináris képet jelöl.

$$lokalis\_maximum\_szurt : X \to Bi$$

$$lokalis\_maximum\_szurt(P) = \begin{cases} 1, & \text{ha } f(Q) = 1 | \forall Q \in X, \\ & \text{ahol } 8szomszed(P,Q) = igaz, \\ 0, & \text{egyébként.} \end{cases}$$



3.8. ábra. Bináris kép



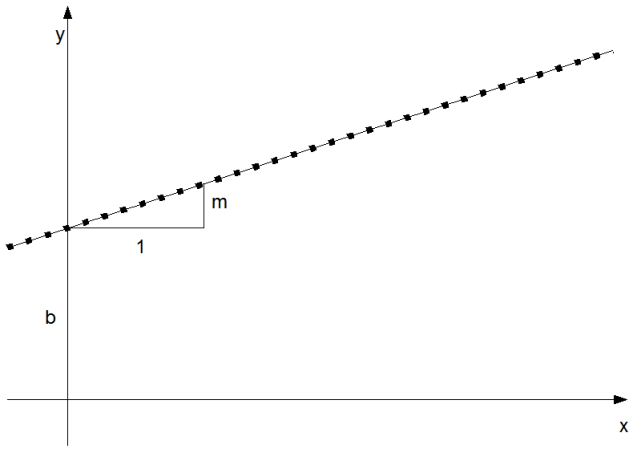
3.9. ábra. Lokális maximumra szűrt bináris kép

3.6. Vonal illesztése

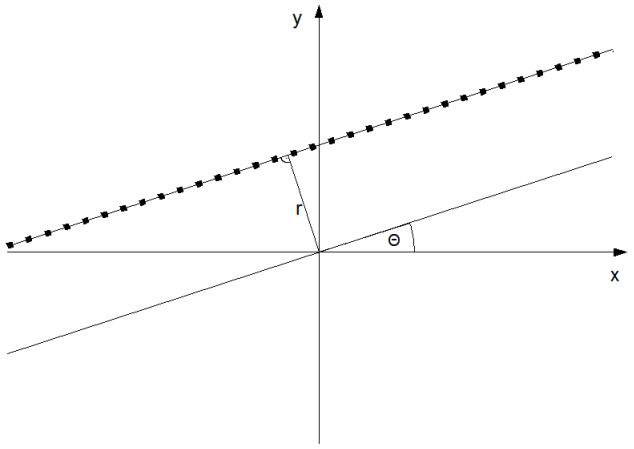
A szűrt képen Hough-transzformáció segítségével próbálom a legjobban illeszkedő egyenest megtalálni.

A képtérben egy egyenest $y = mx + b$ alakban írhatunk le Descartes-féle koordinátarendszerben, ahogyan az a 3.10 ábrán látható. Az egyenes pontjai (x, y) koordinátákkal rendelkeznek. A Hough-transzformáció alapötlete az, hogy a meredekségre (m) és az y tengellyel való metszéstávolságra (b), mint a paramétertér egy (m, b) koordinátájú pontjára tekintünk. Az (m, b) koordinátákkal a függőleges és függőleges közeli vonalaknál adódnak problémák, mert a meredekség és a metszéstávolság is a végtelenbe tart, ezen értékek tárolása pedig számítástechnikai problémákat vet fel.

Helyesebb áttérni polárkoordinátákra θ és r paraméterek mellett, ahol θ az elforgatás szöge és r az egyenes origótól való távolsága. A polárkoordináták szemléltetése a 3.11 ábrán történik. Az egyenes egyenlete a következő módon írható le: $r = x * \cos(\theta) + y * \sin(\theta)$, ahol az egyenes pontjai az (x, y) koordinátákkal adhatóak meg. Amennyiben ismerünk a képen egy (x_0, y_0) koordinátájú pontot, akkor az azon áthaladó egyeneseket az $r = x_0 * \cos(\theta) + y_0 * \sin(\theta)$ egyenletből fejezhetjük ki. Mivel $\theta \in [0, 2\pi]$, minden θ-hoz meghatározhatjuk $r \geq 0$ értékét az $r(\theta) = x_0 * \cos(\theta) + y_0 * \sin(\theta)$ egyenletből. Mivel a θ minden értelmezési intervallumbeli értékéhez nem tudjuk az r-et meghatározni, így a $[0, 2\pi]$ intervallum bizonyos felbontása mellett dolgozunk. Az egyes (θ, r) párok egy egyenest jelentenek.

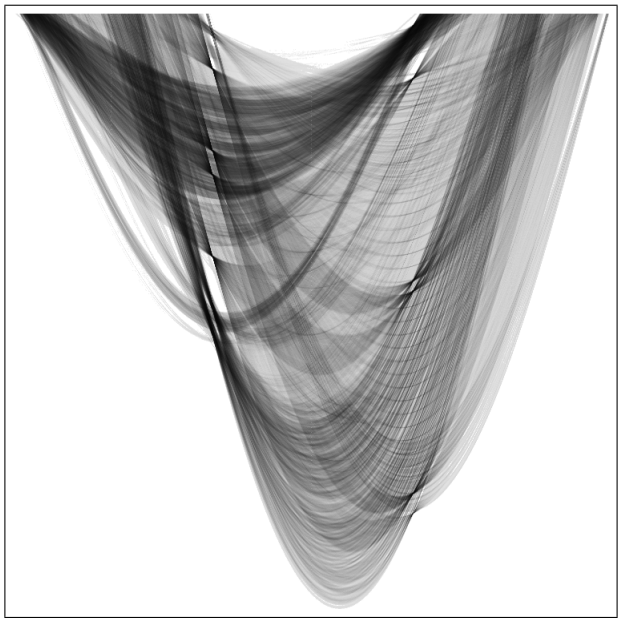


3.10. ábra. Descartes-koordináták



3.11. ábra. Polárkoordináták

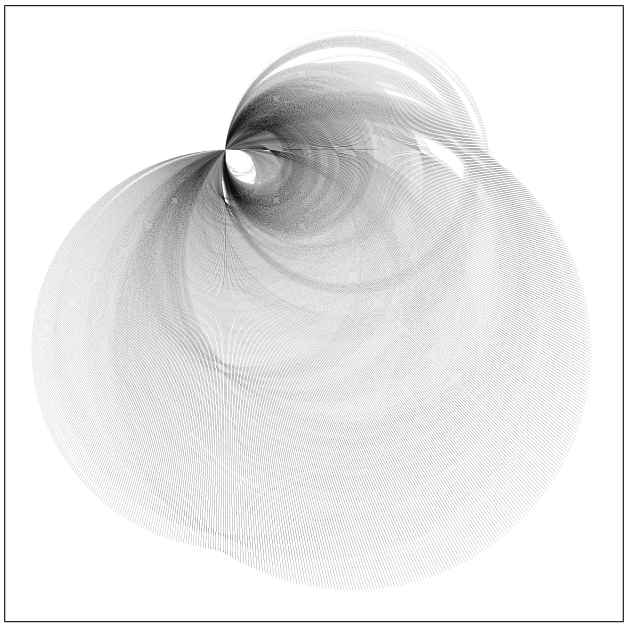
Készítsünk egy 2-dimneziós Hough-mátrixot. Ezen mátrix minden (θ, r) koordinátájú eleme egy egyenest reprezentál. Amennyiben a (θ_0, r_0) megoldása a fenti egyenletnek, akkor a Hough-mátrix θ_0-adik sorának r_0-adik elemét növeljük meg eggyel. Hajtsuk végre a fenti algoritmust minden fekete képpontra. Itt tapasztalhatjuk, hogy a szűrés során elhagyott fekete képpontok sebességnövekedést jelentenek. Végül keressük meg, hogy a mátrix melyik elemének az értéke a legnagyobb. Ezen cella koordinátái adják meg a legjobban illeszkedő egyenest. A Hough-mátrix ábrázolásakor a a mátrix legnagyobb eleméhez tartozik a legfényesebb képpont. Kétféle invertált ábrázolását is megtekinthetjük a 3.12 és a 3.13 ábrákon.



3.12. ábra. A Hough-mátrix Descartes-féle koordinátarendszerben hisztogramkiegyenlítéssel, ahol a koordináták (θ, r) alakúak

3.7. Forgatási szög megállapítása

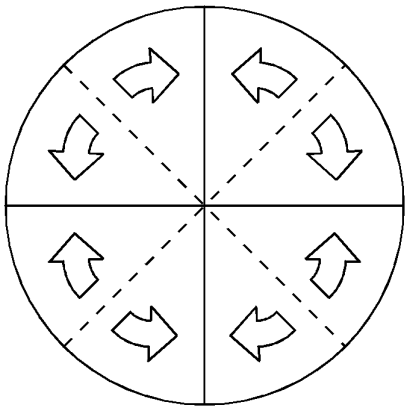
Az előzőleg megtalált egyenes eredetileg függőleges vagy vízszintes volt. A cél az, hogy ezt a vonalat azzal a függőleges vagy vízszintes vonallal párhuzamossá forgassuk, amelyiket a legkisebb forgatási szöggel el tudjuk érni. A teljes 360°-os kört 8 körcikkre osztjuk fel. Az ábrán látható módon az egyes körcikkekbe eső szöghöz tartozó nyílnak megfelelő irányba forgatunk. A forgatás szögét úgy határozzuk meg, hogy addig vonunk ki illetve adunk hozzá 90°-ot a megtalált egyenes θ-jához, amíg -45° és 45° közé eső szöget nem kapunk. Ez lesz a végső



3.13. ábra. A Hough-mátrix polárkoordinátarendszerben, ahol a koordináták (θ, r) alakúak (csak az értékes rész szerepel a képen)

elforgatás szöge, amellyel az illesztett vonalat elforgatva a kép valamelyik oldalával párhuzamos vonalat kapunk.

A 3.14 ábrán látható, hogy az adott meredekségű egyenest melyik irányba kell forgatnunk ahhoz, hogy függőleges vagy vízszintes vonalat kapjunk.



3.14. ábra. Forgatási határok

3.8. Forgatás

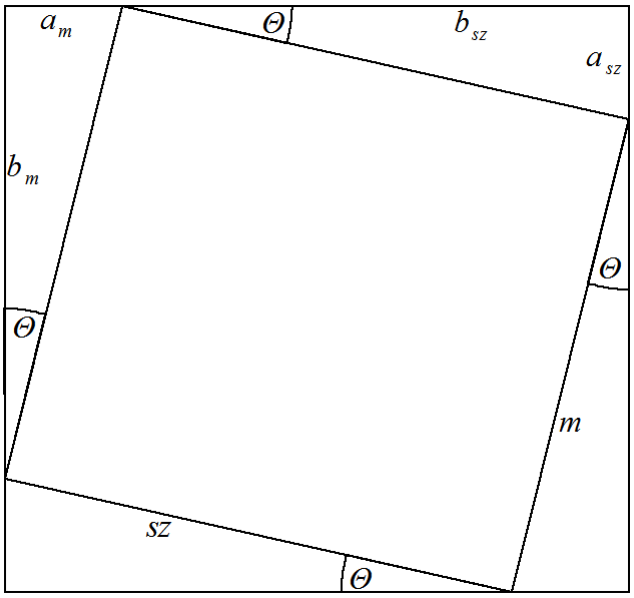
A .NET környezetben egy képet egy Bitmap objektum reprezentál. Nincs olyan beépített függvény, amelyik egy bemeneti Bitmap elforgatottját adná vissza.

A Graphics osztály egy GDI+ rajzfelületet foglal magában, eljárásokat biztosít a megjelenítőre történő rajzoláshoz. A Graphics osztályban található meg a `DrawImage(Image, Point[])` eljárás, amely az átadott képet a 3 elemű ponttömb által meghatározott paralelogrammában rajzolja ki.

A forgatást a James T. Johnson által készített kód végzi[6]. Ennek a lényege, hogy meghatározzuk ezen paralelogramma csúcspontjait, ezek ismeretében a forgatást megoldottnak tekinthetjük.

Az eredeti téglalap szélessége és magassága sz és m, az elforgatás szöge θ. Az elforgatás során az új téglalap csúcsainál egymással egybevágó, θ szögű csúccsal rendelkező derékszögű háromszögek keletkeznek. Az sz hosszúságú átfogóhoz tartozó befogók hossza a_{sz} és b_{sz}, míg az m hosszúságú átfogóhoz a_m és b_m befogóhosszak tartoznak. Ezen adatok alapján az új téglalap szélessége a_m+b_{sz}, a magassága $a_{sz}+b_m$.

A θ elforgatás szögétől függően már meg tudjuk adni az elforgatott kép csúcsainak koordinátáit. A 3.15 ábrán a $0 \leq \theta < \frac{\pi}{2}$ eset látható. Az elforgatott téglalap csúcsainak koordinátái ebben az esetben: $(a_m, 0)$, $(a_m + b_{sz}, a_{sz})$, $(b_{sz}, a_{sz} + b_m)$, $(0, b_m)$. Ezek közül három egymást követő pontot kell a már korábban említett `DrawImage` eljárásnak átadnunk.



3.15. ábra. Jelölések magyarázata a forgatáshoz

3.9. Struktúra-vonalak illesztése

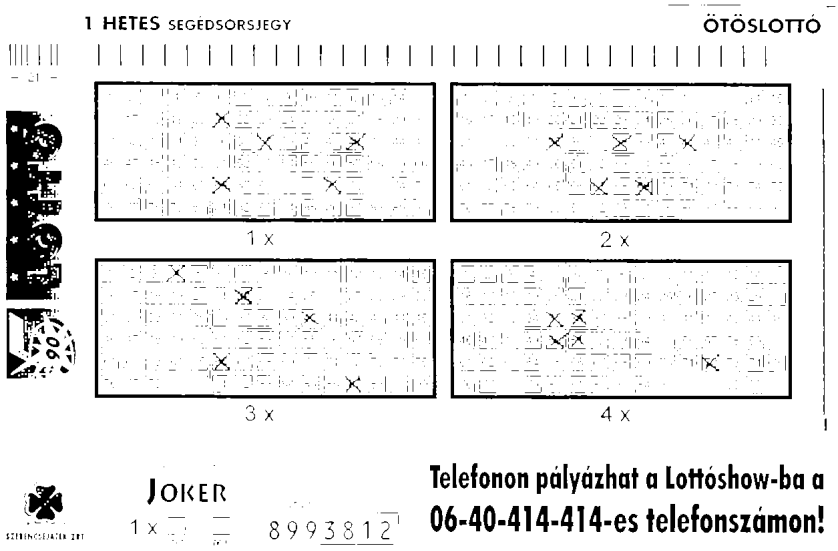
Az eredeti színes képet forgassuk el a korábban meghatározott szögnek megfelelően. Készítsük el ennek a szürkeskálás, majd a bináris képét. Hajtsuk végre lokális maximum szűrést a bináris képen, így a 3.16 ábrán látható eredményt kapjuk.

A szelvény struktúrájának konfigurálásakor megadtuk, hogy az illesztés melyik függőleges és vízszintes vonalak alapján történjen. Külön letárolásra kerül a vízszintes és a függőleges vonalak elhelyezkedésének a két szélső vonalhoz viszonyított aránya a 3.17 ábrán látható módon.

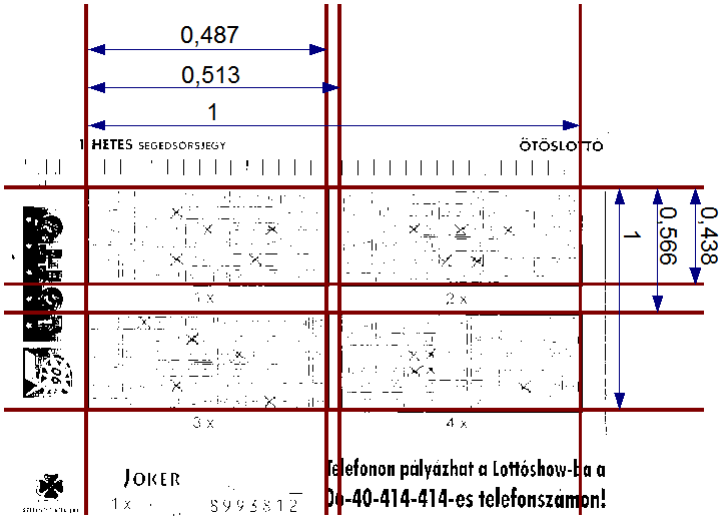
Határozzuk meg az egyes oszlopokban és sorokban lévő fekete képpontok számát. A 3.16 ábrán szereplő szelvényhez a 3.18 és a 3.19 ábrákon szereplő oszlop- és sorösszegek tartoznak.

Határozzuk meg a függőleges vonalak elhelyezkedését az oszlopokban lévő képpontok számának és függőleges vonalak elhelyezkedésének ismeretében. Végezzük el a következő műveleteket a legbalodalibb és legjobboldalibb függőleges vonal minden lehetséges helyzetére, feltételezve, hogy bármely 2 szomszédos függőleges vonal között van egy minimális fix, előre rögzített távolság. Nálam 10 pixel a határ, aminél nem eshet közelebb egymáshoz két vonal.

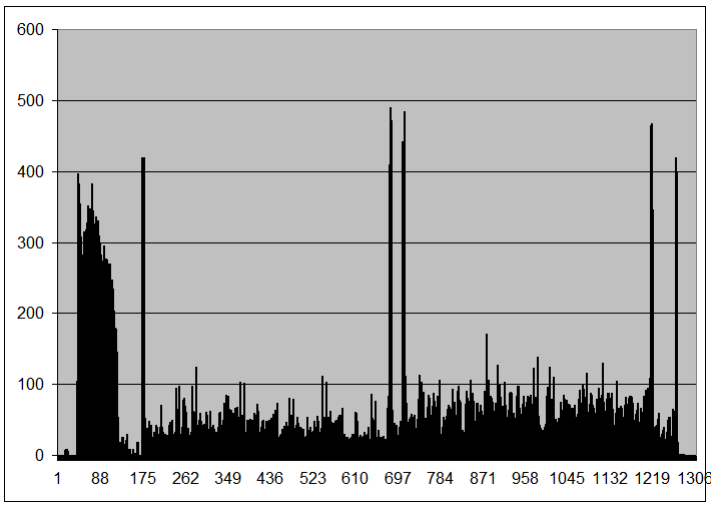
A két szélső függőleges vonal ismeretében lineáris interpolációval határozzuk meg a közbenső függőleges vonalak helyét. Adjuk össze a kiszámolt függőleges vonalakhoz tartozó oszlopokban lévő fekete képpontok számát. Ahol ez az összeg



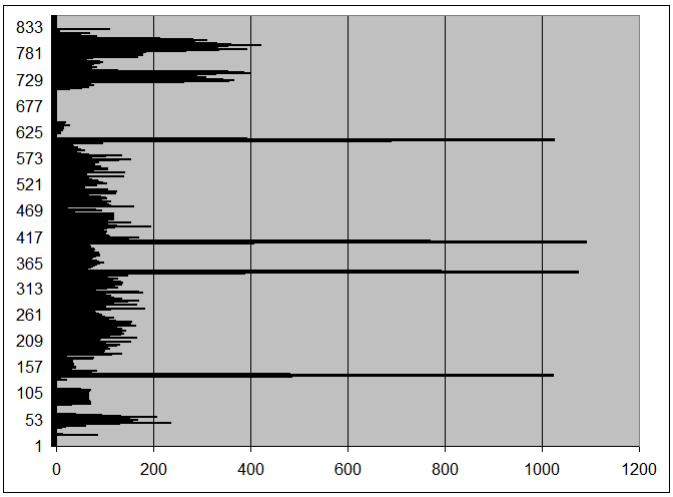
3.16. ábra. Forgatott szűrt bináris kép



3.17. ábra. Vonalak távolságaránya



3.18. ábra. A oszlopokban szereplő fekete képpontok száma



3.19. ábra. A sorokban szereplő fekete képpontok száma

maximális, ott illeszkednek legjobban a függőleges vonalak. Ugyanezt az algoritmust hajtjuk végre vízszintes vonalak esetében is teljesen analóg módon.

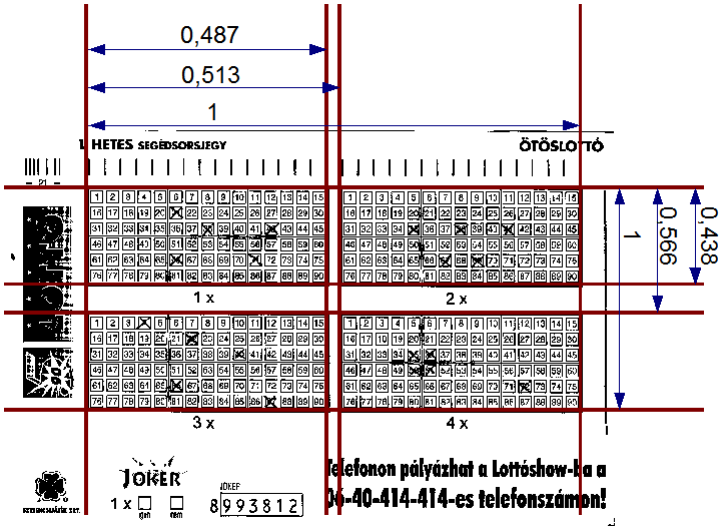
A `keresCsucsokat` függvény az oszlopösszegek és a függőleges vonalak távolságarányainak ismeretében meghatározza a vonalak pontos helyét.

```
public static int[] keresCsucsokat(int[] eredeti, double[] aranyok)
{
    int hossz = eredeti.Length;
    int mintavolsag = 10;
    int max = -1;
    int maxbal = -1;
    int maxjobb = -1;
    for (int i = 0; i < hossz - 1; ++i)
        for (int j = i + 1 + mintavolsag * (aranyok.Length - 1);
             j < hossz; ++j)
        {
            bool tulkozel = false;
            int aktualis = 0;
            int szelesseg = j - i;
            for (int k = 0; k < aranyok.Length - 1; ++k)
            {
                if ((int)((j - i) * aranyok[k + 1]) -
                    (int)((j - i) * aranyok[k]) < mintavolsag)
                {
                    tulkozel = true;
                    continue;
                }
            }
            if (tulkozel) continue;
            for (int k = 0; k < aranyok.Length; ++k)
            {
                aktualis += eredeti[i + (int)((j - i) * aranyok[k])];
            }
            if (max < aktualis)
            {
                max = aktualis;
                maxbal = i;
                maxjobb = j;
            }
        }
    int[] vissza = new int[aranyok.Length];
    for (int k = 0; k < aranyok.Length; ++k)
        vissza[k] = maxbal + (int)((maxjobb - maxbal) * aranyok[k]);
    return vissza;
}
```

4. FELISMERÉS

4.1. Kép kivágása a szelvényből

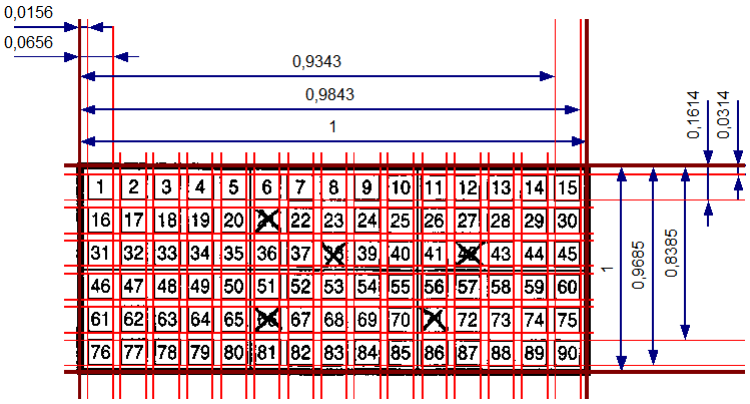
A szelvény a struktúra konfigurálása során téglalapokra, a téglalapot mezőkre osztottuk fel. A téglalapok oldalai a függőleges és vízszintes vonalakhoz való távolságarányon lettek meghatározva, így miután az illesztővonalakat a helyükre igazítottuk, lineáris interpolációval meghatározottnak tekinthetjük a téglalapokat. A mezők oldalai az őket tartalmazó téglalapok oldalaihoz viszonyított arányokkal határozhatók meg, eztáltal a mezők helyét is adottnak tekinthetjük. A téglalapok és mezők lokalizálása a 4.1 és a 4.2 ábrákon látható. Minden egyes mezőt feldolgozunk a következő lépésekben leírt módon.



4.1. ábra. Téglalapok lokalizálása a szelvényen belül

4.2. Kép szélének levágása

Az illesztéskor pontatlanság léphet fel. Ez leggyakrabban akkor fordul elő, ha egy vastag vonal a forgatási szög meghatározásának alapja, mivel erre több



4.2. ábra. Mezők lokalizálása a téglalapon belül

egyenes is illeszthető. Dönthetünk arról, hogy a mező keretvonala része legyen-e a képnek, vagy sem. Tapasztalatom alapján célszerű ezt a keretet elhagyni, mert félrevezeti a felismerést. Százalékos arányban adhatjuk meg, hogy a mező kerete mellett a magasság és szélesség hány százalékát hagyjuk el. Gyakorlatban a 10% már pontos felismerést biztosít. A 4.3 ábrán látható gyakorlatban.

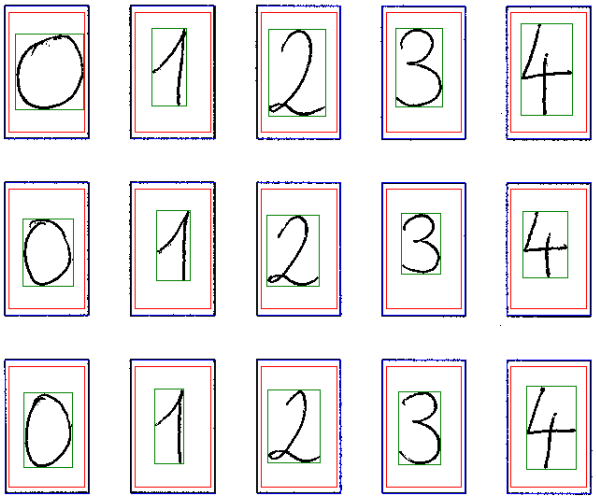
4.3. A kép hasznos részének megkeresése

Különleges esetekben előfordulhat, hogy a mezőbe írt jelet szeretnénk pontosan körülhatárolni a felismerés érdekében. Számjegyek felismerésekor különösen hasznos, ha a mezőbe írt különböző méretű jeleket pontosan körül tudjuk határolni, mivel a későbbi egységes méretre skálázásnál az azonos számjegyek jobban hasonlítanak majd egymásra. A keretet úgy szűkítjük le, hogy minden irányból vizsgáljuk az oldallal párhuzamos egyenesen lévő képpontok közt, hogy van-e köztük fekete. Addig toljuk ezt az egyenes a kép közepe fele, amíg fekete képpontba ütközünk. Az ütközési egyenes a kép közepén lévő alakzat köré írható téglalap keretvonalát adja. A fenti lépéseket minden irányban elvégezve az alakzatot pontosan körülhatároljuk. A 4.3 ábrán számjegyek megtalálása látható.

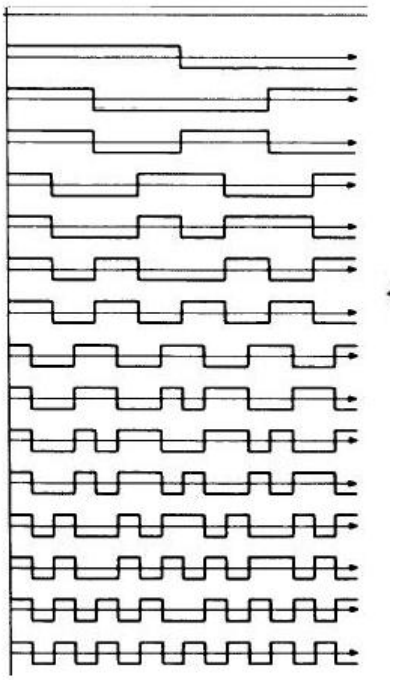
4.4. Felismerés Walsh-transzformációval

Joseph Leonard Walsh amerikai matematikus definiálta a Walsh-függvényeket. A Walsh-függvények azzal a speciális tulajdonsággal rendelkeznek, hogy -1-et vagy 1-et vesznek fel értékül a speciális értéket felvevő törtekkel kiszámított részintervallumain, ahol a törtek értékei a $[0, 1)$ intervallumból származnak.

Két dimenzióban ez számunkra -1-ekből és 1-ekből álló speciális mintázatú négyzetes mátrixokat jelent. A speciális mintázat alatt vízszintes és függőleges



4.3. ábra. Kékkel jelölve a mező megtalált szélei, pirossal a 10%-kal befele tolt határ, zölddel a megtalált hasznos rész



4.4. ábra. A Walsh-függvények mintázata 1 dimenzióban (Jacoby ábrája [5])

középtengelyre való szimmetriát és inverz szimmetriát értek. Példa található erre a 4.5 és a 4.6 ábrákon. Ezek segítségével a felismerendő karaktereket vektorokra képezhetjük le, amelyek egyedisége biztosítja a felismerés alapját. Az elkészített szoftverben a Walsh-transzformációt használtam a felismerésre.

Ebben a részben a Walsh-transzformáción alapuló felismerés egyedi lépései találhatók meg. Ha ezeket más technikán alapuló felismerés lépéseire le tudjuk cserélni, akkor is működő felismerőt kapunk.

4.4.1. Leképezés vektorra

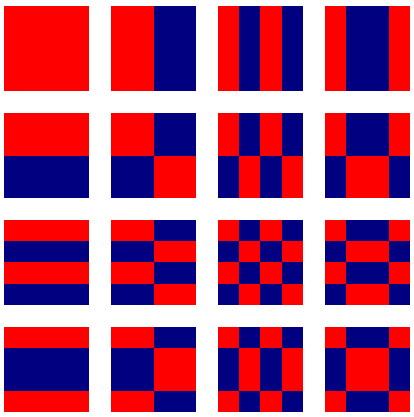
A felismerés Walsh-transzformáció segítségével történik. Ehhez különböző mátrixokat állítunk elő, amelyek segítségével az egyes képeket vektorokra képezhetjük le. A betanítás során az egyes felismert alakzatokhoz is meghatároztuk ezen vektorokat. A felismerés során megkeressük, hogy a felismerendő képhez tartozó vektornak melyik betanított vektortól való távolsága a legkisebb. Ez lesz a felismerés eredménye.

Walsh-mátrixok inicializálása

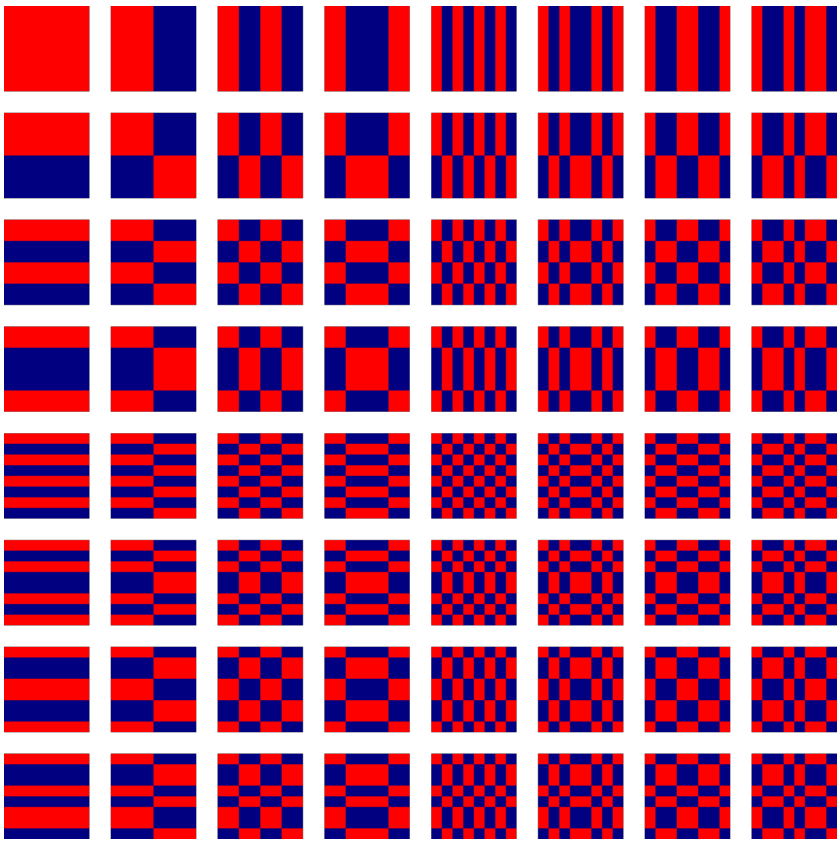
A kétdimenziós diszrkét Walsh-transzformáció $N = 2^n$ esetén a következő formában határozható meg: A $W(u, v)$ mátrix (x, y) koordinátájú elemének az értéke

$$W_{(u,v)}(x, y) = \prod_{i=0}^{n-1} -1^{b_i(x)b_{n-1-i}(u)+b_i(y)b_{n-1-i}(v)},$$

ahol $b_k(z)$ a z bináris reprezentációjának k-adik bitje.



4.5. ábra. 16×16-os Walsh-mátrixok mintázata



4.6. ábra. 64×64-es Walsh-mátrixok mintázata

Kép átméretezés 16×16-os vagy 64×64-es méretre

Következő lépésben a képet a mátrix méretével megegyező méretre kell átalakítani. Az azonos jeleknek ebben a méretben nagyon hasonlóan kell kinézni. A gyakorlat bizonyította, hogy célszerű két különböző felbontással dolgozni. 16×16-os felbontást használtam egyszerű $\times$-ek felismerésekor. Azon túl, hogy a kisebb mátrixok kisebb számolásigénnyel bírnak, a durva felbontás azzal az előnnyel is jár, hogy a mezőben elhelyezett jelek egyre jobban hasonlítanak. Ugyanazon jel különböző rajzaihoz tartozó vektorok egymástól való távolsága kicsi, ami jól jön a felismerésnél. Túl nagy felbontás esetén ezen vektorok egymástól való távolsága nő, nagyobb eséllyel fordulhat elő tévesztés. 64×64-es felbontás megfelelő számjegyek felismeréséhez. Ezek már kellően bonyolult alakzatok ahhoz, hogy kihasználják a magasabb felbontás előnyeit. Tapasztalatom szerint a kisebb felbontás számjegyeknél nagyon sok hibával dolgozik.

Kép összeszorzása a Walsh-mátrixokkal

Ezen lépés előtt rendelkezünk egy átméretezett bináris képpel, és 16×16-os felbontásnál $16(= 4 \times 4)$ db, míg 64×64-es felbontásnál $64(= 8 \times 8)$ db ugyanilyen méretű csak $+1$ és -1-et tartalmazó mátrixszal. Dolgozzuk fel ezeket a mátrixokat sorfolytonosan a következő műveletekkel. Szorozzuk össze páronként a képpontok megfelelő értékét (0,1) a Walsh-mátrix azonos pontban található értékével $(-1,+1)$. Ezen szorzatok összege lesz a képnek az adott Walsh-mátrixra való leképezése. A képet az összes mátrixszal sorfolytonosan összeszorozva kapott értékeket rendezzük egy vektorba. Ez a vektor lesz a bináris kép leképezése.

Legyen $W_{(u,v)}(x,y)$ a $W_{(u,v)}$ Walsh-mátrix (x,y) koordinátájú elemének az értéke, és legyen $K(x,y)$ az átméretezett kép (x,y) koordinátájú pontjának az értéke. A bináris kép leképezése 64×64-es felbontás esetén a v vektor lesz, amely a következőképpen áll elő:

$$\underset{\substack{u=0\ldots7\\v=0\ldots7\\i=8u+v}}{v\,[i]} = \sum_{x=0}^{63}\sum_{y=0}^{63} W_{(u,v)}(x,y)\cdot K(x,y),\ \text{ahol } v \in \mathbb{Z}^{64}.$$

16×16-os felbontás esetén a következő módon határozhatjuk meg a v vektort:

$$\underset{\substack{u=0\ldots3\\v=0\ldots3\\i=4u+v}}{v\,[i]} = \sum_{x=0}^{15}\sum_{y=0}^{15} W_{(u,v)}(x,y)\cdot K(x,y),\ \text{ahol } v \in \mathbb{Z}^{16}.$$

4.4.2. Betanított vektoroktól való távolság vizsgálata

A betanítás során az előző lépésben részletezett módon képezett vektorokat a hozzájuk tartozó betanított értékkel párban egy halmazban tároljuk le. Felismerés során a képünkből kiszámítjuk a vektort, majd ezt a vektort a halmaz minden elemével összehasonlítjuk. Az összehasonlítás során kiszámítjuk a két

vektor távolságát. Ahol ez a távolság minimális, az ahhoz tartozó betanított érték lesz a felismerés eredménye.

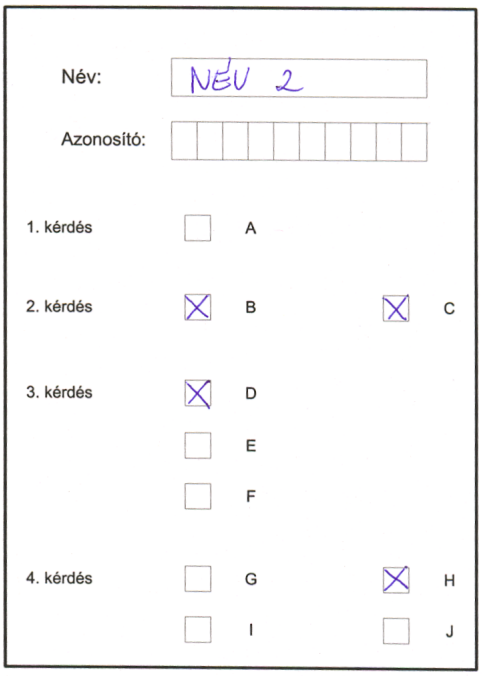
A v_1 és v_2 vektorok távolságának meghatározása:

$$d(v_1, v_2) = \sum_{i=0}^{63} |v_1[i] - v_2[i]|, \text{ ahol } v_1, v_2 \in \mathbb{Z}^{64}.$$

Természetesen 16×16-os felbontás esetén a v_1 és v_2 vektorok $\mathbb{Z}^{16}$-ból valóak, a $\sum$ ciklusváltozója pedig 0-tól 15-ig fut.

4.5. Nem felismerendő mezők begépelése

A struktúra konfigurációs XML `<mezo>` tagjének `felismerendo` attribútumában megadhatjuk `true` illetve `false` értékkel, hogy az adott mezőt fel kelle ismernie a szoftvernek. Amennyiben nem, akkor feldob egy ablakot a felhasználónak a mező képével, aki begépelheti a mező tartalmát. Különösen hasznos ez például a kézzel írott név megadásakor, mivel a program nincs ezek felismerésére felkészítve.



4.7. ábra. Példa a felismerendő és nem felismerendő mezőket tartalmazó szelvényre

5. KONFIGURÁLÁS

A konfigurálás során elkészítjük a konfigurációs XML fájlokat. Ezek a fájlok minden információt tartalmaznak, ami a feldolgozáshoz szükséges lesz. Első lépésben megállapítjuk a két szintre vágáshoz szükséges küszöböt. Következő lépésben megadjuk a szelvény struktúráját. Majd az OCR konfigurációt hajtjuk végre. Végül elkészítünk egy olyan konfigurációs fájlt, amelyik a szelvényhez tartozó összes fájl hivatkozását tartalmazza a későbbi egyszerűbb felhasználás érdekében.

5.1. Két szintre vágás korlátjának megállapítása

A szkennelés körülményeinek köszönhetően azonos fényviszonyok között készülnek a képek. Úgy határozzuk meg a két szintre vágás korlátját, hogy a rajzolt vonalak jól láthatóak legyenek. Vizsgáljuk meg különböző színű és vastagságú tollakkal jelölt alakzatok képeit is, hogy a leghalványabb is felismerhető legyen. A konfigurációs ablakban megtekinthető az eredeti színes kép, a csúszkával pedig állítható a két szintre vágási korlát. Tesztelhetünk fájlból és szkennerről beolvasott képet is. Az ablak bezárásával véglegesítődik az utoljára beolvasott korlát.

A korlát a konfigurációs fájlban a következő módon tárolható:

```
<kepfeldolgozas ... threshold="210" ... />
```

5.2. Struktúra konfigurálása

A szelvény struktúrája speciális XML fájlban tárolódik le. Ennek alkotóelemeit részletezem a következő bekezdésekben.

5.2.1. Függőleges és vízszintes vonalak megadása

A szelvényen jelöljük meg és nevezzük el a vastagabb vízszintes és függőleges vonalakat. Ezeket először a szelvény pontos illesztéséhez használhatjuk. Az illesztés precizitását növelhetjük minél több vonal megadásával.

Nem a vonalak konkrét helyét, hanem a közöttük lévő távolságok arányát fogjuk letárolni. Vegyük sorban a függőleges vonalakat balról jobbra. Tekintsük a legbaloldalibbhoz tartozó arányt 0-nak, a legjobboldalibbhoz tartozót 1-nek. A köztes függőleges vonalakhoz rendeljuk a vonal legbaloldalibb vonaltól való távolságának és a két legszélső függőleges vonal egymástól való távolságának

hányadosát. Ezzel analóg technikával határozhatjuk meg a vízszintes vonalakhoz tartozó arányokat is. A fájlban az alábbi módon szerepelnek:

```
<fuggolegesVonal nev="f2" arany="0,487"/>

<vizszintesVonal nev="v2" arany="0,438"/>
```

5.2.2. Téglalapok meghatározása

A szelvényt feloszthatjuk több téglalapra is, amennyiben a szelvényen szereplő adatokat jobban tudjuk struktúrálni, de legalább egy téglalapnak mindenképpen szerepelnie kell. A téglalapok oldalait az előző lépésben megadott konfigurációs vonalakból választhatjuk ki. Adjunk nevet a téglalapnak a későbbi áttekinthetőség érdekében.

```
<teglalap nev="1. het" bal="0" jobb="0,487"
          felso="0"  also="0,438"> ... </teglalap>
```

5.2.3. Felismerendő/nem felismerendő mezők megadása

A struktúra konfigurálásának utolsó lépésében bejelölhetjük a mezők helyét. Ehhez előbb ki kell választanunk, hogy melyik téglalapon belül található a mező. Kattintsunk a megfelelő gombra aszerint, hogy felismerendő vagy nem felismerendő mező helyét szeretnénk meghatározni. A hely meghatározása után adjuk meg a mező nevét. A mező bal és jobb szélének megállapítása a téglalap bal és jobb szélétől való, korábban már alkalmazott távolságok hányadosán alapul. A felső és alsó szél ugyanilyen módon adható meg.

```
<mezo nev="1" bal="0,015625" jobb="0,065625"
      felso="0,031428" also="0,161428" felismerendo="true"/>
```

Utolsó lépésben mentsük el az elkészült struktúra XML fájlt.

A függelék az A.1 részében megtalálható egy ötöslottó szelvény teljes stuktúra konfigurációja.

5.3. OCR konfigurálás

Eddigi eszközeinkkel már képesek vagyunk a kezdeti képen belül az egyes mezőket szegmentálni, majd ezeket vektorra leképezni. Felvetődik az a probléma, hogy hogyan rendeljük hozzá az egyes vektorokhoz a betanítandó elemet. Tökéletesen megfelel erre a feladatra a felismerés eredményeképpen előálló XML fájl struktúrája. A betanítás során képzett vektorhalmazból egy XML fájlt készítünk, ez lesz az OCR konfiguráció.

5.3.1. Tanítófájl készítése struktúra alapján

A tanítófájl célja, hogy a felismeréskor az egyes mezőket lokalizálja a téglalapnév és a mezőnév segítségével, majd kijelölt mezőhöz a betanítandó értéket hozzárendelje. A tanítófájl pontosan ugyanolyan struktúrájú, mint a felismerés eredményeképpen kapott XML.

Amennyiben lehetséges, próbáljuk meg a betanítást olyan konfiguráló szelvénnyel elvégezni, amelyen minden mező ugyanazon betanítási értékkel rendelkezik. Ehhez a lépéshez a program tartalmaz egy menüpontot, mellyel egyszerűen készíthetünk azonos betanítandó elemeket tartalmazó XML-t. A struktúrát tartalmazó fájlon kívül már csak a mindenhol szereplő értéket kell megadni, ezekből a program generál egy tanító XML-t.

Egy ötöslottó nem bejelölt mezőinek betanítására készült tanítófájl a következőképpen néz ki:

```
<?xml version="1.0" encoding="utf-8"?>
<szelveny>
  <teglalapok>
    <teglalap nev="1. het">
      <mezo nev="1">false</mezo>
      <mezo nev="2">false</mezo>
      ...
      <mezo nev="90">false</mezo>
    </teglalap>
    ...
  </teglalapok>
</szelveny>
```

5.3.2. Leképezés vektorra

A tanítófájlban szereplő téglalap és mezőnév egyértelműen meghatározza a kép egy mezőjét. A struktúra XML alapján a felismeréskor részletezett algoritmussal leképezzük az egyes mezők képét vektorra. A leképezett vektort a tanítófájlban a mezőhöz tartozó felismert értékkel párban, felismert elemként tároljuk.

Egy üres mezőhöz tartozó felismert elem XML alakban:

```
<felismertelem nev="false">
  <walshvektor>
     <koordinata>3807</koordinata>
     <koordinata>-1</koordinata>
     ...
     <koordinata>-27</koordinata>
  </walshvektor>
</felismertelem>
```

5.3.3. Az OCR XML felépítése

Rögzíteni kell, hogy 16×16-os vagy 64×64-es felbontású Walsh-mátrixokkal dolgozunk, mert a vektorok is ennek megfelelően 16 vagy 64 dimenziósok lesznek. Ezt a `<felismertelemek>` elem `meret` attribútumában adhatjuk meg.

Így néz ki egy OCR konfigurációs XML fájl:

```
<?xml version="1.0" encoding="utf-8"?>
<konfiguracio>
  <felismertelemek meret="64">
    <felismertelem nev="false">
      <walshvektor>
        <koordinata>3807</koordinata>
```

```
        <koordinata>-1</koordinata>
        <koordinata>161</koordinata>
        ...
        <koordinata>-27</koordinata>
      </walshvektor>
    </felismertelem>
    <felismertelem nev="false">
    ...
    </felismertelem>
    ...
  </felismertelemek>
</konfiguracio>
```

5.4. Összefoglaló konfiguráció elkészítése

Az összefoglaló konfigurációnak akkor van fontos szerepe, amennyiben egy adott szelvényhez tartozó összes fájlt egyszerre szeretnénk hivatkozni. (struktúra, szintaktikai elemzés, szemantikai elemzés, XSLT konverziós fájlok, ...) Így egy azonos típusú szelvényekből álló szelvénycsoport feldolgozásához elegendő az összefoglaló konfigurációs fájlt betölteni, ez az összes többi fájlra tartalmaz hivatkozást.

A következőképpen néz ki egy ötöslottó szelvényhez készült összefoglaló konfigurációs fájl:

```
<?xml version="1.0" encoding="UTF-8"?>
<konfiguracio nev="ötöslottó">
   <struktura fajlnev="./Otoslotto/otoslotto-struktura.xml"/>
   <elemzes tipus="szintaktika"/>
   <ocr fajlnev="./Otoslotto/otoslotto-ocr.xml"
        xslt="./Otoslotto/otoslotto-xslt.xsl"/>
   <szintaktika fajlnev="./Otoslotto/otoslotto-elemzes.xml"/>
   <kepfeldolgozas keret="0" threshold="210" filter="no"/>
</konfiguracio>
```

A `<struktura/>` tag `fajlnev` mezőjében található a struktúra konfigurációs fájl elérési útja.

Az elemzés típusa lehet `szintaktika`, `szemantika`, továbbá elemzés kihagyása esetén `no` is.

Az `<ocr>` tag alatt az OCR konfigurációs fájl és az XSLT konverziós fájl elérési útja található meg.

A `<szintaktika>` tag alatt a szintaktikai elemzőfájl elérési útja található.

A `<kepfeldolgozas>` tag a képfeldolgozáshoz szükséges adatokat tartalmazza. A `keret` attribútum a keret szélességét tartalmazza százalékban. A `threshold` attribútumban a két szintre vágás korlátja található. A `filter` attribútum értéke `no` vagy `kivagassal` lehet, attól függően, hogy a mezőt le kell-e szűkítenünk pontosan a benne szereplő jelre.

6. ELEMZÉS

A felismerés során előálló eredményeken megpróbálunk bizonyos vizsgálatokat elvégezni. Ezek ismeretében döntünk arról, hogy helyesen van-e a szelvény kitöltve.

6.1. Szintaktikai elemzés

Szintaktikai elemzés során megvizsgáljuk, hogy a szelvény eleget tesz-e egy szabályrendszernek. Logikai műveletekből, elemi felismerésből és darabszámkorlátokból építhetünk fel egy feltételrendszert, nem pontos találatokat keresünk.

6.1.1. Szabályrendszer ismertetése

Minden műveletről eldönthető, hogy helyes-e. A szabályrendszer a következő műveletekből épülhet fel:

- Találat: Az egyetlen elemi művelet. Ha a szelvény adott téglalapjában lévő adott mező felismert értéke *true*, akkor helyes, különben helytelen.

```
<talalat teglalapnev="1. het" mezonev="1"/>
```

- És: Amennyiben az általa összekötött minden művelet helyes, akkor értéke helyes, különben helytelen.

```
<and nev="Mind a 4 hét helyesen van kitöltve"> ... </and>
```

- *(Megengedő)* Vagy: Ha az általa összekötött műveletek között szerepel helyes, akkor értéke helyes, különben helytelen.

```
<or nev="Valamelyik teljesül"> ... </or>
```

- Kizáró vagy: Ha két műveletet köt össze és a két művelet helyessége nem egyezik meg, akkor értéke helyes, különben helytelen.

```
<xor nev="Pontosan egy teljesül"> ... </xor>
```

- Nem: Értéke az alá tartozó művelet helyességének ellentéte.

```
<not nev="Nem teljesül"> ... </not>
```

- Darabszámkorlát: Ha az általa összekötött műveletek között minimum és maximum közé eső helyes művelet van, akkor értéke helyes, különben helytelen. Ha az egyik korlát nincs megadva, akkor csak a másik teljesülését vizsgáljuk.

```
<darabszamkorlat nev="Pontosan 5-öt kell bejelölni"
     minimum="5" maximum="5"> ... </darabszamkorlat>
```

A függelék az A.2 részében található meg egy teljes szintaktikai elemzés konfiguráló XML.

6.2. Szemantikai elemzés

Feltételezzük, hogy a szelvényünk az azonosításhoz szükséges mezőkből, kérdésekből és a kérdésekre adott válaszokból áll. Az azonosításhoz szükséges mezőknek nem vizsgáljuk az értékét, egyszerűen csak átvesszük őket. Minden egyes kérdéshez tartozik egy pontérték. Amennyiben helyesen válaszoltunk minden pontot megkapunk, különben 0-át.

6.2.1. Szabályrendszer ismertetése

Minden válaszról eldönthető, hogy helyes-e. Összetett szabályrendszerrel adhatjuk meg egy válasz helyességét.

- Elemi válasz: Ha a felismerés eredménye megegyezik a várt válasszal, akkor helyes, különben helytelen.

```
<valasz teglalapnev="t" mezonev="A">true</valasz>
```

- És: Ha az általa összefoglalt válaszok mindegyike helyes, akkor értéke helyes, különben helytelen.

```
<and> ... </and>
```

- *(Megengedő)* Vagy: Ha az általa összekötött műveletek között szerepel helyes, akkor értéke helyes, különben helytelen.

```
<or> ... </or>
```

- Kizáró vagy: Ha két műveletet köt össze és a két művelet helyessége nem egyezik meg, akkor értéke helyes, különben helytelen.

```
<xor> ... </xor>
```

- Nem: Értéke az alá tartozó művelet helyességének ellentéte.

```
<not> ... </not>
```

A függelék az A.3 részében található meg egy teljes szemantikai elemzés konfiguráló XML.

7. KONVERZIÓ XSLT STÍLUSLAPPAL

A programba a felismert eredményeket tartalmazó XML és a szemantikai eredményeket tartalmazó XML fájlok XSLT stíluslappal konvertálása van beépítve. Amennyiben valaki új szelvény konfigurálását szeretné elvégezni, saját kezűleg kell az XSLT fájlt elkészítenie. Az XSLT szabvány kellően rugalmas ahhoz, hogy tetszőlegesen formázott kimeneti HTML-t készíthessen a felhasználó. Az XSLT stíluslap használatát egy űrlapon keresztül mutatom be.

Egy eldöntendő kérdésekből álló űrlaphoz tartozó minta XSLT stíluslap

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">
    <xsl:template match="/">
        <html>
            <head>
                <meta content="text/html; charset=UTF-8"
                                http-equiv="Content-Type"/>
                <meta name="author" content="Nagy András"/>
                <title>Urlap</title>
            </head>
            <body>
                <xsl:apply-templates select="szelvenyek"/>
            </body>
        </html>
    </xsl:template>

    <xsl:template match="szelvenyek">
        <h1>Urlap</h1>
        <xsl:apply-templates select="szelveny"/>
    </xsl:template>

    <xsl:template match="szelveny">
        <hr/>
        <h2>
            <xsl:value-of select="position()"/>
            <xsl:text>. lap</xsl:text>
        </h2>
        <xsl:apply-templates select="teglalapok"/>
    </xsl:template>

    <xsl:template match="teglalapok">
        <xsl:apply-templates select="teglalap"/>
    </xsl:template>

    <xsl:template match="teglalap">
        <table border="1" width="25%" align="center">
```

```
            <xsl:apply-templates select="mezo"/>
        </table>
    </xsl:template>

    <xsl:template match="mezo">
        <tr>
            <td align="center">
                <xsl:value-of select="@nev"/>
            </td>
            <td>
                <xsl:value-of select="."/>
            </td>
        </tr>
    </xsl:template>
</xsl:stylesheet>
```

A végrehajtott felismerés eredménye

```
<?xml version="1.0" encoding="utf-8"?>
<szelvenyek>
  <szelveny>
    <teglalapok>
      <teglalap nev="t">
        <mezo nev="Név">Név1</mezo>
        <mezo nev="A">true</mezo>
        <mezo nev="B">false</mezo>
        <mezo nev="C">true</mezo>
        <mezo nev="D">false</mezo>
        <mezo nev="E">true</mezo>
        <mezo nev="F">false</mezo>
        <mezo nev="G">true</mezo>
        <mezo nev="H">false</mezo>
        <mezo nev="I">false</mezo>
        <mezo nev="J">true</mezo>
      </teglalap>
    </teglalapok>
  </szelveny>
</szelvenyek>
```

A gyakorlatban a következőképpen működik a transzformáció.

Elsőként `<xsl:template>` elem `match` attribútumának értékét illesztjük a dokumentumfára. Elkészül a HTML fájl `<head>` blokkja és a `<body>` keret is.

Következik a `<szelvenyek>` elem, kiírásra kerül az Űrlap felirat.

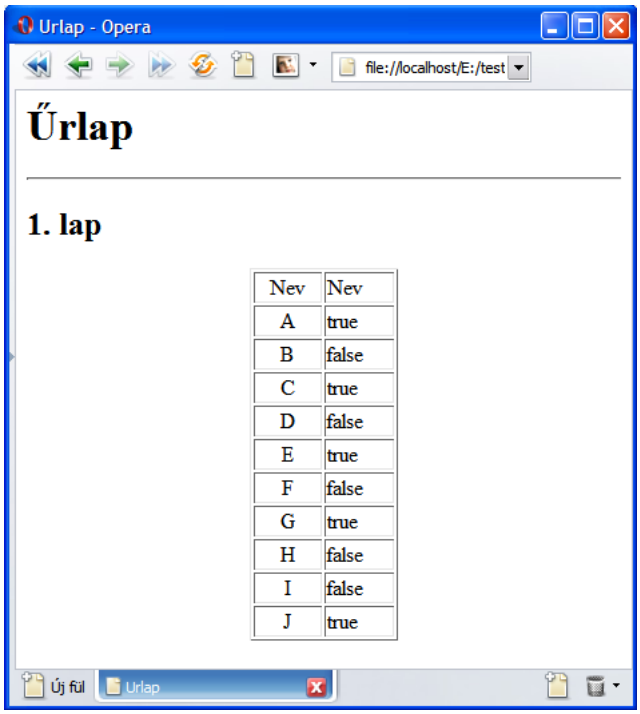
Feldolgozásra kerül az összes `<szelveny>` elem, amiből jelen esetben csak 1 darab van. Minden egyes szelvény leírása egy vízszintes vonallal kezdődik, amit a `<hr/>` HTML elem ír ki. Kiírásra kerül, hogy hanyadik szelvényről van szó.

A `<teglalapok>` elem, majd azon belül a `<teglalap>` elemek kerülnek feldolgozásra. Minden téglalap leírása egy táblázatban történik.

Kiíródnak az aktuális `<teglalap>` elemen belüli `<mezo>` elemek. Egy mező egy táblázatsornak felel meg. Az első oszlopban a `nev` attribútum tartalma, a másodikban az elem tartalma kerül kiíratásra.

Miután a teljes dokumentumfa rekurzívan feldolgozásra kerül, előáll a kimeneti HTML fájl, amit elmenthetünk. Ezt böngészőben megtekintve a 7.1

ábrához hasonló képet kapunk.



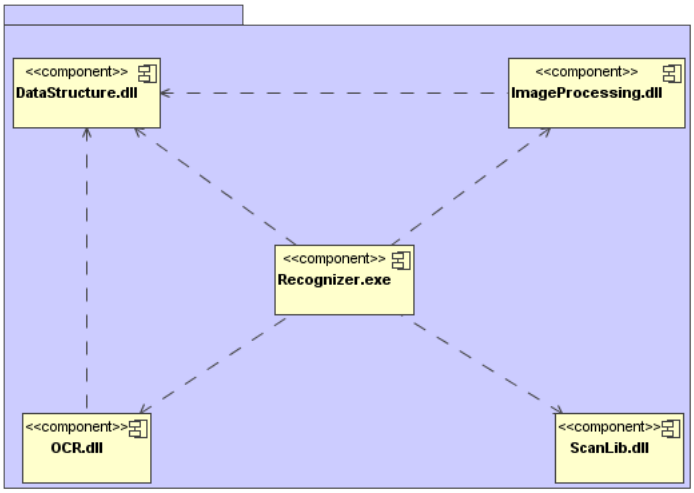
7.1. ábra. A generált HTML fájl Opera böngészővel megjelenítve

8. PROGRAMFEJLESZTÉSI DOKUMENTÁCIÓK

Bonyolult rendszerek megvalósításához előzetes tervek elkészítésére van szükségünk. A különböző részletességű terveket az alkalmazás modelljének nevezzük. A modellek áttekinthetőbben ábrázolják az alkalmazást, mivel eltekintenek a megvalósítási technikáktól. Ábrázolása diagramok segítségével történik.

Az UML[1] az objektummodellezés grafikus jelölőnyelve. Az UML egy általános célú modellező nyelv, az objektumorientált elemzés és tervezés eszköze. Egy UML jelölésekkel elkészített rendszer absztrakt modelljét UML modellnek nevezzük. Az UML modell diagramokból áll, ezek közül csak azokra térek ki részletesebben, amelyeken keresztül az alkalmazás folyamatait szemléltetem.

A komponens diagram a rendszer komponensekre felosztását és ezen komponensek közötti összefüggéseket ábrázolja. Bármely rendszer architektúrájának modellezésére és dokumentálására is használható.



8.1. ábra. A komponens diagram

[1] Unified Modelling Language

Az osztály diagram olyan struktúra diagram, amely leírja a rendszer osztályait, attribútumait és az osztályok közötti kapcsolatokat. Az osztály diagram rögzíti az objektumok közötti kapcsolatok szabályait. A függelékben a B.1 és a B.2 ábrákon tekinthető meg.

Az aktivitás diagram egy rendszer komponenseinek időben lezajló változását fejezi ki. Nem csak szekvenciális, hanem konkurens végrehajtás is meghatározható. A teljes vezérlési folyamatot bemutatja. A függelékben a B.3 és a B.4 ábrákon tekinthető meg.

A szekvencia diagram a párhuzamosan létező objektumokat és a köztük történő üzenetváltásokat a bekövetkezés sorrendjében mutatja meg. Az objektumok életútját a létrehozástól a megszűnésig követhetjük. A függelékben a B.5 ábrán tekinthető meg.

9. TOVÁBBFEJLESZTÉSI LEHETŐSÉGEK

A szoftver tudása természetesen nem éri el egy kereskedelmi szoftverét, de nem is ebből a célból született. Ennek köszönhetően vannak olyan területek, ahol van még továbbfejlesztési lehetőség. Sok ötlet a felismerési sebesség növelésére vonatkozik.

A felismerő sosem dolgozhat elég pontosan. Minél jobban illeszkednek a forgatás és a struktúraillesztés után az igazító vonalak az eredetijükre, annál kisebb részt kell levágni a mező széléből. Ezzel elkerülhetjük azt a problémát, hogy valaki a mező keretéhez hozzáérő jeleket rajzol, és a levágás során értékes információt veszítünk.

A program futtatása során megfigyelhető, hogy a felismerés idejének legnagyobb része a forgatási szög meghatározásához szükséges egyenes illesztésével telik el. A kép méretének növekedésével egyenes arányban növekedik az illesztés ideje. Feltételezésem szerint az eredeti kép kettőhatványad oldalhosszúságú kicsinyítésein vonalat keresve a normál méretű képen megtalálható vonalhoz közel eső vonalat kapunk. A durvább képen megtalált vonal adatai alapján a finomabb képen már elegendő csak a korábbi vonal környezetében a vonalat keresni, amivel időt spórolhatunk.

A szoftver nincs arra felkészítve, hogy egy űrlapon belül többféle típusú jelet próbáljunk meg felismerni. Így még nem oldható meg egy szelvényen belül számjegyek és $\times$-ek felismerése. További problémát okoz, hogy az egyes típusokhoz különböző átméretezés tartozhat a felismerés finomságától függően.

Nagyobb léptékű átalakítást igényelne a nem csak Walsh-transzformáción alapuló karakterfelismerés. Karakterfelismerés történhet még többek között vázkijelöléssel és neurális hálókkal is.

Sok űrlapon keresztül betanított konfigurációs fájllal pontosabb felismerést érhetünk. Nem csak előnye van ám, hanem hátránya is. A fájl egyre több vektort tartalmaz, így a betanított vektorokhoz való hasonlításkor minden alkalommal több hasonlítási lépést végzünk. Ezen vektorok közül az azonos betanított értékhez tartozók egymástól való távolsága kicsi. Úgy képzelhető el, mint ha egy 16 vagy 64 dimenziós *térben* bizonyos pontok körül megnő a betanított vektortok által reprezentált pontok sűrűsége. Amennyiben valamilyen magasabb matematikai eszközzel ezen sokdimenziós vektorokat osztályozhatnánk, akkor az egyes csoportokat elegendő lenne egy átlagvektorral reprezentálni, így kevesebb hasonlítási művelet is elegendő lenne.

IV. rész

ÖSSZEFOGLALÁS

A dolgozatban egy űrlapok kitöltésének automatikus feldolgozását végző szoftver tervezését és megvalósítását tűztem ki célul.

Programozás során nem az elérhető legújabb nyelveket és technológiákat használtam, mert azok sem tartalmaztak olyan eszközöket, amelyek segítségével hatékonyabb, gyorsabb vagy jobban strukturált kódot készíthettem volna.

Az alkalmazott technológiáknak köszönhetően a szoftver integrálható nagyobb rendszerbe. A feldolgozás eredményeképpen előálló XML fájlt tetszőlegesen felhasználhatjuk. Például továbbíthatjuk adatbázisba, készíthetünk weboldalt belőle vagy felhasználhatjuk más program bemeneteként. A program átalakítható oly módon, hogy webszolgáltatásként működjön. Így a telepítés helyétől távolról is elérhető, elegendő egy böngésző a használatához.

A program tervezése során végig szem előtt tartottam, hogy képes legyen teszt típusú dolgozatok automatikus felismerésére, pontozására és az eredmények összesítésére. Nem érdemes azonban kevés kijavítandó dolgozat esetén a szoftvert használni, mert a konfigurálásba és betanításba fektetett idő nem biztos, hogy megtérül. Pontos számításokat nem végeztem arra, hogy hány feldolgozandó lapnál van a kézi-gépi feldolgozás közötti döntés határa, mert ez függ az egy oldalon szereplő mezők számától, az egy kérdéshez tartozó válaszmezőktől és a válaszok kiértékelésének bonyolultságától is.

Az eddigiek során nem ejtettem szót arról, hogy milyen felbontással szkennelt képeket használtam. Okkal nem említettem, mivel a feldolgozásban egyáltalán nem használtam ki ezt a tulajdonságot. Mivel a szelvények konfigurálása során végig arányokkal és nem távolságokkal dolgoztam, így egy nagyobb felbontással beolvasott képpel semmi probléma nem akad. Gyakorlatban a lehető leggyorsabb feldolgozás érdekében 200 dpi-s felbontással szkennelt képeket alkalmaztam.

A legtöbb problémám a képfeldolgozási technikák kidolgozásával adódott, számos zsákutcába futottam bele. Próbáltam a feldolgozás menetét teljesen automatizálni azért, hogy a felhasználónak a lehető legkevesebbet kelljen, illetve lehessen beavatkozni. Alapvetően kétféle tudású felhasználót különböztetek meg: egyszerű felhasználó, aki csak szelvények feldolgozását végzi; és operátor, aki ismeri a háttérben lezajló folyamatokat, így konfigurálást is tud végezni a feldolgozáson túl.

További problémáim adódtak még a szkenner elérésekor. Törekedtem arra, hogy a szoftver ne csak megnyitott képfájlokat tudjon feldolgozni, hanem szkenner kezelésére is fel legyen készítve, mivel annak használata életszerűbb. A szkenner elérése TWAIN interfészen keresztül történik. Szkenner használata által folyamatos, holt idő nélküli feldolgozás is elérhető.

A szoftver hatékonyságát két fő területen lehetne mérni: gyorsaság és felismerési pontosság. A szoftver minden egyes elindításakor a háttérben kiszámítódnak a Walsh-mátrixok, ami eléggé gépigényes feladat. Ezen mátrixok állandóságuk miatt minden mező felismeréséhez használhatók, így meghatározásuk egyszer is elegendő, de nem spórolható meg. Erre az a legmegfelelőbb alkalom, amikor a felhasználó a beállítások betöltésével van elfoglalva. Maga a feldolgozás erősebb processzorral és gyorsabb szkennerrel tovább gyorsítható. Viszont a program nincs felkészítve több szkenner párhuzamos használatára és

több szelvény párhuzamos feldolgozására sem. Természetesen a program valódi használhatósága és gyakorlatban mért pontossága nagyon sok szelvénnyel való, ipari méretű tesztelésen bizonyosodhatna be, de erre nem volt lehetőségem.

Befejezésül elmondhatom, hogy a rendelkezésemre álló matematikai, informatikai és képfeldolgozási eszközökkel illetve szorgalommal egy ilyen szoftver elkészítése egyáltalán nem volt megoldhatatlan feladat.

IRODALOMJEGYZÉK

[1] Attila Fazekas and András Hajdu, „An algorithm using Walsh transformation for compressing typeset documents," Acta Math. Acad. Paedagog. Nyházi. (N.S.) 15 (1999), 61-68.

[2] Fazekas Attila és Kormos János, „Digitális képfeldolgozás matematikai alapjai," mobiDIÁK könyvtár, Debrecen, 2004.

[3] Attila Fazekas and András Hajdu, „Extracting Feature Vectors for Character Recognition by Walsh Transformation," in Proc. of KÉPAF 3 (2002), Domaszék, Hungary, 272-278.

[4] Attila Fazekas and András Hajdu, „Recognizing typeset documents using Walsh transformation," J. Comput. Inf. Technol. 9 (2001), 101-112.

[5] Benjamin Jacoby, „Walsh Functions: A Digital Fouries Series," BYTE Publications, 2/9 (1997), 190.

[6] James T. Johnson, „Image Rotation in .NET," 2002, 5 pages, `http://www.codeproject.com/KB/graphics/rotateimage.aspx`

[7] Jay Leventhal, „The Man and the Machine: An Interview with Ray Kurzweil," AFB AccessWorld, 5/5 (2004), 5 pages, `http://www.afb.org/afbpress/pub.asp?DocID=aw050505`

[8] Douglas Martin, „David H. Shepard, 84, Dies; Optical Reader Inventor," The New York Times, December 11, 2007 `http://www.nytimes.com/2007/12/11/us/11shepard.html`

[9] Vég Csaba, „Alkalmazásfejlesztés a Unified Modeling Language szabványos jelöléseivel," Logos 2000, 1999, 242 oldal.

[10] Wikipedia, „Optical character recognition," `http://en.wikipedia.org/wiki/Optical_character_recognition`

[11] „Feature Comparison of OCR Software", OCR Review, 2007, 5 pages, `http://www.ocrreview.com/ocr-comparison-matrix/`

[12] „C# Version 2.0 Specification", Microsoft Corporation, 2005, 115 pages.

[13] „OMG Unified Modeling Language Superstructure, V2.1.2," Object Management Group, 2007, 738 pages.

FÜGGELÉK

A. XML KONFIGURÁCIÓS PÉLDÁK

A.1. Struktúra konfigurációs XML minta

```
<?xml version="1.0" encoding="UTF-8"?>
<szelveny>
  <vonalak>
    <fuggolegesVonalak>
      <fuggolegesVonal nev="f1" arany="0"/>
      <fuggolegesVonal nev="f2" arany="0,487"/>
      <fuggolegesVonal nev="f3" arany="0,513"/>
      <fuggolegesVonal nev="f4" arany="1"/>
    </fuggolegesVonalak>
    <vizszintesVonalak>
      <vizszintesVonal nev="v1" arany="0"/>
      <vizszintesVonal nev="v2" arany="0,438"/>
      <vizszintesVonal nev="v3" arany="0,566"/>
      <vizszintesVonal nev="v4" arany="1"/>
    </vizszintesVonalak>
  </vonalak>
  <teglalapok>
    <teglalap nev="1. het" bal="0" jobb="0,487"
                           felso="0" also="0,438">
      <mezo nev="1" bal="0,015625" jobb="0,065625"
          felso="0,031428" also="0,161428" felismerendo="true"/>
      <mezo nev="2" bal="0,08125" jobb="0,13125"
          felso="0,031428" also="0,161428" felismerendo="true"/>
      <mezo nev="3" bal="0,146875" jobb="0,196875"
          felso="0,031428" also="0,161428" felismerendo="true"/>
      ...
      <mezo nev="90" bal="0,934375" jobb="0,984375"
          felso="0,838571" also="0,968571" felismerendo="true"/>
    </teglalap>
    <teglalap nev="2. het" bal="0,513" jobb="1"
                           felso="0" also="0,438">
      <mezo nev="1" bal="0,015625" jobb="0,065625"
          felso="0,031428" also="0,161428" felismerendo="true"/>
      ...
      <mezo nev="90" bal="0,934375" jobb="0,984375"
          felso="0,838571" also="0,968571" felismerendo="true"/>
    </teglalap>
    <teglalap nev="3. het" bal="0" jobb="0,487"
                           felso="0,566" also="1">
      <mezo nev="1" bal="0,015625" jobb="0,065625"
          felso="0,031428" also="0,161428" felismerendo="true"/>
      ...
      <mezo nev="90" bal="0,934375" jobb="0,984375"
          felso="0,838571" also="0,968571" felismerendo="true"/>
```

```
    </teglalap>
    <teglalap nev="4. het" bal="0,513" jobb="1"
                           felso="0,566" also="1">
      <mezo nev="1" bal="0,015625" jobb="0,065625"
          felso="0,031428" also="0,161428" felismerendo="true"/>
      ...
      <mezo nev="90" bal="0,934375" jobb="0,984375"
          felso="0,838571" also="0,968571" felismerendo="true"/>
    </teglalap>
  </teglalapok>
</szelveny>
```

A.2. Szintaktikai elemzést konfiguráló XML minta

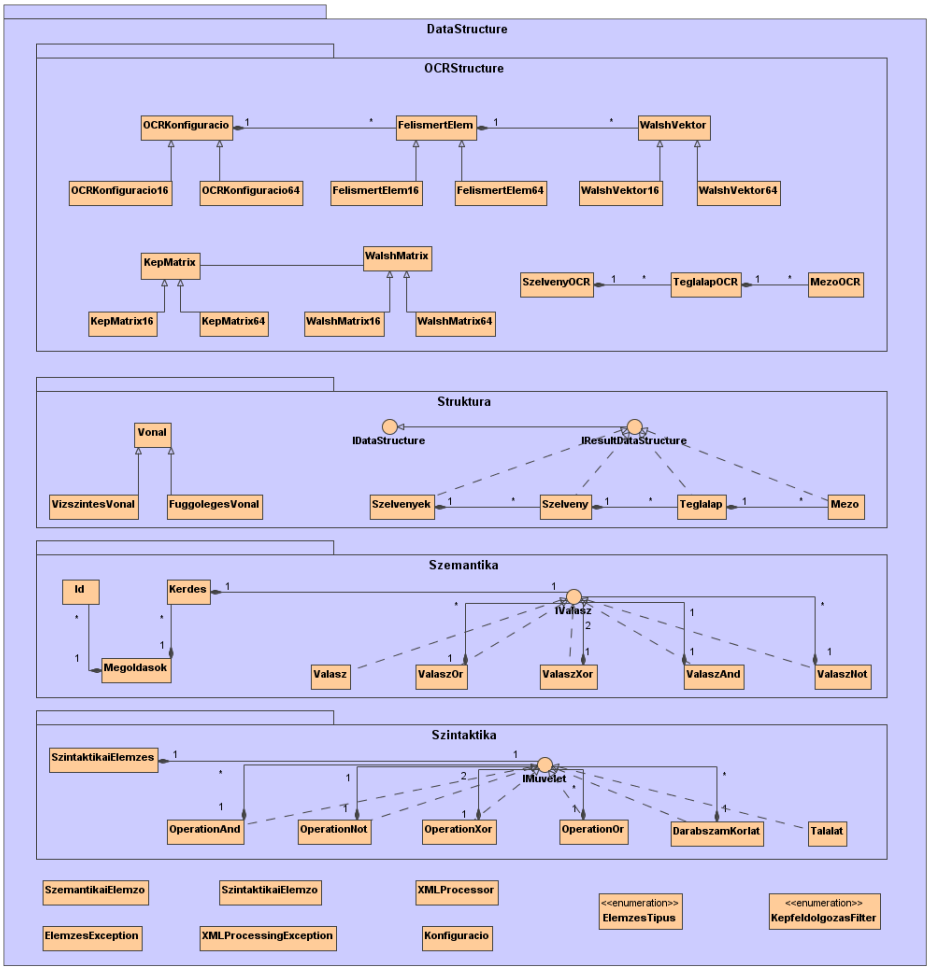
```
<?xml version="1.0" encoding="UTF-8"?>
<elemzes>
  <and nev="Mind a 4 het helyesen van kitoltve">
    <darabszamkorlat nev="Pontosan 5-ot kell bejelolni"
                minimum="5" maximum="5">
      <talalat teglalapnev="1. het" mezonev="1"/>
      <talalat teglalapnev="1. het" mezonev="2"/>
      ...
      <talalat teglalapnev="1. het" mezonev="90"/>
    </darabszamkorlat>
    <darabszamkorlat nev="Pontosan 5-ot kell bejelolni"
                minimum="5" maximum="5">
      <talalat teglalapnev="2. het" mezonev="1"/>
      <talalat teglalapnev="2. het" mezonev="2"/>
      ...
      <talalat teglalapnev="2. het" mezonev="90"/>
    </darabszamkorlat>
    <darabszamkorlat nev="Pontosan 5-ot kell bejelolni"
                minimum="5" maximum="5">
      <talalat teglalapnev="3. het" mezonev="1"/>
      <talalat teglalapnev="3. het" mezonev="2"/>
      ...
      <talalat teglalapnev="3. het" mezonev="90"/>
    </darabszamkorlat>
    <darabszamkorlat nev="Pontosan 5-ot kell bejelolni"
                minimum="5" maximum="5">
      <talalat teglalapnev="4. het" mezonev="1"/>
      <talalat teglalapnev="4. het" mezonev="2"/>
      ...
      <talalat teglalapnev="4. het" mezonev="90"/>
    </darabszamkorlat>
  </and>
</elemzes>
```

A.3. Szemantikai elemzést konfiguráló XML minta

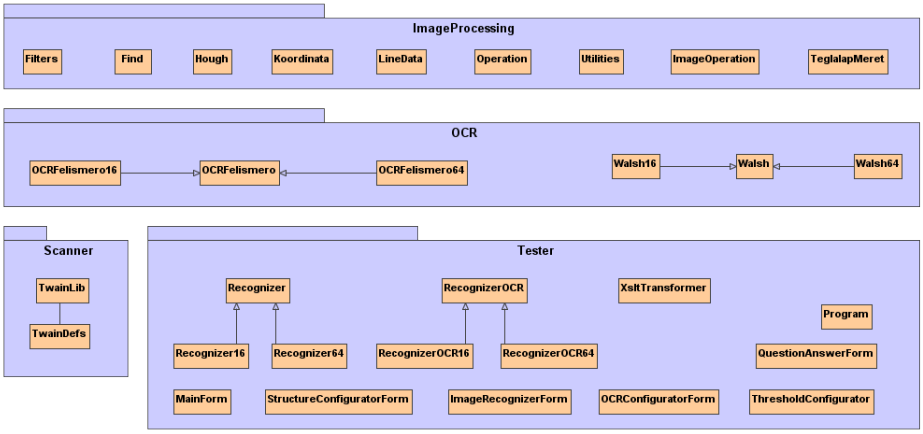
```
<?xml version="1.0" encoding="utf-8"?>
<megoldasok>
  <ids>
    <id teglalapnev="t" mezonev="Név"/>
  </ids>
  <kerdesek>
```

```
    <kerdes nev="1. kérdés" pont="1">
      <valasz teglalapnev="t" mezonev="A">true</valasz>
    </kerdes>
    <kerdes nev="2. kérdés" pont="2">
      <and>
        <valasz teglalapnev="t" mezonev="B">false</valasz>
        <valasz teglalapnev="t" mezonev="C">false</valasz>
      </and>
    </kerdes>
    <kerdes nev="3. kérdés" pont="3">
      <and>
        <valasz teglalapnev="t" mezonev="D">true</valasz>
        <valasz teglalapnev="t" mezonev="E">true</valasz>
        <valasz teglalapnev="t" mezonev="F">true</valasz>
      </and>
    </kerdes>
    <kerdes nev="4. kérdés" pont="4">
      <and>
        <valasz teglalapnev="t" mezonev="G">false</valasz>
        <valasz teglalapnev="t" mezonev="H">false</valasz>
        <valasz teglalapnev="t" mezonev="I">false</valasz>
        <valasz teglalapnev="t" mezonev="J">false</valasz>
      </and>
    </kerdes>
  </kerdesek>
</megoldasok>
```

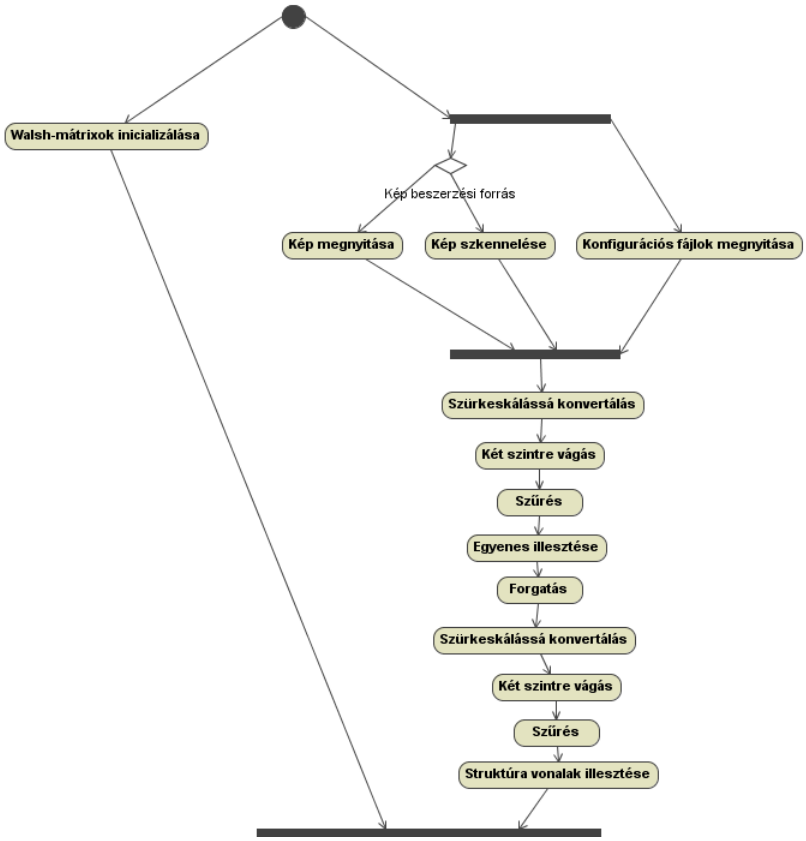
B. UML DIAGRAMOK



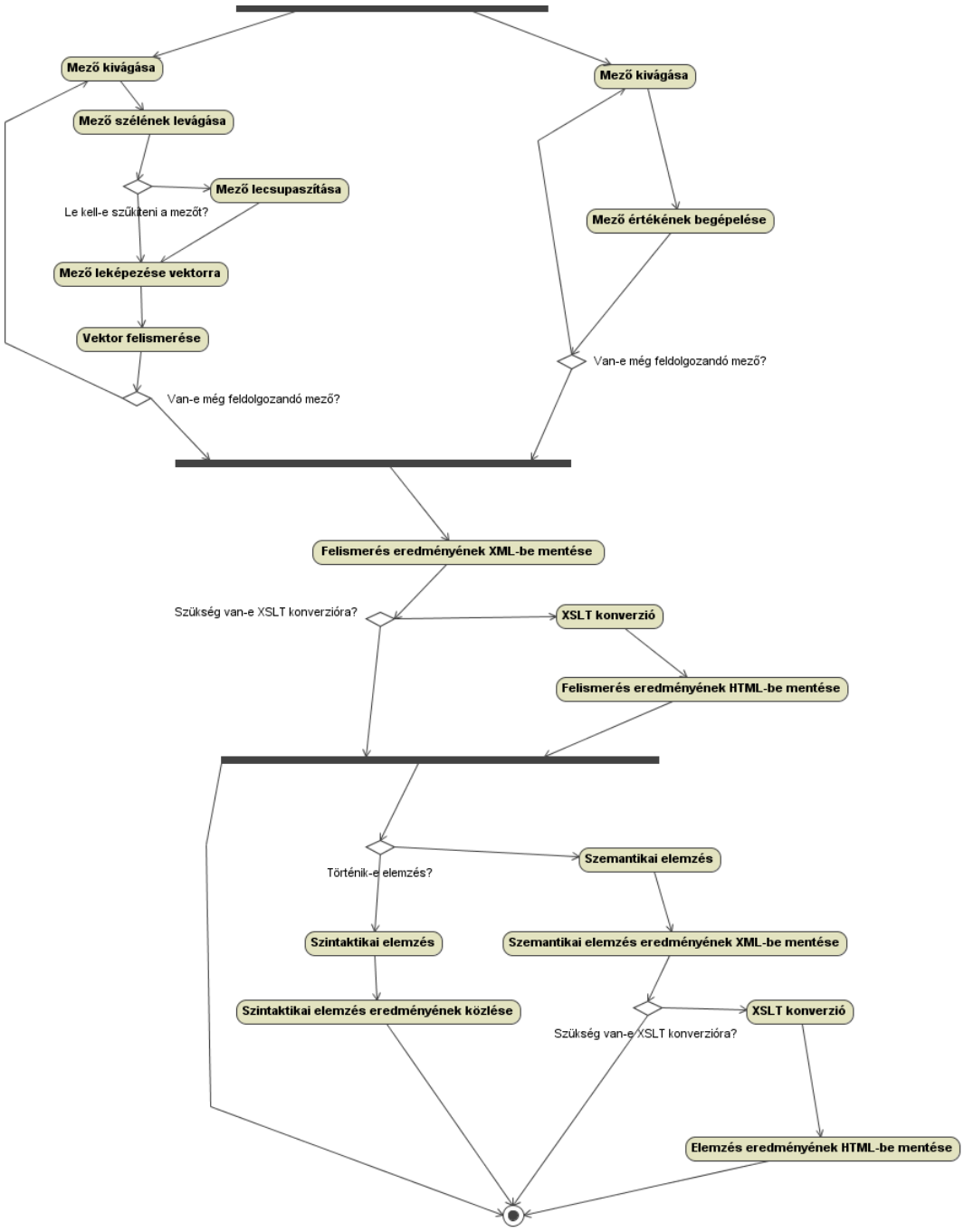
B.1. ábra. Az osztály diagram felső része



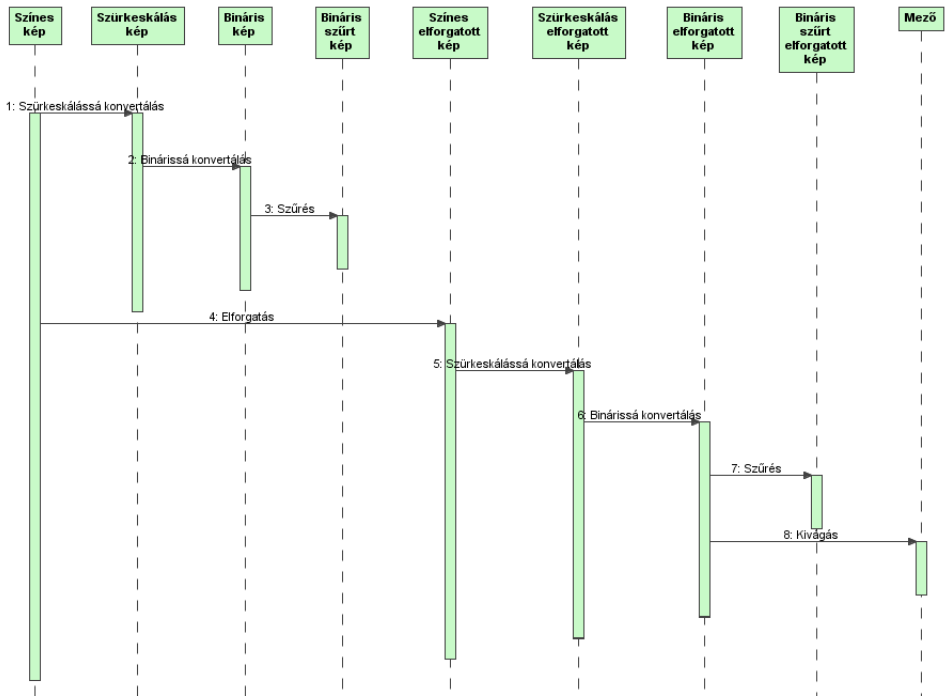
B.2. ábra. Az osztály diagram alsó része



B.3. ábra. Az aktivitás diagram felső része

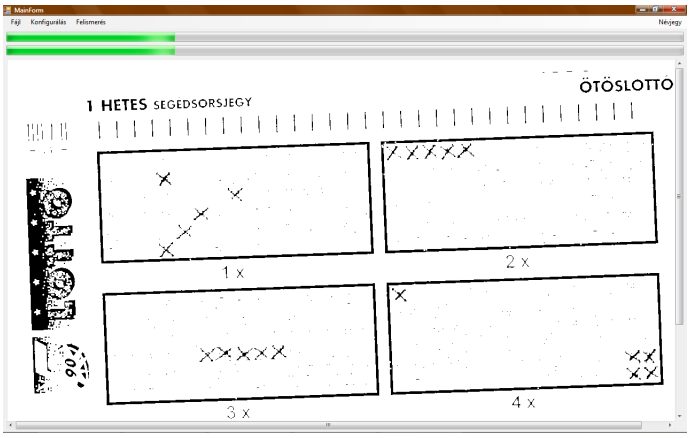


B.4. ábra. Az aktivitás diagram alsó része

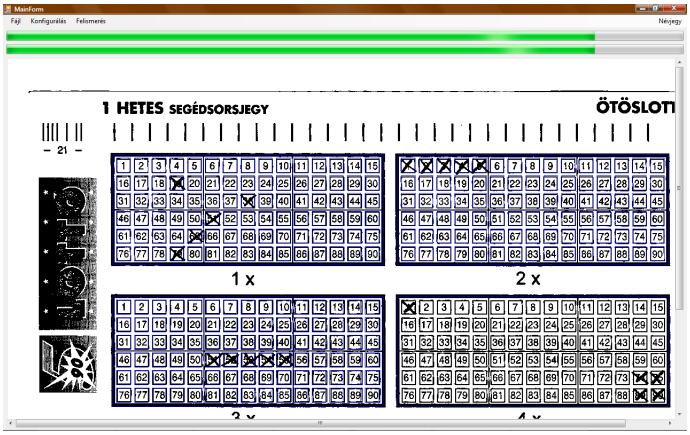


B.5. ábra. A szekvencia diagram

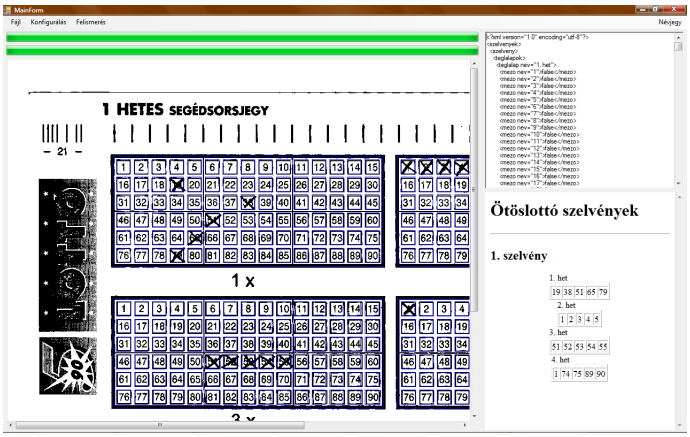
C. KÉPERNYŐKÉPEK A PROGRAM FUTÁSÁRÓL



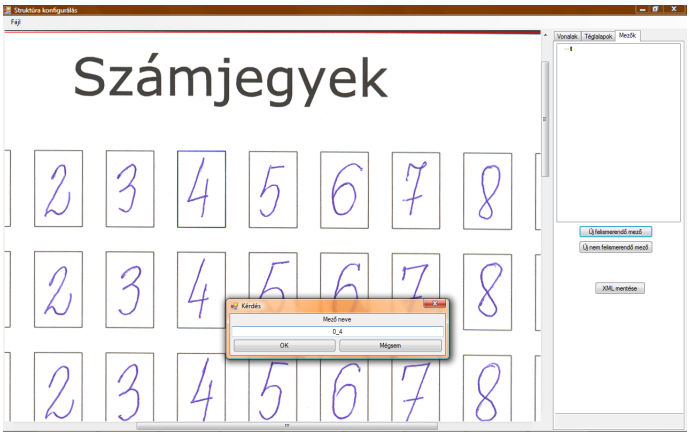
C.1. ábra. Egyenes keresése a szűrt képen



C.2. ábra. Mezők felismerése a forgatott képen



C.3. ábra. A felismerés végén a képernyő jobb szélén megjelenik a felismerés eredménye XML és HTML formátumban



C.4. ábra. Mező helyének meghatározása a struktúra konfigurációs ablakban