EÖTVÖS LORÁND TUDOMÁNYEGYETEM
INFORMATIKAI KAR

Topográfiai térképek automatikus georeferálása OpenCV-vel

DIPLOMAMUNKA
TÉRKÉPÉSZ MESTERSZAK

Készítette:
Varga Lola

Témavezető:
Dr. Gede Mátyás
egyetemi docens
ELTE IK Térképtudományi és Geoinformatikai Intézet

Budapest, 2021

EÖTVÖS LORÁND TUDOMÁNYEGYETEM

INFORMATIKAI KAR

TÉRKÉPTUDOMÁNYI ÉS GEOINFORMATIKAI TANSZÉK

DIPLOMAMUNKA TÉMABEJELENTŐ

Hallgató adatai:

Név: Varga Lola

Neptun kód: DG12LU

Képzési adatok:

Szak: térképész, mesterképzés (MA/MSc)

Tagozat: Nappali

Belső témavezetővel rendelkezem

Témavezető neve: *Dr. Gede Mátyás*

munkahelyének neve: *ELTE IK Térképtudományi és Geoinformatikai Tanszék*

munkahelyének címe: 1117, Budapest, Pázmány Péter sétány 1/A.

beosztás és iskolai végzettsége: egyetemi docens, PhD

A diplomamunka címe: Topográfiai térképek automatikus georeferálása OpenCV-vel

A diplomamunka témája:

(A témavezetővel konzultálva adja meg 1/2 - 1 oldal terjedelemben diplomamunka témájának leírását)

Szelvényezett topográfiai térképsorozatok szelvényeinek automatikus georeferálása. A térképek keretvonalainak detektálása, azok alapján a sarokpontok meghatározása.
A szelvényazonosító automatikus felismerése. Ezek alapján a georeferencia-adatok meghatározása, és a szelvény georeferálása. A feladat implementálása OpenCV/Python környezetben

Budapest, 2020.11.30.

Tartalomjegyzék

Bevezetés

A dolgozatom témája régi topográfiai térképek automatikus georeferálása a számítógépes látás eszközeivel. A cím számos kérdést vet fel, így ebben a tanulmányban részletesen be fogom mutatni a kitűzött célokat, ismertetem a szükséges elméleti hátteret, módszereket és problémákat, melyek megoldására újabb és újabb eszközöket hívtam segítségül.

Témaválasztásom indoklásának elsősorban leendő szakmámat jelölöm meg Térképészként, Geoinformatikusként fontos, hogy az általunk tanult módszereket felhasználjuk és új megoldásokkal gazdagítsuk a szakma szerteágazó területeit.

A rövid irodalmi áttekintést követően ismertetni fogom a felhasznált függvénykönyvtárakat, a képek beolvasására alkalmazott módszereimet, a szelvényeken lévő vonalak detektálását és azok közös metszéspontjainak kiszámítását. Kitekintést nyújtok a szövegfelismerő eszköz alkalmazására, valamint annak pontos definiálására.

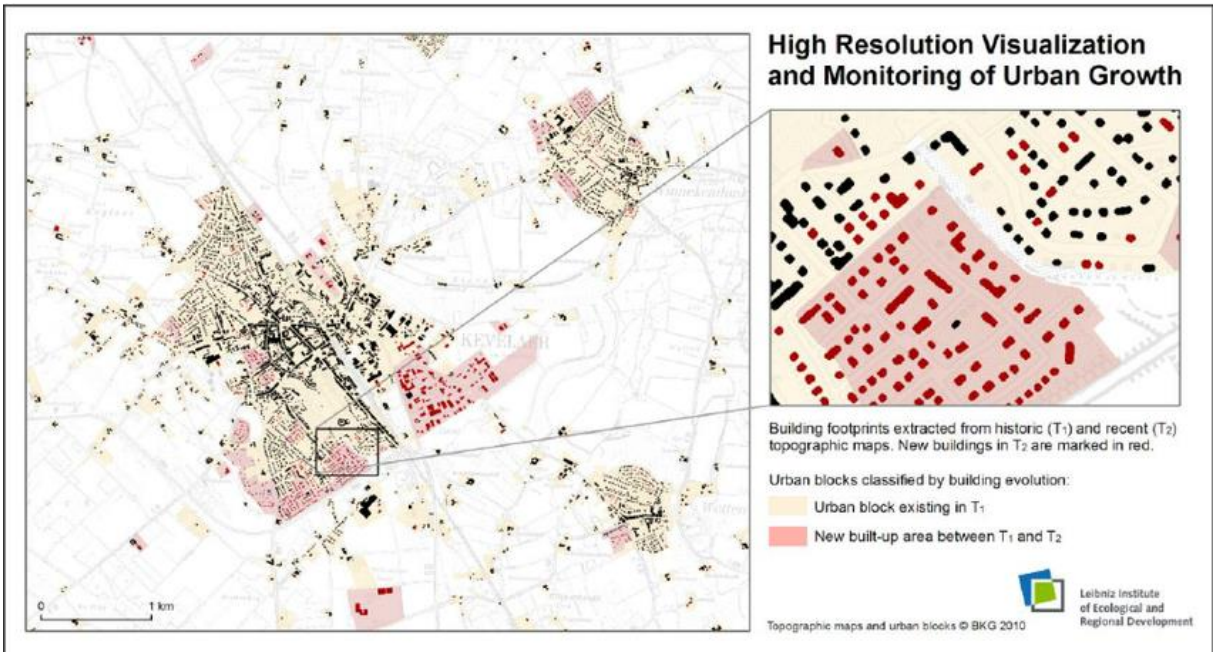
A georeferálás egy olyan folyamat, ami segítségével egy digitális képet – mely lehet egy légifelvétel, szkennelt térkép vagy akár egy műholdfelvétel is – valós térben helyezünk el. Ennek segítségével a felhasználó képes a kép minden egyes pixelét elhelyezni a Föld felszínén, ezáltal bármilyen térinformatikai szoftverben felismerhető és használható lesz a végtermék. Az ilyen végtermékek, a kép pixeleihez társult földrajzi adatokat tárolják. Kezdetben számunkra a raszteres kép pontjainak csak a kép síkkoordinátarendszerében érvényes pixelkoordinátái ismertek, ebben a rendszerben a kép bal felső sarka (0,0) és minden képpont 1-1 növekményt jelent. Ezekhez társítunk illesztőpontokat (GCP: Ground Control Point) majd adjuk át az előzőekben ismertetett pixelkoordinátákat és a térképi koordinátákat (Dr Timár 2008). A GeoPDF, GeoTIFF formátum alkalmas a pixeladatok, georeferencia információk megőrzésére, amik lehetnek különböző koordinátarendszerek, ellipszoidok és minden olyan további adat, melyek szükségesek a fájl pontos térbeli hivatkozásának meghatározásához. Más képformátumok esetén ezeket az információkat kiegészítő fájlokban (többnyire egy vetület és egy úgynevezett „world” fájlban) tároljuk. A georeferálás folyamatának vizsgálatakor valójában négy lépésről beszélhetünk. Az első a kép beolvasása egy számunkra alkalmas szoftverben, ezt követi az illesztőpontok kijelölése a raszteres képen, amihez a harmadik lépés során koordináta párokat társítunk kellő pontossággal. Végezetül következik az ellenőrzés és a georeferálás mentése a már előbb említett formátumokban (ArcGIS.Pro 2017).

A dolgozatom célja Python környezetben, különböző nyílt forráskódú eszközök segítségével az előzőekben ismertetett négy fázis automatizálása. Ehhez nélkülözhetetlen meghatározni a számunkra lényeges négy sarokpontot az adott szelvényeken, melyekhez a későbbiekben földrajzi koordinátákat rendelhetünk, valamint a szelvényazonosító detektálása és beolvasása, hisz ezek ismeretében probléma nélkül véghezvihető a georeferálás.

A felhasznált eszközök között csak nyílt forráskódú könyvtárak találhatók: az OpenCV, amivel a keretvonalak detektálását végezzük, a Tesseract, ami szövegek felismerésére nyújt lehetőséget és a GDAL, mely a georeferálás folyamatához szükséges. A későbbiekben ezen könyvtárak működésére és használatára is ki fogok térni.

Irodalmi áttekintés

A témával kapcsolatban született néhány hasonló kutatás nemzetközi színtereken. I Rus és társai hasonló módszerrel georeferálták a Romániát lefedő Lambert-Cholesky féle 1 : 20 000-es méretarányú térképgyűjteményt Radon transzformáció segítségével (I Rus et al. 2010).
Mivel a képek sok zajt, zavaró pontot tartalmaznak, bizonyos helyekre végezték el a számításokat, a térkép széleinél, a vonalak detektálása érdekében mindezt adott paraméterekkel, ezzel csökkentve az algoritmus futási idejét. A radon számítás végeredménye egy mátrix különböző csúcsértékekkel, amikből már kivitelezhető a fokhálózati vonalak rekonstrukciója. Különböző finomítások és szűrések segítségével detektálták az illesztőpontokat melyeket a GDAL függvénykönyvtár segítségével, a szelvényazonosítóból származtatott koordinátákkal látták el.
Hendrik Herold és kollégái (2011) hasonló módszert alkalmaztak az épület vizualizációs eljárásokra egy városdinamikai adatbázis létrehozására. A módszer alkalmas a városfejlesztési tervek fejlődésének követésére, földterületek figyelemmel kísérésére, valamint település minták és városnövekedési modellek reprodukciójára (1. ábra).



1. ábra: Automatikus vektorizálás eredményének felhasználása tematikus térképként. (Herold, és társai. 2011)

A lett Jānis Jātnieks QGIS-ben létrehozott egy olyan plug-int, melynek segítségével könnyebben és gyorsabban véghezvihető a georeferálás. A MapSheetAutoGeoRef plug-in hátránya, hogy egy ponton még mindig igényel emberi beavatkozást, azonban a georeferálás

folyamatát számottevően gyorsítja. Működéséhez szükséges ismernünk a térkép vetületét és az adathalmaznak tartalmaznia kell a térképszelvények azonosítóját. A pixelek pontos helyzetét azonban kézzel kell kijelölni, ezek után pedig automatikusan rendel hozzá adatot, tölti be a következő szelvényt és nagyít rá az adott területekre. A plug-in nem változtatja meg az eredeti raszteres képet, azonban automatikusan kimenti azt nekünk egy GeoTIFF formátumban. (Jātnieks 2010)

Hazai szinten is foglalkoztak hasonló témával a képfeldolgozás különböző területein, azonban térképszelvények automatikus georeferálásáról eddig még nem esett szó.

Importált modulok

Az OpenCV egy olyan könyvtárcsomag, ami a számítógépes látást segíti elő. A könyvtár több mint 2500 optimalizált algoritmussal rendelkezik, amik számos felhasználási kört fednek le. Ilyenek például: arc és objektum felismerés, videókon való emberi mozgás analizálása, tárgyak osztályozása, pontfelhők készítése vagy akár videófelvételeken való különféle szemmozgások követése (Rosebrock 2016). Én Python környezetben használtam fel a különféle eszközöket, melyeket a későbbiekben részletesen be fogok mutatni, de akár C vagy C++ nyelveken is alkalmazhatóak. (Rosebrock 2016)

```
import cv2
import numpy as np
import math
import pytesseract
import re
import glob
import random
```

A fentiekben felsorolt modulok voltak a segítségemre a program megírásakor, melyeket a kódsorozat elején importáltam és fokozatosan használtam fel.

`cv2`	Történeti okok miatt az OpenCV Pythonban cv2 néven érhető el.
`numpy`	Numerical Python rövidítése. Olyan könyvtár, ami segítségével a tömbökkel és különböző mátrixokkal tudunk könnyedén számolni, valamint az OpenCV a képeket numpy tömbként tárolja el.
`math`	Matematikaifüggvények és értékek vannak eltárolva benne.
`pytesserct`	Optikai karakterfelismerő eszköz.
`re`	Regular Expression (Regex) modul. Segítségével könnyedén kereshetünk összetett mintáknak megfelelő részleteket szövegekben.

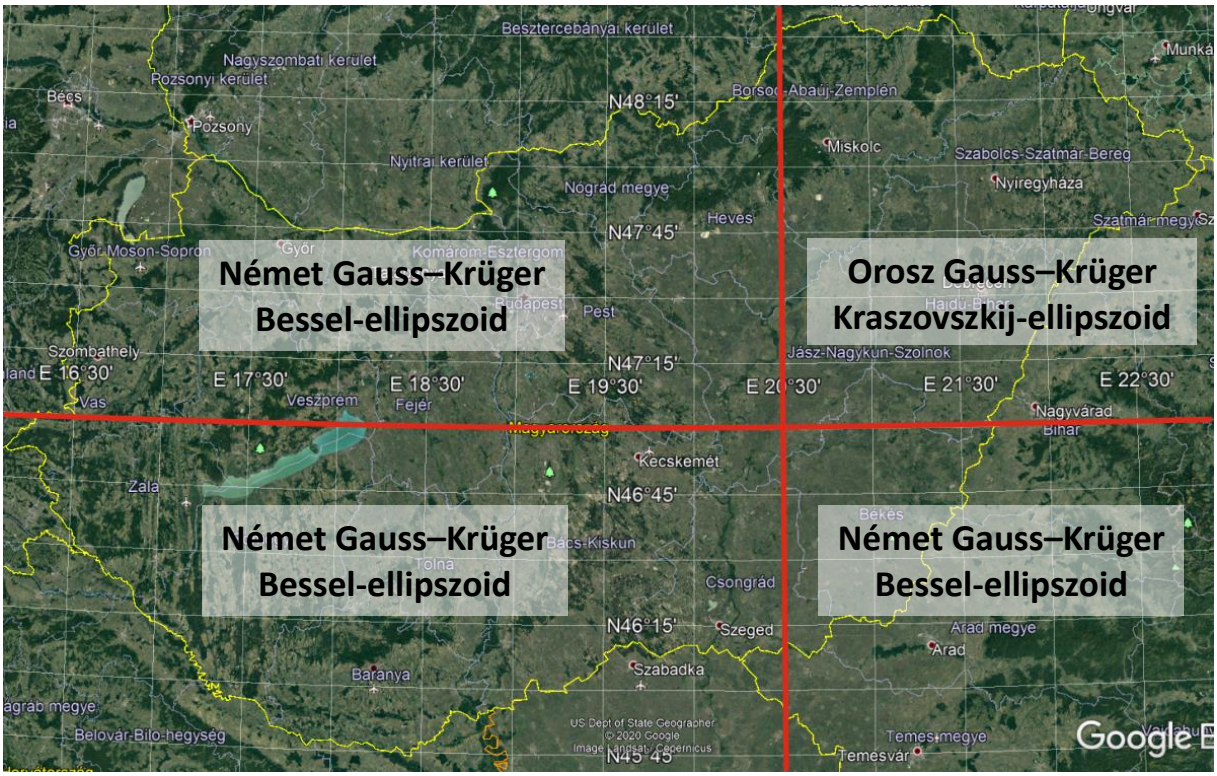
`glob`	A fájlnevek beolvasását segíti.
`random`	A véletlenszámokat generáló random() függvény miatt van rá szükség.

Feldolgozott szelvények

A feldolgozandó térképek a második világháború utáni gyorshelyesbítés szelvényei (1950-52). Az ötvenes években a topográfiai térképészet szovjet behatás alá került és az új felméréshez a harmadik katonai felmérés (1869-1887) szelvényeit vették alapul. (Jankó 2007) A cél, hogy a meglévő alaptérképeket Gauss–Krüger vetületi rendszerbe helyezzék, valamint a nemzetközi szelvényezésnek megfelelő struktúrát alakítsanak ki.

A felmérési méretarány 1 : 25 000 és a technikai paraméterek is változatlanok maradtak, így csak tartalmilag kezdték helyesbíteni a meglévő térképeket (Nagy 1985). Új jelkulcs született az adriai tengertől számított magaságokkal. Szinvonalak alapszintköze 10 m.

A gyorshelyesbítés első két évében a szelvényeket Bessel-féle ellipszoidra vonatkoztatták, így amikor az utolsó évben megkísérelték őket egymáshoz illeszteni a Szovjet rendszerben használt Kraszovszkij-féle ellipszoidra, nem sikerült. A hibát okozó fő faktor, hogy a szovjet szakértő nem adta át az üzenetet a magyar félnek a változásról. A λ=20°30’ hosszúsági körtől keletre és a φ=46°40’ szélességi körtől északra eső területen így már a Kraszovszkij-ellipszoidot használták alapfelületként. (Zentai , Topográfiai térképek előadás diái 2018)



2. ábra: Alapfelületek változása a II. világháború utáni gyorshelyesbítés alatt

Programkód bemutatása

Main

```
def main():

    lista=[]
    mappa = "C:/Lola/szelvenyek/*"

    for nev in glob.glob(mappa):
        lista.append(nev)

    random.shuffle(lista)
    of=open("valami.csv", 'w', encoding="utf-8")
    for i in lista:
        MapSheet_processing(i,of)
    of.close()
    print (len(hiba))

main()
```

A `main()` függvény tartalmazza a főprogramot, ahol a `MapSheet_processing` függvény egy for ciklus segítségével hívódik meg, a `MapSheet_processing` pedig tartalmazza a képbeolvasást annak újra méretezését, a fekete szín leválogatását, az éldetektálást, a szövegfelismerést, a metszéspontszámítást, a finomítási munkálatokat és mindezek kirajzolását valamint a fájlba írást, melyen részeket egyesével fogok kifejteni a dolgozatom későbbi fejezeteiben.

Itt látható, hogy a térképszelvények fájlneveit szedi ki majd keveri össze véletlenszerű módon. Az utóbbira a fejlesztés alatt azért volt szükség, hogy minden indításkor más szelvényekre fusson le a program így több, különféle hibák és problémák jöhettek elő.

MapSheet_processing függvény

A `MapSheet_processing` függvényben történik a program egy jelentős része, melyet most kifejtek részletesen. Fontos, hogy két paramétert vár el, a képnevet (`kepnev`) és az output fájl objektumot (`of`):

Legelső fázisa maga a kép beolvasása és a képek magasságának és szélességének lekérése a későbbi formázásokhoz.

```
kep=cv2.imread(kepnev)
magassag, szelesseg, csatorna=kep.shape
```

Fekete szűrése

Minden három csatornás kép, aminek az alapegységei a pixelek, három színből épülnek fel, így egy képpont információtartalmát annak értékeiből nyerhetjük ki. Az RGB vagyis a piros, zöld, kék színskála intenzitás értéke adja meg egy adott pixel értékét, így a továbbiakban ezzel az összefüggéssel dolgoztam. A három színt 8 bites számokkal jelöljük, így ezek az értékek 0–255-ig terjedhetnek. Ha mindhárom csatorna 255-et vesz fel fehér színt kapunk és ha ezek az értékek 0-t , akkor feketét. Azonban a valóságban az emberi szemnek a sötét szürke is feketének tűnik, így nem csak a 0 értéket felvevő pixelekkel dolgoztam, hanem a közel fekete színeket is leválogattam.

Az alsó sorokban látható a kód, aminek a segítségével meghatároztam mi az az érték, ami még feketének látszik. Erre azért volt szükség, mert a későbbiekben a keretvonalak meghatározása, azok metszéspontjainak kiszámítása a lényegi, melyek a térképszelvényeken feketék vagy közel fekete színűek.

Először szétválasztottam a csatornákat, majd azok egymás közti különbségét, viszonyát határoztam meg, hogy milyen intervallumban vizsgáljuk a színeket a térképszelvényeken. Az utolsó sorban látszódik a `mask` ami a már előzetesen kiszűrt értékeket mutatja, az általam definiált módon. A mask egy egybites kép, aminek a lényege, hogy minden olyan dolgot amire nincs szükségünk a képen, kiszűrje (pixel értéke 0) és csakis az adott paraméterekkel rendelkező tehát csak feketének minősített pixeleket jelenítse meg (pixelérték 1) (Rosebrock 2016).

```
ch=cv2.split(kep)
d1=abs(cv2.absdiff(ch[0],ch[1]))
d2=abs(cv2.absdiff(ch[0],ch[2]))
m1=cv2.inRange(ch[0],0,150)
m2=cv2.inRange(d1,0,50)
m3=cv2.inRange(d2,0,50)
mask=cv2.bitwise_and(cv2.bitwise_and(m1,m2),m3)
```

3.ábra: A fekete színszűrés eredménye

Egyenes vonalak detektálása

Ahogy a 3. ábrán látható, sikeresen kiszűrtem a fekete színt (minden, ami fehér most az ábrán) a képről. Nem szabad elfelejteni, hogy az eredeti cél a sarokpontok keresése és meghatározása, amiket a keretvonalak metszéspontjaiból kapunk meg. Így egyértelművé vált, hogy csak a bizonyos hosszúságú egyenes vonalak megtalálása a fontos, a többi felesleges zaj elhanyagolható.

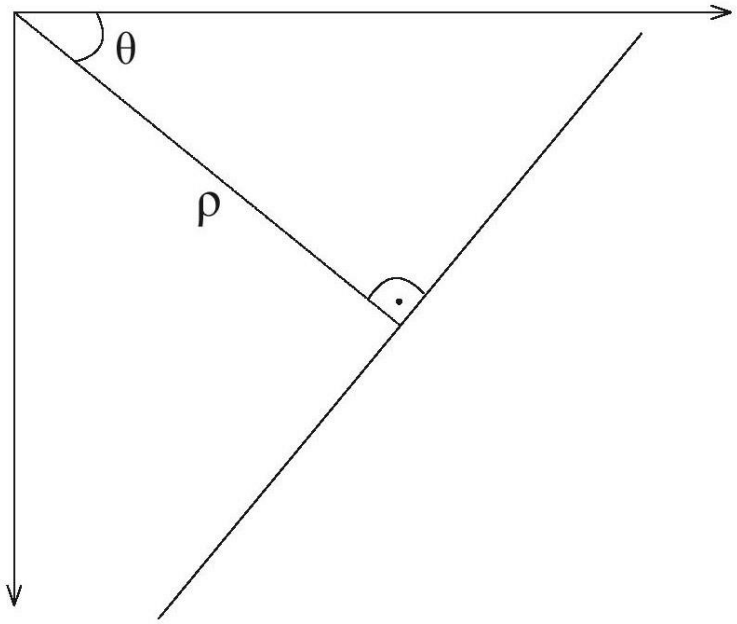
A vonaldetektálásra a leggyakrabban használt algoritmus a Hough transzformáció.

Működése:

A térkép különböző helyein, különböző irányban álló egyenesekhez tartozó pontok számát vizsgálja. Minél több pont tartozik az adott egyeneshez, annál valószínűbb, hogy ott lesz a keresendő vonal. Az algoritmus végig nézi az összes lehetséges egyenest, hogy azok vajon kitesznek-e egy vonalat, azonban ez a művelet végtelen számú lehetőséget jelent, így az általam megadott paraméterekkel teszi ezt meg, a folyamat redukálása érdekében.

A középiskolában jól begyakorolt megoldás, az egyenes egyenlete ($y = Ax + B$), az esetünkben nem alkalmazható, ugyanis az függőlegesek számítására nem alkalmas, azonban egy egyenes egyenlete felírható az Origótól (esetünkben ez a bal felső sarok) vett távolsággal és a bezárt szög nagyságával is (Tanács 2018-2021).

$\rho = xcos\theta + ysin\theta$, ahol ρ a távolságot adja meg, a θ pedig a merőleges és a vízszintes tengely által bezárt szöget óramutató járásával ellentétes irányban (Dimitri van Heesch 2021). A 4. ábrán mindez jól látható.



4. ábra: Vonal iránya és helyzete polárkoodinátákkal kifejezve

Az OpenCV-ben létezik erre az algoritmusra két függvény:

HoughLine és HoughLineP-t (P=Probabilistic). Az utóbbi az előbbi továbbfejlesztése, ami optimalizálja a számításokat, ennek érdekében gyorsabban megy végbe a folyamat. (Dharra 2019)

Különböző paraméterek megszabásával meghatározza az adott hosszúságú egyeneseket.

lines=cv.HoughLinesP(image, rho, theta, threshold[, lines[, minLineLength[, maxLineGap]]])

Az én esetemben ezek az értékek:

```
lines=cv2.HoughLinesP(mask,
                      rho=1,
                      theta=np.pi/180/3,
                      minLineLength=szelesseg/3,
                      maxLineGap=10,
                      threshold=100
                      )
```

A fenti értékek a Hough transzformáció egyeneseinek változásának a mértékét szabják meg. Azért fontos a paraméterek megfelelő beállítása, hogy minden keretvonalat megtaláljon, de lehetőleg más, rövidebb egyenesből minél kevesebbet.

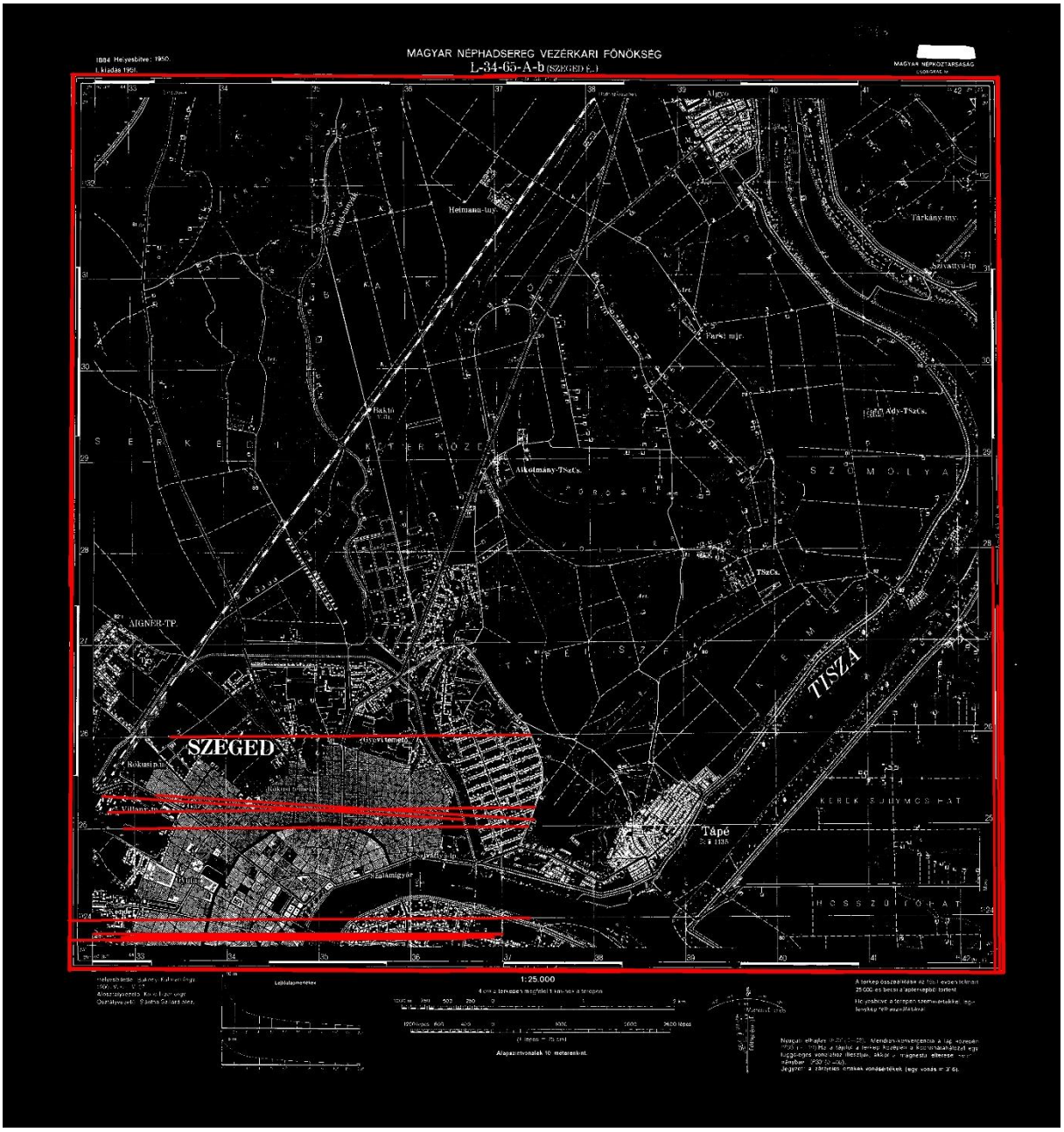
A vízszintes és függőleges vonalak leválasztása

```
for l in lines:
    (x1, y1, x2, y2) = l[0]
    alfa=math.atan2((y2-y1),(x2-x1))
    if (-0.1)<alfa<(0.1) or (math.pi-0.1)<alfa<(math.pi+0.1):
        koordinatak.append((x1,x2,y1,y2))
        vizsz.append((x1,y1,x2,y2))
        vro.append(rho(x1,y1,x2,y2))
    elif -math.pi/2-0.1<alfa<-math.pi/2+0.1 or math.pi/2-0.1
<alfa<math.pi/2+0.1:
        if .01*szelesseg<rho(x1,y1,x2,y2)<.99*szelesseg:
            koordinatak.append((x1, x2, y1, y2))
            fugg.append((x1,y1,x2,y2))
            fro.append(rho(x1,y1,x2,y2))
```

A fenti kódsorozat azt mutatja meg, hogy milyen módon válogattam le és adtam külön listákhoz a vízszintes és függőleges vonalakat. Ehhez végig mentem egy for ciklussal a már előzőekben (HoughLine transzformáció) bemutatott `lines`-on (a vonalat alkotó pontokon). Lekértem négy koordinátát, amik meghatároznak egy egyenest, ezekután pedig az alfa dőlésszöget számítottam ki az atan2 függvénnyel, ami megadja egy vektor irányszögét. Ha egy vonal vízszintes, akkor a program beleteszi a két végpontjának koordinátáit a `vizsz`. listába. Azonban látható, hogy valójában 2x2 listát, hoztam létre, először a közel vízszintes vonalaknak, majd a közel függőleges vonalaknak. A cél, hogy a megtalált függőleges és vízszintes vonalak közül csak a legszélsőt találja meg mind a négy oldalon, hisz az lesz maga a keretvonal, így a másik két lista létrehozása is szükséges volt, amikben már a vonalakhoz tartozó ρ érték is tárolásra került. A függőleges vonalak esetében a ρ a balszéltől, a vízszintes vonalaknál pedig a felső széltől való távolságot mondja meg. Ezek közül kiválasztva a legkisebb, a legnagyobb, „legjobboldalibb”

és „legbaloldalibb” vonalat megkapjuk a keretvonalakat. A lenti képlet volt a segítségemre mindezek meghatározására.

```
def rho(x1,y1,x2,y2):
    l = math.sqrt((y2 - y1) ** 2 + (x2 - x1) ** 2)
    return abs(x2 * y1 - y2 * x1) / l
```



5. ábra: Vonaldetektálás eredménye

Legszélső vonalak keresése

6. ábra: Keretvonalak kiemelése a vonaldetektálás után

```
# balszélső vonal keresése
bal=fro.argmin()
(bx1,by1,bx2,by2)=fugg[bal]
cv2.line(kep, (bx1, by1), (bx2, by2), (0, 255, 255), 6)

# jobb szélső vonal keresése
jobb=fro.argmax()
(jx1,jy1,jx2,jy2)=fugg[jobb]
cv2.line(kep, (jx1, jy1), (jx2, jy2), (0, 255, 255), 6)

# felső vonal keresése
```

```
fent=vro.argmin()
(fx1,fy1,fx2,fy2)=vizsz[fent]
cv2.line(kep, (fx1, fy1), (fx2, fy2), (0, 255, 255), 6)

# alsó vonal
lent=vro.argmax()
(lx1,ly1,lx2,ly2)=vizsz[lent]
cv2.line(kep, (lx1, ly1), (lx2, ly2), (0, 255, 255), 6)
```

A fenti kódon látható, hogy a már listába tett és numpy tömbbé alakított vonalak közül miként válogattam le a keretvonalakat. A numpy.argmax() a tömb maximum elemének indexeit adja vissza, amit hasonló módon vittem véghez a maradék három oldalon is.
A cv2.line szintén egy beépített funkciója az OpenCV-nek, ami a képen kirajzolja (az adott szelvényen) két koordinátapár közt `(bx1, by1), (bx2, by2)` a vonalat, az általam megadott színnel `(0, 255, 255)` és vastagsággal `(6)`.

Külső metszéspont keresés

Két vonal metszéspontjának számításához egy függvény deklaráltam, amit a `MapSheet_processing` függvényben hívtam meg a sarokpont keresés alkalmával.

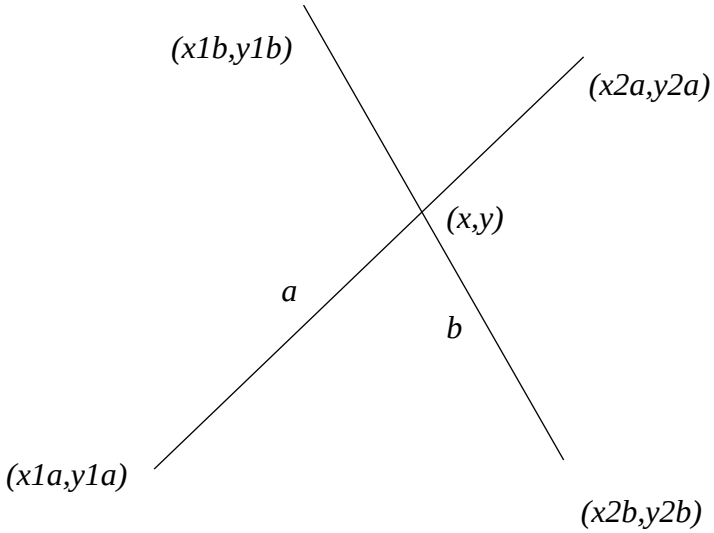
```
def intersection(la,lb):
    (x1a,y1a,x2a,y2a)=la
    (x1b,y1b,x2b,y2b)=lb
    # line steepness
    ma=(y2a-y1a)/(x2a-x1a) if x1a!=x2a and (x2a-x1a)!=0 else None
    mb=(y2b-y1b)/(x2b-x1b) if x1b!=x2b and (x2b-x1b)!=0 else None
    if (ma==mb):
        # lines are parallel
        return None
    if (x2a==x1a):
        # "la" vertical
        x=x1a
    elif (x2b==x1b):
        # "lb" vertical
        x=x1b
    else:
        x=(y1b-mb*x1b-y1a+ma*x1a)/(ma-mb)
    y=y1a+ma*(x-x1a) if ma is not None else y1b+mb*(x-x1b)
    return (int(x),int(y))
```

Ahogy azt a fentiekben látható, két vonalat kap meg a függvény és ezen vonalak metszéspontját számolja ki. Tehát a vonalak egymáshoz viszonyított helyzetét kapjuk meg. Ha ezek párhuzamosak, akkor értelemszerűen nincs közös metszéspontjuk. Ha merőlegesek egymásra,

akkor tudjuk, hogy melyik vonal szerint merőlegesek, ez esetben tudjuk mi az első koordinátája. Ha pedig egyik feltétel sem teljesül alkalmazzuk a lenti képletet.

```
x=(y1b-mb*x1b-y1a+ma*x1a)/(ma-mb)
y=y1a+ma*(x-x1a) if ma is not None else y1b+mb*(x-x1b)
```

Segítségül rajzoltam egy ábrát két vonal metszéspontjának matematikai számításához és levezettem, miként jött ki a felhasznált képlet.



7. ábra: Két vonal metszéspontjának kiszámítása

$$mb = \frac{y2b - y1b}{x2b - x1b}$$

$$ma = \frac{y2a - y1a}{x2a - x1a}$$

$$y - y1a = (x - x1a)ma$$

$$y = y1a + (x - x1a)ma$$

$$y = y1b + (x - x1b)mb$$

$$y1a + (x - x1a)ma = y1b + (x - x1b)mb$$

$$x(ma - mb) = y1b - y1a - x1b * mb + x1a * ma$$

$$x = \frac{y1b - y1a - x1b * mb + x1a * ma}{ma - mb}$$

```
# bf sarok
bf=intersection(vizsz[fent],fugg[bal])
cv2.circle(kep, (bf), 8, (255,0,128), 2)

# jf sarok
jf=intersection(vizsz[fent],fugg[jobb])
cv2.circle(kep, (jf), 8, (255,0,128), 2)

# bl sarok
bl=intersection(vizsz[lent],fugg[bal])
cv2.circle(kep, (bl), 8, (255,0,128), 2)

# jl sarok
jl=intersection(vizsz[lent],fugg[jobb])
cv2.circle(kep, (jl), 8, (255,0,128), 2)
```

A fenti kód segítségével meghatároztam a sarokpontokat, a keretvonalak metszéspontjának kiszámításával. Az előzőekben bemutatott cv2.line-hoz hasonlóan, a cv2.circle a körök kirajzolásáért felelős.

Belső metszéspontok keresése

Az előző oldalakban bemutatott munka során sikeresen leválogattuk a fekete színt, megtaláltuk a vonalakat, meghatároztuk mi számít függőlegesnek és mi vízszintes vonalnak. A továbbiakban ezeken metszéspontot kerestünk majd a sarokpontok koordinátáira is fény derült. Azonban a georeferáláshoz nem a külső keretvonalak metszéspontja szükséges, hanem a térképtükör belső sarokpontjai. Mivel a szelvények Gauss–Krüger vetületű szabványtérképek, ezért tudni, hogy a szabványméret szerint a belső és a külső keretvonal közt pontosan 1 cm különbség van. Ezt átszámolva 371 pixelkoordinátát jelent, levezetve: 1 : 25 000-hez méretarányú szelvényeken:

$$5' = \left(\frac{5}{60}\right)\left(\frac{\pi}{180}\right) 6373\ km = 9269\ m$$

$$\frac{9269}{25000} = 0{,}371\ m = 371\ mm$$

ahol a Föld sugara 6373 km és ehhez a $2 * 10\ mm$ vastag keret adódik, így $391\ mm$ lesz az eredmény.

```
tenMm = int((ly2-fy2)/391*10)

cv2.circle(kep, (jf[0]-tenMm,jf[1]+tenMm), 8, (255,0,128), 2)
cv2.circle(kep, (bf[0]+tenMm,bf[1]+tenMm), 8, (255,0,128), 2)
cv2.circle(kep, (jl[0]-tenMm,jl[1]-tenMm), 8, (255,0,128), 2)
cv2.circle(kep, (bl[0]+tenMm,bl[1]-tenMm), 8, (255,0,128), 2)
```

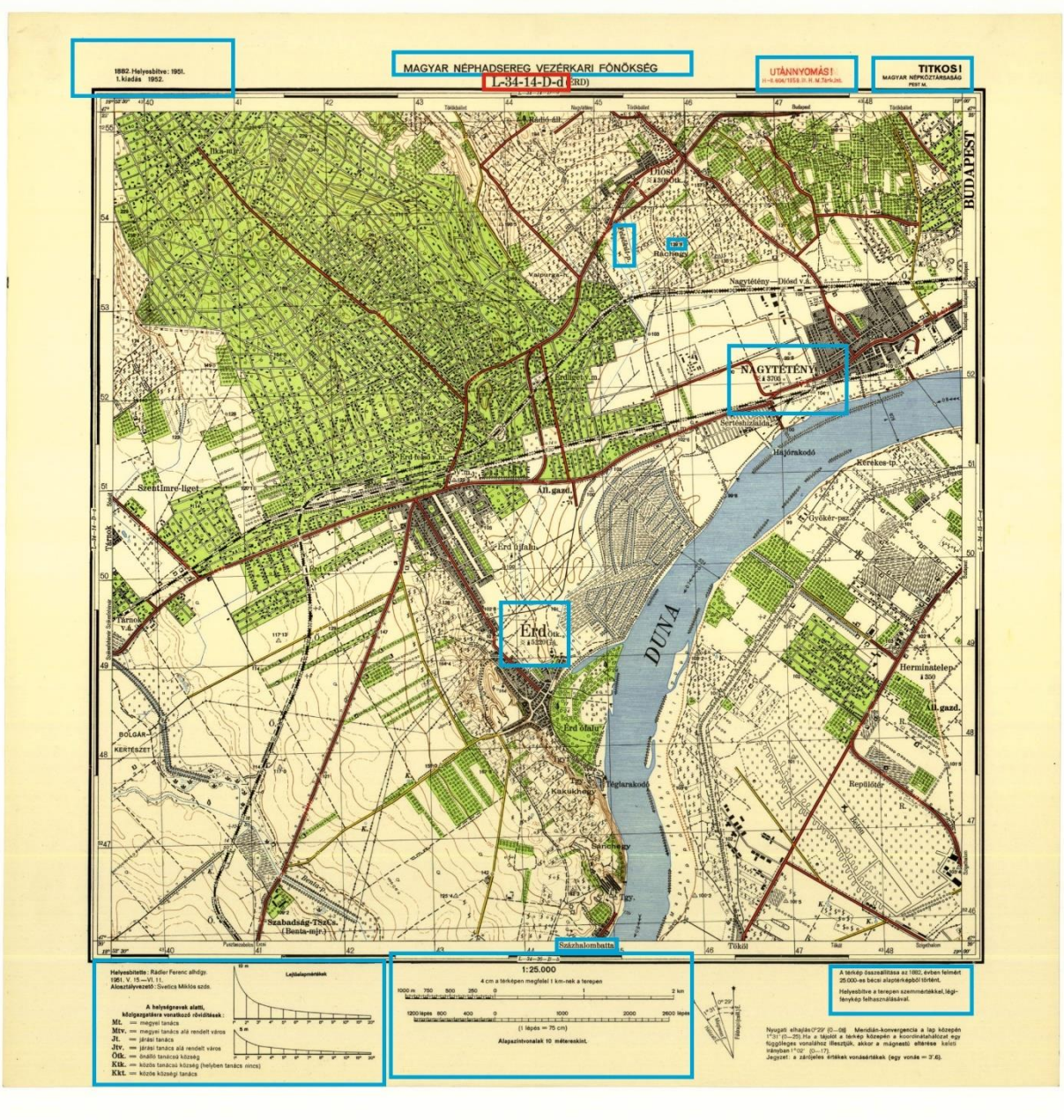
A 13. ábrán lila körrel jelöltem a megtalált sarokpontot.

OCR

Az OCR (Optical Character Recognition), olyan optikai karakterfelismerést elősegítő rendszer, ami alkalmas kétdimenziós képek szöveges tartalmát gépi szöveggé alakítani. Többlépcsős folyamatról beszélünk, mely során a cél egy képről kinyert digitális karakterek detektálása a lehető legpontosabb módon.
Főbb folyamatok:

- A képi előfeldolgozás
- Szöveg lokalizálás
- Karakter tagolás
- Karakterfelismerés
- Utómunkálatok (Zelic és Sable 2021)

A képi előfeldolgozás során, a képet beolvassuk, esetünkben Python környezetben. A következő művelet az általunk keresett szöveg hozzávetőleges helyzetének meghatározása, hisz minél inkább szűkítjük a keresett karaktereket tartalmazó terület méretét, annál gyorsabb és hatékonyabb lesz későbbiekben a szövegfelismerés. Egy térképszelvény esetén a számunkra fontos információ a fejlécnél, a szelvényazonosító körüli terület. Ha a keresési területet nem szűkítenénk a szelvény ezen kis részére, felesleges karaktereket is felismerne, például méretarányt, a mértékléc értékeit, a készítés és kiadás dátumát stb. Ráadásul az egész képen szövegeket detektálni nagyságrendekkel hosszabb ideig tartana, mint egy kis kivágott területen.



8. ábra: OCR számára felismerhető szövegek a szelvényen.

A 8. ábrán bejelöltem néhány helyet (kék színnel), ahol tartalmaz a szelvény szöveget, melyeket felismer az OCR – valószínűleg sok hibával, hisz nagy felületen nehezebb a pontos karakterdetektálás. Számunkra a piros kerettel határolt terület a lényeges, ugyanis a szelvényazonosító ismeretében meghatározhatjuk a szelvény sarokpontjainak koordinátáit.

A lenti sorokban bemutatom, hogy milyen eredményt kapunk amikor az egész szelvényre nézve keresi az OCR a karaktereket és nem egy megadott határolódobozban, valamint látható, hogy a szöveg felismerhetetlen, így a következő fejezetekben ennek kiküszöbölésére fogok kitérni.

iTITKOS

igen:0MAGYARelieerrFONOKSEG__UTANNYOMAS!woveseen

…

Alosztalyvezetd:SveticsMiklésszds.Wooom750200250peeSreJHetyesbave:arenesszemmeteki

etlagi-

aBechtaNex,Bi

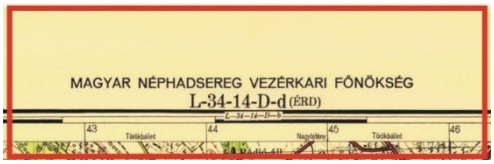
Tesseract

Számos karakterfelismerő eszközt ismerünk (OCRopus, Ocular OCR, SwiftOCR), melyek közül én a Tesseract OCR-t hívtam segítségül.
A Tesseract egy Ph.D kutatási projektnek indult Bristolban, azonban mára már egyre népszerűbb az OCR fejlesztők körében számos programozási nyelvvel való kompatibilitása miatt. 2005-től már nyílt forráskódú szoftverré nőtte ki magát, amit azóta a Google fejleszt (Zelic és Sable 2021). Működése neurális hálókon alapul.

A Pytesseract a TesseractOCR egy csomagja, aminek a legfőbb előnye, hogy szövegeket ismer fel képeken és emellett támogatja a főbb képtípusokat is: jpeg, png, gif, bmp, tiff. (Zelic és Sable 2021). Ahhoz, hogy hiba nélkül funkcionáljon, a szükséges előkészítéseket végre kell hajtanunk (átméretezés, binárissá alakítás, zajeltávolítás).

```
txt = pytesseract.image_to_string(kep[0:int(magassag * 0.1),
int(szelesseg * 0.4):int(szelesseg * 0.6)],
                                  config="--psm 6")
```

A felső két sor kódrészlet mutatja meg az általam beállított paramétereket. pytesseract.image_to_string, vár egy képet, ahol megadtam egy határoló dobozt, amire tekintve végezze el a karakterfelismerési műveleteket és egy szegmentálási kódot[1], hogy tudja milyen betűkről is van szó. Az esetemben a kép maga az adott térképszelvény, a határoló doboz pedig egyik részről a kép felső egytized része, másrészről a képszélesség 40-60%-ig tartó középső területe (9. ábra).



YARNEPHADSEREGVEZERKARIFONOKSEG

L-34-14-D-d(éRp)

SS———_

Tetokbalint44aTordkbalint6

9. ábra: A Pytesseract számára kivágott képrészlet (fent). Szövegfelismerés eredménye (lent).

[1] https://github.com/tesseract-ocr/tesseract/issues/434

Az eredmény egy tagolatlan szöveg minden, amit az adott kivágaton az OCR felismert. A lenti sorokban megfigyelhető, hogy a szövegfelismerés nem elegendő a szelvényazonosító detektálásához és további felhasználásához.

RegEx

A RegEx (Regular Expressions) egy kiegészítő modul Python környezetben, aminek a segítségével definiálhatunk különböző rendszereket, szabályokat és karakterláncokat. Ezen kívül alkalmas ezen karakterláncok módosítására és szétválasztására is egyaránt. (Kuchling 2001-2020) (Python Software Fundation 2021)

```
m = re.search('[LM1.]{1,2}-[0-9]{1,3}-[0-9]{1,3}-[A-D0]-[a-
dD¢8BeC]', txt)
```

Ebben a sorban egy olyan regexet alkalmazok, ami során megadom, hogy milyen karaktereket keressen a már előzőekben definiált területen. Mivel 1: 25 000-es méretarányú Gauss–Krüger szelvényeket dolgoztam fel, így Magyarországot tekintve két zónát tartottam különösen fontosnak az azonosítók detektálására, az L és M zónákat. Ezért kifejezetten azokat a karaktereket kerestem, de sokszor előfordult, hogy az L-t 1-nek érzékelte, vagy éppen a jobb „szárát” pontnak, ezért tettem ezeket is a regular expression-be. A regex következő része: `[0-9]{1,3}`egy 0-tól 9-ig terjedő szám ami lehet 1, 2 vagy 3 számjegyű. Ez adja meg a szelvényazonosító következő részét, ami az országunkra nézve lehet 33 vagy 34. A következő számra ugyanezt az eljárást alkalmaztam, majd a betűkkel is elvégzem ezt a szűrést, ahol `[A-D]` megmutatja, hogy A, B, C vagy D karaktert kell találnia az azonosító detektálása során. Az utolsó betű meghatározásához kisebb változtatásokat tettem bele az előzőekéhez képest, ugyanis ott olyan karaktereket is felismert, amik nem lehetnek egy térképszelvény azonosítójában. Mivel a c-t sokszor ¢-nek ismerte fel, d-t 8-nak, c helyett e stb., ezért elhelyeztem ezeket kitételként, miszerint ezek is elfogadhatóak.

```
sheetId=m.group(0) if m is not None else "error”

if sheetId == "error":
    print("SHEETID: ", m)
if sheetId[1] != "-":
    sheetId = sheetId[0] + sheetId[2:]
if "¢" in sheetId:
```

```
    sheetId = sheetId.replace('¢', 'c')
if sheetId[-1] == "D":
    sheetId = sheetId[:-1] + "b"
if sheetId[-1] == "C":
    sheetId = sheetId[:-1] + "c"
if sheetId[-1] == "8":
    sheetId = sheetId[:-1] + "a"
if sheetId[-1] == "B":
    sheetId = sheetId[:-1] + "b"
if sheetId[-1] == "e":
    sheetId = sheetId[:-1] + "b"
if sheetId[-3] == "0":
    sheetId = sheetId[:-3] + "C" + sheetId[-2:]
if sheetId[0] == "1":
    sheetId = "L" + sheetId[1:]
```

A fenti kódsorozatban látható, milyen féltételeket szabtam meg a továbbiakban, ha a szelvényazonosító detektálása nem lenne teljes mértékben hibátlan és mégis pontos kiíratást szeretnénk látni. Ezek a problémák merültek fel a legtöbbször a program futása alatt, amit itt, ily módon korrigáltam.

Finomítási munkálatok

Miután a lényegi rész valójában kész lett, kezdődhettek a finomítási munkálatok, melyek kiemelkedő hangsúlyt kaptak hisz az első körben számtalan hiba keletkezett. Ilyenek lehettek: az egyenesek nem pontos detektálása, az adott hosszúságú vonalak nem pontos kirajzolása, az OCR nem pontos szelvényazonosító értelmezése, metszéspontok elcsúszása. Ezek kiküszöbölésére különböző módszereket alkalmaztam, így a következő oldalakon kitekintést fogok nyújtani, miként finomítottam a sarokpontok helyét.

Metszéspontok finomítása

Az első és legfontosabb, a metszéspontok finomítása és pontosabb detektálása volt. Ennek érdekében kivágtam a térképszelvény négy sarka körüli területet, hogy kisebb felületen ezáltal pontosabban keressen metszéspontokat a program.
Ehhez egy finomít nevű függvényt hoztam létre, melynek paraméterei:
(kep, kozeppont, meret, fent, bal) ami az adott képen kivág a középpont körül egy adott méretű területet a kép adott sarkában, ami lehet fent bal, fent jobb, lent bal, lent jobb (11. ábra).

```
def finomit(kep, kozeppont, meret, fent, bal):
    meret=int(meret)
    kiskep = kep[kozeppont[1]-meret:kozeppont[1]+meret,kozeppont[0]-
meret:kozeppont[0]+meret].copy()
    cv2.imshow("szia",kiskep)
    cv2.waitKey(0)
    subimg = cv2.erode(kiskep, ek)
    if fent:
        K=Ktl if bal else Ktr
    else:
        K=Kbl if bal else Kbr
    M = cv2.filter2D(subimg, -1, K)
    M = M.flatten()
    if not fent:
        M=M[::-1] # megfordítjuk a tömböt (-1-esével ugrálunk)
    hely = M.argmax()
    if not fent:
        hely = len(M) - hely
    y=hely//(meret*2)
    x=hely%(meret*2)

    cv2.circle(subimg, (x, y), 8, (255, 0, 128), 2)
```

A bal felső sarok esetében a `mask` nevű képről (a kép, ami csak a fekete pixeleket tartalmazza a szelvényen), a középpont körül, ami jelen esetben maga a bal felső sarok pixelkoordinátája, 9 mm méretű képet vág ki (tenMm egy előre definiált változó: `tenMm = int((ly2-fy2)/391*10))`. Ugyanezt megcsináltam a maradék három sarokpont körül is (fent jobb esetén: True False, lent bal: False True és lent jobb esetén: False False értékekkel), így a kapott kis képeken pontosabb eredményt kaphatunk, hisz kisebb területeken könnyebb a metszéspont vizsgálat.

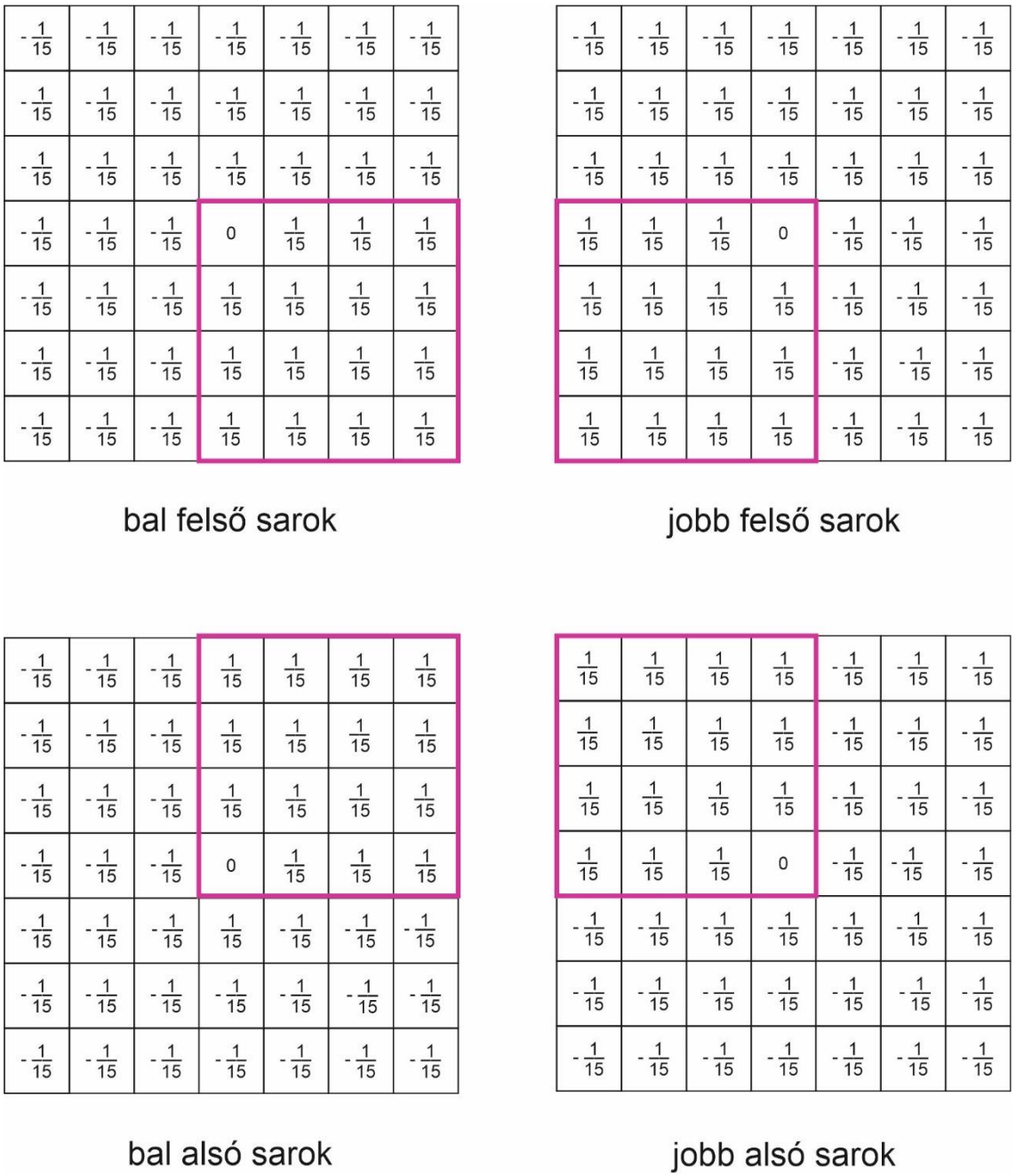
Konvolúciós szűrő

A konvolúciós szűrők a képfeldolgozás egyik legfontosabb összetevői. Számos fajtájukat különböztetjük meg (Szeparábilis szűrők, Gauss szűrő, Boksz szűrő stb.), melyeknek a lényegi közös nevezői az adott mintázandó objektum detektálása és finomítása. Ezen szűrők, kernelek valójában mátrixok formájában épülnek fel. Minden esetben számolnak a képközépponttal és a környezetében lévő átlag értékekkel. Áll egy bemeneti és egy kimeneti függvényből, melyek szorzatának integráltja adja meg a konvolúció értékét, ami egy kxk-méretű mátrix formájában valósul meg. Mi magunk határozzuk meg a mátrix méretét, azonban célszerű páratlan oldalhosszúságú létrehozása, hogy minél könnyebben meghatározható legyen annak középpontja (3x3, 5x5, 7x7). Minden értéket külön konvolvál, így a mátrix elemeit megszorozva az adott képpont pixel értékével kapjuk meg a konvolúció értékét. Ehhez a mátrix középpontját ráhelyezzük a kép minden pixelére. Minél nagyobb mátrixot veszünk alapul, annál erősebb és pontosabb eredményt kapunk, hisz több eredményből számlálódik ki az átlag.
A mátrixot egy NumPy tömbként adjuk meg (NumPy, 2020). A konvolúciós maszkot pedig a `cv2.filter2D()` függvény segítségével érhetjük el, aminek első paramétere a szűrendő kép, a következő az eredmény típusa, melyhez ha -1 értéket rendelünk, megőrizzük az eredeti kép típusát, az utolsó paramétere pedig az a mátrix, melyet szeretnénk alkalmazni.

```
M=cv2.filter2D(subimg,-1,K)
```

Ahol az M a végeredmény, a `subimg` a bemeneti kép (más néven forráskép), -1 az eredmény típusa, ami meg kell hogy egyezzen a bementi kép típusával, hisz nem lehet kevésbé pontos, utolsó sorban maga a kernel, ami a konvolúciót definiálja.

A bemeneti mátrix:



10. ábra: A sarokdetektáláshoz használt mátrixok.

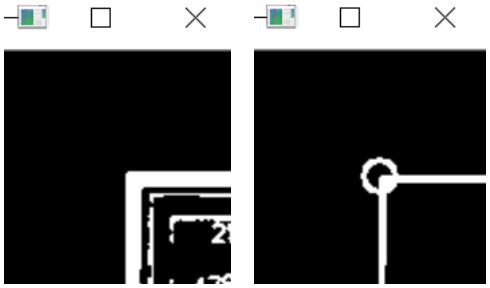
Az 10. ábrán látható az a mátrix, amit lefuttatunk a keret négy sarkára. Mindezek előtt szükséges a vonalak vékonyítása, egy eróziós szűrő segítségével, hogy megelőzzünk további hibákat.

Eróziós szűrő

Működése alapja egy kernel mely végigmegy a kép pontjain, melyek 0 vagy 1 értékeket adnak vissza. Csak akkor lesz értéke 1, ha a mátrix alatt található összes képpont 1, ellenkező esetben erodálódik és nullát ad vissza. A módszer alkalmas a felesleges zajok, vonalak, pontok eltávolítására.

`ek = np.ones((5, 5), np.uint8)` : itt látható, hogy egy 5 x 5-ös kernelt alkalmazok a vékonyításra, hiszen ha ez 3 x 3 lenne, akkor az erózió nem elég nagy és más pontokat is megtalál amiket elszeretnék hagyni és ha ezt a számot 7 x 7 -re növelem, a vékonyítás túl nagynak minősülne.

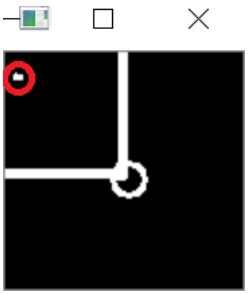
Felületek széleiről x pixel lefaragásához $(2x + 1) \times (2x + 1)$-es eróziós kernel kell, ez példámban 5 x 5 -ös mátrixnál 2 pixelt jelent, de egy 7 x 7- es mátrix esetében már 3 pixelt.



11. ábra: Sarok erózió előtt (bal), erózió után (jobb), a detektált sarokpont körrel jelölve.

Az adott sarokhoz az adott megfelelő mátrixot futtatjuk le (10. ábra).

Ez a mátrix azonban nem elég a maximum érték kiválasztásához. A hely = M.argmax() csakis egydimenziós tömbön tudja kiválasztani az első maximum helyet, ezért először átalakítjuk a kétdimenziós mátrixot egydimenziós tömbbé a `flatten()` függvény segítségével. Mivel a szelvényeken a keret sarka az első fekete pont, ami a mátrix segítségével könnyen meg is található, ezért, ha több ponton is magasabb értéket konvolvál a szűrőnk mindig az első pontot vesszük alapul. Azonban ez csak a fenti sarkokra érvényes. Ha jobban belegondolunk a lenti sarkokban nem a szűrő által kiválasztott első pontot szeretnénk kinyerni, hanem az utolsót, hisz ott lesz a keret, vagyis a vonalak metszéspontja. Ehhez szükséges a tömb megfordítása (mely Pythonban nagyon egyszerű: M=M[::-1]), így már kiválasztható a legnagyobb érték a lenti sarkokra tekintve is.



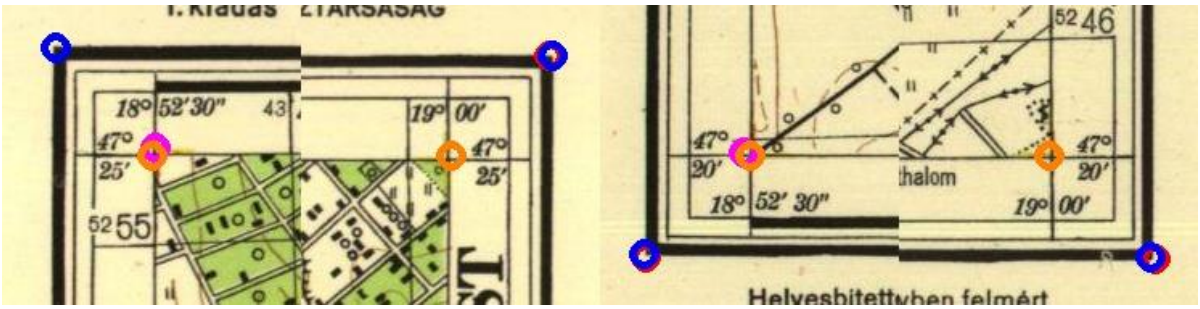
12. ábra: Fals sarokdetektálás eredménye az alsó sarkoknál

Az 12. ábrán látható, hogy ha nem fordítanánk meg az egydimenziós tömböt mit találna meg az argmax függvény első maximumhelynek, ami nyilvánvalóan nem a keret sarkát adná meg. Az újonnan megkapott pontosabb metszéspontokat ismét kirajzoljuk a `cv2.circle` segítségével.

Mivel az erózió során a vonal vastagsága vesztett 2 pixelt értékéből, így ezeket levonva rajzoltatjuk ki az új végleges sarokpontokat.

```
tenMmk = int((ly2 - fy2) / 391 * 10)-2


cv2.circle(kep, (jf[0] - tenMmk, jf[1] + tenMmk), 8, (255, 128,255),
2)
cv2.circle(kep, (bf[0] + tenMmk, bf[1] + tenMmk), 8, (255, 128,255),
2)
cv2.circle(kep, (jl[0] - tenMmk, jl[1] - tenMmk), 8, (255, 128,255),
2)
cv2.circle(kep, (bl[0] + tenMmk, bl[1] - tenMmk), 8, (255, 128,255),
2)
```



13.ábra: A finomítás előtt (piros, lila) és utáni sarokpont keresés eredménye (kék, narancs)

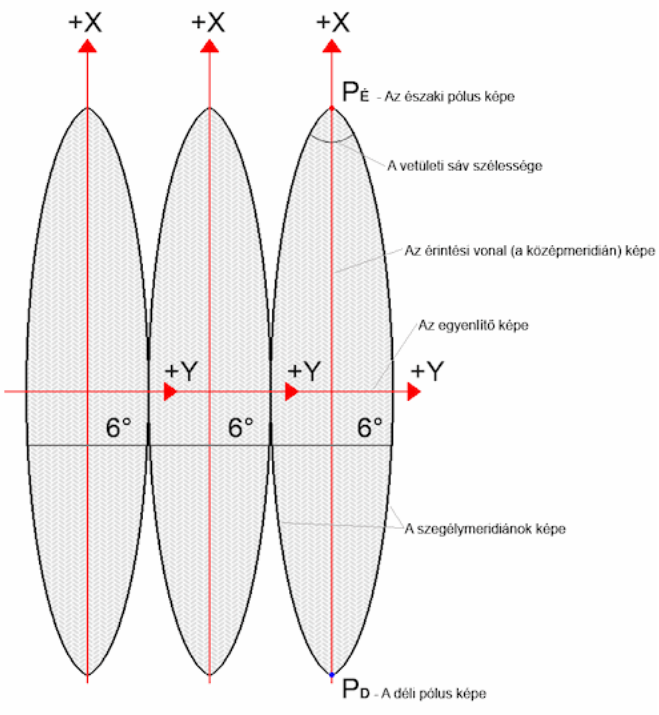
A 13. ábra jól szemlélteti az eddig leírtakat. A piros kör mutatja meg a külső sarokpont keresés eredményét, a lila pedig a belsőét. A finomítás munkálatok után a kék, és a számunkra legfontosabb, narancssárga körök helyén találjuk a végleges sarokpontokat.

Gauss–Krüger szelvényazonosító

A dolgozatom elején már bemutattam a feldolgozott szelvényeimet, azonban a szelvényazonosítók szabályrendszerét nem fejtettem ki részletesen.

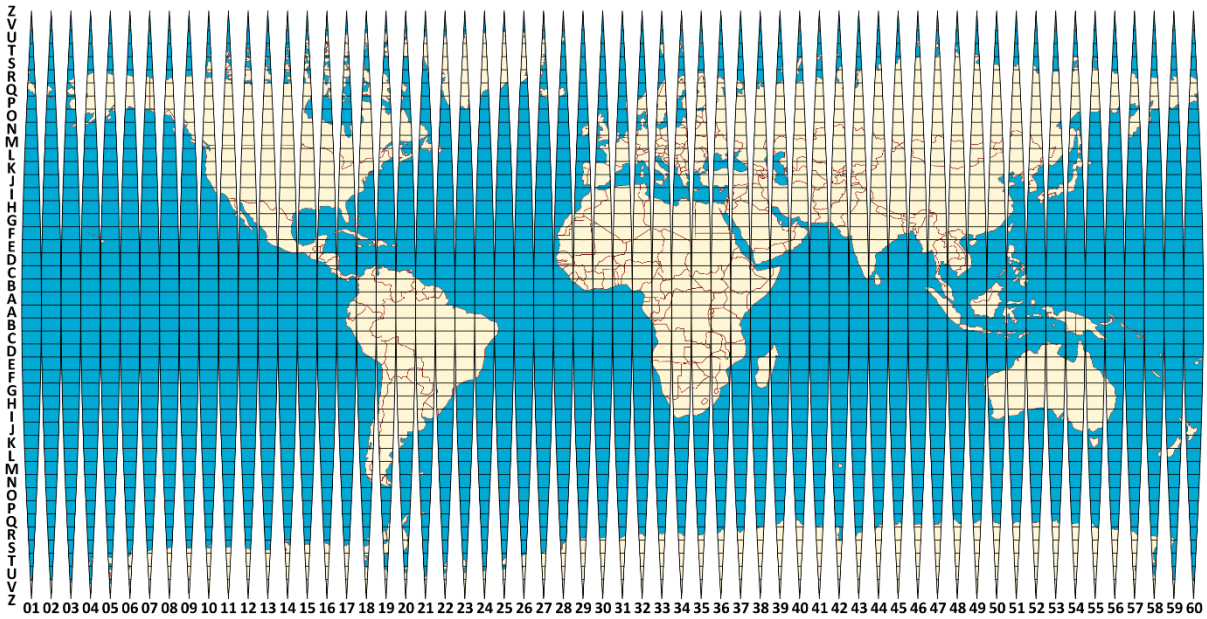
A Gauss–Krüger vetületet 1946-ban vezették be és kezdték el használni, az akkori szocialista rendszer térképészeti alapját szolgáltatta. A Gauss–Krüger vetület vonatkozási ellipszoidja a Kraszovszkij-ellipszoid, ehhez az egész Földre kiterjedő egységes szelvényezés tartozik (Dr Katona 2013).

A vetítés a meridiánok mentén érintő transzverzális elhelyezésű ellipszoidi hengerek felületére történik. Az egyes rendszerek önálló vetületi sávot képeznek, amik egymáshoz a szegélymeridiánok mentén csatlakoznak. Az egyes vetületi sávokon belül a vetületek törvényszerűségei teljesen megegyeznek, így a vetület az egész földfelület egységes rendszerben történő ábrázolására alkalmas (Bazsó, Czimber és Király 2011).



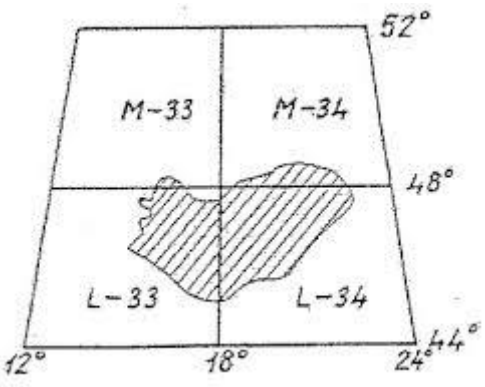
14. ábra: A Gauss–Krüger vetületi koordinátarendszer

A keretvonalak egyben a földrajzi fokhálózat vonalai. Az Egyenlítőtől kezdve 4°-os övekre, a Greenwich-től számítva pedig 6°-os zónákra osztották fel a Földet és az ABC betűit alkalmazták a övek, 1–60-ig tejerdő számokat pedig a zónák megkülönböztetésére (Mélykúti 2010).



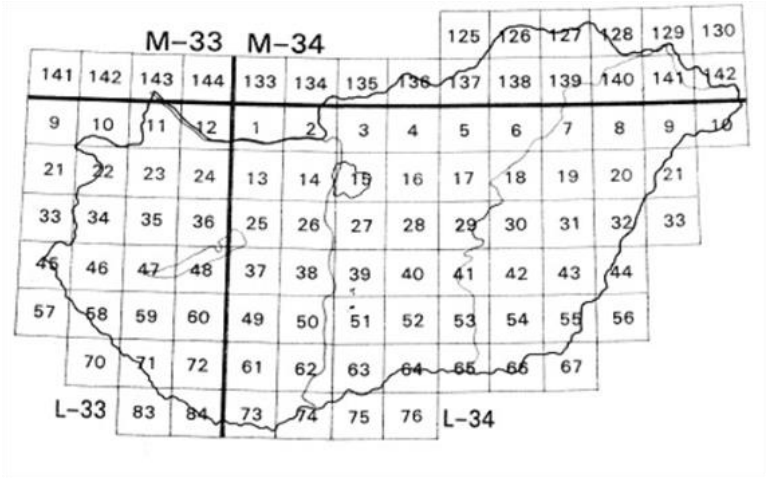
15. ábra: Az 1:1 000 000 méretarányú szelvények számozási rendszere

Az 15. ábrán jól látható, a Földet felosztó 60 zóna, azonban számunkra csak az L és M zónák fontosok az országunk földrajzi helyzetéből kifolyólag.

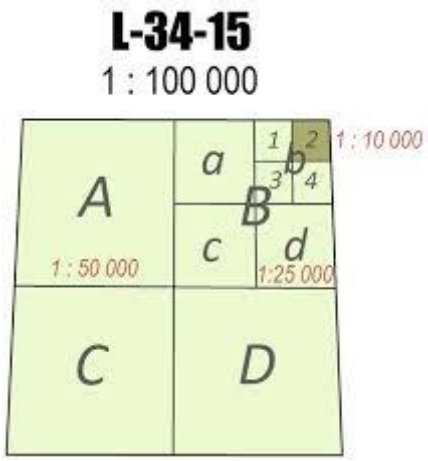


16. ábra: A Magyarország területét érintő 1:1 000 000 szelvények

Itt a 16. ábrán jól látható, hogy Magyarország az L és M övekre és 33–34-es oszlopokra esik. Ez a 4° × 6° -os terület egy 1 : 1 000 000 méretarányú szelvényt jelent, azonban a kutatásom ennek a területnek csak egy kis részére, az 1 : 25 000-es méretarányú szelvényekre koncentrálódik így ezek az 1 : 1 000 000-s szelvények további felbontása is szükséges volt. Az ábrázolt terület negyedelésével jutunk el a kívánt méretarányig, melyeket először az ABC nagybetűivel, majd kisbetűivel jelzünk.



17. ábra: 1 : 1 000 000 méretarányú szelvénybeosztás



18. ábra: 1 : 100 000 méretarányú szelvénybeosztás

Tehát a különböző méretarányú szelvények számai:
1 : 1 000 000 méretarányú szelvény száma L-34,
1 : 100 000 méretarányú szelvény száma L-34-57,
1 : 50 000 méretarányú szelvény száma L-34-57-A,
1 : 25 000 méretarányú szelvény száma L-34-57-A-c.

```
def GK (sheetId):
    try:
        sheetId=sheetId.split('-')
        szelesseg=52 if sheetId[0]=="M" else 48
        hosszusag=(int(sheetId[1])-31)*6
        print (szelesseg, hosszusag)
        szelesseg_perc=((int(sheetId[2])-1)//12)*20
        hosszusag_perc=((int(sheetId[2])-1)%12)*30
        print(szelesseg, szelesseg_perc)
```

```
        print (hosszusag, hosszusag_perc)
        if sheetId[3]=="B":
            hosszusag_perc+=15
        if sheetId[3]=="C":
            szelesseg_perc+=10
        if sheetId[3]=="D":
            hosszusag_perc += 15
            szelesseg_perc += 10
        print(szelesseg, szelesseg_perc)
        print(hosszusag, hosszusag_perc)
        if sheetId[4]=="b":
            hosszusag_perc+=7.5
        if sheetId[4]=="c":
            szelesseg_perc+=5
        if sheetId[4]=="d":
            hosszusag_perc += 7.5
            szelesseg_perc += 5
        print(szelesseg, szelesseg_perc)
        print(hosszusag, hosszusag_perc)
        szelesseg-=szelesseg_perc/60
        hosszusag+=hosszusag_perc/60
        A=(szelesseg,hosszusag)
        B=(A[0],A[1]+7.5/60)
        C=(A[0]-5/60,A[1])
        D=(A[0]-5/60,A[1]+7.5/60.)
        print(A,B)
        print (C,D)

    except:
        print ("bocsi itt hiba van", sheetId)
        pass
```

A fenti sorokban látható az előzőekben bemutatott Gauss–Krüger szelvényazonosítók számítása a programon belül. Kissé hosszú és bonyolultnak minősült az azonosítókból kiszámítani a földrajzi koordinátákat, ezért a későbbiekben egy másik módszert is bemutatok ezek kalkulálására.

Abból indultam ki, hogy minden szelvény mérete $5' \times 7.5'$. Ha a szelvényazonosító első betűje L, akkor 44°– 48° , ha pedig M akkor egy 48°– 52°-ig terjedő területről beszélünk. Ha az azonosító második összetevője, vagyis az oszlop száma 33, akkor 12°–18°, ha 34-es pedig 18°–24° közötti területet jelöl. Ugyanezzel az eljárással a további komponensekkel is elvégeztem az összefüggéseket, ezzel folyamatosan szűkítve a keresendő területet. A következő alkotórésze egy szám, ami 1–144-ig terjedhet (17. ábra), ezekután következhettek a nagybetűk A-B-C-D és a kisbetűk a-b-c-d is (18. ábra). Végeredményül megkaptam, hogy egy azonosító felismerésekor (előző fejezet OCR és Regex) és kiíratásából automatikus kalkulálja a hozzá tartozó földrajzi fokhálózat adatait.

Fájlba írás

```
if sheetId!="error":
    outtext = sheetId + ',' + str(jf[0] - tenMmk) + ',' + str(jf[1]
+ tenMmk) + ',' \
              + str(bf[0] + tenMmk) + ',' +  str(bf[1] + tenMmk) +
',' \
              + str(jl[0] - tenMmk) + ',' + str(jl[1] - tenMmk) +
','\
              + str(bl[0] + tenMmk) + ',' + str(bl[1] - tenMmk)
+','+kepnev + '\n'
    print(outtext)
    of.write(outtext)
    of.flush()

of=open("adatok.csv", 'w', encoding="utf-8")
```

Mivel a későbbiekben a belső sarokpontok pixelkoordinátái a lényegesek így azok kiíratása volt elengedhetetlen, ugyanakkor Python környezetben ez egyszerűen kivitelezhető volt. Szükséges megadni a formátumot, milyen típusú legyen az adott fájl, milyen karakterkódolással vigye véghez, valamint, hogy a ’w’ write vagy a ’r’ read funkciót alkalmazza. Természetesen az esetemben a ’w’ write volt indokolt.

GDAL

A GDAL a Python egy könyvtárcsomagja, ami a georeferálást teszi lehetővé, valamint különböző vetületeket ismer fel és számít át különböző koordinátarendszerekbe. Ezt a modult már nem a szakdolgozat.py programban telepítettem és írtam meg, hanem egy külön Python file-t hoztam létre annak érdekében, hogy biztosan végig fusson az első program probléma nélkül (vonaldetektálás, sarokpontkeresés). Ezek után következhetett a dolgozatom georeferálásért felelős része.
A GDAL (Geospatial Data Abstraction Library) valójában két könyvtárt tartalmaz, magát a GDAL-t a térinformatikai raszteradatokkal való munkálatokhoz és az OGR-t a térinformatikai vektoros adatmodellek manipulálását segítő csomagot.

Az importált modulok:

```
from osgeo import gdal
from osgeo import ogr
from osgeo import osr
```

Az előző fejezetben már bemutattam a szelvényazonosító számolására alkalmazott módszeremet, amikor is az azonosító ismeretében koordinátapárokat tudtam rendelni a sarokpontokhoz. Most egy másik rövidebb módszert fogok alkalmazni és bemutatni:
A karakterekkel kezdtem el dolgozni, hogy milyen messze vagyunk az L és M zónákban az A karaktertől. Ugyanis köztudott, hogy az informatikában mindent számokkal lehet leírni, hasonlóan a betűket is, így minden betűhöz tartozik egy szám, amit egy szabványrendszerbe illeszthetünk. Számtalan féle karakterkódolás létezik, de a számunkra lényeges információt csak a latin ABC nagy (csak az L és M) és kisbetűi (a,b,c,d) jelentenek vagyis az egymástól való távolságuk számokban. Hasonlóan az azonosító többi egységére is megnézve, annak a tudatában, hogy a szelvényazonosító 5 komponensből áll és egy terület 6° X 4°-os nagyságú. A betűhöz tartozó számot (szélesség meghatározása) megszorozva 4-gyel, a sorszámot (hosszúság meghatározása) pedig megszorozva 6-tal megkapjuk az 1 : 1 000 000 méretarányú szelvény helyét. A szelvényazonosító további részét hasonlóan végig gondolva és feldolgozva láthatjuk az lenti kódban. Az eredmény pedig ugyanúgy a sarokpontok földrajzi pixelkoordinátái.

```
def boundsFromGKSheet(s):
    i1=ord(s[0])-ord('A')
    i2=int(s[2:4])
    i3=int(s[5:8])
    i4=s[9]
    i5=s[11]
    n0=i1*4
    e0=i2*6-186
    n1=(11-(i3-1)//12)/3
    e1=((i3-1)%12)/2
    n2=1/6 if i4 in 'AB' else 0
    e2=1/4 if i4 in 'BD' else 0
    n3=1/12 if i5 in 'ab' else 0
    e3=1/8 if i5 in 'bd' else 0
    W=e0+e1+e2+e3
    S=n0+n1+n2+n3
    E=W+1/8
    N=S+1/12
    return (W,S,E,N)


src = osr.SpatialReference()
src.ImportFromEPSG(4284) # s-42 pulkovo földrajzi

targ33 = osr.SpatialReference()
targ34 = osr.SpatialReference()
targ33.ImportFromEPSG(28403) # s-42 pulkovo Gauss-Krüger zone 3
targ34.ImportFromEPSG(28404) # s-42 pulkovo Gauss-Krüger zone 4

transform33 = osr.CoordinateTransformation(src, targ33)
transform34 = osr.CoordinateTransformation(src, targ34)
```

A fenti kódsorban láthatjuk a vetületi beállításokat és transzformációkat. Ehhez létre kellett hoznunk koordinátarendszer típusú objektumokat (`SpatialReference` objektumok) mindkét vetületi zónához (s-42 Pulkovo, Gauss–Krüger zónák) annak érdekében, hogy később műveleteket tudjuk értelmezni rajtuk. A GDAL különlegessége, hogy az EPSG számok ismeretében át lehet számolni egyik koordinátarendszerből a másikba, így innentől kezdve megadott `src`-ből `targ33` (Gauss–Krüger 33-as zóna), vagy `targ34-be` (Gauss–Krüger 34-es zóna) számol át.

```
imgpath="C:/Lola/szelvenyek/"
outpath="C:/Lola/eredmeny/"
corners='C:/Lola/python/Elte/adatok.csv'
```

A fenti három sorban megadtam, milyen szelvényekre szeretném elvégezni a georeferálást (1 : 25 000- es GK szelvények), hova mentse a végeredményt és melyik fájlból vegye az adatokat (előző fejezet fájlba írás).

```
with open(corners) as f:
    sheets=f.read().split('\n')

(W,S,E,N)=boundsFromGKSheet(s[0])
T=transform33 if W<18 else transform34
targ=targ33 if W<18 else targ34
```

A következőkben beolvassa az adatokat az előző fájlban (szakdolgozat.py) létrehozott CSV-ből (adatok.csv). Elkéri a már előzetesen kiszámolt S,W,N,E-t és beállítja önmagának, hogy ez a 33-as vagy 34-es övben helyezkedik el. A GDAL hátránya, hogy az illesztőpontok létrehozása kissé bonyolult. Well-known text formátumban megadott geometriai objektumot hoztunk létre az N,E; N,W; S,E; S,W pontokra és következhetett maga a transzformáció, pulkovo-i földrajziból Gauss–Krüger koordinátákra.

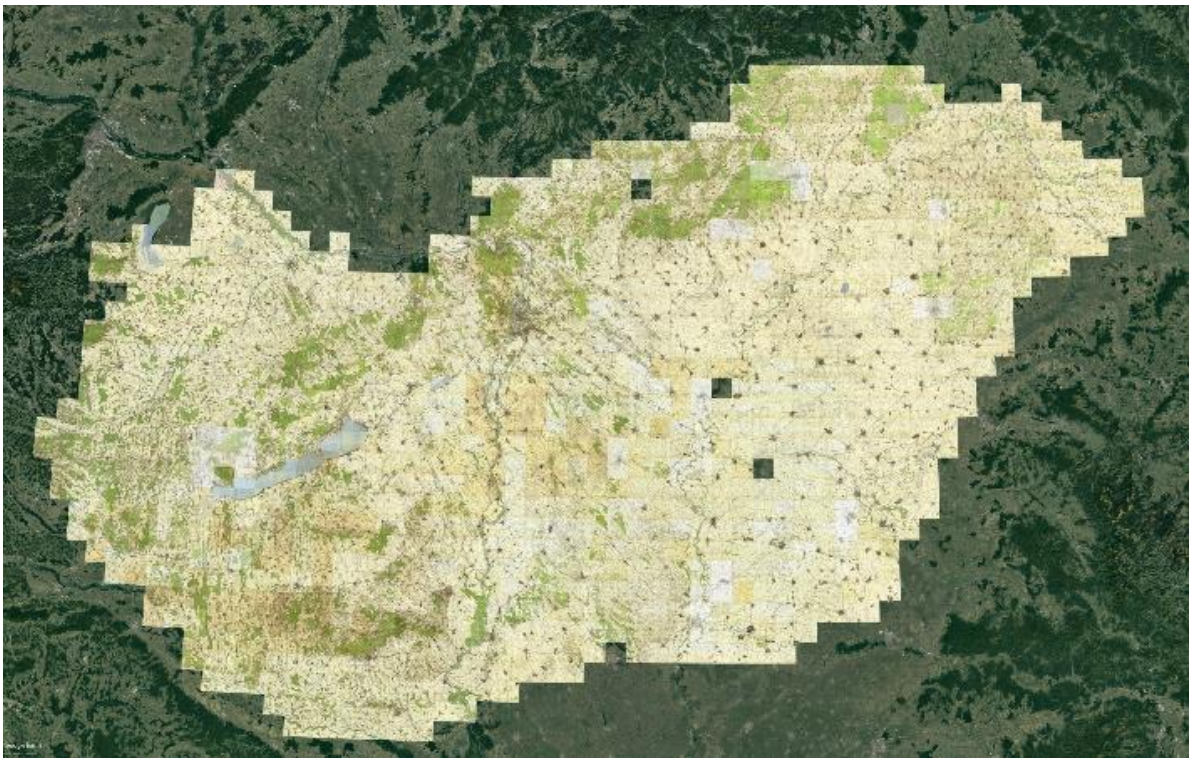
```
G=[]
p=ogr.CreateGeometryFromWkt("POINT ({} {})".format(N,E))
p.Transform(T)
G.append(gdal.GCP(p.GetY(),p.GetX(),0,int(s[1]),int(s[2])))
p=ogr.CreateGeometryFromWkt("POINT ({} {})".format(N,W))
p.Transform(T)
G.append(gdal.GCP(p.GetY(),p.GetX(),0,int(s[3]),int(s[4])))
p=ogr.CreateGeometryFromWkt("POINT ({} {})".format(S,E))
p.Transform(T)
G.append(gdal.GCP(p.GetY(),p.GetX(),0,int(s[5]),int(s[6])))
p=ogr.CreateGeometryFromWkt("POINT ({} {})".format(S,W))
p.Transform(T)
G.append(gdal.GCP(p.GetY(),p.GetX(),0,int(s[7]),int(s[8])))


opts = gdal.TranslateOptions(GCPs=G, format="GTiff", outputSRS=targ)
gdal.Translate('g.tif', s[9], options=opts)
gdal.Warp(outpath+sId+'.tif', 'g.tif', dstSRS=src, outputBounds=(W,
S, E, N), xRes=1e-4, yRes=1.47e-4,
creationOptions=["COMPRESS=JPEG"])
n+=1
print(n,s[0])
```

Utolsó sorban látható miként készítettem GeoTIFF formátumú fájlt a gdal.Translate segítségével, egy raszteres képből egy lista segítségével, amiben előzetesen már beletettem az illesztőpontokat és a pixel x és y koordinátáit. Ehhez szükséges a lista maga (G), kimenő formátum típusa (GTiff), a kész eredménynek a koordinátarendszere (targ33 vagy targ34).

Kivágtam a térképet a keretvonalak körül a gdal.Warp paranccsal, hogy a későbbiekben hézag és átfedésmentesen össze tudjam őket illeszteni egy térinformatikai szoftverben.

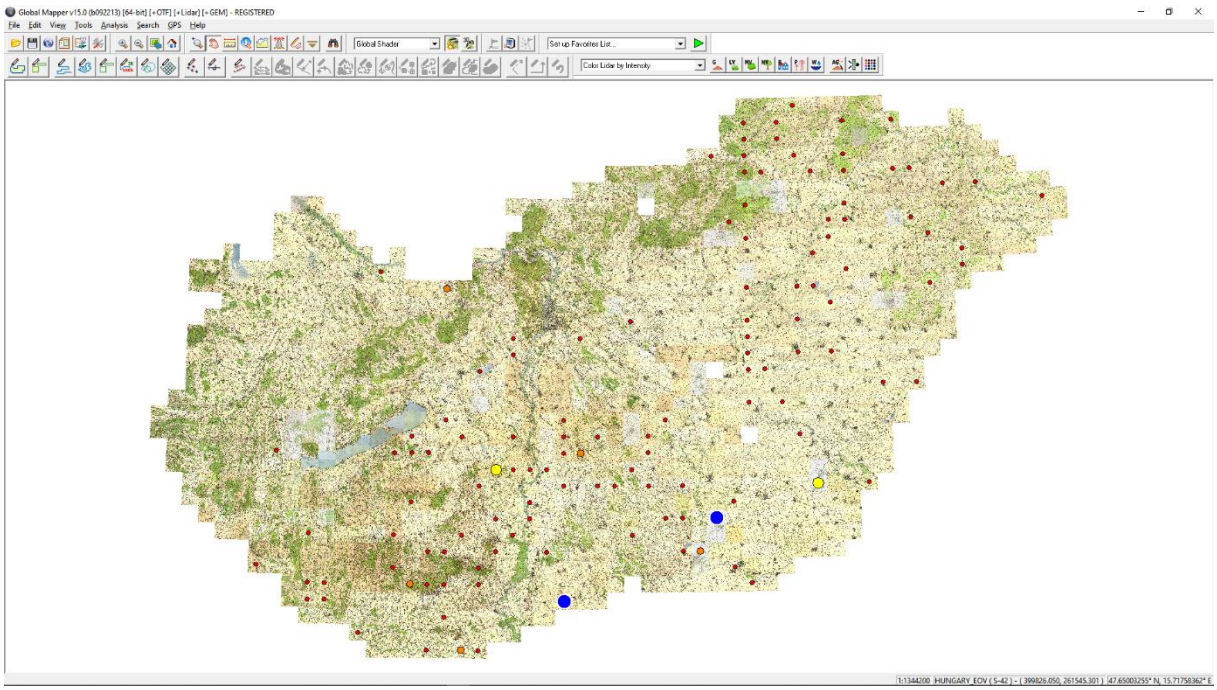
Eredmények



19. ábra: Az 1147 georeferált szelvény Google Earth-ben megnyitva

A 19. ábrán jól látszik a végeredmény, az összes térképszelvény körülvágott, georeferált változata. Pár helyen észrevehető, hogy a térképkollekció hiányos .

Összesen 1147 szelvény került feldolgozásra, egyenként átlagosan 4 másodperces futási idővel. A sarokpontok finomítása után a szelvények 89%-ánál 1% alatt, míg 99%-ánál 2% alatt volt a hibaarány. A szelvényazonosító detektálása 75,9%-ban volt eredményes.
A program legidőigényesebb része a vonaldetektálás volt, így amint a keretvonalak megvoltak, a többi részfolyamat már gyorsan lefutott, körülbelül szelvényenként 0,1 másodperc alatt.
A sarokpontok ellenőrzésére egyszerű matematikai módszereket alkalmaztunk. Összehasonlítottuk a bal és jobb, valamint a vízszintes és a függőleges keret hosszának arányát a szélesség koszinuszával korrigálva.



20. ábra: A georeferált és összeillesztett mozaik. Az 1%-nál nagyobb geometriai hibát a színes korongok jelzik.

A 20. ábrán láthatóak azok az eltérések, melyek nagy részben a különböző alapfelületű szelvények miatt jöttek létre. A korongok jelölik azt, hogy a detektált sarkok alapján, milyen mértékű elmozdulásról beszélünk. A piros 1%, a narancs 2%, a sárga 5% és kék korongok 10% elmozdulást mutatnak, azonban 2%-nál nagyobb hiba csak 9 szelvényen mutatkozott. Ha jól megfigyeljük, a 2. ábrán megrajzolt hosszúsági kör mentén különböző alapfelületű szelvények találkozásánál a 20. ábrán végig piros korongok rajzolódnak ki, az előzőekben említett eltérés miatt.

Mivel a beolvasott térképek fájlnevei már tartalmazták a szelvényazonosítókat, így az OCR ellenőrzése nem okozott problémát. 1147 szelvény közül 261 esetében nem egyezett a fájlnévből kinyert és az OCR által felismert szelvényazonosító. A problémák egy részének kiküszöbölésére a 26. oldalon lévő Regex fejezetben olvashatunk. Csak 16 esetben, vagyis 1,4% -ban volt a javításokon túl teljesen eltérő az azonosító, azonban OCR beállításait tovább hangolva, esetleg a térképeken használt betűtípus betanításával ez is csökkenthető.

Konklúzió

A szakdolgozatomban a II. világháború utáni gyorshelyesbítés szelvényeit georeferáltuk OpenCV segítségével. A kézi georeferálás lassú és körülményes lépéseit elhagyva hoztunk létre egy automatizáló programot mely segítségével az összes szelvényt percek alatt a Föld valós terébe helyeztünk el. Ezeket pontosan összeillesztve, a keretvonalak levágása után kirajzolódott előttünk hazánk térképe, melyet már tényleges navigációra is használni tudunk a georeferencia adatok segítségével.

Felhasznált eszközeink nyílt forráskódú könyvtárak, melyeket alkalmazva jutottunk el a végeredményhez. Az OpenCV (számítógépes látás függvénytár), a Tesseract (szövegfelismerő eszköz) és a GDAL (Geospatial Data Abstraction Library) fő modulok voltak segítségünkre.

Az automatizáló program több szakaszból épült fel, melyek kódrészleteit részletesen bemutattam a dolgozat különböző fejezeteiben. Ezek közül a legfontosabbak voltak, a fekete pixelek kiemelése, a megfelelően hosszú egyenesek detektálása és leválogatása a szelvényeken, azok metszéspontjainak, tehát sarokpontjainak meghatározása, és a belső sarokpont keresése. A szelvényazonosító beolvasása, ami segítségével pontos földrajzi koordinátákat tudtunk illeszteni a már előre kiszámított sarokpontokhoz. Az azonosító detektálásához az OCR-t (Optical Character Recognition) futtattuk a kép egy adott kivágatán, majd a Regex-mintaillesztést alkalmazva rá, nyertük ki az adott karaktereket. Amint megvoltak az illesztőpontok (pixelkoordináták) és az azonosítók (pixelkoordináták földrajzi helyzete) kezdődhetett maga a georeferálás, ami során GeoTIFF formátumban mentettük el a térképeket. Innentől már mindenféle probléma nélkül nyithattuk meg őket különböző térinformatikai programokban, ahol automatikusan a helyükre kerültek a Föld felszínén. A szelvények keretvonalait levágva hézag- és átfedésmentesen összeilleszthetők lettek, ezzel plasztikus képet nyújtva Magyarországról.

Felmerül a kérdés, miért hasznos régi topográfiai szelvényeket georeferálni és valós térben elhelyezni? Számos szempontból, hisz rengeteg információt nyerhetünk ki belőlük a múltról, az akkori viszonyokról és növényzeti fedettségről. Láthatóvá válik, hogyan változott a növényborítás, ami különösen jelentős a természetvédelem és ökológia szempontjából. A régi közlekedés és úthálózat ismerete fontos információt nyújthat a régészeknek, történelemtudósoknak, de a mindennapi életre is befolyással lehet, például építkezéseknél vagy földmunkáknál. Ezek mellett az árvíz- és belvízvédelem aspektusából a mai napig mérvadó merre folytak a folyók és a patakok.

Végső összegzésként elmondható, hogy egy releváns térinformatikai témával találkozhattunk, mely során a programozás területén bővíthettük tudásunkat és nyerhettünk mélyebb betekintést. Elsajátítottuk a Python képfeldolgozásért felelős funkcióit és lehetőségünk nyílt térképészeti tudásunk kiterjesztésére az informatika világában.

Irodalomjegyzék

Tanács , Attila. *Digitális képfeldolgozás.* 2018-2021. http://www.inf.u-szeged.hu/~tanacs/pyocv/

ArcGIS.Pro. *Overview of georeferencing.* 2017. https://pro.arcgis.com/en/pro-app/latest/help/data/imagery/overview-of-georeferencing.htm

Bazsó, Tamás, Kornél Czimber, és Géza Király. *Földmérés.* 2011. https://oszkdk.oszk.hu/storage/00/01/43/03/dd/1/foldmeres.pdf

Dharra, Ratan Singh. *Line detection.* 2019. https://rsdharra.com/blog/lesson/14.html

Dimitri van Heesch. *Hough Line Transform.* 2021. https://docs.opencv.org/master/d6/d10/tutorial_py_houghlines.html

Dr Katona, Endre. *Térinformatika.* 2013. https://www.inf.u-szeged.hu/~katona/gis.pdf

Dr Timár, Gábor. *TÉRKÉPI VETÜLETEK ÉS GEODÉZIAI DÁTUMOK.* 2008. http://sas2.elte.hu/tg/georeferencia.htm

Emri, Miklós. *Orvosi képfeldogozás.* 2011. https://regi.tankonyvtar.hu/hu/tartalom/tamop425/0019_1A_orvosi_kepfeldolgozas/ch09.html

Herold, Hendrik, Patric Roehm, Robert Heck, és Gotthard Meinel. *AUTOMATICALLY GEOREFERENCED MAPS AS A SOURCE FOR HIGH RESOLUTION URBAN GROWTH ANALYSES.* Proceedings of the ICA 25th International Cartographic Conference, July 3 - 8. Paris, France, 2011.

I Rus, C Balint, V Craciunescu, S Constantinescu4, I Ovejanu, Zs Bartos-Elekes. *AUTOMATED GEOREFERENCE OF THE 1:20 000 ROMANIAN MAPS UNDER LAMBERT-CHOLESKY (1916–1959) PROJECTION SYSTEM.* 2010. https://www.researchgate.net/publication/250008142_Automated_georeference_of_the_120_000_Romanian_maps_under_Lambert-Cholesky_1916-1959_projection_system

Jankó, Annamária. *Magyarország katonai felmérései.* Budapest: Argumentum kiadó, 2007.

Jātnieks , Jānis . *Open Source Solution for Massive Map Sheet Georeferencing Tasks for Digital Archiving.* 2010. http://home.lu.lv/~sg30022/ICADL2010_Jatnieks.pdf

Kuchling, A.M. . *Regular Expression.* 2001-2020. https://docs.python.org/3/howto/regex.html#introduction

Mélykúti, Gábor. *Topográfiai térképek jellemző tulajdonságai.* 2010. https://regi.tankonyvtar.hu/hu/tartalom/tamop425/0027_TOP3/ch01s03.html

Nagy, Zoltán. *Magyar topográfiai alaptérképművek.* 1985. http://lazarus.elte.hu/hun/digkonyv/nagy_zoltan/nz.htm

Python Software Fundation. *Regular Expressions.* 2021. https://docs.python.org/3/library/re.html

Rosebrock, Dr. Adrian. *Practical Python and OpenCV: An Introductory, Example Driven Guide to Image Processing and Computer Vision.* 2016. https://minhtn1.github.io/Practical%20Python%20and%20OpenCV,%203rd%20Edition.pdf

Zelic, Filip, és Anuj Sable. *A comprehensive guide to OCR with Tesseract, OpenCV and Python.* 2021. https://nanonets.com/blog/ocr-with-tesseract/#tesseractocr

Zentai , László. *Topográfiai térképek előadás diái .* 2018.

Zentai, László. *A digitális térképek Magyarországon az első digitális adatbázisoktól a kilencvenes évek végéig.* 2012. http://www.rsgis.hu/RS&GIS-2012-1-3.html

Az internetes hivatkozások utoljára ellenőrizve: 2021.05.14.

Ábrajegyzék:

Az ábrák saját szerkesztésűek. Készült: 2021.05.10.

SZAKDOLGOZAT / DIPLOMAMUNKA

EREDETISÉG NYILATKOZAT

Alulírott **Varga Lola**, Neptun-kód: **DG12LU**

ezennel kijelentem és aláírásommal megerősítem, hogy az Eötvös Loránd Tudományegyetem Informatikai Karának, Térképtudományi és Geoinformatikai Intézetében írt,

Topográfiai térképek automatikus georeferálása OpenCV-vel

című diplomamunkám saját, önálló szellemi termékem; az abban hivatkozott szakirodalom felhasználása a szerzői jogok általános szabályainak megfelelően történt.

Tudomásul veszem, hogy szakdolgozat/diplomamunka esetén plágiumnak számít:

- szószerinti idézet közlése idézőjel és hivatkozás megjelölése nélkül;
- tartalmi idézet hivatkozás megjelölése nélkül;
- más publikált gondolatainak saját gondolatként való feltüntetése.

A témavezető által benyújtásra elfogadott szakdolgozat PDF formátumban való elektronikus publikálásához a tanszéki honlapon

HOZZÁJÁRULOK NEM JÁRULOK HOZZÁ

Budapest, 2021. 05. 14.

Varga Lola

……..

hallgató aláírása