Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar

Borszuk Judit

AGRO-TÉRINFORMATIKAI ADATFELDOLGOZÁS

EGYETEMI KONZULENS
Krébesz Tamás István

KÜLSŐ KONZULENS
Dömötör Ákos - ITineris Kft.

BUDAPEST, 2016

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott **Borszuk Judit**, szigorló hallgató kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2016. 05. 29.

...……

Borszuk Judit

Összefoglaló

A modern mezőgazdaságban egyre fontosabbá válik a termelékenység növelése, ehhez szükséges az elvégzett munka pontos nyilvántartása. Ennek köszönhetően a precíziós mezőgazdaságban mára alapvetővé vált a GPS alapú rendszerek alkalmazása.

A diplomaterv agro-térinformatikai adatfeldolgozással foglalkozik. A feladat célja mezőgazdasági gépek mozgása során gyűjtött GPS adatok alapján a munkavégzés mérőszámainak meghatározása, ezen belül is a mezőgazdasági táblákon végzett munka kezdetének és végének meghatározása, a munkaszélesség megállapítása, valamint a megművelt és az esetlegesen kihagyott földterületek meghatározása. A diplomaterv ismerteti és összehasonlítja a különböző térinformatikai algoritmusokat, amelyek segítségével a szükséges adatok meghatározhatók.

A diplomaterv ismerteti a GPS technológia korlátait és az ezek miatt szükséges különböző feldolgozási módszereket, amelyekkel az adatok pontossága növelhető. Majd bemutatja a térképészeti alapokat és azokat a vetülettípusokat, amelyek a számítások elvégzéséhez szükségesek.

A diplomaterv többféle lehetőséget is bemutat a feladat elvégzéséhez. Ezekhez szükséges a képfeldolgozás, azon belül is a morfológia alapjainak ismertetése.

A diplomatervhez hozzátartozik a lehetséges módszerek elemzésén túl a feladat megvalósítása C++ nyelven, majd az eredmény pontosságának és hatékonyságának kiértékelése.

Abstract

In modern agriculture the productiveness is becoming more and more important therefore it is necessary to keep record of field work. Because of this the usage of GPS based applications has become elementary in precision farming.

The topic of this thesis is the usage of geographical information systems in agriculture. The main purpose is to determine the quality of field work based on GPS data that was collected during the movement of agricultural vehicles. Specifically, the aim of this thesis is to determine the beginning and the end of field work and the sowing distance, furthermore to calculate the area of cropped fields and skipped areas. Different geoinformatical algorithms are introduced and compared in order to find the most efficient for the extraction of the required data.

The thesis summarizes the potentials and limits of GPS technology, and presents different methods for precision improvement. Furthermore it gives an overview on the basics of cartography and presents some map projection types that are necessary before executing the mathematical operations.

Various options are presented for the extraction of the required information. For these the basics of image processing, especially mathematical morphology is described.

Besides analyzing the possible methods, this thesis contains the implementation of the necessary algorithms in C++ language and the evaluation of precision and effectiveness.

1 Térinformatika a mezőgazdaságban

A technológiai fejlődésnek köszönhetően ma már széles körben elterjedt a GNSS[1] technológia alkalmazása. Ennek köszönhetően a szállítmányozás, logisztika és járművédelem mellett a mezőgazdaságban is egyre gyakrabban használnak műholdas helymeghatározáson alapú rendszereket. Ennek célja a mezőgazdaság fenntarthatóságának biztosítása, a termelés hatékonyságának növelése, valamint a környezeti ártalmak csökkentése.

Ennek érdekében különböző mezőgazdasági célokra alkalmaznak GPS alapú technológiákat, többek között hozam monitorozásra, különböző ütemben történő vetés, permetezés elvégzésére. Az egyes földterületekről gyűjtött mintákhoz földrajzi pozíciót lehet rendelni, így a megfelelő területekre lehet juttatni a szükséges vizet és tápanyagot. A GNSS technológia segítségével megelőzhető, hogy kimaradjanak megművelni kívánt területek, vagy ugyanazon a földdarabon véletlenül többször is végighaladjanak. Az átfedések megelőzésének köszönhetően csökken a felhasznált üzemanyag és vetőmag mennyisége. Mivel így az is kiküszöbölhető, hogy többször permetezzék ugyanazt a területet, ezért a környezetvédelem szempontjából is előnyös.

Mindebből látható, hogy a precíziós gazdálkodásban elengedhetetlen, hogy meg lehessen határozni a megművelt földterület mennyiségét, a sortávolságokat és fel lehessen deríteni az esetlegesen kihagyott területeket. Ezekhez azonban szükség van a különböző nagy teljesítményű mezőgazdasági gépek szántóföldön végzett munkájának követésére. Ennek megvalósítására a gépekbe speciális nyomkövető és adatrögzítő eszközöket szerelnek. A GNSS eszközökből származó adatokat azonban fel kell dolgozni a felhasználó igényeinek megfelelően. A diplomaterv a feldolgozás menetét és nehézségeit ismerteti.

[1] Global Navigation Satellite System

2 Az adatfeldolgozás célja

A feladatom az volt, hogy egy kontrolling rendszerben keletkezett GPS adatsorok alapján kidolgozzam, hogy milyen módszerekkel lehet a gépek által végzett munkaterületeket elkülöníteni. Az adatsorok feldolgozása utólag történik, post-process módon. A vevő eszköz által többféle adat rendelkezésre áll. Ezek a következők:

- időbélyeg
- sebesség
- szélességi fok
- hosszúsági fok
- magasság
- menetirány
- egyéb mérési jellemzők

A földrajzi koordináták WGS84[2] dátum szerint vannak megadva, felbontásuk ezred szögmásodperc (MAS[3]). Ez körülbelül 2 cm-es felbontást jelent, azonban a vevő egységek által szolgáltatott mérési adatok pontossága ennél lényegesen rosszabb. Többféle vevő adatait kell feldolgozni, ezek között vannak precíziós vevők, melyek pontossága körülbelül 10 cm, vannak 50 cm körüli pontossággal rendelkező egységek, a kereskedelmi forgalomban kapható eszközök ennél jelentősen rosszabbak, 3-5 méter pontosságúak. Az adatok másodpercenként érkeznek.

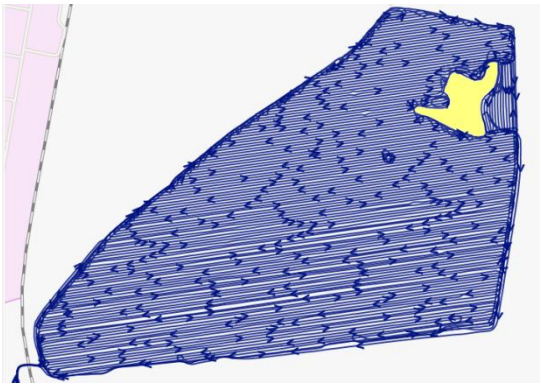
A vagyonvédelmi alkalmazásokban és a szállítmányozás területén alkalmazott adatgyűjtőkkel ellentétben a vevő egységek nem végeznek semmilyen szűrést vagy illesztést, így a nyers adatok állnak rendelkezésre.

A diplomaterv célja, hogy a rendelkezésre álló adatsor alapján meg lehessen határozni a járművekkel végzett szántóföldi munkák mennyiségi jellemzőit. A legfontosabb feladat a munka kezdetének és végének detektálása, tehát, hogy el lehessen különíteni a vonulást, azaz a táblák közti mozgást azoktól a szakaszoktól, ahol tényleges munkavégzés történt. Emellett meg kell határozni a munkaszélességet, azaz a

[2] World Geodetic System 1984

[3] milliarcsecond

sortávolságot. Valamint a megművelt és az esetlegesen kihagyott területek méretét is ki kell számítani. A 2.1. ábrán egy mezőgazdasági tábla látható, ahol valamilyen okból kimaradt egy részlet.



2.1. ábra Mezőgazdasági tábla kihagyott területtel

Sokféle mezőgazdasági gép adatainak feldolgozását kell elvégezni, így figyelembe kell venni az ezek közötti eltéréseket. Jelentős eltérés van a gépek munkaszélessége között. Vannak olyan járművek, amelyek 1-2 méter távolságban végzik a munkát, és vannak olyanok, melyek munkaszélessége akár 20-30 méter is lehet. Számottevő eltérés van a gépek munkavégzési sebessége között is. Vannak olyan gépek, amelyek közel azonos sebességgel végzik a szántóföldi munkákat, mint amilyennel vonulnak, a gépek nagy része viszont sokkal lassabban dolgozik, mint ahogy vonul.

Az adatok feldolgozása egy szerver oldali alkalmazás része lesz, tehát fontos, hogy minél gyorsabb legyen a létrehozott program működése. Az alkalmazás feladata az lesz, hogy bizonyos időközönként kapott adatcsomagok alapján eldöntse, hogy az adott jármű munkát végez-e, vagy vonul. Ritkábban területszámításra is szükség van. Általában rendelkezésre áll táblatérkép, ilyenkor egyszerűen el lehet dönteni, hogy a jármű munkát végez-e, csak össze kell hasonlítani a pozícióadatot a táblatérképpel: ha a térképen belül van, akkor munkát végez. Ehhez egyszerűen azt kell megvizsgálni, hogy az adott koordináta a táblát körülvevő sokszögön belül helyezkedik-e el. Ezt részletesen ismerteti a 8.6 fejezet. Bizonyos esetekben azonban nem áll rendelkezésre táblatérkép, például bérmunkák végzésekor és a nem saját táblákon végzett munka esetén. Ilyenkor a nyers adatokból kell eldönteni, hogy vonulás vagy mezőgazdasági munka történik, és ezekből az adatokból kell meghatározni a megművelt terület méretét és a sortávolságokat is.

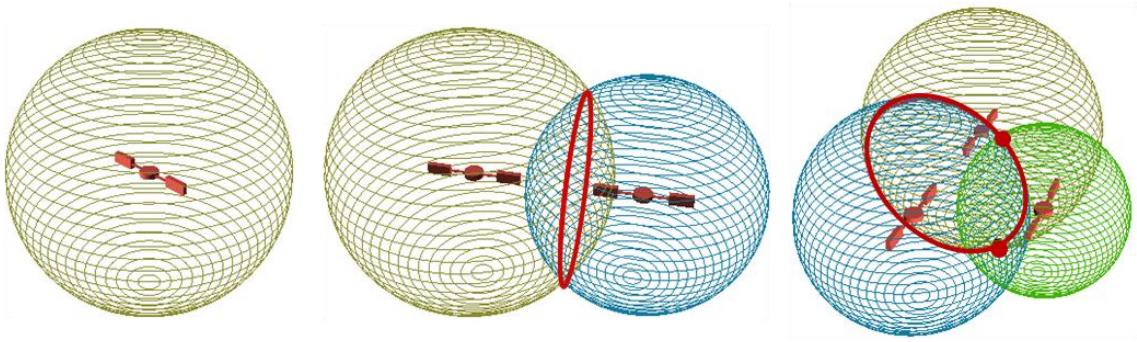
3 GPS adatok feldolgozása

A GPS[4], azaz a Globális Helymeghatározó Rendszer egy széles körben alkalmazott műholdakra alapozott technológia, amely lehetővé teszi földrajzi pozíciónk meghatározását. A következő fejezet ismerteti a GPS technológia alapjait, a helymeghatározás pontosságát, valamint a GPS által szolgáltatott mérési eredmények szokásos feldolgozási módszereit.

3.1 A GPS technológia

A GPS technológia alapja, hogy a földrajzi helyzetünk meghatározható, ha meg tudjuk mérni legalább négy ismert pozíciójú műholdtól vett távolságunkat. A távolság meghatározása időmérésen alapul. A műhold rendelkezik egy órával és ennek segítségével az általa küldött üzenet tartalmazza a küldés időpontját. A vevő egység is rendelkezik egy órával, ami a műhold órájával elméletileg szinkronban jár, így az üzenete vételekor meg tudja határozni a terjedési időt. Továbbá a jel sebessége is ismert, fénysebességgel terjed, így az eltelt idő és a sebesség alapján az adott műhold távolsága kiszámítható. A pozíció meghatározáshoz azonban legalább négy műhold szükséges.

Az 3.1. ábra mutatja hogyan csökken a tényleges pozíciónkat tartalmazó lehetséges terület mérete a vevő egység által látott műholdak számának növelésével.



3.1. ábra Trilateráció

Egy műhold segítségével csak az állapítható meg, hogy a pozíciónk egy gömb felületén található, amelynek középpontja a szatellit, sugara pedig a kiszámított

[4] Global Positioning System

távolság. Ha ezzel a méréssel egy időben egy második műholdat is figyelembe veszünk, akkor megállapítható, hogy a vevő egy gömb felületén lesz amelynek közepén ez a második műhold található. Tehát a vevő a két gömb felületén helyezkedik el, azaz a két gömb által meghatározott kör valamelyik pontján található. Egy harmadik szatellit segítségével megállapítható, hogy a vevő az előző két gömb által kijelölt kör és a harmadik műhold által meghatározott gömb metszéspontján található. Ezzel két pontra szűkítettük a pozíciónkat. Ez a két pont azonban jelentősen eltér egymástól, mind a szélességi és hosszúsági fokokban, mind a magasságban, így eldönthető, hogy melyik az ami nekünk szükséges [1].

Elméletileg három műhold segítségével meghatározható a pozíció, ehhez azonban az kellene, hogy a vevő órája tökéletesen szinkronban legyen a műholdéval. Ez miatt a vevőkben lévő kvarc oszcillátor hibáját is ismeretlenként kezelik. Így a pozíció meghatározásához négy ismeretlent kell meghatároznunk, ezek a jelterjedési idő hibája ε, valamint az (x_0, y_0, z_0) koordinátáink. Ha négy műholdat veszünk figyelembe, akkor négy egyenletünk lesz, amelyekből kiszámíthatóak az ismeretlenek. A négy műhold pozícióját jelölje X_i, Y_i, Z_i, a jel megérkezéséhez szükséges időt Δt_i. A d távolság függ a jel terjedésének a vevő által mért idejétől és a vevőben lévő oszcillátor hibájától.

$$\begin{Bmatrix} (x_0 - X_1)^2 + (y_0 - Y_1)^2 + (z_0 - Z_1)^2 = d(\Delta t_1, \varepsilon)^2 \\ (x_0 - X_2)^2 + (y_0 - Y_2)^2 + (z_0 - Z_2)^2 = d(\Delta t_2, \varepsilon)^2 \\ (x_0 - X_3)^2 + (y_0 - Y_3)^2 + (z_0 - Z_3)^2 = d(\Delta t_3, \varepsilon)^2 \\ (x_0 - X_4)^2 + (y_0 - Y_4)^2 + (z_0 - Z_4)^2 = d(\Delta t_4, \varepsilon)^2 \end{Bmatrix} \quad [3]$$

Általában minél több műholdat lát a GPS vevő annál nagyobb pontosság érhető el, de ez függ a műholdak helyzetétől is. Gyakran négynél több műholdra is rálátása van a vevőnek, ilyenkor numerikus szempontból az a legjobb, ha a négy, egymástól legtávolabbi, azaz a legnagyobb térszöget bezáró műholdat használja.

3.2 A GPS pontossága

A mérési eredményeket többféle, jellemzően zajjal modellezhető hiba, valamint szisztematikus hiba is terhelheti, melyek három fő forrásból származnak, ezek a műhold, a vevő valamint a terjedés által keletkezett hibák.

A szatellit eltérhet a becsült pályájától, ez eltérést okoz a trilateráció során, ez az úgynevezett ephemeris hiba. Minden eszköznél ugyanakkora eltolódást okoz, amennyiben ugyanakkora szögből látják a műholdat. A műhold és a vevő közti távolság

meghatározása időmérésen alapul, így nagyon pontos órákra van szükség, ezért a földi irányító központok rendszeresen korrigálják a szatellitek óráit, azonban így is előfordulhat eltérés. A GPS vevők olcsóbb órát tartalmaznak, így a pontosságuk sokkal kisebb mint a műholdaké. A műholdakról érkező jel visszaverődhet különböző tereptárgyakról, így megnő a terjedési ideje, valamint interferencia keletkezik, ami szisztematikus hibát okoz. A jel terjedése során az ionoszférában és a troposzférában bekövetkező sebesség változás miatt is keletkezhet eltérés a valódi pozíciótól [2]. A 3.1. táblázat összefoglalja a különböző forrásokból származó GPS hibákat és nagyságukat [4].

A hiba forrása	Eltérés nagysága [m]
Ionoszférikus hatás	± 5
Szatellit pályától való eltérése	$\pm 2{,}5$
Órajel hiba	± 2
Többutas terjedés hatása	± 1
Troposzférikus hatás	± 0.5
Kerekítési hiba	± 1

3.1. táblázat A GPS hibák forrása és nagysága

Ezeken kívül ritkábban előfordulhatnak durva hibák, melyek akár több száz vagy több ezer km nagyságúak lehetnek. A vevő egység hardveres megvalósítása miatt is keletkezhet zaj, ezeket okozhatja a jelfeldolgozás, a vevő órajelének hibája, és a nem megfelelő jel-zaj viszony.

A szisztematikus hibák hatása (pályahiba, órahiba, ionoszféra és troposzféra hatása) csökkenthető differenciális korrekcióval, azaz DGPS[5] alkalmazásával. A durva hibák és a véletlen hibák korrigálhatóak szűréssel és simítással. A korábban említett okokból a GPS vevőnek legalább négy műholdra rálátással kell rendelkeznie a pozíció meghatározáshoz. Ha ezek közül valamelyik eltűnik a látóhatár alatt, akkor ehelyett egy másik műholdat kell választani. Ezzel viszont a 3.2 fejezetben felsorolt szisztematikus hibák is ugrásszerűen változni fognak, mivel másik műhold pályáról, másik jelterjedési útról, valamint másik órajelről lesz szó. Az ugrásokkal méréstechnikai szempontból az a baj, hogy nem tudjuk, hogy előtte, vagy utána volt-e pontosabb a jel. Tehát olyan szűrő

[5] Differential Global Positioning System

alkalmazása célszerű, ami az ugrás előtti és utáni pontokat is figyelembe veszi, és úgy simítja egybe őket.

3.3 Adatfeldolgozás

Mivel a GPS vevő által mért jelhez zaj adódik, ezért célszerű elő-feldolgozást végezni mielőtt a szükséges információt megpróbáljuk kinyerni az adatokból, erre a szokásos módszer a szűrés és a simítás.

3.3.1 Szűrés

A szűrés eltávolítja a nem kívánt adatokat, az úgynevezett outlier értékeket, amik időben vagy térben jelentősen eltérnek a sorozattól és ezért nyilvánvalóan hibásak. Ki kell szűrni például a nem időrendben érkező adatokat, és a hirtelen bekövetkező távoli ugrásokat.

3.3.2 Simítás

A hibás adatok kiszűrése után következik a simítás, aminek célja a zaj csökkentése, egy közelítő függvénnyel. GPS adatsorok esetén gyakran alkalmaznak csúszóátlagos simítást, Gauss-Kernelt, és lokális regressziót.

A mozgó átlagos simítás esetén egyszerűen átlagoljuk az aktuális elem környezetében lévő elemeket a következő képlet alapján.

$$\hat{x}_t = \frac{x_t + x_{t-1} + \cdots + x_{t-n-1}}{n}$$

A becsült érték $\hat{x}_t$, a megfigyelt értékek x_{t-i}. A csúszó ablak méretét n határozza meg. Minél nagyobbra állítjuk a csúszó ablak méretét annál radikálisabb lesz a simítás.

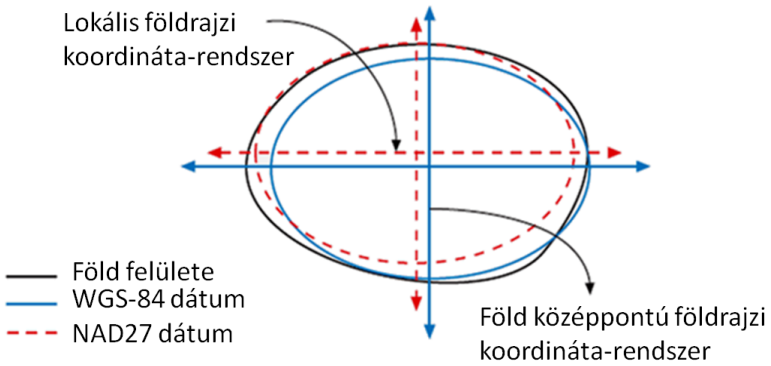
4 Térkép vetületek

Ahhoz, hogy a földrajzi pozíciónkat le tudjuk írni szükség van egy koordináta-rendszerre, amiben megadhatjuk a helyzetünket. Léteznek különböző két és három dimenziós koordináta-rendszerek és többféle referenciarendszer, amik viszonyítási alapot adnak a koordináta-rendszereknek.

A GPS vevő által szolgáltatott adatok WGS-84 dátum szerint adottak és a földrajzi koordinátákat tartalmazza, azaz a szélességi és hosszúsági fokokat. Ahhoz, hogy a Descartes-féle koordináta-rendszert alkalmazni lehessen, ezeket a koordinátákat egy síkbeli koordináta-rendszerben kell ábrázolni. A következő fejezet a különböző térképvetületeket és dátumokat ismerteti.

4.1 A geodéziai dátum

A földrajzi pozíciónk megadását nehezíti, hogy a föld felszíne nem egyenletes, ezért szükség van egy referenciafelületre, egy ellipszoidra, amivel közelíteni lehet a Föld felületét. Ha meghatározzuk ennek a felületnek az alakját, középpontját és a tengelyek irányultságát, akkor megkapjuk a geodéziai dátumot. Ez egy viszonyítási alap, egy referencia, amihez képest megadható egy-egy pozíció. Többféle dátum létezik, ezért a koordináták megadásakor fontos meghatározni, hogy a koordináta-rendszer melyik geodéziai dátumhoz kötődik [2]. A 4.1. ábra a WGS-84 és a NAD27 dátumot mutatja a föld helyzetéhez képest.



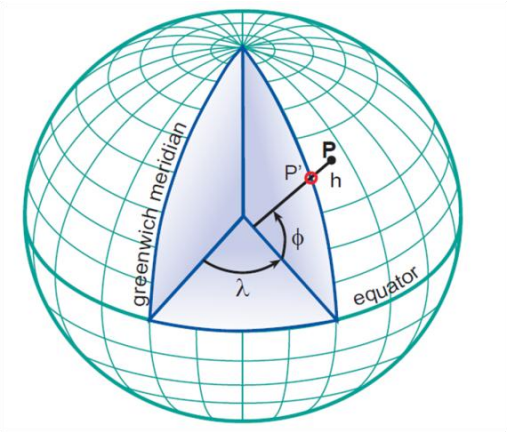
4.1. ábra Geodéziai dátumok helyzete a Földhöz képest [5]

A gyakorlatban alkalmaznak olyan dátumokat, amelyek a Földnek csak egy korlátozott részére vonatkoznak, ilyen például a NAD, de vannak olyanok is, melyek az

egész Földön alkalmazhatóak. Ilyen a GPS rendszerek által használt WGS-84, azaz World Geodetic System 1984, ami az egyik leggyakrabban alkalmazott dátum.

4.2 A földrajzi koordináta-rendszer

A GPS rendszerek által alkalmazott háromdimenziós földrajzi koordináta-rendszerben a referencia felület egy ellipszoid, a tengelyeket és a középpontot két sík jelöli ki, a meridián, valamint az egyenlítő által meghatározott sík. Ebben a rendszerben a föld bármelyik pontja leírható szélességi és hosszúsági fokok, valamint egy magasság érték segítségével.



4.2. ábra A 3D földrajzi koordináta-rendszer [6]

A ϕ földrajzi szélesség (latitude) megadja, hogy a föld középpontjától az egyenlítőhöz húzott szakasz mekkora szöget zár be a föld középpontjától az adott pontig húzott szakasszal. A λ földrajzi hosszúság (longitude) meghatározza egy pont meridián-síkja és a kezdőmeridián síkja által bezárt szöget.

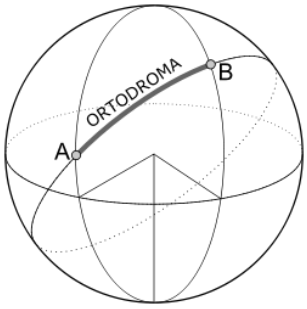
A földrajzi koordináta-rendszerben két pont közti távolság és irány kiszámítása bonyolult, mivel két pont között egy ívet kell meghatározni, nem pedig egy egyenest, így szükség van a következő alfejezetekben ismertetett függvényekre.

4.2.1 Távolság meghatározás

Földrajzi koordináta-rendszer alkalmazásakor a föld felületén történő távolság meghatározására nem alkalmazható a síkbeli koordináta-rendszer. Két szélességi és hosszúsági koordinátákkal adott pont ortodromikus távolságát a haversine formula adja meg. Az ortodroma, vagy más néven gömbi geodéziai vonal a földfelszín két pontja

közti legrövidebb távolságot határozza meg, tehát megegyezik a pontokat összekötő főkör egy szakaszával.

A 4.3 ábrán látható az ortodroma, azaz két pont közti legrövidebb út a föld felszínén.



4.3. ábra Ortodroma

A következő képlet a haversine formula, amely megadja két pont ortodromikus távolságát, amelyet d jelöl. Két pont földrajzi hosszúságának különbségét $\Delta\lambda$ jelzi, φ_i a földrajzi szélesség. A föld sugarát jelöli R, ennek értéke 6371 km. A szögtávolságot c jelöli.

$$a = \sin\left(\frac{\Delta\varphi}{2}\right) + \cos(\varphi_1) \cdot \cos\varphi_2 \cdot \sin^2\left(\frac{\Delta\lambda}{2}\right)$$

$$c = 2 \cdot \operatorname{atan2}\left(\sqrt{a}, \sqrt{1-a}\right)$$

$$d = R \cdot c$$

4.2.2 Irány meghatározás

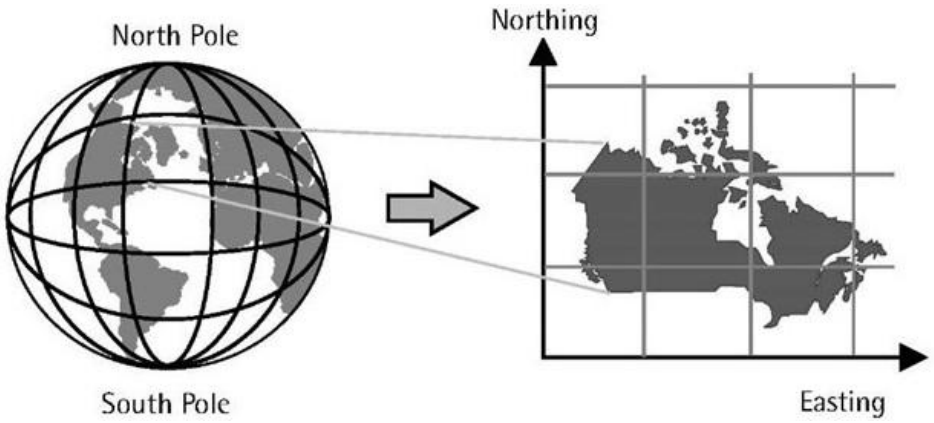
A következő formula azt adja meg hogy a (λ_1, φ_1) pontból a(λ_2, φ_2) felé tartva a kapott szakasz hány fokot zár be az észak felé tartó egyenessel.

$$\theta = atan2(\sin(\Delta\lambda) \cdot \cos(\varphi_2), \cos(\varphi_1) \cdot \sin(\varphi_2) - \sin(\varphi_1) \cdot \cos(\varphi_2) \cdot \cos(\Delta\lambda))$$

4.3 Térkép vetületek

A 4.2 fejezetben ismertetett földrajzi koordináta-rendszerben történő számítás nehézkes. A feladat elvégzését egyszerűsítené, ha Descartes-féle koordináta-rendszerben lehetne műveleteket végezni, ezért érdemes valamilyen vetületet alkalmazni.

A térkép vetület célja, hogy egy sík koordináta-rendszerben ábrázolja egy térbeli alakzat pontjait, ahogy a 4.4. ábra is mutatja. Azt a transzformációt, amellyel a Föld felületének egy részét egy sík felületre képezzük vetítésnek hívjuk. A vetítés elengedhetetlen térképek készítéséhez.



4.4. ábra A térkép leképezés célja [2]

Nagyon sokféle projekció létezik, mindegyiknek megvan a maga előnye és hátránya, de tökéletes leképezés nyilvánvalóan nem hozható létre. Ez olyan mintha a narancs héját próbálnánk meg kiteríteni, anélkül hogy elszakadjon, kinyúljon vagy bármilyen egyéb módon torzuljon. Emiatt mindig a célnak legmegfelelőbb leképezést kell választanunk, kompromisszumot kell kötnünk, tekintettel arra, hogy milyen térképi tulajdonságot szeretnénk megtartani minimális torzulással és melyek azok a tulajdonságok, amelyek kevésbé fontosak, ezek még jobban eltorzulhatnak a transzformáció során. A legfontosabb tulajdonságok, amiket egy térképen mérni lehet a következők:

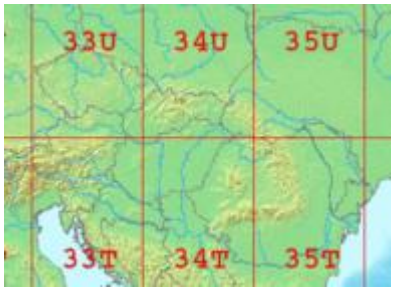
- alak
- méret
- irány
- távolság
- arány

A feladat legfontosabb részei a sortávolságok meghatározása és a területszámítás. A területszámításhoz érdemes területtartó leképezést alkalmazni. A sortávolságok meghatározásához párhuzamos szakaszok közti távolságokat kell mérni, ehhez irány és távolságtartó leképezés szükséges.

4.4 Univerzális Transzverzális Mercator

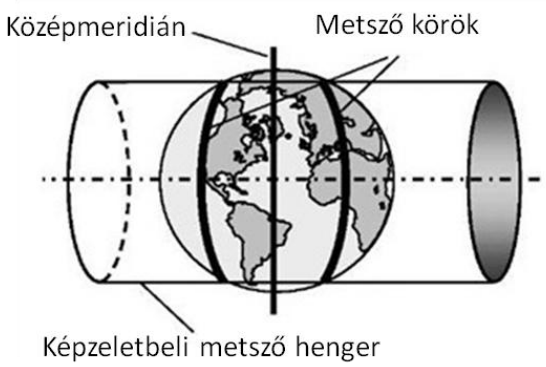
Az UTM az egyik legelterjedtebb leképezés, melyet világszerte alkalmaznak. Egyik előnye, hogy szögtartó, kis méretű területek estén alaktartó is, ezért megfelelő a feladat elvégzésére, mivel a párhuzamos sorokat megtartja párhuzamosnak, így Descartes-féle koordináta-rendszerben lehet számolni és egyszerűen lehet párhuzamos sorokat keresni. Az UTM leképezés a középmeridián mentén távolságtartó, az egyes zónákon belül kis torzítás lép fel a terület leképezés során. A skálafaktor pedig nem lépi túl a 0.1%-ot egy adott zónán belül [5].

Az UTM leképezés egy összetett transzformáció. Először 60 hosszúsági zónára osztja a földet, melyek egyenként 6 fokosak és számozásuk 1-től 60-ig terjed. A zónákat vízszintes irányban 8 szélességi fokonként 20 sávra osztja, ezek jelölésére az ABC betűit használja. A 4.5 ábrán látható Magyarország és környékének UTM felbontása.



4.5. ábra UTM zónák

Mindegyik zóna rendelkezik egy középmeridiánnal, ami a zóna közepén található. Majd mindegyik zónában transzverzális Mercator leképezést végez, ami transzverzális, metsző hengervetületet alkalmaz, ez látható a 4.6 ábrán.



4.6. ábra Transzverzális metsző hengervetület

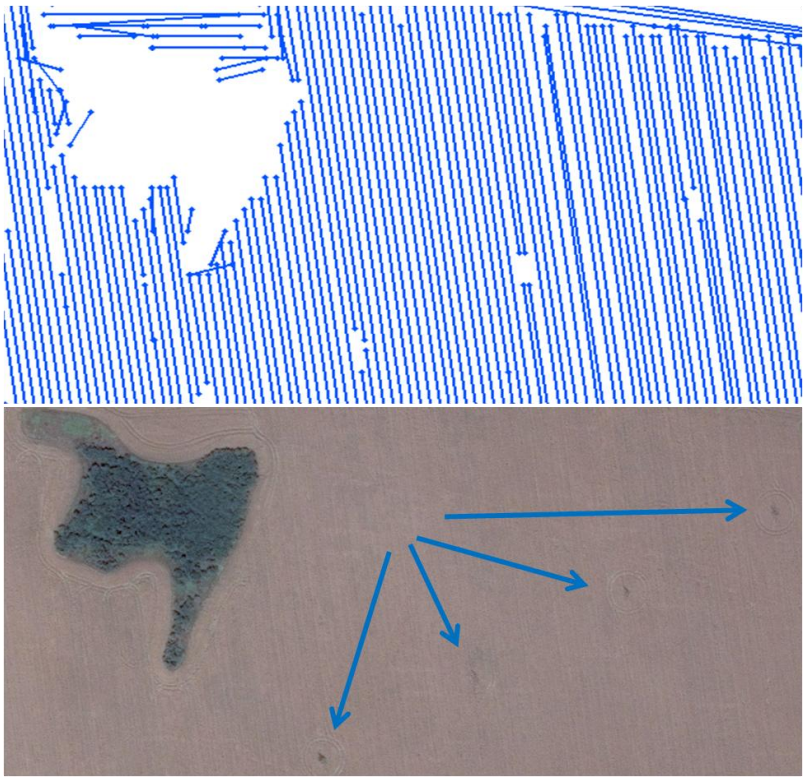
5 Kezdeti próbálkozások

Miután a rendelkezésre álló adatsoron elvégeztem a 3. fejezetben ismertetett szűréseket és a 4. fejezetben részletezett UTM vetületbe történő leképezést, következhet a lényeges információ kinyerése az adatokból. A cél az egyes munkafolyamatok elkülönítése, a megművelt és kihagyott föld darabok területének meghatározása, valamint a sortávolságok kiszámítása. Ezekhez az első lépés a táblákon végzett munka és a táblákhoz történő vonulás elkülönítése. Ez a fejezet ismertet egy módszert, aminek segítségével ez elvégezhető.

5.1 Egyenes szakaszok keresése

A táblák sorainak megtalálásához először is egyenes szakaszokat kell keresni az adatsorban. Egyenes szakaszok esetén az egymás után következő pontpárok által meghatározott vektorok iránya csak kis mértékben tér el egymástól, fordulók esetén viszont nagy az eltérés. Így az egyenes szakaszok detektálásához végig kell haladni az adatokon és kiszámítani az irány átlagértékét. Ha az adott irány és az eddig számolt átlagérték különbsége elér egy küszöbszintet, akkor fordulóhoz értünk. Ilyenkor az eddigi koordinátákat el kell tárolni egy szegmensként. Majd a többi adatra ugyanezt a műveletet végre kell hajtani.

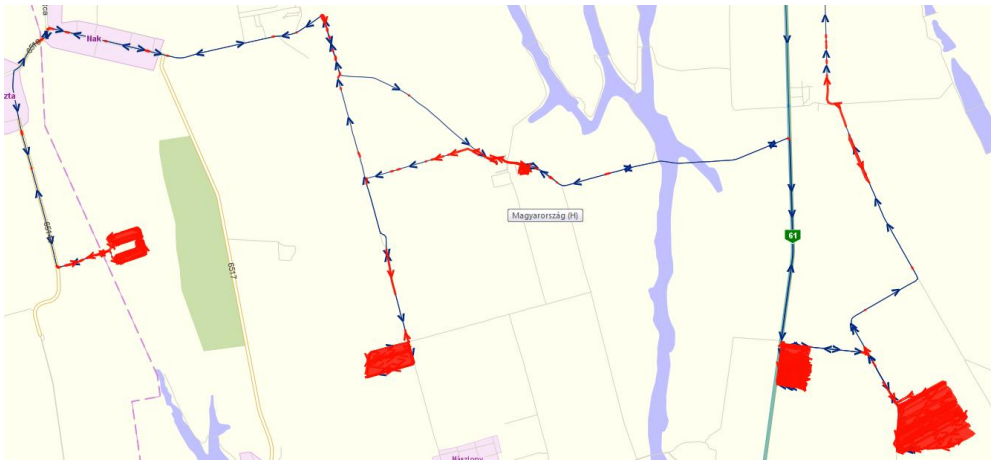
Az 5.1. ábra egy tábla részletet tartalmaz, amin végrehajtottam az előző műveletet, tehát fel van bontva egyenes szakaszokra. Látható, hogy vannak olyan sorok, amelyek több részből állnak. Megvizsgálva az adott táblához tartozó térképet látható, hogy a táblán több akadály, valószínűleg távvezeték pózna is áll, amelyeket a munkavégzés során ki kellett kerülni. Ezt a térképrészletet mutatja az 5.1. ábra.



5.1. ábra Egyenes szakaszokra osztott táblarészlet a hozzá tartozó térképpel

5.2 Sebességprofil figyelembe vétele

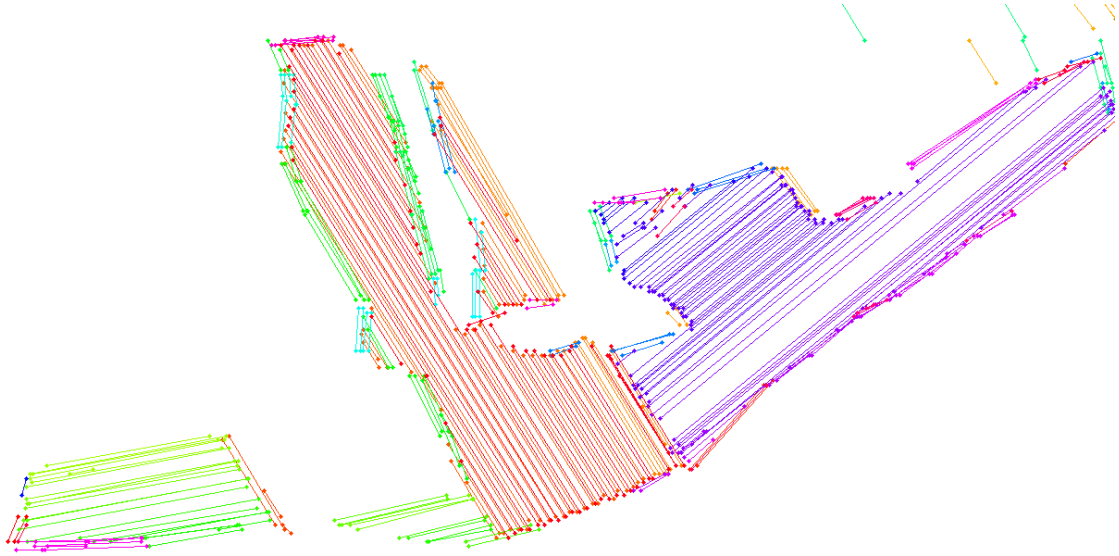
Abban az esetben, ha olyan járművet vizsgálunk, amelynek eltér a munkavégzési sebessége a vonulásitól, akkor figyelembe vehető a sebessége is a táblák elkülönítésénél. Ebben az esetben meg kell határozni egy sebességet, amely fölött már elmondható, hogy biztosan nem végez szántóföldi munkát az adott gép, így egyszerűen több kisebb részletre osztható az adatsor és rögtön ki lehet zárni a vonulás egy részét. A 5.2. ábra egy gép mozgását mutatja, a pirossal jelölt részek a kiválasztott sebesség alatt vannak.



5.2. ábra Sebesség-szűrés

5.3 Párhuzamos szakaszok keresése

Miután sikerült egyenes szakaszokra felbontani az adatsort, minden egyes megtalált szakaszon lineáris regressziót kell végrehajtani. Ezután a kapott szakaszokat párhuzamos csoportokra kell osztani. A sorok valójában nem teljesen párhuzamosak, mivel a munkavégzés nem tökéletes, ezért akkor mondható, hogy két szakasz párhuzamos, ha a rájuk illesztett egyenesek által bezárt kisebbik szög értéke kisebb mint egy meghatározott tolerancia érték, ezt 10°-ra állítottam be. Egy jármű mozgásának párhuzamos csoportokra osztott szakaszait tartalmazza az 5.3. ábra.



5.3. ábra Párhuzamos csoportokra osztott adatsor

5.4 Táblák elkülönítése

A táblák elkülönítéséhez a megtalált párhuzamos sorok között szabályos mintákat kell keresni. Akkor mondható, hogy egy táblát találtunk, ha van legalább 4 sora, a sorok közti távolság egy bizonyos értéktartományon belül van (kisebb mint 40 méter), valamint az átlagos sortávolság legalább 2 méter. A sorok közti távolságok meghatározásához azonban először meg kell keresni a szomszédos sorokat. Ennek a megvalósítását részletesen bemutatja a 9. fejezet.

5.5 A módszer elemzése

Több adatsoron is teszteltem az algoritmust mind futási idő, mind hatékonyság szempontjából.

5.5.1 Futási idő

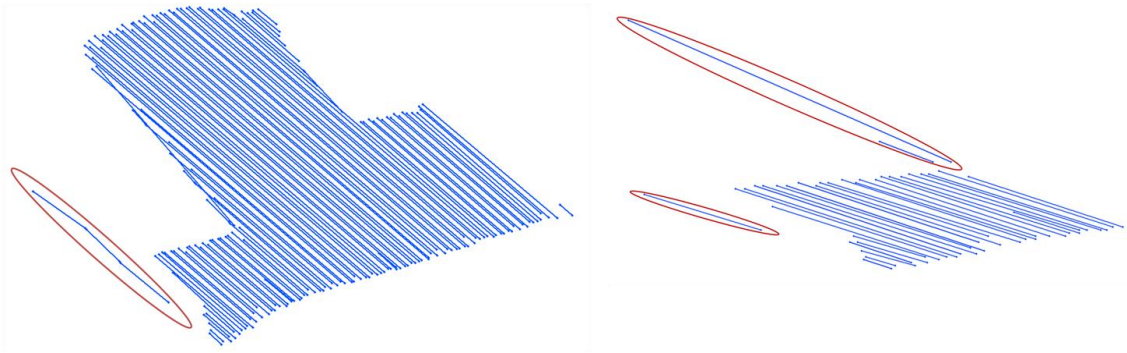
A 5.1. táblázat az egyes műveletek futási idejét mutatja három különböző méretű adatsor esetén, a futási idők ms egységben értendők. Látható, hogy nagy mennyiségű adat feldolgozása is viszonylag gyorsan megtörténik, így futási idő alapján a gyakorlatban is alkalmazható a módszer.

Adatok száma / Művelet	83119	101716	81033
Simítás	109	124	102
Felbontás sebesség alapján	360	435	330
Egyenes szakaszok keresése, regresszió	717	1030	717
Párhuzamos szakaszok keresése	31	16	16
Sorok rendezése	16	46	31
Sortávolságok meghatározása	31	31	47
Összesen	1264	1682	1243

5.1. táblázat Futási idők a képfeldolgozási műveletek nélkül

5.5.2 Hatékonyság

Gyakran előfordul, hogy olyan szakaszok is a tábla sorai közé kerülnek, amelyeknek nem kellene, mert nyilvánvalóan a vonuláshoz tartoznak. Az 5.4. ábra két ilyen táblát is mutat. A pirossal bejelölt szakaszok megfelelnek a kritériumoknak, tehát párhuzamosak a tábla soraival és megfelelő távolságon belül vannak, viszont ránézésre egyértelműen meg lehet állapítani, hogy ezek valójában a vonuláshoz tartoznak, nem pedig a tábla részei.



5.4. ábra Hibásan elkülönített táblák

Vannak olyan permetezőgépek, amelyek vonulási és munkavégzési sebessége között nincs jelentősebb eltérés. Így a sebesség alapján történő felosztás sem vezet eredményre.

6 Táblák elkülönítése morfológia segítségével

Az előzőekben bemutatott módszer a táblák és a vonulás elkülönítésére nem mindig jár sikerrel, gyakran kerül be a tábla pontjai közé néhány, a sorokkal párhuzamos útszakasz. A szükséges feltételek meghatározása is bonyolult, pedig ránézésre nagyon egyszerűen el lehet dönteni, hogy hol vannak a táblák és az utak. Éppen ezért érdemes a számítógépes látórendszerek módszereit alkalmazni a táblák megkeresésére. Morfológiai műveleteket alkalmaztak már multi-track GPS adatsorok feldolgozására térképek generálásához is [18].

Ez a fejezet ismerteti a GPS adatsorok képpé alakítását, az úgynevezett raszterizálást, aminek hatására képfeldolgozási módszerek segítségével el lehet különíteni a táblán végzett műveleteket a vonulástól.

6.1 Raszterizálás

Ahhoz, hogy morfológia műveletek segítségével el tudjuk különíteni a táblákat, először is létre kell hoznunk a képet. Rendelkezésre állnak az adatsorok x, y koordinátái, amelyeket a szélességi és hosszúsági fokok UTM értékekké alakításával kaptunk. Ezeket kell sor és oszlopindexekké alakítanunk.

Fontos lépés a megfelelő felbontást meghatározása, azaz el kell dönteni, hogy egy pixel szélessége (vagy magassága) a valóságban hány méternek feleljen meg. A felbontást nem szabad túl nagyra állítani, mivel a mezőgazdasági gépek egynapi munka során akár 50-60 km széles területet is bejárhatnak, így például egy 1m/px-es felbontás esetén 50000 oszlopos mátrixot kellene létrehozni. A felbontás túl kicsi sem lehet, mivel akkor előfordulhat, hogy egy teljes tábla egy pixelre esik. Ezért kezdetben érdemes 20m/px felbontást választani. Ez alapján a mátrix sorainak és oszlopainak száma:

$$N = \frac{height}{resolution}$$

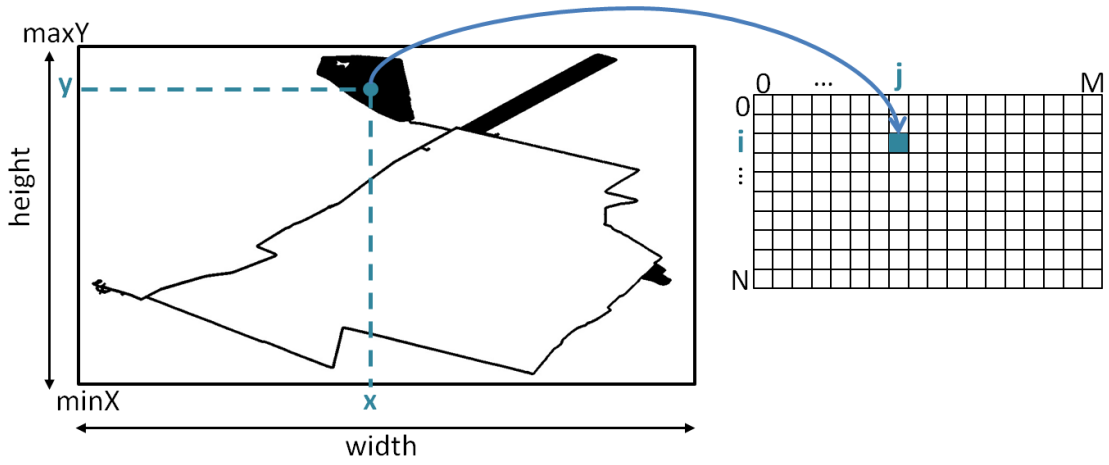
$$M = \frac{width}{resolution}$$

Jelölje az adatsor egy koordinátáját P(x,y), a mátrixban lévő pixelt pedig I(i, j). Először is meg kell határozni az útvonalat befoglaló téglalapot, az úgynevezett bounding boxot. Ehhez meg kell keresni az x és y koordináták között is a legkisebbet és a legnagyobbat. A befoglaló téglalap a mátrix méretének meghatározásához szükséges. A 6.1. ábra jelöléseit használva a P(x,y) ponthoz tartozó mátrixpozíció meghatározása a következő képletekkel lehetséges.

$$i = \frac{maxY - y}{resolution}$$

$$j = \frac{x - minX}{resolution}$$

A 6.1. ábra bal oldalán egy táblatérkép található, jobb oldalán pedig a belőle létrehozott mátrix.



6.1. ábra Mátrix létrehozása az adatsorból

A raszterizálást megvalósítására létrehoztam a Rasterizer osztályt, ami egy mátrixot épít a koordinátákból. A mátrix létrehozását és a képfeldolgozási műveleteket az OpenCV képfeldolgozást megvalósító könyvtár alkalmazása segítségével valósítottam meg, ami ismertetésre kerül a 11.1 fejezetben. Az osztály tagváltozói a mátrix mérete, azaz a sorok és oszlopok száma, a térképet körülvevő téglalap adatai (minX, maxY), valamint a létrehozott mátrix.

```
class Rasterizer
{
      int rowNum;
      int colNum;
      int extendedRowNum;
      int extendedColNum;
      double minX;
      double maxY;
```

```
        double resolution;
        double resolutionReciprok;
        Mat image;
public:
        Rasterizer(double minX, double minY, double maxX, double maxY, double
                resolution);
        void addElement(double x, double y);
        void tableContoures(int strElSize, vector<vector<Point_<double>> >&
                tContoures);
        void convertContour(vector<vector<Point> >& contoursPx,
                vector<vector<Point_<double>> >& contours);
        void paintContoures(Size size, vector<vector<Point> >& contours,
                Mat& hierarchy, vector<vector<Point> >& reducedContours);
};
```

A konstruktornak meg kell adni a térképet körülvevő négyszög szélső határait, azaz a legkisebb és legnagyobb x, valamint y irányú koordinátákat, valamint a választott felbontást, ez létrehozza a megfelelő méretű bináris mátrixot. A mátrix létrehozásához az OpenCV képfeldolgozást megvalósító könyvtár Mat objektumát alkalmaztam. A sorok és oszlopok számát is néhány pixellel nagyobbra kell állítani, azért, hogy a később végrehajtott műveletek során a térkép szélén ne torzuljanak az alakzatok.

```
Rasterizer::Rasterizer(double minX, double minY, double maxX, double maxY,
        double resolution) : resolution(resolution), minX(minX), maxY(maxY) {
                resolutionReciprok = 1.0 / resolution;
                rowNum = (maxY - minY) / resolution + 5;
                colNum = (maxX - minX) / resolution + 5;

                extendedRowNum = rowNum + 10;
                extendedColNum = colNum + 10;
                Mat im(extendedRowNum, extendedColNum, CV_8UC1, Scalar(0));
                image = im;
}
```

Az addElement() függvény segítségével lehet koordinátákat hozzáadni a mátrixhoz. A függvény bemenete az elhelyezni kívánt koordináta x és y értéke. A sor- és oszlopindex meghatározása után a mátrix megfelelő celláját beállítja a függvény, azaz fehérre színezi.

```
void Rasterizer::addElement(double x, double y) {
        int rowIndex = (maxY - y) * resolutionReciprok;
        int colIndex = (x - minX) * resolutionReciprok;

        int extendedRowIndex = rowIndex + 5;
        int extendedColIndex = colIndex + 5;

        if (extendedRowIndex >= 0 && extendedRowIndex < extendedRowNum &&
                extendedColIndex >= 0 && extendedColIndex < extendedColNum)
        {
                uchar* p = image.ptr<uchar>(extendedRowIndex);
                p[extendedColIndex] = 255;
        }
}
```

Az OpenCV könyvtár referencia számlálást alkalmaz a létrehozott mátrixok nyilvántartására, ezért a mátrixok destruktora automatikusan meghívásra kerül, amint az összes rájuk történő hivatkozás megszűnik, így a Rasterizer osztálynak nem kell a mátrix törlésével foglalkoznia.

A tableContoures() függvény elvégzi a táblahatárok megkeresését képfeldolgozási műveletek segítségével, a convertContoure() pedig a megtalált kontúrokat visszaalakítja pixel értékekről koordinátákká. Ezek a műveletek részletesen ismertetésre kerülnek a 8. fejezetben.

Raszterizálás alkalmazása esetén is érdemes figyelembe venni a járművek sebességét, tehát az 5.2 fejezetben említett módon célszerű beállítani egy küszöbsebességet. Nyilván nem mondható az, hogy ahol a küszöbérték alá kerül a gép sebessége ott munkát végez, de így kisebb darabokra vágható az adatsor, ezáltal kisebb mátrixokat kell létrehozni. Így akár nagyobb felbontás is beállítható, továbbá gyorsul a képen végzett műveletek sebessége. A 6.2. ábra egy jármű napi munkavégzését tartalmazza, piros szín jelöli a sebességkorláton belüli (17 km/h) mozgást. Látható, hogy így több kisebb mátrixot kell létrehozni egy nagy helyett.

6.2. ábra Sebesség-szűrés

Miután az adatsor minden elemére megtörtént a leképezés, tehát létrejött egy bináris (fekete-fehér) kép, elvégezhető a táblák elkülönítése képfeldolgozási módszerek segítségével. Az ehhez szükséges morfológiai műveleteket a következő fejezet ismerteti.

7 A képfeldolgozás elméleti háttere

A 6. fejezet bemutatta hogyan történik a raszterizálás, tehát hogyan lehet egy koordináta sorból létrehozni egy képet. Ez a fejezet ismerteti a különböző képfeldolgozási módszereket, amelyek segítségével el lehet különíteni azokat a területeket, ahol munkavégzés történt.

Mivel a raszterizálás során a táblaadatokból létrehozott képek csak kétféle értéket vehetnek fel, az alapján, hogy az adott pontban járt-e a jármű, ezért a következőkben csak a bináris képeken végezhető műveletek kerülnek ismertetésre.

Bináris képek esetén minden pixel kétféle értéket vehet fel, ezek a 0 és az 1. Konvenció szerint általában feketével jelenítjük meg a hátteret, fehérrel az alakzatokat. A bináris képeken végezhető alapvető műveleteket két csoportra oszthatjuk, a képek között végezhető logikai műveletekre, és a morfológiai műveletekre, amely egy képen belüli pixeleket önállóan módosítja.

A morfológia vagy matematikai morfológia a térbeli alakzatok analízisének elmélete. A morfológia elnevezés arra utal, hogy alakzatokat, objektumokat elemezünk a segítségével, és azért matematikai, mert alapvetően halmazelméletre, Bool-algebrára, valamint integrál-geometriára épül. A morfológia egy hatékony eszköz a képfeldolgozás során [15].

7.1 A morfológia alapjai

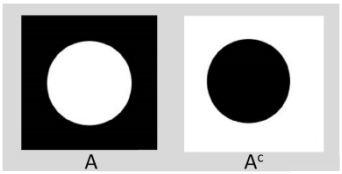
Bináris képeken végezhetünk különböző halmaz műveleteket. A matematikai morfológiában a halmaz azokat az alakzatokat képviseli, amelyek a képen az előtérben szerepelnek [16].

A leggyakoribb morfológiai műveletek az erózió, a dilatáció, a zárás, a nyitás, valamint a kontúrkeresés. Ezeknek a bemutatásához szükség van néhány halmazművelet [17], valamint a strukturáló elem ismertetésére, mivel az alapvető morfológiai műveletek, ezeknek a kombinációjára épülnek. A következő ábrákon a képfeldolgozás során használt konvenció szerint a fekete pixelek 0 értékűek, azaz a hátteret jelentik, a fehérek az előtér elemei, azaz 1 értékűek.

7.1.1 Komplemens

Az A halmaz komplemense azoknak a pixeleknek a halmaza, amelyek nem tartoznak A-hoz.

$$A^c = \{c | c \notin A\}$$

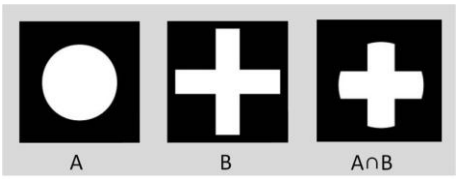


7.1. ábra Komplemens

7.1.2 Metszet

Az A és B halmaz metszete azoknak a pixeleknek a halmaza, amelyek mindkét halmazhoz hozzátartoznak.

$$A \cap B = \{c | ((c \in A) \wedge (c \in B))\}$$

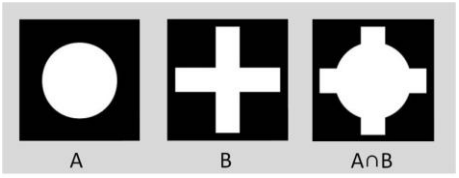


7.2. ábra Metszet

7.1.3 Unió

Az A és B halmaz uniója azokat a pixeleket tartalmazza, amelyek legalább az egyik halmazhoz hozzátartoznak.

$$A \cup B = \{c | (c \in A) \vee (c \in B)\}$$

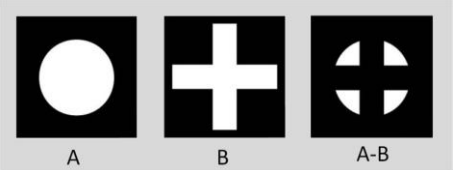


7.3. ábra Unió

7.1.4 Különbség

Az A és B halmaz különbsége azokat az elemeknek a halmaza, amelyek szerepelnek A-ban, viszont B-ben nem.

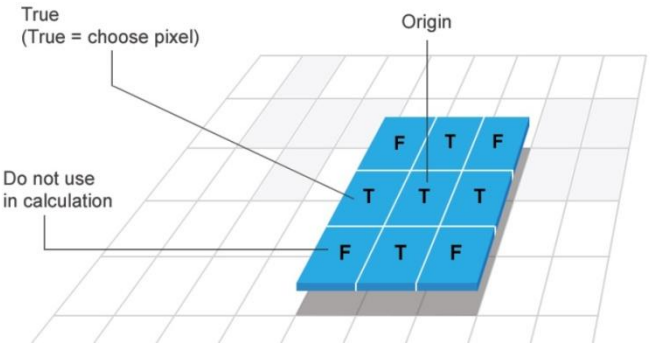
$$A \backslash B = \{c | (c \in A) \wedge (c \notin B)\}$$



7.4. ábra Különbség

7.2 A strukturáló elem

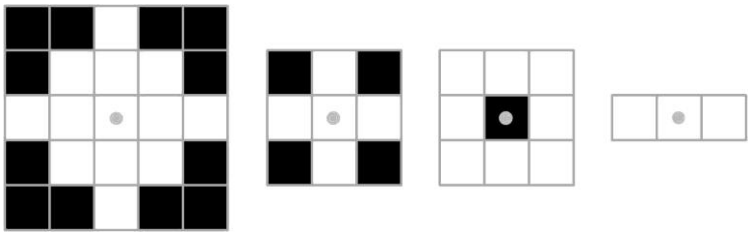
A morfológiai műveletek lényege, hogy pixeleket adunk hozzá vagy veszünk el a képből, bizonyos szabályok alapján, amiket az adott pixel szomszédain végzünk el. A műveletekhez használt szomszédos pixelek kijelölését a strukturáló elem, vagy más néven kernel végzi. A strukturáló elem egy két dimenziós bináris maszk, meghatározott alakkal, mérettel és origóval. A műveleteket a kép minden pixelére el kell végezni, ezt úgy hajtjuk végre, hogy a kernelt végigléptetjük az adott képen. Az elem origója jelöli ki az aktuális pixelt, amit módosítani szeretnénk, az alakja és tartalma pedig meghatározza a vizsgált szomszédokat.



7.5. ábra Strukturáló elem

A legtöbb művelet esetén (pl. a később részletezett erodálás és dilatáció) minél nagyobb egy strukturáló elem annál hosszabb ideig tart a művelet elvégzése. Emellett minél nagyobb az elem mérete annál bővebb szomszédságot veszünk figyelembe. A leggyakrabban alkalmazott méretek a 3x3, 5x5, 7x7. Rengeteg féle kernel létezik, a célnak megfelelően kell kiválasztani, hogy melyiket használjuk. A 7.6. ábra bemutat néhány gyakori strukturáló elemet, ezek a gyémánt, a kereszt, a négyzet és a vonal. Az ábrán a fekete cellák nem vesznek részt a morfológia műveletben, ez azt jelenti, hogy, ha a kernelt ráhelyezzük egy képre, akkor az adott képnek csak azok a pixelei vesznek részt a műveletben, amelyek az elem fehér (1-es értékű) cellái alá kerültek. Az

elemekben szereplő kör jelöli az origót. A harmadik elemhez hasonlóan előfordulhat az is, hogy az origóban szereplő elem nem vesz rész a műveletben.



7.6. ábra Néhány gyakori strukturáló elem

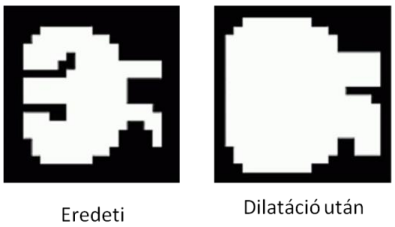
7.3 Dilatáció

A dilatációt leíró műveletet a Minkowski-összeg segítségével fejezhetjük ki. A képletben A jelöli a képet, amin a műveletet elvégezzük, B pedig a strukturáló elemet [17].

$$A \otimes B = \{c | c = a + b, a \in A, b \in B\} = \bigcup_{b \in B} (A)_b = \bigcup_{a \in A} (B)_a$$

Dilatáció esetén az adott pixel kimenetét úgy kapjuk meg, hogy vesszük a pixelnek a strukturáló elem által kijelölt szomszédait, majd megkeressük köztük a maximum értéket. Ez bináris képek esetén nyilvánvalóan 0 vagy 1 lesz. Tehát, ha a kijelölt szomszédai között van 1-es értékű, akkor ez a pixel is ezt az értéket kapja meg.

A dilatáció általában megnöveli az objektumok méretét és eltünteti az objektumokon belüli apróbb réseket.



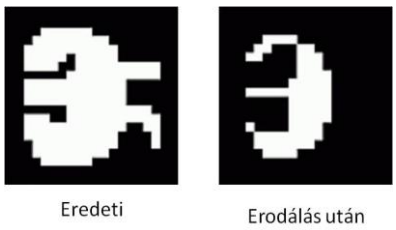
7.7. ábra Dilatáció

7.4 Erózió

Az eróziót leíró műveletet a Minkowski-különbség segítségével fejezhetjük ki. A képletben A jelöli a képet, amin a műveletet elvégezzük, B pedig a strukturáló elemet.

$$A \ominus B = \{c | (B)_c \subseteq A\}$$

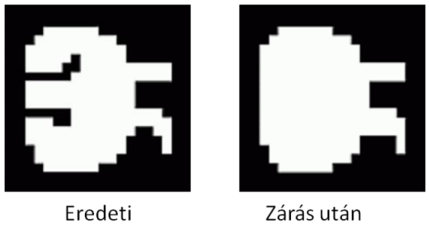
Erózió esetén az adott pixel kimenetét úgy kapjuk meg, hogy vesszük a pixelnek a strukturáló elem által kijelölt szomszédait, majd megkeressük köztük a minimum értéket. Ez bináris képek esetén nyilvánvalóan 0 vagy 1 lesz. Tehát, ha a kijelölt szomszédai között van 0 értékű, akkor ez a pixel is ezt az értéket kapja meg. A strukturáló elemet végig kell léptetni minden egyes pixelen és elvégezni az előbb leírt műveletet. A léptetést transzlációnak nevezzük, ezt jelöli a képletben a $(B)_c$. Az erózió általában csökkenti az objektumok méretét és eltünteti az apróbb részeket a képről.



7.8. ábra Erózió

7.5 Zárás

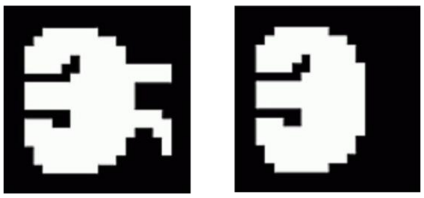
Zárásnak nevezzük, ha végrehajtunk egy dilatációt, majd ezt követően ugyanazzal a strukturáló elemmel végzünk egy eróziót is. A zárás név elég beszédes, arra utal, hogy a művelet összezárja a kis réseket az objektumon belül.



7.9. ábra Zárás

7.6 Nyitás

Nyitásnak nevezzük, ha végrehajtunk egy eróziót, majd utána egy dilatációt ugyanazt a strukturáló elemet felhasználva. A nyitás név arra utal, hogy a művelet felnyitja a vékony résszel összekötött objektumokat. A nyitás során eltűnnek a kis zajok a képről, viszont a megmaradt objektumok körvonala nem tűnik el.



7.10. ábra Nyitás

7.7 Hatékonyság

Az alapvető képfeldolgozási műveletek, az erózió és a dilatáció futási ideje lineárisan növekszik a kép pixeleinek számával és a strukturáló elem méretével. Tehát a szükséges lépésszám O(npq), ahol n a kép pixeleinek száma, p és q pedig a strukturáló elem mérete. Sebességnövekedést lehet elérni a strukturáló elem felbontásával és a művelet többszöri végrehajtásával.

7.8 A morfológiai műveletek végrehajtása

A morfológiai műveletek végrehajtásának célja, hogy az egy táblához tartozó pontok egy csoportba kerüljenek, viszont elkülönüljenek egymástól a táblák és a táblákhoz vezető utak.

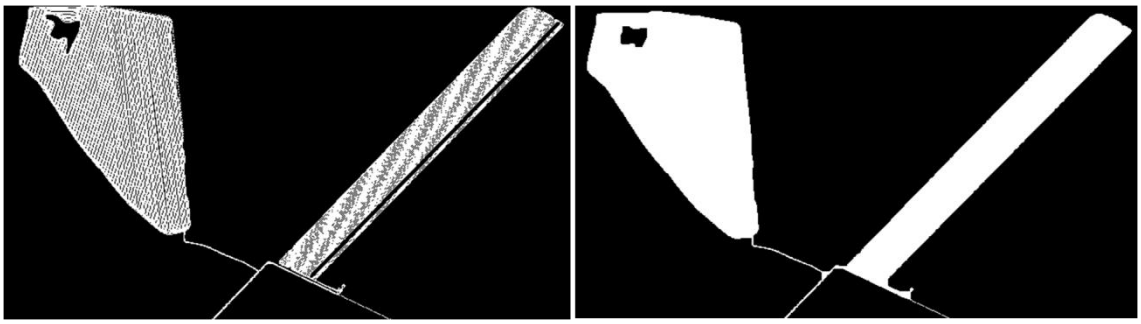
A műveletek elvégzéséhez először egy megfelelő strukturáló elemet kell választani. Tegyük fel, hogy két sor addig tekinthető szomszédosnak, amíg a köztük lévő távolság kisebb mint 40 méter. Ha a felbontás 10 m/px, akkor annak érdekében, hogy ezeket egy kernel le tudja fedni, annak 4 pixel szélesnek kell lennie. Tehát a strukturáló elem mérete függ a felbontástól, valamint attól, hogy milyen távolságot választunk maximális sortávnak. Jelölje a strukturáló elem méretét n, a maximális megengedhető sortávolságot egy táblán belül d, a felbontást pedig r. Ezek alapján a strukturáló elem mérete meghatározható a következő képlettel.

$$n = \frac{d}{r}$$

A létrehozott mátrixon először is zárást kell végezni, azaz egymás után ugyanazzal a strukturáló elemmel egy dilatációt, majd egy eróziót. A következő ábrákon egy adatsoron végzett művelet részlete látható.[6] Az ábra tartalmazza a művelet

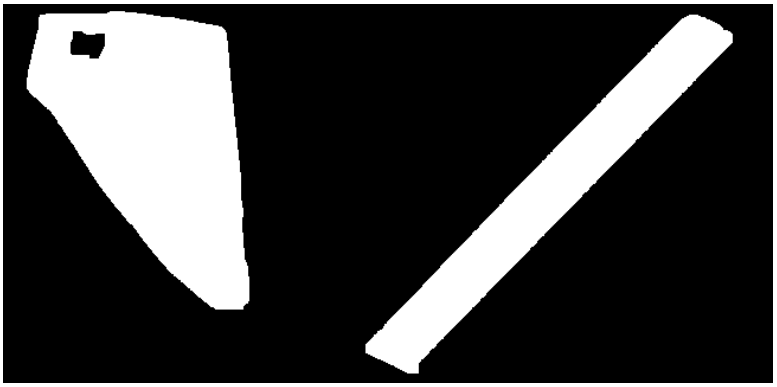
[6] A függelék tartalmaz további táblaképeket a rajtuk végzett morfológiai műveletekkel együtt

eredményét, az egy táblába tartozó pontok összefüggő táblává váltak, mivel a zárás eltünteti az apró réseket. Ennek köszönhetően az egyes pontok és a sorok összeolvadtak egy alakzattá.



7.11 A zárás eredménye

Ezután egy nyitást kell végrehajtani, ami a szétválasztja a kis területen összefüggő alakzatokat, és eltünteti az apró részeket. A 7.12. ábra a nyitás eredményét mutatja, elkülönült egymástól a két tábla, valamint eltűntek a hozzájuk vezető utak.



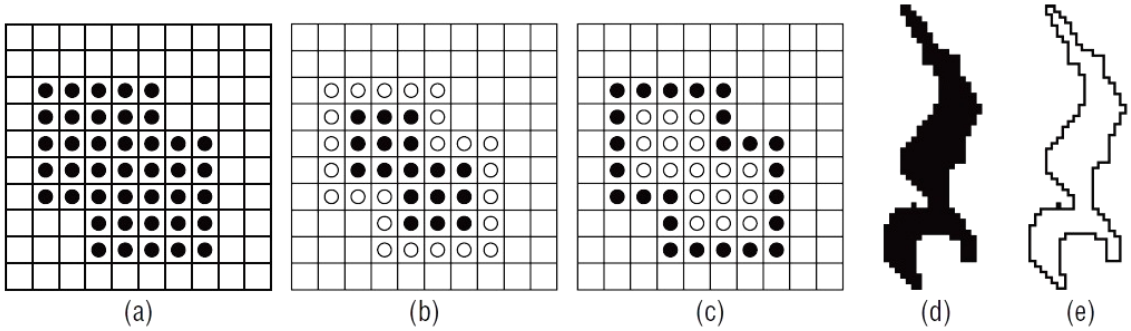
7.12. ábra A nyitás eredménye

8 Táblahatárok meghatározása kontúrkereséssel

Az utak eltávolítása után következik a táblahatárok, azaz a táblák körvonalának megkeresése. A körvonal azoknak a pixeleknek a halmaza, amelyek egy alakzathoz tartoznak és van legalább egy olyan szomszédjuk, ami a háttér része. Képfeldolgozás során gyakran van szükség egy kép körvonalának meghatározására.

8.1 Kontúr megjelenítése

Mivel egy vizsgált pixelnek akármelyik szomszédja tartozhat a háttérhez, így nem tudunk létrehozni egyetlen olyan strukturáló elemet, amellyel erodálást vagy dilatációt végezve rögtön elkülöníthető lehetne a körvonal. Azonban, ha a 8.1. ábra alapján elvégzünk egy eróziót (b), majd az így kapott eredményt kivonjuk az eredeti képből (c), akkor megkapjuk az alakzatok körvonalát.



8.1. ábra A kontúr meghatározás lépései [17]

Halmazelmélet segítségével ez a következő képlettel fejezhető ki, ahol C jelöli a kontúrt, A az eredeti képet, B pedig a strukturáló elemet.

$$C = A - (A \ominus B)$$

Másképpen megfogalmazva:

$$C = A \cap (A \ominus B)^c$$

8.2 Kontúrkeresés

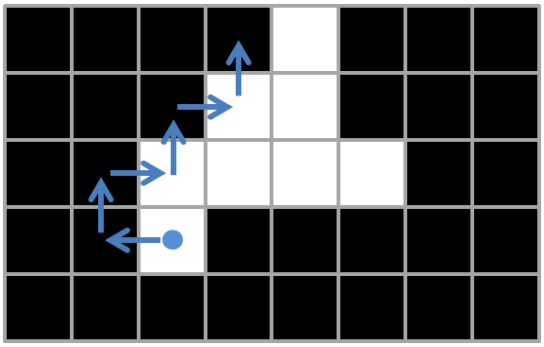
A kontúr vizuális megjelenítésén túl gyakran szükség van arra is, hogy összefüggő objektumokat keressünk egy képen és kinyerjük ezeknek a pozícióját. A

kontúr fontos információt hordoz az alakzatról, amit körülvesz. Segítségével végezhető például kerület- és területszámítás, objektum-felismerés, valamint eltárolható az alakzat.

Egy alakzat körvonalának megkeresésére többféle határkövető algoritmus is rendelkezésre áll. Ezeknek az ismertetéséhez először is tisztázni kell, hogy mit tekintünk egy pixel szomszédságának. Megkülönböztetjük a Moore- és a von Neumann-szomszédságot. Moore-szomszédság esetén egy pixel szomszédai közé csak azok a pixelek tartoznak, amelyekkel közös élen osztoznak. Von Neumann-szomszédság esetén nem csak a közös éllel, de a közös csúccsal rendelkező pixelek is szomszédosnak számítanak.

8.3 Egyszerű határkövető algoritmus

Az egyszerű határkövető algoritmus (square-tracking algorithm) egy előzőleg megtalált határpixelről indul. Ezután tovább kell lépkedni az alapján, hogy 0 vagy 1 értékű pixelen állunk-e. A 8.2. ábra mutatja az algoritmus lépéseit. Ha 1-értékű pixelen, azaz kontúrpixelen állunk, akkor balra, ha 0-értékű, azaz háttérpixelen állunk, akkor jobbra kell haladni.



8.2. ábra Egyszerű határkövető algoritmus

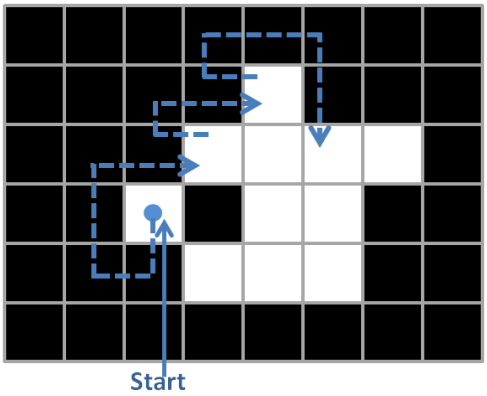
Az algoritmus akkor áll le, ha másodszorra lépünk a start pixelre ugyanabból az irányból [19].

8.4 Moore-szomszédsági algoritmus

A Moore-algoritmus esetén, amelyet a Matlab is alkalmaz, egy adott pixel szomszédságán azokat a pixeleket értjük, amelyekkel közös éle vagy közös csúcsa van, tehát a 8-szomszédú távolságot vesszük figyelembe.

Az algoritmus lépéseit mutatja a 8.3. ábra. Feltételezzük, hogy valamilyen módszerrel már megtaláltuk a kontúr egy tetszőleges pixelét. Ezután a következő pixel

megtalálásához az óramutató járásával megegyező irányban kell haladni, addig amíg nem találunk egy újabb pixelt, ami a kontúrhoz tartozik. A vizsgálódást mindig attól a pixeltől kezdjük, amelyik felől elértük a kontúr aktuális pontját [20].



8.3. ábra A Moore-algoritmus

8.5 Kontúrkereső algoritmusok összehasonlítása

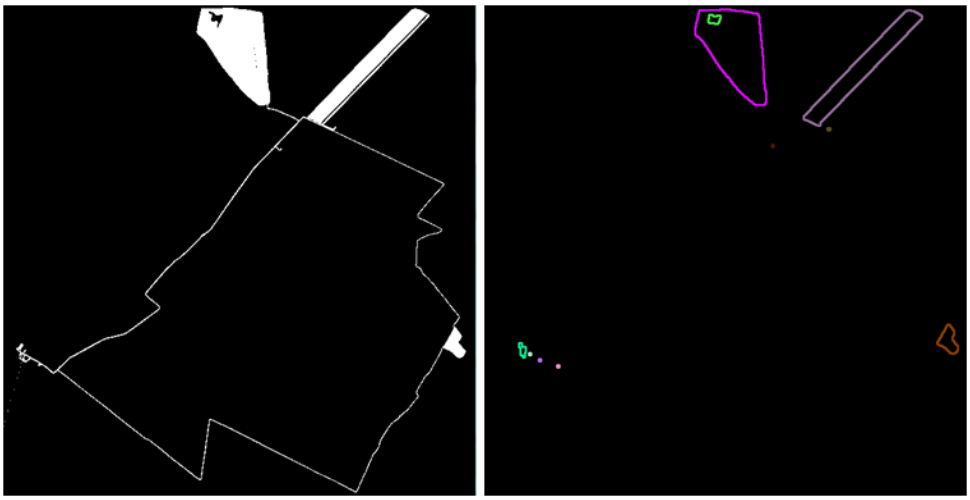
Az egyszerű határkövető algoritmus gyakran nem ismer fel olyan mintákat, amelyek a Moore-szomszédsághoz tartoznak. Ennek az oka az, hogy az algoritmus mindig vagy jobbra vagy balra fordul, ezért az átlós pixelek gyakran kimaradnak. Ezzel szemben a Moore-szomszédsági algoritmus, ahogy a neve is mutatja az adott pixelnek mind a 8 szomszédja között keres szomszédos pixeleket, így az olyan kontúrt is megtalálja, amely átlókat tartalmaz. Azonban mindkét algoritmus hátránya, hogy csak a külső kontúrokat képesek megtalálni, a lyukakat viszont nem.

A feladat megvalósítása során azonban a lyukak megtalálására is szükség van a kihagyott területek felderítéséhez. A Suzuki által létrehozott algoritmus [21], amelyet több képfeldolgozást megvalósító könyvtár többek között az OpenCV[7] és a Java alapú OpenIMAJ[8] is alkalmaz, a határkövető algoritmusokon alapul és alkalmas külső illetve belső kontúrok keresésére is. Az algoritmus egyedi címkéket rendel a kontúrokhoz, és szülő-gyermek kapcsolatot hoz létre köztük az alapján, hogy egymáson kívül vagy belül

[7] http://docs.opencv.org/3.1.0/d3/dc0/group__imgproc__shape.html#ga17ed9f5d79ae97bd4c7cf18403e1689a&gsc.tab=0

[8] http://www.openimaj.org/openimaj-image/image-feature-extraction/apidocs/org/openimaj/image/contour/SuzukiNeighborStrategy.html

helyezkednek-e el. A Suzuki-algoritmus segítségével megtalált táblák láthatóak a 8.4 ábrán.



8.4. ábra A megtalált táblahatárok

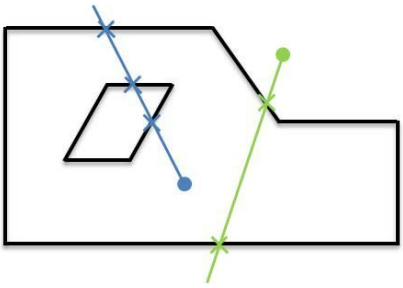
Az esetleges nagy távolságok miatt, amit egy jármű megtehet, alacsony felbontást kell beállítani (5m/px - 10m/px). Emiatt nem lehet a képből egy az egyben visszaállítani az eredeti x-y koordinátákat, mert előfordulhat, hogy egymás mellett lévő sorok koordinátái egy pixelre esnek. Ezért meg kell keresni azokat a sorokat, amelyek az adott kontúron belül találhatóak. Ez úgy történik, hogy a 5. fejezetben ismertetett módszerrel egyenes szakaszokra bontott adatsorban megkeressük, hogy melyik szakasz melyik táblához tartozik. Ehhez elég megnézni, hogy az adott szakasz középpontja beleesik-e az adott táblakontúrba. Ennek megvalósításához szükség van a különböző pont és poligon helyzetét vizsgáló algoritmusok megismerésére.

8.6 Pont és sokszög helyzete

Számítógépes geometriában gyakran előfordul, hogy egy pontról el kell dönteni, hogy egy sokszögön kívül vagy belül helyezkedik el. Fontos, hogy az algoritmus működjön konvex, konkáv és olyan sokszögek esetében is, amelyeknek a belseje lyukas. Először mindig érdemes azt megvizsgálni, hogy az adott pont a sokszöget körülvevő téglalapon (bounding box) belül található-e, ez jelentősen meggyorsítja az algoritmust. Többféle módszer létezik a pont és a poligon helyzetének vizsgálatára, leggyakrabban a Ray casting algoritmust és a szög összegzést alkalmazzák.

8.6.1 Ray casting algoritmus

A ray casting algoritmus, vagy más néven páros-páratlan szabály lényege, hogy a vizsgált pontból indulva felveszünk tetszőleges irányban egy félegyenest, ez általában vízszintes vagy függőleges, majd megszámoljuk, hogy hány pontban metszi a poligon oldalait. Ha ez a szám páratlan, akkor a pont a poligonon belül van, ha páratlan, akkor viszont kívülre esik [14]. A 8.5. ábra erre mutat egy példát. A kék pontból induló félegyenes három helyen metszi a poligont, ezért azon belül helyezkedik el, a zöld pont két helyen metszi, így a poligonon kívül található.

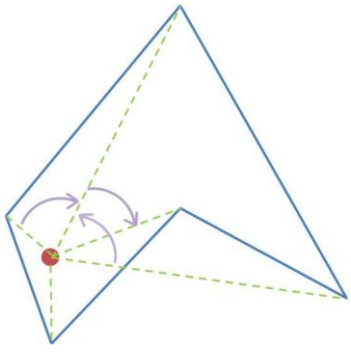


8.5. ábra A ray casting módszer

Ha a félegyenes egy csúcson halad keresztül, akkor a módszer nem működik, így ezt az esetet külön kell kezelni. Véges pontosságú aritmetika esetén, ha a pont nagyon közel található a poligon körvonalához, akkor helytelen eredményt kaphatunk.

8.6.2 Szög összegzés

A módszer lényege, hogy sorban végighaladunk a poligon oldalain és kiszámoljuk a vizsgált pontból az oldalak csúcsaiba húzott egyenesek által bezárt szögek előjelese összegét. Ha az összeg közel van 0-hoz a pont a poligonon kívül van, ha nem, akkor pedig belül.



8.6. ábra Szög összegző módszer bemutatása

8.6.3 A két módszer összehasonlítása

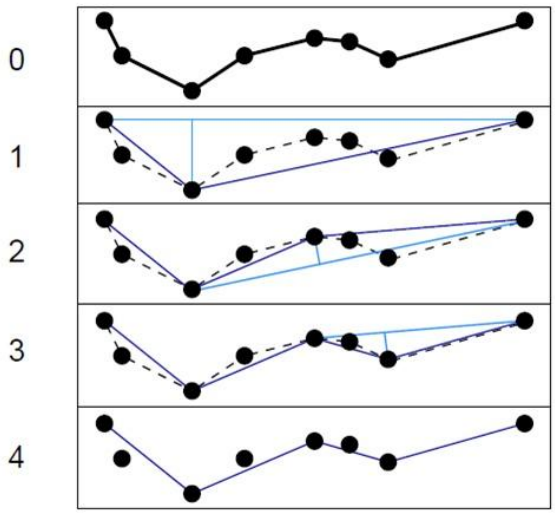
A szög összegzést használó módszer jelentősen lassabb mint a ray casting módszer, mivel költséges trigonometriai műveletek sorozatát kell elvégezni a végrehajtásához. A ray casting algoritmus mellett szól az is, hogy a feladat megvalósításához választott CGAL könyvtár is ezt az algoritmust alkalmazza.

8.7 Pontszámcsökkentési módszerek

A számítógépes geometria egyik fontos problémája, hogy milyen módszerrel lehet csökkenteni egy görbe pontjainak a számát, úgy, hogy a kapott görbe „hasonlítson” az eredetire. Többféle algoritmus létezik a problémára, a leggyakrabban alkalmazott megoldások a Ramer–Douglas–Peucker és a Visvalingam-Whyatt algoritmus.

8.7.1 Ramer–Douglas–Peucker algoritmus

A Ramer–Douglas–Peucker algoritmust gyakran alkalmazzák a kartográfiában például szintvonalak számának csökkentésére. A 8.7 ábra az algoritmus lépéseit illusztrálja.



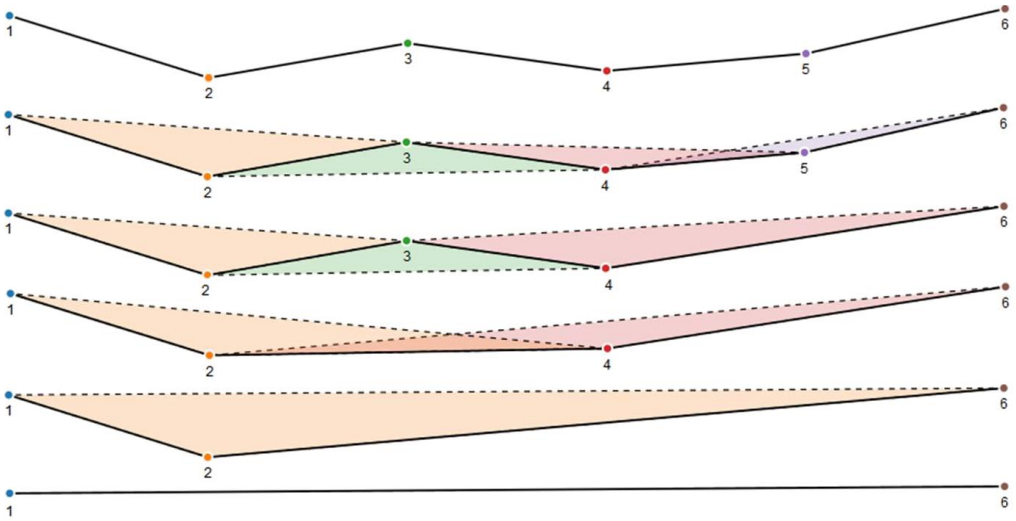
8.7 A Ramer-Douglas-Peucker algoritmus lépései

Az algoritmus bemenete egy rendezett ponthalmaz és egy ε tolerancia érték. Az algoritmus kimenete a bemeneti pontoknak egy halmaza. Az algoritmus rekurzívan felosztja a görbét. Az 1. lépésben a kimeneti pontokhoz adjuk a kezdő és végpontot, mivel ezeket mindenképpen meg kell tartanunk. Ezután megkeressük azt a pontot, amely a legtávolabb esik az előző két pontot összekötő szakasztól. Ha ez a távolság

kisebb mint a tolerancia, akkor véget ért az algoritmus, az összes pontot elhagyjuk ami a szakasz kezdő és végpontja közé esik. Ellenkező esetben megtartjuk az adott pontot, aminek segítségével két részre osztjuk a görbét és ismételjük az előző lépéseket [11].

8.7.2 Visvalingam-Whyatt algoritmus

A Visvalingam-Whyatt algoritmus [12] esetében is meg kell adni egy tolerancia értéket, de itt ez egy területet jelent, nem pedig távolságot a Ramer–Douglas–Peucker algoritmussal ellentétben. A 8.8. ábra mutatja az algoritmus lépéseit.



8.8. ábra A Visvalingam-Whyatt algoritmus lépései

Az algoritmus megkeresi azt a pontot, amelyik a legkisebb területű háromszöget zárja be a két szomszédjával. Ha ez kisebb mint a tolerancia, akkor eltávolítjuk a pontot. Az ábra második sorában a lila terület a legkisebb, ezért a hozzá tartozó 5. pontot eltávolítjuk. Ezután addig ismételjük ezt a lépést, amíg csak a toleranciánál nagyobb háromszögek maradnak.

8.7.3 A pontszámcsökkentési módszerek összehasonlítása

Mindkét algoritmus a bemeneti pontok egy részhalmazát adja vissza. Különböző generalizáló algoritmusokon végzett tesztek alapján [13] amíg a görbe pontszáma el nem éri körülbelül az ezer pontot, addig a Ramer–Douglas–Peucker algoritmus gyorsabb. Minél kevesebb pontot tartalmaz a görbe, annál nagyobb az eltérés a két módszer sebessége között, tehát kis pontszám esetén érdemes a Ramer–Douglas–Peucker algoritmust használni, ha ezer pontnál sokkal többet tartalmaz a görbe, akkor

viszont a Visvalingam-Whyatt algoritmus javasolt. A tapasztalatok alapján az egyes táblák kontúrjának pontszáma a választott felbontás mellett ennél jóval kevesebb.

A tesztek [13] alapján a közelített görbe átlagos távolsága az eredeti görbétől Ramer–Douglas–Peucker algoritmus esetén körülbelül 750 pontig jelentősen kisebb, e fölött körülbelül hasonló eltérést mutatnak, tehát a feladat megvalósításához e szempont alapján is célszerű a Ramer–Douglas–Peucker módszert választani. Mivel a sebesség és a torzítás mértéke alapján is a Ramer–Douglas–Peucker szolgáltat jobb eredményt, ezért ezt az algoritmust alkalmaztam.

8.7.4 A választott algoritmus tesztelése

A választott Ramer–Douglas–Peucker algoritmust különböző tolerancia értékekkel is teszteltem. A 8.1. táblázat tartalmazza egy tábla kontúr esetén az eredeti poligon és a generalizált poligon pontjainak a számát különböző felbontás és tolerancia értékek alkalmazása esetén.

Felbontás [m/px]	Tolerancia	Eredeti pontszám	Csökkentett pontszám
5	2	273	18
2	2	623	139
5	4	273	15
2	4	623	38
5	6	273	10
2	6	643	17

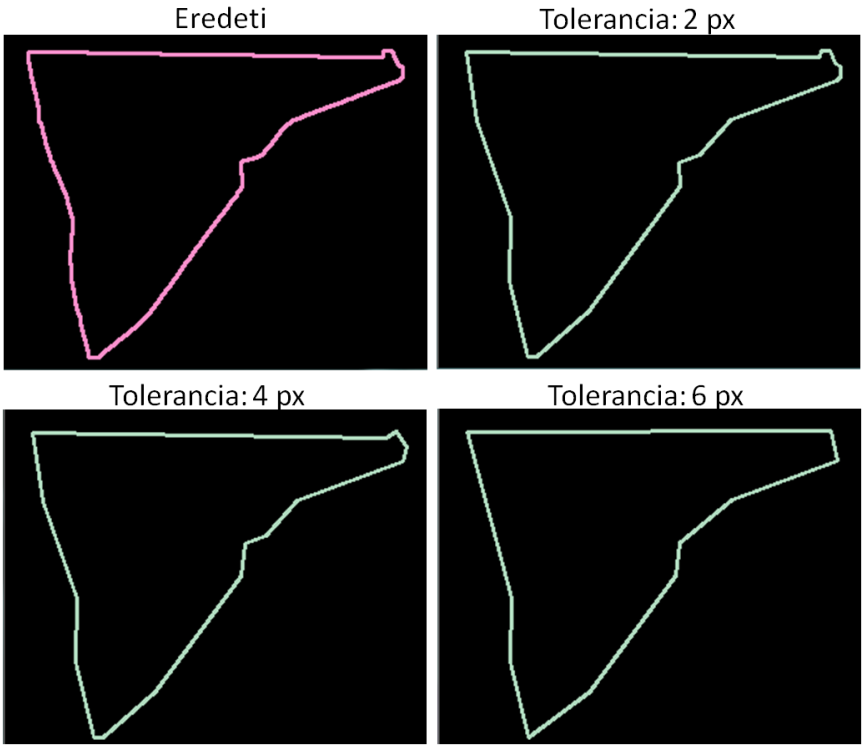
8.1. táblázat Pontszámok csökkenése

Futási idő teszteket is végeztem az alapján, hogy a kontúrok pontszáma mennyire befolyásolja a futási időt a pont és poligon helyzetét meghatározó algoritmus végrehajtásakor. A 8.2. táblázat az egyes poligon pontszámok esetén mutatja az algoritmus futási idejét. Látható, hogy a pontszám csökkentésével jelentős gyorsulás érhető el, ezért mindenképpen érdemes alkalmazni.

Pontok száma	Futási idő [ms]
285	4493
93	1404
16	156
9	78

8.2. táblázat Futási idő különböző pontszámok esetén

A 8.9. ábra ugyanennek a táblának a kontúrját mutatja 5 m/px felbontás mellett. A rózsaszín tábla az eredetileg meghatározott kontúrt mutatja pontszámcsökkentő algoritmus alkalmazása nélkül. A mellette szereplő kontúr 2px tolerancia alkalmazásával keletkezett, az alsó két kontúr pedig sorra 4 és 6px tolerancia használatával jöttek létre.



8.9. ábra A generalizálás eredménye különböző tolerancia értékek esetén

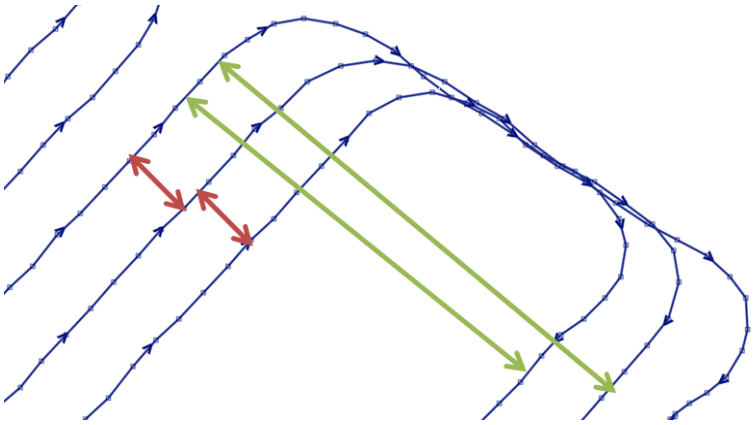
Összevetve a 8.1. táblázat és a 8.9. ábra eredményeit látható, hogy a Ramer-Douglas-Peucker algoritmussal jelentős pontszámcsökkenést lehet elérni, miközben az így létrehozott sokszög alig tér el az eredetitől. A csökkentett pontszámú sokszög meghatározásának célja, hogy el lehessen különíteni egymástól és az utaktól a mezőgazdasági táblákat. Ez úgy történik, hogy az egyenes szakaszokra bontott adatsorban megkeressük, hogy melyik szakasz melyik táblához tartozik. Ehhez elég megnézni, hogy az adott szakasz középpontja beleesik-e az adott táblakontúrba. Ennek megvalósításához nem szükséges, hogy a kontúr pontosan kövesse az eredeti tábla határait, így nagyobb tolerancia érték is megengedhető.

9 Sortávolságok meghatározása

A mezőgazdasági táblák egy fontos jellemzője és így a munkavégzés egyik fontos mérőszáma a sortávolság. Ennek meghatározása elsőre egyszerűnek tűnhet, csak meg kell határozni az egymás után következő sorokra eső egyenes távolságát. Azonban több probléma is felmerül. Egyrészt a párhuzamosnak nevezett sorok nem teljesen párhuzamosak, valamint az egymás melletti sorok megművelése időben nem egymás után történik. Ezeknek a problémáknak a megoldását ismerteti a következő fejezet.

9.1 Sorok rendezése

A szántóföldi munkák végzése gyakran a 9.1 ábrán látható minta szerint történik. Ha egy mezőgazdasági gép hosszú, nehéz eszközöket vontat maga után, akkor egyszerűbb, ha nem fordul vissza rögtön a sor végén, hanem tesz egy körülbelül 90 fokos fordulót a sorok végén, majd egy hosszú, akár 50-60 méteres egyenes szakasz után fordul meg. Majd miután végighaladt a soron egy újabb 90 fokos forduló, és egy egyenes szakasz után visszatér az első sor mellé.



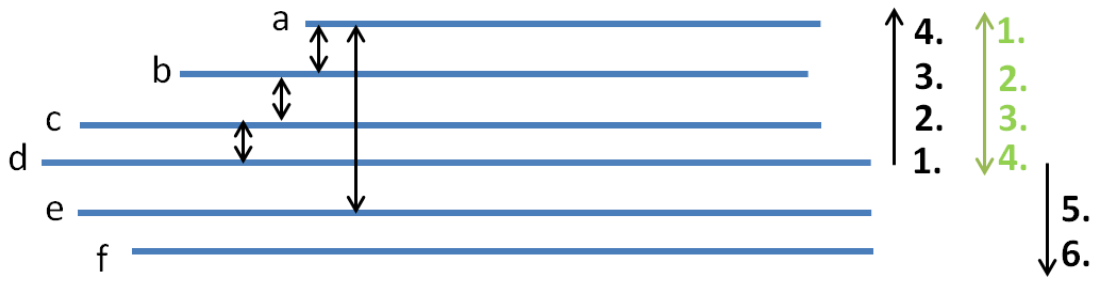
9.1. ábra Gyakori művelési minta

Rendelkezésre állnak a korábban meghatározott párhuzamos egyeneseket tartalmazó csoportok. Ahhoz, hogy a sortávolságokat meg lehessen határozni, először is megfelelő sorrendbe kell rendezni ezeknek a sorait. Erre kétféle lehetőséget is ismertetek.

9.1.1 Egyszerű sorba rendezés

A módszer egy egyszerű sorba rendezést valósít meg. A bemenet egy vector, ami egymással párhuzamos szakaszokat tartalmaz tetszőleges sorrendben, a kimenet egy rendezett vector. Az algoritmus egy tetszőleges sorral indul és megkeresi a hozzá legközelebb eső szakaszt. Ha megtaláltuk a legközelebbi szakaszt, akkor azt ki kell venni a csoportból, majd a maradék szakaszok között kell keresni a hozzá legközelebbit. Mivel a kezdő szakasz nem feltétlenül a tábla szélén van ezért nyilván kell tartani, hogy a tábla melyik széle felé haladunk. Ha oldalváltás történik, akkor meg kell fordítani az eddig tárolt szakaszok sorrendjét és folytatni tovább az algoritmust, amíg el nem fogynak a sorok. Ezt addig kell folyatatni, amíg el nem fogynak a szakaszok.

A 9.2. ábra demonstrálja az algoritmus menetét. Tegyük fel, hogy a d szakasszal kezdünk, azt kivesszük a bementi adatok közül, majd meg kell határozni az összes többi szakasztól vett távolságát, és kiválasztani a legközelebbit, ez lesz a c szakasz. A c szakaszt is kivesszük a szakaszok közül. Amikor az a szakaszhoz érünk, akkor a hozzá legközelebb eső az e szakasz lesz. Ez a legelső szakasznak az ellenkező oldalán van ezért meg kell fordítani az eddig megtalált szakaszok sorrendjét.



9.2. ábra Egyszerű sorba rendezés

Az algoritmus kis méretű táblák esetén jól működik, azonban a sorok számának növekedésével lassú lesz.

9.1.2 Címkézés

Az előző algoritmushoz hasonlóan a bemenet ebben az esetben is egy vector, ami tetszőleges sorrendű egymással párhuzamos szakaszokat tartalmaz, a kimenet pedig egy rendezett vector.

Vesszük az első szakaszt, ami a tábla egy tetszőleges pozíciójában található. Létre kell hozni két csoportot az alapján, hogy az adott szakasz melyik oldalán

helyezkedik el a vizsgált elem. Ezután végighaladunk a szakaszokon és az alapján, hogy a referencia sor melyik oldalán helyezkedik el a megfelelő csoportba kell tenni a távolság értékével együtt.

Miután minden egyes szakasznak meghatároztuk a referencia sortól való távolságát és a helyzete alapján a megfelelő csoportba tettük, mindkét csoport elemeit sorba kell rendezni távolság szerint. Ezután az egyik csoport elemeinek sorrendjét fel kell cserélni, majd hozzáilleszteni a referencia szakaszt, utána pedig a másik csoport elemeit. A módszer a 9.3. ábrán látható.

9.3. ábra Sorba rendezés címkézéssel

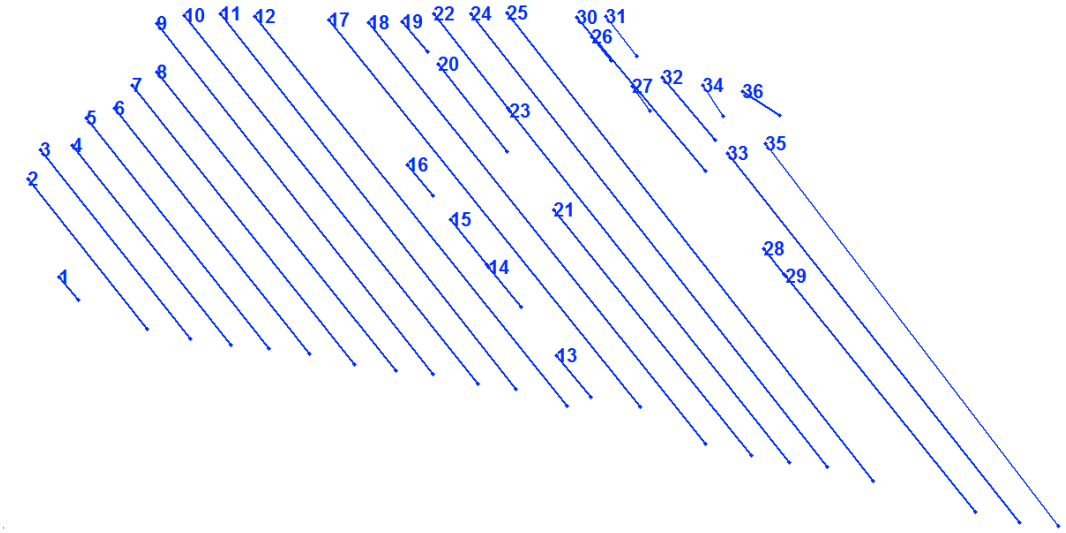
A következő függvény bemenete egy vektor, ez tartalmazza a távolság értékeket. A kimenete egy vektor, ami tartalmazza a növekvő sorrendbe rendezett távolságok indexeit.

```
vector<int> sortOrder(const vector<double>& vec)
{
	vector<int> order(vec.size());
	iota(order.begin(), order.end(), 0);
	sort(order.begin(), order.end(),
		[&](int i, int j){ return vec.at(i) < vec.at(j); });
	return order;
}
```

A következő függvény bemenete egy vektor, amely a szegmenseket tartalmazza, valamint az előzőleg létrehozott vektor, amelyben az indexek szerepelnek. A függvény az indexek alapján meghatározott sorrendbe rendezi a szegmenseket.

```
vector<Segment> sortByOrder(const vector<Segment>& vec,
	const vector<int>& order)
{
	vector<Segment> sorted(order.size());
	transform(order.begin(), order.end(), sorted.begin(),
		[&](int i){ return vec.at(i); });
	return sorted;
}
```

Az alábbi ábrán láthatók egy tábla sorai, a sorok mellett szerepel a sorszámuk is, látszik, hogy az algoritmus sikeresen rendezte a szegmenseket.



9.4. ábra Egy tábla sorba rendezett szegmensei

9.2 Hisztogram

A megművelt mezőgazdasági táblák egy fontos jellemzője a sortávolság. Azonban a sortávolság nyilvánvalóan nem állandó, ezért célszerű valamilyen átlag értéket számítani. A munka folyamán előfordulhat, hogy kimaradnak sorok, valamint a szoftver hibájából adódóan is lehetséges, hogy egy-egy sort nem sikerül megtalálni. Azt is figyelembe kell, venni, hogy mivel az 5. fejezetben ismertetett módon történik a szegmensek keresése, ezért előfordul, hogy a tábla egy tényleges sora több szakaszból áll, így az ezek közti távolságot nem szabad figyelembe venni. Éppen ezért érdemes hisztogramot készíteni a sortávolságokról.

A hisztogram alkalmazásának célja, hogy megtaláljuk azt a sortávolságot, ami körüli értékeket az átlag vagy medián számítása során figyelembe kell venni, a túl közeli, tehát valószínűleg egy egyenesre eső és a kihagyott sorok elhagyásával.

9.2.1 Hisztogram elméleti háttere

A hisztogram a statisztika egyik leggyakrabban alkalmazott eszköze, egy speciális oszlopdiagram. Az adatokat csoportokra osztja és megadja ezek előfordulásának gyakoriságát. Az egyes gyakoriságokat területarányosan ábrázolja. A hisztogram gyakorlatilag becslést ad egy valószínűségi változó sűrűségfüggvényére.

A hisztogram elkészítéséhez először is intervallumokra kell osztani a teljes adattartományt, majd megszámolni, hogy hány érték kerül a meghatározott tartományba, ezt nevezzük gyakoriságnak. A tartományok szomszédosak, nem lapolódnak át és általában megegyezik a szélességük.

Az osztályok számának meghatározására többféle módszer is létezik, az adatok eloszlásától, mennyiségétől és a célunktól is függ, hogy melyiket érdemes alkalmazni. Ha túl sok osztályt választunk, akkor nagyon kevés adat kerül egy csoportba és így töredezett lesz a hisztogram. Viszont, ha túl kevés osztályunk van, akkor durva lesz a felbontás. Az egyik leggyakrabban alkalmazott módszer a Sturges-formula, amit akkor érdemes használni, ha 30-nál több értéket tartalmazó adatsorral rendelkezünk, melyeknek eloszlása megközelítőleg normális eloszlású.

$$k = \lceil \log_2 n \rceil + 1$$

Az osztályok számának meghatározása után az osztályszélesség kiszámítása magától értetődő módon a következő:

$$\frac{x_{max} - x_{min}}{k}$$

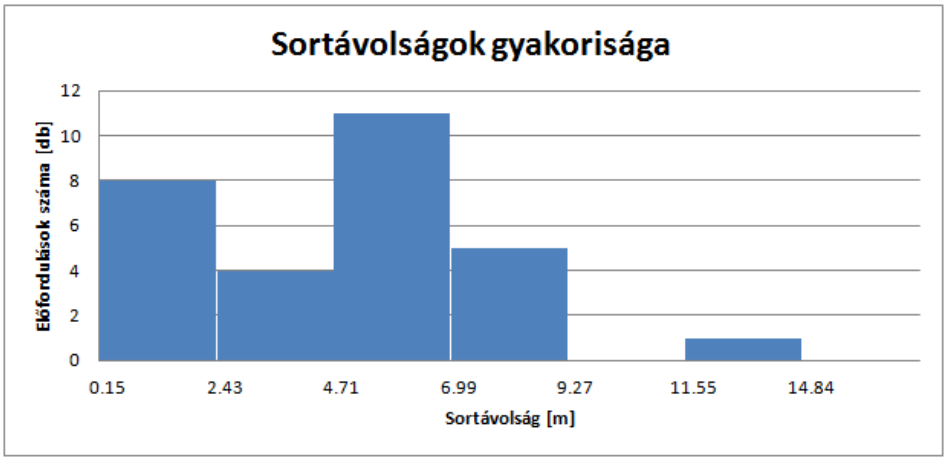
9.2.2 Hisztogram létrehozása

A hisztogramot létrehozó függvény bemenete egy vector, ami a távolság értékeket tartalmazza. A kimenete pedig egy pair, amelyben az egyes tartományok kezdőértéke, valamint a hozzá tartozó gyakoriságok szerepelnek.

Először is meg kell határozni a tartományok számát. Mivel a sorok száma a legtöbb tábla esetén 30 felett van és eloszlásuk közelítőleg normális, ezért a Sturges-formulát alkalmaztam az osztályok számának meghatározására. Ennek az értékét tartalmazza a groupNum változó. Ehhez meg kell keresni a minimum és a maximum értéket az adatok között. Ezekből már kiszámítható az egyes tartományok szélessége is, melynek értékét a groupWidth tárolja. A bounds vector tartalmazza az egyes tartományok kezdőértékét, ezt a minimum értékkel kezdve egy for ciklus feltölti a tartományszélességek hozzáadásával. A frequency vector a gyakoriságok tárolását végzi, a for_each algoritmus tölti fel a megfelelő értékekkel. Először is az upper_bound függvény segítségével megkeresi, hogy melyik tartományba kell kerülnie az adott értéknek, majd a hozzá tartozó gyakoriságok számát megnöveli eggyel.

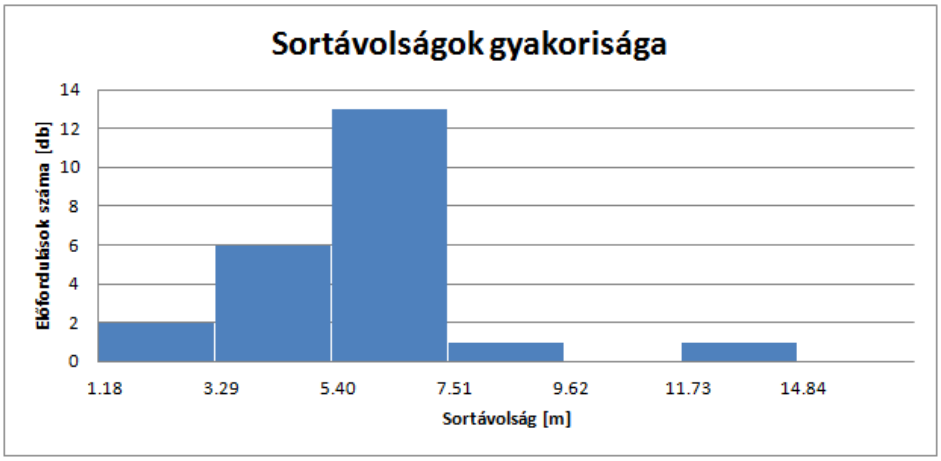
```
pair<vector<double>, vector<double>> histogram(vector<double>& values)
{
	vector<double> bounds;

	double minValue = *min_element(values.begin(), values.end());
	double maxValue = *max_element(values.begin(), values.end());

	int groupNum = ceil(log2(values.size()) + 1);
	double groupWidth = static_cast<double>(maxValue-minValue)/groupNum;

	for (int i = 0; i < groupNum; i++) {
		bounds.push_back(minValue + (i * groupWidth));
	}
	bounds.push_back(maxValue + 1);

	vector<int> frequency(bounds.size(), 0);

	for_each(values.begin(), values.end(), [&](double &val) {
		vector<double>::iterator it = upper_bound(bounds.begin(),
			bounds.end(), val);
		vector<int>::iterator f = frequency.begin() +
			distance(bounds.begin(), it) - 1;
		(*f)++;
	});

	return make_pair(bounds, frequency);
}
```

A 9.5. ábrán látható egy tábla adatai alapján készült hisztogram. A leggyakrabban előforduló sortávolságok értéke 4,71 és 6,99 méter közé esik, ez 11 esetben fordul elő a 29 szakaszból. Nyolc olyan sor is van, amelynek 0,15 és 2,43 méter közti a szélessége, ez valószínűleg a korábban említett probléma miatt lép fel, tehát néhány sor valamiért több részre lett felosztva.



9.5. ábra Hisztogram

A közeli sorokat érdemes a hisztogram elkészítése előtt kiszűrni, egy minimum sortávolság meghatározásával (körülbelül fél méter). Az így létrehozott hisztogramot ábrázolja a 9.6. ábra.



9.6. ábra Hisztogram a fél méternél kisebb sortávolságok kiszűrésével

Látható, hogy a leggyakrabban előforduló távolság minimum és maximum értéke között több mint két méter távolság van, ez viszonylag nagynak számít. Azért ekkora a szórás, mert a pozíciómérés egy kommersz vevővel történt, így a mérés bizonytalansága méteres, a sortávolság pedig ehhez képest viszonylag kicsi. Precíziós vevővel, 30 méteres munkaszélességű permetezőgép esetén nyilván egy vagy két osztályba esne az összes érték.

A leggyakrabban előforduló sortávolságoknak az átlagát meghatározva adódik egy olyan sortávolság, amely már jól jellemzi a munkavégzést. Látható, hogy körülbelül hat méteres az átlagos sortávolság, és nagyjából ennek a duplájánál is szerepel egy érték, ez jelzi, hogy kimaradt egy sor. A hat méteres sortávolság nem meglepő, mivel általában ilyen távolságot alkalmaznak vetéskor, így a vetés utáni talajmunkáknál is ezt kell követni, esetleg ennek egész számú többszörösét. A vetés előtti talajmunkáknál másfajta sortávolság is előfordulhat.

10 Területszámítás

A feladat része az egymástól és a táblák közti vonulástól elkülönített táblák területének meghatározása, valamint a kihagyott területek detektálása.

10.1 Raszterizálás

A megművelt táblák méretének meghatározásához fel lehet használni a 8. fejezetben ismertetett módszereket. Tehát képfeldolgozás segítségével meg lehet keresni a táblák körvonalát, majd ez alapján meg lehet határozni a területét. A pontosságot azonban jelentősen befolyásolja a képfelbontás.

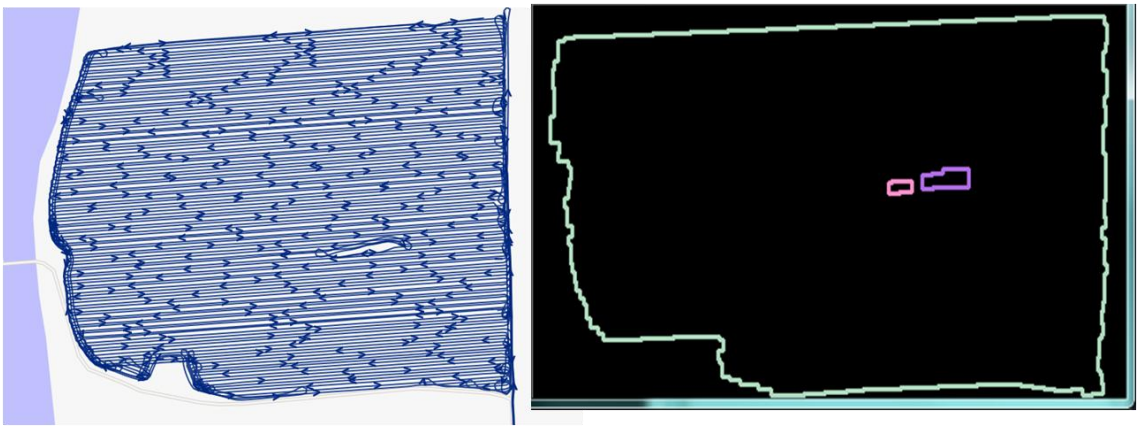
Mivel már rendelkezésre állnak az egymástól elkülönített táblák, így viszonylag kis méretű területet kell egyszerre raszterizálni és feldolgozni, így nagyobb felbontás is beállítható. Emellett már ismertek a sortávolságok is, ezt figyelembe lehet venni a strukturáló elem méretének meghatározásánál. A 7.8 alfejezet ismerteti, hogy a strukturáló elem mérete függ a felbontástól és a maximálisan megengedhető sortávolságtól. Jelölje a strukturáló elem méretét n, a maximális megengedhető sortávolságot egy táblán belül d, a felbontást pedig r. Ebben az esetben a strukturáló elem mérete $n \times n$.

$$n = \frac{d}{r}$$

Mivel már ismert a sortávolság így nem kell feleslegesen nagy strukturáló elemet alkalmazni.

A megvalósított program egy szerver oldali alkalmazás része lesz, amely bizonyos időközönként kapott pozícióadatok alapján eldönti, hogy az adott jármű dolgozik-e. Azonban területszámításra csak ritkán van szükség, emiatt nem szükséges, hogy a számítás gyors legyen, ez is hozzájárul a nagyobb felbontás és így a nagyobb képméret alkalmazásához.

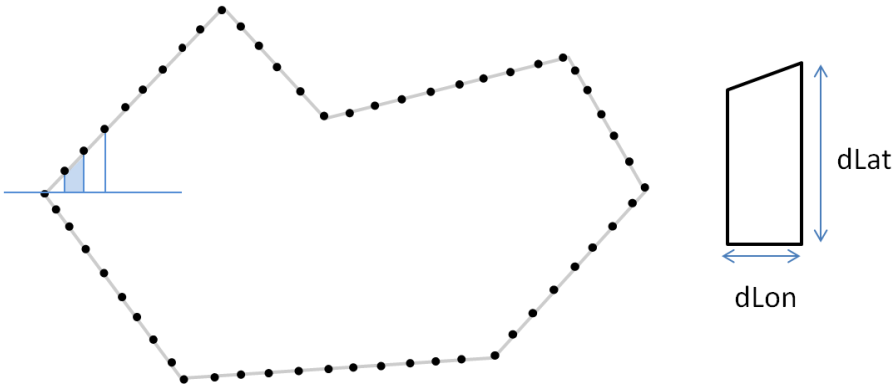
Mindezek miatt a felbontás sokkal nagyobbra állítható, mint, amit a területek szétválasztásakor alkalmazni lehetett, $10\,m/px$ helyett $2\,m/px$ felbontást alkalmaztam.



10.2 Sokszög területe

A tábla kontúrjának meghatározása után meg kell határozni a kapott sokszög területét. A területszámítást el lehet végezni a földrajzi koordináták segítségével, vagy valamilyen területtartó vetületet kell választani és a koordinátákat ebbe a rendszerbe átkonvertálva kiszámolható a terület.

A poligon területét a trapéz szabály segítségével egyszerűen ki lehet számítani. A poligon pontjain sorba haladva a 10.2 ábrán látható módon trapézokra kell bontani a poligont, és ezeknek a trapézoknak a területét kell összegezni.



10.1. ábra Területmérés trapéz szabály segítségével

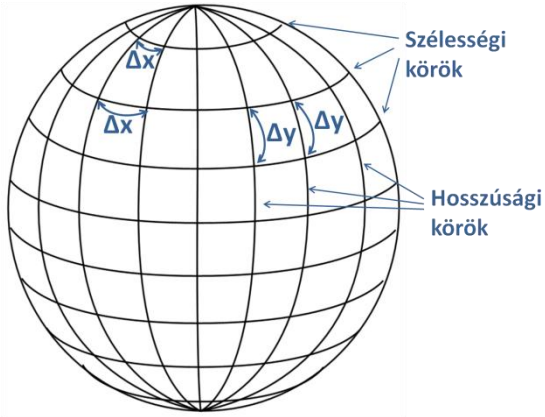
A trapéz területe következő képlettel számítható ki.

$$T = \frac{a + c}{2} m$$

A terület meghatározására a földrajzi koordinátákat alkalmaztam, így a szélességi és hosszúsági fokokban adott értékeket hosszúság értékekké kell alakítani.

Mivel a trapézok területének meghatározásakor nagyon kicsi távolságokról van szó, nem szükséges figyelembe venni a föld görbületét.

A rendelkezésre álló adatok WGS-84 dátum szerint vannak megadva. Ebben a rendszerben a föld egy lapított ellipszoidnak tekinthető, melynek nagy féltengelye $a = 6378137\text{m}$, kis féltengelye pedig $b = 6356752\text{m}$. Ha a pozíciónk y irányban változik, azaz szélességi fokbeli (lat) eltérés esetén a megtett távolság nem függ attól, hogy melyik hosszúsági körön (lon) helyezkedünk el. Mivel a szélességi körök nagysága annál jobban csökken, minél messzebb vagyunk az egyenlítőtől, így x iránybeli eltérés esetén, azaz, ha a hosszúsági koordinátánk változik, akkor a méterben megtett távolság nagyban függ attól, hogy melyik szélességi körön vagyunk. Ezt mutatja a 10.2. ábra.



10.2. ábra Távolságmérés a szélességi és hosszúsági körök mentén

A következő képlet segítségével kiszámolható, hogy egy adott (lat) szélességi körön Δlon változás esetén mekkora a Δx távolság méterben. Látható, hogy ez függ attól, hogy melyik szélességi körön vagyunk.

$$\Delta x = a \cdot \cos(lat) \cdot \Delta lon \cdot \frac{\pi}{180}$$

A következő képlet segítségével kiszámolható, hogy Δlat változás esetén mekkora a megtett Δy távolság méterben. Látható, hogy ez nem függ attól, hogy melyik hosszúsági körön vagyunk.

$$\Delta y = b \cdot \Delta\text{lat} \cdot \frac{\pi}{180}$$

Ezek alapján a területmérést megvalósító programrészlet a következő.

```
double GetElementAreaInSqM(vector<cv::Point_<double>>& coords)
     {
          static const double  MAS2RAD = M_PI / 648000000.0;
          static const double  RLONG_EARTH = 6378137.0;
          static const double  RLAT_EARTH = 6356752.3142451795;
          static const double  MAS2LONG_EARTH = MAS2RAD * RLONG_EARTH;
          static const double  MAS2LAT_EARTH = MAS2RAD * RLAT_EARTH;

          if (coords.size() <= 0)
               return 0.0;

          double area = 0.0;
          int  i;
          int pointCount = coords.size();

          for (i = 0; i < coords.size(); ++i) {
               int32_t idx2 = (i + 1) % pointCount;
               int32_t dx = coords[idx2].x - coords[i].x;

               if (dx == 0)
                    continue;
               int32_t y = (coords[i].y + coords[idx2].y) / 2;

               double cy = cos(y * MAS2RAD);
               double dxM = dx * cy * MAS2LONG_EARTH;
               double dyM = (y - coords[0].y) * MAS2LAT_EARTH;

               area = area + dxM * dyM;
          }

          return ::fabs(area);
     }
```

Vannak olyan mezőgazdasági gépek, például permetező- és öntözőgépek, amelyeknek a munkaszélessége több méter is lehet. A 10.3. ábra egy ilyen gépet ábrázol.

10.3. ábra Nagy munkaszélességű mezőgazdasági gép

A GPS vevő a gép közepén található, így a tábla szélein fél-fél munkaszélességnyi terület kimarad a kontúrkereséskor. Ezért a meghatározott területhez hozzá kell adni ezeket a kimaradt részeket, hogy pontosabb eredményt kapjunk. Ez a következő képlet alkalmazásával történik.

$$area += (length(row_0) + length(row_N)) \cdot \frac{workWidth}{2}$$

11 Alkalmazott technológiák

A diplomaterv készítése során többféle technológia alkalmazására is szükség volt, melyek kiválasztásánál figyelembe kellett venni, hogy az adott könyvtár C++ nyelvű, valamint nyílt forráskódú legyen, emellett nyilvánvaló szempont volt a részletes dokumentáltság is. A fejezet ismerteti a kiválasztott technológiákat, először az OpenCV gépi látás könyvtárat, majd a CGAL számítógépes geometriát alkalmazó könyvtárat, végül pedig a GeographicLib következik, amely különböző vetületi rendszerek közti konverziók elvégzésére alkalmas.

11.1 OpenCV

A feladat megvalósításának egyik alappillére különböző képfeldolgozási módszerek használata. Erre kiválóan alkalmas az OpenCV nyílt forráskódú könyvtár, mely szabadon használható a BSD licensz alatt, és C++ interfésszel is rendelkezik. A könyvtár célja, hogy egy könnyedén használható eszközt nyújtson a képfeldolgozással foglalkozók számára.

Az OpenCV [7] alapvető adattárolásra használt eleme a mátrix osztály. Legnagyobb előnye, hogy a felhasználónak nem kell memória kezelési problémákkal foglalkoznia, nincs szükség manuális memóriafoglalásra és felszabadításra, mivel ezt automatikusan megoldja az osztály.

Mivel a képfeldolgozó műveleteknek nagy a számításigénye, ezért annak érdekében, hogy ne csökkenjen feleslegesen a sebesség, a könyvtár referencia számlálást alkalmaz a felesleges másolások elkerülése érdekében. Ez annyit jelent, hogy egy mátrixot megoszthatunk több változó között mátrix pointerek segítségével. A másoló operátorok pedig csak a mátrix header részét másolják le, ami a méretét, címét és egyéb információkat tartalmazza, magát az adatsort viszont nem kell másolni [8].

A könyvtár széles körű funkcionalitással rendelkezik, a következő felsorolás csak a feladat elvégzéséhez alkalmazott műveletek tartalmazza. Az OpenCV könyvtár támogatja az alapvető képfeldolgozási műveletek elvégzését:

- kép szűrés
    - erodálás
    - dilatáció

    - nyitás
    - zárás
- strukturális elemzés
    - kontúrkeresés
    - alakzat területe
- tulajdonság keresés
    - Hough-transzformáció
    - Line Segment Detector

A könyvtár további előnye, hogy részletes dokumentációval rendelkezik és rengeteg alkalmazási példát biztosítanak a felhasználók számára.

11.2 CGAL

A CGAL (Computational Geometry Algorithms Library) egy C++ könyvtár, amely hatékony és megbízható geometriai algoritmusokat tartalmaz. Számos területen alkalmazzák, többek között földrajzi információs rendszerekben, számítógépes grafikában, robotikában és molekuláris biológiában [9].

A könyvtár több különböző komponensből áll, alapvető része a kernel, ami tartalmazza a legfontosabb két és háromdimenziós objektumokat: pont, szegmens, egyenes, vektor, kör, felület, valamint az ezeken végezhető műveleteket, többek között orientáció, metszéspontok, hossz. Ezek mellett a CGAL könyvtár olyan geometriai struktúrákat és algoritmusokat is tartalmaz, mint például a konvex Hull, poligon, háromszögelés, interpoláció [9]. A könyvtár létrehozásánál a legfontosabb szempont a robusztusság, az általánosság, a hatékonyság és az egyszerű használhatóság volt.

11.3 GeographicLib

A GeographicLib [10] egy C++ könyvtár, amely segítségével egyszerűen lehet konverziókat végezni különböző koordináta-rendszerek között. A konverziók pontossága körülbelül 5–15 nanométer. A támogatott rendszerek a következők:

- földrajzi koordináta-rendszer
- UTM
- UPS
- MGRS
- geocentrikus koordináta-rendszer
- lokális Descartes-féle koordináta-rendszer

12 Elért eredmények

A képfeldolgozás felhasználásával történő megvalósítást többféle adatsoron teszteltem futási idő és hatékonyság szempontjából egyaránt.

12.1 Futási idő analízis

A 12.1. táblázat az egyes műveletek átlagos futási idejét mutatja három különböző méretű adatsor esetén, a futási idők ms egységben értendők.

Adatok száma / Művelet	83119	101716	81033
Simítás	109	125	107
Felbontás sebesség alapján	359	421	328
Raszterizálás 5m/px felbontás mellett	111	95	93
Morfológiai műveletek: zárás, nyitás	327	437	329
Kontúrkeresés	15	15	16
Egyenes szakaszok keresése, regresszió	749	983	718
Kontúrhoz tartozó szakaszok keresése	298	655	389
Párhuzamos szakaszok keresése	14	1	3
Sorok rendezése	1	31	16
Átlagos sortávolság meghatározása	1	1	15
Területszámítás	343	531	282
Összesen	2327	3295	2296

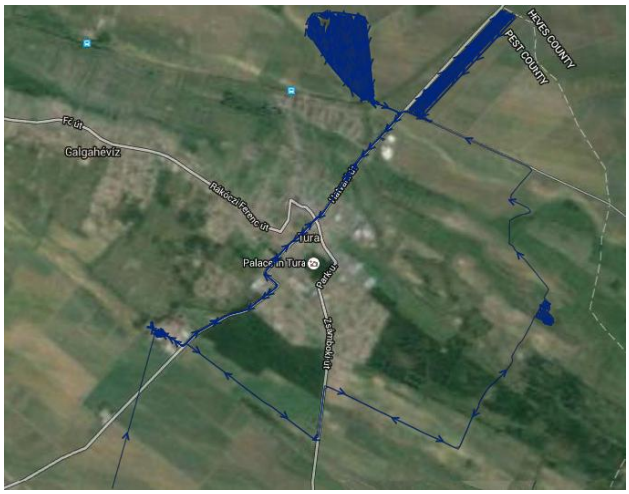
12.1. táblázat Futási idők képfeldolgozás alkalmazása esetén

A mérési eredményekből is látszik, hogy a simítás és a sebesség alapján történő felbontás elvégzésének ideje arányos az adatok számával. A raszterizálás és a morfológiai műveletek elvégzése annál tovább tart, minél nagyobb a vizsgált terület mérete, tehát a futási idő nem csak az adatsorok nagyságától függ. Ha a sebesség alapú felbontás alapján kevesebb adat marad a raszterizálásra, akkor gyorsabban végre lehet hajtani a műveletet. Ezt befolyásolja az is, hogy ténylegesen milyen volt a vonulás és a munkavégzés aránya, valamint az is, hogy mennyire tér el az adott jármű munkavégzési és vonulási sebessége. A sorok rendezésének és az átlagos sortávolság meghatározásának sebességét nyilvánvalóan a megtalált táblákban lévő sorok száma befolyásolja. A területszámítás sebessége egyrészt függ attól, hogy mekkora méretű táblákon történt munkavégzés, valamint függ az átlagos sortávolságtól is, mivel kisebb

sortáv esetén kisebb méretű strukturáló elemet kell alkalmazni a morfológiai műveletek elvégzésekor.

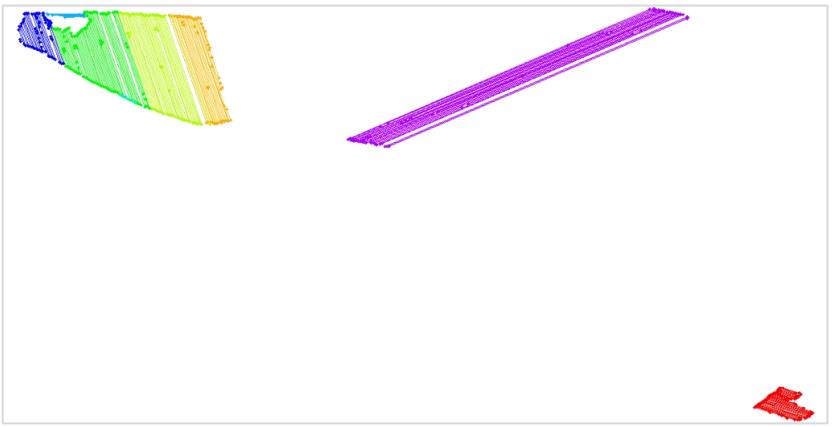
12.2 Hatékonyság

Több adatsoron is ellenőriztem az algoritmus működését, a munkafolyamatok meghatározása, a sortávolságok és a megművelt terület méretének meghatározása is megfelelően működik. A 12.1. ábra egy mezőgazdasági gép egy heti munkavégzését tartalmazza. A 12.2. ábra pedig a megtalált táblákat ábrázolja.



12.1. ábra Egy jármű napi munkavégzése

A bal oldali tábla egy körülbelül 70 hektár nagyságú terület, amelyet több nap alatt sikerült megművelni. Látható, hogy el is vannak különítve az egyes részek, ez a sebesség alapú felbontásnak köszönhető.



12.2. ábra A megtalált táblarészletek

Irodalomjegyzék

[1] Satheesh Gopi: *Global Positioning System: Principles And Applications,* 2005

[2] Ahmed El-Rabbany: *Introduction to GPS: The Global Positioning System*, 2002, Artech House

[3] Richard B. Thompson: *Global Positioning System: The Mathematics of GPS Receivers,* Mathematics Magazine pp. 265.

[4] Terry W. Griffin: GPS CaPPture: *A system for GPS trajectory collection, processing, and destination prediction,* 2012

[5] Melita Kennedy, Steve Kopp: *Understanding Map Projections,* July 1, 2001

[6] Otto Huisman, Rolf A. de By: *Principles of Geographic Information Systems,* 2009 pp.209

[7] http://opencv.org/ (2016.05.29.)

[8] Bernát Gábor: *OpenCV The core functionality:* Mat - The Basic Image Container (2016.05.29.)

[9] http://www.cgal.org/ (2016.05.29.)

[10] http://geographiclib.sourceforge.net/html/index.html (2016.05.29.)

[11] Douglas D. H., Peucker T. K.: *Algorithms for the Reduction of the Number of Points Required to Represent a Digitised Line or its Caricature.* The Canadian Cartographer, 10(2), 1973.

[12] M. Visvalingam and J. Whyatt: *Line generalisation by repeated elimination of points,* The Cartographic J. 1993.

[13] Wenzhong Shi & ChuiKwan Cheung: *Performance Evaluation of Line Simplification Algorithms for Vector Generalization,* 2013, The Cartographic Journal

[14] Paul S. Heckbert: *Graphics gems IV.*, 1994

[15] Pierre Soille: *Morphological Image Analysis: Principles and Applications,* 2007

[16] Robert M. Haralick, Stanley R. Sternberg, Xinhua Zhuang: *Image Analysis Using Mathematical Morphology*, 1987.

[17] J.R.Parker: *Algorithms for Image Processing and Computer Vision*, 2010

[18] Chen Chen, Yinhang Cheng: *Roads Digital Map Generation with Multi-track GPS Data,* 2008.

[19] Jonghoon Seo, Seungho Chae, Jinwook Shim, Dongchul Kim, Cheolho Cheong, Tack-Don Han: *Fast Contour-Tracing Algorithm Based on a Pixel-Following Method for Image Sensors,* 2016

[20] Ratika Pradhan, Shikhar Kumar, Ruchika Agarwal, Mohan P. Pradhan, M. K. Ghose: *Contour Line Tracing Algorithm for Digital Topographic Maps*

[21] Satoshi Suzuki, Keiichi Abe*: Topological Structural Analysis of Digitized Binary Images by Border Following*, 1985. Academic Press

Függelék

F.1 Morfológiai műveletek eredménye

A következő ábrákon néhány adatsoron elvégzett morfológiai művelet eredménye látható. Egymás után szerepel a raszterizált adatsor, majd a zárás, a nyitás és végül a kontúrkeresés eredménye.