T.C.
ERCİYES ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

TOPOGRAFİK HARİTALARDAN YÜKSEKLİK ÇİZGİLERİNİN OTOMATİK ÇEKİLMESİ VE OLUŞAN HATALARIN GİDERİLMESİ

(Yüksek Lisans Tezi)

Hazırlayan
Emrah HANÇER

Danışman
Prof. Dr. Derviş KARABOĞA

Nisan 2012
KAYSERİ

BİLİMSEL ETİĞE UYGUNLUK

Bu çalışmadaki tüm bilgilerin, akademik ve etik kurallara uygun bir şekilde elde edildiğini beyan ederim. Aynı zamanda bu kural ve davranışların gerektirdiği gibi, bu çalışmanın özünde olmayan tüm materyal ve sonuçları tam olarak aktardığımı ve referans gösterdiğimi belirtirim.

Emrah HANÇER

Topografik Haritalardan Yükseklik Bilgilerinin Otomatik Çekilmesi ve Oluşan Hataların Giderilmesi adlı Yüksek Lisans tezi, Erciyes Üniversitesi Lisansüstü Tez Önerisi ve Tez Yazma Yönergesi'ne uygun olarak hazırlanmıştır.

Tezi Hazırlayan

Emrah HANÇER

Danışman

Prof. Dr. Derviş KARABOĞA

Bilgisayar Mühendisliği ABD Başkanı

Derviş KARABOĞA

Prof. Dr. Derviş KARABOĞA danışmanlığında Emrah HANÇER tarafından hazırlanan **"Topografik Haritalardan Yükseklik Bilgilerinin Otomatik Çekilmesi ve Oluşan Hataların Giderilmesi"** adlı bu çalışma jürimiz tarafından Erciyes Üniversitesi Fen Bilimler Enstitüsü Bilgisayar Mühendisliği Anabilim Dalında **yüksek lisans** tezi olarak kabul edilmiştir.

25 /04 / 2012

JÜRİ:

Danışman : Prof. Dr. Derviş KARABOĞA .....

Üye : Prof. Dr. Coşkun ÖZKAN .....

Üye : Doç. Dr. Veysel ASLANTAŞ .....

ONAY:
Bu tezin kabulü Enstitü Yönetim Kurulunun 03/07/2012 tarih ve 2012/28-03 sayılı kararı ile onaylanmıştır.

Prof. Dr. Necmettin MARAŞLI
Enstitü Müdürü

ÖNSÖZ / TEŞEKKÜR

İlgi duyduğum çalışmaları yapmam yönünde destek veren değerli danışmanım Prof. Dr. Derviş Karaboğa'ya, tez ile ilgili yardımcı olan Doç. Dr. Refik Samet'e, ayrıca akademik tecrübe ve bilgilerini benden esirgemeyen değerli hocalarım Yrd. Doç. Dr. Celal Öztürk ve Yrd. Doç. Dr. Necla Özkaya'ya teşekkürü borç bilirim.

Emrah HANÇER

Kayseri, Nisan 2012

TOPOGRAFİK HARİTALARDAN YÜKSEKLİK BİLGİLERİNİN OTOMATİK ÇEKİLMESİ VE OLUŞAN HATALARIN GİDERİLMESİ

Emrah HANÇER

Erciyes Üniversitesi, Fen Bilimleri Enstitüsü

Yüksek Lisans Tezi, Şubat 2012

Danışman: Prof. Dr. Derviş KARABOĞA

KISA ÖZET

Topografik haritalar, coğrafi bilgi sistemleri (CBS) için büyük önem arz etmektedir. Bu haritalardaki yükseklik bilgilerinin çıkarılması ve işlenmesi ise CBS uygulamalarında çok yaygın olan ve önem arz eden bir alandır. Özellikle üç boyutlu modellemeler için yükseklik bilgilerinin elde edilmesine ihtiyaç duyulmaktadır. Geçmişte yapılmış olan çalışmalar yeteri ölçüde başarılı olmaması nedeniyle bu alanda yeni çalışmalara ve araştırmalara ihtiyaç duyulmaktadır. Bu tezin amacı topografik haritalardan yükseklik çizgilerinin otomatik çekilmesi ve doğru bir şekilde bağlanmasını sağlayan özgün yöntemler tasarlamak ve uygulamasını yapmaktır.

Haritalardan yükseklik çizgilerinin çekilmesi, beş aşamadan oluşmaktadır: 1) Topografik haritanın sayısallaştırılması, 2) Segmentasyon ve eşikleme, 3) Gürültülü piksellerin filtrelenmesi, 4) İnceltme ve 5) İnceltilmiş eğrilerin vektörel dönüşümü. Beş aşama ilgili gerekli analiz ve uygulamalar yapılmakta olup, en önemli adım olan 5. aşamanın çözümüne yeni yöntemler geliştirilerek katkı sağlanmaktadır. Önerilen yöntemlerin geçmişte yapılan çalışmalara göre daha iyi performansa ve doğruluğa sahip olduğu sonucuna varılmıştır.

Anahtar Kelimeler: Coğrafi Bilgi Sistemleri, Topografik Harita, Yükseklik Çizgisi, Görüntü İşleme, Segmentasyon, Eşikleme, Filtreleme, İnceltme, Yükseklik Çizgisi Çekme, Yükseklik Çizgisi Bağlama, Eşleştirme, Parabolik Yön.

AUTOMATIC EXTRACTION OF CONTOUR LINES FROM TOPOGRAPHIC MAPS AND ELIMINATION OF OCCURED ERRORS

Emrah HANÇER

Erciyes University, Graduate School of Natural and Applied Sciences

M.Sc. Thesis, February 2012

Supervisor: Prof. Dr. Derviş KARABOĞA

ABSTRACT

Topographic maps are significant for geographic information systems (GIS). The extraction and reconnection of contour lines from topographic maps is popular and interesting area in GIS-based applications. Especially, for construction of DEM data the information of contour lines is needed. The aim of the thesis is to design applications to extract contour lines from topographic maps and then correctly reconnect them.

The extraction of contour lines from topographic maps includes five stages: (1) topographic map digitization by scanner; (2) color image segmentation and thresholding; (3) filtering noisy pixels; (4) thinning and pruning the binary image; and (5) raster to vector conversion of the resulting thinned lines. Necessary analysis and applications are applied for all the five stages, new approaches are improved to overcome the difficulties of stage (5), which is the most important stage. The results show that the proposed approaches performs better than related works in terms of accuracy and run-time.

Keywords: Geographic Information Systems, Topographic Maps, Contour Line, Image Processing, Segmentation, Thresholding, Filtering, Thinning, Contour Extraction, Contour Reconnection, Map Segmentation, Mapping, Parabolic Direction.

İÇİNDEKİLER

Sayfa

ŞEKİLLER LİSTESİ

GİRİŞ

Topografik haritalar, dünyanın bir bölümü veya belirli bir alan hakkında coğrafi bilgi veren haritalardır. Topografik haritalarda su birikintileri, akarsular, yükseklik çizgileri (yükselti eğrileri, kontur), bitki örtüsü ve yapısal alanlarla ilgili bilgilere yer verilmektedir. Bu bilgiler farklı piksel renkleri ile ifade edilmektedir. Örneğin; su birikintileri mavi tonlarda, yükseklik çizgileri kahverengi tonlarda ifade edilmektedir [1].

Topografik haritalarda çok büyük önem arz eden yükseklik çizgileri, deniz yüzeyinden aynı yükseklikte bulunan yerlerin birleştirilmesi ile elde edilen eğriler olarak tanımlanmaktadır. Yükseklik çizgileri yükseklik değerleri hakkında bilgi vermesinin yanında bir alanın yüzeyi hakkındaki üçüncü boyut bilgisini de içermektedir. Günümüzde maden araştırma kurumları, şehir planlama, jeoloji, jeofizik, coğrafi bilgi sistemleri (CBS), uzaktan algılama ve su kaynakları yönetimi topografik haritaların kullanıldığı alanlara örnek olarak verilebilir.

Geleneksel topografik haritalar kâğıtlara basılır ve insanların yorumlayacağı biçimde üretilir. Bilgisayar sistemleri tarafından yorumlanması mümkün değildir. Dolayısıyla bu tür haritaların bilgisayarların yorumlayacağı formata dönüştürmek önem arz etmektedir. Bu dönüşüme harita vektörizasyonu denilir [2].

Topografik haritaların önemli bir uygulama alanı olan CBS için yükseklik bilgilerinin çekilmesi ve tekrar bağlanıp bu bilgilerden sayısal yükseklik modeli (Digital Evaluation Model (DEM)) verisi oluşturmak önemli bir işlemdir. Bu konu ile ilgili birçok çalışma yapılmıştır. Ancak tatmin edici başarıya ulaşılamamıştır [1-3].

Bu çalışmada yükseklik çizgilerinin minimum hata ile elde edilmesi ve tekrar

bağlanması amaçlanmaktadır. Böylece oluşturulacak DEM verisi kalitesinin de yükseltilmesine önemli bir katkı sağlanacağı düşünülmektedir.

Tezin Amacı

Mevcut paket programlar sayısallaştırma işlemini manuel olarak yapabilmektedir. Lakin haritanın boyutu arttıkça sayısallaştırma işleminin bu yazılımlar ile yapılması önemli oranda zaman kaybına neden olmaktadır. Dolayısıyla otomatik olarak sayısallaştırma işlemlerini yapabilen programlara ihtiyaç vardır ve geçmişte yapılan çalışmalar, özellikle yükseklik çizgilerinin tekrar bağlanması işlemlerinde yeterli ölçüde başarı sağlayamamıştır.

Bu tez çalışması ile mevcut yapılan çalışmaların analizi yapılarak eksiklikleri görülüp önerilen iki yeni metot ile yükseklik çizgilerinin yeniden bağlanması aşamasında mevcut sistemlerden daha iyisi ortaya konulmaya çalışılmıştır.

Bu çalışma kapsamında;

- Tarama işleminden geçirilen topografik haritaların gürültülerden arındırılması için mevcut filtreler analiz edilmiş ve en iyisinin hangisi olduğuna karar verilmiştir.
- Tarama işleminden geçirilen ve gürültülerden arındırılan topografik haritalardan yükseklik çizgilerinin çekilmesi için mevcut teknikler (segmentasyon işlemleri) analiz edilmiş ve hangisinin daha iyi olduğuna karar verilmiştir.
- Tarama işleminden geçirilen ve gürültülerden arındırılan topografik haritalardan çekilen yükseklik çizgilerinde meydana gelen bozulmaları ve kopmaları gidermek için en iyi yükseklik çizgileri bağlama metodu belirleme kriteri geliştirilmiş ve önerilen metotlar mevcutlarla karşılaştırılmıştır.

Tezin Organizasyonu

Konu ile ilgili genel bilgiler ve literatürdeki çalışmalar 1. Bölüm'de verilmiştir.

2. Bölüm'de aşamalar ile ilgili uygulanan teknikler anlatılmış ve yükseklik çizgilerinin tekrar bağlanması için iki yeni metot önerilmiştir.

3. Bölüm’de önerilen metotlar belirlenen haritalara uygulanmış ve elde edilen bulguların analizi yapılmıştır.

4.Bölüm’de önerilen yaklaşımlar ve ortaya çıkan sonuçlar ile ilgili gerekli değerlendirmeler yapılmıştır.

1. BÖLÜM

LİTERATÜR TARAMASI

Bu bölümde yükseklik çizgilerinin çıkartılması ve tekrar bağlanması ile ilgili temel bilgilere ve yapılmış olan çalışmalara değinilmiştir.

1.1 Topografik Haritalar

Yeryüzünü görüntüleme teknolojilerindeki gelişmeler, raster verisini CBS verisi için önemli bir konuma getirmiştir. Uzaktan algılama, fotogrametri, hava indirme lazer sistemleri bu teknolojilere örnek verilebilir. Bu teknolojiler sayesinde CBS çok büyük boyutlarda raster verilerini yönetebilecek ve işleyebilecek duruma gelmiştir [3]. Şehir planlama, maden arama kurumları, askeri kurumlar ve daha birçok alanda CBS tarafından işlenmesi gereken haritalara ihtiyaç duyulmaktadır.

CBS'nin en önemli veri kaynaklarından biri olan topografik haritalar yeryüzüne ait bir alanla ilgili önemli bilgi sağlayan veri sistemleridir. Yükseklik çizgileri bu alanlardaki yükseklik farklarını ayırt etmek için kullanılır. Giriş bölümünde bahsedildiği gibi yükseklik çizgilerine ek olarak; bu haritalar akarsular, bitki örtüsü, yapılar ve yollar gibi birçok bilgi içermektedir. Ayrıca yükseklik çizgileri, arka plan ve bölgelerle ile ilgili bazı özellikleri belirtmek için sayısal ve sembolik bilgilere de başvurulmaktadır. Her bir katman farklı bir renk tonu ile ifade edilmektedir [4].

Geçmişte yapılan çalışmalar, yükseklik çizgilerinin çıkarılması konusunda yeterli ölçüde başarı sağlayamamıştır. Bölüm 2.1.1'de bunun nedenlerine ayrıntılı değinilmiştir. Yükseklik çizgilerini standartlaştıracak ve genelleyecek olursak üç özellik gereklidir [3, 5]:

- Yükseklik çizgileri gereksiz ayrıntıdan arındırılmış olmalıdır.
- Yükseklik çizgileri düzgün ve pürüzsüz olmalıdır.

- Yükseklik çizgilerinin karakteristik özellikleri korunmalıdır.

Yükseklik çizgileri bölge yüzeylerinin tanımlanması için kullanılan en önemli katmandır ve ilgili yüzeyin geometrik ve coğrafik özelliklerini tanımlar. Yükseklik çizgileri yükseklik bilgileri ile tanımlanır ve ifade edilir. Genellikle deniz seviyesi 0 (başlangıç) olarak kabul edilir. Dağların tepelerine, doruk noktalarına doğru yükseklik düzenli artarak devam eder. Artış miktarı haritaya ve konuma göre değişmektedir.

Yükseklik çizgileri sürekli ve üzerinde kopukluk olmayan hatta kapalı bir döngü olacak şekilde oluşturulan eğrilerdir. Yükseklik çizgilerinin bir diğer önemli özelliği de birbirleri ile kesişmemeleridir. Fakat yükseklik çizgilerinin bu özellikleri, çeşitli nedenlerden ötürü korunamaya bilmektedir. Yükseklik çizgilerinin özelliklerinin korunamamasına neden olan temel problemler şunlardır:

- Tarayıcılardan kaynaklanan problemler;
- Yükseklik çizgilerine ait yükseklik bilgilerinin yükseklik çizgilerinin üzerine yazılmasından kaynaklanan kopukluklar;
- Bazı yükseklik çizgilerinin diğerleri ile sürekli bir yapı içinde olmamasından kaynaklanan problemler;
- Bazı yükseklik çizgilerinin bazı yüksek yoğunluklu alanlarda net olarak çizilememesinden kaynaklanan problemler [3];
- Topografik haritalardaki katmanların konumundan kaynaklanan problemler

1.2 Literatür Taraması

CBS tarafından kullanılacak ve işlenecek haritaların raster verisinden vektör verisine çevrilmesi yani vektörleştirilmesi gerekir. Vektörleştirme, raster verisinin veya resminin vektör verisine çevrilmesi işlemidir. Taratıcı tarafından taratılmış bir resim veya bilgisayar ekranı tarafından görüntülenen bir duvar resmi, raster verisine örnek olarak verilebilir. Bu tür resimlerde görüntülenen detay, resmin boyutuna ve çözünürlüğüne bağlı olarak değişmektedir. Vektör verisi ise geometrik nesnelerden (çizgiler, açılar) oluşmaktadır. Vektör verileri CBS için kullanıldığı gibi CAD (Computer Aided Systems), OCR (Optical Character Recognition) gibi alanlarda da kullanılmaktadır [6].

Vektörleştirme işlemleri için geliştirilen çeşitli paket programları mevcuttur. Eroiica [6] da bu paket programlarından biridir. Eroiica'da 4 tane vektör çıkartım metodu bulunur:

- Eğimli doğruların sağlanması (Stentiford Method)
- Düz çizgilerin sağlanması (Zhang-Suen Method [7])
- Inner Kenar Algılama Algoritması (Inner Edge Detection)
- Canny Kenar Algılama Algoritması (Edge Detection [8])

Topografik haritalar için ise vektörleştirme işlemleri şu aşamalardan oluşmaktadır [1, 9]:

- Topografik haritanın sayısallaştırılması,
- Segmentasyon ve eşikleme,
- Gürültülü piksellerin filtrelenmesi,
- İnceltme
- İnceltilmiş resmin vektörel dönüşümü.

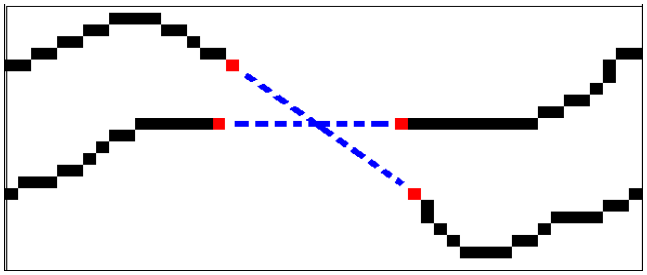
Haritaların vektörleştirilmesi ile ilgili yapılan çalışmalar resim tabanlı yaklaşımlar ve geometrik tabanlı yaklaşımlar olarak ikiye ayırılmıştır [10]. Daha sonra Pouderoux ve Spinello tarafından bu yaklaşımlara ek olarak gradyan vektör akış tabanlı yaklaşım da eklenmiştir [11].

1.2.1 Resim Tabanlı Yaklaşımlar

Görselliğe dayalı prensiplerden oluşmaktadır (Örneğin iki farklı bölge veya pikselin gruplanması). İki temel kriteri vardır: komşuluk (yakınlık) ve süreklilik [10].

Arrighi ve Soille [12], vektörleşme işlemlerinin 5 aşaması yanında matematiksel morfolojik işlemleri de kullanmıştır. Öncelikle kırmızı tondaki renkler resimden çıkarılmıştır. Fakat bu işlem birçok gürültüye neden olmaktadır. Daha sonra bu gürültüleri temizlemek için birden çok morfolojik filtre uygulanmıştır: (1) tekli piksellerin hit-or-miss transformation ile temizlenmesi, (2) bir piksellik kalın boşlukların homotopik olmayan hit-or-miss transformation ve bazı yapısal şablonlar ile doldurulması ve (3) yükselti çizgileri üzerindeki bütün boşlukların yok edilmesi. Skeletonization (belli bir alanı diğer alanlardan ayırmak ve vektörleştirme için ön işlem)

işleminden önce hit-or-miss transformation genelleştirilerek yükselti çizgilerinin çıkartılması işlemi uygulanmıştır. Son olarak da yükselti çizgileri arasındaki Öklid mesafeler ve yönler arasındaki farkları hesaplanır. Bütün ihtimal dâhilindeki uzaklıkların hesaplanması ve daha sonra en küçük uzaklıktaki uç noktaların bağlanması ile uygulama sona erer. Ancak bu yaklaşım gürültüye karşı çok duyarlı olmakla beraber, yükseklik çizgilerinin bilgisi açısından çok zayıf bir yaklaşımdır. Uç noktaların birleştirilmesi işleminde yeterli ölçüde başarı sağlayamamaktadır ve Şekil 1.1'de görülen hataya neden olabilmektedir.



Şekil 1.1 Yanlış bağlama hatası.

Eikvil ve arkadaşları [13], doğru üzerinde ilerlemeye dayalı bir yöntem önermiştir. Bu yöntemde bir boşluğun oluşması durumunda sadece bir yönde devamının mümkün olabileceği ve bu yönün de o çizginin bulunduğu yön olabileceği ve bir uç noktadan belli bir bölgede arama yapılarak boşluğun doldurulması ile problemin çözümü önerilmiştir. Bu metot [14]'teki çalışmada da kullanılmıştır. Bu metot da problem çözmede çok yetersiz kalmaktadır.

Khotanzad ve Zink [4], yükseklik çizgilerinin ve diğer katmanların renk bilgilerini kullanarak bir sistem geliştirmiştir ve dört adımdan oluşan bir metot önermiştir: (1) tarayıcıdan kaynaklanan renklerin karışması ve yanlış renklerin oluşumunun üstesinden gelmek için bir anahtar renk kümesi önermişlerdir; (2) Valley Seeking Algoritması ile doğrusal katmanlar çıkarılırken, diğer katmanların RGB renk uzayı histogramı analizi ile çıkarılması; (3) Oluşan boşlukları gidermek için A yıldız arama algoritmasının kullanılmasıdır. A yıldız arama algoritması San ve arkadaşları [15] tarafından da kullanılmıştır. A yıldız arama algoritması düğümler ve kenarlardan oluşan bir kümeden en yakın uzaklığın bulunması için kullanılan algoritmadır. Algoritma şu aşamalardan oluşmaktadır: (1) A yıldız yollarından en küçük uzaklıkta yolun seçilmesi, (2) İki ucun

bu yolla bağlanması, (3) Bu yolun A yıldız yollarından silinmesi ve (4) Diğer yolların da önceki işlemler takip edilerek kullanılıp A yıldız yollarından silinmesi.

Chang ve arkadaşları topografik haritalardan otomatik yol çıkarma yöntemi önermiştir [16]. Bu yöntem matematiksel morfolojideki koşullu ekleme içermektedir. Aynı renk tonunda olan yükseklik çizgileri ve yolların genişliklerinin birbirinden çok farklı olması sayesinde erosion (açma) morfolojik işlemi yapılarak yollar elimine edilir ve dilation (kapama) morfolojik işlemi ile de gürültüler elenir. Morfolojik işlemler [1]'de de kullanılmıştır. Ayrıca morfolojik işlemlerin dışında, Samet ve Namazov tarafından haritalarda bulanık filtreleme yöntemi önerilmiştir [17].

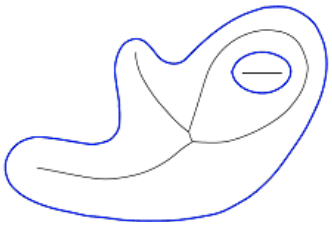
İsmail ve Azam tarafından haritaların sayısallaştırılması ile ilgili geniş çaplı bir doktora çalışması yapılmıştır [3]. Haritaların sayısallaştırılması ve ortaya çıkan hataların giderilebilmesi için analizler yapılmıştır. Ayrıca yükseklik çizgilerinin çıkartılması için kullanılan temel yaklaşımlar analiz edilmiş olup çalışmalarında morfolojik işlemler, median filtre ve renk kuantalaması gibi temel işlemler kullanmışlardır. Yükseklik çizgilerinin bağlanması için yükseklik çizgilerinin geometrik ve topolojik özelliklerinden faydalanılmıştır: (1) En yakın uzaklık, (2) Karşıt yönler ve (3) uç noktalar arsındaki açı. Yükseklik çizgilerinin bağlanması işlemi için geçmişte yapılan en kapsamlı çalışmadır. Ayrıca tam otomatik sistemlerin yeterli oranda başarı sağlayamaması nedeniyle yarı otomatik sistem önermişlerdir.

1.2.2 Geometrik Tabanlı Yaklaşımlar

Çizgeyi yeniden oluşturma işlemi daha genel bir problem olarak değerlendirilebilir: bir çizgiye ait noktalar kümesi V olsun. $G = (V, E)$ olacak şekilde bir çizge oluşturmak istersek, bir kenarın (edge) V kümesindeki iki noktası birbirine komşu olması halinde bağlanır. Biz bu çizgeye de bilinmeyen eğrinin poligon oluşturuşu diyoruz. Eğriyi yeniden oluşturma problemi grafiklerde ve hesapsal geometride sıklıkla kullanılmaktadır [11]. [11]'de belirtildiği gibi ilk algoritmalar uniform örnekleme koşulu üzerine kurulmuş olup, herhangi iki örnek arasındaki uzaklığın belli bir eşik değerden küçük olması prensibine dayanır.

Geometrik tabanlı yaklaşımların belki de en popüleri Amenta ve arkadaşlarının geliştirmiş olduğu metottur [18]. Bu metot, bir noktanın eğrideki orta (medial) eksenine

olan uzaklığını baz alır (Bkz. Şekil 1.2 [19]). Bu özelliği kullanarak, değişken yoğunluğunun örneklenmesini sağlayan uniform olmayan örnekleme koşulu oluşturulmuştur (Delaunay Üçgenleme). Bu örnekleme koşulunu sağlayan düzgün kapalı eğriler kümesinden doğru eğrinin yeniden oluşturulması hesabı yapılmıştır. Algoritma, nokta kümesinin Delaunay Üçgenlemesinin hesaplanması ile çalışır ve daha sonra doğrunun oluşması için filtrelenir. Bu yaklaşım [19]'da da kullanılmıştır. Ancak kompleks haritalarda bu algoritmanın doğruluk oranında çok başarılı olmadığı görülmüştür.



Şekil 1.2. Siyah ile gösterilen çizgiler, mavi çizgilerin medial ekseni.

Delaunay filtrelemeye bağlı algoritma incelemelerini [20]'de bulabilirsiniz. Düzgün kapalı eğrileri ile ilgili birçok çeşitli çalışmalar yapılmıştır [21, 22]. Dey ve arkadaşları [23], düzgün kapalı ve açık eğrileri beraber ele alarak genişletmiştir. Giesen [24], köşe alanlar için farklı bir örnekleme koşulu kullanmıştır. Giesen, Gezgin Satıcı Problemi'ni (GSP) tek kapalı eğriler için hatasız eğri oluşturma yöntemi olarak önermiştir. Daha sonra Althaus ve Mehlhorn tarafından GSP polinom zamanın içinde hesaplanarak genişletilmiştir [25].

Salvatore ve Guitton [10], yükseklik çizgilerinin tekrar oluşturulması amacıyla segmentasyonu yapılmış resmin filtrelenmesi ve inceltilmesi için Delaunay filtrelemeye bağlı bir yaklaşım önermişlerdir. Ayrıca, topografik haritaların bölgesel geometrik ve topolojik özellikleri de bu çalışmada kullanılmıştır. Fakat bu yaklaşımda kullanılan geometrik bilgiler çok sade ve yetersiz olmakla beraber, bu yaklaşım problemlerin çözümünde hiçbir garanti vermemektedir. Renklerin sınıflandırılması işleminde ise RGB renk uzayı yerine HSV renk uzayı kullanılmıştır. Yükseklik çizgilerinin HSV renk uzayındaki tonunun $10 < hue < 30$ arasında olduğu tespit edilmiştir.

1.2.3 Diğer Yaklaşımlar

Pouderoux ve Spinello [11], gradyen vektör akışı tabanlı tam otomatik bir yaklaşım önermişlerdir. Bu yaklaşım üç aşamadan oluşmaktadır: (1) Yükseklik çizgilerinin normallerinden gradyan yönlendime alanı üretilmesi, (2) Uç noktaların eşleştirilmesi ve (3) eşleştirilen uç nokta çiftleri arasındaki boşlukların doldurulması. Uç noktaların eşleştirilmesi için Gabow [26] tarafından önerilen yaklaşım kullanılmıştır. Uç noktaların bağlanması işlemi ise uç noktaların hesaplanan tanjantlarından yola çıkılarak yapılmıştır. Bu yaklaşım ayrıca [27]'de de uygulanmıştır. Bu yaklaşımın en önemli dezavantajı çalışma süresinin çok uzun olmasıdır.

[28]'de temel morfolojik işlemlerin yanında başlangıç noktalarının araştırılması için C-means (fuzzy) kümeleme ve yükseklik çizgilerinin çıkarılması ve bağlanması için geliştirilmiş ve GGVF (Generalized Gradient Vector Flow) aktif yükseklik çizgisi modeli [29, 30] önerilmiştir. Bu yaklaşımın da işlem süresi çok uzundur ve boşlukların doldurulması konusunda yeterli tatmin edici sonuçlar ortaya konulamamıştır.

2. BÖLÜM

UYGULAMA İÇİN ÖNERİLEN YAKLAŞIMLAR

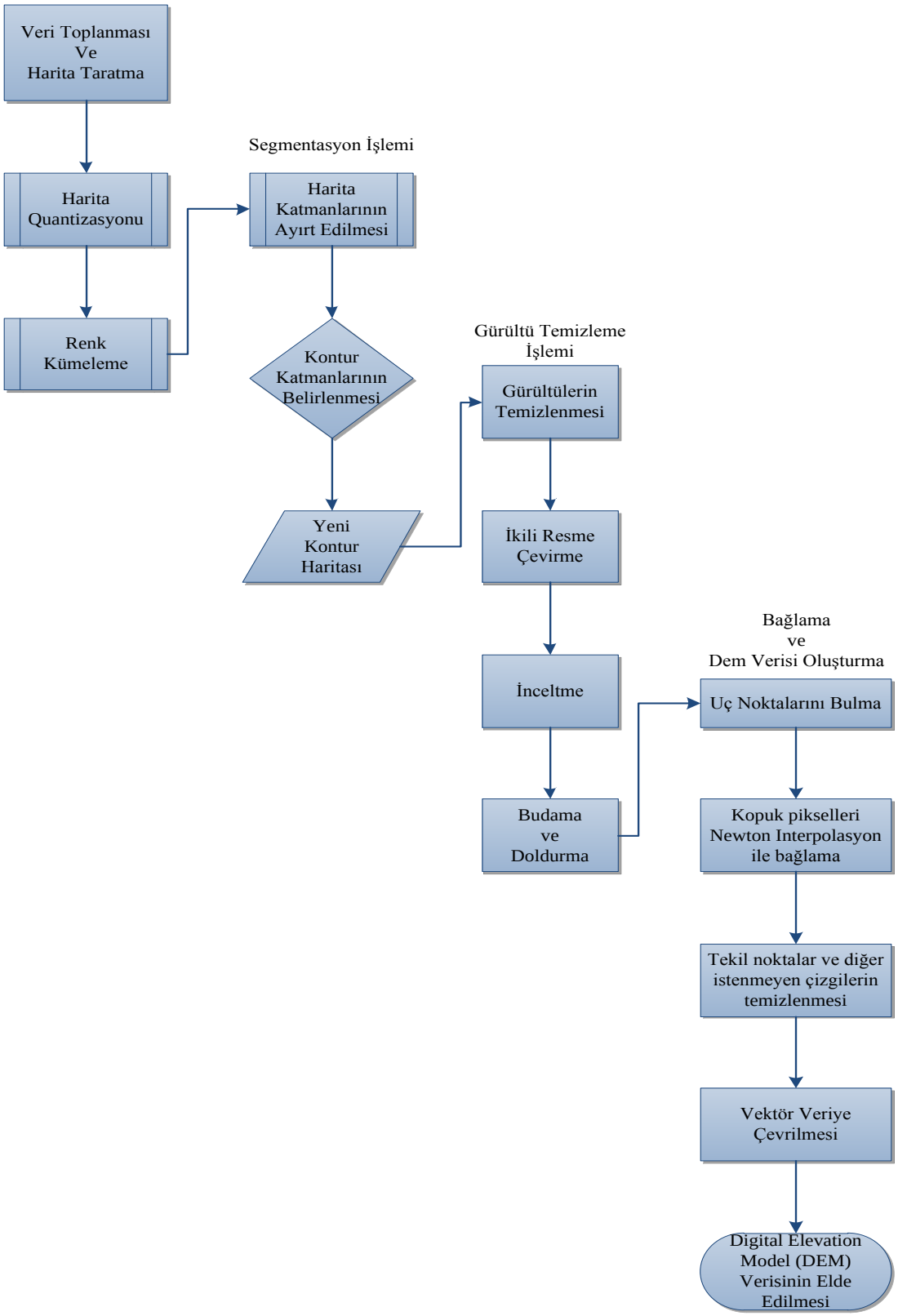
Bu bölümde ilk aşamadan son aşamaya kadar uygulamada neler yapıldığına değinilecektir. Şekil 2.1'de metodolojinin genel akış diyagramı verilmiştir. Uygulamaya Şekil 2.1'de bulunan ilk ve son adımları dahil edilmemiştir. Yani veri olarak taratılmış haritalar kullanıldı ve DEM modeli oluşturma işlemleri paket programlarına bırakıldı.

2.1 Topografik Haritaların Sayısallaştırılması

2.1.1 Veri Toplanması ve Harita Taratma

Günümüzde yazıcı ve tarayıcıların baş döndürücü bir şekilde geliştiği ve gelişmekte olduğu aşikârdır. Dolayısıyla kağıda çizimi yapılmış olan topografik haritalar gayet başarılı bir şekilde bilgisayar ortamına aktarılabilmektedir. Ancak renklerin çakışması, yanlış rengin oluşması gibi bazı eksiklikler de tarayıcılar tarafından tetiklenmektedir. Kullanılan yüksek teknolojili tarayıcılar, verinin daha detaylı bir şekilde alınmasına olanak sağlamakla birlikte verinin de kapasitesini dolaylı olarak arttırmaktadır. Bununla beraber yüksek çözünürlüğün yüksek kaliteli sonuçlar ve vektörleşmenin daha iyi yapılmasına dair sağladığı bir garanti yoktur. Diğer taraftan düşük çözünürlük de bazı süreksiz verilere ve boşluklara neden olmakta ve bu da mevcut veri üzerinde çalışma yapmayı güçleştirmektedir. Sonuç olarak, çözünürlüğün uygun bellekte hem kaliteyi hem doğruluğu sağlayacak şekilde seçilmesi gereklidir [3].

Taratma işleminden önce kağıda basılmış bir haritanın yüksek kartografik kalitede, çizgilerin, sembollerin ve yazıların anlaşılır olması gereklidir. Haritalarda gürültülere neden olmamak için temizlemede kullanılacak araçlar özenle seçilmelidir [3].



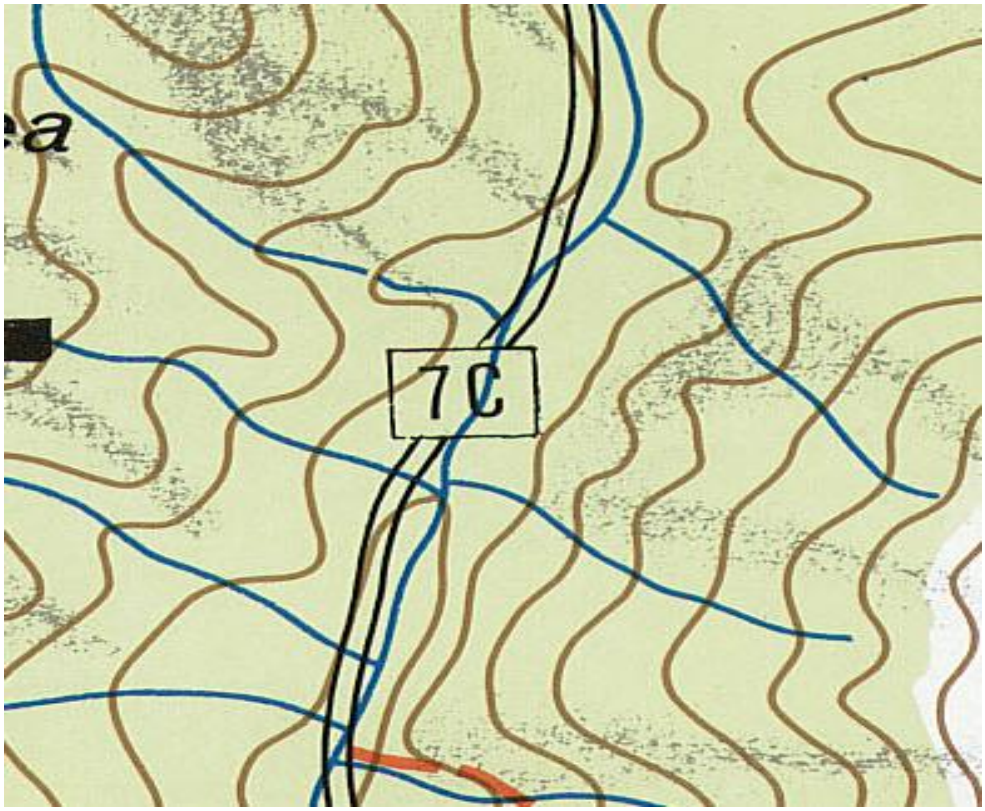
Şekil 2.1. Metodoloji genel akış diyagramı.

2.2 Segmentasyon

2.2.1 Renk Kuantalama

Haritaların birden fazla katmandan oluştuğundan önceki bölümlerde bahsedilmiştir. Bu katmanların her biri farklı bir piksel tonu ile ifade edilmektedir. Örneğin su birikintileri mavi tonlarda ifade edilirken, yükseklik çizgileri kahverengi tonlarda ifade edilmektedir [1]. Bu katmanları ayırt etmek bazı problemler nedeniyle zorlaşmaktadır. Bunların bazıları şunlardır:

- Bazı katmanların birbiri ile çakışması: yolların ve akarsuların yükseklik çizgilerinin üzerinden geçmesi buna örnek verilebilir (Şekil 2.2).
- Bazı sembollerin, yolların, köprülerin de kahverengi tonlarla ifade edilmesi: yükseklik çizgilerinin deniz seviyesinden yüksekliğini belirten numaraların yükseklik çizgisi ile aynı tonda olması örnek olarak verilebilir (Şekil 2.3).
- Yükseklik çizgilerinin bazılarının diğer yükseklik çizgilerinden daha farklı kahverengi tonlarında bulunması



Şekil 2.2. Akarsuların yükseklik çizgisi ile çakışmasına bir örnek.

Şekil 2.3. Yükseklik çizgisi ile aynı tonda sembole bir örnek.

Birçok durumda bir resimde gereğinden fazla renk tonları bulunabilmektedir. Özellikle topografik haritalar tarama formatında kaydedilirken renk tonu sayısı artmaktadır. Dolayısıyla bir harita üzerinde gerekli işlemlerin yapılabilmesi için renk sayısının azaltılması gerekmektedir. Bunun için de renk kuantalama (nicemleme) algoritmalarından faydanılmaktadır. Renk kuantalama bir resimdeki renk sayısının o renklerin en yakınındaki renkler ile değiştirilmesi ile azaltılması prensibine dayanır.

Heckbert, kuantalamayı dört adımda tanımlamıştır [31]:

- Renk istatistiği için orijinal resmin örneklenmesi.
- İstatistiğe göre renk uzayının seçilmesi.
- Renklerin renk uzayındaki örnek renk gruplarına eşleştirilmesi.
- Yeni resmin kuantalanması ve oluşturulması.

Birçok kuantalama algoritması mevcuttur. Median Cut Algoritması, uniform kuantalama, octree algoritması, popularity algoritmaları bunlardan bazılarıdır. Özellikle Heckbert [31] tarafından geliştirilen median cut algoritması yaygın olarak kullanılmaktadır.

Median Cut Algoritması histogram eşitlemeye dayalı bir algoritmadır. Bu algoritma ile kuantalama işleminden sonra resmin renk bilgisinin maksimum seviyede kalması sağlanır [32]. Algoritma dört adımdan oluşmaktadır:

1. RGB renk uzayında verideki tüm renkleri içeren en küçük hücrenin seçilmesi;
2. Hücrenin uzun olan ekseni boyunca renklerin küçükten büyüğe sıralanması;
3. Renkleri sıraladıktan sonra hücrenin uzun ekseni boyunca ortadan iki parçaya bölünmesi;

4. Madde 3'teki işlemin istenilen hücre sayısına bölünene kadar devam etmesi;

Kuantalama işlemlerine yardımcı olan bazı algoritma ve araçlar da kullanılmaktadır. Yapay Sinir Ağları, K-Means Kümeleme [33], Fuzzy Kümeleme [34] bunlardan sadece birkaçıdır ve yaygın olarak kullanılmaktadır. Ancak bu araçlar K-Means ve Fuzzy kümeleme algoritmaları büyük boyutlu haritalarda işlem yükünü arttırmaktadır. Dolayısıyla haritalardaki kuantalama işlemlerinde çok fazla tercih edilmemektedir.

2.3 Filtreleme ve Eşikleme

2.3.1 Gürültü Filtreleme

Gürültü, harita tarayıcıda taratılırken, kuantalama işleminin ardından veya katmanlar ayırt edilirken ortaya çıkmaktadır. Bu gürültüler yükseklik çizgilerinin çıkartılması ve işlenmesi işlemi için problem teşkil etmektedir. Gürültülerin temizlenmesi için çeşitli filtreler kullanılmıştır. Weiner, median, mean ve adaptive gaussian filtresi bunlardan sadece birkaçıdır. Sağladığı başarı nedeniyle bu çalışmada adaptive gaussian filtresi tercih edilmiştir. Ghircoias ve Brad, gürültü filtreleme işlemini kuantalama işleminden önce de uygulamıştır [19].

2.3.2 İkili Resme Çevirme

Bir eşik değer belirlenir ve vektörleşme işlemlerinin daha rahat yapılabilmesi için filtrelenmiş harita ikili resme çevirilir. Belli bir eşik değerin altında kalan değerler arka plan diğer değerler ise yükseklik eğrileri ve lineer şekiller olarak değerlendirilir [8].

$$g(x,y) = \begin{cases} 1 \text{ if } f(x,y) > T \\ 0 \qquad otherwise \end{cases} \tag{2.1}$$

2.3.3 Kenar Belirleme İşlemleri

İkili resme çevrilmesi işleminin ardından yükseklik çizgilerinin belirgin bir şekilde ortaya konulması gerekmektedir. Bunun için Sobel, Gradient, Laplacian, Prewitt, alçak geçirgen, yüksek geçirgen filtreler kullanılabilir [8]. Bu filtrelerin sağladığı başarı oranı haritaya göre değişmektedir. Bu işlemlerden sonra ayrıca morfolojik işlemler, dilation,

erosion, hit or miss transform, connected component analysis gibi işlemler de düzeltme işlemlerinde kullanılmaktadır.

2.4 İnceltme ve Budama

2.4.1 İnceltme İşlemleri

İnceltme (Thinning) işlemleri ile bir taraftan gereksiz veri yok edilirken, diğer taraftan da gerekli verinin de zarar görmemesi amaçlamaktadır. Çünkü gerekli verinin zarar görmesi, yükseklik çizgilerini tekrar bağlama işlemlerinde problem oluşturabilmektedir ve bu da oluşacak DEM verisinin istenen düzeyde olmamasına neden olacaktır. Birçok inceltme algoritması vardır. Lakin bunların birçoğu verinin (yükselti eğrilerinin) korunması açısından yeterli başarı sağlayamamaktadır. AForge.NET Framework [35] tarafından sunulan basit inceltme metodunun uygulanmış olduğu bir harita Şekil 2.4'te görülmektedir. Şekil 2.4'te de görüldüğü gibi inceltme işlemleri sonrası yer yer gürültülü pikseller göze çarpmaktadır.

Şekil 2.4 Simple Skeletonization yöntemi ile inceltilmiş harita.

Zhang-Seun algoritması ile istenilen inceltme sonuçlarına elde edilmiştir [7]. Zhang-Seun algoritması bu alanda birçok araştırmacı tarafından uygulanmış ve genel olarak çok iyi sonuçlar vermiştir [3, 19, 28]. Bu algoritma her bir pikselin 8 bağlantılı komuşuluğu analiz edilerek geliştirilmiştir.

Zhang-Suen algoritması iki iterasyondan oluşmaktadır. Piksellerin Şekil 2.5'teki gibi numaralandırıldığını kabul edersek;

p8	p1	p2
p7	p	p3
p6	p5	p4

Şekil 2.5. Piksellerin Zhang-Suen Algoritması için numaralandırılması.

1. İterasyon: Merkez piksel p aşağıdaki dört koşul sağlandığı takdirde silinmek için işaretlenir:

- $Connectivity = 1$ (Şekil 2.6'da $Connectivity$ için kullanılan metod verilmiştir.)
- p'nin en az 2 ve en fazla 6 tane nesne komşuluğu olabilir.
- p1, p3 ve p5 ten en az biri arka plan pikseli olmalı ($p1 \bullet p3 \bullet p5 = 0$).
- p3, p5 ve p7 den en az biri arka plan pikseli olmalı ($p3 \bullet p5 \bullet p7 = 0$).

2. İterasyon: Merkez piksel p aşağıdaki dört koşul sağlandığı takdirde silinmek için işaretlenir:

- $Connectivity = 1$
- p'nin en az 2 ve en fazla 6 tane nesne komşuluğu olabilir.
- p1, p3 ve p7 ten en az biri arka plan pikseli olmalı ($p1 \bullet p3 \bullet p7 = 0$)
- p1, p5 ve p7 ten en az biri arka plan pikseli olmalı ($p1 \bullet p5 \bullet p7 = 0$)

İki iterasyonun sonunda işaretlenecek başka piksel kalmamışsa, işlem durdurulur ve işaretlenmiş pikseller elimine edilir.

```
private int connectivity(int i,int j)
{
        int connected=0;

        if (greyPixelArray[i + 1, j] == 255
         && greyPixelArray[i + 1, j - 1] == 0) connected ++;

        if (greyPixelArray[i + 1, j - 1] == 255
         && greyPixelArray[i , j - 1] == 0)    connected ++;

        if (greyPixelArray[i , j - 1] == 255 &&
           greyPixelArray[i - 1, j - 1] == 0) connected ++;

        if (greyPixelArray[i - 1, j - 1] == 255
         && greyPixelArray[i - 1, j ] == 0)    connected ++;

        if (greyPixelArray[i - 1, j] == 255
         && greyPixelArray[i - 1, j + 1] == 0) connected ++;

        if (greyPixelArray[i - 1, j + 1] == 255
         && greyPixelArray[i , j + 1] == 0)    connected ++;

        if (greyPixelArray[i , j + 1] == 255
         && greyPixelArray[i + 1, j + 1] == 0) connected ++;

        if (greyPixelArray[i + 1, j + 1] == 255
         && greyPixelArray[i + 1, j ] == 0)    connected ++;

        return connected;

}
```

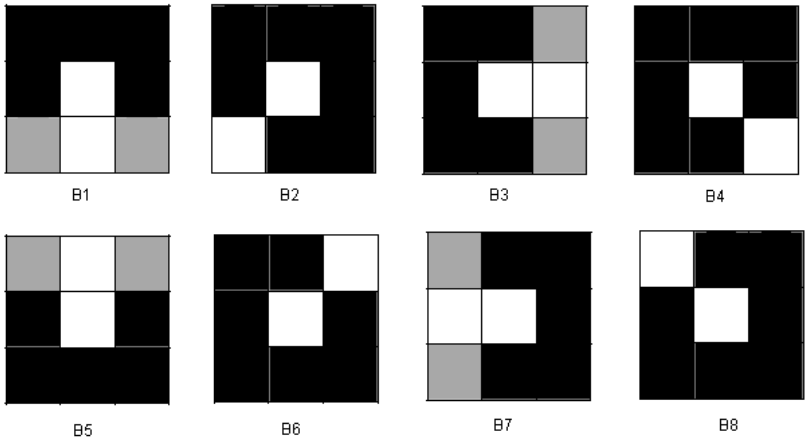
Şekil 2.6. Connectivity için kullanılan metod.

2.4.2 Budama ve Doldurma İşlemleri

İnceltme işlemlerinin ardından bazı yükseklik çizgilerinin üzerinde istenmeyen çıkıntılar meydana gelebilmektedir. Bu çıkıntılarda bulunan uç noktalar yükseklik çizgilerinin tekrar bağlanmasında yanlış bağlama problemlerine neden olmaktadır. Çıkıntıları yok etmek için budama (pruning) yöntemine başvurulur. Bu yöntemde belli bir eşik değere kadar olan tüm çıkıntılar yok edilirken diğer yükseklik çizgilerinin durumu korunur. Yükseklik çizgilerinin durumunun korunması ise doldurma işlemleri ile yapılmaktadır. Şekil 2.7 ve 2.8'de budama ve doldurma işlemleri için kullanılan şablonlar gösterilmiştir [8].

Budama işlemi 4 tane morfolojik işlem adımı içermektedir [19]:

- Şekil 2.7 deki şablonlar ile n adet erosion işlemine tabi tutulur (n: kullanıcı tarafından belirlenen iteration sayısı).
- Bir önceki işlemin ardından uç noktalar bulunur.
- Mevcut resim Şekil 2.7 de belirtilen şablonlar ile n adet dilation işlemine tabi tutulur.
- 1. Adım ve 3. Adım daki resimlerin bağlanması ile budanmış resim elde edilir.

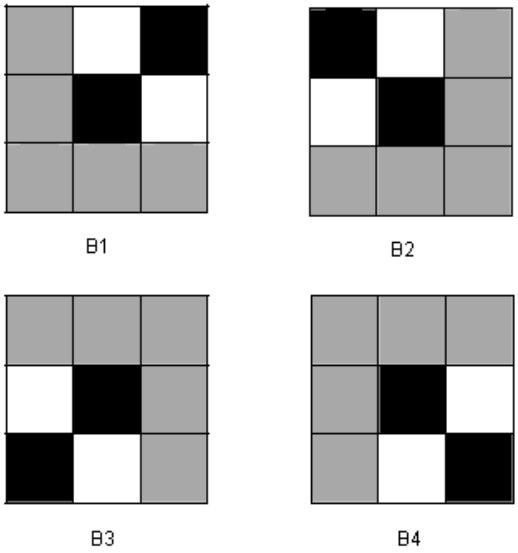


Şekil 2.7. Budama işlemleri için kullanılan şablonlar.

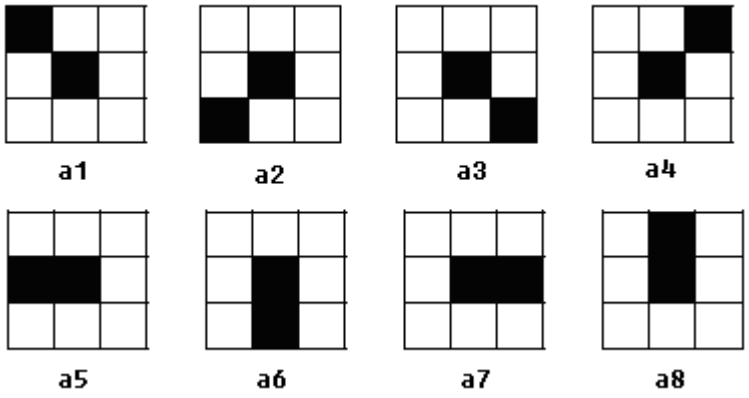
2.5 İnceltilmiş Çizgilerin Vektörel Dönüşümü

2.5.1 Uç Noktaların Bulunması

Uç noktalar Şekil 2.9'daki şablonlar kullanılarak bulunmaktadır. Şekil 2.10'da uç noktaların bulunması için kullanılan metot verilmiştir.



Şekil 2.8 Doldurma işlemleri için kullanılan şablonlar.



Şekil 2.9. Uç noktaların bulunması için kullanılan şablonlar.

p8	p1	p2
p7	p	p3
p6	p5	p4

if *(p8==0 & p7==255 & p6==255 & p5==255 & p4==255 & p3==255 & p2==255 & p1==255 & p==0)*

p yi uç nokta olarak ata;

elseif *(p8==255 & p7==0 & p6==255 & p5==255 & p4==255 & p3==255 & p2==255 & p1==255 & p==0)*

p yi uç nokta olarak ata;

elseif *(p8==255 & p7==255 & p6==0 & p5==255 & p4==255 & p3==255 & p2==255 & p1==255 & p==0)*

p yi uç nokta olarak ata;

elseif *(p8==255 & p7==255 & p6==255 & p5==0 & p4==255 & p3==255 & p2==255 & p1==255 & p==0)*

p yi uç nokta olarak ata;

elseif *(p8==255 & p7==255 & p6==255 & p5==255 & p4==0 & p3==255 & p2==255 & p1==255 & p==0)*

p yi uç nokta olarak ata;

elseif *(p8==255 & p7==255 & p6==255 & p5==255 & p4==255 & p3==0 & p2==255 & p1==255 & p==0)*

p yi uç nokta olarak ata;

elseif *(p8==255 & p7==255 & p6==0 & p5==255 & p4==255 & p3==255 & p2==0 & p1==255 & p==0)*

p yi uç nokta olarak ata;

elseif *(p8==255 & p7==255 & p6==0 & p5==255 & p4==255 & p3==255 & p2==255 & p1==0 & p==0)*

p yi uç nokta olarak ata;

Şekil 2.10. Uç noktaların bulunması için kullanılan metot.

2.5.2 Yükseklik çizgilerinin Klasik Metot ile Bağlanması

Yükseklik çizgilerinde meydana gelen kopukluklar genel olarak şu nedenlerden kaynaklanabilir [3]:

- Haritaların karanlık bölümlerindeki yükseklik çizgilerinin saptanamaması.
- Taratıcılardan kaynaklanan problemler

- Bazı katmanların üst üste gelmesi ve çakışması; bundan dolayı da katmanların bazı bölümlerinin sınıflandırılamaması
- Bazı yükseklik çizgilerinin kesikli çizgiler olarak görünmesi ve bunun harita üreticileri tarafından ihmal edilmesi
- Ön işlemler yapılırken kaynaklanan kopukluklar

Yükseklik çizgilerinin bağlanması aşaması daha önceki bölümde bahsi geçen beş aşamanın içinde en zor bölümdür. Çok fazla ve değişik türdeki kopukluklar, bu problemin yapay zeka teknikleri ile veya optimizasyon algoritmaları [36, 37] ile çözümünü zorlaştırmaktadır. Bu yüzden bu probleme yönelik özel yöntemlerin geliştirilmesi gereklidir. Önceki bölümde bahsedildiği gibi geçmişte bu problemi çözmeye yönelik yapılmış olan çalışmalar yeterli düzeyde değildir. Dolayısıyla özellikle bu problemin çözümüne ilişkin çalışmalar üzerine yoğunlaşılmıştır.

Temel resim tabanlı uygulamalardan biri olan iskelet çıkarma ve uygun olan ilgili uçları bağlama metodu incelendi. Ancak bu metodun [3] ve [10]'da belirtildiği gibi gürültülere karşı çok hassas olması başarı oranını düşürmektedir.

Tam otomatik sistemler [11, 12, 19] yetersiz olmakla beraber sistemlerin daha da geliştirilmesinin ve sistemlere müdahalenin çok fazla mümkün olmadığı görülmüştür. Tam otomatik sistemlerin yetersiz kalması ve bu sitemlerden kaynaklanan hataların düzeltme şansının yok denecek kadar az olması nedeniyle, yarı otomatik sistem geliştirilmesi uygun görüldü.

[3]'de yükseklik çizgilerinin geometrik ve topolojik özelliklerinden yola çıkılarak öne sürülen yarı otomatik klasik yaklaşım detaylı olarak incelenmiş ve uygulaması yapılmıştır (Bu yaklaşım **Klasik Metot** olarak bahsedilecektir).

Klasik Metot temel olarak üç temel adımdan oluşmaktadır:

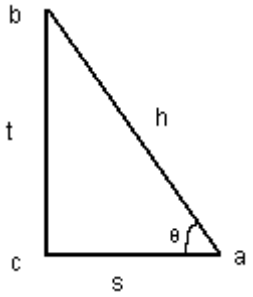
- Uç noktaların bulunması
- Uç noktaların eşleştirilmesi
- Cubic Spline Interpolasyon tekniği ile eşleştirilmiş uç noktaların bağlanması

Uç noktaların bulunması Bölüm 2.3.1'de anlatılmıştır. Uç noktaların eşleştirilmesi için

üç bilgiden faydalanılmıştır: (1) en yakın uzaklık, (2) yön ve (3) açı. Öncelikle bir uç nokta merkez olacak şekilde kullanıcı tarafından belirlenen $2hx2h$ büyüklüğünde bir şablon etrafındaki uç noktalar aranır. Şayet şablonun merkezinde bulunan uç nokta ile şablondaki herhangi bir uç nokta uzaklık, yön ve açı şartını sağlıyorsa bu iki uç nokta bağlanır.

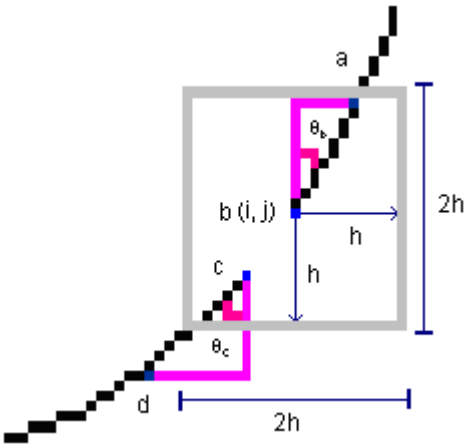
İki uç nokta arasındaki uzaklığı bulmak için 2.2'deki denklem (Öklid Uzaklık) kullanılır (Bkz. Şekil 2.11).

$$dis\tan ce = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{2.2}$$



Şekil 2.11. Açı hesabı için kullanılan üçgen modeli.

İki uç nokta arasındaki açıyı bulmak için bu uç noktalardan 10 piksel kadar geri gidilir. Geri gidilmesinin akabinde bulunan pikseller ile uç noktalar arasında üçgensel bölge oluşturularak açı hesaplanır. Şekil 2.12'de açı ve yönün hesaplanması ile ilgili süreç gösterilmiştir [3].



Şekil 2.12. Kopuk yükseklik çizgisi parçası için uzaklık, açı ve yön belirlenmesi [3].

h : Uç noktadan öklit uzaklık yaklaşımı

$a(x_1, y_1)$: $b(x_2, y_2)$ noktasının 10 piksel gerisindeki piksel

$b(x_2, y_2)$: Uç nokta

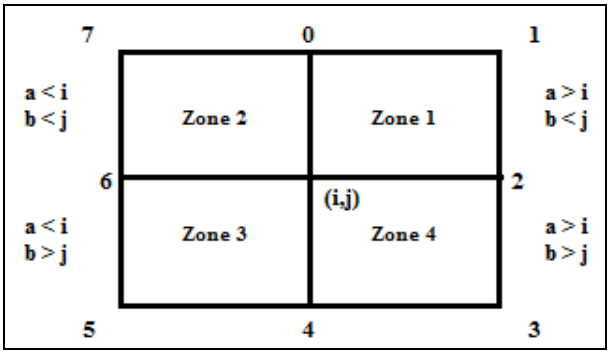
$c(x_3, y_3)$: $2hx2h$ şablonun içindeki en yakın uç nokta

$d(x_4, y_4)$: $c(x_3, y_3)$ noktasının 10 piksel gerisindeki piksel

$$\theta_b = \sin^{-1}\left(\frac{|x_2 - x_1|}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}}\right) \tag{2.3}$$

$|\theta_b - \theta_c| < Açı, \theta_b$ ab ile Y-axis arasındaki açı ve θ_c cd ile Y-axis arasındaki açı. Bu iki açı arasındaki farkın belli bir açı değerinden küçük olması gereklidir. Açı 90 dereceden fazla olamaz.

Yön hesabı için toplam 8 tane yön vardır. Bunlar 0 ile 7 arasında yer alır. Şekil 2.13'de gösterilmiştir. Yönlerin eşleştirilmesine örnek verecek olursak yön 5 yön 1 ile eşleşir, yön 7 yön 3 ile eşleşir. Yön hesabı için kullanılan metotlar Şekil 2.14 ve 2.15'te verilmiştir.



Şekil 2.13. 8 yönün dizilimi.

```
i: uç noktanın x koordinatı;
j: uç noktanın y koordinatı;
[a,b] uç noktadan 10 piksellik geri gidilen noktanın x ve y koordinatı;
direction = -1;
if (a > i && b < j) direction = 1;
else if (a < i && b < j) direction = 7;
else if (a < i && b > j) direction = 5;
```

```
else if (a > i && b > j) direction = 3;
else if (a == i && b < j) direction = 0;
else if (a == i && b > j) direction = 4;
else if (a > i && b == j) direction = 2;
else if (a < i && b == j) direction = 6;
```

Şekil 2.14. Yönün hesaplanması için kullanılan metot.

```
private int[,] backwardtenpixel(int i, int j)
    {
        int[,] A = new int[1, 2];
        int x = i;
        int y = j;
        int back = 0;
        while (back < 10)
        {
            if ( x-1 <= 0 || x+1>=BitmapFilter.height-1 || y-1<=0 || y+1 >= BitmapFilter.width-1)
                break;
            if (greyPixelArray[x - 1, y - 1] == 0)
            {
                greyPixelArray[x - 1, y - 1] = 1;
                x = x - 1; y = y - 1;
            }

            else if (greyPixelArray[x, y - 1] == 0)
            {
                greyPixelArray[x, y - 1] = 1;
                y = y - 1;
            }

            else if (greyPixelArray[x + 1, y - 1] == 0)
            {
                greyPixelArray[x + 1, y - 1] = 1;
                x = x + 1; y = y - 1;
            }

            else if (greyPixelArray[x - 1, y] == 0)
            {
                greyPixelArray[x - 1, y] = 1;
                x = x - 1;
            }

            else if (greyPixelArray[x + 1, y] == 0)
            {
                greyPixelArray[x + 1, y] = 1;
```

```
            x = x + 1;
        }

        else if (greyPixelArray[x - 1, y + 1] == 0)
        {
            greyPixelArray[x - 1, y + 1] = 1;
            x = x - 1; y = y + 1;
        }

        else if (greyPixelArray[x, y + 1] == 0)
        {
            greyPixelArray[x, y + 1] = 1;
            y = y + 1;
        }

        else if (greyPixelArray[x + 1, y + 1] == 0)
        {
            greyPixelArray[x + 1, y + 1] = 1;
            x = x + 1; y = y + 1;
        }
        back = back + 1;
    }

    A[0, 0] = x; A[0, 1] = y;
    birlerisıfırla();
    return A;
}
```

Şekil 2.15. 10 piksel geri gidilmesi için kullanılan metot.

Bütün kriterler sağladıktan sonra iki uç nokta birbiriyle eşleşir ve bunların bağlanması gerekir. Bağlama işlemi için çeşitli interpolasyon yöntemleri vardır. Klasik metotta küçük kopukluklar için Cubic Spline Interpolasyon [38], büyük kopukluklar için Newton Interpolasyon kullanılmış. Lakin Newton Initerpolasyon hem büyük hem de küçük kopukluklarda düzgün bir bağlama sağlaması ve Cubic Spline Interpolasyon'un işlem hacminin daha fazla olması sebebiyle sadece Newton Interpolasyon [39] kullanılmıştır. Denklem 2.4, 2.5, 2.6 ve 2.7'de Newton Interpolasyon yönteminin nasıl çalıştığı gösterilmiştir.

$$P_n(x) = c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) + \ldots + c_n(x - x_0)(x - x_0)\ldots(x - x_n) \quad (2.4)$$

$$f[x_0] = c_0 = f(x_o); \quad (2.5)$$

$$f[x_0, x_1] = c_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0} \tag{2.6}$$

$$f\left[x_0, x_{1,} x_{2,} \ldots, x_n\right] = c_n \tag{2.7}$$

2.5.3 Yükseklik çizgilerinin I. Yeni Metot ile Bağlanması

Klasik Metot temel, hızlı ve yarı otomatik bir yaklaşım olmasına karşın önemli eksiklikleri de bünyesinde bulundurmaktadır. Şekil 2.16'da görüleceği gibi parabolik yön çözümü için çaresiz kalmaktadır. Uç noktadan 10 piksel geri gidilemediği durumlarda yine bağlama mümkün olmamaktadır. Ayrıca Öklid mesafesi dikkate alınarak yapılan eşleştirme işlemi yanlış bağlama sorunlarına yol açabilmektedir.

Şekil 2.16. Klasik metot ile bağlanmayan uç noktalara örnek.

Bu problemleri çözmek için Klasik Metot'a bazı parametreler ilave edilerek yeni bir metot geliştirildi (Bu yaklaşımdan "I. Yeni Metot" olarak bahsedilecektir) [40]. I. Yeni Metot klasik metoda göre üç yeni parametre içerir: (1) Parabolik Yön, (2) Uç noktadan 10 pikselden daha az gidilebilen noktaları işleme dahil etmek ve (3) Şablonun içindeki toplam piksel sayısı tek sayı ise şablon boyutunu arttırarak piksel sayısını çift sayıya tamamlama.

Parabolik yön bilgisi [40]'da ilk defa kullanılmıştır. Bu bilgi ile bağlanmayan kopuklukların hepsinin bağlanması ve bu sayede doğruluk oranının arttırılması amaçlanmıştır.

Haritalarda çok küçük yükseklik çizgileri kopukluklarının ortaya çıkabilmesi mümkündür. Bu kopukluklarda bir uç noktadan 10 piksel geriye gitmek mümkün değildir. Dolayısıyla bu pikseller işleme dahil edilerek doğruluk oranının arttırılması amaçlanmıştır.

Şablon düzenli olarak büyütülerek işlem süresinin kısaltılması amaçlanmıştır. İşlem süresinin kısaltılması, her bir uç noktanın bağlanacağı bir çifti olduğunu düşünürsek şablonun içindeki piksel sayısının tek sayı olduğu hatta bir olduğu durumda (şablon büyütülüp) uç nokta sayısı çift sayıya tamamlanarak, uç noktaların o anda kullanılan şablon içinde bağlanması ve dolayısıyla da kullanılan şablon sayısının azaltılması sağlanmıştır. Sonuç olarak çalışma süresinde önemli oranda iyileşme sağlanmıştır.

I. Yeni Metot ile önce karşıt yönlerdeki uç noktalar bağlanır. Daha sonra parabolik yönlerdeki uç noktalar bağlanarak işlem tamamlanır. Özellikle kompleksliğin az olduğu ve küçük boyutlu haritalarda çok iyi sonuçlar verdiği görülmüştür. Yarı otomatik bir sistem olması sayesinde seçilen bir şablonda hatalı bir bağlama olması durumunda mevcut işlem iptal edilip yeni bir şablon seçilerek hatanın düzeltilmesine de imkan sağlamaktadır. Şekil 2.17 ve 2.18'de I. yeni metodun sözde kodu görülmektedir.

Modül 1: En Yakın Uç Nokta Bulma

Adım1: Resim üzerinde bir uç nokta için satır satır arama yap

Adım2: Eğer bir uç nokta bulunursa, bu uç noktayı (e_0) merkez olacak şekilde $2hx2h$ boyutunda bir kare şablon tanımla

Adım3: Kare şablonun içinde kalan diğer tüm uç noktaları $e_1, e_2, e_3, ...e_i..., e_k$ bul

Adım4: Toplam uç nokta sayısı $k+1$ tek sayıysa; şablonun boyutunu büyüt, çift sayıya ulaşana kadar Adım2 ve Adım3'ü tekrarla.

Adım5: e_0 ile en yakın uzaklıktaki uç noktayı bul.

Adım6: Şablonun boyutunu ilk haline çevir.

Modül 2: Yön Kontrolü

Adım1: Eğer Modül 1 ile gelen iki uç nokta bulunmuşsa, bu uç noktalardan 10 piksel kadar geriye git.

Adım2: Eğer 10 piksel geriye gidilemiyorsa, gidilebildiği kadar git.

Adım3: Şekil 2.13'deki yön sistemini oluştur.

Adım4: Karşıt yöndeki boşlukları doldurmak için karşıt yönleri kullan; Karşıt yönler:

0 Yönü 4 Yönü ile

1 Yönü 5 Yönü ile

2 Yönü 6 Yönü ile

3 Yönü 7 Yönü ile

Adım5: Parabolik yöndeki doldurmak için parabolik yönleri kullan; Parabolik yönler:

1 Yönü 3 Yönü ile

3 Yönü 5 Yönü ile

5 Yönü 7 Yönü ile

7 Yönü 1 Yönü ile

Adım6: Uç noktaların yönünü bul.

Adım7: Eğer karşıt yöndeki boşlukların doldurulması aşamasında isen, Adım4'ü uygula; eğer parabolik yöndeki boşlukların doldurulması aşamasında isen Adım5'i uygula.

Modül 3: Açı Kontrolü

Adım1: Eğer Modül 2 den gelen iki uç nokta bulunmuşsa, bu uç noktaların gerideki pikselleri ile olan açılarını hesapla

Adım2: Eğer açılar arasındaki fark eşik değerinden küçükse, bu uç noktaları bağlanmak üzere birbirine eşle.

Şekil 2.17. I. Yeni metot eşleştirme aşamaları.

Adım1: Eğer iki uç nokta eşlenmişse, bunların birine b diğerine c olarak ata.

Adım2: b nin gerideki pikseline a ve c nin gerideki pikseline d olarak ata.

Adım3: Bu dört noktayı kontrol noktaları olarak interpolasyon için kullan:

$$a = (x_0, f(x_0))$$

$$b = (x_1, f(x_1))$$

$$c = (x_2, f(x_2))$$

$$d = (x_3, f(x_3))$$

Adım4: Newton polinomunun katsayılarını hesapla:

$1st$ coefficient $\quad f[x_0] = c_0 = f(x_o);$

$2nd$ coefficient $\quad f[x_0, x_1] = c_1 = \dfrac{f(x_1) - f(x_0)}{x_1 - x_0};$

...

nth coefficient $\quad f[x_0, x_1, x_2, \ldots, x_n] = c_n.$

Adım5: Newton Interpolasyon Polinomunu oluştur:

$$P_n(x) = c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1)$$

$$+ \cdots + c_n(x - x_0)(x - x_0)\ldots(x - x_n).$$

Şekil 2.18. Eşleşmiş uç noktaların bağlanma aşaması.

2.5.4 Yükseklik çizgilerinin II. Yeni Metot ile Bağlanması

I. Yeni Metot özellikle kompleksliğin az olduğu ve küçük boyutlu haritalarda hem sonuç hem de performans olarak iyi sonuç vermektedir. Ancak büyük boyutlu işlenmiş topografik haritaların da varlığı azımsanmayacak sayıda olup, bu haritalarda kompleks kopuklukların da bulunabilmesi mümkündür. Dolayısıyla bu haritaları işleyebilmek için bağımsız bir metot geliştirilmiştir (Bu çalışma "II. Yeni Metot" olarak anılacaktır.) [41].

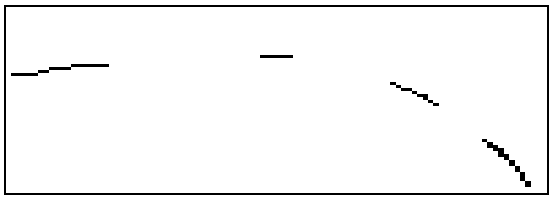
II. yeni metodun klasik metot ve I. yeni metot ile yarı otomatik olması, şablon boyutunun kullanıcıdan istenmesi gibi bazı benzerlikleri olmasına karşın kullanılan özellikler tamamen farklıdır. Bu özellikler şöyledir: (1) parabolik ve zıt yönlerin birlikte kullanımı, (2) x ekseni veya y ekseni üzerindeki yönlerin saatin tersi yönündeki komşu yön ile değiştirilmesi, (3) uç noktaların ordinatları arasındaki fark ve (4) bir şablonun içinde yer alan küçük yükseklik çizgisi parçasının bir ucu eşleştirme için kullanılıyorsa diğer ucun bu işlem sırasında kullanılmamasıdır.

Parabolik ve Zıt Yönlerin Birlikte Kullanımı: Daha önce de belirtildiği gibi zıt yönlerin tek başına kullanımı çok etkili olmamaktadır. Parabolik ve zıt yönlerin birlikte kullanımı ile birlikte bütün yön alternatifleri değerlendirilip doğru bağlanma şansı arttırılmıştır. Uç noktaların çakışarak bağlanmasının engellenmesi amaçlanmıştır.

X ve Y Ekseni Üzerindeki Yönlerin Komşuluğundaki Yöne Atanması: Genel olarak yükseklik çizgileri zıt ve parabolik yönlerde dağılım gösterirler. Çok nadir x ve y ekseni üzerinde görünürler. Bu şekilde görünmesinde 10 piksel geriye gidilmesinin de etkisi vardır. Yani 10 piksel geri gidişlerde çizginin yönü hesaplandığı zaman çizgi x ve y ekseni üzerindeymiş gibi görünebilir. Dolayısıyla x ve y ekseni üzerinde bulunan bir uç noktayı da çifti ile eşleştirmek çok zordur çünkü başta da söylediğimiz gibi yükseklik çizgileri genel olarak zıt ve parabolik yönelimlidir. Dolayısıyla x ve y ekseni üzerinde görünen yönlerinin saatin tersi yöndeki en yakınında bulunan diğer bir söylemle zıt veya parabolik yönlere çevrilmesi ile bu problemin çözülmesi amaçlanmıştır.

Uç Noktaların Ordinatları Arasındaki Fark: Bir yükseklik çizgisindeki kopukluğun üzerinde yer alan iki uç nokta topografik haritaların özelliğinden dolayı aynı hizada bulunmaktadır. Hiza bilgisinin nasıl kullanılacağına ilişkin araştırmalar yapılmıştır. Yapılan araştırmalar sonucunda genel olarak aynı yükseklik çizgisi üzerinde yer alan, bağlanması gereken iki uç noktanın y'lerinin farkının diğer uç noktalara göre minimum olduğu sonucuna varılmıştır.

Aynı Yükseklik Çizgisi Üzerindeki Yakın Uç Noktaların Kullanılma Şekli: Topografik haritalarda Şekil 2.19'dan da görüleceği gibi küçük kopukluklardan oluşan parçacıklar bulunabilmektedir. Bu parçacık üzerinde yer alan iki nokta yön değerlendirmesinde birbirini sağlamakta ve yanlış bağlama işlemi neticesi verebilmektedir. Bunu engellemek için şablonun altında kalan parçacıktaki iki uç noktadan biri kullanılıyorsa diğerinin aynı anda kullanılmamasına karar verilmiştir.



Şekil 2.19. Küçük kopuk yükseklik çizgileri parçacıkları.

II. Yeni metot performans açısından diğer metotlara göre daha yavaş olmasına karşın doğru bağlama oranı olarak klasik ve I. yeni metotlardan daha iyi sonuç vermektedir. Ayrıca ilk aşamalarda klasik metot, diğer aşamalarda II. yeni metot kullanılarak bu metodun performansının iyileştirilmesi amaçlanmıştır. Şekil 2.20'de II. yeni metodun eşleştirme adımları görülmektedir. Bağlama aşaması Şekil 2.18'de belirtildiği gibi yapılmaktadır.

Modül 1: Şablondaki uç noktaların bulunması

Adım1: Resim üzerinde bir uç nokta için satır satır arama yap

Adım2: Eğer bir uç nokta bulunursa, bu uç noktayı (e_0) merkez olacak şekilde $2h x 2h$ *boyutunda bir kare şablon tanımla*

Adım3: e_0'dan en fazla 20 piksel kadar geri gitmek kaydıyla; eğer bir uç noktaya rast gelinirse, o uç noktayı e_0 kullanılırken değerlendirmeye alma

Adım4: Şablonun içinde kalan diğer uç noktaları $e_0, ..., e_i, ..., e_N$ *bul*

Modül 2: Yön kontrol

Adım1: Şekil 2.13'deki yön sistemini oluştur.

Adım2: Uç noktaların yönlerini, uç noktalardan 10 piksel geriye giderek hesapla. Eğer 10 piksel geri gitmek mümkün olmuyorsa gidebileceği kadar geriye git ve o pikseli yön hesaplamasında kullan.

Adım3: Eğer yön x ekseni veya y ekseni üzerinde görünüyorsa, yönü saatin tersi yönündeki en yakın yön ile değiştir:

Yön 0'ı Yön 7 ile

Yön 6'yı Yön 5 ile

Yön 4'ü Yön 3 ile

Yön 2'yi Yön 1 ile

Adım4: Merkez uç nokta e_0 *ile karşıt ve parabolik yön özelliğini sağlamayan uç noktaları ele.*

Zıt yönler için:

1 Yönü 5 Yönü ile

3 Yönü 7 Yönü ile

Parabolik yönler için:

1 Yönü 3 Yönü ile

3 Yönü 5 Yönü ile

5 Yönü 7 Yönü ile

7 Yönü 1 Yönü ile

Modül 3: y ekseni kontrolü

Adım1: Modül 2'den gelen merkezi uç nokta ile diğer uç noktaların y'lerinin farklarını bulunuz.

Adım2: $min \mid y_0 - y_{i \in D} \mid$ *denklemini kullanarak merkez uç nokta ile minimum y ekseni farkına sahip uç noktayı merkez uç nokta ile bağlanmak üzere eşleştiriniz.*

Şekil 2.20. II. Yeni metot eşleştirme aşamaları.

3. BÖLÜM

UYGULAMA SONUÇLARI

Bu bölümde üç tane harita üzerinde gerekli uygulamalar ve analizler yapılıp sonuçları verilecektir. Haritalar komplekslik ve büyüklük olarak küçükten büyüğe doğru seçilmiştir. Topografik haritaların segmentasyonu ve yükseklik çizgilerinin çıkarılması işlemleri Bölüm 2'de anlatılan yöntemler ile yapılacaktır. Yükseklik çizgilerinin bağlanması işlemleri ise daha ayrıntılı bir şekilde işlenecektir. Değerlendirmeler kullanılan haritalar, boyutları ve uygulanan metotlar ile beraber doğru bağlanma sayısı, hatalı bağlanma sayısı, boşluk (bağlanmama) sayısı, çalışma zamanı (saniye) ve doğruluk oranı temel alınarak yapılmıştır ve Tablo 3.1, 3.2 ve 3.3'de verilmiştir.

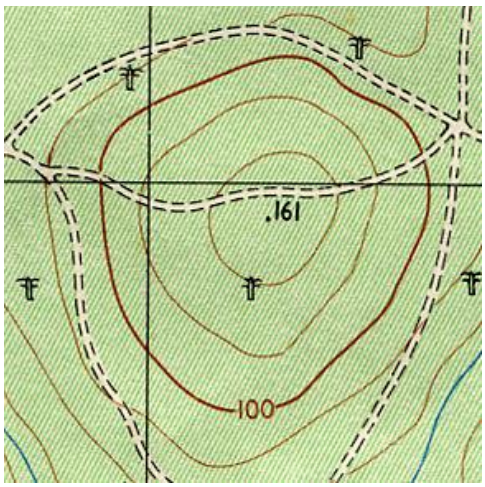
Uygulama, Visual Studio Net. platformunda C# dili kullanılarak geliştirilmiştir. Matlab Image Processing Toolbox ve AFORGE.NET kütüphanelerinden filtreleme ve segmentasyon işlemlerinde yararlanılmıştır. Kullanılan bilgisayar Inter Core 2 Duo 2.00 GHz işlemcilidir. Kullanılan haritalar jpeg, bmp ve png formatındadır.

3.1 UYGULAMA 1

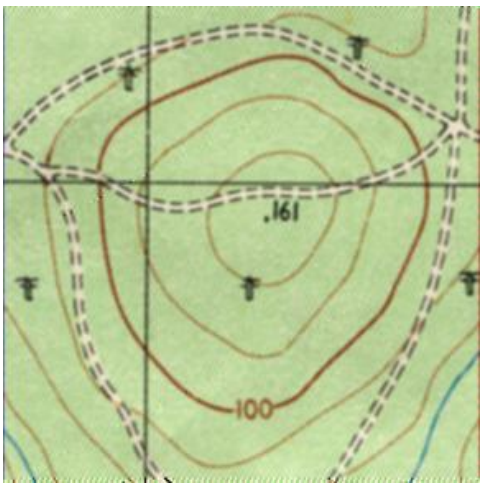
Harita 1, Şekil 3.1 (a)'daki harita Malezya'nın Klang şehrinin kuzey bölümü olup Jabatan Ukur Dan Pemetaan Malaysia (JUPEM) kaynaklıdır [3]. Şekil 3.1 (c)'de görüldüğü gibi katman ayrımında oluşan gürültü çok azdır. Bunun nedeni. Şekil 3.1 (b)'de uygulanmış olan gaussian filtre ile hata oranı katman ayrıştırılmasına başlanmadan en aza indirilmiştir. Yükseklik çizgilerinin değerini belirten rakamın silinmesi manuel olarak yapılmıştır. Yükseklik çizgilerini ifade eden sayısal değerlerin yükseklik çizgileri ile bitişil konumda bulunması OCR ile tanınmasını mümkün kılmamaktadır.

Bağlama işlemleri önerilen üç metot ile yapılmış ve ilgili sonuçlar Tablo 3.1'de

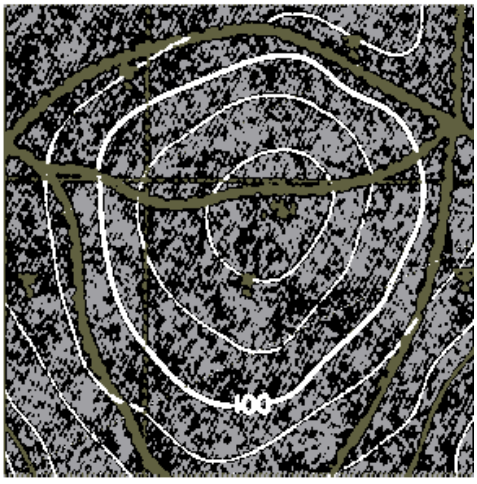
verilmiştir. Tablo 3.1'de görüleceği gibi klasik metot dışındaki önerilen bütün metotlarda doğruluk oranında %100 e varan bir başarı elde edilmiştir. Çalışma zamanı olarak en iyi sonucu I. yeni metot sağlamıştır. Ayrıca klasik metot ve II. yeni metot birlikte kullanılarak II. yeni metodun çalışma zamanda iyileştirme sağlanmıştır. Klasik ve II. yeni metodun birlikte kullanılması önce klasik metot ile bağlanabilecek kopukların bağlanması ve geri kalan kopuklukların ise II. yeni metot ile bağlanması ile gerçekleştirilmiştir.



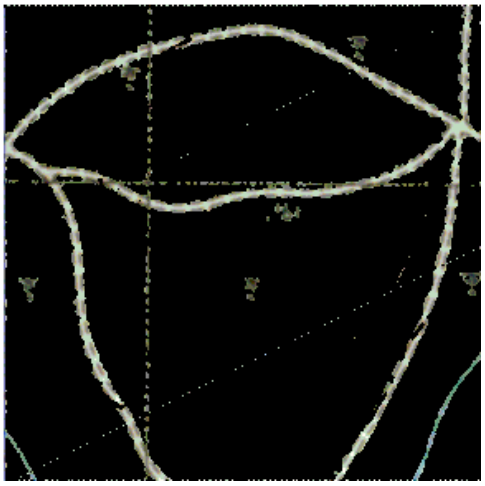
a) Taranmış Harita



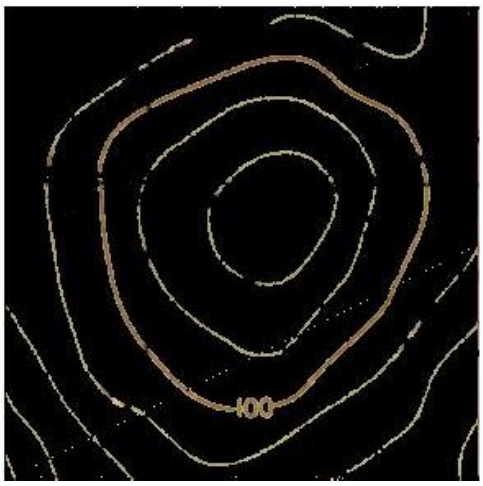
b) Gaussian Filtrelenmiş Harita



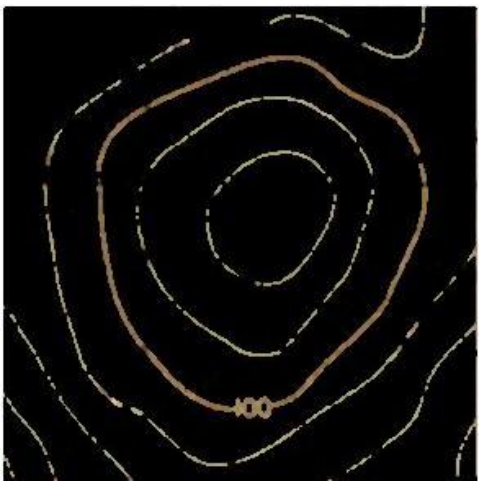
c) Oluşan Katmanlar



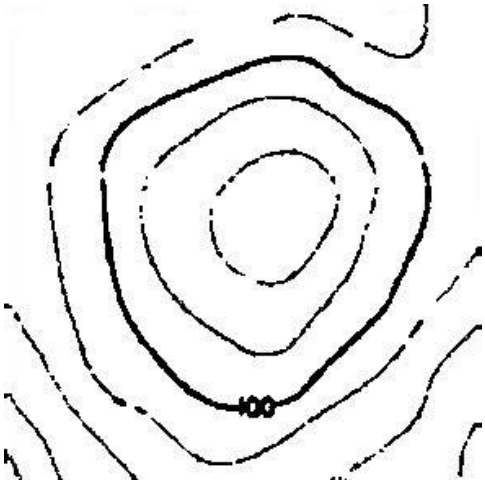
d) Oluşan katmanlardan biri



c) Color clustering

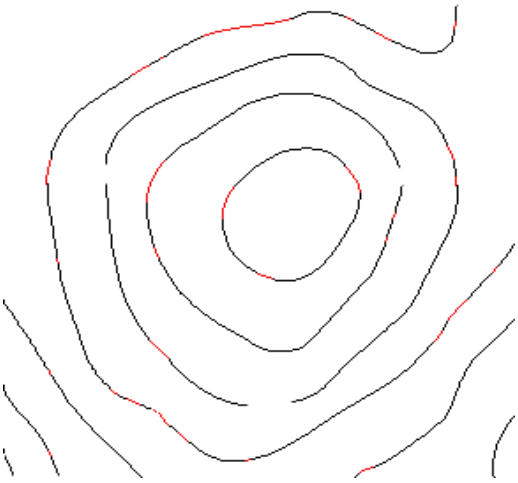


d) Median Filtreleme

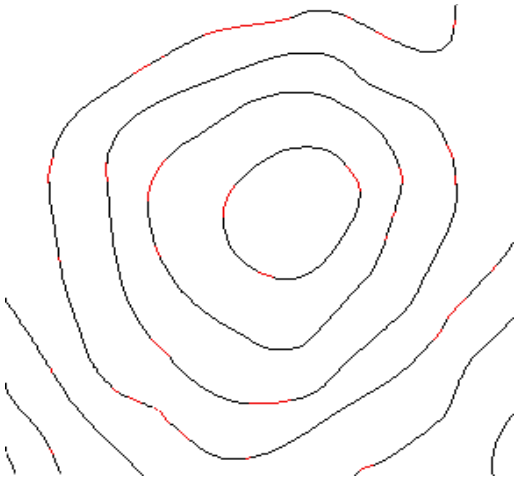


e) İkili Resme Çevirme

f) Morfolojik ve inceltme işlemleri



g) Klasik Metot ile Bağlama



h) I. Yeni Metot ile Bağlama ve II. Yeni Metot ile Bağlama

Şekil 3.1. Harita 1 ait yükseklik çizgilerinin bağlanması.

Harita 1, Şekil 3.1 (a)'daki harita Malezya'nın Klang şehrinin kuzey bölümü olup Jabatan Ukur Dan Pemetaan Malaysia (JUPEM) kaynaklıdır [3]. Şekil 3.1 (c)'de görüldüğü gibi katman ayrımında oluşan gürültü çok azdır. Bunun nedeni. Şekil 3.1 (b)'de uygulanmış olan gaussian filtre ile hata oranı katman ayrıştırılmasına başlanmadan en aza indirilmiştir. Yükseklik çizgilerinin değerini belirten rakamın silinmesi manuel olarak yapılmıştır. Yükseklik çizgilerini ifade eden sayısal değerlerin yükseklik çizgileri ile bitişil konumda bulunması OCR ile tanınmasını mümkün kılmamaktadır.

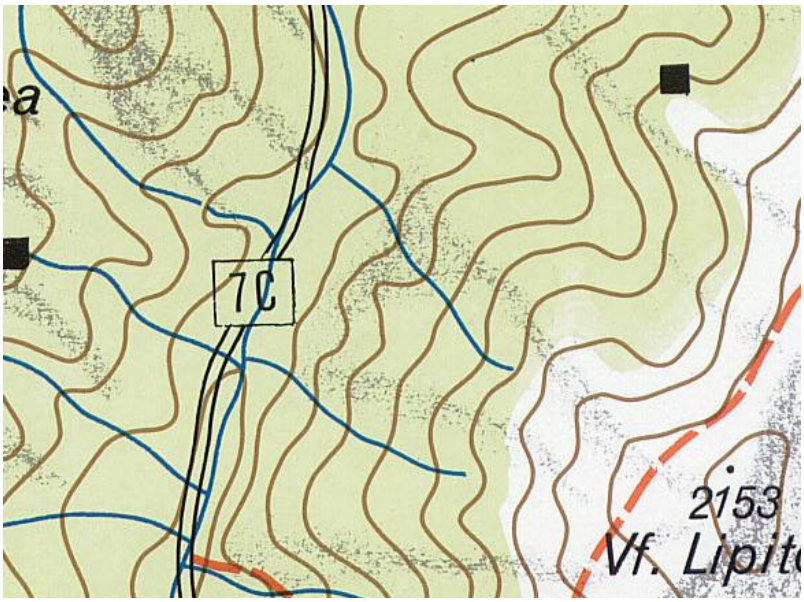
Bağlama işlemleri önerilen üç metot ile yapılmış ve ilgili sonuçlar Tablo 3.1'de verilmiştir. Tablo 3.1'de görüleceği gibi klasik metot dışındaki önerilen bütün metotlarda doğruluk oranında %100 e varan bir başarı elde edilmiştir. Çalışma zamanı olarak en iyi sonucu I. yeni metot sağlamıştır. Ayrıca klasik metot ve II. yeni metot birlikte kullanılarak II. yeni metodun çalışma zamanda iyileştirme sağlanmıştır. Klasik ve II. yeni metodun birlikte kullanılması önce klasik metot ile bağlanabilecek kopukların bağlanması ve geri kalan kopuklukların ise II. yeni metot ile bağlanması ile gerçekleştirilmiştir.

Tablo 3.1. Harita 1'e ait boşlukların doldurulması sonuçları.

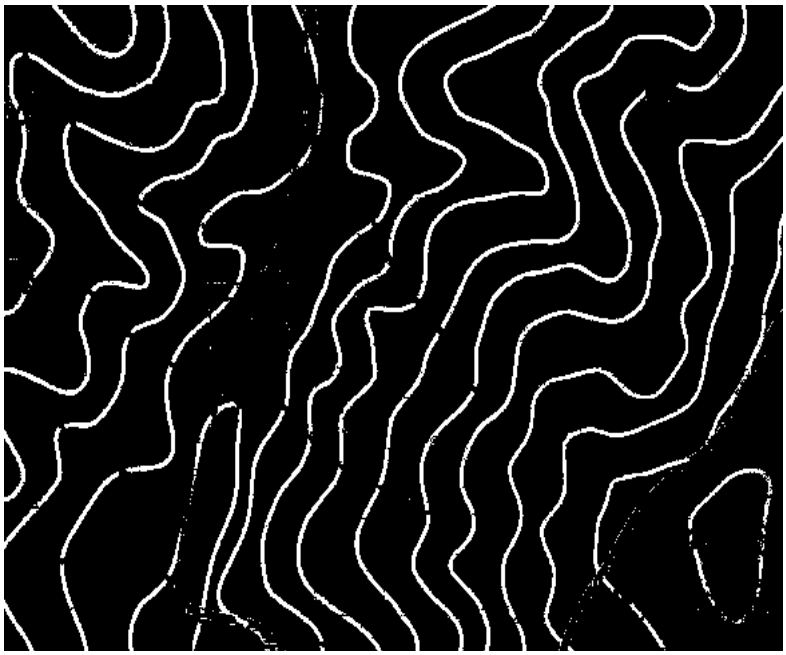
Harita Adı	**Harita Boyutu**	**Method**	**Bağlanma Sayısı**	**Boşluk Sayısı**	**Hatalı Bağlama**	**Çalışma Zamanı (sn.)**	**Doğruluk (%)**
Şekil 3.1	290x290	Klasik Metot	31	3	0	0.474	91.18
		I. Yeni Metot	34	0	0	0.266	100
		II. Yeni Metot	34	0	0	0.879	100
		Klasik + II. Yeni Metot	34	0	0	0.544	100

3.2 UYGULAMA 2

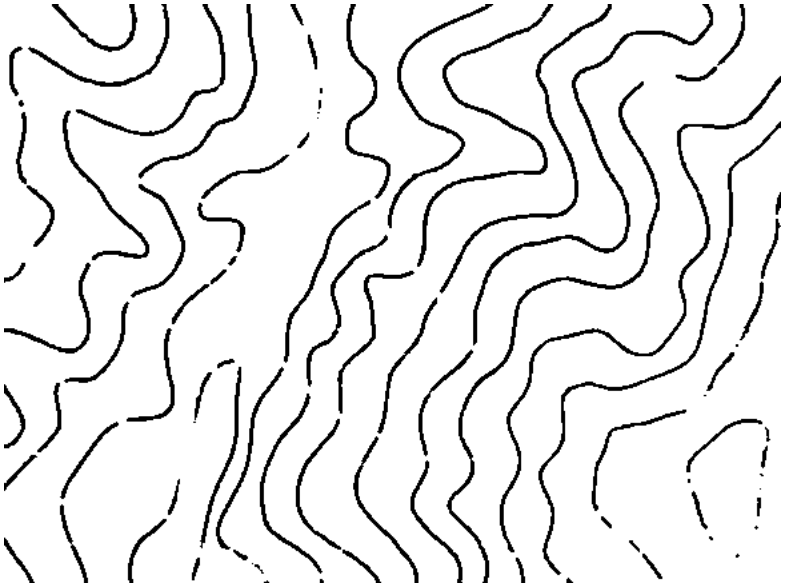
Harita 2, geometrik bir yaklaşım olan Delaunay Triangulization [18] yöntemi ile önerilen yöntemlerin karşılaştırılması amacıyla [19]'dan alınmıştır. Fakat haritanın orijinalinin olmaması ve PDF dosyasından alınan haritanın kalitesinin düşmesi nedeniyle katmanlar ayrıştırılırken problemler ortaya çıkmaktadır. Dolayısıyla katmanların ayrışmış olduğunu [19]'deki gibi kabul edip işlemlere devam edilmiştir.



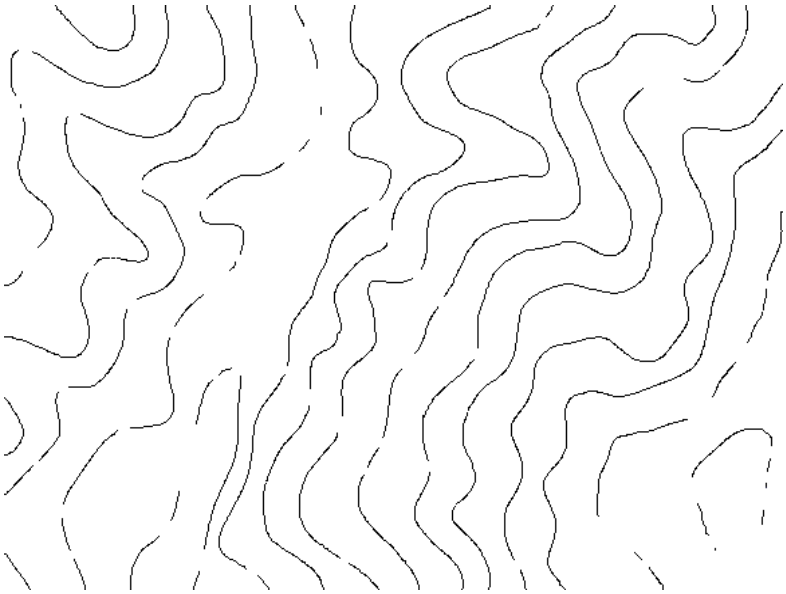
a) Taranmış Harita [19]



b) Katmanları ayrılmış harita [19]



c) Median filtre ve reklerin ters çevrilmesi



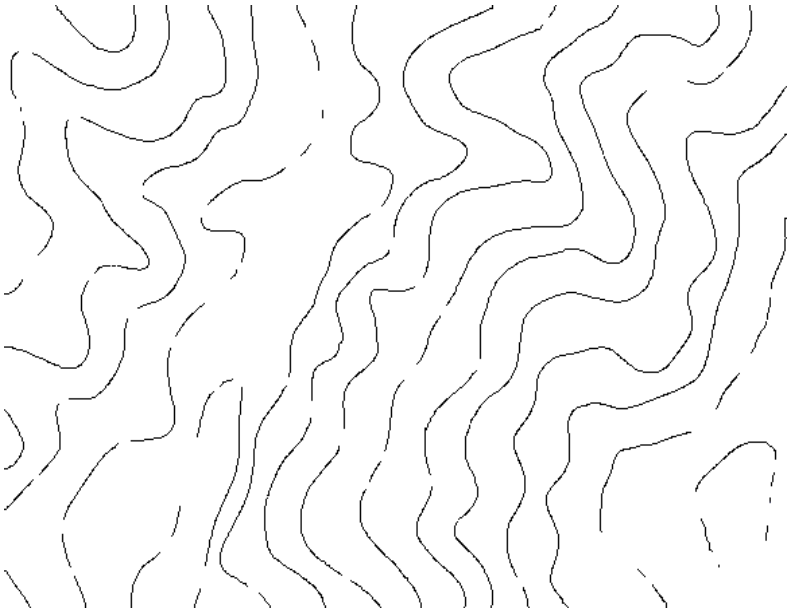
d) Morfolojik işlemler ve inceltme

Şekil 3.2 Harita 2'nin katmanlarına ayrılması ve filtrelenmesi.

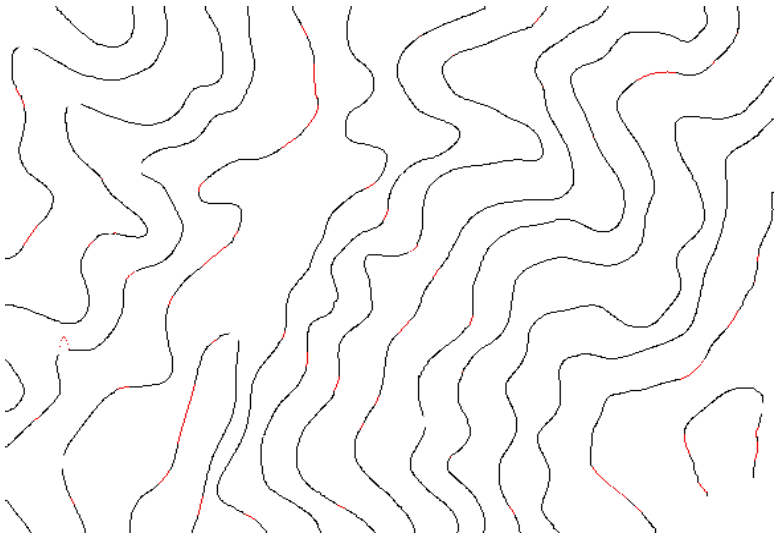
Şekil 3.2 (b)'de görüldüğü gibi yarı otomatik sistemin sağlamış olduğu avantajla segmentasyon işlemi başarılı bir şekilde gerçekleştirilmiştir. Segmentasyon işleminden maksimum verim alınması sayesinde oluşan gürültüler Şekil 3.2 (c)'de görüldüğü gibi başarılı bir şekilde elimine edilmiştir. Akabinde ise çeşitli morfolojik işlemler ve inceltme işlemi ile Şekil 3.2 (d)'deki resim elde edilmiştir.

Şekil 3.3'de ikinci haritanın bağlama sonuçları verilmiştir. Şekil 3.3 (d)'de II. yeni metodun performansını arttırmak amacıyla klasik metot ve I. yeni metot ile birlikte kullanılmıştır.

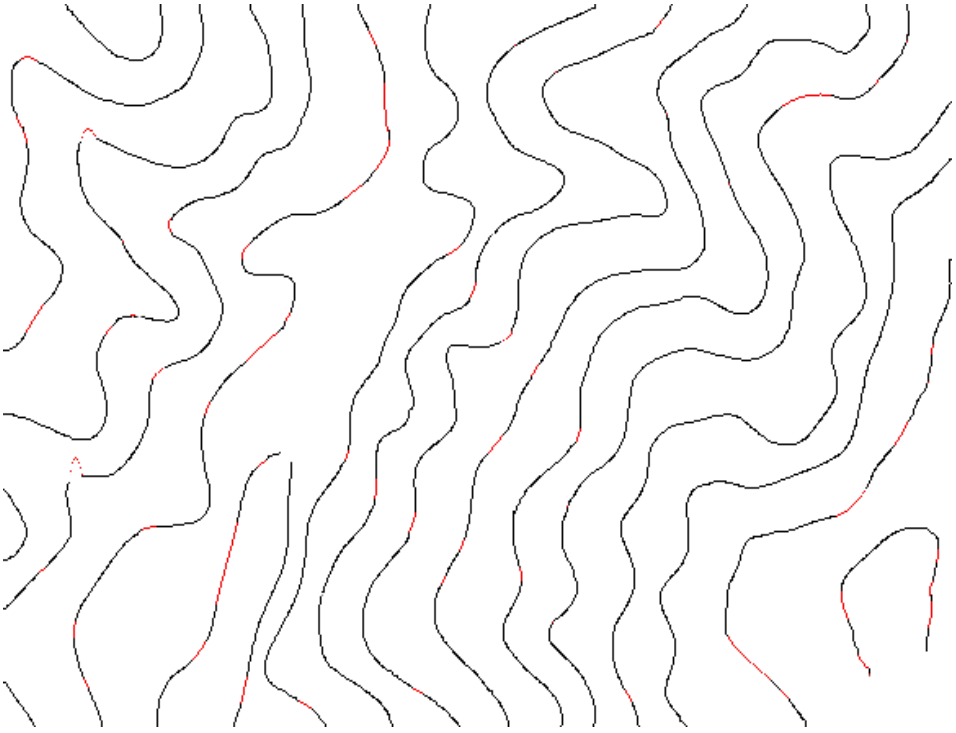
Tablo 3.2'de ikinci haritanın bağlanması ile ilgili sonuçlar verilmiştir. Delaunay Triangulization ve klasik metoda göre %10'lara varan daha doğru bağlama sonuçları elde edilmiştir. Ayrıca II. yeni metodun diğer metotlar ile kullanılması ile de çalışma zamanında %100'lere varan bir iyileşme sağlanmıştır.



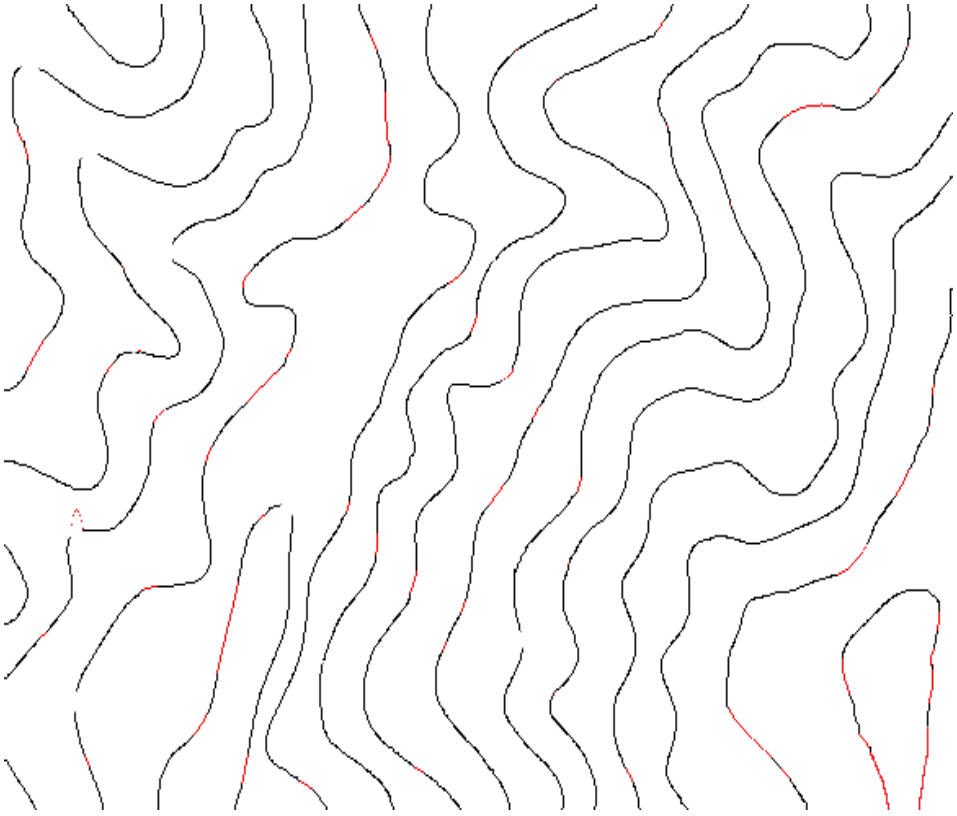
a) Uç noktaların bulunması



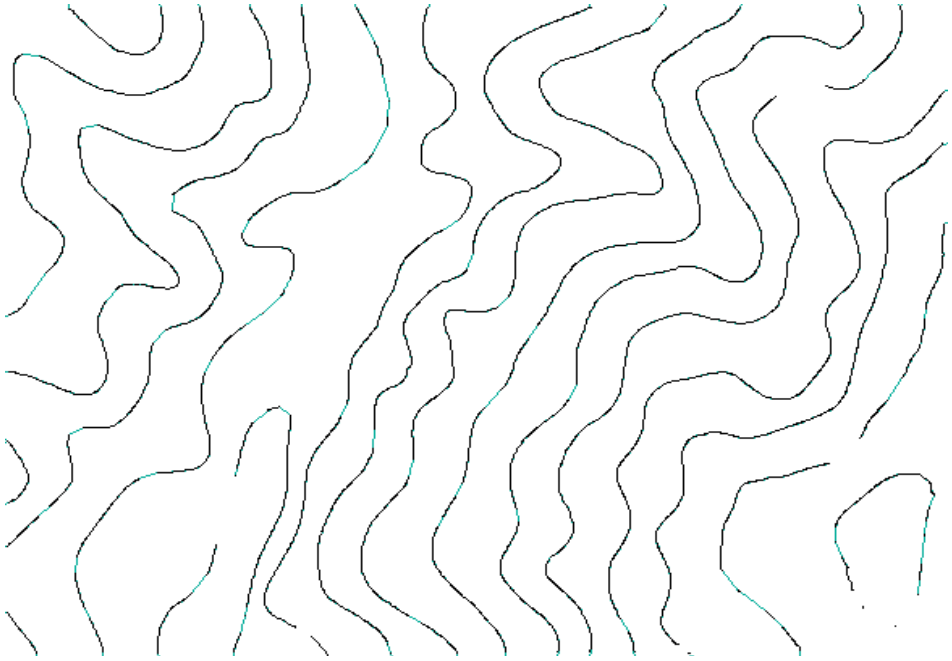
b) Klasik metot ile bağlama



c) I. Yeni Metot ile bağlama



d) II. Yeni Metot; I. Yeni + II. Yeni Metot; II. Yeni + Klasik Metot ile bağlama



e) Delaunay Triangulization ile bağlama

Şekil 3.3 Harita 2'ye ait yükseklik çizgilerinin bağlanması.

Tablo 3.2. Harita 2'ye ait boşlukların doldurulmasının sonuçları.

Toplam Boşluk	Harita Boyutu	Method	Bağlanma Sayısı	Boşluk Sayısı	Düzgün Bağlanmama	Çalışma Zamanı(sn.)	Doğruluk (%)
50	581x431	I. Yeni + II. Yeni Metot	48	1	1	1.676	96
		Klasik+ II. Yeni Metot	48	1	1	1.81	96
		II. Yeni Metot	48	1	1	3.50	96
		I. Yeni Metot	47	2	1	0.986	94
		Klasik Metot	43	6	1	1.85	86
		Delaunay Triangulization	43	7	0	2.18	86

3.3 UYGULAMA 3

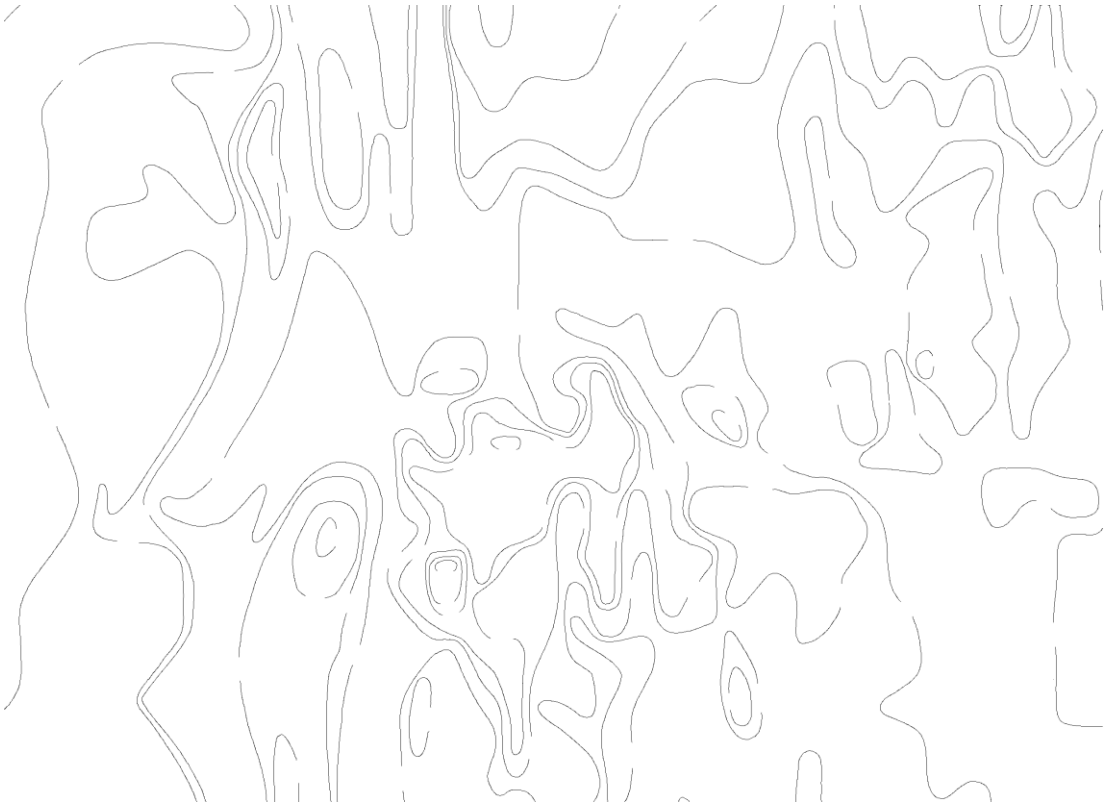
Harita 3, gradyen vektor akışı tabanlı bir yaklaşım olan mükemmel eşleme algoritması ile karşılaştırılmak amacıyla seçilmiştir ve bu algoritmaya ait sonuçlar [11]'den alınmıştır. Harita 3, Minesota Gölü'nün etrafındaki bir alanı içermektedir. [11]'de bu

bölgeye ait kullanılan harita 1278*903 (piksel) boyutundadır. Ancak aynı bölgeye ait bulunan orijinal harita 2568*1932 (piksel) boyutundadır. Şekil 3.4'te bu haritaya ait yapılan aşamalar verilmiştir.

Üçüncü haritanın bağlanması ile ilgili sonuçlar Tablo 3.3'te verilmiştir. Mükemmel eşleme metoduna göre kullanılan harita boyutu 3 katı büyüklüğünde olmasına rağmen klasik ve II. yeni metodun bağlanması zaman olarak daha iyi sonuç vermiştir. Ayrıca kullanılan harita boyutunun üç katı büyüklüğünde olmasına rağmen II. yeni metot 4 saniyelik bir fark ile daha hızlı olarak işlemi gerçekleştirmektedir. Dolayısıyla II. yeni metodun da mükemmel eşleme algoritmasına göre çalışma zamanı olarak daha iyi olduğu ortaya çıkmaktadır. Doğru bağlama sonuçları olarak da klasik metot dışındaki metotlarda %90'ların üzerinde başarı sağlanmış olup II. yeni metot'ta bu oran %100'e çıkmıştır.

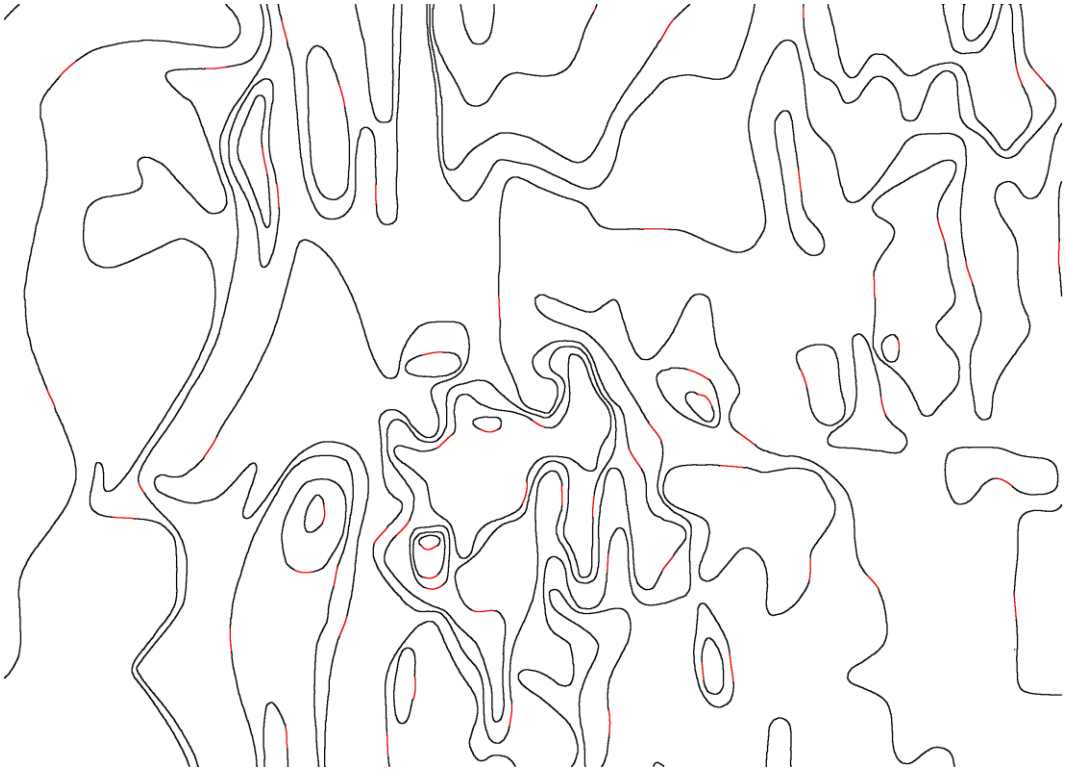
a) Orijinal taranmış harita

b) Katakterlerin Temizlenmesi



c) İnceltme

d) Klasik metot ile bağlama



e) II. Yeni Metot ile bağlanma

Şekil 3.4. Harita 3'e ait yükseklik çizgilerinin bağlanması.

Tablo 3.3. Harita 3'e ait boşlukların doldurulmasının sonuçları.

Toplam Boşluk	**Harita Boyutu**	**Method**	**Bağlanma Sayısı**	**Boşluk Sayısı**	**Yanlış Bağlanma**	**Çalışma Zamanı(sn.)**	**Doğruluk (%)**
60	2568*1932	Klasik+ II. Yeni Metot	60	0	0	21.371	100
		II. Yeni Metot	60	0	0	27.947	100
		I. Yeni + II. Yeni Metot	58	1	1	21.220	96.67
		I. Yeni Metot	55	3	2	24.320	91.67
		Klasik Metot	50	9	1	18.485	83.33
	1278*903	Mükemmel Eşleme	60	0	0	23	100

4. BÖLÜM

TARTIŞMA-SONUÇ ve ÖNERİLER

Bu çalışmada, topografik haritalardan yükseklik çizgilerinin çekilmesi ve oluşan hataların giderilmesi ile ilgili bir uygulama geliştirilmiştir. Filtreleme, segmentasyon ve morfolojik işlemlerinin yükseklik çizgilerinin çıkarılması hususundaki başarı oranları araştırılmış ve başarılı olanları uygulamaya dahil edilmiştir.

Yükseklik çizgilerinin tekrar bağlanması ile ilgili yapılan literatür taramalarında yeterli ve tatmin edici başarı sağlanamadığı görülmüş ve özellikle bu konu üzerindeki çalışmalara ağırlık verilmiştir. Çalışmalar neticesinde iki tane metot geliştirilmiştir [40, 41]. Bu iki metottan biri mevcut klasik algoritmanın gelişmiş versiyonu olup (I. Yeni Metot), diğeri tamamen farklı ve yeni bir metottur (II. Yeni Metot). Klasik yaklaşımdan yola çıkarılarak geliştirilen I. metodun özellikle küçük boyutlu ve kompleksliğin az olduğu haritalarda iyi sonuçlar verdiği görülmüştür. Diğer geliştirilen II. yeni metodun ise hem problemi çözmede, hem de çalışma zamanı olarak en iyi sonuçları verdiği görülmüştür. Böylece verilerin vektörleşme kalitesinin artacağı sonucuna varılmıştır.

Tam otomatik sistemlerin problem çözmede yetersiz kaldığı ve hataları düzeltmeye imkan vermemesi nedeniyle yarı otomatik sistemlerin daha iyi sonuçlar sağladığı kanısına varılmıştır.

İleriki çalışmalarda, yükseklik çizgilerinin bağlanması için yükseklik bilgisi kullanılarak sanal bir doğru ekseni oluşturulup daha sağlam bir algoritma geliştirilebilir. Bağlamak için kullanılan interpolasyon tekniklerinin optimize edilmesi de sağlanabilir. Ayrıca Gradyen vektör akışı tabanlı algoritmaların bu alanda uygulamaları çok azdır. Bu tarz çalışmalara daha fazla ağırlık verilebilir.

Çözülmeyi bekleyen diğer bir problem ise, haritalardan yükseklik bilgilerinin çıkarılması için segmentasyon işlemlerinin belli bir standarda henüz oturtulamamasıdır. Dolayısıyla farklı yaklaşımlı segmentasyon ve filtreleme işlemlerine ihtiyaç vardır.

Sonuç olarak yükseklik çizgilerinin bağlanması ile ilgili yeni metotlar geliştirilmiş olup bu yaklaşımların bilgisayarla görme alanlarında (feature detection and extraction) da kullanılabileceği sonucuna varılmıştır.

KAYNAKÇA

1.Samet, R., Askerbeyli, I. N. A., Varol, C., 2010. An Implementation of Automatic Contour Line Extraction. **An International Journal Applied and Computational Mathematics, 9** (1): 116-127.

2.Açıcı, K., 2008. Yükseklik Bilgilerinin Çekilmesi ve Yeniden Bağlanması. Ankara Üniversitesi, Bilgisayar Mühendisliği, Seminer, Ankara.

3.Ismail, N. I. N., Sheiriff, N. A. M., 2006. A Novel Technique for Contour Reconstruction to DEM. University Technology Malaysia, Phd Thesis.

4.Khotanzad, A., Zink, E., 2003. Contour Line and Geographic Feature Extraction from USGS Color Topographic Paper Maps. **IEEE Trans. On Pattern Analysis and Machine Intelligence, 25** (1)

5.Li, Z., Sui, H., Gong, J., 1999. A System for Automated Generalisation of Contour Lines, pp. *Proceedings of International Carthographic Conference Ottawa*.

6.Visualization, E. E. Raster to Vector Conversion, 2011 -- Vectorization **http://www.parallax69software.com/vectorize.htm** Erişim Tarihi 7 Temmuz.

7.Zhang, T. Y., Suen, C. Y., 1984. A Fast Parallel Algorithm for Thinning Digital Patterns. **Comm. of the ACM, 27** (3): 236-239.

8.Gonzalez, R. C., Woods, R. E., 2002. Digital Image Processing. Prentice Hall.

9.Leberl, F., Olson, D., 1982. Raster scanning for operational digitizing of gaphical data. **Photogrammetric Engineering and Remote Sensing, 48** (4): 615-627.

10.Salvatore, S., Guitton, P., 2004. Contour Lines Recognition from Scanned Topographic Maps. **Journal of WSCG, 1** (3)

11.Pouderoux, J., Spinello, S., 2007. Global Contour Lines Reconstruction in Topographic Maps, pp. 779-783. *Proceedings of the Ninth International Conference on Document Analysis and Recognition*.

12.Arrighi, P., Soille, P., 1999. From scanned topographic maps to digital elevation models, pp. *Proceedings of Geovision'99, International Symposium on Imaging Applications in Geology, University of Liege, Belgium*.

13.Eikvil, L., Aas, K., Koren, H., 1995. Tools for interactive map conversion and vectorization, pp. 927. *Third International Conference on Document Analysis and Recognition: ICDAR 1995*.

14.Chen, Y., Wang, R., Qian, J., 2006. Extracting Contour Lines From Common-Conditioned Topographic Maps. **IEEE Trans. On Geosience and Remote Sensing, 44** (4): 10.

15.San, L. M., Yatim, S. M., Sheriff, N. A. M., Ismail, N. I. N., 2004. Extracting contour lines from scanned topographic maps, pp. *International Conference CGIV*.

16.Chang, A., Kim, K. S., Rhee, S. B., Lee, K. Y., 1997. A Road Extraction From Topographic Maps, pp. 839-842. *IEEE Pasific Rim Conference on Communications, Computers and Signal Processing*.

17.Samet, R., Namazov, M., 2008. Using Fuzzy Sets for Filtering Topographic Map Images. **Int. J. Applied and Computational Mathematics, 7** (2): 242-254.

18.Amenta, N., Bern, M., Eppstein, D., 1998. The crust and the skeleton: Combinatorial curve reconstruction. **Graphical Models and image processing, 60** (2): 125-135.

19.Ghircoias, T., Brad, R., 2011. Contour Lines Extraction and Reconstruction from Topographic Maps. **Ubiquitous Computing and Communication Journal, 6** (2)

20.Edelsbrunner, H., 1998. Shape reconstruction with the Delaunay Complex. **Lecture Notes in Computer Sience, 1380**: 119-132.

21.Dey, T. K., Kumar, P., 1999. A simple provable algorithm for curve reconstruction, pp. 893-894. *In Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms (SODA'99)*.

22.Gold, C., 1999. Crust and anti-crust: A one-step boundaru and skeleton extraction algorithm, pp. 189-196. *In Proceedings of the 15th Annual ACM Symposium on Computational Geometry (SCG'99)*.

23.Dey, T. K., Mehlhorn, K., Ramos, E. A., 1999. Curve Reconstruction: Connecting dots with good reason, pp. 197-206. *In Proceedings of the 15th Annual ACM Symposium on Computational Geometry (SCG'99)*.

24.Giesen, D. P. J., 2000. Curve Reconstruction. Swiss Federal Institute of Technology, Phd Thesis.

25.Althaus, E., Mehlhorn, K., "TSP-based curve reconstruction in polynomial time," *In Proceedings of 11th Sympos. Discrete Algorithms, ACM and SIAM*, 2000, pp. 125-135.

26.Gabow, H. N., 1974. Implementation of algorithms for maximum matching on nonbipartite graphs. Stanford University, PhD Thesis.

27.Wu, R.-Q., Cheng, X.-R., Yang, C.-J., 2009. Extracting contour lines from topographic maps based on cartography and graphics knowledge. **JCS&T, 9** (2)

28.Xin, D., Zhou, X., Zheng, H., 2006. Contour Line Extraction from Paper Based Topographic Maps. **Journal of Information and Computing Sience, 1** (5): 275-283.

29.Xu, C. Y., Prince, J. L., 1998. Generalized Gradient Vector Flow External Forces for Active Contours. **Signal Processing, 71**: 131-139.

30.Xu, C. Y., Prince, J. L., 1998. Snakes, Shapes and Gradient Vector Flow. **IEEE transactions on Image Processing, 3**: 359-369.

31.Heckbert, P., 1982. Color Image Quantization for Frame Buffer Display. **Computer Graphics, 16** (3): 297-303.

32.Ağaoğlu, S. 2009. Median Cut ve Huffman Uygulayarak Görüntü Sıkıştırma **http://www.club3e.org/kb.php?mode=article&k=20&page_num=2&start=0** Erişim Tarihi 23 Aralık 2011.

33.Jie, X., Peng-Fei, S., 2003. Natural Color Image Segmentation, pp. 973-976. *International Conference on Image Processing*.

34.Louverdis, G., Andreadis, I., 2003. Morphological Filtering Using a Fuzzy Model and Its Application to Colour Image Processing. **Springer-Verlag London,**

35.Kirillov, A. AForge.Net Framework, 2011 **http://www.aforgenet.com/framework/** Erişim Tarihi 23 Haziran.

36.Kennedy, J., Eberhart, R. C., 1995. Particle swarm optimization, pp. 1942-1948. *IEEE International Conference on Neural Networks*.

37.Karaboga, D., Ozturk, C., 2011. A novel clustering approach: Artificial Bee Colony (ABC) algorithm. **Applied Soft Computing, 11** (1): 652-657.

38.Burden, R. L., Faires, J. D., Reynolds, A. C., 1997. Numerical Analysis 6th ed, MA: Brooks/Cole.

39.Heath, M. T., 1997. Scientific Computing An Introduction Survey. The McGraw-Hill Companies.

40.Hancer, E., Samet, R., "Advanced contour reconnection in scanned topographic maps," *Application of Information and Communication Technologies (AICT), 2011 5th International Conference on*, 2011, pp. 1-5.

41.Samet, R., Hancer, E., 2012. A New Approach To The Reconstruction Of Contour Lines Extracted From Topographic Maps. **Journal of Visual Communication and Image Representation, 23** (4): 642-647

EKLER

Bu bölümde uygulamada kullanılan önemli kod parçacılarına yer verilmiştir

Yön kontrolü için kullanılan kod parçası:

```
private bool directioncontrol(int direction1, int direction2)
        {
            if ((direction1 == 1 && direction2 == 5) || (direction1 == 5 &&
direction2 == 1)) return true;
            else if ((direction1 == 0 && direction2 == 4) || (direction1 ==
4 && direction2 == 0)) return true;
            else if ((direction1 == 7 && direction2 == 3) || (direction1 ==
3 && direction2 == 7)) return true;
            else if ((direction1 == 6 && direction2 == 2) || (direction1 ==
2 && direction2 == 6)) return true;
            else return false;
        }

private bool directioncontrolforparabolic(int direction1, int direction2)
        {
            if ((direction1 == 1 && direction2 == 3) || (direction1 == 3 &&
direction2 == 1)) return true;
            else if ((direction1 == 3 && direction2 == 5) || (direction1 ==
5 && direction2 == 3)) return true;
            else if ((direction1 == 5 && direction2 == 7) || (direction1 ==
7 && direction2 == 5)) return true;
            else if ((direction1 == 1 && direction2 == 7) || (direction1 ==
7 && direction2 == 1)) return true;
            else return false;
        }
```

Thinning ve Pruning işlemleri için:

```
private void thinning()
        {
            hazirla_();
            int komsusiyah;
            int index=0;
            for (int i = 1; i < BitmapFilter.height-1; i++)
            {
                for (int j = 1; j < BitmapFilter.width-1; j++)
                {
                    if (greyPixelArray[i,j]==0)
                    {
                        komsusiyah = siyahpixelsayisi(i, j);
                        //first iteration
                        if (connectivity(i, j) == 1 && (2 <= komsusiyah &&
komsusiyah <= 6) && firstiteration3_4(i, j) == true)
                        {
                            marked[index].i = i;
                            marked[index].j = j;
```

```
                            index++;
                        }

                    }
                }
            }

            for (int i = 0; i < index; i++)
            {
                greyPixelArray[marked[i].i, marked[i].j] = 255;
            }
            index = 0;

            for (int i = 1; i < BitmapFilter.height - 1; i++)
            {
                for (int j = 1; j < BitmapFilter.width - 1; j++)
                {
                    if (greyPixelArray[i, j] == 0)
                    {
                        komsusiyah = siyahpixelsayisi(i, j);

                        //second iteration
                        if (connectivity(i, j) == 1 && (2 <= komsusiyah &&
komsusiyah <= 6) && seconditeration3_4(i, j) == true)
                        {
                            marked[index].i = i;
                            marked[index].j = j;
                            index++;
                        }
                    }
                }
            }

            for (int i = 0; i < index; i++)
            {
                greyPixelArray[marked[i].i, marked[i].j] = 255;
            }

        }

 private void pruning(int n_iteration)
 {
   MarkedData[,] eroted=new MarkedData[n_iteration,300];
   //preparing eroted data
   for (int i = 0; i < n_iteration; i++)
   {
        for (int j = 0; j < 300; j++)
        {
             eroted[i, j] = new MarkedData();
        }
   }


   int iteration = 0;
   int sayac=0;
   int[] anasayac = new int[n_iteration];
   while (iteration < n_iteration)
   {
```

```
     for (int i = 1; i < BitmapFilter.height - 1; i++)
     {
        for (int j = 1; j < BitmapFilter.width - 1; j++)
        {
            if (i - 1 <= 0 || i + 1 >= BitmapFilter.height || j - 1 <= 0 ||
j + 1 > BitmapFilter.width)
            break;
                        if (greyPixelArray[i - 1, j - 1] == 0 &&
greyPixelArray[i, j - 1] == 255 && greyPixelArray[i + 1, j - 1] == 255 &&
greyPixelArray[i - 1, j] == 255 && greyPixelArray[i, j] == 0 &&
greyPixelArray[i + 1, j] == 255 && greyPixelArray[i - 1, j + 1] == 255 &&
greyPixelArray[i, j + 1] == 255 && greyPixelArray[i + 1, j + 1] == 255)
                        {

                            eroted[iteration, sayac].i = i;
                            eroted[iteration, sayac].j = j;
                            sayac++;

                        }
                        else if (greyPixelArray[i - 1, j - 1] == 255 &&
greyPixelArray[i, j - 1] == 0 && greyPixelArray[i + 1, j - 1] == 255 &&
greyPixelArray[i - 1, j] == 255 && greyPixelArray[i, j] == 0 &&
greyPixelArray[i + 1, j] == 255 && greyPixelArray[i - 1, j + 1] == 255 &&
greyPixelArray[i, j + 1] == 255 && greyPixelArray[i + 1, j + 1] == 255)
                        {

                            eroted[iteration, sayac].i = i;
                            eroted[iteration, sayac].j = j;
                            sayac++;

                        }
                        else if (greyPixelArray[i - 1, j - 1] == 255 &&
greyPixelArray[i, j - 1] == 255 && greyPixelArray[i + 1, j - 1] == 0 &&
greyPixelArray[i - 1, j] == 255 && greyPixelArray[i, j] == 0 &&
greyPixelArray[i + 1, j] == 255 && greyPixelArray[i - 1, j + 1] == 255 &&
greyPixelArray[i, j + 1] == 255 && greyPixelArray[i + 1, j + 1] == 255)
                        {

                            eroted[iteration, sayac].i = i;
                            eroted[iteration, sayac].j = j;
                            sayac++;

                        }
                        else if (greyPixelArray[i - 1, j - 1] == 255 &&
greyPixelArray[i, j - 1] == 255 && greyPixelArray[i + 1, j - 1] == 255 &&
greyPixelArray[i - 1, j] == 0 && greyPixelArray[i, j] == 0 &&
greyPixelArray[i + 1, j] == 255 && greyPixelArray[i - 1, j + 1] == 255 &&
greyPixelArray[i, j + 1] == 255 && greyPixelArray[i + 1, j + 1] == 255)
                        {

                            eroted[iteration, sayac].i = i;
                            eroted[iteration, sayac].j = j;
                            sayac++;

                        }
                        else if (greyPixelArray[i - 1, j - 1] == 255 &&
greyPixelArray[i, j - 1] == 255 && greyPixelArray[i + 1, j - 1] == 255 &&
greyPixelArray[i - 1, j] == 255 && greyPixelArray[i, j] == 0 &&
```

```
greyPixelArray[i + 1, j] == 0 && greyPixelArray[i - 1, j + 1] == 255 &&
greyPixelArray[i, j + 1] == 255 && greyPixelArray[i + 1, j + 1] == 255)
                        {

                            eroted[iteration, sayac].i = i;
                            eroted[iteration, sayac].j = j;
                            sayac++;

                        }
                        else if (greyPixelArray[i - 1, j - 1] == 255 &&
greyPixelArray[i, j - 1] == 255 && greyPixelArray[i + 1, j - 1] == 255 &&
greyPixelArray[i - 1, j] == 255 && greyPixelArray[i, j] == 0 &&
greyPixelArray[i + 1, j] == 255 && greyPixelArray[i - 1, j + 1] == 0 &&
greyPixelArray[i, j + 1] == 255 && greyPixelArray[i + 1, j + 1] == 255)
                        {

                            eroted[iteration, sayac].i = i;
                            eroted[iteration, sayac].j = j;
                            sayac++;

                        }
                        else if (greyPixelArray[i - 1, j - 1] == 255 &&
greyPixelArray[i, j - 1] == 255 && greyPixelArray[i + 1, j - 1] == 255 &&
greyPixelArray[i - 1, j] == 255 && greyPixelArray[i, j] == 0 &&
greyPixelArray[i + 1, j] == 255 && greyPixelArray[i - 1, j + 1] == 255 &&
greyPixelArray[i, j + 1] == 0 && greyPixelArray[i + 1, j + 1] == 255)
                        {

                            eroted[iteration, sayac].i = i;
                            eroted[iteration, sayac].j = j;
                            sayac++;

                        }
                        else if (greyPixelArray[i - 1, j - 1] == 255 &&
greyPixelArray[i, j - 1] == 255 && greyPixelArray[i + 1, j - 1] == 255 &&
greyPixelArray[i - 1, j] == 255 && greyPixelArray[i, j] == 0 &&
greyPixelArray[i + 1, j] == 255 && greyPixelArray[i - 1, j + 1] == 255 &&
greyPixelArray[i, j + 1] == 255 && greyPixelArray[i + 1, j + 1] == 0)
                        {

                            eroted[iteration, sayac].i = i;
                            eroted[iteration, sayac].j = j;
                            sayac++;

                        }
                    }
                }
 for (int ind = 0; ind < sayac; ind++)
 {
 greyPixelArray[eroted[iteration, ind].i, eroted[iteration, ind].j] = 255;
 }
      anasayac[iteration] = sayac;
      iteration++;
      sayac = 0;
 }

//for dilation
iteration--;
while (iteration >= 0)
```

```
{
     for (int i = 1; i < BitmapFilter.height-1; i++)
     {
        for (int j = 1; j < BitmapFilter.width-1; j++)
        {
                        if (greyPixelArray[i - 1, j - 1] == 0 &&
greyPixelArray[i, j - 1] == 255 && greyPixelArray[i + 1, j - 1] == 255 &&
greyPixelArray[i - 1, j] == 255 && greyPixelArray[i, j] == 0 &&
greyPixelArray[i + 1, j] == 255 && greyPixelArray[i - 1, j + 1] == 255 &&
greyPixelArray[i, j + 1] == 255 && greyPixelArray[i + 1, j + 1] == 255)
                        {


                       for (int ind = 0; ind < anasayac[iteration]; ind++)
                                {
                                    if ((i - 1 <= eroted[iteration, ind].i
&& eroted[iteration, ind].i <= i + 1) && (j - 1 <= eroted[iteration, ind].j
&& eroted[iteration, ind].j <= j + 1))
                                        greyPixelArray[eroted[iteration,
ind].i, eroted[iteration, ind].j] = 0;
                                }


                        }
                        else if (greyPixelArray[i - 1, j - 1] == 255 &&
greyPixelArray[i, j - 1] == 0 && greyPixelArray[i + 1, j - 1] == 255 &&
greyPixelArray[i - 1, j] == 255 && greyPixelArray[i, j] == 0 &&
greyPixelArray[i + 1, j] == 255 && greyPixelArray[i - 1, j + 1] == 255 &&
greyPixelArray[i, j + 1] == 255 && greyPixelArray[i + 1, j + 1] == 255)
                        {

                       for (int ind = 0; ind < anasayac[iteration]; ind++)
                            {
                                if ((i - 1 <= eroted[iteration, ind].i &&
eroted[iteration, ind].i <= i + 1) && (j - 1 <= eroted[iteration, ind].j &&
eroted[iteration, ind].j <= j + 1))
                                    greyPixelArray[eroted[iteration,
ind].i, eroted[iteration, ind].j] = 0;
                            }

                        }
                        else if (greyPixelArray[i - 1, j - 1] == 255 &&
greyPixelArray[i, j - 1] == 255 && greyPixelArray[i + 1, j - 1] == 0 &&
greyPixelArray[i - 1, j] == 255 && greyPixelArray[i, j] == 0 &&
greyPixelArray[i + 1, j] == 255 && greyPixelArray[i - 1, j + 1] == 255 &&
greyPixelArray[i, j + 1] == 255 && greyPixelArray[i + 1, j + 1] == 255)
                        {

                       for (int ind = 0; ind < anasayac[iteration]; ind++)
                            {
                                if ((i - 1 <= eroted[iteration, ind].i &&
eroted[iteration, ind].i <= i + 1) && (j - 1 <= eroted[iteration, ind].j &&
eroted[iteration, ind].j <= j + 1))
                                    greyPixelArray[eroted[iteration,
ind].i, eroted[iteration, ind].j] = 0;
                            }

                        }
```

```
                        else if (greyPixelArray[i - 1, j - 1] == 255 &&
greyPixelArray[i, j - 1] == 255 && greyPixelArray[i + 1, j - 1] == 255 &&
greyPixelArray[i - 1, j] == 0 && greyPixelArray[i, j] == 0 &&
greyPixelArray[i + 1, j] == 255 && greyPixelArray[i - 1, j + 1] == 255 &&
greyPixelArray[i, j + 1] == 255 && greyPixelArray[i + 1, j + 1] == 255)
                        {

                         for (int ind = 0; ind < anasayac[iteration]; ind++)
                             {
                                 if ((i - 1 <= eroted[iteration, ind].i &&
eroted[iteration, ind].i <= i + 1) && (j - 1 <= eroted[iteration, ind].j &&
eroted[iteration, ind].j <= j + 1))
                                     greyPixelArray[eroted[iteration,
ind].i, eroted[iteration, ind].j] = 0;
                             }

                         }
                        else if (greyPixelArray[i - 1, j - 1] == 255 &&
greyPixelArray[i, j - 1] == 255 && greyPixelArray[i + 1, j - 1] == 255 &&
greyPixelArray[i - 1, j] == 255 && greyPixelArray[i, j] == 0 &&
greyPixelArray[i + 1, j] == 0 && greyPixelArray[i - 1, j + 1] == 255 &&
greyPixelArray[i, j + 1] == 255 && greyPixelArray[i + 1, j + 1] == 255)
                        {

                        for (int ind = 0; ind < anasayac[iteration]; ind++)
                             {
                                 if ((i - 1 <= eroted[iteration, ind].i &&
eroted[iteration, ind].i <= i + 1) && (j - 1 <= eroted[iteration, ind].j &&
eroted[iteration, ind].j <= j + 1))
                                     greyPixelArray[eroted[iteration,
ind].i, eroted[iteration, ind].j] = 0;
                             }

                         }
                        else if (greyPixelArray[i - 1, j - 1] == 255 &&
greyPixelArray[i, j - 1] == 255 && greyPixelArray[i + 1, j - 1] == 255 &&
greyPixelArray[i - 1, j] == 255 && greyPixelArray[i, j] == 0 &&
greyPixelArray[i + 1, j] == 255 && greyPixelArray[i - 1, j + 1] == 0 &&
greyPixelArray[i, j + 1] == 255 && greyPixelArray[i + 1, j + 1] == 255)
                        {

                        for (int ind = 0; ind < anasayac[iteration]; ind++)
                             {
                                 if ((i - 1 <= eroted[iteration, ind].i &&
eroted[iteration, ind].i <= i + 1) && (j - 1 <= eroted[iteration, ind].j &&
eroted[iteration, ind].j <= j + 1))
                                     greyPixelArray[eroted[iteration,
ind].i, eroted[iteration, ind].j] = 0;
                             }

                         }
                        else if (greyPixelArray[i - 1, j - 1] == 255 &&
greyPixelArray[i, j - 1] == 255 && greyPixelArray[i + 1, j - 1] == 255 &&
greyPixelArray[i - 1, j] == 255 && greyPixelArray[i, j] == 0 &&
greyPixelArray[i + 1, j] == 255 && greyPixelArray[i - 1, j + 1] == 255 &&
greyPixelArray[i, j + 1] == 0 && greyPixelArray[i + 1, j + 1] == 255)
                        {

                       for (int ind = 0; ind < anasayac[iteration]; ind++)
```

```
                            {
                                if ((i - 1 <= eroted[iteration, ind].i &&
eroted[iteration, ind].i <= i + 1) && (j - 1 <= eroted[iteration, ind].j &&
eroted[iteration, ind].j <= j + 1))
                                    greyPixelArray[eroted[iteration,
ind].i, eroted[iteration, ind].j] = 0;
                            }

                        }
                        else if (greyPixelArray[i - 1, j - 1] == 255 &&
greyPixelArray[i, j - 1] == 255 && greyPixelArray[i + 1, j - 1] == 255 &&
greyPixelArray[i - 1, j] == 255 && greyPixelArray[i, j] == 0 &&
greyPixelArray[i + 1, j] == 255 && greyPixelArray[i - 1, j + 1] == 255 &&
greyPixelArray[i, j + 1] == 255 && greyPixelArray[i + 1, j + 1] == 0)
                        {

                    for (int ind = 0; ind < anasayac[iteration]; ind++)
                            {
                                if ((i - 1 <= eroted[iteration, ind].i &&
eroted[iteration, ind].i <= i + 1) && (j - 1 <= eroted[iteration, ind].j &&
eroted[iteration, ind].j <= j + 1))
                                    greyPixelArray[eroted[iteration,
ind].i, eroted[iteration, ind].j] = 0;
                            }

                        }

                    }
                }
                iteration--;
            }

        }
```

Newton Interpolasyon ile bağlamak için kullanılan kod parçası:

```
private void NewtonInterpolation(int[,] data)
{

  double t1, t2, t3, t4, f_t1, f_t2, f_t3, f_t4, f_t1_t2, f_t2_t3, f_t3_t4,
f_t1_t2_t3, f_t2_t3_t4, f_t1_t2_t3_t4;

            if (Math.Abs(data[2, 1] - data[1, 1]) > Math.Abs(data[1, 0] -
data[2, 0]))
            {
                for (int i = 0; i < 2; i++)
                {
                    for (int j = 0; j < 3; j++)
                    {
                        if (data[j, i] == data[j + 1, i])
                            data[j, i] = data[j, i] - 1;
                    }

                }
                t1 = data[0, 1]; t2 = data[1, 1]; t3 = data[2, 1]; t4 =
data[3, 1];
                f_t1 = data[0, 0]; f_t2 = data[1, 0]; f_t3 = data[2, 0];
f_t4 = data[3, 0];
```

```
                f_t1_t2 = (f_t2 - f_t1) / (t2 - t1);
                f_t2_t3 = (f_t3 - f_t2) / (t3 - t2);
                f_t3_t4 = (f_t4 - f_t3) / (t4 - t3);
                f_t1_t2_t3 = (f_t2_t3 - f_t1_t2) / (t3 - t1);
                f_t2_t3_t4 = (f_t3_t4 - f_t2_t3) / (t4 - t2);
                f_t1_t2_t3_t4 = (f_t2_t3_t4 - f_t1_t2_t3) / (t4 - t1);

                int y;
                for (int t = Convert.ToInt32(t2); t <= Convert.ToInt32(t3);
t++)
                {
                    try
                    {
                        y = Convert.ToInt32(f_t1 + f_t1_t2 * (t - t1) +
f_t1_t2_t3 * (t - t1) * (t - t2) + f_t1_t2_t3_t4 * (t - t1) * (t - t2) * (t
- t3));
                        if (y < BitmapFilter.height && y>=1)
                        greyPixelArray[y, t] = 0;

                    }
                    catch (Exception)
                    {

                        throw;
                    }

                }

            }
            else
            {
                data = sirala_y(data);
                /*int test;
                if (data[0, 0] == 73)
                    test=1;*/
                int[,] yedekdata = data;
                for (int i = 0; i < 2; i++)
                {
                    for (int j = 0; j < 3; j++)
                    {
                        if (yedekdata[j, i] == yedekdata[j + 1, i])
                            data[j, i] = data[j, i] - 1;
                    }

                }

                t1 = data[0, 0]; t2 = data[1, 0]; t3 = data[2, 0]; t4 =
data[3, 0];
                f_t1 = data[0, 1]; f_t2 = data[1, 1]; f_t3 = data[2, 1];
f_t4 = data[3, 1];
                f_t1_t2 = (f_t2 - f_t1) / (t2 - t1);

                f_t2_t3 = (f_t3 - f_t2) / (t3 - t2);
                f_t3_t4 = (f_t4 - f_t3) / (t4 - t3);
                f_t1_t2_t3 = (f_t2_t3 - f_t1_t2) / (t3 - t1);
                f_t2_t3_t4 = (f_t3_t4 - f_t2_t3) / (t4 - t2);
                f_t1_t2_t3_t4 = (f_t2_t3_t4 - f_t1_t2_t3) / (t4 - t1);

                int y;
```

```
                for (int t = Convert.ToInt32(t2); t <= Convert.ToInt32(t3);
t++)
                {
                    try
                    {
                        y = Convert.ToInt32(f_t1 + f_t1_t2 * (t - t1) +
f_t1_t2_t3 * (t - t1) * (t - t2) + f_t1_t2_t3_t4 * (t - t1) * (t - t2) * (t
- t3));
                        if (y < BitmapFilter.width - 1 && y >= 1)
                        {
                            greyPixelArray[t, y] = 0;
                        }

                    }
                    catch (Exception)
                    {

                        continue;
                    }

                }


            }
```

Açı hesabı ve kontrolü için kullanılan kod parçası:

```
private double anglehesapla(int[,] a, int[,] b)
{
        double h = distance(a[0, 1], a[0, 0], b[0, 1], b[0, 0]);
        double angle = Math.Asin(Math.Abs(a[0, 0] - b[0, 0]) / h);
        return angle;
}

private bool anglecontrol(double angle1, double angle2)
{
   if (Math.Abs(angle1 - angle2) < 90)
   {
        return true;
   }

   else
        return false;
}
```

End point ve bağlama noktalarını renklendirmek için kullanılan kod parçası:

```
public static bool EndPointColor(Bitmap b)
{

    // GDI+ still lies to us - the return format is BGR, NOT RGB.
    BitmapData bmData = b.LockBits(new Rectangle(0, 0, b.Width, b.Height),
ImageLockMode.ReadWrite, PixelFormat.Format24bppRgb);
    int stride = bmData.Stride;
    System.IntPtr Scan0 = bmData.Scan0;
    unsafe
    {
```

```
                byte* p = (byte*)(void*)Scan0;

                int nOffset = stride - b.Width * 3;
                int nPixel;

                for (int y = 0; y < b.Height; ++y)
                {
                    for (int x = 0; x < b.Width; ++x)
                    {
                        if (p[0] == 127 && p[1] == 127 && p[2] == 127)
                        {
                            p[0] = 0; p[1] = 0; p[2] = 238;
                        }
                        p += 3;
                    }
                    p += nOffset;
                }
            }

            b.UnlockBits(bmData);

            return true;

        }
```

Klasik metot için kullanılan kod parçası:

```
private void searchwindow(int height, int width)
{
  hazirla();
  int dizin_y_sol; int dizin_x_sol;
  int dizin_y_sağ; int dizin_x_sağ;
  for (int i = 0; i < BitmapFilter.height - 1; i++)
  {
      for (int j = 0; j < BitmapFilter.width - 1; j++)
      {
           if (greyPixelArray[i, j] == 128)
           {
               if (i - height < 0)
               {
                   dizin_y_sol = 0;
               }
               else
               {
                   dizin_y_sol = i - height;
               }
               if (i + height > BitmapFilter.height - 1)
               {
                   dizin_y_sağ = BitmapFilter.height - 1;
               }
               else
               {
                   dizin_y_sağ = i + height;
               }
               if (j - width < 0)
               {
                   dizin_x_sol = 0;
               }
               else
```

```
                {
                    dizin_x_sol = j - width;
                }
                if (j + width > BitmapFilter.width - 1)
                {
                    dizin_x_sağ = BitmapFilter.width - 1;
                }
                else
                {
                    dizin_x_sağ = j + width;
                }

                for (int ii = dizin_y_sol; ii < dizin_y_sağ; ii++)
                 //search window içi
                {
                 for (int jj = dizin_x_sol; jj < dizin_x_sağ; jj++)
                 {
                if (greyPixelArray[ii, jj] == 128 && (ii != i || jj != j))
                 {

                   double uzaklık = distance(i, j, ii, jj);
                   ends[say].i = ii;
                   ends[say].j = jj;
                   ends[say].distance = uzaklık;
                   say++;
                  }
                  }
                 }
                       /*int test;

        double min = ends[0].distance;
        int index = 0;
        //en yakın uzaklıktaki end point bulma

        for (int j_ = 0; j_ < say; j_++)
        {
            if (min > ends[j_].distance)
            {
                 min = ends[j_].distance;
                 index = j_;
            }

        }


       if (ends[index].i != 0 && ends[index].j != 0)
//search window içindeki en yakın point
       {

          int[,] c = new int[1, 2];
          int[,] d = new int[1, 2];
          c = backwardtenpixel(i, j);
          d = backwardtenpixel(ends[index].i, ends[index].j);
          int[,] endpoint2 = new int[1, 2];
          endpoint2[0, 0] = ends[index].i;
          endpoint2[0, 1] = ends[index].j;
          int[,] anaendpoint = new int[1, 2];
          anaendpoint[0, 0] = i;
          anaendpoint[0, 1] = j;
```

```
            int yon1 = directions(i, j, c);
            int yon2 = directions(i, j, d);
            if (yon1 == 2 || yon1 == 4 || yon1 == 6)
                   yon1 = yon1 - 1;
            else if (yon1 == 0)
                   yon1 = 7;
            if (yon2 == 2 || yon2 == 4 || yon2 == 6)
                   yon2 = yon2 - 1;
            else if (yon2 == 0)
                   yon2 = 7;

            if (directioncontrol(yon1, yon2) == true)
            {
             //MessageBox.Show("zıt yonlerde yani doğru, açıları hesap et");
             double angle1 = anglehesapla(anaendpoint, c);
             double angle2 = anglehesapla(endpoint2, d);
             if (anglecontrol(angle1, angle2) == true)
             {
               //MessageBox.Show("açılar da tamam, birleştirebilirsin");

               int[,] data = new int[4, 2];
               data[0, 0] = c[0, 0];
               data[0, 1] = c[0, 1];
               data[1, 0] = anaendpoint[0, 0];
               data[1, 1] = anaendpoint[0, 1];
               data[2, 0] = endpoint2[0, 0];
               data[2, 1] = endpoint2[0, 1];
               data[3, 0] = d[0, 0];
               data[3, 1] = d[0, 1];

               data = sirala(data);
               NewtonInterpolation(data);
               greyPixelArray[anaendpoint[0, 0], anaendpoint[0, 1]] = 0;
               greyPixelArray[endpoint2[0, 0], endpoint2[0, 1]] = 0;
               }

          }
         }
         }
        else { goto t; }

         adim: ;
         say = 0;
         sifirla();
         t: ;
}

}
}
```

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı, Soyadı: Emrah HANÇER
Uyruğu: Türkiye (TC)
Doğum Tarihi ve Yeri: 6 Temmuz 1986, Ankara
Medeni Durumu: Bekâr
Tel: +90 352 437 49 01 (Dahili: 32583)
email: emrahhanc@gmail.com
Yazışma Adresi: Erciyes Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliği Bölümü 38039 Talas/KAYSERİ

EĞİTİM

Derece	Kurum	Mezuniyet Tarihi
Yüksek Lisans	Erciyes Üniversitesi Bilgisayar Mühendisliği	2011(Yatay Geçiş)-2012
Yüksek Lisans	Ankara Üniversitesi Bilgisayar Mühendisliği	2009-2011
Lisans	Çankaya Üniversitesi Matematik-Bilgisayar Bilimleri	2009
Lise	Yıldırım Beyazit Anadolu Lisesi	2004

İŞ DENEYİMLERİ

Yıl	Kurum	Görev
2011-2012	Erciyes Üniversitesi Bilgisayar Mühendisliği	Araştırma Görevlisi
2010-2011	Mehmet Akif Ersoy Üniversitesi Bilgisayar Tek. ve Bilişim Sistemleri	Araştırma Görevlisi

YABANCI DİL

İngilizce

YAYINLAR

1) Hancer, E., Samet, R., "Advanced contour reconnection in scanned topographic maps," *Application of Information and Communication Technologies (AICT), 2011 5th International Conference on*, 2011, pp. 1-5.

2) Samet, R., Hancer, E., 2012. A New Approach To The Reconstruction Of Contour Lines Extracted From Topographic Maps. **Journal of Visual Communication and Image Representation, 23** (4): 642-647