

T.C.
YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

**YERSEL LAZER TARAMA TEKNİĞİ İLE ELDE EDİLEN NOKTA
BULUTLARININ İŞLENMESİ VE SEGMENTASYON
YÖNTEMLERİNİN ARAŞTIRILMASI**

Duygu ARICAN

YÜKSEK LİSANS TEZİ

Harita Mühendisliği Anabilim Dalı

Uzaktan Algılama ve CBS Programı

Danışman

Dr. Öğr. Üyesi Melis UZAR

Temmuz, 2019

T.C.
YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

**YERSEL LAZER TARAMA TEKNİĞİ İLE ELDE EDİLEN NOKTA
BULUTLARININ İŞLENMESİ VE SEGMENTASYON YÖNTEMLERİNİN
ARAŞTIRILMASI**

Duygu ARICAN tarafından hazırlanan tez çalışması 12.07.2019 tarihinde aşağıdaki jüri tarafından Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü Harita Mühendisliği Anabilim Dalı, Uzaktan Algılama ve CBS Programı **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Dr. Öğr. Üyesi Melis UZAR

Yıldız Teknik Üniversitesi

Danışman

Jüri Üyeleri

Dr. Öğr. Üyesi Melis UZAR, Danışman

Yıldız Teknik Üniversitesi

Prof. Dr. Bülent BAYRAM, Üye

Yıldız Teknik Üniversitesi

Prof. Dr. Zafer Dursun ŞEKER, Üye

İstanbul Teknik Üniversitesi

Danışmanım Dr. Öğr. Üyesi Melis UZAR sorumluluğunda tarafımda hazırlanan Yersel Lazer Tarama Tekniği ile Elde Edilen Nokta Bulutlarının İşlenmesi ve Segmentasyon Yöntemlerinin Araştırılması başlıklı çalışmada veri toplama ve veri kullanımında gerekli yasal izinleri aldığımı, diğer kaynaklardan aldığım bilgileri ana metin ve referanslarda eksiksiz gösterdiğimi, araştırma verilerine ve sonuçlarına ilişkin çarpıtma ve/veya sahtecilik yapmadığımı, çalışmam süresince bilimsel araştırma ve etik ilkelerine uygun davrandığımı beyan ederim. Beyanımın aksinin ispatı halinde her türlü yasal sonucu kabul ederim.

Duygu ARICAN

İmza



Anneme

TEŞEKKÜR

Bu tez çalışmasında, yersel lazer tarama verisine uygulanan segmentasyon yöntemleri değerlendirilmiştir. İlk olarak, çalışmanın konusunun belirlenmesinde ve hazırlanma sürecinin her aşamasında değerli bilgilerini ve zamanını benden esirgemeyerek her fırsatta çalışmamla yakından ilgilenen, yol gösteren ve destekleyen danışman hocam Dr. Öğr. Üyesi Melis UZAR'a; çalışmam boyunca bilgisi, tecrübesi ve destekleri ile her zaman yardımcı olan Doç. Dr. Nursu TUNALIOĞLU'na; hayatımın her anında maddi ve manevi tüm desteği ile arkamda duran canım anneme ve kardeşime; çalışmam boyunca yardımlarını esirgemeyen arkadaşım Tümay ARDA'ya; çalışmamda kullandığım veriyi elde etmede yardımcı olan arkadaşım Hakan ÖĞÜTCÜ'ye ve OSTEM Mühendislik'e; son olarak tez çalışmam boyunca hayatımdan kafeini eksik etmeyen Orkun ÜSTEL'e teşekkürü borç bilirim.

Duygu ARICAN

İÇİNDEKİLER

SİMGE LİSTESİ	vii
KISALTMA LİSTESİ	viii
ŞEKİL LİSTESİ	ix
TABLO LİSTESİ	x
ÖZET	xi
ABSTRACT	xiii
1 Giriş	1
1.1 Literatür Özeti.....	1
1.2 Tezin Amacı.....	6
1.3 Hipotez.....	6
2 Yersel Lazer Tarama Tekniği	8
2.1 Yersel Lazer Tarayıcılar.....	9
2.1.1 Yersel Lazer Tarayıcıların Çalışma İlkeleri.....	11
2.2 Yersel Lazer Tarama Verisinin İşlenmesi	11
2.2.1 Ön İşleme	12
2.2.2 Segmentasyon.....	16
3 Nesne Yönelimli Programlama	21
3.1 C++ Programlama Dili	21
3.1.1 C++ Programlama Dili Yapısı	24
3.2 Entegre Geliştirme Ortamı	26
3.3 Nokta Bulutu Kütüphanesi.....	26
4 Materyal ve Yöntem	28

4.1 Çalışma Alanı ve Kullanılan Materyaller	28
4.2 Nokta Bulutu Verisinin işlenmesi	31
4.2.1 Nokta Bulutlarının Renklendirilmesi	33
4.2.2 Nokta Bulutlarının Temizlenmesi	33
4.2.3 Nokta Bulutlarının Birleştirilmesi	36
4.2.4 Nokta İnceltme	38
4.2.5 Segmentasyon	40
5 Sonuç ve Öneriler	54
A Voxel Grid Alt-örnekleme	59
B Bölge Büyütme Algoritması	60
C Bölge Büyütme Algoritmasında Kullanılan Parametreler	62
D RANSAC Algoritması	65
E RANSAC Algoritmasında Kullanılan Parametreler	68
F Normallerin Farkına Dayalı Segmentasyon Algoritması	71
G Normallerin Farkına Dayalı Segmentasyon Algoritmasında Kullanılan Parametreler	75
Kaynakça	78
Tezden Üretilmiş Yayınlar	85

SİMGE LİSTESİ

c	Işık hızı
d	Öklid mesafesi
m	model için gerekli minimum nokta sayısı
N	İterasyon sayısı
p	örneklem kümelerinden en az birinin kaba hata içermeme olasılığı
r_i	Referans nokta bulutu
r_j	Hedef nokta bulutu
u	Seçilen noktanın modele uygun olma olasılığı
v	Kaba hata gözleme olasılığı
x_i	Referans nokta bulutundaki noktanın x koordinatı
x_j	Hedef nokta bulutundaki noktanın x koordinatı
y_i	Referans nokta bulutundaki noktanın y koordinatı
y_j	Hedef nokta bulutundaki noktanın y koordinatı
z_i	Referans nokta bulutundaki noktanın z koordinatı
z_j	Hedef nokta bulutundaki noktanın z koordinatı
α	Gelen ışının nadir doğrultusu ile yaptığı açı
β	Giden ışının nadir doğrultusu ile yaptığı açı
π	Pi sayısı

KISALTMA LİSTESİ

2B	2 Boyutlu
3B	3 Boyutlu
BSD	Berkeley Software Distribution
CPU	Central Processing Unit
EGO	Entegre Geliştirme Ortamı
FLANN	Fast Library for Approximate Nearest Neighbors
GB	Giga Byte
GHz	Giga Hertz
GPS	Global Positioning System
GUI	Graphical User Interface
INS	Inertial Navigation System
İEYN	İteratif En Yakın Nokta
LiDAR	Light Detection and Ranging
MFC	Microsoft Foundation Classes
Norvana	NORmal Varyasyon Analizi
NYP	Nesne Yönelimli Programlama
PCL	Point Cloud Library – Nokta bulutu kütüphanesi
RANSAC	RANdom SAMple Consensus – Rastgele Örnek Konsensüsü
RGB	Red Green Blue
RMS	Root Mean Square
SIMULA	SIMUlation LAnguage
YLT	Yersel Lazer Tarama

ŞEKİL LİSTESİ

Şekil 2.1	YLT tekniğinin kullanıldığı alanlar [7].....	9
Şekil 2.2	Optik 3B ölçme sistemlerinin sınıflandırılması [34]	10
Şekil 2.3	3B yüzeylerin optik ölçülmesinde kullanılan aktif yöntemler: (a) ışık ulaşım süresinin kestirimi, (b) triangülasyon [34]	11
Şekil 2.4	YLT prosedürü genel iş akışı [37]	12
Şekil 2.5	İEYN yöntemiyle referans nokta bulutundaki noktanın hedef nokta bulutundaki karşılığı en yakın noktanın bulunması: (a) Noktadan-noktaya, (b) noktadan-yüzeye, (c) noktadan-projeksiyon merkezine mesafenin belirlenmesi [42]	14
Şekil 2.6	İEYN yöntemiyle referans nokta bulutundaki noktalara hedef nokta bulutundaki en yakın noktaların bulunması: (a) Eşik değerden daha uzak mesafede olan noktalara yapılan eşleştirmeler, (b) Hedef nokta bulutundaki bir noktayla referans nokta bulutundaki birden fazla noktanın eşleşmesi [42]	15
Şekil 2.7	(a) Bir voksel ve (b) bu voksele karşılık gelen octree'nin gösterimi [28]	16
Şekil 2.8	Segmentasyon Yöntemleri [45]	17
Şekil 3.1	Tipik bir C ortamı [50].....	22
Şekil 4.1	Çalışma alanı	28
Şekil 4.2	Veri setleri: (a) Kupa, (b) Masa, (c) Tesisat.....	31
Şekil 4.3	Uygulama genel iş akışı.....	32
Şekil 4.4	Homojen olmayan nokta dağılımı.....	38
Şekil 4.5	Test verisinde nokta inceltme işlemi: (a) öncesi (b) sonrası.....	40
Şekil 4.6	İnceltilecek nokta bulutu: (a) öncesi (b) sonrası.....	40
Şekil 4.7	PCL kütüphanesi kullanılarak nokta normallerinin hesaplanması	42
Şekil 4.8	Bölge büyütme algoritmasının programlanması.....	44
Şekil 4.9	Bölge büyütme algoritması ile segmentasyon: (a) Kupa nokta bulutu (b) Masa nokta bulutu (c) Tesisat nokta bulutu (d) Tez nokta bulutu.....	45
Şekil 4.10	RANSAC algoritması ile segmentasyon: (a) Kupa nokta bulutu (b) Masa nokta bulutu (c) Tesisat nokta bulutu (d) Tez nokta bulutu	49
Şekil 4.11	Yüzey normallerinde kullanılan farklı yarıçapların etkisi [52]	50
Şekil 4.12	Normallerin farkı algoritması ile segmentasyon: (a) Kupa nokta bulutu (b) Masa nokta bulutu (c) Tesisat nokta bulutu (d) Tez nokta bulutu.....	53
Şekil 5.1	Veri inceltme: (a) öncesi ve (b) sonrası tez verisi.....	57

TABLO LİSTESİ

Tablo 1.1	Literatürdeki bazı YLT uygulamaları	2
Tablo 2.1	RANSAC algoritması [47]	19
Tablo 3.1	C++'ta kullanılan veri türleri [51]	25
Tablo 4.1	Kullanılan yersel lazer tarayıcının teknik özellikleri [54]	29
Tablo 4.2	Ham veri setindeki nokta bulutları ve nokta sayıları	30
Tablo 4.3	Test verileri ve özellikleri	31
Tablo 4.4	Filtrelemede kullanılan küre yarıçap parametreleri	34
Tablo 4.5	Çalışmada kullanılan nokta bulutlarına ait nokta sayıları	35
Tablo 4.6	Birleştirmede kullanılan referans ve hedef nokta bulutlarıyla elde edilen RMS değerleri	36
Tablo 4.7	Birleştirme işlemi sonucu elde edilen RMS değerleri	37
Tablo 4.8	Voksel grid alt-örnekleme yöntemine ait sözde kod	39
Tablo 4.9	Bölge büyütme algoritmasına ait sözde kod	41
Tablo 4.10	RANSAC algoritmasına ait sözde kod	46
Tablo 4.11	Normallerin Farkına dayalı segmentasyon algoritmasına ait sözde kod	51
Tablo 5.1	Çalışma içerisinde türetilen veri setleri	54
Tablo 5.2	Tüm veri setlerindeki birleştirmelere ait RMS değerlerinin ortalamaları	55

Yersel Lazer Tarama Tekniği ile Elde Edilen Nokta Bulutlarının İşlenmesi ve Segmentasyon Yöntemlerinin Araştırılması

Duygu ARICAN

Harita Mühendisliği Anabilim Dalı

Yüksek Lisans Tezi

Danışman: Dr. Öğr. Üyesi Melis UZAR

Tarihi yapılar, kültürel mirasın nesilden nesile aktarılması için önemli bir bileşendir. Bu eserlerin zamanla yıpranması ve zarar görmesi durumunda, restorasyonları için aslına uygun modellenmesi oldukça önemlidir. Lazer tarama tekniği ile bu eserlerin topoğrafyasına ait üç boyutlu nokta bulutu verisi hızlı, detaylı ve doğruluğu yüksek bir şekilde elde edilebilmektedir. Ancak, elde edilen nokta bulutundan nesnenin doğru bir sayısal modelinin üretilebilmesi için bir dizi işlem adımı uygulanmalıdır. Ham ve işlenmiş verilerdeki hatalar, özellikle stokastik yöntemlerin kullanılması sebebiyle iş akış yolu ile yayılarak tüm çalışmadaki doğruluğu azaltma potansiyelindedir. Bu nedenle, veri işlemenin tüm adımları titizlikle planlanarak gerçekleştirilmelidir. Bu tez çalışmasının amacı yersel lazer tarama tekniği ile elde edilen nokta bulutlarının işlenmesi ve segmentasyon çalışmalarında kullanılan algoritmaların araştırılmasıdır. Bu kapsamda İstanbul, Silivri İlçesi'nde bulunan Değirmenköy-Germiyan Rum Ortodoks Kilisesi'nin ön cephesine ait nokta bulutu verisi kullanılarak veri işleme adımları detaylı bir şekilde incelenmiştir. Nokta

bulutunun temizlenmesi ve birleştirilmesi aşamasında CloudCompare yazılımından yararlanılmıştır. Nokta bulutunun temizlenmesi, filtreleme ve manuel temizleme işlem adımlarını kapsayarak; birleştirme adımı ise İteratif En Yakın Nokta yönteminin uygulanmasıyla incelenmiştir. Nokta inceltme ve segmentasyon yöntemlerinin incelenmesinde ise C++ programlama dilinde PCL kütüphanesinden yararlanılmıştır. Nokta inceltme için Voksel Grid Alt-Örnekleme yöntemi incelenmiştir. Segmentasyon yöntemlerinden ise bölge büyütme, RANSAC ve normallerin farkına dayalı segmentasyon algoritmaları araştırılmıştır. Veri işlemenin ilk adımı olan ön işleme adımlarının farklı sırayla uygulanması ile nokta bulutlarının birleştirilmelerinden elde edilen sonuçlara göre “Ön işleme adımlarında her nokta bulutu ayrı ayrı ele alınmalıdır” sonucu ortaya çıkmıştır. Ayrıca segmentasyon algoritmalarının uygulanmasında parametre seçiminin zorlayıcı, fakat uygun parametrelerin belirlenmesi durumunda nesneyi temsil eden segmentlerin oluşturulabilmesinin mümkün olduğu vurgulanmıştır.

Anahtar Kelimeler: Yersel lazer tarama, iteratif en yakın nokta, bölge büyütme, RANSAC, normallerin farkı

Investigation of Terrestrial Laser Scanning Point Cloud Processing and Segmentation Methods

Duygu ARICAN

Department of Geomatics Engineering

MSc. Thesis

Advisor: Assist. Prof. Melis UZAR

Historical buildings are an important component of cultural heritage to convey through generations. It is crucial to create proper models for restoration in case of deterioration and damage. The three-dimensional point cloud data of the topography of these artefacts can be obtained fast, detailed and accurately by using laser scanning method. However, a series of processing steps must be followed to produce an accurate model of the object from the point cloud that obtained. Errors in raw and processed data, especially due to the use of stochastic methods, have the potential to decrease accuracy of whole study by spread through workflow. Thus, all steps of data processing must be carefully planned and performed. The aim of this thesis study to investigate the processing steps and segmentation algorithms for point clouds obtained by terrestrial laser scanning method. Within this scope, data processing steps were detailly examined using point cloud data which belongs Değirmenköy-Germiyan Greek Orthodox Church's front façade located in Silivri, İstanbul. CloudCompare software was used for cleaning and registering point cloud. Filtering and manual cleaning for cleaning point cloud; Iterative closest point method for registering were investigated. PCL library was utilized for C++ programming language to analyze point thinning and segmentation methods. Voxel grid down sampling method was implemented for point thinning. Region growing,

RANSAC and difference of normals segmentation algorithms were implemented to investigate segmentation methods. According to the obtained results by registering point cloud, it was concluded as "each point cloud must be handled separately" for the pre-processing steps as the first step of data processing. Furthermore, it was emphasized that it is coercive to selecting appropriate parameters for the implementation of the algorithms, but it is possible to create segments that represent the object surface, with appropriate parameters.

Keywords: Terrestrial laser scanning, iterative closest point, region growing, RANSAC, difference of normals



1.1 Literatür Özeti

Tarihi yapılar, kültürel mirasın nesilden nesile aktarımı için önemli bir bileşen olmakla birlikte, medeniyetlerin de sembolüdür. Bu eserlerin zamanla yıpranması veya doğal afetler gibi yıkıcı faktörlerin etkisi ile zarar görmesi durumunda restorasyonları için aslına uygun bir şekilde modellenmesi oldukça önemlidir [1]. Günümüzde kültürel mirasın korunmasında, bilgi toplama ve sayısal dokümantasyon uygulamaları sürecin önemli bir parçasını oluşturmaktadır [2].

Lazer tarama teknolojisi ile nesne yüzeyinin topoğrafyasına ait doğruluğu yüksek üç boyutlu nokta bulutu verisi, hızlı bir şekilde elde edilmektedir. Bu gibi özellikleri nedeniyle son zamanlarda lazer tarama teknolojilerine ilgi oldukça artmıştır. Büyük ölçekli modelleme çalışmalarında, otomatik ve etkili yöntemlerin geliştirilmesi ile binaların modellenmesi için, doğruluğu yüksek üç boyutlu veri toplama aşamasında, lazer tarama tekniği maliyet ve zaman bakımından kazanç sağlamaktadır [3]. Bunların yanı sıra lazer tarama tekniği, geleneksel tekniklerle ölçümü yapılamayan alanların ve kompleks geometriye sahip nesnelerin detaylı bir şekilde ölçülmesi gibi olanaklara da sahiptir [4].

Lazer tarama tekniği ile elde edilen verilerin işlenerek nokta bulutundan nesnenin sayısal modelinin elde edilmesi için bir dizi işlem adımı uygulanmalıdır. Bu işlem adımları genel olarak 2 ayrı aşamada toplanmaktadır [5]:

- Ön işleme,
- Katı modelleme.

“Ön işleme” filtreleme, nokta bulutlarının birleştirilmesi ve jeoreferanslandırılması gibi doğrudan nokta bulutları üzerinden gerçekleştirilen işlemleri içermektedir. Bu işlemlerin sonucunda bir sonraki aşama için komplike yapıda “gürültüsüz” (kaba hata ve sistematik hataları içermeyen) bir nokta bulutu elde edilir [5].

Bir sonraki aşama olan katı model üretimi, yüzey modelin oluşturulmasını sağlayan segmentasyon, sınıflandırma, yüzey ağı oluşturma gibi bir dizi işlemten oluşmaktadır [5].

Farklı uygulamalar için üretilen modellerden farklı doğruluk ve hassasiyet beklenmektedir. Kültürel mirasın dokümantasyonu ve restorasyonu için yapılan çalışmalarda modellerin oldukça yüksek doğrulukta ve hassasiyette olması beklenmektedir. Ham ve işlenmiş verilerdeki hatalar, özellikle stokastik yöntemlerin kullanılması sebebiyle, iş akışı yoluyla yayılma potansiyeline sahiptir. Bu sebeple veri işlemedeki tüm adımlar titizlikle gerçekleştirilmelidir [6].

İlk yıllarda lazer tarama, ağırlıklı olarak otomotiv ve endüstriyel tasarım sürecini kolaylaştırmak için kullanılmıştır. Teknolojinin gelişmesi ile lazer tarayıcıların diğer alanlarda da kullanımı artmıştır [7]. YLT tekniğinin kullanım alanları; ulaşım ve altyapı uygulamaları, mimari ve bina ölçme uygulamaları, kıyı uygulamaları, afet izleme ve risk tespit uygulamaları, kültürel mirasın korunması ve arkeolojik uygulamalar, endüstriyel uygulamalar, madencilik, taş ocağı ve kazı uygulamaları, deformasyon ölçmeleri uygulamaları şeklinde sıralanabilir [8]. Buna göre bazı YLT uygulamaları Tablo 1.1’de belirtilmiştir.

Tablo 1.1 Literatürdeki bazı YLT uygulamaları

Çalışma Adı	Yazarlar	Yılı	Uygulama alanı
Laser scanned point clouds to support autonomous vehicles [9]	Vivien Poto, Jozsef Arpad Somogyi, Tamas Lovas, Arpad Barsi	2017	Ulaşım
Terrestrial laser scanning digitalization in underground constructions [10]	Cesar Porras-Amores, Fernando R. Mazarron, Ignacio Canas, Paola Villoria Saez	2019	Altyapı
Estimating architecture-based metabolic scaling exponents of tropical trees	Alvaro Lau, Christopher Martius, Harm Bartholomeus, Alexander Shenkin,	2019	Mimari

using terrestrial LiDAR and 3D modelling [11]	Tobias Jackson, Yadvinder Malhi, Martin Herold, Lisa Patrick Bentley		
Generation of visually aesthetic and detailed 3D models of historical cities by using laser scanning and digital photogrammetry [12]	Jose Balsa-Barreiro, Dieter Fritsch	2018	Bina modelleme
Evaluation of beach profile changes by using terrestrial laser scanner [13]	Aysun Koroglu, Dursun Z. Seker, M. Sedat Kabdasli, Hakan Karabork, Ayhan Goktepe, Evren Varol, Bulent Bayram	2017	Kıyı
Using the terrestrial laser scanning technique for aeolian sediment transport assessment in the coastal zone in seasonal scale [14]	Aleksandr Danchenkov, Nikolay Belov, Zhanna Stont	2019	Kıyı
Quantitative assessment for the rockfall hazard in a post-earthquake high rock slope using terrestrial laser scanning [15]	Hai-bo Li, Xiao-wen Li, Wan-zhou Li, Shi-lin Zhang, Jia-wen Zhou	2019	Afet izleme
Quantifying erosion of 'at risk' archaeological sites using repeat terrestrial laser scanning [16]	Mark Kinney, Chris Gerrard, Jeff Warburton	2017	Risk tespit
Documentation of cultural heritage using digital photogrammetry and laser scanning [17]	Naci Yastikli	2007	Kültürel miras
3D Documentation of a historical monument using terrestrial laser scanning	T. Temizer, G. Nemli, A. Ekizce, S. Demir, B. Bayram, H. İ. Aşkın,	2013	Kültürel miras

case study: Byzantine Water Cistern, Istanbul [18]	A. V. Coban, H. F. Yilmaz,		
Terrestrial laser scanning and close-range photogrammetry for 3D archaeological documentation: the Upper Palaeolithic Cave of Parpalló as a case study [19]	Jose Luis Lerma, Santiago Navarro, Miriam Cabrelles, Valentin Villaverde	2010	Arkeoloji
3D reconstruction of as-built industrial instrumentation models from laser-scan data and a 3D CAD database based on prior knowledge [20]	Hyojoo Son, Changmin Kim, Changwan Kim	2015	Endüstri
Octree-based segmentation for terrestrial LiDAR point cloud data in industrial applications [21]	Yun-Ting Su, James Bethel, Shuowen Hu	2016	Endüstri
Application of laser scanning for rock mass characterization and discrete fracture network generation in an underground limestone mine [22]	Juan J. Monsalve, Jon Baggett, Richard Bishop, Nino Ripepi	2019	Madencilik
Deformation measurement using terrestrial laser scanning data and least squares 3D surface matching [23]	O. Monserrat, M Crosetto	2008	Deformasyon
Deformation monitoring of typical composite structures based on terrestrial laser scanning technology [24]	Xu, Xiangyang; Yang, Hao; Neumann, Ingo	2018	Deformasyon

Abmayr, Hartli Reinköster ve Fröhlich 2005 yılında [25], IMAGER 5003 lazer tarayıcı kullanarak görselleştirme ve modelleme çalışmaları yapmışlardır.

Çalışmada sistemin özellikleri ve kullandıkları yöntemleri açıklamanın yanı sıra kültürel miras ve inşaat mühendisliği alanında iki uygulama gösterilmektedir. Hayakawa ve Oguchi tarafından 2016 yılında [26] jeomorfolojideki YLT uygulamaları derlenmiştir.

Oniga, Savu ve Negrila tarafından gerçekleştirilen 2016 yılındaki çalışmada [27], birleştirme işleminin 3B model oluşturma sürecinin ilk ve en önemli adımı olduğu ayrıca bu adımın mümkün olduğunca en doğru şekilde yapılması gerektiği vurgulanmıştır. Çalışmada CloudCompare yazılımında YLT nokta bulutları için bağlama noktası temelli birleştirme yaklaşımına ait doğruluk değerlendirmesi yapılmıştır. İki farklı YLT nokta bulutu kullanılarak beş nokta çiftinin manuel olarak seçilmesi ve ölçülmesi ile sonuçlar elde edilmiştir. Birleştirme parametrelerinin kestiriminde 3B konformal dönüşüm ve en küçük kareler yöntemi kullanılmıştır.

Morena ve Li tarafından 2016 yılında [28], hangi yöntemlerin en iyi ve hangi durumda işe yaradığını belirlemek için karşılaştırmalı bir çalışma yapılmıştır. Çalışmada veri işleme için nokta bulutu kütüphanesinden (PCL) yararlanılmıştır. Veri setine geçiş (pass through) filtresi, voksel grid filtresi ve iki taraflı (bilateral) filtre uygulanmıştır.

Sahin tarafından 2014 yılında [29], yersel lazer tarayıcı verileri ile düzlemsel yüzeylerin ayrılmasına yönelik matematiksel bir filtreleme algoritması ile segmentasyon çalışması yapılmıştır. Segmentasyon için uygulanan filtreleme, noktanın düzleme olan mesafesine dayanmaktadır.

Che ve Olsen tarafından 2017 yılında [30], iki ana adımdan oluşan bir segmentasyon yöntemi sunulmuştur. İlk adımda, Norvana adı verilen özellik çıkarma yaklaşımı uygulanmıştır. Bu yaklaşımla gürültülerin bir kısmının giderilmesinin yanı sıra genel bir normal kestirimi yapılmadan nesnenin kenar noktaları çıkarılmıştır. İkinci adımda ise segmentasyon sonucunun elde edilmesi için bölge büyütme yöntemi ile pürüzsüz bir yüzey üzerindeki noktalar gruplandırılmıştır. Ayrıca bu yöntem birden fazla tarama noktasından taranan ve birleştirilen verinin segmentasyonu gerçekleştirilebilmektedir.

Çömert tarafından 2014 yılında [31], yapılan tez çalışmasında YLT verilerinin 3B modellenmesi işlemi, “Yüzey ağı modelleme” ve “Geometrik temelli modelleme” olarak ikiye ayrılmış, geometrik temelli modelleme için RANSAC algoritması nokta bulutu verisine uygulanmıştır. Bunun sonucunda nokta bulutu verisinde yer alan düzlem, silindir, koni ve küre geometrik şekilli yüzeye sahip nesnelerin otomatik olarak çıkarılması sağlanmıştır.

Ioannou, Taati, Harrap ve Greenspan tarafından 2012 yılında [32], normallerin farkına dayalı bir segmentasyon yöntemi sunulmuştur. Uygulamada 2 farklı kentsel alan LiDAR veri seti kullanılmıştır. İki veri setinde de Normallerin Farkı operatörünün otomobiller, insanlar ve sokak lambaları gibi nesneleri otomatik belirleyerek kümelerle ayırdığı gösterilmiştir.

Bu tez çalışmasının ikinci bölümünde YLT tekniği hakkında bilgi verilmiştir. Üçüncü bölümde, Nesne Yönelimli Programlama anlatılmıştır. Dördüncü bölümde, kullanılan materyal ve yöntem sunulmuştur. Son olarak, beşinci bölümde sonuçlar ve önerilerle çalışma tamamlanmıştır.

1.2 Tezin Amacı

Tez çalışması kapsamında YLT tekniği ile elde edilen nokta bulutu verilerinin, veri setine uygun işlem adımları ile işlenmesi, istenmeyen noktaların etkin bir şekilde kaldırılması ve segmentasyonun hassasiyetini etkilemeyecek şekilde verinin büyük ölçüde azaltılması amaçlanmıştır. Segmentasyon yöntemlerinin değerlendirilmesi amacıyla Bölge büyütme, RANSAC ve Normallerin farkına dayalı segmentasyon algoritmalarıyla veri setlerindeki nesneleri temsil eden noktaların segment olarak ayrılabilmesi hedeflenmiştir.

1.3 Hipotez

YLT verilerinin segmente edilebilmesi için öncelikle veri seti segmentasyon adımına uygun bir şekilde hazırlanmalıdır. Segmentasyon adımında ise hazırlanan veriler farklı algoritmalarla segmente edilebilmektedir. Bölge büyütme gibi algoritmalarla komşuluk ilişkisi kullanılarak segmentler oluşturulması, RANSAC gibi algoritmalarla düzgün nesne geometrisine sahip verilerin veya verideki nokta normalleri gibi özelliklerin hesaplanması ile segmentlere ayrılması mümkündür.

Seilen yntemin bařarılı bir řekilde uygulanması veri setinin ierięi ile doęrudan ilgilidir. Bu alıřmada Blge Bytme, RANSAC ve Normallerin farkı segmentasyon algoritmaları kullanılarak bu algoritmaların alıřma prensipleri, farklı veri setlerine uygulanarak incelenmiřtir. Tez alıřması kapsamında gerekleřtirilen alıřmalar “YLT nokta bulutu segmentasyon algoritmaları ile nesneyi temsil eden noktaların segmentlere ayrılabilmesi hem veri setine hem de segmentasyon yntemine baęlıdır.” hipotezi zerine inřa edilmiřtir.



YLT teknolojisi, 20. yüzyılın son yarısında yeryüzünün ve nesne yüzeylerinin gerçeğe yakın bir şekilde, üç boyutlu, yeniden oluşturulması için üretilmiş ve günümüzde hala gelişmekte olan bir teknolojidir [33].

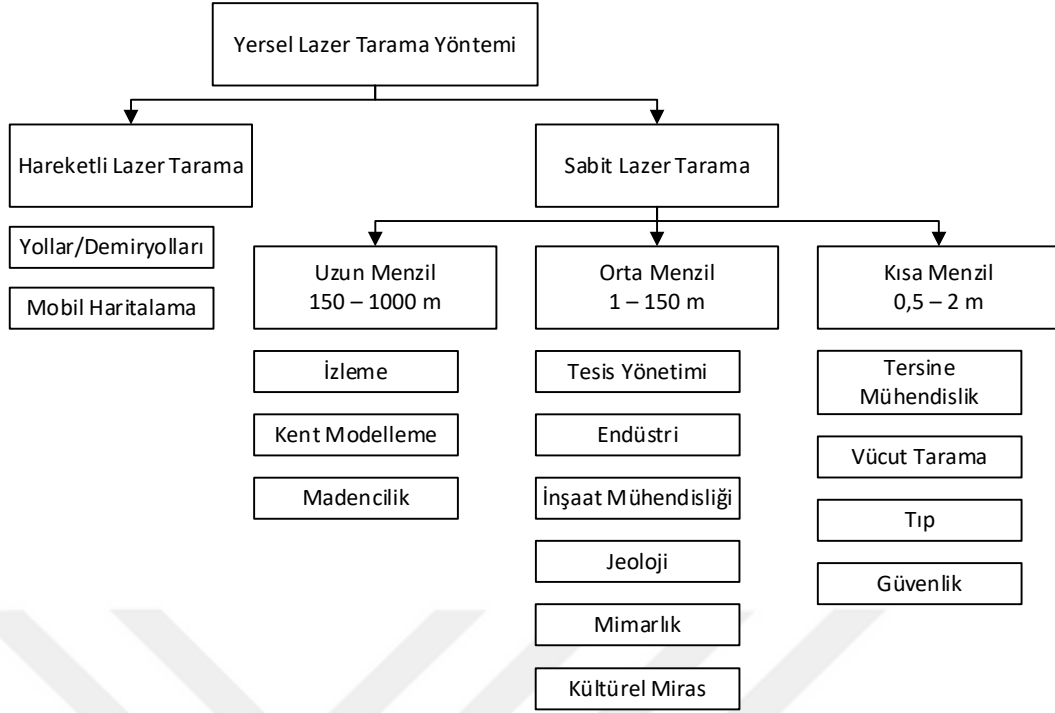
YLT tekniğinde nesnelerin kapladığı alan, yatay ve düşey doğrultuda saniyede binlerce noktanın lazer tarayıcı ile taranmasıyla ölçülmektedir. Ölçüm sonucunda “nokta bulutu” olarak adlandırılan, nesnenin milyonlarca noktadan oluşan oldukça ayrıntılı 3B görüntüsü elde edilmektedir. Her nokta için, tarayıcıya göre ayarlanan koordinat sistemindeki 3B koordinatlar ve lazer sinyalinin yansıma değeri kaydedilir. Yersel lazer tarayıcılar ile hem sabit hem de hareketli platformlardan tarama yapılabilmektedir [4].

Veri alımı sırasında tarayıcının sabit bir noktada konumlandırılması ile yapılan tarama işlemine sabit lazer tarama denir. Bu teknikle hareketli lazer tarama tekniğine göre, daha yoğun ve yüksek doğrulukta nokta bulutu elde edilebilmektedir. Sabit lazer taramaların tamamı YLT tekniği olarak görülebilir, ancak tüm yersel lazer taramalar sabit lazer tarama olarak sınıflandırılmamaktadır [7].

Hareketli lazer tarama durumunda, tarayıcı mobil bir platform üzerine monte edilir. Bu sistemler, INS ve GPS gibi ek konumlandırma sistemleri gerektirmektedir [7].

Lazer tarama tekniği, kompleks nesne geometrisini detaylı, yüksek çözünürlüklü ve hızlı bir şekilde yakalama gibi avantajlara sahiptir. Ayrıca, geleneksel tekniklerle ölçümü yapılamayan afet alanları veya ulaşılamayan alanlarda nesnelerin modellenmesini mümkün kılma, projenin daha hızlı tamamlanması ve proje masraflarının azaltılmasını sağlama gibi avantajlara da sahiptir [4].

Lazer tarayıcılarla elde edilen veriler daha sonra çeşitli uygulamalar için sayısal 2B çizimler veya 3B modeller oluşturmak için kullanılabilmektedir [7].



Şekil 2.1 YLT tekniğinin kullanıldığı alanlar [7]

İlk yıllarda lazer tarama, ağırlıklı olarak otomotiv ve endüstriyel tasarımda Bilgisayar Destekli Tasarım sürecini kolaylaştırmak için kullanılmıştır. Teknolojinin gelişmesi ile lazer tarayıcılar diğer alanlarda da kullanılarak, kullanım alanları giderek genişlemiştir. Şekil 2.1’de YLT tekniğinin kullanıldığı alanlar gösterilmiştir [7].

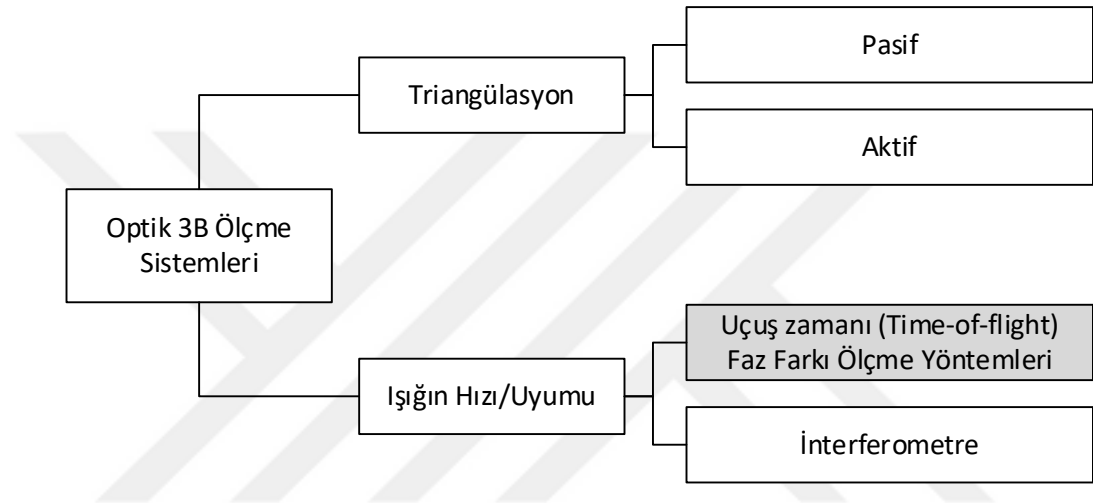
2.1 Yersel Lazer Tarayıcılar

Lazer tarayıcılar nesnelerin ve alanların görünür yüzeylerinin doku ve geometri bilgisini yakalayan ve kayıt eden ölçme sistemleridir. Bu sistemler temassız ölçüm cihazlarıdır ve belirli bir ölçüm belirsizliği ile belirli bir görüş alanındaki bir yüzeyin nicel bir 3B sayısal gösterimini (örneğin, nokta bulutu) üretir [34].

Yersel lazer tarayıcıların sınıflandırılması ölçüm prensibine veya teknik özelliklerine göre birçok şekilde yapılabilir. Tüm mühendislik ve mimarlık uygulamalarında kullanılabilecek tek bir yersel lazer tarayıcı yoktur [35]. Bu nedenle örneğin genellikle orta menzilli (150-350 m’ye kadar menzilli) tarayıcılar iç mekân için uygunken uzun menzilli tarayıcılar (350 m’den fazla menzilli) dış mekân

kullanımı için uygundur. Ayrıca yakın mesafe uygulamaları için yüksek hassasiyetli kısa menzilli (50-100 m'ye kadar) tarayıcılar da vardır [36].

Yersel lazer tarayıcılar mesafe ölçüm ilkelerine göre kategorize edilebilir. Mesafe ölçüm sistemi hem menzil hem de sistemin doğruluğu ile doğrudan ilişkilidir. Lazer tarayıcılarda menzil ölçümleri için uçuş zamanı ilkesine dayanan ölçme sistemleri, faz farkı ölçme ilkesine dayanan ölçme sistemleri ve triangülasyon olmak üzere üç teknoloji kullanılmaktadır [35]. Genel olarak optik 3B ölçme sistemleri Şekil 2.2'deki gibi kategorilere ayrılmaktadır [34].



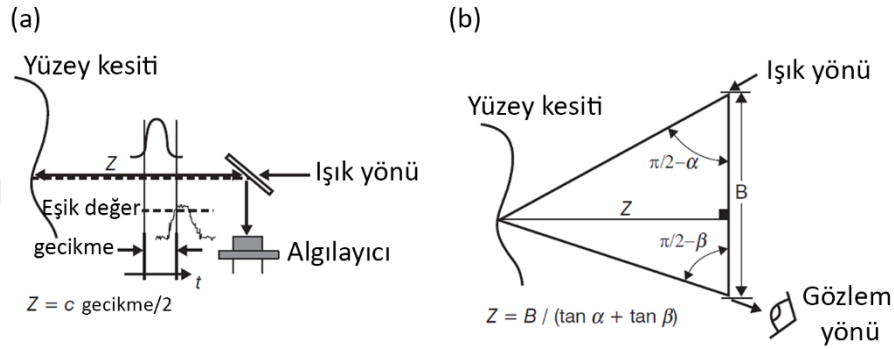
Şekil 2.2 Optik 3B ölçme sistemlerinin sınıflandırılması [34]

Teknik özelliklere göre yapılan sınıflandırmalar sistemin olanaklarını ve performansını belirttiği için kullanıcının ihtiyacını bulabilmesi açısından daha kullanışlıdır. Bu teknik özellikler:

- Tarama hızı: lazer ölçüm sisteminin örnekleme hızı
- Görüş alanı: kamera görüşü, görüntüleme
- Mekânsal çözünürlük: görüş alanında taranan nokta sayısı
- Doğruluk: ölçme sisteminin doğruluğu
- Diğer cihazlarla kombinasyonu: lazer tarayıcının üzerine monte edilen cihazlar (kamera, GPS vb.) [35].

2.1.1 Yersel Lazer Tarayıcıların Çalışma İlkeleri

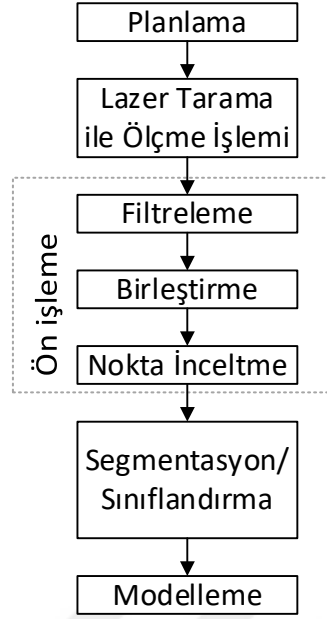
3B bir yüzeyin optik olarak ölçülmesinde iki temel aktif yöntem vardır: ışık ulaşım süresinin kestirimi ve triangölasyon. Şekil 2.3 (a)'da gösterildiği gibi, ışık dalgaları belirli bir ortamda bilinen bir hızla ilerler. Böylece, kaynaktan yansıtıcı hedef yüzeye çarpıp ve algılayıcıya geri dönen ışığın yarattığı zaman farkının ölçümü, mesafeyi değerlendirmek için uygun bir yöntem sunar. Bu sistemler aynı zamanda uçuş zamanı veya LiDAR sistemleri olarak da bilinir. Uçuş zamanı ölçümü, ayrıca sürekli dalga lazerlerindeki faz ölçümü ile dolaylı olarak da gerçekleştirilebilir. Bu ölçme prensibi faz farkı ilkesine dayanan ölçme sistemlerinde uygulanmaktadır. Triangölasyon yönteminde ise, yansıtıcı bir yüzeye yönelik bir ışık yönü açısı ($\pi/(2-\alpha)$) ve ışık kaynağından bilinen bir mesafede (B) bir gözlem yönü açısı ($\pi/(2-\beta)$) kullanılarak bir üçgen oluşturulması sağlanır ve kosinüs yasasından yararlanılarak mesafe ölçülür (Şekil 2.3 (b)) [34].



Şekil 2.3 3B yüzeylerin optik ölçülmesinde kullanılan aktif yöntemler: (a) ışık ulaşım süresinin kestirimi, (b) triangölasyon [34]

2.2 Yersel Lazer Tarama Verisinin İşlenmesi

YLT verisinin işlenmesinde izlenen iş akışı Şekil 2.4'teki diyagramdaki gibi özetlenebilmektedir [37].



Şekil 2.4 YLT prosedürü genel iş akışı [37]

2.2.1 Ön İşleme

Tarama işleminde, lazer görüş alanı içerisindeki tüm nesnelere ait 3B nokta verisi kaydedildiği için istenmeyen veriler de nokta bulutu içerisinde bulunmaktadır. Bu istenmeyen noktalar ve gürültüler, modelin doğruluğunu etkilemektedir. Ayrıca görüş alanındaki tüm nesnelerin kaydedilmesi sebebiyle bulutu verisi oldukça yoğun ve büyük bir veridir. Ön işleme adımları ile bu gürültülerin ve istenmeyen noktaların etkin bir şekilde ortadan kaldırılması ve modelleme hassasiyetini etkilemeden verilerin büyük ölçüde azaltılması amaçlanmaktadır [38]. Ön işleme adımları birleştirme, filtreleme ve nokta inceltme adımlarını kapsamaktadır [39].

2.2.1.1 Filtreleme

YLT tekniği ile taranan nesnenin parçaları, genellikle kolaylıkla tanımlanabilmektedir. Ancak bu tanımlama işlemi sonraki aşamalar için yeterli değildir [7]. Elde edilen nokta bulutu verileri kalibrasyon, sistematik hatalar ve ölçüm hatalarının etkisi ile oluşan gürültü ve kaba hataları içermektedir [40]. Bu nedenle nokta bulutunun ilk analizinde gereksiz tüm noktalar veri setinden manuel olarak silinmelidir. Ancak nokta bulutunun manuel temizlenmesi, bu kusurların giderilmesi için yeterli değildir [7]. Bu nedenle, bu kusurların giderilmesi için filtreleme işlemleri yapılmalıdır [40]. Filtrelenen verinin kalitesi model

doğruluğuna etki etmektedir. Bu nedenle filtreleme, veri işlemenin önemli bir adımıdır [41].

Filtreleme ile gürültü gideren otomatik algoritmalar temel olarak iki prensibe dayanmaktadır. İlk prensip, çevrelerinde az veya hiç nokta bulunmayan noktaların kaba hata sayılmasıdır. Kaba hatalar genellikle tarama işlemi devam ederken tarayıcının önünde hareket eden insanlardan veya diğer engellerden kaynaklanmakta ve taranacak nesnenin bir parçası olmamaktadır. Bu noktalar belirli parametre ayarları kullanılarak kolayca tanımlanabilmekte ve nokta bulutundan kaldırılabilir [7].

Diğer gürültü giderme prensibi, yüzeyde optimum bir pürüzsüzlük elde etmek için noktaların hareket ettirilmesine dayanmaktadır. Bu algoritmalar, yerel (serbest biçimli) düzlemleri nokta bulutundaki noktalara oturtmaya çalışmaktadır. Merkez nokta oturtulan düzlemden çok uzağa uzandığında, komşularına daha fazla tutarlılık sağlamak için düzleme doğru hareket ettirilerek filtrelenmektedir [7].

2.2.1.2 Birleştirme

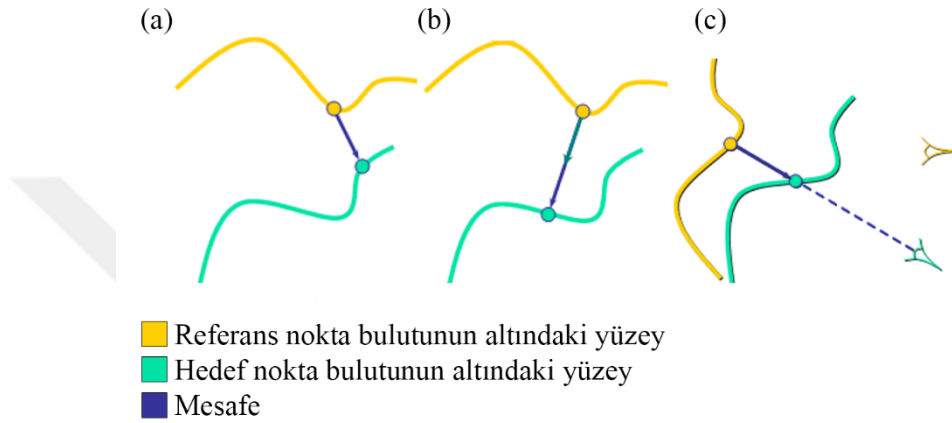
Lazer tarayıcı kullanılarak elde edilen verilerle 3B modelleme yapılırken en önemli işlem adımlarından biri, farklı tarama istasyonlarından bindirmeli olarak yapılan taramaların ortak bir koordinat sisteminde birleştirilmesidir. Nokta bulutlarının birleştirilmesinde kullanılan yöntemlerden biri olan İEYN yöntemi hem yaygın olarak kullanılan hem de doğruluğu oldukça yüksek bir yöntemdir [42].

İEYN yönteminin uygulanabilmesi için nokta bulutlarının manuel olarak birbirlerine yaklaşık yöneltilmeleri gerekmektedir. Ardından bindirmeli olarak taranmış nokta bulutlarının, ortak tarama alanlarında bulunan noktalar yardımıyla dönüşüm parametreleri hesaplanmaktadır. Bu yöntemde, nokta bulutlarından biri referans (sabit) olarak belirlenmektedir. Ardından diğer taramalar hesaplanan dönüşüm parametreleri yardımıyla referans nokta bulutunun koordinat sistemine dönüştürülmektedir. Referans nokta bulutunda bulunan tüm noktaların, hedef (döndürülüp ötelenecek olan) nokta bulutunda bulunan en yakın eşlenik noktaları aranmaktadır. En kısa Öklid mesafesine sahip nokta “en yakın eşlenik nokta” olarak tanımlanmaktadır. Referans nokta bulutundaki $r_i(x_i, y_i, z_i)$ noktası ve hedef nokta

bulutundaki $r_j(x_j, y_j, z_j)$ noktası eşlenik noktalar olmak üzere; bu noktalar arasındaki Öklid mesafesi (2.1) bağıntısı ile hesaplanmaktadır [42].

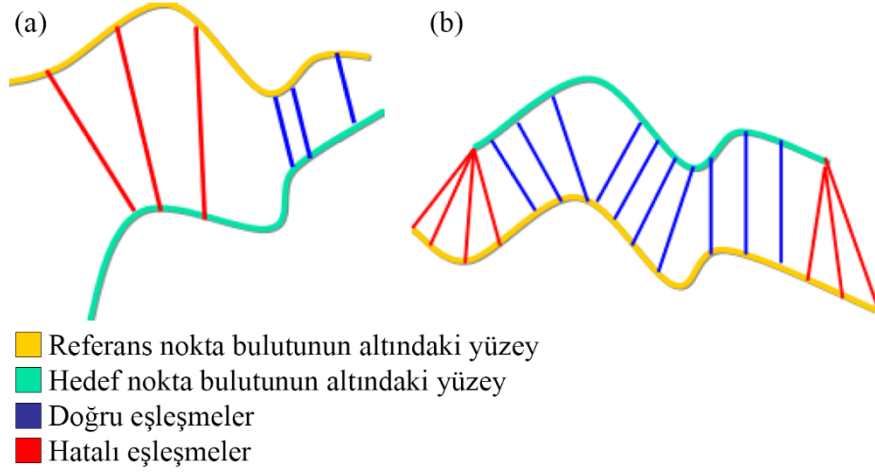
$$d(\vec{r}_i, \vec{r}_j) = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2} \quad (2.1)$$

Eşlenik noktalar, noktadan-noktaya, noktadan-yüzeye veya noktadan-projeksiyon merkezine olan en kısa Öklid mesafeleri ile belirlenmektedir (Şekil 2.5).



Şekil 2.5 İEYN yöntemiyle referans nokta bulutundaki noktanın hedef nokta bulutundaki karşılığı en yakın noktanın bulunması: (a) Noktadan-noktaya, (b) noktadan-yüzeye, (c) noktadan-projeksiyon merkezine mesafenin belirlenmesi [42]

En yakın eşlenik noktalar bulunduktan sonra, belirlenen eşik değerden daha büyük uzaklığa sahip eşleşmeler ve hedef nokta bulutundaki bir noktaya referans nokta bulutundan birden fazla nokta ile yapılan eşleşmeler hatalı olarak kabul edilmektedir (Şekil 2.6).



Şekil 2.6 İEYN yöntemiyle referans nokta bulutundaki noktalara hedef nokta bulutundaki en yakın noktaların bulunması: (a) Eşik değerden daha uzak mesafede olan noktalara yapılan eşleştirmeler, (b) Hedef nokta bulutundaki bir noktaya referans nokta bulutundaki birden fazla noktanın eşleşmesi [42]

Dönüşüm parametreleri, hatalı kabul edilen eşleşmeler göz ardı edilerek kalan eşlenik noktalarla hesaplanmakta ve bu işlem iteratif olarak devam etmektedir. Eşlenik noktalar arasındaki mesafe minimum veya dönüşümün hatası belirlenen bir değere ulaşıncaya kadar bu işleme devam edilir. Genellikle 20-30 iterasyon sonucunda dönüşüm işlemi gerçekleştirilmektedir [42]. Birleştirmeye ait RMS değerleri (2.2)'de gösterilen bağıntıya göre hesaplanmaktadır.

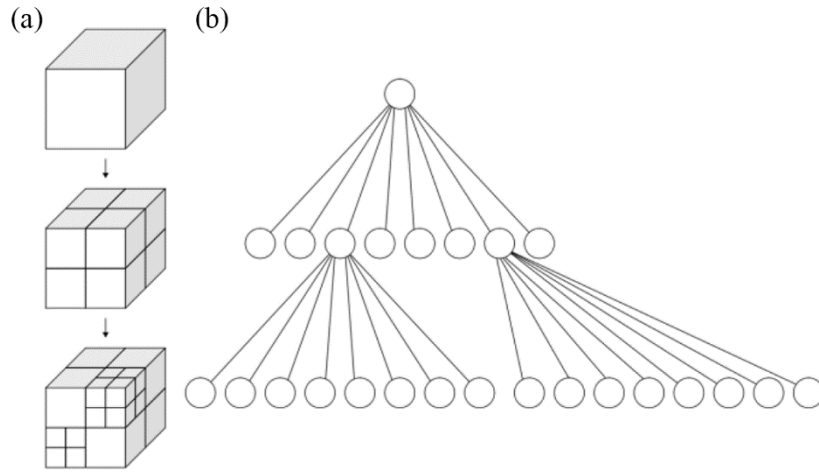
$$RMS = \sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2} \quad (2.2)$$

2.2.1.3 Nokta İnceltme

Veri inceltme, çoklu çözünürlükteki bir yüzey yaklaşımı için nokta örneklerinin veri hiyerarşisini ortaya çıkaran tekrarlamalı bir nokta kaldırma algoritmasına dayanmaktadır. İnceltme algoritması, yerel komşu noktaların önemini ölçen ve her adımda en az önemli bir noktayı kaldıran bir nokta kaldırma kriteri ile çalışır [43]. Veri inceltme, ortalama yüzeyden yüksek sapmaya sahip noktaların silmesi sebebiyle filtreleme ile benzer bir etkiye sahiptir. Farklı istasyonlardan yapılan tarama işlemlerinde veri inceltme işlemi birleştirme işleminden sonra yapılmalıdır. Verinin çok sayıda nokta içermesi ve lazer ışınının tarayıcıya olan uzaklığının ve

açısının her istasyon için farklı olması nedeniyle bindirilen alanlarda nokta yoğunluğu farklı olacaktır [44].

Voksel grid alt-örnekleme yönteminde, tüm bir nokta bulutunu temsil etmek için kullanılan temel veri yapılarından biri octree'dir. Bir octree nokta bulutunu yinelemeli olarak voksel adı verilen sekiz küçük bölüme ayırır (Şekil 2.7). Bu vokseller, nokta bulutundan bir nokta alt kümesini içine alan 3B küplerdir. Girdi bulutunun üzerine bir 3B voksel ızgarası oluşturur. Bu, tüm nokta bulutunun üzerine bir dizi küp yerleştirildiği anlamına gelir. Çerçevenin çözünürlüğü octree'nin derinlik seviyesi ile belirlenir, yani daha yüksek bir çözünürlük daha fazla voksel ve daha derin bir octree anlamına gelir. Her bir voksel içindeki noktalar, merkezlerine göre alt-örneklenir [28].

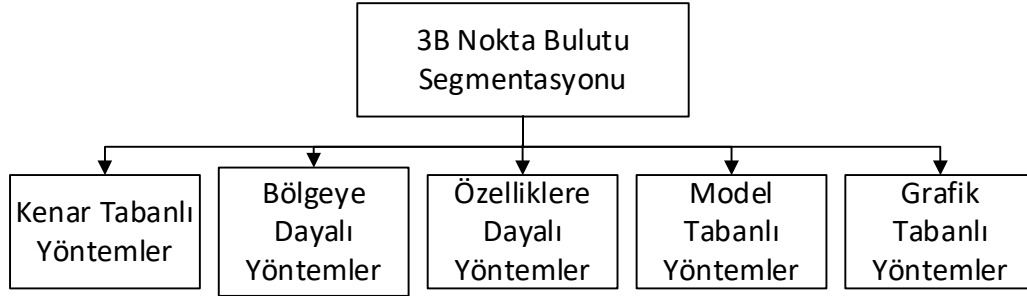


Şekil 2.7 (a) Bir voksel ve (b) bu voksele karşılık gelen octree'nin gösterimi [28]

2.2.2 Segmentasyon

3B nokta bulutlarının işlenmesinde segmentasyon önemli işlem adımlarından biridir. Segmentasyon işleminin amacı, benzer karakteristik özelliklere sahip noktaları homojen bölgelere ayırarak kümelemektir. Bu izole bölgeler anlamlı olmalıdır. Segmentasyon işlemi, alandaki nesnelerin konumlanması ve tanımlanması, sınıflandırılması ve nesne çıkarımı gibi çeşitli açılardan analiz etmek için yardımcı olmaktadır.

Segmentasyon yöntemleri genel olarak; kenar tabanlı yöntemler, bölgeye dayalı yöntemler, özelliklere dayalı yöntemler, model tabanlı yöntemler ve grafik tabanlı yöntemler olmak üzere beş gruba ayrılmaktadır (Şekil 2.8) [45].



Şekil 2.8 Segmentasyon Yöntemleri [45]

2.2.2.1 Kenar Tabanlı Yöntemler

Kenarlar, nesnelerin şekli ile ilgili karakteristik özellikleri olarak tanımlanmaktadır. Kenar tabanlı yöntemlerle, segmentlerine ayrılmış bölgeler elde etmek için nokta bulutlardaki çeşitli bölgelerin sınırlarını algılanır. Bu yöntemlerin prensibi farklı yansıma değerine sahip noktaları bulmaktır [45].

Her ne kadar kenar tabanlı yöntemler hızlı olsa da doğruluk problemleri vardır. Çünkü bu yöntemlerin hepsi, genellikle nokta bulutu verilerinde meydana gelen gürültüye ve nokta bulutlarının eşit olmayan yoğunluğuna karşı çok hassastır [45].

2.2.2.2 Bölgeye Dayalı Yöntemler

Bölgeye dayalı yöntemler yakın noktaları birleştirerek izole bölgeler elde etmek ve bunun sonucunda farklı bölgeleri ayırmak bulmak için komşuluk bilgisini kullanmaktadır. Bölgeye dayalı yöntemler, gürültüye karşı kenar tabanlı yöntemlerden daha hassastır. Bölgeye dayalı yöntemler çekirdekli bölge ve çekirdeksiz bölge yöntemleri olmak üzere iki kategoriye ayrılmaktadır.

Çekirdekli bölge yöntemlerinde, çekirdek noktası sayısının seçilmesi ile segmentasyon süreci başlatılır, daha sonra bu noktalarda, belirli kriterler veya uygun eşik değerleri sağlandığında her bölge, komşu noktaların eklenmesi ile büyütülmektedir.

Çekirdekli bölge yaklaşımları, seçilen çekirdek noktalarına büyük ölçüde bağımlıdır. Yanlış çekirdek noktalarının seçimi, segmentasyon sürecini etkiler ve segment

olarak ayrılacak bölgenin eksik veya fazla kümelenmesine sebep olabilmektedir. Çekirdek noktalarının seçilmesi ve büyüme sürecinin kontrol edilmesi oldukça zaman alan işlemlerdir. Diğer bir zorluk ise yöntemin gürültüye duyarlı olmasından dolayı, yerel olarak tamamlanan işlemde noktanın verilen bölgeye eklenip eklenmeyeceğine karar vermektir.

Çekirdeksiz bölge yöntemleri, çekirdekli bölge yöntemlerinin tam tersidir, “yukarıdan aşağıya” yaklaşımı temel almaktadır. İlk olarak, tüm noktalar bir bölgeye ayrılmaktadır. Daha sonra alt bölme işlemi ile bölge daha küçük bölgelere bölünmektedir. Başarım ölçüsü, eşik değerden daha yüksek olduğu sürece alt bölümlere ayırma işlemine devam edilmektedir.

Çekirdeksiz bölge yöntemlerinin başlıca zorluğu bölgelerin nereye ve nasıl bölüneceklerine karar vermektir. Bu yöntemlerin bir diğer kısıtlaması ise karmaşık alanlarda genellikle bilinmeyen ön bilgi (nesne modelleri, bölge sayısı vb.) gerektirmesidir [45].

2.2.2.3 Özelliklere Dayalı Yöntemler

Bu yöntemlerde, ilk olarak her noktanın geometrik ve radyometrik karakteristikleri esas alınarak özellikler tanımlanır. Bu özellikler genellikle bir noktanın konumu, yerel olarak tahmin edilen yüzey normali, en uygun yüzey artık değerleri ve nokta bulutu verisinin yansıma değerleridir. Her noktanın n özelliğini eşleştirmek için n -boyutlu bir özellik alanı oluşturulur. Daha sonra, özellik alanında kümeler tanımlanır. Her bir kümeye ait noktalar, nesne alanında ayrı bir segment olarak etiketlenir [46].

Özelliklere dayalı yöntemler, nokta bulutu verilerinin özelliklerinin kümelenmesine dayanan güçlü yaklaşımlardır [45]. Bu tekniğin sonucu özellik alanını bölmede kullanılan yöntemler kadar seçilen özelliklere ve bu özelliklerin üretilme yöntemlerine de bağlıdır. Her bir noktanın özellikleri genellikle yerel komşu noktalar kullanılarak tanımlandığı için, bu segmentasyon yöntemi de verideki gürültüye karşı hassastır ve komşuluk tanımından etkilenir [46]. Nokta bulutu verilerinin özellikleri, farklı sınıflar arasında en iyi ayrılmayı sağlamak için tam

olarak hesaplanmalıdır. Bu yöntemlerin bir başka sınırlaması ise, çok sayıda girdi noktası için çok boyutlu özelliklerle uğraşırken zaman alıcı olmalarıdır [45].

2.2.2.4 Model Tabanlı Yöntemler

Model tabanlı yöntemler, noktaların gruplanması için temel geometrik şekilleri (küre, koni, düzlem ve silindir gibi) kullanmaktadır. Bu yöntemler tamamen matematiksel bir ilkeye sahiptir. Aynı matematiksel gösterime sahip olan noktalar aynı segment içerisine kümelenmektedir. Kaba hataların varlığında bile hızlı ve güçlü algoritmalarıdır. Bu yaklaşımların kısıtlamaları, farklı nokta bulutu kaynaklarıyla işlem yapılırken hata yapmalarıdır [45].

- **RANSAC Algoritması**

Tablo 2.1 RANSAC algoritması [47]

Algoritma 1 RANSAC	
1	Model parametrelerinin belirlenmesi için gerekli minimum sayıda nokta seç
2	Model parametrelerini hesapla
3	Tüm noktalar kümesinden kaç tane noktanın önceden tanımlanmış toleransla uyduğunu belirle
4	Eğer seçilen modele uygun nokta sayısının, kümedeki toplam sayı noktalarının üzerindeki payı önceden tanımlanmış bir eşiği geçerse, tanımlanmış tüm modele uygun noktaları kullanarak model parametrelerini yeniden hesapla ve sonlandır
5	Aksi takdirde, 1 ile 4 arasındaki adımları tekrarla (en fazla N kez)

RANSAC algoritması literatürde yaygın olarak kullanılan model tabanlı bir yöntemdir. RANSAC doğrular, daireler vb. gibi matematiksel özellikleri belirleyen güçlü bir modeldir [45]. İlk olarak Fischler ve Bolles tarafından 1981 yılında tasarlanmıştır [48]. Temel algoritma Tablo 2.1'deki gibi özetlenebilir [47].

İterasyon sayısı N , rastgele örneklem kümelerinden en az birinin bir kaba hata içermemesi olasılığını p (genellikle 0.99'a ayarlanır) sağlayacak kadar yüksek seçilmelidir. u , seçilen herhangi bir veri noktasının bir modele uygun olma olasılığını belirtirse, $v=1-u$ bir kaba hata gözlemlene olasılığı olarak hesaplanır. Model için gerekli olan minimum nokta sayısı m 'nin N iterasyonu,

$$1 - p = (1 - u^m)^N \quad (2.3)$$

şeklinde ifade edilmektedir ve bu da (2.4)'te verilen bağıntıdaki gibi çözümlenebilmektedir [47].

$$N = \frac{\log(1 - p)}{\log(1 - (1 - v)^m)} \quad (2.4)$$

2.2.2.5 Grafik Tabanlı Yöntemler

Grafik tabanlı yöntemler nokta bulutlarını grafik olarak ele alır. Bu yöntemde basit bir modelde her köşe, verideki bir noktaya karşılık gelir ve kenarlar belirli komşu nokta çiftlerine bağlanır. Grafik tabanlı yöntemlerin doğruluğu yüksektir ve özellikle robotik uygulamalarda verimliliği sebebiyle popülerlik kazanmıştır.

Diğer yöntemlerle karşılaştırıldığında grafik tabanlı yöntemler, karmaşık alanlardaki nokta bulutu verilerinde, gürültü veya düzensiz yoğunluğa sahip nokta bulutlarında daha iyi sonuçlarla segmentlere ayırabilmektedir. Ancak, bu yöntemler genellikle gerçek zamanlı olarak çalıştırılmamaktadır [45].

Nesne Yönelimli Programlama kavramları, ilk kez 1960'ların başlarında Oslo'daki Norveç Bilgisayar Merkezinde, Ole-Johan Dahl ve Kristen Nygaard tarafından tanıtılmıştır. Nesneler, sınıflar, kalıtım vb. gibi NYP kavramlarının kullanıldığı, SIMULA adlı ayırık olay simülasyonu için bir programlama dili geliştirmişlerdir. Smalltalk, LISP, Object Pascal ve C ++ gibi daha sonra geliştirilen birçok NYP dili, SIMULA dilini temel almaktadır. NYP, zamanla büyük yazılım sistemleri için vazgeçilmez bir programlama yöntemi haline gelmiştir [49].

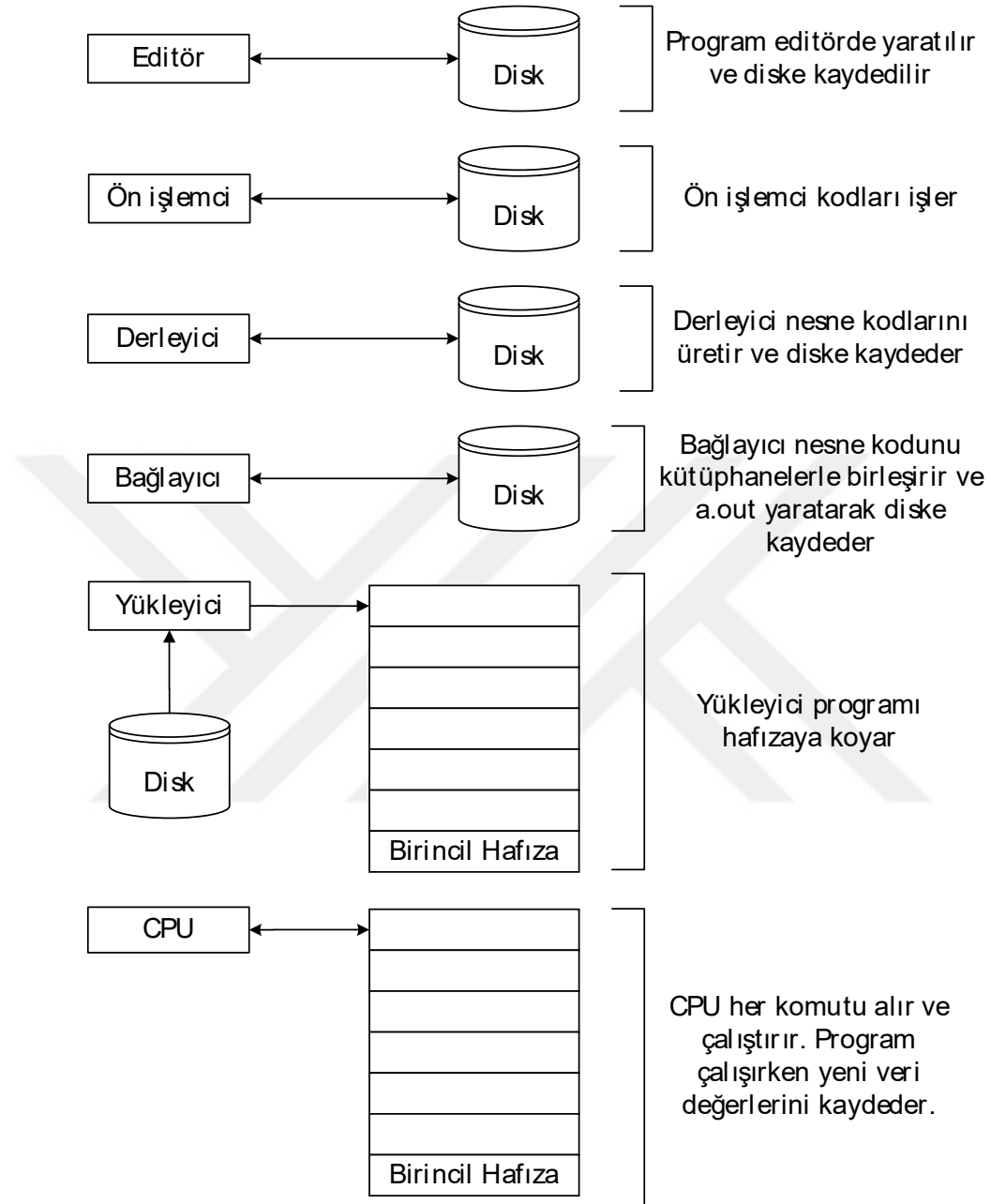
NYP dillerinden önce programlama dilleri (FORTRAN, PASCAL, BASIC ve C gibi), nesneler yerine eylemlere odaklanmaktadır. Bu durum, program yazmayı yavaş hale getirmektedir. Java, C++ ve birçok NYP dilleri sayesinde programcılar nesnelerle uğraşırlar. Bu şekilde yapılan programlamada, programcılar dünyayı gördükleri biçimde yazmaktadırlar. Bu da yordamsal (procedural) programlamadan daha kolay bir yoldur ve verimliliğin artmasında önemli katkı sağlamaktadır [50].

Nesne teknolojisi ile, yazılım girdileri (nesneler) oluşturulmakta ve nesneler iyi tasarlanmış olursa sonraki projelerde yeniden kullanılmaları mümkün hale gelmektedir. MFC ya da Roque Wave ve diğer yazılım geliştirme organizasyonlarının kütüphanelerindeki yeniden kullanılabilir parçaları kullanmak, belli sistemleri oluşturmakta harcanacak gücü önemli derecede azaltmaktadır [50].

3.1 C++ Programlama Dili

C ++, hem ticari hem de akademik amaçlar için yaygın olarak kullanılan popüler bir yazılım geliştirme dilidir. Bjarne Stroustrup tarafından, 1980 yılında Bell Laboratuvarlarında geliştirilmiştir. C++, C diline dayanmaktadır ve C özelliklerine ek olarak, aynı zamanda NYP'yi de destekler. Java dilinden farklı olarak C ++, çoklu kalıtım ve operatör aşırı yüklemesini destekler [24]. Ayrıca, geometri, üçgen

modeller ve doğrusal cebir paketleri gibi, sonlu elemanlar yazılımı geliştirmek için gereken kütüphanelerin çoğu genellikle C ++ dilinde yazılmıştır [49].



Şekil 3.1 Tipik bir C ortamı [50]

C sistemleri genellikle birkaç kısımdan oluşmaktadır: entegre geliştirme ortamı, dilin kendisi ve C standart kütüphanesi. Şekil 3.1’de C geliştirme ortamı açıklanmıştır [50].

Genellikle bir C programı, çalışmadan önce düzenleme (edit), ön işleme (preprocess), derleme (compile), bağlama (link), yükleme (load) ve çalışma (execute) olmak üzere altı aşamadan geçer [50].

Düzenleme aşaması, programın yazım aşamasıdır. Bunun için bir editör program kullanılmaktadır. Borland C++ ve Microsoft Visual Studio gibi entegre geliştirme ortamları yazılım paketlerinde, editörler paketin içerisinde [50].

Program düzenlendikten sonra, kullanıcı tarafından programa “derle” komutu verilir. Derleyici (compiler), C programını makine diline çevirir. C sistemleri, ön işlemci komutları (preprocessor directives) denilen özel komutlara uyar. Bu komutlar, program derlenmeden önce program üzerinde çeşitli işlemlerin yapılmasını sağlar. Bu işlemler genellikle, derlenecek dosyanın içine başka dosyalar katmak ve özel sembolleri program metninin içine yerleştirmek için yapılır [50].

Bir sonraki aşama, bağlama (link) adımıdır. C programları genelde kütüphanelerde ya da belirli bir proje üzerinde çalışan programcılarının oluşturduğu, özel kütüphanelerde tanımlanmış fonksiyonlar içerebilir. C derleyicisi tarafından oluşturulan makine kodunda bu kısımlar boş bırakılır. Bağlayıcı (linker), makine koduyla kütüphanelerde tanımlanmış fonksiyonları birleştirme işlemini yaparak, tamamlanmış çalıştırılabilir programı oluşturur [50].

Ardından, yükleme (loading) işlemi gerçekleştirilir. Yükleme, programın çalıştırılabilmesi için hafızaya yerleştirilmesini sağlar. Bu işlem, yükleyici (loader) tarafından gerçekleştirilir. Yükleyici, programın çalıştırılabilir biçimini diskten alıp hafızaya yerleştirir. Programı destekleyen bazı özel bileşenler, ortak kütüphanelerden alınarak programla birlikte yüklenir [50].

Son olarak CPU kontrolü altında, her komut teker teker çalıştırılır. Programlar, her zaman ilk denemede çalıştırılamayabilir. Bu durumda, sıralanan aşamalardan herhangi birinde çeşitli hatalar olmuş olabilir. Bu durum, bilgisayarın bir hata mesajı yazdırmasına sebep olur. Programcı, yazım aşamasına geri dönüp, gereken düzeltmeleri yapıp, geri kalan aşamaları yeniden gerçekleştirerek programı yeniden çalıştırmalıdır [50].

3.1.1 C++ Programlama Dili Yapısı

C++ programlama dili, C dili gibi iki temel bölümden oluşmaktadır. Bunlar;

- Ön işlemci direktifleri,
- Ana fonksiyon ve yardımcı fonksiyonlardır [51].

3.1.1.1 Ön işlemci Direktifleri

C++ programlarının kendi derleyicileriyle olan ilişkisi, C++ ön işlemcisi ile sağlanmaktadır. Çeşitli emirlerden oluşan ön işlemciler, C++ derleyicisinin kaynak kodunu denetlemekte kullanılmaktadır [51].

Ön işlemci direktifleri program içinde “#” işareti ile başlar. En çok kullanılan ön işlemci direktifleri “#include” ve “#define” ile tanımlanmaktadır. Ön işlemci direktifleri “;” işareti ile sonlandırılmaz [51].

#include direktifi bir kaynak dosyasının programa dahil edilmesini sağlar. C++’a dahil edilen dosyalar “Başlık dosyası” olarak adlandırılır. Başlık dosyalarında standart kütüphane fonksiyonları hakkında bilgiler yer almaktadır ve bu fonksiyonları kullanabilmek için söz konusu dosyaları programa dahil etmek gerekir [51].

#define direktifi ise sembolik değişmezlerin tanımlanmasını sağlar. Söz konusu ön işlemci, bir isim ve bir değer ile tanımlanır [51].

3.1.1.2 Fonksiyonlar

Fonksiyonlar, C++ programlama dilinin temel yapı taşlarıdır. Belirli bir adı olan program parçacıklarıdır ve bu isimle çağırılırlar. Fonksiyonlar, bir değer veya bir sonuç üretmek üzere tasarlanırlar. Bu sebeple, bir veri türüne sahip olmaları gerekir. Fonksiyonların genel yapısı şu şekildedir [51];

türü fonksiyon adı (parametre listesi)

{

deyimler

return;

}

Bir C++ programının içerisinde çok sayıda fonksiyon yer alabilir. Ancak mutlaka bir `main()` fonksiyonu bulunmalıdır. Program yürütülmeye başladığında öncelikle, programın içinde bağlanmış bulunan bir başlangıç yordamı çalışır ve bu yordam da `main()` fonksiyonunu çağırır [51].

3.1.1.3 Açıklama Satırları

C++ programının herhangi bir yerine açıklama satırları eklemek için, açıklamanın başına `/**` eklenir. Bu işaretle başlayan ifadeler, C++ derleyicisi tarafından yok sayılır ve herhangi bir işleme tabii tutulmaz. Özellikle uzun C++ programlarında, bölümlerin hangi işlemi yerine getirdiğinin bilinmesi için, satırlar arasına söz konusu kodların hangi amaçla yazıldığını belirten ifadeler belirtmekte yarar vardır [51].

3.1.1.4 Temel Veri Türleri ve Değişkenler

C++'ta kullanılacak olan tüm değişkenler, kullanılmadan önce programa bildirilmeli ve tanıtilmelidir. Bu bildirim esnasında, değişkenin veri türünün de belirlenmesi gerekmektedir. C++'ta yaygın olarak kullanılan veri türleri Tablo 3.1'de verilmiştir [51].

Tablo 3.1 C++'ta kullanılan veri türleri [51]

Tür	Tanım	Kontrol Karakteri	Sınırlar
İŞARETLİ TÜRLER			
<code>int</code>	Tamsayı		-32768 ile +32767
<code>short</code>	Kısa Tamsayı		-32768 ile +32767
<code>long</code>	Uzun Tamsayı	<code>l</code> veya <code>L</code>	-2147483648 ile +2147483647
<code>float</code>	Kayan Ondalıklı Sayı	<code>f</code> veya <code>F</code>	$3.4 \cdot 10^{-38}$ ile $3.4 \cdot 10^{38}$
<code>double</code>	Çift Ondalıklı Sayı		$1.7 \cdot 10^{-308}$ ile $1.7 \cdot 10^{308}$

long double	Uzun Ondalıklı Sayı	$1.2 \cdot 10^{-4932}$ ile $1.2 \cdot 10^{4932}$
char	Karakter	-128 ile +127
İŞARETSİZ TÜRLER		
unsigned int	İşaretsiz Tamsayı	0 ile +4294967295
unsigned short	İşaretsiz Kısa Tamsayı	0 ile +65535
unsigned long	İşaretsiz Uzun Tamsayı	0 ile +4294967295
unsigned char	İşaretsiz Karakter	0 ile +255
DİĞER		
bool	doğru ya da yanlış	doğru için 1, yanlış için 0 değeri alır

3.2 Entegre Geliştirme Ortamı

Entegre geliştirme ortamları veya diğer adıyla tümleşik geliştirme ortamları, yazılım geliştiricilere yazılım geliştirme, bakım ve değiştirme işlemlerinde yardımcı olmak için özel olarak tasarlanmış programlardır. EGO'lar, genellikle bir grafik kullanıcı arayüzü (GUI) oluşturucu, bir kaynak kod editörü, bir derleyici ve bir hata ayıklayıcı olmak üzere dört bileşenden oluşmaktadır. Program geliştirmek için GUI oluşturucu ve kaynak kod editörü; yazılı kaynak kodları nesne kodlarına çevirmek için derleyici; yazılımdaki olası hataları bulmak ve düzeltmek için hata ayıklayıcı kullanılmaktadır. En çok kullanılan EGO'lardan bazıları Microsoft Visual Studio, Oracle Netbeans, Xcode ve Eclipse'dir [49].

3.3 Nokta Bulutu Kütüphanesi

Nokta bulutu kütüphanesi, Mart 2011'de Radu B. Rusu tarafından oluşturulan bağımsız bir girişim projesi olarak ortaya çıkmıştır. Günümüzde, PCL'in temelini

oluřturan ilk algoritmaların bazıları, dünya apındaki birok arařtırmacı tarafından bağımsız olarak geliřtirilmiřtir ve PCL topluluęu tarafından yayınlardan kodlara geirilmiřtir [52].

PCL, BSD lisansına sahip 2B/3B grnt ve nokta bulutu iřleme iin oluřturulmuř cretsiz bir C++ ktphanesidir. PCL; filtreleme, zellik kestirimi, yzey modelleme, birleřtirme, model belirleme ve segmentasyon gibi ok sayıda algoritma iermektedir.

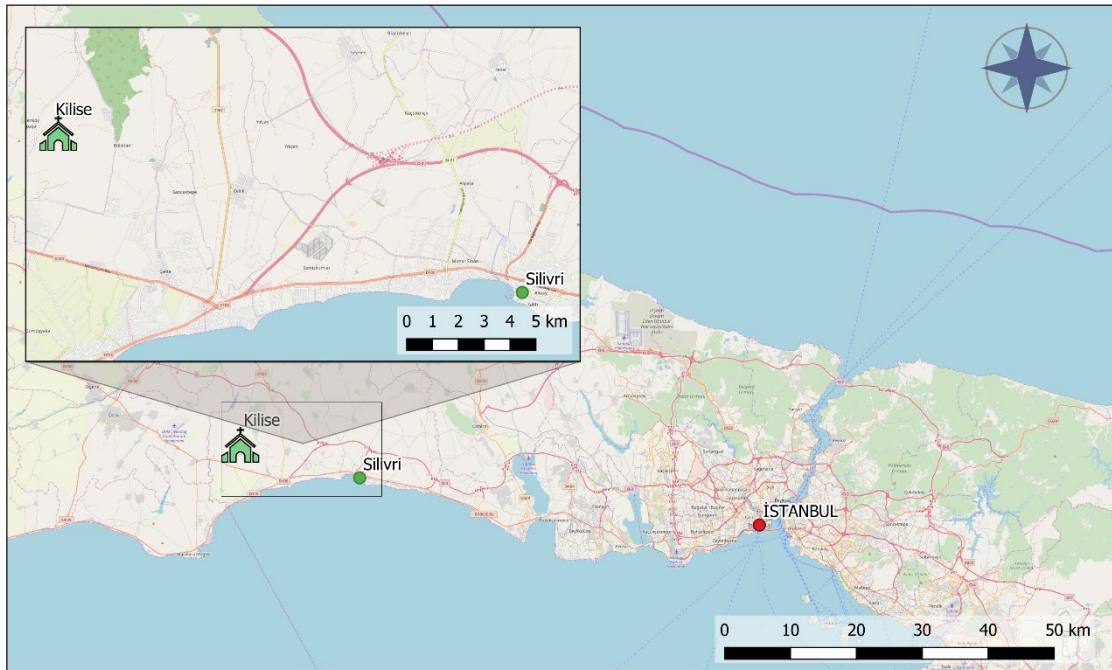
Ktphane ierisinde bulunan matematiksel iřlemlerin biroęu, lineer cebir iin bir aık kaynak ktphane olan Eigen'ı temel alarak uygulanmaktadır. Hızlı k-en yakın komřuluk arama operatr iin ise FLANN ktphanesi temel alınmıřtır [53].



Bu tez çalışması kapsamında, YLT tekniği ile elde edilen nokta bulutu verisinin işlenmesi ve nesne yönelimli programlama ile YLT çalışmalarında kullanılan segmentasyon yöntemlerinin veri setlerine uygulanması araştırılmıştır.

4.1 Çalışma Alanı ve Kullanılan Materyaller

Bu çalışmada kullanılan veri seti, İstanbul'un Silivri ilçesindeki Değirmenköy-Germiyan Rum Ortodoks Kilisesi'nin ön cephesine ait 3B yersel lazer tarayıcı ile elde edilen nokta bulutudur. Çalışma alanının konumu Şekil.4.1'deki haritada gösterilmiştir.



Şekil 4.1 Çalışma alanı

Bu çalışmada, Tablo 4.1'de teknik özellikleri verilen olan Faro Focus 3D 330X yersel lazer tarayıcı ile kilisenin ön cephesine ait 6 dış ve 7 iç olmak üzere; toplam 13 istasyon noktasından tarama işlemi gerçekleştirilmiştir. Elde edilen ham nokta

bulutlarındaki nokta sayıları ve yersel lazer tarayıcının kurulmuş olduğu istasyon noktasının yapıya göre konumu Tablo 4.2’de verilmiştir.

Tablo 4.1 Kullanılan yersel lazer tarayıcının teknik özellikleri [54]

Özellik	Faro Focus 3D 330X
Minimum Ölçüm Mesafesi	0,6 m
Maksimum Ölçüm Mesafesi	330 m
Ölçüm Hızı (nokta/saniye)	122.000 / 244.000 / 488.000 / 976.000
Mesafe Doğruluğu	± 2 mm
Kamera Çözünürlüğü	70 mega piksel
Entegre GPS	Var
Ağırlık	5,2 kg

Çalışmada izlenen işlem adımlarının nokta bulutlarının renklendirilmesi, filtrelenmesi ve birleştirilmesi aşamaları, CloudCompare yazılımının 2.10 Alpha sürümü kullanılarak gerçekleştirilmiştir. Ardından, nokta inceltme aşamasında Voksel grid alt örnekleme; segmentasyon aşamasında ise bölge büyütme, RANSAC ve normallerin farkına dayalı segmentasyon algoritmaları Visual Studio 2017 derleyicisi kullanılarak C++ kodlama dilinde programlanmıştır.

Algoritmaların kodlanmasında PCL’den yararlanılmıştır. Nokta bulutu kütüphanesi PCL 1.9.1’in ilişkilendirilmesi için CMake yazılımı kullanılmıştır.

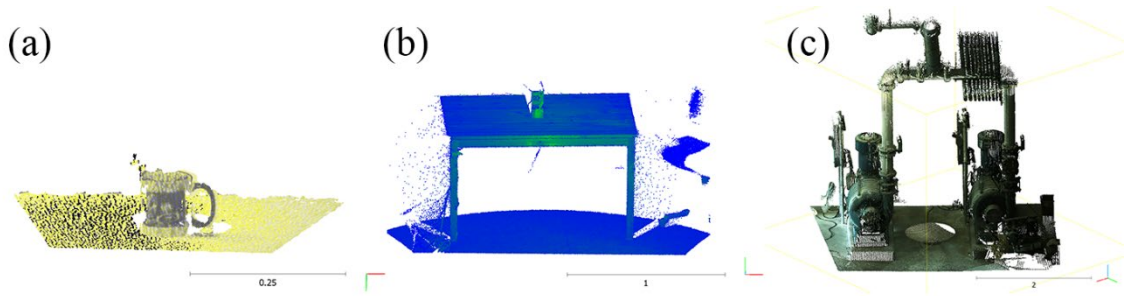
Tablo 4.2 Ham veri setindeki nokta bulutları ve nokta sayıları

Tarama No	Nokta Sayısı (adet)	İstasyon Noktası Konumu
1	97.855.418	Dış
2	90.149.347	
3	76.659.254	
4	83.221.524	
5	90.717.069	
6	99.948.229	
7	136.763.667	İç
8	164.343.695	
9	165.981.754	
10	165.679.563	
11	169.317.938	
12	170.232.007	
13	169.978.688	

Programlanan algoritmaların farklı özellikteki verilerle çalışma prensiplerinin incelenebilmesi amacıyla, Tablo 4.3'te nokta bulutlarında bulunan nokta sayıları ve nokta bulutlarının temin edildiği kaynaklar gösterilmiştir. Verilere ait görüntüler ise Şekil 4.2'de verilmiştir.

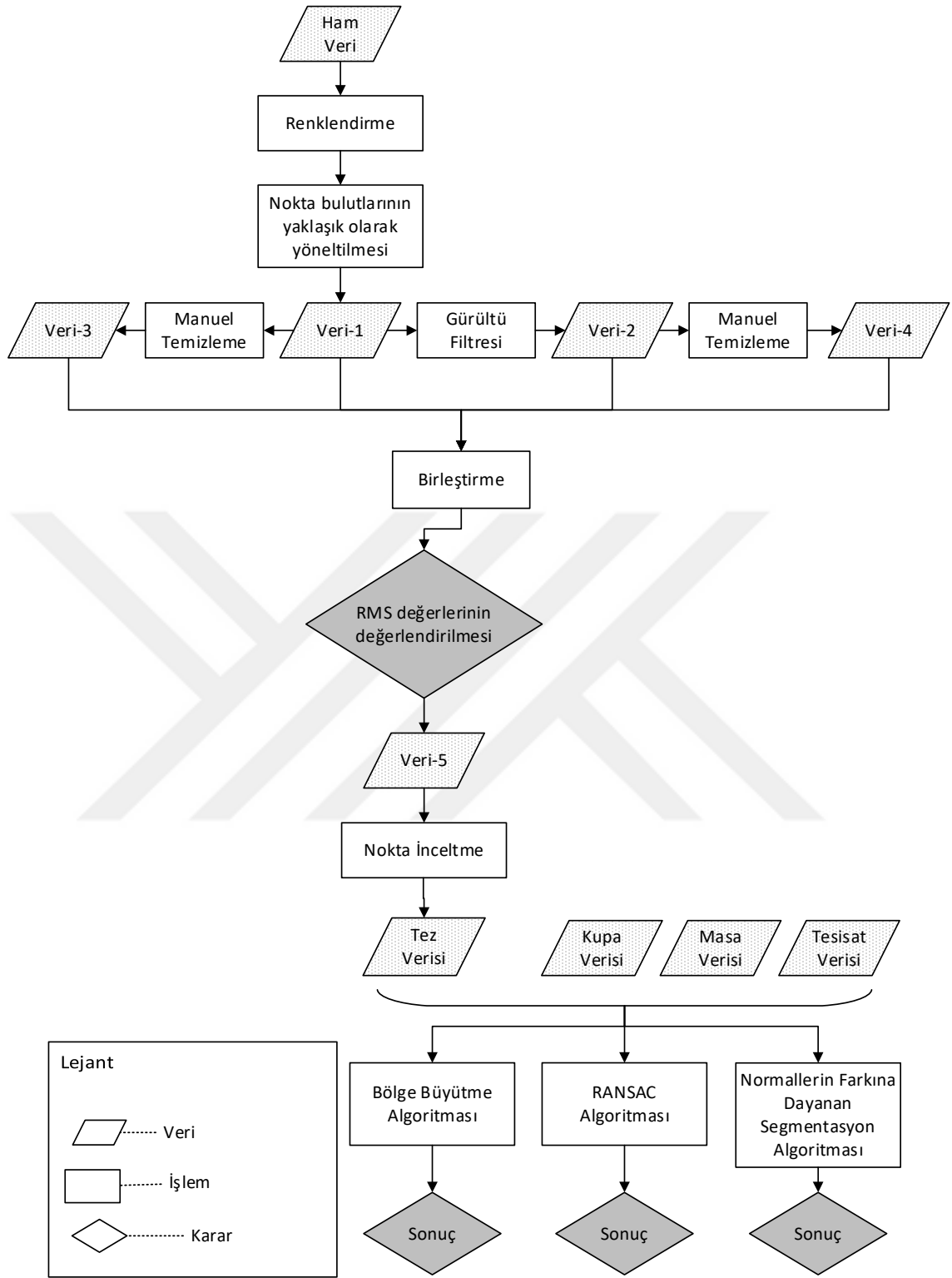
Tablo 4.3 Test verileri ve özellikleri

Nokta Bulutu	Nokta Sayısı (adet)	Veri Kaynağı
Kupa	139.656	PCL [52]
Masa	460.400	PCL [52]
Tesisat	1.213.990	Libe57 [55]

**Şekil 4.2** Veri setleri: (a) Kupa, (b) Masa, (c) Tesisat

4.2 Nokta Bulutu Verisinin işlenmesi

Uygulamada izlenen iş akışı Şekil 4.3'te verilmiştir. Ham verinin renklendirilmesi ve birleştirme işlemi için nokta bulutlarının yaklaşık olarak yöneltilmesi sonucunda Veri-1; Veri-1'e gürültü filtresi uygulanarak Veri-2 ve manuel temizleme yapılarak Veri-3; Veri-2 ise manuel temizlenerek Veri-4'ün oluşturulmuştur. Oluşturulan dört adet veri setinin İEYN yöntemi ile birleştirilmiş ve birleştirme işlemi analiz edilmiştir. Bunun sonucunda en iyi değerler elde edilecek şekilde, nokta bulutu birleştirilerek Veri-5 elde edilmiştir. Ardından, nokta inceltme işlemi ile Veri-5'teki nokta sayısı azaltılmış ve segmentasyon yöntemleri uygulanmıştır.



Şekil 4.3 Uygulama genel iş akışı

4.2.1 Nokta Bulutlarının Renklendirilmesi

Ham nokta bulutları CloudCompare yazılımı kullanılarak RGB renk bandı kombinasyonunda, her noktaya karşılık gelen piksel, RGB renk değerine göre renklendirilmiştir.

4.2.2 Nokta Bulutlarının Temizlenmesi

Nokta bulutlarının temizlenmesi, filtreleme ve manuel temizleme işlemlerini kapsamaktadır. Tez çalışmasının bu adımında birleştirme işleminin doğruluğunun artırılmasına uygun veri işleme stratejisinin belirlenmesi için 4 adet veri seti elde edilmiştir.

Verilerin filtrelenmesinde CloudCompare yazılımının gürültü filtresi kullanılmıştır. Filtreleme işlemi vakit alan bir işlem olduğu için ham veriler filtrelenmeden önce cephe dışındaki alanlar kesilerek nokta sayısı azaltılmıştır. Bu nokta bulutları “Veri-1” olarak adlandırılmıştır.

Bir sonraki adımda, İEYN yöntemi ile nokta bulutlarının birleştirilebilmesi için, nokta bulutu verileri yaklaşık olarak yönlendirilmiş olmalıdır. Bu nedenle kullanılan veri setlerinin ayrı ayrı yönlendirilmesi, objektif bir değerlendirme yapılmasına engel olacağı düşünülerek öncelikle, filtrelenmemiş nokta bulutlarını içeren Veri-1 veri setindeki nokta bulutları, yaklaşık olarak birbirlerine göre yönlendirilmiştir. Veri-2’nin ve Veri-3’ün oluşturulmasında yaklaşık olarak yönlendirilmiş olan Veri-1 kullanılmıştır. Veri-4’ün elde edilmesinde ise Veri-2’den faydalanılmıştır.

Veri-1’in gürültü filtresi ile temizlenmesi ile elde edilen nokta bulutları ise “Veri-2” olarak adlandırılmıştır. Veri-2’nin elde edilmesinde kullanılan gürültü filtresi, nokta bulutunun altında uzanan yüzeye olan uzaklığı dikkate almaktadır. Bu yöntemde, nokta bulutunun her bir noktasının çevresine yerel bir düzlem ayarlanmaktadır. Nokta, bu düzlemden çok uzaktaysa silinmektedir. Düzlem kestirimi için bir yarıçap değeri belirlenerek bir küre oluşturulmakta ve bu kürenin içerisinde kalan noktalar komşu olarak belirlenmektedir. Bu küre minimum 6 nokta içermek zorundadır. Bu komşu noktalardan noktaya olan mesafe belirlenerek eşik değer olarak tanımlanmaktadır. Noktanın yüzeyden uzaklığı, eşik değerden büyükse nokta

silinmektedir. Son olarak veri setindeki hiçbir komşu noktaya sahip olmayan izole noktalar bu adımın devamında kaldırılmaktadır.

Gürültü filtresinde kullanılan kürenin belirlenmesinde, yazılım tarafından her nokta için otomatik olarak belirlenen yarıçap değerleri kullanılmıştır. Buna göre, belirlenen yarıçap değerleri Tablo 4.4'teki gibidir. Gürültüler temizlendikten sonra izole noktalar da veri setinden silinerek temizlenmiştir.

Tablo 4.4 Filtrelemede kullanılan küre yarıçap değerleri

Tarama No	Küre Yarıçapı (mm)
1	9,568
2	10,692
3	16,748
4	17,581
5	12,115
6	9,370
7	6,203
8	5,228
9	5,521
10	5,318
11	6,516
12	5,218
13	7,632

Veri-1 veri setindeki nokta bulutlarının, manuel olarak temizlenmesi ile elde edilen veri seti “Veri-3” olarak adlandırılmıştır. Filtrelenen veri seti olan Veri-2’deki nokta bulutlarının da manuel olarak temizlenmesi ile “Veri-4” elde edilmiştir.

Tablo 4.5’te ham, Veri-1, Veri-2, Veri-3 ve Veri-4 veri setlerindeki nokta bulutlarına ait nokta sayıları gösterilmiştir.

Tablo 4.5 Çalışmada kullanılan nokta bulutlarına ait nokta sayıları

Tarama No	Nokta Sayıları (adet)				
	Ham Veri	Veri-1	Veri-2	Veri-3	Veri-4
1	97.855.418	22.336.038	14.360.915	21.656.969	14.272.279
2	90.149.347	17.098.701	10.922.764	17.007.068	10.873.512
3	76.659.254	7.212.024	4.558.445	7.148.867	4.520.924
4	83.221.524	6.532.577	4.139.891	6.346.550	4.080.294
5	90.717.069	13.945.378	8.884.949	13.299.899	8.728.860
6	99.948.229	22.279.438	14.183.249	22.001.054	14.092.951
7	136.763.667	28.711.626	17.750.137	28.564.597	17.744.091
8	164.343.695	47.088.462	28.340.446	47.041.446	28.262.356
9	165.981.754	47.628.063	28.575.166	47.454.773	28.563.901
10	165.679.563	49.915.337	30.074.399	49.832.325	30.071.697
11	169.317.938	31.154.264	18.920.146	30.951.273	18.907.940
12	170.232.007	50.990.125	30.739.355	50.522.066	30.695.726
13	169.978.688	32.870.988	20.811.625	32.695.840	20.768.424
Toplam	1.680.848.153	377.763.021	232.261.487	374.522.727	231.582.955

4.2.3 Nokta Bulutlarının Birleştirilmesi

Nokta bulutlarının birleştirilmesinde literatürde yaygın olarak kullanılan İEYN yöntemi kullanılmıştır. Bu aşamada, veri setinin doğruluğu en yüksek olacak şekilde birleştirilmesi için, bir önceki aşamada elde edilen veri setlerindeki nokta bulutları ayrı ayrı birleştirilerek sonuçlar karşılaştırılmıştır.

Nokta bulutlarının birleştirilmesinde kullanılan referans nokta bulutları ve hedef nokta bulutları Tablo 4.6’da gösterildiği gibidir. Referans nokta bulutundan rastgele seçilen 100.000 örnek noktanın, hedef nokta bulutundaki eşlenikleri aranmış ve en kısa Öklid mesafesine göre belirlenmiştir. Bu işlem tüm veri setlerindeki her nokta bulutu için 20 iterasyonla sınırlandırılarak birleştirme sağlanmıştır. Her dönüşüm sonucunda elde edilen RMS değerleri Tablo 4.6’da gösterilmiştir.

Tablo 4.6 Birleştirmede kullanılan referans ve hedef nokta bulutlarıyla elde edilen RMS değerleri

Nokta Bulutları		RMS (mm)			
Referans	Hedef	Veri-1	Veri-2	Veri-3	Veri-4
1	2	7,164	7,123	7,260	7,261
2	3	7,616	7,415	7,711	7,908
3	4	30,610	23,779	24,627	27,617
4	5	12,483	7,554	7,423	7,353
5	6	10,741	9,187	8,188	8,613
6	7	35,355	31,185	36,593	31,538
7	8	16,072	13,380	15,074	13,785
8	9	21,405	20,888	21,338	20,713
9	10	16,062	16,808	17,015	16,195

10	11	26,322	26,296	26,208	26,238
8	12	51,574	52,294	49,653	51,736
12	13	27,386	26,161	26,603	26,546

Elde edilen en düşük RMS değerlerine göre, sonraki aşamalarda kullanılmak üzere Veri-5 oluşturulmuştur. Veri-5'te bulunan nokta bulutları, Veri-2'den 1, 2, 3, 4, 7, 8 ve 13; Veri-3'ten 6, 11 ve 12; Veri-4'ten 5, 9 ve 10 numaralı nokta bulutları alınarak oluşturulmuştur. Referans nokta bulutunda rastgele seçilen 100.000 örnek noktanın, hedef nokta bulutundaki eşlenikleri aranmış ve en kısa Öklid mesafesine göre belirlenmiştir. Bu işlem, tüm veri setlerindeki nokta bulutları için 20 iterasyonla sınırlandırılarak nokta bulutları birleştirilmiştir. Her dönüşüm sonucunda elde edilen RMS değerleri Tablo 4.7'de gösterilmiştir.

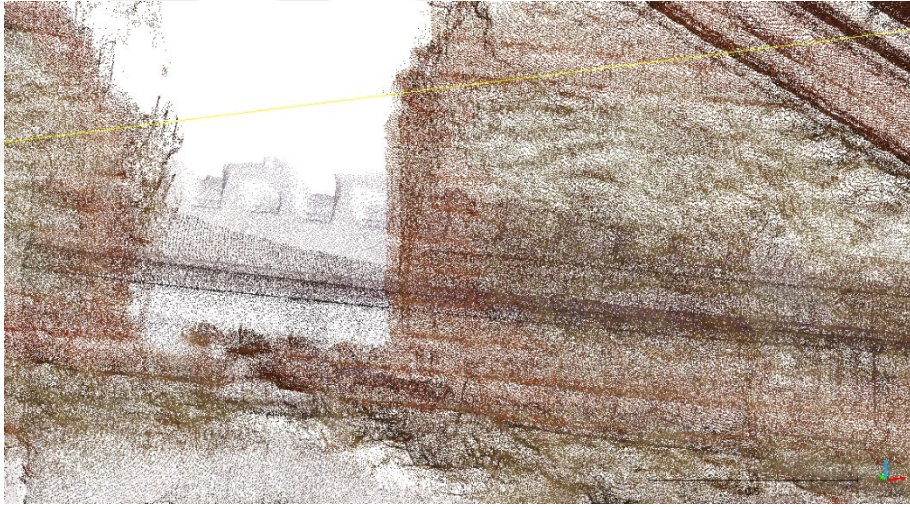
Tablo 4.7 Birleştirme işlemi sonucu elde edilen RMS değerleri

Referans Nokta Bulutu	Hedef Nokta Bulutu	RMS (mm)
1	2	7,0617
2	3	7,521
3	4	24,262
4	5	7,347
5	6	8,621
6	7	31,368
7	8	13,850
8	9	20,696
9	10	16,586

10	11	26,220
8	12	50,602
12	13	14,787

4.2.4 Nokta İnceltme

Çalışmada, farklı iç ve dış tarama istasyonlarından alımı yapılan nokta bulutları birleştirilmiştir. Bu nedenle, her tarama istasyonda, lazer ışınının tarayıcıya olan uzaklığı ve açısının farklı olmasından dolayı, taramaların bindirmeli olduğu alanlarda nokta yoğunluğunun homojen bir dağılıma sahip olmadığı görülmüştür (Şekil 4.4). Bu sebeple, nokta inceltme işlemine ihtiyaç duyulmuştur. Bu adımda, vöksel grid alt-örnekleme yönteminden yararlanılmıştır. Yöntemin uygulanmasında C++ programlama dili ve nokta bulutu kütüphanesinden faydalanılmıştır.



Şekil 4.4 Homojen olmayan nokta dağılımı

Nokta bulutu kütüphanesi kullanılarak yazılan kaynak kod Ek-A'da verilmiştir. Öncelikle kaynak kod, nokta bulutu kütüphanesinden alınan masa verisi ile test edilmiştir. Algoritmaya ait sözde kod Tablo 4.8'de verilmiştir.

Tablo 4.8 Voksel grid alt-örnekleme yöntemine ait sözde kod

Sözde Kod 1 <i>Voksel Grid Alt-Örnekleme</i>	
1	Başla
2	Nokta bulutu verisini oku
3	3B voksel boyutunu ayarla
4	Filtrele
5	Filtrelenmiş veriyi kaydet
6	Bitir

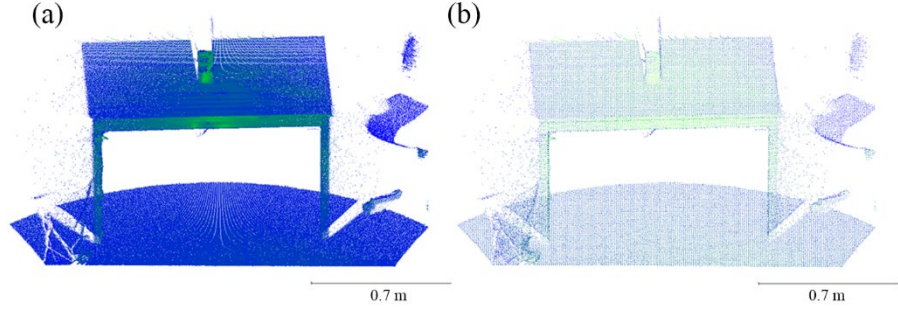
Nokta bulutu kütüphanesinden yararlanılarak voksel grid alt-örnekleme yönteminin uygulanmasında aşağıdaki fonksiyonlar kullanılmaktadır.

setInputCloud (): Kullanılacak nokta bulutunu tanımlar.

setLeafSize (float lx, float ly, float lz): Vokselin boyutunu ayarlar. Buradaki float, birimin ondalık değerde olacağını belirtmektedir. lx, ly ve lz ise sırasıyla vokselin x, y ve z yönündeki boyutlarını belirtir.

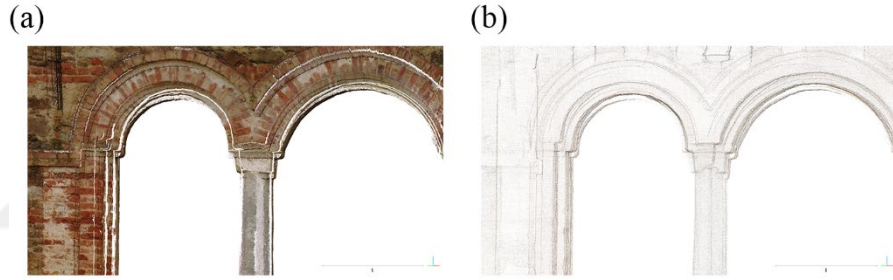
filter (): filtreleme yöntemini çağırır ve filtrelenmiş veri kümesini çıktı ürüne çevirir.

Bir vokselin boyutu x, y ve z yönünde belirlenmektedir. Kaynak koddaki voksel boyutları 1x1x1 cm olacak şekilde belirlenmiştir ve algoritma test edilmiştir. 460.400 noktadan oluşan masa test verisinin nokta inceltme işleminden sonra nokta sayısının 41.049 olduğu tespit edilmiştir. Test verisinin nokta inceltme işlemi öncesi ve sonrasına ait görüntüsü Şekil 4.5'te gösterilmiştir.



Şekil 4.5 Test verisinde nokta inceltme işlemi: (a) öncesi (b) sonrası

Test verisiyle test edilen algoritma, Veri-5'e uygulanmıştır. Veri-5, 69.882.286 noktadan oluşmaktadır. Veri-5 1x1x1 cm'lik vöksel boyutu belirlenerek inceltmiştir. Elde edilen nokta bulutu, "Tez nokta bulutu" olarak adlandırılmış ve 7.633.803 noktadan oluşmaktadır. İnceltile nokta bulutu Şekil 4.6'da gösterilmiştir.



Şekil 4.6 İnceltile tez nokta bulutu: (a) öncesi (b) sonrası

4.2.5 Segmentasyon

Tez çalışmasında bölge tabanlı yöntemlerden bölge büyütme algoritması, model tabanlı yöntemlerden RANSAC algoritması ve özelliklere dayalı yöntemlerden ise yüzey normallerinin farkına dayalı segmentasyon yöntemi, nokta bulutları verilerine uygulanarak değerlendirilmiştir. Bu yöntemlerin değerlendirilmesinde nokta bulutu kütüphanesinden yararlanılmıştır. Belirlenen yöntemlerin değerlendirilmesinde test verileri ve vöksel grid alt-örnekleme yöntemi ile inceltile Veri-5 nokta bulutu verisi kullanılmıştır.

4.2.5.1 Bölge Büyütme Algoritması

Bu çalışmada bölgeye dayalı yöntemlerden çekirdekli bölge büyütme yöntemi uygulanmıştır. Buna göre, kullanılan algoritmaya ait sözde kod Tablo 4.9'daki gibidir.

Tablo 4.9 Bölge büyütme algoritmasına ait sözde kod

Sözde Kod 2 Bölge Büyütme Algoritması	
1	Başla
2	Nokta bulutu verisini oku
3	Arama yöntemini ayarla
4	K aramada kullanılacak nokta sayısını ayarla
5	Normalleri hesapla
6	Kümede bulunacak minimum nokta sayısını ayarla
7	Kümede bulunacak maksimum nokta sayısını ayarla
8	Arama yöntemini ayarla
9	Komşu nokta sayısını ayarla
10	Yumuşatma eşik değerini ayarla
11	Eğrilik eşik değerini ayarla
12	Kümeleri çıkar
13	Nokta bulutunu kaydet
14	Bitir

Bölge büyütme algoritmasının uygulanmasında öncelikle yüzey normallerinin hesaplanması gerekmektedir. Nokta bulutu kütüphanesi kullanılarak, nokta

bulutuna ait yüzey normallerinin hesaplanmasında aşağıda açıklamaları verilmiş olan fonksiyonlar kullanılmıştır.

setSearchMethod (): K en yakın komşunun bulunmasında kullanılacak olan arama yöntemini ayarlar.

setInputCloud (): Kullanılacak nokta bulutunu tanımlar.

setKSearch (): Özellik kestirimi için kullanılacak k en yakın komşu sayısını ayarlar.

setRadiusSearch (): Girilen yarıçap değerine göre tüm komşuları arar.

compute (): Girdi nokta bulutuna uygun olan en küçük kareler düzlemini hesaplar ve tahmin edilen düzlem parametrelerini yüzey eğriliği ile birlikte verir.

Yüzey normallerinin hesaplanmasında kullanılacak olan komşu noktaların aranmasında KSearch veya RadiusSearch yöntemlerinden birisi kullanılmaktadır [31]. Visual Studio ortamında C++ programlama dili ile nokta normalinin hesaplanması Şekil 4.7'deki gibi gerçekleştirilmektedir.

```
pcl::search::Search<pcl::PointXYZ>::Ptr tree(new pcl::search::KdTree<pcl::PointXYZ>);
pcl::PointCloud<pcl::Normal>::Ptr normals(new pcl::PointCloud<pcl::Normal>);
pcl::NormalEstimation<pcl::PointXYZ, pcl::Normal> normal_estimator;
normal_estimator.setSearchMethod(tree);
normal_estimator.setInputCloud(cloud);
normal_estimator.setKSearch(50);
normal_estimator.compute(*normals);
```

Şekil 4.7 PCL kütüphanesi kullanılarak nokta normallerinin hesaplanması

Veri setindeki tüm noktalar için yüzey normalleri hesaplandıktan sonra, bölge büyütme algoritması, PCL kütüphanesindeki bölge büyütme sınıfı “pcl::RegionGrowing” kodu ile başlatılmıştır. Bu sınıfta iki parametre mevcuttur:

PointT: Kullanılan noktanın türü (pcl::PointXYZ)

NormalT: Kullanılan normallerin türü (pcl::Normal)

Bir sonraki adımda minimum ve maksimum küme boyutu ayarlanmıştır. Buna göre, segmentasyon tamamlandığında ayarlanan minimum değerden daha az sayıda (veya ayarlanan maksimum değerden daha fazla) noktaya sahip tüm kümeler veri setinden çıkarılmıştır. Maksimum ve minimum küme için varsayılan değerler 1'dir.

Maksimum ve minimum küme değerlerinin oluşturulmasında kullanılan fonksiyonlar aşağıdaki gibidir.

setMinClusterSize (): Kümenin geçerli sayılabilmesi için içermesi gereken minimum nokta sayısını belirler.

setMaxClusterSize (): Kümenin geçerli sayılabilmesi için içermesi gereken maksimum nokta sayısını belirler.

Bu algoritmanın uygulanmasında, komşu noktaların belirlenmesi için K en yakın komşuluk aramasına ihtiyaç duyulmaktadır. Bu sebeple arama yöntemi ve komşu sayısı ayarlanmıştır. Ardından segmentasyon işlemi yapılacak olan nokta bulutu ve normalleri çağrılmıştır. Komşuluk aramasına ve nokta bulutunun çağrılmasına ait fonksiyonlar, normallerin hesaplanmasında kullanılan fonksiyonlarla aynıdır. Kullanılacak komşu noktaların sayısı ve normallerinin kullanılmasında ise aşağıdaki fonksiyonlardan faydalanılmıştır.

setNumberOfNeighbours (): Komşu sayısının belirlenmesini sağlar.

setInputNormals (): Normalleri belirler.

Bölge büyütme yöntemiyle, segmentasyonun uygulanabilmesi için belirlenmesi gereken iki önemli parametre bulunmaktadır. Bunlar pürüzsüzlük eşik değeri ve eğrilik eşik değeridir. Pürüzsüzlük eşik değeri radyan ölçü biriminde belirlenmiştir. Bu değer nokta normallerinin arasındaki sapma için uygun aralığın belirlenmesinde kullanılmıştır. Nokta normalleri arasındaki standart sapma, pürüzsüzlük eşik değerinden daha az ise aynı kümede olduğu şeklinde belirlenmiştir (yeni nokta -test edilen nokta- kümeye dahil edilmiştir).

İki noktanın normallerinin sapması küçük bir değerse, eğrilikleri arasındaki farklılığın test edilmesi gerekmektedir. Bu değer eğrilik eşik değerinden küçükse, algoritma yeni noktayı kümeye ekleyerek bölgeyi büyütmeye devam eder. Bu iki parametrenin hesaplanmasında kullanılan fonksiyonlar aşağıdaki gibidir.

setSmoothnessThreshold (): Noktaların test edilmesi için kullanılacak pürüzsüzlük eşik değerini belirler.

setCurvatureThreshold (): Noktaların test edilmesi için kullanılacak eğrilik eşik değerini belirler.

extract (): Segmentasyon algoritmasını başlatır ve segmentasyon boyunca elde edilen kümeleri alır.

Visual Studio ortamında C++ dilinde bölge büyütme algoritması ile segmentasyon işleminin programlanması Şekil 4.8'deki gibidir.

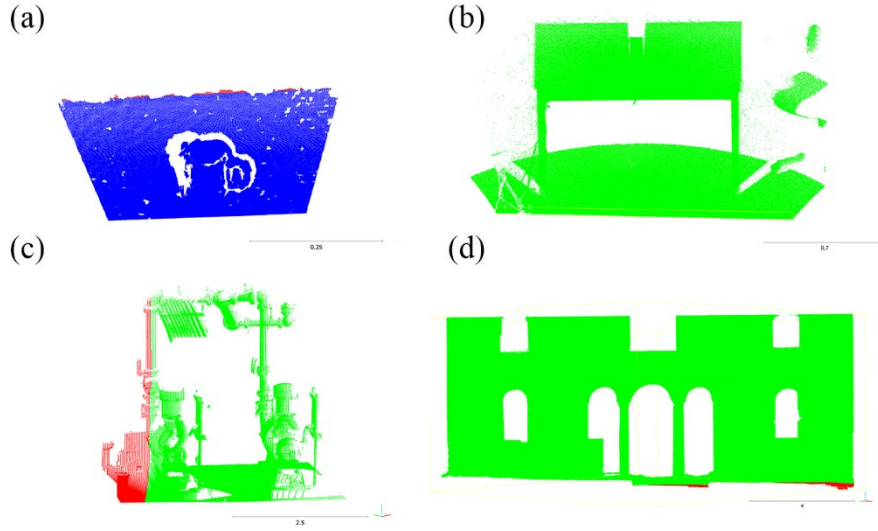
```
pcl::RegionGrowing<pcl::PointXYZ, pcl::Normal> reg;  
reg.setMinClusterSize(50);  
reg.setMaxClusterSize(1000000);  
reg.setSearchMethod(tree);  
reg.setNumberOfNeighbours(30);  
reg.setInputCloud(cloud);  
reg.setInputNormals(normals);  
reg.setSmoothnessThreshold(3.0 / 180.0 * M_PI);  
reg.setCurvatureThreshold(1.0);  
  
std::vector<pcl::PointIndices> clusters;  
reg.extract(clusters);
```

Şekil 4.8 Bölge büyütme algoritmasının programlanması

PCL kütüphanesinden alınan test verisi ile kaynak kod test edilmiştir. Kullanılan kaynak kod Ek-B'de verilmiştir.

Kaynak kodun test edilmesi için kullanılan test verilerine ve tez verisine uygulanan parametreler deneysel olarak belirlenmiştir. Buna göre, veri setleri için belirlenen parametreler Ek-C'de verilmiştir.

Şekil 4.9 Bölge büyütme algoritmasının uygulanması ile segmentlerine ayrılan nokta bulutlarını göstermektedir. Ek-C'de belirtilen parametrelere göre (a) Kupa veri setinin deneme 2'deki, (b) Masa veri setinin deneme 6'daki, (c) Tesisat veri setinin deneme 3'teki ve (d) ise Tez verisinin deneme 3'teki segmentasyondan elde edilen sonuçtur. Sonuç nokta bulutu görüntülerinde, segmentleri gösteren nokta kümeleri farklı renklerde boyanarak gösterilmiştir.



Şekil 4.9 Bölge büyütme algoritması ile segmentasyon: (a) Kupa nokta bulutu (b) Masa nokta bulutu (c) Tesisat nokta bulutu (d) Tez nokta bulutu

4.2.5.2 RANSAC Algoritması

PCL kütüphanesi kullanılarak RANSAC algoritmasıyla çıkarılan modeller aşağıda listelenmiştir.

SACMODEL_PLANE: Düzlem modellerin belirlenmesinde kullanılır.

SACMODEL_LINE: Çizgi modellerin belirlenmesinde kullanılır.

SACMODEL_CIRCLE2D: Bir düzlemdeki 2B dairelerin belirlenmesinde kullanılır.

SACMODEL_SPHERE: Küre modellerin belirlenmesinde kullanılır.

SACMODEL_CYLINDER: Silindir modellerin belirlenmesinde kullanılır.

SACMODEL_CONE: Koni modellerin belirlenmesinde kullanılır.

SACMODEL_PARALLEL_LINE: Maksimum bir açısal sapma içinde, belirli bir eksene paralel bir çizgiyi belirlemek için kullanılan bir modeldir.

SACMODEL_PERPENDICULAR_PLANE: Maksimum belirlenen bir açısal sapma içinde, kullanıcı tarafından belirlenen bir eksene dik bir düzlem belirlemek için kullanılan bir modeldir.

SACMODEL_NORMAL_PLANE: Her bir noktadaki yüzey normalleri, belirlenen maksimum bir açısal sapma içinde, çıktı düzleminin normal yüzeyine paralel olması kısıtlaması kullanılarak düzlem modellerini belirlemek için kullanılan bir modeldir.

SACMODEL_PARALLEL_PLANE: Ek yüzey normal kısıtlamaları kullanarak 3B düzlem segmentasyonu için bir model tanımlar. Düzlem, kullanıcı tarafından belirlenen bir eksene paralel olmalıdır.

Kullanılacak örnek konsensüs kestirim modelinin tanımlanmasında ise aşağıda tanımlanmış olan yöntemler kullanılır.

SAC_RANSAC: Rastgele örneklem konsensüsü

SAC_LMEDS: Karelerin en küçük medyanı

SAC_MSAC: M-kestirici örneklem konsensüsü

SAC_RRANSAC: Rastgele edilmiş RANSAC

SAC_RMSAC: Rastgele edilmiş MSAC

SAC_MLESAC: Maksimum benzerlik kestirim örneklem konsensüsü

SAC_PROSAC: Progresif örneklem konsensüsü

Tablo 4.10 RANSAC algoritmasına ait sözde kod

Sözde Kod 3 *RANSAC*

- 1 **Başla**
- 2 Nokta bulutu verisini oku
- 3 Arama yöntemini ayarla
- 4 K aramada kullanılacak nokta sayısını ayarla
- 5 Normalleri hesapla
- 6 Model türünü düzlem olarak ayarla
- 7 Normal mesafe ağırlığını ayarla
- 8 Kestirim modelini ayarla
- 9 Maksimum iterasyon sayısını ayarla

- 10 Mesafe eşik değerini ayarla
 - 11 Eşik değerler içeisindeki noktaları düzlem olarak segmente et
 - 12 Düzlem olarak kümelenen noktaları kaydet
 - 13 Filtre oluştur
 - 14 Düzlem noktaları filtreleyerek veri setinden çıkar
 - 15 Model türünü silindir olarak ayarla
 - 16 Kestirim modelini ayarla
 - 17 Normal mesafe ağırlığını ayarla
 - 18 Maksimum iterasyon sayısını ayarla
 - 19 Mesafe eşik değerini ayarla
 - 20 Yarıçap limitini ayarla
 - 21 Eşik değerler içerisindeki noktaları silindir olarak segmente et
 - 22 Silindir olarak kümelenen noktaları kaydet
 - 23 **Bitir**
-

Bu çalışmada, RANSAC algoritması, rastgele örneklem konsensüsü kestirim modeli yöntemi ile iki adımda uygulanmıştır. İlk adımda, veri setinden SACMODEL_PLANE modeli ile düzlem yüzeylerin çıkarılması, ikinci adımda ise SACMODEL_CYLINDER modeli ile silindir yüzeylerin çıkarılması üzerine çalışılmıştır. Her iki adımın uygulanmasında nokta normallerinden yararlanılmıştır. Nokta normallerinin hesaplanması 4.2.5.1 Bölge Büyütme Algoritması bölümünde ifade edildiği şekilde gerçekleştirilmiştir. RANSAC algoritmasına ait sözde kod Tablo 4.10'da sunulmuştur.

- **Düzlem Yüzeylerin Çıkarılması**

Nokta normalleri hesaplandıktan sonra, RANSAC algoritması ile düzlemlerin çıkarılmasında kullanılan fonksiyonlar ve açıklamaları aşağıdaki gibidir.

setOptimizeCoefficients (): Parametre gerektiği durumda “true”, gerekmiyorsa “false” olarak ayarlanır.

setModelType (): Kullanılacak model türünü tanımlar.

setNormalDistanceWeight (): Nokta normalleri ve düzlem normalleri arasındaki açılal mesafeyi vermek için bağıl ağırlığı (0 ve 1 arasında) ayarlar.

setMethodType (): Kullanılacak örnek konsensüs yönteminin türünü tanımlar.

setMaxIterations (): Maksimum iterasyon sayısını ayarlar.

setDistanceThreshold (): Model eşik değerine olan mesafeyi tanımlar.

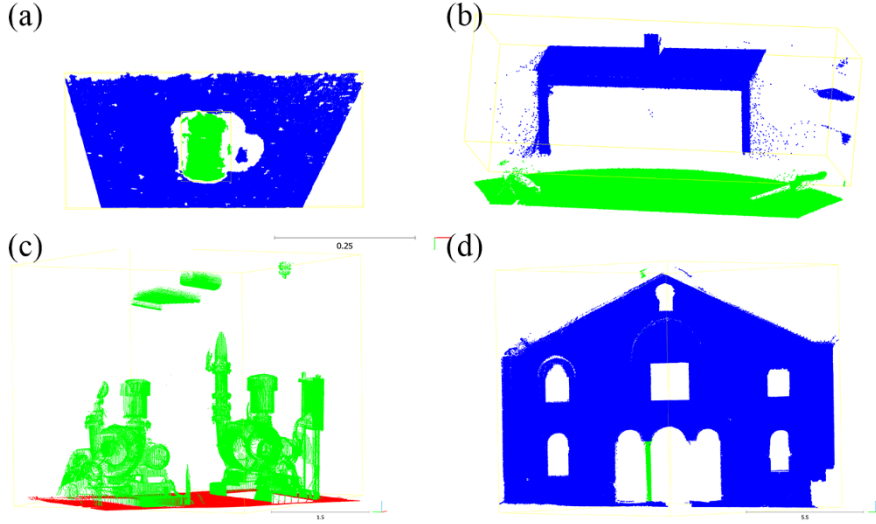
segment (): Verilen nokta bulutundaki bir modelin segmentasyonu için temel yöntemdir. Parametre olarak çıktı ürünler tanımlanır. Bunlar; model noktaları ve model parametreleridir.

Çalışmada, düzlem yüzeylerin çıkarılmasında model türü SACMODEL_PLANE, yöntem türü ise SAC_RANSAC olarak tanımlanmıştır.

- **Silindir Yüzeylerin Çıkarılması**

RANSAC algoritması kullanılarak silindir yüzeylerin çıkarılmasında kullanılan fonksiyonlar düzlem yüzeylerin çıkarılmasında kullanılan fonksiyonlar ile aynıdır. Ancak bu aşamada kullanılacak olan model silindir yüzeylerin çıkarılmasında kullanılan SACMODEL_CYLINDER olarak değiştirilmiştir. Ek olarak silindir modelin çapının oluşturulması için setRadiusLimits() fonksiyonu kullanılmıştır.

Çalışmada kullanılan kaynak kod Ek- D’de verilmiştir. Algoritma PCL kütüphanesinden ve libE57’den elde edilen test verileri ve tez verisi ile Ek-E’de sunulan parametrelerle uygulanmıştır.



Şekil 4.10 RANSAC algoritması ile segmentasyon: (a) Kupa nokta bulutu (b) Masa nokta bulutu (c) Tesisat nokta bulutu (d) Tez nokta bulutu

Şekil 4.10 RANSAC algoritmasının uygulanması ile segmentlerine ayrılan nokta bulutlarını göstermektedir. Ek-E'de belirtilen parametrelere göre (a) Kupa veri setinin deneme 7'deki, (b) Masa veri setinin deneme 6'daki, (c) Tesisat veri setinin deneme 8'deki ve (d) ise Tez verisinin deneme 1'deki segmentasyonunun sonucudur. Sonuç nokta bulutu görüntülerindeki segmentleri gösteren nokta kümeleri farklı renklerde boyanarak gösterilmiştir.

4.2.5.3 Normallerin Farkına Dayalı Segmentasyon Algoritması

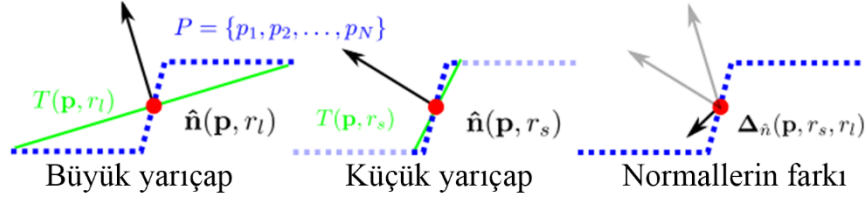
Normallerin farkına dayalı segmentasyon yönteminde, bir P nokta bulutundaki her p noktası için, $r_l > r_s$ olmak üzere $\hat{n}(p, r_l)$ ve $\hat{n}(p, r_s)$ iki ayrı normal kestirilir. Bu noktaların normallerinin normalleştirilmiş (vektör) farkı, operatörü tanımlar. Normallerin farkı operatörü (4.1)'deki şekilde formüle edilmektedir.

$$\Delta \hat{n}(p, r_s, r_l) = \frac{\hat{n}(p, r_s) - \hat{n}(p, r_l)}{2} \quad (4.1)$$

Burada, $r_s, r_l \in \mathbb{R}$ ve $\hat{n}(p, r)$ ise bir p noktasının verilen r yarıçapındaki yüzey normal kestirimidir.

Normallerin farkı temel anlamda, herhangi bir yarıçapta tahmin edilen yüzey normallerinin, yarıçap ölçeğinde yüzeyin temel geometrisini yansıttığı gözlemine dayanmaktadır. Yüzey normallerini tahmin etmenin birçok farklı yöntemi olmasına

rağmen, normaller her zaman bir yarıçap (veya belirli sayıda komşu) ile tahmin edilmektedir. Bu yarıçapla, normalin temsil ettiği yüzey yapısındaki ölçek belirlenir.



Şekil 4.11 Yüzey normallerinde kullanılan farklı yarıçapların etkisi [52]

Şekil 4.11'deki diyagram tek boyutta ölçeğin etkisini göstermektedir. Normaller $\hat{n}$ ve tanjantlar T , r_s küçük ölçek yarıçapı ile kestirildiğinde yüzey yapısından (ve benzer şekilde gürültüden) etkilenmekte; r_l büyük ölçek yarıçapı ile kestirildiğinde ise yüzey yapısından daha az etkilenmekte ve daha büyük ölçekli yüzey yapılarının geometrisini temsil etmektedir.

Segmentasyon işleminde yüzey normallerinin kullanılması, sırasıyla aşağıdaki adımların gerçekleştirilmesi ile sağlanmaktadır:

1. r_l büyük yarıçap kullanılarak her nokta için normallerin kestirilmesi,
2. r_s küçük yarıçap kullanılarak her nokta için normallerin kestirilmesi,
3. Her nokta için normallerin farkının normalize edilmesi,
4. İlgili ölçeğe/alana ait noktaları izole etmek için elde edilen vektör alanının filtrelenmesi.

PCL kütüphanesi kullanılarak algoritmanın kodlanmasında ilk olarak nokta normallerinin hesaplanmıştır. Nokta normallerinin hesaplanmasında, Bölge Büyütme algoritmasında uygulanmasından farklı olarak, komşu noktaların aranmasında "RadiusSearch" yöntemi kullanılmıştır. Algoritmaya ait sözde kod Tablo 4.11'de verilmiştir.

Tablo 4.11 Normallerin Farkına dayalı segmentasyon algoritmasına ait sözde kod

Sözde Kod 4 *Normallerin Farkına dayalı Segmentasyon Algoritması*

- 1 **Başla**
 - 2 Nokta bulutu verisini oku
 - 3 Arama yöntemini ayarla
 - 4 Bakış noktası ayarla
 - 5 Küçük ölçekte normal hesaplamak için yarıçap değeri ayarla
 - 6 Küçük ölçekte normalleri hesapla
 - 7 Büyük ölçekte normal hesaplamak için yarıçap değeri ayarla
 - 8 Büyük ölçekte normalleri hesapla
 - 9 Normal farkı özelliklerini hesapla
 - 10 Filtreleme için mesafe koşulu oluştur
 - 11 Filtreyi uygula
 - 12 Filtrelenmiş veriyi kaydet
 - 13 Küme tolerans değerini ayarla
 - 14 Kümede bulunacak minimum nokta sayısını ayarla
 - 15 Kümede bulunacak maksimum nokta sayısını ayarla
 - 16 Arama yöntemini ayarla
 - 17 Kümelenen nokta indislerini çıkar
 - 18 Kümeleri kaydet
 - 19 **Bitir**
-

Normallerin farkı operatörünün oluşturulmasında kullanılan fonksiyonlar ise aşağıda tanımlandığı gibidir.

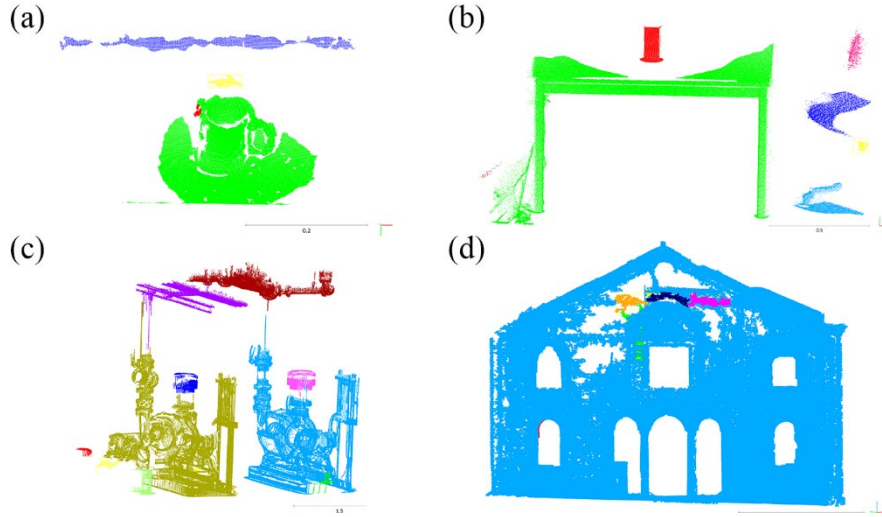
setNormalScaleLarge (): Normallerin farkı operatörü için daha büyük bir arama yarıçapı (ölçek) kullanarak hesaplanan normalleri ayarlar.

setNormalScaleSmall (): Normallerin farkı operatörü için daha küçük bir arama yarıçapı kullanarak hesaplanan normalleri ayarlar.

computeFeature (): Girdi nokta bulutundaki her nokta için normallerin farkı vektörünü hesaplar ve vektör buluta çıktı olarak verir.

Bu algoritmanın uygulanmasında kullanılan kaynak kod Ek-F'de verilmiştir. Kullanılan parametreler; küçük ve büyük ölçek için yarıçap değerleri, büyüklüğe göre filtreleme için büyüklük değeri, tolerans değeri, maksimum ve minimum küme boyutudur. Bu değerler deneysel olarak belirlenmiş olup, test verilerine ve tez verisine uygulanan parametreler Ek-G'de sunulmuştur.

Şekil 4.12 Normallerin farkı algoritmasının uygulanması ile segmentlerine ayrılan nokta bulutlarını göstermektedir. Ek-E'de belirtilen parametrelere göre (a)'da kupa veri setinin deneme 5'teki, (b)'de masa veri setinin deneme 4'teki, (c)'de tesisat veri setinin deneme 9'daki ve (d)'de ise Tez verisinin deneme 10'daki segmentasyonun sonucunda elde edilen nokta bulutlarına ait görüntülerdir. Sonuç nokta bulutu görüntülerindeki segmentleri gösteren nokta kümeleri farklı renklerde boyanarak gösterilmiştir.



Şekil 4.12 Normallerin farkı algoritması ile segmentasyon: (a) Kupa nokta bulutu (b) Masa nokta bulutu (c) Tesisat nokta bulutu (d) Tez nokta bulutu

Bu tez çalışmasında YLT verilerinin işlenmesi ve literatürde bulunan segmentasyon yöntemleri incelenmiştir. Yöntemlerin incelenmesinde PCL ve Libe57'den elde edilen test verileri ve İstanbul'un Silivri ilçesindeki Değirmenköy-Germiyan Rum Ortodoks Kilisesi'nin ön cephesine ait nokta bulutlarından yararlanılmıştır.

Tablo 5.1 Çalışma içerisinde türetilen veri setleri

Veri seti	Yapılan işlem	Nokta Sayısı (adet)
Ham Veri	-	1.680.848.153
Veri-1	Verinin kesilmesi	377.763.021
Veri-2	Verinin kesilmesi, filtreleme	232.261.487
Veri-3	Verinin kesilmesi, manuel temizleme	374.522.727
Veri-4	Verinin kesilmesi, filtreleme, manuel temizleme	231.582.955

YLT verisinin işlenmesinde izlenen adımlar, tez verisine uygulanarak çalışmada sunulmuştur. Farklı noktalardan taraması yapılan YLT verilerinin işlenmesinde uygulanan ilk adım olan birleştirmede literatürde yaygın olarak kullanılan İEYN yöntemi uygulanmıştır. Bu adım için YLT verisi önceden yaklaşık olarak yönlendirilmiştir. Ayrıca ham nokta bulutu verisinden cepheyi gören alanlar kabaca kesilerek çevrede bulunan ağaç, taş vb. nesnelerin veri setinden çıkarılması sağlanmıştır. Yöntemin verimliliğinin incelenebilmesi amacıyla yönlendirilen nokta bulutu, filtrelenmiş, manuel temizlenmiş ve hem filtrelenip hem de manuel temizlenerek 4 veri seti elde edilmiştir. Ham veri seti 1.680.848.153 adet nokta içermekteyken; kabaca kesilen ve yönlendirilen Veri-1 nokta bulutu 377.763.021; Veri-

1'in filtrelenmesi ile elde edilmiş olan Veri-2 nokta bulutu 232.261.487; Veri-1'in manuel temizlenmesi ile elde edilen Veri-3 nokta bulutu 374.522.727 ve son olarak Veri-2 nokta bulutunun manuel temizlenmesi ile oluşturulan Veri-4 nokta bulutu ise 231.582.955 adet nokta içermektedir. Çalışma sırasında elde edilen tüm veri setlerindeki nokta sayıları Tablo 5.1'de verilmiştir. Buna göre, manuel temizleme ile verinin temizlenmesi işleminin hem zaman alıcı olduğu hem de veri setinin tam olarak temizlenemediği gözlemlenmiştir. Sadece filtreleme işlemlerinin uygulanmasının sonucunda ise aşırı gürültü içeren alanlarda, gürültülerin tamamen temizlenemediği görülmüştür.

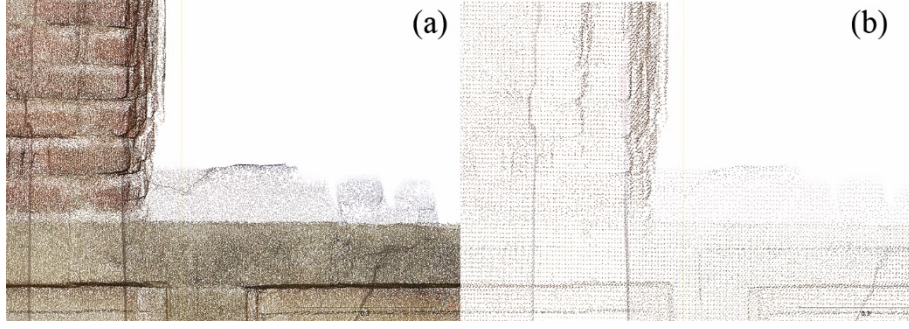
Tablo 5.2 Tüm veri setlerindeki birleştirmelere ait RMS değerlerinin ortalamaları

Veri Seti	RMS değerlerinin ortalaması (mm)
Veri-1	21,90
Veri-2	20,17
Veri-3	20,64
Veri-4	20,46
Veri-5	19,08

Birleştirme adımında incelenen dört veri setinin birleştirilerek elde edilen RMS değerlerinin karşılaştırılması yapılmıştır. Birleştirmede tüm veri setlerindeki nokta bulutları için hedef nokta bulutundan 100.000 örnek nokta seçilerek eşlenikleri aranmıştır. Bu işlem her nokta bulutu için 20 iterasyonla sınırlandırılmıştır. Buna göre, elde edilen RMS değerlerinin ortalamaları Veri-1, Veri-2, Veri-3 ve Veri-4 için sırasıyla 0,02190, 0,02017, 0,02064 ve 0,02046 olarak elde edilmiştir (Tablo 5.2). Belirlendiği üzere, en düşük RMS değerleri Veri-2'den elde edilmiştir. Her birleştirme, ayrı ayrı irdelendiğinde ise Veri-2'ye uygulanan birleştirme işleminin her zaman en düşük RMS değerlerine ulaştırmadığı sonucuna varılmıştır. Bu nokta bulutlarının, Veri-3 ve Veri-4'teki karşılıklarının birleştirilmesinden ise daha düşük RMS değerlerine ulaşıldığı tespit edilmiştir. Bu sebeple, gürültülerin azaltılmasında

filtreleme işleminin yeterli olmadığı sonucuna varılmıştır. Sonraki aşamalarda yüksek doğrulukta sonuçların alınması hedeflendiği için, birleştirilen veri setlerinden elde edilen en düşük RMS değerlerini sağlayan nokta bulutları tespit edilerek, Veri-5 elde edilmiştir. Veri-5'teki tüm nokta bulutlarının birleştirilmesi ile elde edilen RMS değerlerinin ortalaması 0,01908 olarak hesaplanmıştır. Bu işlemlerden elde edilen tüm veri setleri ve RMS değerlerinin ortalamaları Tablo 5.2'de verilmiştir. Elde edilen sonuçlar değerlendirildiğinde ise, birden fazla tarama noktasından ölçümü gerçekleştirilen YLT verilerinin birleştirme işlemine hazırlanmasında, her nokta bulutu verisinin ayrı ayrı ele alınması önerilmektedir.

Farklı istasyonlardan bindirmeli ölçme işlemi yapılan YLT verilerinin birleştirilmesiyle elde edilen nokta bulutunda, kesişim alanlarında nokta yoğunluğunun diğer alanlara oranla daha fazla olduğu tespit edilmiştir. Bu nedenle, sonraki işlemlerde daha homojen bir nokta bulutu ile işlem yapabilmek için voksel grid alt-örnekleme yöntemi ile nokta inceltme işlemi gerçekleştirilmiştir. Bu yöntemin uygulanmasında C++ programlama dili ile PCL kütüphanesinden yararlanılmıştır. Ancak, kütüphane güncel olmadığı için kurulum aşaması oldukça zor ve zaman alıcı olmuştur. Voksel grid alt-örnekleme yönteminin uygulanmasında kullanılan kodun test edilmesinde, PCL kütüphanesinden elde edilen masa veri seti kullanılmıştır. 1x1x1 cm'lik voksel boyutu ile 460.400 adet noktadan oluşan nokta bulutu, seyrelerek 41.049 adet noktaya düşmüştür. Bu yöntemde uygulanan voksel boyutu büyüdükçe, nokta sayısındaki artışın da bir o kadar fazla olduğu tespit edilmiştir. Algoritma Veri-5'e aynı voksel boyutu ile uygulanarak, nokta sayısı 69.882.286'ten, 7.633.803'e düşmüştür. Nokta sayısındaki bu azalma ileride uygulanacak işlemlerin, daha az sayıda nokta ile işlem yapılacağı için işlem süresinin düşürüldüğü düşünülmektedir. Ayrıca, bu sayede daha sonraki aşamalar için noktalar arası mesafenin 1'er cm olduğu homojen bir veri seti elde edilmiştir (Şekil 5.1).



Şekil 5.1 Veri inceltme: (a) öncesi ve (b) sonrası tez verisi

7.633.803 noktadan oluşan Tez verisi, PCL ve libe57 kaynaklarından elde edilen nokta bulutu verileri ile Bölge büyütme, RANSAC ve Normallerin farkı algoritmaları ile segmente edilmiştir. Bu algoritmaların kodlanmasında da PCL kütüphanesinden yararlanılmıştır.

Bölge büyütme algoritmasında belirlenmesi gereken parametreler; kümede bulunan minimum ve maksimum nokta sayısı, komşu noktaların sayısı, yumuşatma eşik değeri ve eğrilik eşik değeridir. Çalışmada bu değerler deneysel olarak belirlenmiştir. Belirlenen parametreler ve bu parametreler sonucunda elde edilen segment sayıları Ek-C’de verilmiştir. Bu algoritmanın çalışmasında belirlenen parametreler incelendiğinde; belirlenen minimum ve maksimum nokta sayısı, segmentasyon için kümelerin boyutunu oluşturacağından bu değerler aralığında bulunmayan kümeler segmente edilememiştir. Belirlenen komşu sayısı artırıldığında, segmentlerin nesneyi daha uygun bir şekilde ifade ettiği ancak işlem süresinin ise uzadığı tespit edilmiştir. Yumuşatma eşik değeri artırıldığında ise, segment sayısının azaldığı görülmüştür. Bu değerlerin pürüzsüz yüzeylerde daha düşük yumuşatma eşik değeri kullanılması gerektiği sonucunu ortaya çıkarmıştır. Eğrilik eşik değerinin ise sonuçlara fazla etkisi olmadığı tespit edilmiştir. Bu algoritma genel olarak değerlendirildiğinde, çalışma prensibi nesne geometrisine dayanmadığı için nesne çıkarmada yeterince başarılı sonuçlar elde edilememiştir.

RANSAC algoritmasının uygulanmasında iki adım izlenmiştir. Bunlardan ilki düzlemlerin çıkarılmasıdır. Düzlemlerin çıkarılmasında belirlenmesi gereken parametreler; Normal mesafe ağırlığı, maksimum iterasyon sayısı ve mesafe eşik değeridir. İkinci adım olan silindirik yüzeylerin çıkarılmasında ise belirlenmesi gereken parametreler; normal mesafe ağırlığı, maksimum iterasyon sayısı, mesafe

eşik değeri ve yarıçap limitidir. Belirlenen parametreler ve bu parametrelerle çıkarılan yüzeylerdeki nokta sayıları Ek-E'deki tabloda gösterilmiştir. Algoritmanın uygulanmasında, özellikle iterasyon sayısı işlem süresini oldukça etkilemektedir. Bu algoritmanın düzgün geometrik şekle ve yüzeye sahip nesnelerin segmentasyonunda oldukça başarılı bir şekilde uygulanabildiği tespit edilmiştir.

Normallerin farkına dayalı segmentasyon algoritmasının uygulanmasında ise belirlenmesi gereken parametreler; iki farklı ölçekte yüzey normallerinin farkının alınabilmesi için büyük ve küçük ölçekte yarıçap değerleri, normallerin farkının büyüklüğü, küme tolerans değeri, minimum ve maksimum nokta sayılarıdır. Bu yöntem daha önce Bölge büyütme ve RANSAC yöntemlerine göre daha fazla işlem süresi gerektirmekte fakat daha anlamlı sonuçlar üretebilmektedir. RANSAC gibi belirli bir nesne geometrisine dayanan çalışma prensibine sahip değildir. Buna rağmen düzgün geometrili nesnelerin segmentasyonunun yapılabildiği tespit edilmiştir.

Çalışma genel olarak değerlendirildiğinde, YLT verisinin işlenmesinin ilk adımı olan ön işleme adımlarında, her nokta bulutu ayrı ayrı ele alınmalıdır. Segmentasyon algoritmalarının başarılı olarak uygulanabilmesi için, ön işleme adımlarının titizlikle yapılması gerekmektedir. Belirlenen segmentasyon algoritmalarının uygulanmasında parametre seçiminin oldukça zorlayıcı fakat uygun parametrelerin belirlenmesi ile verinin segmente edilebilmesinin mümkün olduğu görülmüştür. Ayrıca segmentasyon işleminde kullanılacak yöntemin başarısının, doğrudan veri seti ile ilgili olduğu görülmüştür. Bu sebeple, yapılacak segmentasyon çalışmalarında öncelikle veri seti incelenmeli, veri setinden çıkarılması hedeflenen nesnenin geometrik yapısı değerlendirilmelidir. Buna göre uygulanacak segmentasyon yöntemi belirlenmelidir.

```

#include <iostream>
#include <pcl/io/pcd_io.h>
#include <pcl/point_types.h>
#include <pcl/filters/voxel_grid.h>
int
main(int argc, char** argv)
{
    pcl::PCLPointCloud2::Ptr cloud(new pcl::PCLPointCloud2());
    pcl::PCLPointCloud2::Ptr cloud_filtered(new pcl::PCLPointCloud2());
    // Nokta bulutu verisini aç
    pcl::PCDReader reader;
    reader.read("CLEANED.pcd", *cloud);
    std::cerr << "Nokta bulutu filtrelenmeden önce: " << cloud->width * cloud->height
    << "nokta içermektedir" << pcl::getFieldsList(*cloud) << ").";
    // Voksel grid alt-örnekleme
    pcl::VoxelGrid<pcl::PCLPointCloud2> sor;
    sor.setInputCloud(cloud);
    sor.setLeafSize(0.01f, 0.01f, 0.01f);
    sor.filter(*cloud_filtered);
    std::cerr << "Nokta bulutu filtrelendikten sonra: " << cloud_filtered->width
    * cloud_filtered->height
    << "nokta içermektedir" << pcl::getFieldsList(*cloud_filtered) <<
    ").";
    // Nokta bulutu verisini kaydet
    pcl::PCDWriter writer;
    writer.write("THINNED.pcd", *cloud_filtered,
        Eigen::Vector4f::Zero(), Eigen::Quaternionf::Identity(), false);
    return (0);
}

```

```

#include <iostream>
#include <vector>
#include <pcl/point_types.h>
#include <pcl/io/pcd_io.h>
#include <pcl/search/search.h>
#include <pcl/search/kdtree.h>
#include <pcl/features/normal_3d.h>
#include <pcl/filters/passthrough.h>
#include <pcl/segmentation/region_growing.h>
int
main(int argc, char** argv)
{
    int minSize, maxSize, neigh;
    double smooth, curve;
    minSize = 100;
    maxSize = 100000;
    neigh = 30;
    smooth = 3.0;
    curve = 1.0;
    pcl::PointCloud<pcl::PointXYZ>::Ptr cloud(new
pcl::PointCloud<pcl::PointXYZ>);
    if (pcl::io::loadPCDFile <pcl::PointXYZ>("kupa.pcd", *cloud) == -1)
    {
        std::cout << "Nokta bulutu okunamadı." << std::endl;
        return (-1);
    }
    pcl::search::Search<pcl::PointXYZ>::Ptr tree(new
pcl::search::KdTree<pcl::PointXYZ>);
    pcl::PointCloud <pcl::Normal>::Ptr normals(new pcl::PointCloud
<pcl::Normal>);
    pcl::NormalEstimation<pcl::PointXYZ, pcl::Normal> normal_estimator;
    normal_estimator.setSearchMethod(tree);
    normal_estimator.setInputCloud(cloud);
    normal_estimator.setKSearch(50);
    normal_estimator.compute(*normals);

```

```

pcl::IndicesPtr indices(new std::vector<int>);
pcl::RegionGrowing<pcl::PointXYZ, pcl::Normal> reg;
reg.setMinClusterSize(minSize);
reg.setMaxClusterSize(maxSize);
reg.setSearchMethod(tree);
reg.setNumberOfNeighbours(neigh);
reg.setInputCloud(cloud);
reg.setInputNormals(normals);
reg.setSmoothnessThreshold(smooth / 180.0 * M_PI);
reg.setCurvatureThreshold(curve);
std::vector<pcl::PointIndices>* clusters = new std::vector<pcl::PointIndices>;
reg.extract(*clusters);
std::cout << clusters->size() << '\n';
pcl::PointCloud<pcl::PointXYZ>::Ptr extracloud(new pcl::PointCloud<pcl::PointXYZ>);
std::cout << "Ptr\n";
for (int j = 0; j < clusters->size(); j++)
{
    for (int i = 0; i < clusters->operator[](j).indices.size(); i++)
    {
        extracloud->push_back(cloud->operator[](i));
    }
    std::stringstream ss;
    ss << "cluster_" << j << ".pcd";
    std::cout << ss.str() << '\n';
    pcl::io::savePCDFile(ss.str(), *extracloud);
    extracloud->clear();
    ss.clear();
}
delete clusters;
return (0);
}

```

Bölge Büyütme Algoritmasında Kullanılan Parametreler

Veri	Nokta Sayısı	Deneme No	Minimum Nokta Sayısı	Maksimum Nokta Sayısı	Komşu Sayısı	Yumuşatma Eşik Değeri	Eğrili k Eşik Değeri	Segment Sayısı
Kupa	139.656	1	100	100.000	30	3	1	13
		2	100	150.000	30	3	1	14
		3	100	150.000	30	30	1	4
		4	100	150.000	3	3	1	25
		5	10	150.000	3	3	1	3295
		6	10	150.000	3	30	1	1807
		7	100	150.000	150	3	1	10
		8	100	150.000	30	30	0,5	4
		9	100	150.000	30	150	0,5	3
		10	100	150.000	30	3	10	14
Masa	460.400	1	100	100.000	30	3	1	54
		2	100	400.000	30	3	1	56
		3	1000	500.000	30	3	1	12
		4	1000	500.000	5	3	1	6
		5	1000	500.000	50	3	1	13
		6	500	500.000	30	30	1	2
		7	500	500.000	30	10	1	7

Tesisat	1.213.990	8	5000	500.000	30	3	1	3
		9	500	500.000	30	3	3	19
		10	500	500.000	30	30	1	1
		1	100	100.000	30	3	1	562
		2	1000	1.000.000	30	3	1	98
		3	10.000	1.000.000	30	3	1	8
		4	10.000	100.000	1000	3	1	1
		5	10.000	100.000	100	3	1	15
		6	10.000	100.000	100	1	1	4
		7	1000	100.000	100	3	0,5	92
Tez	7.633.803	8	10.000	1.000.000	30	3	0,5	8
		9	10.000	1.000.000	30	1	1	2
		10	10.000	1.000.000	30	5	1	15
		1	100	100.000	30	3	1	1873
		2	10.000	1.000.000	30	3	1	23
		3	10.000	7.000.000	30	5	1	3
		4	10.000	7.000.000	30	3	1	24
		5	10.000	7.000.000	50	3	1	14
		6	10.000	7.000.000	100	1	1	17
		7	10.000	7.000.000	30	5	0,5	3
		8	10.000	7.000.000	30	3	0,5	25
		9	10.000	7.000.000	30	8	0,5	2

10 1000 7.000.000 30 8 0,5 7



D

RANSAC Algoritması

```
#include <pcl/ModelCoefficients.h>
#include <pcl/io/pcd_io.h>
#include <pcl/point_types.h>
#include <pcl/filters/extract_indices.h>
#include <pcl/features/normal_3d.h>
#include <pcl/sample_consensus/method_types.h>
#include <pcl/sample_consensus/model_types.h>
#include <pcl/segmentation/sac_segmentation.h>
typedef pcl::PointXYZ PointT;
int
main(int argc, char** argv)
{
    // Nesnelerin tanımlanması
    pcl::PCDReader reader;
    pcl::NormalEstimation<PointT, pcl::Normal> ne;
    pcl::SACSegmentationFromNormals<PointT, pcl::Normal> seg;
    pcl::PCDWriter writer;
    pcl::ExtractIndices<PointT> extract;
    pcl::ExtractIndices<pcl::Normal> extract_normals;
    pcl::search::KdTree<PointT>::Ptr tree(new pcl::search::KdTree<PointT>());
    // Veri setleri
    pcl::PointCloud<PointT>::Ptr cloud(new pcl::PointCloud<PointT>);
    /*pcl::PointCloud<PointT>::Ptr cloud_filtered(new
pcl::PointCloud<PointT>);*/
    pcl::PointCloud<pcl::Normal>::Ptr cloud_normals(new
pcl::PointCloud<pcl::Normal>);
    pcl::PointCloud<PointT>::Ptr cloud_filtered2(new pcl::PointCloud<PointT>);
    pcl::PointCloud<pcl::Normal>::Ptr cloud_normals2(new
pcl::PointCloud<pcl::Normal>);
    pcl::ModelCoefficients::Ptr coefficients_plane(new pcl::ModelCoefficients),
coefficients_cylinder(new pcl::ModelCoefficients);
    pcl::PointIndices::Ptr inliers_plane(new pcl::PointIndices),
inliers_cylinder(new pcl::PointIndices);
    // Nokta bulutu verisinin okunması
    reader.read("kupa.pcd", *cloud);
```

```

        std::cerr << "Nokta bulutunda: " << cloud->points.size() << "nokta vardır."
<< std::endl;
        // Estimate point normals
        ne.setSearchMethod(tree);
        ne.setInputCloud(cloud);
        ne.setKSearch(50);
        ne.compute(*cloud_normals);
        // Düzlem model için segmentasyon nesnesinin oluşturulması ve tüm
parametrelerin ayarlanması
        seg.setOptimizeCoefficients(true);
        seg.setModelType(pcl::SACMODEL_NORMAL_PLANE);
        seg.setNormalDistanceWeight(0.1);
        seg.setMethodType(pcl::SAC_RANSAC);
        seg.setMaxIterations(100);
        seg.setDistanceThreshold(0.1);
        seg.setInputCloud(cloud);
        seg.setInputNormals(cloud_normals);
        // Düzlem noktalarının ve katsayılarının elde edilmesi
        seg.segment(*inliers_plane, *coefficients_plane);
        std::cerr << "Düzlem katsayıları: " << *coefficients_plane << std::endl;
        // Girdi nokta bulutundan düzlem noktalarının çıkarılması
        extract.setInputCloud(cloud);
        extract.setIndices(inliers_plane);
        extract.setNegative(false);
        // Düzlem noktalarının diske yazılması
        pcl::PointCloud<PointT>::Ptr cloud_plane(new pcl::PointCloud<PointT>());
        extract.filter(*cloud_plane);
        std::cerr << "Nokta bulutunda düzlemi temsil eden: " << cloud_plane-
>points.size() << " nokta vardır." << std::endl;
        writer.write("kupa_plane.pcd", *cloud_plane, false);
        // Düzlem noktalarının silinmesi, geri kalan noktaların çıkarılması
        extract.setNegative(true);
        extract.filter(*cloud_filtered2);
        extract_normals.setNegative(true);
        extract_normals.setInputCloud(cloud_normals);
        extract_normals.setIndices(inliers_plane);
        extract_normals.filter(*cloud_normals2);
        // Silindir model için segmentasyon nesnesinin oluşturulması ve tüm
parametrelerin ayarlanması
        seg.setOptimizeCoefficients(true);

```

```

seg.setModelType(pcl::SACMODEL_CYLINDER);
seg.setMethodType(pcl::SAC_RANSAC);
seg.setNormalDistanceWeight(0.1);
seg.setMaxIterations(100);
seg.setDistanceThreshold(0.1);
seg.setRadiusLimits(0, 0.1);
seg.setInputCloud(cloud_filtered2);
seg.setInputNormals(cloud_normals2);
// Silindir noktalarının ve katsayılarının elde edilmesi
seg.segment(*inliers_cylinder, *coefficients_cylinder);
std::cerr << "Silindir katsayıları: " << *coefficients_cylinder <<
std::endl;

// Düzlem noktalarının diske yazılması
extract.setInputCloud(cloud_filtered2);
extract.setIndices(inliers_cylinder);
extract.setNegative(false);
pcl::PointCloud<PointT>::Ptr cloud_cylinder(new pcl::PointCloud<PointT>());
extract.filter(*cloud_cylinder);
if (cloud_cylinder->points.empty())
    std::cerr << "Silindir bileşeni bulunamadı." << std::endl;
else
{
    std::cerr << "Nokta bulutunda silindiri temsil eden: " <<
cloud_cylinder->points.size() << " nokta vardır." << std::endl;
    writer.write("kupa_cylinder.pcd", *cloud_cylinder, false);
}
return (0);
}

```

RANSAC Algoritmasında Kullanılan Parametreler

		Düzlem Parametreleri				Silindir Parametreleri						
Veri	Nokta Sayısı	Dene me No	Nor mal Mesa fe Ağırl ığı	İter as yon sayısı	Mes afe eşik değ e ri	Düzle m nokta sayısı	Nor mal Mesa fe Ağırl ığı	İter as yon sayısı	Mes afe eşik değ e ri	Yarıç ap limiti	Silin dir nokt a sayısı	
Kup a	139.656	1	0.1	100	0.1	123079	0.1	100	0.1	0, 0.1	13646	
		2	0.1	10	0.1	123079	0.1	10	0.1	0, 0.1	13646	
		3	0.5	100	0.1	102421	0.5	100	0.1	0, 0.1	7496	
		4	0.01	100	0.1	138794	0.01	100	0.1	0, 0.1	862	
		5	0.05	100	0.1	126509	0.05	100	0.1	0, 0.1	13147	
		6	0.05	1000	0.1	126509	0.05	1000	0.1	0, 0.1	13147	
		7	0.1	100	0.05	121113	0.1	100	0.05	0, 0.1	11472	
		8	0.1	10	0.05	121113	0.1	10	0.05	0, 0.5	11472	
		9	0.1	10	0.05	121113	0.1	10	0.05	0, 0.05	11472	
		10	0.1	10	0.5	139656	0.1	10	0.5	0, 0.5	0	

Mas a	460.40 0	1	0.1	100	0.1	2671 17	0.1	100	0.1	0, 0.1	5790 1
		2	0.1	100	0.15	2793 58	0.1	100	0.15	0, 0.08	0
		3	0.1	100	0.05	2425 66	0.1	100	0.05	0, 0.1	3552 2
		4	0.1	100	0.5	4394 25	0.1	100	0.5	0, 0.1	0
		5	0.5	100	0.5	7766 1	0.5	100	0.5	0, 0.1	0
		6	0.01	100	0.1	2830 38	0.01	100	0.1	0, 0.1	9758 0
		7	0.01	100	0.1	2830 38	0.01	100	0.1	0, 0.05	0
		8	0.01	100	0.1	2830 38	0.01	100	0.1	0, 0.5	1422 27
		9	0.05	100	0.1	2798 37	0.05	100	0.1	0, 0.5	8812 7
		10	0.05	100	0.1	2798 37	0.03	100	0.1	0, 0.5	9968 9
Tesi sat	1.213. 990	1	0.1	100	0.1	2775 14	0.1	100	0.1	0, 0.1	0
		2	0.1	100	0.1	2775 14	0.1	100	0.1	0, 0.5	7026 8
		3	0.1	100	0.1	2775 14	0.1	100	0.1	0, 1	1419 62
		4	0.1	100	0.1	2775 14	0.1	100	0.1	0, 5	9696 6
		5	0.1	100	0.1	2775 14	0.05	100	0.1	0, 5	1306 16
		6	0.1	100	0.1	2775 14	0.00 5	100	0.1	0, 5	2063 68

		7	0.1	100	0.1	2775 14	0.05	100	0.5	0, 5	5535 28
		8	0.05	100	0.1	2919 43	0.05	100	0.5	0, 5	5757 58
		9	0.01	100	0.1	3239 01	0.01	100	0.5	0, 5	6261 45
		10	0.00 5	100	0.1	3281 47	0.00 5	100	0.5	0, 5	6159 93
Tez	7.633. 803	1	0.1	100	0.1	2456 228	0.1	100	0.1	0, 0.1	4889 9
		2	0.05	100	0.1	2529 575	0.05	100	0.1	0, 0.1	6398 6
		3	0.05	100	0.1	2579 575	0.05	100	0.1	0, 0.5	1775 45
		4	0.01	100	0.1	2179 185	0.1	100	0.1	0, 0.5	1654 82
		5	0.1	100	0.1	2456 228	0.1	100	0.1	0, 0.3	9678 4
		6	0.1	100	0.1	2456 228	0.1	100	0.1	0, 0.2	8544 4
		7	0.1	100	0.1	2456 228	0.1	100	0.1	0, 0.25	0
		8	0.1	1000	0.1	2381 788	0.1	1000	0.1	0, 0.05	0
		9	0.1	100	0.1	2456 228	0.1	100	0.1	0,	0
		10	0.1	10	0.1	2456 228	0.1	10	0.1	0, 0.1	1568 0

Normallerin Farkına Dayalı Segmentasyon Algoritması

```
#include <pcl/point_types.h>
#include <pcl/io/pcd_io.h>
#include <pcl/filters/extract_indices.h>
#include <pcl/features/normal_3d.h>
#include <pcl/kdtree/kdtree.h>
#include <pcl/segmentation/extract_clusters.h>
#include <pcl/features/don.h>
#include <pcl/ModelCoefficients.h>
#include <pcl/sample_consensus/method_types.h>
#include <pcl/sample_consensus/model_types.h>
#include <pcl/segmentation/sac_segmentation.h>
#include <string>
#include <pcl/search/organized.h>
#include <pcl/features/normal_3d_omp.h>
#include <pcl/filters/conditional_removal.h>
#include <pcl/segmentation/impl/extract_clusters.hpp>
using namespace pcl;
using namespace std;
int
main(int argc, char** argv)
{
    // Nokta bulutu verisini oku
    pcl::PCDReader reader;
    pcl::PointCloud<pcl::PointXYZ>::Ptr cloud(new
pcl::PointCloud<pcl::PointXYZ>), cloud_f(new pcl::PointCloud<pcl::PointXYZ>);
    reader.read("table_scene_lms400.pcd", *cloud);
    std::cout << "Nokta bulutunda: " << cloud->points.size() << " nokta vardır."
<< std::endl; //
    // Normal kestirim sınıfını oluştur ve girdi veri setini ayarla
    pcl::NormalEstimation<pcl::PointXYZ, PointNormal> ne;
    ne.setInputCloud(cloud);
    // Boş bir kdtree gösterimi oluştur, normal kestirim nesnesinin içine
yerleştir
    // İçeriği, verilen giriş veri setine bağlı olarak (başka bir arama yüzeyi
belirtilmediğinden) nesnenin içinde doldurulacaktır.
```

```

        pcl::search::KdTree<pcl::PointXYZ>::Ptr tree(new
pcl::search::KdTree<pcl::PointXYZ>());
        ne.setSearchMethod(tree);
        /* NOT: bakış noktasının ayarlanması oldukça önemli, bu sayede normallerin
hepsinin aynı yönü işaret ettiğinden emin olunur*/
        ne.setViewPoint(std::numeric_limits<float>::max(),
std::numeric_limits<float>::max(), std::numeric_limits<float>::max());
        // Çıktı veri seti
        std::cout << "Normaller küçük ölçeğe göre hesaplanıyor:" << 0.03 <<
std::endl;

        pcl::PointCloud<PointNormal>::Ptr normals_small_scale(new
pcl::PointCloud<PointNormal>);
        // 3 cm'lik bir küre içerisindeki tüm noktaları kullan
        ne.setRadiusSearch(0.03);
        // Özelliklerin hesaplanması
        ne.compute(*normals_small_scale);
        // Normalleri büyük ölçekte hesapla
        std::cout << "Normaller büyük ölçeğe göre hesaplanıyor:" << 0.10 <<
std::endl;

        pcl::PointCloud<PointNormal>::Ptr normals_large_scale(new
pcl::PointCloud<PointNormal>);
        ne.setRadiusSearch(0.10);
        ne.compute(*normals_large_scale);
        // Normallerin farkı (DoN/Difference of Normals) sonuçları için çıktı bulutu
        oluştur
        PointCloud<PointNormal>::Ptr doncloud(new pcl::PointCloud<PointNormal>);
        copyPointCloud<PointXYZ, PointNormal>(*cloud, *doncloud);
        cout << "Calculating DoN... " << endl;
        // DoN operatorunu oluştur
        pcl::DifferenceOfNormalsEstimation<PointXYZ, PointNormal, PointNormal> don;
        don.setInputCloud(cloud);
        don.setNormalScaleLarge(normals_large_scale);
        don.setNormalScaleSmall(normals_small_scale);
        if (!don.initCompute())
        {
            std::cerr << "Error: Could not initialize DoN feature operator" <<
std::endl;

            exit(EXIT_FAILURE);
        }
        // DoN hesapla

```

```

don.computeFeature(*doncloud);
// DoN özelliklerini kaydet
pcl::PCDWriter writer;
writer.write<PointNormal>("don.pcd", *doncloud, false);
// Büyüklüğe göre filtrele
cout << "don büyüklüğünü filtrele<= " << 0.05 << "..." << endl;
// Filtreleme koşulu oluştur
pcl::ConditionOr<PointNormal>::Ptr range_cond(new
pcl::ConditionOr<PointNormal>()
);
range_cond->addComparison(pcl::FieldComparison<PointNormal>::ConstPtr(
new pcl::FieldComparison<PointNormal>("curvature",
pcl::ComparisonOps::GT, 0.05))
);
// Filtreyi oluştur
pcl::ConditionalRemoval<PointNormal> condrem;
condrem.setCondition(range_cond);
condrem.setInputCloud(doncloud);
pcl::PointCloud<PointNormal>::Ptr doncloud_filtered(new
pcl::PointCloud<PointNormal>);
// Filtreyi uygula
condrem.filter(*doncloud_filtered);
doncloud = doncloud_filtered;
// Filtrelenmiş çıktıyı kaydet
std::cout << "Filtrelenmiş nokta bulutunda: " << doncloud->points.size() <<
" nokta vardır." << std::endl;
writer.write<PointNormal>("don_filtered.pcd", *doncloud, false);
// Büyüklüğe göre filtrele
cout << "Öklid Küme çıkarmayla kümeleniyor. Tolerans değeri <= " << 0.02 <<
"..." << endl;
pcl::search::KdTree<PointNormal>::Ptr segtree(new
pcl::search::KdTree<PointNormal>);
segtree->setInputCloud(doncloud);
std::vector<pcl::PointIndices> cluster_indices;
pcl::EuclideanClusterExtraction<PointNormal> ec;
ec.setClusterTolerance(0.02);
ec.setMinClusterSize(50);
ec.setMaxClusterSize(100000);
ec.setSearchMethod(segtree);
ec.setInputCloud(doncloud);

```

```

        ec.extract(cluster_indices);
        int j = 0;
        for (std::vector<pcl::PointIndices>::const_iterator it =
cluster_indices.begin(); it != cluster_indices.end(); ++it, j++)
        {
            pcl::PointCloud<PointNormal>::Ptr cloud_cluster_don(new
pcl::PointCloud<PointNormal>);
            for (std::vector<int>::const_iterator pit = it->indices.begin(); pit
!= it->indices.end(); ++pit)
            {
                cloud_cluster_don->points.push_back(doncloud->points[*pit]);
            }
            cloud_cluster_don->width = int(cloud_cluster_don->points.size());
            cloud_cluster_don->height = 1;
            cloud_cluster_don->is_dense = true;
            // Kümeyi kaydet
            stringstream ss;
            ss << "don_cluster_" << j << ".pcd";
            writer.write<PointNormal>(ss.str(), *cloud_cluster_don, false);
        }
    }
}

```

Normallerin Farkına Dayalı Segmentasyon Algoritmasında Kullanılan Parametreler

Veri	Nokta sayısı	Dene me no	Küçük ölçek yarı çapı	Büyük ölçek yarı çapı	DoN büyüklü ğü	Küme toleran sı	Minimu m küme boyutu	Maksimu m küme boyutu	Segme nt Sayısı
Kupa	139.656	1	0.03	0.10	0.05	0.02	50	100.000	6
		2	0.05	0.10	0.05	0.02	50	100.000	2
		3	0.01	0.10	0.05	0.02	50	100.000	4
		4	0.01	0.05	0.05	0.02	50	100.000	5
		5	0.03	0.10	0.03	0.02	50	100.000	4
		6	0.03	0.10	0.01	0.02	50	100.000	5
		7	0.03	0.10	0.1	0.02	50	100.000	2
		8	0.03	0.10	0.05	0.1	50	100.000	3
		9	0.03	0.10	0.05	0.1	1000	100.000	2
		10	0.03	0.10	0.03	0.005	1000	100.000	2
Masa	460.400	1	0.03	0.10	0.05	0.02	50	100.000	15
		2	0.03	0.10	0.05	0.02	50	500.000	16
		3	0.05	0.50	0.05	0.02	50	500.000	8
		4	0.05	0.50	0.1	0.02	50	500.000	7
		5	0.05	0.50	0.1	0.5	50	500.000	0
		6	0.05	0.50	0.1	0.1	50	500.000	3

Tesis at	1.213.9 90	7	0.05	0.10	0.1	0.02	50	500.000	11
		8	0.05	0.50	0.1	0.02	500	500.000	5
		9	0.05	0.50	0.1	0.05	500	500.000	3
		10	0.05	0.20	0.1	0.05	500	500.000	2
		1	0.03	0.10	0.05	0.02	50	100.000	72
		2	0.03	0.05	0.05	0.02	50	100.000	123
		3	0.03	0.05	0.05	0.02	1000	100.000	21
		4	0.03	0.05	0.05	0.1	1000	100.000	3
		5	0.03	0.05	0.05	0.1	10.000	100.000	0
		6	0.03	0.05	0.05	0.05	1000	100.000	6
Tez	7.633.8 03	7	0.03	0.05	0.1	0.02	1000	100.000	25
		8	0.03	0.05	0.1	0.02	1000	1.000.00 0	25
		9	0.05	0.1	0.05	0.02	1000	1.000.00 0	12
		10	0.05	0.1	0.05	0.05	1000	1.000.00 0	5
		1	0.03	0.10	0.05	0.02	50	100.000	186
		2	0.03	0.10	0.1	0.02	50	100.000	351
		3	0.03	0.50	0.1	0.02	50	100.000	470
		4	0.03	0.05	0.05	0.02	50	100.000	4362
		5	0.03	0.10	0.05	0.2	500	5.000.00 0	1
		6	0.03	0.10	0.05	0.05	500	5.000.00 0	3

7	0.03	0.10	0.50	0.05	500	5.000.00 0	7
8	0.03	0.10	0.50	0.01	500	5.000.00 0	20
9	0.03	0.10	0.03	0.01	500	5.000.00 0	139
10	0.03	0.10	0.03	0.01	5000	5.000.00 0	7



- [1] R. C. Erenoglu, O. Akcay ve O. Erenoglu, "An UAS-assisted Multi-sensor Approach for 3D Modeling and Reconstruction of Cultural Heritage Site," *Journal of Cultural Heritage*, no. 26, pp. 79-90, 2017.
- [2] L. Cheng, Y. Wang, Y. Chen ve M. Li, "Using Lidar for Digital Documentation of Ancient City Walls," no. 17, pp. 188-193, 2016.
- [3] L. Troung-Hong ve D. F. Laefer, "Octree-Based, Automatic Building Façade Generation from Lidar Data," *Computer-Aided Design*, no. 53, pp. 46-61, 2014.
- [4] Y. Reshetyuk, *Self-Calibration and Direct Georeferencing in Terrestrial Laser Scanning*, Stockholm: Doctoral Thesis, Department of Transport and Economics Division of Geodesy, 2009.
- [5] L. Bornaz ve F. Rinaudo, "Terrestrial Laser Scanner Data Processing," XXth ISPRS Congress, İstanbul, 2004.
- [6] S. J. Buckley, J. A. Howell, H. D. Enge ve T. H. Kurz, "Terrestrial Laser Scanning in Geology: Data Acquisition, Processing and Accuracy Considerations," *Journal of the Geological Society*, no. 165, pp. 625-638, 2008.
- [7] B. Van Genechten, "Theory and Practice on Terrestrial Laser Scanning: Training Material Based on Practical Applications," 2008.
- [8] K. Gümüş, *Yersel Lazer Tarayıcılar ve Konum Doğruluklarının Araştırılması*, İstanbul: Yüksek Lisans Tezi, Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü, 2008.

- [9] V. Poto, J. A. Somogyi, T. Lovas ve A. Barsi, "Laser Scanned Point Clouds to Support Autonomous Vehicles," *Transportation Research Procedia*, no. 27, pp. 531-537, 2017.
- [10] C. Porras-Amores, F. R. Mazarron, I. Canas ve P. V. Saez, "Terrestrial Laser Scanning Digitalization in Underground Constructions," *Journal of Cultural Heritage*, no. In Press, 2019.
- [11] A. Lau, C. Martius, H. Bartholomeus, A. Shenkin, T. Jackson, Y. Malhi, M. Herold ve L. P. Bentley, "Estimating Architecture-Based Metabolic Scaling Exponents of Tropical Trees Using Terrestrial Lidar and 3D Modelling," *Forest Ecology and Management*, no. 439, pp. 132-145, 2019.
- [12] J. Balsa-Barreiro ve D. Fritsch, "Generation of Visually Aesthetic and Detailed 3D Models of Historical Cities by Using Laser Scanning and Digital Photogrammetry," *Digital Applications in Archaeology and Cultural Heritage*, no. 8, pp. 57-64, 2018.
- [13] A. Koroglu, D. Z. Seker, M. S. Kabdasli, H. Karabork, A. Goktepe, E. Varol ve B. Bayram, "Evaluation of Beach Profile Changes by Using Terrestrial Laser Scanner," *Fresenius Environmental Bulletin*, cilt 26, no. 1, pp. 19-28, 2017.
- [14] A. Danchenkov, N. Belov ve Z. Stont, "Using the Terrestrial Laser Scanning Technique for Aeolian Sediment Transport Assessment in The Coastal Zone in Seasonal Scale," *Estuarine, Coastal and Shelf Science*, no. 223, pp. 105-114, 2019.
- [15] H.-b. Li, X.-w. Li, W.-z. Li, S.-l. Zhang ve J.-w. Zhou, "Quantitative Assessment for the Rockfall Hazard in a Post-Earthquake High Rock Slope Using Terrestrial Laser Scanning," *Engineering Geology*, no. 248, pp. 1-13, 2019.

- [16] M. Kincey, C. Gerrard ve J. Warburton, "Quantifying Erosion of 'at Risk' Archaeological Sites Using Repeat Terrestrial Laser Scanning," *Journal of Archaeological Science: Reports*, no. 12, pp. 405-424, 2017.
- [17] N. Yastikli, "Documentation of Cultural Heritage Using Digital Photogrammetry and Laser Scanning," *Journal of Cultural Heritage*, no. 8, pp. 423-427, 2007.
- [18] T. Temizer, G. Nemli, A. Ekizce, S. Demir, B. Bayram, H. İ. Aşkın, A. V. Coban ve H. F. Yilmaz, "3D Documentation of a Historical Monument Using Terrestrial Laser Scanning Case Study: Byzantine Water Cistern, Istanbul," *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, Strasbourg, 2013.
- [19] J. L. Lerma, S. Navarro, M. Cabrelles ve V. Villaverde, "Terrestrial Laser Scanning and Close Range Photogrammetry for 3D Archaeological Documentation: The Upper Palaeolithic Cave of Parpallo as a Case Study," *Journal of Archaeological Science*, no. 37, pp. 499-507, 2010.
- [20] H. Son, C. Kim ve C. Kim, "3D Reconstruction of As-Built Industrial Instrumentation Models from Laser-Scan Data and a 3D CAD Database Based on Prior Knowledge," *Automation in Construction*, no. 49, pp. 193-200, 2015.
- [21] Y.-T. Su, J. Bethel ve S. Hu, "Octree-Based Segmentation for Terrestrial Lidar Point Cloud Data in Industrial Applications," *ISPRS Journal of Photogrammetry and Remote Sensing*, no. 113, pp. 59-74, 2016.
- [22] J. J. Monsalve, J. Baggett, R. Bishop ve N. Ripepi, "Application of Laser Scanning for Rock Mass Characterization and Discrete Fracture Network Generation in an Underground Limestone Mine," no. 29, pp. 131-137, 2019.

- [23] O. Monserrat ve M. Crosetto, "Deformation Measurement Using Terrestrial Laser Scanning Data and Least Squares 3D Surface Matching," *ISPRS Journal of Photogrammetry & Remote Sensing*, no. 63, pp. 142-154, 2008.
- [24] X. Xu, H. Yang ve I. Neumann, "Deformation Monitoring of Typical Composite Structures Based on Terrestrial Laser Scanning Technology," *Composite Structures*, no. 202, pp. 77-81, 2018.
- [25] T. Abmayr, F. Hartl, M. Reinköster ve C. Fröhlich, "Terrestrial Laser Scanning: Applications in Cultural Heritage Consercation and Civil Engineering," *Proceedings of the ISPRS working group V4*, 2005.
- [26] Y. S. Hayakawa ve T. Oguchi, "Applications of Terrestrial Laser Scanning in Geomorphology," *Journal of Geography (Chigaku Zasshi)*, cilt 3, no. 125, pp. 299-324, 2016.
- [27] E. Oniga, A. Savu ve A. Negrila, "The Evaluation of CloudCompare Software in the Process of TLS Point Clouds Registration," *RevCAD Journal of Geodesy and Cadastre*, no. 21, pp. 117-124, 2016.
- [28] C. Moreno ve M. Li, "A Comparative Study of Filtering Methods for Point Clouds in Real-Time Video Streaming," *Proceedings of the World Congress on Engineering and Computer Science*, San Francisco, 2016.
- [29] C. Sahin, "Planar Segmentation of Indoor Terrestrial Laser Scanning Point Clouds via Distance Function from a Point to a Plane," *Optics and Lasers in Engineering*, no. 64, pp. 23-31, 2014.
- [30] E. Che ve M. J. Olsen, "Multi-Scan Segmentation of Terrestrial Laser Scanning Data Based on Normal Variation Analysis," *ISPRS Journal of Photogrammetry and Remote Sensing*, no. 143, pp. 233-248, 2018.

- [31] R. Çömert, Yersel Lazer Tarayıcılardan Elde Edilen Veriler Kullanılarak Otomatik Nesne Çıkarımı, Eskişehir: Yüksek Lisans Tezi, Anadolu Üniversitesi Fen Bilimleri Enstitüsü, 2014.
- [32] Y. Ioannou, B. Taati ve R. Harrap, "Difference of Normals as a Multi-Scale Operator in Unorganized Point Clouds," 2012 Second International Conference on 3D Imaging, Modeling, Processing, Visualization & Transmission, IEEE, 2012.
- [33] M. Abdel, 3D Laser Scanners: History, Applications, and Future, 2011.
- [34] G. Vosselman ve H.-G. Maas, Airborne and Terrestrial Laser Scanning, CRC, 2010.
- [35] C. Fröhlich ve M. Mettenleiter, "Terrestrial Laser Scanning - New Perspectives in 3D Surveying," International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences, 2014.
- [36] G. Petrie, C. K. Toth ve J. Shan, "Terrestrial Laser Scanners," Topographic Laser Ranging and Scanning Principles and Processing, 2009, pp. 87-128.
- [37] T. Lovas, "Digitalis Tankönyvtar," 2010. [Çevrimiçi]. Available: <https://www.tankonyvtar.hu/>. [Erişildi: 05 01 2019].
- [38] X. D. Zhang, C. Sun, C. Wang ve S. H. Ye, "Study on Preprocessing Methods for Color 3D Point Cloud," Materials Science Forum. Trans Tech Publications., no. 471, p. 716-721, 2004.
- [39] M. Wu, J. Wang ve H. Luo, "Research on 3D Modeling of Point Cloud Data Based on Terrestrial Laser Scanner," 3rd International Conference on Mechatronics, Robotics and Automation (ICMRA), 015.

- [40] D. Levin, "Mesh-Independent Surface Interpolation," Geometric modeling for scientific visualization, Berlin, Heidelberg, 2004, pp. 37-49.
- [41] G. Sithole, Segmentation and Classification of Airborne Laser Scanner Data, Phd Thesis, Publications on Geodesy, Netherland, 2005.
- [42] C. Altuntaş, "İteratif En Yakın Nokta Yöntemi ile Lazer Tarayıcı Nokta Bulutlarının Birleştirilmesine Analitik Bir Bakış," IV. Uzaktan Algılama ve Coğrafi Bilgi Sistemleri Sempozyumu (UZAL-CBS 2012), Zonguldak, 2012.
- [43] N. Dyn, A. Iske ve W. Holger, "Meshfree Thinning of 3D Point Clouds," Foundations of Computational Mathematics, no. 8, pp. 409-425, 2008.
- [44] W. Boehler, G. Heinz, A. Marbs ve M. Siebold, "3D Scanning Software: An Introduction," CIPA Heritage Documentation, International Workshop on Scanning for Cultural Heritage Recording, 2002.
- [45] A. Nguyen ve B. Le, "3D Point Cloud Segmentation: A Survey," 2013 6th IEEE Conference on Robotics, Automation and Mechatronics (RAM), 2013.
- [46] P. P. Sapkota, Segmentation of Coloured Point Cloud Data, ITC: Master Thesis, International Institute for Geo-Information Science and Earth Observation, 2008.
- [47] K. G. Derpanis, "Overview of RANSAC Algorithm," ImageRochester NY, cilt 1, no. 4, pp. 2-3, 2010.
- [48] M. A. Fischler ve R. C. Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," Graphics and Image Processing, cilt 6, no. 24, pp. 381-395, 1981.

- [49] H. E. Adak, Developing a 3D Finite Element Software with an Object Oriented Approach, İstanbul: Master Thesis, ITU Graduate School of Science Engineering and Technology, 2014.
- [50] P. Deitel ve H. Deitel, C++ How to Program, Pearson, 2016.
- [51] A. Çakır, C++ Programlama Dili Ders Notları, Zonguldak: Bülent Ecevit Üniversitesi, 2015.
- [52] "Point Cloud Library," [Çevrimiçi]. Available: <http://www.pointclouds.org/documentation/tutorials/index.php>. [Erişildi: 2019].
- [53] R. B. Rusu ve S. Cousins, "3D is Here: Point Cloud Library (PCL)," 2011 IEEE International Conference on Robotics and Automation, 2011.
- [54] FARO, "Faro Laser Scanner Focus 3D330X Features, Benefits & Technical Specifications," 2013.
- [55] "LibE57," [Çevrimiçi]. Available: <http://www.libe57.org/>. [Erişildi: 2019].

Tezden Üretilmiş Yayınlar

İletişim Bilgisi: duyguaricann@gmail.com

Konferans Bildirileri

1. Arıcan D., Uzar M., “Yersel Lazer Tarayıcı Verileri ile 3 Boyutlu Modelleme Çalışmalarında Filtreleme ve Birleştirme Analizi”, III. Uluslararası Mühendislik ve Teknik Bilimler (UMTEB) Kongresi, GAZİANTEP, TÜRKİYE, 21-22 Haziran 2018.

