R. SİNEKLİ, 2015

T.C.
NİĞDE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI

EŞ ZAMANLI KONUMLANDIRMA VE HARİTALAMADA KULLANILMAK ÜZERE ÜÇ BOYUTLU TARAMA EŞLEŞTİRME ALGORİTMALARININ GERÇEKLENMESİ

RECAİ SİNEKLİ

AĞUSTOS 2015

YÜKSEK LİSANS TEZİ

NİĞDE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

T.C.
NİĞDE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI

EŞ ZAMANLI KONUMLANDIRMA VE HARİTALAMADA KULLANILMAK ÜZERE ÜÇ BOYUTLU TARAMA EŞLEŞTİRME ALGORİTMALARININ GERÇEKLENMESİ

RECAİ SİNEKLİ

Yüksek Lisans Tezi

Danışman

Yrd. Doç. Dr. Mehmet Kürşat YALÇIN

AĞUSTOS 2015

Recai SİNEKLİ tarafından **Yrd. Doç. Dr. Mehmet Kürşat YALÇIN** danışmanlığında hazırlanan "**Eş Zamanlı Konumlandırma ve Haritalamada Kullanılmak Üzere Üç Boyutlu Tarama Eşleştirme Algoritmalarının Gerçeklenmesi**" adlı bu çalışma jürimiz tarafından Niğde Üniversitesi Fen Bilimleri Enstitüsü **Elektrik-Elektronik Mühendisliği** Ana Bilim Dalı'nda Yüksek Lisans tezi olarak kabul edilmiştir.

Başkan : Yrd. Doç. Dr. Tuğrul OKTAY (Erciyes Üniversitesi)

Üye : Yrd. Doç. Dr. Fuat KARAKAYA (Niğde Üniversitesi)

Üye : Yrd. Doç. Dr. Mehmet Kürşat YALÇIN (Niğde Üniversitesi)

ONAY:

Bu tez, Fen Bilimleri Enstitüsü Yönetim Kurulunca belirlenmiş olan yukarıdaki jüri üyeleri tarafından …./…./20.... tarihinde uygun görülmüş ve Enstitü Yönetim Kurulu'nun …./…./20.... tarih ve ..... sayılı kararıyla kabul edilmiştir.

...../...../20...

Doç. Dr. Murat BARUT
MÜDÜR

TEZ BİLDİRİMİ

Tez içindeki bütün bilgilerin bilimsel ve akademik kurallar çerçevesinde elde edilerek sunulduğunu, ayrıca tez yazım kurallarına uygun olarak hazırlanan bu çalışmada bana ait olmayan her türlü ifade ve bilginin kaynağına eksiksiz atıf yapıldığını bildiririm.

Recai SİNEKLİ

ÖZET

EŞ ZAMANLI KONUMLANDIRMA VE HARİTALAMADA KULLANILMAK ÜZERE ÜÇ BOYUTLU TARAMA EŞLEŞTİRME ALGORİTMALARININ GERÇEKLENMESİ

SİNEKLİ, Recai
Niğde Üniversitesi
Fen Bilimleri Enstitüsü
Elektrik-Elektronik Mühendisliği Ana Bilim Dalı

Danışman :Yrd. Doç. Dr. Mehmet Kürşat YALÇIN

Ağustos 2015, 64 sayfa

Günümüzde robotlar genellikle endüstride kullanılır ve parça birleştirme, kaynak ve boya yapma gibi alanlarda görevlendirilirler. Bir robot, operatör kontrolünde çalışabileceği gibi bir bilgisayar programının kontrolünde de görevini yerine getirebilir. Gelişen kontrol teknikleri ve günümüz bilgisayarlarının işlem gücü sayesinde robotlar aşırı miktarda veriyi işleyerek, karar verebilir, özerk olarak hareket edebilir ve bir takım görevleri yerine getirebilirler. Bu sayede yerleri kendi kendine süpürebilecek veya enkaz altından bir insanı kurtarabilecek kadar yetenekli robotların geliştirilmesi mümkün kılınmıştır. Mobil robotların özerk olarak hareket edebilmesi bulundukları ortamın haritalandırılmasına bağlıdır. Eş zamanlı konumlandırma ve haritalama (SLAM) tekniği ile hareket halindeki bir robotun, içerisinde bulunduğu ancak bilmediği bir ortamın haritasını oluşturması ve kendisini bu haritaya göre konumlandırması sağlanır. Robotun ortamdan edindiği yeni bilgilerin sürekli olarak haritada uygun noktalara yerleştirilmesi gerekir. Bu işlem genellikle tarama eşleştirme algoritmaları kullanılarak yapılır. Bu çalışmada, hem simülasyon ortamından hem de gerçek ortamdan alınan veriler üzerinde çalışarak lazer tarama verilerini eşleştiren açık kaynak bir bilgisayar yazılımı geliştirilmiştir.

Anahtar Sözcükler: Tarama eşleştirme, mobil robot, eş zamanlı konumlandırma ve haritalama, SLAM

SUMMARY

IMPLEMENTATION OF SCAN MATCHING ALGORITHMS TO BE USED FOR SIMULTANEOUS LOCALIZATION AND MAPPING

SİNEKLİ, Recai

Niğde University

Graduate School of Natural and Applied Science

Department of Electrical-Electronics Engineering

Supervisor :Assistant Professor Dr. Mehmet Kürşat YALÇIN

August 2015, 64 pages

Today, robots are mainly used in industry and they are employed in areas such as part consolidation, welding and painting. A robot can perform its duty under the control of a computer program as well as working under the control of an operator. Thanks to the developments in the control techniques and the computing power of today's computers, robots can decide, move autonomously and perform a number of tasks by handling excessive amounts of data. Thus, development of robots that can sweep the floors autonomously or save the people trapped under the debris has become possible. Autonomous movement of mobile robots depends on mapping of their environment. A robot in motion uses the Simultaneous Localization and Mapping (SLAM) technique to build a map of its unknown environment and locate itself in the map. The new information received from the robot's environment must be placed in the appropriate spot on the map. This process usually done using scan matching algorithms. In this study, an open source software was developed for scan matching by using laser scan data that were collected both from the simulated and real environment.

Keywords: Scan matching, mobile robot, simultaneous localization and mapping, SLAM

ÖN SÖZ

Bu yüksek lisans çalışmasında, hem Gazebo simülasyon ortamından hem de gerçek ortamdan alınan lazer verileri ile çalışılarak üç boyutlu bir tarama eşleştirme algoritması bilgisayar ortamında gerçeklenmiştir.

Bu tez çalışması sürecinde desteğini esirgemeyen, değerli danışmanım Yrd. Doç. Dr. Mehmet Kürşat YALÇIN'a, lisans ve yüksek lisans eğitimim boyunca çalışmalarım esnasında tecrübelerine başvurduğum Yrd. Doç. Dr. Fuat KARAKAYA'ya, beraber çalıştığım tüm çalışma arkadaşlarıma, tüm bunların olmasını sağlayan maddi manevi desteklerini benden esirgemeyen aileme ve 113E210 numaralı proje kapsamında bu çalışmaya yapmış oldukları maddi ve manevi desteklerinden dolayı TÜBİTAK'a teşekkürlerimi sunarım.

İÇİNDEKİLER

ÇİZELGELER DİZİNİ

ŞEKİLLER DİZİNİ

KISALTMALAR

Kısaltmalar	Açıklama
ICP	Iterative Closest Point
SLAM	Simultaneous Localization and Mapping
IMU	Inertial Measurement Unit
PCL	Point Cloud Library
SVD	Singular Value Decomposition
SDF	Simulation Description Format
GPS	Global Positioning System

BÖLÜM I

GİRİŞ

1.1 Amaç ve Kapsam

Bilgisayarlardaki giderek artan işlem gücü, gelişen kontrol teknikleri ve fiziksel algılayıcılar robotların gündelik hayata daha fazla dahil olması için yapılan çalışmaları ivmelendirmiştir. Gündelik hayatta insanların yapabileceği işleri yapacak ya da yardımcı olabilecek robotların geliştirilmesi robotların daha akıllı hale getirilmelerine bağlıdır. Özerk hareket edebilecek, bir görevi yerine getirebilmek için karar verebilecek, bir hedefe ulaşmak için uygun rotayı planlayacak robotların ilk önce bulunduğu ortamı algılayabilmesi gerekmektedir. Bu işlem çeşitli algılayıcılar aracılığı ile ortamdan toplanan bilgilere dayanarak gerçekleştirilir. Görüntüye veya lazer tarama verilerine bağlı olarak ortamdan edinilen bilgiler bir harita oluşturmak için kullanılabilir. Ancak, hareket halindeki bir aracın haritayı doğru bir şekilde oluşturabilmesi için ortam verilerinin yanında kendi hareketini de kestirebilmesi gerekmektedir. Bu kestirim enkoderler veya ataletsel ölçüm birimleri (Inertial Measurement Unit) yardımıyla gerçekleştirilebilir. Ortam koşullarındaki değişiklikler ve ölçüm tekniğinden veya algılayıcılardan kaynaklı hatalar sebebiyle ölçüm sonuçlarında hatalar meydana gelmektedir. Bu hatalı verilerin oluşturulacak harita üzerinde de etkisi olmaktadır. Bu nedenle robot hareketinin kestirilmesinde algılayıcılardan elde edilen verilere destek olması ve daha doğru sonuçlar elde edebilmek adına tarama eşleştirme algoritmaları kullanılmaktadır. Tarama eşleştirme problemine çözüm olarak çeşitli yöntemler önerilmiş olsa da bunlardan en bilineni “Iterative Closest Point” algoritmasıdır (Besl ve McKay, 1992).

ICP algoritması, içerisinde serbest biçimli şekillerin bulunduğu iki boyutlu veya üç boyutlu nokta bulutlarının eşleştirilmesi için geliştirilmiş bir yöntemdir. Bu algoritma iki nokta bulutu arasında ilişki bulunduğu sürece yakınsamayı garanti eder. Bu başarısından dolayı öne sürüldüğü günden beri eş zamanlı konumlandırma ve haritalama (Costa vd., 2010; Fujita, 2012; Yoshitaka vd., 2006), insan vücudunun veya bir nesnenin hareketinin takip edilmesi (Kim ve Kim, 2010), bir mobil robotun kendi

hareketini kestirmesi (Bonaccorso vd., 2012; Martínez vd., 2006), tarama ile üç boyutlu şekillerin yeniden oluşturulması (Besl ve McKay, 1992; Estépar vd., 2004; Neugebauer, 1997; Wang vd., 2014) gibi alanlarda ICP temelli algoritmalar kullanılmaktadır.

ICP algoritması filtreleme, ilişkili noktaların tespiti, sıra dışılık analizi ve transformasyonun bulunması gibi adımlardan oluşur. ICP algoritmasının geliştirilmesine yönelik çalışmalarda genellikle bu adımlar üzerinde durulmuştur.

Filtreleme işlemi yapılmadığında her iki nokta bulutundaki bütün noktaların (Besl ve McKay, 1992), homojen bir şekilde filtreleme yapıldığında geriye kalan noktaların (Turk ve Levoy, 1994) veya her tekrarda rastgele seçilen noktaların (Masuda vd., 1996) kullanıldığı çalışmalar bulunmaktadır.

İlişkili noktaların tespiti için, bir nokta bulutundaki noktanın diğer nokta bulutundaki en yakın komşusunun bulunduğu (Besl ve McKay, 1992), en yakın komşu bulma işlemi ile renk bilgisinin (Joung vd., 2009; S. Druon vd., 2006) veya lazer ışını yoğunluk bilgisinin (Weik, 1997; Yoshitaka vd., 2006) birleştirildiği, klasik yaklaşımda olduğu gibi noktadan noktaya değil noktadan yüzeye mesafeye bakıldığı (Censi, 2008), noktaların Kartezyen koordinat sistemi yerine polar koordinat sisteminde incelendiği (Diosi ve Kleeman, 2007), geleneksel ICP algoritmasının genetik algoritma ile birleştirildiği (Martínez vd., 2006), köşe bulma algoritması ile tespit edilen özniteliklerin kullanıldığı (Ray vd., 2012) çalışmalar yapılmıştır.

İlişkili noktaların tespitinden sonra gelen aykırı noktaların tespiti olarak da isimlendirebileceğimiz sıra dışılık analizi adımında ise, ilişkili noktaların aralarındaki mesafenin önceden belirlenmiş sabit bir eşik değerden daha büyük olanlarının elenmesi ilk akla gelen en basit yöntemdir. Bu basit yöntem yerine ilişkili noktaların aralarındaki mesafelerin ortalama değer ve standart sapmasının toplamının veya medyan değerinin sabit bir katının eşik değer olarak belirlenmesi gibi istatistiksel metotlar içeren çalışmalar bulunmaktadır (Diebel vd., 2004; Pomerleau vd., 2010). Rastgele örnekleme yaparak tekrarlı bir şekilde en uygun noktaların seçilmesine dayalı yöntemler de önerilmiştir (Kim vd., 2009).

ICP algoritmasının son adımı olan bir nokta bulutunu diğeri ile eşleştirecek uygun transformasyonun bulunması adımında ise genellikle tercih edilen yöntem tekil değer ayrışımına dayalı (Singular Value Decomposition) bir yöntemdir (Arun vd., 1987). İki nokta bulutundaki ilişkili noktalar doğru tespit edildiği sürece doğrudan sonuca ulaştıran, tekrarlı olmayan bir yöntemdir.

Bu çalışmada, bahsedilen ICP algoritması C++ programlama dili kullanılarak kodlanmış, hem simülasyon ortamından hem de gerçek ortamdan alınan verilerle testleri yapılmıştır. Yapılan testler sonucunda filtre boyutu ve sıra dışılık analizi adımında kullanılan medyan katsayısının seçimi için uygun aralıklar belirlenmiştir.

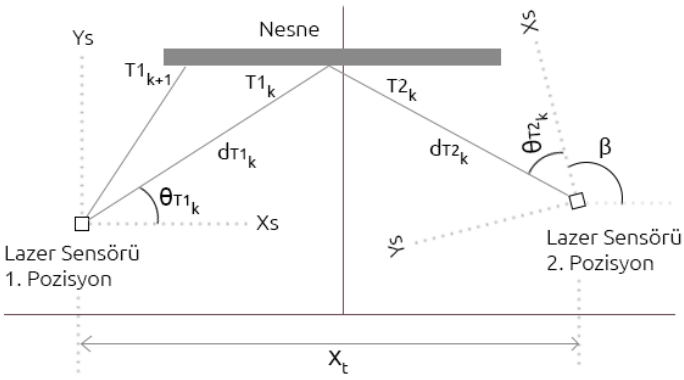
BÖLÜM II

TARAMA EŞLEŞTİRME

2.1 Nokta Bulutu Eşleştirme Problemi

Nokta bulutu veren bir algılayıcıdan alınan ardışık iki nokta bulutunun en uygun şekilde eşleştirilmesi, çözümüne çok sayıda kişinin katkıda bulunduğu önemli bir problemdir. Hareket halindeki bir algılayıcıdan farklı zamanlarda alınan veriler farklı koordinat sistemlerinde olacaktır. Bu verilerin en az hatayla aynı koordinat sistemine taşınması işlemi tarama eşleştirme olarak adlandırılır.

İki boyutlu bir lazer mesafe algılayıcısının bir taramasında aralarında belirli bir açı bulunan belirli sayıda lazer ışını bulunur. Çevresindeki herhangi bir nesne ile kesişen bir lazer ışını, kutupsal koordinat sisteminde bir açı ve bir mesafe ile temsil edilir. Şekil 2.1'de algılayıcının birinci pozisyonu için k anındaki ölçüm ($\mathrm{T1}_k$), d_{T1_k} mesafesi ve θ_{T1_k} açısı ile temsil edilmiştir. İkinci pozisyonda algılayıcı $\mathrm{X_t}$ kadar ötelenmiş ve β kadar döndürülmüştür. Birinci pozisyonda nesne üzerinde görülen bir nokta, ikinci pozisyonda d_{T2_k} mesafesi ve θ_{T2_k} açısı ile temsil edilen $\mathrm{T2}_k$ lazer ışını ile kesişmiştir. Ardışık iki lazer taramasının eşleştirilmesi sonucunda, ikinci pozisyondaki $\mathrm{T2}_k$ lazer ışını ile ölçülen noktanın birinci pozisyondaki görüntüsü bulunur. Böylece algılayıcının evrensel koordinat sistemine göre ne kadar ötelendiği ve döndürüldüğü bilgisi elde edilmiş olur.



Şekil 2.1. Ardışık iki lazer taraması

Kutupsal koordinat sisteminde olan ölçüm sonuçları denklem 2.1 yardımıyla kartezyen koordinat sisteminde ifade edilir.

$$X_{T1}(k) = d_{T1_k}\cos(\theta_{T1_k})$$
$$Y_{T1}(k) = d_{T1_k}\sin(\theta_{T1_k}) \tag{2.1}$$

Bu çalışmada algılayıcıdan gelen ilk nokta bulutu hedef, ikinci nokta bulutu ise kaynak olarak isimlendirilmiştir. Geleneksel olarak yapılan işlem, kaynak nokta bulutuna bir transformasyon uygulayarak bu noktaların hedef nokta bulutu ile en uygun şekilde eşleşmesini sağlamaktır. Kaynak nokta bulutu $K = \{k_1, k_2, k_3, \dots, k_N\}$, hedef nokta bulutu ise $H = \{h_1, h_2, h_3, \dots, h_N\}$ olarak tanımlanırsa, nokta bulutları arasındaki eşleşmenin nokta bulutları arasındaki eşleşmenin ne seviyede olduğunu gösterebilmek için denklem 2.2'de görüldüğü gibi bir hata ölçütü yazılabilir. Bu hata ölçütü, kaynak nokta bulutuna bir transformasyon matrisi (T) uygulandıktan sonra bu noktalar ile hedef nokta bulutundaki en yakın noktalar arasındaki mesafelerin karelerinin toplamıdır.

$$E = \sum_{i=1}^{N} \|T(k_i) - h_i\|^2 \tag{2.2}$$

Üç boyutlu uzayda transformasyonun, üç farklı eksende dönme ve üç farklı eksende öteleme işlemi gerçekleştirebileceğinden dolayı, altı serbestlik derecesi vardır. Dönme işlemi $\boldsymbol{R}$ rotasyon matrisi, öteleme işlemi $\vec{\boldsymbol{t}}$ öteleme vektörü ile tanımlanırsa, $\boldsymbol{E}$ hata ölçütü denklem 2.3'deki gibi yazılabilir.

$$E = \sum_{i=1}^{N} \left\|\boldsymbol{R}k_i + \vec{\boldsymbol{t}} - h_i\right\|^2 \tag{2.3}$$

Denklem 2.4'te rotasyon matrisi ve öteleme vektöründen oluşan transformasyon matrisi görülmektedir.

$$T_{4x4} = \begin{bmatrix} \boldsymbol{R}_{3x3} & t_{3x1} \\ 0_{1x3} & 1 \end{bmatrix} \tag{2.4}$$

Sırayla x-ekseni etrafında α, y-ekseni etrafında β ve z-ekseni etrafında γ açısıyla dönmeyi saylayacak rotasyon matrisleri denklem 2.5, 2.6 ve 2.7 ile verilmiştir.

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\alpha & -sin\alpha \\ 0 & sin\alpha & cos\alpha \end{bmatrix} \tag{2.5}$$

$$R_y(\beta) = \begin{bmatrix} cos\beta & 0 & sin\beta \\ 0 & 1 & 0 \\ -sin\beta & 0 & cos\beta \end{bmatrix} \tag{2.6}$$

$$R_z(\gamma) = \begin{bmatrix} cos\gamma & -sin\gamma & 0 \\ sin\gamma & cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.7}$$

Bir genelleştirilmiş rotasyon matrisi hesaplamak da mümkündür. Genelleştirilmiş rotasyon matrisi dönme yönüne bağlıdır. Örneğin; önce x, sonra y ve daha sonra z-ekseni etrafında dönmek için denklem 2.8 kullanılırken, önce z, sonra y ve daha sonra x-ekseni etrafında dönmek için denklem 2.9 kullanılmalıdır (Slabaugh, 1999).

$$R_z R_y R_x = \begin{bmatrix} cos\beta cos\gamma & cos\gamma sin\alpha sin\beta - cos\alpha sin\gamma & cos\alpha cos\gamma sin\beta + sin\alpha sin\gamma \\ cos\beta sin\gamma & cos\alpha cos\gamma + sin\alpha sin\beta sin\gamma & -cos\gamma sin\alpha + cos\alpha sin\beta sin\gamma \\ -sin\beta & cos\beta sin\alpha & cos\alpha cos\beta \end{bmatrix} \tag{2.8}$$

$$R_x R_y R_z = \begin{bmatrix} cos\beta cos\gamma & -cos\beta sin\gamma & sin\beta \\ cos\alpha sin\gamma + sin\alpha sin\beta cos\gamma & cos\alpha cos\gamma - sin\alpha sin\beta sin\gamma & -sin\alpha cos\beta \\ sin\alpha sin\gamma - cos\alpha sin\beta cos\gamma & sin\alpha cos\gamma + cos\alpha sin\beta sin\gamma & cos\alpha cos\beta \end{bmatrix} \tag{2.9}$$

Nokta bulutu eşleştirmede, hata ölçütündeki göreceli değişim istenilen bir eşik değerin altına indiğinde veya maksimum tekrarlama sayısına ulaşıldığında hesaplanan transformasyon matrisi kaynak nokta bulutunu hedef nokta bulutuna eşleştiren transformasyon matrisi olarak alınır.

Tarama eşleştirme algoritmaları, lazer taramaların eşleştirilerek, bir nesnenin üç boyutlu modelinin oluşturulması, nesnelerin hareket takibi veya bir ortamın haritalandırılmasını sağlar. Bu işlemler, eğer tarama verisini elde etmek için kullanılan algılayıcının hareketi bilinirse tarama eşleştirme algoritmasına gerek kalmadan halledilebilir. Örneğin, bir robot üzerindeki algılayıcının ortam içerisindeki pozisyonu ve duruşu bilinseydi tarama

eşleştirme algoritmalarına gerek duyulmadan ortam haritalandırılabilirdi. Ortam haritalandırma için algılayıcının ortam içerisindeki konumunun yanında duruşunun da bilinmesi gerekir. Yani, küresel konumlandırma verisi olsa bile haritalandırma için yeterli olmayacaktır. Geliştirilen tarama eşleştirme algoritmalarının hedefi, algılayıcının konumu ve duruşundan bağımsız olarak iki taramanın en doğru şekilde eşleştirilmesidir.

2.2 Nokta Bulutu Eşleştirmede Kullanılan Algoritmalar

Nokta bulutu eşleştirme problemine çözüm getiren çalışmalardan bazıları aşağıda listelenmiştir.

ICP (Iterative Closest Point) Algoritması : ICP algoritması, tarama eşleştirme algoritmaları içerisinde en bilinen ve en çok kullanılan algoritmadır (Besl ve McKay, 1992). ICP, nokta bulutları arasında bir ilişki bulunduğu sürece yakınsamayı garanti eden bir algoritmadır ancak bu algoritmayı daha güçlü bir hale getirmek için yapılan çalışmalar sonucunda ortaya bir çok ICP varyantı çıkarılmıştır. Bu algoritma, kendi içerisinde filtreleme, noktaların ilişkilendirilmesi, aykırı noktaların elenmesi ve transformasyonun bulunması gibi bazı adımlara ayrılır. Bu adımlarda yapılan geliştirmeler eşleşme sonucunu doğrudan etkiler. Örneğin, ICP algoritması ile genetik algoritmanın gürbüzlüğünü bir araya getiren bir çalışma ICP varyantı olarak sunulmuştur (Lenac vd., 2011). Algoritmadaki ilişkili noktaların bulunması adımında renk bilgisini de dahil ederek arama uzayını daraltan çalışmalar da bulunmaktadır (Joung vd., 2009; S. Druon vd., 2006). Benzer şekilde, lazer algılayıcının ışın şiddeti bilgisini dahil ederek ilişkili noktaların bulunduğu çalışma da bir başka ICP varyantı olarak gösterilebilir (Yoshitaka vd., 2006). Bir çok çalışmada ilişkili noktalar bulunurken tüm noktalar kullanılmıştır ancak rastgele örnekleme yapılarak seçilen noktalar arasında ilişkili noktaların tespit edildiği bir çalışma da bulunmaktadır (Masuda vd., 1996).

Hough Dönüşümü : Lazer verisi üzerinde Hough dönüşümü kullanılarak geliştirilen bu tarama eşleştirme yönteminde Hough düzleminin özelliklerinden faydalanılmıştır. Öznitelik temelli çalışan tarama eşleştirme yöntemlerine benzer ve üretilen lazer verileri

daha az gürültülü olduğundan ICP algoritması ile birlikte daha iyi sonuçlar elde etmek için kullanılabilir (Censi vd., 2005).

Kutupsal Koordinat Kullanarak Tarama Eşleştirme : Bu çalışmada önerilen yöntemle ICP algoritmasında yer alan noktaların eşleştirilmesi problemini ortadan kaldırılmıştır. Ancak yine ICP gibi interaktif olan bu yöntem optimum transformasyonun bulunmasından önce bir bölümleme adımı içermektedir (Diosi ve Kleeman, 2007).

Normal Dağılım Transformu : Nokta bulutlarındaki noktaların ilişkilendirilmesi yerine bir noktanın referans modeldeki ilişkili hücresinin bulunduğu, normal dağılım transformuna dayalı, bir tarama eşleştirme algoritması önerilmiştir (Ulaş ve Temeltaş, 2013). Renk bilgisini de içerek üç boyutlu nokta bulutlarında noktaların renk uzayındaki dağılımı ele alarak çalışan bir alternatif tarama eşleştirme algoritması da bulunmaktadır (Huhle vd., 2008).

RANSAC : "RAndom Sample and Consensus" yöntemine dayalı, öznitelik çıkarmaya ve başlangıç transformasyonuna ihtiyaç duyulmayan, rastgele ve tekrarlı bir biçimde en uygun transformasyonun arandığı bir yöntem de tarama eşleştirme algoritması olarak sunulmuştur (Kim vd., 2009).

Bu çalışmada, tarama eşleştirme algoritmalarının bir çoğunun temelini oluşturan ICP algoritması nokta bulutları arasında ilişki bulunduğu sürece yakınsamayı garanti ettiğinden dolayı tercih edilmiş ve detayları aşağıda verilmiştir.

2.3 Iterative Closest Point (ICP) Algoritması

ICP algoritması genel olarak iki veya daha fazla nokta seti arasındaki farkı en aza indirmek için kullanılır (Besl ve McKay, 1992; Chen ve Medioni, 1992). Eş zamanlı konumlandırma ve haritalama (Costa vd., 2010; Diebel vd., 2004; Fujita, 2012), insan vücudunun veya bir nesnenin hareketinin takip edilmesi (Kim ve Kim, 2010), bir mobil robotun kendi hareketini kestirmesi (Bonaccorso vd., 2012; Martínez vd., 2006), tarama

ile üç boyutlu şekillerin yeniden oluşturulması (Besl ve McKay, 1992; Estépar vd., 2004) gibi alanlarda ICP temelli algoritmalar kullanılmaktadır. Bu algoritma ayrıntılı biçimde incelenecek olursa aşağıdaki adımlar takip edilir.

i. **Başlatma :** Eğer iki nokta bulutunu birbirine yaklaştıracak bir başlangıç transformasyonu bilinmiyorsa T_0 başlangıç transformasyonu 4x4 birim matris olarak alınır.
ii. **Filtreleme :** İşlem hızını artırmak için her iki nokta bulutundaki nokta sayısını azaltacak şekilde örnekleme yapılır.
iii. **En yakın komşu bulma :** Hedef nokta bulutundaki her bir nokta için diğer nokta bulutundaki en yakın nokta bulunur ve bu iki nokta birbiriyle ilişkilendirilir.
iv. **Aykırı nokta bulma :** Bir $D_{maksimum}$ eşik değeri belirlenir ve eşleşen iki nokta arasındaki mesafe $D_{maksimum}$ değerinden daha büyük ise bu iki nokta eşleşme listesinden çıkarılır. Bu test bütün eşleşen noktalar için yapılır.
v. **Transformasyonun bulunması :** Geriye kalan eşleşen noktalara göre iki nokta bulutu arasında bir transformasyon hesaplanır. Hesaplanan transformasyon başlangıç transformasyonu ile çarpılır ve en son durum saklanır (T_0 = T_0*Hesaplanan transformasyon). Yeni T_0 transformasyonu kaynak nokta bulutuna uygulanır.
vi. **Tekrarlama işleminin sonlandırılması (Durdurma kriteri) :** Maksimum tekrarlama sayına ulaşılıncaya veya o anda hesaplanan hata ölçütü (denklem 2.2) E ile bir önceki tekrarlamada hesaplanan E arasındaki fark önceden belirlenen bir eşik değerin altına ininceye kadar iii. Adıma dönülür ve işlemler tekrarlanır.

Bu adımlardan ayrıntılı olarak incelenmesi gerekenler devam eden kısımlarda anlatılmıştır.

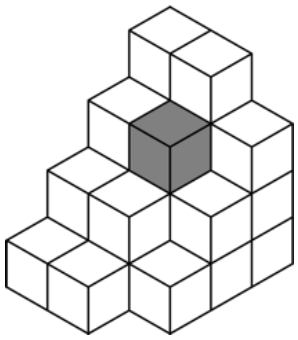
2.3.1 Filtreleme

Filtreleme işlemi yapılmadan her iki nokta bulutundaki bütün noktaların (Besl ve McKay, 1992), homojen bir şekilde filtreleme yapıldığında geriye kalan noktaların (Turk ve Levoy, 1994) veya her tekrarda rastgele seçilen noktaların (Masuda vd., 1996)

kullanıldığı çalışmalar bulunmaktadır. Bu çalışmada, ICP algoritmasının hızını arttırmak için nokta bulutu filtreleme işlemi olarak PCL kütüphanesinde kullanılan pcl::VoxelGrid sınıfı kullanılmıştır (PCL - Point Cloud Library).

PCL, iki boyutlu ve üç boyutlu resim ve nokta bulutlarını işlemek için geliştirilmiş açık kaynak kodlu bir kütüphanedir. Bu kütüphane bir çok filtreleme, öznitelik çıkarma, tarama eşleştirme ve bölümleme algoritmalarını içermektedir.

Voxel kelimesi "volume" ve "pixel" kelimelerinden oluşur ve bir pikseli üç boyutlu uzayda tanımlar. Şekil 2.2'de bir voxel grubu görülmektedir.



Şekil 2.2. Voxel grubu ve bir voxel (gri)

pcl::VoxelGrid sınıfı, verilen bir nokta bulutu üzerinde üç boyutlu bir ızgara oluşturur ve nokta bulutu üzerinde her bir voxeldeki noktaları, o noktaların merkezindeki bir nokta ile ifade edecek şekilde filtreler. Bu işlem her bir voxelin merkezini bir nokta olarak almaktan daha uzun sürer ancak nokta bulutunu daha doğru bir şekilde temsil eder.

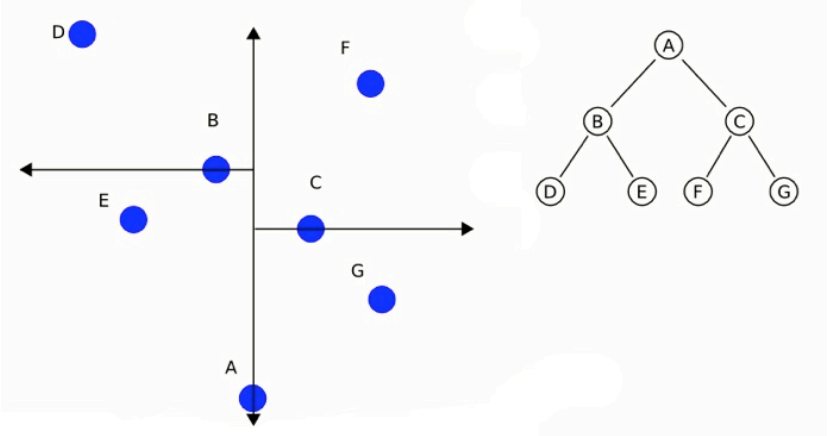
pcl::VoxelGrid sınıfı kullanılırken nokta bulutu üzerinde oluşturulacak her bir voxelin hacmi x, y ve z için üç ayrı parametre ile belirlenir ve bu parametreler filtre boyutu (leaf size) olarak isimlendirilmiştir. Voxel hacmi arttıkça nokta bulutunda filtreleme sonrasında geriye kalan nokta sayısı azalır.

2.3.2 En Yakın Komşu Bulma

En yakın nokta arama veya benzerlik arama olarak da bilinen en yakın komşu bulma algoritması, en çok benzeyen veya en yakın noktanın bulunmasını sağlayan bir optimizasyon problemidir. Bu problem genelde, M uzayındaki bir q noktasına ($q \in M$), S noktalar kümesi içindeki en yakın noktanın bulunması olarak tanımlanır. Bu problem daha genelleştirilmiş bir ifadeyle k-NN olarak isimlendirilir ve S kümesinden k adet en yakın noktanın bulunması olarak ifade edilir.

En yakın komşu bulma problemine en basit yaklaşım sıralı arama algoritmasıdır. Aranılan eleman bulununcaya veya liste sonlanıncaya kadar arama yapılır. Üzerinde çalışılan listenin sıralı olmasına gerek yoktur (Knuth, 1998). Bu çalışmada, M uzayındaki bir q noktasının S noktalar kümesindeki her bir noktaya olan mesafesini hesaplayarak ve o ana kadar bulunan en küçük mesafeyi ve hangi nokta olduğunu saklayarak çözüm getiren bir C++ sınıfı yazılmıştır. S kümesindeki nokta sayısı N, M uzayının nokta boyutu d ise sıralı arama algoritmasının zaman karmaşıklığı $O(Nd)$ olarak hesaplanır.

Uzay bölümleme yöntemleri arasında en basiti olan, PCL kütüphanesinin de kullandığı "k-d tree" algoritmasıdır (Bentley, 1975). Bu algoritma Şekil2.3'te görüldüğü gibi, arama alanını tekrarlı olarak bir üst bölgedeki noktaların yarısını ihtiva eden iki alt bölgeye ayırarak çalışır.



Şekil 2.3. İki boyutlu uzayda k-d tree

Bir k-d tree oluşturma süresi, N nokta için $O(NlogN)$, oluşturulmuş bir k-d tree içerisinde bir noktaya en yakın komşuyu bulmak için ise $O(logN)$'dir (Andrew Moore, 1991; Bentley, 1975).

PCL kütüphanesi en yakın komşu bulmak için yazılmış FLANN (Fast Library for Approximate Nearst Neighbours) kütüphanesini içermektedir. Bu çalışmada, sıralı arama algoritmasının yanında pcl::KdTreeFLANN sınıfı kullanılarak da en yakın komşu bulma işlemi gerçekleştirilmiştir. Her iki algoritmanın sonuçları birebir aynıdır fakat zaman karmaşıklığı hesabından da anlaşıldığı üzere k-d tree daha hızlı çalışmaktadır.

2.3.3 Aykırı Nokta Bulma

Aykırı nokta bulma, nokta bulutlarında bulunan ve bir önceki adımda birbirine en yakın komşu olarak atanan noktalar arasından bazı ikililerin elenmesi işlemidir. ICP algoritmasının başarısını doğrudan etkileyen adımlardan biridir. Bu nedenle aykırı noktaların tespit edilebilmesi için çeşitli yöntemler önerilmiştir. Bu yöntemlerden bazıları aşağıda verilmiştir.

Sabit : Eşleşen noktalar için sabit bir $D_{maksimum}$ değerinin belirlendiği yöntemdir. Eşleşen noktalar arasından mesafesi $D_{maksimum}$ değerinden daha büyük olanlar aykırı nokta olarak belirlenir ve eşleşme listesinden çıkarılır. En basit yöntemdir ancak farklı eşik değer gerektiren durumlara adapte olamaz (Pomerleau vd., 2010).

Ortalama : Bu yöntem, $D_{maksimum}$ mesafesini eşleşen noktalar arası mesafelerin ortalama değeri ile standart sapmasının toplamı ($\mu + \sigma$) olarak alır. $D_{maksimum}$ değerinin sabit kabul edildiği yönteme göre daha esnektir ve farklı eşik değerler gerektiren durumlara adapte olabilir. Eşleşen noktalar arası mesafelerin normal dağılımda olduğu yani hareketli nesnelerin olmadığı ortamlarda aykırı noktaların tespit edilmesi için uygun bir yöntemdir (Pomerleau vd., 2010).

Medyan : Eşleşen noktalar arasındaki mesafelerin medyan değerinin 3 katının $D_{maksimum}$ olarak belirlendiği bir yöntemdir (Diebel vd., 2004; Pomerleau vd., 2010). Farklı eşik değer gerektiren durumlara adapte olabilir ancak iki nokta bulutu arasındaki mesafe

değerlerinin medyanının bulunması sabit veya ortalama değer kullanan yöntemlere göre hesaplama açısından daha maliyetlidir. Medyan hesaplanırken aşağıda görüldüğü gibi std::sort fonksiyonu kullanılmıştır.

```
float IterativeClosestPoint::median(std::vector<float> distance){
    float median;
    std::sort(distance.begin(),distance.end());
    if(distance.size()%2==0){
        median = (distance[distance.size()/2-1]+distance[distance.size()/2])/2;
    }else{
        median = distance[distance.size()/2];
    }
    return median;
}
```

RANSAC (RAndom Sample and Consensus) : RANSAC algoritması, aykırı noktalar barındıran bir veri seti içerisinden elde edilecek matematiksel modelin parametrelerini kestirmek için kullanılan tekrarlı bir yöntemdir. Bu algoritma ilk olarak Fischler ve Bolles tarafından yayınlanmıştır (Fischler ve Bolles, 1981).

RANSAC algoritması, üzerinde çalışacağı veri setinin hem aykırı hem de aykırı olmayan noktalar içerdiğini, aykırı olmayan noktalardan bir model çıkarılabileceğini ve aykırı noktaların bu modele uymayacağını varsayar.

Bu algoritma, veri seti içerisinden rastgele bir alt küme seçer. Bu alt kümedeki verileri aykırı olmayan noktalar olarak kabul eder ve aşağıdaki işlemlerle devam eder.

i. Aykırı olmayan noktalardan bir model oluşturulur.
ii. Diğer noktalar bu model ile test edilir ve modele uyan noktalar da aykırı olmayan noktalara dahil edilir.
iii. Belirlenen bir eşik değere bağlı olarak yeterince çok sayıda nokta aykırı olmayan nokta olarak tespit edilirse birinci adımda oluşturulan modelin iyi bir model olduğu kabul edilir.
iv. Sadece başlangıçta seçilen aykırı noktalardan bir model oluşturulduğundan dolayı yeni bulunan aykırı olmayan noktalar da hesaba katılarak yeni bir model oluşturulur.

v. Son olarak model, aykırı olmayan noktaların modele göre hatası kestirilerek değerlendirilir.

Bu işlemler maksimum tekrarlama sayısına ulaşılıncaya kadar devam eder. Her tekrarlamada oluşturulan model, çok az sayıda aykırı olmayan nokta içeriyorsa kabul edilmez veya çok sayıda aykırı olmayan nokta içeriyorsa beşinci adımda hesaplanan hata değeriyle birlikte kabul edilir. Kabul edilmesi durumunda, daha önceki tekrarlamalar sonucu bulunan en iyi modelin hata değeriyle karşılaştırılır ve hatanın daha düşük olması durumunda en iyi model olarak saklanır.

RANSAC algoritması veri setinde çok sayıda aykırı nokta olsa bile yüksek doğruluk derecesi ile doğru modeli oluşturur. Tekrarlama sayısı arttıkça doğru modeli bulma ihtimali de artar ancak bu durum sonuca ulaşmak için geçen süreyi artıracaktır. RANSAC algoritması ile ilgili en büyük dezavantaj rastgele örneklemeye dayalı olmasından kaynaklı olarak tekrarlanabilir olmamasıdır (Hast ve Nysjö, 2013). RANSAC algoritmasının bir dezavantajı da kullanıcı tarafından belirlenecek birden fazla keyfi parametreye ihtiyaç duyması ve bu parametre değerlerinin belirlenmesi problemidir (Fischler ve Bolles, 1981).

2.3.4 Transformasyonun Hesaplanması

İki nokta bulutu arasında ilişkili noktalar bulunduktan ve aykırı noktalar elendikten sonra yapılması geren işlem optimum dönme ve ötelemenin bulunmasıdır. İki nokta bulutu arasındaki ilişkili noktalar biliniyorsa, kaynak nokta bulutunu hedef nokta bulutuna taşıyacak dönme ve öteleme, "Singular Value Decomposition" (SVD) işlemine bağlı olarak bulunabilir (Arun vd., 1987). Noktalar hatalı eşleştirildiğinde bulunacak dönme ve öteleme de hatalı sonuç verecektir.

SVD işleminin ayrıntıları aşağıdaki gibidir.

SVD, lineer cebirde, $\boldsymbol{M_{nxp}}$ matrisinin $\boldsymbol{M_{nxn} = U_{nxn} S_{nxp} V^{T}_{pxp}}$ şeklinde çarpanlarına ayrılmasıdır. Sinyal işleme ve istatistikte birçok uygulama alanına sahiptir (Strang, 2005).

SVD teoremine göre aşağıdaki işlemler sağlanmalıdır.

$$U^T U = I_{nxn}$$

$$V^T V = I_{pxp}$$

U matrisinin sütunları sol tekil vektörler ve V^T matrisinin satırları sağ tekil vektörler olarak adlandırılır. S matrisi ise tekil değerlerden oluşan sözde-köşegen matristir.

SVD hesabının yapılması MM^T ve $M^T M$ 'nin öz değer ve öz vektörlerinin bulunmasından oluşur. $M^T M$'nin öz vektörleri V matrisinin sütunlarını, MM^T matrisinin öz vektörleri ise U matrisinin sütunlarını oluşturur. S matrisindeki tekil değerler ise MM^T veya $M^T M$'nin öz değerlerinin kareköküdür ve bu değerler artan sırada S matrisinin köşegen elemanlarıdır. Tekil değerler her zaman reel sayılardır ve M matrisi reel ise U ve V matrisleri de reeldir (Singular Value Decomposition (SVD) tutorial).

Kaynak nokta bulutunun, hedef nokta bulutu ile ilişkili noktaları bulunduktan sonra SVD tabanlı optimum dönme ve ötelemenin bulunması 2.10 denkleminde $\boldsymbol{R}$ rotasyon matrisi ve $\vec{\boldsymbol{t}}$ öteleme vektörünün hesaplanmasıdır.

$$min\left(E = \sum_{i=1}^{N} \left\| \boldsymbol{R} k_i + \vec{\boldsymbol{t}} - h_i \right\|^2\right) \tag{2.10}$$

Kaynak nokta bulutu hedefe taşınırken, kaynak nokta bulutundaki noktalar arası mesafeyi değiştirecek veya noktaları kırpacak şekilde herhangi bir bozulmaya sebebiyet verilmemelidir. Bu işleme rijit transformasyon denir. Optimum rijit transformasyonun bulunması işlemi aşağıdaki üç temel adımdan oluşur.

i. Kaynak nokta bulutunda, hedef nokta bulutu ile ilişkili olan noktaların ağırlık merkezi bulunur. Hedef nokta bulutunun ise ilişkili olmayan noktalar çıkarıldıktan sonra geriye kalanlarının ağırlık merkezleri hesaplanır.

ii. Her iki nokta bulutunun tüm elemanlarından kendi ağırlık merkezleri çıkarılarak nokta bulutları orijine taşınır ve rotasyon matrisi hesaplanır.

iii. Ağırlık merkezleri arasındaki farktan faydalanarak da öteleme vektörü hesaplanır.

Bu adımların detayları aşağıdaki şekildedir.

Nokta bulutlarının ağırlık merkezleri denklem 2.11'de görüldüğü gibi hesaplanır. Bu işlemle ilgili C++ kod parçacığı aşağıda verilmiştir.

$$p_1 = \frac{1}{N_K}\left(\sum_{i=1}^{N_K} k_i\right)$$
$$p_2 = \frac{1}{N_H}\left(\sum_{i=1}^{N_H} h_i\right) \quad (2.11)$$

```
//Ağırlık merkezlerini hesapla
mean(*set1_tr, p1, true, kdcorresp); //Sadece ilişkili noktalar
mean(*set2_tr, p2, false);

void IterativeClosestPoint::mean(pcl::PointCloud<pcl::PointXYZ> &cloud, Eigen::VectorXf &p,
                                 bool correspOnly,const std::vector<int> &corresp){
    float x,y,z;
    x=y=z=0;
    if(correspOnly){
        for(int i=0;i<corresp.size();i++){
            x+=cloud.points[corresp[i]].x;
            y+=cloud.points[corresp[i]].y;
            z+=cloud.points[corresp[i]].z;
        }
        p(0)=x/corresp.size();
        p(1)=y/corresp.size();
        p(2)=z/corresp.size();
    }else{
        for(int i=0;i<cloud.points.size();i++){
            x+=cloud.points[i].x;
            y+=cloud.points[i].y;
            z+=cloud.points[i].z;
        }
        p(0)=x/cloud.points.size();
        p(1)=y/cloud.points.size();
        p(2)=z/cloud.points.size();
    }
}
```

Ağırlık merkezleri bulunduktan sonra her iki nokta bulutunun da ağırlık merkezleri orijine taşınarak SVD işlemine tabi tutulacak M_{3x3} matrisi denklem 2.12'deki gibi hesaplanır. Bu işlem için yazılmış C++ kod parçacığı aşağıda verilmiştir.

$$\mathbf{M} = \sum_{i=1}^{N} (k_i - p_1)(h_i - p_2)^{\mathrm{T}} \tag{2.12}$$

```
Eigen::MatrixXf M(3,3);
M.setZero(3,3);
for(int i=0;i<set2_tr->points.size();i++){
    Eigen::VectorXf temp(3);
    temp(0)=set1_tr->points[kdcorresp[i]].x-p1(0);
    temp(1)=set1_tr->points[kdcorresp[i]].y-p1(1);
    temp(2)=set1_tr->points[kdcorresp[i]].z-p1(2);

    Eigen::RowVectorXf temp2(3);
    temp2(0)=set2_tr->points[i].x-p2(0);
    temp2(1)=set2_tr->points[i].y-p2(1);
    temp2(2)=set2_tr->points[i].z-p2(2);

    M += temp*temp2;
}
```

M_{3x3} matrisi tekil değerlerine ayrıştırıldıktan sonra U ve V matrisleri kullanılarak denklem 2.13'de görüldüğü gibi rotasyon matrisi elde edilir. Bu çalışmada "Eigen" kütüphanesinin "JacobiSVD" sınıfı kullanılarak transformasyon hesabı yapılmıştır ve işlemle ilgili gerekli C++ kod parçacığı aşağıda verilmiştir.

$$R = VU^T \tag{2.13}$$

```
//SVD
Eigen::JacobiSVD<Eigen::MatrixXf> svd(M, Eigen::ComputeFullU | Eigen::ComputeFullV);
Eigen::MatrixXf R(3,3);
Eigen::MatrixXf U(3,3);
Eigen::MatrixXf V(3,3);

U = svd.matrixU();
V = svd.matrixV();
```

Rotasyon matrisi hesaplanırken nadiren de olsa yansıma olarak adlandırılan özel bir durumla karşılaşılır. Rotasyon matrisinin determinantının -1'e eşit olması durumunda

oluşan bu özel durum V matrisinin herhangi bir sütununun işareti değiştirilerek çözülür (Fischler ve Bolles, 1981). Bu işlemle ilgili C++ kod parçacığı aşağıda verilmiştir.

```
if (U.determinant () * V.determinant () < 0)
{
    for (int x = 0; x < 3; ++x)
        V (x, 2) *= -1;
}

R = V*U.transpose();
```

Rotasyon matrisi hesaplandıktan sonra $\vec{\boldsymbol{t}}$ vektörü denklem 2.14'te görüldüğü gibi hesaplanır (Fischler ve Bolles, 1981).

$$\vec{\boldsymbol{t}} = -\mathbf{R} * p_1 + p_2 \tag{2.14}$$

Rotasyon matrisi ve öteleme vektörü denklem 2.15'te görüldüğü gibi birleştirilerek T_{4x4} transformasyon matrisi elde edilir.

$$\mathrm{T}_{4\mathrm{x}4} = \begin{bmatrix} \boldsymbol{R}_{3\mathrm{x}3} & \vec{\boldsymbol{t}}_{3x1} \\ 0_{1\mathrm{x}3} & 1 \end{bmatrix} \tag{2.15}$$

BÖLÜM III

SİMÜLASYON ORTAMI

3.1 Neden Bir Simülasyon Ortamına İhtiyaç Duyulmuştur?

Tarama eşleştirme algoritmalarının testi için hazır veri setleri kullanılabileceği gibi gerçek ortam verisi toplayacak bir düzenek de oluşturulabilir. Test işlemleri sırasında çoğu zaman hazır veri setlerinin sunduğundan fazlasına ihtiyaç duyulur. Bu nedenle gerçek ortam verisi toplayacak bir düzenek ile yapılan testler ve sonuçları çalışmayı daha değerli kılar. Fakat, böyle bir düzenek ve sürekli olarak farklı ortamlar oluşturmak hem zaman hem de parasal açıdan maliyetli olmaktadır. Bu bağlamda, üç boyutlu olarak hem robot hem de ortam tasarımı yapılabilecek, gerçeğe yakın dinamiklere sahip ve açık kaynak bir robot simülatörü önem kazanmaktadır. Bir robot simülatörü kullanıldığında maliyet düşer ve tasarlanan robot veya geliştirilen algoritma çok hızlı bir şekilde alternatif senaryolarla test edilebilir.

Robotik sistemlerin tasarımı ve geliştirilen algoritmaların test edilebilmesi için bir çok robot simülatörü bulunmaktadır. Bu çalışmada, açık kaynak kodlu olması, C++ programlama dili desteği, robot ve ortam tasarımı konusunda oldukça esnek olması, mobil robotlardan endüstriyel robotlara kadar hem iç hem de dış ortamlarda ileri seviye simülasyonlara olanak sağlaması ve akademik çalışmalarda kullanılması (Meyer vd., 2012; Suárez-Ruiz vd., 2014; Unhelkar vd., 2014) sebebiyle Gazebo robot simülatörü tercih edilmiştir.

3.2 Gazebo

Gazebo, hem kapalı hem de açık mekanlar için geliştirilmiş, açık kaynak kodlu bir çoklu robot simülatörüdür. Birden fazla robot, sensör ve cismi üç boyutlu ortamda simüle edebilme yeteneğine sahiptir.

Gazebo projesi 2002 yılının sonbaharında Güney Kaliforniya Üniversitesi'nde Dr. Andrew Howard ve öğrencisi Nate Koenig tarafından başlatılmıştır. Böyle bir simülatör

konsepti, robotların dış ortamda değişik koşullar altında simüle edilebilme ihtiyacından kaynaklanmıştır. Başlangıçta dış ortama yönelik olarak tasarlandığı için Gazebo ismi tercih edilmiştir ancak kullanıcıların birçoğu bu simülatörü kapalı ortamlar için kullanmaktadır.

Kuruluşundan bu yana, Andrew ve Nate, Andrew JPL'e (Jet Propulsion Laboratory) katılmak için Güney Kaliforniya Üniversitesinden ayrılana kadar Gazebo üzerinde çalışmışlardır. Andrew ayrıldıktan sonra Nate bu projeye karşı hissettiği sorumluluk sayesinde robotik camiasına bu genel amaçlı aracı kazandırmak için çalışmalarına doktora kariyeri boyunca devam etmiştir.

2009 yılında "Willow Garage"da kıdemli araştırma mühendisi olan Jhon Hsu, ROS (Robot Operating System) ve PR2'yi (Cousins, 2010) Gazebo'ya entegre etmiştir. 2011 yılının ilkbaharında ise "Willow Garage" Gazebonun gelişimi için finansal destek sağlamaya başlamıştır.

3.2.1 Gazebo Sistem Gereksinimleri

Gazebo açık kaynak kodlu bir yazılımdır ve Ubuntu Linux üzerinde geliştirilmeye başlanmıştır. Bu çalışmanın yapıldığı tarihte Gazebo, OS X, Ubuntu, Debian, Fedora ve Arc Linux dağıtımları için hazır fakat Windows işletimi için çalışmalar devam etmektedir. Eğer bahsi geçenlerin dışında bir linux dağıtımı kullanılıyorsa Gazebo kaynak kodlarından derlenerek sisteme kurulmalıdır.

Gazebo kullanılırken simüle edilen ortama ve robotlara bağlı olarak işlemci (CPU) gücü ihtiyacı değişecektir. Ancak mümkün olduğunca güçlü bir işlemciye sahip olunması önerilmektedir.

Gazebo sahneleme işlemi için açık kaynak kodlu bir grafik motoru olan OGRE'yi (Object-Oriented Graphics Rendering Engine) kullanmaktadır. Bu sebepten işletim sistemi ile uyumlu güçlü bir grafik kartına sahip olmak gerekmektedir.

3.2.2 Paket Yöneticisi ile Kurulum

Bu çalışma yapılırken Gazebo Ubuntu 12.04 işletim sistemi üzerinde çalıştırılmıştır. Gazebo'nun Ubuntu işletim sistemine apt-get paket yöneticisini kullanarak kurulması için aşağıdaki adımlar takip edilmelidir.

Kullanılacak bilgisayarın packages.osrfoundation.org adresinden yazılım kabul etmesi için ayar yapılmalıdır. Bu işlem için aşağıdaki komutlar sırayla terminal ekranında çalıştırılır.

```
$ sudo sh -c 'echo "deb http://packages.osrfoundation.org/gazebo/ubuntu `lsb_release -cs` main" > /etc/apt/sources.list.d/gazebo-latest.list'

$ wget http://packages.osrfoundation.org/gazebo.key -O - | sudo apt-key add -
```

apt-get paket yöneticisini güncelleştirerek Gazebo'yu kurmak için aşağıdaki komutlar çalıştırılır.

```
$ sudo apt-get update
$ sudo apt-get install gazebo5
```

Gazebo'yu başlatmak için terminal ekranından adının yazılarak çağırılması yeterli olacaktır.

```
$ gazebo
```

3.2.3 Kaynak Koddan Derlenerek Kurulum

http://gazebosim.org/download adresinden Gazebo'nun herhangi bir versiyonunun kaynak kodları indirilir. Sıkıştırılmış dosya aşağıdaki komut ile açılır.

```
$ tar -xvf gazebo-5.0.1.tar.bz2
```

Gazebo kaynak kodlarının bulunduğu klasöre gitmek için aşağıdaki komut çalıştırılır.

```
$ cd gazebo-5.0.1
```

Aşağıdaki komutlar çalıştırılarak Gazebo için gerekli olan kütüphaneler kurulur.

```
$ sudo apt-get install build-essential libtinyxml-dev libtbb-dev libxml2-dev libqt4-dev pkg-config libprotoc-dev libfreeimage-dev

$ sudo apt-get install libprotobuf-dev protobuf-compiler libboost-all-dev freeglut3-dev cmake libogre-dev libtar-dev

$ sudo apt-get install libcurl4-openssl-dev libcegui-mk2-dev
```

Kütüphanelerin kurulumu tamamlandıktan sonra aşağıdaki komut çalıştırılarak "build" adında bir klasör oluşturulur, klasörünün içine girilir ve ardından Gazebo'nun konfigüre edilmesi sağlanır.

```
$ mkdir build
$ cd build
$ cmake ../
```

cmake ../ komutu çalıştırıldığında eksik paketler olduğuna dair bazı hatalar alınabilir. Bu paketler yüklendikten sonra cmake ../ komutu tekrar çalıştırılmalıdır. Gazebo konfigüre edildikten sonra aşağıdaki komut çalıştırılarak derlenmelidir.

```
$ make
```

Daha fazla çekirdek kullanarak derleme zamanı kısaltılabilir. Bunun için aşağıdaki komutta X yerine kullanılacak çekirdek sayısı yazılmalıdır.

```
$ make -jX
```

Derleme işlemi tamamlandıktan sonra aşağıdaki komut çalıştırılarak Gazebo işletim sistemine yüklenmelidir.

```
$ make install
```

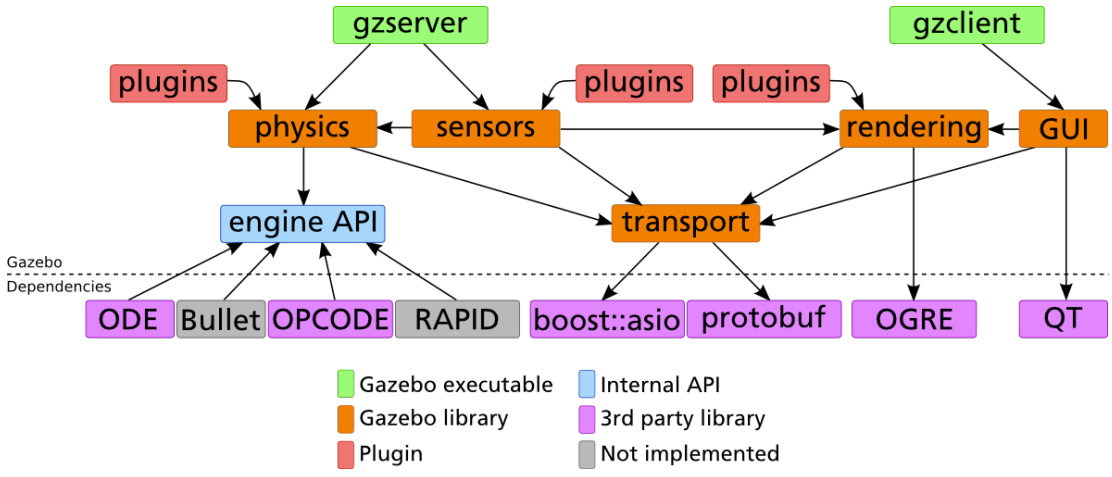
Gazebo'yu başlatmak için terminal ekranından adının yazılarak çağırılması yeterli olacaktır.

```
$ gazebo
```

3.2.4 Gazebo'nun Mimarisi

Gazebo, Şekil 3.1'de görüldüğü gibi çeşitli kütüphanelere ayrılmıştır.

- **Fizik**: Simülasyonun fiziksel durumunu günceller.
- **Sahneleme**: Simülasyon durumunu görselleştirir.
- **Sensör**: Sensör verilerini oluşturur.
- **Taşıma**: İşlemler arası iletişimi yönetir.
- **Grafik Kullanıcı Arayüzü (GUI)**: Simülasyonun görselleştirilmesi ve manipülasyonunu sağlar.



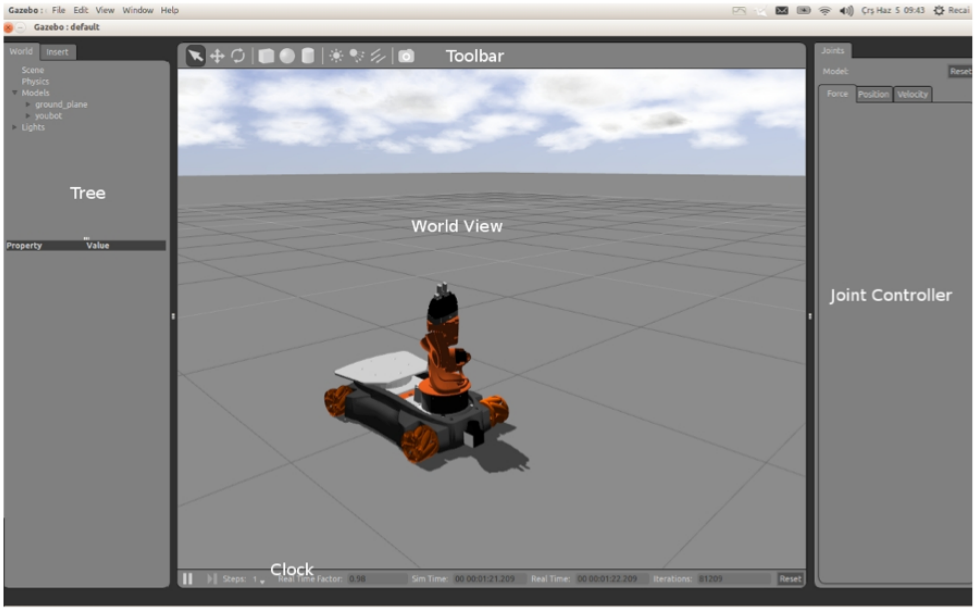
Şekil 3.1. Gazebonun Mimarisi

Bu kütüphaneler iki ana işlem tarafından kullanılmaktadır.

- **Sunucu (gzserver)**: Fizik döngülerini çalıştırır ve sensör verilerini üretir.
- **İstemci (gzclient)**: Kullanıcı etkileşimi ve simülasyonun görselleştirilmesini sağlar.

3.2.5 Grafik Kullanıcı Arayüzü

Grafik kullanıcı arayüzü (Şekil 3.2), simülasyonun gösterilmesini ve beş bileşeni aracılığıyla kullanıcı ile simülasyon arasındaki etkileşimi sağlamaktadır.



Şekil 3.2. Grafik Kullanıcı Arayüzü

World View: Oluşturulan ortamın ve modellerin görüntülendiği alandır. Bu alana model eklenebilir, kaldırabilir veya manipüle edilebilir.

Toolbar: Bu araç çubuğu kullanılarak sahneye yakınlaşıp uzaklaşılabilir, bir model sahne üzerinde taşınabilir, döndürebilir, kamera açısı ve pozisyonu değiştirilebilir, küp, silindir, küre ve ışık kaynağı eklenebilir.

Tree: Bu liste sahnedeki modellerin hiyerarşik olarak görüntülenmesini sağlar. Ayrıca sahneye yeni model eklemek için de kullanılır.
Clock: Simülasyonu durdurmak, başlatmak için kullanılır. Ayrıca simülasyonun gerçek zamanlı veya gerçek zamana göre ne kadar yavaş çalıştığını görüntüler.

Joint Controller: Bu bileşen, sahnedeki modellerin eklemlerine kuvvet uygulamak, hız kazandırmak veya açısını değiştirmek için kullanılır.

3.2.6 Komut Satırı Arayüzü

Gazebo komut satırından "gazebo" komutu verilerek çalıştırılabileceği gibi önceden kaydedilmiş bir ortam parametre olarak verilerek de başlatılabilir.

```
$ gazebo worlds/empty.world
```

Aşağıdaki gibi –u parametresi ile simülasyon duraklatılmış bir şekilde başlatılabilir.

```
$ gazebo -u worlds/empty.world
```

Gazebo server ve client olarak ayrı ayrı da başlatılabilir.

```
$ gzserver worlds/empty.world
$ gzclient
```

Ayrıca çalışan bir simülasyon hakkında bir takım bilgilerin alınabileceği "gzstats", "gztopic", "gzfactory" gibi komutlar da mevcuttur.

Örneğin, belirtilen adresteki model çalıştırıldıktan sonra komut ile o modelin öteleme (x, y, z) ve dönme (roll, pitch, yaw) değerleri verilebilir.

```
$ gzfactory spawn -f ~/.gazebo/models/bowl/model.sdf -m bowl -x -10 -y 0 -z 1 -R 0 -P 0 -Y 0
```

3.2.7 SDF (Simulation Description Format)

Gazebo bir simülasyon ortamı veya bir model hakkındaki bilgileri yüklemek veya saklamak için XML kullanır. Gazebonun geliştiricileri XML içeren SDF adındaki kendi formatlarını tanımlamışlardır. Bu format ile ilgili ayrıntılı bilgi http://gazebosim.org/sdf.html adresinden alınabilir.

3.2.8 Model

Temel olarak Gazebo için bir model, bir robotun veya çevresinin özelliklerinin tanımlanmasıdır. Modeller, SDF kullanılarak tanımlanmaktadır. Bir model basit bir şekil olabileceği gibi çok karmaşık bir robot da olabilir. Kullanıcılar kendi modellerini oluşturabilir veya çevrimiçi model veri tabanından alınan modelleri kullanabilir. Modeller hiyerarşik bir yapıda da kullanılabilir. Yani bir model başka bir modeli kendi içerisine dahil edebilir. Örneğin, yeni oluşturulan bir robota önceden tanımlanmış bir lazer mesafe algılayıcısı eklenebilir.

3.2.9 Bir Modelin Bileşenleri

Gazebo'daki bir model aşağıdaki bileşenlerden oluşur.

Link: Bir link, modelin gövdesinin fiziksel özelliklerini içerir. Her link çok sayıda "Collision" ve "Visual" bileşeni içerebilir. Ancak bir modeldeki link sayısı azaldıkça performans artacaktır. Örneğin, bir masayı birbirlerine Joint bileşenleri ile bağlanmış 4 ayak ve 1 yüzey olmak üzere 5 linkten oluşturabilirsiniz. Ancak, bu tasarım, ayaklar ile yüzeyi birleştiren eklemler hareket etmeyeceğinden gereğinden fazla kompleks olacaktır. Bu nedenle masayı 5 Collision bileşeni içeren bir linkten meydana getirmek daha doğru olacaktır.

Collision: "Collision" bileşeni çarpışma kontrolü için kullanılan bir geometriyi kapsar. Genelde az kaynak tüketmesi açısından basit bir şekil olması tercih edilir.

Visual: "Visual" bileşeni bir linkin parçalarını görselleştirmek için kullanılır. Bir link sıfır veya daha fazla "visual" bileşeni içerebilir.

Inertial: "Inertial" bileşeni bir linkin kütle, eylemsizlik matrisi gibi dinamik özelliklerinin tanımlandığı bileşendir.

Sensor: Bu bileşen, "plugin" bileşeninde kullanmak üzere ortamdan veri toplar.

Joint: Bir “joint” bileşeni iki linki birbirine bağlar.

Plugin: Bu bileşen simülasyon içerisinde modeli kontrol etmek üzere üçüncü parti olarak geliştirilmiş kütüphaneleri eklenti olarak kullanmayı sağlar.

3.2.10 Model Oluşturma

Birim kütle ve birim uzunlukta bir kutu modeli oluşturup Gazeboyu çalıştırdıktan sonra bu modeli sahneye eklemek için gerekli adımlar bu kısımda anlatılmıştır.

Aşağıdaki komut çalıştırılarak kutu adında yeni bir klasör oluşturulmuştur.

```
$ mkdir ~/.gazebo/models/kutu
```

Aşağıdaki kod herhangi bir metin editörüne yapıştırılıp ~/.gazebo/models/kutu/ yolu altına “kutu.sdf” olarak kaydedilir.

```
<?xml version='1.0'?>
<sdf version="1.3">
<model name="Kutu">
  <pose>0 0 2 0 0 0</pose>
  <static>true</static>
    <link name="link">
      <inertial>
        <mass>1.0</mass>
        <inertia>
          <ixx>1.0</ixx>
          <ixy>0.0</ixy>
          <ixz>0.0</ixz>
          <iyy>1.0</iyy>
          <iyz>0.0</iyz>
          <izz>1.0</izz>
        </inertia>
      </inertial>
      <collision name="collision">
        <geometry>
          <box>
            <size>1 1 1</size>
          </box>
        </geometry>
      </collision>
```

```
    <visual name="visual">
      <geometry>
        <box>
          <size>1 1 1</size>
        </box>
      </geometry>
    </visual>
  </link>
 </model>
</sdf>
```

Görüldüğü gibi bu model tek bir link içermektedir. Daha önceden de bahsedildiği gibi bir link bileşeni "inertial", "collision", "visual" gibi bileşenler içerebilir. Bu model bir "collision" içermekte ve bu "collision" bir "visual" bileşeni ile görselleştirilmektedir. Daha önceden bahsetmediğimiz <pose> ve <static> etiketleri göze çarpmaktadır. <pose> etiketi kutunun sahneye eklendiğinde duracağı pozisyonu ve <static> etiketi ise modele yerçekiminin etki edip etmeyeceğini ayarlamaktadır. Model tasarımı sırasında <static> etiketinin "true" olarak tanımlanması kolaylık sağlar. Tasarım tamamlandıktan sonra bu etiket "false" olarak değiştirilebilir.

Gazebo tarafından modelin görülebilmesi için **model.config** dosyası oluşturulmuştur. Bunun için aşağıdaki kod bir metin editörüne yapıştırılıp istenilen değişiklikler yapıldıktan sonra ~/.gazebo/models/kutu/ yolu altına **model.config** olarak kaydedilir.

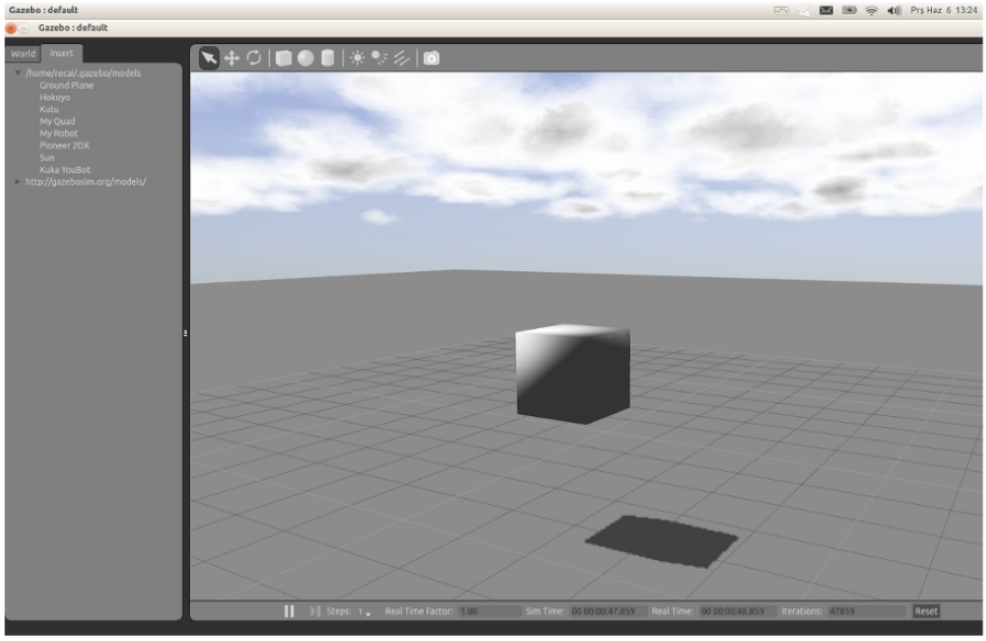
```
<?xml version="1.0"?>
<model>
  <name>Kutu</name>
  <version>1.0</version>
  <sdf version='1.3'>kutu.sdf</sdf>
  <author>
   <name>Recai Sinekli</name>
   <email>recai@sinekyazilim.com.tr</email>
</author>

  <description>
    Kutu Modeli
  </description>
</model>
```

Daha sonra bir terminal ekranından gazebo başlatılır.

```
$ gazebo
```

Gazebo açıldıktan sonra sol taraftaki “Insert” sekmesinden “Kutu” ismindeki model seçilip sahneye eklenmiş ve Şekil 3.3’teki gibi bir görünüm elde edilmiştir.



Şekil 3.3. Kutu Modeli

3.2.11 Ortam Oluşturma

Gazebo çalışılan bir ortamın “dosya_adı.world” olarak kaydedilmesine izin vermekle birlikte önceden oluşturulmuş bir ortamın da aşağıdaki şekilde çalıştırılmasına imkan tanımaktadır.

```
$ gazebo dosya_adi.world
```

Ortam oluşturmanın model oluşturmaktan çok farkı yoktur. SDF formatında yazılır ve çalıştırılır. Aşağıdaki kod bir metin editörüne yapıştırılarak “**kutu.world”** adıyla herhangi bir dizine kaydedilir.

```
<?xml version="1.0"?>
<sdf version="1.3">
  <world name="default">
    <include>
```

```
      <uri>model://ground_plane</uri>
    </include>
    <include>
      <uri>model://sun</uri>
    </include>
    <model name="Kutu">
      <pose>0 0 2 0 0 0</pose>
      <static>true</static>
      <link name="link">
        <inertial>
          <mass>1.0</mass>
          <inertia>
            <ixx>1.0</ixx>
            <ixy>0.0</ixy>
            <ixz>0.0</ixz>
            <iyy>1.0</iyy>
            <iyz>0.0</iyz>
            <izz>1.0</izz>
          </inertia>
        </inertial>
        <collision name="collision">
          <geometry>
            <box>
              <size>1 1 1</size>
            </box>
          </geometry>
        </collision>
        <visual name="visual">
          <geometry>
            <box>
              <size>1 1 1</size>
            </box>
          </geometry>
        </visual>
      </link>
    </model>
  </world>
</sdf>
```

Aşağıdaki komut ile bu ortam çalıştırılmıştır. Gazebo açıldığında daha önceden tasarlanan kutu modelinin <pose> etiketi ile belirtilen yere yerleştirildiği görülmüştür.

```
$ gazebo kutu.world
```

3.2.12 Üç Boyutlu Çizimleri Kullanarak Görselleştirme

Bu kısıma kadar yapılan tasarımlarda görselleştirme amacıyla "visual" bileşeni altında "collision" bileşeninde yapılan tanımlamanın aynısını kullanıldı. Bu kısımda, Google Sketchup veya Blender gibi 3D tasarım programlarıyla hazırlanmış bir modelin "visual" bileşeni altında nasıl kullanılacağı incelenmiştir.

Gazebo model dosyalarına görsel olarak "collada" formatındaki (.dae) dosyaları dahil edebilmektedir. Kullanılan çizim programından görsel bu formatta kaydedilmelidir. Bu kısımda gösterilen örnekte hazır bir çizim http://sketchup.google.com/3dwarehouse/ adresinden indirilerek kullanılmıştır.

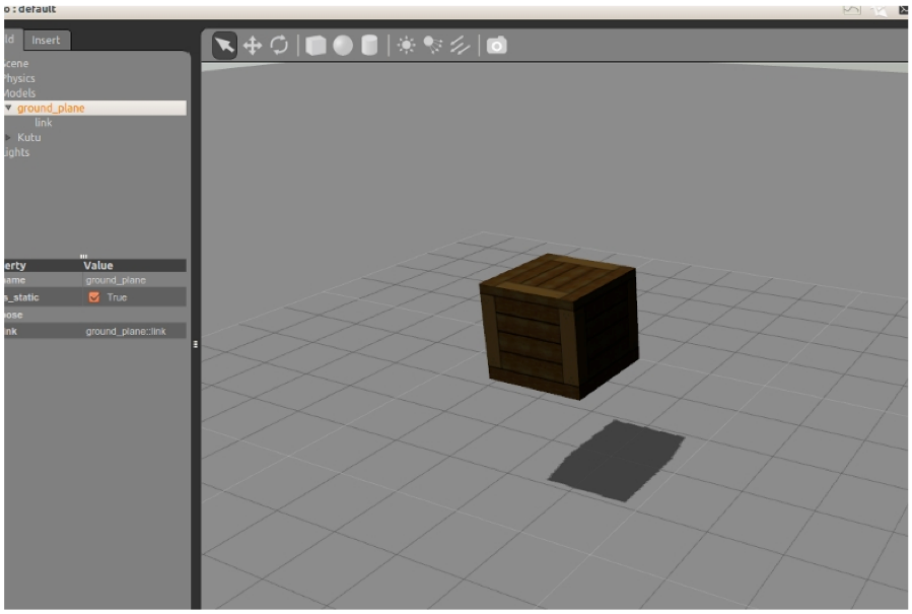
Bir kutu modeli seçilip ve collada formatında indirilmiştir. Eğer seçilen model collada formatında yüklenmemişse "sketchup" dosyası indirilerek collada formatına dönüştürülebilir.

İndirilen dosyaların arasında "dae" uzantılı bir dosya ve görselleri içeren bir klasör bulunacaktır. Daha önceden oluşturulan "kutu.world" dosyasının yanında "mesh" adında bir klasör oluşturulmuş "dae" uzantılı dosyanın adı kutu.dae olarak değiştirilmiş ve bu klasör altına taşınmıştır. Modelle ilgili görsellerin bulunduğu klasör de kutu.world dosyasının bulunduğu yere taşınmıştır. Daha sonra kutu.world dosyasındaki "visual" bileşenini aşağıdaki ile değiştirilmiştir.

```
<visual name="visual">
      <geometry>
            <mesh>
                  <uri>file://mesh/kutu.dae</uri>
                  <scale>0.5 0.5 0.5</scale>
            </mesh>
      </geometry>
</visual>
```

<mesh> etiketi arasında collada formatındaki dosyanın yolu belirtilmiş ve <scale> etiketi ile istenilen boyuta göre ölçeklenmiştir. Aşağıdaki komutla kutu.world dosyası çalıştırılmış ve Şekil 3.4'teki gibi bir görüntü elde edilmiştir.

```
$ gazebo kutu.world
```



Şekil 3.4. Görsel Eklenmiş Kutu Modeli

3.2.13 Eklentiler

Bir eklenti, paylaşımlı kütüphane olarak derlenen ve simülasyona eklenen bir C++ kodudur. Eklentiler standart C++ sınıflarını kullanarak Gazebonun tüm fonksiyonlarına erişebilirler. Gazebo simülasyonunda program aracılığıyla ortama veya bir modele müdahale etmek, algılayıcılardan veri okumak istenildiğinde eklenti kullanılmalıdır.

Dört tip eklenti vardır:

- World
- Model
- Sensor
- System

"Model" eklentisi Gazebo içerisinde herhangi bir modele, "World" eklentisi ortama, "Sensor" eklentisi bir algılayıcıya eklenir. "System" eklentisi ise komut satırı için özelleşmiş bir eklentidir. Bu nedenle istenilen fonksiyonelliğe göre uygun bir eklenti

seçilmelidir. Örneğin, ortam özellikleri değiştirilecekse bir "World" eklentisi, bir modelin durumu hakkında bilgi alınacaksa, bir modelin eklemleri kontrol edilecekse de "Model" eklentisi kullanılmalıdır.

3.2.14 Eklenti Örneği

İlk olarak basit bir "World" eklentisi oluşturulmuştur. Böylece bir eklentinin nasıl oluşturulacağı ve kullanılacağı açıklanmıştır. Bu eklenti için aşağıdaki adımlar takip edilmelidir.

Aşağıdaki komut çalıştırılarak "gazebo_eklentisi" adında bir klasör oluşturulur.

```
$ mkdir ~/gazebo_eklentisi
```

Aşağıdaki komut çalıştırılarak "hello_world.cc" dosyası oluşturulur.

```
$ gedit ~/gazebo_eklentisi/hello_world.cc
```

Aşağıdaki C++ kodu açılan metin editörüne yapıştırılarak kaydedilir.

```
#include <gazebo/gazebo.hh>
namespace gazebo
{
 class WorldPluginTutorial : public WorldPlugin
 {
   public: WorldPluginTutorial() : WorldPlugin()
           {
            printf("Hello World!\n");
           }
   public: void Load(physics::WorldPtr _world, sdf::ElementPtr_sdf)
{ }
 GZ_REGISTER_WORLD_PLUGIN(WorldPluginTutorial)
}
```

Yazılan C++ eklentisi cmake kullanılarak derlenmiştir. Eğer sistemde cmake kurulu değilse aşağıdaki komut çalıştırılarak kurulmalıdır.

```
$ sudo apt-get install cmake
```

Aşağıdaki komut çalıştırılarak CmakeLists.txt dosyası oluşturulur.

```
$ gedit ~/gazebo_eklentisi/CMakeLists.txt
```

Metin editörü açıldıktan sonra aşağıdaki satırlar yapıştırılarak kaydedilir.

```
cmake_minimum_required(VERSION 2.8 FATAL_ERROR)
include (FindPkgConfig)
if (PKG_CONFIG_FOUND)
  pkg_check_modules(GAZEBO gazebo)
endif()
include_directories(${GAZEBO_INCLUDE_DIRS})
link_directories(${GAZEBO_LIBRARY_DIRS})
add_library(hello_world SHARED hello_world.cc)
target_link_libraries(hello_world ${GAZEBO_libraries})
```

Aşağıdaki komutlar sırayla çalıştırılarak build adında bir klasör oluşturulur ve o klasör içerisine gidilir.

```
$ mkdir ~/gazebo_eklentisi/build
$ cd ~/gazebo_eklentisi/build
```

Kodu derlemek için sırayla aşağıdaki komutlar çalıştırılır.

```
$ cmake ..
$ make
```

Derleme tamamlandığında build klasörü altında **libhello_world.so** adında bir paylaşımlı kütüphane oluşturulur. Bu kütüphane bir sonraki adımda oluşturulacak hello.world dosyasının yanına taşınmalı ya da kütüphanenin erişilebilir olması için kütüphane yolu GAZEBO_PLUGIN_PATH'e eklemelidir. Bu işlemi yapmak için aşağıdaki komut çalıştırmalıdır.

```
$ export GAZEBO_PLUGIN_PATH=${GAZEBO_PLUGIN_PATH}:~/gazebo_eklentisi/build
```

Sırada bu eklentinin kullanılacağı ortam dosyasının oluşturulması var.

```
$ gedit ~/gazebo_eklentisi/hello.world
```

```
<?xml version="1.0"?>
<sdf version="1.3">
```

```
  <world name="default">
    <plugin name="hello_world" filename="libhello_world.so"/>
  </world>
</sdf>
```

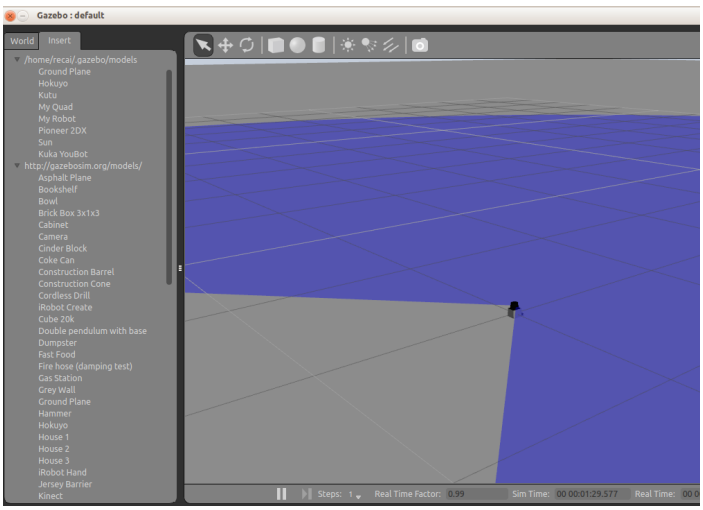
```
$ gzserver ~/gazebo_eklentisi/hello.world
```

Server başlatıldığında “Hello World!” çıktısı görülür. Böylece bir eklenti yardımıyla ekrana çıktı verilmiş olur.

3.2.15 Bir Modele Hokuyo Lazer Algılayıcı Eklenmesi

Şekil 3.5. Hokuyo Lazer Algılayıcı

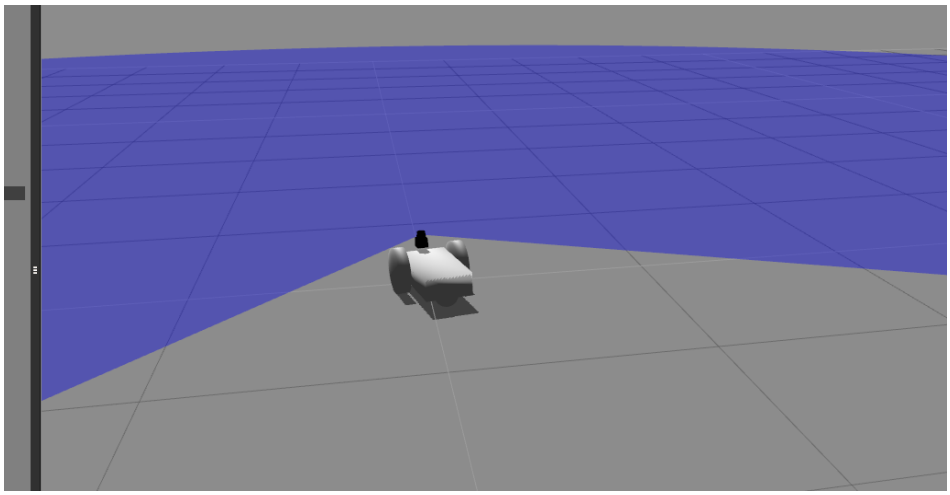
Hokuyo lazer algılayıcı (Şekil 3.5), Hokuyo Automatic tarafından üretilen, 30m ölçüm mesafesine ve 270° tarama açısına sahip bir algılayıcıdır. Bu algılayıcı Gazebo simülatör ortamında mesafe ölçmek için kullanılabilir.



Şekil 3.6. Gazebo’da Hokuyo Lazer Algılayıcı

Gazeboyu çalıştırdıktan sonra sol taraftaki "Insert" sekmesine gelerek http://gazebosim.org/models/ adresi altındaki Hokuyo algılayıcı bir kere sahneye eklenir (Şekil 3.6). Böylece bu algılayıcıya ait model dosyaları "~/.gazebo/models" dizini altına indirilmiş olur.

Şekil 3.7'de görüldüğü gibi bir robot ile Hokuyo lazer algılayıcı birleştirilerek ortamdan veri toplanabilmektedir.



Şekil 3.7. Hokuyo Lazer Algılayıcı Eklenmiş Mobil Robot

3.2.16 Gazebo ile İletişim

Gazebo, harici programlar ile TCP soket kullanarak haberleşir. Harici programlar aracılığıyla Gazebo simülasyon ortamı ile ilgili parametrelere ve modellere erişilebilir, değişiklik yapılabilir veya algılayıcı verileri okunabilir. Gazebo, iletişim katmanını yönetmek için Boost ASIO kütüphanesini, gönderilecek mesajların serileştirilmesi işlemi için ise Google tarafından geliştirilen "Protocol Buffers" veri yapısını kullanır. Gazebo'da veriler "topic" adı verilen kanallardan, yayıncılar (publishers) aracılığıyla gönderilir. Yayınlanan bu mesajın okunması istenilen tarafta ise bir abone (subscriber), mesajın yayınlandığı kanalı sürekli olarak dinler.

Gazebo simülasyon ortamından verileri dışarıya aktaracak veya dışarıdan gelen verileri okuyarak simülasyon ortamına müdahale edecek programlar kısım 3.2.13'te anlatılan

C++ programlama dili kullanılarak geliştirilen eklentilerdir. Bir eklenti, simülasyon ortamıyla haberleşebileceği gibi Gazebo kütüphanelerini kullanarak mesaj yayınlayan veya dinleyen başka programlarla da haberleşebilir. Bir eklenti yazarken haberleşmeyi sağlamak için aşağıdaki adımlar takip edilmelidir.

Eklenti, aşağıdaki fonksiyonu çalıştırarak iletişim sistemi ile ilgili gerekli ayarların yapılmasını sağlar.

```
gazebo::transport::init();
```

Eklentiler, yayıncı ve dinleyici fonksiyonlarını kullanabilmek için "Node" sınıfından bir nesne oluşturmalıdır.

```
gazebo::transport::NodePtr node(new transport::Node());
node->Init();
```

Eklentinin, Gazebo tarafından mesaj gönderilen herhangi bir kanala abone olabilmesi için "Node" sınıfından oluşturulan nesnenin "Subscribe" metodu kullanılmalıdır. Aşağıdaki örnekte, Gazebo'nun varsayılan olarak ortam bilgilerini gönderdiği bir kanal olan "world_stats" kanalına abone olması sağlanmış ve bir mesaj aldığında hangi fonksiyonu çağıracağı belirtilmiştir.

```
node->Subscribe("~/world_stats", &Class::Function, this);

void Class::Function(ConstWorldStatisticsPtr &_msg)
{
    std::cout << _msg->DebugString();
}
```

Yukarıdaki örnekte, geri dönüş fonksiyonu "world_stats" kanalından okuduğu değerleri ekrana yazdıran bir fonksiyon olarak tanımlanmıştır. Kanala abone olma işlemi tamamlandıktan sonra aşağıdaki komut çalıştırılarak iletişim sistemi başlatılır.

```
gazebo::transport::run();
```

Programın çalışması sonlandırıldığında iletişim sisteminin kapatılması için aşağıdaki komut çalıştırılmalıdır. Yazılan sınıfın yıkıcı fonksiyonu içerisinde yazılması yeterli olacaktır.

```
gazebo::transport::fini();
```

Bir eklentinin, harici bir yazılımı dinleyip, aldığı verilere göre simülasyon ortamına müdahale edebilmesi için, o harici yazılıma bir "Subscriber" aracılığı ile abone olması ve aynı zamanda bir "Publisher" aracılığı ile mesaj göndermesi gerekir. "Publisher" ve "Subscriber" türlerine "transport" ad alanı altından erişilebilirken gönderilecek mesajlar "msgs" ad alanı altındaki "GzString" türünde tanımlanmalıdır.

```
transport::PublisherPtr msgPub;
msgs::GzString strMsg;
transport::SubscriberPtr sub;
```

Bu eklenti artık yukarıda tanımlanan "msgPub" işaretçisi üzerinden mesaj gönderebilir, "sub" işaretçisi üzerinden de mesaj dinleyebilir. Tanımlanan bu işaretçilere bir atama yapılması gerekmektedir. Mesaj yayınlamak için bir kanal oluşturulmalıdır. Aşağıdaki örnekte "Node" sınıfının üyelerinden olan "Advertise" fonksiyonu çağırılmış, "custom_msg" adında bir kanal oluşturulmuş ve "msgPub" işaretçisine atanmıştır. "Node" sınıfının üyelerinden olan "Subscribe" fonksiyonu ile de harici yazılımın yayın yapacağı "command" kanalına abone olunmuştur. Böylece eklenti, "custom_msg" kanalı üzerinden yayın yapıp, harici yazılımın "command" kanalını dinleyebilecek duruma gelir.

```
msgPub = node->Advertise<msgs::GzString>("~/custom_msg");
sub = node->Subscribe("~/command", &ModelPush::callBack,this);
```

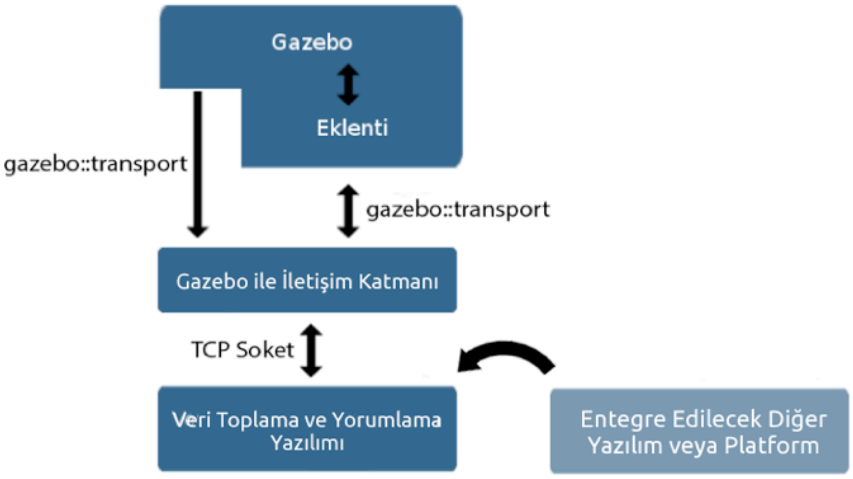
Eklentiler kısım 1.9'da anlatıldığı gibi C++ sınıfları olarak yazılırlar. Bu uygulama için eklenti, "ModelPush" isimli bir sınıf olarak yazılmıştır. Eklenti, dinlediği "command" kanalından bir veri okuduğunda "ModelPush" isimli sınıfın yani kendisinin "callBack" fonksiyonunu çağıracak ve kanaldan aldığı mesajı parametre olarak bu fonksiyona gönderecektir.

Eklenti, kendi oluşturduğu “custom_msg” kanalından mesaj göndermek istediğinde ise, “strMsg” nesnesinin “set_data” fonksiyonunu kullanarak bir veri yazar. Daha sonra, “msgPub” işaretçisi ile “transport” ad alanı altındaki “Publisher” sınıfının “Publish” fonksiyonuna erişir ve mesajı yayınlar.

```
strMsg.set_data("Mesaj eksiksiz alındı.");
msgPub->Publish(strMsg);
```

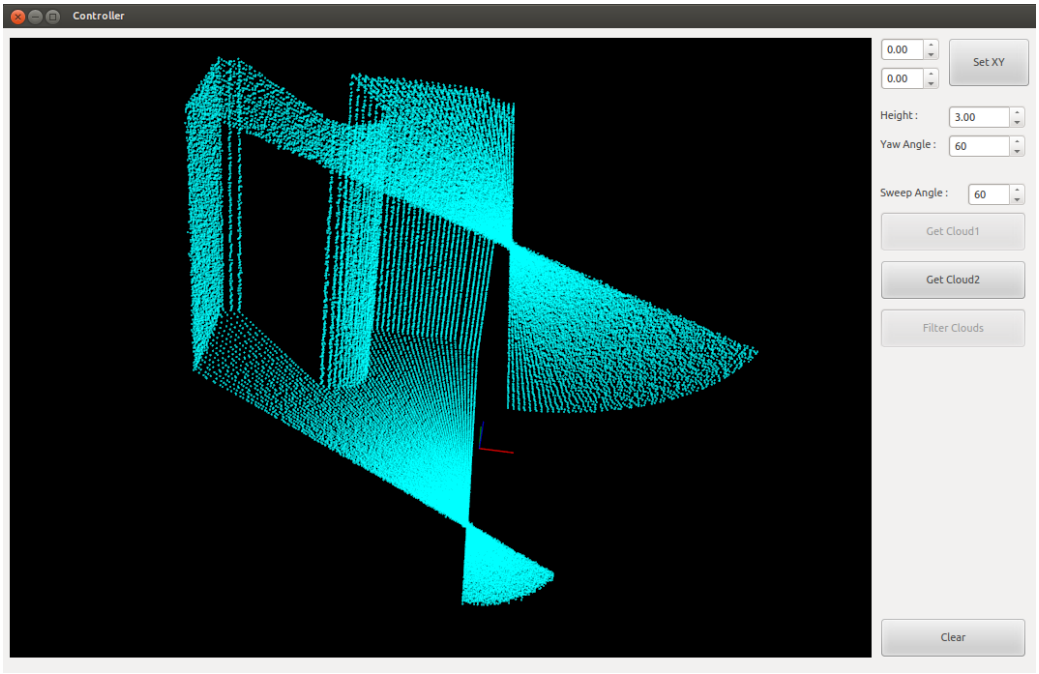
3.2.17 Gazebo ile İletişim İçin Oluşturulan Yapı

Bu iş paketinde, simülasyon ortamındaki çok rotorlu uçan platforma bağlı bir lazer algılayıcıdan veri alınıp ICP algoritması kullanılarak tarama eşleştirme yapılmıştır. Uçan platformun pozisyonunun değiştirilebilmesi için bir eklenti yazılmalı ve bu eklentiye harici yazılım tarafından konum bilgisi gönderilmelidir. Bu nedenle, Qt Creator geliştirme ortamında C++ programlama dili kullanılarak iki adet uygulama ve bir adet eklenti geliştirilmiştir. Bu yazılımlar ve simülasyon ortamı arasındaki iletişim Şekil 3.8’de gösterildiği gibidir. İlk uygulama (Şekil 3.9), Gazebo ile gerektiğinde simülasyon ortamına müdahale edecek ve verileri yorumlayacak diğer yazılım (Şekil 3.10) arasında bir ara katman görevini üstlenmektedir.



Şekil 3.8. Yazılımlar Arası İletişim Şeması

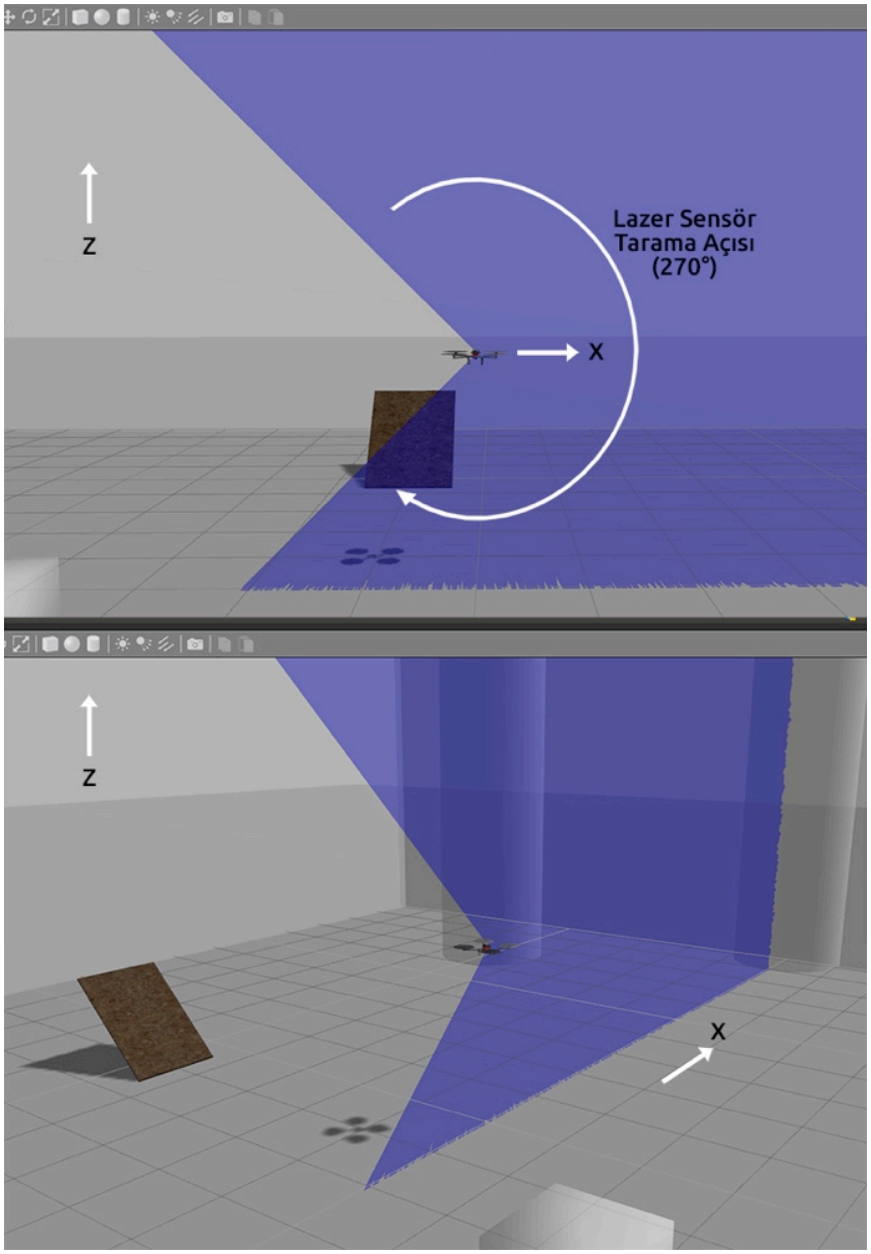
Şekil 3.9. Gazebo İletişim Katmanı



Şekil 3.10. Veri Toplama ve Yorumlama Yazılımı

Bu çalışmada, kullanılmak üzere Gazebo simülasyon ortamı için bir quadrotor modeli oluşturulmuş ve hokuyo lazer algılayıcı ile birleştirilerek kullanılmıştır. Şekil 3.11'de görülen simülasyon ortamından lazer verileri Şekil 3.9'da görülen iletişim katmanı aracılığıyla alınmakta ve Şekil 3.10'da görülen veri toplama ve yorumlama yazılımına aktarılmaktadır. Aynı zamanda veri toplama ve yorumlama yazılımından gönderilen uçan platformun pozisyon bilgileri de iletişim katmanı aracılığıyla eklentiye aktarılmaktadır. İletişim katmanından aldığı bilgileri yorumlayan eklenti, simülasyon ortamına müdahale ederek uçan platformun pozisyonunu güncellemektedir. Bu iki katmanlı tasarım Şekil 3.8'de gösterildiği üzere, veri toplama ve yorumlama görevini

üstlenen yazılım yerine başka yazılımların veya platformların kolayca entegre edilebilmesini sağlamaktadır.



Şekil 3.11. Gazebo Simülasyon Ortamından Bir Görüntü

BÖLÜM IV

ALGORİTMANIN TEST EDİLMESİ

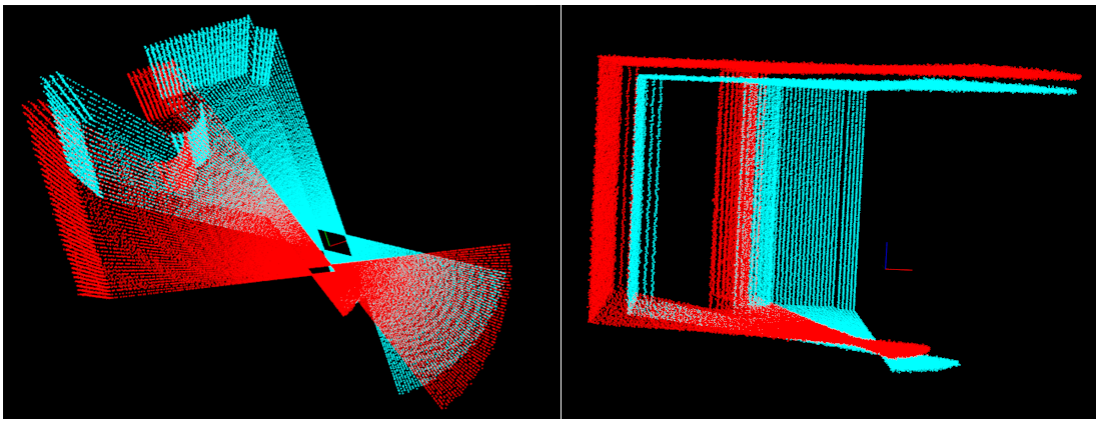
4.1 ICP Algoritmasının Testleri

Bölüm 2.3'te anlatılan ICP algoritması Qt Creator geliştirme ortamında C++ programlama dili kullanılarak kodlanmıştır. Kodlanan algoritmanın doğruluğu hem simülasyon ortamından alınan hem de ETHZ – ASL (Autonomous Systems Lab) tarafından oluşturulan veri setleri kullanılarak test edilmiştir (Pomerleau vd., 2012a).

4.2 Simülasyon Ortamı Verileri ile Yapılan Testler

Bu çalışmada kullanılmak üzere Gazebo simülasyon ortamı için bir quadrotor modeli oluşturulmuş ve hokuyo lazer algılayıcı ile birleştirilerek kullanılmıştır. Hokuyo lazer algılayıcı iki boyutlu verileri kullanılarak üç boyutlu nokta bulutu oluşturulabilmesi için algılayıcı, quadrotor üzerine Şekil 3.11'de görüldüğü gibi, tarama açısı quadrotorun yunuslama açısıyla aynı olacak şekilde yerleştirilmiştir. Böylece, quadrotor, z-ekseninde 1'er derece aralıklarla döndürülerek üç boyutlu nokta bulutu oluşturulmuştur. Simülasyon ortamından toplanan veriler ortalama değeri sıfır, standart sapması 5 cm olan gauss gürültüsü eklenmiş verilerdir.

Simülasyon ortamından veri toplama işlemi sırasında quadrotor, iki farklı konumda, z-ekseninde döndürülerek nokta bulutları oluşturulmuştur. Bu şekilde toplanan nokta bulutları ICP algoritması ile eşleştirilerek bulunan sonuçlar gerçek dönme öteleme değerleri ile karşılaştırılmıştır. Şekil 4.1'de gösterilen örnekte, quadrotor ilk konumunda ve x, y eksenlerinde 1'er metre, z-ekseninde -0.5m hareket ettikten sonra 60 derecelik bir açıyı tarayarak iki farklı üç boyutlu nokta bulutu oluşturulmuştur. Bu nokta bulutlarının 40 derecelik bölümleri kesişmektedir. ICP algoritmasının başarıya ulaşabilmesi için iki nokta bulutu arasında kesişen bölümler olmak zorundadır. İkinci konumda oluşturulan nokta bulutunu birinci konumdaki ile eşleştirecek transformasyon matrisi ICP algoritması ile bulunmuştur.



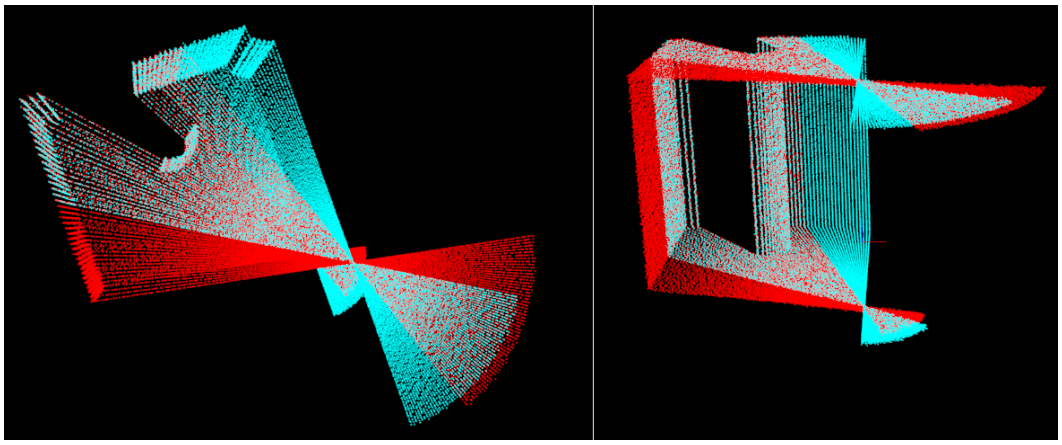
Şekil 4.1. Simülasyon Ortamından Toplanan Verilerle Oluşturulan İki Nokta Bulutunun Farklı Açılardan Görünümü

Bu transformasyon matrisinde denklem 2.15'te belirtilen öteleme vektörüne bakılarak eşleşmenin hangi oranda başarılı olduğu anlaşılabilir. Bu oran Tablo 4.1'de verilmiştir.

Çizelge 4.1. ICP Algoritmasının Başarımı

	x	y	z
Gerçek Öteleme (m)	1	1	-0.5
Bulunan Öteleme (m)	0.994372	1.00409	-0.501881
Hata (%)	0.5628	0.409	0.1881

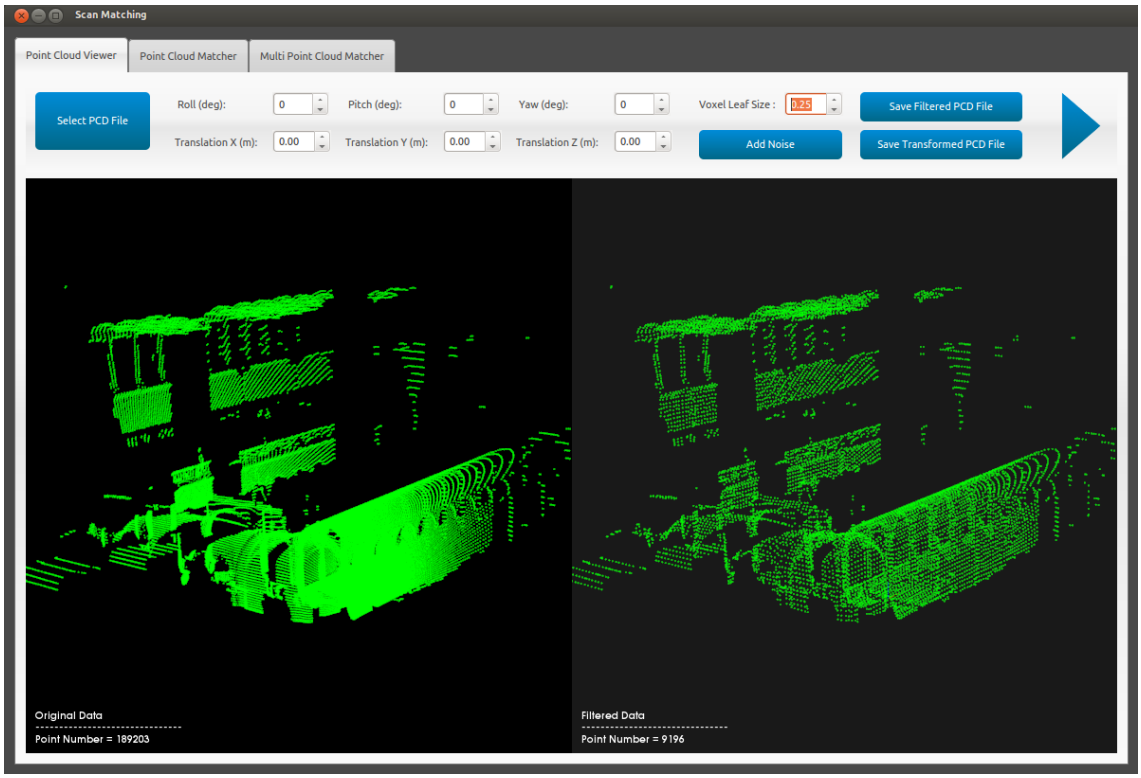
ICP algoritması ile bulunan transformasyon matrisi kaynak nokta bulutuna uygulanması ile elde edilen nokta bulutlarının eşleştirilmiş hali Şekil 4.2'de görülmektedir.



Şekil 4.2. ICP Algoritması ile Eşleştirilmiş Nokta Bulutlarının Farklı Açılardan Görünümü

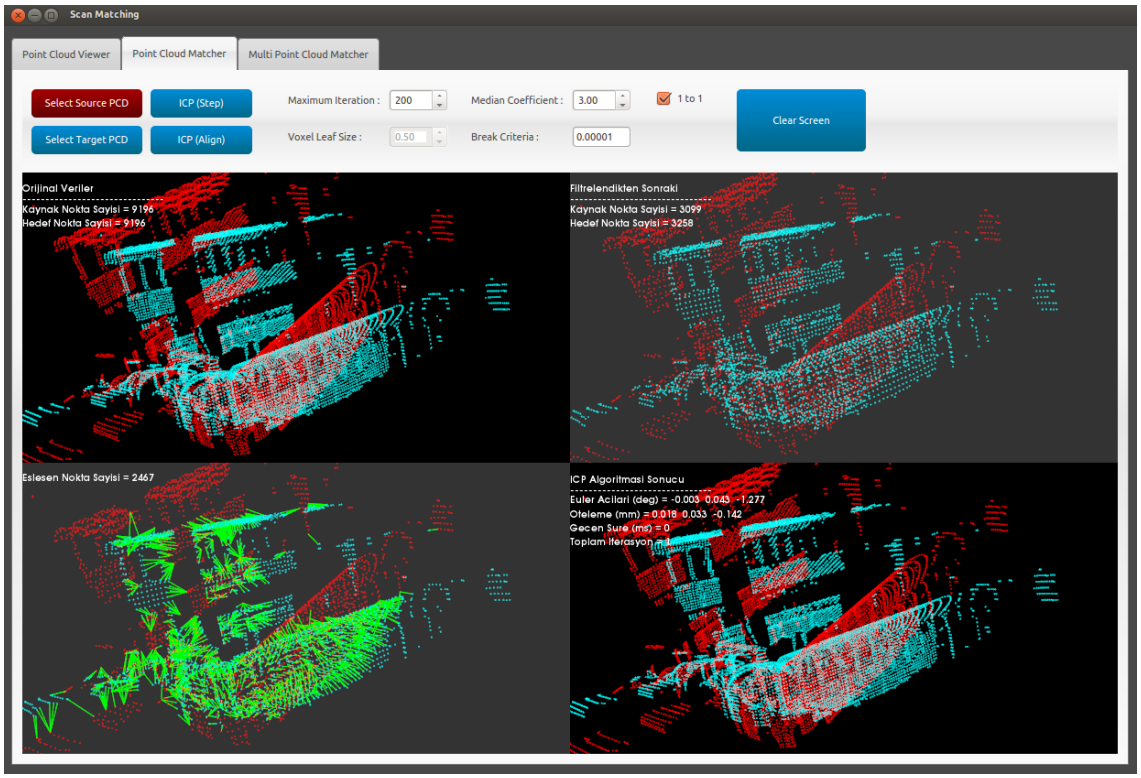
4.3 Tarama Eşleştirme Yazılımı

Bu çalışmada, bir dosyadan nokta bulutu okuma, nokta bulutunu filtreleme, dönme ve öteleme işlemlerine tabi tutma, işlem görmüş nokta bulutunu kaydetme, iki veya daha fazla nokta bulutunu ICP algoritması ile eşleştirme işlemlerinin yapılabilmesi için bir yazılım (Şekil 4.3) geliştirilmiştir.

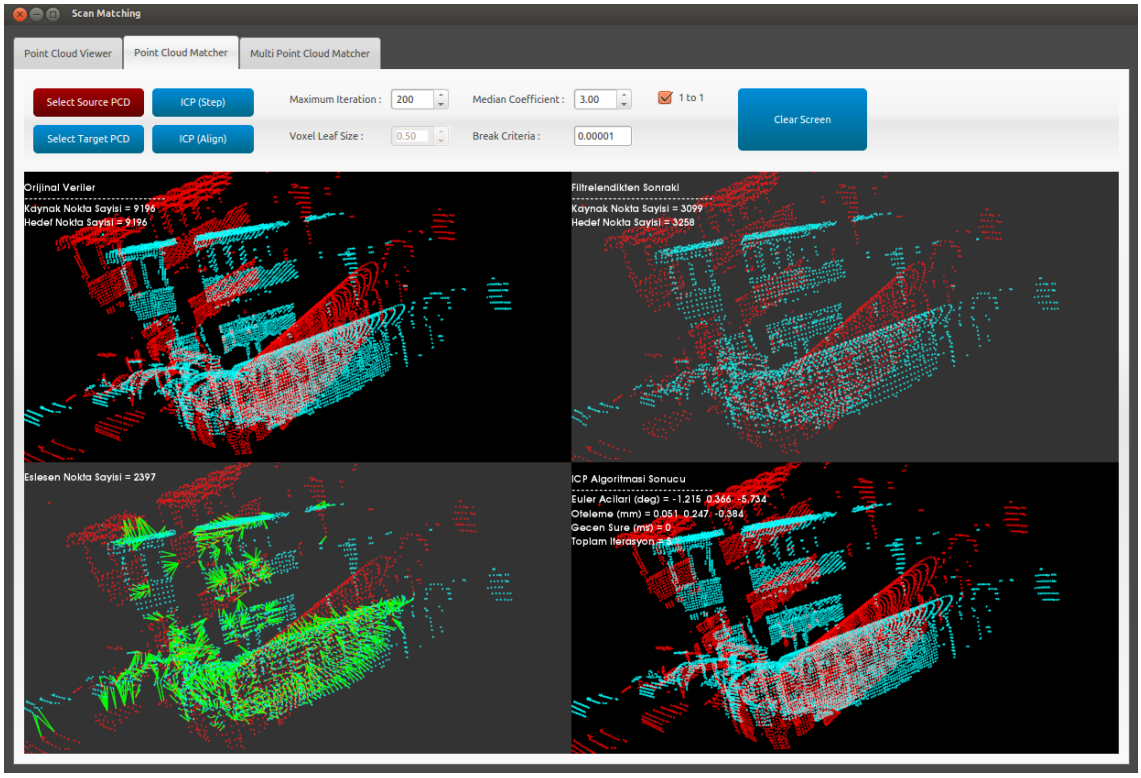


Şekil 4.3. Geliştirilen Tarama Eşleştirme Yazılımı

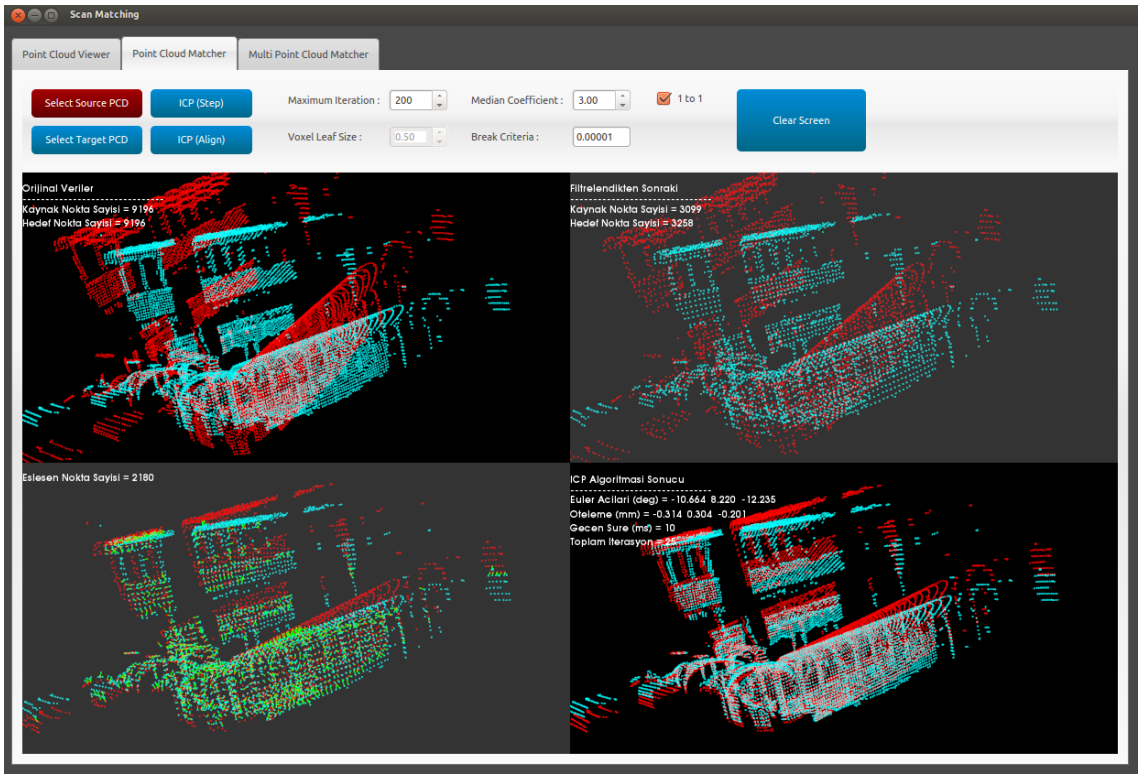
Bu yazılım kullanılarak nokta bulutları eşleştirilirken algoritmanın adım adım çalıştırılma özelliği sayesinde her tekrarlamada ilişkili noktalar takip edilebilir. Böylece eşleştirmenin gidişatı hakkında bir yorum yapılabilir ve gerekirse algoritma parametrelerinde değişiklik yapılarak daha iyi bir sonuca varılabilir. Şekil 4.4, Şekil 4.5, Şekil 4.6 ve Şekil 4.7'de bir çift nokta bulutunun adım adım eşleştirilmesinden bir kaç adım gösterilmiştir.



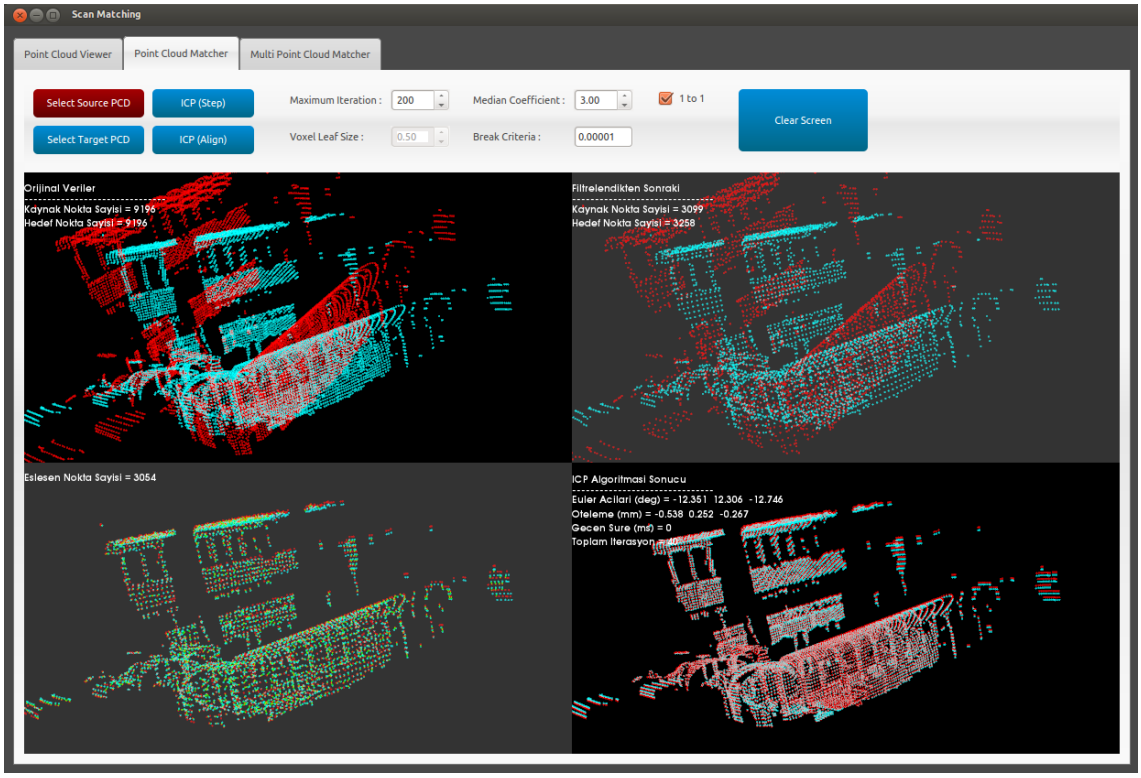
Şekil 4.4. ICP Algoritması 1. Adım



Şekil 4.5. ICP Algoritması 5. Adım



Şekil 4.6. ICP Algoritması 25. Adım



Şekil 4.7. ICP Algoritması 40. Adım

Geliştirilen bu yazılım ile gerçek dönme ve ötelenme değerleri bilinen nokta bulutları üzerinde testler yapılmış ve bulunan sonuçlar üzerinden filtreleme adımı için filtre boyutu ve aykırı nokta bulma adımında kullanılacak medyan katsayısı için uygun aralıklar belirlenmiştir.

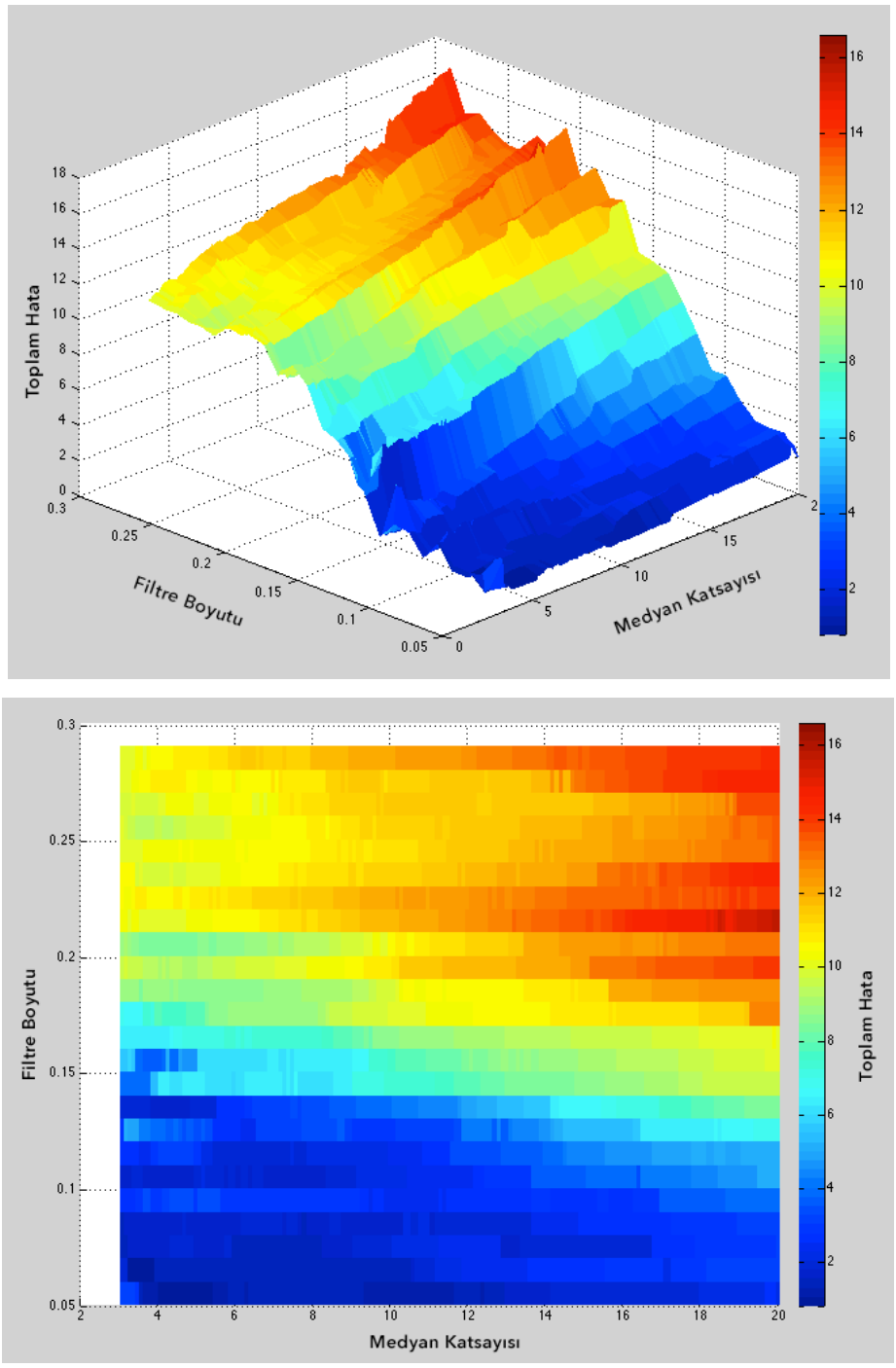
4.4 ETHZ – ASL Veri Setleri Kullanılarak Yapılan Testler

ETHZ – ASL (Autonomous Systems Lab) tarafından oluşturulan veri setlerinden iki tanesi seçilmiş ve bunlar üzerinde testler yapılmıştır (Pomerleau vd., 2012b). Bu veri setlerinden bir tanesi her bir taramada ortalama 191000 nokta barındıran 31 tarama içermektedir. Bir bina içinde ve dışında taramalar yapılarak hem iç hem de dış ortam verilerini ihtiva eden bir veri seti oluşturulmuştur. Ayrıca taramalar esnasında hareket eden insanlar da bulunduğundan oldukça zorlu bir veri setidir. İkinci veri seti de her bir taramada ortalama 191000 nokta barındırmakta ve toplam 36 tarama içermektedir. Sadece iç ortamda tarama yapılarak oluşturulan bu veri seti ise birbirine çok benzeyen tekrarlı yapıları barındırdığından dolayı algoritmanın gürbüzlüğünün test edilebilmesi için kullanıma uygun zorlu bir veri setidir.

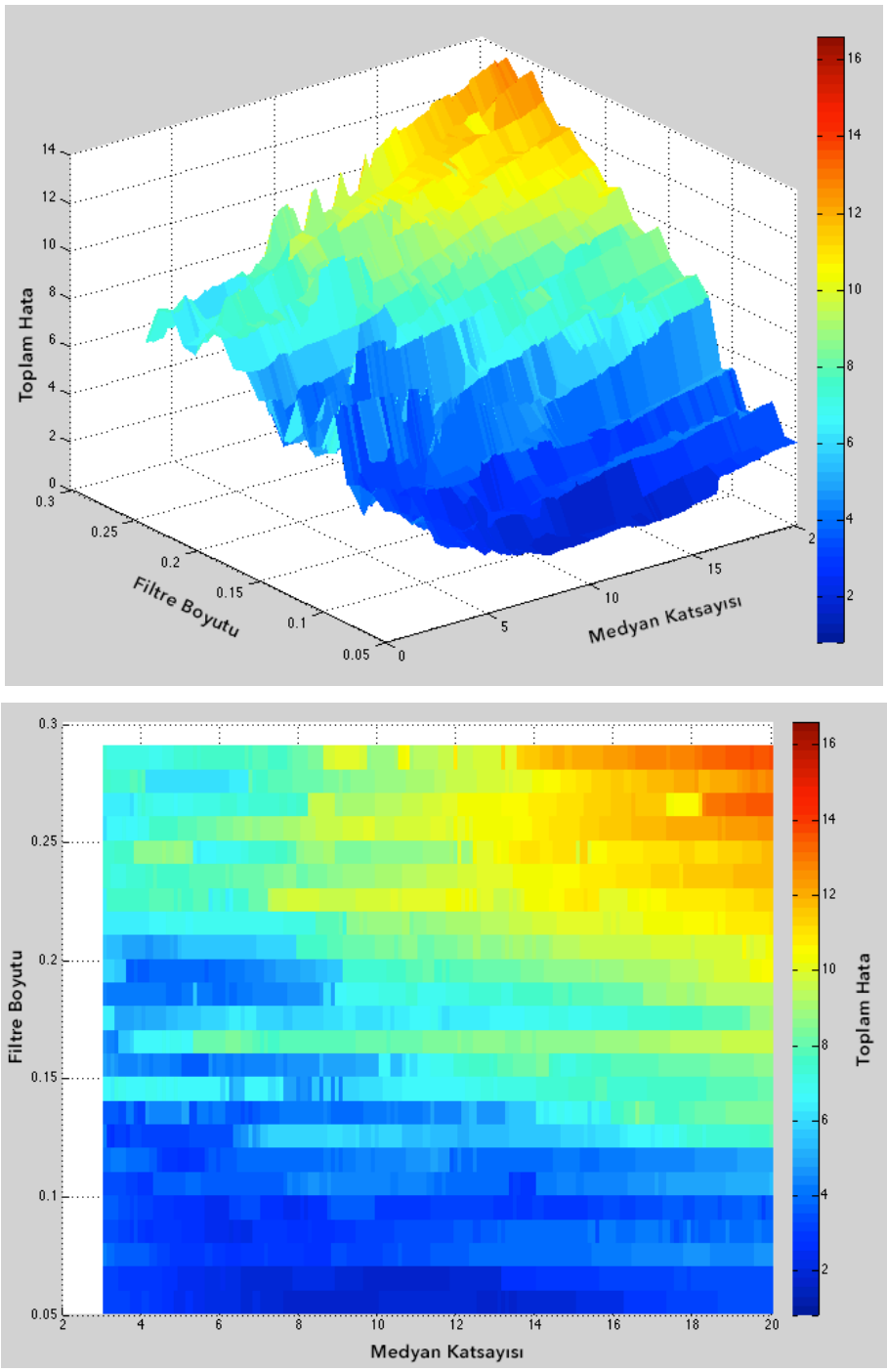
Bu veri setleri oluşturulurken Hokuyo UTM-30LX lazer algılayıcı, Xsens MTi-G ataletsel ölçüm birimi ve GPS, Leica TS15 teodolit kullanılmıştır. Bu sayede tarama verilerinin yanında lazer algılayıcının pozisyon bilgisi de hassas bir şekilde elde edilmiştir. Bu bilgilere -2.5 derece ile 2.5 derece aralığında rastgele bir dönme ve -10 cm ile 10 cm aralığında rastgele bir öteleme değeri eklenerek gürültülü IMU verisi oluşturulmuştur. Bu gürültülü IMU verisi ICP algoritması için başlangıç transformasyonu olarak kullanılmıştır. Ayrıca gerçek pozisyon bilgileri kullanılarak tarama eşleştirme sonuçlarındaki hata miktarları hesaplanabilmiştir.

Geliştirilen tarama eşleştirme yazılımı ara yüzünde filtre boyutu ve medyan katsayısı istenilen değere ayarlanarak ICP algoritması koşturulabilmektedir. Bu çalışmada, bu iki parametrenin değişiminin tarama eşleştirme sonuçlarına nasıl etki ettiği gözlemlenmiştir. Her iki veri seti için de ardışık taramalar arasında eşleştirme işlemi gerçekleştirilmiştir. Her eşleştirme 3 ile 20 arasında 0.1 adım aralığı ile 171 farklı

medyan katsayısı ve 0.05 ile 0.3 arasında 0.01 adım aralığı ile 25 farklı filtre boyutu ile test edilmiştir. Bu da her bir veri seti için 4275 farklı ICP kombinasyonu demektir. Bu testlerin sonuçları gerçek pozisyon bilgileri ile kıyaslanarak bir hata ölçümü yapılmış ve bu hata değerleri Şekil 4.8 ve Şekil 4.9'da gösterilmiştir.



Şekil 4.8. Birinci Veri Setindeki Tüm Tarama Eşleştirmeler İçin Toplam Hatalar



Şekil 4.9. İkinci Veri Setindeki Tüm Tarama Eşleştirmeler İçin Toplam Hatalar

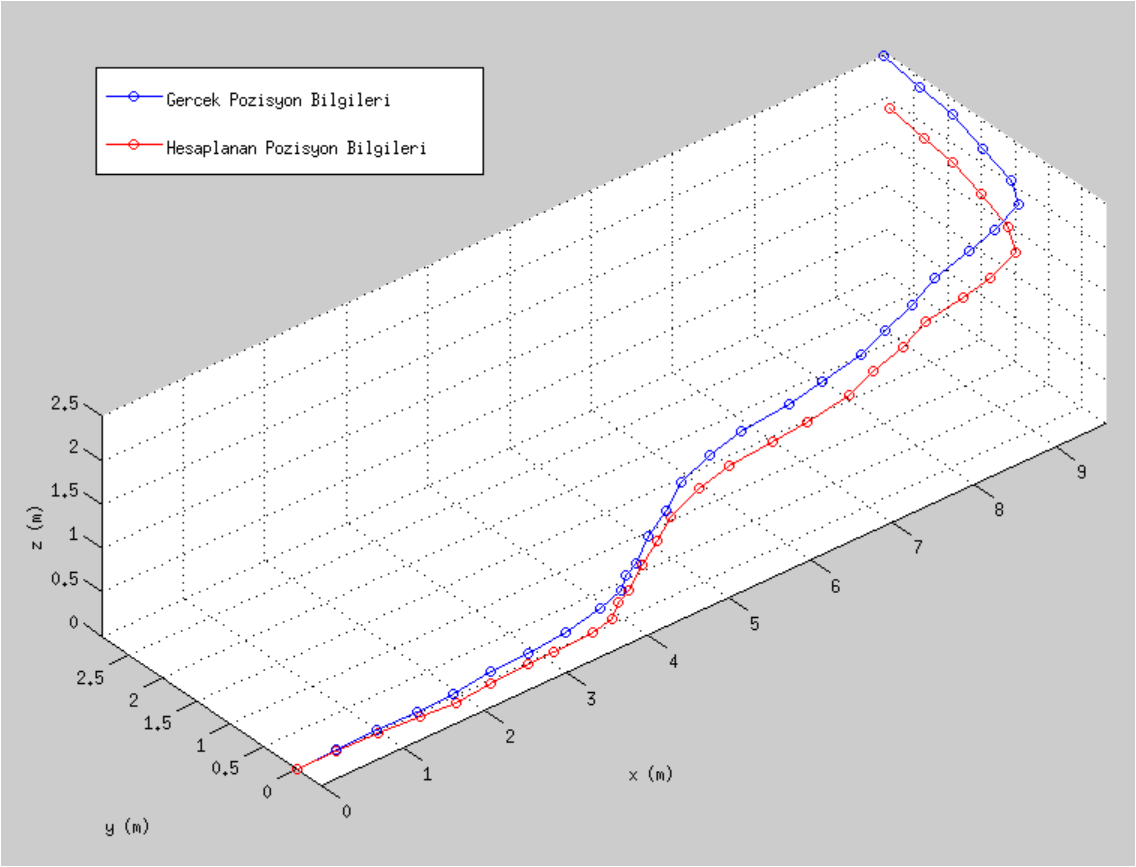
Her iki veri setinde yapılan eşleştirme sonuçlarına göre en az hatanın görüldüğü bölgeler koyu mavi, en çok hatanın görüldüğü bölgeler ise kırmızı olarak gösterilmiştir.

Seçilen filtre boyutu ve medyan katsayısı değerlerine göre bulunan pozisyon bilgilerinden bazıları gerçek değerlerle karşılaştırmalı olarak verilmiştir. Bunlardan ilki medyan katsayısının 3, filtre boyutunun 0.25 m olarak seçildiği durumdur ve sonuçlar Çizelge 4.2 ile Şekil 4.10'da gösterilmiştir.

Çizelge 4.2. Birinci Veri Seti: 3*Medyan, 0.25 m Filtre Boyutu için Gerçek ve Hesaplanan Pozisyon Bilgileri

	Gerçek Değerler			**Hesaplanan Değerler**		
Pozisyon	**x (m)**	**y (m)**	**z (m)**	**x (m)**	**y (m)**	**z (m)**
1	0	0	0	0	0	0
2	0.432	-0.05082	0.06705	0.4318	-0.05461	0.06138
3	0.8806	-0.1118	0.1385	0.8887	-0.1251	0.09469
4	1.306	-0.1964	0.203	1.313	-0.2195	0.1485
5	1.697	-0.2505	0.2628	1.697	-0.3046	0.1795
6	2.158	-0.2512	0.3192	2.139	-0.2833	0.2169
7	2.574	-0.295	0.3765	2.57	-0.3187	0.2574
8	3.004	-0.333	0.4427	2.876	-0.3231	0.2706
9	3.451	-0.3141	0.5133	3.354	-0.3126	0.2903
10	3.752	-0.2629	0.5625	3.624	-0.2775	0.3066
11	3.825	-0.2324	0.6966	3.717	-0.2523	0.443
12	3.951	-0.2316	0.7818	3.849	-0.2521	0.5265
13	4.103	-0.2231	1.01	4.011	-0.2503	0.7326
14	4.305	-0.2661	1.244	4.19	-0.2721	0.9464
15	4.489	-0.2569	1.477	4.356	-0.269	1.155
16	4.803	-0.2901	1.678	4.68	-0.2934	1.354
17	5.17	-0.329	1.806	5.018	-0.3184	1.48
18	5.721	-0.3582	1.896	5.538	-0.341	1.528
19	6.134	-0.3439	1.963	5.976	-0.3282	1.568
20	6.663	-0.2878	2.014	6.514	-0.2916	1.628
21	7.049	-0.1729	2.063	6.903	-0.188	1.676
22	7.467	-0.08461	2.129	7.336	-0.1041	1.717
23	7.859	0.07219	2.191	7.732	0.04877	1.758

24	8.337	0.1389	2.262	8.221	0.08547	1.808
25	8.691	0.1907	2.313	8.586	0.125	1.849
26	9.086	0.3042	2.372	8.981	0.2287	1.92
27	9.255	0.6319	2.407	9.159	0.5557	1.962
28	9.365	1.175	2.433	9.298	1.114	1.984
29	9.492	1.777	2.465	9.455	1.726	1.965
30	9.525	2.31	2.482	9.519	2.235	1.936
31	9.578	2.916	2.505	9.614	2.855	1.922



Şekil 4.10. Birinci Veri Seti: 3*Medyan, 0.25 m Filtre Boyutu için Gerçek ve Hesaplanan Pozisyon Bilgileri

Hesaplanan ve gerçek pozisyon bilgileri arasındaki mesafeye bakılarak her pozisyonda yapılan hatalar hesaplanmış ve Çizelge 4.3'te verilmiştir.

Çizelge 4.3. Birinci Veri Seti: 3*Medyan, 0.25 m Filtre Boyutu için Hata Miktarları, İterasyon Sayıları ve İşlem Süreler

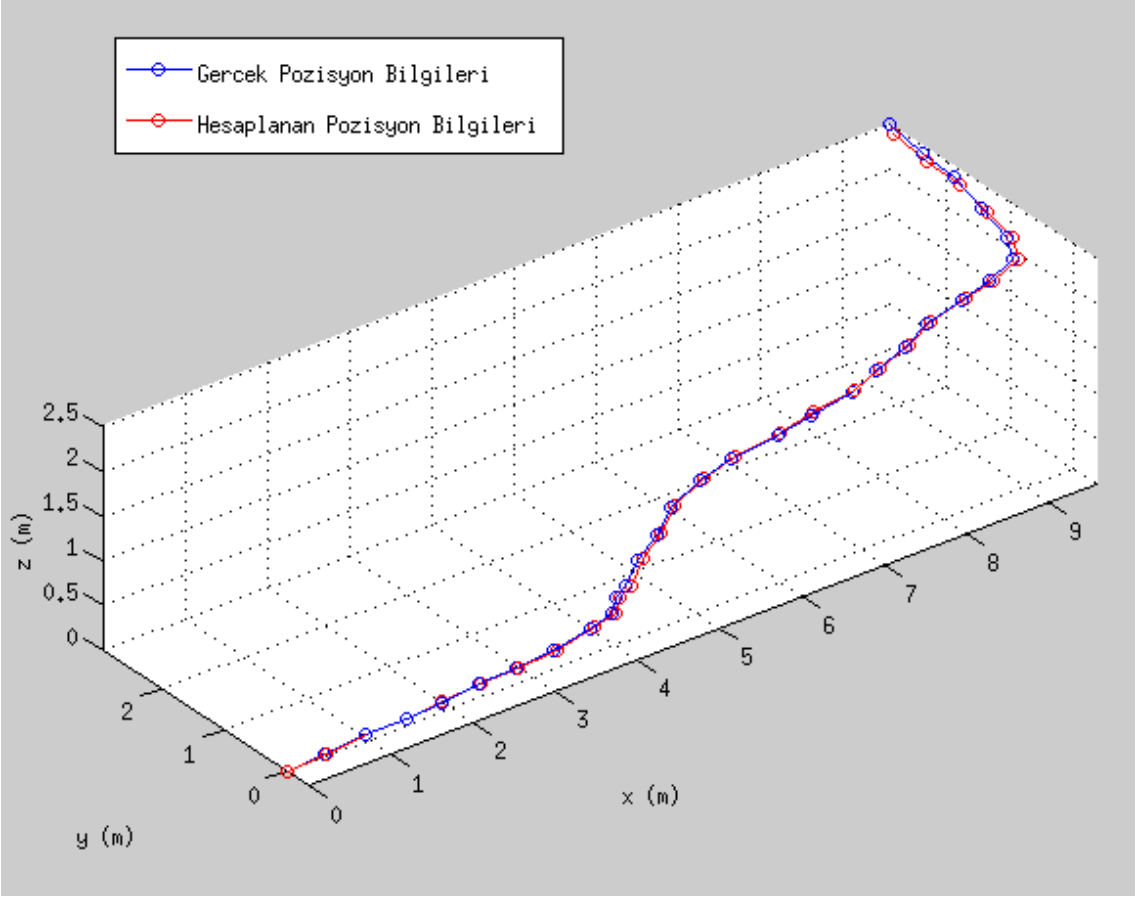
Pozisyon	Hata (m)	İterasyon Sayısı	İşlem Süresi (ms)
1	0	0	0
2	0.0109	79	630
3	0.0464	26	270
4	0.0596	30	220
5	0.0994	26	190
6	0.1088	23	210
7	0.1215	22	50
8	0.2147	200	460
9	0.2434	9	20
10	0.2866	12	20
11	0.2764	36	60
12	0.2756	18	30
13	0.2933	11	30
14	0.3193	8	20
15	0.3485	22	70
16	0.3462	35	90
17	0.3598	22	50
18	0.4107	68	140
19	0.4258	137	320
20	0.4139	21	50
21	0.4139	42	110
22	0.4326	21	60
23	0.4515	13	50
24	0.4713	33	160
25	0.4807	29	200
26	0.4706	31	340
27	0.4611	23	300
28	0.4578	40	370
29	0.5042	26	240
30	0.5514	22	190
31	0.5896	70	560

Çizelge 4.3'te verilen bilgilere göre medyan katsayısının 3, filtre boyutunun 0.25 m seçildiği durum için ortalama hata, ortalama iterasyon sayısı ve ortalama işlem süresi Çizelge 4.4'te verilmiştir.

Çizelge 4.4. Birinci Veri Seti: 3*Medyan, 0.25 m Filtre Boyutu için Ortalama Hata, İterasyon Sayısı ve İşlem Süresi

Ortalama Hata (m)	Ortalama İterasyon Sayısı	Ortalama İşlem Süresi (ms)
0.3208	38.5	183.67

Medyan katsayısının 3, filtre boyutunun ise 0.1 m seçildiği durumda elde edilen sonuçlar Şekil 4.11'de, hata miktarları ise Çizelge 4.5'te verilmiştir.



Şekil 4.11. Birinci Veri Seti: 3*Medyan, 0.1 m Filtre Boyutu için Gerçek ve Hesaplanan Pozisyon Bilgileri

Çizelge 4.5. Birinci Veri Seti: 3*Medyan, 0.1 m Filtre Boyutu için Hata Miktarları, İterasyon Sayıları ve İşlem Süreleri

Pozisyon	Hata (m)	İterasyon Sayısı	İşlem Süresi (ms)
1	0	0	0
2	0.0109	79	630
3	0.0464	26	270
4	0.0596	30	220
5	0.0994	26	190
6	0.1088	23	210
7	0.1215	22	50
8	0.2147	200	460
9	0.2434	9	20
10	0.2866	12	20
11	0.2764	36	60
12	0.2756	18	30
13	0.2933	11	30
14	0.3193	8	20
15	0.3485	22	70
16	0.3462	35	90
17	0.3598	22	50
18	0.4107	68	140
19	0.4258	137	320
20	0.4139	21	50
21	0.4139	42	110
22	0.4326	21	60
23	0.4515	13	50
24	0.4713	33	160
25	0.4807	29	200
26	0.4706	31	340
27	0.4611	23	300
28	0.4578	40	370
29	0.5042	26	240
30	0.5514	22	190
31	0.5896	70	560

Çizelge 4.5'te verilen bilgilere göre medyan katsayısının 3, filtre boyutunun 0.1 m seçildiği durum için ortalama hata, ortalama iterasyon sayısı ve ortalama işlem süresi Çizelge 4.6'da verilmiştir.

Çizelge 4.6. Birinci Veri Seti: 3*Medyan, 0.1 m Filtre Boyutu için Ortalama Hata, İterasyon Sayısı ve İşlem Süresi

Ortalama Hata (m)	Ortalama İterasyon Sayısı	Ortalama İşlem Süresi (ms)
0.0531	41.47	3284

Medyan katsayısının 3, filtre boyutunun 0.06 m seçildiği bir durum için ortalama hata, ortalama iterasyon ve ortalama işlem süresi Çizelge 4.7'de verilmiştir.

Çizelge 4.7. Birinci Veri Seti: 3*Medyan, 0.06 m Filtre Boyutu için Ortalama Hata, İterasyon Sayısı ve İşlem Süresi

Ortalama Hata (m)	Ortalama İterasyon Sayısı	Ortalama İşlem Süresi (ms)
0.0623	49.1	15748

Medyan katsayısının 3.3, filtre boyutunun 0.06 m seçildiği bir durum için ortalama hata, ortalama iterasyon ve ortalama işlem süresi Çizelge 4.8'de verilmiştir.

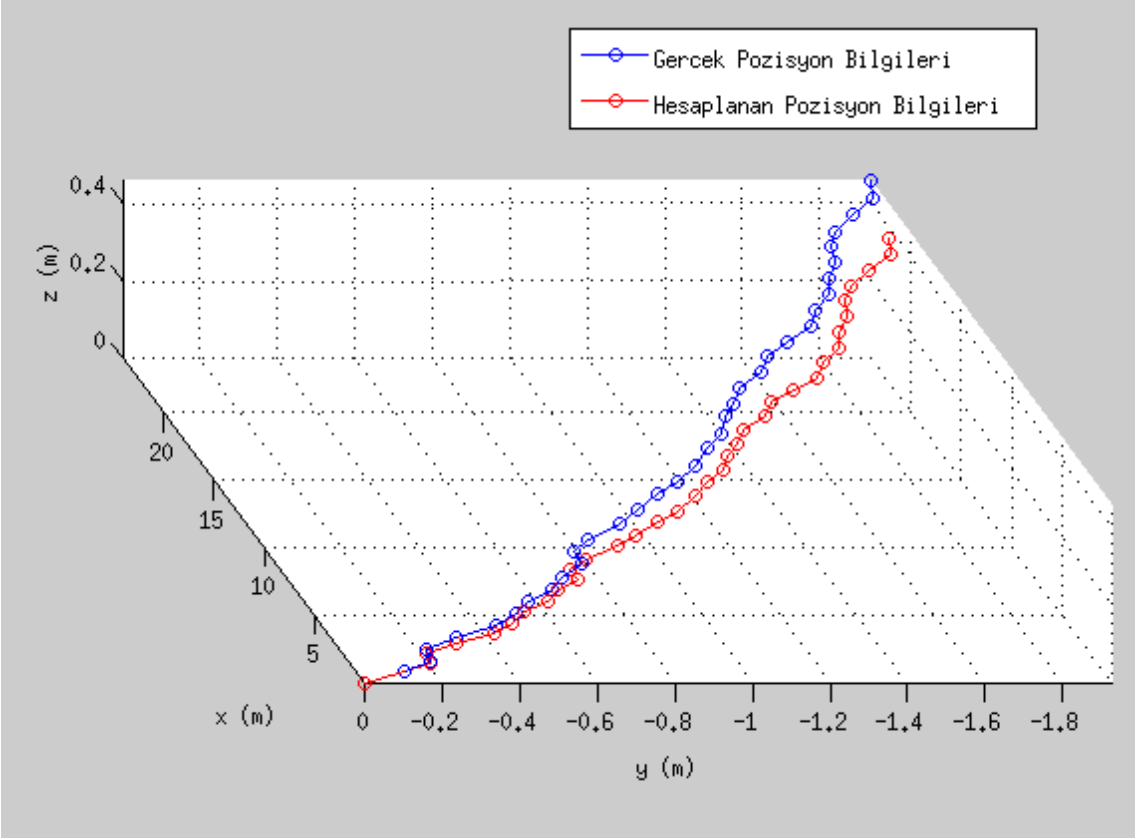
Çizelge 4.8. Birinci Veri Seti: 3.3*Medyan, 0.06 m Filtre Boyutu için Ortalama Hata, İterasyon Sayısı ve İşlem Süresi

Ortalama Hata (m)	Ortalama İterasyon Sayısı	Ortalama İşlem Süresi (ms)
0.0267	50.6	14664

İkinci veri seti için de seçilen filtre boyutu ve medyan katsayısı değerlerine göre bulunan pozisyon bilgilerinden bazıları gerçek değerlerle karşılaştırmalı olarak verilmiştir. Bunlardan ilki medyan katsayısının 3, filtre boyutunun 0.05 m olarak seçildiği durumdur ve sonuçlar Çizelge 4.9 ile Şekil 4.12'de gösterilmiştir.

Çizelge 4.9. İkinci Veri Seti: 3*Medyan, 0.05 m Filtre Boyutu için Ortalama Hata, İterasyon Sayısı ve İşlem Süresi

Ortalama Hata (m)	Ortalama İterasyon Sayısı	Ortalama İşlem Süresi (ms)
0.0882	71.7	84893

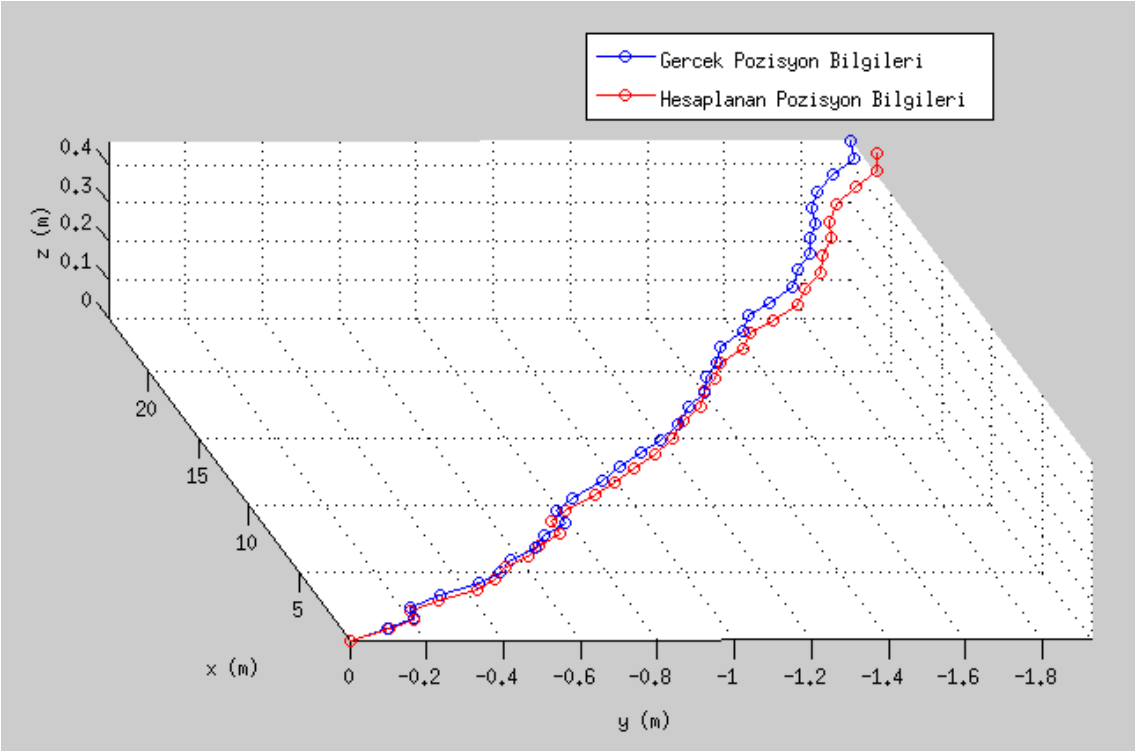


Şekil 4.12. İkinci Veri Seti: 3*Medyan, 0.05 m Filtre Boyutu için Gerçek ve Hesaplanan Pozisyon Bilgileri

Medyan katsayısının 8, filtre boyutunun 0.05 m seçildiği bir durum için ise pozsiyon bilgileri Şekil 4.13'te, ortalama hata, ortalama iterasyon ve ortalama işlem süresi Çizelge 4.10'da verilmiştir.

Çizelge 4.10. İkinci Veri Seti: 8*Medyan, 0.05 m Filtre Boyutu için Ortalama Hata, İterasyon Sayısı ve İşlem Süresi

Ortalama Hata (m)	Ortalama İterasyon Sayısı	Ortalama İşlem Süresi (ms)
0.0467	72.34	62806



Şekil 4.13. İkinci Veri Seti: 8*Medyan, 0.05 m Filtre Boyutu için Gerçek ve Hesaplanan Pozisyon Bilgileri

Medyan katsayısının 10, filtre boyutunun 0.2 m seçildiği bir durum için ortalama hata, ortalama iterasyon ve ortalama işlem süresi Çizelge 4.11'de verilmiştir.

Çizelge 4.11. İkinci Veri Seti: 10*Medyan, 0.2 m Filtre Boyutu için Ortalama Hata, İterasyon Sayısı ve İşlem Süresi

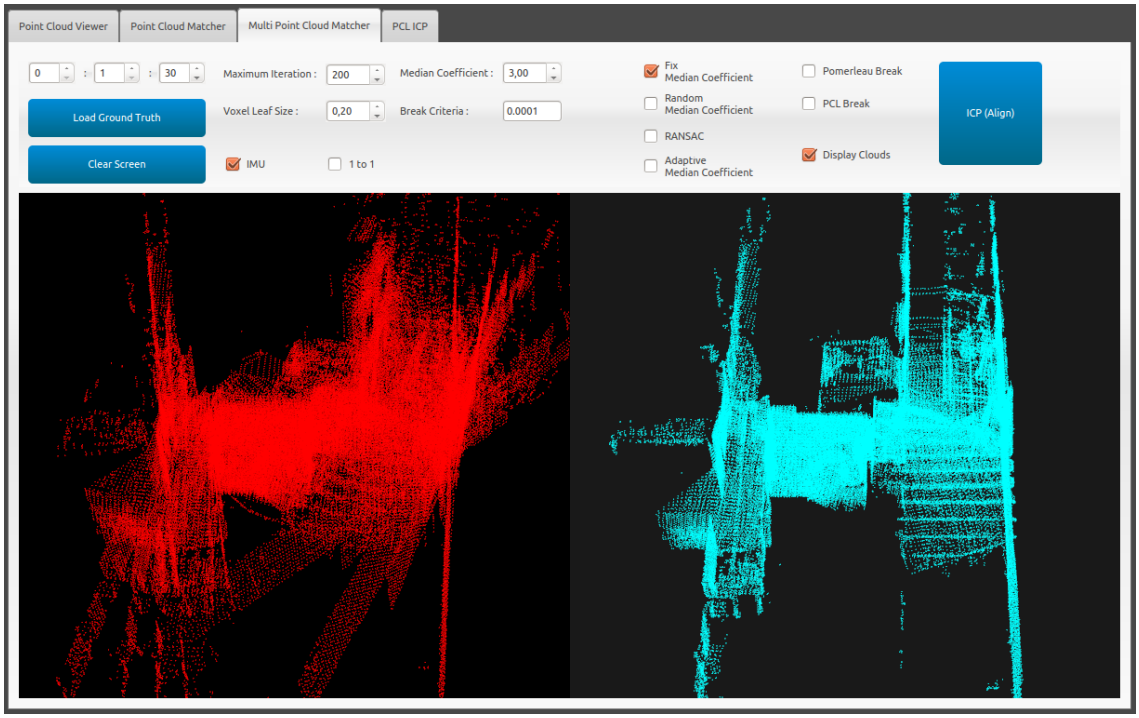
Ortalama Hata (m)	Ortalama İterasyon Sayısı	Ortalama İşlem Süresi (ms)
0.23	49.66	2296

Medyan katsayısının 10, filtre boyutunun 0.1 m seçildiği bir durum için ortalama hata, ortalama iterasyon ve ortalama işlem süresi Çizelge 4.12'de verilmiştir.

Çizelge 4.12. İkinci Veri Seti: 10*Medyan, 0.1 m Filtre Boyutu için Ortalama Hata, İterasyon Sayısı ve İşlem Süresi

Ortalama Hata (m)	Ortalama İterasyon Sayısı	Ortalama İşlem Süresi (ms)
0.1208	64.97	15314

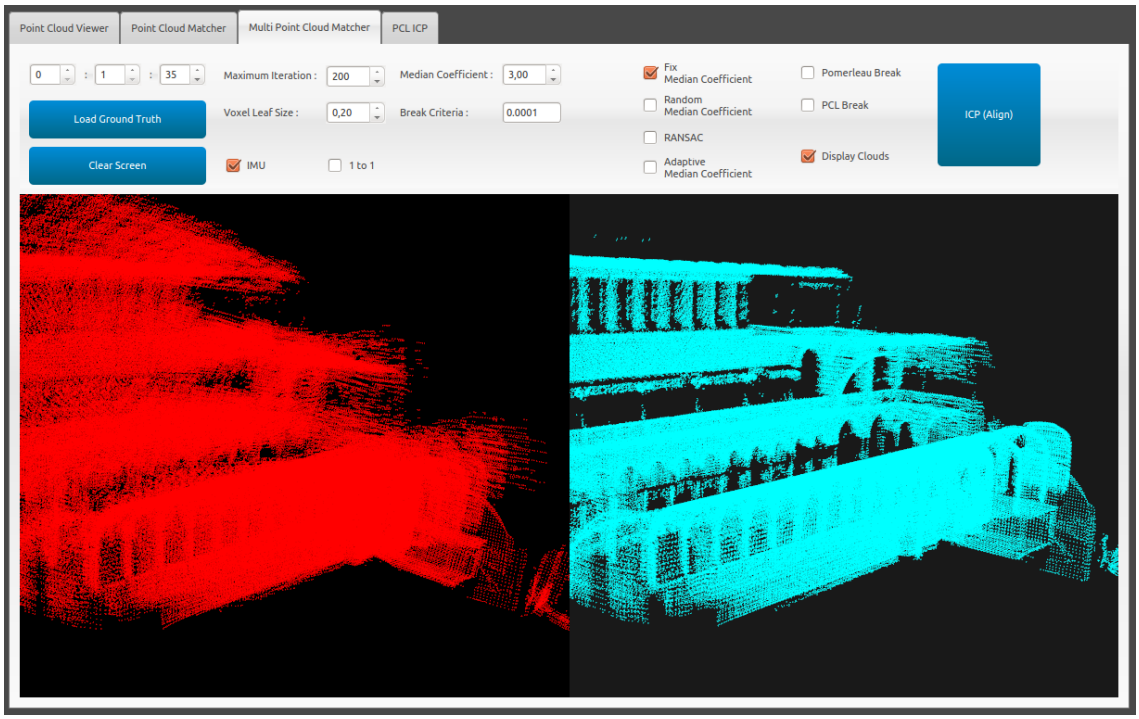
Geliştirilen yazılımın çoklu tarama eşleştirme bölümü kullanıldığında ICP algoritması ile oluşturulan harita görülebilmektedir. Şekil 4.14'te birinci veri setindeki tüm taramalar ardışık olarak eşleştirilmiş ve harita olarak gösterilmiştir. İkiye bölünmüş olarak görünen ekranda sol tarafta sadece gürültülü IMU verisi ile oluşturulan harita, sağ tarafta ise gürültülü IMU verisi ve ICP algoritması ile oluşturulan harita görülmektedir.



Şekil 4.14. Birinci Veri Seti: 3*Medyan, 0.2m Filtre Boyutu ile Oluşturulan Harita

Şekil 4.15'te ikinci veri setindeki tüm taramalar ardışık olarak eşleştirilmiş ve harita olarak gösterilmiştir. İkiye bölünmüş olarak görünen ekranda sol tarafta sadece gürültülü IMU verisi ile oluşturulan harita, sağ tarafta ise gürültülü IMU verisi ve ICP algoritması ile oluşturulan harita görülmektedir.

Her iki veri setinden oluşturulan haritalara bakıldığında sadece IMU verisinin bir harita oluşturabilmek için yeterli olmadığı, bir tarama eşleştirme ihtiyaç duyulduğu rahatlıkla görülmektedir.



Şekil 4.15. İkinci Veri Seti: 3*Medyan, 0.2m Filtre Boyutu ile Oluşturulan Harita

BÖLÜM V

SONUÇLAR

Tez çalışması sonucunda Gazebo simülasyon ortamından veri toplayacak ve eşleştirebilecek bir yazılım ile nokta bulutu dosyaları üzerinde dönme, öteleme, filtreleme gibi işlemlerin yapılabileceği, parametre değerlerinin değiştirilerek ICP algoritmasının koşturulabileceği bir yazılım geliştirilmiştir. Geliştirilen bu yazılım hız konusunda optimize edilmemiştir. Olabildiğinde yüksek doğrulukla eşleştirme yapabilmek adına uygun parametrelerin tespit edilmesi üzerinde çalışılmıştır.

ICP algoritmasının başarımını doğrudan etkileyen aykırı noktaların tespit edilmesi ve filtreleme işlemleri üzerinde çalışmalar yapılmıştır. Filtreleme işlemi sonucunda eşleştirme işlemi için yeterince noktanın kalması gerektiği açıktır. Bu nedenle nokta bulutundan aşırı miktarda nokta filtrelendiği takdirde eşleşme sonucunda hata miktarının artması beklenen bir durumdur ve bu çalışma sonucunda da gözlemlenmiştir. Ancak filtreleme sonucu kalan nokta sayısı da işlem hızını doğrudan etkilemektedir. Gereğinden fazla noktanın kullanılması işlem süresini gereksiz yere uzatmakla sonuçlanmaktadır. Bu nedenle aynı şekilde aykırı noktaların tespit edilip eşleştirme işlemi sırasında kullanılmaması son derece önemlidir. Aykırı noktaların doğru tespit edilememesi eşleştirme işleminin süresini uzatmakta ve hata miktarını artırmaktadır. Bu noktadan yola çıkılarak farklı iki veri seti üzerinde aykırı noktaların tespitinde kullanılan medyan katsayısının 3 ile 20 arasındaki tüm değerlerle 0.1 adım aralığı ile test yapılmıştır. Aynı zamanda filtre boyutu da 0.05 m ile 0.3 m aralığında 0.01 adım aralığı ile her bir veri seti için 4275 farklı kombinasyonda ICP algoritması koşturulmuştur. Yapılan bu çalışma sonucunda medyan katsayısının 3 ile 10 aralığında, filtre boyutunun ise 0.05 m ile 0.15 m aralığında seçildiği bölgede eşleştirme hatasının en az olduğu tespit edilmiştir. Bu sonuçtan da anlaşıldığı üzere tarama eşleştirme hatasının en az olması için medyan katsayısının ve filtre boyutunun belirlenen aralıkta seçilmesi uygun olacaktır. Daha başarılı eşleştirme sonuçları için belirlenen bölgede adaptif bir katsayı kullanılmasının veya akıllı bir arama algoritmasının geliştirilmesi gerekmektedir.

KAYNAKLAR

Arun, K.S., Huang, T.S. and Blostein, S.D., "Least-Squares Fitting of Two 3-D Point Sets", ***IEEE Trans. Pattern Anal. Mach. Intell.***, 698–700, 1987. doi:10.1109/TPAMI.1987.4767965

Bentley, J.L., "Multidimensional binary search trees used for associative searching", ***Communications of the ACM*** 18, 509–517, 1975. doi:10.1145/361002.361007

Besl, P.J. and McKay, N.D., "A method for registration of 3-D shapes", ***IEEE Trans. Pattern Anal. Mach. Intell.*** 14, 239–256, 1992. doi:10.1109/34.121791

Bonaccorso, F., Muscato, G. and Baglio, S., "Laser range data scan-matching algorithm for mobile robot indoor self-localization", ***World Automation Congress (WAC)***, 1–5, 2012.

Censi, A., "An ICP variant using a point-to-line metric", ***Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on***, 19–25, 2008.

Censi, A., Iocchi, L. and Grisetti, G., "Scan Matching in the Hough Domain", ***Proceedings of the 2005 IEEE International Conference on Robotics and Automation***, 2739–2744, 2005. doi:10.1109/ROBOT.2005.1570528

Chen, Y. and Medioni, G., "Object modelling by registration of multiple range images", ***Image and Vision Computing*** 10, 145–155, 1992. doi:10.1016/0262-8856(92)90066-C

Costa, W.F., Matsuura, J.P., Santana, F.S. and Saraiva, A.M., "Evaluation of an ICP Based Algorithm for Simultaneous Localization and Mapping Using a 3D Simulated P3DX Robot", ***Robotics Symposium and Intelligent Robotic Meeting (LARS)***, 103–108, 2010. doi:10.1109/LARS.2010.23

Cousins, S., "ROS on the PR2 [ROS Topics]", ***IEEE Robot. Autom. Mag.*** 17, 23–25, 2010. doi:10.1109/MRA.2010.938502

Diebel, J., Reutersward, K., Thrun, S., Davis, J. and Gupta, R., "Simultaneous localization and mapping with active stereo vision", ***2004 IEEE/RSJ International Conference on Intelligent Robots and Systems, (IROS 2004)***, 3436–3443, 2004. doi:10.1109/IROS.2004.1389948

Diosi, A. and Kleeman, L., "Fast Laser Scan Matching using Polar Coordinates", ***The International Journal of Robotics Research*** 26, 1125–1153, 2007. doi:10.1177/0278364907082042

Estépar, R.S.J., Brun, A. and Westin, C.-F., "Robust generalized total least squares iterative closest point registration", ***Medical Image Computing and Computer-Assisted Intervention***, 234–241, 2004.

Fischler, M.A. and Bolles, R.C., “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography”. ***Communications of the ACM*** 24, 381–395, 1981.

Fujita, T., “3D Sensing and Mapping for a Tracked Mobile Robot with a Movable Laser Ranger Finder”. ***Int. J. Electron. Electr. Eng.*** 6, 243–268, 2012.

Hast, A. and Nysjö, J., “Optimal RANSAC - Towards a Repeatable Algorithm for Finding the Optimal Set”. ***Journal of WSCG*** 21, 21–30, 2013.

Huhle, B., Magnusson, M., Strasser, W. and Lilienthal, A.J., “Registration of colored 3D point clouds with a Kernel-based extension to the normal distributions transform”, ***IEEE International Conference on Robotics and Automation, 2008. ICRA 2008***, 4025–4030, 2008. doi:10.1109/ROBOT.2008.4543829

Joung, J.H., An, K.H., Kang, J.W., Chung, M.-J. and Yu, W., “3D environment reconstruction using modified color ICP algorithm by fusion of a camera and a 3D laser range finder”, ***IEEE/RSJ International Conference on Intelligent Robots and Systems, 2009. IROS 2009***, 3082–3088, 2009. doi:10.1109/IROS.2009.5354500

Kim, D. and Kim, D., “A Fast ICP Algorithm for 3-D Human Body Motion Tracking”, ***IEEE Signal Process. Lett.*** 17, 402–405, 2010. doi:10.1109/LSP.2009.2039888

Kim, H.-Y., Lee, S.-O. and You, B.-J., “Robust laser scan matching in dynamic environments”, ***Robotics and Biomimetics (ROBIO), 2009 IEEE International Conference on***, 2284–2289, 2009.

Knuth, D.E., The Art of Computer Programming: Volume 3: Sorting and Searching, ***Upper Saddle River, NJ***, 1998.

Lenac, K., Mumolo, E. and Nolich, M., Robust and Accurate Genetic Scan Matching Algorithm for Robotic Navigation, Editörleri, Jeschke, S., Liu, H., Schilberg, D. (Eds.), ***Lecture Notes in Computer Science, Springer Berlin Heidelberg***, 584–593, 2011.

Martínez, J.L., González, J., Morales, J., Mandow, A. and García-Cerezo, A.J., “Mobile robot motion estimation by 2D scan matching with genetic and iterative closest point algorithms”, ***J. Field Robot.*** 23, 21–34, 2006. doi:10.1002/rob.20104

Masuda, T., Sakaue, K. and Yokoya, N., “Registration and integration of multiple range images for 3-D model construction”, ***Proceedings of the 13th International Conference on Pattern Recognition***, 879–883, 1996. doi:10.1109/ICPR.1996.546150

Meyer, J., Sendobry, A., Kohlbrecher, S., Klingauf, U., Stryk and O. von, “Comprehensive simulation of quadrotor uavs using ros and gazebo”, ***Simulation, Modeling, and Programming for Autonomous Robots. Springer***, 400–411, 2012.

Moore, A., An introductory tutorial on KD trees, PhD Thesis, ***Carnegie Mellon University***. 1991.

Neugebauer, P.J., “Reconstruction of real-world objects via simultaneous registration and robust combination of multiple range images”, ***Int. J. Shape Model.*** 3, 71–90, 1997.

PCL - Point Cloud Library (PCL), http://pointclouds.org

Pomerleau, F., Colas, F., Ferland, F. and Michaud, F., “Relative motion threshold for rejection in ICP registration”, ***Field and Service Robotics. Springer***, 229–238, 2010.

Pomerleau, F., Liu, M., Colas, F. and Siegwart, R., “Challenging data sets for point cloud registration algorithms”, ***Int. J. Robot. Res.*** 31, 1705–1711, 2012. doi:10.1177/0278364912458814

Ray, R., Banerji, D., Nandy, S. and Shome, S.N., “Keypoints based laser scan matching - A robust approach”, ***2012 IEEE International Conference on Robotics and Biomimetics (ROBIO)***, 741–746, 2012. doi:10.1109/ROBIO.2012.6491056

S. Druon, M. Aldon and A. Crosnier, “Color constrained icp for registration of large unstructured 3d color data sets”. ***Information Acquisition, 2006 IEEE International Conference on***, 249–255, 2006.

Singular Value Decomposition (SVD) tutorial, http://web.mit.edu/be.400/www/SVD/Singular_Value_Decomposition.htm

Slabaugh, G.G., Computing Euler angles from a rotation matrix, ***Technical Report***, 1999.

Strang, G., Linear Algebra and Its Applications, 4th Edition, ***Cengage Learning, Belmont, CA***, 2005.

Suárez-Ruiz, F., Owen-Hill, A. and Ferre, M., “Internet-Based Supervisory Teleoperation of a Virtual Humanoid Robot”, ***Advances in Intelligent Systems and Computing. Springer International Publishing***, 345–358, 2014.

Turk, G. and Levoy, M., “Zippered polygon meshes from range images”, ***Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques***, 311–318, 1994.

Ulaş, C. and Temeltaş, H., “3D Multi-Layered Normal Distribution Transform for Fast and Long Range Scan Matching”, ***J. Intell. Robot. Syst.*** 71, 85–108, 2013. doi:10.1007/s10846-012-9780-8

Unhelkar, V.V., Perez, J., Boerkoel, J.C., Bix, J., Bartscher, S. and Shah, J.A., “Towards control and sensing for an autonomous mobile robotic assistant navigating assembly lines”, ***2014 IEEE International Conference on Robotics and Automation (ICRA)***, 4161–4167, 2014. doi:10.1109/ICRA.2014.6907464

Wang, K., Wang, X., Pan, Z. and Liu, K., “A Two-Stage Framework for 3D Face Reconstruction from RGBD Images”, ***IEEE Trans. Pattern Anal. Mach. Intell.*** 36, 1493–1504, 2014. doi:10.1109/TPAMI.2013.235

Weik, S., “Registration of 3-D partial surface models using luminance and depth information”, ***International Conference on Recent Advances in. Presented at the 3-D Digital Imaging and Modeling***, 93–100, 1997. doi:10.1109/IM.1997.603853

Yoshitaka, H., Hirohiko, K., Akihisa, O. and Shin’ichi, Y., “Mobile robot localization and mapping by scan matching using laser reflection intensity of the sokuiki sensor”, ***IEEE Industrial Electronics, IECON 2006***, 3018–3023, 2006.

ÖZ GEÇMİŞ

Recai SİNEKLİ 09.06.1988 tarihinde Mersin'de doğdu. İlk ve ortaöğrenimini Mersin'de tamamladı. 2007 yılında Niğde Üniversitesi Elektrik-Elektronik Mühendisliği Bölümü'nü kazandı ve 2011 yılında bu bölümden birincilikle mezun oldu. Askerlik görevinin ardından 2012 yılında Niğde Üniversitesi Elektrik-Elektronik Mühendisliği Ana Bilim Dalı'nda yüksek lisans öğrenimine başladı ve aynı yıl Niğde Üniversitesi Mekatronik Mühendisliği Bölümü'ne araştırma görevlisi olarak atandı ve halen bu bölümde çalışmaktadır. İlgi alanı mobil platformlar, web teknolojileri, gömülü sistemler gibi alanlarda yazılım geliştirmek olan Recai SİNEKLİ, Bilim, Sanayi ve Teknoloji Bakanlığı'nın 2015 yılı Teknogirişim Sermayesi Desteğini kazanarak Sinek Yazılım San. ve Tic. Ltd. Şirketi'ni kurmuştur.

TEZ ÇALIŞMASINDAN ÜRETİLEN ESERLER

Bu tez çalışmasından 3 (üç) adet ulusal bildiri üretilmiştir. Bu üretilen çalışmalar aşağıda sunulmuştur.

Sinekli, R., Budak, Ö.F., Yalçın, M.K., "Kinect Sensör ve ICP Algoritması Kullanarak Üç Boyutlu Nokta Bulutu Eşleştirme", TOK'2014 Otomatik Kontrol Ulusal Toplantısı, İzmit, 947-951, 2014.

Sinekli, R., Budak, Ö.F., Yalçın, M.K., "ICP Algoritmasında Medyan Değerine Bağlı Aykırı Nokta Bulmanın Parametrik İncelenmesi", TORK'2014 Türkiye Otonom Robotlar Konferansı, Ankara, 2014

Sinekli, R., Yalçın, M.K., "Gazebo Simülasyon Ortamında Quadrotor Kullanılarak ICP Algoritmasına Dayalı 3 Boyutlu Tarama Eşleştirme", TOK'2015 Otomatik Kontrol Ulusal Toplantısı, Denizli, 2015.