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

ÜÇ BOYUTLU TARAMA EŞLEŞTİRME ALGORİTMASININ FPGA PLATFORMUNDA GERÇEKLENMESİ

BAUYRZHAN ANARBAYEV

Nisan 2017

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

ÜÇ BOYUTLU TARAMA EŞLEŞTİRME ALGORİTMASININ FPGA PLATFORMUNDA GERÇEKLENMESİ

BAUYRZHAN ANARBAYEV

Yüksek Lisans Tezi

Danışman

Yrd. Doç. Dr. Mehmet Kürşat YALÇIN

Nisan 2017

Bauyrzhan ANARBAYEV tarafından **Yrd. Doç. Dr. Mehmet Kürşat YALÇIN** danışmanlığında hazırlanan **"ÜÇ BOYUTLU TARAMA EŞLEŞTİRME ALGORİTMASININ FPGA PLATFORMUNDA GERÇEKLENMESİ"** adlı bu çalışma jürimiz tarafından Ömer Halisdemir Üniversitesi Fen Bilimleri Enstitüsü **Elektrik-Elektronik Mühendisliği** Ana Bilim Dalı'nda Yüksek Lisans tezi olarak kabul edilmiştir.

Başkan : Yrd. Doç. Dr. Tuba KURBAN (Erciyes Ün. Müh. Fak. Harita Müh.)

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

Üye : Yrd. Doç. Dr. Murat PEKER (Ömer Halisdemir 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 V.

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.

Bauyrzhan ANARBAYEV

ÖZET

ÜÇ BOYUTLU TARAMA EŞLEŞTİRME ALGORİTMASININ FPGA PLATFORMUNDA GERÇEKLENMESİ

ANARBAYEV, Bauyrzhan
Ömer Halisdemir Üniversitesi
Fen Bilimleri Enstitüsü
Elektrik-Elektronik Mühendisliği Ana Bilim Dalı

Danışman :Yrd. Doç. Dr. Mehmet Kürşat YALÇIN

Nisan 2017, 35 sayfa

Günümüzde otonom araçların ve mobil robotların, insan müdahalesi olmadan, bilgisayar kontrolünde belirlenen bir amaca ulaşması sağlanabilmektedir. Robotlar, kendilerine verilen görevleri yerine getirmek için çeşitli verileri işleyerek karar almak durumundadırlar. Bir mobil robotun içinde bulunduğu ortamda özerk olarak hareket edebilmesi, elindeki ortam bilgilerine bağlıdır. Bu bilgilerden birisi ve en önemli sayılabileni ortamın haritasıdır. Mobil Robotun bünyesinde bulunan mesafe sensörlerinden aldığı verileri hafızasında tuttuğu haritada uygun yerlere koyması, bir başka deyişle ortamın üç boyutlu nokta bulutu verisini oluşturup bu nokta bulutu verisini işlemesi gerekir. Bu işlem çoğunlukla tarama eşleştirme algoritması kullanarak yapılır. Nokta bulutu verisinin işlenme süresi aracın hareket ve karar alma hızını etkilemektedir. Aracın görevini ve hareketini hızlı yapması için verileri hızlı işleme kapasitesine sahip işlemciye ihtiyaç duymaktadır. Ancak böyle işlemcilerin yüksek güç tüketimi olduğu için kendi enerji kaynağını üzerinde taşıyan hareketli bir robotta kullanılması pek uygun değildir. Bu çalışmada tarama eşleştirme algoritması az güç tüketimi olan ve paralel işleme kapasitesi sayesinde hızlı veri işleyebilen FPGA donanımı üzerinde gerçeklenmiştir.

Anahtar Sözcükler: FPGA, tarama eşleştirme, mobil robot, nokta bulutu verisi

SUMMARY

FPGA IMPLEMENTATION OF 3D SCAN MATCHING ALGORITHM

ANARBAYEV, Bauyrzhan
Ömer Halisdemir University
Graduate School of Natural and Applied Science
Department of Electrical-Electronics Engineering

Supervisor : Assistant Professor Dr. Mehmet Kürşat YALÇIN

April 2017, 35 pages

Nowadays autonomous vehicles and mobile robots can achieve predetermined tasks without computer and human intervention. Robots must take decisions by working on various data to fulfill their assigned tasks. The ability of a mobile robot to act autonomously in the environment depends on the information about its surroundings. Maybe the most significant information among others that mobile robot needs is the map of the surroundings. It is necessary to place the data received from the distance sensors installed on the mobile robot to the appropriate points in the map that is stored in the memory. In other words, it has to create the three-dimensional point cloud data of the environment and process this point cloud data appropriately. This is often done using the scan-matching algorithm. The processing time of point cloud data affects the speed of the vehicle's movement and decision making. The robot needs a processor with a fast processing capacity to speed up its task and movement. However, since such processors have high power consumption, it is not appropriate to use them in a mobile robot that carries its own energy source. In this study, the scan-matching algorithm was implemented on FPGA hardware, which consumes little power and can process data quickly due to its parallel processing capacity.

Keywords: FPGA, scan matching, mobile robot point cloud data.

ÖN SÖZ

Bu tez çalışmasında, tarama eşleştirme algoritması FPGA kartı üzerinde gerçekleşmiştir.

Tez çalışması sürecinde desteğini esirgemeyen, değerli danışmanım Yrd. Doç. Dr. Mehmet Kürşat YALÇIN'a, teşekkürlerimi sunarım. Ayrıca yüksek lisans eğitimim boyunca tez çalışmalarım esnasında tecrübelerine başvurduğum Alper EMLEK, Mehmet Muzaffer KÖSTEN hocalarıma ve beraber çalıştığım tüm çalışma arkadaşlarıma teşekkürlerimi sunarım. 113E210 numaralı proje kapsamında bu çalışmaya yapmış oldukları maddi ve manevi desteklerinden dolayı TÜBİTAK'a teşekkürlerimi sunarım.

İÇİNDEKİLER

ÇİZELGELER DİZİNİ

ŞEKİLLER DİZİNİ

SİMGE VE KISALTMALAR

Kısaltmalar	Açıklama
FPGA	Field Programmable Gate Arrays
PS	Processing System
PL	Programlanabilir Lojik
ICP	Iterative Closest Point
IMU	Interial Measurement Unit
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuits
RAM	Random Access Memory
IP	Intellectual Property
SVD	Singular Value Decomposition

BÖLÜM I

GİRİŞ

Teknolojinin gelişmesi ile endüstriyel, tarım, askeri, maden vb. birçok alanda insan gücüyle yapılan işleri robotlar devralmaktadır. Robotlar çalıştığı alana göre belli verilere ihtiyaç duymaktadır. Hareketli robotların özerk olarak engellere takılmadan belirlenen hedefe gidebilmesi ve gerekli işlemi yapabilmesi için ortamı algılayabilmesi yani ortamın haritasının olması gerekmektedir. Ortam haritası kameradan veya lazer tarayıcısından veriler alınarak oluşturulabilmektedir. Hareket halindeki robotun haritayı doğru bir şekilde oluşturması için sensörlerden aldığı verilerin yanı sıra kendi pozisyonunu ve hareketini kestirmesi lazımdır. Kendi hareketini kestirebilmesi için ataletsel ölçüm birimleri (Interial Measurement Unit) veya enkoder kullanılmaktadır. Ortam koşullarındaki değişikler nedeniyle enkoder ve ataletsel ölçüm birimlerinden kaynaklanan, mobil robotun harita oluşturması sırasında hatalar olmaktadır. Bu hataları gidermek ve robotun doğru bir şekilde hareketini kestirmek için lazer ya da kameradan alınan veriler tarama eşleştirme algoritmasında işlenir ve daha doğru sonuçlar elde edilir. Tarama eşleştirme probleminin birçok çözümü bulunmaktadır. Bu yöntemlerin en bilineni ve en yaygın olarak kullanılanı “Iterative Closest Point” (ICP) algoritmasıdır. (Besl ve McKay, 1992).

ICP, iki veya üç boyutlu nokta bulutlarının eşleştirmesi için geliştirilmiş bir algoritmadır. 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; Martinez vd., 2006), tarama ile üç boyutlu şekillerin yeniden oluşturulması (Besl ve McKay, 1992; Estepar vd., 2004; Neugebauer, 1997; Wang vd., 2014) alanlarında ICP temelli algoritmalar kullanılmaktadır.

ICP algoritması genel olarak dört adımdan oluşmaktadır. Bunlar filtreleme, ilişkili noktaların bulunması, dışsal noktaların bulunması ve uygun dönüşümün bulunmasıdır. Literatürde ICP algoritmasının geliştirilmesi ile ilgili çalışmalar bu adımlar üzerinde yoğunlaşmaktadır.

ICP algoritmasının ilk adımı olan filtreme işleminde temel olarak iki farklı yöntem bulunmaktadır. Birincisi nokta bulutlarını homojen bir şekilde filtreleyerek (Turk ve Levoy, 1994), ikincisi ise filtrelemeyi her iterasyonda rastgele seçilen noktaları (Masuda vd., 1996) kullanarak yapmaktır.

Filtreleme işleminden sonra gelen adım ilişkili noktaların tespitidir. İlişki noktaların tespitinde, akla gelen ilk yöntem şu şekildedir; birinci nokta bulutundaki herbir noktanın ikinci nokta bulutundaki bütün noktalara olan uzaklıkları bulunur ve bu uzaklıkların en küçüğünü veren nokta çifteleri ilişkili olarak addedilir (Besl ve McKay, 1992). Literatürde mesafenin noktadan noktaya değil, noktadan düzleme hesaplandığı metotlar da bulunmaktadır (Censi, 2008). Ayrıca noktaların polar koordinat sisteminde incelendiği (Diosi ve Kleeman, 2007) çalışmalar öne çıkmaktadır.

ICP algoritmasında ilişkili noktaların tespitinden sonra üçüncü adım olarak dışsal noktaların tespiti gelir. Dışsal nokta tespiti adımında bazı noktalar elenecektir. Bu eleme işlemi sabit bir eşik kullanılarak yapılabilir. İlişkili noktalar arasındaki mesafeler eğer bu eşik değerden büyük ise bu nokta çiftleri dışsal noktalar olarak etiketlenir ve elenir. Literatürde bu eleme işleminde kullanılan eşik ile ilgili farklı yöntemler bulunmaktadır. Örneğin, eşik olarak ilişkili noktaların aralarındaki mesafelerinin medyan değerinin sabit bir katı alınan çalışmalar vardır. Bir başka çalışmada ise eşik değeri mesafelerin standart sapmasının toplamı olarak alınmıştır. (Diebel vd., 2004; Pomerlau vd., 2010).

Dışsal nokta tespiti işleminden sonraki adım dönüşüm kestirimidir. Bir nokta bulutunun referans eksen takımı ile diğer nokta bulutunun referans eksen takımı arasındaki dönüşümün bulunması adımında en çok tercih edilen yöntem tekil değer ayrışımına dayalı (Singular Value Decomposition) bir yöntemdir (Arun vd., 1987). Önceki adımlarda bulunan ilişkili noktalar eğer doğru ise SVD kullanılarak ulaşılan dönüşüm doğru olacaktır.

Bu çalışmada, bahsedilen ICP algoritması Zedboard geliştirme kartındaki FPGA (Field Programmable Gate Arrays) donanımı üzerinde VHDL dili kullanılarak kodlanmıştır. Zedboard üzerindeki FPGA çipinin içinde programlanabilen mantık bloklarının yanı sıra ARM tabanlı çift çekirdekli işlemci bulunmaktadır. ICP algoritması VHDL dilinde yazılıp, sonuçlar alınmıştır. Aynı zamanda algoritma ARM çekirdekler üzerinde çalışan

Xillinux işletim sistemi altında C diliyle yazılıp gerçek sonuç ve hız karşılaştırılması yapılmıştır.

BÖLÜM II

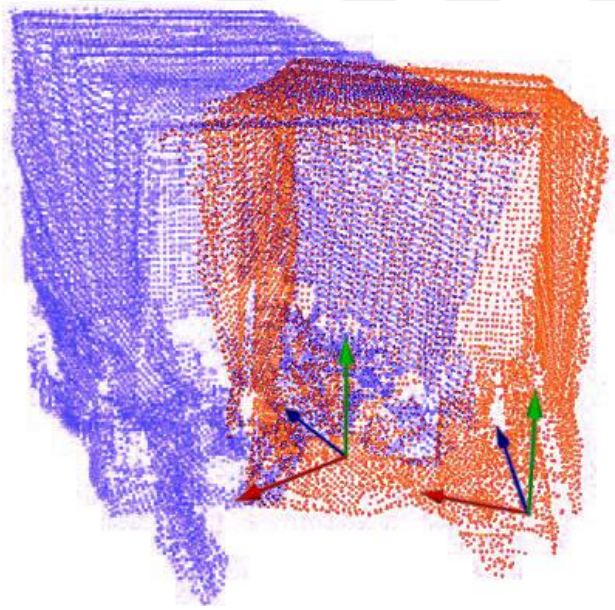
TARAMA EŞLEŞTİRME

2.1 Problem Tanımı

Hareket halindeki bir robota yerleştirilmiş bir sensörden farklı zamanlarda alınan veriler farklı koordinat sistemlerinde olacaktır. Bu verilerin en az hatayla aynı koordinat sistemine taşınması işlemi tarama eşleştirme olarak adlandırılır. Eğer bir robota sabitlenmiş sensörün ortam içerisindeki pozisyonu ve oryantasyonu bilinirse, tarama eşleştirme algoritmalarına gerek duyulmadan ortam haritası çıkarılabilir.

Geliştirilen tarama eşleştirme algoritmalarının hedefi, iki nokta bulutunun referans eksen takımlarını en doğru şekilde çakıştırmaktır.

Şekil 2.1'de bir robotun aynı ortam için iki farklı pozisyondan almış olduğu üç boyutlu nokta bulutları görülmektedir. Her bir nokta bulutu verisi kendi referans eksen takımına göre tanımlıdır.



Şekil 2.1. Robotun iki farklı pozisyonu için üç boyutlu nokta bulutları

Bu çalışmada sensörden gelen ilk nokta bulutu hedef, ikinci nokta bulutu ise kaynak olarak isimlendirilmiştir. Sıklıkla yapılan işlem, kaynak nokta bulutunun referans eksen takımına bir dönüşüm uygulayarak bu eksen takımını, hedef nokta bulutu referans eksen takımına çakıştırmaktır. Kaynak nokta bulutu $K = \{k_1, k_2, k_3, \dots, k_N\}$, hedef nokta bulutu ise $H = \{h_1, h_2, h_3, \dots, h_M\}$ olarak tanımlanırsa, 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 dönüşüm matrisi (T) uygulandıktan sonra bu noktalar ile hedef nokta bulutundaki en yakın noktalar arasındaki mesafelerin karelerinin toplamıdır.

$$E = \sum_{i=1}^{N} \|T(k_i) - h_i\|^2 \tag{2.1}$$

Üç boyutlu uzayda dönüşüm, üç farklı eksende dönme ve üç farklı eksende öteleme işlemi içerdiğinden, altı serbestlik derecesi vardır. Rotasyon matrisi ve öteleme vektöründen oluşan dönüşüm matrisi T Denklem 2.3'deki gibi yazılırsa, E hata ölçütü denklem 2.4'deki gibi yazılabilir.

$$T_{4x4} = \begin{bmatrix} R_{3x3} & t_{3x1} \\ 0_{1x3} & 1 \end{bmatrix} \tag{2.2}$$

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

Tarama eşleştirmede, hata ölçütünü en aza indiren bir dönüşüm matrisi bulunmalıdır. Optimum dönüşüm matrisine ulaşma işlemi döngüsel bir işlemdir. Bu döngüsel yapının sonlandırılması iki şekilde olmaktadır. Birincisi belirli bir döngü sayısına ulaşıldığında elde edilen dönüşüm matrisi en uygun matris olarak kabul edilir. İkincisi ardışık döngüler arasında hata ölçütü değişiminin belirli bir eşik altında kaldığında elde edilen matris en uygun olarak kabul edilir.

2.2 Iterative Closest Point (ICP) 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 birçok ICP çeşiti çıkarılmıştır. Bu algoritma, kendi içerisinde filtreleme, noktaların ilişkilendirilmesi, dışsal noktaların elenmesi ve dönüşümün bulunması gibi bazı adımlara ayrılır. Bu adımlarda yapılan geliştirmeler eşleşme sonucunu doğrudan etkiler.

ICP algoritması genel olarak iki veya daha fazla nokta bulutu 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.

1. Başlatma**:** Eğer iki nokta bulutunu birbirine yaklaştıracak bir başlangıç dönüşümü bilinmiyorsa T0 başlangıç dönüşümü 4x4 birim matris olarak alınır.
2. Filtreleme: İşlem hızını artırmak için her iki nokta bulutundaki nokta sayısını azaltacak şekilde örnekleme yapılır.
3. En yakın komşu bulma: Hedef nokta bulutundaki her bir nokta için diğer nokta bulutundaki en yakın nokta bulunur ve bu iki nokta birbiriyle ilişkilendirilir.
4. Dışsal nokta bulma: Bir eşik değeri belirlenir ve eşleşen iki nokta arasındaki mesafe bu eşik değerden daha büyük ise bu iki nokta eşleşme listesinden çıkarılır. Bu test bütün eşleşen noktalar için yapılır.
5. Dönüşümün bulunması: Geriye kalan eşleşen noktalara göre iki nokta bulutu arasında bir dönüşüm hesaplanır. Hesaplanan dönüşüm başlangıç dönüşümü ile çarpılır ve en son durum saklanır (T0 = T0 * Hesaplanan dönüşüm). Yeni T0 dönüşümü kaynak nokta bulutuna uygulanır.
6. 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 3. adıma dönülür ve işlemler tekrarlanır.

Bu adımlardan ayrıntılı olarak incelenmesi gerekenler devam eden kısımlarda anlatılmıştır.

2.2.1 Filtreleme

Filtreleme işlemi yapılmadan her iki nokta bulutundaki bütün noktalar kullanılarak ICP gerçekleştirilebilir (Besl ve McKay, 1992). Bu çalışmada, ICP algoritmasının hızını arttırmak için nokta bulutu filtreleme işlemi PCL (Point Cloud Library) kütüphanesi

kullanılarak bir bilgisayarda yapılmıştır.

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

2.2.2 En yakın komşu bulma

En yakın nokta arama olarak da bilinen en yakın komşu bulma algoritması, en yakın noktanın bulunmasını sağlayan bir eniyileme problemidir. Bu problem genelde, H uzayındaki bir q noktasına ($q \in H$), K noktalar kümesi içindeki en yakın noktanın bulunması olarak tanımlanır. Bu problem daha genelleştirilmiş bir ifadeyle k-NN (Nearest Neighbour) olarak isimlendirilir ve K 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, H uzayındaki bir q noktasının K noktalar kümesindeki her bir noktaya olan mesafesini hesaplayarak ve o ana kadar bulunan en küçük mesafeyi ve hangi nokta olduğunu saklayarak çözüm getiren VHDL kodlar yazılmıştır. K kümesindeki nokta sayısı N, H uzayının nokta sayısı M ise sıralı arama algoritmasının zaman karmaşıklığı $O(NM)$ olarak hesaplanır.

2.2.3 Dışsal nokta bulma

Dışsal 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 dışsal noktaların tespit edilebilmesi için çeşitli yöntemler önerilmiştir. Bu yöntemlerden bazıları aşağıda verilmiştir.

Sabit eşik: Eşleşen noktalar için sabit bir eşik değerinin belirlendiği yöntemdir. Eşleşen noktalar arasındaki mesafe bu eşik değerinden daha büyük ise o nokta çifti dışsal noktalar olarak belirlenir ve eşleşme listesinden çıkarılır. En basit yöntemdir ancak farklı eşik değer gerektiren durumlara adapte olamaz (Pomerleau vd., 2010).

Ortalama eşik: Bu yöntem, eşik mesafesini eşleşen noktalar arası mesafelerin ortalama

değeri ile standart sapmasının toplamı ($\mu + \sigma$) olarak alır. Eşik 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 dışsal noktaların tespit edilmesi için uygun bir yöntemdir (Pomerleau vd., 2010).

Medyan eşiği: Eşleşen noktalar arasındaki mesafelerin medyan değerinin 3 katının eşik olarak alındığı 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.

2.2.4 Dönüşüm hesaplanması

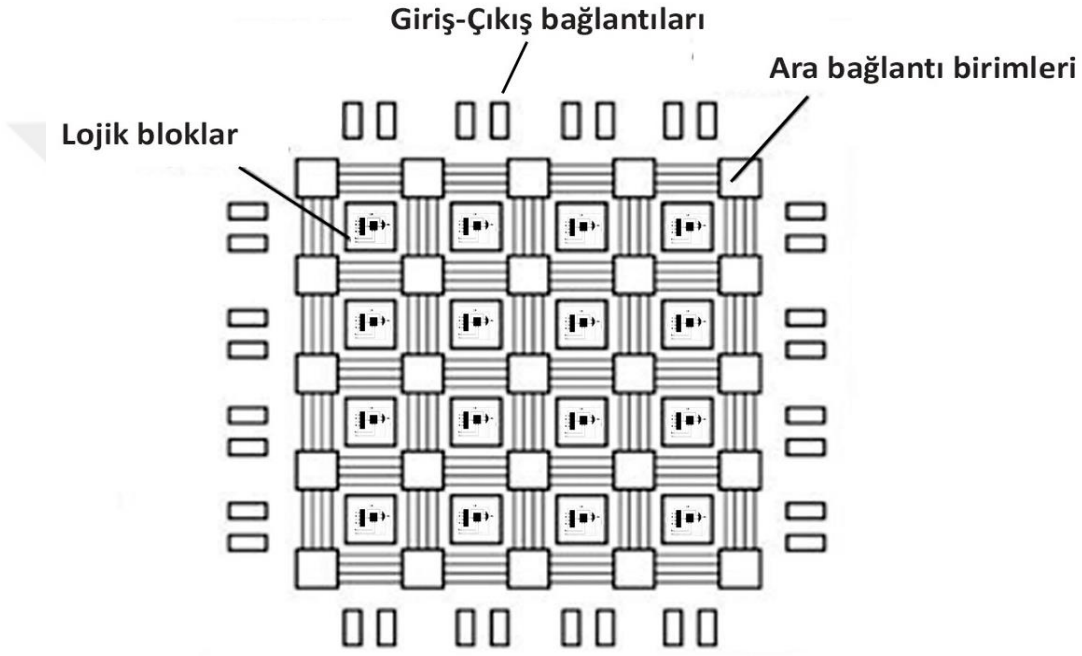
İki nokta bulutu arasında ilişkili noktalar bulunduktan ve dışsal noktalar elendikten sonra yapılması gereken işlem en iyi dönüşüm matrisinin bulunmasıdır. İki nokta bulutu arasındaki ilişkili noktalar biliniyorsa, kaynak nokta bulutunu hedef nokta bulutuna taşıyacak dönme ve öteleme, "Singular Value Decomposition" (SVD) işlemine bağlı olarak bulunabilir (Arun vd., 1987). Noktalar hatalı eşleştirildiğinde bulunacak dönme ve öteleme de hatalı olacaktır.

BÖLÜM III

ALGORİTMANIN GERÇEKLEŞTİRİLECEĞİ GÖMÜLÜ SİSTEM

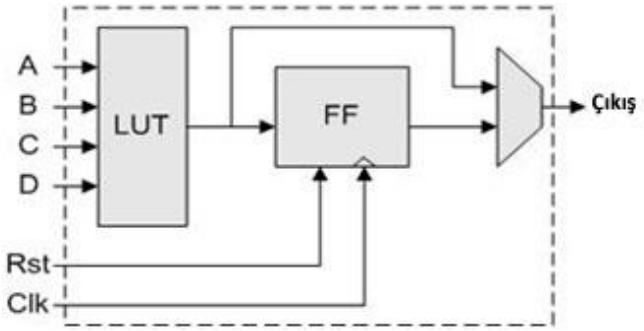
Bu bölümde, tarama eşleştirme algoritmasının gerçekleştirileceği geliştirme kartı ve gömülü sistemden bahsedilecektir. Geliştirme kartı, Xillinx firmasına ait bir FPGA çip içermektedir. Bu çip içerisinde programlanabilen bir kısım ile birlikte çift çekirdekli ARM işlemci bulunmaktadır. Aşağıdaki alt bölümlerde detaylar verilmiştir.

3.1 FPGA Nedir?



Şekil 3.1. FPGA içyapısı

FPGA'lar temel olarak programlanabilen lojik bloklar, bağlantı noktaları ve giriş-çıkış bloklarından oluşan yarı iletken elemanlardır. Programlanabilen lojik bloklar, AND, OR, XOR, NOT gibi temel lojik kapıları veya kod çözücüler, matematiksel fonksiyonlar gibi daha karmaşık işlemleri gerçekleyecek şekilde programlanabilirler. Günümüz FPGA yongaları bünyelerinde bellek blokları içermektedir. FPGA'lar 1985 yılında Xilinx firması tarafından 9.000 kapıdan oluşan ilk ticari model olan XC2064'ün geliştirilmesi ile yaygınlaşmıştır. Şekil 3.1.'de FPGA in içyapısı, Şekil 3.2'de programlanabilen lojik blok içyapısı görülmektedir.



Şekil 3.2. Programlanabilir lojik blok içyapısı (Wikibooks, 2014)

Bu çalışmada Xilinx firmasına ait Zynq-7000 System on Chip (SoC) entegresi barındıran Zedboard FPGA geliştirme kartı kullanılmıştır.

3.2 Zynq-7000 Entegresi

Zynq 7000 entegresi Xilinx firmasının tamamen programlanabilen (All Programmable) mimarisine dayanmaktadır. Bu entegre, içerisinde çift çekirdekli ARM Cortex-A9 tabanlı bir işlemci sistemi (Processing System:PS) ve programlanabilen lojik (Programmable Logic:PL) barındırmaktadır. İşletim sistemi olmadan çalışabilmektedir veya istenirse Linux tabanlı işletim sistemleri kurulabilir. Bunlar için gerekli aygıt sürücüleri Xilinx firması tarafından sağlanmakta, kart destek paketleri Xilinx'in ortakları tarafından sağlanmaktadır.

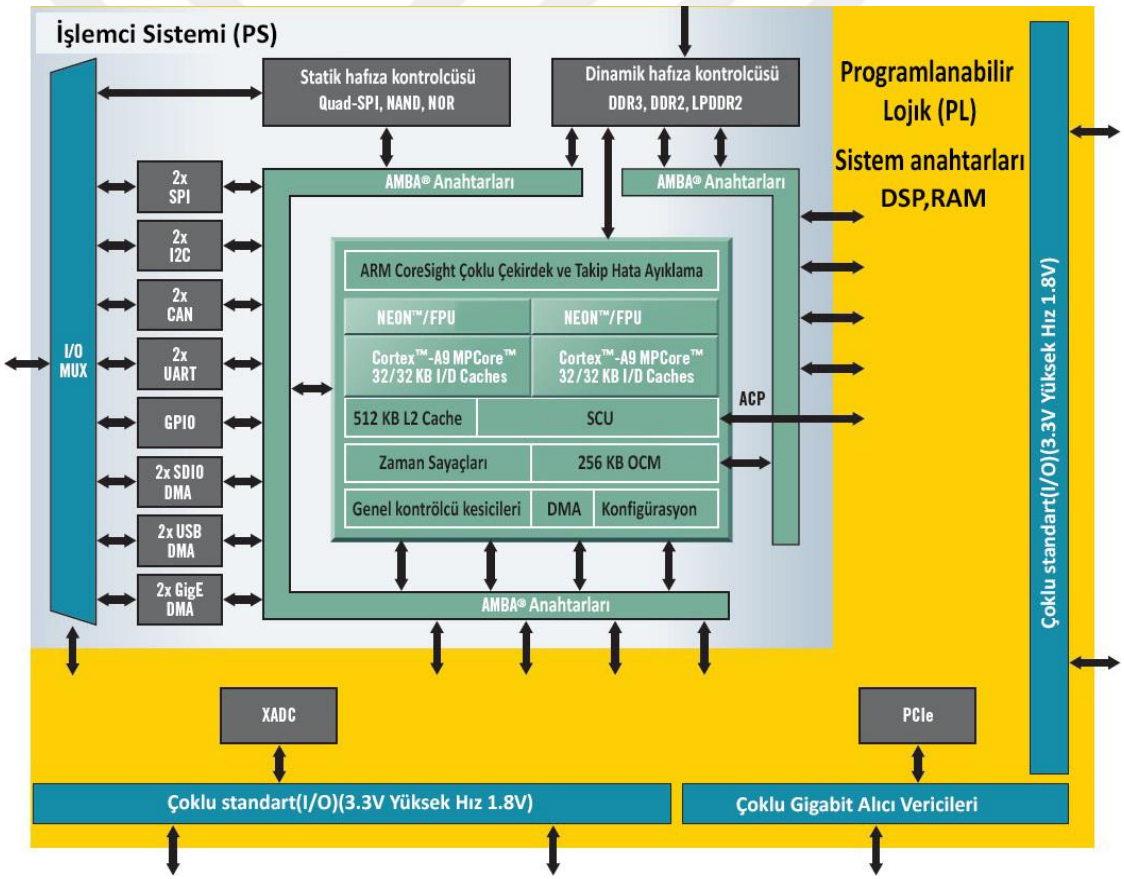
Zynq-7000 Şekil 3.3'te görülen aşağıdaki temel fonksiyonel bloklardan oluşmaktadır.
İşlemci sistemi (PS)

- Uygulama İşlemci Birimi
  Uygulama işlemci birimi çift çekirdekli bir ARM Cortex-A9 işlemciye sahiptir. NEON yardımcı işlemcisi, 32 KB L1,512 KB L2 ön belleği ile mikro işlemci gerektiren işlemleri rahat bir şekilde yapabilmektedir.
- Bellek ara yüzleri
  Bellek ara yüzleri birden fazla bellek teknolojisini içermektedir. Bu teknolojiler başta DDR (DDR2,DDR3) kontrolcüsü olmak üzere, QUAD-SPI kontrolcüsü, NAND kontrolcüsüdür.
- Giriş Çıkış çevresel birimi
  I/O çevresel birimleri (IOP) harici haberleşme için endüstri standardı ara yüzlerin bir araya getirilmesinden oluşmuştur. Bu ara yüzler; genel amaçlı giriş çıkış

(GPIO), Ethernet, SD/SDIO, Ethernet, USB, SPI, CAN, UART, I2C ve PS MIO dur.

- Ara Bağlantı Birimi

  Zynq-7000 aygıtları, fonksiyon bloklarının ihtiyaçlarını ve belirli iletişim ihtiyaçlarını karşılayacak şekilde optimize edilmiş çeşitli bağlantı teknolojileri kullanır. PS ve PL arası iletişimlerde "Advanced eXtensible Interface" (AXI) ara bağlantısı lojik IP'ler (Intellectual Property core) ile iletişimde kullanılmaktadır. MIO (multiplexed I/O) dış dünya ile PS arası doğrudan bağlantılarda kullanılır. EMIO (extended multiplexed I/O) ise PS ve PL arasındaki iletişimlerde ve lojik bloğa bağlı olan ara yüzlerin PS tarafından erişiminin sağlanmasında kullanılmaktadır.



Şekil 3.3. Zynq-7000 Soc blok diyagramı (GREG BROWN, 2011)

Programlanabilir lojik (PL)

Programlanabilir lojik kullanıcı tarafından kontrol edilebilir zengin bir mimariye sahiptir. İçerisinde 220 adet Sayısal sinyal işleme blokları (DSP48E1), 560 KB blok RAM, 85.000 lojik hücre bulunmaktadır.

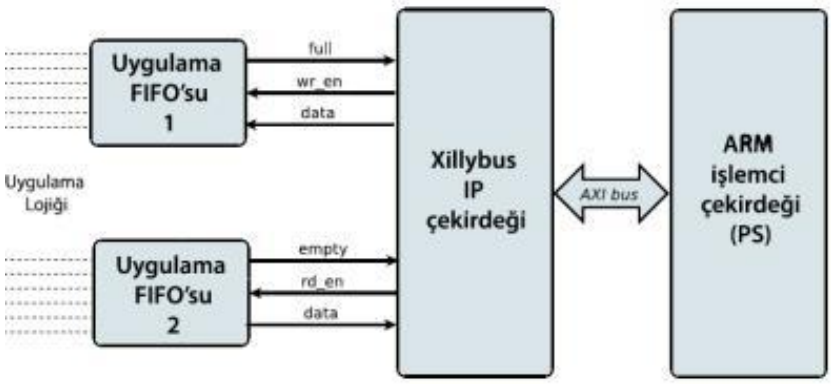
3.3 Zedboard, İşletim Sistemi (PS) ile Programlanabilir Lojik (PL) Haberleşmesi (Xillinux, Xillybus)

Xillinux, Ubuntu 12.04 LTS tabanlı, yazılım ve lojik karışımı projelerin hızlı geliştirilmesi için platform oluşturmak amacıyla geliştirilmiş grafik ara yüzlü bir işletim sistemidir. Diğer Linux dağıtımları gibi Xillinux da üzerinde Linux çalışan bir kişisel bilgisayarın sahip olduğu yazılım kabiliyetlerinin çoğuna sahiptir.

Şekil 3.4. Zedboard üzerinde kurulmuş xillinux

Bu dağıtımda Şekil 3.4'te görüldüğü gibi klasik klavye, fare monitör de kullanılabilmektedir. Ayrıca USB-UART portundan komut satırı ile kontrol de yapılabilmektedir, ancak bu özellik çoğunlukla bir sorunla karşılaşıldığında kullanılmaktadır. İçerdiği Xillybus IP çekirdeği ve sürücüsü ile FPGA ve Linux tabanlı bir projenin gerçekleştirilebilmesi için temel programlama ve lojik tasarım yeteneği yeterli olmaktadır.

Xillybus sade, portatif, verimli bir DMA (Direct Memory Access) tabanlı FPGA ve işletim sistemi arasında veri transferi için geliştirilmiş kullanıma hazır bir IP çekirdeğidir. Xillybus kişisel bilgisayarlar ve gömülü sistemler için PCI Express veri yolunu, ARM tabanlı işlemciler için de "Advanced Microcontroller Bus Architecture" (AMBA) veri yolunu (AXI3/AXI4) destekleyecek şekilde geliştirilmiştir. Şekil 3.5'te gösterildiği gibi, FPGA üzerindeki uygulama lojiği yalnızca standart FIFO'lara ihtiyaç duymaktadır.

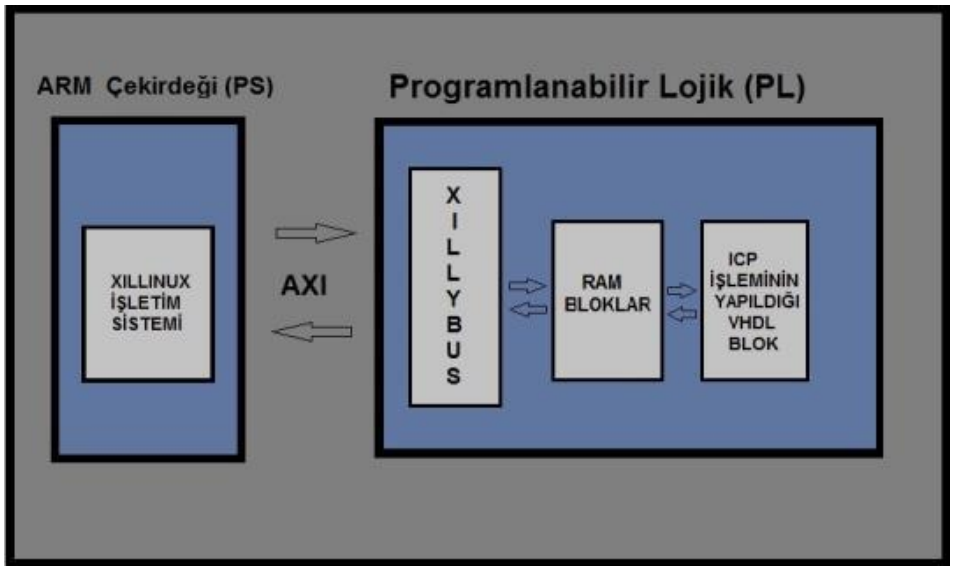


Şekil 3.5. Xillybus çalışma diyagramı

Bu FIFO'lardan 1 numaralı FIFO Xillybus'dan veriyi alıp uygulama lojiğine iletir, 2 numaralı FIFO ise uygulama lojiğinden veriyi alır Xillybus'a iletir. İstenildiğinde bu FIFO'lar yerine FIFO gibi davranacak herhangi bir modül de eklenebilir. Xillybus ve ARM çekirdeği arasındaki bağlantı DMA-AXI veri yolu üzerinden DMA istekleri ile sağlanır. Xillybus IP çekirdeği kullanıcının isteği doğrultusunda yapılandırılabilmektedir.

3.3.1 Haberleşmenin sağlanması

İşletim sistemi (PS) ile Programlanabilir Lojik (PL) sisteminin haberleşme diyagramı Şekil 3.6'da görülmektedir.



Şekil 3.6. PS ile PL haberleşmesini sağlandığı blok diyagram

Kullanıcının erişebileceği kanallar şöyledir;

- Xillybus Lite: 32 bit adres 32 bit veri

- Memory: 5 bit adres 8 bit veri
- Fifo 32: 32 bit veri
- Fifo 8: 8 bit veri

Dikkat edileceği üzere Fifo 32 kanalı için adres bilgisi bulunmamaktadır. Bu projede, nokta bulutu verileri paketler halinde PS'den PL'e gönderileceğinden, herhangi bir veri kaybı ya da veri karışıklığını gidermek amacıyla Xillybus Lite kanalı kullanılmıştır. Xillybus Lite kanalı ise 32 bit adres ve 32 bit veri genişliği sunmaktadır. Bu özellik sisteme hız, bant genişliği ve adresleme doğruluğu kazandırır. PS-PL veri trafiğini kontrol etmek için kontrol kelimesi 8 bit olan Memory kanalı kullanılmıştır. Bu kontrol komutları arasında "Başlat", "Xillybus Lite üzerinden hangi verinin gönderildiği", "işlemin bittiği" gibi komutları bulunmaktadır.

BÖLÜM IV

YÖNTEM

Daha önceki bölümlerde detayları verilen ICP algoritması temel olarak altı adımdan oluşmaktadır. Bunlardan "Başlatma" ve "Filtreleme" adımları bilgisayar sistemleri kullanılarak gerçeklenmektedir. Bilgisayar sisteminden elde edilen veriler programlanabilen mantık devrelerine aktarılmakta ve ICP'nin geriye kalan "En yakın komşu bulma", "Dışsal nokta bulma", "Dönüşümün bulunması" ve "Tekrarlama işleminin sonlandırılması (Durdurma kriteri)" adımları burada çalıştırılmaktadır. Bu dört adım aşağıda detaylandırılmıştır. Örnek nokta bulutu verileri ile sonuçlar ve her adım için detaylı hesaplama süreleri elde edilmiştir. Hesaplama süreleri göz önüne alındığında "En yakın komşu bulma" adımının en zayıf halka olduğu tespit edilmiştir. Bu yüzden bu adımla ilgili ayrı bir yöntem denemesi yapılmış ve detaylar takip eden alt bölümlerde sunulmuştur.

4.1 En Yakın Komşu Bulma

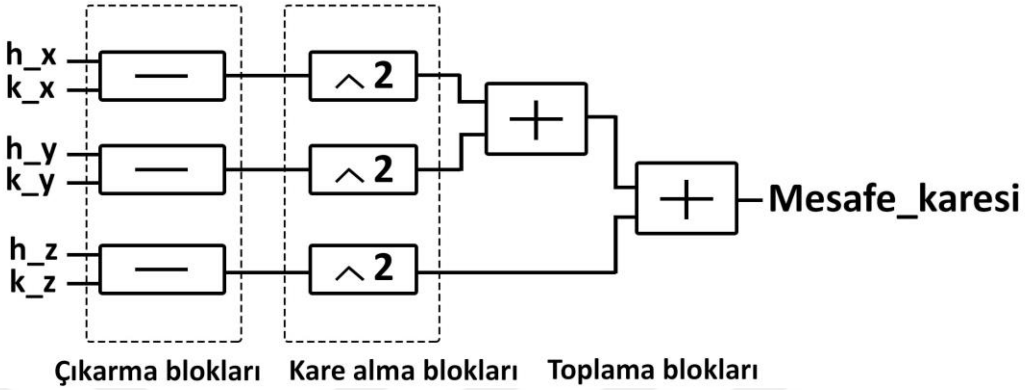
En yakın komşu bulma adımı için lineer arama yöntemi kullanılmıştır. Bu yöntemle aşağıda verilen denklemdeki gibi kaynak nokta bulutu ve hedef nokta bulutu arasındaki mesafelerin kareleri hesaplanır.

$$Mesafe\_kare(i) = (h_x(i) - k_x(j))^2 + (h_y(i) - k_y(j))^2 + (h_z(i) - k_z(j))^2 \quad (4.1)$$

Bu denklemde, h hedef nokta bulutunu, k kaynak nokta bulutunu ve alt indisler ilgili noktaların x, y ve z koordinatlarını göstermektedir. Hedef nokta bulutundaki bütün noktalar için, kaynak nokta bulutundaki bütün noktalar arasındaki mesafesinin karesi $Mesafe\_kare(i)$ hesaplanır ve bu mesafelerden en küçük olanı hedef nokta bulutu ile eşleştirilir. Hesaplama işlemleri 32-bitlik kayar nokta aritmetiği (Floating Point Arithmetic) kullanılarak yapılmıştır. Aritmetik işlemlerin hepsi Xillinx – Vivado Design Suite programının sunduğu hazır IP çekirdekler ile yapılmıştır. Bu hesaplama işlemlerini yapmak için paralel çalışan bir algoritma geliştirilmelidir. Algoritma geliştirme aşamasında bir ön hesap yapılmıştır. Bu ön hesap single port ram ve Dual Port ram kullanımının hesap süresi üzerine etkisini içermektedir.

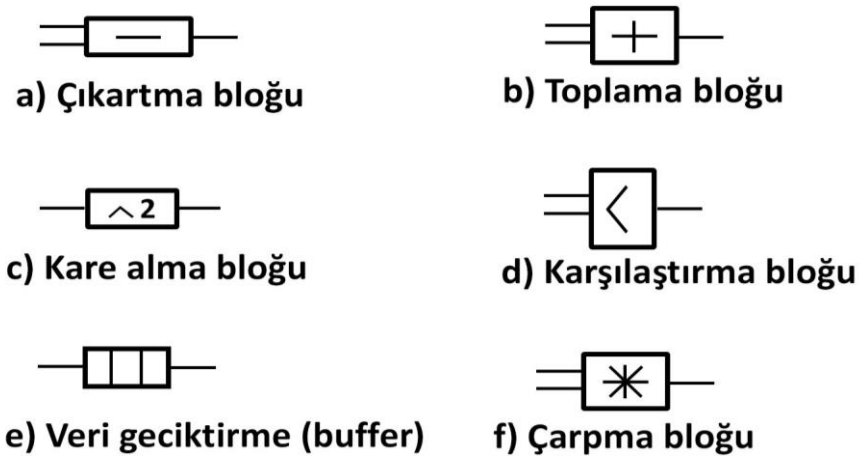
İlk olarak en yakın komşu bulma algoritmasındaki nokta arası mesafe karesi hesaplama işlemini Single Port RAM kullanarak yapılacaktır. Şekil 4.2'de bu algoritma gerçekleşmesini blok diyagram şeklinde görsel olarak anlatan çizelge verilmiştir ve

paralel yapılan işlemler kesikli çizgi içerisine alınmıştır. Burada çıkarma ve toplama işlemi için tek bir blok kullanılmıştır. Yapmak istenen işleme göre kontrol girişi verilerek toplama ya da çıkarma işlemi yaptırılabilir. Bu sebeple bir hedef kaynak nokta çifti için mesafe karesi hesaplama işi bitmeden ikinci bir nokta çiftine geçilememektedir.



Şekil 4.1. Mesafe karesi işlemi blok diyagramı

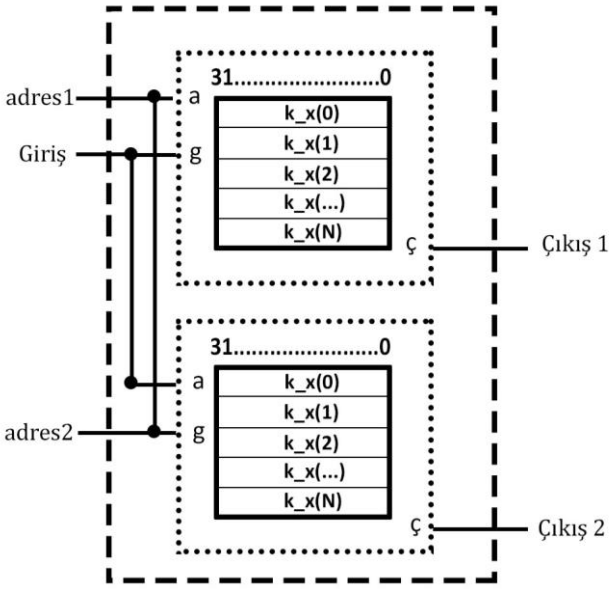
Şekil 4.2'te ise Şekil 4.1'deki blokların ne işlem yaptığı açıklanarak gösterilmiştir.



Şekil 4.2. Blokların açıklaması

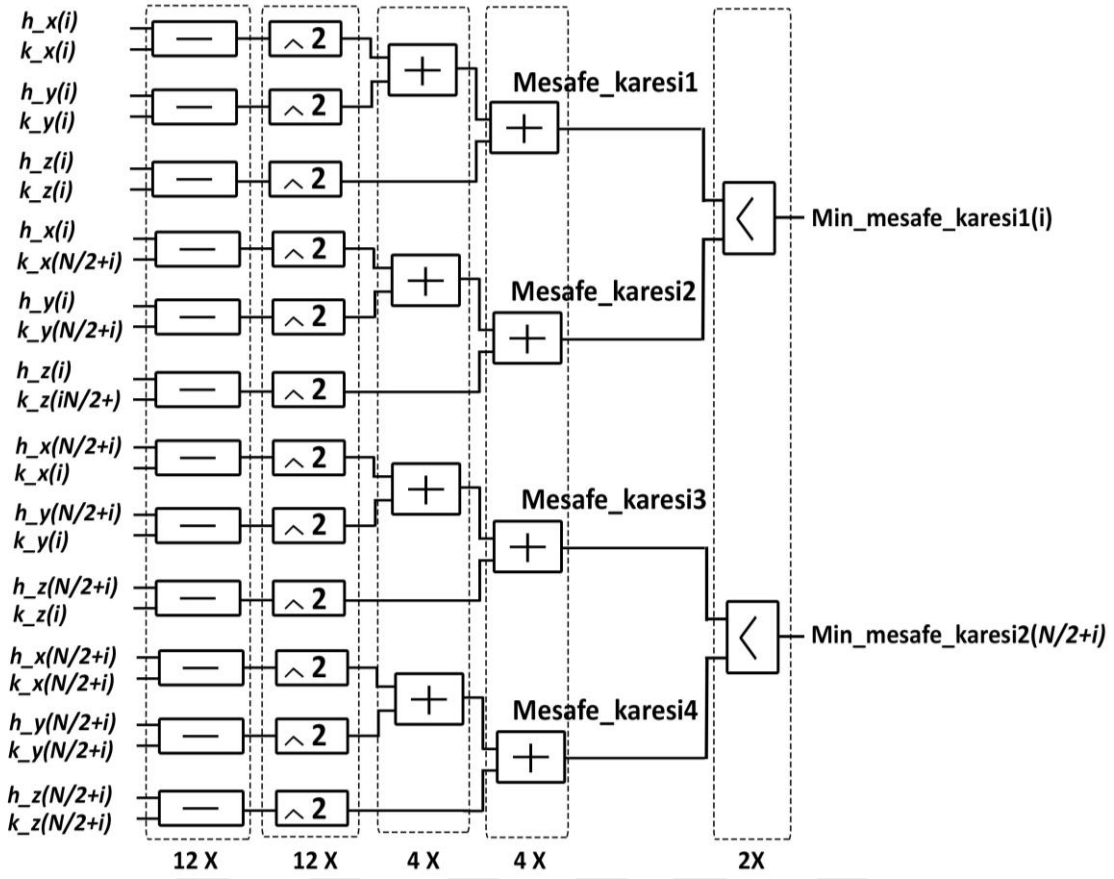
Single Port Ram kullanarak denklem 4.1'deki işlemi yaptırmak istersek önce 3 adet fark alma işlemi aynı anda paralel olarak yapılır ve sonucu karelerini alma işlemine paslanır. Kare alma işlemi de paralel olarak yapılır ve işleminin sonuçları toplama işlemine gönderilerek sonuçlar alınır. Vivado'nun sunduğu IP çekirdekler aynı anda maksimum iki veriyi toplama kapasitesine sahip olduğu için önce iki veri toplanıp sonucu üçüncü veri ile toplanır. Her seri işlenen işlem için geçen süreyi T birim olarak tanımlarsak kaynak nokta bulutundan bir nokta ve hedef nokta bulutundan bir nokta için mesafelerinin karesini hesaplama işlemi, çıkarma için 1T, kare alma için 1T, 2 adet toplama için 2T toplam 4T birim sürmektedir. Mesafe hesaplama işleminden sonra karşılaştırma işlemi

yapılır. Karşılaştırma işlemi ilk gelen mesafe verisini en küçük mesafe olarak alır sonra her gelen veriyi bu veri ile karşılaştırır. Eğer karşılaştırılan veri en küçük olarak atanan veriden küçük ise en küçük mesafeye yeni gelen değer atanır. Değil ise bir sonraki veri ile karşılaştırılır. Bu işlem kaynak nokta bulutundaki her veri için tekrarlanır. Karşılaştırma işlemi için geçen süreyi de T birim olarak kabul ettiğimizde hedef nokta bulutundan 1 nokta için kaynak nokta bulutundan 1 nokta ile mesafe hesaplayıp karşılaştırılması toplam 5T sürmektedir. Örnek olarak N adet kaynak ve M adet hedef nokta olduğunu varsayarsak, single port ram kullanarak en yakın komşu bulma işlemi N*(5T*M)= N*M*5T birim sürede hesaplanmaktadır.



Şekil 4.3. Dual Port RAM veri saklama

Daha sonra nokta bulutu verilerinin Dual Port RAM'lerde kaydedilmesi durumunda süre hesabı yapılmıştır. N uzunluğunda bir Dual Port RAM, içerisinde 2*N'lik bir hafıza bölgesi barındırır. Bu tip RAM'lere yazma yapılırken bir adres ve veri sağlanır. Bu veri, içerisindeki iki RAM'in aynı adresine yazılır. Okuma yaparken ise durum farklıdır. İki adet adres bilgisine ihtiyaç duymaktadır. Aynı anda bu iki farklı adresten veri okunabilir. Şekil 4.3'te kaynak nokta bulutunun *x* koordinat verisinin RAM'de kaydedilmesinin temsili şekli gösterilmiştir. RAM bloğunun iki kısmına da kaynak nokta bulutunun *x* koordinat verisi (k_x) yüklendiği görülmektedir. Bu yöntem ile kaynak ve hedef noktalarının *x*, *y* ve *z* verileri de RAM'e kaydedilmiştir.



Şekil 4.4. Dual Port RAM Kullanarak yapılan işlem diyagramı

Şekil 4.4'te noktalar arası mesafe hesaplama işleminin Dual Port RAM kullanımı ile nasıl paralelleştirildiği gösterilmiştir. Bu gösterimde eş zamanlı yapılan işlemler kesik çizgi içerisine alınmıştır ve eş zamanlı çalışan süreç sayısı görselin altında "X" olarak gösterilmiştir. Kaynak nokta bulutunun veri sayısı N, hedef nokta bulutunun veri sayısı M olsun. Kaynak nokta bulutu için oluşturulan Dual Port RAM'den okuma yaparken "adres1" hattına 0 ile N/2 arasında değerler girilirken, "adres2" hattına ise N/2 ile N arasında değerler girilecektir. Böylece veriler karışmadan eş zamanlı çalışmada eşleme sağlanmaktadır. Bahsedilen yöntem ile adresleme yapıldıktan sonra kaynak nokta bulutu RAM'lerinin birinci çıkışı $k_x(i)$, $k_y(i)$, $k_z(i)$, ikinci çıkışını ise $k_x(\frac{N}{2}+i)$, $k_y(\frac{N}{2}+i)$, $k_z(\frac{N}{2}+i)$ olmaktadır. Hedef nokta bulutu RAM'lerinin birinci çıkışı h_x(i), $h_y(i)$, $h_z(i)$, ikinci çıkışı ise $h_x(\frac{N}{2}+i)$, $h_y(\frac{N}{2}+i)$, $h_z(\frac{N}{2}+i)$ olmaktadır. Böylece aynı anda 2 adet kaynak ve 2 adet hedef verisinin okunabiliyor olması aşağıdaki dört farklı denklemi doğurur.

$$Mesafe\_kare\_0(i) = (h_x(i) - k_x(j))^2 + (h_y(i) - k_y(j))^2 + (h_z(i) - k_z(j))^2 \qquad (4.2)$$

$$Mesafe\_kare\_1(i) = (h_x(i) - k_x\left(\frac{N}{2} + j\right))^2 + (h_y(i) - k_y(\frac{N}{2} + j))^2 + (h_z(i) - k_z(\frac{N}{2} + j))^2 \quad (4.3)$$

$$Mesafe\_kare\_2(\frac{N}{2} + i) = (h_x\left(\frac{N}{2} + i\right) - k_x(j))^2 + (h_y\left(\frac{N}{2} + i\right) - k_y(j))^2 + (h_z\left(\frac{N}{2} + i\right) - k_z(j))^2 \quad (4.4)$$

$$Mesafe\_kare\_3(\frac{N}{2} + i) = (h_x\left(\frac{N}{2} + i\right) - k_x\left(\frac{N}{2} + j\right))^2 + (h_y\left(\frac{N}{2} + i\right) - k_y(\frac{N}{2} + j))^2 + (h_z\left(\frac{N}{2} + i\right) - k_z(\frac{N}{2} + j))^2 \quad (4.5)$$

Denklem 4.2'deki $Mesafe\_kare\_0$ değişkeni, hedef nokta bulutunun *i.* verisinin kaynak nokta bulutunun *i.* verisi arasındaki uzaklıktır. Denklem 4.3'deki $Mesafe\_kare\_1$ değişkeni ise hedef nokta bulutunun *i.* verisi ile kaynak nokta bulutunun $(\frac{N}{2} + i)$. verisi arasındaki uzaklıktır. Denklem 4.4'teki $Mesafe\_kare\_2$ değişkeni, hedef nokta bulutunun $(\frac{N}{2} + i)$. verisinin kaynak nokta bulutunun *i.* verisi arasındaki uzaklıktır. Denklem 4.5'teki $Mesafe\_kare\_3$ değişkeni ise hedef nokta bulutunun $(\frac{N}{2} + i)$. verisi ile kaynak nokta bulutunun $(\frac{N}{2} + i)$. verisi arasındaki uzaklıktır. Görüldüğü üzere hedef nokta bulutunun *i.* verisi için kaynak nokta bulutundan 2 adet mesafe hesaplanmış olunur. Sonraki adımda bu iki veri, küçük olanını ($Min\_mesafe\_kare\_0$) seçmek için, karşılaştırma işlemine tabi tutulur. Bu adım aynı anda hedef noktanın $(\frac{N}{2} + i)$. verisi içinde yapılır ve $Min\_mesafe\_kare\_1$ bulunur. Bu işlemlerin sonucu bize 2 adet hedef nokta için 2 adet kaynak nokta verisi içerisinden en yakını bulunmuş olur.

Dual Port RAM'ların çıkışı birbirinden bağımsız olduğundan 4 adet mesafe karesi hesapla işlemi paralel olarak 4T sürede hesaplanmaktadır. Bu süreye ilaveten 2 adet karşılaştırma süresini eklendiğinde bir çift veri hesaplaması için 6T süre harcanmaktadır. Hem kaynak hem hedef nokta bulutu ikiye bölünerek hesaplandığı için toplam süre (N/2)*6T*(M/2)= N*M(1,5T) olmaktadır. Dual Port Ram kullanarak en yakın komşu bulma algoritmasını single port RAM' ile hız oranı 5T/1,5T= 3,3 olmaktadır. Görüldüğü üzere yaklaşık ~3.5 kat hızlı olmaktadır.

4.2 Aykırı Noktaların Tespiti

Aykırı noktaların tespiti için bir dizinin medyanı hesaplanması gereklidir. Medyan değeri bulunduktan sonra o medyan değerinin belirli bir katı eşik olarak alınır, örn. 3 katı.

Dizinin bu eşikten büyük olan elemanları aykırı nokta olarak değerlendirilir ve diziden çıkarılır. Bu işlem FPGA platformunda başarı ile gerçekleştirilmiştir.

4.2.1 Medyan değerinin bulunması

Medyan sıralanmış bir dizide, dizinin orta noktasındaki sayıdır. Eğer dizi tek sayıda elemana sahip ise, medyan ortadaki değerdir. Dizi çift sayıda elemana sahip ise medyan ortadaki iki değerin aritmetik ortalamasıdır. En yakın komşu buluma işleminde ilişkili noktaların arasındaki mesafe bilgileri medyan değerinin bulunması için ayrı bir RAM'a kaydedilmiştir. İlk geliştirdiğimiz yöntemde medyan değeri bulunurken aşağıdaki adımlar izlenerek yapıldı.

i. Bir taramada vektör içerisindeki değeri en büyük nokta bulunur

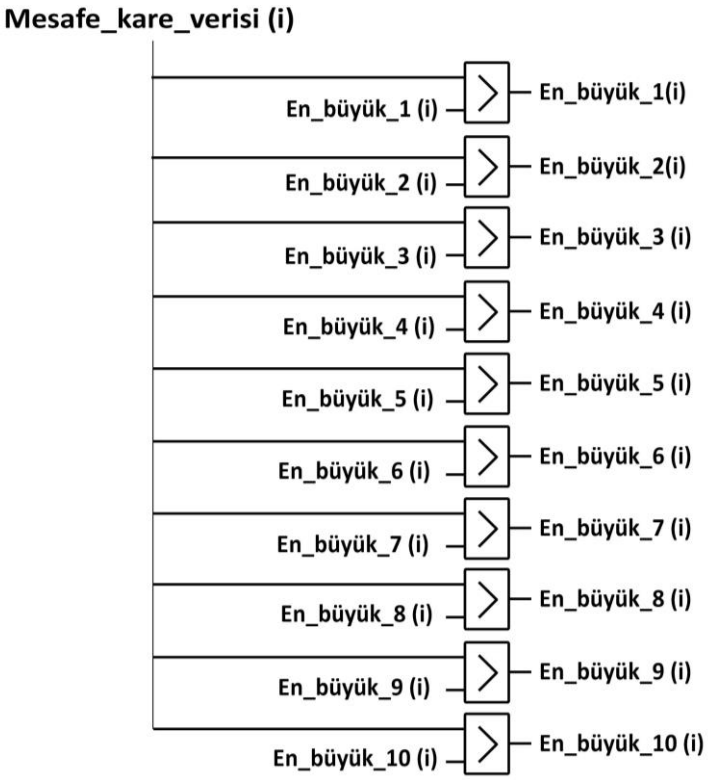
ii. Bulunan noktanın değeri, indisi ve bu noktanın bulunan kaçıncı büyük nokta olduğu kaydedilir.

iii. Bulunan büyük noktanın bulunduğu adrese "0" yazılır.

iv. Bulunan büyük nokta sayısı toplam nokta sayısının yarısına eşit değilse i. adıma gidilir eşitse en son bulunan değer ve indisler sonuç olarak geri döndürülür.

Sıralama için gerekli adım sayısı nokta sayısın yarısı olduğu için tüm noktaları sıralayıp orta noktayı bulmaya göre daha kısa zamanda tamamlanır. Örnek olarak N adet veri için bu yöntem ile süre hesabı (N*N/2) şeklinde olmaktadır. Bu işlem süresine tatmin olmadığımız için, medyan bulma işlemine yeni yaklaşım gerçekleştirilmiştir.

Yeni medyan bulma yöntemi eski yöntemle adımları hemen hemen aynıdır. i. adımda bir taramada vektör içerisinden değeri en büyük 10 veriyi buluyoruz. Bu adımdaki bir taramada en büyük 10 veriyi bulma işlemi paralel 10 adet karşılaştırıcı yardımı ile yapılmıştır. Şekil 4.5'te işlemin şeması verilmiştir.



Şekil 4.5. Mesafe kare verisini on adet veri ile paralel karşılaştırılması

Bu işlemde karşılaştırma işlemi için Vivado'nun hazır Float sayıları karşılaştıran IP çekirdeği kullanılmıştır. Şekil 4.5'te görüldüğü üzere mesafe verisi aynı anda başlangıç değeri sıfır olan 10 adet en_büyük veri seti ile karşılaştırılır. Bu karşılaştırılan verilerden yukarıdan aşağı sıralanarak hangisinden büyük ise o veri yerine mesafe verisi atanır, atanan verinin aşağısında kalan veriler kayma yaptırılarak bir birine atanır. Eğer karşılaştırılan mesafe verisi hepsinden küçük ise herhangi atama ya da kaydırma yaptırılmadan bir sonraki veri okuma işlemine geçer.

ii. Bulunan noktaların değerleri, indisleri ve bu noktaların bulunan kaçıncı büyük nokta oldukları kaydedilir.
iii. Bulunan büyük noktaların bulunduğu adreslerine "0" yazılır.
iv. Bulunan büyük nokta sayısı toplam nokta sayısının yarısına eşit değilse en_büyük veri seti ve indisleri sıfırlanarak i. adıma gidilir eşitse en son bulunan değer ve indisler sonuç olarak geri döndürülür.

Bu en büyük işleminde geliştirdiğimiz yeni yöntem hızını hesaplarsak, N adet veri için N*(N/20) şeklinde olmaktadır. İlk gösterilen yöntem ile hız farkı ~10 kat artmıştır. Hız artımı paralel karşılaştırıcı kullanma sayısı ile artığı görülebilmektedir.

4.3 Dönüşüm Kestirimi

İki nokta bulutu arasında ilişkili noktalar bulunduktan ve aykırı noktalar elendikten sonra yapılması geren işlem en iyi 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. Noktalar hatalı eşleştirildiğinde bulunacak dönme ve öteleme de hatalı sonuç verecektir. Optimum dönüşümün bulunması işlemi aşağıdaki dört temel adımdan oluşur.

Kaynak ve hedef nokta bulutlarından sadece aralarında ilişki olan noktaların ağırlık merkezleri bulunur.

$$\mathrm{k}_{ort} = \frac{1}{N_c}(\sum_{\mathrm{i=1}}^{\mathrm{N_c}} \mathrm{k_i}) \tag{4.6}$$

$$\mathrm{h}_{ort} = \frac{1}{M_c}(\sum_{\mathrm{i=1}}^{\mathrm{M_c}} \mathrm{h_i}) \tag{4.7}$$

Bu denklemde N_c ve M_c sırasıyla kaynak ve hedef nokta bulutlarının ilişkili nokta sayısını gösterirler.

Her iki nokta bulutunun da ağırlık merkezleri orijine taşınarak SVD işlemine tabi tutulacak $\mathrm{M_{3x3}}$ matrisi aşağıdaki denklemdeki gibi hesaplanır.

$$\mathrm{M} = \sum_{\mathrm{i=1}}^{\mathrm{N}} (\mathrm{k_i} - \mathrm{k_{ort}})(\mathrm{h_i} - \mathrm{h_{ort}})^{\mathrm{T}} \tag{4.8}$$

M matrisi daha sonra tekil değer ayrıştırma işlemine tabi tutulur.

$$\mathrm{M} = \mathrm{U\,S\,V^T} \tag{4.9}$$

Rotasyon matrisi aşağıdaki şekilde hesaplanır.

$$R = VU^T \tag{4.10}$$

Öteleme vektörü ise bulunan rotasyon matrisi de hesaba katılarak Denklem 11'de görüldüğü gibi bulunur.

$$\vec{t} = -\mathrm{R} * \mathrm{k}_{\mathrm{ort}} + h_{ort} \quad (4.11)$$

Dönüşüm Kestirimi işleminin ilk iki adımı olan ilişkili noktaların ağılık merkezini ve SVD işlemine tabi tutulacak $\mathrm{M}_{3\mathrm{x}3}$ matrisi bulma işlemi VHDL olarak kodlanabilmektedir. Bu işlem, aynı anda kaynak ve hedef nokta bulutlarının x,y ve z verilerine ulaşıldığından paralel olarak yapılabilmektedir.

Ancak dönüşüm kestirimi işleminin daha sonraki adımları olan, rotasyon matrisi, öteleme vektörü, toplam dönme ve toplam öteleme vektörünün hesaplanması işini yapacak kod parçacığı Vivado ailesinin ürettiği "Vivado High-Level Synthesis" programı kullanılarak üretilmiştir. Vivado High-Level Synthesis, Xilinx tarafından sağlanan bir programdır. Özelliği ise karmaşık işlemlerin, C ya da C++ dilinde kodlanarak VHDL ya da Verilog projesinde kullanılabilecek düzeyde hazır IP çekirdeklerine dönüştürülmesidir. "Vivado HLS" programının içerisinde C ve C++ dilinde kodlanan matris çarpımı, sayıların köklerini hesaplama, SVD algoritması gibi bir takım hazır örnek algoritmalar mevcuttur ve geniş bir kütüphaneye sahiptir. "Vivado HLS" soncunda üretilen IP çekirdek, hız ve alan açısından en iyi şekilde hazırlanır ve kullanıcıya sunulur. "Vivado HLS"de sadece rotasyon matrisi ve öteleme vektörünü bulma işlemi kodlanmamış, aynı zamanda iterasyon durdurma kriterini hesaplayan alt programda yazılmıştır.
Aşağıda C++ kodundan bir parça görülmektedir:

```
// Call SVD
   hls::svd<3, 3, MATRIX_IN_T, MATRIX_OUT_T>(m_i, s_i, u_i, v_i);

   u_i_temp[0][0] = u_i[0][2];
   u_i_temp[1][0] = u_i[1][2];
   u_i_temp[2][0] = u_i[2][2];

   u_i[0][2] = u_i[0][0];
   u_i[1][2] = u_i[1][0];
   u_i[2][2] = u_i[2][0];

   u_i[0][0] = u_i_temp[0][0];
   u_i[1][0] = u_i_temp[1][0];
   u_i[2][0] = u_i_temp[2][0];

   v_i_temp[0][0] = v_i[0][2];
   v_i_temp[1][0] = v_i[1][2];
   v_i_temp[2][0] = v_i[2][2];

   v_i[0][2] = v_i[0][0];
   v_i[1][2] = v_i[1][0];
   v_i[2][2] = v_i[2][0];

   v_i[0][0] = v_i_temp[0][0];
   v_i[1][0] = v_i_temp[1][0];
```

```
v_i[2][0] = v_i_temp[2][0];

    hls::matrix_multiply <hls::NoTranspose, hls::Transpose, 3, 3, 3, 3, 3, 3, MATRIX_OUT_T,
    MATRIX_OUT_T> (v_i, u_i, r_i);

    float determinant = r_i[0][0]*(r_i[1][1]*r_i[2][2]-r_i[1][2]*r_i[2][1])-r_i[1][0]*(r_i[0][1]*r_i[2][2]-
    r_i[0][2]*r_i[2][1])+r_i[2][0]*(r_i[0][1]*r_i[1][2]-r_i[0][2]*r_i[1][1]);

if(determinant<0)
{
   for (int r=0;r<3;r++) {
      v_i[r][2] *= -1;
   }
        hls::matrix_multiply <hls::NoTranspose, hls::Transpose, 3, 3, 3, 3, 3, 3, MATRIX_OUT_T,
    MATRIX_OUT_T> (v_i, u_i, r_i);}

    hls::matrix_multiply <hls::NoTranspose, hls::NoTranspose, 3, 3, 3, 1, 3, 1, MATRIX_OUT_T,
    MATRIX_OUT_T> (r_i, s_cp_i, t_i);

comp_row_loop : for (int r=0;r<3;r++) {
   comp_col_loop : for (int c=0;c<1;c++) {
      t_i[r][c] = -t_i[r][c]+ t_cp_i[r][c];
   }
}

    hls::matrix_multiply <hls::NoTranspose, hls::NoTranspose, 3, 3, 3, 3, 3, 3, MATRIX_OUT_T,
    MATRIX_OUT_T> (Res_R_in_lc, r_i, Res_R_out_lc);

    hls::matrix_multiply <hls::NoTranspose, hls::NoTranspose, 3, 3, 3, 1, 3, 1, MATRIX_OUT_T,
    MATRIX_OUT_T> (Res_R_in_lc, t_i, Res_t_out_lc);
```

"Vivado HLS"'in üretmiş olduğu IP çekirdeğin, yapacağı işlemi kaç saat darbesinde tamamladığı ve kullandığı kaynak miktarı aşağıdaki Çizelge 4.1'de verilmektedir. Nokta bulutu verilerini İşletim Sisteminden alıp, ICP algoritmasını belirli bir iterasyon sayısına kadar tekrarlayan ve sonucu bulan, bulunan sonucu tekrar işletim sistemine gönderen ve bu işlemler için gerekli tüm veri yolu yapısını içeren Programlanabilir Lojik bölümünün toplam kaynak tüketimi Çizelge 4.2'de verilmiştir.

Çizelge 4.1. Vivado HLS'in ürettiği IP çekirdeğin kaynak tüketimi ve hesaplama süresi

Kaynak Türü	Toplam Kaynak	Kullanılan Kaynak	Kullanılan Kaynak (%)
LUT	53200	23350	44
FF	106400	13948	13
Blok RAM	140	16	11
DSP	220	82	37
Clock darbe sayısı * Clock Periyodu = Süre			
7740 * 100 µs = 0.0774 ms			

Çizelge 4.2. Programlanabilir Lojik tarafının toplam kaynak tüketimi

Kaynak Türü	Toplam Kaynak	Kullanılan Kaynak	Kullanılan Kaynak (%)
LUT	53200	27612	51,90
LUTRAM	17400	3153	18,12
FF	106400	33250	31,25
Blok RAM	140	27	19,29
DSP	220	130	59,09
IO	200	79	39,50
BUFG	32	2	6,25
PLL	4	1	25,00

Çizelge 4.3. ICP algoritmasının 10 iterasyon için bulunan sonuçları ve süreleri

Zedboard – PS (İşletim Sistemi) (666 MHZ)			
Rotasyon matrisi (3 x 3) (PS)	1.0000	-0.0001	-0.0002
	-0.0000	0.9986	-0.0524
	0.0002	0.0524	0.9886
Öteleme vektörü (PS)	0.0004 (X)	0.0995 (Y)	-0.0007 (Z)
Hesaplama Süresi (PS)	**4302 milisaniye**		
Zedboard – PL (Programlanabilir Lojik) (100 MHZ)			
Rotasyon matrisi (3 x 3) (PL)	1.0000	-0.0000	-0.0000
	-0.0000	0.9986	-0.0523
	0.0000	0.0523	0.9986
Öteleme vektörü(PL)	-0.0000 (X)	0.0996 (Y)	0.0001 (Z)
Hesaplama Süre(PL)	**1693 milisaniye**		
Ulaşılması gereken sonuç			
Rotasyon matrisi (3 x 3)	1	0	0
	0	0.9986	-0.0524
	0	0.0523	0.9886
Öteleme vektörü	0.0000 (X)	0.1000 (Y)	0.0000 (Z)

Bu tezde aynı nokta bulutu çifti için, dönüşüm kestirim işlemi, sonuçların kıyaslanması adına hem PS hem de PL tarafında yapılır. ARM çekirdekler üzerinde koşan Linux işletim sistemi 666 MHZ frekansta çalışmaktadır. Programlanabilir Lojik tarafı ise 100 MHZ frekansa sahiptir. 1022 adet kaynak nokta bulutu verisi, 1022 adet hedef nokta bulutu verileri için PS tarafı ile PL tarafının sonuçları ve hesaplama sürelerini kıyaslayan Çizelge 4.3 verilmiştir. ICP algoritmasının sonuca yakınsama süreleri kıyaslandığında, çizelgeden de görüleceği üzere, PL, ARM işlemciden süre olarak yaklaşık olarak ~2.5 kat daha performanslı çıkmıştır. Fakat saat frekansları da göz önüne alındığında neredeyse 16 katına kadar işlem kapasitesine sahiptir.

Tarama eşleştirme algoritmasının beş adet farklı veri seti eşleştirmesinde, sadece 1 iterasyon için geçirdiği süre Çizelge 4.4'te görülmektedir. Görüldüğü üzere en çok süre nokta eşleştirme kısmında harcanmaktadır. Bu süreyi azaltmak için en yakın komşu bulma adımı üzerinde hızlandırıcı etki oluşturacak bir yenilik yapılmıştır. Detaylar tezin ilerleyen kısımlarında sunulmuştur.

Çizelge 4.4. ICP algoritması adımlarının süreleri

ICP adımı Veri boyutu (NxM)	Veri 1 1022x1022	Veri 2 995x995	Veri 3 749x749	Veri 4 495x495	Veri 5 989x989
En yakın komşu bulma (ms)	161,9	153,2	86,7	37,7	151,3
Dışsal nokta tespiti (ms)	6,3	5,9	3,4	1,4	5,9
Dönüşüm kestirimi (ms)	0,007	0,007	0,007	0,007	0,007
Diğer ara işlemler(ms)	1,01	0,95	0,83	0,82	1,00
1 iterasyon süresi (ms)	169,3	160,1	91,0	40,0	158,3
En yakın komşu bulma zaman yüzdesi (%)	95.62	95.69	95.27	94.25	95.57

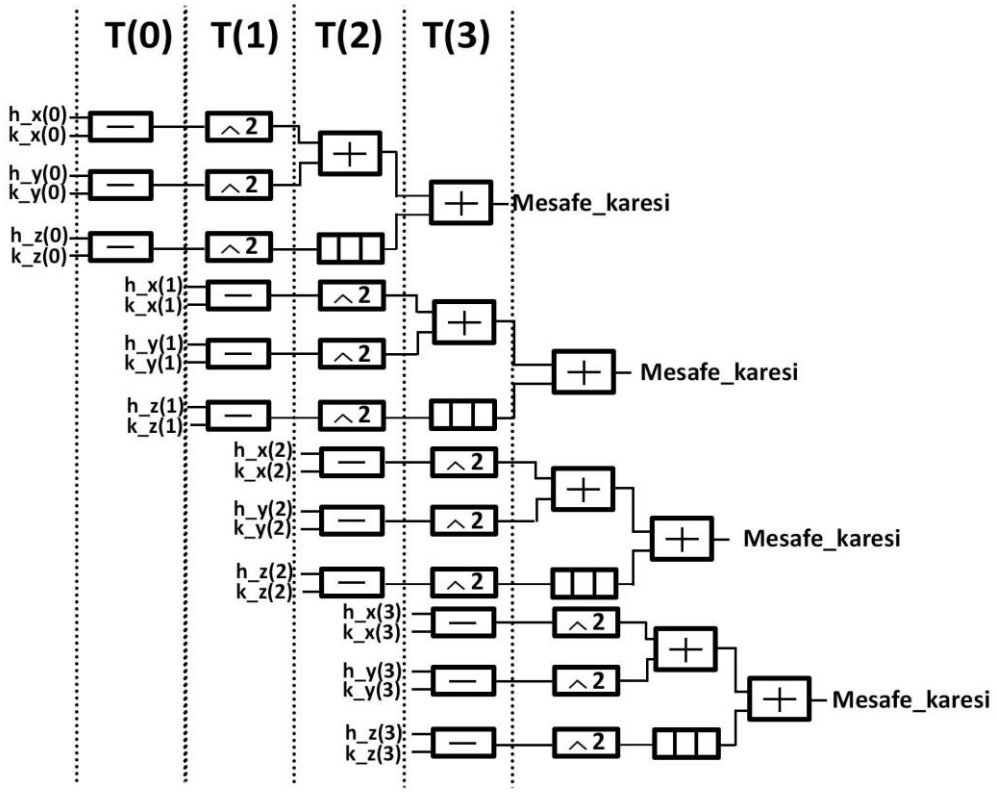
Daha önce geliştirilen en yakın komşu bulma yönteminin içerdiği çıkarma ve toplama işlemi için aynı blok kullanılmıştır. Genel ICP algoritmasında çıkarma ve toplama işlemi sık olduğundan, her bir işlem için ayrı ayrı blok kullanmaktansa en az blok ile bütün toplama çıkarma işlemlerinin yapılabileceği ön görülmüştü. Bunun arkasında yatan sebep daha az IP çekirdeği kullanarak daha az kaynak tüketmektir. Fakat bu stil beraberinde bir dezavantaj getirir. Bu dezavantaj, ardışık gelen toplama veya çıkarma işlemi için aynı blok kullanıldığından, işlemler seri yapılmak zorundadır.

Yöntemdeki hız kazandıran yenilik, tek toplama – çıkarma bloğunun yerine ayrı ayrı toplama ve çıkarma blokları tercih edilmesidir. Bu sayede mesafe kare hesaplama işleminde çıkarma ve toplama işlemi bağımsız bir şekilde yürütülebilmektedir. Ardışık gelen çıkarma ve toplama işlemini eş zamanlı gerçekleştirebiliriz. Şekil 4.6'da yeni komşu bulma algoritmasındaki çıkarma, kare alma toplama bloklarının paralel çalışma şeması verilmiştir.

T(0) anında, çıkarma blokları $h_x(0), h_z(0), h_z(0)$ ve $k_x(0), k_z(0), k_z(0)$ noktaların sıfırıncı verisini işler. T(1) zaman diliminde ise kare alma blokları çıkarma bloklarının sıfırıncı veri için ürettiği sonuçları işleme alır, aynı anda çıkarma blokları bir sonraki hedef ve kaynak verilerini işlemektedir. T(2) zaman biriminde ise 2 adet kare alma bloğu sonucu toplama bloğuna gönderilirken, üçüncü kare alma bloğunun sonucu geciktirme (Buffer) bloğuna iletilmektedir.

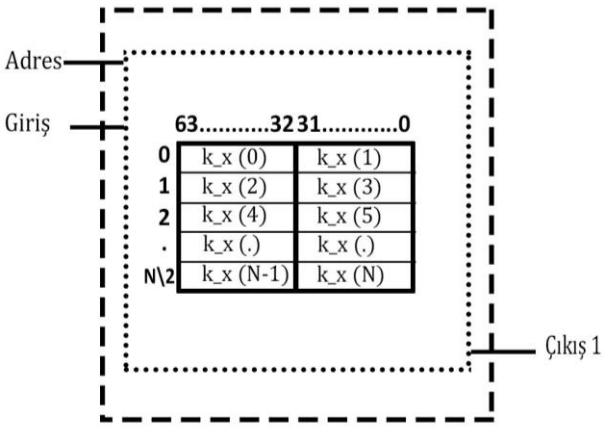
Bu işlemler ile eş zamanlı olarak çıkarma blokları ikinci verileri işlemektedir, kare alma blokları birinci veriyi işleme almaktadır. T(3) zaman biriminde ise geciktirme bloğundan gelen sonuç ile ilk toplama bloğundan gelen sonuç ikinci toplama bloğunda işleme tabi tutulacaktır. Bu işlemler yapılırken çıkarma blokları üçüncü verileri, kare alma ikinci verileri toplama ve bekletme ise birinci verileri işleme almaktadır. T(3) biriminin bitiminde hedefe kaynak nokta bulutunun sıfırıncı verisi için mesafe hesaplamasının ardından her saat darbesinde bir sonraki işlemin sonucunu vermektedir.

Yöntemde kaynak tüketimini azaltmayı hedefleyen bir yenilik daha sunulmuştur. Bu yenilik veri kaydetme stilinde meydana gelmiştir. Verilerimiz 32 bitlik genişliğe sahiptir. 32 bit genişliğinde N derinliğinde Dual Port RAM'lar kullanmaktansa, 64 bit genişliğinde N/2 derinliğinde Single Port RAM kullanmak daha az kaynak tüketmektedir.



Şekil 4.6. Mesafe kare hesaplama işleminin paralel hesaplaması

Bu sayede bir adreste 32 bitlik veriden 2 adet saklayabileceğiz. Tek bir adres vererek iki veriye aynı anda ulaşılabilmektedir. Şekil 4.7'de genişliği arttırılmış RAM şeması çizilmiştir.



Şekil 4.7. Genişliği arttırılmış blok RAM'lar

Şekilde görüldüğü üzere RAM'ın sıfırıncı adresinde kaynak nokta bulutunun *x* koordinatına ait sıfırıncı ve birinci veri bulunmaktadır. Sıfırıncı veri 63. bit ile 32. bit aralığında, birinci veri ise 31. bit ile 0. bit aralığında kaydedilmiştir. Bu yöntemle bir

dizgideki çift indisliler 63-32 bit aralığında, tek indisliler ise 31-0 bit aralığında kaydedilir. Dual Port RAM'lerde bir veriyi iki kere kaydedilmekteydi. Single Port RAM durumunda tek bir kere kaydedilmiştir. Genişliğin 2 kat arttırılması RAM'ın derinliğini 2 kat azalmasını sağlar.

Bu yenilikler sonucunda ortaya çıkan şema Şekil 4.8'de verilmiştir. RAM türü değiştirildiğinden adresleme sitili de değişmiştir. Şekilde gösterilen yatay kalın kesik çizgiler içerisindeki bloklar eş zamanlı olarak çalışmaktadır.

Yeni oluşturulan denklemler aşağıda verilmiştir ve eskisi ile benzer olduğu görülmektedir. Bu denklemlerde yukarıda bahsedildiği üzere *i.* verinin 63-32 bir aralığında kaydedilen çift indisliler $i_ç$, 31-0 bit aralığında kaydedilen tek indisliler ise i_t olarak gösterilmiştir.

$$Mesafe\_kare\_1(i_ç) = (h_x(i_ç) - k_x(i_ç))^2 + (h_y(i_ç) - k_y(i_ç))^2 + (h_z(i_ç) - k_z(i_ç))^2 \quad (4.12)$$

$$Mesafe\_kare\_2(i_ç) = (h_x(i_ç) - k_x(i_t))^2 + (h_y(i_ç) - k_y(i_t))^2 + (h_z(i_ç) - k_z(i_t))^2 \quad (4.13)$$

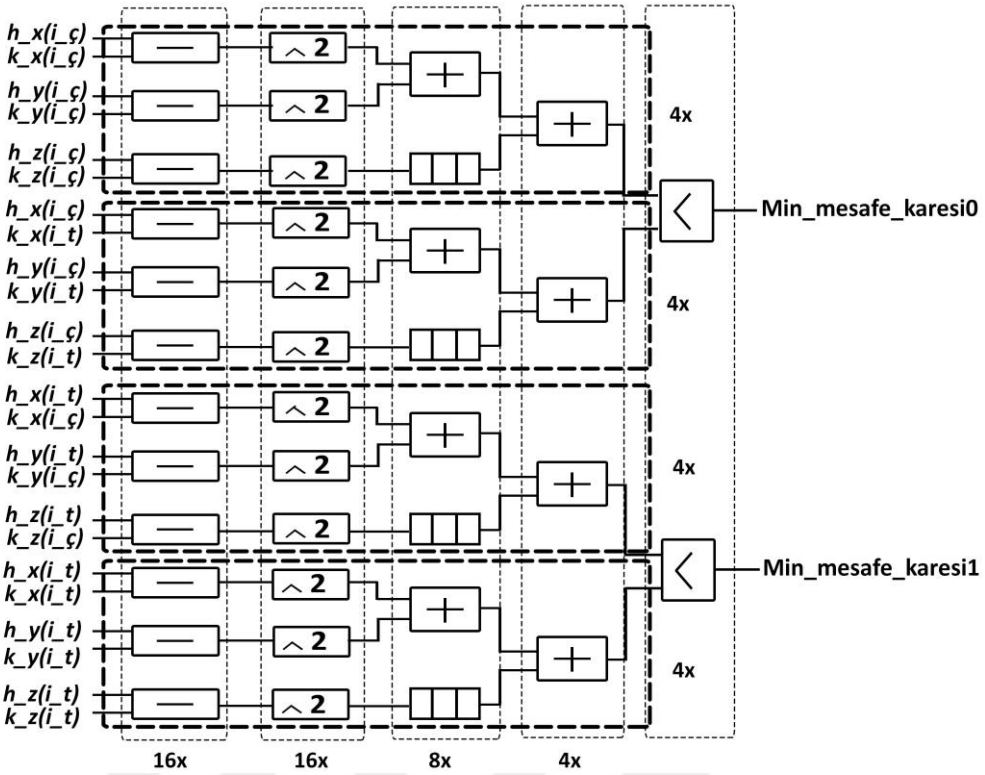
$$Mesafe\_kare\_3(i_t) = (h_x(i_t) - k_x(i_ç))^2 + (h_y(i_t) - k_y(i_ç))^2 + (h_z(i_t) - k_z(i_ç))^2 \quad (4.14)$$

$$Mesafe\_kare\_4(i_t) = (h_x(i_t) - k_x(i_t))^2 + (h_y(i_t) - k_y(i_t))^2 + (h_z(i_t) - k_z(i_t))^2 \quad (4.15)$$

Çizelge 4.5'te 5 adet farklı veri ile PS ve PL için hız, mutlak hata ve hatalı eşleşme sayısını gösteren sonuçlar verilmiştir.

Çizelge 4.5. Yeni En yakın komşu bulma yöntemi sonuçları

Veri boyutu (NxM)	Veri 1 1022x1022	Veri 2 995x995	Veri 3 749x749	Veri 4 495x495	Veri 5 989x989
Eski en yakın komşu bulma PL (ms)	161,9	153,2	86,7	37,7	151,3
Yeni en yakın komşu bulma PL (ms)	2,6	2,475	1,403	0,613	2,445
PL(eski) / PL (yeni) yöntem hız oranı	62.2	61.8	61.9	61.5	61.8
PS (ms)	204,118	193,003	109,155	48,016	189,741
Yeni yöntem PL(yeni) / PS Hız Oranı	78.6	78.1	77.9	78.6	77.4
Eşleşme hatası olan nokta sayısı	1	2	1	2	0



Şekil 4.8. Ayrı Toplam Çıkarma IP çekirdeğini kullanarak yapılan işlem

Çizelgeden görüldüğü üzere yeni geliştirdiğimiz yöntem eski yöntemden yaklaşık 61 kat daha hızlı sonuç üretmektedir. Ayrıca PS tarafı ile hız oranı ise yaklaşık ~78 kat olduğu görülmektedir. Eşleşme hatası olan nokta sayısı göz ardı edilebilecek durumunda olduğu görülmektedir.

BÖLÜM V

SONUÇLAR VE GELECEK ÇALIŞMALAR

Tez çalışması sonucunda tarama eşleştirme probleminde kullanılan ICP algoritması FPGA kartı üzerinde başarıyla gerçeklenmiştir. Öncelikli olarak hem işlemci sitemi (PS) hem programlanabilir lojik (PL) kısmında sonuçlar elde edilip karşılaştırıldığında PL tarafının PS tarafına göre 2.5 kat hızlı sonuç verildiği görülmektedir. İki tarafın frekanslarını da göz önünde bulundurarak hız oranı yapıldığında bu farkın 16 kata çıktığı görülmektedir. PL tarafında gerçeklenen ICP algoritmasın adımlarının ayrı ayrı süreleri tutulup, en fazla zaman alan adımın, en yakın komşu bulma adımı olduğu tespit edilmiştir. Bu adım üzerine hızlandırma çalışması yapılmıştır. İlk çalışma toplama - çıkarma aritmetik işlemini yapan tek blok kullanımı yerine sadece toplama ya da sadece çıkarma işlemini yapan ayrı ayrı blok kullanarak yapılmıştır. Bu çalışma mesafe karesi hesaplama denklemindeki aritmetik işlemlerin eş zamanlı olarak yapılmasını sağlar. Hafızadaki veriler hesaplama bloklarına her darbede ardı ardına gönderildiği için bloklardan çıkan mesafe karesi hesaplama sonucu ilk veriden sonra her darbede bir sonraki verilerin sonuçlarını vermektedir. İkinci hızlandırma çalışması verilerin RAM'e kaydetme ve okuma stilini değiştirerek yapılmıştır. RAM'lerin genişliğini 32 bit'ten 64 bit'e yükselterek bir okumada aynı anda iki veriye ulaşılması sağlanmıştır. Bu sayede aynı anda iki adet hedef nokta bulutu verisi için kaynak nokta bulutundan iki adet veriye mesafe karesi hesaplama işlemi yapılmıştır. En yakın komşu bulma algoritması için yapılan hızlandırma çalışması sonucu önceki yöntemden yaklaşık 62 kat hızlı bir yöntem geliştirilmiştir. RAM genişliği arttırılarak ICP algoritmasını daha hızlı çalıştırmanın mümkün olduğu görülmektedir. Single Port RAM'lar 256 bit genişliğine kadar arttırılabilmektedir. Bu özelliği kullanarak FPGA'da alan yeterliliği sağlandığı sürece algoritmayı daha fazla parçalara ayırıp daha hızlı sistemlerin yapılabileceği görülmektedir.

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.*** 9, 698-700, 1987.

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.

Bonaccorso, F., Muscato, G. and Baglio, S., "Laser range data scan-matching algorithm for mobile robot indoor self-localization", ***World Automation Congress (WAC)***, Puerto Vallarta, Mexico, s.1-5, 24-28 June, 2012.

Censi, A., "An ICP variant using a point-to-line metric", ***Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on***, Pasadena, California, USA, s.19–25, 24-28 June, 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***, Barcelona, Spain, s.2739-2744, 18-22 April, 2005.

Chen, Y. and Medioni, G., "Object modelling by registration of multiple range images", ***Image and Vision Computing*** 10, 145-155, 1992.

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)***, Sao Bernardo do Campo, Brazil, s.103–108, 23-28 October, 2010.

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***, Sendai, Japan, s.3436–3443, 28 September - 2 October, 2004.

Diosi, A. and Kleeman, L., "Fast Laser Scan Matching using Polar Coordinates", ***The International Journal of Robotics Research*** 26, 1125-1153, 2007.

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*** 41, 234–241, 2004.

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.

Kim, D. and Kim, D., "A Fast ICP Algorithm for 3-D Human Body Motion Tracking", ***IEEE Signal Process. Lett.*** 17, 402–405, 2010.

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***, Guilin, China, s.2284–2289, 19-23 December, 2009.

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

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.

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***, Vienna, Austria, s.879–883, 25-26 August, 1996.

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.

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.

Xillybus Turotials, http://xillybus.com/tutorials, 10 Mart 2016

Turk, G. and Levoy, M., "Zippered polygon meshes from range images", ***Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques,*** Orlando, USA, s.311–318, 24-29 July, 1994.

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.

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,*** Paris, France, s.3018–3023, 7-10 November, 2006

Wikipedia, “Programmable Logic/FPGAs?” , https://en.wikibooks.org/wiki/Programmable_Logic/FPGAs, 5 Ağustos 2014.

ÖZ GEÇMİŞ

Bauyrzhan ANARBAYEV 10.02.1992 tarihinde Kazakistan'da Kentau şehrinde doğdu. İlk orta ve lise öğretimini Kentau'da tamamladı. 2009 yılında girdiği ERCİYES Üniversitesi Elektrik-Elektronik Mühendisliği Bölümünden Haziran 2014'te mezun oldu. 2014 yılında Ömer Halisdemir Üniversitesi Elektrik-Elektronik Mühendisliği Ana Bilim Dalı'nda yüksek lisans öğrenimine başladı ve yüksek lisans öğrenimine devam etmektedir. İlgi alanı Gömülü Sistemlerdir.