İSTANBUL TEKNİK ÜNİVERSİTESİ
ELEKTRİK-ELEKTRONİK FAKÜLTESİ

MATRİS AYRIŞIMLARININ GÖRÜNTÜ İSLEME UYGULAMALARI VE FPGA GERÇEKLENMELERİ

LİSANS BİTİRME TASARIM PROJESİ

Furkan Emre IŞIK

Egemen DENİZERİ

ELEKTRONİK VE HABERLEŞME MÜHENDİSLİĞİ BÖLÜMÜ

HAZİRAN, 2022

İSTANBUL TEKNİK ÜNİVERSİTESİ
ELEKTRİK-ELEKTRONİK FAKÜLTESİ

MATRİS AYRIŞIMLARININ GÖRÜNTÜ İSLEME UYGULAMALARI VE FPGA GERÇEKLENMELERİ

LİSANS BİTİRME TASARIM PROJESİ

Furkan Emre IŞIK
(040170089)

Egemen DENİZERİ
(070160217)

Proje Danışmanı: Prof. Dr. Sıddıka Berna ÖRS YALÇIN
Proje Danışmanı: Prof. Dr. Işın ERER

ELEKTRONİK VE HABERLEŞME MÜHENDİSLİĞİ BÖLÜMÜ

HAZİRAN, 2022

İTÜ, Elektronik ve Haberleşme Mühendisliği Bölümü'nün ilgili Bitirme Tasarım Projesi yönergesine uygun olarak tamamen kendi çalışmamız sonucu hazırladığımız "MATRİS AYRIŞIMLARININ GÖRÜNTÜ İSLEME UYGULAMALARI VE FPGA GERÇEKLENMELERİ" başlıklı Bitirme Tasarım Projesi'nin Ara Raporu'nu sunmaktayız. Bu çalışmayı intihal olmaksızın hazırladığımızı taahhüt eder; intihal olması durumunda bitirme tasarım projesinin başarısız sayılacağını kabul ederiz.

Furkan Emre IŞIK .....
(040170089)

Egemen DENİZERİ .....
(070160217)

ÖNSÖZ

Bu proje, İstanbul Teknik Üniversitesi Elektronik ve Haberleşme Mühendisliği Bitirme Tasarım Projesi olarak gerçekleştirilmiştir. Bitirme Tasarım Projesi boyunca bize verdiği değerli destek, gösterdiği sabır ve sağladığı katkılarından dolayı tez danışmanlarıımız Prof. Dr. Berna Sıddıka ÖRS YALÇIN ve Prof. Dr. Işın ERER akademisyenlerimize çok teşekkür ederiz. Ayrıca değerli bilgi ve görüşlerini bizimle paylaştıkları için tüm hocalarımıza ve projeyi geliştirme sürecimiz boyunca bizlere destek olan arkadaşlarımız ve ailelerimize teşekkürlerimizi sunarız.

Haziran 2022

Furkan Emre IŞIK
Egemen DENİZERİ

İÇİNDEKİLER

Sayfa

KISALTMALAR

FPGA	**:** Field-Programmable Gate Array
RAM	**:** Random Access Memory
VSLI	**:** Very Large Scale Integration
SoC	**:** Kırmık Üstü Sistem
IP	**:** Intellectual Property
YNR	**:** Yere Nüfuz Eden Radar
TBA	**:** Temel Bileşen Analizi
TDA	**:** Tekil Değer Ayrışımı
PSNR	**:** Peak Signal to Noise Ratio
NMF	**:** Negatif Olmayan Matris Ayrışımı
MEX	**:** MATLAB Executable File
UART	**:** Universal Asynchronous Receiver-Transmitter

SEMBOLLER

$\boldsymbol{\phi}$	**:** Öz vektör
$\boldsymbol{\lambda}$	**:** Gürbüz negatif olmayan matris ayrışımı algoritması düzenleme parametresi
$\boldsymbol{\Omega}$	**:** Rastgele oluşturulmuş matris
$\boldsymbol{\epsilon}$	**:** Rastgele hale getirilmiş negatif olmayan matris ayrışımı durdurma kriteri

ÇİZELGE LİSTESİ

Sayfa

ŞEKİL LİSTESİ

Sayfa

MATRİS AYRIŞIMLARININ GÖRÜNTÜ İŞLEME UYGULAMALARI VE FPGA GERÇEKLENMESİ

ÖZET

Pojede görüntü işleme uygulamalarında kullanılan bazı temel algoritmalar ve negatif olmayan matris ayrışımları bilgisayar ortamında ve FPGA üzerinde çalıştırılmıştır. Projede üzerinde durulan algortimaların başında temel bileşen analizi, tekil değer ayrışımı ve negatif olmayan matris ayrışımları gelmektedir. Öklid, Kullback-Leibler, İtakiro Saito ıraksama fonksiyonları kullanan negatif olmayan matris ayrışımlarına ek olarak, gürbüz negatif olmayan matris ayrışımı ve rastgele hale getirilmiş negatif olmayan matris ayrışımı kullanılmıştır.

Projede yere nüfuz eden radar görüntüsü kullanılmıştır. Görüntü işleme algoritmaları ile yere nüfuz eden radar görüntülerinde kargaşa giderme yapılmıştır. Kargaşa giderme için kullanılan negatif olmayan matris ayrışım algoritmalarında rank değeri en düşük seçilerek güçlü bileşen yani kargaşa görüntüden çıkarılmış ve hedef görüntü elde edilmiştir. Yapılan kargaşa giderme uygulaması sonucunda literatürdeki sonuçlarla uyumlu olarak gürbüz negatif olmayan matris ayrışımı algoritması en yüksek sinyal gürültü oranını veren algoritma olmuştur. Ancak rastgele hale getirilmiş negatif olmayan matris ayrışımı algoritması uygulama yapılan görüntülerin boyutları hali hazırda küçük olmasından dolayı beklendiği gibi bir hesaplama süresinde hızlanmayı sağlamamıştır.

Projede gürültü giderme uygulaması da yapılmıştır. Bunun için öncelikle yere nüfuz eden radar görüntüsüne ve 512x512 boyutlarında başka bir görsele Gausian gürültü eklenmiştir. Daha sonra gürültü eklenmiş görüntülerde rank değeri görsel boyutuna yakın yüksek bir değer seçilerek tekil değer ayrışımı ve negatif olmayan matris ayrışımı yöntemleri ile gürültü giderme yapılmıştır. Yere nüfuz eden radar görüntüsü için gürültü giderme öncesi ve sonrasında kargaşa giderme işlemi yapılmış ve gürültü gidermenin kargaşa gidermeye olan etkisi gösterilmiştir.

Projede FPGA kullanılmıştır. Bu sayede proje sürecinde tasarım esnekliği sağlanmış ve sonuç vermeyen girişimler söz konusu olduğunda yeni tasarımlara uyumsuzluk yaşanmaksızın gidilmiştir. Bu sayede gerekli gerçeklemeler için daha fazla teknik dokümana ulaşılmıştır. Bunun yanı sıra, Zynq işlemcili donanımlar tercih edilmiştir. Bu tercihlerde ortaya çıkan tek zorluk kullanılan Zynq-7000 işlemcili Zedboard FPGA kartında Linux işletim sisteminin dahili olarak bulunmaması olmuştur. Bu sorunun giderilmesi için Linux işletim sistemine sahip bir RaspberryPi donanımı ana donanım olarak kullanılmış ve kullanılan FPGA bu donanımın uydusu olarak kullanılmıştır.

Bu projede ana donanımdan beklenen giriş vektörlerini ve gerekli rank değerini uydu donanıma iletmesi olmuştur. Uydu donanımından beklenen ise işletim sistemine sahip ana donanımdan gelen verilerin işlenmesi ve geri iletilmesi olmuştur.

Radardan gelen görüntüyü en iyi şekilde işleyen algoritma seçilerek donanım üzerinde gerçeklenmesi sağlanmıştır. Bu gerçeklemenin mümkün olması için uydu bir FPGA donanımı üzerinde radar verisini Gürbüz Negatif Olmayan Matris Ayrışımları algoritması kullanarak işleyebilmesini ve işlediği veriyi ana donanıma iletmesini sağlayacak olan programlar gerçeklenmiştir. Ana donanım olarak Linux işletim sistemine sahip RaspberryPi donanımı tercih edilmiştir. Bu iki taşınabilir donanım sayesinde radardan gelen görüntü verisinin mobil olarak işlenebilmesi amaçlanmıştır. Gürültü giderme algoritması için seçilecek rank değeri hem yazılım hem donanım gerçeklemelerinde toplam piksel hatası bakımından en uygun sonucu veren değer olarak seçilmiş ve nihai sonucun elde edilmesinde bu değer kullanılmıştır.

APPLICATION AND FPGA IMPLEMENTATION OF MATRIX DECOMPOSITION BASED IMAGE PROCESSING

SUMMARY

In this project the fundamental image processing algorithms and non-negative matrix factorization are applied in computer and implemented on FPGA. The algortihms that are included in this project are principal component analysis, singular value decomposition and three different non-negative matfix factorization algorithms. The basic non-negative matrix factorization algortihms that uses Euclid, Kullback-Leibler and Itarkiro-Saito divergence functions, robust non-negative matrix factorization and randomized non-negative matrix factorization are applied in computer work environment.

In applications ground penetrating radar images are used for image processing. The image processing algorithms are used for clutter removal from ground penetrating radar images. Ground penetration radars are very useful devices to detect shallow buried objects under sand or earth but in these images clutter which reduces the performance of target detection algorithms is always presented and it is needed to be removed using image processing algorithms. The clutter removal applications are done by using non-negative matrix factorization instead of singular value decomposition or principal component analysis because non-negative matrix factorizations are much more simpler to implement on FPGA since they are only using matrix operations like multiplication or summation. In order to do clutter removal, the non-negative matrix factorization algortihms are applied by taking the value of rank as small as possible and the most powerful signal is extracted from raw data to left with only the target image except robust non-negative matrix factorization. Robust non-negative matrix factorization assumes the result of matrix factorization will have a sparse error matrix. In robust non-negative matrix factorization sparse error matrix is taken as target image. Our results are mostly compatible with literature. Robust non-negative matrix factorization perform as the best algorithm considering peak signal to noise ratio. However, due to small image sizes randomized non-negative matrix factorization does not perform as hoped in term of compuational power.

In addition to clutter removal, denoising application is also conducted in this project. On this purpose a Gaussian noise is added to ground penetrating radar image and another image with size of 512x512 pixels. After that using high rank non-negative matrix factorization and singular value decomposition algorithms the noisy images are denoised. To evaluate the denoising permormance peak singal to noise ratios are compared and for both of the images singular value decomposition had better singal to noise ratiof. In order to emphasize the importance of denoising for clutter removal in ground penetrating radar images, a clutter removal result with and without denoising is shown.

FPGA was used in the project. In this way, design flexibility was ensured during the project process, and in the case of unsuccessful initiatives, new designs were made without incompatibility. In this way, more technical documents have been reached for the necessary implementations. In addition, hardware with Zynq processors is preferred. The only difficulty that arose in these preferences was the lack of Linux operating system internally in the Zedboard FPGA board with the Zynq-7000 processor used. To overcome this issue, it is decided to use two hardwares. There would be a master-slave relationship between the hardwares to be able to take, transfer and process the data. RaspberryPi hardware with Linux operating system was used as the master hardware and the FPGA used was used as a slave of this hardware. With this cooperation, master hardware RaspberryPi takes the data from radar and transfers it to slave FPGA with Zynq processor and slave FPGA processes the noisy data with Robust Non-negative Matrix Factorization algorithm and transfers it back to master hardware for further applications.

In this project, the input vectors expected from the master hardware and the required rank value were transmitted to the slave hardware. What was expected from the slave hardware was the processing and transmission of the data coming from the master hardware with the operating system.

The algorithm that processes the image coming from the radar in the best way is selected and implemented on the hardware. In order for this implementation to be possible, programs have been implemented on a slave FPGA hardware that will enable it to process radar data using the Robust Non-Negative Matrix Decomposition algorithm and transmit the processed data to the master hardware. RaspberryPi hardware with Linux operating system was preferred as the master hardware. Thanks to these two portable hardware, it is aimed to process the image data coming from the radar mobile.

The rank value to be selected for the noise removal algorithm was chosen as the value that gives the most appropriate result in terms of total pixel error in both software and hardware implementations, and this value was used to obtain the final result.

1. GİRİŞ

Projede görüntü işleme yöntemleri yere nüfuz eden radar görüntülerinde kargaşa giderme ve gürültü giderme problemleri için bilgisayarar üzerinde uygulanmıştır. Ardından uygulanan algoritmalardan negatif olmayan matris ayrışımı ile kargaşa giderme ve gürültü giderme algoritmaları FPGA için gerçeklenmiştir. Bilgisayar ortamında yere nüfuz eden radar görüntülerinde gerçekleştirilen görüntü işleme uygulamalarının FPGA üzerinde gerçekleştirilebileceği gösterilmiştir.

1.1 Projenin Amacı

Projede yere nüfuz eden radar görüntülerinin matris ayrışım yöntemleri ile işlenmesi ve seçilen bir algoritmanın FPGA üzerine (taşınabilir olacak şekilde) düşük maliyetli yüküyle gerçeklenmesi amaçlanmaktadır. MATLAB ortamında gerçekleştirilecek uygulamalarda literatür taraması ile uyumlu sonuçlar elde edilmesi ve çıktıları verecek olan donanımın hızlı çalışması beklenmektedir.

2. TEMEL BİLGİLER

Bu bölümde projede kullanılacak bazı temel kavramlar ile ilgili bilgiler yerilecektir. İlk olarak görüntü işleme ve gürültü giderme hakkında temel bilgiler verilecek daha sonra FPGA temel olarak anlatılacaktır.

2.1 Görüntü işleme

Görüntü işleme, dijital bir ortamdaki görsellerden istenen amaca göre farklı tekniklerle görsel bilginin elde edilmesidir. Görüntü işleme uygulamaları dijital ortamdaki görüntünün kalitesini arttırmak, gürültüyü gidermek, görüntüdeki belirli desenleri ayırmak ve görünmesi zor nesneleri görünür yapmak gibi amaçlarla çok yaygın bir biçimde kullanılmaktadır. Özellikle günümüzde yüz tanıma, görüntülerin iyileştirilmesi, nesnelerin ayrıştırılması ve yer nüfuz eden radar görüntülerinde kargaşa giderme ile hedef tespiti gibi birçok farklı alanda görüntü işleme uygulamalarından faydalanılır.

Görüntü işleme problemlerinde dijital görüntü iki boyutlu bir f(x,y) fonksiyonu olarak düşünülebilir. Burada x ve y uzaysal koordinatları belirtirken f fonksiyonunun genliği ise resmin (x,y) koordinatlarındaki yoğunluğunu ya da grilik seviyesini belirtmektedir. Bilgisayar tarafından işlenecek dijital görüntüler sonlu adet elemandan oluşmaktadır ve bu elemanlara piksel adı verilmektedir.

2.2 Gürültü Giderme

Dijital görüntüler elde edilirken ışık, kamera gibi unsurlar görüntüde bozulmalara, gürültülere neden olabilmektedir. Orijinal görüntünün gürültülerden arındırılması görüntü işleme uygulamaları için büyük önem arz etmektedir. Günümüzde gürültü giderme için temel bileşen analizi ve tekil bileşen ayrışımı gibi birçok farklı yöntemden faydalanılmaktadır. Gürültü giderme problemi Y orijinal görüntü, X elde edilmek istenen görüntü ve N gürültü olacak şekilde Y = X + N olarak formüle edilebilir. Çözüm formüle edilirken N, ortalaması sıfır olan Gauss gürültüsü olarak alınabileceği gibi, birçok çözümde gürültü istatistiksel olarak minimuma yakınsanarak da formüle edilmektedir.

2.3 FPGA

Yapılandırılabilir mantık blokları (CLB'ler), giriş/çıkış blokları (IOB'ler) ve programlanabilir ara bağlantılar, bir FPGA'yı oluşturan üç temel parçadır. RAM, kablolu çoğaltıcılar, çoklu biriktirme birimi, yüksek hızlı saat yönetimi devresi ve seri alıcı-vericiler gibi özel bloklar da gelişmiş FPGA sistemlerine dahildir. Sabit nokta uygulamaları, erken FPGA cihazlarının birincil odak noktasıydı. Ancak, VLSI teknolojisindeki önemli gelişmeler nedeniyle, günümüzün FPGA'ları kayan nokta işlemlerini desteklemek için yeterli donanım kaynaklarına sahiptir.

2.4 Kırmık Üstü Sistem

SoC olarak da bilinen Kırmık Üstü Sistem, tüm elektronik veya bilgisayar sistemini tek bir alt tabaka üzerinde birleşen bir entegre devredir. Merkezi bir işlemci birimi, giriş ve çıkış bağlantı noktaları, dahili bellek ve analog giriş ve çıkış blokları, bir Kırmık Üstü Sistem kendi üzerinde barındırdığı alt komponentlerdir. Bir çip boyutuna küçültülmüş sistemde tasarıma bağlı olarak sinyal işleme, kablosuz iletişim, yapay zeka gibi fonksiyonlar gerçeklenebilir.

2.5 IP Entegratörü ve Blok Tasarım

IP (Intellectual Property) Entegratörü, ortak bir kullanıcı arabirimi aracılığıyla dahili bir kütüphanede barındırılan hazır fonksiyonel blokların birbirlerine hiyerarşik düzende bağlanmasına izin veren bir grafik kullanıcı arabirimidir. Blok tasarım ise gömülü sistemi elle kodlamak yerine var olan fonksiyonel blokları tasarımın amacına göre entegre ederek örnek yazılım uygulamasının donanım üzerinde gerçeklenmesidir.

3. MATRİS AYRIŞIMLARININ GÖRÜNTÜ İŞLEME UYGULAMALARI

Bu bölümde yere nüfuz eden radar görüntülerinin işlenmesinde kullanılan algoritmalar ile ilgili bir literatür taraması yer almaktadır. YNR görüntülerindeki gürültü ve kargaşa giderimi problemi için hangi algoritmaların nasıl kullanıldığı, zaman içerisinde hangi gelişmelerden geçtikleri ve hangisinin nasıl sonuçlar verdiği anlatılacaktır.

3.1 Yere Nüfuz Eden Radar Görüntüsünün İşlemesi ile İlgili Literatür Araştırması

Yere nüfuz eden radar (YNR) görüntülerinde yüzeye yakın gömülü bir cismi tespit etmek için radar görüntüsü ile görüntüde yaygın bir biçimde bulunan kargaşayı (clutter) birbirinden ayırmak gerekmektedir. Bu amaç için farklı görüntü işleme algoritmaları ve yaklaşımlarından faydalanılabilir. Matris ayrışımı, YNR görüntüsü işleme alanında gürültü gidermede yaygın bir şekilde kullanılan bir yöntemdir. Matris ayrışımları ile görüntü sinyal ve gürültü olarak bölünür, hedeflenen cisim kargaşaya oranla küçük kaldığından elde edilen gürültü hedeflenen cismin görüntüsünü verir. YNR görüntüsü işlemede, temel bileşen analizi [1,2], tekil değer ayrışımı [3,4] ve negatif olmayan matris ayrışımı [5,6], [7], [8] yaygınca kullanılan matris ayrıştırma yöntemlerinden bazılarıdır.

3.1.1 Temel bileşen analizi ve tekil değer ayrışımı

Temel bileşen analizi, birbirleriyle ilişkili değişkenleri birbiri ile ilişkili olmayan bir grup değişkene dönüştüren bir boyut azaltma yöntemidir ve görüntü işleme uygulamalarında sıklıkla kullanılır [9]. 2001 yılında yapılan bir araştırmada [1], kuru kum altında demir ve plastik hedeflerin YNR görüntüsü ile tespiti üzerine çalışılmıştır. Çalışmalarında temel bileşen analizi ile demir ve plastik hedefleri kargaşadan ayrıştırmayı başarmışlardır. Aynı çalışmada temel bileşen analizi ve bağımsız bileşen analizi karşılaştırılmış, bağımsız bileşen analizinin daha iyi sonuç verdiğini belirtilmiştir. Yapılan başka bir araştırmada daha temel bileşen analizi ve bağımsız bileşen analizi karşılaştırılmıştır [2]. İki algoritma karşılaştırıldığında bağımsız bileşen analizinin daha iyi sonuç verdiği ancak temel bileşen analizinin nümerik olarak uygulanmasının daha kolay olduğu belirtilmiştir. Tekil değer

ayrışımı, görüntü işleme uygulamalarında esas sinyal ve gürültüyü ayrıştırmada kullanılan başka bir matris ayrışım yöntemidir. 2005 yılında yapılan bir çalışmada [3], YNR görüntüsündeki metal olmayan hedef tespiti için tekil değer ayrışımı algoritmaları kullanılmıştır. Tekil değer ayrışımı algoritmaları kargaşa giderme konusunda iyi sonuçlar vermiştir. Yakın dönemde yapılan bir araştırmada temel bileşen analizi ve tekil değer ayrışımının performansları karşılaştırılmıştır [8]. Elde edilen sonuçlara göre tekil değer ayrışımı %25'e kadar daha iyi performans göstermiştir.

3.1.2 Negatif olmayan matris ayrışımı

Bir diğer matris ayrışımı yöntemi de negatif olmayan matris ayrışımıdır ve görüntü işleme uygulamalarında sıklıkla kullanılmaktadır. YNR görüntülerinde negatif olmayan matris ayrışımı ile hedef tespiti üzerine birçok araştırma bulunmaktadır. [5] Negatif olmayan matris ayrışımı kullanarak YNR görüntüsünden kargaşa gidermeyi başarmış ve önceki çözüme göre daha iyi performans göstermiştir. 2018 yılında yapılan bir çalışmada [6] negatif olmayan matris ayrışımı ile temel bileşen analizi ve tekil değer ayrışımı karşılaştırılmıştır. Çalışmaya göre negatif olmayan matris ayrışımı diğer yöntemlerden daha başarılı sonuçlar vermiştir. 2018 yılında yayınlanan bir başka araştırmada hesaplama süresi diğer negatif olmayan matris ayrışım algoritmalarına göre daha hızlı olan rastegele hale getirilmiş negatif olmayan matris ayrışımı üzerine çalışılmıştır [7]. Sunulan rastgele hale getirilmiş hiyerarşik dönüşümlü en küçük kareler algoritması, karşılaştırıldığı deterministik algoritmaya göre daha hızlı bir performans göstermiştir. 2020 yılında yapılan çalışmada [8] YPR görüntüleri gürbüz negatif olmayan matris ayrışımı yöntemi kargaşa gidermek için kullanılmıştır. Araştırma sonucunda gürbüz negatif olmayan matris ayrışımının, hedef tespitinde temel bileşen analizi ve negatif olmayan matris ayrışımından daha iyi sonuçlar verdiği sonucuna varılmıştır. Üstelik gürbüz negatif olmayan matris ayrışımı işlem hızı olarak da diğer algoritmalara üstünlük sağlamıştır. Bu sebeple projede, bahsedilen bütün yöntemler yazılım ile uygulanacak ancak FPGA gerçeklemesi gürbüz negatif olmayan matris ayrışımı için yapılacaktır.

3.2 Gürültü Giderme ile İlgili Literatür Araştırması

Günümüze dek literatüre gürültü giderme problemi için birçok farklı yöntem ile katkıda bulunulmuştur. Literatür incelendiğinde kısmi diferansiyel denklem temelli yöntemler [10], toplam varyans temelli yöntemler [11] ve komşu pikselleri temel alan filtreler [12] gibi gürültü giderme yöntemleri üzerine çalışmalar yapıldığı görülmektedir. Yaygın bir şekilde kullanılan diğer bir teknik ise frekans bileşeni temelli filtreleme yöntemleridir, özellikle frekans bölgesinde alçak geçiren süzgeç kullanmak ve wavelet (dalgacık) bölgesinde filtreleme yapmak bu yöntemler arasında sayılabilir [13]. Bu yöntemlerin yanı sıra gürültü giderme problemi için boyut indirgeme tekniklerinden de faydalanılmaktadır. Bağımsız bileşen analizi bu tekniklerden biridir. Temel olarak bağımsız bileşen analizi, çok bileşenli bir sinyali birbirinden bağımsız olduğu kabul edilen toplanabilir alt bileşenlere bölmektedir ve bu alt bileşenler kullanılarak gürültü giderme uygulamaları yapılabilir [14].

Kullanılan bir diğer boyut indirgeme yöntemi temel bileşen analizidir (TBA). TBA, bağımsız bileşen analizinden farklı olarak değişkenlerin birbiri ile ilişkili olduğu varsayar ve sinyalin en temel bileşenleri ile gürültüden arınmış görüntü elde edilir. 2004 yılında yapılan bir çalışmada görüntülerdeki aydınlatma ve bozulma gibi lineer olmayan unsurların TBA ile tam olarak açıklanamayacağı öne sürülerek çekirdek temel bileşen analizi kullanılmıştır [15]. Uzaklık temelli çekirdek temel bileşen analizini farklı poz, ifade ve aydınlıktaki yüz görselleri ile test etmişlerdir. Elde edilen sonuçlara göre önerdikleri yöntem temel bileşen analizinden daha iyi sonuç vermiştir. 2010 yılında yapılan bir araştırmada görüntüdeki bölgesel dokunun daha iyi korunması için ise TBA analizi bölgesel piksel gruplama algoritması ile birlikte kullanılmıştır [16]. Önerdikleri yöntemde önce görüntü bölgesel piksel gruplama kullanılarak benzerlik gösteren piksel bloklarına bölünmekte ardında da her bir blok için TBA yapılmaktadır. Sonuçlara göre önerdikleri algoritma hem görüntüdeki detayları koruduğu gibi hem de karşılaştırdıkları diğer yöntemlere göre daha iyi sonuç vermiştir.

Son olarak literatür araştırması sonucu değinilmesi gereken boyut indirgeme yöntemi tekil değer ayrışımıdır (TDA). TDA bir matris ayrışımı tekniğidir. TDA ile görüntü parçalara bölünüp ham görüntü TDA sonucunda bulunan öz resimlerin

toplamı şeklinde ifade edilebilir ve düşük dereceli öz resimler gürültü olarak elde edilir. TDA görüntü işleme alanında gürültü gidermede sıklıkla kullanılan yöntemlerdendir. 2005 yılında yayınlanan bir makalede resim ve hava haritalarında test etmek üzere iki boyutlu ve düşük derece varsayımlı tekil değer ayrışımı algoritması üzerine çalışma yapılmıştır [17]. Elde edilen sonuçlarda sundukları algoritma standart TDA ile kıyaslandığında daha iyi sonuçlar vermiştir. 2013 yılında yapılan bir araştırmada ise gürültü giderme problemi için bir yüksek dereceli TBA algoritması üzerine çalışılmıştır [18]. Önerdikleri yöntem elde edilen sonuçlar göre karşılaştırdıkları diğer yöntemlere göre ya daha iyi sonuçlar elde etmiş ya da daha hızlı bir performans göstermiştir. Çalışmada önerilen yöntem TBA temelli tekniğe göre de daha başarılı olmuştur. 2015 yılında gerçekleştirilen bir çalışmada gürültü giderme için TBA ve TDA yöntemleri karşılaştırılmıştır [19]. Yapılan çalışmanın sonucuna göre blok tabanlı TDA gri ölçekli görüntüler için daha başarılı olurken renkli görüntüler için TBA daha iyi sonuç vermiştir.

3.3 Yöntem

Bu bölümde YNR görüntülerinde gürültü ve kargaşa giderme amacıyla kullanılan temel bileşen analizi, tekil değer ayrışımı, negatif olmayan matris ayrışımı, rastgele hale getirilmiş negatif olmayan matris ayrışımı ve gürbüz negatif olmayan matris ayrışımı yöntemleri anlatılmıştır.

3.3.1 Temel bileşen analizi

Temel bileşen analizi (TBA) için $X_{ij,}$ M x N boyutunda (i = 1,2,..,M; j = 1,2,…, N) bir görüntüyü temsil edecek şekilde olursa X matrisinin N temel bileşeni (1) ile gösterilebilir.

$$Y = A^T X \tag{1}$$

$X = [x_1, x_2,\ldots, x_n]^T$ sıfır ortalamalı giriş vektörü, $Y = [y_1, y_2,\ldots, y_n]^T$ temel bileşenler vektörü isimli çıkış vektörüdür. A ise büyükten küçüğe sıralanmış N öz vektörden oluşan $A = [\phi_1, \phi_2,\ldots, \phi_N]^T$ olacak şekilde bir dönüşüm matrisidir.

Temel bileşenler matrisi S, (2) eşitliğindeki gibi olursa

$$S = A^T X \tag{2}$$

kargaşa giderme problemi için en büyük öz vektör ϕ_1 kargaşayı temsil edecek şekilde görüntünün matrisi aşağıdaki gibi ayrışabilir [9].

$$X = X_K + X_{Hedef} = A_1^T S_1 + \sum_{i=2}^{N} A_i^T S_i \quad (3)$$

TBA ile elde edilen temel bileşenlerden düşük dereceli olanları gürültüyü ifade etmektedir. Gürültü giderme işlemi için yüksek dereceli temel bileşenler ile görüntü yeniden oluşturulur.

YNR görüntüsünde kargaşa giderme problemi ise gürültü gidermeden farklı olarak giderilecek olan kargaşa yüksek dereceli bileşenler ile elde edilmektedir. Bu durumda kargaşa giderme problemi için TBA kullanılırsa yüksek dereceli bileşenler kargaşayı elde etmemizi sağlarken düşük dereceli bileşenler YNR görüntüsündeki hedefi elde etmemizi sağlar.

3.3.2 Tekil değer ayrışımı

Tekil değer ayrışımı (TDA), görüntü işleme uygulamalarında esas sinyal ve gürültüyü ayrıştırmada kullanılan başka bir matris ayrışım yöntemidir. Görüntüyü temsil eden aynı X matrisinin tekil bileşen ayrışımı

$$X = USV^T \quad (4)$$

şeklinde verilebilir [3]. U matrisi (M x M) boyutunda, V matrisi de (N x N) boyutunda bir üniter matristir. S = diag($\sigma_1, \sigma_2, \ldots, \sigma_r$) olacak şekilde r = N < M için

$$X = \sum_{i=1}^{N} \sigma_i u_i v_i^T \quad (5)$$

şeklinde belirtilebilir. Kargaşa giderme problemi için S büyükten küçüğe doğru sıralı olduğunda ve ilk öz resim kargaşa olarak kabul edilirse X görüntüsünün ayrışımı aşağıdaki gibidir [8].

$$X = X_K + X_{Hedef} = \sigma_1 u_1 v_1^T + \sum_{i=2}^{N} \sigma_i u_i v_i^T \quad (6)$$

TDA yönteminde elde edilen öz resimlerden düşük dereceli olanları gürültüyü ifade etmektedir. Gürültü giderme problemi için yüksek dereceli öz resimler ile görüntü yeniden oluşturulabilir.

Kargaşa giderme probleminde ise gürültü gidermeden farklı olarak giderilmek istenen olan kargaşa yüksek dereceli bileşenler ile elde edilmektedir. Bu durumda kargaşa giderme problemi için TDA kullanılırsa yüksek dereceli bileşenler kargaşayı elde etmemizi sağlarken düşük dereceli bileşenler yere nüfuz eden radar görüntüsündeki hedefi elde etmemizi sağlar.

3.3.3 Negatif olmayan matris ayrışımı

Negatif olamayan matris ayrışımı, negatif olmayan (M x N) boyutlu X matrisinin negatif olmayan (M x K) boyutlu W ve (K x N) boyutlu H matrislerinin çarpımı olarak yakınsanması şeklinde ifade edilebilir [6].

$$X \approx WH \tag{7}$$

K < min(M,N) olacak şekilde ayrışımın derecesini belirtmektedir. Yakınsama problemi aşağıdaki gibi bir optimizasyon problemine dönüştürülebilir.

$$W,H = \arg min\, D(X;W,H) \;\; ; \;\; W,H \geq 0 \tag{8}$$

D, W ve H değerini bulan ve minimize edilen bir maliyet ya da ıraksama fonksiyonudur. Çözüm için farklı ıraksama ya da maliyet fonksiyonları kullanılabilir. Bu fonksiyonlardan bazıları Öklid uzaklığı, Kullback-Leibler uzaklığı ve Itakura-Saito uzaklığıdır. Negatif olmayan matris ayrışımları algoritmalarında başlangıçta rasgele oluşturulan W ve H matrisleri bir döngü içerisinde maliyet fonksiyonunu minimize edecek şekilde güncellenirler. Farklı maliyet fonksiyonları için belirlenen güncelleme kuralları aşağıdaki gibidir.

$$\text{Öklid} : W \leftarrow W \bullet \frac{WH^T}{WHH^T}, \quad H \leftarrow H \bullet \frac{W^T X}{W^T WH} \tag{9a}$$

$$Kullback-Leibler : W \leftarrow W \bullet \frac{\frac{X}{WH}H^T}{\mathbf{1}H^T}, \quad H \leftarrow H \bullet \frac{W^T\frac{X}{(WH)^2}}{W^T\frac{\mathbf{1}}{WH}} \tag{9b}$$

$$Itakiro-Saito : W \leftarrow W \bullet \frac{\frac{X}{(WH)^2}H^T}{\frac{1}{WH}H^T}, \quad H \leftarrow H \bullet \frac{W^T\frac{X}{(WH)^2}}{W^T\frac{\mathbf{1}}{WH}} \tag{9c}$$

Kargaşa giderme problemi için negatif olmayan matris ayrışımı kullanılmak istenirse görüntüdeki kargaşa, hedeften çok daha kuvvetli olduğundan en anlamlı

ayrışımı veren derece K = 1 olacak şekilde belirlenirse X görüntüsü aşağıdaki şekilde ayrıştırılabilir.

$$X = X_{Hedef} + X_K = X_{Hedef} + W_{Mx1}H_{1xN} \tag{10}$$

Negatif olmayan matris ayrışı gürültü giderme problemi için de kullanılabilir. Gürültü giderme problemi için negatif olmayan matris ayrışımı kullanılmak istendiğinde daha büyük rank değerleri ile negatif olmayan matris ayrışımı yapılabilir. Bu durumda yakınsanan W ve H matrislerinin çarpımı resimdeki güçlü bileşeni, yani gürültüden arınmış bileşeni verecektir.

3.3.4 Gürbüz negatif olmayan matris ayrışımı

2020 yılında yapılan bir çalışmada ise YNR görüntüsü işlemede gürbüz negatif olmayan matris ayrışımı uygulanmıştır [8]. Gürbüz negatif olmayan matris ayrışımında görüntü matrisi X baştan seyrekçe bozuk kabul edilir. Bu durumda eşitlik (7) aşağıdaki gibi değişir.

$$X \approx WH + S \tag{11}$$

S, (M x N) boyutunda seyrek hata matrisidir. Bu durumda optimizasyon problemi aşağıdaki gibidir.

$$\min_{W,H,S} \left|\left|X - WH - S\right|\right|_F^2 + \lambda \left|\left|S\right|\right|_1 \quad ; W, H \geq 0\,, \lambda > 0 \tag{12}$$

ve

$$\left|\left|S\right|\right|_1 = \sum_{i=1}^{m}\sum_{j=1}^{n}\left|S_{ij}\right| \tag{13}$$

$||.||_F$, Froebenius normu, λ düzenleme parametresidir. W ve H için güncelleme kuralları aşağıdaki gibi belirlenmiştir.

$$W_{ij} \leftarrow \left[\frac{\left|\left((S-X)H_T\right)_{ij}\right| - \left((S-X)H_T\right)_{ij}}{2(WHH^T)_{ij}}\right] W_{ij} \tag{14a}$$

$$H_{ij} \leftarrow \left[\frac{\left|\left(W^T(S-X)\right)_{ij}\right| - \left(W^T(S-X)\right)_{ij}}{2(W^TWH)_{ij}}\right] H_{ij} \tag{14b}$$

W ve H aşağıdaki şekilde normalleştirilmektedir.

$$W_{ij} \leftarrow \frac{W_{ij}}{\sqrt{\sum_{r=1}^{n} W_{kj}^2}} \tag{15a}$$

$$H_{ij} \leftarrow H_{ij} \sqrt{\sum_{r=1}^{n} W_{ki}^2} \tag{15b}$$

S için güncelleme kuralı da aşağıda belirtildiği gibidir.

$$S \leftarrow T_{\frac{\lambda}{2}}(X - WH) \tag{16}$$

Kargaşa giderme problemi için rank değeri 1 gibi çok düşük bir değer seçilerek matris ayrışımı gerçekleştirilir. Yakınsanan W ve H matrislerinin çarpımı kargaşa ve S hedef görüntü olarak elde edilir.

Gürültü giderme için gürbüz olmayan negatif matris ayrışımı kullanılmak istenir ise rank değeri daha büyük seçilerek matris ayrışımı gerçekleştirilir. Yapılan ayrıştırma sonucu elde edilen S matrisi gürültüyü W ve H matrisi çarpımı ise gürültüden arındırılmış görüntüyü verecektir.

3.3.5 Ranstegele hale getirilmiş negatif olmayan matris ayrışımı

Rastgele hale getirilmiş negatif olmayan matris ayrışımı algortimaso olasılıksal yapıyı kullanmaktadır [7]. Olasılıksal yapıyı anlatmak için (M x N) boyutlarında bir X matrisinin olduğunu var sayalım. X matrisinin dizinini yakınsamak için (17) kullanılır.

$$Y := X\Omega \tag{17}$$

Burada Ω, K << N olacak şekilde, (N x K) boyutlarında rastgele oluşturulmuş test matrisidir. Sonrasında Y matrisinin QR-ayrıştırması (M x K) boyutlarındaki orthonormal Q matrisini oluşturmak için kullanırlır. Elde edilen orthonormal Q matrisi ile aşağıdaki eşitlik sağlanmış olur.

$$X \approx QQ^T X \tag{18}$$

Son olarak (K x N) boyutlarındaki B matrisi, A matrisinin düşük boyutlu düzlemdeki izdüşümü ile hesaplanır. Böylece A matrisi QB ayrışımı olarak da bilinen orthonormal Q ve düşük boyut izdüşüm B matrislerine ayrıştırılmış olur.

$$B := Q^T X \tag{19}$$

Hiyerarşik dönüşümlü en düşük kareler algoritması ile rastegele hale getirilmiş negatif olmayan matris ayrışımı için minimize edilmesi gereken fonksiyon aşağıdaki gibidir.

$$\min_{\widetilde{W},H} \left|\left|B - \widetilde{W}H\right|\right|_F^2 \quad ; Q\widetilde{W} \geq 0 \, , H \geq 0 \tag{20}$$

Burada $\widetilde{W}$, W matrisinin düşük boyutlu izdüşümüdür ve W matrisi $\widetilde{W}$ kullanılarak tekrardan elde edilebilir.

$$\widetilde{W} := Q^T W \tag{21}$$

$\widetilde{W}$ ve H matrislerinin j sütunu için her bir yinelemedeki güncelleme kuralı aşağıdaki gibidir.

$$H^+_{(j,:)} \leftarrow \left[H_{(j,:)} + \frac{\left[B^T\widetilde{W}\right]_{(:,j)} - H^T\left[\widetilde{W}^T\widetilde{W}\right]_{(:,j)}}{\left[\widetilde{W}^T\widetilde{W}\right]_{(j,j)}}\right] \tag{22a}$$

$$\widetilde{W}^+_{(j,:)} \leftarrow \left[\widetilde{W}_{(j,:)} + \frac{[BH^T]_{(:,j)} - \widetilde{W}[HH^T]_{(:,j)}}{[HH^T]_{(j,j)}}\right] \tag{22b}$$

Ardınından W matrisi $\widetilde{W}$ kullanılarak güncellenir ve $\widetilde{W}$ de tekrardan hesaplanır.

$$W^+_{(:,j)} \leftarrow \left[Q\widetilde{W}_{(:,j)}\right]_+ \tag{23}$$

$$\widetilde{W}^+_{(:,j)} \leftarrow Q^T W^+_{(:,j)} \tag{24}$$

Yineleme, hedef fonksiyonu belirlenen durdurma kriteri ϵ değerinden düşük olduğunda son bulur.

$$\left|\left|X - WH\right|\right|_F^2 < \epsilon \tag{25}$$

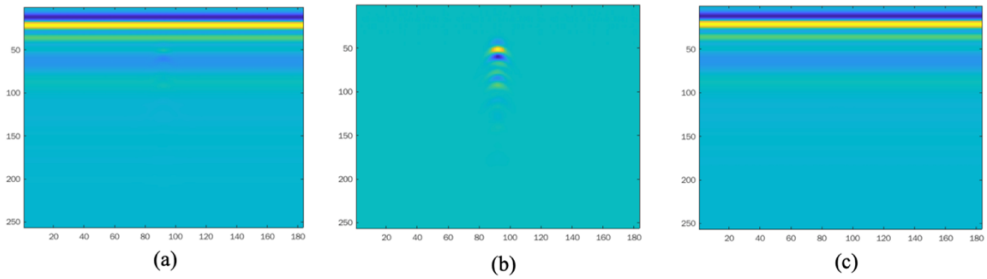
3.4 Kargaşa Giderme Uygulaması

Bu bölümde, 3.3'te anlatılan yöntemler ile gerçekleştirilen karagaşa giderme uygulaması yer almaktadır. Uygulamanın nasıl gerçekleştirildiği anlatılmış ve kullanılan algoritmalar açıklanmıştır. Uygulama sonunda elde edilen sonuçlar paylaşılmış ve uygulanan yöntemlerin sinyal gürültü oranı ve hesaplama süresi karşılaştırması yapılmıştır. Yapılan uygulama MATLAB Online ortamında kodlanmış ve gerçekleştirilmiştir.

3.4.1 Gürbüz negatif olmayan matris ayrışımı

İlk olarak gerçekleştirilen uygulama gürbüz negatif olmayan matris ayrışımı ile yene nüfuz eden radar görüntülerinde kargaşa gidermedir. Algoritma MATLAB yazılımı ile MATLAB yazılımına ait hazır matris operatörleri kullanılmadan yazılmıştır. Çünkü MATLAB ortamında yazılan kod daha sonrasında FPGA uygulamaları için C diline çevrilecektir. C dilinde hazır matris operatörleri olmadığı için algoritma kodlanırken döngüler ile yazılmış matris işlemleri kullanılmıştır.

Algoritma için seçilmesi gereken regülarizasyon parametresi (λ) ve toplam yineleme sayısı literatür araştırmasında edinilen bilgiler sonucunda λ = 0.0003 ve yineleme sayısı da 10000 olarak alınmıştır [8]. Uygulamada 256x183 boyutunda yere nüfuz eden radar görüntüsü kullanılmıştır. Uygulamalar MATLAB Online üzerinde yapılmıştır. Uygulama sonucu hedef cisim ve kargaşa görüntüsü başarılı bir şekilde ayrıştırılmıştır.



Şekil 3.1 : Gürbüz negatif olmayan matris ayrışımı kargaşa giderme sonuçları: (a) Orijinal yere nüfuz eden radar görüntüsü, (b) hedef görüntüsü, (c) kargaşa görüntüsü.

Algoritma 1: Gürbüz Negatif Olmayan Matris Ayrışımı

Girdi: N x M büyüklüğündeki görüntü matrisi (**X**), **W** ve **H** matrislerinin başlangıç değerleri (**W**^{0} ve **H**^{0}), regülarizasyon parametresi (λ), toplam yineleme sayısı (t).

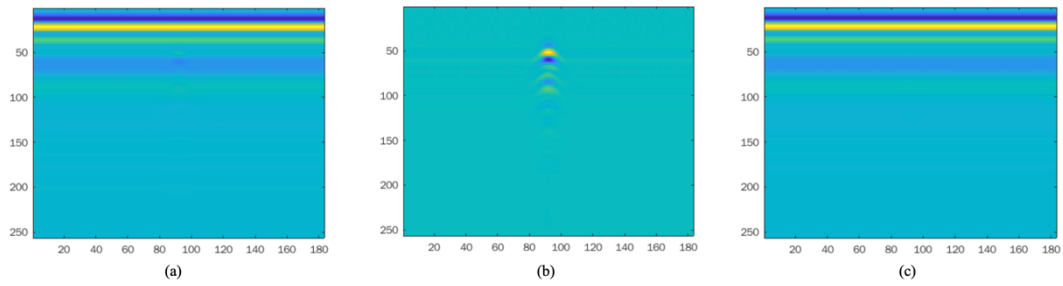
Çıktı: W, H ve **S** matrislerinin sonuç değerleri.

1. **W** ← **W**^{0}
2. **H** ← H^0
3. **for** i = 1:t **yap**
4. S ← X - WH
5. $$S_{ij} \leftarrow \begin{cases} S_{ij} - \frac{\lambda}{2}, eğer\ S_{ij} > \frac{\lambda}{2} \\ S_{ij} + \frac{\lambda}{2}, eğer\ S_{ij} < -\frac{\lambda}{2} \\ 0, diğer\ durumlarda \end{cases}$$
6. W ve H, (14.a) ve (14.b) ile güncellenir.
7. W ve H, (15.a) ve (15.b) ile güncellenir.
8. **döngünün sonu**
9. $X_{kargaşa}$ = WH
10. X_{hedef} = S

Şekil 3.2 : Gürbüz negatif olmayan matris ayrışımı sözde kodu

3.4.2 Rastgele hale getirilmiş negatif olmayan matris ayrışımı

İkinci olarak rastgele hale getirilmiş negatif olmayan matris ayrışımı algoritmasının uygulaması yapılmıştır. Algoritma için seçilmesi gereken durdurma kriteri (ϵ) ve toplam yineleme sayısı, ϵ = 0.0003 ve yineleme sayısı da 10000 olarak alınmıştır. Uygulamada 256x183 boyutunda yere nüfuz eden radar görüntüsü kullanılmıştır. Uygulamalar MATLAB Online üzerinde yapılmıştır. Uygulama sonucu hedef cisim ve kargaşa görüntüsü başarılı bir şekilde ayrıştırılmıştır.



Şekil 3.3 : Rastgele hale getirilmiş negatif olmayan matris ayrışımı kargaşa giderme sonuçları: (a) Orijinal yere nüfuz eden radar görüntüsü, (b) hedef görüntüsü, (c) kargaşa görüntüsü.

Algoritma 2: Rastgele Hale Getirilmiş Negatif Olmayan Matris Ayrışımı
Girdi: N x M büyüklüğündeki görüntü matrisi (**X**), **W** ve **H** matrislerinin başlangıç değerleri ($\mathbf{W^0}$ ve $\mathbf{H^0}$), rastgele oluşturulmuş Ω matrisi, durdurma kriteri (ϵ), toplam yineleme sayısı (t).
Çıktı: W, H matrislerinin sonuç değerleri. 1. $\mathbf{W} \leftarrow \mathbf{W^0}$ 2. $\mathbf{H} \leftarrow H^0$ 3. $Y = X\Omega$ 4. $[Q, R] = qr(Y)$ 5. $B = Q^T X$ 6. $\widetilde{W} = Q^T W$ 7. **for** i = 1:t **yap** 8. $\widetilde{W}$ ve H, (22.a) ve (22.b) ile güncellenir. 9. W, (23) ile güncellenir. 10. $\widetilde{W}$, (24) ile güncellenir. 11. **eğer** (25) sağlanırsa 12. **döngünün sonu** 13. **döngünün sonu** 14. $X_{kargaşa}$ = WH 15. X_{hedef} = A-WH

Şekil 3.4 : Rastgele hale getirilmiş negatif olmayan matris ayrışımı sözde algoritması

3.4.3 Öklid, Kullback-Leibler, ve Itakura-Saito uzaklığı

Bu bölümde temel negatif olmayan matris ayrışımı algoritması 3.3.3'te anlatıldığı gibi Öklid uzaklığı, Kullback-Leibler uzaklığı ve Itakura-Saito uzaklığı için farklı güncelleme kuralları kullanılarak uygulanmıştır. Yinelemeyi sonlandıran eşik değeri üç yöntem için de 0.0001 olarak seçilmiştir. Algoritma MATLAB dili ile yazıldı ve MATLAB Online ortamında çalıştırıldı. Üç farklı uzaklığa dayanan güncelleme kuralları ile de negatif olmayan matris ayrışımı başarı ile gerçekleştirildi ve hedef görüntüler elde edildi.

Şekil 3.5 : Negatif olmayan matris ayrışımı kargaşa giderme sonuçları: (a) Orijinal yere nüfuz eden radar görüntüsü; (b) Öklid uzaklığı, (c) Kullback-Leibler, (d) Itakura-Saito ile elde edilen hedef.

3.4.4 Kargaşa giderme sonuçlarının karşılaştırması

Bu bölümde uygulanan negatif olmayan matris ayrıştırma yöntemlerinin sonuçları incelendi ve karşılaştırıldı. Gürbüz negatif olmayan matris ayrışımı, rastgele hale getirilmiş negatif olmayan matris ayrışımı ve Kullback-Leibler uzaklığı kullanan negatif olmayan metris ayrışımı algoritmalarının sinyal gürültü oranı (PSNR) ve hesaplama süreleri karşılaştırıldı.

Çizelge 3.1 : Negatif olmayan matris ayrışımı algoritmalarının sinyal gürültü oranları

Algoritma	Sinyal Gürültü Oranı
Negatif Olmayan Matris Ayrışımı (Kullback-Leibler)	66.01
Gürbüz Negatif Olmayan Matris Ayrışımı	79.39
Rastgele Hale Getirilmiş Negatif Olmayan Matris Ayrışımı	66.60

Çizelge 3.2 : Negatif olmayan matris ayrışımı algoritmalarının hesaplama süreleri

Algoritma	Hesaplama Süresi (s)
Negatif Olmayan Matris Ayrışımı (Kullback-Leibler)	0.28
Gürbüz Negatif Olmayan Matris Ayrışımı	1.58
Rastgele Hale Getirilmiş Negatif Olmayan Matris Ayrışımı	0.57

Yapılan literatür araştırmasına göre gürbüz negatif olmayan matris ayrışımının diğer algoritmalara göre daha yüskek sinyal gürültü oranının daha yüksek olması

[8] ve rastgele hale getirilmiş negatif olmayan matris ayrışımının da diğer algoritmalara göre daha hızlı performans vermesi [7] beklenmekteydi. Çizelge 3.1'te görülmektedir ki en iyi sinyal gürültü oranını gürbüz negatif olmayan matris ayrışımı algoritması ile elde edilmiştir. Kullback-Leibler uzaklığı kullanan negatif olmayan matris ayrışımı ve rastgele hale getirilmiş negatif olmayan matris ayrışımı arasında ise sinyal gürültü oranı olarak ciddi bir fark görülmemektedir ve hesaplama süresi bakımından da rastgele hale getirilmiş negatif olmayan matris ayrışımının beklentinin aksine daha yavaş olduğu söylenebilir. Literatüre göre [7] rastgele hale getirilmiş negatif olmayan matris ayrışımı için veri büyüdükçe hesaplama avantajı sağladığı bilinmektedir. Bu çalışmada Kullback-Leibler uzaklığı kullanan negatif olmayan matris ayrışımının daha hızlı çıkmasının sebebi kargaşa giderme probleminde rank değerinin çok düşük olarak seçilmesi olabilir.

Uygulama sonucunda bütün algoritmalar ile ham görüntü hedef ve kargaşa olarak başarıyla ayrıştırılmış ve algoritmaların karşılaştırılmalarında elde edilen bulgular literatür taramasında elde edilenler ile büyük oranda uyumlu gerçekleşmiştir.

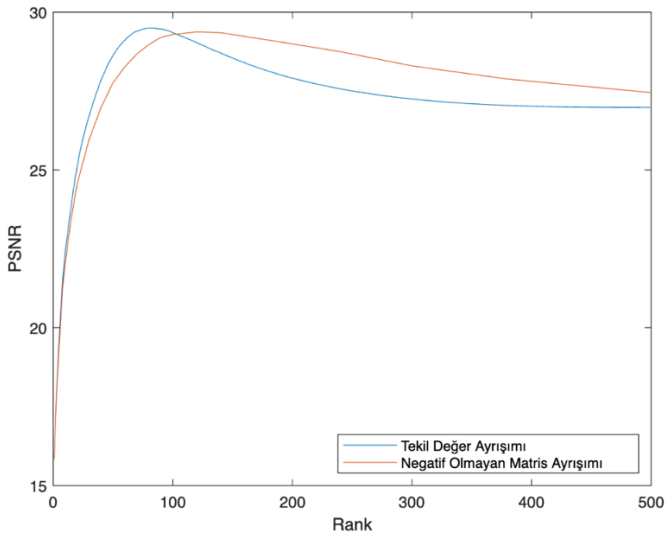
3.5 Gürültü Giderme Uygulamaları

Bu bölümde negatif olmayan matris ayrışımı ve tekil değer ayrışımı kullanılarak görüntü işleme çalışmalarında sıklıkla kullanılan görüntülerden birisi üzerinde gürültü giderme uygulamaları yapılacaktır. Sonrasında yere nüfuz eden radar görüntüsüne gürültü karıştığı durumda kargaşa giderme uygulamasının nasıl etkilendiği incelenip, yere nüfuz eden radar görüntüsüne eklenen gürültü giderilerek tekrar kargaşa giderme uygulaması yapılıp sonuçlar karşılaştırılacaktır.

3.5.1 Negatif olmayan matris ayrışımı ve tekil değer ayrışımı ile gürültü giderme uygulaması

İlk olarak görüntü işleme çalışmalarında kullanılan görsellerden biri olan 512x512 piksel boyutlarındaki "Lena" görseline gürültü eklendi. Gürültü eklemek için MATLAB içerisindeki "imnoise" fonksiyonundan yararlanıldı. Gürültü olarak ortalaması 0 ve varyansı 0.002 olan Gaussian gürültüsü kullanıldı. Negatif olmayan matris ayrışımı algoritmasının hangi rank değeri için en yüksek sinyal gürültü oranını verdiğini bulmak için farklı rank değerleri ile gürültü giderme işlemi yapıldı ve sinyal gürültü oranları karşılaştırıldı. Aynı şekilde tekil değer ayrışımı yöntemi için de ideal

sinyal gürültü oranını veren rank değeri bulundu. Farklı rank değerleri için elde edilen sinyal gürültü oranı değerleri karşılaştırması Şekil 3.6'da paylaşılmıştır. Alınan sonuçlara göre "Lena" görselinde gürültü giderme uygulaması için negatif olmayan matris ayrışımı rank değeri 120 iken en yüksek sinyal gürültü oranını verirken, tekil değer ayrışımı yöntemi rank değeri 85 iken en yüksek sinyal gürültü oranını vermektedir.



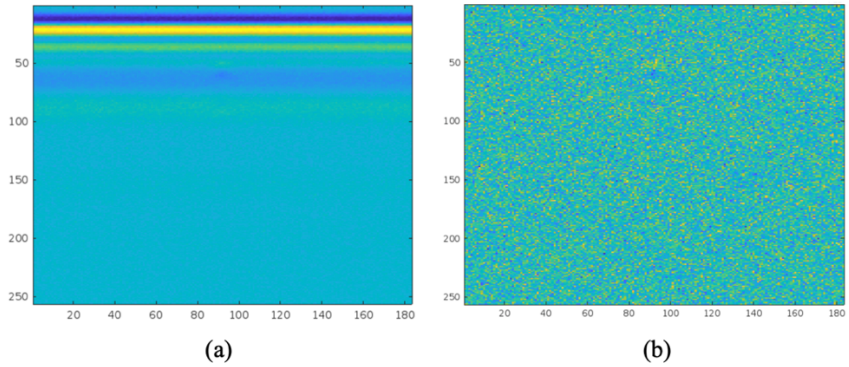
Şekil 3.6 : Lena görselinde gürültü giderme uygulaması için algoritmaların rank ve sinyal gürültü oranı karşılaştırması

Ardından elde edilen ideal rank değerleri hem negatif olmayan matris ayrışımı hem de tekil değer ayrışımı algoritması için kullanılarak gürültü eklenmiş "Lena" görseli üzerinde gürültü giderme uygulaması yapıldı. Elde edilen temizlenmiş görseller yeniden oluşturuldu. Negatif olmayan matris ayrışımı algortiması kullanarak 29.3 sinyal gürültü oranı elde edilirken tekil değer ayrışımı algoritması ile gerçekleştirilen gürültü giderme uygulaması sonucunda 29.58 ile biraz daha iyi bir sinyal gürültü oranı performansı gözlendi.

Şekil 3.7 : Lena gürültü giderme; (a) Ham görüntü, (b) gürültülü görüntü, (c) tekil değer ayrışımı, (d) negatif olmayan matris ayrışımı.

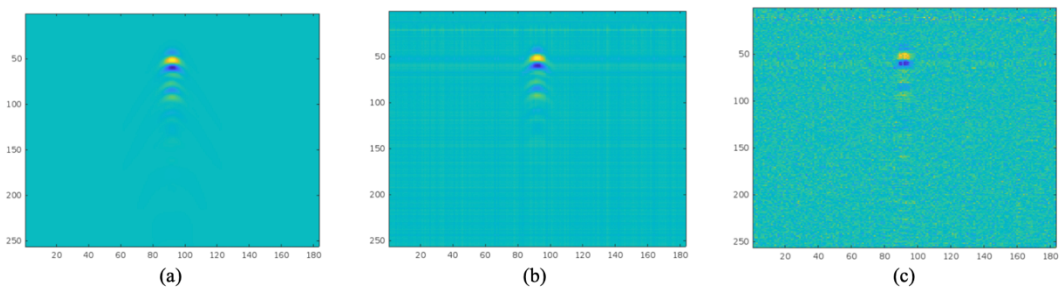
3.5.2 Yere nüfuz eden radar görüntüsünde gürültü giderme uygulaması

Yere nüfuz eden radar görüntüsünde kargaşa giderme işlemi yapmadan önce görüntünün gürültüden arındırılması önem arz etmektedir. Yere nüfuz eden radar görüntüsüne MATLAB "imnoise" fonksiyonu kullanılarak ortalaması 0 ve varyansı 0.0002 olan Gaussian gürültü eklenmiş ve gürültülü görselde kargaşa giderme yapılmaya çalışılmıştır. Şekil 3.8'de gürültü karışmış yere nüfuz eden radar görüntüsünde kargaşa giderme işlemi yapılması durumdan yaşanan problem gösterilmiştir, gürültülü görselde hedef görüntüsünü elde etmek mümkün olmamaktadır.



Şekil 3.8 : (a) Gürültülü yere nüfuz eden radar görüntüsü ve (b) gürültülü görüntüden elde edilen hedef görüntüsü.

Bu sebeple hem tekil değer ayrışımı hem de negatif olmayan matris ayrışımı ile gürültü giderme işleminden sonra kargaşa giderme uygulaması yapılmıştır. Yere nüfuz eden radar görüntülerinde gürültü giderme işlemi için iki algoritmanın da rank değeri görsel boyutlarından daha düşük ancak görselin boyutlarına yakın bir değer olarak alınmıştır. Bu uygulamada öncelikle büyük rank değeri ile gürültü giderilmiş daha sonra rank değeri 1 seçilerek kargaşa giderilmiştir. Ardından elde elilen hedef görüntüsü ile referans hedef görüntüsü kullanılarak sinyal gürültü oranı hesaplanmıştır.



Şekil 3.9 : Gürültü giderme sonrası kargaşa giderme; (a) referans hedef, (b) tekil değer ayrışımı, (c) negatif olmayan matris ayrışımı.

Gürültü giderme sonrası yapılan kargaşa giderme sonrası elde edilen hedeflerin referansa göre tepe sinyal gürültü oranı tekil değer ayrışımı için 64.46 ve negatif olmayan matris ayrışımı için 45.83 olarak gerçekleşmiştir. Yapılan uygulama sonucunda yere nüfuz eden radar görüntüsünü gürültüden arındırmanın kargaşa gidermek için ne kadar önemli olduğu anlaşılmıştır. Şekil 3.9'da elde edilen hedef görüntüleri Şekil 3.8'e göre çok daha belirgin ve tespit edilebilirdir. Elde edilen sonuçlarda tekil değer ayrışımının gürültü gidermede negatif olmayan matris ayrışıma göre daha avantajlı olduğu gözlemlenmektedi

4. FPGA ÜZERİNDE MATRİS AYRIŞIMI GERÇEKLEMELERİ

Bu bölümde FPGA donanımı üzerinde bilgisayar yazılımlarının gerçeklenmesi ile ilgili bir literatür taraması yer almaktadır. FPGA donanımının teknik özellikleri, yazılımın gerçeklenmesinde izlenecek yöntem, kullanılacak ara yazılımlar ve örnek uygulama anlatılacaktır.

4.1 FPGA Üzerinde Yazılım Gerçeklenmesi ile İlgili Literatür Araştırması

2013 yılında yapılan bir çalışmada; NASA'nın Havadan Görünür Kızılötesi Görüntüleme Spektrometresi (AVIRIS) tarafından toplanan veriler, Piksel Saflık İndisi (PPI) adı verilen ve her bir pikselin birim vektöre yansıtılması sonucu hangi pikselin kaç kez uç noktada kaldığını sayarak bir görüntüdeki ana unsur pikselleri seçen bir algoritma tarafından PowerPC işlemcili Virtex-4 model bir FPGA ve Nvidia CUDA model bir Grafik İşlemci Ünitesi üzerinde çalıştırılmıştır [20]. Sonuçlar göstermektedir ki; donanım uygulamalarının, dikkate alınan FPGA ve GPU mimarilerinde veri işleme kaynaklarını efektif şekilde kullanmaktadır. Ayrıca, yalnızca bir donanım cihazı kullanarak dahi düşük maliyetle önemli hızlanmalar sağladığını göstermiştir. Bu çalışmada kullanılan Arty Z7 model FPGA'nın sahip olduğu işlemci ailesi (Zynq), Virtex modellerinin sahip olduğu işemcilere; işlemci hızı, içerdiği mantık bloğu ve giriş-çıkış bloğu sayıları bakımından üstündür. 2020 yılında yapay bir veri seti ile yapılan bir başka çalışmada yine PPI ve benzer algoritmalar olan Uç Bileşen Analizi (VCA) ile N-FINDR algoritmaları donanım üzerinde gerçeklenmiştir [21]. İlgili çalışmada Virtex ailesi FPGA ve Nvidia Fermi Grafik İşlemci Ünitesi kullanılmış ve sonuçlar Hiperspektral çözümleme algoritmaların donanım üzerinde işlenmesi için, FPGA'ların ideal seçim olduğunu, Grafik İşlemci Üniteleri ve çok çekirdekli Merkezi İşlem Birimleri gibi diğer platformlarla karşılaştırıldığında daha az güç tükettiğini, bunun yanı sıra hafif oluşu sayesinde daha kompakt olduğunu göstermiştir. 2015 yılında yapılan bir çalışmada AVIRIS veri seti ve VCA algoritması Zynq ailesi işlemci kullanılan Artix-7 model FPGA üzerinde gerçeklenmiştir [22]. 2013 yılında Virtex-4 ile yapılan çalışmada [20] çalışmada bir verinin uç bileşenlerinin bulunması yaklaşık 5.2 milisaniye sürerken 2015 yılındaki ilgili çalışmada [22] bu süre 1.6 milisaniye kadardır.

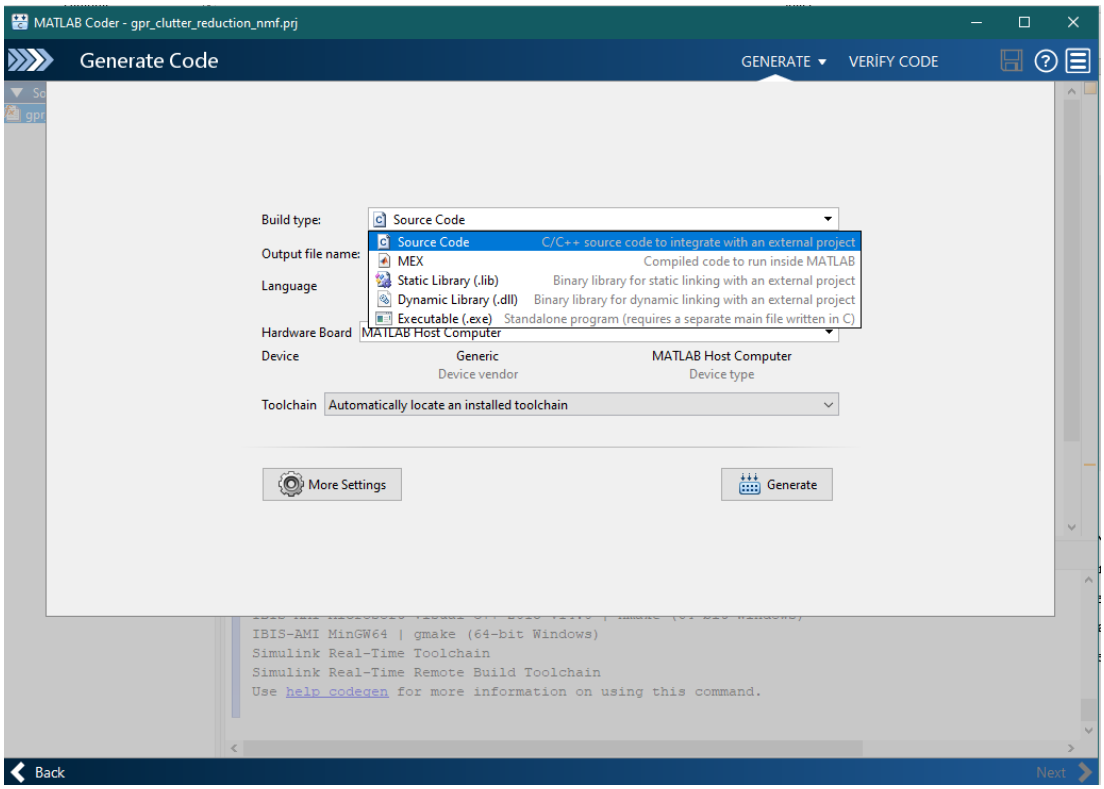
4.2 Yöntem

MATLAB kullanılarak yazılan Negatif Olmayan Matris Ayrıştırması algoritmasındaki fonksiyonlar, MATLAB bünyesinde bulunan ve MATLAB kodlarını C dilindeki kütüphanelere çeviren MATLAB Coder uygulaması kullanılarak donanım için uygun yazılım diline çevrilecektir. Belirtmek gerekir ki bu çevrim gerçekleşirken kodlar hem kütüphane hem de kendi kendine çalışabilen .exe uzantılı dosyalar olarak üretilebilmektedir.

```
function target = gpr_clutter_reduction_nmf(raw_data, d)
V = raw_data;
[row, col] = size(V);
th = 0.00001;
maxIter = 10000;
rdim = d;
W=rand(row,rdim);
H=rand(rdim,col);
iter=1;
D=zeros(maxIter,1);
D(1,1)=sum(sum((V.*log((V+eps)./(W*H+eps)))-V+(W*H))); % K-L divergence
thresh=D(1,1);
while abs(thresh)>th %% th is 0.0005
    iter=iter+1;
    W = W./(ones(row,1)*sum(W,1));
    H = H./(ones(rdim,1)*sum(H,1));
        W = W.*(((V./(W*H))*H')./(ones(row,col)*H'));
        H = H.*((W'*(V./(W*H)))./(W'*ones(row,col)));
    %% Metrics
        D(iter,1) = sum(sum((V.*log((V+eps)./(W*H+eps)))-V+(W*H))); % K-L divergence
    thresh=(D(iter,1)-D(iter-1,1))/D(iter-1,1);
    if iter==maxIter
        break;
    end
end
rec_im = W*H;
target = rec_im;
clutter_removed_data = raw_data - target;
```

Şekil 4.1: C Çevrimi Yapılan MATLAB Fonksiyonu

Şekil 4.1'de gösterilen Negatif Olmayan Matris Ayrışımı (NMF) algoritması gösterilmektedir. Bu fonksiyon rastgele *W* ve *H* üreildikten sonra MATLAB Coder kullanılarak tamamen C dilinde bir projeye çevrilecektir. Kodun vereceği sonuçlar çevrim esnasında MATLAB üzerinden ana fonksiyon gövdesi değiştirilmeksizin sınanacaktır. Daha sonra FPGA üzerinde ölçümler alınacak ve MATLAB çıktıları ile kıyaslanacaktır.

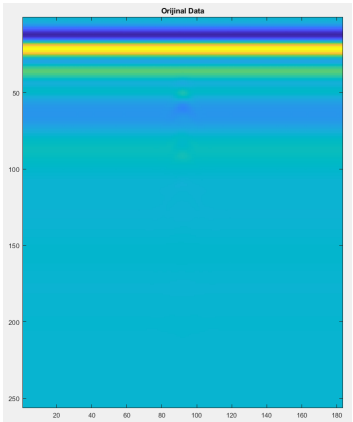


Şekil 4.2: MATLAB Coder Arayüzü

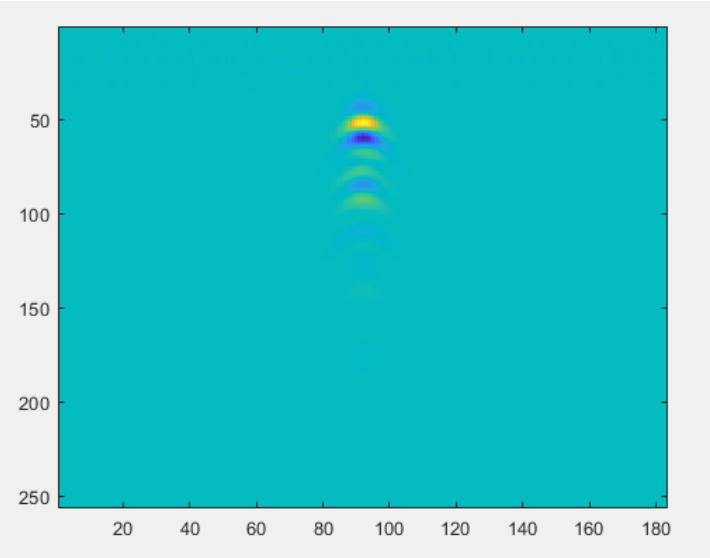
Projenin kaynak kodu şeklinde üretilmesine karar verildikten sonra çevrim işlemi tamamlandı.

4.3 FPGA Üzerinde Yazılım Gerçeklenmesi ile İlgili Uygulama

Şekil 4.3 elde edilen ham YNR görüntüsünü göstermektedir. Gerçeklemenin test edilmesi için ham veeri test görseli vektör formatında işlemci aracılığıyla DDR3 hafızasına kaydedilmiştir. Elde edilen çıktılar UART arayüzü ile sunucu bilgisayarda gözlemlenmiştir.



Şekil 4.3: Orijinal Data



Şekil 4.4: Donanım Gerçeklemesi Çıktısı

Bu uygulamada, GNOMA algoritmasının ARM Cortex A9 üzerinde işletim sistemi olmadan sonuç üretebilmesi 256x183 boyutunda bir ham veriden kargaşa ve hedef görüntü vektörlerini oluşturabilmesi- için 184.945.604.510 adet saat döngüsüne ihtiyaç duyulmaktadır. İşlemcinin içerisinde kullanılan saat frekansı ise 666 MHz olarak seçilmiştir. Bu nedenle GNOMA'nın 256x183 boyutunda bir ham veriden çıkış vektörlerini üretme süresi 277.7 saniye sürmüştür. Seri arayüzden veri göndermenin hızı uygulamaya göre değişeceğinden algoritma gerçekleme süresince ölçülen süre ve saat döngüsü sayısına seri arayüzden veri gönderme işlemleri dahil edilmemiştir. Uygulama için üretilen C kodu şu şekildedir:

```
/*
 * Academic License - for use in teaching, academic research, and meeting
 * course requirements at degree granting institutions only.  Not for
 * government, commercial, or other organizational use.
 * File: rnmf_in.c
 *
 * MATLAB Coder version            : 4.2
 * C/C++ source code generated on  : 04-Jan-2022 22:34:09
 */

/* Include Files */
#include <math.h>
#include <string.h>
#include "rnmf_in.h"
#include "sqrt.h"
#include "sum.h"
#include "sign.h"
#include "abs.h"


void rnmf_in(double target[46848], double clutter[46848])
{
  double W[256];
  static const double dv0[256] = { 0.80747046945450007, 0.26463438417187, ...
  double H[183];
  static const double dv1[183] = { 0.197970832005718, 0.624752290113142, ...
  int iter;
  int i0;
  int i1;
  static double varargin_2[46848];
  int target_tmp;
  double norms;
  static const double X[46848] = { 0.438652820170039, 0.432653295627544, ...
  static double z1[46848];
  double b;
  double MSXHt[256];
  double MWtSX[183];

  memcpy(&W[0], &dv0[0], sizeof(double) << 8);
  memcpy(&H[0], &dv1[0], 183U * sizeof(double));
  for (iter = 0; iter < 10000; iter++) {
    for (i0 = 0; i0 < 256; i0++) {
      for (i1 = 0; i1 < 183; i1++) {
        target_tmp = i0 + (i1 << 8);
        target[target_tmp] = X[target_tmp] - W[i0] * H[i1];
      }
    }

```

Şekil 4.5: NMF Algoritması C Kodu (1 –11929 satırları arası)

```
11930      b_abs(target, varargin_2);
11931      for (target_tmp = 0; target_tmp < 46848; target_tmp++) {
11932        norms = varargin_2[target_tmp] - 0.00015;
11933        varargin_2[target_tmp] -= 0.00015;
11934        z1[target_tmp] = fmax(0.0, norms);
11935      }
11936
11937      b_sign(target);
11938      for (i0 = 0; i0 < 46848; i0++) {
11939        target[i0] *= z1[i0];
11940      }
11941
11942      for (target_tmp = 0; target_tmp < 256; target_tmp++) {
11943        norms = 0.0;
11944        b = 0.0;
11945        for (i0 = 0; i0 < 183; i0++) {
11946          i1 = target_tmp + (i0 << 8);
11947          norms += target[i1] * H[i0];
11948          b += X[i1] * H[i0];
11949        }
11950
11951        norms -= b;
11952        MSXHt[target_tmp] = -norms;
11953        if (norms > 0.0) {
11954          MSXHt[target_tmp] = 0.0;
11955        }
11956      }
11957
11958      norms = 0.0;
11959      for (i0 = 0; i0 < 183; i0++) {
11960        norms += H[i0] * H[i0];
11961      }
11962
11963      for (target_tmp = 0; target_tmp < 256; target_tmp++) {
11964        W[target_tmp] = MSXHt[target_tmp] * W[target_tmp] / fmax(W[target_tmp] *
11965          norms, 1.0E-20);
11966      }
11967
11968      for (target_tmp = 0; target_tmp < 183; target_tmp++) {
11969        MWtSX[target_tmp] = 0.0;
11970        norms = 0.0;
11971        b = 0.0;
11972        for (i0 = 0; i0 < 256; i0++) {
11973          i1 = i0 + (target_tmp << 8);
11974          norms += W[i0] * target[i1];
11975          b += W[i0] * X[i1];
11976        }
11977
```

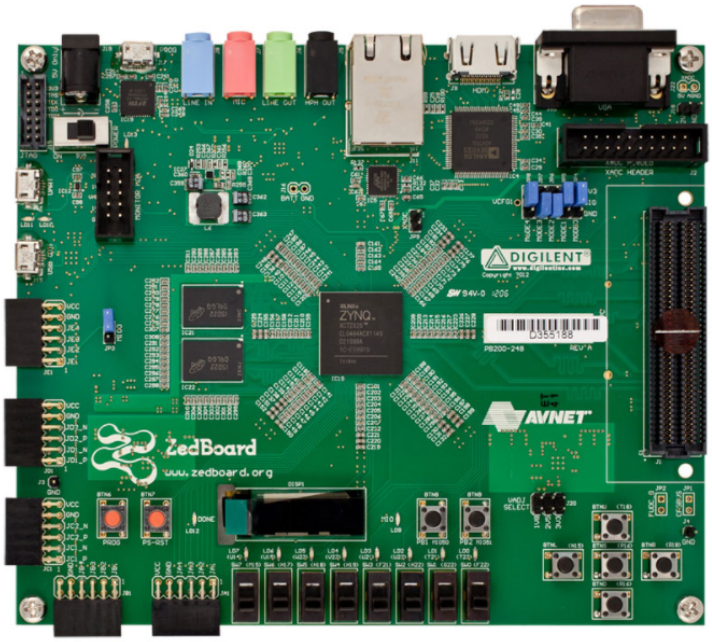
Şekil 4.6: NMF Algoritması C Kodu (11929 - 11977 satırları arası)

```
11978         norms -= b;
11979         MWtSX[target_tmp] = -norms;
11980         if (norms > 0.0) {
11981           MWtSX[target_tmp] = 0.0;
11982         }
11983       }
11984
11985       norms = 0.0;
11986       for (i0 = 0; i0 < 256; i0++) {
11987         norms += W[i0] * W[i0];
11988       }
11989
11990       for (target_tmp = 0; target_tmp < 183; target_tmp++) {
11991         H[target_tmp] = MWtSX[target_tmp] * H[target_tmp] / fmax(norms *
11992           H[target_tmp], 1.0E-20);
11993       }
11994
11995       for (i0 = 0; i0 < 256; i0++) {
11996         MSXHt[i0] = W[i0] * W[i0];
11997       }
11998
11999       norms = sum(MSXHt);
12000       b_sqrt(&norms);
12001       b = 1.0 / norms;
12002
12003       for (i0 = 0; i0 < 256; i0++) {
12004         W[i0] *= b;
12005       }
12006
12007       for (i0 = 0; i0 < 183; i0++) {
12008         H[i0] *= norms;
12009       }
12010     }
12011
12012     for (i0 = 0; i0 < 256; i0++) {
12013       for (i1 = 0; i1 < 183; i1++) {
12014         clutter[i0 + (i1 << 8)] = W[i0] * H[i1];
12015       }
12016     }
12017   }
12018
12019   /*
12020    * File trailer for rnmf_in.c
12021    *
12022    * [EOF]
12023    */
```

Şekil 4.7: NMF Algoritması C Kodu (11977 - 12023 satırları arası)

4.4 FPGA Üzerinde Gömülü Linux Gerçeklenmesi ile İlgili Uygulama

FPGA üzerinde radar yazılımını gerçekleyebilmek için öncelikle İTÜ Gömülü Sistem Laboratuvarı'ndan üzerinde Zynq – 7000 işlemci bulunan Zedboard geliştirme kartı temin edildi. Temin edilen kartın Evrensel Asenkron Alıcı Verici (UART) çıkışından veri okuyabilmek için sunucu bilgisayar Cypress USB-UART sürücü yazılımı ve sürücüye erişim sağlamak adına TeraTerm sanal terminal yazılımı kurumuştur. [23].



Şekil 4.8: Zedboard Geliştirme Kartı

Temin edilen geliştirme kartının bilgisayar ortamı ile veri alışverişinde bulunabilmesi için, UART karakteristiğine sahip iki adet microUSB-USB kablosu, programlama için J17 koldu "PROG" ve seri haberleşme için J14 kodlu "UART" girişleri üzerinden bilgisayara bağlandı. 12V adaptör ile geliştirme kartına besleme sağlandı. Geliştirme kartının harici hafızasında bulunan gömülü Linux'a erişim sağlayabilmek için JP9 ve JP10 kodlu ara bağlantı kabloları beslemeye; JP7, JP8 ve JP11 ara bağlantı kabloları ise toprağa bağlandı [24].

Şekil 4.9: Zedboard SDCard Erişim Modu

Temin edilen kartın bağlantılarının doğruluğu kontrol etmek adına terminal üzerinden kartın üzerindeki LED'lere komut gönderildi.

Şekil 4.10: Linux'un başlatıldığına dair terminal mesajı

Şekil 4.11: Kart donanımlarının başlatıldığına dair terminal mesajı

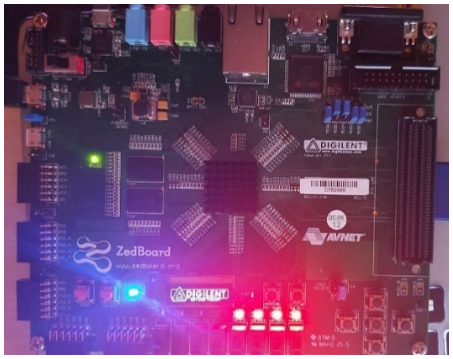
```
++ Configure static IP 192.168.1.10
[    1.510000] GEM: lp->tx_bd ffdfb000 lp->tx_bd_dma 18fd3000 lp->tx_skb d807028
0
[    1.510000] GEM: lp->rx_bd ffdfc000 lp->rx_bd_dma 18fd4000 lp->rx_skb d807038
0
[    1.520000] GEM: MAC 0x00350a00, 0x00002201, 00:0a:35:00:01:22
[    1.520000] GEM: phydev d8b6f400, phydev->phy_id 0x1410dd1, phydev->addr 0x0
[    1.530000] eth0, phy_addr 0x0, phy_id 0x01410dd1
[    1.530000] eth0, attach [Marvell 88E1510] phy driver
++ Starting telnet daemon
++ Starting http daemon
++ Starting ftp daemon
++ Starting dropbear (ssh) daemon
++ Starting OLED Display
[    1.570000] pmodoled-gpio-spi [zed_oled] SPI Probing
++ Exporting LEDs & SWs
rcS Complete
zynq>
```

Şekil 4.12: Zedboard IP Konfigürasyonu

Kartın üzerinde bulunan sekiz adet anahtarın, SW0 anahtarından başlayarak SW7 anahtarına kadar, “read_sw” komutuyla adreslerine erişildi. Daha sonra ilk 4 LED’in adresi alındı ve “write_led” komutu ile lojik-1 verisi gönderildi [25].

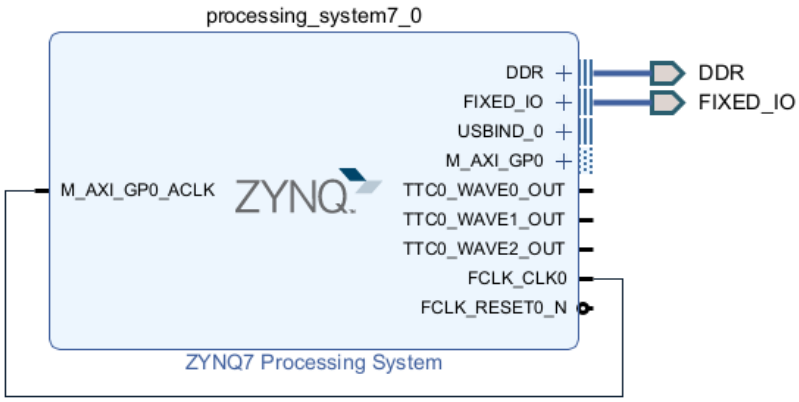
```
zynq> read_sw
0x0 0
zynq> read_sw
0x1 1
zynq> read_sw
0x2 2
zynq> read_sw
0x4 4
zynq> read_sw
0x8 8
zynq> read_sw
0x10 16
zynq> read_sw
0x20 32
zynq> read_sw
0x40 64
zynq> read_sw
0x80 128
zynq> read_sw
0xf 15
zynq> write_led 0xf0
```

Şekil 4.13: Anahtar Adresleri



Şekil 4.14: Komut Çıktısı

Yapılan işlemler neticesinde bağlantıların doğru olduğuna ve geliştirme kartının doğru şekilde çalıştığına karar verildi. Daha sonra Vivado yazılımı kullanılarak Zynq-7000 işlemcisine erişim sağlamak ve işlemci üzerinde radar yazılımını koşturmak için gereken blok tasarımı oluşturuldu. Oluşturulan blok tasarımdan Donanım Tasarım Dili (HDL) kodu üretildi.



Şekil 4.15: Zedboard Zynq-7000 İşlemci Blok Tasarımı

```
module zynq_wrapper
   (DDR_addr,
    DDR_ba,
    DDR_cas_n,
    DDR_ck_n,
    DDR_ck_p,
    DDR_cke,
    DDR_cs_n,
    DDR_dm,
    DDR_dq,
    DDR_dqs_n,
    DDR_dqs_p,
    DDR_odt,
    DDR_ras_n,
    DDR_reset_n,
    DDR_we_n,
    FIXED_IO_ddr_vrn,
    FIXED_IO_ddr_vrp,
    FIXED_IO_mio,
    FIXED_IO_ps_clk,
    FIXED_IO_ps_porb,
    FIXED_IO_ps_srstb);
```

Şekil 4.15: Zynq-700 İşlemci RAM ve Giriş-Çıkış Tanımlamaları

```
inout [14:0]DDR_addr;
inout [2:0]DDR_ba;
inout DDR_cas_n;
inout DDR_ck_n;
inout DDR_ck_p;
inout DDR_cke;
inout DDR_cs_n;
inout [3:0]DDR_dm;
inout [31:0]DDR_dq;
inout [3:0]DDR_dqs_n;
inout [3:0]DDR_dqs_p;
inout DDR_odt;
inout DDR_ras_n;
inout DDR_reset_n;
inout DDR_we_n;
inout FIXED_IO_ddr_vrn;
inout FIXED_IO_ddr_vrp;
inout [53:0]FIXED_IO_mio;
inout FIXED_IO_ps_clk;
inout FIXED_IO_ps_porb;
inout FIXED_IO_ps_srstb;
```

Şekil 4.16: Zynq-7000 İşlemci Veri Bitleri

```
wire [14:0]DDR_addr;
wire [2:0]DDR_ba;
wire DDR_cas_n;
wire DDR_ck_n;
wire DDR_ck_p;
wire DDR_cke;
wire DDR_cs_n;
wire [3:0]DDR_dm;
wire [31:0]DDR_dq;
wire [3:0]DDR_dqs_n;
wire [3:0]DDR_dqs_p;
wire DDR_odt;
wire DDR_ras_n;
wire DDR_reset_n;
wire DDR_we_n;
wire FIXED_IO_ddr_vrn;
wire FIXED_IO_ddr_vrp;
wire [53:0]FIXED_IO_mio;
wire FIXED_IO_ps_clk;
wire FIXED_IO_ps_porb;
wire FIXED_IO_ps_srstb;
```

Şekil 4.17: Zynq-7000 Modül Bağlantıları

```
design_1 design_1_i
     (.DDR_addr(DDR_addr),
      .DDR_ba(DDR_ba),
      .DDR_cas_n(DDR_cas_n),
      .DDR_ck_n(DDR_ck_n),
      .DDR_ck_p(DDR_ck_p),
      .DDR_cke(DDR_cke),
      .DDR_cs_n(DDR_cs_n),
      .DDR_dm(DDR_dm),
      .DDR_dq(DDR_dq),
      .DDR_dqs_n(DDR_dqs_n),
      .DDR_dqs_p(DDR_dqs_p),
      .DDR_odt(DDR_odt),
      .DDR_ras_n(DDR_ras_n),
      .DDR_reset_n(DDR_reset_n),
      .DDR_we_n(DDR_we_n),
      .FIXED_IO_ddr_vrn(FIXED_IO_ddr_vrn),
      .FIXED_IO_ddr_vrp(FIXED_IO_ddr_vrp),
      .FIXED_IO_mio(FIXED_IO_mio),
      .FIXED_IO_ps_clk(FIXED_IO_ps_clk),
      .FIXED_IO_ps_porb(FIXED_IO_ps_porb),
      .FIXED_IO_ps_srstb(FIXED_IO_ps_srstb));
endmodule
```

Şekil 4.18: Zynq-7000 Modül Atamaları

HDL kodunun sentez ve yerleştirme (implementation) aşamaları koşturulduktan sonra .xsa uzantılı bit serisi (bitstream) oluşturuldu ve Zynq-7000 işlemcisi üzerinde çalışma yapabilmek için Vitis yazılımına aktarıldı. Oluşturulan bit serisi kullanılarak işletim sistemi barındırmayan bir platform oluşturuldu. Geliştirme kartı üzerinde, oluşturulan platform kullanılarak Vitis yazılımında hazır bulunan basit bir "Hello World" uygulaması koşturuldu.

```c
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"


int main()
{
    init_platform();

    print("Hello World\n\r");
    print("Successfully ran Hello World application");
    cleanup_platform();
    return 0;
}
```

Şekil 4.19: Vitis Örnek Uygulama Kodu

Şekil 4.20: Program Çıktısı

Daha sonra geliştirme kartı üzerinde bulunan Zynq-7000 işlemcisi ile RAM arasında veri alışverişi sağlanabildiğini görmek adına 5 elemanlı iki adet sayı dizisinin elemanlarını toplayan bir program Vitis yardımıyla C dili kullanılarak oluşturulup geliştirme kartına yüklendi. Böylece, radardan gelen verinin RAM üzerine yazılarak işlemci tarafından okunabileceği teyit edilmiş oldu.

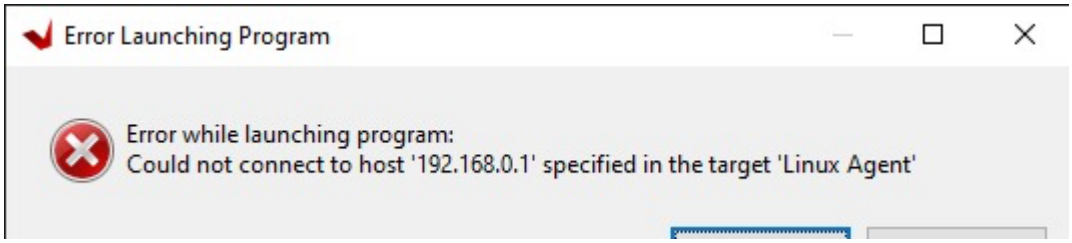
```c
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"


int main()
{
    init_platform();

    int demoMatrix1[5] = {0, 1, 2, 3, 4};
     int demoMatrix2[5] = {5, 6, 7, 8, 9};
     int sumMatrix[5];
     int i;

     for(i == 0; i < 5; i = i + 1)
     {
        sumMatrix[i] = demoMatrix1[i] + demoMatrix2[i];
     }

     print("Result: %d", &sumMatrix)
    cleanup_platform();
    return 0;
}
```

Şekil 4.21: Dizi Toplama Programı

```
DRAM:  512 MiB
MMC:   SDHCI: 0
Using default environment

In:    serial
Out:   serial
Err:   serial
Net:   zynq_gem
Hit any key to stop autoboot:  0
Result: 5 7 9 11 13
```

Şekil 4.22: Program Çıktısı

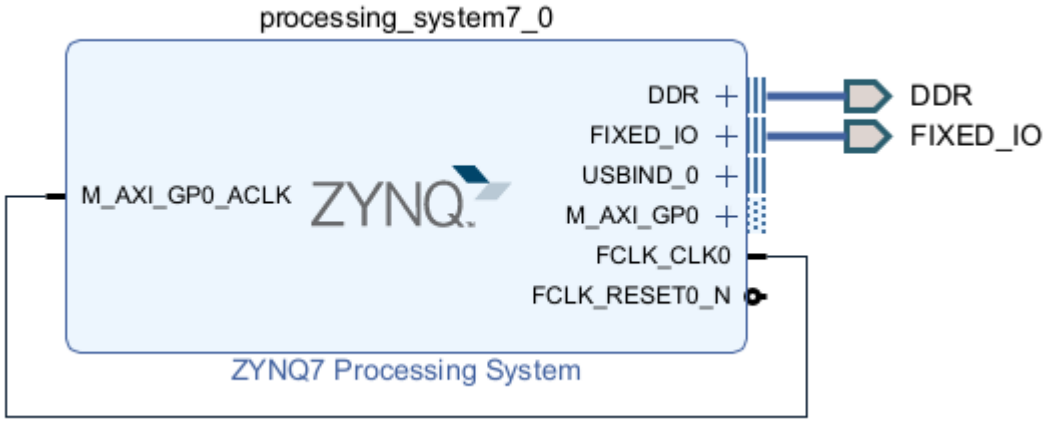
Daha sonra radarın kaynak kodlarının çalışması için Linux işletim sistemi gerektiğinden, geliştirme kartının harici hafıza birimi içerisindeki Linux sistemini kullanan bir platform projesi ve içerisindeki Gömülü Linux sistemini kullanarak MRM radarı için gereken arpa/inet.h ve sys/socket.h kütüphanelerine erişebilen ve radarın kaynak kodlarını derleyen bir uygulama proje oluşturuldu. Programın derlenmesi esnasında geliştirme kartının sunucu bilgisayar ile IP adresi üzerinden iletişim kuramayışından kaynaklanan bir hata ile karşılaşıldı.



Şekil 4.23: Hata Mesajı

Hatanın çözümü için, geliştirme kartı ile sunucu bilgisayar arasında ethernet bağlantısı kuruldu ancak hata giderilemedi. Bunun yanında, sunucu bilgisayara ve geliştirme kartına statik IP'ler verildiği halde sorun devam etti. Kablosuz bağlantı sağlanması adına geliştirme kartı ile modem arasında ethernet bağlantısı sağlandı ve statik IP atamaları denendi. Bahsedilen yöntemler hatanın çözülmesini sağlamadığından MRM yazılımını gerçeklemek için başka bir geliştirme kartı talep edildi.

Zynq-7000 işlemcili Arty-Z7 kartı bir adet UART karakteristiğine sahip microUSB-USB kablosu kullanılarak sunucu bilgisayara bağlandı. Vitis yazılımı kullanılarak geliştirme kartı üzerindeki Zynq-7000 işlemci üzerinde çalışabilmek için gereken blok tasarım oluşturuldu.



Şekil 4.24: Arty-Z7 Zynq-7000 İşlemci Blok tasarımı

Oluşturulan blok tasarımı ile üretilen HDL kodundaki giriş-çıkış bağlantıları, veri bitleri, modül tanımlamaları ve atamaları Şekil 4.12, Şekil 4.13, Şekil 4.14, ve Şekil 4.15'deki gibidir. Bağlantıların doğruluğunu teyit etmek adına Şekil 4.18'deki uygulama kodu kullanıldı.

Şekil 4.25: Program Çıktısı

Arty-Z7 geliştirme kartında kendi mini-SD Card olmadığından; Linux görüntüsü, geliştirme kartına yüklenemedi. Sunucu bilgisayar Windows işletim sistemi ile çalıştığından, sanal makina kullanılarak yapay bir Linux işletim sistemi oluşturuldu. Donanımlar için Gömülü Linux yükleme arayüzü olan Petalinux yazılımı kuruldu. Petalinux kök dizinine elle erişim sağlamaya çalışıldı [26].

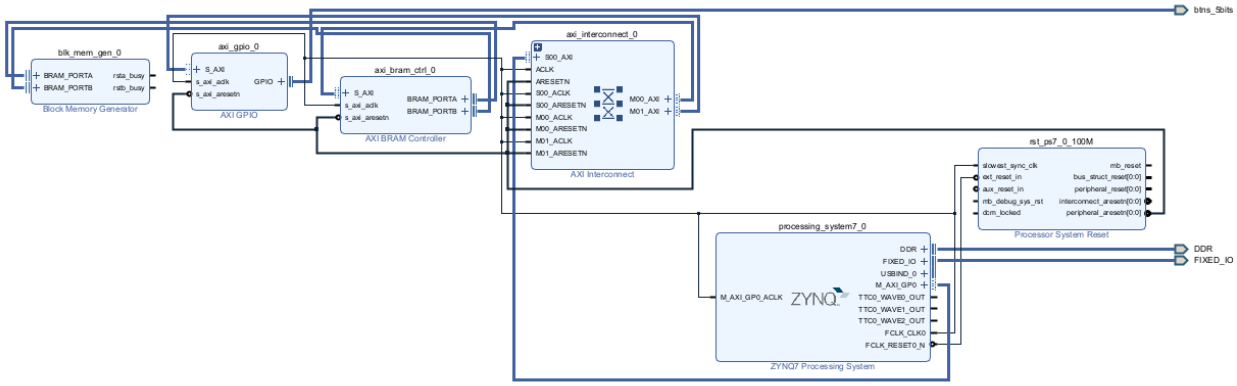
Bu girişimler sonuç vermeyince, MRM yazılımının bir sistem görüntüsü Zedboard geliştirme kartının harici hafıza birimine yüklendi. Ancak, MRM yazılımının sistem görüntüsü, geliştirme kartının kendi sistem görüntüsünün üzerine yazıldığından, Gömülü Linux sistemine, dolayısıyla Linux kütüphanelerine ulaşılamadı. Gerekli Linux kütüphaneleri elle bir araya getirilerek proje dosyasına eklendi [27].

Ancak, kütüphanelerin farklı kaynaklardan bulunarak bir araya getirilmesinden kaynaklı uyumsuzluklar yazılımın doğru çalışmamasına sebep oldu [28].

Daha sonra sorunun Windows işletim sisteminin Linux kütüphaneleriyle uyumsuzluk yaratmasından kaynaklandığı düşünüldü. Linux işletim sistemi kullanan bir sunucu bilgisayar üzerinden, Vitis kütüphanesinde bulunan “Linux Hello World” uygulaması derlendi [29].

4.5 RaspberryPi – Zedboard İşbirliğiyle Radar Görüntüsünün İşlenmesi

Zedboard üzerinde Gömülü Linux gerçeklemesi girişimi sonuçsuz kalınca, Linux çalıştırabilen bir RaspberryPi donanımı ana donanım olarak kullanılması ve uydu Zedboard kartının RaspberryPi üzerinde koşan işletim sistemiyle veriyi işleyerek ana donanıma geri iletmesiyle radar görüntüsünün işlenmesi düşünüldü. Zedboard için blok tasarımı aşağıdaki şekilde değiştirildi:



Şekil 4.26: Zedboard Zynq-7000 İşlemci Güncellenmiş Blok Tasarımı

Yeni blok tasarım sayesinde Zedboard veri alışverişi yapabilir ve veri saklayabilir hale geldi.[30]

4.5.1 Zedboard üzerinde NMF algoritmasının gerçeklenmesi

Bölüm 4.3'de bahsedilen NMF algoritması giriş olarak uygulanan matrislerle farklı rank değerlerinde işlem yapabilmek için yine aynı bölümde bahsedilen C kodu güncellendi. Bu giriş matrisleri ve seçilecek rank değeri kullanıcı tanımlı olarak ayarlandı. C kodu aşağıdaki görsellerdeki gibidir:

```
 2   * Academic License - for use in teaching, academic research, and meeting
 3   * course requirements at degree granting institutions only.  Not for
 4   * government, commercial, or other organizational use.
 5   * File: gpr_clutter_reduction_nmf.c
 6   *
 7   * MATLAB Coder version            : 5.2
 8   * C/C++ source code generated on  : 08-Jun-2022 13:16:34
 9   */
10
11  /* Include Files */
12  #include "gpr_clutter_reduction_nmf.h"
13  #include "combineVectorElements.h"
14  #include "gpr_clutter_reduction_nmf_data.h"
15  #include "gpr_clutter_reduction_nmf_emxutil.h"
16  #include "gpr_clutter_reduction_nmf_initialize.h"
17  #include "gpr_clutter_reduction_nmf_types.h"
18  #include "mtimes.h"
19  #include "rand.h"
20  #include "sum.h"
21  #include <math.h>
22  #include <string.h>
23
24  void gpr_clutter_reduction_nmf(const emxArray_real_T *raw_data, double d,
25                                 emxArray_real_T *target)
26  {
27    emxArray_real_T *H;
28    emxArray_real_T *W;
29    emxArray_real_T *b_raw_data;
30    emxArray_real_T *r;
31    emxArray_real_T *x;
32    emxArray_real_T *x_tmp;
33    emxArray_real_T *y_tmp;
34    double D[10000];
35    double iter;
36    double thresh;
37    int col;
38    int i;
39    int i1;
40    int k;
41    int nx;
42    int row;
43    boolean_T exitg1;
44    if (!isInitialized_gpr_clutter_reduction_nmf) {
45      gpr_clutter_reduction_nmf_initialize();
46    }
47    emxInit_real_T(&W, 2);
48    emxInit_real_T(&H, 2);
```

Şekil 4.27: NMF Algoritması C Kodu (2 – 48 satırları arası)

```
50    col = raw_data->size[1];
51    row = raw_data->size[0];
52    b_rand(raw_data->size[0], d, W);
53    b_rand(d, raw_data->size[1], H);
54    iter = 1.0;
55    memset(&D[0], 0, 10000U * sizeof(double));
56    emxInit_real_T(&y_tmp, 2);
57    emxInit_real_T(&x_tmp, 2);
58    mtimes(W, H, y_tmp);
59    i = x_tmp->size[0] * x_tmp->size[1];
60    x_tmp->size[0] = raw_data->size[0];
61    x_tmp->size[1] = raw_data->size[1];
62    emxEnsureCapacity_real_T(x_tmp, i);
63    nx = raw_data->size[0] * raw_data->size[1];
64    for (i = 0; i < nx; i++) {
65      x_tmp->data[i] = raw_data->data[i] + 2.2204460492503131E-16;
66    }
67    emxInit_real_T(&x, 2);
68    i = x->size[0] * x->size[1];
69    x->size[0] = x_tmp->size[0];
70    x->size[1] = x_tmp->size[1];
71    emxEnsureCapacity_real_T(x, i);
72    nx = x_tmp->size[0] * x_tmp->size[1];
73    for (i = 0; i < nx; i++) {
74      x->data[i] = x_tmp->data[i] / (y_tmp->data[i] + 2.2204460492503131E-16);
75    }
76    nx = x->size[0] * x->size[1];
77    for (k = 0; k < nx; k++) {
78      x->data[k] = log(x->data[k]);
79    }
80    emxInit_real_T(&b_raw_data, 2);
81    i = b_raw_data->size[0] * b_raw_data->size[1];
82    b_raw_data->size[0] = raw_data->size[0];
83    b_raw_data->size[1] = raw_data->size[1];
84    emxEnsureCapacity_real_T(b_raw_data, i);
85    nx = raw_data->size[0] * raw_data->size[1];
86    for (i = 0; i < nx; i++) {
87      b_raw_data->data[i] =
88          (raw_data->data[i] * x->data[i] - raw_data->data[i]) + y_tmp->data[i];
89    }
90    emxInit_real_T(&r, 2);
91    combineVectorElements(b_raw_data, r);
92    D[0] = sum(r);
93    thresh = D[0];
94    exitg1 = false;
95    while ((!exitg1) && (fabs(thresh) > 1.0E-5)) {
96      iter++;
97      combineVectorElements(W, r);
```

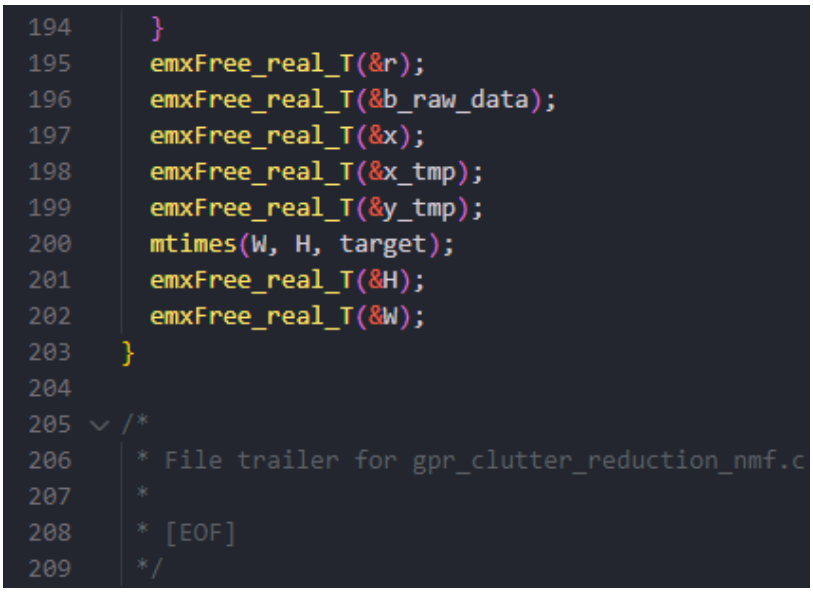
Şekil 4.28: NMF Algoritması C Kodu (49 – 97 satırları arası)

```
 98      i = W->size[0] * W->size[1];
 99      W->size[0] = row;
100      W->size[1] = r->size[1];
101      emxEnsureCapacity_real_T(W, i);
102      nx = r->size[1];
103      for (i = 0; i < nx; i++) {
104        for (i1 = 0; i1 < row; i1++) {
105          W->data[i1 + W->size[0] * i] /= r->data[i];
106        }
107      }
108      combineVectorElements(H, r);
109      k = (int)d;
110      i = H->size[0] * H->size[1];
111      H->size[0] = (int)d;
112      H->size[1] = r->size[1];
113      emxEnsureCapacity_real_T(H, i);
114      nx = r->size[1];
115      for (i = 0; i < nx; i++) {
116        for (i1 = 0; i1 < k; i1++) {
117          H->data[i1 + H->size[0] * i] /= r->data[i];
118        }
119      }
120      mtimes(W, H, y_tmp);
121      i = b_raw_data->size[0] * b_raw_data->size[1];
122      b_raw_data->size[0] = raw_data->size[0];
123      b_raw_data->size[1] = raw_data->size[1];
124      emxEnsureCapacity_real_T(b_raw_data, i);
125      nx = raw_data->size[0] * raw_data->size[1];
126      for (i = 0; i < nx; i++) {
127        b_raw_data->data[i] = raw_data->data[i] / y_tmp->data[i];
128      }
129      b_mtimes(b_raw_data, H, y_tmp);
130      i = b_raw_data->size[0] * b_raw_data->size[1];
131      b_raw_data->size[0] = row;
132      b_raw_data->size[1] = col;
133      emxEnsureCapacity_real_T(b_raw_data, i);
134      k = row * col;
135      for (i = 0; i < k; i++) {
136        b_raw_data->data[i] = 1.0;
137      }
138      b_mtimes(b_raw_data, H, x);
139      nx = W->size[0] * W->size[1];
140      for (i = 0; i < nx; i++) {
141        W->data[i] *= y_tmp->data[i] / x->data[i];
142      }
143      mtimes(W, H, y_tmp);
144      i = b_raw_data->size[0] * b_raw_data->size[1];
145      b_raw_data->size[0] = raw_data->size[0];
146      b_raw_data->size[1] = raw_data->size[1];
```

Şekil 4.29: NMF Algoritması C Kodu (98 – 146 satırları arası)

```
147     emxEnsureCapacity_real_T(b_raw_data, i);
148     nx = raw_data->size[0] * raw_data->size[1];
149     for (i = 0; i < nx; i++) {
150       b_raw_data->data[i] = raw_data->data[i] / y_tmp->data[i];
151     }
152     c_mtimes(W, b_raw_data, y_tmp);
153     i = b_raw_data->size[0] * b_raw_data->size[1];
154     b_raw_data->size[0] = row;
155     b_raw_data->size[1] = col;
156     emxEnsureCapacity_real_T(b_raw_data, i);
157     for (i = 0; i < k; i++) {
158       b_raw_data->data[i] = 1.0;
159     }
160     c_mtimes(W, b_raw_data, x);
161     nx = H->size[0] * H->size[1];
162     for (i = 0; i < nx; i++) {
163       H->data[i] *= y_tmp->data[i] / x->data[i];
164     }
165     mtimes(W, H, y_tmp);
166     i = x->size[0] * x->size[1];
167     x->size[0] = x_tmp->size[0];
168     x->size[1] = x_tmp->size[1];
169     emxEnsureCapacity_real_T(x, i);
170     nx = x_tmp->size[0] * x_tmp->size[1];
171     for (i = 0; i < nx; i++) {
172       x->data[i] = x_tmp->data[i] / (y_tmp->data[i] + 2.2204460492503131E-16);
173     }
174     nx = x->size[0] * x->size[1];
175     for (k = 0; k < nx; k++) {
176       x->data[k] = log(x->data[k]);
177     }
178     i = b_raw_data->size[0] * b_raw_data->size[1];
179     b_raw_data->size[0] = raw_data->size[0];
180     b_raw_data->size[1] = raw_data->size[1];
181     emxEnsureCapacity_real_T(b_raw_data, i);
182     nx = raw_data->size[0] * raw_data->size[1];
183     for (i = 0; i < nx; i++) {
184       b_raw_data->data[i] =
185           (raw_data->data[i] * x->data[i] - raw_data->data[i]) + y_tmp->data[i];
186     }
187     combineVectorElements(b_raw_data, r);
188     D[(int)iter - 1] = sum(r);
189     thresh = D[(int)(iter - 1.0) - 1];
190     thresh = (D[(int)iter - 1] - thresh) / thresh;
191     if (iter == 10000.0) {
192       exitg1 = true;
193     }
194   }
```

Şekil 4.30: NMF Algoritması C Kodu (147 – 194 satırları arası)



Şekil 4.31: NMF Algoritması C Kodu (194 – 209 satırları arası)

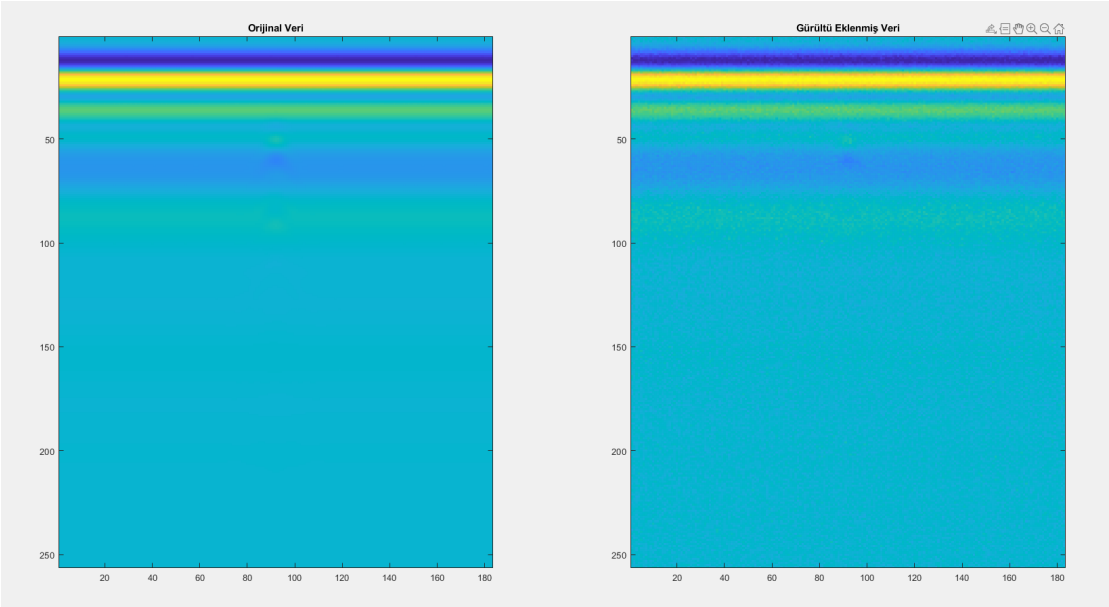
GNOMA algoritması içerisinde rastgele *W* ve *H* vektörlerine ihtiyaç duyulmaktadır. Ancak kullanıcı tanımlı olabilmesi açısından *W* ve *H* vektörleri test için kullanılacak MATLAB ana fonksiyonuyla oluşturuldu. Bu vektörler DDR3 hafıza elemanı içerisine yazılarak kullanıldı. Bu yapılan işlem dinamik hafıza ataması gerektirdiğinden çalışma süresi artmıştır ancak uygulama gürültü giderme yapabilir hale getirildi. Vektörlerin MATLAB'da oluşturulup daha sonra ARM üzerinde kullanılmasının sonuç performansına etki etmediği de gözlemlendi.

MATLAB Coder uygulaması oluşturulan C kodunu MEX (Matlab Executable File) dosyası olarak kullanmaktadır.[31]

Böylelikle MATLAB programlama dilinden çevrilen fonksiyon MATLAB dahili kütüphanelerinden biri gibi işlenerek test için kullanılabilir. Eldeki görüntüye, üzerinde gürültü giderme ve kargaşa giderme kodları test edilmeden önce, 0 ortalamalı ve 0.0002 varyanslı, Gauss dağılımına sahip gürültü eklendi. Oluşturulan MEX kütüphanesini denemek için kullanılan test programı ve çıktıları aşağıda verilmiştir:

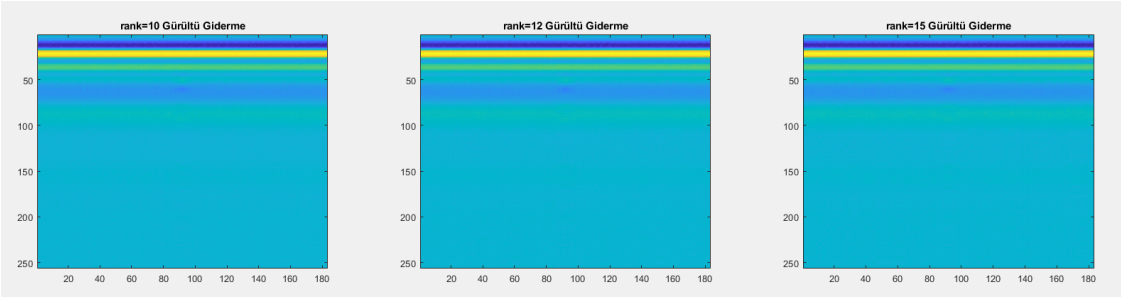
```
raw_data = im2double(imread("gpr2.png"));
noisy_data = imnoise(raw_data,'gaussian',0, 0.0002);
d1 = 10;
d2 = 12;
d3 = 15;
r = 1;

target1 = gpr_clutter_reduction_nmf(noisy_data, d1);
target2 = gpr_clutter_reduction_nmf(noisy_data, d2);
target3 = gpr_clutter_reduction_nmf(noisy_data, d3);

cr1 = gpr_clutter_reduction_nmf(noisy_data, r) - noisy_data;
cr2 = gpr_clutter_reduction_nmf(noisy_data, r) - noisy_data;
cr3 = gpr_clutter_reduction_nmf(noisy_data, r) - noisy_data;

figure
subplot(1, 2, 1)
imagesc(raw_data)
title('Orijinal Data')
subplot(1, 2, 2)
imagesc(noisy_data)
title('Gürültü Eklenmiş Data')

figure
subplot(2, 3, 1)
imagesc(target1)
title('rank=10 Gürültü Giderme')
subplot(2, 3, 2)
imagesc(target2)
title('rank=12 Gürültü Giderme')
subplot(2, 3, 3)
imagesc(target3)
title('rank=15 Gürültü Giderme')
subplot(2, 3, 4)
imagesc(cr1)
title('rank=10 Kargaşa Giderilmiş Görüntü')
subplot(2, 3, 5)
imagesc(cr2)
title('rank=12 Kargaşa Giderilmiş Görüntü')
subplot(2, 3, 6)
imagesc(cr3)
title('rank=15 Kargaşa Giderilmiş Görüntü')
```

Şekil 4.32: Test Kodu

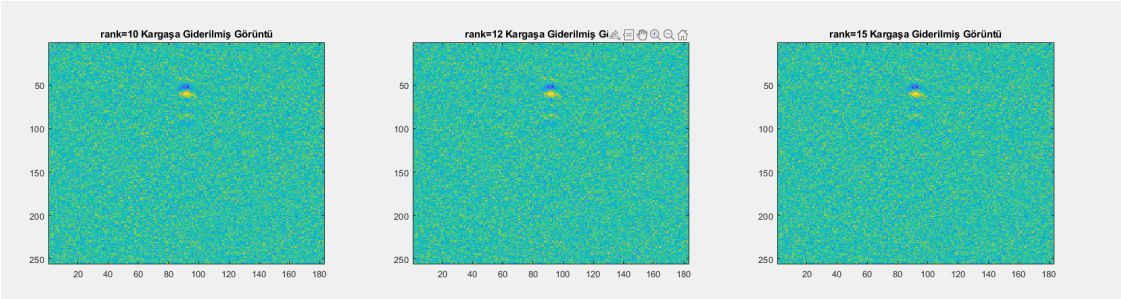


Şekil 4.33: Orijinal Görüntü ve Gürültü Eklenmiş Görüntü

Gürültü eklenen görüntü rank 10, 12 ve 15 değerleri için gürültü gidermeye tabi tutuldu. Daha sonra gürültü giderilmiş görüntülere kargaşa giderme uygulandı.

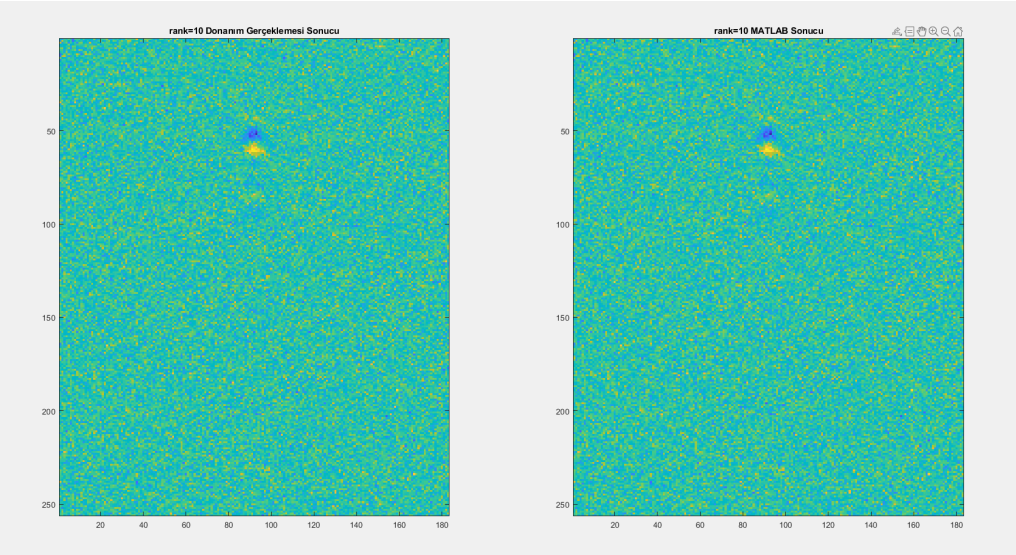


Şekil 4.34: Farklı Rank Değerleri İçin Gürültü Giderme



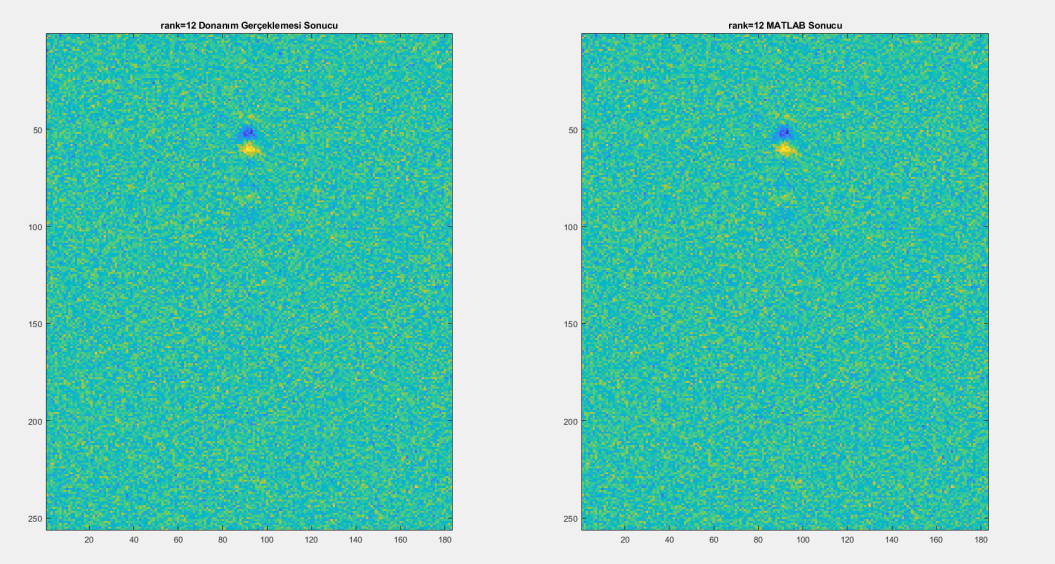
Şekil 4.35: Kargaşa Giderme Çıktıları

Görsel çıktılar sonuçların uygunluğunu göstermekte yetersiz kaldığından toplam mutlak piksel hata değeri verilen rank değerleri için hesaplanmıştır. Sonuçlar şu şekildedir:



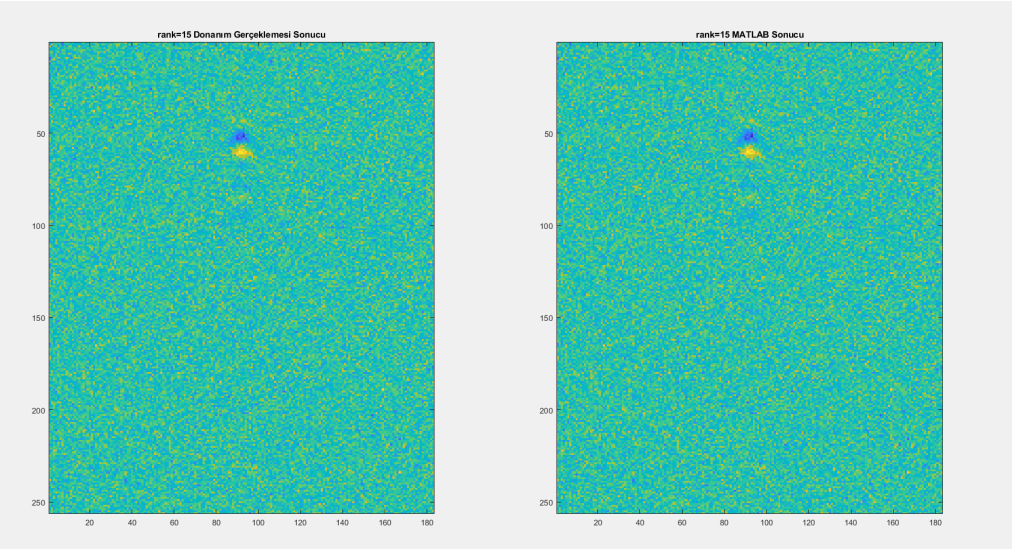
Şekil 4.36: MATLAB ve FPGA Sonuçları (rank=10)

Rank = 10 değeri için toplam piksel hata değeri 3.634×10^{-12} olarak elde edilmiştir.



Şekil 4.37: MATLAB ve FPGA Sonuçları (rank=12)

Rank = 12 değeri için toplam piksel hata değeri 4.637×10^{-12} olarak elde edilmiştir.



Şekil 4.38: MATLAB ve FPGA Sonuçları (rank=15)

Rank = 15 değeri için toplam piksel hata değeri 5.160×10^{-12} olarak elde edilmiştir.

Sonuç olarak, donanım gerçeklemesinden elde edilen çıktılar MATLAB çıktıları ile paralellik göstermektedir. Böylece NMF algoritmasının C diline başarıyla çevrildiği sonucuna varılmıştır. Ayrıca eklemek gerekir ki algoritmanın hem MATLAB yazılımı gerçeklemesinde hem de donanım gerçeklemesinde en iyi gürültü giderme ve kargaşa giderme sonuçları rank 10 değeri ile elde edilmiştir ve rank değeri arttıkça MATLAB sonucu ile donanım gerçeklemesinden elde edilen çıktı arasındaki piksel farkı artmaktadır.

4.5.2 Zedboard üzerinde veri alışverişi uygulamasının gerçeklenmesi

Daha önce Zynq işlemcili bir FPGA kullanılarak yapılan veri transferi projeleri incelendi. Amaç bakımından benzer projeler örnek alınarak uydu donanım tarafından işlenen verileri ana donanıma ileten ve kontrol amacıyla ilettiği verileri terminale bastıran bir C kodu yazıldı. Yazılan C kodu Bölüm 4.4'de gösterilen SDCard erişim modu ile Zedboard donanımı üzerinde gerçeklendi. [32]

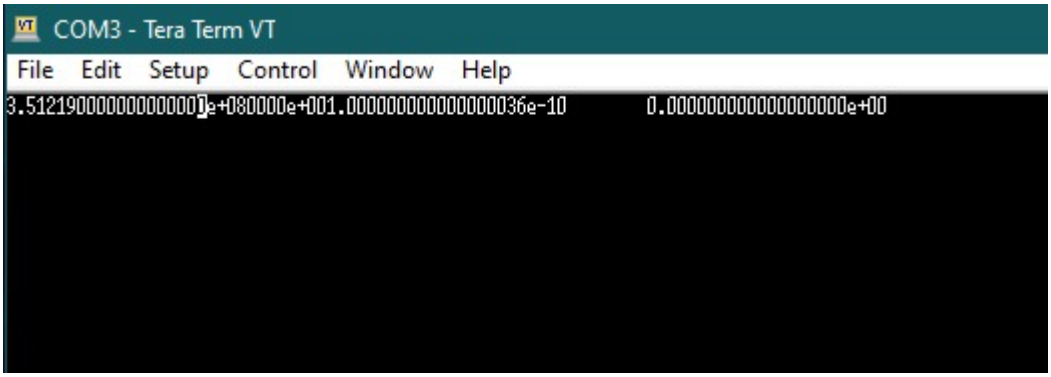
```
 *  Created on: 18 May 2022
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include "xil_types.h"
#include "xuartps.h"
#include "xparameters.h"

#define dataSize    5095*3
#define headSize    10
#define fileSize dataSize + headSize

int main(){

    u8 *radarData;
    u32 receivedBytes=0;
    u32 totalReceivedBytes=0;
    u32 status;
    u32 transmittedBytes=0;
    u32 totalTransmittedBytes=0;
    XUartPs_Config *myUartConfig;
    XUartPs myUart;

    radarData = malloc(sizeof(u8)*(fileSize));

    myUartConfig = XUartPs_LookupConfig(XPAR_PS7_UART_1_DEVICE_ID);
    status = XUartPs_CfgInitialize(&myUart, myUartConfig, myUartConfig->BaseAddress);
    if(status != XST_SUCCESS)
        print("UART initialization failed...\n\r");

    status = XUartPs_SetBaudRate(&myUart, 115200);
    if(status != XST_SUCCESS)
            print("Baud rate initialization failed...\n\r");

    while(totalReceivedBytes < fileSize){
        receivedBytes =  XUartPs_Recv(&myUart,(u8*)&radarData[totalReceivedBytes],fileSize);
        totalReceivedBytes += receivedBytes;
    }

    while(totalTransmittedBytes< fileSize){
        transmittedBytes = XUartPs_Send(&myUart,(u8*)&radarData[totalTransmittedBytes],1);
        totalTransmittedBytes += transmittedBytes;
    }

}
```

Şekil 4.39: Veri Alışverişi için C Kodu

```
# Freq  Real    Image
3.0000000000000000e+05        1.0000000000000000e-10        0.0000000000000000e+00
6.7985000000000000e+06        1.0000000000000000e-10        0.0000000000000000e+00
1.3297000000000000e+07        1.0000000000000000e-10        0.0000000000000000e+00
1.9795500000000000e+07        1.0000000000000000e-10        0.0000000000000000e+00
2.6294000000000000e+07        1.0000000000000000e-10        0.0000000000000000e+00
3.2792500000000000e+07        1.0000000000000000e-10        0.0000000000000000e+00
3.9291000000000000e+07        1.0000000000000000e-10        0.0000000000000000e+00
4.5789500000000000e+07        1.0000000000000000e-10        0.0000000000000000e+00
5.2288000000000000e+07        1.0000000000000000e-10        0.0000000000000000e+00
5.8786500000000000e+07        1.0000000000000000e-10        0.0000000000000000e+00
6.5285000000000000e+07        1.0000000000000000e-10        0.0000000000000000e+00
7.1783500000000000e+07        1.0000000000000000e-10        0.0000000000000000e+00
7.8282000000000000e+07        1.0000000000000000e-10        0.0000000000000000e+00
8.4780500000000000e+07        1.0000000000000000e-10        0.0000000000000000e+00
9.1279000000000000e+07        1.0000000000000000e-10        0.0000000000000000e+00
9.7777500000000000e+07        1.0000000000000000e-10        0.0000000000000000e+00
1.0427600000000000e+08        1.0000000000000000e-10        0.0000000000000000e+00
1.1077450000000000e+08        1.0000000000000000e-10        0.0000000000000000e+00
1.1727300000000000e+08        1.0000000000000000e-10        0.0000000000000000e+00
1.2377150000000000e+08        1.0000000000000000e-10        0.0000000000000000e+00
1.3027000000000000e+08        1.0000000000000000e-10        0.0000000000000000e+00
1.3676850000000000e+08        1.0000000000000000e-10        0.0000000000000000e+00
1.4326700000000000e+08        1.0000000000000000e-10        0.0000000000000000e+00
1.4976550000000000e+08        1.0000000000000000e-10        0.0000000000000000e+00
1.5626400000000000e+08        1.0000000000000000e-10        0.0000000000000000e+00
1.6276250000000000e+08        1.0000000000000000e-10        0.0000000000000000e+00
1.6926100000000000e+08        1.0000000000000000e-10        0.0000000000000000e+00
1.7575950000000000e+08        1.0000000000000000e-10        0.0000000000000000e+00
1.8225800000000000e+08        1.0000000000000000e-10        0.0000000000000000e+00
1.8875650000000000e+08        1.0000000000000000e-10        0.0000000000000000e+00
1.9525500000000000e+08        1.0000000000000000e-10        0.0000000000000000e+00
2.0175350000000000e+08        1.0000000000000000e-10        0.0000000000000000e+00
2.0825200000000000e+08        1.0000000000000000e-10        0.0000000000000000e+00
2.1475050000000000e+08        1.0000000000000000e-10        0.0000000000000000e+00
2.2124900000000000e+08        1.0000000000000000e-10        0.0000000000000000e+00
2.2774750000000000e+08        1.0000000000000000e-10        0.0000000000000000e+00
2.3424600000000000e+08        1.0000000000000000e-10        0.0000000000000000e+00
2.4074450000000000e+08        1.0000000000000000e-10        0.0000000000000000e+00
2.4724300000000000e+08        1.0000000000000000e-10        0.0000000000000000e+00
2.5374150000000000e+08        1.0000000000000000e-10        0.0000000000000000e+00
2.6024000000000000e+08        1.0000000000000000e-10        0.0000000000000000e+00
2.6673850000000000e+08        1.0000000000000000e-10        0.0000000000000000e+00
2.7323700000000000e+08        1.0000000000000000e-10        0.0000000000000000e+00
2.7973550000000000e+08        1.0000000000000000e-10        0.0000000000000000e+00
2.8623400000000000e+08        1.0000000000000000e-10        0.0000000000000000e+00
2.9273250000000000e+08        1.0000000000000000e-10        0.0000000000000000e+00
2.9923100000000000e+08        1.0000000000000000e-10        0.0000000000000000e+00
3.0572950000000000e+08        1.0000000000000000e-10        0.0000000000000000e+00
3.1222800000000000e+08        1.0000000000000000e-10        0.0000000000000000e+00
3.1872650000000000e+08        1.0000000000000000e-10        0.0000000000000000e+00
3.2522500000000000e+08        1.0000000000000000e-10        0.0000000000000000e+00
3.3172350000000000e+08        1.0000000000000000e-10        0.0000000000000000e+00
```

Şekil 4.40: Test Verisi

Gerçeklemeyi denemek için Şekil 4.37'deki örnek radar verisinin uydu donanımdan bilgisayara aktarımı sağlandı.



Şekil 4.41: C Kodu Çıktısı

Yukarıdaki şekilde görüleceği üzere sadece rastgele bir veri terminale bastırıldığından veri iletiminin tamamlanıp tamamlanmadığı belirsizlik gösterse de bu durumun veri türü ya da yazılım ile haberleşmenin yapıldığı terminal arasındaki sembol oranı uyumsuzluğundan kaynaklandığı düşünüldü. Veri iletiminin iyileşmesi için çalışmalar devam etmektedir.

5. GERÇEKÇİ KISITLAR, SONUÇLAR VE ÖNERİLER

5.1 Çalışmanın Uygulama Alanı

Çalışmanın uygulama alanı olabilecek yere nüfuz eden radar birçok farklı sektörde toprağa gömülü nesneleri bulmak için kullanılmaktadır. Özellikle anti personel mayınlarının tespitinde askeri alanda kullanılan radarlarda hedef tespiti için görüntünün işlenmesi ve kargaşanın giderilmesi gerekmektedir. Projede uygulanacak matris ayrışımları yere nüfuz eden radar görüntülerinin işlenmesinde yaygınca kullanılan ve FPGA gerçeklenmesi için görece basit algoritmalardır. Gürbüz negatif matris ayrışımı gibi başarılı bir görüntü işleme algoritmanın FPGA üzerinde gerçeklenmesi ile doğrudan radar üzerinde taşınabilir, ucuz ve hızlı bir çözüm geliştirilmiş olacaktır.

5.2 Gerçekçi Tasarım Kısıtları

5.2.1 Maliyet

Projenin yazılım kısmında ücretsiz MATLAB öğrenci paketlerinden faydalanılacaktır. Projenin maliyet bölümünün hazırlandığı tarihlerde dolar kuru yaklaşık 17 lira ve saatlik asgari ücret yaklaşık 1.5$ olduğundan yeni mezun mühendis maaşı 3 $/saat varsayılmıştır. Elektronik ve Haberleşme Tasarım Projesi iki dönemde toplam 28 hafta sürmekte ve dersin AKTS'si 5.5 olduğundan her öğrencinin haftalık 5.5 saat ayıracağı varsayılmıştır. Bu hesap sonucunda toplam işçi maaliyeti 924$ olarak gerçekleşecektir. İşçilik masrafına ek olarak bir FPGA temin edilmesi gereklidir onun da maaliyeti 336.75$ olacaktır. Bu durumda projenin toplam maaliyeti 1260.75$ olacaktır.

5.2.2 Standartlar

Bu proje IEEE standartlarına ve İstanbul Teknik Üniversitesi'nin öğrenci etik kurallarına uyularak hazırlanmıştır.

5.2.3 Sosyal, çevresel ve ekonomik etki

Yere nüfuz eden radar ile gömülü nesne tespiti askeriye, arkeoloji ve inşaat alanlarında sıklıkla başvurulan bir yöntemdir. Sunulan çözümde radar görüntülerinin doğrudan cihaz üzerinde işlenmesi ve gömülü nesnenin tespiti söz konusu olacaktır. Maliyeti

düşük ve hızlı sonuç veren bir donanım kolaylıkla üretilebilir hem de ilgili uygulamalarda pratiklik kazandırır.. Bu sayede gömülü nesne tespitinin kritik uygulamalarında yaygınlaşır.

5.2.4 Sağlık ve güvenlik riskleri

Yere nüfuz eden radar yere gömülü nesneleri bulmak için kullanılan toprağı kazmayı gerektirmeyen son derece güvenli bir ölçüm yöntemidir. Özellikli askeri amaçlı mayın arama çalışmalarında oldukça faydalı ve güvenli bir teknik olarak karşımıza çıkmaktadır.

5.3 Sonuçlar

Projede birçok negatif olmayan matris ayrışım algoritması yere nüfuz eden radar görselinde kargaşa giderme amacıyla kullanılmıştır ve sonuçlar karşılaştırılmıştır. En iyi sinyal gürültü oranını veren gürbüz negatif olmayan matris ayrışımı algoritmasının FPGA'de gerçeklenmesine karar verilmiştir. Ardından gürültülü yere nüfuz eden radar görüntüsü kargaşa giderme işleminde performans düşmesine sebep olduğundan gürültü giderme uygulaması yapılmıştır. Gürültü giderme uygulamasında tekil değer ayrışımı ve negatif olmayan matris ayrışımı kullanılmıştır. Gürültü giderme sonrası yapılan kargaşa giderme işlemi ile gürültülü resim üzerinde yapılan kargaşa giderme işlemi karşılaştırılmış ve gürültü gidermenin kargaşa giderme işlemi için olan önemi görülmüştür. FPGA sınıfının sağladığı esneklik Linux gerçeklemesi sonuç vermeyince proje yeniden düzenlenmiştir. Hem gerekli görüntü işleme algoritmaları hem de veri transferi donanım üzerinde işletim sistemi olmaksızın gerçeklenmiştir. Zynq işlemci üzerinde işletim sistemi gerçeklenemeyince FPGA donanımının işletim sistemine sahip farklı bir donanıma uydu donanım olarak bağlanmasıyla hareketlilik ve işlem paralelliği sağlanmıştır. Daha sonra farklı rank değerleri için donanım gerçeklemesi test edilerek toplam piksel hata değeri baz alınarak en uygun rank değeri 10 olarak elde edilmiştir.

5.4 Geleceğe Yönelik Öneriler

Yazılım ortamında denenen algoritmalardan gürbüz negatif olmayan matris ayrışımı algoritmasının sinyal gürültü oranı olarak büyük bir avantaj sağladığı ancak hız olarak diğer algoritmalara göre düşük bir performansa sahip olduğu gözlenmiştir. Bu

bağlamda sinyal gürültü oranı kaybı yaşamadan hesaplama hızı performansında iyileşme sağlayabilecek bir negative olmayan matris ayrışımı algoritması üzerine araştırmalara devam etmek sonraki çalışmaların konusu olabilir. Ek olarak işletim sistemi Linux olan bir sunucu bilgisayar kullanılarak FPGA üzerinde Gömülü Linux gerçeklenebilir ve bu sayede donanım gerekesinimi azaltılabilir. Matris işlemleri MATLAB Coder ile ayrı ayrı kütüphane haline getirilerek üretilen kodlar farklı görüntü işleme algoritmaları için yeniden kullanılır hale getirilebilir.

KAYNAKLAR

[1] Karlsen, B., Larsen, J., Sorensen, H., B., D. & Jakobsen, K., B. (2001). Comparison of PCA and ICA based clutter reduction in GPR systems for anti-personal landmine detection, *Proc. 11th IEEE Signal Process*ing *Workshop on Statist. Signal Processing*. (146-149), Singapur, Ağustos 6-8.

[2] Abujarad, F. & Omar, A. (2006). GPR data processing using the component separation methods PCA and ICA. *Proc. Int. Workshop Imag. Syst.Techn.*, (60-64), İtalya: Minori, Nisan 29.

[3] Abujarad, F., Nadim, G. & Omar, A. (2005). Clutter reduction and detection of landmine objects in ground penetrating radar data using singular value decomposition (SVD). In Lambot, S. & Gorriti, A., G. (Ed.), *Proc. IEEE 3rd Int. Workshop Adv. Ground Penetrating Radar*, (37-42), Hollanda: Delft University of Technology , Mayıs 2-3 .

[4] Riaz, M., M. & Ghafoor, A. (2012). Information theoretic criterion based clutter reduction for ground penetrating radar, *Progress In Electromagnetics Research, 14,* 147- 164.

[5] Nabelek, D. & Ho, K., C. (2015). Detection of deeply buried non-metal objects by ground penetrating radar using non-negative matrix factorization. In Bishop, S.S. (Ed), *Proc. SPIE 9454, Detection and Sensing of Mines, Explosive Objects, and Obscured Targets XX*, Mayıs 21.

[6] Kumlu, D. & Erer, I. (2018). Clutter removal in GPR images using non-negative matrix factorization, *Journal of Electromagnetic Waves and Applications, 32* (16), 2055–2066.

[7] Erichson, N., B., Mendible, A., Wihlborn, S. & Kutz, J. (2017). Randomized Nonnegative Matrix Factorization. *Pattern Recognition Letters. 104* (1).

[8] Kumlu, D. & Erer, I. (2020). Improved Clutter Removal in GPR by Robust Nonnegative Matrix Factorization, *IEEE Geoscience and Remote Sensing Letters*, *17* (6), 958-962.

[9] Kumlu, D. & Erer, I. (2017). A comparative study on clutter reduction techniques in GPR images, *2017 4th International Conference on Electrical and Electronic Engineering (ICEEE)*, (323-328), Türkiye: Ankara, Nisan 8-10 .

[10] Jiang, Y. & Chai, T. (2016). Image Processing and Analysis Based on Partial Differential Equation Method. *Proceedings of the 2015 4th National Conference on Electrical, Electronics and Computer Engineering,* (985-990), Çin: Xi'an, Aralık 12-13.

[11] Rudin, L., I., Osher, S. & Fatemi, E. (1992). Nonlinear total variation based noise removal algorithms, *Physica D: Nonlinear Phenomena, 60* (1-4), 259-268.

[12] Tomasi, C. & Manduchi, R. (1998). Bilateral filtering for gray and color images. *Sixth International Conference on Computer Vision,* (839-846), Hindistan: Bombay, Ocak 4-7.

[13] Alisha, P., B. & Gnana Sheela K. (2016). Image denoising techniques-an overview, *IOSR Journal of Electronics and Communication Engineering, 11* (1), 78-84.

[14] Hyvarinen, A., Oja, E., Hoyer, P. & Hurri, J. (1998). Image feature extraction by sparse coding and independent component analysis. *Proceedings. Fourteenth International Conference on Pattern Recognition (Cat. No.98EX170)*, (1268-1273), Avustralya: Brisbane, Ağustos 16-20.

[15] Liu, Q., Cheng, J., Lu, H. & Ma, S. (2004). Distance based kernel PCA image reconstruction. *Proceedings of the 17th International Conference on Pattern Recognition,* (670-673), Birleşik Krallık: Cambridge, Ağuston 26.

[16] Zhang, L., Dong, W., Zhang, D. & Shi, G. (2010). Two-stage image denoising by principal component analysis with local pixel grouping. *Pattern Recognition, 43* (4), 1531-1549.

[17] Ding, C. & Ye, J. (2005). Two-dimensional singular value decomposition for 2D maps and images**.** *In Proceedings of the 2005 SIAMInternational Conference on Data Mining,* (32-43), Nisan.

[18] Rajwade, A., Rangarajan, A. & Banerjee, A. (2013). Image denoising using the higher order singular value decomposition, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *35* (4), 849-862.

[19] James, R., Jolly, A., M, Anjaldi, C. & Michael, D. (2015). Image denoising using adaptive PCA and SVD. *Fifth International Conference on Advances in Computing and Communications (ICACC)*, (383-386), Hindistan: Kochi, Eylül 2-4.

[20] González, C., Sánchez, S., Paz, A., Resano, J., Mozos, D., & Plaza, A. (2013). Use of FPGA or GPU-based architectures for remotely sensed hyperspectral image processing. *Integration*, *46*(2), 89–103. https://doi.org/10.1016/j.vlsi.2012.04.002

[21] Guda, M., Gasser, S., El-Mahallawy, M. S., & Shehata, K. (2020). FPGA Implementation of L1/2 Sparsity Constrained Nonnegative Matrix Factorization Algorithm for Remotely Sensed Hyperspectral Image Analysis. *IEEE Access*, *8*, 12069–12083. https://doi.org/10.1109/access.2020.2966044

[22] Nascimento, J. M. P., Vestias, M., & Martin, G. (2015, July). FPGA-based architecture for hyperspectral unmixing. *2015 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*. https://doi.org/10.1109/igarss.2015.7326130

[23] Digilent Inc. (2017). Cypress CY7C64225 USB-to-UART Setup Guide

[24] Lowe, S. (2021, June). *Zedboard Programming Guide in SDK*. Digilent Reference. https://digilent.com/reference/learn/programmable-logic/tutorials/zedboard-programming-guide/start

[25] Digilent Inc. (2017). ZedBoard™ Getting Started Guide

[26] Digilent Forum. (2018, September). *Vivado-SDK 2017.2 Linux*. https://forum.digilentinc.com/topic/5352-vivado-sdk-20172-linux/,

[27] Digilent Inc. (2013, January). Getting Started With Embedded Linux – ZedBoard

[28] *Eclipse Community Forums: C / C++ IDE (CDT) » Unresolved inclusion | The Eclipse Foundation*. (2018). Unresolved Inclusion While Developing under Windows for Linux. https://www.eclipse.org/forums/index.php/t/1096058/

[29] **Xilinx** (2016, November). SDSoCEnviroment Tutorial. Digilent Reference

[30] *İTÜ GSTL Staj Raporu*. (2017). Mehmet Onur Demirtürk.

[31] *Automatically Converting MATLAB Code to C Code - Video*. (2020, September 3). MATLAB. https://www.mathworks.com/videos/automatically-converting-matlab-code-to-c-code-96483.html

[32] *Data Transfer between PC and ZedBoard through UART Interface*. (2020, March 2). [Video]. YouTube. https://m.youtube.com/watch?v=lzQ9hJ-wevg