

**T.C.**  
**GEBZE TEKNİK ÜNİVERSİTESİ**  
**FEN BİLİMLERİ ENSTİTÜSÜ**

**PARÇACIK FİLTRELERİ İLE HEDEF**  
**TAKİBİ İÇİN YENİDEN ÖRNEKLEME**  
**ALGORİTMALARININ GELİŞTİRİLMESİ**

**AHMET BACAK**  
**DOKTORA TEZİ**  
**ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI**

**GEBZE**  
**2021**

**T.C.**  
**GEBZE TEKNİK ÜNİVERSİTESİ**  
**FEN BİLİMLERİ ENSTİTÜSÜ**

**PARÇACIK FİLTRELERİ İLE HEDEF  
TAKİBİ İÇİN YENİDEN ÖRNEKLEME  
ALGORİTMALARININ GELİŞTİRİLMESİ**

**AHMET BACAK**  
**DOKTORA TEZİ**  
**ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI**

**DANIŞMANI**  
**DR. ÖĞR. ÜYESİ ALİ KÖKSAL HOCAOĞLU**

**GEBZE**  
**2021**

**T.R.**  
**GEBZE TECHNICAL UNIVERSITY**  
**GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES**

**DEVELOPMENT OF RESAMPLING  
ALGORITHMS FOR TARGET TRACKING  
WITH PARTICLE FILTERS**

**AHMET BACAK**  
**A THESIS SUBMITTED FOR THE DEGREE OF  
DOCTOR OF PHILOSOPHY**  
**DEPARTMENT OF ELECTRONICS ENGINEERING**

**THESIS SUPERVISOR**  
**ASSIST. PROF. DR. ALİ KÖKSAL HOCAOĞLU**

**GEBZE**  
**2021**

|                                  |                                |
|----------------------------------|--------------------------------|
| <b>GEBZE TEKNİK ÜNİVERSİTESİ</b> | <b>DOKTORA JÜRİ ONAY FORMU</b> |
|----------------------------------|--------------------------------|

GTÜ Fen Bilimleri Enstitüsü Yönetim Kurulu'nun 22/01/2021 tarih ve 2021/04 sayılı kararıyla oluşturulan jüri tarafından 25/06/2021 tarihinde tez savunma sınavı yapılan Ahmet BACAK'ın tez çalışması Elektronik Mühendisliği Anabilim Dalında DOKTORA tezi olarak kabul edilmiştir.

### **JÜRİ**

ÜYE

(TEZ DANIŞMANI) : Dr. Öğr. Üyesi Ali Köksal HOCAOĞLU

ÜYE

: Prof. Dr. Fikret ÇALIŞKAN

ÜYE

: Doç. Dr. Koray KAYABOL

ÜYE

: Prof. Dr. Hasari ÇELEBİ

ÜYE

: Doç. Dr. Sultan Aldırmaz ÇOLAK

### **ONAY**

Gebze Teknik Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu'nun  
...../...../..... tarih ve ...../..... sayılı kararı.

İMZA/MÜHÜR

## ÖZET

Doğrusal olmayan sistemlerde ve ölçüm ve süreç gürültülerinin Gauss dağılımına sahip olmadığı durumlarda, Parçacık filtreleri yaygın olarak kullanılmaktadır. Parçacık filtreleri olasılık yoğunluğu temsillerine dayalı, örneklem tabanlı sıralı bir Monte Carlo yöntemidir. Parçacık filtreleri algoritmalarında dört temel adım vardır. Bunlar, sırasıyla parçacıkların ilklendirilmesi, parçacıklara ait ağırlıkların güncellemesi, yeniden örnekleme adımı ve kestirme adımıdır. Parçacıkların oluşturulması ve parçacıkların ağırlıklarının güncelleme işlemi önem yoğunluk fonksiyonundan yararlanılarak yapılır. Zaman içinde bazı parçacıkların ağırlıkları çok artar ve diğer parçacıklarının ağırlıkları ise çok küçük kalır. Bu durum, az sayıda örnek ile sonsal dağılımın kestirilmesine neden olur. Bu problemin önüne geçmek amacıyla, yeniden örnekleme algoritmaları kullanılmaktadır. Yeniden örnekleme algoritmaları tahminlerdeki hataları azaltmaktadır.

Bu tez çalışmasında, Açgözlü yaklaşım, torba problemini çözerek oluşturulan dinamik programlama yaklaşımı yeniden örnekleme algoritması ve LSTM derin öğrenme yaklaşımı tabanlı yeniden örnekleme algoritmaları geliştirilmiştir. Önerilen algoritmaların performansları farklı hareket modellerinde test edilmiştir. Doğrusal olmayan hareket modeli, kartezyen hareket modeli, dairesel hareket modelinde hedef takibi yapılmıştır. Önerilen yeniden örnekleme algoritmalarının performansları literatürde sık karşılaşılan yeniden örnekleme algoritmalarının performansları ile karşılaştırılmıştır. Ayrıca, özellikle su altı sistemlerinde sık karşılaşılan sadece kerterize bağlı hedef hareket takibi problemi için, Parçacık filtresi yaklaşımına dayalı hedef takip algoritması önerilmiştir. Önerilen hedef takip algoritmasının performansı literatürdeki çalışmalarla karşılaştırılmıştır.

**Anahtar Kelimeler:** Parçacık Filtresi, Yeniden Örnekleme Algoritması, Açgözlü yaklaşım, Dinamik programlama, Derin öğrenme, Hedef Hareket Analizi.

## SUMMARY

Particle filters are widely used in nonlinear systems and where measurement and process noises do not have Gaussian distribution. Particle filters are a sample-based sequential Monte Carlo method based on probability density representations. There are four basic steps in particle filter algorithms. These are the initialization of the particles, the update of the weights of the particles, the resampling step and the estimation step, respectively. Creating particles and updating the weight of the particles is done by using the importance density function. Over time, the weights of some particles increase greatly, and the weights of other particles remain very small. This situation causes posterior distribution to be estimated with a small number of samples. Resampling algorithms are used to avoid this problem. Resampling algorithms reduce errors in predictions.

In this thesis, the Greedy approach, the dynamic programming approach resampling algorithm created by solving the bag problem, and LSTM deep learning approach based resampling algorithms have been developed. The performances of the proposed algorithms have been tested in different motion models. Target tracking has been carried out in the nonlinear motion model, cartesian motion model, circular motion model. The performances of the proposed resampling algorithms were compared with the performances of the resampling algorithms that are frequently encountered in the literature. In addition, a target tracking algorithm using particle filters approach has been proposed for the bearing-only target motion tracking problem, which is frequently encountered especially in underwater systems. The performance of the proposed target tracking algorithm has been compared with the studies in the literature.

**Key Words: Particle Filter, Resampling Algorithm, Greedy approach, Dynamic programming, Deep learning, Target Motion Analysis.**

## TEŞEKKÜR

Başta, doktora eğitimimde ve akademik hayatımda desteğini ve yardımlarını hiçbir zaman esirgemeyip bilgisi ile bu çalışmanın oluşmasının yolunu açan danışmanım Dr. Öğr. Üyesi Ali Köksal HOCAOĞLU'na,

Doktora tez izleme süresince yorumlarıyla katkı sağlayan ve vakit ayıran Prof. Dr. Fikret ÇALIŞKAN'a ve Doç. Dr. Koray KAYABOL'a,

Tezin inceleme sürecindeki yorumlarıyla katkı sağlayan Doç. Dr. Sultan Aldırmaz ÇOLAK'a,

Tez savunması için zaman ayıran, Prof. Dr. Hasari ÇELEBİ'ye,

ve göstermiş olduğu manevi desteklerinden dolayı sevgili aileme en içten teşekkürlerimi sunarım.

# İÇİNDEKİLER

|                                                     | <b><u>Sayfa</u></b> |
|-----------------------------------------------------|---------------------|
| ÖZET                                                | v                   |
| SUMMARY                                             | vi                  |
| TEŞEKKÜR                                            | vii                 |
| İÇİNDEKİLER                                         | viii                |
| SİMGELER VE KISALTMALAR DİZİNİ                      | xi                  |
| ŞEKİLLER DİZİNİ                                     | xii                 |
| TABLolar DİZİNİ                                     | xiv                 |
| <br>                                                |                     |
| 1. GİRİŞ                                            | 1                   |
| 1.1. Tezin Amacı                                    | 1                   |
| 1.2. Tezin Katkısı                                  | 2                   |
| 1.3. Tezin İçeriği                                  | 3                   |
| 2. LİTERATÜR TARAMASI                               | 5                   |
| 2.1. Hedef Takip Algoritmaları                      | 5                   |
| 2.1.1. Alfa-Beta Filtresi                           | 5                   |
| 2.2. Yeniden Örnekleme Algoritmaları                | 11                  |
| 2.2.1. Systematic Yeniden Örnekleme Algoritması     | 11                  |
| 2.2.2. Stratified Yeniden Örnekleme Algoritması     | 13                  |
| 2.2.3. MultiNominal Yeniden Örnekleme Algoritması   | 15                  |
| 2.2.4. Residual Yeniden Örnekleme Algoritması       | 16                  |
| 2.2.5. Minimum Örnekleme Varyansı Algoritması       | 19                  |
| 3. AÇGÖZLÜ YAKLAŞIMLA YENİDEN ÖRNEKLEME ALGORİTMASI | 21                  |
| 3.1. Açgözlü Yaklaşım                               | 21                  |
| 3.2. Açgözlü Yeniden Örnekleme Algoritması          | 22                  |
| 3.3. Açgözlü Yeniden Örnekleme Algoritması Testleri | 25                  |
| 3.3.1. Lineer olmayan modelde hedef takibi          | 25                  |
| 3.3.2. Kartezyen koordinatlarda hedef takibi        | 31                  |
| 3.3.3. Sürekli dönüş modelinde hedef takibi         | 34                  |

|                                                                             |    |
|-----------------------------------------------------------------------------|----|
| 3.4. Genel Değerlendirme                                                    | 37 |
| 4. TORBA PROBLEMİ ÇÖZÜMÜ TABANLI YENİDEN ÖRNEKLEME ALGORİTMASI              | 38 |
| 4.1. Dinamik Programlama                                                    | 39 |
| 4.2. Torba Problemi                                                         | 40 |
| 4.3. Torba Problemi Çözümü Tabanlı Yeniden Örneklemeye                      | 44 |
| 4.4. Torba Problemi Çözümü Tabanlı Yeniden Örneklemeye Algoritması Testleri | 48 |
| 4.4.1. Lineer olmayan modelde hedef takibi                                  | 48 |
| 4.4.2. Kartezyen koordinatlarda hedef takibi                                | 53 |
| 4.4.3. Sürekli dönüş modelinde hedef takibi                                 | 56 |
| 4.5. Genel Değerlendirme                                                    | 58 |
| 5. LSTM DERİN ÖĞRENME ALGORİTMASI TABANLI YENİDEN ÖRNEKLEME ALGORİTMASI     | 59 |
| 5.1. Uzun Kısa Vadeli Bellek Ağları (LSTM)                                  | 60 |
| 5.2. LSTM Tabanlı Yeniden Örneklemeye Algoritması                           | 65 |
| 5.2.1. LSTM Tabanlı Yeniden Örneklemeye Algoritması                         | 65 |
| 5.2.2. Böl Fethet Yaklaşımının LSTM Dizi Tahmininde Kullanılması            | 69 |
| 5.3. Parametrelerin Etkisi                                                  | 72 |
| 5.3.1. Gizli Birim Sayısının Etkisi                                         | 72 |
| 5.3.2. Algoritmaların Etkisi                                                | 73 |
| 5.3.3. Tur Sayısının Etkisi                                                 | 73 |
| 5.3.4. Batch Sayısının Etkisi                                               | 74 |
| 5.3.5. Katman Sayısının Etkisi                                              | 75 |
| 5.4. LSTM Tabanlı Yeniden Örneklemeye Algoritması Test Sonuçları            | 76 |
| 5.4.1. Lineer Olmayan Hareket Modelindeki Sonuçlar                          | 76 |
| 5.4.2. Torba Problemindeki Sonuçlar                                         | 80 |
| 5.4.3. Gürültülü Verilerdeki Sonuçlar                                       | 82 |
| 5.4.4. Algoritma Çalışma Süreleri                                           | 85 |
| 6. HEDEF HAREKET ANALİZİNİN PARÇACIK FİLTRESİ İLE YAPILMASI                 | 87 |
| 6.1. Hedef Hareket Analizi                                                  | 87 |
| 6.1.1. Sadece Kerterize Bağlı Hedef Takibi Durum Modeli                     | 89 |
| 6.2. Açgözlü Yeniden Örneklemeye Kullanan Parçacık Filtresi ile TMA         | 90 |

|                                                                       |     |
|-----------------------------------------------------------------------|-----|
| 6.2.1. Başlangıç Değerleri için Arama Algoritması                     | 90  |
| 6.2.2. Parçacık Filtresi ile TMA Çözümü                               | 92  |
| 6.3. Parametre Optimizasyonu                                          | 98  |
| 6.3.1. Maliyet Fonksiyonlarının Etkisi                                | 99  |
| 6.3.2. Parçacık Sayısının Etkisi                                      | 100 |
| 6.3.3. Süreç Gürültüsü Standart Sapma Değerinin Etkisi                | 101 |
| 6.3.4. Ölçüm Gürültüsü Standart Sapma Değerinin Etkisi                | 102 |
| 6.3.5. Açgözlü Yeniden Örnekleme Algoritmasında Eşik Değerinin Etkisi | 103 |
| 6.3.6. Yeniden Örnekleme Algoritmasının Etkisi                        | 104 |
| 6.4. Test Sonuçları                                                   | 105 |
| 6.4.1. Hedef Hareket Analizi Çözümü Algoritması Sonuçları             | 105 |
| 6.4.2. Büyük Patlama Büyük Çöküş Algoritması İle Karşılaştırma        | 111 |
| 6.4.3. Algoritma Çalışma Süreleri                                     | 113 |
| 6.5. Genel Değerlendirme                                              | 114 |
| 7. SONUÇLAR VE GELECEK ÇALIŞMA ÖNERİLERİ                              | 115 |
| KAYNAKLAR                                                             | 117 |
| ÖZGEÇMİŞ                                                              | 125 |
| EKLER                                                                 | 126 |

## SİMGELER VE KISALTMALAR DİZİNİ

### Simgeler ve Açıklamalar

#### Kısaltmalar

|         |   |                                                                             |
|---------|---|-----------------------------------------------------------------------------|
| GPU     | : | Grafik İşlemci Ünitesi (Ing: Graphical Processing Unit)                     |
| LSTM    | : | Uzun Kısa Vadeli Bellek (Ing: Long Short Term Memory)                       |
| PF      | : | Parçacık Filtresi (Ing: Particle Filter)                                    |
| RNN     | : | Tekrarlayan Sinir Ağı (Ing: Recurrent Neural Network)                       |
| MSV     | : | Minimum Örnekleme Varyansı (Ing: Minimum Sampling Variance)                 |
| SIR     | : | Sıralı Önem Yeniden Örnekleme (Ing: Sequence Importance Resampling)         |
| R       | : | Mesafe (Ing: Range)                                                         |
| C       | : | Rota (Ing: Course)                                                          |
| S       | : | Sürat (Ing: Speed)                                                          |
| CPU     | : | Merkezi İşlemci Ünitesi (Ing: Central Processing Unit)                      |
| m       | : | Metre                                                                       |
| km      | : | Kilometre                                                                   |
| sn      | : | Saniye                                                                      |
| EKF     | : | Genişletilmiş Kalman Filtresi (Ing: Extended Kalman Filter)                 |
| UKF     | : | Kokusuz Kalman Filtresi (Ing: Unscented Kalman Filter)                      |
| TMA     | : | Hedef Hareket Analizi (Ing: Target Motion Analysis)                         |
| GTÜ     | : | Gebze Teknik Üniversitesi                                                   |
| RMSE    | : | Ortalama Karesel Hatanın Karekökü                                           |
| MAE     | : | Ortalama Mutlak Hata                                                        |
| MAPE    | : | Ortalama Yüzdesel Mutlak Hata (Ing: Mean Absolute Percentage Error)         |
| RMSElog | : | Ortalama Karesel Log Hatasının Karekökü (Ing: Root Mean Squared Log Error ) |

## ŞEKİLLER DİZİNİ

| <b><u>Sekil No:</u></b>                                                                            | <b><u>Sayfa</u></b> |
|----------------------------------------------------------------------------------------------------|---------------------|
| 3.1: Hedefin parçacık filtreleri ile takip edilmesi.                                               | 26                  |
| 3.2: Eşik değerinin ortalama karesel hataya etkisi.                                                | 27                  |
| 3.3: Farklı parçacık sayılarında farklı algoritmalarının performansı.                              | 28                  |
| 3.4: Ölçüm gürültüsünün ortalama karesel hataya etkisi.                                            | 29                  |
| 3.5: Gürültü tipinin ortalama karesel hataya etkisi.                                               | 30                  |
| 3.6: Algoritmaların çalışma süreleri.                                                              | 31                  |
| 3.7: 3 boyutlu kartezyen koordinatlarda hareket izleme.                                            | 33                  |
| 3.8: 3 boyutlu kartezyen koordinatlarda algoritmaların sonuçları.                                  | 33                  |
| 3.9: Dairesel hareket modelinde hareket izleme.                                                    | 34                  |
| 3.10: Dairesel hareket modelinde 0.5 standart sapmalı ölçüm gürültüsünde algoritmaların sonuçları. | 36                  |
| 3.11: Dairesel hareket modelinde 1 standart sapmalı ölçüm gürültüsünde algoritmaların sonuçları.   | 36                  |
| 4.1: Eşik değerinin ortalama karesel hataya etkisi.                                                | 49                  |
| 4.2: Genişletme katsayısı etkisi.                                                                  | 50                  |
| 4.3: Farklı parçacık sayılarında farklı algoritmalarının performansı.                              | 51                  |
| 4.4: Ölçüm gürültüsünün ortalama karesel hataya etkisi                                             | 52                  |
| 4.5: Algoritmaların çalışma süreleri.                                                              | 53                  |
| 4.6: 3 boyutlu kartezyen koordinatlarda hareket izleme.                                            | 55                  |
| 4.7: 3 boyutlu kartezyen koordinatlarda algoritmaların sonuçları                                   | 56                  |
| 4.8: Dairesel hareket modelinde hareket izleme                                                     | 57                  |
| 4.9: Dairesel hareket modelinde 1 standart sapmalı ölçüm gürültüsünde algoritmaların sonuçları.    | 58                  |
| 5.1: LSTM hücresi blok diyagramı.                                                                  | 60                  |
| 5.2: Diziden diziye problemler için LSTM tabanlı yapay sinir ağı blok diyagramı.                   | 63                  |
| 5.3: Sohbet botu örneği metinleri.                                                                 | 64                  |
| 5.4: Şekil 5.3'deki sohbet botu örneğindeki veri setinin çalışması.                                | 65                  |

|       |                                                                                       |    |
|-------|---------------------------------------------------------------------------------------|----|
| 5.5:  | LSTM yeniden örnekleme algoritması blok şeması.                                       | 66 |
| 5.6:  | LSTM yeniden örnekleme algoritması için giriş dizileri.                               | 67 |
| 5.7:  | LSTM yeniden örnekleme algoritması için çıkış dizileri.                               | 67 |
| 5.8:  | Gizli birim sayısının etkisi.                                                         | 72 |
| 5.9:  | LSTM ve RNN başarıları.                                                               | 73 |
| 5.10: | Tur sayısının etkisi.                                                                 | 74 |
| 5.11: | Algoritmaların Torba Problemi optimum çözümdeki indislerin<br>dice benzerlik yüzdesi. | 81 |
| 6.1:  | Parçacık filtresi tahmini açıları.                                                    | 95 |
| 6.2:  | Senaryo 1 ait hedef ve gözlemcinin x ve y konumları.                                  | 98 |

## TABLÖLAR DİZİNİ

| <b><u>Tablo No:</u></b>                                                                               | <b><u>Sayfa</u></b> |
|-------------------------------------------------------------------------------------------------------|---------------------|
| 3.1: Diziden diziye problem için Ağgözlü yaklaşım çözücüsü.                                           | 22                  |
| 3.2: Sürekli dönüş modelindeki konumların hesaplanması.                                               | 34                  |
| 4.1: Torba problemi çözümünün Dinamik Programlama sözde kodu.                                         | 41                  |
| 4.2: Torba problemi örneği.                                                                           | 43                  |
| 4.3: Torba probleminde dinamik programlama çözümü tablosu.                                            | 44                  |
| 4.4: Dinamik Programlama ile örneğin çözümü.                                                          | 44                  |
| 4.5: Sayısal örnek için dinamik algoritma tablosu.                                                    | 47                  |
| 4.6: Sürekli dönüş modelindeki konumların hesaplanması.                                               | 57                  |
| 5.1: Yaklaşık ağırlıklı olacak şekilde dizileri ikiye bölen algoritmanın sözde kodu.                  | 70                  |
| 5.2: Sayısal örneğin çözümü adımları.                                                                 | 71                  |
| 5.3: Batch sayısının başarıya etkisi.                                                                 | 75                  |
| 5.4: Katman sayısının başarıya etkisi.                                                                | 75                  |
| 5.5: Gizli birim sayısının RMSE etkisi ( $R=1$ , $Q=3.16$ , $N=25$ ).                                 | 77                  |
| 5.6: Orijinal algoritmalar ile LSTM tabanlı algoritmaların RMSE değeri ( $R=1$ , $Q=3.16$ , $N=25$ ). | 78                  |
| 5.7: Orijinal algoritmalar ile LSTM tabanlı algoritmaların RMSE değeri ( $R=1$ , $Q=3.16$ , $N=50$ ). | 79                  |
| 5.8: Orijinal algoritmalar ile LSTM tabanlı algoritmaların RMSE değeri ( $R=1$ , $Q=1$ ).             | 79                  |
| 5.9: Algoritmaların Torba Problemi optimum çözümdeki indislerin dice benzerlik yüzdesi.               | 81                  |
| 5.10: Gürültülü verilerdeki sonuçlar.                                                                 | 82                  |
| 5.11: Yeniden örnekleme algoritmaların düşük parçacık içeren indis sayıları.                          | 83                  |
| 5.12: En yüksek ağırlıklı parçacık sayısının oranı.                                                   | 85                  |
| 5.13: Algoritma çalışma süreleri.                                                                     | 85                  |

|       |                                                                                   |     |
|-------|-----------------------------------------------------------------------------------|-----|
| 6.1:  | Genel TMA problemi için basit arama metodu.                                       | 91  |
| 6.2:  | Genel TMA problemi çözüm algoritması.                                             | 95  |
| 6.3:  | Parçacık filtresi algoritması.                                                    | 97  |
| 6.4:  | Senaryo 1-2.                                                                      | 98  |
| 6.5:  | Senaryo 1 Maliyet fonksiyonlarının etkisi.                                        | 100 |
| 6.6:  | Senaryo 2 Maliyet fonksiyonlarının etkisi.                                        | 100 |
| 6.7:  | Parçacık sayısının etkisi.                                                        | 101 |
| 6.8:  | Süreç gürültüsü standart sapma değerinin etkisi.                                  | 102 |
| 6.9:  | Ölçüm gürültüsü standart sapma değerinin etkisi.                                  | 103 |
| 6.10: | Açgözlü yeniden örnekleme algoritmasında eşik değerinin etkisi.                   | 104 |
| 6.11: | Yeniden örnekleme algoritmalarının etkisi.                                        | 105 |
| 6.12: | Senaryo 1-9.                                                                      | 105 |
| 6.13: | Senaryo 1 sonuçları.                                                              | 107 |
| 6.14: | Senaryo 2 sonuçları.                                                              | 107 |
| 6.15: | Senaryo 3 sonuçları.                                                              | 108 |
| 6.16: | Senaryo 4 sonuçları.                                                              | 108 |
| 6.17: | Senaryo 5 sonuçları.                                                              | 109 |
| 6.18: | Senaryo 6 sonuçları.                                                              | 109 |
| 6.19: | Senaryo 7 sonuçları.                                                              | 110 |
| 6.20: | Senaryo 8 sonuçları.                                                              | 110 |
| 6.21: | Senaryo 9 sonuçları.                                                              | 111 |
| 6.22: | Senaryo 10-15.                                                                    | 112 |
| 6.23: | Önerilen algoritma ve Büyük Patlama Büyük Çöküş algoritmasının<br>performansları. | 113 |
| 6.24: | Algoritma çalışma süreleri.                                                       | 113 |

# 1. GİRİŞ

Hedef takibi, kara, hava, sualtı ve su üstü hedeflerinin konum, hız ve ivme gibi hedef hareket dinamiklerinin elektro-optik, sonar ve radar gibi algılayıcılar yoluyla doğru olarak tahmin edilmesi işlemi olarak tanımlanır. Hedef izleme algoritmaları genel olarak hedefin davranışını doğru olarak tanımlayan bir ölçüm ve durum modeline ihtiyaç duyar. Hedef izlemede kullanılan algoritmaların en bilineni Kalman filtresidir. Ölçüm gürültüsünün Gauss dağılımına sahip olduğu doğrusal sistemlerin durum değişkenlerini kestirmek için Kalman filtresi sıklıkla kullanılır [1-3]. Kalman filtresi 1960 yılında Rudolf Kalman tarafından önerilmiştir [4].

Kalman filtresinin yalnızca doğrusal sistemler için ve ölçüm hatalarının Gauss dağılımlı olması durumunda optimal çözüm üretmesi kullanım alanını daraltmaktadır [1-3]. Doğrusal olmayan sistemlerde ve ölçüm ve süreç gürültülerinin Gauss dağılımına sahip olmadığı durumlarda Parçacık filtreleri yaygın olarak kullanılmaktadır. Parçacık filtreleri olasılık yoğunluğu temsillerine dayalı, örneklem tabanlı sıralı bir Monte Carlo yöntemidir [1].

Parçacık filtreleri algoritmalarında üç temel adım mevcuttur. Bu adımlar, sırasıyla parçacıklara başlangıç değerleri atanması, parçacıkların ağırlıkların güncellemesi ve yeniden örnekleme adımlarıdır [1]. Parçacıkların oluşturulması ve parçacıklara ait ağırlıklarının güncelleme işlemi önem yoğunluk fonksiyonundan yararlanılarak yapılır [1]. Zaman içinde bazı parçacıklarının ağırlıkları çok artmaktadır ve diğer parçacıklarının ağırlıkları ise çok küçük kalmaktadır. Bu durum, az sayıda örnek ile sonsal dağılım kestirilmesine neden olur. Bu problemin önüne geçmek amacıyla yeniden örnekleme algoritmaları kullanılmaktadır [5-9].

## 1.1. Tezin Amacı

Parçacık filtrelerinde hatayı azaltan en önemli yaklaşım, yeniden örnekleme yaklaşımıdır [1]. Yeniden örnekleme algoritmasında doğruluk, hız, işlemci gücü kullanımı gibi parametrelerde iyileştirme yapılması önem arz etmektedir. Yeniden örnekleme algoritmasında yapılan iyileştirmeler Parçacık filtrelerinin uygulandığı problemlerin daha yüksek performansla çözülmesine katkı sağlamaktadır. Parçacık filtrelerinin uygulandığı en önemli problemlerden biri de hedef izleme problemidir.

Hedef takibi problemi, hava savunma, balistik füze savunma, okyanus ve uydu gözleme, hava trafik kontrol sistemleri gibi pek çok alanda karşılaşılan bir konudur. Yeniden örnekleme algoritmasının iyileştirmesi ve hedef takibi probleminde kullanılmasının hem askeri ve hem de sivil alanlara katkısı vardır. Literatürdeki yeniden örnekleme algoritması incelendiğinde, parçacıkların ağırlıkları önemli bilgi olarak kullanılarak, algoritma üretme teknikleri ve optimizasyon algoritmaları yaklaşımları ile üretilmiş, bir yeniden örnekleme algoritmasına rastlanmamıştır. Bu çalışmanın amacı, bu tür yaklaşımlarla geliştirilmiş, özgün yeniden örnekleme algoritmalarını literatüre kazandırmaktır.

## 1.2. Tezin Katkısı

Literatürde, daha önceki var olan parçacık filtrelerinde kullanılan yeniden örnekleme algoritmaları detaylı bir şekilde incelenmiştir. Tez önerisinde, literatürde var olan Açgözlü algoritmalar, dinamik algoritmalar, LSTM derin öğrenme ağı vb. yaklaşımlar kullanılarak üretilen yeniden örnekleme algoritmalarına rastlanmamıştır. Tez çalışmasının dayandığı en önemli hipotez, parçacıkların ağırlıkları önemli bilgi olarak kullanılırsa ve literatürdeki bu yaklaşımlar uyarlanırsa daha iyi sonuçlar elde edilebilmektedir. Açgözlü algoritmalar, en önemli algoritma üretme yaklaşımlarından olan dinamik programlama yaklaşımı, derin öğrenme yaklaşımları yeniden örnekleme algoritması üzerine uygulanmıştır. Çalışmalar sonucunda literatürdeki en çok kullanılan problem üzerinde diğer algoritmalara göre daha iyi sonuçlar elde edilmiştir.

Algoritmaların performanslarını ölçmek için, bilgisayar simülasyonu yapılmıştır. Çalışma için öncelikle aşağıdaki çalışmalar tamamlanmıştır.

- Modele uygun veri üretilmiştir.
- Parçacık filtresi algoritmasının geliştirilmiştir.
- Literatürde var olan yeniden örnekleme algoritmaları gerçekleştirilmiştir.
- Açgözlü yaklaşım kullanılarak yeniden örnekleme algoritması geliştirilmiştir.
- Torba problemi çözülerek yeniden örnekleme algoritması geliştirilmiştir.
- LSTM derin öğrenme tabanlı yeniden örnekleme algoritması geliştirilmiştir.
- Hedef hareket analizi problemi için, arama, parçacık filtresi algoritması

tabanlı çözüm geliştirilmiştir.

- Doğrusal olmayan model, kartezyen hareket modeli, sürekli dönüş hareket modelinde hedef takibinde yeniden örnekleme algoritmasının performansları karşılaştırılmıştır.
- Önerilen algoritmalar, hız, doğruluk gibi performans parametreleri açısından karşılaştırılmıştır.

### 1.3. Tezin İçeriği

Tezin ikinci bölümünde, literatürdeki hedef takip algoritmaları, Parçacık filtreleri ve yeniden örnekleme algoritmaları anlatılmaktadır. Hedef takip algoritmalarında Alfa-Beta filtresi, Kalman filtresi, Genişletilmiş Kalman filtresi ve Unscented (Kokusuz) Kalman filtresi anlatılmaktadır. Önerilen yeniden örnekleme algoritmaları ile testlerde karşılaştırması yapılmış yeniden örnekleme algoritmaları anlatılmaktadır. Bu yeniden örnekleme algoritmaların sözde kodları da bu bölümde yer almaktadır.

Tezin üçüncü bölümünde, Açgözlü yaklaşım yeniden örnekleme algoritmasına uyarlanarak özgün bir algoritma geliştirilmiştir. Açgözlü algoritmalar, ilk en iyi arama algoritmalarıdır. Yeniden örnekleme algoritmasının giriş değeri parçacıkların ağırlıkları olmaktadır. Sistemden çıkartılacak parçacıkları belirlemek için eşik değeri belirlenmektedir. Bu yaklaşımda, parçacıkların ağırlıkları büyükten küçüğe sıralanmaktadır. Eşik değeri aşınca kadar, ağırlıkları büyük olandan küçüğe doğru parçacıklar eklenmektedir. Çoğaltılacak parçacıklar belirlendikten sonra ağırlıkları ve başlangıç parçacık sayısıysa orantılı olacak şekilde parçacıklar çoğaltılmaktadır. Geliştirilen algoritmanın doğrusal olmayan hareket modeli, kartezyen hareket modeli ve sürekli dönüş hareket modelindeki performansı elde edilmiştir ve diğer yeniden örnekleme algoritmalarıyla karşılaştırılmıştır.

Tezin dördüncü bölümünde, torba problemi çözümü, yeniden örnekleme algoritmasına uyarlanarak özgün bir algoritma geliştirilmiştir. Bu problemde, bir torbaya eşyalar konulmak isteniyor. Eşyaların ağırlıkları ve değerleri bilinmektedir. Torbanın ise bir ağırlık kısıtlaması varken hangi eşyaları seçmenin maksimum fayda sağlanacağı bulunmaktadır. Bu problemin dinamik programlama yaklaşımıyla çözümü yeniden örnekleme algoritmasına uygulanmıştır. Doğrusal olmayan hareket

modeli, kartezyen hareket modeli sürekli dönüş hareket modelindeki performansı elde edilmiştir ve diğer yeniden örnekleme algoritmalarıyla karşılaştırılmıştır.

Tezin beşinci bölümünde, LSTM derin öğrenme algoritmasının, diziden diziye problemleri çözme yaklaşımı kullanarak yeniden örnekleme algoritmaları geliştirilmiştir. Giriş dizileri parçacıkların ağırlıkları orantılı doğal sayı dizisidir. Çıkış dizileri ise, herhangi bir yeniden örnekleme algoritmasının sonucudur. LSTM derin öğrenme algoritması kullanılarak, bu giriş ve çıkış dizileriyle eğitimi yapıp model elde edilmiştir. Bu eğitilen model ile üretildiği algoritmalar ile performansı doğrusal olmayan hareket modelinde karşılaştırması yapılmıştır.

Tezin altıncı bölümünde, hedef hareket analizi çözümü için algoritma önerilmiştir. Hedef hareket analizinde, belirli süre hedeften kerteriz açıları elde edilip, mesafe, rota ve sürat kestirimi yapılmaktadır. Mesafe, rota, sürat ve başlangıç değerlerini bulmak için, hareket denklemleri ve kerteriz açıları kullanarak kaba kuvvet arama algoritması geliştirilmiştir. Parçacık filtresinin başlangıç konumları arama algoritmasına elde edilen değerler kullanılarak elde edilmektedir. Parçacık filtresi algoritması ve hareket denklemleri kullanarak, daha az gürültülü açı değerleri elde edilmektedir. Bu açı değerleri kullanılarak yeniden arama algoritması ile nihai mesafe, rota ve sürat kestirimi yapılabilmektedir.

Tezin yedinci bölümü sonuçlar ve gelecekte yapılabilecek çalışma önerileri ile tez sonlandırılmaktadır.

## 2. LİTERATÜR TARAMASI

Bu bölümde, literatürde yaygın olarak kullanılan hedef takibi algoritmaları ve parçacık filtresinin avantajları, tez çalışmasının ana hedefi olan yeniden örnekleme algoritmalarına neden ihtiyaç duyulduğu ve literatürde yaygın olarak kullanılan yeniden örnekleme algoritmalarının adımları ve çalışma prensipleri sayısal bir örnek üzerinden anlatılmaktadır.

### 2.1. Hedef Takip Algoritmaları

Literatürden en çok kullanılan takip algoritmaları, Alfa-Beta Filtresi, Kalman Filtreleri, Genişletilmiş Kalman Filtresi Unscented (Kokusuz) Kalman Filtresi, ve Parçacık Filtreleri şeklindedir [1-3,5].

#### 2.1.1. Alfa-Beta Filtresi

Sabit katsayılı iki boyutlu bir filtredir. Bu filtrede hedefin sabit hızla hareket ettiği sıfır ortalamalı beyaz Gauss gürültüsü ile ölçümlerin bozulduğu kabul edilmektedir. Filtrede kazanç değerinin sabit katsayılı olması uygulama kolaylığı getirmektedir. Bu filtre sadece konuma ait ölçümlerin alındığı durumda kullanılır.

Alfa-Beta filtresi ilk defa [11] makalede önerilmiştir. Alfa-Beta filtresinin genişletilmiş hali Alfa-Beta-Gamma filtresidir. Alfa-Beta-Gamma filtresi Alfa-Beta filtresinden farklı olarak, ivme tahmini de yapabilmektedir [12,13].

Alfa-Beta-Gamma filtresi ilk defa [14] makalede sunulmuştur. Kalman filtresinin basitleştirilmiş hali olarak düşünebilir. Basit ve hızlıdır [3]. Alfa-Beta filtresinin formülleri Eşitlik (2.1) ile (2.5) arasında verilmektedir. Aşağıdaki gibidir. Burada,  $x_{kp}$  gelecek konum değerini,  $x_k$  şimdiki konum değerini,  $dt$  zaman aralığı,  $v_{kp}$  gelecek hız değeri,  $v_k$  şimdiki hız değerini,  $r_k$  hatayı,  $x_m$  ölçülen konumu belirtir.  $\alpha$  ve  $\beta$  katsayıları probleme göre test edilerek hesaplanır.

$$x_{kp} = x_k + dt * vk \quad (2.1)$$

$$v_{kp} = v_k \quad (2.2)$$

$$x_{kp} = x_m - x_{kp} \quad (2.3)$$

$$x_{kp} = x_{kp} + \alpha * r_k \quad (2.4)$$

$$v_{kp} = v_{kp} + \frac{\beta}{dt} * r_k \quad (2.5)$$

### 2.1.2. Kalman Filtresi

1960 yılında Rudolf Kalman tarafından ortaya atılmıştır [4]. Kalman filtresinin en önemli özelliği modellenen sisteme ait geçmiş, şimdiki ve gelecekteki durumları kestirebilmesidir. Hatta bu özellik tam olarak modellenememiş sistemler için de geçerlidir. Kalman filtresinde sistemin doğrusal bir sistem olduğu, ölçümlerin ve gürültülerin Gauss dağılımına sahip olduğu varsayılır. Sadece Gauss dağılımına sahip modeller ile kullanılabilmesinden dolayı, Kalman filtresinin kullanım alanı çok sınırlıdır [2]. Kalman filtresi iki farklı denklem grubunu içermektedir. Bunlar zaman güncellemesi ve ölçüm güncellemesi denklemleridir. Zaman güncellemesinde, şimdiki durumdan bir sonraki durum tahmin edilir ve oluşan hatanın kovaryansı hesaplanır. Ölçüm güncellemesinde ise, Kalman kazancı hesaplanarak, son ölçüm değeri ile durum güncellemesi yapılmaktadır ve ardından hata kovaryansı güncellenir [2]. Eşitlik (2.6) ile Eşitlik (2.10) arası bu denklemler görülmektedir. Burada,  $x'_k$  tahmin adımındaki durum vektörünü,  $P'_k$  tahmin adımındaki kovaryans matrisini,  $x_k$  durum vektörünü,  $A$  durum değişim matrisini,  $B$  bozulma matrisini,  $u_k$  sıfır ortalamalı Gauss gürültüsünü,  $Q$  süreç gürültüsü varyansını,  $H$  ölçüm matrisini,  $P_k$  kovaryans matrisini,  $R$  ölçüm gürültüsü varyans matrisini,  $K_k$  kalman kazancını belirtir.

Kalman Filtresi Tahmin Adımı:

$$x'_k = Ax'_{k-1} + Bu_k \quad (2.6)$$

$$P'_k = AP_{k-1}A^T + Q \quad (2.7)$$

Kalman Filtresi Güncelleme Adımı:

$$K_k = P'_k H^T (HP'_k H^T + R)^{-1} \quad (2.8)$$

$$x_k = x'_k + K_k(z_k - Hx'_k) \quad (2.9)$$

$$P_k = (I - K_k H) P'_k \quad (2.10)$$

### 2.1.3. Genişletilmiş Kalman Filtresi

Gerçek dünyadaki çoğu süreç doğrusal değildir; dolayısıyla Kalman filtresiyle tahmin edilebilmeleri için doğrusal hale getirilmelidirler. Genişletilmiş Kalman Filtresi (EKF, Ing: Extended Kalman Filtresi) doğrusal olmayan problemleri durum ve ölçüm fonksiyonlarının tahminini durum denklemleri etrafında Jacobian matrisini hesaplayarak çözer. Bu matris durum denklemlerinin etrafına merkezlenen model fonksiyonun eğrisini verir. Jacobian matrisi bir vektörün bütün kısmi türevlerini içermektedir. EKF, doğrusal olmayan sistem fonksiyonlarının Taylor serisi açılımı ile doğrusal hale getirilmesi yaklaşımı ile Kalman filtrenin uygulanmasıdır [2].

### 2.1.4. Unscented (Kokusuz) Kalman Filtresi

Julier ve Uhlman 1997 yılında, , Kokusuz Kalman Filtresi (UKF, Ing: Unscented Kalman Filter) algoritmasını, GKF'ye alternatif olarak önermiştir [16]. UKF, dağılım fonksiyonlarının doğrusal olmayan dönüşümleri yerine, o dağılımları temsil eden örneklemelerin doğrusal olmayan dönüşümlerinin daha kolay yapılabileceği gerçeğinden esinlenilerek ortaya konmuştur [17,18]. Pasif dinleme ile doppler, yanca ve yanca hızı ölçümlerini yapma kabiliyetine sahip bir sistem için

hızlı ve yüksek doğrulukta çalışan UKF tabanlı hedef takip yaklaşımı önerilmiştir [19]. Önerilen takip algoritmasında, yinelemeli GKF algoritmasından yola çıkılarak UKF yinelemeli hale getirilmiştir [19].

Rastgele Gauss değişkenlerinin doğrusal bir fonksiyon üzerinde ilerletilmesi problemi unscented dönüşüm olarak isimlendirilen başka bir yaklaşımla ele alınabilir [19]. Bu dönüşüm, fonksiyonları doğrusal hale getirmek yerine belirli bir noktalar kümesini kullanır ve bu noktaları doğrusal olmayan fonksiyon üzerinde ilerletir. Bu noktalar, ortalama, kovaryans ve muhtemelen yüksek dereceli momentleri rastgele Gauss değişkeni ile eşleştirecek şekilde seçilir. Ortalama ve kovaryans, fonksiyon üzerindeki bu noktalardan geleneksel fonksiyon doğrusallaştırma yöntemine göre daha doğru sonuç verecek şekilde yeniden hesaplanabilir. Bu yaklaşımdaki temel fikir, fonksiyonun kendisinin yerine olasılık dağılım fonksiyonuna yaklaşım sağlamaktır. Bu yaklaşım GKF'ye göre, genel olarak hem daha doğru ve hızlı sonuçlar ortaya çıkarmakta ve hesaplama karmaşıklığını azaltarak, kestirim doğruluğunu artırmaktadır [19-20].

### **2.1.5. Parçacık filtreleri**

Parçacık filtreleri algoritmalarının hedef takibinde, doğrusal olmayan ve gürültünün Gauss dağılımı olmadığı durumlarda kullanabilmesi en önemli avantajıdır. Parçacık filtresi ile hedef takibi dört temel adımdan oluşmaktadır. Bu adımlar sırasıyla örnekleme adımı, ağırlıkları güncelleme adımı, yeniden örnekleme adımı ve kestirme adımıdır [1]. Hedefe ait konum ve hız bilgileri örnekleme adımında kestirilir. Kestirilen değerler her bir parçacık için yapılan kestirimler olup, hedefe ait en son kestirilen değerler değildir. Ağırlıkları güncelleme adımında, her bir parçacığın olabilirliklerine göre ağırlıklandırılarak, hedefin olası konumu kestirilmeye çalışılır. Ağırlıklandırma işleminde, ölçülen konum ile tahmin edilen konum değerlerinin farklarına bağlı olarak parçacıklar ağırlandırılır [1]. Tahmin değeri ile ölçüm değerinin yakın olduğu parçacıkların ağırlıkları yüksek olur. Düşük ağırlıklı parçacıkların ağırlıkları sıfıra yaklaşması etkin örnek sayısının azalmasına sebep olmaktadır. Etkin örnek sayısının azalması kestirim hatasının artmasına neden olmaktadır [1]. Bu problemin önüne geçmek amacıyla literatürde yeniden örnekleme algoritmaları kullanılmaktadır [1]. Yeniden örnekleme adımında daha düşük ağırlıklı

parçacıkların sistemden uzaklaştırılarak yerlerine yüksek ağırlıklı parçacıklar yerleştirilir. Böylece hedefin en son konum bilgisi daha doğru bir şekilde kestirilecektir. Son adım olan kestirme adımında ise, hedefe ait olan en son konum bilgisi elde etmek için, parçacıkların ağırlıkları ile kestirim yapılacak değerler çarpılır ve hedefin son konumu tahmin edilir [1].

Parçacık sayısı  $N$  olmak üzere ve parçacık filtresinin algoritmanın adımları aşağıda maddeler halinde verilmektedir.

Adım 1: Parçacıkların  $k - 1$  anında parçacıkların ağırlıkları  $w_1, w_2, \dots, w_N$  ve durum dizisinin değerleri  $x_1, x_2, \dots, x_N$  ve  $z_k$  konum değeri alınır.  $N$  tane parçacık için giriş değeri olarak alınır.

Adım 2: Durum denkleminin önsel olasılık dağılım fonksiyonu ve  $k - 1$  anında durum dizilerin değerleri kullanılarak parçacıkların  $k$  anındaki durum dizileri  $(x_k | x_k^{(i)}, z_k) \approx p(x_k | x_{k-1}^{(i)})$  şeklinde hesaplanır.

Adım 3: Önsel olasılık dağılım fonksiyonu ve  $k - 1$  anında parçacıkların ağırlıkları kullanılarak parçacıkların  $k$  anındaki ağırlıkları hesaplanır. Ve  $k$  anında ölçülen değeri kullanılarak ağırlıklar hesaplanır. Parçacıkların ağırlıkları  $W_k^{(i)} \approx w_k^{(i-1)} p(z_k | x_k^{(i)})$  şeklinde hesaplanabilir. Gerçek değer ile parçacığın hesapladığı değer yakınsa o parçacığın ağırlığı daha yüksektir.

Adım 4: Tüm parçacıkların ağırlıklarının toplamı hesaplanır.

Toplam ağırlık  $W_T = \sum_{i=1}^N w_i$  eşitliği ile hesaplanır.

Adım 5: Her bir parçacığın ağırlığı Adım 4'te elde edilen toplam ağırlığa bölünerek normalize ağırlıklar  $W_i = w_i / W_T$  bulunur.

Adım 6:  $N_{eff} = \frac{1}{\sum_{i=1}^N W_i^2}$  eşitliği hesaplanmaktadır.

Adım 7:  $N_t = \text{örnekleme\_oranı} * N$ .  $\text{örnekleme\_oranı}$  tipik olarak 0.5 olarak belirlenebilir.

Adım 8:  $N_{eff} < N_t$  ise yeniden örnekleme yapılır. Düşük ağırlıklı parçacıklar sistemden çıkarılarak yerine yüksek ağırlıklı parçacıklar çoğaltılır. Yeniden örnekleme yapıldığında  $w_1, w_2, \dots, w_N$  tüm parçacıkların ağırlıkları  $w_i = 1/N$  şeklinde olur.

Adım 9: Son adım kestirme adıımıdır.  $x_{est} = \sum_{i=1}^N x_i * w_i$  şeklinde hesaplanır.

Adım 10:  $k$  anında parçacıkların ağırlıkları  $w_1, w_2, \dots, w_N$  ve durum dizisinin değerleri  $x_1, x_2, \dots, x_N$  geri dönüş yapılarak algoritma sonlanır.

Parçacık sayısı  $N$  olmak üzere; Basit bir sayısal örnek üzerinden algoritmanın adımlarını açıklamak gerekirse;

Adım 1: Parçacıkların  $k - 1$  anında parçacıkların ağırlıkları  $w_1, w_2, \dots, w_N$  ve durum dizisinin değerleri  $x_1, x_2, \dots, x_N$  ölçülen değer 14.6

$$w_{k-1} = [0.1; 0.1; 0.1; 0.1; 0.1; 0.1; 0.1; 0.1; 0.1; 0.1],$$

$$x_{k-1} = [17.0; 17.0; 17.0; 17.0; 17.0; 17.0; 17.0; 17.0; 17.0; 17.0]$$

Adım 2: Önsel olasılık dağılım fonksiyonu ve  $k - 1$  anında durum dizilerinin değerleri kullanılarak parçacıkların  $k$  anındaki durum dizileri hesaplanır.

$$x_k = [15.4; 16.5; 15.6; 10.9; 15.5; 13.8; 13.4; 12.8; 14.8; 10.1]$$

Adım 3: Önsel olasılık dağılım fonksiyonu ve  $k - 1$  anında parçacıkların ağırlıkları kullanılarak parçacıkların  $k$  anındaki ağırlıkları hesaplanır.

$$w_k = [0.06; 0.18; 0.069; 0.0003; 0.064; 0.009; 0.005; 0.003; 0.027; 0.0001]$$

Adım 4: Tüm parçacıkların ağırlıklarının toplamı hesaplanır.

$$\text{Toplam ağırlık } W_T = \sum_{i=1}^N w_i$$

Adım 5: Her bir parçacığın ağırlığı Adım 4'te elde edilen toplam ağırlığa bölünerek normalize ağırlıklar  $W_i = w_i / W_T$  bulunur.

$$W_i = [0.145; 0.43; 0.165; 0.0008; 0.152; 0.022; 0.013; 0.007; 0.064; 0.0003]$$

$$\text{Adım 6: } N_{eff} = \frac{1}{\sum_{i=1}^N W_i^2}, N_{eff} = 3.80$$

$$\text{Adım 7: } N_t = 0.5 * 10 = 5$$

Adım 8:  $N_{eff} < N_t$  ise yeniden örnekleme yapılır. düşük ağırlıklı parçacıklar sistemden çıkarılarak yerine yüksek ağırlıklı parçacıklar çoğaltılır. yeniden örnekleme yapıldığında  $w_1, w_2, \dots, w_n$  tüm parçacıkların ağırlıkları  $w_i = 1/N$  şeklinde olur. Yeniden örnekleme adımı sonrasında geri dönüş yapılan indisler Geri dönüş indisleri = [ 2; 2; 2; 2; 2; 3; 3; 5; 5]

$$x_k = [16.5; 16.5; 16.5; 16.5; 16.5; 16.5; 15.6; 15.6; 15.5; 15.5]$$

$$w_k = [0.1; 0.1; 0.1; 0.1; 0.1; 0.1; 0.1; 0.1; 0.1; 0.1] \text{ olarak hesaplanır.}$$

Adım 9: Son adım kestirme adımdır.  $x_{est} = \sum_{i=1}^N x_i * w_i$  şeklinde hesaplanır.  $x_{est} = 16.1$

Adım 10:  $x_k = [16.5;16.5;16.5;16.5;16.5;16.5;15.6;15.6;15.5;15.5]$   
 $w_k=[0.1;0.1;0.1;0.1;0.1;0.1;0.1;0.1;0.1;0.1]$  şeklinde geri dönüş yapılır.

## 2.2. Yeniden Örnekleme Algoritmaları

Hedef takibinde Parçacık filtrelerinde yeniden örnekleme algoritması olarak bir çok algoritma mevcuttur. Literatürde en çok kullanılan dört algoritma Systematic, Stratified, MultiNominal, Residual, yeniden örnekleme algoritmalarıdır. Tez çalışmasında, performans karşılaştırması yapmak için bu algoritmalara ek olarak, son dönemde geliştirilen MSV (Ing: Minimum Sampling Variance, Minimum Örnekleme Varyansı) algoritması kullanılmıştır. Bu algoritmalar ilgili detaylar [21-25]'de bulunmaktadır.

### 2.2.1. Systematic Yeniden Örnekleme Algoritması

İlk defa [23] makalede sunulmuştur. Öncelik birikimli dağılım fonksiyonu  $Q$  hesaplanmaktadır. Sistematik yeniden örnekleme algoritmasında, parçacık sayısı  $N$  olmak üzere; 0 ile 1 arasında üretilen rastgele sayının  $1/N$ 'i kadar rastgele sayı ile toplanmaktadır. Bu fonksiyonu  $T$  fonksiyonu olarak adlandırılabilir.  $T$  fonksiyondaki elemanların  $Q$  fonksiyonundaki elemanlardan ilk küçük olduğu elemanın indis değerini bulunur.  $U$  birim dağılımlı rastgele bir gürültü fonksiyonu belirtir.  $U[0,1)$  fonksiyonu 0-1 arasında eşit olasılıklı rastgele sayı üreten fonksiyondur.

Algoritma adımları aşağıda maddeler halinde sıralanmaktadır:

Adım 1: Parçacıkların ağırlıkları  $w_1, w_2, \dots, w_N$   $N$  tane parçacık için giriş değeri olarak alınır.

Adım 2: Tüm parçacıkların ağırlıklarının toplamı hesaplanır.

Toplam ağırlık  $W_T = \sum_{i=1}^N w_i$

Adım 3: Her bir parçacığın ağırlığı Adım 3'te elde edilen toplam ağırlığa bölünerek normalize ağırlıklar  $W_i = w_i/W_T$  bulunur.

Adım 4:  $Q_k = \sum_{i=1}^k W_i$   $k = 1, 2, \dots, N$  kümülatif fonksiyon hesaplanıyor

Adım 5: 0 ile 1 arasında  $1/N$  eğimli ve 0 ile 1 arasında üretilen rastgele sayının  $1/N$ 'i kadar sabit sayı eklenir. Bu fonksiyonu  $T$  fonksiyonu olarak adlandırabiliriz.

$$T_i = \sum_{i=0}^{N-1} \frac{i}{N} + \left( \frac{1}{N} * U[0,1) \right)$$

Adım 6: Algoritmanın geri dönüş yapılacak indisler için hafızada yer ayrılır. İndisler,  $\delta_1, \delta_2, \dots, \delta_n$  şeklinde ifade edilir.

Adım 7:  $\delta_k = \text{argmin } T_k < Q_i \quad k = 1, 2, \dots, N$ .  $T$  fonksiyondaki elemanların  $Q$  fonksiyonundaki elemanlardan ilk küçük olduğu elemanın indis değerini bulunur ve geri dönüş yapılacak indis değerine atanır.

Bu algoritmayı, basit bir sayısal örnekle açıklayalım. Parçacıkların ağırlıkları [1;2;3;8;16] olsun.

Adım 1:  $\{1, 2, 3, 8, 16\}$

Adım 2:  $W_T = \sum_{i=1}^N w_i = 1+2+3+8+16=30$

Adım 3:  $W_i = w_i/W_T$  şeklinde hesaplanır.

$W_1 = 1/30; W_2 = 2/30; W_3 = 3/30;$

$W_4 = 8/30; W_5 = 16/30$  şeklinde hesaplanır.

Adım 4:  $Q_k = \sum_{i=1}^k W_i$  şeklinde hesaplanır.

$Q_1 = 1/30; Q_2 = 3/30; Q_3 = 6/30; Q_4 = 14/30; Q_5 = 1.00$  şeklinde hesaplanır.

Adım 5:  $U[0,1) = 0.60$  olmak üzere,  $i = 0, 1, \dots, N - 1$

$T_{i+1} = \frac{i}{N} + \left( \frac{1}{N} * U[0,1) \right)$  olmak üzere ,

$T_1 = \frac{0}{5} + \left( \frac{1}{5} * 0.60 \right), T_2 = \frac{1}{5} + \left( \frac{1}{5} * 0.60 \right), T_3 = \frac{2}{5} + \left( \frac{1}{5} * 0.60 \right), T_4 = \frac{3}{5} + \left( \frac{1}{5} * 0.60 \right), T_5 = \frac{4}{5} + \left( \frac{1}{5} * 0.60 \right)$  olmak üzere,

$T_1 = 0.12, T_2 = 0.32, T_3 = 0.52, T_4 = 0.72, T_5 = 0.92$

Adım 6: Algoritmanın geri dönüş yapılacak indisler için hafızada yer ayrılır.  $\delta_1, \delta_2, \dots, \delta_N$

Adım 7:  $\delta_k = \text{argmin } T_k < W_i$

$\delta_1 = 3$ . Çünkü  $0.12 \geq 0.10$  ve  $0.12 < 0.23$

$\delta_2 = 4$ . Çünkü  $0.32 \geq 0.23$  ve  $0.32 < 0.50$

$\delta_3 = 5$ . Çünkü  $0.52 \geq 0.50$  ve  $0.52 < 1.00$

$\delta_4 = 5$ . Çünkü  $0.72 \geq 0.50$  ve  $0.72 < 1.00$

$\delta_5 = 5$ . Çünkü  $0.92 \geq 0.50$  ve  $0.92 < 1.00$

Bu durumda, Systematic yeniden örnekleme algoritması [3;4;5;5;5] şeklinde indisleri geri döner.

### 2.2.2. Stratified Yeniden Örnekleme Algoritması

Stratified algoritması da Systematic yeniden örnekleme algoritmasına benzer, Ancak,  $N$  parçacık sayısını belirtmek üzere, 0 ile 1 arasında üretilen rastgele sayının  $1/N$ 'i kadar rastgele sayı ile toplamak yerine,  $N$  tane 0 ile 1 arasında üretilen  $N$  tane farklı sayı eklenir. İlk defa [22] makalede önerilmiştir.

Algoritma adımları aşağıda maddeler halinde sıralanmaktadır:

Adım 1: Parçacıkların ağırlıkları  $w_1, w_2, \dots, w_N$   $N$  tane parçacık için giriş değeri olarak alınır.

Adım 2: Tüm parçacıkların ağırlıklarının toplamı hesaplanır.

$$\text{Toplam ağırlık } W_T = \sum_{i=1}^N w_i$$

Adım 3: Her bir parçacığın ağırlığı Adım 3'te elde edilen toplam ağırlığa bölünerek normalize ağırlıklar  $W_i = w_i/W_T$  bulunur.

Adım 4:  $Q_k = \sum_{i=1}^k W_i$   $k = 1, 2, \dots, N$  kümülatif fonksiyon hesaplanıyor.

Adım 5: 0 ile 1 arasında  $1/N$  eğimli ve 0 ile 1 arasında üretilen  $N$  adet rastgele sayı eklenir. Bu fonksiyonu  $T$  fonksiyonu olarak adlandırabiliriz.  $T_i = \sum_{i=0}^{N-1} \frac{1}{N} + \left( \frac{1}{N} * U_i[0,1) \right)$

Adım 6: Algoritmanın geri dönüş yapılacak indisler için hafızada yer ayrılır. İndisler,  $\delta_1, \delta_2, \dots, \delta_n$  şeklinde ifade edilir.

Adım 7:  $\delta_k = \text{argmin } T_k < Q_i$   $k = 1, 2, \dots, N$ .  $T$  fonksiyondaki elemanların  $Q$  fonksiyonundaki elemanlardan ilk küçük olduğu elemanın indis değerini bulunur ve geri dönüş yapılacak indis değerine atanır.

Bu algoritmayı, Basit bir sayısal örnekle açıklayalım. Parçacıkların ağırlıkları[1;2;3;8;16] olsun.

Adım 1:  $\{1, 2, 3, 8, 16\}$

Adım 2:  $W_T = \sum_{i=1}^N w_i = 1+2+3+8+16=30$

Adım 3:  $W_i = w_i/W_T$  şeklinde hesaplanır.

$$W_1 = 1/30; W_2 = 2/30; W_3 = 3/30;$$

$$W_4 = 8/30; W_5 = 16/30 \text{ şeklinde hesaplanır.}$$

Adım 4:  $Q_k = \sum_{i=1}^k W_i$  şeklinde hesaplanır.

$$Q_1 = 1/30; Q_2 = 3/30; Q_3 = 6/30;$$

$$Q_4 = 14/30; Q_5 = 1.00 \text{ şeklinde hesaplanır.}$$

Adım 5: Üretilen N farklı sayımız  $[0.2; 0.4; 0.6; 0.8; 0.9]$  olsun.,  $i = 0, 1, \dots, N - 1$

$$T_i = \sum_{j=0}^{N-1} \frac{1}{N} + \left( \frac{1}{N} * U_i[0,1) \right) \text{ olmak üzere ,}$$

$$T_1 = \frac{0}{5} + \left( \frac{1}{5} * 0.20 \right), T_2 = \frac{1}{5} + \left( \frac{1}{5} * 0.40 \right), T_3 = \frac{2}{5} + \left( \frac{1}{5} * 0.60 \right)$$

$$T_4 = \frac{3}{5} + \left( \frac{1}{5} * 0.80 \right), T_5 = \frac{4}{5} + \left( \frac{1}{5} * 0.90 \right)$$

olmak üzere  $T_1 = 0.04, T_2 = 0.28, T_3 = 0.52, T_4 = 0.76, T_5 = 0.98$  elde edilir.

Adım 6: Algoritmanın geri dönüş yapılacak indisler için hafızada yer ayrılır.

$$\delta_1, \delta_2, \dots, \delta_N$$

Adım 7:  $\delta_k = \operatorname{argmin} T_k < W_i$

$$\delta_1 = 2. \text{ Çünkü } 0.04 \geq 0.033 \text{ ve } 0.04 < 0.10$$

$$\delta_2 = 4. \text{ Çünkü } 0.28 \geq 0.23 \text{ ve } 0.28 < 0.50$$

$$\delta_3 = 5. \text{ Çünkü } 0.52 \geq 0.50 \text{ ve } 0.52 < 1.00$$

$$\delta_4 = 5. \text{ Çünkü } 0.76 \geq 0.50 \text{ ve } 0.76 < 1.00$$

$$\delta_5 = 5. \text{ Çünkü } 0.98 \geq 0.50 \text{ ve } 0.98 < 1.00$$

Bu durumda Stratified yeniden örnekleme algoritması  $[2;4;5;5;5]$  şeklinde indisleri geri döner.

### 2.2.3. MultiNominal Yeniden Örnekleme Algoritması

İlk defa [21] makalede sunulmuştur. Öncelikle birikimli dağılım (Cumulative) fonksiyon hesaplanır. Bu fonksiyonu  $Q$  fonksiyonu olarak adlandırabiliriz. 0 ile 1 arasında  $N$  tane rastgele sayı üretilir. Bu fonksiyonu  $T$  fonksiyonu olarak adlandırabiliriz. Algoritmanın geri dönüş yapılacak indisler için hafızada yer ayrılır.  $T$  fonksiyondaki elemanların  $Q$  fonksiyonundaki elemanlardan ilk küçük olduğu elemanın indis değerini bulunur.

Parçacık sayısı  $N$  olmak üzere; Algoritma adımları aşağıda maddeler halinde sıralanmaktadır:

Adım 1: Parçacıkların ağırlıkları  $w_1, w_2, \dots, w_N$   $N$  tane parçacık için giriş değeri olarak alınır.

Adım 2: Tüm parçacıkların ağırlıklarının toplamı hesaplanır.

$$\text{Toplam ağırlık } W_T = \sum_{i=1}^N w_i$$

Adım 3: Her bir parçacığın ağırlığı Adım 3'te elde edilen toplam ağırlığa bölünerek normalize ağırlıklar  $W_i = w_i/W_T$  bulunur.

Adım 4:  $Q_k = \sum_{i=1}^k W_i$   $k = 1, 2, \dots, N$  kümülatif fonksiyon hesaplanıyor.

Adım 5: 0 ile 1 arasında üretilen  $N$  adet rastgele sayı üretilir. Bu fonksiyonu  $T$  fonksiyonu olarak adlandırabiliriz.  $T_i = U_i[0,1)$

Adım 6: Algoritmanın geri dönüş yapılacak indisler için hafızada yer ayrılır.  $\delta_1,$

$$\delta_2, \dots, \delta_n$$

Adım 7:  $\delta_k = \text{argmin } T_k < Q_i$   $k = 1, 2, \dots, N$ .  $T$  fonksiyondaki elemanların  $Q$  fonksiyonundaki elemanlardan ilk küçük olduğu elemanın indis değerini bulunur ve geri dönüş yapılacak indis değerine atanır.

Bu algoritmayı, Basit bir sayısal örnekle açıklayalım. Parçacıkların ağırlıkları  $[1;2;3;8;16]$  olsun.

Adım 1:  $\{1, 2, 3, 8, 16\}$

Adım 2:  $W_T = \sum_{i=1}^N w_i = 1+2+3+8+16=30$

Adım 3:  $W_i = w_i/W_T$  şeklinde hesaplanır.

$$W_1 = 1/30; W_2 = 2/30; W_3 = 3/30;$$

$$W_4 = 8/30; W_5 = 16/30 \text{ şeklinde hesaplanır.}$$

Adım 4:  $Q_k = \sum_{i=1}^k W_i$  şeklinde hesaplanır.

$$Q_1 = 1/30; Q_2 = 3/30; Q_3 = 6/30;$$

$$Q_4 = 14/30; Q_5 = 1.00 \text{ şeklinde hesaplanır.}$$

Adım 5: Üretilen  $N$  farklı sayımız  $[0.2; 0.4; 0.6; 0.8; 0.9]$  şeklinde olsun.

$$T_1 = 0.2, T_2 = 0.40, T_3 = 0.60, T_4 = 0.80, T_5 = 0.90$$

Algoritmanın geri dönüş yapılacak indisler için hafızada yer ayrılır. İndisler,  $\delta_1, \delta_2, \dots, \delta_n$  şeklinde ifade edilir.

Adım 6:  $\delta_k = \operatorname{argmin} T_k < W_i$

$$\delta_1 = 3. \text{ Çünkü } 0.20 \geq 0.10 \text{ ve } 0.20 < 0.23$$

$$\delta_2 = 4. \text{ Çünkü } 0.40 \geq 0.23 \text{ ve } 0.40 < 0.50$$

$$\delta_3 = 5. \text{ Çünkü } 0.60 \geq 0.50 \text{ ve } 0.60 < 1.00$$

$$\delta_4 = 5. \text{ Çünkü } 0.80 \geq 0.50 \text{ ve } 0.80 < 1.00$$

$$\delta_5 = 5. \text{ Çünkü } 0.90 \geq 0.50 \text{ ve } 0.90 < 1.00$$

Bu durumda algoritma  $[3;4;5;5;5]$  şeklinde indisleri geri döner.

## 2.2.4. Residual Yeniden Örnekleme Algoritması

İlk defa [24] makalede önerilmiştir. Parçacık sayısı  $N$  olmak üzere, algoritmanın adımları aşağıda maddeler halinde sıralanmaktadır.

- Her bir parçacığının ağırlıklarına göre olasılık değeri hesaplanır.
- Bu olasılık değerleri, parçacık sayısı ile çarpılır.
- Elde edilen değer en yakın ve daha küçük olan tam sayıya yuvarlanır. Bu işlemin sonucunda elde edilen sonuç, hangi indisli elemanda kaç adet çoğaltılacağını ifade eder. Algoritmanın ilk aşaması tamamlanır.
- İlk aşama sonucunda, en yakın ve daha küçük sayıya yuvarlandığı için, başlangıçtaki parçacık sayısından daha az sayıda parçacık oluşmuştur.

Başlangıçtaki parçacık sayısı ile algoritmanın ilk aşaması sonucunda elde edilen toplam parçacık sayısı farkı bulunur.

- Bulunan fark sayısı kadar, MultiNominal yeniden örnekleme algoritmasıyla yeniden örneklenecek parçacıkların indisleri hesaplanır.

Parçacık sayısı  $N$  olmak üzere; Algoritma adımları aşağıda maddeler halinde sıralanmaktadır:

Adım 1: Parçacıkların ağırlıkları  $w_1, w_2, \dots, w_N$   $N$  tane parçacık için giriş değeri olarak alınır.

Adım 2: Tüm parçacıkların ağırlıklarının toplamı hesaplanır.

$$\text{Toplam ağırlık } W_T = \sum_{i=1}^N w_i$$

Adım 3: Her bir parçacığın ağırlığı Adım 3'te elde edilen toplam ağırlığa bölünerek normalize ağırlıklar  $W_i = w_i / W_T$  bulunur.

$$Q_k = \sum_{i=1}^k W_i \quad k = 1, 2, \dots, N \text{ kümülatif fonksiyon hesaplanıyor.}$$

Adım 4:  $k_i = \text{floor}[N * w_i]$ , Bu olasılık değerleri, parçacık sayısı ile çarpılır. Elde edilen değer en yakın ve daha küçük olan tam sayıya yuvarlanır.  $n_s = \{k_1, k_2, \dots, k_n\}$  Bu işleminin sonucunda elde edilen sonuç, hangi indisli elemanda kaç adet çoğaltılacağını  $n_s$  çoğaltma dizisi ifade eder.

Adım 5:  $n_t = \sum_{s=1}^N n_s$  ve  $\text{remain} = N - n_t$  en yakın ve daha küçük sayıya yuvarlandığı için, başlangıçtaki parçacık sayısından daha az sayıda parçacık oluşmuştur. Başlangıçtaki parçacık sayısı ile algoritmanın ilk aşaması sonucunda elde edilen toplam parçacık sayısı farkı bulunur.  $\text{remain}$  hesaplanır.

$$\text{Adım 6: } R_t = \frac{N * W - n_s}{\text{remain}} \text{ hesaplanır.}$$

Adım 7: 6. Adımda hesaplanan  $R_t$  fonksiyonun birikimli dağılım fonksiyonu hesaplanır.  $Q(R_t)$  hesaplanır.

Adım 8: 0 ile 1 arasında üretilen  $\text{remain}$  adet rastgele sayı üretilir. Bu fonksiyonu  $T$  fonksiyonu olarak adlandırabiliriz.  $T_i = U_i[0,1]$

Adım 9: Algoritmanın geri dönüş yapılacak indisler için hafızada yer ayrılır.

$$\delta_1, \delta_2, \dots, \delta_{\text{remain}}$$

Adım 10:  $\delta_k = \text{argmin } T_k < Q(R_t)_i$  geriye kalan indis değerleri hesaplanır.

Bu algoritmayı, Basit bir sayısal örnekle açıklayalım. Parçacıkların ağırlıkları [1,2,3,8,16] olsun.

Adım 1:  $\{1, 2, 3, 8, 16\}$

Adım 2:  $W_T = \sum_{i=1}^N w_i = 1+2+3+8+16=30$

Adım 3:  $W_i = w_i/W_T$  şeklinde hesaplanır.

$W_1 = 1/30; W_2 = 2/30; W_3 = 3/30;$

$W_4 = 8/30; W_5 = 16/30$  şeklinde hesaplanır.

Adım 4:  $k_i = \text{floor}[N * w_i],$

$k_1 = \text{floor}[5/30], ; k_2 = \text{floor}[10/30], k_3 = \text{floor}[15/30], ;$

$k_4 = \text{floor}[40/30], ; k_5 = \text{floor}[80/30]$  şeklinde hesaplanır.

$$n_s = \{k_1, k_2, k_3, k_4, k_5\}$$

$$n_s = \{0,0,0,1,2\} \text{ ve } n_t = 1 + 2 = 3$$

$$\text{remain} = N - n_t = 5 - 3 = 2$$

Adım 5:  $N * W - n_s = [0.17; 0.33; 0.67; 1.33; 2.50] - [0; 0; 0; 1; 2] = [0.17; 0.33; 0.67; 0.33; 0.50]$  olur.

Adım 6:  $R_t = \frac{N*W-n_s}{\text{remain}} = [0.083; 0.17; 0.33; 0.17; 0.25]$

Adım 7:  $Q(R_t) = [0.083; 0.25; 0.58; 0.75; 1.0]$  birikimli dağılım fonksiyonu hesaplanır.

Adım 8: Üretilen 2 sayımız  $[0.2; 0.8]$  şeklinde olsun.  $T_1 = 0.2, T_2 = 0.80,$

Adım 9: Algoritmanın geri dönüş yapılacak indisler için hafızada yer ayrılır. İndisler,  $\delta_1, \delta_2, \dots, \delta_n$  şeklinde ifade edilir.

Adım 10:  $\delta_k = \text{argmin } T_k < Q(R_t)_i$

$\delta_1 = 2.$  Çünkü  $0.20 \geq 0.083$  ve  $0.20 < 0.25$

$\delta_2 = 5.$  Çünkü  $0.80 \geq 0.75$  ve  $0.80 < 1.00$

Geriye kalan indis değerleri [2;5] şeklindedir. Daha önceki indis değerleriyle beraber Residual yeniden örnekleme algoritması [4;5;5;2;5] şeklinde geri döner. Eğer bu algoritmanın, ikinci aşamasında ‘multinomial’ algoritması yerine, stratified yeniden örnekleme algoritması kullanılmış olsaydı bu algoritma literatürde Residual Stratified yeniden örnekleme algoritması olarak adlandırılmaktadır.

### 2.2.5. Minimum Örneklem Varyansı Algoritması

Bu algoritma Li tarafından 2015 yılında sunulmuştur [25]. Bu metot örnekleme öncesi ve sonrası eşlenik dağılımı elde etmek üzere örnekleme sapmasını minimum olmasını sağlamaktadır. MSV (Ing: Minimum Sampling Variance, Minimum Örneklem Varyansı) yeniden örnekleme algoritması iki parçadan oluşmaktadır.

Parçacık sayısı  $N$  olmak üzere; Algoritma adımları aşağıda maddeler halinde sıralanmaktadır:

Adım 1: Parçacıkların ağırlıkları  $w_1, w_2, \dots, w_N$   $N$  tane parçacık için giriş değeri olarak alınır.

Adım 2: Tüm parçacıkların ağırlıklarının toplamı hesaplanır.

Toplam ağırlık  $W_T = \sum_{i=1}^N w_i$

Adım 3: Her bir parçacığın ağırlığı Adım 2’de elde edilen toplam ağırlığa bölünerek normalize ağırlıklar  $W_i = w_i/W_T$  bulunur.

$Q_k = \sum_{i=1}^k W_i$   $k = 1, 2, \dots, N$  kümülatif fonksiyon hesaplanıyor.

Adım 4:  $k_i = \text{floor}[N * W_i]$ , Bu olasılık değerleri, parçacık sayısı ile çarpılır. Elde edilen değer en yakın ve daha küçük olan tam sayıya yuvarlanır.  $n_s = \{k_1, k_2, \dots, k_n\}$  Bu işleminin sonucunda elde edilen sonuç, hangi indisli elemanda kaç adet çoğaltılacağını  $n_s$  çoğaltma dizisi ifade eder.

Adım 5:  $[N * W_i] - n_s$  hesaplanır.

Adım 6: 5.adımda hesaplanan dizi sıralanır.  $\text{sort}([N * W_i] - n_s)$  hesaplanır.  $[N * W_i] - n_s$   $D$  olarak sıralanmış indis değerleri bulunur.  $D_{\delta_1} \leq D_{\delta_2} \leq \dots, D_{\delta_N}$ .  $\delta_i$  fonksiyonu sıralama indisini ifade eder.

Adım 7:  $j = 1, 2, \dots, N$  ve  $\text{count} = 1$  olmak üzere,  $\text{count} \leq n_s \{j\}$  olduğu tüm durumlarda  $D_{\delta_i} = j$  atama yapılır. Bu durumda,  $i = i + 1$  ve  $\text{count} = \text{count} + 1$  yapılarak sıralanmış indis değerleri güncellenir. Bu indis değerleri geri dönülerek algoritma sonlanır.

Parçacık sayısı  $N$  olmak üzere; algoritma adımlarını basit bir örnek üzerinden verelim.

Adım 1:  $\{1, 2, 3, 8, 16\}$

Adım 2:  $W_T = \sum_{i=1}^N w_i = 1+2+3+8+16=30$

Adım 3:  $W_i = w_i/W_T$  şeklinde hesaplanır.

$$W_1 = 1/30; W_2 = 2/30; W_3 = 3/30;$$

$$W_4 = 8/30; W_5 = 16/30 \text{ şeklinde hesaplanır.}$$

Adım 4:  $k_i = \text{floor}[N * W_i]$ ,

$$k_1 = \text{floor}[5/30], ; k_2 = \text{floor}[10/30], k_3 = \text{floor}[15/30],$$

$$k_4 = \text{floor}[40/30], ; k_5 = \text{floor}[80/30] \text{ şeklinde hesaplanır.}$$

$$n_s = \{k_1, k_2, k_3, k_4, k_5\}$$

$$n_s = \{0,0,0,1,2\}$$

$$\text{Adım 5: } N * W - n_s = [0.17; 0.33; 0.50; 1.33; 2.67] - [0; 0; 0; 1; 2] = [0.17; 0.33; 0.50; 0.33; 0.67]$$

Adım 6:  $\text{sort}([N * W_i] - n_s)$  hesaplanır.  $D_{\delta_1} \leq D_{\delta_2} \leq \dots, D_{\delta_N}$

$$0.17 \leq 0.33 \leq 0.33 \leq 0.50 \leq 0.67 \quad [1; 4; 2; 3; 5] \text{ şeklindedir.}$$

Adım 7:  $j = 1, 2, \dots, N$  ve  $\text{count} = 1$  olmak üzere,  $\text{count} \leq n_s\{j\}$

$$n_s\{4\} = 1, n_s\{5\} = 2 \text{ } D_1 \text{ için, } j = 4 \text{ için, } 1 \leq n_s\{4\} = 1 \text{ olduğundan, } D_1 = 4$$

$$D = [4; 4; 2; 3; 5] \text{ şeklinde olur. } \text{count} = 2 \text{ olur.}$$

$$D_2 \text{ için, } j = 5 \text{ için, } 2 \leq n_s\{5\} = 2 \text{ olduğundan } D_2 = 5 \text{ olur.}$$

$$[4; 5; 2; 3; 5] \text{ şeklinde olur. } \text{count} = 1 \text{ olur.}$$

$$D_3 \text{ için, } j = 5 \text{ için, } 2 \leq n_s\{5\} = 2 \text{ } D_3 = 5 \text{ olur. } [4; 5; 5; 3; 5] \text{ şeklinde olur.}$$

$$D_4 \text{ ve } D_5 \text{ değışiklik olmaz. } [4; 5; 5; 3; 5] \text{ şeklinde algoritma sonlanır.}$$

### 3. AÇGÖZLÜ YAKLAŞIMLA YENİDEN ÖRNEKLEME ALGORİTMASI

Bu bölümde, algoritma üretme yaklaşımlarından olan Açgözlü (Ing: Greedy) yaklaşımın yeniden örnekleme algoritması üzerine uyarlanması anlatılmaktadır. Açgözlü yaklaşım, ilk en iyi arama algoritmasıdır. Bu yaklaşımı yeniden örnekleme algoritmasına uyarlamak için, 0-1 arasında bir eşik değeri belirlenir. Parçacıkların normalize edilmiş ağırlıkları büyükten küçüğe doğru sıralanır. Daha sonra ise eşik değeri aşmıyaya kadar parçacıklar eklenir. Böylece, eşik değerini aşamayan düşük ağırlıklı parçacıklar sistemden çıkartılmış olur. Onların yerine yüksek ağırlıklı parçacıklar geçmektedir. Önerilen algoritmayla, farklı hareket modellerinde elde edilen sonuçlar ikinci bölümde detayları verilen diğer yeniden örnekleme algoritmalarıyla karşılaştırılmıştır. Doğrusal olmayan hareket modeli, kartezyen hareket modeli ve dairesel hareket modelinde en iyi performansı Açgözlü yeniden örnekleme algoritması vermiştir.

#### 3.1. Açgözlü Yaklaşım

Algoritma üretme yaklaşımlarından birisi olan Açgözlü yaklaşımına göre mümkün olan ve sonuca en yakın olacak seçim yapılmaktadır. Basit bir seçim yapılması gerektiğinde sonuca en çok yaklaştıracak olan seçimin yapılması önerilmektedir [26]. Ancak, bu seçim her zaman için en iyi seçim olmayabilir. Açgözlü yaklaşımı kullanılarak üretilen farklı problemler üzerinde kullanılan algoritmalar mevcuttur. Asgari tarama ağaçları kullanılan Kruskal algoritması [27,28], Prim algoritması [27,29], en kısa yol bulma problem için kullanılan Dijkstra algoritması [30], veri sıkıştırımda kullanılan Huffman Kodlama algoritması [31] örnek olarak verilebilir. Açgözlü algoritmalar ilk en iyi arama algoritması olarak adlandırılır. Bu kapsamda, sezgisel arama problemlerinde kullanılırlar [32,33]. Atama [34], iş zamanlama [35] ve iş seçme gibi [36] optimizasyon problemlerinde de kullanılırlar. Öbek tabanlı makine çevirisinde, eksponansiyel zamandaki aramayı polinomsal zamana dönüştürerek kullanılabilirler [37]. Torba problemini çözmek için Açgözlü yaklaşımda kullanılabilir [38]. Tablo 3.1 diziden diziye problem için Açgözlü yaklaşım çözücü oluşturulmuştur. Bu problem için, eğer kaba kuvvet yaklaşımla arama yapılmış olsaydı, en yüksek olasılığı geri dönen kelime

dizisi bulmak için  $5^{10}$  karşılaştırma yapılmalıdır. Ancak Açgözlü yaklaşımla her bir 5 kelimenin en yüksek olasılığı için 5 karşılaştırma yapılır. Toplam kelime dizisinin uzunluğu 10 olduğundan  $5 \times 10$  işlem yapılır. Arama problemi eksponansiyel zamandan polinomsal zamana çözülmektedir. En iyi dizi  $\{1,1,1,1,1,5,5,5,5,5\}$  olarak geri dönmektedir.

Tablo 3.1: Diziden diziye problem için Açgözlü yaklaşım çözücüsü.

| Olasılık<br>Kelime | Ola.<br>1  | Ola.<br>2  | Ola.<br>3  | Ola.<br>4  | Ola.<br>5  | Ola.<br>6  | Ola.<br>7  | Ola.<br>8  | Ola.<br>9  | Ola.<br>10 |
|--------------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|
| 1                  | <b>0.5</b> | <b>0.5</b> | <b>0.5</b> | <b>0.5</b> | <b>0.5</b> | 0.1        | 0.1        | 0.1        | 0.1        | 0.1        |
| 2                  | 0.2        | 0.2        | 0.2        | 0.2        | 0.2        | 0.2        | 0.2        | 0.2        | 0.2        | 0.2        |
| 3                  | 0.1        | 0.1        | 0.1        | 0.1        | 0.1        | 0.1        | 0.1        | 0.1        | 0.1        | 0.1        |
| 4                  | 0.1        | 0.1        | 0.1        | 0.1        | 0.1        | 0.1        | 0.1        | 0.1        | 0.1        | 0.1        |
| 5                  | 0.1        | 0.1        | 0.1        | 0.1        | 0.1        | <b>0.5</b> | <b>0.5</b> | <b>0.5</b> | <b>0.5</b> | <b>0.5</b> |

Örneğin, literatürde sık olarak ele alınan para üzeri verilmesi probleminin Açgözlü yaklaşım kullanılarak çözülmesini düşünelim. Bu problemde bir satıcı kendisinden alışveriş yapan kişiye para üzeri verirken en az sayıda para vererek bunu yapması gerekmektedir [26]. Örneğin, ödenmesi gereken miktar 40 birim olsun. Ödeme için kullanılacak paraların birimleri ise 30, 15, 5 ve 1 olsun. Açgözlü yaklaşımına göre, satıcı 40 birim para üstünü tamamlamak için elindeki paralardan sonuca en çok yaklaştıran ve en büyük para olan 30 birimlik parayı en önce seçecektir. Geriye kalan 10 birimlik parayı elindeki en iyi alternatif olan 2 tane 5'lik birimle dolduracaktır. Böylece, toplam 3 adet parayı müşteriye verecektir.

### 3.2. Açgözlü Yeniden Örneklem Algoritması

Bu bölümde, Açgözlü yaklaşım kullanılarak geliştirilen özgün yeniden örneklem algoritması hakkında bilgi verilmektedir. Yeniden örneklem algoritmasında, zayıf olan parçacıkların sistemden çıkarılması ve ağırlıkları yüksek olan parçacıkların çoğaltılması gerekir.

Bu bölümde, Açgözlü yaklaşım, yeniden örnekleme algoritmasına uyarlanmaktadır. Bazı parçacıkları sistemden çıkarmak için, belli bir eşik değeri belirlenmelidir. Parçacıkların ağırlıkları 0-1 arasında normalize edildiği için, eşik değeri 0.01 ile 0.99 arası belirlenmelidir. Açgözlü yaklaşım ilk en iyi arama algoritmasıdır. Burada en önemli parçacıklar, ağırlıkları en yüksek parçacıklardır. Bu durumda parçacıklar büyükten küçüğe doğru sıralanmalıdır. Eşik değeri aşınca kadar, ağırlıkları büyük olan parçacıktan küçük olan parçacığa doğru parçacıkların ağırlıkları toplanmalıdır. Böylece hangi parçacıkları çoğaltılacağı Açgözlü yaklaşımla bulunmuş olur. En son ise, bir parçacığın kaç adet olacağı başlangıçta parçacık sayısı ile parçacığın ağırlığı çarpımının, sistemde kalan parçacıkların toplam ağırlığına bölünmesiyle hesaplanmaktadır.

Bu yaklaşımın en önemli avantajı parçacıkların ağırlıklarını önemli bilgi olarak kullanılmasıdır. Probleme göre eşik değeri iyileştirme parametresi olarak kullanılabilir. Yapılan deneysel sonuçlarda, diğer algoritmalarından daha iyi performans göstermiştir. Parçacık filtresinde hedef takibinde, parçacık filtresinin yeniden örnekleme adımı bu algorithmada çok kolaylıkla kullanılabilir. Diğer yeniden örnekleme algoritmaları gibi parçacıkların ağırlıkları giriş parametresi olacaktır. Ayrıca, eşik değeri giriş parametresi olarak verilmelidir. Eşik değeri varsayılan olarak 0.5 olarak verilebilir. Parçacık sayısı 200'den büyükse eşik değeri 0.5 den küçük olarak belirlenebilir. Parçacık sayısı 50'den daha az ise, 0.5 den büyük olarak belirlenebilir. Algoritma diğer yeniden örnekleme algoritmalarında olduğu gibi zayıf parçacıkları sistemden çıkartarak, başlangıçta parçacık sayısı kadar, parçacıkların indislerini geri dönecektir. Kestirme adımı yeniden örneklenmiş parçacıkların indisleri kullanılarak parametre kestirimi yapılacaktır. Açgözlü yaklaşımla önerilen algoritma adımları aşağıdaki gibidir.

Adım 1: Parçacıkların ağırlıkları  $w_1, w_2, \dots, w_N$ ,  $N$  tane parçacık için giriş değeri olarak alınır.

Adım 2: 0.01-0.99 arası bir eşik değeri  $\tau$  belirlenir.

Adım 3: Tüm parçacıkların ağırlıklarının toplamı hesaplanır.

Toplam ağırlık  $W_T = \sum_{i=1}^N w_i$  ifade edilir.

Adım 4: Her bir parçacığın ağırlığı Adım 3'te elde edilen toplam ağırlığa bölünerek normalize ağırlıklar  $W_i = w_i/W_T$  bulunur.

Adım 5: Adım 4’de elde edilen parçacıkların ağırlıkları büyükten küçüğe doğru sıralanır,  $W_{\sigma(1)} \geq W_{\sigma(2)} \geq \dots \geq W_{\sigma(N)}$ . Burada,  $\sigma(\cdot)$  fonksiyonu sıralama indisini ifade eder.

Adım 6: Kümülatif normalize ağırlıklar  $V_r = \sum_{i=1}^r W_{\sigma(i)}$  ile bulunur.

Adım 7: Eşik değerini aşan en küçük ağırlık indisi  $k$  bulunur.

$$k = \underset{r}{\operatorname{argmin}} \quad V_r \geq \tau$$

Adım 8:  $\sigma(1), \sigma(2), \dots, \sigma(k)$  indisli parçacıklar ağırlıkları oranınca çoğaltılır. Her bir parçacık için çoğaltılma sayısı  $K_{\sigma(i)} = NW_{\sigma(i)}/V_R$  eşitliğiyle hesaplanır. Eşitlik sonucu reel sayı olduğu için tam sayıya çevirmek için yuvarlama fonksiyonları kullanılır.

Adım 9: Çoğaltma işlemi sonunda elde edilen toplam parçacık sayısı  $K_t = \sum_{i=1}^k K_{\sigma(i)}$  başlangıçtaki parçacık sayısı  $N$  ile karşılaştırılır. Yuvarlama işleminden dolayı  $K$  ve  $N$  eşit olmayabilir.  $K_t < N$  ise, başlangıçta yüksek ağırlığa sahip parçacıklar eklenir,  $K_t > N$  ise, başlangıçta düşük ağırlığa sahip indisli parçacıklar çıkartılır.

Açgözlü yeniden örnekleme yöntemini basit bir örnek ile açıklamak üzere sistemde başlangıçta 5 adet parçacığın ağırlıkları  $\{1,2,3,8,16\}$  verilsin. Eşik değeri 0.79 olarak belirlensin. Bu örnek üzerinde algoritmanın çalışması aşağıdaki maddeler halinde sıralanmıştır.

Adım 1:  $\{1, 2, 3, 8, 16\}$

Adım 2:  $\tau = 0.79$

Adım 3:  $W_t = \sum_{i=1}^N w_i = 1+2+3+8+16=30$

Adım 4:  $W_i = w_i/(W_t)$  şeklinde hesaplanır.

$W_1 = 1/30; W_2 = 2/30; W_3 = 3/30;$

$W_4 = 8/30; W_5 = 16/30$  şeklinde hesaplanır.

Adım 5:  $[1/30; 2/30; 3/30; 8/30; 16/30]$  büyükten küçüğe doğru sıralanırsa;  $16/30 \geq 8/30 \geq 3/30 \geq 2/30 \geq 1/30$  şeklindedir.  $W_5 \geq W_4 \geq W_3 \geq W_2 \geq W_1$  şeklindedir. Sıralanmış parçacıkların indisleri  $[5; 4; 3; 2; 1]$  şeklindedir.

$\sigma(1) = 5; \sigma(2) = 4; \sigma(3) = 3; \sigma(4) = 2; \sigma(5) = 1.$

Adım 6:  $r=2$  için;  $V_r = \sum_{i=1}^2 W_{\sigma(i)} = W_5 + W_4 = 16/30 + 8/30 = 24/30.$

Adım 7:  $r=2$  için;  $24/30 \geq 0.79$  olduğundan;  $V_r \geq \tau$  eşitliğini sağlayan en küçük  $r$  değeri 2'dir. Yani  $k=2$ 'dir. En yüksek ağırlıklı 2 indis çoğaltılacaktır. Yani,  $[5;4]$  indisli parçacıklar çoğaltılacaktır.

Adım 8:  $K_5 = NW_5/V_R = (5*16/30)/(24/30)=3.33$

$K_4 = NW_4/V_R = (5*8/30)/(24/30)=1.67$ . En yakın sayıya yuvarlama fonksiyonları kullanılırsa;  $[3.33;1.67]$  değerleri  $[3;2]$  şeklinde yuvarlanır.

Adım 9:  $K_t = K_5 + K_4 = 3+2=5$ .  $K_t = N$  olduğundan 8. adımda herhangi bir değişiklik yapmadan 5 indisli parçacıktan 3 adet 4 indisli parçacıktan 2 adet olmak üzere;  $[5;5;5;4;4]$  çıktı vererek algoritma sonlanır

### 3.3. Açgözlü Yeniden Örnekleme Algoritması Testleri

#### 3.3.1. Lineer olmayan modelde hedef takibi

Denklem (3.1) ile Denklem (3.4)'de verilen süreç ve ölçüm fonksiyonları yeniden örnekleme algoritmalarının performansını ölçmek için birçok makalede kullanılmaktadır. Bu makalelere, [1] ve [25] yapılan çalışmalar örnek olarak verilebilir. 2015 yılında, özgün ve yeni olan Minimum Örnek Varyansı yeniden örnekleme algoritması önerildiğinde bu denklemler kullanılmıştır [25]. Doğrusal olmayan problem olması nedeniyle tercih edilmektedir. Açgözlü yeniden örnekleme algoritmasının performansını ölçmek için [1] ve [25]'de verilen ve yeni önerilen örnekleme algoritmasının performansını ölçmek için kullanılan Parçacık filtresi olarak [1]'de verilen jenerik parçacık filtresi kullanılmıştır.

,

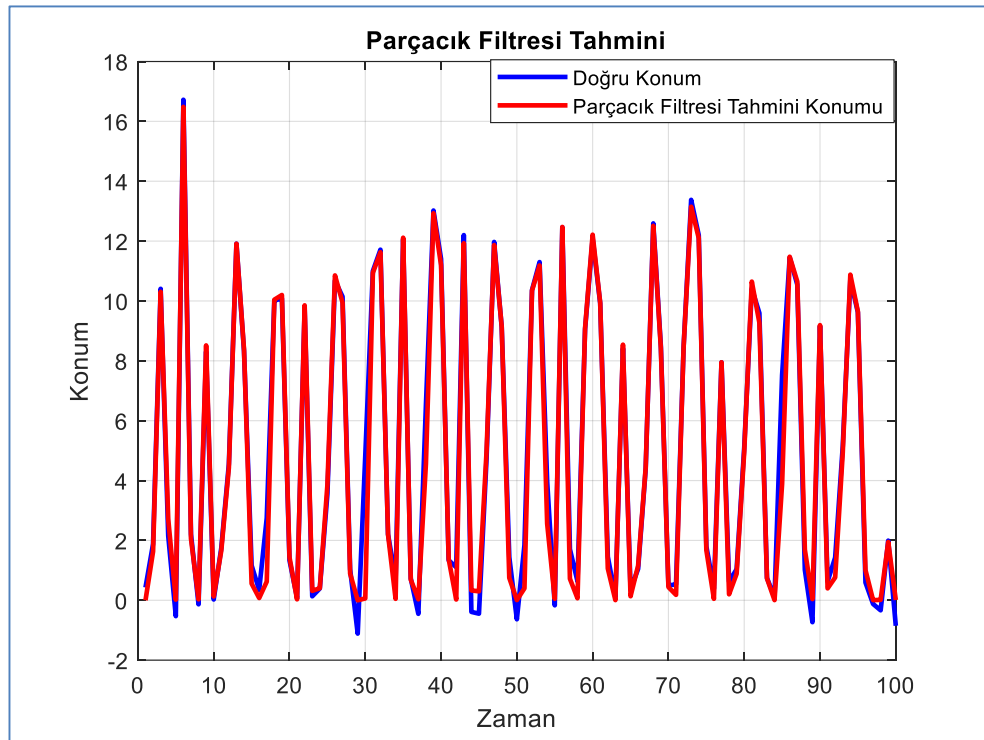
$$x_k = \frac{x_{k-1}}{2} + \frac{25x_{k-1}}{1 + x_{k-1}^2} + 8\cos(1.2k) + v_k \quad (3.1)$$

$$Z_k = \frac{x_k^2}{20} + r_k \quad (3.2)$$

$$v_k \sim \mathcal{N}(0, Q) \quad (3.3)$$

$$r_k \sim \mathcal{N}(0, R) \quad (3.4)$$

Burada,  $x_k$  durum denklemini,  $Z_k$  ölçümleri,  $v_k$  süreç gürültüsünü ve  $r_k$  ise ölçüm gürültüsünü belirtmektedir. Bu çalışmada,  $v_k$  sıfır ortalamalı  $Q$  kovaryanslı bir Gauss gürültüsü,  $r_k$  ise sıfır ortalamalı  $R$  kovaryanslı bir Gauss gürültüsü olarak alınmıştır. Monte Carlo benzetimi testleri yapılmıştır. Şekil 3.1’de (3.1) ’deki durum denklemi ve (3.2) ’deki ölçüm denklemine sahip bir hedefin parçacık filtreleri ile 100 sn. boyunca izlenmesi görülmektedir. Ölçüm gürültüsü varyansı 10 standart sapması 3.16, süreç gürültüsünün standart sapması 1’dir. Parçacık sayısı 100 olarak alınmıştır. Ölçüm zaman aralığı ise 1 sn’dir.

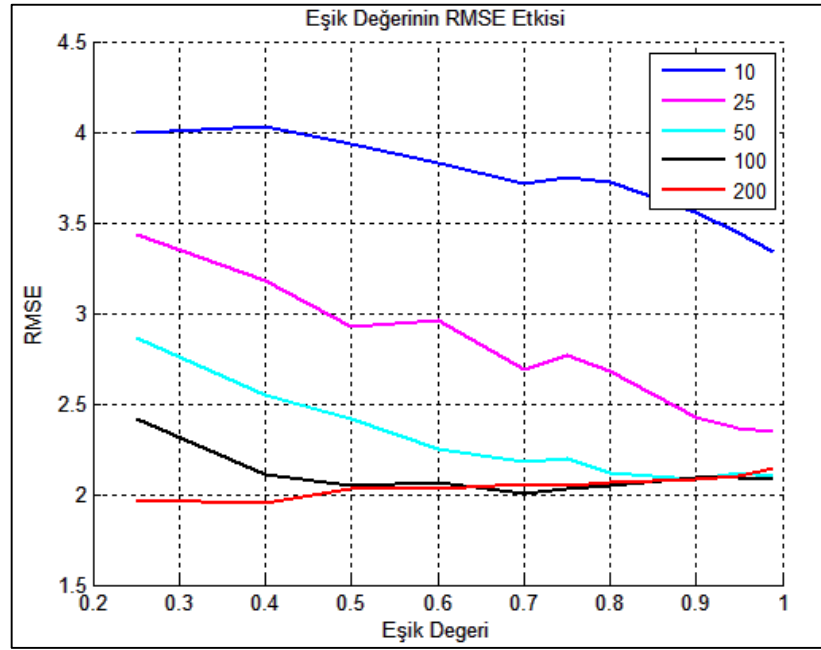


Şekil 3.1: Hedefin parçacık filtreleri ile takip edilmesi.

Değerlendirme metriği olarak ortalama karesel hatanın karekökü (RMSE, Ing: Root Mean Square Error) kullanılmıştır. Denklem 3.5’de RMSE formülü görülmektedir.  $x_k^g$   $k$  anındaki gerçek değeri,  $x_k^t$   $k$  anındaki tahmini değeri belirtmektedir.

$$RMSE = \sqrt{\frac{1}{N} \sum_{k=1}^N (x_k^g - x_k^t)^2} \quad (3.5)$$

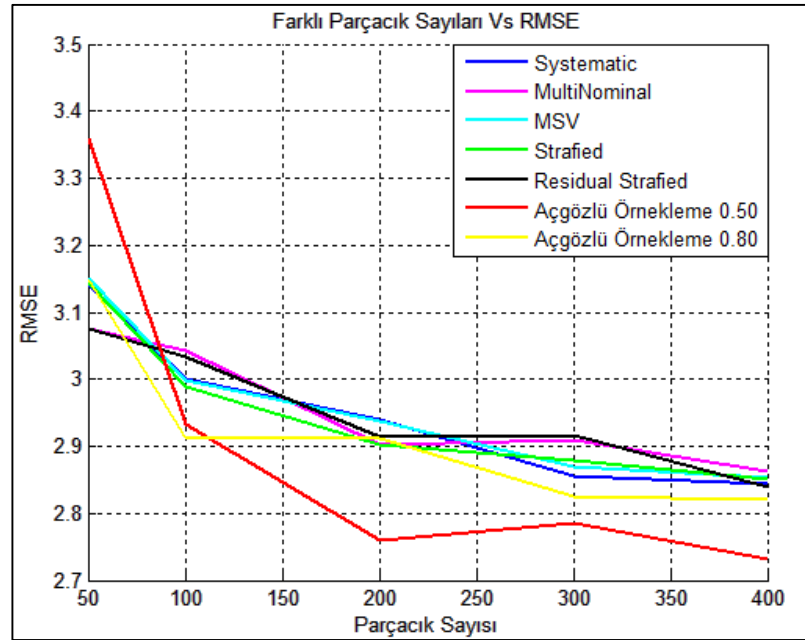
Şekil 3.2’de, bu hareket modelinde, eşik değerinin farklı parçacık sayıları için ortalama karesel hataya olan etkisi görülmektedir. 200 Monte Carlo benzetimi testi yapılmıştır. Parçacık sayısının 10, 25 ve 50 gibi düşük değere sahip olması durumlarında en iyi performansı 0.99 eşik değeri vermiştir. 100 parçacık sayısında en iyi kestirim performansını 0.70 eşik değeri verirken, 200 parçacık sayısında en iyi kestirim performansı 0.40 eşik değeri ile elde edilmiştir. Eşik değerini, az sayıda parçacık kullanılacaksa yüksek seçmek gerekmektedir. Bunun temel sebebi sürekli yüksek parçacıkların seçilmesi örnek fakirleşmesine neden olmaktadır [1]. Az sayıda parçacık olduğunda ve düşük eşik değeri belirlendiğinde örnek fakirleşmesine neden olmaktadır. Daha çok parçacığın olduğu durumlarda ise eşik değeri daha düşük seçilmelidir.



Şekil 3.2: Eşik değerinin ortalama karesel hataya etkisi.

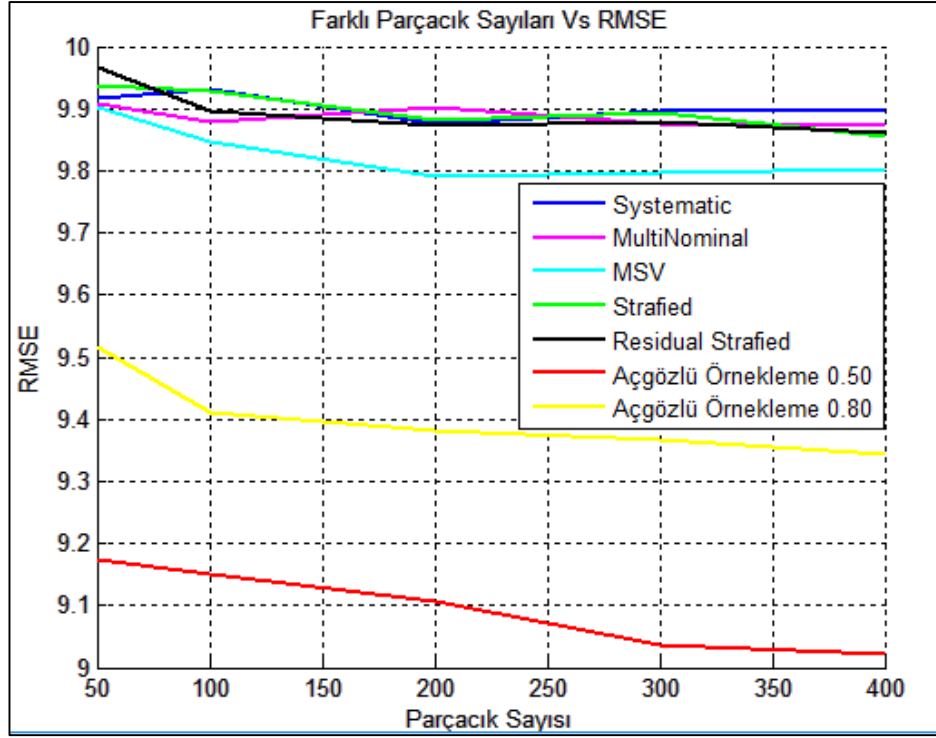
Şekil 3.3’de farklı parçacık sayıları ve farklı yeniden örnekleme algoritmaları kullanılarak elde edilen hedef konumlarının ortalama karesel hataları görülmektedir. Parçacık sayısının artması, ortalama karesel hatanın azalmasına, hedef konumunun daha doğru kestirilmesini sağlamaktadır. Ölçüm ve süreç gürültüleri Gauss dağılımlıdır. Ölçüm gürültüsünün varyansı 10, standart sapması 3.16 ve süreç gürültüsünün standart sapması 1’dir. 100 örnekli Monte Carlo benzetimi testi yapılarak için elde edilen sonuçlarda, en iyi performansı Açgözlü yaklaşımla

oluşturulan önerilen algoritma vermiştir. 200, 300, 400 parçacık sayısında 0.50 eşik değerinde önerilen algoritma vermiştir.



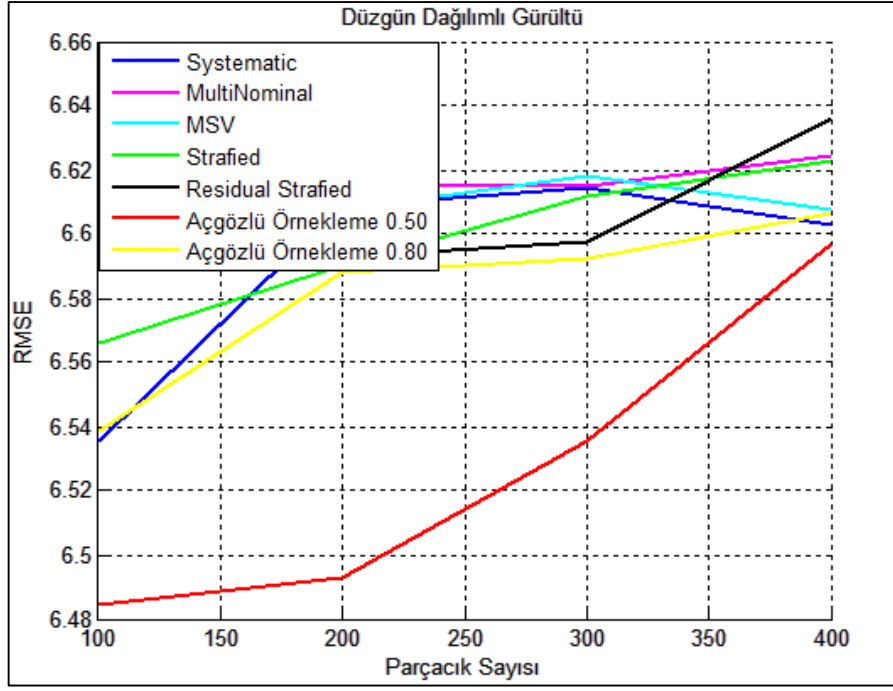
Şekil 3.3: Farklı parçacık sayılarında farklı algoritmalarının performansı.

Şekil 3.4’de gürültü değerinin ortalama karesel hataya etkisi tespit etmek için testler yapılmıştır. 100 Monte Carlo benzetimi testi yapılmıştır. Şekil 3.2 ve Şekil 3.3 karşılaştırılırsa, gürültünün standart sapmasının artması beklendiği gibi ortalama karesel hatayı artırmıştır. En iyi performansı 0.50 eşik değerinde önerilen algoritma vermiştir. Diğer algoritmalar 200 parçacık sayısında 9.9 ortalama karesel hata verirken, Ağgözlü algoritmanın ortalama karesel hata değeri 0.50 eşik değerinde 9.1 ve 0.80 eşik değerinde 9.4’tür. Ölçüm gürültüsünün standart sapması 10 ve süreç gürültüsünün standart sapması 1’dir.



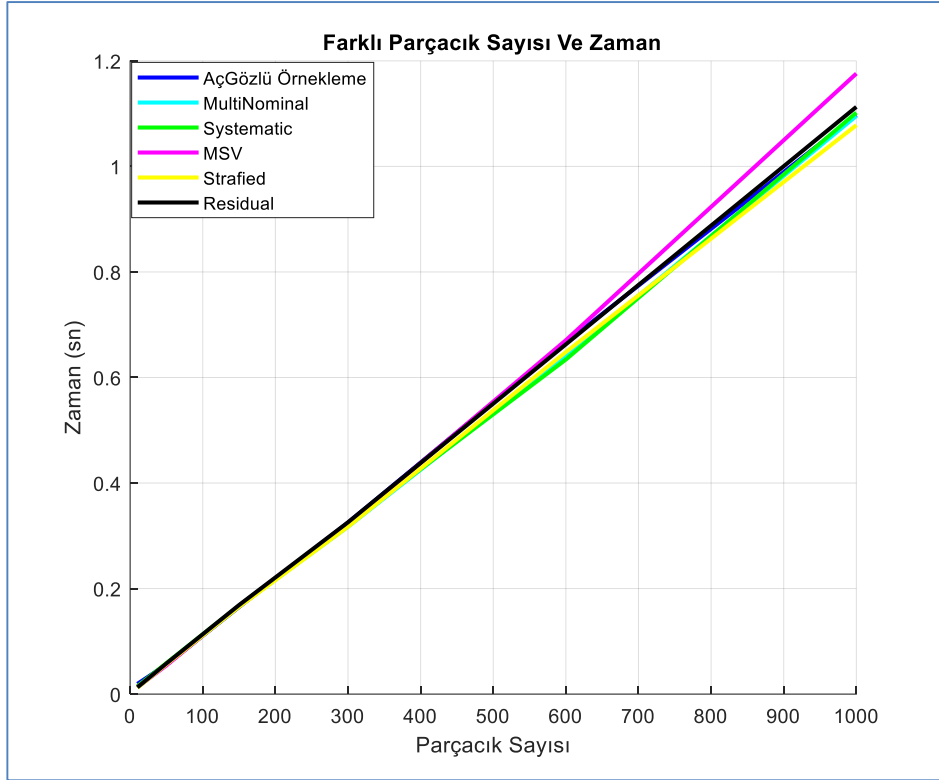
Şekil 3.4: Ölçüm gürültüsünün ortalama karesel hataya etkisi.

Şekil 3.5'te gürültü tipini ortalama karesel hataya etkisini ölçmek için testlerin sonuçları görülmektedir. 100 Monte Carlo benzetimi testi yapılmıştır. Gauss olmayan düzgün dağılımlı gürültü örnekleri 0 ile 100 arasında oluşturulmuştur. Tüm algoritmaların performansları birbirine yakındır. En iyi performansı 0.50 eşik değerinde Ağgözlü yaklaşımla önerilen algoritma vermiştir. 200 parçacık sayısında diğer algoritmalar 6.6 karesel hata verirken, 0.50 eşik değerinde Ağgözlü örnekleme algoritması 6.48 karesel hata değerini vermiştir.



Şekil 3.5. Gürültü tipinin ortalama karesel hataya etkisi.

Şekil 3.6’da algoritmalarının hızlarını ölçmek için testler yapılmıştır. Algoritmalar 1000 defa çalıştırıldığında elde edilen süreler görülmektedir. Tüm algoritmalar genel olarak doğrusal zamanda çalışmaktadır. Önerilen algoritma da yaklaşık olarak doğrusal zamanda çalışmaktadır. Önerilen algoritmada çalışma zamanını etkileyen en önemli metod parçacıkların sıralanmasıdır. Sıralama algoritmaları Big O notasyonuna göre karmaşıklığı  $N \log N$ ’dir. Bu sebepten ötürü, önerilen algoritmanın karmaşıklığı  $N \log N$ ’dir.



Şekil 3.6: Algoritmaların çalışma süreleri.

### 3.3.2. Kartezyen koordinatlarda hedef takibi

Üç-boyutlu bir düzlemde hareket eden sabit hızlı hedefin kartezyen koordinatlardaki konum ölçümlerine ilişkin hareket modelinin durum modeli, ve ölçüm modeli sırasıyla Denklem (3.6) ile (3.12) arasında verilmektedir.  $X_k$  durum vektörü  $k$  anındaki  $x$ ,  $y$  ve  $z$  yönündeki konumları ile hızlarını içermektedir. Ölçüm vektörü  $Z_k$  radar tarafından ölçülen hedef konumlarını içermektedir.  $v_k$  süreç gürültüsünü,  $r_k$  ise ölçüm gürültüsünü belirtmektedir. Bu çalışmada,  $v_k$  sıfır ortalamalı  $Q$  kovaryanslı bir Gauss gürültüsü,  $r_k$  ise sıfır ortalamalı  $R$  kovaryanslı bir Gauss gürültüsü olarak alınmıştır.  $dt$  ölçüm zaman aralığını belirtmektedir.

$$X_{k+1} = FX_k + v_k \quad (3.6)$$

$$Z_k = HX_k + r_k \quad (3.7)$$

Burada,

$$X_k = [x_k \ y_k \ z_k \ \dot{x}_k \ \dot{y}_k \ \dot{z}_k]^t \quad (3.8)$$

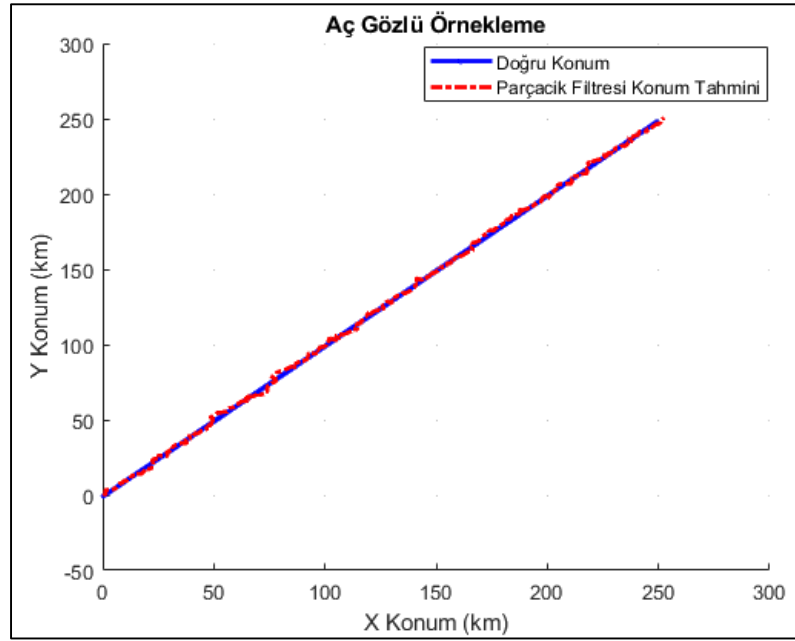
$$F = \begin{bmatrix} 1 & 0 & 0 & dt & 0 & 0 \\ 0 & 1 & 0 & 0 & dt & 0 \\ 0 & 0 & 1 & 0 & 0 & dt \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.9)$$

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \quad (3.10)$$

$$r_k \sim \mathcal{N}(0, Q_k) \quad (3.11)$$

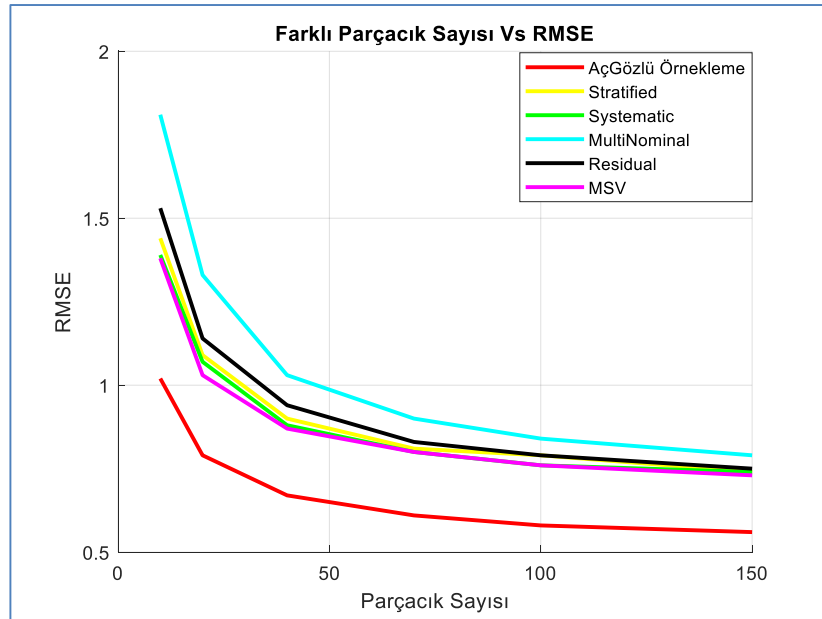
$$v_k \sim \mathcal{N}(0, R_k). \quad (3.12)$$

Algoritmaların performans karşılaştırmaları için mono statik bir radar ile uçak takibi senaryosu ele alınıp sentetik veriler üretilmiştir. Uçağın hareketini modellemek için z yönünde 6,5 km sabit yükseklikte olduğunun varsayımı altında, X ve Y yönündeki başlangıç referans noktaları 0 alınarak uygun veriler üretilmiştir. X ve Y yönünde başlangıç hızları 100 m/s, zaman aralığı 1 saniye olmak üzere, 100 saniye boyunca hareket eden hedef izlenmiştir. Süreç gürültülerinin 0 olmadığı durumlar için örnekli Monte Carlo benzetimi testleri yapılmıştır. Süreç gürültüleri, standart sapmayla orantılı Gauss gürültüleridir. Ölçümlere standart sapmayla orantılı gauss gürültüsü de eklenmiştir. Şekil 3.7’de bu parametreler için üretilen bir hareketlinin parçacık filtrelerinde Açgözlü yeniden örnekleme algoritması kullanılarak izlenmesi görülmektedir. Ölçüm gürültüsünün standart sapma değeri 10 ve süreç gürültüsünün standart sapma değeri 1’dir.



Şekil 3.7 3 boyutlu kartezyen koordinatlarda hareket izleme.

Şekil 3.8’de 4 standart sapmalı ölçüm gürültüsünde, kartezyen koordinat hareket modelinde farklı parçacık sayısında yeniden örnekleme algoritmalarının sonuçları görülmektedir. RMSE birimi km’dir. 250 parçacık sayısında en iyi sonucu açgözlü yeniden örnekleme algoritması vermiştir. Diğer parçacık sayılarında diğer algoritmalara yakın sonuçlar vermiştir. Açgözlü algoritmanın eşik değeri 0.5 olarak alınmıştır.



Şekil 3.8: 3 boyutlu kartezyen koordinatlarda algoritmaların sonuçları.

### 3.3.3. Sürekli dönüş modelinde hedef takibi

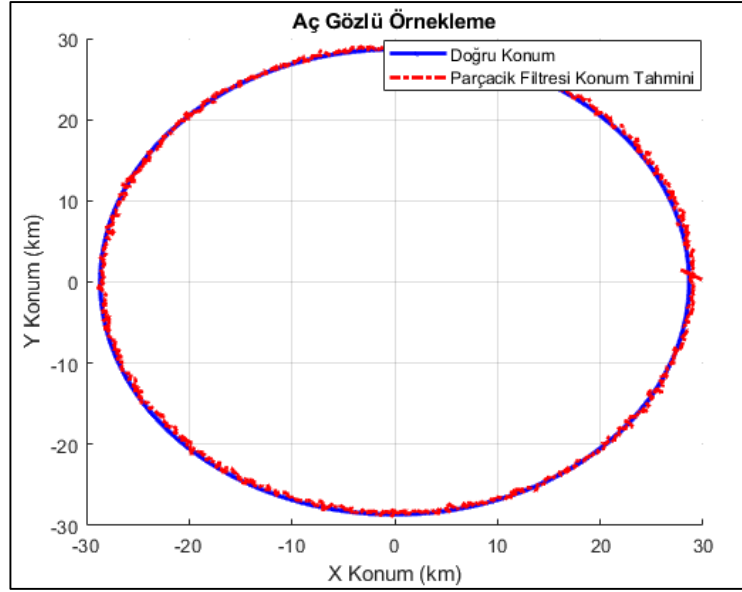
X ve Y yönünde 30 km çapında dairesel hareket yapan saatte 720 km ilerleyen bir hedef için parçacık filtreleri ile hareketinin izlenmesi Şekil 3.9’da verilmiştir. Hedefin X ve Y yönündeki hızları değişken olmasına rağmen, açısal hızı sabittir. Ölçüm gürültüsü standard sapması 1, süreç gürültüsü standart sapması 0.1’dir. Aç gözlü yeniden örnekleme algoritması kullanılmıştır. Bu dairenin etrafında 1 tur atabilmek için 300 saniye geçmektedir. Zaman aralığı 1 saniye olarak alınmıştır. 100 Monte Carlo benzetimi testi yapılmıştır. Şekil 3.9’da bu parametreler için üretilen bir hareketlinin parçacık filtrelerinde Açgözlü yeniden örnekleme algoritması kullanılarak izlenmesi görülmektedir. Aç gözlü algoritmada eşik değeri 0.5 alınmıştır. Burada  $R$  çapı belirtmektedir.  $V$  hedefin hızını belirtmektedir. 720 km/h dir. Saati saniye çevirmek için 3600 ile çarpılmıştır.  $N_{sample}$  değeri Eşitik (3.13) formülle hesaplanmaktadır. Tablo 3.2’de Sürekli dönüş modelindeki x ve y konumların nasıl elde edildiği görülmektedir.  $q_x$  x konumlarına gürültüyü,  $q_y$  y konumlarına eklenen gürültüyü belirtmektedir.  $x_k$  x konumlarını  $y_k$  y konumlarını belirtmektedir.

$$N_{sample} = \frac{2 * \pi * R * 3600}{V * 2 * dt} \quad (3.13)$$

Tablo 3.2: Sürekli dönüş modelindeki konumların hesaplanması.

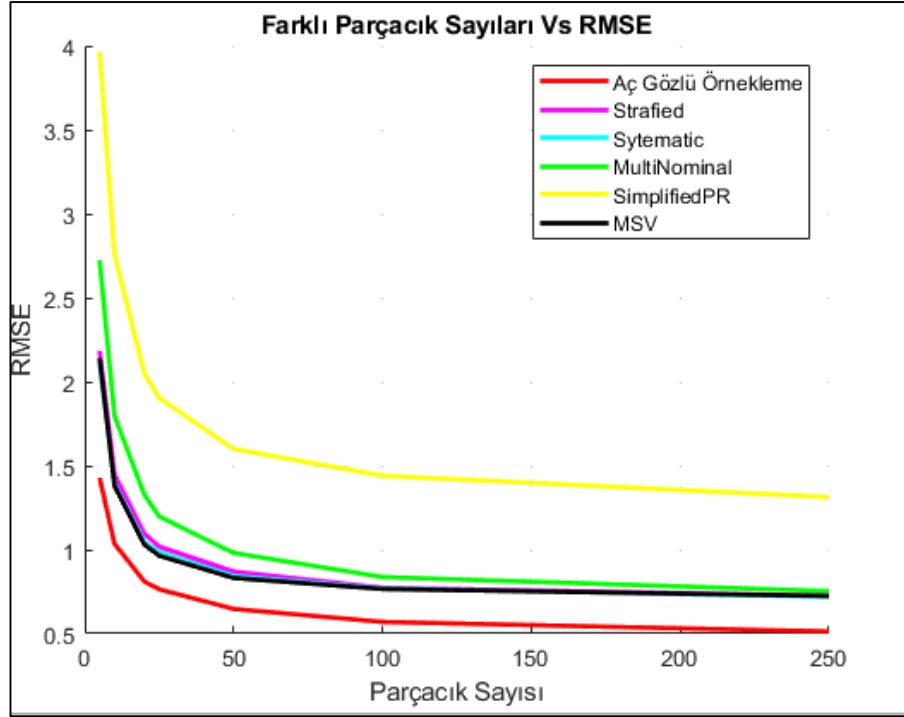
```
for i=0:pi/N_sample:2*pi  
     $x_k = R * \cos(i) + q_x$   
     $y_k = R * \sin(i) + q_y$   
end
```

Şekil 3.9’da sürekli dönüş modeli yapan bir hedefin parçacık filtrelerinde Açgözlü yeniden örnekleme algoritması kullanılarak izlenmesi görülmektedir. Aç gözlü algoritmada eşik değeri 0.5 alınmıştır.

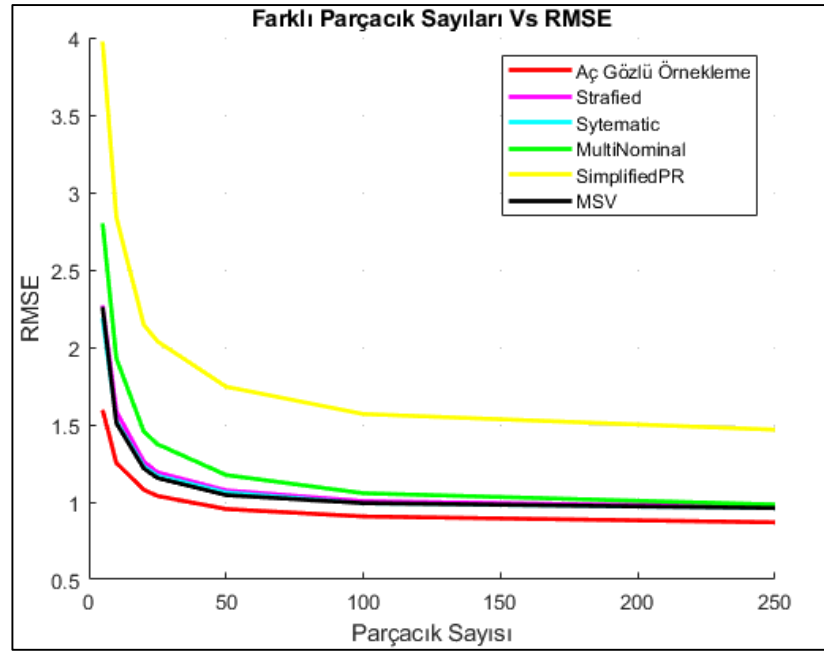


Şekil 3.9: Dairesel hareket modelinde hareket izleme.

Şekil 3.10, 3.11’de dairesel hareket modelinde farklı parçacık sayısında yeniden örnekleme algoritmalarının sonuçları görülmektedir. RMSE birimi km’dir. Şekil 3.10’da, ölçüm gürültüsünün standart sapması 0.5, Şekil 3.11’de, ölçüm gürültüsünün standart sapması 1, süreç gürültüsü standart sapması 0.1 ‘dir. Şekil 3.10 ve Şekil 3.11’de görüldüğü üzere tüm farklı parçacık sayısında en iyi performansı Açgözlü algoritma vermiştir. Dairesel hareket modelinde ölçüm gürültüsünün düşük olduğu durumda Açgözlü algoritma daha iyi sonuç vermektedir.



Şekil 3.10: Dairesel hareket modelinde 0.5 standart sapmalı ölçüm gürültüsünde algoritmaların sonuçları.



Şekil 3.11: Dairesel hareket modelinde 1 standart sapmalı ölçüm gürültüsünde algoritmaların sonuçları.

### 3.4. Genel Değerlendirme

Açgözlü yaklaşımla bir yeniden örnekleme algoritması sunuldu. Açgözlü algoritmanın performansı etkileyen parametre eşik değeridir. Az sayıda parçacık sayısında bu eşik değeri yüksek olabilir. Daha fazla parçacık sayısında daha düşük seçilmelidir. Lineer olmayan hareket modeli, sabit hızlı Kartezyen koordinatlarda hareket modeli, dairesel hareket modelinde en iyi performansı farklı parçacık sayılarındaki Açgözlü algoritma vermiştir. O algoritmanın performansına en yakın algoritma MSV vermiştir.

## 4. TORBA PROBLEMİ ÇÖZÜMÜ TABANLI YENİDEN ÖRNEKLEME ALGORİTMASI

Bu bölümde, dinamik programlama yaklaşımıyla torba problemi çözülerek geliştirilen özgün bir yeniden örnekleme algoritması sunulmaktadır. Torba problemi, bir torbanın belirli bir kapasiteye sahip belirli sayıda öğenin yerleştirilebildiği bir senaryodan alan, kısıtlı bir optimizasyon problemidir. Her öğenin bir değeri ve ağırlığı vardır. Amaç, torbanın içindeki eşyaların toplam değerini maksimize ederken, eşyaların toplam ağırlığının çantanın kapasitesinden az veya ona eşit olacak şekilde çantada tutulacak her bir maddenin sayısını belirlemektir. Bu problemi çözmek için birçok yaklaşım kullanılmaktadır. Bu problemi hızlı ve en optimum çözen yaklaşım ise, dinamik programlama yaklaşımıdır. Tez çalışmasında, yeniden örnekleme problemi önce torba problemine dönüştürülmekte, daha sonra ise, dinamik programlama yaklaşımıyla torba problemi çözülmektedir. Torba problemi için giriş parametresi olarak üç parametre vardır. Bu parametreler eşyaların ağırlığı, eşyaların değerleri ve torbanın kapasitesi şeklindedir.

Torbanın kapasitesi, yeniden örnekleme algoritmasında belirlenen eşik değeri, 100-1000 gibi parçacık sayısına bağlı olarak bir sayıyla genişletilerek, torba problemindeki ağırlık kısıtlaması olarak varsayılır. Parçacıkları ağırlıklarının karesi eşyaların değerleri ve parçacıkların ağırlıkları ise eşyaların ağırlıkları olarak varsayılarak bir çözüm geliştirilmiştir. Böylece, eşyaların değerinin eşyaların ağırlıkları oranı, başlangıçtaki parçacıkları ağırlıklarına eşitlenmiştir. Parçacıkların ağırlıkları önemli bilgi olarak kullanılmıştır. Yeniden örnekleme algoritması, Torba problemi dönüştürüldükten sonra, dinamik programlama yaklaşımıyla çözülerek, hangi parçacıkların yer alacağı tespit edilmiştir. Önerilen algoritmayla, farklı hareket modellerinde elde edilen sonuçlar 2. bölümde detayları verilen diğer yeniden örnekleme algoritmalarıyla karşılaştırılmıştır. Doğrusal olmayan hareket modeli, kartezyen hareket modeli ve dairesel hareket modelinde en iyi performansı torba problemi çözümü tabanlı yeniden örnekleme algoritması vermiştir.

## 4.1. Dinamik Programlama

Dinamik programlama karışık problemlerin daha basit düzeylere indirilerek çözülmesini esas alan bir optimizasyon yaklaşımıdır. Optimizasyondaki amaç, problemdeki sınırlayıcı koşullar altında bu problemle ilgili en iyi karara ulaşmaktır.

İlk defa 1952 yılında Richard Bellman tarafından bu yaklaşım önerilmiştir. [39,40]. Bir problem üzerinde dinamik programlama uygulayabilmek için bir önceki problemin karakteristiğini koruyacak şekilde çözümü daha kolay başka probleme dönüştürülebilir olması ve o problemin alt problemlere parçalanabilir olması gerekir [40,41]. Dinamik programlama yaklaşımına göre, optimum çözüm başlangıç durumundan bağımsız olarak diğer çözümler ile çözüm sonuçlarına göre optimum çözümler ardışıklığındadır. Yani, başlangıçta alt problemlerin çözümü bulunduğu, buradan elde edilen verilerle daha büyük alt problemler çözüldüğünde problemin kendisi de çözülmüş olacaktır.

Dinamik programlama, aynı çözümlü küçük problemlere parçalanabilen tüm problemler için uygulanabilir. Fakat kaba kuvvet yaklaşımı ile eksponansiyel zamanda çözülebilen problemlerde gerçek değerini gösterir.

Literatürde dinamik programlama ile çözülebilecek bazı problemler aşağıda verilmiştir.

- Parçalı en küçük kareler yaklaşımı problemi [42]
- Levensthein uzaklık algoritması [43]
- Viterbi çözümleme algoritması [44]
- Alt küme toplama problemi [45]
- Metinlerde, en uzun ortak küme bulma algoritması [46]
- Gezgin satıcı problemi [47]
- Matris zincir çarpımı problemi [48]
- İş zamanlaması problemi [49]
- Para değişimi problemi [50]
- Torba problemi [51-56]

Bir problem için dinamik programlama algoritması yapmak için aşağıdaki dört adım uygulanmalıdır [41]:

Adım 1: En iyi çözümün özyinelemeli tanımı

Adım 2: En iyi çözümün özyinelemeli tanımı

Adım 3: En iyi çözümün değerini yukardan aşağıya veya alttan yukarıya yaklaşımdan birini kullanarak hesaplanması

Adım 4: Hesaplanmış verileri kullanarak en iyi çözümü bulunması

Yazılan dört adımın ilk üç adımı dinamik programlamanın temel üç adımıdır. Eğer optimal çözümün yalnızca değerine ihtiyacımız varsa, yalnızca ilk üç adımı kullanmak yeterli olacaktır.

## 4.2. Torba Problemi

Torba probleminin literatürde popüler bir problem olduğu bilinmektedir [51-57]. Bu, adını bir torbanın belirli bir kapasiteye sahip belirli sayıda öğenin yerleştirilebildiği bir senaryodan alan kısıtlı bir optimizasyon problemidir. Her öğenin bir değeri ve ağırlığı vardır. Amaç, torbanın içindeki eşyaların toplam değerini maksimize ederken, eşyaların toplam ağırlığının çantanın kapasitesinden az veya ona eşit olacak şekilde çantada tutulacak her bir maddenin sayısını belirlemektir. 0-1 torba problemi ve kesirli torba problemi olarak adlandırılan iki farklı sırt çantası problemi vardır. Kesirli torba probleminde, eşyaların bir kısmı da alınabilir. 0-1 torba probleminde ise eşyalar ya alınır ya da hiç alınmaz. Bu adlandırma kuralının ana nedeni, her öğenin çantaya yalnızca bir kez ve herhangi bir kesir olmadan eklenebilmesidir [52-57]. Dinamik programlama tekniğini kullanarak ilk kez torba problemine bir çözüm [57] 'de önerildi. Dinamik programlama temelde düz özyinelemeden ziyade bir optimizasyondur. Dinamik programlama algoritması ile torba problemini çözmek için sözde bir kod Algoritma 2'de verilmiştir [57]. Algoritma, öğelerin değerlerini, ağırlıklarını, ayrıca torbanın kapasitesini alır ve sadece 0 ve 1 sayılarından oluşan çözüm vektörünü döndürür.

Bu sonuç, dinamik programlamanın tüm alt problemler için optimal olduğunu göstermektedir. Dinamik programlama, akıllı bir kaba kuvvet algoritması olarak düşünülebilir. Bu basit yaklaşım, işlemleri yeniden gerçekleştirmek yerine alt problem çözümünü sakladığı için; üstel karmaşıklıklarını polinomsal zaman karmaşıklıkları olarak azaltır [58-59].

Torba probleminde, hırsız eve giriyor. Çantası her eşyayı almaya yetmiyor. Hırsız yükte hafif, pahada ağır eşyaları alarak maksimum faydayı sağlamaya çalışır.

0-1 torba probleminde, eşyalar ya tamamen alınır ya da tamamen bırakılır. Alınacak olan eşyanın bir kısmını almak mümkün değildir. Örneğin, A eşyası 6 kg değeri 600, B eşyası 5 kg değeri 400, C eşyası 4 kg değeri 300 olduğunda. Çantamız 9 kg alırsa, Açgözlü yaklaşım, eşyaları değer/ağırlıklara göre sıralar: A eşyası 600/6, B eşyası 400/5, C eşyası 300/4. Buradan eşyaların değer/ağırlık oranları sırasıyla A=100, B=80, C=75 şeklindedir. Açgözlü yaklaşım, 6 kg'lık eşyayı çözüm olarak geri döner. Öncelikle eşyaları değer/ağırlığa göre sıralar. Sırasıyla toplar. Önce A eşyasını torbaya ekler. Sonra B eşyasını eklemeye çalışır. Ancak bu örnekte torbanın ağırlığı maksimum kapasitesini aştığı için ekleyemez. Çünkü 6+5 =11 olduğunda bu değer çantamızın ağırlığından büyüktür. Değeri 600'dür. Bu problem için, Dinamik algoritma, 5kg+4kg ağırlığında, 700 değerinde eşyaları geri döner. Ancak dinamik programlama yaklaşımlı algoritmalar Açgözlü yaklaşımlara göre çalışma süreleri daha yavaşlardır.

Torba probleminin dinamik programlama ilgili sözde kodu Tablo 4.1'de verilmektedir.

Tablo 4.1: Torba problemi çözümünün Dinamik Programlama sözde kodu.

```
%  $G_k$  k indisli eşyanın ağırlığını belirtir.
%  $V_k$  k indisli eşyanın değerini belirtir.
% C torbanın alabileceği maksimum kapasiteyi belirtir.
% N eşyaların sayısını belirtmektedir.
Procedure Knapsack( $G_k, V_k, C, N$ ):

% Values hafızada yer ayrılıyor.
Values[0... N][0... C]

% birinci bölüm

% Values 0. Sütunlarına 0 atama yapılır.
Values[n][0] = 0, n = 0, ..., N

% Values 0th satırlarına atama yapılır.
for c from 1 to C
    if  $c \geq G_k[0]$ 
        Values[0][c] =  $G_k[0]$ 
    else
        Values[0][c] = 0
    end if
end for
```

Tablo 4.1: Devam.

```

%Eşyaların değerlerini belirten Values dizisine atama yapılır.

for  $n$  from 1 to  $N$ 
  for  $c$  from 1 to  $C$ 
    if  $G_k[n] \geq c$ 
       $Values[n][c] \leftarrow Values[n-1][c]$ 
    else
       $Values[n][c] = \max(V_k[n] + Values[n-1][c - G_k[n]], Values[n-1][c])$ 
    end if
  end for
end for

% ikinci bölüm:
% Solution Values tablosu kullanılarak hesaplanmaktadır.
% Solution dizisi 0 ve 1 oluşmaktadır.
 $n = N$ 
 $c = C$ 
 $Solution = \text{zeros}[n]$ 

while  $c$  is not equal to 0
  if  $Values[n][c]$  is equal to  $Values[n-1][c]$ 
     $n \leftarrow n - 1$ 
  else
     $Solution[n-1] = 1$ 
     $c \leftarrow c - G_k[n-1]$ 
     $n \leftarrow n - 1$ 
  end if
end while

return  $Solution$ 

```

Dinamik programlamanın torba problemi çözümü için optimal olduğunu ispat edelim.

Teorem:

İlk  $n$  eşya ve sırt çantası kapasitesi  $c$  ile tanımlanan sırt çantası problemi örneği için, torbanın alabileceği maksimum değeri,  $Values[n][c]$  belirtsin.

$1 \leq n \leq N$ ,  $0 \leq c \leq C$ ,  $Values \leq [n][c]$  optimal çözümü eşitlik (4.1) kullanılarak hesaplanır.

$$Values[n][c] = \max(Values[n-1][c], V_k[n] + Values[n-1][c - G_k[n]]) \quad (4.1)$$

### İspat:

$Values[n][c]$  hesaplamak için,  $n$  tane eşyada sadece iki seçeneğimiz var. Kapasite  $c$  sırt çantasına uyan ilk  $n$  eşyanın tüm alt kümelerini iki kategoriye ayırabiliriz: i)  $n$ . eşyayı içermeyen alt kümeler, ii)  $n$ . eşyayı içeren alt kümeler.

- Eşyayı içermeyen durum: Eğer  $n$ . eşya çantaya sığmazsa, çantaya alınmaz ve bu durumda,  $Values[n][c]$  ile  $Values[n-1][c]$  eşit olur.
- Eşyayı alan durum:  $G_k$   $k$  indisli eşyanın ağırlığını belirlemek üzere;  $G_k[n] < c$ . olduğu durumda,  $n$ . eşyayı kullanmazsak, maksimum değer  $Values[n-1][c]$  olur. Eğer,  $n$ . eşyayı alındığında,  $Values[n]$  değerinde bir değer kazanırız ve çantanın alabileceği kapasitesini  $G_k[n]$  kadar azalır. En iyi çözüm kalan eşyalar  $\{1,2,...,n-1\}$  ve ağırlıkları  $(c-G_k[n])$  ile yapılır.  $V_k$   $k$  indisli eşyanın değerini belirtir  $V_k[n] + Values[n-1][c-G_k[n]]$ . Ardından, ilk  $n$  parça ve sırt çantası kapasitesi  $c$  için maksimum değer,  $Values[n][c] = \max(V_k[n] + Values[n-1][c-G_k[n]], Values[n-1][c])$  şeklinde eşitlikle hesaplanır.

Daha önce, bu problem için örnek verdiğimiz örneği dinamik programlama yaklaşımıyla çözelim. Çantanın değerleri aşağıdaki Tablo 4.2’de görülmektedir. Tablo 4.3’de bu problemin dinamik programlama çözümü tablosu görülmektedir. Tablo 4.4’de ise, Tablo 4.3 değerler kullanılarak, maksimum değeri ve çözüm eşyalarının indisleri nasıl bulunduğu anlatılmaktadır.  $c$  torbanın alabileceği maksimum kapasiteyi belirtmektedir.  $v(3,c)$  toplam torbaya 3 eşya alınmaya çalışıldığında ve torbanın kapasitesi  $c$  kapasitesinde aldığı maksimum değeri belirtmektedir. Örneğin,  $v(3,9)$  olduğunda, torbaya eşya 1, eşya 2, eşya 3 konulabilir ve torbanın kapasitesi 9 kg’dır.  $v(2,9)$  torbaya eşya 1, eşya 2 torbanın kapasitesi 9 kg’dır.

Tablo 4.2: Torba problemi örneği.

| Eşya $i$ | Değer $v_i$ | Ağırlık $w_i$ |
|----------|-------------|---------------|
| 1        | 600         | 6             |
| 2        | 400         | 5             |
| 3        | 300         | 4             |

Tablo 4.3: Torba probleminde dinamik programlama çözümü tablosu.

|           | $c=0$ | $c=1$ | $c=2$ | $c=3$ | $c=4$ | $c=5$ | $c=6$ | $c=7$ | $c=8$ | $c=9$ |
|-----------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| $v(0, c)$ | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     |
| $v(1, c)$ | 0     | 0     | 0     | 0     | 0     | 0     | 600   | 600   | 600   | 600   |
| $v(2, c)$ | 0     | 0     | 0     | 0     | 0     | 400   | 600   | 600   | 600   | 600   |
| $v(3, c)$ | 0     | 0     | 0     | 0     | 300   | 400   | 600   | 600   | 600   | 700   |

Tablo 4.4: Dinamik Programlama ile örneğin çözümü.

|                                                                                                                                                                                                              |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| $k = 3: v(3,9) \neq v(2,9), \quad \text{Solution}(3) = 1$<br>$c = c - w_3 = 9 - 4 = 5$<br>$k = 2: v(2,4) \neq v(1,4), \quad \text{Solution}(2) = 1$<br>$c = c - w_3 = 5 - 5 = 0$<br>$\text{Solution}(1) = 0$ |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### 4.3. Torba Problemi Çözümü Tabanlı Yeniden Örneklem

Bu bölümde, torba problemi çözümü tabanlı geliştirilen özgün yeniden örneklem algoritması hakkında bilgi verilmektedir. Yeniden örneklem algoritmasında, zayıf olan parçacıkların sistemden çıkarılması ve ağırlıkları yüksek olan parçacıkların çoğaltılması gerekir.

Dinamik programlama yaklaşımı kullanılarak, yeniden örneklem algoritması üretmek için aşağıdaki adımlar uygulanmaktadır.

- Yeniden örneklem algoritmasında belirlenen eşik değeri, torba problemindeki belli bir genişletme katsayısı ile çarpılarak, ağırlık kısıtlaması olarak varsayılabılır.
- Değer/ağırlık oranı ile parçacıkların ağırlıkları ile doğru orantılı olacak şekilde çözüm geliştirilmelidir. Olası çözümlerden biri, parçacıkların ağırlık oranı sabit tutulup, parçacıkların değerleri ise parçacıkların ağırlık oranının karesi alınır, böylece başlangıçtaki oran korunmuş olur.

- Torba problemi çözmek için hem eşyaların ağırlıkları ve hem de eşyaların değerleri doğal sayı olmalıdır. Bu sebepte en yakın doğal sayıya ağırlıklar yukarı yuvarlanmalıdır.
- 0 ve 1 torba probleminde, optimal sonuçta hangi eşyaların çantada yer alıp almayacağı sonucunu geri döner. Yeniden örnekleme algoritması problemde hangi parçacıkların yeniden örneklenip örneklenmeyeceğini belirlenmiştir.

Bu yaklaşımın en önemli avantajı parçacıkların ağırlıklarının önemli bilgi olarak kullanılmasıdır. Probleme göre ağırlık kısıtlaması iyileştirme parametresi olarak kullanılabilir. Yapılan deneysel sonuçlarda, torba problemi çözümlü yeniden örnekleme algoritması, ikinci bölümde detayları anlatılan diğer algoritmalarından ve önerdiğimiz Açgözlü algoritmadan daha iyi performans göstermiştir. Parçacık filtresinde hedef takibinde, parçacık filtresinin yeniden örnekleme adımı bu algorithmada çok kolaylıkla kullanılabilir.

Diğer yeniden örnekleme algoritmaları gibi parçacıkların ağırlıkları giriş parametresi olacaktır. Ayrıca, eşik değeri ve genişletme katsayısı giriş parametresi olarak verilmelidir. Eşik değeri varsayılan olarak 0.5 verilebilir. Genişletme katsayısı parçacık sayısı olarak varsıyabilir. Algoritma diğer yeniden örnekleme algoritmalarında olduğu gibi zayıf parçacıkları sistemden çıkartarak, başlangıçta parçacık sayısı kadar, parçacıkların indislerini geri dönecektir. Kestirme adımıda yeniden örneklenmiş parçacıkların indisleri kullanılarak parametre kestirimi yapılacaktır.

Torba problemi çözerek önerilen algoritma adımları aşağıdaki gibidir. Parçacıkların ağırlıkları  $w_1, w_2, \dots, w_N$   $N$  tane parçacık için giriş değeri olarak alınır.

Adım 1: 0.01-0.99 arası bir eşik değeri  $T$  belirlenir.

Adım 2: Tüm parçacıkların ağırlıklarının toplamı hesaplanır.

Toplam ağırlık  $W_T = \sum_{i=1}^N w_i$

Adım 3: Her bir parçacığın ağırlığı Adım 2’de elde edilen toplam ağırlığa bölünerek normalize ağırlıklar  $W_i = w_i/W_T$  bulunur.

Adım 4: Değer/ağırlık oranı ile parçacıkların başlangıçta verilen ağırlıkları doğru orantılı olacak şekilde çözüm geliştirilmelidir. Parçacıkların ağırlık oranı

sabit tutulup, parçacıkların değerleri ise parçacıkların ağırlık oranının karesi alınarak, başlangıçtaki oran korunmuş olur. Torba problemi çözmek için hem eşyaların ağırlıkları ve hem de eşyaların değerleri doğal sayı olmalıdır. Bu sebepte en yakın yukarı doğal sayıya ağırlıklar yuvarlanmalıdır. Eşyalarının ağırlıkları  $G$  ve eşyaların değerleri  $V$  olarak ifade edilirse, Genişletme katsayısı  $E$  olarak ifade edilirse;  $G_i = \text{yukariyuvarla}(E * W_i)$ .  $V_i = (G_i)^2$  şeklinde ifade edilir. Ağırlık kısıtlaması  $C$  olarak ifade edilirse,  $C = (T * E)$  şeklinde hesaplanır.  $E$  tipik olarak  $0.5 * N$  ile  $5 * N$  arasında olabilir.

Adım 5: Adım 4’de elde edilen parçacıkların ağırlıkları, değerleri ve ağırlık kısıtlaması giriş değerleri kabul edilerek torba problemi çözülür.  $Knapsack(G_1, V_1, C, N)$  fonksiyonu kullanılır. Bu torba problemin çözümüyle hangi parçacıkların çoğaltılacağı ve hangi parçacıkların sistemden çıkarılacağı belirlenmiş olur.  $\sigma(.)$  fonksiyonu çoğaltılacak parçacıkların indisini ifade etmek üzere, Çözüm,  $\sigma(1), \sigma(2), \dots, \sigma(k)$  şeklinde ifade edilir.

Adım 6:  $\sigma(1), \sigma(2), \dots, \sigma(k)$  indisli parçacıklar ağırlıkları oranınca çoğaltılır. Her bir parçacık için çoğaltılma sayısı  $K_{\sigma(i)} = \frac{NW_{\sigma(i)}}{V_k}$  eşitliğiyle hesaplanır. Eşitlik sonucu reel sayı olduğu için tam sayıya çevirmek için yuvarlama fonksiyonları kullanılır.

Adım 7: Çoğaltma işlemi sonunda elde edilen toplam parçacık sayısı  $K_t = \sum_{i=1}^k K_{\sigma(i)}$  başlangıçtaki parçacık sayısı  $N$  ile karşılaştırılır. Yuvarlama işleminden dolayı  $K$  ve  $N$  eşit olmayabilir.  $K_t < N$  ise, başlangıçta yüksek ağırlığa sahip parçacıklar eklenir,  $K_t > N$  ise, başlangıçta düşük ağırlığa sahip indisli parçacıklar çıkartılır.

Torba problemini çözerek oluşturulan yeniden örnekleme algoritmasının çalışmasını basit bir örnek ile açıklayalım. Sistemde başlangıçta 5 adet parçacığın ağırlıkları  $\{1, 2, 3, 8, 16\}$ , olsun. Eşik değeri 0.81 olarak belirlensin. Bu örnek üzerinde algoritmanın çalışması aşağıdaki maddeler halinde sıralanmıştır.

Adım 1:  $T = 0.81$

Adım 2:  $W_T = \sum_{i=1}^N w_i = 1+2+3+8+16=30$

Adım 3:  $W_i = w_i / W_T$  şeklinde hesaplanır.

$$W_1 = 1/30; W_2 = 2/30; W_3 = 3/30;$$

$$W_4 = 8/30; W_5 = 16/30 \text{ şeklinde hesaplanır.}$$

$$\text{Adım 4: } G_1 = [4; 7; 10; 27; 53] \quad , \quad V_1 = [16; 49; 100; 729; 2809]$$

$$C = (E * T) = (0.80 * 100) = 81$$

Adım 5: Torba problemi, dinamik programlama ile  $Knapsack(G_1, V_1, C, N)$  çözülürse,  $[0; 0; 0; 1; 1]$  şeklinde sonuç elde edilir.  $\sigma(1) = 4, \sigma(2) = 5$  şeklindedir.

Yani 4 ve 5 indisli parçacıklar çoğaltılacaktır.

$$\text{Adım 6: } K_5 = NW_5/V_R = (5 * 16/30)/(24/30) = 3.33$$

$K_4 = NW_4/V_R = (5 * 8/30)/(24/30) = 1.67$ . En yakın sayıya yuvarlama fonksiyonları kullanılırsa;  $[3.33; 1.67]$  değerleri  $[3; 2]$  şeklinde yuvarlanır.

Adım 7:  $K_t = K_5 + K_4 = 3 + 2 = 5$ .  $K_t = N$  olduğundan 8. Adımda herhangi bir değişiklik yapmadan 5 indisli parçacıktan 3 adet 4 indisli parçacıktan 2 adet olmak üzere;  $[5; 5; 5; 4; 4]$  çıktı vererek algoritma sonlanır.

Tablo 4.5: Sayısal örnek için dinamik algoritma tablosu.

| $c \backslash n$ | 0 | 4  | 7  | 10  | 27  | 53   | 81   |
|------------------|---|----|----|-----|-----|------|------|
| 0                | 0 | 0  | 0  | 0   | 0   | 0    | 0    |
| 1                | 0 | 16 | 16 | 16  | 16  | 16   | 16   |
| 2                | 0 | 16 | 49 | 49  | 65  | 65   | 65   |
| 3                | 0 | 16 | 49 | 100 | 165 | 165  | 165  |
| 4                | 0 | 16 | 49 | 100 | 729 | 894  | 894  |
| 5                | 0 | 16 | 49 | 100 | 729 | 2809 | 3538 |

$G_1 = [4; 7; 10; 27; 53]$  ,  $V_1 = [16; 49; 100; 729; 2809]$  ,  $C = 81$  olmak üzere,  $N = 1$  olduğunda, torbaya sadece parçacık 1 alınır.  $N = 2$  olduğunda, torbaya sadece parçacık 1 ve parçacık 2 yerleştirilir.  $N = 5$  olduğunda, tüm parçacıklar torbaya alınır.  $C$  satırı, torbanın ağırlık kısıtlaması  $c$  olduğunda, torbaya yerleştirilebilecek nesnelerin maksimum değerini temsil eder. Verilen sayısal örnek için  $C$  kapasitesi 81 idi ve benzer şekilde eşyaların ağırlıkları şu şekildedir:  $G_1[0] = 4$ ,  $G_1[1] = 7$ ,  $G_1[2] = 10$ ,

$G_1[3] = 27$ ,  $G_1[4] = 53$ . Tablo 4.1’de sözde kodu verilen, Torba problemi çözümü algoritmasının, ikinci bölümü kapasite indisi  $c$ , eşya sayısı  $n$  ve çözüm dizisi, *Solution* dizisine 0 başlangıç değerleri atanır. Bu değerler sırasıyla 81, 5 ve [0; 0; 0; 0;0] olarak ayarlanır. Torba problemi çözümü, algoritmasının ikinci bölümü uygulandığında, Tablo değerleri ve *Solution* dizisi aşağıdaki şekilde güncellenir:

$$Values[5][81] \neq Values[4][81], 3538 \neq 894$$

$$Solution[4]=1, n=5-1=4, c = 81- G[4] =81-53=27.$$

$$Values[4][27] \neq Values[3][27], 729 \neq 165$$

$$Solution[3]=1, n=4-1=3, c= 27-G[3] =27-27=0.$$

$C=0$  olduğundan, while döngüsü sona erer ve algoritma [0;0;0;1;1] değerleriyle *Solution* dizisini döndürür. Yani 4. ve 5. parçacıklar seçilip çoğaltılmalı ve toplam değerleri 3538’dir. Diğer parçacıklar sistemden çıkarılır.

## 4.4. Torba Problemi Çözümü Tabanlı Yeniden Örneklemeye Algoritması Testleri

### 4.4.1. Lineer olmayan modelde hedef takibi

Torba problemi çözümü tabanlı yeniden örneklemeye algoritmasının performansını ölçmek için [1] ve [25]’de verilen ve yeniden örneklemeye algoritmasının performansını ölçmek için kullanılan ve önerilen Denklem (4.2) ile Denklem (4.5)’de arasında verilen süreç ve ölçüm fonksiyonları kullanılmıştır. Parçacık filtresi olarak [1]’de verilen jenerik parçacık filtresi kullanılmıştır.

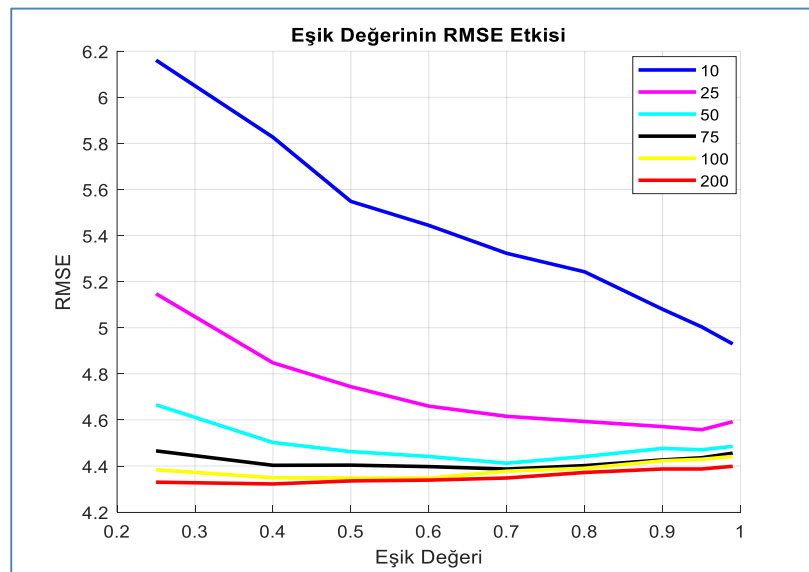
$$x_k = \frac{x_{k-1}}{2} + \frac{25x_{k-1}}{1 + x_{k-1}^2} + 8\cos(1.2k) + v_k \quad (4.2)$$

$$Z_k = \frac{x_k^2}{20} + r_k \quad (4.3)$$

$$v_k \sim \mathcal{N}(0, Q) \quad (4.4)$$

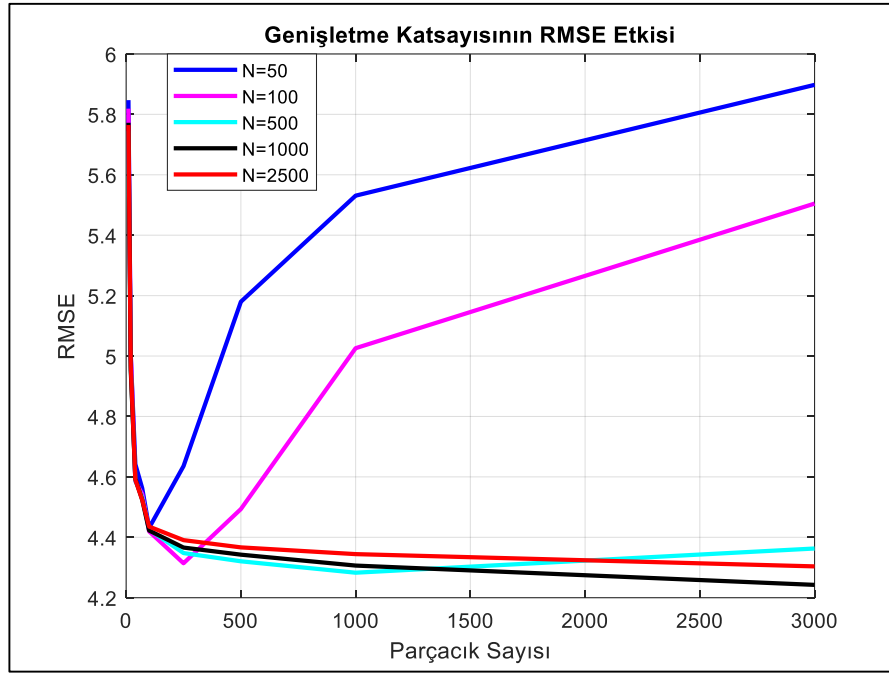
$$r_k \sim \mathcal{N}(0, R) \quad (4.5)$$

Burada,  $x_k$  durum denklemini,  $Z_k$  ölçümü,  $v_k$  süreç gürültüsünü ve  $r_k$  ise ölçüm gürültüsünü belirtmektedir. Bu çalışmada,  $v_k$  sıfır ortalamalı  $Q$  kovaryanslı bir Gauss gürültüsü,  $r_k$  ise sıfır ortalamalı  $R$  kovaryanslı bir Gauss gürültüsü olarak alınmıştır. Monte Carlo benzetimi testleri yapılmıştır. Şekil 4.1’de bu hareket modeline sahip bir hedefin parçacık filtreleri ile 100 sn. boyunca izlenmesi görülmektedir. Ölçüm gürültüsü varyansı 10, standart sapması 3.16 ve süreç gürültüsü standart sapması 1 ve parçacık sayısı 100 olarak alınmıştır. Ölçüm zaman aralığı ise 1sn’dir. Şekil 4.1’ de eşik değerinin farklı parçacık sayıları için ortalama karesel hataya etkisi görülmektedir. 200 Monte Carlo benzetimi testi yapılmıştır. Parçacık sayısının 10 gibi düşük değere sahip olması durumlarında en iyi performansı 0.99 eşik değeri vermiştir. 25 parçacık sayısında 0.95 eşik değeri, 50 ve 75 parçacık sayısında 0.70 eşik değeri, 100 parçacık sayısında en iyi kestirim performansı 0.40 eşik değeri verirken, 200 parçacık sayısında en iyi kestirim performansı 0.25 eşik değeri ile elde edilmiştir. Eşik değeri, az sayıda parçacık kullanılacaksa eşik değerini yüksek seçmek gerekmektedir. Bunun temel sebebi sürekli yüksek ağırlıklı parçacıkların seçilmesi örnek fakirleşmesine neden olmaktadır [1]. Daha çok parçacığın olduğu durumlarda ise eşik değeri daha düşük seçilebilmektedir. Burada genişletme katsayısı 100 olarak alınmıştır. Farklı parçacık sayısında en ideal eşik değerleri 0.5-0.8 arasındadır. Eşik değeri olarak 0.6 olarak belirlenebilir.



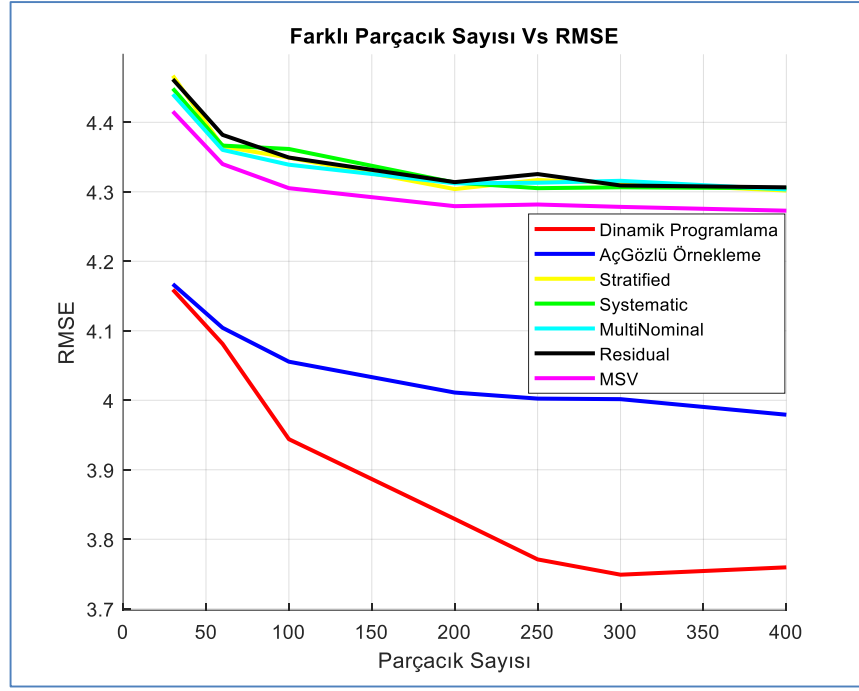
Şekil 4.1: Eşik değerinin ortalama karesel hataya etkisi.

Şekil 4.2’de genişletme katsayısının hataya etkisi görülmektedir.  $0.5*N$  ile  $5*N$  arasında olduğu sürece konum hatası fazla değişmektedir. Parçacık filtresinden çok düşük olması durumunda RMSE artmaktadır. Örneğin, 1000 parçacıkda 50 ve 100 seçilmesi hatanın yüksek olduğu görülmektedir.  $E$  genişletme katsayısı en maksimum parçacık sayısı olarak seçilebilir. 50 ile 400 arasında farklı parçacık sayısında testler yapılacaksa,  $E=400$  olarak seçilebilir. Deneylerimizde  $E=400$  olarak seçilmiştir.



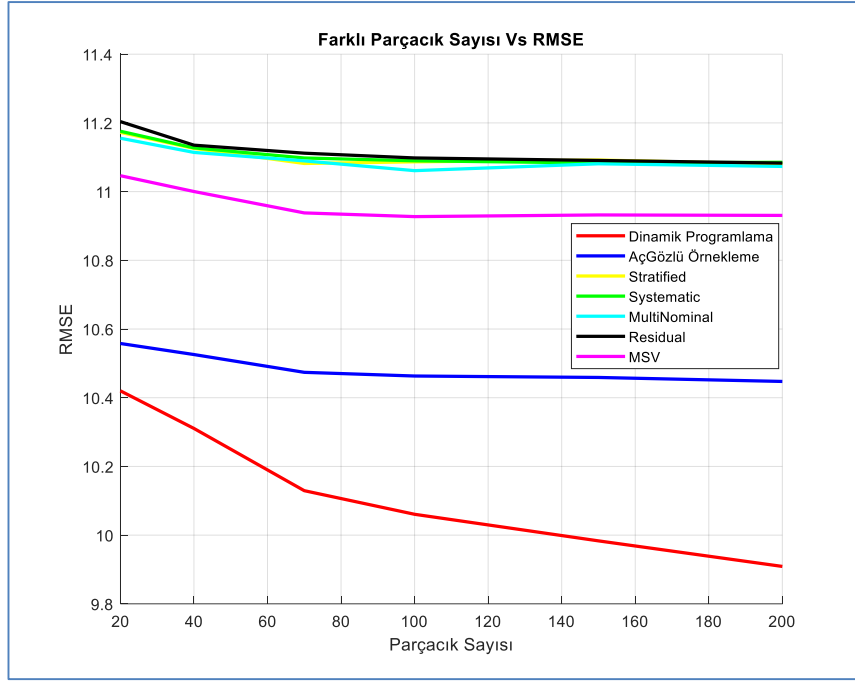
Şekil 4.2: Genişletme katsayısı etkisi.

Şekil 4.3’de farklı parçacık sayıları ve farklı yeniden örnekleme algoritmaları kullanılarak elde edilen hedef konumlarının ortalama karesel hataları görülmektedir. Parçacık sayısının artması, ortalama karesel hatanın azalmasına, hedef konumunun daha doğru kestirilmesini sağlamaktadır. Ölçüm ve süreç gürültüleri Gauss dağılımlıdır. Ölçüm gürültüsünün varyansı 20, standart sapması 4.47 ve süreç gürültüsünün standart sapması 1’dir. 100 örnekli Monte Carlo benzetimi testi yapılarak için elde edilen sonuçlarda, en iyi performansı dinamik programlama oluşturulan önerilen algoritma vermiştir. 25,60,100, 200, 300, 400 parçacık sayısında 0.60 eşik değerinde önerilen algoritma vermiştir.



Şekil 4.3: Farklı parçacık sayılarında farklı algoritmalarının performansı.

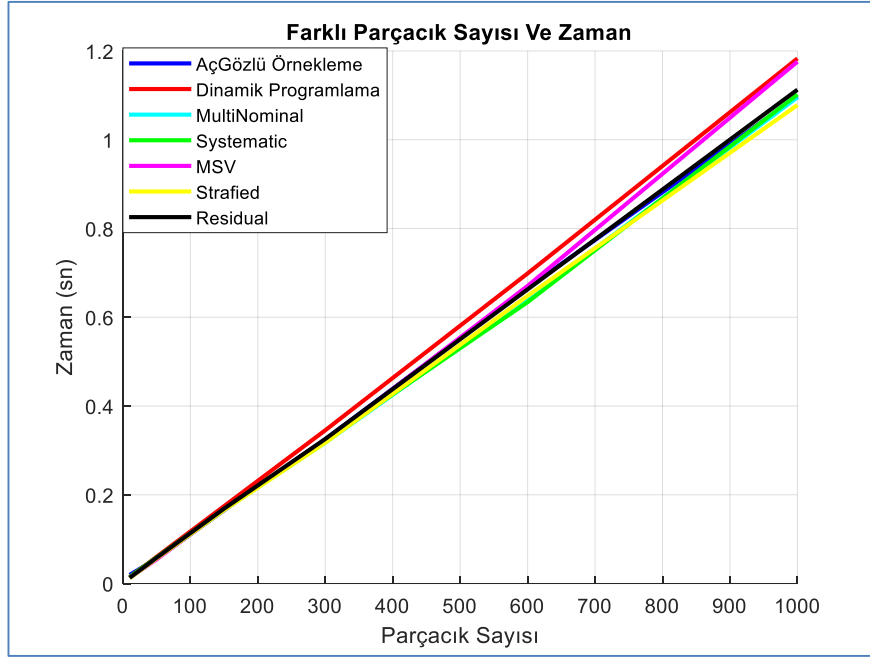
Şekil 4.4’de gürültü değerinin ortalama karesel hataya etkisi tespit etmek için testler yapılmıştır. 100 Monte Carlo benzetimi testi yapılmıştır. Şekil 4.3 ve Şekil 4.4 karşılaştırılırsa, gürültünün standart sapmasının artması beklendiği gibi ortalama karesel hatayı artırmıştır. En iyi performansı 0.60 eşik değerinde torba problemi çözümü tabanlı önerilen algoritma vermiştir. Diğer algoritmalar 200 parçacık sayısında 11.0 ortalama karesel hata verirken, Açgözlü algoritmanın ortalama karesel hata değeri 0.60 eşik değerinde 10.6 ve önerilen algoritma 0.60 eşik değerinde RMSE 9.9’tür.



Şekil 4.4: Ölçüm gürültüsünün ortalama karesel hataya etkisi.

Önerilen algoritmada çalışma zamanını etkileyen en önemli kısıtlama parçacıkların sayısı ( $N$ ) ve ağırlık kısıtlamasıdır ( $C$ ). Algoritmanın karmaşıklığı  $O(N \cdot C)$ . Problemimizde ise, ağırlık kısıtlaması sınırlandırılmakta ve eşik değerine göre değişmektedir. Örneğin 0.25 eşik değeri belirlendiğinde, genişletme katsayısı 100 olduğunda, ağırlık kısıtlaması 25 olmaktadır. Ağırlık kısıtlamasının maksimum 100 olacağı düşünülürse, bu da algoritmanın maksimum çalışma zamanının  $100 \cdot N$  olacağını göstermektedir. Bu sonuca, göre algoritmanın karmaşıklığı Big O notasyonu göre  $N$ 'dir. Yani, önerilen algoritma yaklaşık doğrusal zamanda çalışmaktadır. Parçacık sayısı arttığında, önerilen algoritmanın çalışma süresi aynı oranda artmaktadır. Şekil 4.5'de algoritmaların çalışma süreleri görülmektedir. Parçacık filtresi algoritmasında hangi yeniden örnekleme olursa olsun algoritmaların çalışma süreleri benzerdir. Torba problemin, dinamik programlama yaklaşımı ile çözümü çok az daha yavaştır. Yeniden örnekleme problemi, torba problemine dönüştürüldükten sonra, parçacıklar değer/ağırlık oranı göre sıralanarak, Açgözlü yaklaşımla çözüm de yapılabilir. Yüksek parçacıklarda,  $N=1000$  daha büyük olan durumlarda, genişletme katsayısını ( $E$ ) yüksek belirlemek gerekir. Bu durumda, ağırlık kısıtlaması  $C$ 'de,  $E \cdot T$  eşit olduğundan yüksek olmaktadır. Ağırlık kısıtlamasının yüksek olduğu bu tür durumlarda, torba problemi, Açgözlü yaklaşım

kullanılarak, big O notasyonuna göre,  $O(N \cdot \log(N))$  karmaşıklıkla daha hızlı olarak ve yaklaşık olarak da çözülebilmektedir.



Şekil 4.5: Algoritmaların çalışma süreleri.

#### 4.4.2. Kartezyen koordinatlarda hedef takibi

Üç-boyutlu bir düzlemde hareket eden sabit hızlı hedefin Kartezyen koordinatlardaki konum ölçümlerine ilişkin hareket modelinin durum modeli, ve ölçüm modeli sırasıyla Denklem (4.5) ile (4.11) arasında verilmektedir.  $X_k$  durum vektörü  $k$  anındaki  $x$ ,  $y$  ve  $z$  yönündeki konumları ile hızlarını içermektedir. Ölçüm vektörü  $Z_k$  radar tarafından ölçülen hedef konumlarını içermektedir.  $v_k$  süreç gürültüsünü,  $r_k$  ise ölçüm gürültüsünü belirtmektedir. Bu çalışmada,  $v_k$  sıfır ortalamalı  $Q$  kovaryanslı bir Gauss gürültüsü,  $r_k$  ise sıfır ortalamalı  $R$  kovaryanslı bir Gauss gürültüsü olarak alınmıştır.  $dt$  ölçüm zaman aralığını belirtmektedir.

$$X_{k+1} = FX_k + v_k \quad (4.5)$$

$$Z_k = HX_k + r_k \quad (4.6)$$

Burada,

$$X_k = [x_k \ y_k \ z_k \ \dot{x}_k \ \dot{y}_k \ \dot{z}_k]^t \quad (4.7)$$

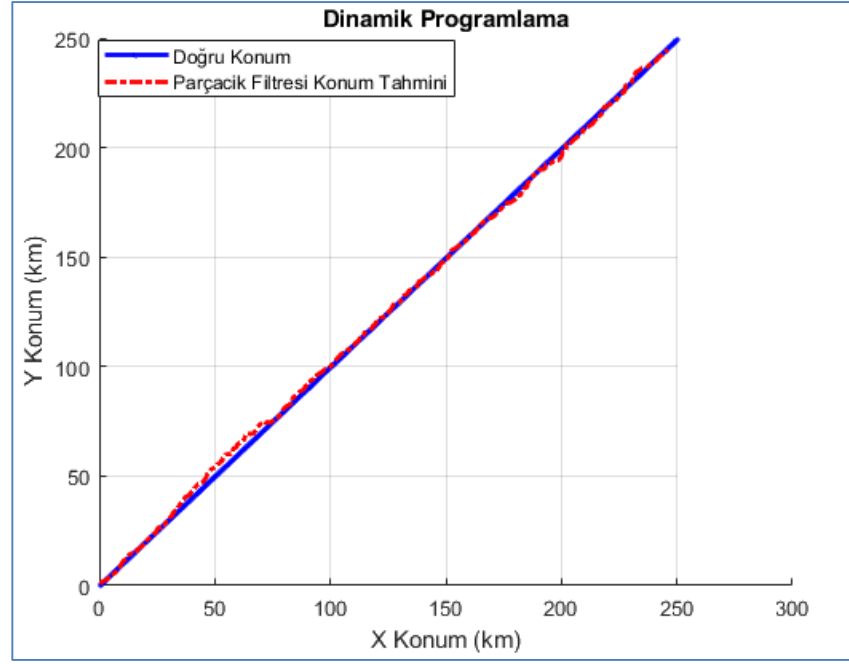
$$F = \begin{bmatrix} 1 & 0 & 0 & dt & 0 & 0 \\ 0 & 1 & 0 & 0 & dt & 0 \\ 0 & 0 & 1 & 0 & 0 & dt \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.8)$$

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \quad (4.9)$$

$$v_k \sim \mathcal{N}(0, Q_k) \quad (4.10)$$

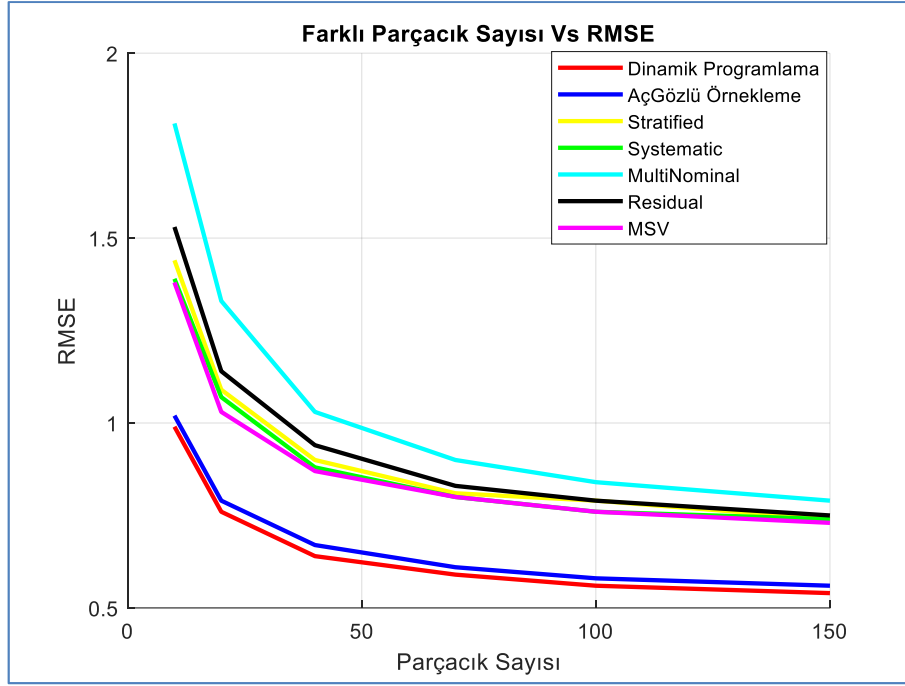
$$r_k \sim \mathcal{N}(0, R_k) \quad (4.11)$$

Algoritmaların performans karşılaştırmaları için monostatik bir radar ile uçak takibi senaryosu ele alınıp sentetik veriler üretilmiştir. Uçağın hareketini modellemek için Z yönünde 6,5 km sabit yükseklikte olduğunun varsayımı altında, X ve Y yönündeki başlangıç referans noktaları 0 alınarak uygun datalar üretilmiştir. X ve Y yönünde başlangıç hızları 100 m/s, zaman aralığı 1 saniye olmak üzere, 100 saniye boyunca hareket eden hedef izlenmiştir. Süreç gürültülerinin 0 olmadığı durumlar için örnekli Monte Carlo benzetimi testleri yapılmıştır. Süreç gürültüleri, standart sapmayla orantılı Gauss gürültüleridir. Ölçümlere standart sapmayla orantılı Gauss gürültüsü de eklenmiştir. 20 varyanslı ve 4.47 standart sapmalı ölçüm gürültüsünde, 1 standard sapmalı süreç gürültüsünde, Şekil 4.6’da bu parametreler için üretilen bir hareketlinin parçacık filtrelerinde torba çözümü tabanlı yeniden örnekleme algoritması kullanılarak izlenmesi görülmektedir. Genişletme katsayısı  $E=400$  ve  $T=0.6$  olarak alınmıştır.



Şekil 4.6: 3 boyutlu kartezyen koordinatlarda hareket izleme.

Şekil 4.7’de 4 standart sapmalı ölçüm gürültüsü ve 1 standard sapmalı süreç gürültüsünde, kartezyen koordinat hareket modelinde farklı parçacık sayısında yeniden örnekleme algoritmalarının sonuçları görülmektedir. RMSE birimi km’dir. 100, 150, 200 parçacık sayısında en iyi sonucu torba problemi çözerek, önerilen algoritma dinamik yeniden örnekleme algoritması vermiştir. Torba problemi çözerek, önerilen algoritma vermiştir. 20, 40, 70 parçacık sayısında Açgözlü örnekleme algoritması vermiştir. Bu parçacık sayılarında en iyi sonucu önerilen algoritma vermiştir.



Şekil 4.7: 3 boyutlu kartezyen koordinatlarda algoritmaların sonuçları.

#### 4.4.3. Sürekli dönüş modelinde hedef takibi

X ve Y yönünde 30 km çapında dairesel hareket yapan saatte 720 km ilerleyen bir hedef için parçacık filtreleri ile hareketinin izlenmesi Şekil 4.8’de verilmiştir. Hedefin X ve Y yönündeki hızları değişken olmasına rağmen, açısal hızı sabittir. Ölçüm gürültüsü standard sapması 1, süreç gürültüsü standart sapması 0.1’dir. Aç gözlü yeniden örnekleme algoritması kullanılmıştır. Bu dairenin etrafında 1 tur atabilmek için 300 saniye geçmektedir. Zaman aralığı 1 saniye olarak alınmıştır. 100 Monte Carlo benzetimi testi yapılmıştır. Şekil 4.8’da bu parametreler için üretilen bir hareketlinin parçacık filtrelerinde torba çözümlü yeniden örnekleme algoritması kullanılarak izlenmesi görülmektedir.  $E=400$  ve  $T=0.6$  alınmıştır. Burada  $R$  çapı belirtmektedir.  $V$  hedefin hızını belirtmektedir. 720 km/h dir. Saati saniye çevirmek için 3600 ile çarpılmıştır.  $N_{sample}$  değeri Eşitlik (4.5) formülle hesaplanmaktadır. Tablo 4.6’de Sürekli dönüş modelindeki x ve y konumların nasıl elde edildiği görülmektedir  $q_x$  x konumlarına gürültüyü,  $q_y$  y konumlarına eklenen gürültüyü belirtmektedir.  $x_k$  x konumlarını  $y_k$  y konumlarını belirtmektedir.

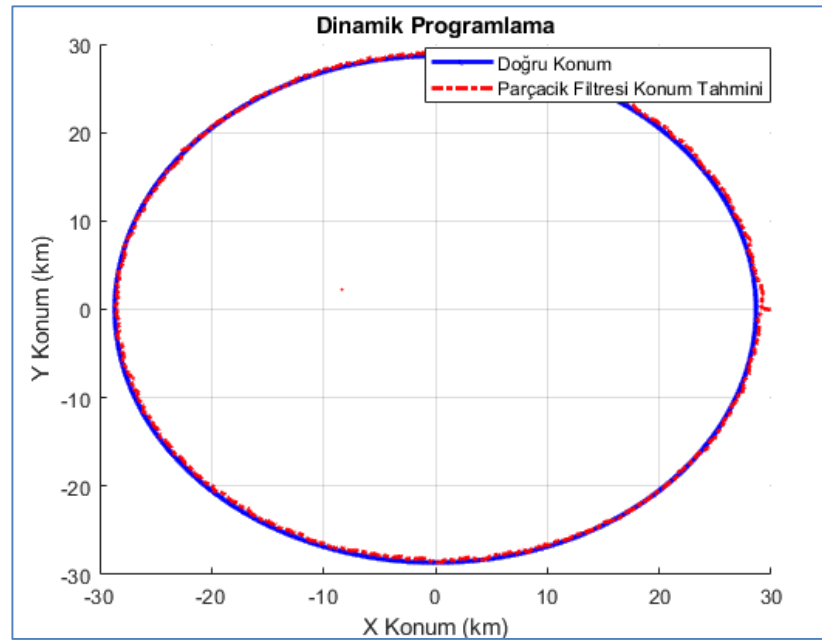
$$N_{sample} = \frac{2 * \pi * R * 3600}{V * 2 * dt} \quad (4.5)$$

Tablo 4.6: Sürekli dönüş modelindeki konumların hesaplanması.

```

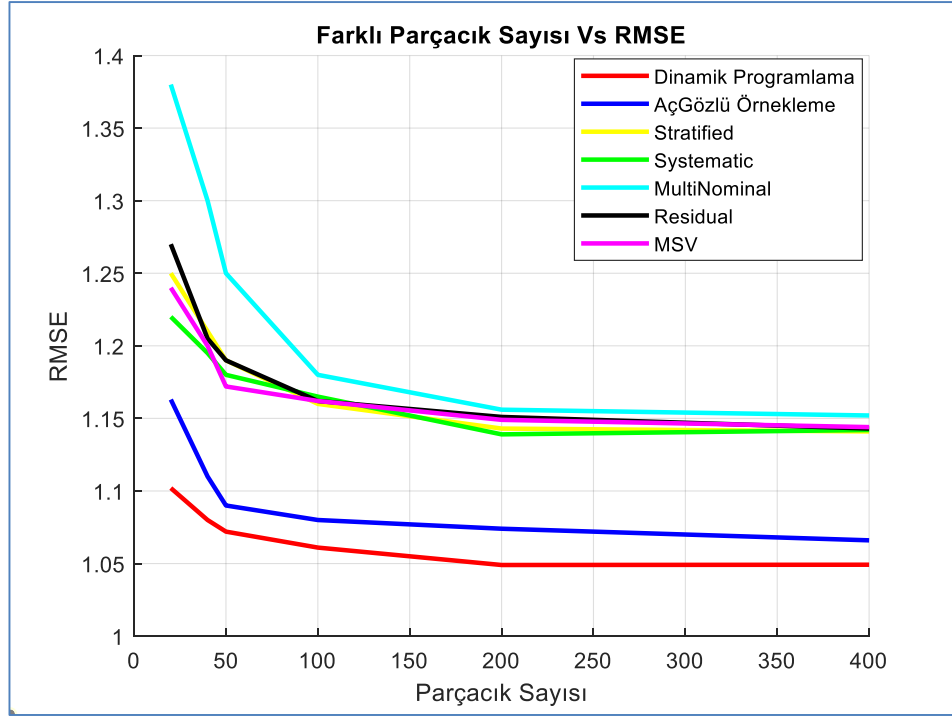
for i=0:pi/N_sample:2*pi
     $x_k = R * \cos(i) + q_x$ 
     $y_k = R * \sin(i) + q_y$ 
end

```



Şekil 4.8: Dairesel hareket modelinde hareket izleme.

Şekil 4.9’da dairesel hareket modelinde farklı parçacık sayısında yeniden örnekleme algoritmalarının sonuçları görülmektedir. Ölçüm gürültüsü standart sapması 1 ve süreç gürültüsünün standart sapması 0.1 dir. Tüm farklı parçacık sayısında en iyi performansı Açgözlü algoritma ve dinamik programlama vermiştir. RMSE birimi km’dir.



Şekil 4.9: Dairesel hareket modelinde 1 standart sapmalı ölçüm gürültüsünde algoritmaların sonuçları.

## 4.5. Genel Değerlendirme

Bu bölümde, Torba problemi çözümü yaklaşımlı bir yeniden örnekleme algoritması sunuldu. Torba problemi çözümlü yeniden örnekleme algoritmasının performansı etkileyen parametre eşik değeridir. Az sayıda parçacık sayısında bu eşik değeri yüksek olabilir. Daha fazla parçacık sayısında daha düşük seçilmelidir. Lineer olmayan hareket modeli, sabit hızlı kartezyen koordinatlarda hareket modeli, dairesel hareket modelinde en iyi performansı farklı parçacık sayılarındaki ve farklı gürültü değerlerinde torba problemi çözümlü algoritma vermiştir. O algoritmanın performansına en yakın algoritma Açgözlü yeniden örnekleme algoritmasıdır.

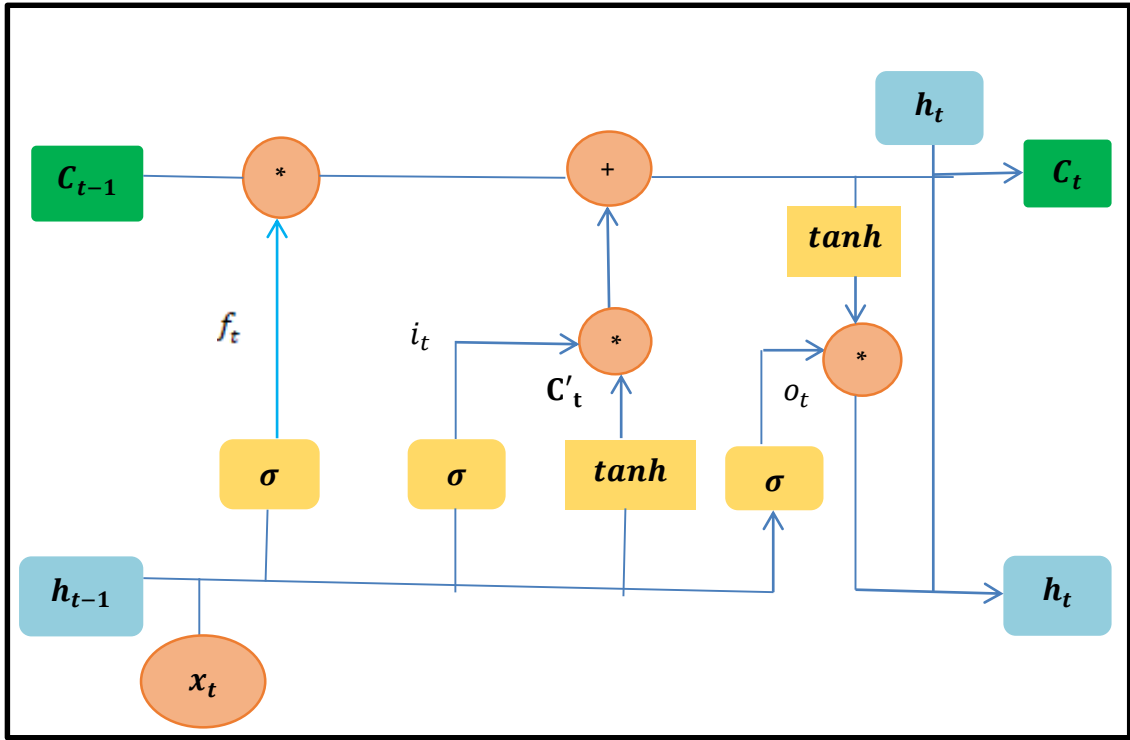
## 5. LSTM DERİN ÖĞRENME ALGORİTMASI TABANLI YENİDEN ÖRNEKLEME ALGORİTMASI

Bu bölümde, Uzun Kısa Vadeli Bellek (LSTM, Ing: Long Short-Term Memory) diziden diziye eşleme özelliğinden esinlenerek, özgün yeniden örnekleme algoritmalarının geliştirilmesine yönelik bir çerçeve sunulmaktadır. LSTM günümüzde, birçok diziden diziye problemini çözmektedir. LSTM mimarileri konuşma/metin işleme konularında oldukça iyi sonuçlar vermektedir. Doğal dil işleme alanında, dil modeli için kullanılmaktadır [60,61]. LSTM Google tarafından 2016 yılında, diziden-diziye problem olan makine çevirisindeki nöral makine çevirisinde kullanılmıştır [62]. Cümleleri isim, sıfat, zamir, fiil gibi etiketleme probleminde de kullanılmaktadır [63]. İmla düzeltici sistemi [64], duygu metinlerin sınıflandırma [65], resimleri yazılandırma [66], metin üretme [67] gibi problemlerde kullanılmaktadır.

Bu çalışmada ise, LSTM tabanlı yeniden örnekleme algoritmaları geliştirmek için, parçacıkların ağırlıkları doğal sayılara dönüştürülür ve problemten bağımsız olarak ve rastgele giriş dizileri oluşturulur. Çıkış dizileri ise, yeniden örnekleme algoritmaları sonucunda oluşan diziler olmaktadır. Önerilen yaklaşımın etkinliğini göstermek üzere bu bölümde ikinci bölümde detayları verilen Systematic ve Stratified yeniden örnekleme algoritmaları üçüncü bölümde önerdiğimiz Açgözlü yeniden örnekleme algoritması algoritmaları kullanılarak çıkış dizileri elde edilerek farklı parçacık sayıları için LSTM modelleri eğitilmiştir. Öncelikle eğitim esnasındaki parametrelerin etkisi araştırılmıştır. Gizli birim sayısının etkisi, Tekrarlayan Sinir Ağları (RNN, Ing: Recurrent Neural Network) ile LSTM performans karşılaştırılması, batch büyüklüğü (blok büyüklüğü), tur sayısının etkisi, katman sayısının etkisi gibi farklı parametrelerin etkisi araştırılmıştır. Daha sonra ise, oluşturulan LSTM tabanlı yeniden örnekleme algoritmalarıyla, orijinal yeniden örnekleme algoritmaları doğrusal olmayan problem üzerinde, farklı parçacık sayısı ve gürültülerde test edilmiştir. LSTM algoritmasıyla üretilen algoritmaların bütün durumlarda, üretildiği orijinal algoritmalarla benzer performansa sahiptir. Ayrıca gürültülü eğitim verisiyle eğitildiğinde LSTM performansına bakılmıştır. LSTM gerçek değerleri bulabildiği gözlemlenmiştir.

## 5.1. Uzun Kısa Vadeli Bellek Ağları (LSTM)

LSTM, zamana bağımlı, bir dizi oluşturan veriler için kullanılabilen tekrarlı sinir ağlarıdır. Tekrarlayan Sinir Ağları (RNN, Ing: Recurrent Neural Network) veri sürekliliğini sağlayan iterasyonlara sahiptir. Bu iterasyonlar, verinin  $t$  zamanından  $t'$  zamanına geçmesini sağlar [68]. Zincir şeklinde sıralanan yapısıyla, dizi şeklinde ve sıralı veri yapıları için oldukça uygundur. Her bir RNN hücresi bir  $t$  zamanındaki veriyi giriş olarak alırken, öncesindeki hücreden geçmiş olan  $t$  zamanlarına ait geçmiş verileri de alabilmektedir [68]. LSTM'ler RNN'lerin özelleşmiş bir hali olan algoritmalarıdır. Her bir LSTM hücresi iterasyonlar halinde kullanılarak LSTM ağlarını oluşturmaktadır. LSTM hücrelerinde bulunan kapılar ile, giriş verisinin ve önceki hücrelerden gelen verinin akışı sağlanmaktadır [68]. LSTM hücresi blok diyagramı Şekil 5.1'de görülmektedir [69].



Şekil 5.1: LSTM hücresi blok diyagramı.

Tipik bir LSTM hücresinin denklemleri Eşitlik (5.1) ile Eşitlik (5.8) arasında verilmiştir [69]. Bu eşitliklerde,  $x_t = t$  anındaki giriş değeri,  $h_{t-1}$  = Önceki hücre çıktısı,  $C_{t-1}$  = Önceki hücre hafızası,  $h_t$  = Şimdiki hücre çıktısı,  $C_t$  = Şimdiki hücre hafızası,  $i_t$  giriş kapısını,  $f_t$  unut kapısını,  $o_t$  çıkış kapısını,  $C'_t$  aday katmanını

belirtir.  $U$  ve  $W$  her bir kapıdaki ağırlık vektörleridir.  $b$  bu kapılardaki sabitleri belirten vektördür.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (5.1)$$

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (5.2)$$

$$i_t = \sigma(x_t U^i + h_{t-1} W^i + b^i) \quad (5.3)$$

$$f_t = \sigma(x_t U^f + h_{t-1} W^f + b^f) \quad (5.4)$$

$$o_t = \sigma(x_t U^o + h_{t-1} W^o + b^o) \quad (5.5)$$

$$C'_t = \tanh(x_t U^c + h_{t-1} W^c + b^c) \quad (5.6)$$

$$C_t = \tanh(f_t C_{t-1} + i_t C'_t) \quad (5.7)$$

$$h_t = \tanh(C_t) * o_t \quad (5.8)$$

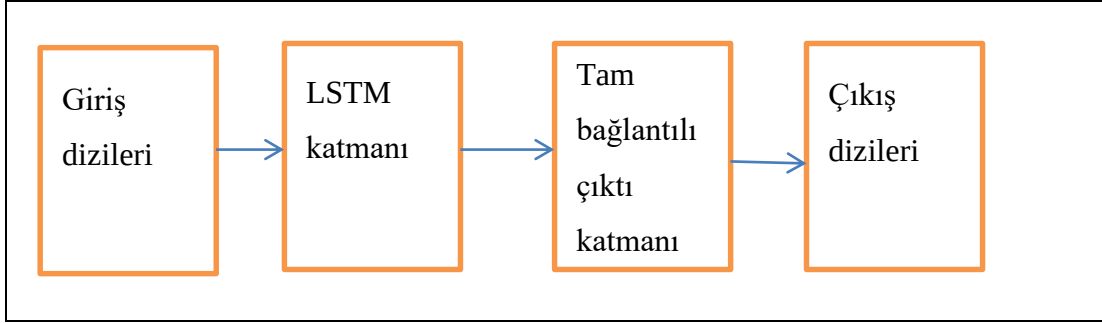
Tipik olarak, bir LSTM hücresinde; giriş kapıları, çıkış kapıları, unut kapıları, sigmoid aktivasyon kapıları gibi farklı kapılar bulunabilir [68,69]. Giriş kapısı,  $C'_t$  hangi bilgilerin güncelleneceğine karar verir. Çıkış kapıları, çıkış verisini aktarmak amacıyla kullanılır. Şimdiki hücre çıktılarını  $h_t$ , o anki hücre hafızasıyla  $C_t$  ve  $o_t$  kullanılarak belirlenir. Herbir zaman adımında,  $h_t$  ve  $C_t$  güncellenmektedir. Bu iki vektörün boyutu eşittir.

Unut kapıları, önceki hücreden gelen verinin ne kadarlık kısmının unutulması gerektiğine karar vermektedir [68]. Eşitlik 5.4'de görüldüğü üzere,  $x_t$  şimdiki giriş değeri,  $h_{t-1}$  önceki hücre çıktısı alınarak ve ağırlık katsayıları çarpılarak bir değer elde edilir. Bu değer, sigmoid aktivasyonundan geçirilir. Eşitlik 5.1'deki sigmoid fonksiyonundan geçirilen değerler 0 ile 1 arasında değerler almaktadır. Unut kapısında ise, 0 ve 0 yakın değerler o bilginin unutulması gerektiğini belirtir. 1 ve 1

yakın değerler o bilginin hatırlanması gerektiğini belirtmektedir. RNN hücreleri tek bir *tanh* fonksiyonundan oluşurken, LSTM algoritmasında ise, bu çıkış katmanına ek olarak unut kapısı, giriş ve sigmoid aktivasyon kapıları olmak üzere kapılar bulunmaktadır. Yapay sinir ağlarında, öğrenme katsayısı kullanılarak, türev alınmaktadır ve türeve öğrenme katsayısı kullanarak ağırlıklar güncellenmektedir. Hata fonksiyonu en küçük yapan katsayılar öğrenilmektedir. RNN algoritması da dizi tahminde kulanabilen bir algoritmadır. Ancak, RNN algoritması, uzun dizilerde türevlerin sıfıra yaklaşması probleminden ötürü (Ing: Vanishing Gradient), uzun geçmişteki hatırlanması gereken bilgiyi kullanmada problemleri vardır [69]. Türevler sıfıra yaklaştığı için ilk katmanlarda ağırlık güncellemesini çok az yapmaktadır. LSTM algoritmasında, RNN algoritmasındaki türevlerin yok olması problemi kapılama yapılarak aşılmalıdır. Böylece, unutulmaması gereken bilgiler, 1 ve 1'e yakın değerler verilerek o bilgiler taşınmaktadır ve tahmin yapılırken kullanılmaktadır. Google çalışan araştırmacıların 2015 yılında yaptığı bir çalışmada, literatürde var olan RNN mimarilerinden türetilerek ve aktivasyon kapılarındaki fonksiyonlar değiştirilmiştir. 10000 farklı hücre eşitliği kullanarak ve farklı hiperparametreler ile yapılan bu çalışmada LSTM algoritmasından tüm görevlerde daha iyi performansla sahip olan hücre eşitliğine rastlanmamıştır [69]. Aynı zamanda, yalnızca bir LSTM kapısı devre dışı bırakılarak elde ettiği eşitliklerde, dil modeli hariç tüm görevlerde unut kapısının en önemli kapı olduğu, daha sonra ise giriş kapısının önemli olduğu ve en az önemli kapının çıktı kapısı olduğu gözlemlenmiştir [69]. LSTM katmanında hesaplanan ağırlık katsayısı sayısı,  $4 * (\text{özellik vektörü boyutu} + (\mathbf{h}_t \text{ boyutu} + 1)) * (\mathbf{h}_t \text{ boyutu})$  ile hesaplanır. Örneğin, 25 giriş dizi uzunluğuna sahip bir dizi, 0 ile 100 arasında doğal sayı değerlerini içeriyorsa, özellik vektörü boyutu 101 olur.  $\mathbf{h}_t$  boyutu 100 olduğunda, toplam LSTM katmanında hesaplanan ağırlık katsayısı  $4 * (101 + (100 + 1)) * 100 = 80800$  olarak bulunur. Giriş dizi uzunluğunun hesaplanan ağırlık katsayısına etkisi yoktur. Ancak eğitim süresini etkilemektedir.

Şekil 5.2'de diziden diziye problem çözümü için LSTM katmanın kullanan blok diyagramı görülmektedir. Herbir zaman adımında, LSTM hücresindeki,  $\mathbf{h}_t$  ve  $\mathbf{C}_t$  değerleri güncellenmektedir. Giriş dizisi uzunluğu ile çıkış dizisinin uzunlukları eşit olmaktadır. Giriş uzunluğu kadar,  $\mathbf{h}_t$  ve  $\mathbf{C}_t$  değerleri güncellenmektedir. Giriş dizi uzunluğunun LSTM katmanında öğrenebilir parametre sayısına hiçbir etkisi

yoktur. Ancak giriş dizi uzunluğu arttıkça eğitim süresi artmaktadır. LSTM katmanındaki tüm zaman adımında elde edilen tüm  $h_t$  değerleri tam bağlantılı katmanın giriş değerleri olmaktadır. Tam bağlantılı çıktı katmanının çıkışında, çıkış dizilerinin değerleri yer almaktadır.



Şekil 5.2: Diziden diziye problemler için LSTM tabanlı yapay sinir ağı blok diyagramı.

RNN mimarilerinde önceki bilgi kullanımına dayalı bir yaklaşım vardır [69-70]. Örneğin, RNN doğal dil işleme alanında, dil modeli için kullanılmaktadır [71]. Örneğin “Meyve toprakta yetişir” cümlesinde, yetişir kelimesinden önce “toprak” kelimesini tahmin etmek kolaydır. Fakat bağlamlar arası boşluk arttığında, türevlerin yok olması probleminden dolayı, RNN modelin geçmişten gelen bir bilgiyi kullanması, türevlerin oldukça küçülmesi yada yok olması nedeniyle oldukça zordur. Örneğin, “Türkiye’de büyüdüm. ... Akıcı bir şekilde Türkçe konuşurum.” gibi bir metinde “Türkçe” kelimesini tahmin ederken, içinde bulunduğu cümleden yola çıkarak bir dil adı olacağı tahmin edilebilir, ancak doğru kelimenin “Türkçe” olduğunu tahmin etmek için, metnin başındaki cümledeki yer bilgisini de hafızada tutmak gerekmektedir. Teoride mümkün olan “uzun-vadeli bağımlılıklar”, pratikte büyük problemlere yol açtığı görülmüştür. Bu problemi çözmek için, uzun vadeli bağımlılıkları öğrenebilen özel bir RNN türü olan LSTM ağları kullanılmaktadır. LSTM algoritması, Hochreiter ve Schmidhuber tarafından 1997 yılında tanıtılmıştır [72]. LSTM algoritmaları ile diziden diziye problemler çözülebilmektedir. Örneğin, LSTM algoritmasıyla sohbet botu uygulamasında, giriş dizileri sorulardan oluşmaktadır. O sorulara verilen cevaplar ise, çıkış dizilerini oluşturmaktadır. Kelimeler, kelime-kökleri, heceler ya da harfler doğal sayıya çevrilir. Dil çevirisi uygulamasında aynı şekilde yapılır. Kaynak metin İngilizce olur ve hedef metin Türkçe olursa, LSTM ile İngilizce-Türkçe otomatik dil çevirisi yapılmış olur. Gelen

doğal sayı dizisine karşılık düşecek doğal sayı dizisine eğitilerek geri dönüş yapılmaktadır. Şekil 5.2’de basit bir sohbet botu ya da soru cevaplama sisteminin örneğinin veri seti görülmektedir. Şekil 5.3’de Python dilinde yazılmış Keras kütüphanesi kullanılarak, LSTM algoritmasıyla 5.2’deki veri seti kullanılarak üretilen bir sohbet botu örneği görülmektedir. Metin verileri, doğal sayı dizisine çevrilerek sayısallaştırılmaktadır. Dizideki doğal sayılarda, 0’lar ve 1’den oluşacak şekilde ikili olarak temsil edilmektedir. Doğal sayının değerine sahip indis değeri 1’dir. Diğer elemanları 0 olmaktadır. Örneğin, 2 doğal sayısı [0,0,1,0,0,...,0] şeklinde ifade edilmektedir. Şekil 5.3’de görüldüğü üzere, herbir farklı kelime farklı bir doğal sayıyla ifade edilmektedir.

```
adın nedir adım bot
yaşınız kaçtır yaşı 10
en çok sevdiğin ad nedir en çok sevdiğim ad ahmet
kaç yaşındasınız 10 yaşımdayım
nasılsınız iyiyim siz nasılsınız
siz nereden geliyorsun ben istanbuldan geliyorum
film izlemeyi seviyor musun evet film izlemeyi seviyorum
kitap okumayı seviyor musun evet kitap okumayı seviyorum
sinemaya gitmeyi seviyor musun evet sinemaya gitmeyi seviyorum
en çok sevdiğin film nedir en çok sevdiğim film inception
en çok sevdiğin müzik türü nedir en çok sevdiğim müzik türü rock
en çok hoşlandığın müzik türü nedir en çok hoşlandığım müzik türü rock
müzik dinlemeyi sever misin evet müzik dinlemeyi severim
en çok sevdiğin hayvan nedir en çok sevdiğim hayvan kedidir
en çok sevdiğin ders nedir en çok sevdiğim ders matematik
kedi dokuz canlı mıdır evet kedi dokuz canlıdır
en çok sevdiğin kitap nedir en çok sevdiğim kitap harry potter
kitap okumayı sever misin evet kitap okumayı severim
en çok sevdiğin renk nedir en çok sevdiğim renk kırmızıdır
en çok sevdiğin arkadaşın adı nedir en çok sevdiğim arkadaşın adı ahmet
en çok sevdiğin kitap türü nedir en çok sevdiğim kitap türü macera
```

Şekil 5.3: Sohbet botu örneği metinleri.

```

ubuntu@ubuntu:~/seq2seq$ sudo python3 chatbot.py
Using TensorFlow backend.
2018-12-03 01:06:23.993517: I tensorflow/core/platform/cpu_device_compiler.cc:117: Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2 FMA
Loaded model from disk
question: adın nedir
[23, 33]
[0, 0, 0, 0, 4, 5]
answer: adım bot

question: yaşınız kaçtır
[1, 29]
[0, 0, 0, 0, 13, 9]
answer: yaşıım 10

question: siz nereden geliyorsun
[27, 36, 30]
[0, 0, 0, 11, 25, 28]
answer: ben istanbuldan geliyorum

question: film izlemeyi seviyor musun
[19, 20, 22, 3]
[0, 0, 37, 19, 20, 32]
answer: evet film izlemeyi seviyorum

question: kitap okumayı seviyor musun
[15, 24, 22, 3]
[0, 0, 37, 15, 24, 32]
answer: evet kitap okumayı seviyorum

question: en çok sevdiğin film nedir
[6, 2, 14, 19, 33]
[0, 6, 2, 7, 19, 39]
answer: en çok sevdiğim film inception

question: en çok sevdiğin ad nedir
[6, 2, 14, 18, 33]
[0, 6, 2, 7, 18, 34]
answer: en çok sevdiğim ad ahmet

```

Şekil 5.4: Şekil 5.3’deki sohbet botu örneğindeki veri setinin çalışması.

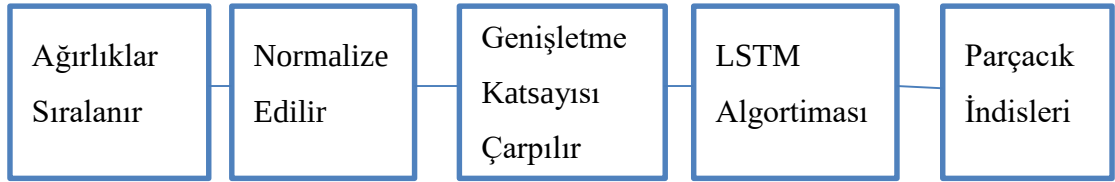
İki doğal sayının toplamı problemi, örneğin 345 ve 452 sayılarının toplanması için, 10 sayıslı toplama (+) işaretini belirtmek üzere, 7 uzunluklu [3,4,5,10,4,5,2] doğal sayı dizisi eşit uzunluklu [0,0,0,0,7,9,7] doğal sayı dizisine dönüştürülerek LSTM algoritması ile kolaylıkla çözülmektedir. Tipik olarak giriş ve çıkış dizi boyutları eşit olmaktadır. Sonuç olarak, birçok diziden diziye problemde LSTM kullanılabilir.

## 5.2. LSTM Tabanlı Yeniden Örnekleme Algoritması

### 5.2.1. LSTM Tabanlı Yeniden Örnekleme Algoritması

Bu bölümde, LSTM algoritması kullanılarak geliştirilen özgün yeniden örnekleme algoritması hakkında bilgi verilmektedir. Yeniden örnekleme algoritmasında, zayıf olan parçacıkların sistemden çıkarılması ve ağırlıkları yüksek olan parçacıkların çoğaltılması gerekir. Yeniden örnekleme algoritmaları aslında

diziden diziye bir problemdir. Tüm yeniden örnekleme algoritmaları parçacıkların ağırlıklarını giriş olarak alır. Çıkış olarak çoğaltılacak parçacıkların indisini geri döner. Ağırlık dizileri reel sayı olduğu için uygun bir çözünürlük sağlayacak 100, 1000 gibi bir genişletme katsayısı ile çarpılıp yuvarlanarak doğal sayıları dönüştürülebilir. Şekil 5.4’de görüldüğü üzere, ağırlıkları doğal sayı dizisi LSTM algoritmasının giriş dizisi olur. Yeniden örnekleme algoritmalarında elde edilen sonuçlar ise LSTM algoritmasının çıkış dizisi olur. Giriş dizileri probleminden bağımsız olması için rastgele ve tüm örnekleri iyi yansıtacak şekilde, tüm farklı doğal sayı değerlerini içerecek şekilde üretilmelidir.



Şekil 5.5: LSTM yeniden örnekleme algoritması blok şeması.

Parçacık filtresinde hedef takibinde, parçacık filtresinin yeniden örnekleme adımı bu algorithmada çok kolaylıkla kullanılabilir. Şekil 5.6’de LSTM’in eğitimi için üretilmiş örnek giriş dizileri gösterilmektedir. Bunlar doğal sayıya dönüştürülmüş ağırlık dizileridir. Şekil 5.7’de ise LSTM’in eğitimi için üretilmiş örnek çıkış dizileri gösterilmektedir.

Yeniden örnekleme yaklaşımında, giriş olarak parçacıkların ağırlıkları ve çıkış olarak çoğaltılacak indisler doğal sayı olarak mevcuttur. Parçacıkların ağırlıkları büyükten küçüğe doğru sıralanmıştır. Ağırlıkları normalize edilip, hassasiyetine göre 100, 200 ya da 1000 gibi bir genişletme katsayısı ile çarpılıp yuvarlama yapıldığında ağırlıklar doğal sayıya dönüştürülmüş olacaktır. Şekil 5.6’da verilen örnek verisetinde, eğitim aşamasında toplamı 100 olacak şekilde rastgele ağırlık dizileri oluşturulmuştur. Giriş dizileri tüm farklı ağırlık değerlerini içerecek şekilde oluşturulmalıdır.

|                                                                              |
|------------------------------------------------------------------------------|
| [57, 5, 4, 3, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0] |
| [7, 7, 6, 6, 6, 5, 5, 5, 4, 4, 4, 4, 4, 4, 4, 4, 3, 3, 3, 3, 3, 2, 0, 0]     |
| [25, 8, 8, 5, 5, 5, 5, 4, 4, 3, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1]       |
| [20, 8, 7, 6, 6, 5, 5, 4, 4, 4, 4, 3, 3, 3, 2, 2, 2, 2, 2, 2, 1, 1, 1]       |
| [77, 3, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0]       |
| [22, 8, 5, 4, 4, 4, 4, 4, 4, 4, 4, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 1, 0]       |
| [94, 3, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]       |
| [42, 7, 6, 6, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1]       |
| [41, 7, 6, 5, 5, 4, 4, 4, 3, 3, 3, 2, 2, 2, 2, 2, 1, 1, 1, 1, 0, 0, 0]       |
| [28, 8, 7, 7, 6, 6, 4, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 0]       |
| [12, 7, 7, 6, 6, 6, 5, 5, 4, 4, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 2, 2, 1]       |
| [8, 7, 6, 6, 6, 6, 5, 5, 5, 5, 5, 5, 4, 4, 4, 3, 2, 2, 2, 2, 2, 1, 0]        |
| [9, 9, 8, 5, 5, 5, 5, 5, 5, 4, 4, 4, 4, 4, 3, 3, 3, 2, 2, 2, 1, 0, 0]        |
| [10, 7, 6, 5, 5, 5, 5, 5, 4, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 1, 1, 0]          |
| [44, 7, 6, 5, 4, 4, 4, 4, 3, 3, 3, 2, 2, 2, 1, 1, 1, 1, 1, 1, 0, 0, 0]       |
| [10, 10, 9, 8, 6, 6, 5, 4, 4, 3, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 1, 0]      |
| [7, 7, 7, 7, 7, 6, 6, 5, 5, 4, 4, 4, 3, 3, 3, 3, 2, 2, 2, 1, 1, 1, 0]        |
| [23, 8, 6, 6, 5, 5, 4, 4, 4, 4, 4, 4, 3, 2, 2, 2, 2, 2, 2, 1, 1, 1, 0]       |
| [17, 9, 6, 5, 5, 5, 5, 5, 4, 4, 4, 4, 4, 3, 3, 3, 3, 3, 2, 1, 1, 1, 0]       |
| [12, 11, 8, 7, 6, 5, 5, 5, 4, 4, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 1, 1, 1]      |

Şekil 5.6: LSTM yeniden örnekleme algoritması için giriş dizileri.

Bu rastgele diziler Açgözlü, Systematic, Stratified yeniden örnekleme algoritmaları kullanılarak çıkış dizileri oluşturulmuştur. Çıkış dizisinde algoritmalarının verdiği indisler küçükten büyüğe doğru sıralanmıştır. Şekil 5.7’de eğitim için kullanılan çıkış dizileri görülmektedir.

|                                                                                   |
|-----------------------------------------------------------------------------------|
| [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 2, 3, 4, 6, 11]         |
| [0, 0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 9, 10, 11, 12, 13, 14, 15] |
| [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 8, 12]         |
| [0, 0, 0, 0, 0, 0, 1, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 8, 9, 10, 11, 12]    |
| [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]             |
| [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 2, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]   |
| [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]             |
| [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 2, 2, 3, 3, 4, 5, 6, 7, 8, 9]       |
| [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 5, 6, 7, 8]       |
| [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 7, 8, 9, 11]      |
| [0, 0, 0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 9, 10, 11, 12, 13, 15]  |
| [0, 0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8, 9, 9, 10, 11, 11, 13]   |
| [0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8, 9, 9, 11, 13]     |
| [0, 0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8, 9, 10, 11, 12, 13, 14]  |
| [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 2, 2, 3, 3, 4, 5, 6, 7]       |
| [0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 8, 9, 10, 11, 12, 13, 14]  |
| [0, 0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8, 9, 9, 10, 11, 12, 13]   |
| [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 7, 8, 9, 10, 11]     |
| [0, 0, 0, 0, 0, 1, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 9, 10, 11, 13]    |
| [0, 0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 9, 10, 11, 13]    |

Şekil 5.7: LSTM yeniden örnekleme algoritması için çıkış dizileri.

Derin öğrenme kütüphanesi olarak, Google Tensorflow kütüphanesi arka planda kullanan, Python dilinde yazılmış keras kütüphanesi kullanılmıştır. Şekil 5.6

giriş dizileri ve Şekil 5.7'deki çıkış dizileri kullanılarak, Keras kütüphanesindeki LSTM algoritması ile eğitim aşaması yapılmıştır ve model katsayıları elde edilmiştir. Elde edilen model katsayıları, hatayı minimize ederek elde edilmiş olan unut kapısı, giriş kapısı, çıkış kapısı ve hücre çıktısındaki  $W$  ve  $U$  ağırlık katsayılarıdır.

Kestirme adımında, diğer yeniden örnekleme algoritmaları gibi parçacıkların ağırlıkları giriş parametresi olacaktır. Parçacıkların ağırlıkları büyükten küçüğe doğru sıralanmaktadır. Veriler sıralandığında eğitim için daha az veriye ihtiyaç olmaktadır. Örneğin, 50 ağırlık değerine indisli parçacık 25 uzunluğunda bir dizi için 25 farklı yerde bulunmak yerine, veriler sıralandığında yalnızca 1 farklı yerde bulunmaktadır. Daha sonra genişletme katsayısı  $K$  ile çarpılarak doğal sayı giriş dizisi  $g_i$  oluşturulmaktadır.  $K$  genişletme katsayısı parçacık sayısından düşük bir değerde olmamalıdır. Eğitilen LSTM modelinde giriş olarak bu doğal sayı dizileri kullanılmaktadır. Çıkış dizisi olarak yeniden örneklenmiş parçacıkların indisleri  $c_i$  üretilmektedir. Çıkış dizisinin uzunluğu başlangıçta belirlenen parçacık sayısı kadardır. LSTM modeli eğitim esnasında herhangi bir yeniden örnekleme algoritması kullanılabilir.

Kestirme adımında, yeniden örneklenmiş parçacıkların indisleri kullanılarak parametre kestirimi yapılacaktır. LSTM tabanlı yeniden örnekleme algoritmasının adımları aşağıda verilmektedir.

#### **Algoritma: LSTM tabanlı yeniden örnekleme algoritması**

- Adım 1: Giriş:  $N$  tane parçacık için parçacıkların ağırlıkları  $w_1, w_2, \dots, w_N$
- Adım 2: Parçacık ağırlıklarını büyükten küçüğe sırala:  $w_1 \geq w_2 \geq \dots \geq w_n$
- Adım 3: Normalize ağırlıkları bul:  $W_i = w_i / \sum_{i=1}^N w_i$
- Adım 4: Doğal sayı giriş dizisi oluştur:  $g_i = \text{round}(K * W_i), i = 1, 2, \dots, N$
- Adım 5: Çıkış dizisi üret:  $c_i = \text{LSTM}(g_i), i = 1, 2, \dots, N$

Parçacık filtreleri ile hedef takibindeki yeniden örnekleme adımında, daha önce eğitilmiş LSTM modelleri kullanılmaktadır. Hedef takibinde yeniden örneklenmiş parçacık indisleri kullanılarak kestirme yapılmaktadır.

### 5.2.2. Böl Fethet Yaklaşımının LSTM Dizi Tahmininde Kullanılması

Böl-Fethet (Ing: Divide And Conquer) algoritmaları, orijinal problemi daha kolay çözmek için, problemi daha küçük alt problemlere böler. Daha sonra bölünen alt problemleri çözmektedir, orijinal problemin çözümünü oluşturmak için, alt problemlerin çözümlerini birleştirir. LSTM dizi tahmininde, Böl-Fethet yaklaşımın kullanılması, problemi kolaylaştırır. Daha uzun dizilerin eğitiminden kurtarır. Gizli birim sayısı, tur sayısı, batch sayısı, algoritma seçimi, gibi hiperparametrelerde en iyi parametrelerin bulunması işlemi de kolaylaşmış olur. Örneğin, Böl-Fethet yaklaşımı ile 25 dizi uzunluğuna sahip bir LSTM modeli ile daha yüksek parçacıkta 50, 100 uzunluklu dizi tahmini de eğitim yapmadan rahatlıkla yapılır.

N dizi uzunluğundaki LSTM modelinde  $2*N$  uzunluklu dizi tahmini gerçekleştirmek için, algoritma adımları aşağıdaki gibidir.

Adım 1: Giriş:  $2N$  tane parçacık için parçacıkların ağırlıkları:  $w_1, w_2, \dots, w_{2N}$

Adım 2: Tüm parçacıklarının ağırlıklarını 2 ile çarp:  $2w_1, 2w_2, \dots, 2w_{2N}$

Adım 3:  $N$  uzunluklu ağırlıkları toplamı birbirine yakın iki dizi oluşturulur. Tablo 5.1'deki sözde kodu verilen algoritma gibi bir algoritma kullanılabilmektedir.

Adım 4: Dizi1  $d1$  ve Dizi2  $d2$  toplamalarını bul. Toplam  $d1$ :  $\sum_{i=1}^N d1_i$  Toplam  $d2$ :  $\sum_{i=1}^N d2_i$

Adım 5: Genişletme katsayısını  $K$  dizinin toplamına bölünerek birleştirme katsayıları bulunur.  $(K/\text{Toplam } d1)$  ve  $(K/\text{Toplam } d2)$  elde edilir.

Adım 6:  $N$  uzunluklu eğitilen LSTM modeli ile iki adet sonuç elde edilir. İndis sayıları elde edilir.

Adım 7: Adım 5 elde edilen birleştirme katsayıları kullanılarak Adım-6 elde edilen indis sayıları ile çarpılır.

Basit sayısal örnek üzerinden anlatmak gerekirse, Toplam 100 ağırlığına sahip 50 uzunluğuna sahip dizi,  $[25, 25, 15, 15, 5, 5, 5, 5, 0, \dots, 0]$  olsun.

Adım 1: Giriş:  $[25, 25, 15, 15, 5, 5, 5, 5, 0, \dots, 0]$

Adım 2: Tüm parçacıkların ağırlıkları 2 ile çarpılırsa,  
 $[50, 50, 30, 30, 10, 10, 10, 10, \dots, ]$

Adım 3:  $d1=[50,30,10,10,0,\dots,0]$   $d2=[50,30,10,10,0,\dots,0]$  şekline dizi ikiye ayrılır. Dizilerin indisleri  $[0,3,5,\dots]$  ve  $[1,2,4,\dots]$  şeklinde ayrılabilir.

Adım 4: Dizi1  $d1$  toplamı 100 ve dizi2  $d2$  toplamı 100'dür.

Adım 5: İki dizinin toplamı 100 olduğu için diziler 1 ile çarpılmalıdır. Birleştirme katsayıları 1 ve 1'dir.

Adım 6: LSTM dizi  $[15 \text{ tane } 0, 10 \text{ tane } 1]$  olarak geri döndüğü varsayılın.

Adım 7: Başlangıç indisler ve birleştirme katsayıları kullanılırsa, Dizi1  $[15 \text{ tane } 0, 10 \text{ tane } 3]$  ve Dizi2  $[15 \text{ tane } 1, 10 \text{ tane } 1]$  olur. Sonuç Dizisi  $[15 \text{ tane } 0, 15 \text{ tane } 1, 10 \text{ tane } 2, 10 \text{ tane } 3]$  olarak geri dönecektir.

Bu önerilen yaklaşım, dizi ağırlıkları eşit olarak ayrılmasa bile, birleştirme katsayısı kullanılarak, çözüm üretilmektedir. Ancak, eğer dizinin iki yakın ağırlıklı ayrılması istenirse, Tablo 5.1'de sözde kodu verilen algoritma da kullanılabilir. Liste sıralı değilse sıralamak gerekir. Listenin toplamı iki değişkende tutulmaktadır. Eğer, birinci toplam ikinci toplam dan küçükse. Eleman liste1 eklenmektedir. Değilse, eleman liste2'ye eklenmektedir. Son aşamada ise iki listenin elemanları kontrol edilmektedir. Sayısı fazla olan listenin son elemanları, sayısı az olan listeye eklenmektedir.

Tablo 5.1:Yaklaşık ağırlıklı olarak dizileri ikiye bölen algoritmanın sözde kodu.

**Procedure** splitListTwoPartsApproximateWeight(sortedList)

*List1* = {}, *List2* = {}

*List1Indice* = {}, *List2Indice* = {}

*Sum1* = 0, *Sum2* = 0

**for** *indice* in size(sortedList):

*element* = sortedList[*indice*]

**if** (*Sum1* < *Sum2*):

*List1 element* ekle.

*List1Indice indice* ekle

*Sum1* = *Sum1* + *element*

**else:**

Tablo 5.1: Devam.

|                                                                                               |
|-----------------------------------------------------------------------------------------------|
| <i>List2 element ekle.</i>                                                                    |
| <i>List2Indice indice ekle.</i>                                                               |
| <i>Sum2=Sum2 +element</i>                                                                     |
| % List1 ve List2 listelerini eleman sayısını kontrol et.                                      |
| % İki dizi arasındaki eleman sayısının yarısı kadar olan ( <i>differenceCount</i> )           |
| % sondaki elemanları, eleman sayısı büyük olan diziden küçük olana taşı.                      |
| <b>if</b> (size( <i>List1</i> )<size( <i>List2</i> )):                                        |
| <i>differenceCount</i> =( size( <i>List2</i> )-size( <i>List1</i> ))/2                        |
| <i>List2</i> 'den <i>differenceCount</i> sayısındaki sondaki elemanları <i>List1</i> 'e taşı. |
| <b>else if</b> (size( <i>List1</i> )>size( <i>List2</i> ))                                    |
| <i>differenceCount</i> =( size( <i>List1</i> )-size( <i>List2</i> ))/2                        |
| <i>List2</i> 'den <i>differenceCount</i> sayısındaki sondaki elemanları <i>List1</i> 'e taşı. |
| <b>return</b> <i>List1, List1Indice, List2, List2Indice</i>                                   |

Basit bir sayısal örnek üzerinden algoritmayı açıklamak gerekirse; Ağırlık sıralanmış bir şekilde, 8,7,6,5,4,3,2,1 giriş dizimiz olsun. Bu sayısal örneğin çözümü adımları Tablo 5.2'de görülmektedir.

Tablo 5.2: Sayısal örneğin çözümü adımları.

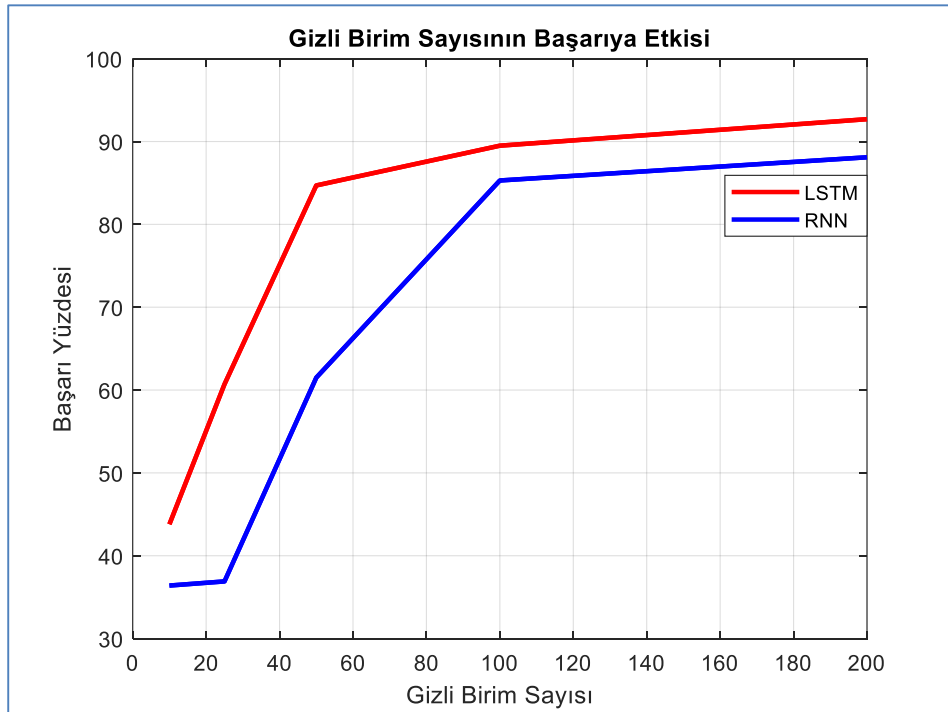
| Liste; İndis; Toplam       | Liste; İndis; Toplam       |
|----------------------------|----------------------------|
| [] ; [] ; 0                | [8] ; [0] ; 8              |
| [7] ; [1] ; 7              | [8] ; [0] ; 8              |
| [7,6] ; [1,2] ; 13         | [8] ; [0] ; 8              |
| [7,6] ; [1,2] ; 13         | [8,5] ; [0,3] ; 13         |
| [7,6] ; [1,2] ; 13         | [8,5,4] ; [0,3,4] ; 17     |
| [7,6,3] ; [1,2,5] ; 16     | [8,5,4] ; [0,3,4] ; 17     |
| [7,6,3,2] ; [1,2,5,6] ; 18 | [8,5,4] ; [0,3,4] ; 17     |
| [7,6,3,2] ; [1,2,5,6] ; 18 | [8,5,4,1] ; [0,3,4,7] ; 18 |

### 5.3. Parametrelerin Etkisi

Tüm deneysel sonuçlarda, kayıp (loss) fonksiyonu olarak ortalama mutlak hata alınmıştır. Optimizer olarak Adam optimizer algoritması kullanılmıştır. Adam optimizer algoritması adaptif bir algoritmadır. Adaptif algoritmalar öğrenme katsayısını (Ing: learning rate) kendisi öğrenmektedir ve dinamik olarak öğrenme katsayısı değişmektedir. Ölçüm metriği olarak doğruluk (Ing: accuracy) metriği kullanılmıştır. Eğitim verisinin boyutu yaklaşık 200.000'dir. Verilerin %10 doğrulama (Ing: validasyon) için kullanılmıştır.

#### 5.3.1. Gizli Birim Sayısının Etkisi

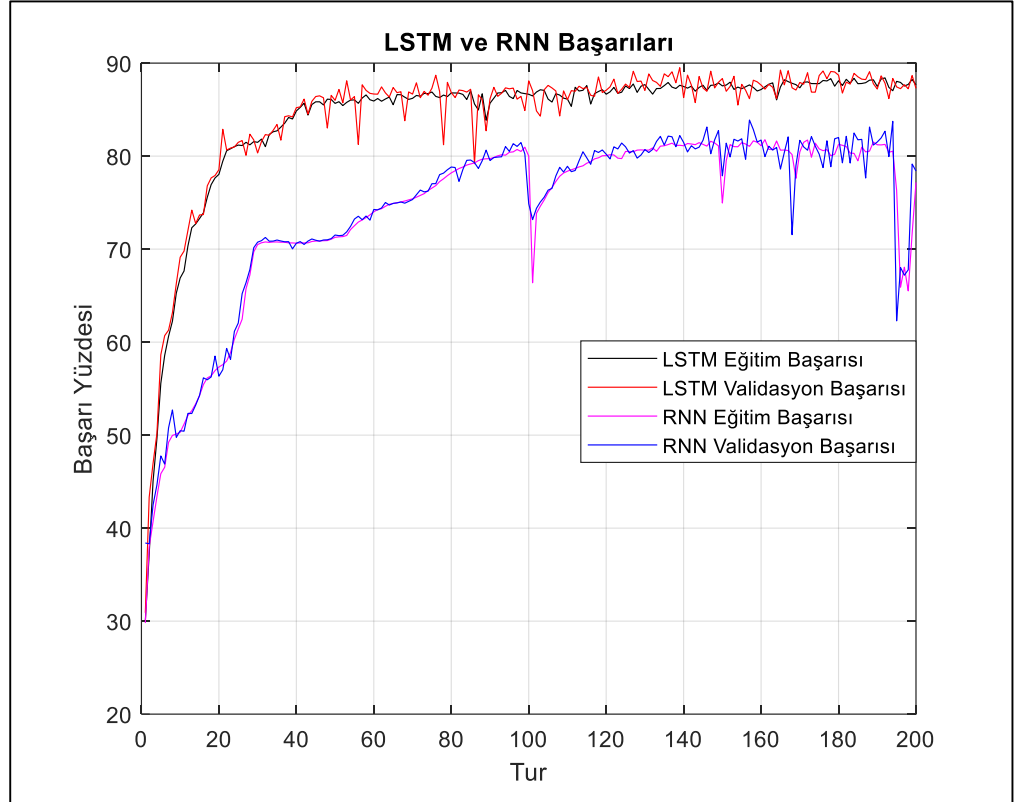
Şekil 5.8'de görüldüğü üzere, gizli birim sayısı (Ing: hidden units) arttıkça başarı belli bir noktaya kadar artmaktadır. Gizli birim sayısı, LSTM matematiksel eşitliklerindeki,  $h_t$  vektörünün uzunluğunu vermektedir. LSTM 10 ve 25 gibi değerler 400 tura sahip olmasına rağmen başarıları düşüktür. En azından 50 gizli birim sayısı seçilmez. 25 giriş dizi uzunluğu için, 100 ile 200 arası seçilebilir.



Şekil 5.8: Gizli birim sayısının etkisi.

### 5.3.2. Algoritmaların Etkisi

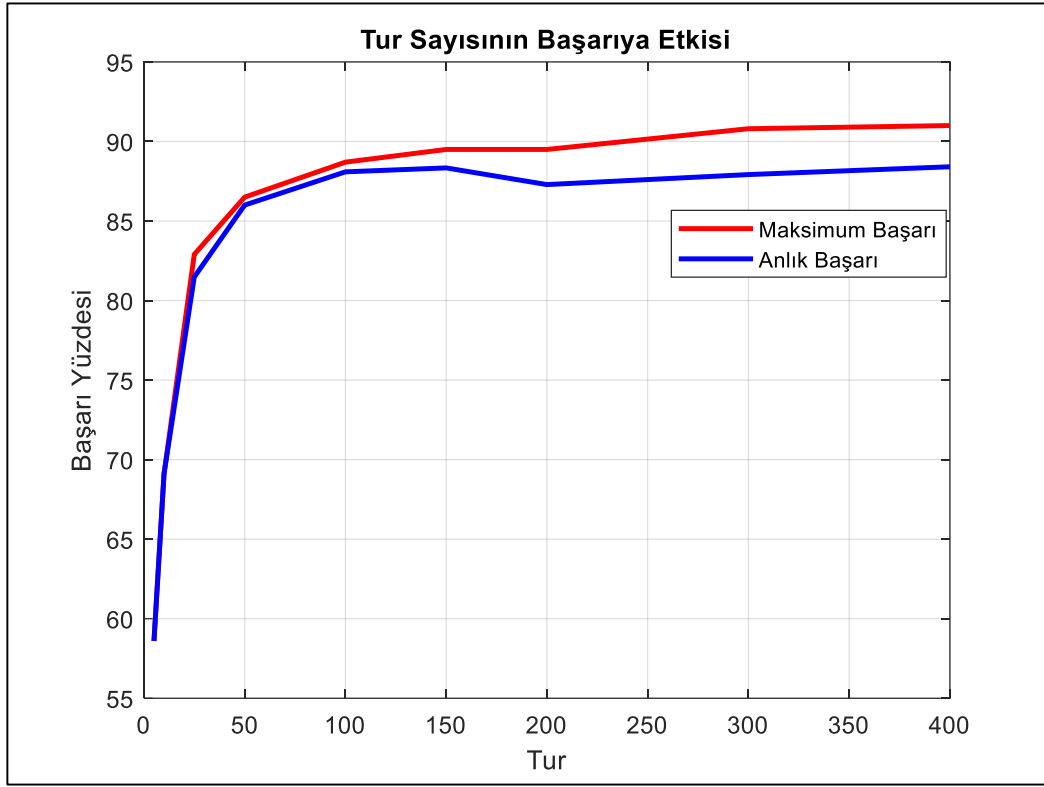
Şekil 5.9’da görüldüğü üzere, LSTM başarısı RNN başarısından daha iyidir. Daha hızlı yakınsamaktadır. Daha az oynaktır. 200 tur, 128 batch 100 gizli birim sayısında, RNN maksimum %83.9 validasyon başarısına ulaşmıştır. LSTM ise maksimum %89.5 başarıya ulaşmıştır.



Şekil 5.9: LSTM ve RNN başarıları.

### 5.3.3. Tur Sayısının Etkisi

Şekil 5.10’da görüldüğü üzere, tur sayısı artıkça belli bir noktaya kadar başarı artmaktadır. En azından 200 tur sayısı gereklidir. 50 turda yüzde 82.9 100 turda, yüzde 86.5 başarıya ulaşmıştır. 300 turda yüzde 90.8, 400 turda yüzde 91.0 ulaşmıştır. İdealde 200-400 tur seçilebilir. 400. turda 88.4 başarıdadır. 400 tur içinde en iyi validasyon başarısına sahip modeli kaydetmek ve oradaki eğitilmiş modeli kullanmak mantıklı olacaktır. Yapılan testlerde eğer validasyon başarısında iyileşme sağlanırsa kaydedilmektedir. Böylece 400. turdaki anlık başarı yerine, 400 tur boyunca en iyi validasyon başarısına sahip model kullanılmaktadır.



Şekil 5.10: Tur sayısının etkisi.

#### 5.3.4. Batch Sayısının Etkisi

Batch (blok) sayısı, modelin aynı anda kaç veriyi işleyeceğini belirten parametredir. Tablo 5.3’de görüldüğü üzere, çok yüksek batch (blok) sayısı başarıya olumsuz etkilemektedir. Batch sayısı 1024 en kötü performansa sahiptir. Gizli birim sayısı 100 olarak alınmıştır. 128 ile 256 arasındaki fark yoktur. 128 ile 256 gibi değerler seçilebilir. Batch sayısı 512’den büyük olmamalıdır.

Tablo 5.3: Batch sayısının başarıya etkisi.

| Batch Sayısı | Validasyon Başarısı (%100) |
|--------------|----------------------------|
| 128          | 89.5                       |
| 256          | 89.5                       |
| 512          | 87.3                       |
| 1024         | 84.8                       |

### 5.3.5. Katman Sayısının Etkisi

Çift yönlü, LSTM iki LSTM katmanından oluşmaktadır. 1. LSTM diziyi normal sırada çözerken, ikinci LSTM diziyi ters çevrilerek çözmektedir. Bu durum öğrenebilir parametre sayısını artırmaktadır. Aynı zamanda daha fazla veri de eğitim yapılmaktadır. Tablo 5.4’de görüldüğü üzere, katman sayısını artırmak başarıyı artırmaktadır. Ancak yakın öğrenebilir parametresine sahip daha fazla birim sayısındaki 156 gizli birim sayısında, tek LSTM daha fazla başarıya sahiptir. Gizli birim sayısını artırmak, katman sayısını artırmaktan başarıya daha fazla katkı sağlamıştır. Bu sebepten ötürü, yeniden örnekleme problemi için, gizli birim sayısı artırmak katman sayısına artırmaya göre tercih edilebilir.

Tablo 5.4: Katman sayısının başarıya etkisi.

| Algoritma                                   | Validasyon Başarısı (%100) | LSTM Katmanında Öğrenebilir Parametre Katsayısı |
|---------------------------------------------|----------------------------|-------------------------------------------------|
| 100 gizli birim 1 LSTM                      | 89.5                       | 80800                                           |
| 100 gizli birim<br>Çift Yönlü LSTM (2 LSTM) | 90.3                       | 161600                                          |
| 156 gizli birim 1 LSTM                      | 92.5                       | 160992                                          |
| 200 gizli birim 1 LSTM                      | 92.7                       | 241600                                          |

## 5.4. LSTM Tabanlı Yeniden Örneklemeye Algoritması Test Sonuçları

### 5.4.1. Lineer Olmayan Hareket Modelindeki Sonuçlar

LSTM tabanlı yeniden örneklemeye algoritmalarının performansını ölçmek için Denklem (5.9) ile Denklem (5.12)'de verilen süreç ve ölçüm fonksiyonları kullanılmıştır. Söz konusu fonksiyonlar [1] ve [25] numaralı çalışmalarda da parçacık filtresi ve yeniden örneklemeye algoritmasının performansını karşılaştırmak için kullanılmıştır. Bu model hem durum ve hem de ölçüm denklemleri nonlineer bir denklemdir. Parçacık filtresi olarak [1]'de verilen jenerik parçacık filtresi kullanılmıştır.

$$x_k = \frac{x_{k-1}}{2} + \frac{25x_{k-1}}{1 + x_{k-1}^2} + 8\cos(1.2k) + v_k \quad (5.9)$$

$$Z_k = \frac{x_k^2}{20} + r_k \quad (5.10)$$

$$v_k \sim \mathcal{N}(0, Q) \quad (5.11)$$

$$r_k \sim \mathcal{N}(0, R) \quad (5.12)$$

Burada,  $x_k$  durum denklemini,  $Z_k$  ölçümü,  $v_k$  süreç gürültüsünü ve  $r_k$  ise ölçüm gürültüsünü belirtmektedir. Bu çalışmada,  $v_k$  sıfır ortalamalı  $Q$  kovaryanslı bir Gauss gürültüsü,  $r_k$  ise sıfır ortalamalı  $R$  kovaryanslı bir Gauss gürültüsü olarak alınmıştır. Monte Carlo testleri yapılmıştır. Denklem 5.1'deki durum denklemini ve 5.2'deki ölçüm denklemini sahip bir hedefin derin öğrenmeli parçacık filtreleri ile 100 sn. izlenerek testler yapılmıştır. Açgözlü algoritmada 25 ve 50 parçacık sayısında, eşik değeri 0.8 olarak alınmıştır. Orijinal algoritmalar ile LSTM tabanlı algoritmalarının benzerlik yüzdeleri almak için, Dice benzerliği kullanılmıştır. İki kümenin ortak elemanları sayısının iki katı, kümelerin eleman sayılarının toplamı oranıdır.

$$\text{Dice Benzerliği (küme 1, küme 2)} = \frac{2|(küme 1 \cap küme 2)|}{|küme 1| + |küme 2|} \quad (5.13)$$

Örneğin, küme 1=[1,2,3,4,5] küme 2= [1,2,3,4,6] şeklide olsun.

$$\text{Dice Benzerliği (küme 1, küme 2)} = \frac{2(4)}{5+5} = 0.8$$

Tablo 5.5’de  $R=1$  ve  $Q=10$  varyans değerine, 3.16 standart sapma değerine sahiptir. Bu değerler, parçacık filtreleri algoritmalarını kıyaslamak için kullanılmış değerlerdir [1]. Parçacık sayısı  $N=25$  olarak alınmıştır. Yeniden örnekleme algoritması olarak Açgözlü algoritma kullanılmıştır. Gizli birim sayısı artıka RMSE azalmaktadır. 10 ve 25 gizli birim sayısında hata orijinal algoritmadan daha fazladır. Çünkü, 10 ve 25 gizli birim sayısında eğitim ve validasyon başarıları oldukça düşüktü. Beklenen sonuçları vermiştir.

Tablo 5.5: Gizli birim sayısının RMSE etkisi ( $R = 1$ ,  $Q = 3.16$ ,  $N=25$ ).

| Algoritma                   | RMSE        | Orijinal Algoritma ile<br>Dice Benzerliği (%100) |
|-----------------------------|-------------|--------------------------------------------------|
| Orijinal Açgözlü Algoritma  | 7.06        | 100.0                                            |
| 10 Gizli Birim Sayısı LSTM  | 9.21        | 56.32                                            |
| 25 Gizli Birim Sayısı LSTM  | 8.75        | 69.67                                            |
| 50 Gizli Birim Sayısı LSTM  | 6.66        | 73.30                                            |
| 100 Gizli Birim Sayısı LSTM | 6.39        | 71.52                                            |
| 200 Gizli Birim Sayısı LSTM | <b>6.29</b> | 71.50                                            |

Tablo 5.6’da görüldüğü üzere, 25 parçacıkta, 200 gizli birim sayısında LSTM tabanlı yeniden örnekleme algoritmasının hem orijinal algoritmalara hem de RNN tabanlı yeniden örnekleme algoritmalarına göre RMSE düşüktür. RNN tabanlı algoritmalar da orijinal algoritmalarından daha iyi performans göstermiştir.

Tablo 5.6: Orijinal algoritmalar ile LSTM tabanlı algoritmaların RMSE değeri ( $R=1$ ,  $Q=3.16$ ,  $N=25$ ).

| Algoritma                              | RMSE        | Orijinal Algoritma ile Dice Benzerliği (%100) |
|----------------------------------------|-------------|-----------------------------------------------|
| Açgözlü Yeniden Örnekleme              | 7.06        | 100                                           |
| 100 Gizli Birim Sayısı LSTM Açgözlü    | 6.39        | 71.52                                         |
| 200 Gizli Birim Sayısı LSTM Açgözlü    | <b>6.29</b> | 71.50                                         |
| 200 Gizli Birim Sayısı RNN Açgözlü     | 6.83        | 75.43                                         |
| Systematic Yeniden Örnekleme           | 6.70        | 100                                           |
| 100 Gizli Birim Sayısı LSTM Systematic | 6.74        | 71.72                                         |
| 200 Gizli Birim Sayısı LSTM Systematic | <b>6.41</b> | 69.58                                         |
| 200 Gizli Birim Sayısı RNN Systematic  | 6.46        | 74.48                                         |
| Stratified Yeniden Örnekleme           | 6.74        | 100                                           |
| 100 Gizli Birim Sayısı LSTM Stratified | 6.72        | 78.34                                         |
| 200 Gizli Birim Sayısı LSTM Stratified | <b>6.49</b> | 69.65                                         |
| 200 Gizli Birim Sayısı RNN Stratified  | 6.62        | 69.57                                         |

Tablo 5.7’de görüldüğü üzere, 50 parçacıkta, 200 gizli birim sayısında önerilen Böl Fethet LSTM tabanlı yeniden örnekleme algoritması, hem orijinal hem de alt problemlere bölmeden 50 uzunluklu dizi için eğitilen modele göre RMSE daha düşüktür.

Tablo 5.7: Orijinal algoritmalar ile LSTM tabanlı algoritmaların RMSE değeri ( $R=1$ ,  $Q=3.16$ ,  $N=50$ ).

| Algoritma                                              | RMSE        | Orijinal Algoritma ile Dice Benzerliği (%100) |
|--------------------------------------------------------|-------------|-----------------------------------------------|
| Açgözlü Yeniden Örnekleme                              | 5.95        | 100                                           |
| 200 Gizli Birim Sayısı LSTM Açgözlü                    | 6.40        | 78.80                                         |
| 200 Gizli Birim Sayısı 2*25 Böl Fethet LSTM Açgözlü    | <b>5.44</b> | 71.50                                         |
| Systematic Yeniden Örnekleme                           | 6.15        | 100                                           |
| 200 Gizli Birim Sayısı LSTM Systematic                 | 5.83        | 70.0                                          |
| 200 Gizli Birim Sayısı 2*25 Böl Fethet LSTM Systematic | <b>5.46</b> | 55.81                                         |
| Stratified Yeniden Örnekleme                           | 6.17        | 100                                           |
| 200 Gizli Birim Sayısı LSTM Stratified                 | 6.29        | 67.30                                         |
| 200 Gizli Birim Sayısı 2*25 Böl Fethet LSTM Stratified | <b>5.54</b> | 55.92                                         |

Tablo 5.8’de görüldüğü üzere, farklı gürültü ve farklı parçacık değerlerinde de, 50 parçacıkta, 200 gizli birim sayısında önerilen Böl Fethet LSTM tabanlı yeniden örnekleme algoritması, RMSE daha düşüktür ve tüm orijinal algoritmalarından daha iyi performans göstermiştir.

Tablo 5.8: Orijinal algoritmalar ile LSTM tabanlı algoritmaların RMSE değeri ( $R=1$ ,  $Q=1$ ).

| Algoritma/ Parçacık Sayısı                          | 25          | 50          |
|-----------------------------------------------------|-------------|-------------|
| Açgözlü Yeniden Örnekleme                           | 5.38        | 4.59        |
| 200 Gizli Birim Sayısı 2*25 Böl Fethet LSTM Açgözlü | <b>4.73</b> | <b>3.98</b> |
| Systematic Yeniden Örnekleme                        | 5.39        | 5.10        |

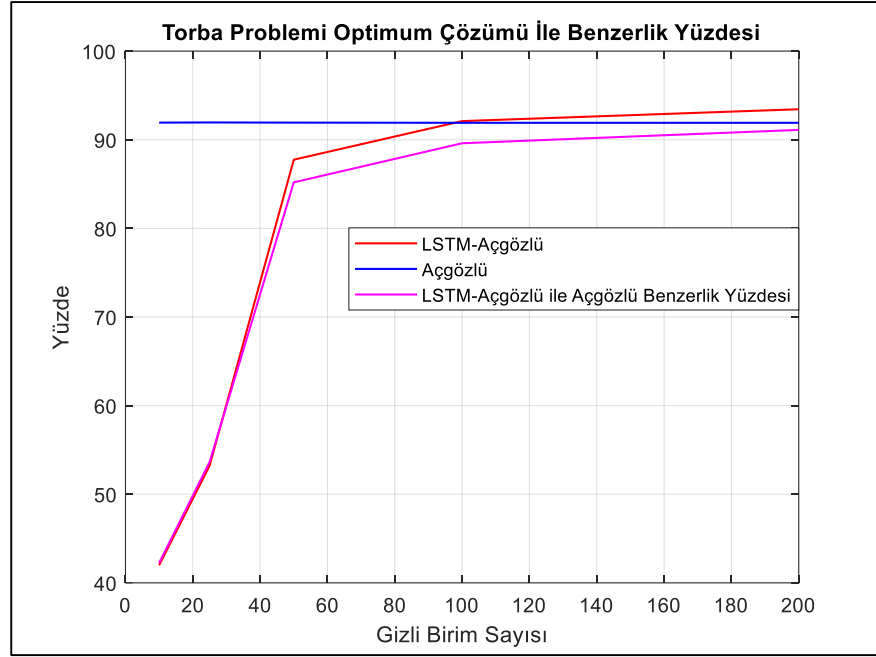
Tablo 5.8: Devam.

|                                                           |             |             |
|-----------------------------------------------------------|-------------|-------------|
| 200 Gizli Birim Sayısı 2*25 Böl Fethet<br>LSTM Systematic | <b>4.74</b> | <b>3.92</b> |
| Stratified Yeniden Örnekleme                              | 5.46        | 5.12        |
| 200 Gizli Birim Sayısı 2*25 Böl Fethet<br>LSTM Stratified | <b>4.78</b> | <b>3.92</b> |

#### 5.4.2. Torba Problemindeki Sonuçlar

Bölüm 4’de detayları verilen, yeniden örnekleme algoritmasında, eşik değeri belirlendiğinde, torba problemine dönüştürülmektedir. Örneğin, [10,10,20,20,40] eşik değeri 0.8 verildiğinde; Ağırlıklar= [10,10,20,20,40], Değerler=[100,100,400,400,1600] ve ağırlık kısıtlaması 80 olarak torba problemi dinamik programlama ile çözüldüğünde [3,4,5] olarak çözülecektir.

Test için, yaklaşık toplamı 100 olan ve 50 uzunluğunda 41000 dizi oluşturuldu. Açgözlü algoritma ve LSTM tabanlı Açgözlü algoritmanın performansları, Torba problemindeki indislerin dice benzerliği hesaplanarak karşılaştırılmıştır. Bu kapsamda, yapılan test sonuçları Şekil 5.11 ve Tablo 5.9’da görülmektedir. Gizli birim sayısı artıkça, Açgözlü orijinal algoritma ile benzerlik artmakta ve Torba probleminin optimum çözümündeki indislerine daha da yaklaşılmaktadır. 10 ve 25 gizli birim sayısında, optimum çözümlerden oldukça uzaktır. 50, 100 gizli birim sayısında, Açgözlü algoritmaya benzer performans göstermiştir. 200 birim sayısında orijinal algoritmadan daha iyi olarak Torba problemini çözmektedir.



Şekil 5.11: Algoritmaların torba problemi optimum çözümdeki indislerin dice benzerlik yüzdesi.

Tablo 5.9: Algoritmaların torba problemi optimum çözümdeki indislerin dice benzerlik yüzdesi.

| Algoritma / Benzerlik                               | Optimum Çözümdeki İndislerle Dice Benzerlik Yüzdesi | Açgözlü Algoritma İle Dice Benzerlik Yüzdesi |
|-----------------------------------------------------|-----------------------------------------------------|----------------------------------------------|
| Açgözlü Algoritma                                   | 91.93                                               | 100                                          |
| 10 Gizli Birim Sayısı 2*25 Böl Fethet LSTM Açgözlü  | 41.99                                               | 42.25                                        |
| 25 Gizli Birim Sayısı 2*25 Böl Fethet LSTM Açgözlü  | 53.23                                               | 53.64                                        |
| 50 Gizli Birim Sayısı 2*25 Böl Fethet LSTM Açgözlü  | 87.75                                               | 85.19                                        |
| 100 Gizli Birim Sayısı 2*25 Böl Fethet LSTM Açgözlü | 92.1                                                | 89.60                                        |
| 200 Gizli Birim Sayısı 2*25 Böl Fethet LSTM Açgözlü | <b>93.44</b>                                        | 91.11                                        |
| 200 Gizli Birim Sayısı 1*50 LSTM Açgözlü            | 68.25                                               | 69.03                                        |

### 5.4.3. Gürültülü Verilerdeki Sonuçlar

LSTM modelinde gürültülü giriş dizilerine sahip eğitim verisinde eğitim yapıldı. Bu testin amacı, gürültüsüz çıkış dizi değerlerini bulup bulamayacağı ilgili test yapıldı. Bu test için Açıgözlü yeniden örnekleme algoritması seçildi. Bu algoritmanın eşik değeri sabitse, gerçek değerleri sabit olması, aynı giriş değerleri için farklı çıkış dizileri üretmemesi, rastgelelik barındırmamasıdır. Eğitim verisinin %80 değiştirilmedi. %20'sinde rastgele sayıda ve 10 parçacık için, 2 rastgele indis ([2, 2, 2, 7, 7, 7, 7, 7, 7, 7]) gibi, 25 parçacık 3 rastgele indisi içerecek şekilde veriler oluş. Yaklaşık %85 civarında bir gerçek değerleri içeren bir eğitim verisi modeli eğitilmiş bulunmaktadır. Tablo 5.10'a görüldüğü üzere, validasyon ve eğitim başarıları %90'larda değildir düşmüştür. Fakat, gerçek çıkış dizisi değerlerine eğitim verisinin sahip olduğu başarıdan daha fazla başarıya ulaşmıştır. İyileşme sağlanmıştır.

Tablo 5.10: Gürültülü verilerdeki sonuçlar.

| Parçacık sayısı | Eğitim Verisinin Gerçek Değerlerle Benzerliği Yüzdesi | Eğitim Verisinin Gerçek Değerlerle Hata Yüzdesi | LSTM alg. Gerçek Değerlerle Benzerliği Yüzdesi | LSTM alg. Gerçek Değerlerle Hata Yüzdesi | LSTM alg. Validasyon Başarı Yüzdesi |
|-----------------|-------------------------------------------------------|-------------------------------------------------|------------------------------------------------|------------------------------------------|-------------------------------------|
| 10              | 85.53                                                 | 17.72                                           | <b>98.56</b>                                   | <b>2.71</b>                              | 79.8                                |
| 25              | 83.74                                                 | 18.56                                           | <b>94.79</b>                                   | <b>9.19</b>                              | 73.8                                |

Öğrenmenin temelinde, Gradyan iniş (Ing: Gradient descent) algoritmasıyla ve adım adım öğrenme katsayısı (Ing: learning rate) ağırlıklar güncellenerek, en optimum yani en az hataya sahip ağırlık katsayıları belirlenmektedir.

Yeniden örnekleme algoritmasında, [27, 14, 11, 10, 10, 8, 7, 6, 5, 2] giriş dizisi [2, 2, 2, 7, 7, 7, 7, 7, 7, 7] çıkış dizisi olarak eğitilse bile, daha önce gördüğü örneklerden, hatayı minimum yapmak için, [0, 0, 0, 0, 1, 1, 2, 3, 4, 5] tahmin ederek, gerçek değeri bulabilmektedir. Çünkü daha önce gördüğü örneklerden 27 giriş ağırlığında 4 adet civarı 0 indisli parçacıklı indisinin birçok dizide çözümde olacağını gördü. Çok nadir olarak gördüğü 2 indisli parçacığın çözümü yerine hatayı

minimum yapmak için 0 indisli parçacığa doğru eğilim yaparak ağırlık katsayılarını hesaplamıştır. Tablo 5.10'daki sonuçlar ışığında, LSTM algoritmasının, yeniden örnekleme algoritmalarına düzgünleştirme (Ing: smoothing) etkisi yaptığı da düşünülebilir.

Bu, düzgünleştirme etkisini, algoritmaları çıkış dizilerinde elde edilen değerleri detaylı analiz ederek irdeleyelim. Toplam 194529 örnek ve 25 dizi uzunluğunda  $194529 \times 25 = 4863225$  toplam indis sayısının yer aldığı tüm verisetinde, Systematic ve Stratified algoritmaları, LSTM Systematic ve LSTM Stratified algoritmaların en düşük ağırlığa sahip indisli parçacık sayıları Tablo 5.11'de görülmektedir. İçerisinde rastgeleliği içeren Systematic ve Stratified algoritmaları çok nadir de olsa düşük parçacık indisleri içermektedir. Yeniden örnekleme algoritmaların mantığı, düşük ağırlıklı parçacıklar yerine yüksek ağırlıklı parçacıkları çoğaltmaktadır. Düşük ağırlıklı parçacıkların sayısı azaltıldığında olumlu etki vermesi beklenir. LSTM tabanlı yeniden örnekleme algoritmalarının daha az sayıda en düşük parçacıkları içerdikleri görülmektedir.

Tablo 5.11: Yeniden örnekleme algoritmaların düşük parçacık içeren indis sayıları.

| Algoritma          | 25.<br>İndisinin<br>Sayısı | 25.<br>İndisinin<br>Yüzdesi | [24.,25.]<br>İndisinin<br>Sayısı | [24,25]<br>İndisinin<br>Sayısı | [23,24,25]<br>İndisinin<br>Sayısı | [23,24,25]<br>İndisinin<br>Yüzdesi |
|--------------------|----------------------------|-----------------------------|----------------------------------|--------------------------------|-----------------------------------|------------------------------------|
| Açgözlü            | 6                          | 0                           | 412                              | 0.01                           | 412                               | 0.01                               |
| LSTM<br>Açgözlü    | <b>0</b>                   | <b>0</b>                    | <b>0</b>                         | <b>0</b>                       | <b>0</b>                          | <b>0</b>                           |
| Stratified         | 17468                      | 0.36                        | 54128                            | 1.11                           | 106460                            | 2.19                               |
| LSTM<br>Stratified | <b>0</b>                   | <b>0</b>                    | <b>0</b>                         | <b>0</b>                       | <b>0</b>                          | <b>0</b>                           |
| Systematic         | 17428                      | 0.36                        | 54302                            | 1.12                           | 107042                            | 2.20                               |
| LSTM<br>Systematic | <b>0</b>                   | <b>0</b>                    | <b>2832</b>                      | <b>0.06</b>                    | <b>38297</b>                      | <b>0.79</b>                        |

Açgözlü algoritma yüksek ağırlıktan başladığı için düşük ağırlıklı parçacıklı indisler yok denecek kadar azdır. Aynı zamanda rastgelelikte mevcut değildir. Aynı giriş değerine karşı eşik değeri sabitse, aynı çıkış dizileri üretmektedir. Yaklaşık 200 bin veride, elde edilen çıkış dizileri analiz edildi. En yüksek ağırlıklı parçacık değerindeki ortalama sayısının, toplam dizi uzunluğu olan 25 ile bölümünden bir oran elde edildi. Tablo 5.12’de yeniden örnekleme algoritmaları ile LSTM tabanlı algoritmalarındaki, en yüksek parçacık oranı görülmektedir. En yüksek parçacık değeri 10, 20, ... , 90 değerlerinde sonuçlar görülmektedir. Tüm algoritmalarda en yüksek ağırlıklı parçacık sayısı oldukça benzerdir. En yüksek parçacık değeri arttıkça tüm algoritmalarda, benzer şekilde artmaktadır. Genel olarak benzerdir. Stratified algoritması ile LSTM tabanlı Stratified algoritması karşılaştırıldığında, LSTM tabanlı algoritmanın en yüksek parçacıktaki ortalamaları daha yüksektir. Bu sonuçlar ışığında en düşük parçacık yerine, en yüksek parçacığa önem verildiği göstermektedir. LSTM yeniden örnekleme algoritmasının, en yüksek parçacığın ağırlığının sayısını birikimli dağılım fonksiyonuna daha yakın bir şekilde çıktı verdiği anlaşılmaktadır. En yüksek ağırlıklı parçacık yerine diğer önemli parçacıkları çoğaltmaktadır. Parçacık ağırlığı çok düşük olmamak kaydıyla, çözümde farklı parçacık indislerinin yer alması da olumlu bir şeydir. Çünkü, tek bir parçacık yerine birden fazla parçacık, ağırlıkları oranınca kestirmeyi etkilemiş olur. Açgözlü algoritmada eşik değeri önemlidir. Eşik değeri aşan parçacıklar ağırlık değerleri ne olursa olsun doğası gereği dikkate alınmamaktadır. LSTM algoritmalarında, her bir ağırlık değeri özelliktir. Özelliklerin önemini, ağırlık katsayıları ile zamanla öğrenir. Açgözlü algoritmada, eşik değeri aşıya bile, o ağırlık önemli ise, o ağırlık değerini çoğaltacaktır. Örneğin, [80,5,5,2,2,2,1,1,...,0] şeklinde bir diziye Açgözlü algoritma [0, 0, 0,..., 0] şeklinde sonuç verirken, LSTM algoritması [0, 0, 0, 0, ..., 0,, 1,2,3,5] gibi sonuç verebilir. Eşik değerini aşılmasına rağmen, 5 ağırlığında özellikler dikkate alınmış olur. Kısaca, LSTM Açgözlü algoritması, Açgözlü algoritmanın eşik değerinden kaynaklı önemli ağırlık değerinin alınmaması problemini çözmektedir. Özetle, farklı yeniden örnekleme algoritmalarında rastgelelikler ve rastgelelikler nedeniyle düşük ağırlıkların çoğaltılması, yuvarlamalar, eşik değeri gibi parametrelerden kaynaklı birçok farklı nedenden dolayı optimal olmayan durumlar yani bir nevi gürültüler, LSTM algoritmasıyla düzeltilebilmektedir.

Tablo 5.12: En yüksek ağırlıklı parçacık sayısının oranı.

| Algoritma/<br>Yüksek<br>Ağırlık | 10   | 20   | 30   | 40    | 50    | 60    | 70    | 80    | 90    |
|---------------------------------|------|------|------|-------|-------|-------|-------|-------|-------|
| Açgözlü                         | 3.39 | 6.39 | 9.95 | 12.98 | 16.03 | 19.37 | 22.20 | 25.0  | 25.0  |
| LSTM<br>Açgözlü                 | 3.0  | 6.1  | 9.89 | 13.21 | 16.04 | 19.39 | 22.19 | 25.0  | 25.0  |
| Systematic                      | 2.50 | 5.0  | 7.49 | 10.0  | 12.50 | 15.0  | 17.47 | 20.0  | 22.47 |
| LSTM<br>Systematic              | 5.75 | 6.69 | 8.0  | 10.78 | 12.57 | 15.83 | 17.73 | 20.58 | 22.29 |
| Stratified                      | 2.50 | 5.0  | 7.5  | 10.0  | 12.48 | 15.0  | 17.53 | 20.0  | 22.58 |
| LSTM<br>Stratified              | 3.01 | 5.01 | 7.01 | 10.08 | 12.18 | 15.42 | 17.51 | 20.69 | 23.35 |

#### 5.4.4. Algoritma Çalışma Süreleri

2.2 Ghz CPU saat frekansına sahip bir PC, Keras kütüphanesi, python dili ve tahmin ederken CPU kullanılması durumunda algoritmaların çalışma süreleri Tablo 5.9’da görülmektedir. Tablo 5.13’de görüldüğü üzere, çok az proses gücüne sahip bir PC’de bile hızları gayet iyidir. 25 parçacıkta saniyede 230 ile 325 dizi tahmini yapılmaktadır. 50 uzunluğunda dizilerde Saniyede 110 ile 135 arası tahmin yapılmaktadır. Daha hızlı bir dil C++ gibi, daha hızlı CPU frekansına sahip PC yada CPU yerine ekran kartı (GPU) kullanılarak tahmin yapılması durumunda algoritmanın tahmin süresi daha da azalacaktır.

Tablo 5.13: Algoritma çalışma süreleri.

| Algoritma                           | Çalışma Süresi |
|-------------------------------------|----------------|
| N=25 ve 200 Gizli Birim Sayısı RNN  | 3.08 ms        |
| N=25 ve 100 Gizli Birim Sayısı LSTM | 3.66 ms        |
| N=25 ve 200 Gizli Birim Sayısı LSTM | 4.36 ms        |

Tablo 5.13: Devam.

|                                                        |         |
|--------------------------------------------------------|---------|
| N=50 ve 200 Gizli Birim Sayısı 2*25<br>Böl Fethet RNN  | 7.53 ms |
| N=50 ve 100 Gizli Birim Sayısı 2*25<br>Böl Fethet LSTM | 8.57 ms |
| N=50 ve 200 Gizli Birim Sayısı<br>2*25 Böl Fethet LSTM | 9.17 ms |

## 5.4. Genel Değerlendirme

LSTM, makine çevirisi, dil modelleri, soru cevap sistemleri, cümlelerin öğeleri bulma gibi birçok diziden diziye problem çözümünde kullanılmaktadır. Bu kadar popüler olmasının sebebi, LSTM algoritmasının belleğinin olmasıdır. Daha uzun geçmişini hatırlayabilmesidir. Bu bölümde, LSTM diziden diziye çözme yaklaşımının kullanılarak yeniden örnekleme algoritmaları geliştirilmektedir. Bunun için ağırlıklar sıralanmakta, normalize edilmekte, genişletme katsayısı ile çarpılarak doğal sayıları dönüştürülmektedir. Bu doğal sayı dizisi ve yeniden örnekleme algoritmasının sonuçları olan parçacık indisleri çıktı dizisi olacak şekilde, LSTM algoritmasıyla yeniden örnekleme algoritması geliştirilmektedir. Yapılan farklı parçacık, farklı gürültü değerleri ve farklı algoritmalarda LSTM tabanlı yeniden örnekleme algoritmasında üretildiği algoritmalarından daha iyi performansa sahip oldukları gözlemlenmiştir.



zorlukları içermektedir [75-76]. Yalnızca kerteriz açıları kullanılarak hedef hareket analizi probleminin çözümü için, bir çok yöntem önerilmiştir. Genişletilmiş Kalman Filtresi (EKF, Ing: Extended Kalman Filter) [75-76], Xu ve Liping, Unscented Kalman Filter'ı (UKF) bu problem çözümü için, önerilen yöntemlerdendir [77]. Ayrıca, TMA problemi, geniş bir parametre uzayında bir optimizasyon problemi olarak kabul edilmektedir ve bu problemi çözmek için evrimsel strateji tabanlı çözümler [73] ve Big Bang Big Crunch [78] gibi optimizasyon tabanlı yaklaşımlar da önerilmektedir.

Parçacık Filtreleri (PF) [79-81], TMA problemi için son zamanlarda yaygın olarak uygulanan yöntemlerdendir. TMA problemi, birçok çalışmada parçacık filtreleri kullanılarak çözülmüştür. Parçacık filtresiyle ve genişletilmiş Kalman filtresiyle yapılan bir çalışmada daha iyi performans, SIR PF'den elde edilmiştir ve Genişletilmiş Kalman filtresi ıraksamıştır [82]. Başka bir sunulan çalışmada ise, bu problem için, mesafe ( $R$ ) parametresi parçacık filtresi kullanılarak tahmin edilmiştir [83]. Yalnızca kerteriz ile izleme problemine uygulanan parçacık filtrelemenin sıralı önem yeniden örnekleme (SIR) yöntemi de önerilmiştir [84]. Yalnızca kerterizlere yönelik birden fazla ölçüm kullanan yerde hareket eden hedefler için yinelemeli bir parçacık filtresi algoritması önerilmiştir [85]. Bu problem için, Parçacık filtresi sunulan bir çalışmada ise, bilgisayar simülasyon sonuçları, parçacık filtresi algoritmasının filtrelemenin kararlılığını artırdığını ve EKF ve UKF algoritmasından daha iyi izleme doğruluğuna sahip olduğunu göstermiştir [86]. Parçacık filtresiyle yapılan başka bir çalışmada ise, parçacık filtresi algoritması EKF ve UKF'den daha iyi izleme performansı göstermiştir. [87]. Başka bir çalışmada ise, kerteriz izleme problemi için, paralel olarak çalışan parçacık filtresi algoritması önerilmiştir [88].

Kerteriz tabanlı hedef hareketi analizinde, takip edilen hedefe ait bir veya birden fazla gözlemciler üzerinde elde edilen açı ölçümleri (kerteriz açıları) kullanılarak hedefin mesafe ( $R$ ), rota ( $C$ ) ve sürat ( $S$ ) bilgileri kestirilmeye çalışılmaktadır [73]. Bu problemde hedefin sabit rota ve süratde ilerlediği; gözlemcinin en az bir manevra yaparak, ikinci bir rota bacağına geçtiği ve gözlemcinin üzerinde bulunan sensörler vasıtasıyla belli bir zaman periyodunda hedefe ilişkin kerteriz açılarının alınabildiği varsayılmaktadır. Gözlemci ilk rota bacağında sabit bir rota izleyerek kerteriz ölçümlerini almaktadır. Ancak bu kerteriz açıları sonsuz sayıda hedef çözüm parametre seti tarafından sağlanabilir [78]. Gözlemcinin manevra yaptıktan ve ikinci rota bacağında kerteriz ölçümleri

alınmasından sonra, her iki bacakta elde edilmiş olan kerteriz açısı ölçümlerinin kesişimi ile tek bir çözüm elde edilmektedir [78].

### 6.1.1. Sadece Kerterize Bağlı Hedef Takibi Durum Modeli

Hedefin sabit sürat ve sabit rotada ilerlediği varsayımı altında, tek bir gözlemcinin sadece kerteriz ölçümü  $B_k$  ile hedefi takip etme senaryosuna ilişkin durum denklemleri bu bölümde verilmektedir.  $x_k^t$  ve  $y_k^t$  hedefin  $k$  anındaki sırasıyla  $x$  ve  $y$  eksenleri yönündeki konumunu,  $x_k^o$  ve  $y_k^o$  gözlemcinin  $k$  anındaki sırasıyla  $x$  ve  $y$  eksenleri yönündeki konumunu,  $R$  başlangıç anında hedefin gözlemci olan mesafesi belirtmektedir. Hedefin başlangıç  $x$  konumu eşitlik 6.1 ile hedefin başlangıç  $y$  konumu ise eşitlik (6.2) kullanılarak hesaplanır [73].  $B_k, k$  anında ölçülen kerteriz açılarını ifade etmektedir. Eşitlik 6.3 ile 6.6'deki formüller kullanılarak hesaplanmaktadır [89].  $C$  rotayı,  $v$  süratı ifade eder.  $\Delta T$ , ardışıl iki ölçüm arasındaki zaman aralığını ifade eder.  $\dot{x}_k$ ,  $k$  anındaki  $x$  yönündeki sürati,  $\dot{y}_k$ ,  $k$  anındaki  $y$  yönündeki hızı ifade eder [73].

$$x_1^t = x_1^o + \sin(B_1) * R \quad (6.1)$$

$$y_1^t = y_1^o + \cos(B_1) * R \quad (6.2)$$

$$B_k = \arctan\left(\frac{x_k^t - x_k^o}{y_k^t - y_k^o}\right), \text{ eğer } y_k^t - y_k^o > 0 \quad (6.3)$$

$$B_k = \arctan\left(\frac{x_k^t - x_k^o}{y_k^t - y_k^o}\right) + 180, \text{ eğer } y_k^t - y_k^o < 0 \text{ ve } x_k^t - x_k^o \geq 0 \quad (6.4)$$

$$B_k = \arctan\left(\frac{x_k^t - x_k^o}{y_k^t - y_k^o}\right) - 180, \text{ eğer } y_k^t - y_k^o < 0 \text{ ve } x_k^t - x_k^o < 0 \quad (6.5)$$

$$B_k = 90, \text{ eğer } y_k^t - y_k^o = 0 \text{ ve } x_k^t - x_k^o > 0 \quad (6.6)$$

$$B_k = -90, \text{ eğer } y_k^t - y_k^o = 0 \text{ ve } x_k^t - x_k^o < 0 \quad (6.7)$$

Eşitlik (6.8) ile (6.11) arasında kerteriz tabanlı hareket modelinde  $x$  ve  $y$  yönündeki konum ve hız denklemleri yer almaktadır [73].

$$x_{k+1}^t = x_k^t + v^* (\sin (C)) * \Delta T \quad (6.8)$$

$$y_{k+1}^t = y_k^t + v^* (\cos (C)) * \Delta T \quad (6.9)$$

$$\dot{x}_k = v^* (\sin (C)) \quad (6.10)$$

$$\dot{y}_k = v^* (\cos (C)) \quad (6.11)$$

## 6.2. Açgözlü Yeniden Örnekleme Kullanan Parçacık Filtresi ile TMA

Parçacık filtresine uygun başlangıç değerleri verilmediğinde, parçacık filtresi iraksamaktadır [1]. Bu sebepten dolayı, başlangıç konum ve sürat değerlerini bulmak için bir algoritmaya ihtiyaç vardır. Bu bölümde TMA problemi için önereceğimiz parçacık filtresine uygun bir başlangıç değeri sağlamak üzere öncelikle bir arama algoritması uygulanmaktadır. Arama algoritması ile elde edilen mesafe ( $R$ ), rota ( $C$ ) ve sürat ( $S$ ) değerleri, parçacık filtresini ilklendirme için kullanılarak parçacık filtresinin gözlemlenen kerteriz açılarındaki gürültüyü azaltması sağlanmaktadır. Parçacık filtresinin sağladığı konum kestirimlerinden yararlanarak  $R$ ,  $C$  ve  $S$  parametrelerinin kestirimleri gerçekleştirilmektedir. Bu bölümde öncelikle başlangıç değerlerini sağlayacak olan arama algoritması verilecek, ardından ise parçacık filtresi ile hedef konum kestiriminin nasıl yapıldığı açıklanmaktadır.

### 6.2.1. Başlangıç Değerleri için Arama Algoritması

Parçacık filtresi başlangıç değerlerini bulmak için Tablo 6.1 sözde kodu verilen arama algoritması geliştirilmiştir. Bu yöntem, gürültüsüz bir sistemde belirli hassasiyetlerde mesafe ( $R$ ), rota ( $C$ ), ve sürat ( $S$ ) 'nin optimum çözümünü garanti eder. Hassasiyet aralıkları mesafe için 100-1000 metre, rota için 0.5-2 derece, sürat için 0.25-1 m/sn seçilebilir. Tablo 6.1'de sözde kodu verilen algorithmada,  $RInterval$ ,

*CInterval* ve *SInterval* değerleri hassasiyeti belirleyen parametrelerdir. Varsayılan değer olarak *RInterval* 400 metre mesafe, rota için *CInterval* 1 derece ve sürat için *SInterval* 0.5 m/sn seçilmiştir.

Bu algoritma bir kaba kuvvet yaklaşımıdır. Ek olarak, bu yöntemin paralel olarak çalıştırılması çok uygundur. Örneğin 4 işlemci çalıştırılması isteniyorsa 0-10000 aralığı, 10000-20000 arasında başka bir işlem aralığı, 20000-30000 arasında başka bir işlem aralığı ve 30000-40000 arası. başka bir işlem aralığı çözülerek dört iş parçacıklı bir çözüm de yapılabilir. Paralel olarak çalıştırılarak çözüme çok daha kısa sürede ulaşılabilir. Parçacık filtrelerinde elde edilen kerteriz açıları tekrar arama metodu uygulanarak, *R*, *C*, *S* değerleri elde edilebilir. Ayrıca ilk başta geniş aralıkla aranıp, buradan elde edilen çözüm etrafında daha hassas aranabilmektedir.

Elde edilen bu değerler parçacık filtre çözümünde başlangıç değerlerini atamak için kullanılabilir. *RInterval*, *CInterval* ve *SInterval* değerleri hassasiyeti belirleyen parametrelerdir. Tablo 6.1’de Genel TMA problemi için basit arama metodunun sözde kodu görülmektedir.

Tablo 6.1: Genel TMA problemi için basit arama metodu.

```
% RInterval =400 metre; CInterval=1 derece; SInterval=0.5 m/s
procedure GetRCS( $x_k^o, y_k^o, B_k, \Delta T$ , RInterval, CInterval, SInterval,
RBegin=0,REnd=40000,CBegin=0,CEnd=180, SBegin=0,SEnd=25)
    Class RMSEResults { R, C, S ,rmseValue }
    List<RMSEResults>listRmseResults
    for R=RBegin:RInterval:REnd
        for C=CBegin:CInterval:CEnd
            for S=SBegin:SInterval:SEnd
                initbear=B(1); % başlangıç kerterizi
                 $x_1^t = \sin(\text{initbear}) * R + x_1^o$ 
                 $y_1^t = \cos(\text{initbear}) * R + y_1^o$ 
                for k=2:len(Bear)
                     $x_k^t = x_{k-1}^t + (S * \sin(C) * \Delta T)$ 
                     $y_k^t = y_{k-1}^t + (S * \cos(C) * \Delta T)$ 
```

Tablo 6.1: Devam.

|                                                                            |
|----------------------------------------------------------------------------|
| $estB_k = \arctan\left(\frac{x_k^t - x_k^o}{y_k^t - y_k^o}\right)$         |
| $\text{if}(y_k^t - y_k^o < 0 \text{ ve } x_k^t - x_k^o \geq 0)$            |
| $estB_k = estB_k + 180$                                                    |
| $\text{if}(y_k^t - y_k^o < 0 \text{ ve } x_k^t - x_k^o < 0)$               |
| $estB_k = estB_k - 180$                                                    |
| <b>end for</b>                                                             |
| $rmseValue = \sqrt{\frac{1}{m} * \sum_{k=1}^m (estB_k - B_k)^2}$           |
| <code>listRmseResults.add(RMSEResults(R,C,S,rmseValue);</code>             |
| <b>end for</b>                                                             |
| <b>end for</b>                                                             |
| <b>end for</b>                                                             |
| %en küçük hataya sahip R, C, S bulunur.                                    |
| <code>RmseResults minRmseResults=findMinRmseValue(listRmseResults )</code> |
| <code>estRange=minRmseResults.R</code>                                     |
| <code>estCourse=minRmseResults.C</code>                                    |
| <code>estSpeed=minRmseResults.S</code>                                     |
| <code>minRmse=minRmseResults.rmse</code>                                   |
| <b>return</b> estRange, estCourse, estSpeed, minRmse                       |

### 6.2.2. Parçacık Filtresi ile TMA Çözümü

Bir önceki alt bölümde sözde kodu verilen arama algoritması kullanılarak TMA problemini çözmek için, önerilen algoritmanın sözde kodu Tablo 6.2’de verilmektedir. Arama algoritmasından elde edilen başlangıç değerleri kullanılarak parçacık filtresi için başlangıç değerleri olarak hesaplanmaktadır. Parçacık filtreleri sonucunda elde edilen filtrelenmiş açı değerleri kullanılarak Tablo 6.1’deki arama algoritması kullanılarak nihai R, C, S hesaplanmaktadır. İlk arama sonucu elde edilen R, C, S değerleri kullanılarak belli bir hata payını göz önünde bulundurarak, daha hassas aralıkla aranabilir. Mesafe için 5000 metre, rota için 15 derece ve hız için 5 m/s eklenebilir. Mesafe ilk aramada, 10000 metre olarak tahmin edilmişse. 5000-15000 metre arasında olması kuvvetle ihtimaldir. Rota, 100 derecede tahmin

edilmişse, 85-115, arası ve hız için 10 m/s tahmin edilmişse 5-15 m/s arası taranabilir. İlk arama sonucunda elde edilen açıdaki RMSE değerine göre bu aralıklar daraltılıp çoğaltılabilir. Parçacık filtresi yaklaşımında, ölçülen açı ile parçacık filtresindeki parçacıklardaki hesaplanan açı değerlerinin farkı alınarak, ölçülen açı değerine yakın olan parçacıkların ağırlıkları daha yüksek ağırlığa sahip olmaktadır. Daha sonra ise yeniden örnekleme adımıyla düşük ağırlıklı parçacıklar elenerek, yüksek ağırlıklı parçacıklar çoğaltılmaktadır. Kerteriz tabanlı hareketin sabit hızlı, rotası aynı ve doğrusal hareket ettiği varsayılmaktadır. Parçacık filtresi çözümü için, gözlemcinin sabit hızlı hareket ettiği ve gözlemcinin bir kez manevra yaptığı düşünülürse, iki sabit hızlı hareket olduğu düşünülebilir. Parçacık filtresi çözümünde, hedef ile gözlemcinin konum farkları alınarak çözüm geliştirilmiştir. Eşitlik (6.12) ve Eşitlik (6.13)'de bağıl hareket eşitlikleri görülmektedir. Gözlemcinin manevra yaptığı noktadan başlangıç kerterizine doğru olan hareket birinci bacaktaki doğrusal harekettir. Gözlemcinin manevra yaptığı noktadan son kerterize doğru olan hareket de ikinci bacaktaki doğrusal harekettir.

$$x_k^r = x_k^t - x_k^o \quad (6.12)$$

$$y_k^r = y_k^t - y_k^o \quad (6.13)$$

Eşitlik (6.14) kerteriz tabanlı modelde, hareket modeli ve Eşitlik (6.15)'de ise ölçüm modeli görülmektedir.

$$X_{k+1} = FX_k + q_k \quad (6.14)$$

$$Z_k = B_k + r_k \quad (6.15)$$

Eşitlik (6.14) ve (6.15)'deki denklemlerin açılımı, Eşitlik (6.16) ile Eşitlik (6.19) arasında görülmektedir.  $q_k$  hareket modeli gürültüsünü  $r_k$  ölçüm modeli gürültüsünü ifade eder. Gürültü tipik olarak Gauss gürültüsü olabilir.

$$X_k = [x_k^r \ y_k^r \ \dot{x}_k \ \dot{y}_k]^t \quad (6.16)$$

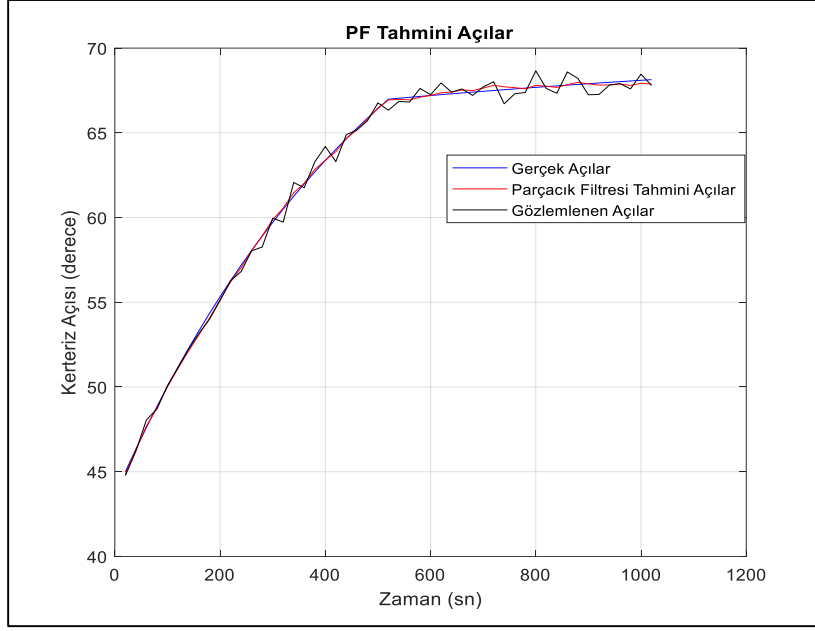
$$F = \begin{bmatrix} 1 & 0 & \Delta T & 0 \\ 0 & 1 & 0 & \Delta T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6.17)$$

$$q_k \sim \mathcal{N}(0, Q_k) \quad (6.18)$$

$$r_k \sim \mathcal{N}(0, R_k) \quad (6.19)$$

Tablo 6.2’de genel TMA çözümü algoritmasında görüldüğü üzere, her iki bacakdaki hareket ayrı ayrı parçacık filtresiyle çözülmektedir. Hedef ve gözlemci arasındaki bağıl konum ve hızlar kullanılmaktadır. İlk bacak ters olarak çözülmektedir. Bunun temel sebebi başlangıç kerterizindeki hatayı azaltmanın daha iyi R, C, S kestirimini sağlamasıdır. İlk bacakdaki hareket ters olarak çözüldüğünden, hızlar -1.0 ile çarpılmıştır. Bu sebepten dolayı, manevra yaptığı konumdan başlangıç konumuna göre açılar ters çevrilerek çözüm geliştirilmiştir. Bu yaklaşımla başlangıç kerteriz açısındaki hatada azalmaktadır.

Şekil 6.1’de görüldüğü üzere, Parçacık filtresi algoritması üzere, gözlemlenen kerteriz açılarında hata oldukça azalmaktadır. Gerçek kerteriz değerlerine yaklaşmaktadır. Tablo 6.3’de ise, parçacık filtresi algoritmasının sözde kodu görülmektedir. Parçacıkların ağırlıkları elde edilirken, Gauss dağılım fonksiyonu kullanılmıştır. Ölçülen kerteriz açılarına en yakın değerlere daha yüksek ağırlık verilmektedir.



Şekil 6.1: Parçacık filtresi tahmini açılar.

Tablo 6.2: Genel TMA problemi çözüm algoritması.

$\% x_k^o$  gözlemci x pozisyonu  
 $\% y_k^o$  gözlemci y pozisyonu  
 $\% manoeuvreIndex$  gözlemcinin birinci bacadan ikinci bacağa geçtiği nokta.  
 $\% B$  bearing açıları  
 $\% \Delta T$  = zaman aralığı  
 $\% q$  = süreç gürültüsü standart sapması  
 $\% r$  = ölçüm gürültüsü varyansı,  $\% N$  = parçacık sayısı  
 $\% resampleMethod$  =yeniden örnekleme algoritması seçimi  
 $\% threshold$  =Açgözlü yeniden örnekleme algoritması eşik değeri  
 $\% 400$  metre mesafe aralığı  
 $\% 1$  derece rota aralığı  
 $\% 0.5$  m/s hız aralığı  
 $\% estRange$  tahmin edilen mesafe  
 $\% estCourse$  tahmin edilen rota

Tablo 6.2: Devam.

```

% estSpeed tahmin edilen hız
% merge birinci dizinin sonuna ikinci diziyi ekleyen fonksiyondur.
function PFilterForTMA( $x_k^o, y_k^o, manoeuvreIndex, B, \Delta T, q, r, N,$ 
    resampleMethod, threshold)

    [estRange1, estCourse1, estSpeed1] = GetRCS( $x_k^o, y_k^o, B, \Delta T, 400, 1, 0.5$ )
    m = manoeuvreIndex
     $x_1^r = \sin(B_1) * \text{estRange1} + x_1^o$ 
     $y_1^r = \cos(B_1) * \text{estRange1} + y_1^o$ 
     $\dot{x}_1 = \text{estSpeed1} * \sin(\text{estCourse1}) - (x_2^o - x_1^o)$ 
     $\dot{y}_1 = \text{estSpeed1} * \cos(\text{estCourse1}) - (y_2^o - y_1^o)$ 
     $x_m^r = x_1^r + (\dot{x}_1 * \Delta T * (m-1))$ 
     $y_m^r = y_1^r + (\dot{y}_1 * \Delta T * (m-1))$ 
     $x_{m+1}^r = x_1^r + (\dot{x}_1 * \Delta T * (m))$ 
     $y_{m+1}^r = y_1^r + (\dot{y}_1 * \Delta T * (m))$ 

    % birinci bacaktaki başlangıç konumu ve hızları
     $X_1 = [x_{m+1}^r \ y_{m+1}^r - \dot{x}_1 - \dot{y}_1]$ 
     $\dot{x}_{m+1} = \sin(\text{estCourse}) * \text{estSpeed1} - (x_{m+2}^o - x_{m+1}^o)$ 
     $\dot{y}_{m+1} = \cos(\text{estCourse}) * \text{estSpeed1} - (y_{m+2}^o - y_{m+1}^o)$ 
    % ikinci bacaktaki başlangıç konumu ve hızları
     $X_2 = [x_m^r \ y_m^r \ \dot{x}_{m+1} \ \dot{y}_{m+1}]$ 

     $B_{leg1} = [B_m, B_{m-1}, \dots, B_1]$ 
     $B_{leg2} = [B_{m+1}, B_{m+2}, \dots, B_{end}]$ 
    th = threshold
    [B_filtered1] = ParticleFilter( $X_1, B_{leg1}, \Delta T, q, r, N, \text{resampleMethod}, th$ )
    [B_filtered2] = ParticleFilter( $X_2, B_{leg2}, \Delta T, q, r, N, \text{resampleMethod}, th$ )
    B_filtered = merge(B_filtered_1, B_filtered_2)

    return estRange1, estCourse1, estSpeed1

```

Tablo 6.3: Parçacık filtresi algoritması.

```

function ParticleFilter(x0, B, ΔT, q, r, N, resampleMethod, th)
F =[1 0 ΔT 0; 0 1 0 ΔT; 0 0 1 0; 0 0 0 1] % positionX, positionY, speedX, speedY
x_Pf={}
for i = 1 to N
    x_Pf(:,i) = x0
end for
T=size(B)
estAngle={}
for t = 1 to T
    angleMeas=B(t)
    x_Pf_update={}
    anglePf={}
    for i = 1 to N:
        x_Pf_update(:,i) = F * x_Pf(i) + q * N(0, 1) % Gauss dağılımlı süreç gürültüsü
        anglePf(i) = arctan2(x_Pf_update(1,i), x_Pf_update(2,i))


$$Pw(i) = \frac{1}{\sqrt{2 * \pi * r}} * e^{\frac{-(angleMeas - anglePf(i))^2}{2*r}}$$

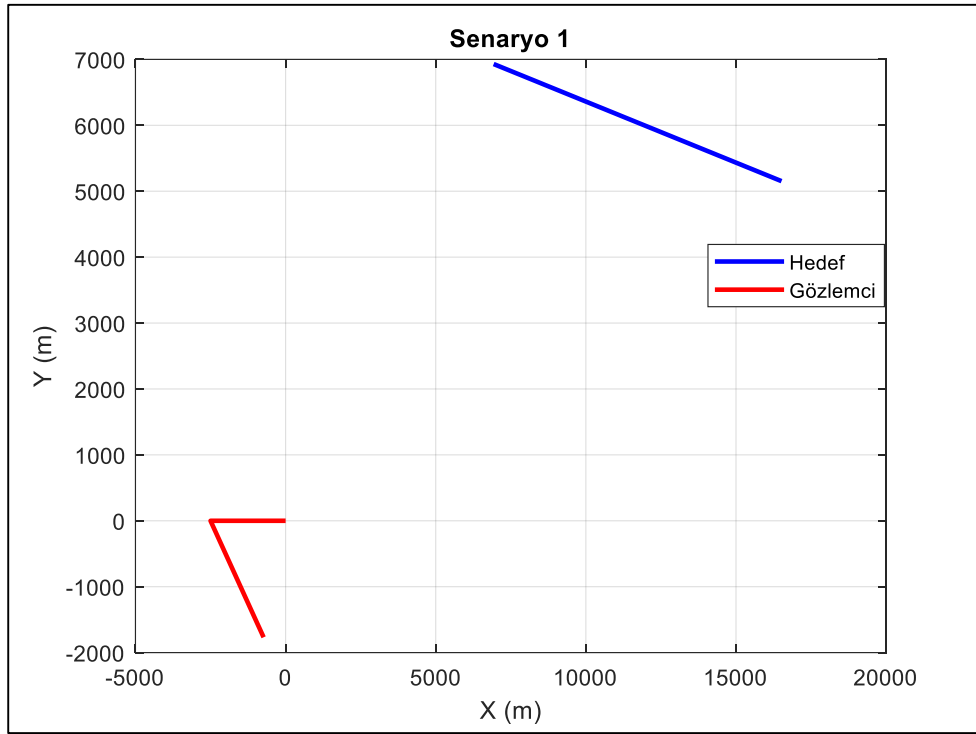
    end for
    sumPw =  $\sum_{k=1}^N Pw(i)$ 

    if(sumPw < 0.00000001):
        Pw(i) = 1/N
    else:
        for i = 1 to N
            Pw(i) = Pw(i) / sumPw
        end for
    index=resample (resampleMethod,th)
    sumAngle=0
    for i = 1 to N:
        x_Pf(:,i)= x_Pf_update(:,index(i))
        sumAngle=sumAngle+anglePf(index(i))
    end for
    estAngle(t)=sumAngle/N
end for
return estAngle

```

### 6.3. Parametre Optimizasyonu

Bu alt bölümde, maliyet fonksiyonu, parçacık filtresindeki parçacık sayısı, parçacık filtresindeki süreç ve ölçüm gürültüleri katsayıları, yeniden örnekleme algoritması ve Açgözlü yeniden örnekleme algoritmasındaki eşik değerinin mesafe, rota, sürat kestirimine etkileri araştırılmaktadır. Şekil 6.1’de gözlemci konumları ile Senaryo 1 ait hedefin x ve y konumları görülmektedir. Bu parametrelerdeki ideal değerler aranmaktadır. Parametre optimazasyonu için kullanılan senaryolar Tablo 6.4’de görülmektedir. Kerteriz açılarının gürültüsü Gauss dağılımlı ve gürültülerin standart sapması 0.4 derecedir.



Şekil 6.1: Senaryo 1 ait hedef ve gözlemcinin x ve y konumları.

Tablo 6.4: Senaryo 1-2.

| Senaryo   | Mesafe<br>(metre) | Rota<br>(derece) | Sürat<br>(m/sn) | Süre<br>(sn) | Zaman<br>Aralığı (sn) |
|-----------|-------------------|------------------|-----------------|--------------|-----------------------|
| Senaryo 1 | 9800              | 100.5            | 9.75            | 1000         | 20                    |
| Senaryo 2 | 5000              | 100.5            | 9.75            | 1000         | 20                    |

### 6.3.1. Maliyet Fonksiyonlarının Etkisi

Gözlemlenen kerterizler ile  $R$ ,  $C$ ,  $S$  arama algoritmasıyla elde edilen kerterizleri karşılatırarak en az hataya sahip değerler bulunmaktadır. Burada en az hataya sahip değerler maliyet fonksiyonları kullanılarak hesaplanmaktadır. Maliyet fonksiyonları Ortalama Karesel Hatanın Karekökü (RMSE, Ing: Root Mean Squared Error), Ortalama Mutlak Hata (MAE, Ing: Mean Absolute Error), Ortalama Karesel Log Hatasının Karekökü (RMSElog, Ing: Root Mean Squared Log Error), Ortalama Yüzdesel Mutlak Hata (MAPE, Ing: Mean Absolute Percantage Error) şeklindedir. Maliyet fonksiyonları eşitlikleri Eşitlik (6.18) ile (6.21) arasında görülmektedir.

$$RMSE = \sqrt{\frac{1}{N} \sum_{k=1}^N (x_k^g - x_k^t)^2} \quad (6.18)$$

$$MAE = \frac{1}{N} |x_k^g - x_k^t| \quad (6.19)$$

$$RMSElog = \sqrt{\frac{1}{N} \sum_{k=1}^N (\log x_k^g - \log x_k^t)^2} \quad (6.20)$$

$$MAPE = \frac{1}{N} \frac{|x_k^g - x_k^t|}{x_k^g} \quad (6.21)$$

Tablo 6.5 ve 6.6’da maliyet fonksiyonlarının etkisi görülmektedir. Arama algoritmasında, en iyi mesafe, rota ve sürat kestirimini RMSE maliyet fonksiyonu vermiştir.

Tablo 6.5: Senaryo 1 Maliyet fonksiyonlarının etkisi.

| Alg/Sonuçlar | Ortalama<br>Mesafe<br>Rota<br>Sürat | StandardSapma<br>Mesafe<br>Rota<br>Sürat | Medyan<br>Mesafe<br>Rota<br>Sürat |
|--------------|-------------------------------------|------------------------------------------|-----------------------------------|
| RMSE         | 9827<br>100.95<br>9.90              | 316<br>5.05<br>1.56                      | 9800<br>100.5<br>9.75             |
| RMSELog      | 9818<br>101.17<br>9.88              | 315<br>5.51<br>1.68                      | 9800<br>101<br>9.63               |
| MAE          | 9792<br>101.05<br>9.87              | 364<br>5.48<br>1.67                      | 9800<br>100<br>9.75               |
| MAPE         | 9792<br>101.26<br>9.85              | 366<br>5.75<br>1.74                      | 9800<br>100.5<br>9.75             |

Tablo 6.6: Senaryo 2 Maliyet fonksiyonlarının etkisi.

| Alg/Sonuçlar | Ortalama<br>Mesafe<br>Rota<br>Sürat | StandardSapma<br>Mesafe<br>Rota<br>Sürat | Medyan<br>Mesafe<br>Rota<br>Sürat |
|--------------|-------------------------------------|------------------------------------------|-----------------------------------|
| RMSE         | 5013<br>100.56<br>9.83              | 145<br>2.08<br>0.90                      | 5000<br>100.5<br>9.75             |
| RMSELog      | 5009<br>100.64<br>9.81              | 144<br>2.40<br>1.03                      | 5000<br>100.5<br>9.75             |
| MAE          | 5027<br>100.4<br>9.91               | 166<br>2.09<br>0.96                      | 5000<br>100<br>10                 |
| MAPE         | 5028<br>100.45<br>9.89              | 168<br>2.18<br>1.00                      | 5000<br>100<br>10                 |

### 6.3.2. Parçacık Sayısının Etkisi

0.4 standard sapmalı gürültülü kerterizlerdeki, Senaryo 1 için, parçacık sayısının etkisi araştırılmıştır. Bu testte, süreç gürültüsünün 1 derece standart sapmalı gürültü için standart sapma değeri dizisi  $q=[50 \ 0 \ 0 \ 0; 0 \ 50 \ 0 \ 0; 0 \ 0 \ 0.5 \ 0; 0 \ 0 \ 0 \ 0.5]$  şeklinde alınmıştır. Yani, 1 derece standart sapmalı gürültü için gürültünün standart sapma değerleri, 50 metre konum 0.5 m/s sürat alınmıştır. Minimum örnekleme

varyansı yeniden örnekleme algoritması ve ölçüm gürültüsünün standart sapma değeri ise arama sonucu elde edilen en iyi  $R$ ,  $C$ ,  $S$  ait RMSE değeri kullanılmıştır. 200 Monte Carlo simülasyonu yapılmış ve parçacık filtresi 20 kez çalıştırıp elde edilen açıların ortalaması alınmıştır. Parçacık filtresi sonucu elde edilen açılar yeniden aranarak Tablo 6.7'deki sonuçlar elde edilmiştir.  $R$ ,  $C$ ,  $S$  kestirimi açısından ideal parçacık sayısı 50 ile 200 arası olabilir.

Tablo 6.7: Parçacık sayısının etkisi.

| Alg/Sonuçlar | Ortalama<br>Mesafe<br>Rota<br>Sürat | StandardSapma<br>Mesafe<br>Rota<br>Sürat | Medyan<br>Mesafe<br>Rota<br>Sürat | Medyan<br>Mesafe<br>Rota<br>Sürat |
|--------------|-------------------------------------|------------------------------------------|-----------------------------------|-----------------------------------|
| 10           | 9826<br>100.79<br>9.87              | 351<br>4.24<br>1.38                      | 9800<br>100.5<br>9.75             | 45.00<br>0.26<br>0.16             |
| 20           | 9824<br>100.80<br>9.87              | 346<br>4.24<br>1.38                      | 9800<br>101<br>9.50               | 45.00<br>0.26<br>0.17             |
| 50           | 9832<br>100.73<br>9.89              | 341<br>4.12<br>1.37                      | 9800<br>101<br>9.63               | 45.00<br>0.26<br>0.17             |
| 100          | 9826<br>100.83<br>9.86              | 336<br>4.14<br>1.36                      | 9800<br>101<br>9.50               | 45.00<br>0.26<br>0.18             |
| 200          | 9825<br>100.76<br>9.87              | 340<br>4.07<br>1.35                      | 9800<br>101<br>9.50               | 45.00<br>0.26<br>0.18             |

### 6.3.3. Süreç Gürültüsü Standart Sapma Değerinin Etkisi

Senaryo 1 ve 0.4 standard sapmalı senaryo için, süreç gürültüsünün etkisi araştırılmıştır. Bu testte, minimum örnekleme varyansı yeniden örnekleme algoritması ve ölçüm gürültüsünün standart sapma değeri ise arama sonucu elde edilen en iyi  $R$ ,  $C$ ,  $S$  ait RMSE değeri kullanılmıştır. 200 Monte Carlo simülasyonu yapılmış ve parçacık filtresi 20 kez çalıştırıp elde edilen açıların ortalaması alınmıştır. Parçacık sayısı 200 olarak alınmıştır. Parçacık filtresi sonucu elde edilen açılar yeniden aranarak, Tablo 6.8'deki sonuçlar elde edilmiştir. 1 derecelik gözlemlenen kerteriz gürültüleri için konumun standart sapması 50 metre ve süratın standart sapması 0.5 m/s olarak belirlenebilir.

Tablo 6.8: Süreç gürültüsü standart sapma değerinin etkisi.

| Alg/Sonuçlar                        | Ortalama<br>Mesafe<br>Rota<br>Sürat | StandardSapma<br>Mesafe<br>Rota<br>Sürat | Medyan<br>Mesafe<br>Rota<br>Sürat | Medyan<br>Mesafe<br>Rota<br>Sürat |
|-------------------------------------|-------------------------------------|------------------------------------------|-----------------------------------|-----------------------------------|
| 10 m std konum<br>0.1 m/s std sürat | 9816<br>100.88<br>9.85              | 334<br>4.22<br>1.38                      | 9800<br>101<br>9.50               | 44.99<br>0.27<br>0.12             |
| 20 m std konum<br>0.2 m/s std sürat | 9834<br>100.6<br>9.94               | 343<br>4.27<br>1.39                      | 9800<br>100.5<br>9.75             | 45.00<br>0.26<br>0.13             |
| 50 m std konum<br>0.5 m/s std sürat | 9825<br>100.76<br>9.87              | 340<br>4.07<br>1.35                      | 9800<br>101<br>9.50               | 45.00<br>0.26<br>0.18             |
| 100 m std konum<br>1 m/s std sürat  | 9821<br>100.89<br>9.83              | 332<br>4.12<br>1.35                      | 9800<br>101<br>9.75               | 45.00<br>0.28<br>0.22             |
| 200 m std konum<br>2 m/s std sürat  | 9836<br>100.92<br>9.84              | 318<br>4.26<br>1.37                      | 9800<br>100.5<br>9.75             | 45.00<br>0.30<br>0.27             |

#### 6.3.4. Ölçüm Gürültüsü Standart Sapma Değerinin Etkisi

Senaryo 1 ve 0.4 standard sapmalı senaryo için, ölçüm gürültüsü standart sapma değerinin etkisi araştırılmıştır. Bu testte, 0.4 derece standart sapmalı gürültülü kerteriz açıları için süreç gürültüsünde;  $x$  ve  $y$  konum standart sapma değeri 20 metre ve  $x$  ve  $y$  süratın standart sapma değeri 0.2 m/s alınmıştır. Minimum örnekleme varyansı yeniden örnekleme algoritması kullanılmıştır. Bu elde edilen değer 200 Monte Carlo simülasyonu yapılmış ve parçacık filtresi 20 kez çalıştırıp elde edilen açıların ortalaması alınmıştır. Parçacık sayısı 200 olarak alınmıştır. Parçacık filtresi sonucu elde edilen açıları yeniden aranarak, Tablo 6.9'daki sonuçlar elde edilmiştir. 0.4 standard sapmalı açıları için, 0.1 ve 1 derecenin seçilmesi en kötü performansı vermiştir. En iyi performansı 0.4 ve arama sonucu elde edilen en iyi  $R$ ,  $C$ ,  $S$  ait RMSE değerinde elde edilmiştir. Sonuç olarak ölçüm gürültüsü standart sapma değeri olarak, arama sonucu elde edilen RMSE değerinin kullanılabilir.

Tablo 6.9: Ölçüm gürültüsü standart sapma değerinin etkisi.

| Alg/Sonuçlar               | Ortalama<br>Mesafe<br>Rota<br>Sürat | StandardSapma<br>Mesafe<br>Rota<br>Sürat | Medyan<br>Mesafe<br>Rota<br>Sürat | Medyan<br>Mesafe<br>Rota<br>Sürat |
|----------------------------|-------------------------------------|------------------------------------------|-----------------------------------|-----------------------------------|
| 0.1                        | 9814<br>100.70<br>9.90              | 371<br>4.47<br>1.44                      | 9800<br>100.5<br>9.75             | 45.00<br>0.30<br>0.22             |
| 0.2                        | 9822<br>100.70<br>9.89              | 331<br>4.16<br>1.36                      | 9800<br>100.75<br>9.75            | 45.00<br>0.27<br>0.20             |
| 0.4                        | 9821<br>100.82<br>9.85              | 339<br>4.08<br>1.35                      | 9800<br>101<br>9.63               | 45.00<br>0.26<br>0.18             |
| 0.4 civarı<br>arama sonucu | 9825<br>100.76<br>9.87              | 340<br>4.07<br>1.35                      | 9800<br>101<br>9.50               | 45.00<br>0.26<br>0.18             |
| 1                          | 9831<br>101.01<br>9.84              | 371<br>4.55<br>1.44                      | 9800<br>101<br>9.75               | 45.00<br>0.27<br>0.15             |

### 6.3.5. Ağgözlü Yeniden Örnekleme Algoritmasında Eşik Değerinin Etkisi

Senaryo 1 ve 0.4 standard sapmalı senaryo için, Ağgözlü algoritmadaki eşik değerinin etkisi araştırılmıştır. Bu testte, süreç gürültüsünün 1 derece standard sapmalı gürültülü kerterizler için standart sapma değeri  $q=[50 \ 0 \ 0 \ 0; 0 \ 50 \ 0 \ 0; 0 \ 0 \ 0.5 \ 0; 0 \ 0 \ 0 \ 0.5]$  alınmıştır. Yani, 1 derece standard sapmalı gürültü için gürültünün standart sapma değerleri, 50 metre konum 0.5 m/s sürat alınmıştır. Ölçüm gürültüsünün standart sapma değeri ise arama sonucu elde edilen en iyi  $R$ ,  $C$ ,  $S$  ait RMSE değeri kullanılmıştır. 200 Monte Carlo simülasyonu yapılmış ve parçacık filtresi 20 kez çalıştırıp elde edilen açıların ortalaması alınmıştır. Parçacık sayısı 200 olarak alınmıştır. Parçacık filtresi sonucu elde edilen açılar yeniden aranarak, Tablo 6.10'daki sonuçlar elde edilmiştir.  $R$ ,  $C$ ,  $S$  kestirimi açısından en çok hatayı 0.25 eşik değeri vermiştir. Eşik değerini aşan hiçbir parçacık eklenmemiştir. 0.25 eşik değerinin tahmin değerlerinin standart sapma değerleri daha yüksektir. En iyi sonucu 0.7 ile 0.9 arası eşik değerleri vermiştir. Tipik olarak, Ağgözlü yeniden örnekleme algoritmasının eşik değeri 0.8 olarak belirlenebilir.

Tablo 6.10: Açgözlü yeniden örnekleme algoritmasında eşik değerinin etkisi.

| Alg/Sonuçlar | Ortalama<br>Mesafe<br>Rota<br>Sürat | StandardSapma<br>Mesafe<br>Rota<br>Sürat | Medyan<br>Mesafe<br>Rota<br>Sürat | Medyan<br>Mesafe<br>Rota<br>Sürat |
|--------------|-------------------------------------|------------------------------------------|-----------------------------------|-----------------------------------|
| 0.25         | 9803<br>100.72<br>9.89              | 342<br>4.37<br>1.40                      | 9800<br>100.5<br>9.75             | 44.98<br>0.28<br>0.19             |
| 0.50         | 9813<br>100.62<br>9.91              | 321<br>4.21<br>1.34                      | 9800<br>100.5<br>9.75             | 44.98<br>0.27<br>0.18             |
| 0.70         | 9818<br>100.53<br>9.93              | 333<br>4.11<br>1.33                      | 9800<br>100<br>9.75               | 44.98<br>0.26<br>0.17             |
| 0.80         | 9812<br>100.55<br>9.91              | 317<br>3.99<br>1.29                      | 9800<br>100.5<br>9.75             | 44.98<br>0.26<br>0.17             |
| 0.90         | 9806<br>100.60<br>9.89              | 313<br>4.03<br>1.31                      | 9800<br>100.5<br>9.75             | 44.99<br>0.26<br>0.16             |
| 0.95         | 9803<br>100.76<br>9.85              | 320<br>4.07<br>1.31                      | 9800<br>100.5<br>9.75             | 44.99<br>0.25<br>0.16             |

### 6.3.6. Yeniden Örnekleme Algoritmasının Etkisi

Senaryo 1 ve 0.4 standard sapmalı senaryo için, yeniden örnekleme algoritmalarının etkisi araştırılmıştır. Bu testte, süreç gürültüsünün 1 derece standart sapmalı gürültü için standart sapma değeri  $q=[50 \ 0 \ 0 \ 0; 0 \ 50 \ 0 \ 0; 0 \ 0 \ 0.5 \ 0; 0 \ 0 \ 0 \ 0.5]$  alınmıştır. Yani, 1 derece standart sapmalı kerteriz gürültüleri için gürültünün standart sapma değerleri, 50 metre konum 0.5 m/s sürat alınmıştır. Ölçüm gürültüsünün standart sapma değeri ise arama sonucu elde edilen en iyi  $R$ ,  $C$ ,  $S$  ait RMSE değeri kullanılmıştır. 200 Monte Carlo simülasyonu yapılmış ve parçacık filtresi 20 kez çalıştırıp elde edilen açıların ortalaması alınmıştır. Parçacık sayısı 200 olarak alınmıştır. Parçacık filtresi sonucu elde edilen açılar yeniden aranarak Tablo 6.11'deki sonuçlar elde edilmiştir. Açgözlü yeniden örnekleme algoritmasının eşik değeri 0.8 olarak alınmıştır. 0.8 eşik değerini aşan parçacıklar eklenmemiştir. Genel olarak tüm algoritmalar yakın sonuçlar vermiştir. Açgözlü algoritma en iyi  $R$ ,  $C$  kestirimi sahip olan algoritma olduğu, en iyi  $S$  kestirimide Residual ve Systematic

algoritması vermiştir. Tipik olarak Ağgözlü yeniden örnekleme algoritması kullanılabilir.

Tablo 6.11: Yeniden örnekleme algoritmalarının etkisi.

| Alg/Sonuçlar | Ortalama<br>Mesafe<br>Rota<br>Sürat | StandardSapma<br>Mesafe<br>Rota<br>Sürat | Medyan<br>Mesafe<br>Rota<br>Sürat | Medyan<br>Mesafe<br>Rota<br>Sürat |
|--------------|-------------------------------------|------------------------------------------|-----------------------------------|-----------------------------------|
| Systematic   | 9809<br>100.78<br>9.85              | 322<br>4.00<br>1.31                      | 9800<br>100.5<br>9.75             | 44.99<br>0.25<br>0.17             |
| Stratified   | 9812<br>100.74<br>9.86              | 326<br>4.02<br>1.32                      | 9800<br>100.5<br>9.75             | 45.00<br>0.25<br>0.16             |
| MultiNominal | 9812<br>100.78<br>9.85              | 333<br>4.02<br>1.33                      | 9800<br>101<br>9.75               | 44.99<br>0.25<br>0.17             |
| Residual     | 9806<br>100.79<br>9.84              | 328<br>4.03<br>1.31                      | 9800<br>100.5<br>9.75             | 44.99<br>0.25<br>0.17             |
| MSV          | 9825<br>100.76<br>9.87              | 340<br>4.07<br>1.35                      | 9800<br>101<br>9.50               | 45.00<br>0.26<br>0.18             |
| Ağgözlü      | 9812<br>100.55<br>9.91              | 317<br>3.99<br>1.29                      | 9800<br>100.5<br>9.75             | 44.98<br>0.26<br>0.17             |

## 6.4. Test Sonuçları

### 6.4.1. Hedef Hareket Analizi Çözümü Algoritması Sonuçları

Arama algoritması ile Parçacık filtresi ile filtrelenmiş açılar sonucundaki arama algoritmasının karşılaştırmak için kullanılan senaryolar Tablo 6.12’de görülmektedir.

Tablo 6.12: Senaryo 1-9.

| Senaryo<br>No | Mesafe<br>(metre) | Rota<br>(derece) | Sürat<br>(m/sn) | Başlangıç<br>Kerteriz (der) | Süre<br>(sn) | Zaman<br>Aralığı (sn) |
|---------------|-------------------|------------------|-----------------|-----------------------------|--------------|-----------------------|
| Senaryo 1     | 9800              | 100.5            | 9.75            | 45                          | 1000         | 20                    |
| Senaryo 2     | 5000              | 100.5            | 9.75            | 45                          | 1000         | 20                    |

Tablo 6.12: Devam.

|           |       |       |       |    |      |    |
|-----------|-------|-------|-------|----|------|----|
| Senaryo 3 | 7600  | 121.5 | 12.25 | 45 | 1000 | 20 |
| Senaryo 4 | 9800  | 60.5  | 9.75  | 45 | 1000 | 20 |
| Senaryo 5 | 5000  | 60.5  | 9.75  | 45 | 1000 | 20 |
| Senaryo 6 | 7600  | 60.5  | 9.75  | 45 | 1000 | 20 |
| Senaryo 7 | 12600 | 121.5 | 12.25 | 45 | 1000 | 20 |
| Senaryo 8 | 7600  | 100.5 | 9.75  | 45 | 1000 | 20 |
| Senaryo 9 | 7600  | 100.5 | 7.25  | 45 | 1000 | 20 |

Test edilen tüm senaryolarda, parametreler sabit tutulmuştur. Bu testte, süreç gürültüsünün 1 derece standart sapmalı gürültü için standart sapma değeri  $q=[50 \ 0 \ 0 \ 0; 0 \ 50 \ 0 \ 0; 0 \ 0 \ 0.5 \ 0; 0 \ 0 \ 0 \ 0.5]$  alınmıştır. Yani, 1 derece standart sapmalı kerteriz gürültüleri için süreç gürültünün standart sapma değerleri, 50 metre konum 0.5 m/s sürat alınmıştır. Ölçüm gürültüsünün standart sapma değeri ise arama sonucu elde edilen en iyi  $R$ ,  $C$ ,  $S$  ait RMSE değeri kullanılmıştır. Gözlemcinin konumları tüm deneysel sonuçlarda sabit tutulmuştur. Gözlemcinin  $x$  ve  $y$  başlangıç noktaları 0 metre olarak alınmıştır. Gözlemcinin hızı 5 m/s'dir. Gözlemci ilk bacakta 270 derece bir rotada ilerlemiş, ikinci bacakta 135 derece bir rotada ilerlemiştir. 200 Monte Carlo simülasyonu yapılmış ve parçacık filtresi 20 kez çalıştırıp elde edilen açıların ortalaması alınmıştır. Parçacık sayısı 200 olarak alınmıştır. Parçacık filtresi sonucu elde edilen açılar yeniden aranarak aşağıdaki sonuçlar elde edilmiştir. Yeniden örnekleme algoritması olarak, Açgözlü yeniden örnekleme algoritması kullanılmıştır. Açgözlü yeniden örnekleme algoritmasının eşik değeri 0.8 olarak alınmıştır. Tablo 6.13 ile Tablo 6.20 arasında 200 Monte Carlo simülasyonun tahmindeki sonuçların ortalama değeri, standart sapma değeri, medyan değeri, başlangıç kerteriz açısı, başlangıç kerteriz açısının standart sapma değeri, filtrelemiş açının RMSE değerleri görülmektedir. Tüm senaryolarda, Parçacık filtresindeki açı RMSE değeri daha düşük, başlangıç kerterizlerindeki RMSE daha düşük ve  $R$ ,  $C$ ,  $S$  standart sapma değerleri de daha düşüktür.

Tablo 6.13: Senaryo 1 sonuçları.

| Alg/Sonuçlar        | Ortalama<br>Mesafe<br>Rota<br>Sürat | StandartSapma<br>Mesafe<br>Rota<br>Sürat | Medyan<br>Mesafe<br>Rota<br>Sürat | B.kerterizOrt<br>B.kerterizStd<br>F.AçırMse |
|---------------------|-------------------------------------|------------------------------------------|-----------------------------------|---------------------------------------------|
| Arama<br>0.1 std    | 9802<br>100.59<br>9.77              | 110<br>1.77<br>0.54                      | 9800<br>100.5<br>9.75             | 45.00<br>0.10<br>0.10                       |
| PF+Arama<br>0.1 std | 9800<br>100.68<br>9.73              | 125<br>1.51<br>0.49                      | 9800<br>100.5<br>9.75             | 45.00<br>0.07<br>0.04                       |
| Arama<br>0.2 std    | 9826<br>100.52<br>9.85              | 194<br>2.78<br>0.89                      | 9800<br>100.5<br>9.75             | 45.00<br>0.19<br>0.20                       |
| PF+Arama<br>0.2 std | 9820<br>100.45<br>9.85              | 210<br>2.43<br>0.82                      | 9800<br>100.5<br>9.75             | 45.00<br>0.14<br>0.09                       |
| Arama<br>0.4 std    | 9827<br>100.95<br>9.90              | 316<br>5.05<br>1.56                      | 5000<br>100.5<br>9.75             | 45.00<br>0.37<br>0.40                       |
| PF+Arama<br>0.4 std | 9812<br>100.55<br>9.91              | 317<br>3.99<br>1.29                      | 9800<br>100.5<br>9.75             | 44.98<br>0.26<br>0.17                       |

Tablo 6.14: Senaryo 2 sonuçları.

| Alg/Sonuçlar        | Ortalama<br>Mesafe<br>Rota<br>Sürat | StandardSapma<br>Mesafe<br>Rota<br>Sürat | Medyan<br>Mesafe<br>Rota<br>Sürat | B.kerterizOrt<br>B.kerterizStd<br>F.AçırMse |
|---------------------|-------------------------------------|------------------------------------------|-----------------------------------|---------------------------------------------|
| Arama<br>0.1 std    | 4999<br>100.61<br>9.70              | 14<br>0.45<br>0.19                       | 5000<br>100.5<br>9.75             | 45.02<br>0.10<br>0.10                       |
| PF+Arama<br>0.1 std | 4999<br>100.55<br>9.73              | 14<br>0.34<br>0.15                       | 5000<br>100.5<br>9.75             | 45.02<br>0.08<br>0.05                       |
| Arama<br>0.2 std    | 5006<br>100.55<br>9.76              | 72<br>0.95<br>0.44                       | 5000<br>100.5<br>9.75             | 45.00<br>0.20<br>0.20                       |
| PF+Arama<br>0.2 std | 5004<br>100.57<br>9.75              | 80<br>0.84<br>0.42                       | 5000<br>100.5<br>9.75             | 45.00<br>0.08<br>0.16                       |
| Arama<br>0.4 std    | 5013<br>100.56<br>9.83              | 145<br>2.08<br>0.90                      | 5000<br>100.5<br>9.75             | 45.00<br>0.45<br>0.41                       |
| PF+Arama<br>0.4 std | 5021<br>100.44<br>9.87              | 148<br>1.83<br>0.86                      | 5000<br>100.5<br>9.75             | 44.99<br>0.34<br>0.19                       |

Tablo 6.15: Senaryo 3 sonuçları.

| Alg/Sonuçlar        | Ortalama<br>Mesafe<br>Rota<br>Sürat | StandardSapma<br>Mesafe<br>Rota<br>Sürat | Medyan<br>Mesafe<br>Rota<br>Sürat | B.kerterizOrt<br>B.kerterizStd<br>F.AçırMse |
|---------------------|-------------------------------------|------------------------------------------|-----------------------------------|---------------------------------------------|
| Arama<br>0.1 std    | 7657<br>121.22<br>12.49             | 187<br>1.04<br>0.57                      | 7600<br>121.5<br>12.25            | 45.00<br>0.10<br>0.10                       |
| PF+Arama<br>0.1 std | 7638<br>121.34<br>12.36             | 155<br>0.82<br>0.45                      | 7600<br>121.5<br>12.25            | 45.00<br>0.07<br>0.05                       |
| Arama<br>0.2 std    | 7620<br>121.52<br>12.31             | 275<br>1.74<br>0.87                      | 7600<br>121.5<br>12.25            | 45.01<br>0.21<br>0.20                       |
| PF+Arama<br>0.2 std | 7615<br>121.51<br>12.29             | 272<br>1.41<br>0.76                      | 7600<br>121.5<br>12.25            | 45.01<br>0.14<br>0.09                       |
| Arama<br>0.4 std    | 7628<br>121.90<br>12.33             | 535<br>3.44<br>1.73                      | 7600<br>121.5<br>12.25            | 45.03<br>0.36<br>0.39                       |
| PF+Arama<br>0.4 std | 7643<br>121.75<br>12.37             | 546<br>3.13<br>1.65                      | 7600<br>121.5<br>12.25            | 44.98<br>0.26<br>0.18                       |

Tablo 6.16: Senaryo 4 sonuçları.

| Alg/Sonuçla<br>r    | Ortalama<br>Mesafe<br>Rota<br>Sürat | StandardSapma<br>Mesafe<br>Rota<br>Sürat | Medyan<br>Mesafe<br>Rota<br>Sürat | B.kerterizOrt<br>B.kerterizStd<br>F.AçırMse |
|---------------------|-------------------------------------|------------------------------------------|-----------------------------------|---------------------------------------------|
| Arama<br>0.1 std    | 9788<br>60.46<br>9.83               | 203<br>0.60<br>0.60                      | 9800<br>60.5<br>9.75              | 45.00<br>0.10<br>0.10                       |
| PF+Arama<br>0.1 std | 9783<br>60.44<br>9.84               | 166<br>0.57<br>0.55                      | 9800<br>60.5<br>9.75              | 44.99<br>0.07<br>0.04                       |
| Arama<br>0.2 std    | 9789<br>60.56<br>9.81               | 376<br>1.13<br>1.06                      | 9800<br>60.5<br>9.75              | 44.99<br>0.19<br>0.20                       |
| PF+Arama<br>0.2 std | 9792<br>60.51<br>9.86               | 305<br>1.03<br>0.97                      | 9800<br>60.5<br>9.75              | 44.99<br>0.13<br>0.09                       |
| Arama<br>0.4 std    | 9865<br>61.26<br>9.69               | 875<br>3.14<br>2.26                      | 9800<br>60.5<br>9.50              | 45.00<br>0.41<br>0.39                       |
| Arama<br>0.4 std    | 9868<br>60.99<br>9.68               | 614<br>2.21<br>1.83                      | 9800<br>60.5<br>9.75              | 45.00<br>0.27<br>0.16                       |

Tablo 6.17: Senaryo 5 sonuçları.

| Alg/Sonuçla<br>r    | Ortalama<br>Mesafe<br>Rota<br>Sürat | StandardSapma<br>Mesafe<br>Rota<br>Sürat | Medyan<br>Mesafe<br>Rota<br>Sürat | B.kerterizOrt<br>B.kerterizStd<br>F.AçıRmse |
|---------------------|-------------------------------------|------------------------------------------|-----------------------------------|---------------------------------------------|
| Arama<br>0.1 std    | 5010<br>60.53<br>9.73               | 91<br>0.24<br>0.26                       | 5000<br>60.5<br>9.75              | 45.00<br>0.10<br>0.10                       |
| PF+Arama<br>0.1 std | 5004<br>60.52<br>9.74               | 63<br>0.17<br>0.20                       | 5000<br>60.5<br>9.75              | 45.00<br>0.07<br>0.05                       |
| Arama<br>0.2 std    | 5010<br>60.56<br>9.73               | 175<br>0.50<br>0.56                      | 5000<br>60.5<br>9.75              | 45.00<br>0.20<br>0.20                       |
| PF+Arama<br>0.2 std | 4998<br>60.53<br>9.74               | 137<br>0.41<br>0.47                      | 5000<br>60.5<br>9.75              | 45.00<br>0.15<br>0.09                       |
| Arama<br>0.4 std    | 5023<br>60.57<br>9.77               | 304<br>0.78<br>0.93                      | 5000<br>60.5<br>9.75              | 45.01<br>0.40<br>0.40                       |
| PF+Arama<br>0.4 std | 5009<br>60.53<br>9.79               | 252<br>0.72<br>0.88                      | 5000<br>60.5<br>9.75              | 45.01<br>0.30<br>0.18                       |

Tablo 6.18: Senaryo 6 sonuçları.

| Alg/Sonuçla<br>r    | Ortalama<br>Mesafe<br>Rota<br>Sürat | StandardSapma<br>Mesafe<br>Rota<br>Sürat | Medyan<br>Mesafe<br>Rota<br>Sürat | B.kerterizOrt<br>B.kerterizStd<br>F.AçıRmse |
|---------------------|-------------------------------------|------------------------------------------|-----------------------------------|---------------------------------------------|
| Arama<br>0.1 std    | 7609<br>60.53<br>9.73               | 156<br>0.44<br>0.44                      | 7600<br>60.5<br>9.75              | 45.01<br>0.10<br>0.10                       |
| PF+Arama<br>0.1 std | 7598<br>60.49<br>9.77               | 124<br>0.39<br>0.40                      | 7600<br>60.5<br>9.75              | 45.00<br>0.07<br>0.04                       |
| Arama<br>0.2 std    | 7625<br>60.66<br>9.65               | 289<br>0.81<br>0.79                      | 7600<br>60.5<br>9.75              | 45.00<br>0.20<br>0.20                       |
| PF+Arama<br>0.2 std | 7606<br>60.58<br>9.71               | 223<br>0.69<br>0.68                      | 7600<br>60.5<br>9.75              | 45.00<br>0.14<br>0.09                       |
| Arama<br>0.4 std    | 7630<br>60.71<br>9.84               | 605<br>1.70<br>1.66                      | 7600<br>60.5<br>9.88              | 44.98<br>0.45<br>0.40                       |
| PF+Arama<br>0.4 std | 7613<br>60.60<br>9.86               | 430<br>1.36<br>1.42                      | 7600<br>60.5<br>10                | 44.99<br>0.31<br>0.18                       |

Tablo 6.19: Senaryo 7 sonuçları.

| Alg/Sonuçlar        | Ortalama<br>Mesafe<br>Rota<br>Sürat | StandardSapma<br>Mesafe<br>Rota<br>Sürat | Medyan<br>Mesafe<br>Rota<br>Sürat | B.kerterizOrt<br>B.kerterizStd<br>F.AçırMse |
|---------------------|-------------------------------------|------------------------------------------|-----------------------------------|---------------------------------------------|
| Arama<br>0.1 std    | 12634<br>121.43<br>12.32            | 301<br>1.68<br>0.64                      | 12600<br>121.5<br>12.25           | 45.00<br>0.10<br>0.10                       |
| PF+Arama<br>0.1 std | 12632<br>121.38<br>12.33            | 336<br>1.66<br>0.66                      | 12600<br>121.5<br>12.25           | 44.99<br>0.07<br>0.04                       |
| Arama<br>0.2 std    | 12636<br>121.63<br>12.36            | 600<br>3.75<br>1.35                      | 12600<br>121.5<br>12.25           | 45.00<br>0.20<br>0.20                       |
| PF+Arama<br>0.2 std | 12692<br>121.25<br>12.45            | 529<br>2.71<br>1.05                      | 12600<br>121.5<br>12.25           | 45.00<br>0.13<br>0.08                       |
| Arama<br>0.4 std    | 12779<br>122.12<br>12.75            | 1286<br>7.72<br>2.99                     | 12600<br>121.25<br>12.5           | 44.99<br>0.43<br>0.40                       |
| PF+Arama<br>0.4 std | 12709<br>121.84<br>12.54            | 1086<br>5.74<br>2.16                     | 12600<br>121.5<br>12.25           | 44.99<br>0.28<br>0.17                       |

Tablo 6.20: Senaryo 8 sonuçları.

| Alg/Sonuçla<br>r    | Ortalama<br>Mesafe<br>Rota<br>Sürat | StandardSapma<br>Mesafe<br>Rota<br>Sürat | Medyan<br>Mesafe<br>Rota<br>Sürat | B.kerterizOrt<br>B.kerterizStd<br>F.AçırMse |
|---------------------|-------------------------------------|------------------------------------------|-----------------------------------|---------------------------------------------|
| Arama<br>0.1 std    | 7610<br>100.28<br>9.84              | 74<br>0.99<br>0.34                       | 7600<br>100.5<br>9.75             | 44.99<br>0.10<br>0.10                       |
| PF+Arama<br>0.1 std | 7610<br>100.34<br>9.82              | 72<br>0.91<br>0.32                       | 7600<br>100.5<br>9.75             | 44.99<br>0.07<br>0.05                       |
| Arama<br>0.2 std    | 7610<br>100.32<br>9.86              | 141<br>1.73<br>0.63                      | 7600<br>100.5<br>10               | 44.98<br>0.18<br>0.20                       |
| PF+Arama<br>0.2 std | 7625<br>100.28<br>9.88              | 148<br>1.68<br>0.62                      | 7600<br>100.5<br>10               | 44.99<br>0.14<br>0.09                       |
| Arama<br>0.4 std    | 7615<br>100.73<br>9.85              | 223<br>3.63<br>1.27                      | 7600<br>100.5<br>9.75             | 44.99<br>0.40<br>0.40                       |
| PF+Arama<br>0.4 std | 7626<br>100.44<br>9.90              | 251<br>2.92<br>1.07                      | 7600<br>100<br>10                 | 44.99<br>0.27<br>0.18                       |

Tablo 6.21: Senaryo 9 sonuçları.

| Alg/Sonuçlar        | Ortalama<br>Mesafe<br>Rota<br>Sürat | StandardSapma<br>Mesafe<br>Rota<br>Sürat | Medyan<br>Mesafe<br>Rota<br>Sürat | B.kerterizOrt<br>B.kerterizStd<br>F.AçıRmse |
|---------------------|-------------------------------------|------------------------------------------|-----------------------------------|---------------------------------------------|
| Arama<br>0.1 std    | 7605<br>100.41<br>7.29              | 65<br>1.39<br>0.29                       | 7600<br>100.5<br>7.25             | 44.99<br>0.10<br>0.10                       |
| PF+Arama<br>0.1 std | 7601<br>100.44<br>7.28              | 47<br>1.25<br>0.25                       | 7600<br>100.5<br>7.25             | 45.00<br>0.07<br>0.05                       |
| Arama<br>0.2 std    | 7601<br>100.41<br>7.32              | 109<br>2.64<br>0.54                      | 7600<br>100.25<br>7.25            | 44.98<br>0.18<br>0.20                       |
| PF+Arama<br>0.2 std | 7599<br>100.42<br>7.30              | 100<br>2.26<br>0.46                      | 7600<br>100.25<br>7.25            | 44.99<br>0.14<br>0.09                       |
| Arama<br>0.4 std    | 7609<br>101.09<br>7.32              | 201<br>5.51<br>1.06                      | 7600<br>101<br>7.25               | 44.99<br>0.40<br>0.40                       |
| Arama<br>0.4 std    | 7604<br>100.68<br>7.32              | 191<br>4.04<br>0.83                      | 7600<br>100<br>7.25               | 44.99<br>0.27<br>0.18                       |

#### 6.4.2. Büyük Patlama Büyük Çöküş Algoritması İle Karşılaştırma

TMA için, önerilen Parçacık filtresi algoritması ile, aynı senaryolarda, bir evrimsel arama yöntemi olan, büyük patlama büyük çöküş en iyi yenileme algoritması performansı ile da karşılaştırılmıştır. Amaç Değerlemeli Büyük Patlama Büyük Çöküş algoritması bir evrimsel arama yöntemidir. İlk defa [91] makalede önerilmiştir. Bu metotta popülasyonun en kötü bireyleri az sayıda ve çok daha büyük bir çapta saçılırken, en iyi bireyler daha fazla ve daha kısa mesafede saçılmaktadır. Bu şekilde aramanın ana dayanağı, “iyi” bireylerin yakınlığında daha fazla derinlemesine arama sağlayıp “kötü” bireylerde daha geniş çaplı saçılma etkisi yaratıp, mutasyon etkisi oluşturmaktır. [78]. Test edilen senaryolarda zaman aralığı [78] makaleye uygun şekilde 20 saniye olarak belirlenmiştir. Her bir bacadta 10 dakika kalınarak toplam 20 dakikalık bir veri üzerinde 60 kerteriz ölçümünde deneyler yapılmıştır. Gözlemcinin hızı 5m/s olarak alınmıştır. Gözlemcinin başlangıç noktası [0,0] olarak alınmıştır. Test edilen senaryolar Tablo 6.22’de görülmektedir.

Tablo 6.22: Senaryo 10-15.

| Senaryo No | Hedef Mesafesi | Hedef Rotası | Hedef Süratı | Kerteriz Gürültüsü |
|------------|----------------|--------------|--------------|--------------------|
| 10         | 5000 m         | 120°         | 9.26m/s      | 0.01°              |
| 11         | 13000 m        | 120°         | 5m/s         | 0.01°              |
| 12         | 13000 m        | 140°         | 9.26m/s      | 0.05°              |
| 13         | 20000 m        | 120°         | 15m/s        | 0.01°              |
| 14         | 20000 m        | 100°         | 25m/s        | 0.05°              |
| 15         | 40000 m        | 100°         | 25m/s        | 0.05°              |

Tablo 6.23’de önerilen algoritmanın 6.22’deki tüm senaryolarda performansı ile Büyük Patlama Büyük Çöküş algoritmasının bu senaryolarda performansı görülmektedir. Tablo 6.23’deki değerler ortalama hata ile standart sapma değerlerini belirtmektedir. Monte Carlo simülasyonu sonucu elde edilen tahmin değerlerinin ortalaması ile gerçek değerlerin farkı olan ortalama hata değeri ve bu tahmin değerlerinin standart sapması görülmektedir. Amaç Değerlemeli Büyük Patlama Büyük Çöküş en iyi yenileme algoritması ile önerdiğiiz parçacık filtreleri çözüm algoritması karşılaştırıldığında, özellikle 0.05 kerteriz gürültülü 14. ve 15. Senaryolarda, Parçacık filtresi ile önerilen yaklaşımın tartışılmaz bir şekilde üstün olduğu görülmektedir. Örneğin, 14. senaryoda Büyük Patlama Büyük Çöküş algoritmasında gerçek değer ile ortalama değer arasında, Range 1666 metre hata rotada 0.50 derece ve hızda 1.10 m/s hata varken, Önerilen algoritmada ise, Range 22 metre rotada 0.01 derece ve hızda ise 0.04m/s hata mevcuttur. Genel olarak standart sapma değerleri de düşüktür.

Tablo 6.23: Önerilen algoritma ve Büyük Patlama Büyük Çöküş algoritmasının perfomansları.

| Senaryo Numarası | Mesafe Önerilen Alg. | Mesafe Big Bang | Rota Önerilen Alg. | Rota Big Bang | Sürat Önerilen Alg. | Sürat Big Bang |
|------------------|----------------------|-----------------|--------------------|---------------|---------------------|----------------|
| 10               | 0                    | 9.6             | 0                  | 0.17          | 0.01                | 0.02           |
|                  | 0                    | 23.94           | 0                  | 0.19          | 0                   | 0.05           |
| 11               | 0                    | 114             | 0                  | 3.75          | 0                   | 0.22           |
|                  | 0                    | 55.4            | 0                  | 3.34          | 0                   | 0.28           |
| 12               | 181                  | 228             | 1.16               | 0.44          | 0.22                | 0.10           |
|                  | 231                  | 303             | 1.47               | 0.08          | 0.29                | 0.14           |
| 13               | 0                    | 103             | 0                  | 0.02          | 0                   | 0.06           |
|                  | 0                    | 198             | 0                  | 0.08          | 0                   | 0.12           |
| 14               | 22                   | 1666            | 0.01               | 0.53          | 0.04                | 1.10           |
|                  | 333                  | 741.34          | 0.34               | 0.77          | 0.70                | 1.15           |
| 15               | 37                   | 1825            | 0.02               | 1.16          | 0.05                | 1.34           |
|                  | 1194                 | 1688            | 1.15               | 1.02          | 1.36                | 1.28           |

#### 6.4.3. Algoritma Çalışma Süreleri

2,2 Ghz bir PC’de, Matlab programlama dili kullanılarak, tek çekirdek kullanarak, elde edilen algoritma çalışma süreleri Tablo 6.24’de görülmektedir. Algoritmanın hızları gayet iyidir. Daha hızlı CPU saat frekansına sahip bir PC’de, Matlab yerine daha hızlı programlar yazmak mümkün olan C programlama dilinin kullanılması, çok çekirdek kullanarak paralel bir şekilde arama yapılması durumunda, daha da hızlanacaktır.

Tablo 6.24: Algoritma çalışma süreleri.

| Algoritma                    | Çalışma süresi |
|------------------------------|----------------|
| Arama Algoritması            | 1.7 sn         |
| PF Algoritması N=200, 1 iter | 0.02 sn        |

Tablo 6.24: Devam.

|                               |               |
|-------------------------------|---------------|
| PF Algoritması N=200, 20 iter | 0.4 sn        |
| Tüm süreç                     | 3 sn – 3.5 sn |

## 6.5. Genel Değerlendirme

Bu bölümde, kerteriz tabanlı hedef hareket analizi için çözüm algoritması önerilmiştir. Öncelikle parçacık filtresinin başlangıç değeri atanması için arama algoritması ve gözlemlenen kerteriz açıları kullanılarak ve belli aralıklarla aranarak en iyi  $R$ ,  $C$ ,  $S$  belirlenmektedir. Daha sonra ise, filtrelenmiş açı değerleri yeniden aranmaktadır. Parçacık filtresi çözümü için iki bacaktaki hareket ayrı ayrı çözülmektedir. Başlangıçtaki kerterizdeki hatayı azaltmak için ilk bacaktaki hareket sondan başa doğru olarak çözülmektedir.

Arama algoritmasında, maliyet fonksiyonu olarak RMSE değeri kullanılmalıdır. Parçacık sayısı 50 ile 200 arası seçilebilir. Süreç gürültüsünde  $x$  ve  $y$  konumlarının 1 derecelik gürültüde, standart sapmaları 50 metre ve sürat için 0.5 m/s olarak alınabilir. Ölçüm gürültüsünün standart sapma değeri ise arama sonucu elde edilen en iyi  $R$ ,  $C$ ,  $S$  ait RMSE değeri kullanılabilir. Yeniden örnekleme algoritmalarının performansları birbirine yakındır. Açgözlü algoritma eşik değeri olarak 0.7 ile 0.9 arasında seçilebilir. Test edilen tüm senaryolarda, parçacık filtresinin açı RMSE değeri daha düşük, başlangıç kerteriz açısındaki RMSE daha düşük ve  $R$ ,  $C$ ,  $S$  standart sapma değerleri de daha düşüktür.

## 7. SONUÇLAR VE GELECEK ÇALIŞMA ÖNERİLERİ

Hedef takibi probleminde, doğrusal olmayan modeller ve Gauss olasılıksal dağılıma sahip olmayan gürültüde kullanılması nedeniyle tercih edilmektedir. Parçacık filtresinde hatayı azaltan en iyi yaklaşım yeniden örnekleme algoritmasıdır. Bu tez çalışmasında, yeniden örnekleme algoritmaları üzerine çalışılmıştır. Açgözlü yeniden örnekleme algoritması, torba problemi çözümü yeniden örnekleme algoritması, LSTM derin öğrenme tabanlı özgün yeniden örnekleme algoritması geliştirilmiştir. Kerteriz tabanlı hareket modeli içinde parçacık filtresini kullanan çözüm algoritması geliştirilmiştir.

Açgözlü yeniden örnekleme algoritmasında, parçacıkların ağırlıkları önemli bilgi olarak kullanılmıştır. Açgözlü algoritmada daha yüksek ağırlığa sahip parçacıkların ilk olarak seçilmektedir. Eşik değeri farklı parametreler için iyileştirme parametresi olarak kullanılabilir. Parçacıkların ağırlıkları sıralanarak ve eşik değeri belirlenerek, düşük ağırlıklı parçacıklar sistemden çıkarılmakta ve onların yerine daha yüksek ağırlıklı parçacıklar çoğaltılmaktadır.

Torba problemi çözümünde daha yüksek ağırlığa sahip parçacıklar en değerli parçacıklar olduğu düşünülerek çözüm geliştirilmiştir. Parçacıkların ağırlıkları torbaya konulacak eşyaların ağırlıkları, parçacıkların ağırlıklarının karesi torbaya konulacak eşyaların değerleri olduğu varsayılmıştır. Böylece eşyaların değerleri değer/ağırlık oranı parçacıkların ağırlıkları doğru orantılı olabilir. Ağırlık kısıtlaması da bu algoritmada iyileştirme parametresi olarak kullanılabilir. Doğrusal olmayan hareket modeli, kartezyen hareket modeli, sürekli dönüş modelindeki deneysel sonuçlarda, torba çözümü yeniden örnekleme algoritması en iyi performansı verirken, Açgözlü yeniden örnekleme algoritması ikinci en iyi performansı vermiştir.

LSTM diziden diziye problemi çözme özelliği kullanılarak geliştirilen Derin öğrenme tabanlı yeniden örnekleme algoritması da, üretildiği algoritmalarından daha iyi performans göstermiş ve daha az tahmin hatasına sahiptir.

Hedef hareket analizinde ise; arama algoritması, parçacık filtresi ve doğrusal regresyon kullanılarak oluşturulan algoritma bir çok farklı senaryoda çok az hatayla problemi çözebilmektedir. Mesafe, rota ve sürat kestirimi çok az hatayla kestirilebilmektedir. Ayrıca, önerilen algoritma, birçok farklı senaryoda, Büyük

Patlama Büyük Çöküş en iyi yenileme algoritmasından da daha iyi performansa sahiptir.

Gelecek çalışma önerileri için, LSTM yeniden örnekleme algoritmaları ilgili daha kapsamlı çalışmalar yapılabilir. Kaliteli grafik işlemci kartımıza sahip olmadığımız için, yani donanım kısıtlaması sebebiyle, detaylı testler yapılamadı. Örneğin, bir dil çevirisinde, bir cümle için birden fazla referans cümle olabilir. Yeniden örnekleme algoritmasında ise, LSTM aynı yada farklı giriş dizileri için, çıkış dizilerinde birden fazla yeniden örnekleme algoritmasının sonuçları yer alabilir. Böylece hibrit yeniden örnekleme algoritmaları geliştirilebilir. LSTM algoritması ile üretilen algoritmalar farklı hareket modellerinde test edilebilir. Bu çalışmada önemli parametrelerin etkisi araştırılmıştır. LSTM ilgili diğer parametrelerin de sonuçlara nasıl etki ettiği de araştırılabilir. Örneğin, eğitim verimizin boyutu 200 bin idi. Eğitim setinin boyutu, 500 bin ve 1 milyon olduğunda doğruluğa etki edip etmediği araştırılabilir. LSTM algoritması ile yapılan çalışmalar, LSTM doğal dil işleme alanında sıkça kullanıldığı için, doğal dil işleme alanında katkılar sağlayacaktır.

## KAYNAKLAR

- [1] Arulampalam M. S., Maskell S., Gordon N., Clapp T., (2002), “A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking”, IEEE Proceedings on Signal Processing, 50 (2), 174-188.
- [2] Welch G., Bishop G., (1995), “An introduction to the kalman filter”, Department of Computer Science, University of North Carolina, USA.
- [3] Çayiroğlu İ., (2012), “Kalman filtresi ve programlama”, Mekatronik Mühendisliği Bölümü, Karabük Üniversitesi, Türkiye.
- [4] Kalman R. E., (1960), “A new approach to linear filtering and prediction problems”, Transaction of the ASME-Journal of Basic Engineering, 82 (1), 35-45.
- [5] Gustafsson F., Gunnarsson F., Bergman N., Forssell U., Jansson J., Karlsson R., Nordlund P., (2002), “Particle filters for positioning, navigation and tracking”, IEEE Transactions On Signal Processing, 50 (2), 425-432.
- [6] Djuric P. M., Kotecha J. H., Zhang J., Y. Huang, Ghirmai T., M. F. Bugallo M. F., Miguez J., (2003), “Particle Filtering”, IEEE Signal Processing Magazine, 20 (5), 19-38.
- [7] Koziarski P., Lis M., Tkiewicz J. Z., (2013), “Resampling in particle filtering – comparison”, Poznan Teknoloji Üniversitesi, Polonya.
- [8] Li T., Bolic M., Djuric P. M., (2015), “Resampling methods for particle filtering: Classification, Implementation, and Strategies”, IEEE Signal Processing Magazine, 70-86.
- [9] Yan Z., Zhaobin L., Zhang B., Yu F., (2017), “A novel adaptive resampling particle filtering algorithm”, CSE and EUC 2017 IEEE International Conference, pp.1-4, Guangzhou, China, 21-24 Temmuz.
- [10] Stordal A. S., (2008), “Sequential monte carlo methods for bayesian filtering”, Yüksek Lisans Tezi, Bergen Üniversitesi.
- [11] Benedict T. R., Bordner G. W., (1962), “Synthesis of an optimal set of radar track-while-scan smoothing equations”, IRE Trans. on Automatic Control, 7(4), 27-32.
- [12] Ng K. H., Yeong C. F., Su E. L. M., Wong L. X., (2012), “Alpha beta gamma filter for cascaded PID motor position control”, Procedia Engineering, 41, 244-250.

- [13] Simpson H. R., (1963), "Performance measures and optimization condition for a third order sampled-data tracker", IEEE Transactions on Automatic Control, 8(2), 182-183.
- [14] Jatoth R. K., Gopisetty S., Hussain M., (2015), "Performance analysis of alpha beta filter, kalman filter and meanshift for object tracking in video sequences", International Journal of Image, Graphics and Signal Processing, 7 (3), 24-30.
- [15] Ljung L., (1979), "Asymptotic behavior of the extended kalman filter as a parameter estimator for linear systems", IEEE Transactions on Automatic Control, 24 (1), 36-50.
- [16] Julier S. J., Uhlmann J. K., (1997). "New extension of the kalman filter to nonlinear systems", In Signal Processing, Sensor Fusion and Target Recognition VI, 3068 (1), 182-193.
- [17] Wan E. A, Van Der Merwe R.,(2000), "The unscented kalman filter for nonlinear estimation", In Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No.00EX373), 153-158, Lake Louise, Canada, 4 Ekim.
- [18] LaViola J. J., (2003), "A comparison of unscented and extended kalman filtering for estimating quaternion motion", Proceedings of the 2003 American Control Conference, 2435-2440, Denver, Co, USA, 4-6 Haziran.
- [19] Uçar M. B., (2015), "Çoklu model parçacık filtrelerinde ağırlıklandırılmış istatistiksel model seçimi", Yüksek Lisans Tezi, Başkent Üniversitesi.
- [20] Manavoğlu B., (2014), "İki boyutta hedef izleme ve simülasyonu", Yüksek Lisans Tezi, Hacettepe Üniversitesi.
- [21] Gordon N. J., Salmond D. J., Smith A. F. M., (1993), "Novel approach to nonlinear/non-gaussian bayesian state estimation", IEEE Proceedings-F, 140 (2), 107–113.
- [22] Kitagawa G., "Monte carlo filter and smoother for non-gaussian nonlinear state space models", Journal of Computational and Graphical Statistics", 5 (1), 1–25, 1996.
- [23] Carpenter J., Clifford P., Fearnhead P., (1999), "Improved particle filter for nonlinear problems", IEEE Proceedings – Radar, Sonar and Navigation, 146 (1), 2–7.
- [24] Douc R., Cappe O., Moulines E., (2005), "Comparison of resampling schemes for particle filtering", Proceedings of the 4th International Symposium on Image and Signal Processing and Analysis, 64-69, Zagreb, Croatia, 15-17 Eylül.

- [25] Li T., Villarrubia G., Sun S., Corchado J. M., Bajo J., (2015), “Resampling methods for particle filtering: identical distribution, a new method, and comparable study”, *Frontiers of Information Technology & Electronic Engineering*, 16 (11), 969-984.
- [26] Cai X. (2009), “Canonical coin systems for change-making problems”, In 2009 IEEE Ninth International Conference on Hybrid Intelligent Systems, 1, 499-504, Shenyang, China, 12-14 Augustos.
- [27] Greenberg H. J., (1998), “Greedy algorithms for minimum spanning tree”, University of Colorado, Denver, USA.
- [28] Kruskal J. B., (1956), “On the shortest spanning tree of a graph and the traveling salesman problem”, *Proceedings of the American Mathematical Society*, 7 (1), 48-50.
- [29] Prim R. C., “Shortest connection networks and some generalizations”, *Bel 1 Systems Technology Journal*, 36 (6), 1389-1401.
- [30] Mirino A. E., (2017), “Best routes selection using dijkstra and floyd-warshall algorithm”, In 2017 11th IEEE International Conference on Information & Communication Technology and System (ICTS), 155-158, Surabaya, Indonesia, 31 October.
- [31] Moffat A., (2019). “Huffman coding”, *ACM Computing Surveys (CSUR)*, 52 (4), 1-35.
- [32] Wilt C. M., Thayer J. T., Ruml W., (2010), “A comparison of greedy search algorithms”, In third Annual Symposium on Combinatorial Search, pp. 1-8, Stone Mountain, Atlanta, Georgia, USA, 8-10 Temmuz.
- [33] Wilt C. M., Ruml W., (2015), “Building a Heuristic for Greedy Search”, In SOCS, Department of Computer Science, University of New Hampshire, USA.
- [34] Romeijn H. E., Morales D. R., (2000). “A class of greedy algorithms for the generalized assignment problem”, *Discrete Applied Mathematics*, 103 (1-3), 209-235.
- [35] Li J., Feng L., Fang S., (2014). “An greedy-based Job Scheduling Algorithm in Cloud Computing”, *JSW*, 9 (4), 921-925.
- [36] Chuzhoy J., Ostrovsky R., Rabani Y., (2006). “Approximation algorithms for the job interval selection problem and related scheduling problems”, *Mathematics Of Operations Research*, 31 (4), 730-738.
- [37] Langlais P., Patry A., Gotti F., (2007), “A greedy decoder for phrase-based statistical machine translation”, *Proc. of TMI*, 1-269.

- [38] Hristakeva M., Shrestha, D., (2005), "Different approaches to solve the 0/1 knapsack problem", In The Midwest Instruction and Computing Symposium, Computer Science Department, Simpson College, Indianola, USA.
- [39] Bellman R., (1952), "On the theory of dynamic programming", Proceedings of the National Academy of Sciences of the United States of America, 38 (8), 716-719.
- [40] Bellman R., (1956), "Notes on the theory of dynamic programming IV-maximization over discrete sets", Naval Research Logistics Quarterly, 3 (1-2), 67-70.
- [41] Web 1, (2021), <http://www.eolss.net/Sample-Chapters/C02/E6-05-01-04.pdf>, (Erişim Tarihi: 15/06/2021).
- [42] Web 2, (2021), <https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/06DynamicProgrammingI-2x2.pdf>, (Erişim Tarihi: 15/06/2021).
- [43] Levenshtein V. I, (1966), "Binary codes capable of correcting deletions, insertions, and reversals" In Soviet Physics Doklady, 10 (8), 707-710.
- [44] Forney G. D., (1973), "The viterbi algorithm", Proceedings of the IEEE, 61 (3), 268-278.
- [45] Koiliaris K., Xu C., (2018), "Subset sum made simple", arxiv preprint arxiv:1807.08248, Department of Computer Science, University of Illinois, USA.
- [46] Tsai Y. T., (2003), "The constrained longest common subsequence problem", Information Processing Letters, 88 (4), 173-176.
- [47] Malandraki C., Dial R. B., (1996), "A restricted dynamic programming heuristic algorithm for the time dependent traveling salesman problem", European Journal of Operational Research, 90 (1), 45-55.
- [48] Barthels, H., Copik M., Bientines P., (2018), "The generalized matrix chain algorithm", In Proceedings of the 2018 International Symposium on Code Generation and Optimization, 138-148, Vienna, Austria, 24-28 Şubat.
- [49] Gromicho J. A., Van Hoorn J. J., Saldanha-da-Gama F., Timmer G. T. (2012), "Solving the job-shop scheduling problem optimally by dynamic programming", Computers & Operations Research, 39 (12), 2968-2977.
- [50] Mehmood A., (2019), "Ash cc algo coin change algorithm optimization", International Journal of Computer Applications 975-8887, 178 (5), 1-9.

- [51] Horowitz E., Sahni S., (1974), “Computing partitions with applications to the knapsack problem”, *Journal of the Association for Computing Machinery*, 21 (2), 77–292.
- [52] Toth P., (1980), “Dynamic programming algorithms for the zero-one knapsack problem”, *Computing* 25, 25 (1), 29-45.
- [53] Figueira J., Paquete L., Simoes M., Vanderpooten D., (2013), “Algorithmic improvements on dynamic programming for the bi-objective 0,1 knapsack problem”, *Computational Optimization and Applications*, 50 (1), 97-111.
- [54] Pan X., Zhang T., (2018), “Comparison and analysis of algorithms for the 0/1 knapsack problem”, *Journal of Physics Conference Series*, 1069 (1), 1-8.
- [55] Axiotis K., Tzamos C., (2018), “Capacitated dynamic programming: faster knapsack and graph algorithms”, 45th International Colloquium on Automata Languages and Programming (ICALP), 1-18, Prague, Czech Republic, 9-13 Temmuz.
- [56] Furini F., Ljubi I., Sinnl M., (2016), “Effective dynamic programming algorithm for the minimum-Cost maximal knapsack packing”, *European Journal of Operational Research*, 262 (2), 1-23.
- [57] Levitin A, (2011), “The design and analysis of algorithms”, 3rd edition, Pearson Education Inc., 292-297.
- [58] Web 3, (2021), [http://www.yisongyue.com/courses/cs155/2018\\_winter/lectures/Recitation5.pdf](http://www.yisongyue.com/courses/cs155/2018_winter/lectures/Recitation5.pdf), (Erişim Tarihi: 15/06/2021).
- [59] Hristakeva M., Shrestha D., (2005), “Different approaches to solve the 0/1 knapsack problem”, *Computer Science Department of Simpson College*, Indianola, USA.
- [60] Verwimp L., Pelemans J., Wambacq P., (2017), “Character-word LSTM language models”, *arXiv preprint arXiv:1704.02813*, 1-11.
- [61] Athiwaratkun B., Stokes J. W., (2017), “Malware classification with LSTM and GRU language models and a character-level CNN”, In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2482-2486, New Orleans, LA, USA, 5-9 Mart.
- [62] Wu Y., Schuster M., Chen Z., Le Q. V., Norouzi M., Macherey W., Klingner J., (2016). “Google's neural machine translation system: bridging the gap between human and machine translation”, *arXiv preprint arXiv:1609.08144*, 1-23.

- [63] Alharbi R., Magdy W., Darwish K., Abdelali A., Mubarak H., (2018), “Part-of-speech tagging for arabic gulf dialect using bi-LSTM”, In LREC Proceedings of the Eleventh International Conference on Language Resources and Evaluation, 1-8, Miyazaki, Japan, 7-12 Mayıs.
- [64] Zaky D., Romadhony A., (2019), “An LSTM-based spell checker for indonesian text”, In 2019 IEEE International Conference of Advanced Informatics: Concepts, Theory and Applications (ICAICTA), 1-6, Yogyakarta, Indonesia, 20-21 Eylül.
- [65] Nowak J., Taspinar A., Scherer R., (2017), “LSTM recurrent neural networks for short text and sentiment classification”, In Springer, Cham International Conference on Artificial Intelligence and Soft Computing, 553-562, Zakopane, Poland, 11-15 Haziran.
- [66] Wang C., Yang H., Bartz C., Meinel C., (2016). “Image captioning with deep bidirectional LSTMs”, In Proceedings of the 24th ACM international conference on Multimedia, 988-997, Amsterdam, Netherlands, 15-19 October.
- [67] Islam M. S., Mousumi S. S. S., Abujar S., Hossain S. A., (2019), “Sequence-to-sequence bangla sentence generation with LSTM recurrent neural networks”, Procedia Computer Science, 152, 51-58.
- [68] Yücer Ş., (2018), “Üç boyutlu iskelet verilerinden metrik öğrenme tabanlı hareket tanıma”, Yüksek Lisans Tezi, Gebze Teknik Üniversitesi.
- [69] Jozefowicz R., Zaremba, W., Sutskever I., (2015), “An empirical exploration of recurrent network architectures”, In International conference on machine learning, pp. 2342-2350, Lille, France, 7-9 Temmuz.
- [70] Medsker L. R., Jain L. C., (2001). “Recurrent neural networks”, Design and Applications, 5, 1-391.
- [71] Mikolov T., Kombrink S., Burget L., Černocký J., Khudanpur S., (2011), “Extensions of recurrent neural network language model”, In 2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 5528-5531, Prague, Czech Republic, 22-27 Mayıs.
- [72] Hochreiter S., Schmidhuber J., (1997), “Long short-term memory”, Neural Computation, 9 (8), 1735-1780.
- [73] Sönmez H. H., Hocaoglu A. K., Genç H. M., (2017), “A new solution method for The passive target motion analysis problem using evolutionary strategies”, 25th IEEE Signal Processing and Communications Applications Conference (SIU), 1-4, Antalya, Turkey, 15-18 Mayıs.
- [74] Nardone S. C., Aidala V. J., (1981), “Observability criteria for bearings-only target motion analysis”, IEEE Transactions on Aerospace and Electronic Systems, 17 (2), 162–166.

- [75] Lin X., Kirubarajan; T., Bar-Shalom Y., Maskell S., (2002), "Comparison of EKF, pseudomeasurement and particle filters for a bearing-only target tracking problem", Signal and Data Processing of Small Targets 2002 International Society for Optics and Photonics, 4728, 240-250.
- [76] Li X., Zhao C., Zhang J., Li X., (2018), "Bearing-only and bearing-doppler target tracking based on EKF", In International Conference on E-Learning and Games Springer, Cham, 241-244, Xi'an, China, 28-30 Haziran.
- [77] Li X., Zhao C., Yu J., Wei W., (2016), "Underwater Bearing-Only and Bearing-Doppler Target Tracking Based on Square Root Unscented Kalman Filter", Entropy, 21 (8), 1-16,
- [78] Tokta A., Hocaoglu A. K., Genç H. M., (2017), "Target Motion Analysis with Fitness Adaptive Big Bang-Big Crunch Optimization Algorithm", 25th IEEE Signal Processing and Communications Applications Conference (SIU), 1-4, Antalya, Turkey, 15-18 Mayıs.
- [79] Wang X., Li L., Xie W., (2019), "A Novel FEM Based TS Fuzzy Particle Filtering for Bearings-Only Maneuvering Target Tracking", IEEE Transactions on Control Systems Technology, 19 (9), 1-8.
- [80] Vaishnavi G., Damodhar B. K., Rao S. K., Jahan K., (2019), "Underwater bearings-only tracking using particle filter", International Journal of Innovative Technology and Exploring Engineering (IJITEE), 8 (5), 1-5.
- [81] Zhang H., Li L., Xie W., (2018), "Constrained multiple model particle filtering for bearings-only maneuvering target tracking", IEEE Access, 6, 51721-51734.
- [82] Gordon N., Pitt M., (1998), "A comparison of sample based filters and the extended kalman filter for the bearings-only target problem", 9th European Signal Processing Conference (EUSIPCO), 8-11, Island of Rhodes, Greece, 8-11 Eylül.
- [83] Karlsson R., Gustafsson F., (2001), "Range estimation using angle-only target tracking with particle filters", In Proceedings of the 2001 IEEE American Control Conference, 3743-3748, Arlington, VA, USA, 25-27 Haziran.
- [84] Bolic M., Hong S., Djuric P. M., (2002), "Finite precision effect on performance and complexity of particle filters for bearing-only tracking", Record of the Thirty-Sixth Asilomar IEEE Conference on Signals Systems and Computers, 838-842, Pacific Grove, CA, USA, 3-6 Kasım.
- [85] Arulampalam M. S., Ristic B., Gordon N., Mansell T., (2004), "Bearings-only tracking of manoeuvring targets using particle filters", EURASIP Journal on Advances in Signal Processing, 15, 2351-2365.
- [86] Fei Z., Xing-peng Z., Xiao-Hui C., Rui-lan L., (2008), "Particle filter for underwater bearings-only passive target tracking", IEEE Pacific-Asia

Workshop on Computational Intelligence and Industrial Application, 2, 847-851.

- [87] Cao B., Ma C., Liu Z., (2012), “Particle Filter With Fine Resampling for Bearings-Only Tracking”, *Procedia Engineering*, 29, 3685-3690.
- [88] Yu J. Y., Coates M. J., Rabbat M. G., Blouin S., (2016), “A distributed particle filter for bearings-only tracking on spherical surfaces”, *IEEE Signal Processing Letters*, 23 (3), 326-330.
- [89] Kose E., (2020), “Analysis of Genetic Algorithm on Bearings-Only Target Motion Analysis”, *arXiv preprint arXiv:2001.05381*, 1-5.
- [90] Brown S. H., (2009), “Multiple linear regression analysis: a matrix approach with matlab”, *Alabama Journal of Mathematics*, 34, 1-3.
- [91] Erol O., Eksin İ., (2006), “A new optimization method: big bang–big crunch”, *Advances in Engineering Software*, 37 (2), 106-111.

## ÖZGEÇMİŞ

Ahmet Bacak 1987 yılında Isparta’da doğdu. 2005 yılında başladığı İstanbul Teknik Üniversitesi Mühendislik Fakültesi Elektronik Mühendisliği Bölümünü 2010 yılında başarıyla tamamlayarak, 2011 yılında yüksek lisans eğitimine Gebze Teknik Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalında başladı. 2014 yılında ise Gebze Teknik Üniversitesi Fen Bilimleri Enstitüsü Elektronik Mühendisliği Anabilim Dalında Doktora eğitimine başladı. 2017 yılında ise doktora yeterlilikte başarılı oldu. 2011 yılından bu yana yazılım, siber güvenlik, makine öğrenmesi ve yapay zeka alanlarında araştırmalarda ve projelerde aktif olarak yer almaktadır.

## **EKLER**

### **Ek A: Tez Çalışması Kapsamında Yapılan Yayınlar**

Bacak A., Hocaoglu A. K., (2020), “A novel resampling algorithm based on the knapsack problem”, Signal Processing, 170, 1-10.

Bacak A., Hocaoglu A. K., (2018), “Development of the resampling algorithm in particle filters using greedy approach”, 26th IEEE Signal Processing and Communications Applications Conference (SIU), 1-4, İzmir, Turkey, 2-5 Mayıs.

Bacak A., Hocaoglu A. K., (2017), “Using the minimum sampling variance resampling algorithm on target tracking”, 25th IEEE Signal Processing and Communications Applications Conference (SIU), 1-4, Antalya, Turkey, 15-18 Mayıs.

.