Otonom Robotlar İçin KU-MCL Tabanlı Yeni Bir Hibrit Konum Belirleme Algoritması Tasarımı ve Uygulaması

Design and Implementation of a New Hybrid Localization Algorithm Based on SA-MCL for Autonomous Robots

Ozan Vahit Altınpınar[1,2], *Volkan Sezer*[1,2]

[1]Kontrol ve Otomasyon Mühendisliği Bölümü
İstanbul Teknik Üniversitesi, İstanbul
altinpinaro@itu.edu.tr, sezerv@itu.edu.tr

[2]Akıllı ve Otonom Sistemler Laboratuvarı (Smart and Autonomous System Laboratory, SASLab)
İstanbul Teknik Üniversitesi, İstanbul
altinpinaro@itu.edu.tr, sezerv@itu.edu.tr

Özetçe

Lokalizasyon, bir konum tahmin problemidir ve otonom mobil robotlar üzerine yapılan çalışmalar arasında en kritik öneme sahip alanlardan biridir. Özellikle başlangıç anında robot kendi konumunu bilmiyorsa problemin zorluğu daha da artmaktadır. Robotun başlangıç anında kendi konumunu bilmemesi problemine global lokalizasyon problemi denilmiştir ve bu problemi çözmek için literatürde parçacık filtre tabanlı algoritmalar mevcuttur. Bu çalışmada ise bir global lokalizasyon algoritması olan ve başlangıç anında parçacıkların daha akıllı ve efektif bir şekilde harita üzerine atanmasını sağlayan enerji tabanlı Kendinden Uyarlamalı Monte Carlo Lokalizasyon (KU-MCL) algoritması incelenerek benzer enerji bölgelerinin daha optimal bir şekilde belirlenebilmesi için bir yöntem önerilmiştir. Bunun yanında KU-MCL algoritması nispeten daha az parçacık kullanan standart MCL algoritması ile hibrit olarak çalıştığında, orijinal KU-MCL algoritmasına göre daha doğru ve güvenilir konum tahminlerinin yapıldığı simülasyon ve gerçek ortam deneyleri ile gösterilmiştir.

Abstract

Localization is a position estimation problem and is one of the most critical areas of research in the field of autonomous mobile robots. Particularly, the difficulty of the problem increases when the robot lacks knowledge of its initial position. This situation is referred to as the global localization problem, and particle filter-based algorithms have been proposed in the literature to address this issue. In this study, we investigate an energy-based Self-Adaptive Monte Carlo Localization (SA-MCL) algorithm, which is a global localization algorithm, and propose a method to enhance the determination of similar energy regions more optimally on the map during the initialization phase. Furthermore, we demonstrate through simulations and real-world experiments that when the SA-MCL algorithm is used in a hybrid manner with the standard MCL algorithm, which employs relatively fewer particles, it provides more accurate and reliable position estimates compared to the original SA-MCL algorithm.

1. Giriş

Son yıllarda, robotik ve otonomi alanında teknolojinin hızla ilerlemesi sayesinde temizlik ve hizmet sektöründen akıllı fabrikalara kadar birçok alanda otonom mobil robotların yer aldığını görebilmekteyiz [1][2]. Dahası, otonom mobil robotlar, fiziksel engelli kişileri güvenli bir şekilde bir yerden başka bir yere taşımak ve bu kişilerin bakımını yapmak gibi tıbbi görevleri yerine getirmek için de kullanılabilirler [3]. Robotların bu tarz görevleri başarılı bir şekilde yerine getirebilmeleri ve belirlenen hedeflere güvenli bir şekilde varabilmeleri için farklı alt sistemler veya bileşenler arasındaki koordinasyon ve iş birliğinin sağlanması büyük önem arz etmektedir. Özellikle konum tahmini, haritalama, rota planlama ve engelden kaçma gibi algoritmaların birbirleriyle uyumlu bir şekilde çalışması otonom bir robotun performansı açısından hayati öneme sahiptir [4].

Bir robotun güvenli ve doğru bir şekilde hedeflenen bir konuma otonom olarak ulaşabilmesi için öncelikle bulunduğu konumu bilmesi gerekmektedir. Robot, konumunu belirlemek için ya Küresel Navigasyon Uydu Sistemi (GNSS) sinyallerini kullanır ya da üzerine monte edilmiş algılayıcılardan elde edilen ölçüm bilgilerinden konum tahmini yapabilen lokalizasyon algoritmalarından yararlanır. GNSS, açık alanlarda yüksek doğrulukta konum bilgisi sağlayabilir, ancak fabrikalar, hastaneler, madenler, depolar ve binalar gibi iç mekanlarda çalışan mobil robotlar için tercih edilmez; çünkü GNSS sinyalleri binalar, ağaçlar ve diğer engeller tarafından engellenebilir [5]. Bu nedenle, bu tür ortamlarda çalışan mobil robotlar, iç mekan (indoor) konum tahmin algoritmalarına ihtiyaç duymaktadır.

Konumlama diğer bir tabirle lokalizasyon, robotun konumunu tahmin etme problemidir. Konum vektörü, iki boyutlu konumlama için x-y konumlarını ve yönelimi içerir. Konumlama sorunu, bilinen bir çevre haritası, algılayıcı ölçümleri ve odometri verileri kullanılarak ele alınır [6]. Eğer robot, konumunu doğru bir şekilde tahmin edemezse seyir halindeyken kullanılan diğer algoritmalar görevlerini düzgün bir şekilde yerine getiremez. Bu durum, konumlama algoritmalarının otonomiyi sağlayan diğer algoritmalar

arasında ne kadar önemli bir yere sahip olduğunu vurgular. Tam bir konumlama, üç temel alt problemi içerir. Bunlar: kendi konumunu takip etme (veya lokal lokalizasyon), küresel konumlama (veya global lokalizasyon) ve anlık konum değişimi anlamına da gelen kaçırılmış robot problemleridir [2][6][7]. Lokal lokalizasyon probleminde robot, başlangıç konumunu $\mathbf{x_0}$ bilmektedir; ancak hedef noktasına gidinceye kadar konum hesabında kullanacağı kontrol işaretleri $\mathbf{u_t}$ gürültülü veya hatalı olabileceğinden bir konum tahmin problemi oluşur. Bu problemden kaynaklanan belirsizlikler unimodal bir dağılıma sahiptir ve problemin çözümü nispeten daha kolaydır. Global lokalizasyonda ise robot başlangıç konumunu **bilmemektedir.** Robot, bilenen bir harita üzerinde üzerine önceden monte edilmiş algılayıcıları tarafından alınan ölçümleri kullanarak konumunu tahmin etmeye çalışmaktadır. Bu süreç lokal lokalizasyona göre çok daha zordur ve bu probleminden dolayı oluşan belirsizlikler multimodal yapıya sahiptir. Kaçırılmış robot problemi ise robotun konumunun anlık değişimi sonucunda meydana gelir. Bu değişim algılandığında robot için konum tahmini, artık bilinmeyen bir rastlantısal süreçtir ve problem global lokalizasyon problemine dönüşmüştür. Yani robot kendisini her kaçırılmış gibi hissettiğinde hemen arkasından problemin çözümü bir global lokalizasyon algoritması ile yapılmaktadır.

Kalman filtre tabanlı konumlama algoritmaları unimodal Gaussian dağılıma sahip olup yalnızca lokal lokalizasyon veya kendi konumunun takip sorununu çözebilirken, parçacık filtre tabanlı algoritmalar hem lokal lokalizasyon hem de global lokalizasyon problemlerini çözebilir [6][7]. Parçacık filtre tabanlı algoritmalar, kaçırılan robotu tespit eden yöntemleri ile birlikte kullanıldığında, lokalizasyonun tüm üç alt problemini aşabilir. Literatürde yer alan, belki de parçacık filtre tabanlı en popüler lokalizasyon algoritması Monte Carlo Lokalizasyon (MCL) algoritmasıdır [6][8][9]. MCL algoritması, mutlimodal yapıdaki rastgele dağılıma sahip konumlama problemlerini çözmek için kullanılır. Yani global ve lokal lokalizasyon problemlerini çözmek için tasarlanmıştır. Lokalizasyon problemini çözerken, bilinen bir haritaya her biri robotu temsil eden parçacıklar atar ve robottan gelen ölçümleri kullanarak her bir parçacığı ağırlıklandırır. Algoritma içindeki yeniden örnekleme (resampling) algoritması ile bu ağırlıklara göre parçacıkların konumu yeniden düzenlenir. Ağrılığı düşük olan parçacıklar elenirken, ağırlığı yüksek olan parçacıklar hayatta kalır. Elenen parçacık sayısı kadar parçacık ağırlığı yüksek olan parçacıkların etrafına atanır. Böylece her adımda parçacıklar robotun etrafında toplanmaya başlar ve belli bir süre sonra parçacıklar robotun konumunu tahmin eder. Geniş boyuta sahip haritalarda lokalizasyon problemini çözmek için çok fazla parçacığa ihtiyaç duyulur. Çevrimiçi çalışırken, her bir parçacığın haritadan ölçüm alma sürecinin belirli bir işlem yükü vardır. Parçacık sayısı arttıkça doğal olarak bu işlem yükü de artmaktadır. Bu işlem yükünü hafifletmek için literatürde bazı lokalizasyon algoritmaları bulunmaktadır. Bunların arasında en popüler olanlarından biri Uyarlamalı (Adaptif) MCL algoritmasıdır [10]. Lokalizasyonun başında çok fazla parçacık kullanılması gerektiğinde, AMCL sayesinde konum belirsizliğine bağlı olarak parçacık sayısı azaltılabilir veya sonradan tekrar artırılabilir. Bu yöntem ile zamanla parçacık sayısı azaltılabilir ve işlem yükü hafifletilebilir. MCL'deki işlem yükünü hafifletmeye yönelik tasarlanan bir diğer lokalizasyon algoritması ise Kendinden Uyarlamalı (Self-Adaptive) MCL algoritmasıdır [2][6][7].

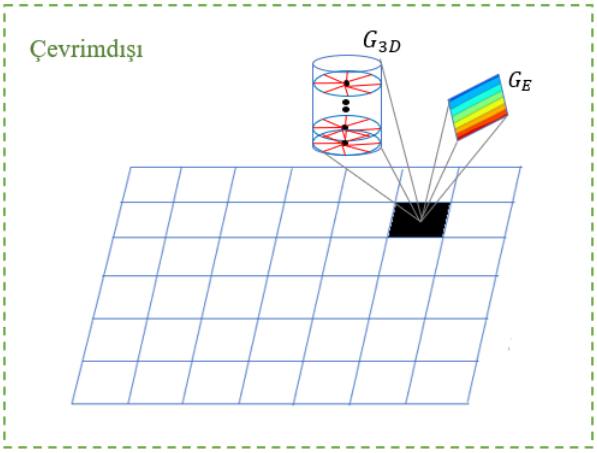
Bu çalışmada Kendinden Uyarlamalı (KU) MCL algoritmasının yapısı ve çalışma prensibi incelenmiş, global lokalizasyonun başında parçacıkların haritaya akıllı bir şekilde atanması için kullanılan benzer enerji bölgelerinin optimal olarak belirlenmesine yönelik bir eşik değeri hesabı önerilmiştir. Bunun yanında KU-MCL ile standart MCL algoritmalarının hibirt olarak kullanılması önerilmiş ve bu hibrit olarak çalışan lokalizasyon algoritmasının performansının, KU-MCL algoritmasından daha iyi olduğu hem simülasyon hem de gerçek dünya deneyleri ile gösterilmiştir.

Bölüm 2'de KU-MCL algoritmasının yapısından ve çalışma prensibinden bahsedilmiş, Bölüm 3'te KU-MCL ile standart MCL'nin önerilen hibrit modeline değinilmiştir. Bölüm 4'te doğrulama için simülasyon ve gerçek ortamda yapılan deney sonuçları verilmiş, Bölüm 5'te ise yapılan çalışmaların sonuçlarının kısa bir değerlendirmesi yapılmış ve gelecekte yapılacak çalışmalardan bahsedilmiştir.

2. Kendinden Uyarlamalı Monte Carlo Lokalizasyon Algoritması

Kendinden Uyarlamalı (KU) MCL algoritması, temelinde diğer MCL algoritmaları gibi bir global lokalizasyon algoritmasıdır. Algoritmanın içindeki robotun kaçırıldığını başka bir deyişle robotun anlık konum değişimini tespit eden mekanizmalar sayesinde kaçırılmış robot problemini de çözebilmektedir. KU-MCL yönteminin sunulmasındaki nedenlerin başında standart MCL algoritmasındaki parçacıkların, haritadan ölçümleri algoritma çevrimiçi çalışırken almaları ve global lokalizasyonun başında parçacıkların haritanın tamamına atanmaları gelmektedir [6]. Birincisi, parçacıkların haritadan ölçümleri algoritma çevrimiçi çalışırken almaları sırasında bir iş yükü oluşmaktadır. Geniş boyutlara sahip haritalarda çalışıldığında çok sayıda parçacığa ihtiyaç duyulur. Tüm parçacıklar düşünüldüğünde toplamda çok fazla iş yükü oluşmakta ve algoritma yavaş çalışmaktadır. Algoritmanın yavaş çalışmasından dolayı bazı anlarda algoritmanın pozisyon tahmini robotun gerçek konumunun gerisinde kalabilmekte bu da anlık pozisyon hatalarına neden olabilmektedir. Bu sorunu çözmek için öncelikle çevrimdışı çalışırken harita ön belleğe alınır ve Şekil 1'deki gibi harita, n×m boyutlarında ızgara (grid) hücrelerine ayrılır. Her bir grid hücresinin merkezine tek düze (uniform) ve belli çözünürlükte 360˚ ölçüm alabilecek şekilde algılayıcılar atanır. Bu algılayıcıların yönelim açıları da belli bir açı çözünürlüğüne göre değişir ve 0˚'den 360˚'ye kadar belli açı adımları ile yönelim açıları artar; o yönelim açısında haritadan ölçümler alınıp üç boyutlu ölçüm deposuna G_{3D} kaydedilir [6]. Çevrimiçi çalışırken parçacıklar, her güncelleme anında etraftan ölçüm almak yerine kendilerine en yakın ızgara hücresinin daha önce depolanmış ölçümlerini kullanırlar. Bu sayede algoritma muazzam hızlı çalışabilmektedir. İkinci durumda ise KU-MCL algoritmasında, standart MCL algoritmasında olduğu gibi lokalizasyonun başlangıç anında parçacıklar haritanın tamamına **atanmamaktadır.** Yine Şekil 1'de görüleceği üzere harita ön belleğe alındığında ve algoritma çevrimdışı çalışırken her bir ızgara hücresindeki algılayıcıların 360˚ aldığı tek bir ölçüm dizisi kullanılarak o ızgara hücresinin enerjisi hesaplanır ve enerji deposuna G_E kaydedilir [2][6]. Yani enerji hesabı yönelimden bağımsız yapılır. Orijinal yayında [6] parçacıkların haritanın tamamına atanması yerine enerji bilgileri kullanılarak haritanın belirli bölgelerine daha akıllı bir şekilde atanması önerilmiştir. Bu sayede başlangıç

anında robotun konumunun daha hızlı ve doğru bir şekilde tahmin edilmesi amaçlanmıştır.



Şekil 1: Harita ön belleğe alındığında yapılan işlemler.

Tüm ızgara hücrelerinin enerjileri aşağıdaki denklem ile hesaplanır ve hesaplanan bu enerji skaler bir büyüklüktür [2].

$$e_k^i = 1 - \frac{a_k^i}{a_{max}} \quad (1)$$

Yukarıdaki (1) denkleminde a_k^i, *k.* ızgara hücresindeki algılayıcının *i.* ölçüm değeridir. e_k^i, *k.* ızgara hücresindeki algılayıcının *i.* ölçümünün enerjisidir. a_{max} ise algılayıcının ölçebileceği maksimum değerdir. Enerji değerleri [0, 1] aralığında olup algılayıcının aldığı maksimumum ölçüm değerinin enerjisi 0 iken, çok küçük ölçümlerin enerjisi 1'e yakındır. Bu enerji kavramı aslında ızgara hücresinin haritadaki nesnelere yakınlığı ve uzaklığı ile ilgilidir. Eğer ızgara hücresi haritadaki nesnelere veya duvarlara çok yakınsa yani dar bir alandaysa enerjisi yüksek, geniş bir alanda ise enerjisi düşüktür. Bir ızgara hücresinin enerjisi, aşağıdaki denklemde de görüldüğü üzere algılayıcının tüm ölçüm değerleri için hesaplanan enerjilerin ortalamasıdır.

$$\mathbf{e}_k = \frac{1}{I}\sum_{i=1}^{I} e_k^i \quad (2)$$

(2) denkleminde yer alan $\mathbf{e}_k$ değeri, *k.* ızgara hücresinin normalize enerji değeridir. Benzer enerji bölgelerinin belirlenebilmesi için robotun çevrimiçi çalışırken etraftan aldığı ölçümlerin yine (1) denklemine göre enerjisi bulunup (2)'ye göre de bu ölçüm değerlerine göre hesaplanan enerjilerin ortalaması alınarak robotun bulunduğu konumdaki enerjisi hesaplanır. Aşağıdaki eşitsizlik kullanılarak robotun enerjisi, haritadaki tüm ızgara hücrelerinin enerjileri ile karşılaştırılır. Enerjiler arasındaki fark belli bir eşik değerinin altında ise o ızgara hücresinin enerjisi robotun enerjisine çok yakındır ve o ızgara hücresi benzer enerji bölgesi olarak kabul edilir [2][6].

$$|\mathbf{e}_R - \mathbf{e}_k| < \delta \quad (3)$$

(3) denkleminde yer alan $\mathbf{e}_R$ robotun enerjisi, δ ise benzer enerji bölgelerinin belirlenmesi için kullanılan eşik değeridir. [6]'da bu eşik değerinin öneminden bahsedilse de neye göre ve nasıl belirleneceği hakkında detaylı bir açıklama yapılmamıştır.

3. Kendinden Uyarlamalı MCL Algoritması ile Standart MCL Algoritmasının Hibrit Çalışması

Bu bölümde KU-MCL algoritmasında yer alan benzer enerji bölgelerinin (BEB) optimal olarak belirlenebilmesi için bir yöntem önerilmiş ve eşik değeri bu yönteme göre hesaplanmıştır. Robotun konumu KU-MCL ile tespit edildikten belli bir adım sonra daha az parçacık ile çalışan standart MCL algoritmasına geçilerek yapılan konum tahminin daha iyi performansa sahip olduğu gösterilmiştir.

3.1. BEB'lerin Optimal Olarak Belirlenebilmesi İçin Önerilen Eşik Değeri

Haritada üzerinde tanımlanmış *k.* ızgara hücresinin merkezinde olan robotun ***t*** anında çevreden aldığı gürültülü ölçümlerin $\boldsymbol{z_t}$, robotun bulunduğu ızgara hücresinin merkezinden robotun yönelim açısına en yakın açıyla önceden alınmış ve depolanmış ölçümler cinsinden bir modeli, aşağıdaki gibi oluşturulabilir.

$$z_t^i = a_k^i + \varepsilon_k^i \quad (4)$$

Yukarıdaki (4) denkleminde yer alan ε_k^i sıfır ortalamalı Gaussian ölçüm hatasını temsil etmektedir. Bu modele göre robotun normalize enerjisi aşağıdaki gibi hesaplanabilir.

$$\mathbf{e}_R = 1 - \frac{1}{a_{max}}\frac{1}{I}\left(\sum_{i=1}^{I} a_k^i + \sum_{i=1}^{I} \varepsilon_k^i\right) \quad (5)$$

(5) denkleminde yer alan ε_k^i sıfır ortalamalı bir ölçüm hatası olarak kabul edildiği için etraftan yeterli sayıda ölçüm alındığında $\frac{1}{I}\sum_{i=1}^{I}\varepsilon_k^i$ ifadesinin sıfıra yaklaşacağı kabul edilmektedir. Robotun, merkezi üzerinde olduğu ızgara hücresine en yakın ızgara hücresinin merkezleri arasındaki uzaklık, ızgara hücreleri kare şeklinde ve hepsi eşit boyutlarda ise ızgara hücresinin bir kenarının uzunluğu kadardır. Yani robotun *i.* ölçümünün uzunluğu ile en yakın ızgara hücresinin *i.* ölçümü arasındaki farkın mutlak değeri $|\Delta a + \varepsilon_k^i|$ kadar olabilir. Δa, kare ızgara hücresinin bir kenar uzunluğunu temsil etmektedir. Robotun enerjisi ile robota en yakın ızgara hücrelerinin enerjileri $\mathbf{e}_k^*$ arasındaki fark, aşağıdaki denklemdeki gibi hesaplanabilir.

$$\mathbf{e}_R - \mathbf{e}_k^* = -\frac{1}{I}\sum_{i=1}^{I}\frac{\Delta a}{a_{max}} = -\frac{\Delta a}{a_{max}} \quad (6)$$

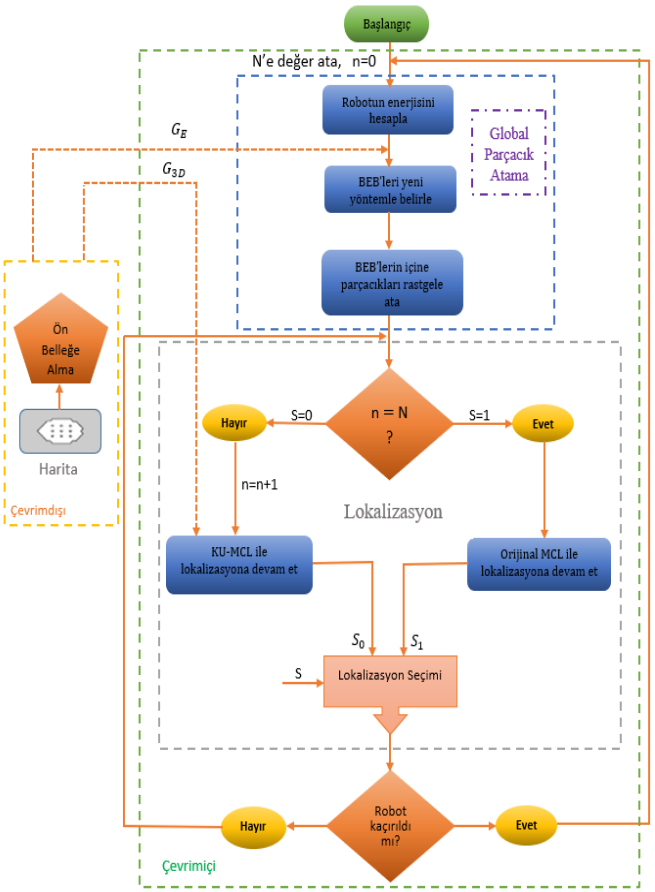
(6) denklemindeki sonuca göre robotun enerjisine en yakın ızgara hücrelerinin optimal bir şekilde belirlenebilmesi için bir eşik değeri belirlenmiş olup bu değer denklem (7)'de verilmiştir.

$$\delta = \left| - \frac{\Delta a}{a_{max}} \right| \qquad (7)$$

BEB'lerin optimal olarak belirlenmesi, başlangıç anında robotun konumunun tahmin olasılığını artırmak için önemlidir; çünkü robotun enerjisine benzeyen bölgelerin sayısı ne kadar optimal belirlenirse belirli sayıda BEB'lere rastlantısal olarak uniform dağılımla atanan parçacıkların robotun bulunduğu bölgelere düşme olasılığı o kadar artar. Robotun bulunduğu bölgede olmayan fakat haritanın farklı bölgelerindeki robotun enerjisine yakın enerji bölgeleri, aslında fazlalıktan ibaret olup robotun konumunun tahmin edilme olasılığını düşürmektedirler.

3.2. MCL Tabanlı Algoritmaların Hibrit Çalışması

KU-MCL ile standart MCL'nin birbirlerine göre avantajlarının olduğu kısımlarda, bu algoritmaları devreye sokmak amacıyla bir hibit lokalizasyon yöntemi önerilmiştir. KU-MCL, BEB'leri kullanarak global lokalizasyonun başında robotun konumunu MCL'ye göre çok daha hızlı ve doğru bir şekilde

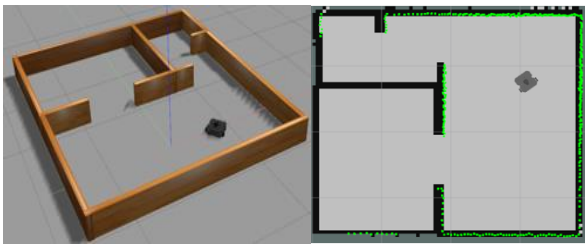


Şekil 2: Önerilen KU-MCL + MCL hibirt algoritmanın akış diyagramı.

tahmin edebilmektedir. Ancak, KU-MCL algoritması ile konum tahmini yapılmaya devam edildiğinde parçacıklar ölçümleri kendi bulundukları konumdan değil de kendilerine en yakın ızgara hücresinin önceden depolanmış bölümlerinden aldıklarından, lokalizasyon boyunca bazı hatalar oluşabilmektedir. Standart MCL'de ise parçacıklar ölçümlerini harita üzerinde bulundukları konumdan alırlar; fakat başlangıçta çok fazla parçacıkla lokalizasyona başlanmak zorunda kalındığından, algoritmanın işlem süresi uzamakta ve bu olumsuz durum anlık konumlama hatalarına neden olmaktadır. Şekil 2'deki akış diyagramında da görüleceği üzere önerilen hibrit algoritmada global lokalizasyonun başlangıç kısmı KU-MCL tarafından yönetilir. Belli bir *n* adım geçtikten sonra parçacıkların robot etrafında toplanıp robotun konumunu tahmin ettiği farz edilir ve bundan sonra daha az parçacıkla çalışan MCL algoritması devreye girer. Bu şekilde yapılacak olan bir lokalizasyon sürecinin sadece KU-MCL ile yapılacak lokalizasyon sürecinden daha doğru ve güvenilir olacağı ileri sürülmektedir ve bu önerinin geçerliliğini doğrulamak için de birtakım testler yapılmıştır.

4. Deneysel Analiz

Önerilen KU-MCL+MCL hibrit lokalizasyon algoritmasının geçerliliğini doğrulamak, daha doğru ve güvenilir konum tahminleri yapabildiğini göstermek için hem simülasyon ortamında hem de gerçek ortamda birtakım testler yapılmıştır. Öncelikle simülasyon testleri için Şekil 3'teki gibi ROS (Robot Operating System) [11] ortamında bir test platformu oluşturulmuştur ve lokalizasyon algoritmasının ihtiyaç duyduğu harita bilgilerini sağlamak için ROS ortamındaki hazır GMapping paketi ile ortamın haritası çıkarılmıştır.



Şekil 3: ROS Gazebo ortamında oluşturulan test platformu (sol) ve bu platformun çıkarılan haritası (sağ).

Şekil 4'te ise ROS Gazebo ortamında oluşturulan platforma benzer bir deney platformu gerçek ortamda oluşturulmuştur ve bu platformun ROS ortamında haritası çıkarılıp MATLAB ortamına aktarılmıştır.

Şekil 4: Gerçek ortamda oluşturulan deney platformu (sol) ve bu platformun çıkarılan haritası (sağ).

Şekil 3'te verilen platform üzerinde Gazebo ortamında URDF modeli (Universal Robot Description Format) olan Turtlebot3 waflle_pi [12] mobil robotu ile global lokalizasyon testleri

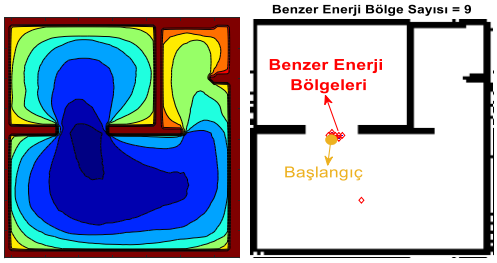
yapıldıktan sonra Şekil 4'te verilen gerçek ortamda oluşturulmuş platform üzerinde bu sefer gerçek Turtlebot3 waffle_pi robotu kullanılarak gerçek dünya deneyleri yapılmıştır. Gerçek ortamda oluşturulan platformun boyutları yaklaşık olarak 4.2m × 3.5m olup bu ortamda çıkarılan haritanın hataları simülasyon ortamında çıkarılan haritaya göre daha fazladır. Gerçek ortamda yapılan deneylerde mobil robot tarafından alınan ölçümler simülasyon ortamındaki kadar mükemmel olmadığı için hataların ve gürültülerin seviyesine göre bozulmalar da artmaktadır. Aslında bu olay gerçek dünya deneylerinin ne kadar önemli olduğunu göstermektedir. Simülasyon ve gerçek dünya deneylerinde kullanılan bilgisayarın ve programların özellikleri Tablo 1'de verilmiştir.

Tablo 1: Deneyde kullanılan yazılım ve donanımın özellikleri

Özellikler
RAM: 16 GB DDR4 2400 SSD: Intel 128GB HDD: 1TB CPU: Intel i7-8750H 2.2-4.1 GHz GPU: Intel UHD Graphics 630 OS: Ubuntu 16.04 Programlar: MATLAB® 2021a, ROS kinetic, Python

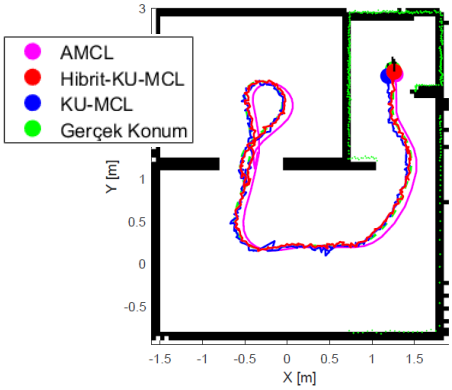
4.1. ROS Ortamında Yapılan Simülasyon Sonuçları

Önerilen lokalizasyon algoritmasının doğruluğunu test etmek için öncelikle Şekil 3'te de görüldüğü gibi bir test platformu oluşturulmuş ve ROS ortamında tanımlanmıştır. Turtlebot3 waffle_pi mobil robotu ile önce ortamın haritası çıkarılmış sonra bu harita kullanılarak 360˚ RPLiDAR ölçümleri ile global lokalizasyon testleri yapılmıştır. Yapılan lokalizasyon simülasyonları baştan sona ***"bagfile"*** olarak kaydedilmiştir ve MATLAB ortamındaki ROS araç kutusu sayesinde bagfile olarak kaydedilen verilere erişilebilmektedir. Bu veriler kullanılarak önerilen lokalizasyon algoritması, MATLAB ortamında test edilmiştir. Algoritma çevrimdışı çalışırken Şekil 5'teki gibi harita ızgara hücrelere ayrılmış, her bir ızgara hücresine ölçümler ve enerji bilgileri kaydedilmiş, enerji kontur grafikleri oluşturulmuş ve enerji dağılımları gözlenmiştir. Izgara hücrelerin boyutları 0.05m×0.05m olup robotun ölçebileceği maksimum mesafe değeri a_{max} = 3.5m olarak ayarlanmıştır. Daha sonra yayında önerilen **eşik değeri** kullanılarak benzer enerji bölgeleri oluşturulmuştur.



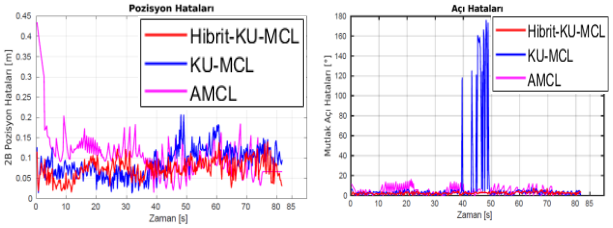
Şekil 5: Haritanın enerji kontur grafiği (sol), benzer enerji bölgelerinin dağılımı (sağ).

Şekil 5'teki benzer enerji bölgelerinin dağılımını gösteren haritaya bakıldığında, önerilen eşik değeriyle BEB'ler hesaplanınca, sadece 9 BEB hücresi ile robotun başlangıç konumunun doğru tahmin edildiği görülmektedir. ROS ortamında toplanan ölçüm verileri, kaydedildikleri zaman bilgileri ile beraber MATLAB ortamına aktarılmış, Şekil 6'da görüldüğü gibi algoritmaların global lokalizasyon testleri MATLAB ortamında yapılmış ve tüm deneylerde KU-MCL için 5000, MCL için 50 parçacık kullanılmıştır.



Şekil 6: Global lokalizasyon algoritmalarının performansları.

Şekil 7'de lokalizasyon testinin performans grafikleri verilmiştir.



Şekil 7: Algoritmaların 2B pozisyon hata grafikleri (sol), mutlak açı hata grafikleri (sağ).

Tablo 2'de algoritmaların performans sonuçları verilmiştir.

Tablo 2: Algoritmaların performans sonuçları

Algoritmalar	**Ortalama 2B Konumlama Hatası (m)**	**Standart Sapmalar (m)**	**Ortalama Mutlak Açı Hatası (°)**
Hibrit-KU-MCL	0.0727	0.0244	2.4470
KU-MCL	0.0889	0.0353	9.4664
AMCL	0.1067	0.0587	4.4616

Tablo 2'ye göre Hibrit-KU-MCL algoritması KU-MCL ile karşılaştırıldığında, 2B konumlama ve mutlak açının ortalama hataları sırasıyla yaklaşık 18.22% ve 74.1507% oranında azaltılmıştır. Hibrit-KU-MCL algoritması (Adaptif) MCL ile karşılaştırıldığında, 2B konumlama ve mutlak açının ortalama hataları sırasıyla yaklaşık 31.865% ve 45.1542% oranında azaltılmıştır. Global lokalizasyonun başında Hibrit-KU-MCL ve KU-MCL algoritmaları robotun konumunu tek adımda tahmin ederken AMCL algoritması birkaç adım sonra ancak robotun konumunu yaklaşık olarak tahmin edebilmiştir. Lokalizasyon süreci boyunca Hibrit-KU-MCL algoritması 2 defa robotu kaçırılmış gibi hissederken, KU-MCL algoritması 11 defa kaçırılmış gibi hissetmiştir. Ancak robot, lokalizasyon

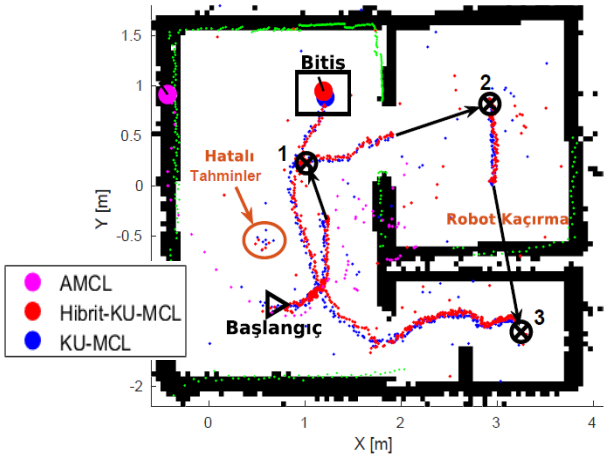
boyunca hiç kaçırılmamıştır. Bu durum Hibrit-KU-MCL algoritmasının daha doğru tahminler yaptığını göstermektedir.

4.2. Gerçek Ortamda Yapılan Deney Sonuçları

ROS ortamında simülasyon testleri yapıldıktan sonra Şekil 8'deki gibi bir deney ortamı kurulmuş, algoritmaların global lokalizasyon problemlerini çözme hızlarını ve doğruluklarını test etmek için robot, haritadaki belirli bölgelerden 3 farklı noktaya kaçırılmıştır.

Şekil 8: Robotun kaçırılma anı.

Şekil 9'da ise robot kaçırma deneyinde toplanan ölçüm verileri kullanılarak MATLAB ortamında yapılan performans testleri görülmektedir.



Şekil 9: Algoritmaların robot kaçırma deneyindeki performansları.

Deneyin başlangıcından bitişine kadar geçen sürede toplamda Hibrit-KU-MCL algoritması 56 kez robotun kaçırıldığını algılarken, KU-MCL algoritması 61 kez algılamıştır. Algoritmaların ortalama 45 kez yanlış kaçırılma algılamaları, kaçırılma anında robotun hatalı ölçümler almasından kaynaklanmıştır. Kaçırılma anında alınan yanlış ölçümlerden kaynaklanan hatalı algılamalar çıkarıldığında en iyi ihtimalle robotun 3 kez kaçırıldığının algılanması beklenebilir. Sonuç olarak Hibrit-KU-MCL algoritması, robotun gerçek konumuna daha hızlı yakınsayıp daha başarılı konum tahmini yaparken, AMCL algoritması ise robotun kaçırıldığı noktaların hiçbirini doğru tahmin edememiştir.

5. Sonuçlar

Bu çalışmada orijinal KU-MCL algoritmasındaki BEB'lerin optimal olarak belirlenebilmesi böylece başlangıç anında robotun konumunun daha hızlı ve doğru bir şekilde tahmin edilebilmesi için yeni bir eşik değeri hesabı önerilmiştir. Buna ilaveten, standart MCL algoritması ile KU-MCL algoritmasının hibrit bir şekilde çalıştırılması sonucunda daha doğru lokalizasyon tahminlerinin yapılabileceği deneylerle gösterilmiştir. Gelecekte ise önerilen bu algoritmanın farklı otonom araçlar üzerinde denenmesi amaçlanmaktadır.

Teşekkür

Bu çalışma TÜBİTAK'ın 121E537 nolu projesi tarafından desteklenmektedir.

Kaynakça

[1] F. O. Coelho, J. P. Carvalho, M. F. Pinto, and A. L. Marcato, "Ekf and computer vision for mobile robot localization," in *2018 13th APCA International Conference on Automatic Control and Soft Computing (CONTROLO)*, 2018, pp. 148–153.

[2] A. Yılmaz and H. Temeltaş, "An Improvement on SA-MCL Algorithm: Ellipse Based Energy Grids," in *2018 6th International Conference on Control Engineering & Information Technology (CEIT)*, 2018, pp. 1–6.

[3] Y. Wang *et al.*, "An improved adaptive Monte Carlo localization algorithm fused with ultra wideband sensor," in *2019 IEEE International Conference on Advanced Robotics and its Social Impacts (ARSO)*, 2019, pp. 421–426.

[4] O. V. Altinpinar, E. C. Contarli, A. Kağizman, U. Uguzlar, E. Cansu, and V. Sezer, "Comparison of Autonomous Robot's Mapping Performance Based on Number of Lidars And Number of Tours," in *2022 Innovations in Intelligent Systems and Applications Conference (ASYU)*, 2022, pp. 1–6.

[5] F. Gu, S. Valaee, K. Khoshelham, J. Shang, and R. Zhang, "Landmark graph-based indoor localization," *IEEE Internet Things J.*, vol. 7, no. 9, pp. 8343–8355, 2020.

[6] L. Zhang, R. Zapata, and P. Lepinay, "Self-adaptive Monte Carlo localization for mobile robots using range finders," *Robotica*, vol. 30, no. 2, pp. 229–244, 2012.

[7] A. W. Li and G. S. Bastos, "A hybrid self-adaptive particle filter through KLD-sampling and SAMCL," in *2017 18th International Conference on Advanced Robotics (ICAR)*, 2017, pp. 106–111.

[8] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte carlo localization for mobile robots," in *Proceedings 1999 IEEE international conference on robotics and automation (Cat. No. 99CH36288C)*, 1999, vol. 2, pp. 1322–1328.

[9] S. Thrun, W. Burgard, and D. Fox, "Probalistic Robotics," *Kybernetes*, vol. 35, no. 7/8, pp. 1299–1300, Jan. 2006, doi: 10.1108/03684920610675292.

[10] D. Fox, "Kld-sampling: Adaptive particle filters and mobile robot localization," *Adv. Neural Inf. Process. Syst.*, vol. 151, p. 152, 2002

[11] M. Quigley *et al.*, "ROS: an open-source Robot Operating System," in *ICRA workshop on open source software*, 2009, vol. 3, no. 3.2, p. 5.

[12] R. Amsters and P. Slaets, "Turtlebot 3 as a robotics education platform," in *Robotics in Education: Current Research and Innovations 10*, 2020, pp. 170–181.