



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

Ozan Vahit Altınpınar¹ , Volkan Sezer² 

- ¹ Kontrol ve Otomasyon Mühendisliği Bölümü, Akıllı ve Otonom Sistemler Laboratuvarı (Smart and Autonomous System Laboratory, SASLab), İstanbul Teknik Üniversitesi, İstanbul, Türkiye
² Kontrol ve Otomasyon Mühendisliği Bölümü, Akıllı ve Otonom Sistemler Laboratuvarı (Smart and Autonomous System Laboratory, SASLab), İstanbul Teknik Üniversitesi, İstanbul, Türkiye

Özet: 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 lokalizasyon algoritmaları 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.

Anahtar Kelimeler: Otonom robotlar, Lokalizasyon, KU-MCL, Standart MCL

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

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.

Keywords: Autonomous robots, Localization, SA-MCL, Standard MCL

RESEARCH PAPER

Corresponding Author: Ozan Vahit Altınpınar, altinpinaro@itu.edu.tr

Reference: O. V. Altınpınar, *et al.*, (2024), Design and Implementation of a New Hybrid Localization Algorithm Based on SA-MCL for Autonomous Robots, *ITU-Journ. Computer Science, AI and Robotics.*, 6-15.

Submission Date: January, 20, 2024

Acceptance Date: March, 23, 2024

Online Publishing: July, 20, 2024

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 bakımının yapılmasına yardımcı olmak 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 alt 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.

Konum belirleme 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 otonomiye sağlayan diğer algoritmalar arasında ne kadar önemli bir yere sahip olduğunu vurgular. Tam bir konumlama, üç temel alt problemi içerir ([7]). 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], [8]). Lokal lokalizasyon probleminde robot, başlangıç konumunu x_0 bilmektedir; ancak hedef noktasına gidinceye kadar konum hesabında kullanacağı kontrol işaretleri u_t gürültülü veya hatalı olabileceğinden bir konum tahmin problemi oluşur. Bu problemde kaynaklanan belirsizlikler unimodal bir dağılıma sahiptir ve problemin çözümü nispeten daha kolaydır ([7]). Global lokalizasyonda ise robot, başlangıç konumunu bilmemektedir. Robot, bilinen 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 problemde 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], [8]). Parçacık filtre tabanlı algoritmalar, kaçırılan robotu tespit eden yöntemler 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], [7], [9], [10]). MCL algoritması, multimodal 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. Ağırlığı düşük olan parçacıklar elenirken, ağırlığı yüksek olan parçacıklar hayatta kalır. Elenen parçacık sayısı kadar parçacık, ağırlığı yüksek olan parçacıkların bulunduğu yerlere atanır. Böylece her adımda parçacıklar, robotun bulunduğu konuma doğru daha çok yakınsar 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 (Adaptive) MCL (AMCL) algoritmasıdır ([11]). 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], [8]). Bu çalışmada Kendinden Uyarlamalı (KU) MCL algoritmasının yapısı ve çalışma prensibi incelenmiş, global lokalizasyon için aşağıdaki katkılar sunulmuştur:

- Global lokalizasyonun başında parçacıkların haritaya akıllı bir şekilde atanması için kullanılan benzer enerji bölgelerinin optimal olarak belirlenmesine yönelik bir eşik değeri hesabı önerilmiştir.
- KU-MCL ile standart MCL algoritmalarının hibirt bir şekilde kullanılması önerilmiştir.

Önerilen hibrit lokalizasyon algoritmasının performansının, orijinal 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 verilmiştir. Bölüm 5’te ise yapılan çalışmaların sonuçlarının kısa bir değerlendirmesi yapılmış ve gelecekte yapılacak çalışmalardan bahsedilmiştir.

2 Kendinden uyarlamalı Monte Carlo lokalizasyon algoritması (KU-MCL)

Kendinden Uyarlamalı (KU) MCL algoritması, temelinde diğer MCL algoritmaları gibi bir global lokalizasyon algoritmasıdır. Algoritmanın içindeki robotun kaçırıldığı başka bir deyişle robotun anlık konum değişimini tespit eden mekanizmalar sayesinde kaçırılmış robot problemi de çözülebilmektedir. Standart MCL, Bayes tabanlı bir Markov lokalizasyon algoritmasıdır. Bayes filtre tekniği, robot lokalizasyon problemlerini anlamak ve çözmek için güçlü bir istatistiksel araç sağlar ([6]). Ölçüm ve kontrol verilerinden inanç (belief) dağılımını $bel(*)$, yinelemeli olarak hesaplar. Bayes filtresi bir Markov varsayımı yapar; yani eğer mevcut durum biliniyorsa geçmiş ve gelecek veriler bağımsızdır. $bel(s_t)$, robotun t zamanında s_t konumunda olduğuna dair öznel inancını gösterir. Burada $s_t = (x_t, y_t, \theta_t)^T$, Kartezyen koordinat sistemindeki x-y koordinatlarını ve θ yönelimini içeren 3B bir değişkendir ([6]). İnanç dağılımı, t zamanındaki s_t durumu üzerindeki tüm geçmiş ölçümlere ($z_{0:t}$) ve tüm geçmiş kontrol işaretlerine ($u_{1:t}$) bağlı olan sonsal (posterior) olasılıktır ve denklem (1)’deki gibi ifade edilebilir ([2], [6]).

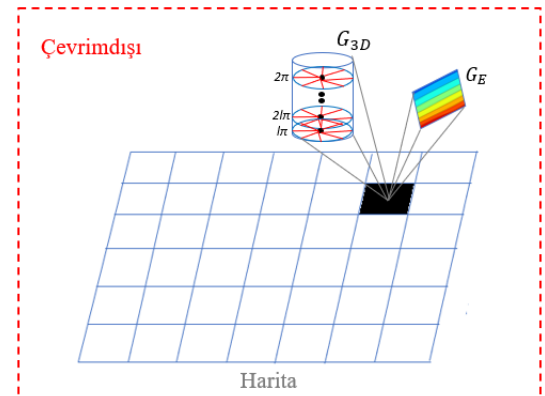
$$bel(s_t) = p(s_t | u_{1:t}, z_{0:t}) \quad (1)$$

Parçacık filtresinde ise s_t ’in olasılık yoğunluğunu veren $bel(s_t)$ inanç fonksiyonu, aşağıdaki gibi ağırlıklı parçacıkları içeren bir küme ile temsil edilebilir ([2]).

$$bel(s_t) \propto S_t = \left\{ s_t^{[m]}, \omega_t^{[m]} \right\}_{m=1, \dots, M} \quad (2)$$

Denklem (2)’deki ifadede $s_t^{[m]}$ her bir parçacığı, M parçacık sayısını, $\omega_t^{[m]}$ her bir parçacığın negatif olmayan önem ağırlığını, S_t ise ağırlıklı parçacıkları içeren kümeyi temsil etmektedir ([6]). Robotun başlangıç konumu s_0 , belirli bir standart sapma ile yaklaşık olarak biliniyorsa ilk inanç dağılımı $bel(s_0)$, Gauss dağılımı ile; bilinmiyorsa rastgele uniform dağılım ile modellenilebilir ([2]). Robotun başlangıç konumu bilinmediğinde, t_0 anında durum uzayına atanan tüm parçacıkların robotun gerçek konumunda bulunma olasılıklarının eşit olduğu kabul edilir ve $\omega_0^{[m]} = 1/M$ olarak

hesaplanır. KU-MCL yönteminin sunulmasındaki nedenlerin başında; standart MCL algoritmasındaki parçacıkların haritadan ölçümleri algoritma çevrimiçi çalışırken almaları, bunun da algoritmanın iş yükünün artmasına sebep olması ve global lokalizasyonun başında parçacıkların haritanın tamamına atanması gelmektedir ([6]). Birincisi, parçacıkların haritadan ölçümleri algoritma çevrimiçi çalışırken almaları sırasında bir iş yükü oluşmaktadır. Geniş boyutlara sahip haritalarda çalışıldığı zaman 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, $n \times m$ boyutlarında ızgara (grid) hücrelerine ayrılır. Her bir ızgara hücresinin merkezine tekdüze (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 çevrimdışı yapılan işlemler.

Şekil 1'deki ölçümler, $l\pi$ yönelim açısından başlayıp yönelim açısı 2π olana kadar alınmaktadır ve $l\pi$, açı aralığını temsil etmektedir. l değeri, bütün uygulamalarda 1/32 olarak alınmıştır. Tüm ızgara hücrelerinin enerjileri aşağıdaki denklem ile hesaplanır ve hesaplanan bu enerji skaler bir büyüklüktür [2].

$$e_k^i = 1 - \frac{a_k^i}{a_{max}} \quad (3)$$

Yukarıdaki denklem (3)'te 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 maksimum 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 alandaysa enerjisi düşüktür. Bir ızgara hücresinin enerjisi, aşağıdaki denklemde de görüldüğü üzere algılayıcının tüm ölçüm değerleri için hesaplanan enerjilerin ortalamasıdır.

$$\Xi_k = \frac{1}{I} \sum_{i=1}^I e_k^i \quad (4)$$

Denklem (4)'te yer alan Ξ_k değeri, k . ızgara hücresinin normalize enerji değeridir. I , değeri ise alınan toplam ölçüm değeridir. Benzer enerji bölgelerinin belirlenebilmesi için robotun çevrimiçi çalışırken etraftan aldığı ölçümlerin her birinin enerjisi yine denklem (3)'tekine benzer şekilde aşağıdaki gibi hesaplanmalıdır.

$$e_R^i = 1 - \frac{z_t^i}{a_{max}} \quad (5)$$

Denklem (5)'te yer alan z_t^i , robotun i . ölçüm değeridir. e_R^i , ise robotun i . ölçümünün enerjisidir. Robotun normalize ya da ortalama enerjisi ise aşağıdaki denklemde verilmiştir.

$$\Xi_R = \frac{1}{I} \sum_{i=1}^I e_R^i \quad (6)$$

Denklem (6)'da yer alan Ξ_R , robotun normalize enerji değerini temsil etmektedir. 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 veya hücresi olarak kabul edilir ([2][6]).

$$|\Xi_R - \Xi_k| < \delta \quad (7)$$

Denklem (7)'de yer alan δ , benzer enerji bölgelerinin belirlenmesi için kullanılan eşik değeridir. ([6])'da bu eşik değerinin öneminden bahsedilse de neye göre ve nasıl

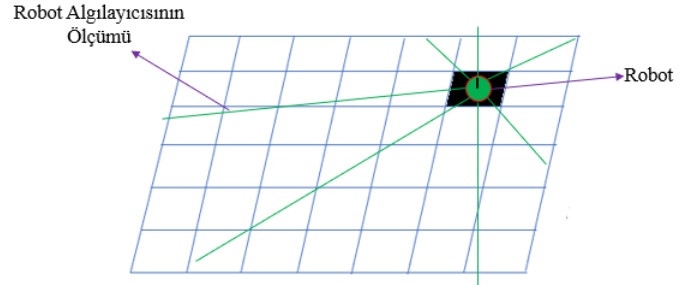
belirleneceği hakkında detaylı bir açıklama yapılmamıştır.

3 Kendinden uyarlamalı MCL algoritması ile standart MCL algoritmasının hibrit olarak çalışması

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

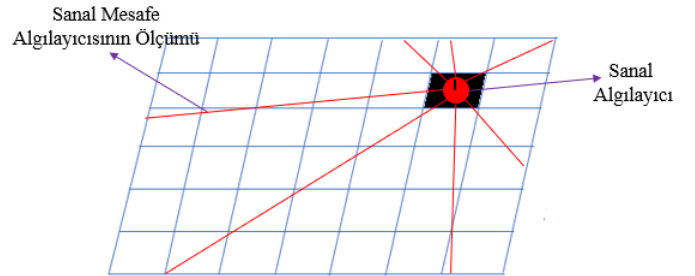
3.1 BEB'lerin optimal olarak belirlenebilmesi için önerilen eşik değeri

Haritada üzerinde tanımlanmış k . ızgara hücresinin merkezinde olan robotun t anında belli bir açıyla çevreden aldığı gürültülü ölçümler z_t , Şekil 2'deki gibi modellenebilir.



Şekil 2 Robot, haritanın k . ızgara hücresinin merkezindeyken t anında θ açısıyla etraftan aldığı gürültülü z_t ölçümler.

Harita önbelleğe alındığında, haritanın k . ızgara hücresinin merkezine yerleştirilmiş 360° ölçüm alabilen sanal bir algılayıcının yine θ açısıyla haritadan aldığı düzgün ölçümler, Şekil 3'teki gibi modellenebilir.



Şekil 3 Haritanın k . ızgara hücresinin merkezindeki sanal algılayıcının θ açısıyla haritadan aldığı ölçümler a_k .

k . ızgara hücresinin merkezinde olan robotun; t anında çevreden aldığı gürültülü ölçümlerin z_t , robotun bulunduğu ızgara hücresinin merkezinden robotun yönelim açısına en yakın açıyla önceden alınmış ve depolanmış ölçümler cinsinden bir denklem modeli, aşağıdaki gibi oluşturulabilir.

$$z_t^i = a_k^i + \epsilon_k^i \quad (8)$$

Denklem (8)'de ϵ_k^i , sıfır ortalamalı Gauss dağılımlı ölçüm

hatasını temsil etmektedir. Bu modele göre robotun normalize enerjisi, denklem (9)'daki gibi hesaplanabilir.

$$\bar{\varepsilon}_R = 1 - \frac{1}{a_{max}} \frac{1}{I} \left(\sum_{i=1}^I a_k^i + \sum_{i=1}^I \varepsilon_k^i \right) \quad (9)$$

Denklem (9)'da yer alan ε_k^i sıfır ortalamalı bir ölçüm hatası olarak kabul edildiği için etraftan yeterli sayıda ölçüm alındığında $\left(\frac{1}{I} \sum_{i=1}^I \varepsilon_k^i \right) \rightarrow 0$ ifadesinin sıfıra yaklaşacağı kabul edilmektedir. Robotun, merkezi üzerinde olduğu ızgara hücrelerine en yakın ızgara hücrelerinin merkezleri arasındaki uzaklık, ızgara hücreleri kare şeklinde ve hepsi eşit boyutlarda ise ızgara hücrelerinin bir kenarının uzunluğu kadardır. Yani robotun i . ölçümünün uzunluğu ile en yakın komşu ızgara hücrelerinin i . ölçümü arasındaki fark $(-\Delta a - |\varepsilon_k^i|, \Delta a + |\varepsilon_k^i|)$ aralığında olabilir. Δa , kare ızgara hücrelerinin bir kenar uzunluğudur. Robotun enerjisi ile robota en yakın ızgara hücrelerinin enerjileri ε_k^* arasındaki fark, aşağıdaki denklemdeki gibi hesaplanabilir.

$$|\bar{\varepsilon}_R - \varepsilon_k^*| < \frac{1}{I} \sum_{i=1}^I \frac{\Delta a}{a_{max}} = \frac{\Delta a}{a_{max}} \quad (10)$$

Denklem (10)'daki sonuca göre robotun enerjisine en yakın ızgara hücrelerinin optimal bir şekilde belirlenebilmesi için bir eşik değeri belirlenmiş olup bu değer aşağıda verilmiştir.

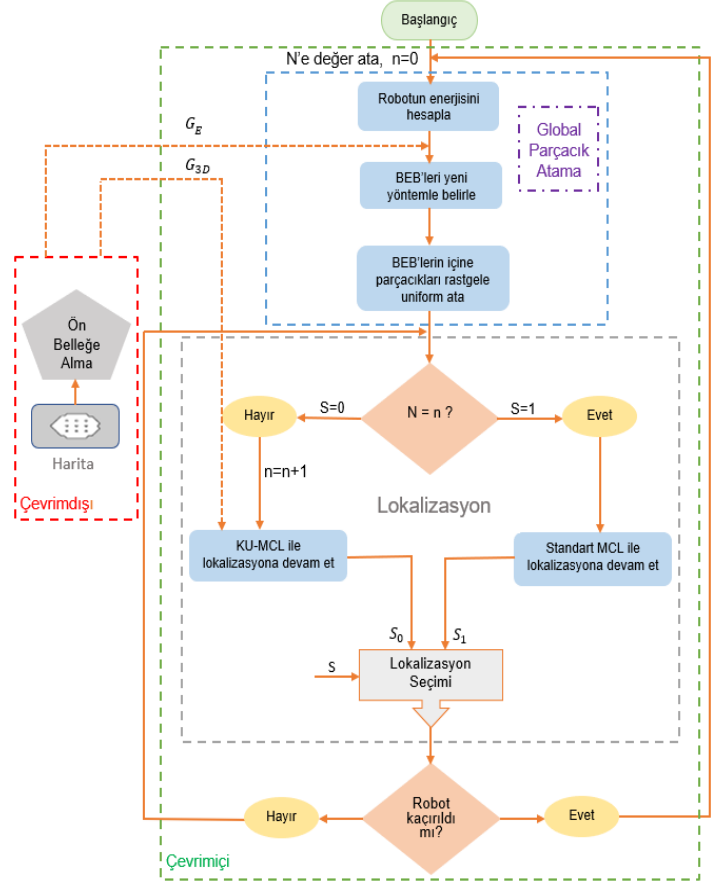
$$\delta = \frac{\Delta a}{a_{max}} \quad (11)$$

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 düşürmektedir.

3.2 MCL tabanlı algoritmaların hibrit olarak çalışması

KU-MCL ile standart MCL'nin birbirlerine göre avantajlarının olduğu kısımlarda, bu algoritmaları devreye sokmak amacıyla bir hibrit lokalizasyon yöntemi önerilmiştir. KU-MCL, BEB'leri kullanarak global lokalizasyonun başında robotun konumunu MCL'ye göre çok daha hızlı ve doğru bir şekilde tahmin edebilmektedir. Ancak, KU-MCL algoritması ile konum tahmini yapılmaya devam edildiğinde parçacıklar, ölçümleri kendi buldukları konumdan değil de kendilerine en yakın ızgara hücrelerinin önceden depolanmış bölümlerinden aldıklarından lokalizasyon boyunca bazı hatalar oluşabilmektedir. Standart MCL'de ise parçacıklar ölçümlerini harita üzerinde buldukları konumdan alırlar;

fakat başlangıçta çok fazla parçacıkla lokalizasyona başlamak zorunda kalındığından, algoritmanın işlem süresi uzamakta ve bu olumsuz durum anlık konumlama hatalarına neden olmaktadır. Şekil 4'teki 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. Global lokalizasyonun başlangıcından itibaren parçacıkların tamamının robotun bulunduğu konuma yakınsadığını garanti edecek N adım sayısı, test ve deneme yoluyla belirlenmektedir. 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.

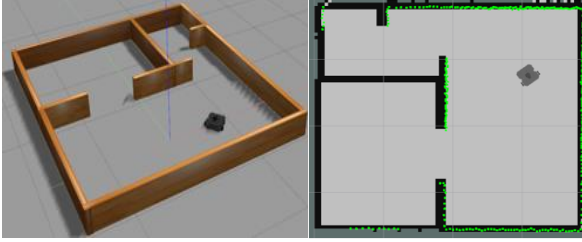


Şekil 4 Önerilen KU-MCL + MCL hibrit algoritmanın akış diyagramı.

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 5'teki gibi ROS (Robot Operating System) ([12]) ortamında bir test platformu oluşturulmuştur ve lokalizasyon algoritmasının ihtiyaç duyduğu harita bilgilerini sağlamak için ROS

ortamındaki hazır GMapping paketi ile ortamın haritası çıkarılmıştır.



Şekil 5 ROS Gazebo ortamında oluşturulan test platformu (sol) ve bu platformun çıkarılan haritası (sağ).

Şekil 6'da ise ROS Gazebo ortamında oluşturulan platforma benzer bir deney platformu gerçek ortamda oluşturulmuştur ve bu platformun ROS ortamında haritası çıkarılıp MATLAB ortamına aktarılmıştır.



Şekil 6 Gerçek ortamda oluşturulan deney platformu (sol) ve bu platformun çıkarılan haritası (sağ).

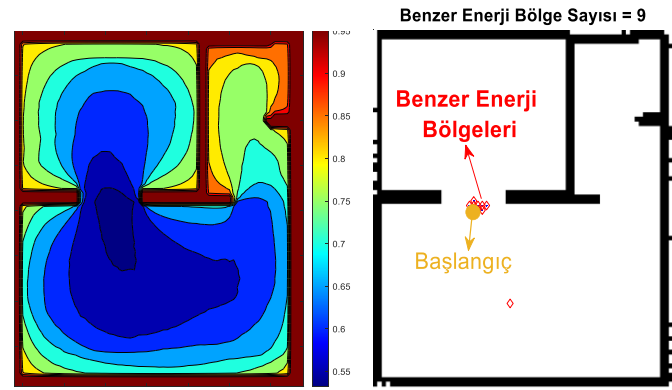
Şekil 5'te verilen platform üzerinde Gazebo ortamında URDF modeli (Universal Robot Description Format) olan Turtlebot3 waffle_pi ([13]) mobil robotu ile global lokalizasyon testleri yapıldıktan sonra Şekil 6'da verilen gerçek ortamda oluşturulmuş platform üzerinde bu sefer gerçek Turtlebot3 waffle_pi robotu kullanılarak gerçek dünya deneyleri yapılmıştır. Gerçek ortamda oluşturulan platformun boyutları yaklaşık olarak 4.2m x 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: NVIDIA® GTX 1050 4 GB
OS: Ubuntu 16.04
Programlar: MATLAB® 2021a, ROS kinetic, Python

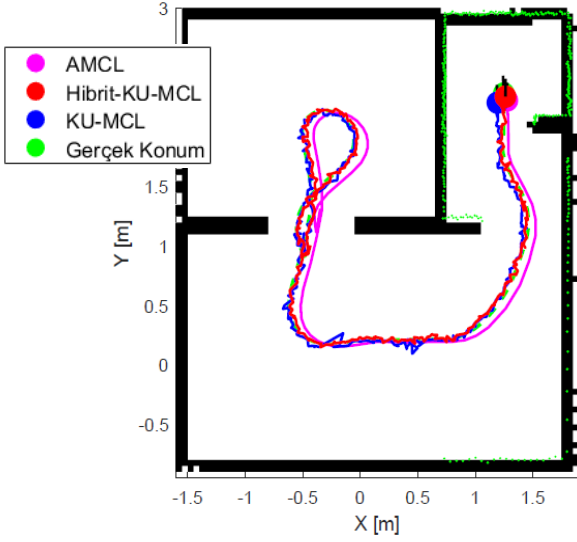
4.1 ROS ve MATLAB ortamları kullanılarak yapılan simülasyon sonuçları

Önerilen lokalizasyon algoritmasının doğruluğunu test etmek için öncelikle Şekil 5'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 7'deki gibi harita ızgara hücrelere ayrılmıştır, her bir ızgara hücrelerine ölçümler ve enerji bilgileri kaydedilmiştir, enerji kontur grafikleri oluşturulmuştur 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.



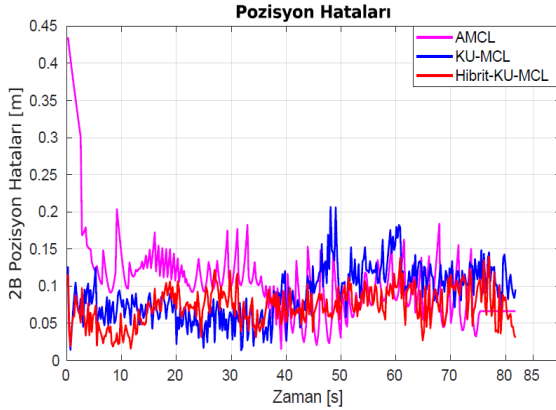
Şekil 7 Haritanın enerji kontur grafiği (sol), benzer enerji bölgelerinin dağılımı (sağ).

Şekil 7'deki 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 8'de görüldüğü gibi algoritmaların global lokalizasyon testleri MATLAB ortamında yapılmış ve tüm deneylerde KU-MCL için 5000, standart MCL için 50 parçacık kullanılmıştır. Global lokalizasyonun başlangıç noktası haritanın merkezine yakın bir yerde olduğu için AMCL algoritması kısa bir süre sonra robotun bulunduğu konuma yakınsayabilmiştir. Bunun nedeni ise global lokalizasyonun başlangıcında, AMCL algoritması parçacıkları haritanın tamamına rastgele uniform dağılımla atamaktadır ve bu parçacıkların başlangıç anında ağırlıkları eşit olduğundan konumlarının ağırlıklı ortalaması yaklaşık olarak haritanın merkezinde olmaktadır.

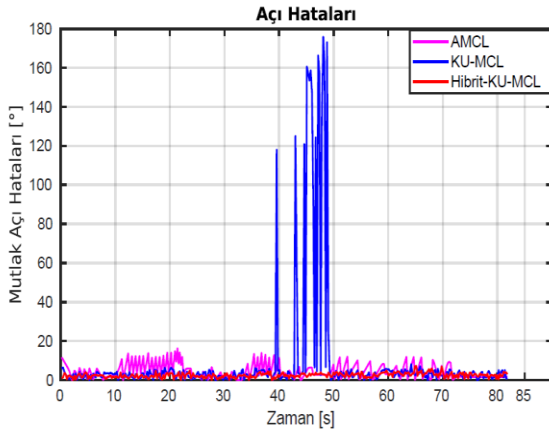


Şekil 8 Global lokalizasyon algoritmalarının performansları.

Lokalizasyon algoritmalarının performanslarını daha iyi analiz edebilmek için Şekil 9'da 2 boyutlu pozisyon hata grafikleri, Şekil 10'da ise mutlak açı hata grafikleri verilmiştir.



Şekil 9 Algoritmaların 2B pozisyon hata grafikleri.



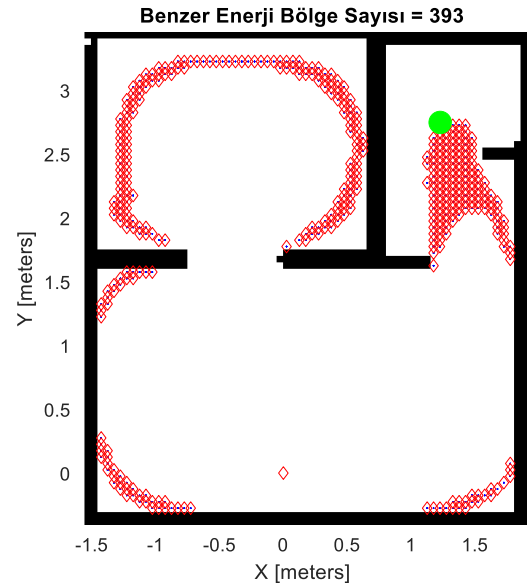
Şekil 10 Algoritmaların mutlak açı hata grafikleri.

Algoritmaların lokalizasyon performanslarının daha net görülebilmesi için Tablo 2'de performans sonuçları nümerik olarak verilmiştir.

Tablo 2 Algoritmaların performans sonuçları

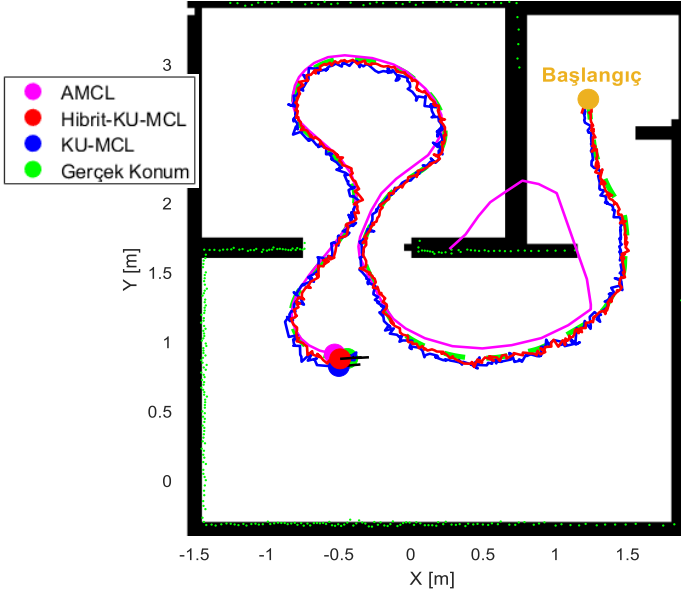
Algoritmalar	Ortalama 2B Konumlama Hatası (m)	Standart Sapmalar (m)	Ortalama Mutlak Açı Hatası (°)
Hibrit-KU-MCL	0.0727	0.0244	2.4470
KU-MCL	0.0889	0.0353	9.4664
AMCL	0.1067	0.0587	4.4616

Tablo 2 incelenip Hibrit-KU-MCL algoritması orijinal KU-MCL ile karşılaştırıldığında, 2B konumlama ve mutlak açının ortalama hataları sırasıyla yaklaşık 18.22% ve 74.15% oranında azaltılmıştır. Hibrit-KU-MCL algoritması AMCL ile karşılaştırıldığında, 2B konumlama ve mutlak açının ortalama hataları sırasıyla yaklaşık 31.865% ve 45.154% 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 kaçırmış gibi hissederken, KU-MCL algoritması 11 defa kaçırmış gibi hissetmiştir. Ancak robot, lokalizasyon boyunca hiç kaçırmamıştır. Bu durum Hibrit-KU-MCL algoritmasının daha doğru tahminler yaptığını göstermektedir. Başlangıç noktasının haritanın merkezinden daha uzakta olduğu ikinci simülasyon deneyi yapılmış, robotun başlangıç noktasına göre benzer enerji bölgelerinin dağılımı Şekil 11'deki gibi oluşmuştur.



Şekil 11 Benzer enerji bölgelerinin dağılımı.

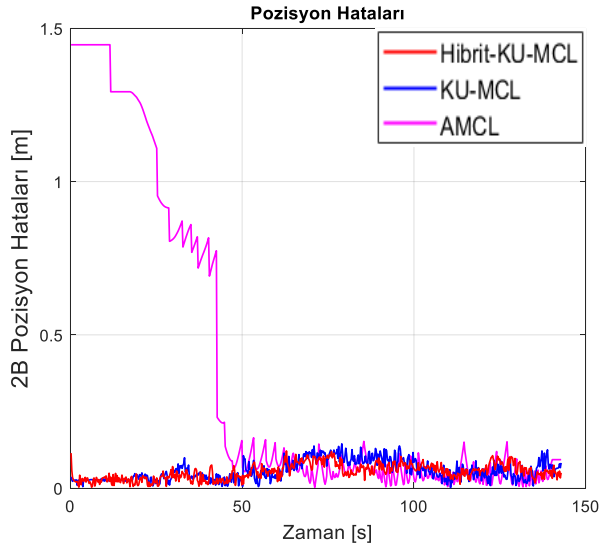
Şekil 12'de ikinci simülasyon deneyinin global lokalizasyon performansları verilmiştir.



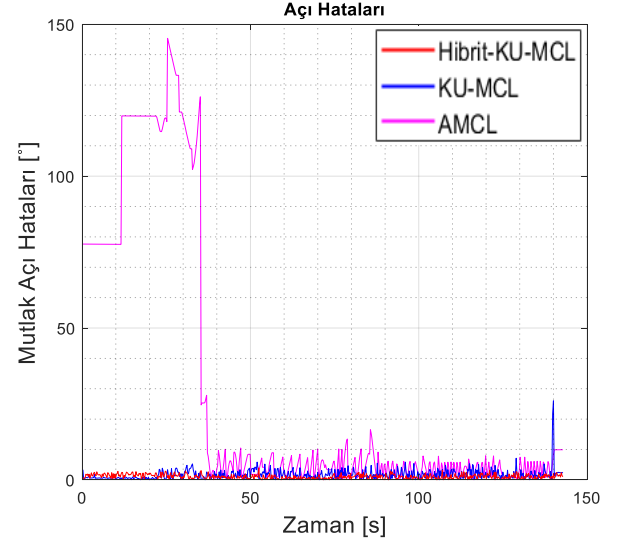
Şekil 12 Global lokalizasyon algoritmalarının performansları.

Şekil 12'de görüldüğü üzere global lokalizasyona haritanın merkezinden uzak bir yerde başladığında AMCL algoritması robotun konumuna daha uzun zaman sonra yakınsayabilmiştir. Ancak KU-MCL tabanlı algoritmalar yine ilk adımda robotun konumunu tahmin etmeyi başarabilmişlerdir.

Lokalizasyon algoritmalarının performanslarını daha iyi analiz edebilmek için Şekil 13'te 2 boyutlu pozisyon hata grafikleri, Şekil 14'te ise mutlak açı hata grafikleri verilmiştir.



Şekil 13 Algoritmaların 2B pozisyon hata grafikleri.



Şekil 14 Algoritmaların mutlak açı hata grafikleri.

Algoritmaların lokalizasyon performanslarının daha net görülebilmesi için Tablo 3'te performans sonuçları nümerik olarak verilmiştir.

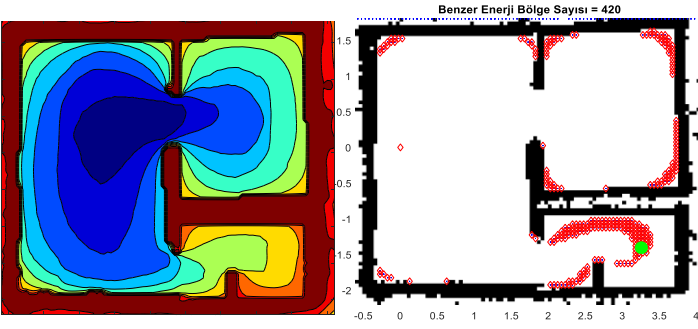
Tablo 3 Algoritmaların performans sonuçları

Algoritmalar	Ortalama 2B Konumlama Hatası (m)	Standart Sapmalar (m)	Ortalama Mutlak Açı Hatası (°)
Hibrit-KU-MCL	0.0526	0.0257	1.0768
KU-MCL	0.0587	0.0323	1.6684
AMCL	0.3807	0.5096	29.4588

Tablo 3 incelendiğinde Hibrit-KU-MCL algoritması KU-MCL ile karşılaştırıldığında, 2B konumlama ve mutlak açının ortalama hataları sırasıyla yaklaşık 10.39% ve 35.46% oranında azaltılmıştır. Hibrit-KU-MCL algoritması AMCL ile karşılaştırıldığında, 2B konumlama ve mutlak açının ortalama hataları sırasıyla yaklaşık 86.18% ve 96.34% oranında azaltılmıştır. Birinci deneyle ikinci deney kıyaslandığında hibrit KU-MCL ile orijinal KU-MCL algoritmalarının 2B pozisyon ve mutlak açı hataları azalırken AMCL algoritmasının tam tersine 2B pozisyon ve mutlak açı hataları artmıştır. Bunlara ilaveten, Global lokalizasyon süreci boyunca robot hiç kaçırılmamasına rağmen orijinal KU-MCL algoritması robotun 3 kez kaçırıldığını algılamakta Hibrit KU-MCL algoritması 1 kez kaçırıldığını algılamıştır. Bu hatalı algılamalar orijinal KU-MCL algoritmasını lokalizasyon tahmininde iki adım geriye düşürmüştür.

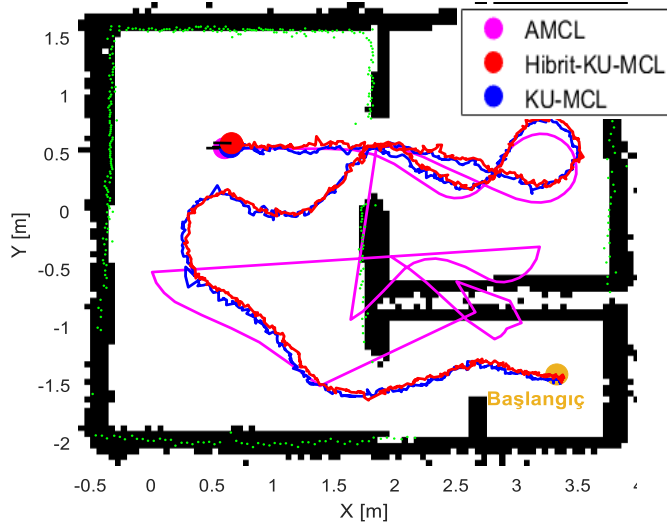
4.2 Gerçek ortamda yapılan deney sonuçları

Şekil 6'da verilen deney platformunun önce haritası çıkarılmış daha sonra Şekil 15'teki gibi enerji kontur grafiği oluşturulmuştur ve bir global lokalizasyon performans deneyi yapılmıştır.



Şekil 15 Haritanın enerji kontur grafiği (sol), benzer enerji bölgelerinin dağılımı (sağ).

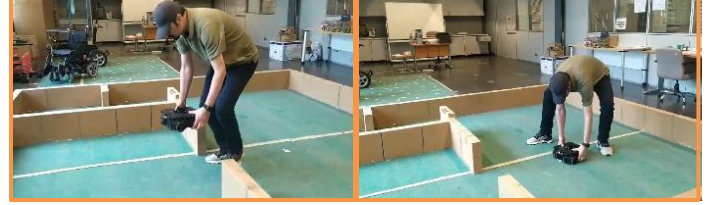
Harita önbelleğe alınıp ızgara hücrelerindeki depolara enerji ve ölçüm değerleri kaydedildikten sonra gerçek ortamda Şekil 16'daki gibi global lokalizasyon deneyi yapılmıştır. ROS ortamında mobil araç ile ssh protokolü kullanılarak haberleşme sağlandıktan sonra mobil araçtan gelen veriler ROS ortamında laptopa kaydedilmiştir.



Şekil 16 Algoritmaların global lokalizasyon performansları.

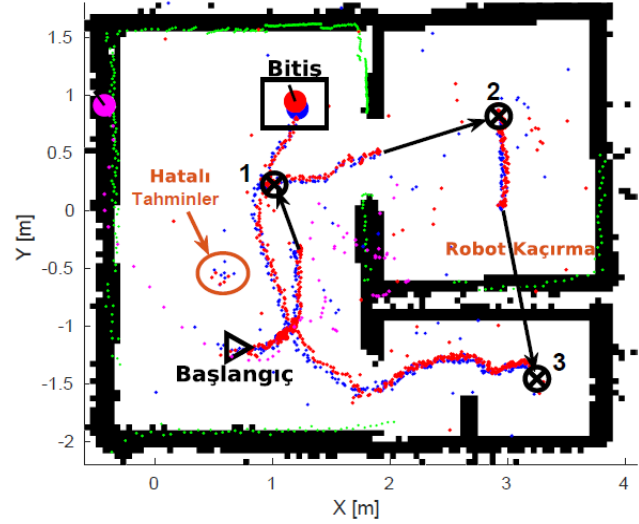
Şekil 16'da verilen algoritmaların global lokalizasyon performansları hakkında bir değerlendirme yapılacak olursa Hibrit KU-MCL algoritmasının orijinal KU-MCL algoritmasına göre daha düzgün bir rotaya sahip olduğu, orijinal KU-MCL'nin rotasının daha zig-zaglı olduğu söylenebilir. AMCL algoritması için ise bu algoritmanın ROS ortamında yapılan simülasyon performansından daha kötü bir performansa sahip olduğu, neredeyse gerçek robotun konumuna, lokalizasyon işleminin bitişine doğru yakınsadığı söylenebilir. Global lokalizasyonun başlangıcında, Hibrit-KU-MCL ile orijinal KU-MCL algoritmalarının her ikisi de robotun konumuna ilk adımda yakınsamıştır. Global lokalizasyon süreci boyunca, Hibrit KU-MCL algoritması 9 kez robotu kaçırılmış gibi algılarken, orijinal KU-MCL algoritması 12 kez kaçırılmış gibi algılamıştır.

Gerçekte ise lokalizasyon boyunca robot hiç kaçırılmamıştır. Yapılan son deneyde ise bu sefer gerçekten robot daha önceden belirlenmiş noktalara fiziksel olarak kaçırılmıştır. Deney senaryosuna göre önce robot, harita üzerinde belirli bir noktaya yerleştirildi; fakat algoritmalar robotun başlangıç konumunu bilmiyorlardı. Daha sonra Şekil 17'deki gibi robot, önceden belirlenmiş 3 farklı noktaya kaçırıldı.



Şekil 17 Robotun kaçırılma anı.

Şekil 18'de ise robot kaçırma deneyinde toplanan ölçüm verileri kullanılarak MATLAB ortamında yapılan performans testi görülmektedir.



Şekil 18 Algoritmaların robot kaçırma deneyindeki performansları.

Deneyin başlangıcından bitişine kadar geçen sürede toplamda Hibrit-KU-MCL algoritması 56 kez robotun kaçırıldığını algılayarak, 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 kaçırıldığını algılaması 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 hem simülasyon ortamında hem de gerçek ortamda yapılan deneylerle gösterilmiştir. Gelecekte ise önerilen KU-MCL tabanlı hibrit global lokalizasyon algoritması ile otonom tekerlekli sandalye gibi farklı kara robotlarının konumlarının belirlenmesi üzerine çalışmalar yapılacaktır.

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. Altınpinar, E. C. Contarli, A. Kağızman, 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] O. V. Altınpinar and V. Sezer, "A novel indoor localization algorithm based on a modified EKF using virtual dynamic point landmarks for 2D grid maps," Robotics and Autonomous Systems, vol. 170, (2023): 104546.
- [8] 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.
- [9] 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.
- [10] S. Thrun, W. Burgard, and D. Fox, "Probabilistic Robotics," Kybernetes, vol. 35, no. 7/8, pp. 1299–1300, Jan. 2006, doi: 10.1108/03684920610675292.
- [11] D. Fox, "Kld-sampling: Adaptive particle filters and mobile robot localization," Adv. Neural Inf. Process. Syst., vol. 151, p. 152, 2002.
- [12] 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.
- [13] 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.