TinyOS Tabanlı Telsiz Duyarga Ağları için Bir Konumlandırma ve *k*-Bağlılık Denetleme Sistemi

Orhan DAĞDEVİREN[1], Vahid Khalilpour AKRAM[2]

[1,2] Uluslararası Bilgisayar Enstitüsü, Ege Üniversitesi, İzmir, Türkiye
orhan.dagdeviren@ege.edu.tr, vahid59@gmail.com
(Geliş/Received:09.06.2016; Kabul/Accepted:03.02.2017)
DOI: 10.17671/gazibtd.309291

***Özet*—** Telsiz duyarga ağlarında (TDAlarında) düğümlerin konumlarının bulunması ve aralarındaki bağlantıların denetimi önemli konulardandır. Konumlandırma probleminde, amaç konumları belli olmayan mobil veya statik düğümlerin koordinatlarını diğer kök düğümlerin yardımıyla bulmaktır. *k*-Bağlılık denetleme probleminde amaç en az kaç adet düğüm bozulduğunda ağın bağlılığının bozulduğunu bulmaktır. Bu çalışmada TDA için bir konumlandırma ve *k*-bağlılık denetleme sistemi tasarlanmıştır ve uygulanmıştır. Bu sistemde trilaterasyon algoritması kullanılarak düğümlerin güncel pozisyonları diğer kök düğümlerden gelen mesajların üzerinden bulunmaktadır. Uygulanan sistemde düğümlerin arasındaki mesafeler gelen sinyallerin RSSI (Received Signal Strength Indicator) değerinden tahmin edilir. Konumu belli olmayan bir düğüm, en az 3 kök düğümden mesaj aldıktan sonra kendi koordinatını hesaplayabilir. Tüm gönderilen mesajlar ve bulunan koordinatlar, ağın çıkış düğümü tarafından algılanıp, bir Java uygulamasına aktarılır ve arayüzler üzerinden kullanıcıya sunulur. Ağın son bağlılık durumunu göstermek için gelen mesajlar üzerinden oluşturulan güncel topoloji üzerinde bir *k*-bağlılık denetleme algoritması çalıştırılır. Önerilen ve MEMSIC-IRIS düğümlerin üzerinde denetlenen sistem tüm TinyOS işletim sistemini destekleyen düğümlerin üzerinde çalışabilir. Önerilen sistem çeşitli uygulamalarda bir altyapı olarak kullanılabilir.

***Anahtar Kelimeler*—** Telsiz Duyarga Ağları, Konumlandırma, Trilaterasyon, Bağlılık Denetleme, *k*-Bağlılık

A Localization and *k*-Connectivity Detection System for TinyOS based Wireless Sensor Networks

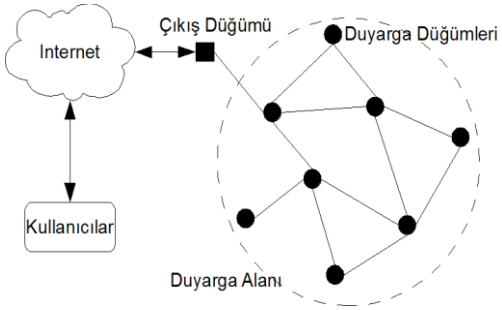
***Abstract*—** Localization and connectivity detection are two important topics in Wireless Sensor Networks (WSNs). In localization problem the aim is finding the position of mobile or static nodes with unknown coordinates. The purpose of the *k*-connectivity problem is to find the number of nodes whose removal destroys the connectivity of the network. In this paper, we have designed and implemented a localization and *k*-connectivity detection system. The proposed system uses trilateration approach to find the up to date locations of nodes where the trilateration method uses the messages from anchor nodes. In our system, the RSSI (Received Signal Strength Indicator) values of transmitted messages are used to estimate the distance between nodes. Using the proposed system, nodes can estimate their locations after receiving messages from at least 3 anchor nodes. The exchanged message and detected positions are snooped by the sink node, transferred to a Java program and presented to the user on GUIs. To find the latest connectivity statuses of a network, a *k*-connectivity testing algorithm periodically is run on the current detected topology from incoming messages. The proposed system has been tested with MEMSIC-IRIS motes and supports all WSNs which use TinyOS based motes. This system can be used as an infrastructure for various applications in WSNs.

***Keywords*—** Wireless Sensor Networks, Localization, Trilateration, Connectivity Detection, *k*-Connectivity

1. GİRİŞ (INTRODUCTION)

Günümüzdeki sistemler artık küçük ve ucuz bir entegre devre üzerinde, az güç kullanan haberleşme ünitesiyle tasarlanmaya başlanmıştır. Bu tasarım tekniği sayesinde telsiz duyarga ağlarının (TDAlarının) kullanılması mümkün olmuştur. TDA, çevreden algılama yapan ve telsiz haberleşen düğümlerin sabit bir altyapı olmadan oluşturdukları bir ağdır. Bir TDAnda yüzlerce hatta binlerce duyarga düğüm kendini yönetebilecek şekilde tasarsız bir yapıda kurulur [1][2]. Çevre gözetimi, askeri takip, sağlık hizmetleri ve akıllı tarım TDAyla yapılabilinecek örnek uygulamalardır [3-5].

Şekil 1'de örnek bir TDA yapısı verilmiştir. Duyarga düğümlerin çevreden topladığı veri diğer düğümler üzerinden çok zıplamalı (hop) olarak çıkış düğümüne (sink node) ulaşır. Çıkış düğümü, diğer düğümlerden gelen verileri toplar ve kullanıcıya yönlendirmek üzere diğer ağ yapısına iletilen ağ geçiti (gateway) vazifesini görür. Çıkış düğümü şekilde görüldüğü üzere Internet üzerinden verilerini gönderebileceği gibi uydu teknolojisi ve hücresel ağ teknolojisi de kullanabilir. Şekilde çift taraflı oklarla da gösterildiği üzere bazı durumlarda kullanıcılar TDAyı değiştirmek için gerekli bilgileri gönderebilmektedir.



Şekil 1. Telsiz duyarga ağlarının genel yapısı
(General architecture of wireless sensor networks)

TDAlar üzerinde düğümlerin konumlarının bulunması ve ağın bağlılığının denetlenmesi önemli problemlerdir. TDAda konumlandırma işlemi, konumunu bilmeyen bir düğümün, konumunu bilen diğer düğümleri temel alarak konumunu bulması işlemidir [6-11,15-27]. Birçok durumda, TDA içindeki bir düğümün konumunu öğrenmesi gerekmektedir. Örneğin hedef takip (target tracking) ve olay tespit (event detection) uygulamalarında konumlandırma işleminin önemi çok büyüktür. Büyük ölçekli TDA uygulamalarında, her düğümün konumunun kullanıcı tarafından verilmesi çok zordur. Bunun yanında, GPS pahalı bir tercih olabilmektedir. Bu sebeplere dayanarak konumlandırma işlemi TDAda çok önemlidir [6].

Genel olarak TDAda düğümlerin arasında bir güvenilir iletişim altyapısı olmadığından dolayı, düğümlerin arasında bulunan bağlantılar tüm ağın güvenilirliğini etkilemektedir. Bağlı bir ağda bütün düğümler arasında bir patika bulunur başka bir ifadeyle bağlı bir ağda her düğüm diğer düğümlerin yardımıyla verilerini hedef düğümüne ulaştırır. En kötü durumda bir düğümün bozulması ağı iki bağımsız parçaya bölüp, bazı düğümleri ulaşılmaz hale getirebilir. Bu sebepten ağın bağlılığının periyodik olarak kontrol edilmesi önemlidir. Eğer bir ağda herhangi iki düğümün arasında en az k bağımsız patika varsa o ağ k-bağlıdır. Bir k-bağlı ağda herhangi k-1 düğümün bozulması, diğer düğümlerin arasındaki bağlantıları koparmaz. Bu nedenle güvenilir ağların k değeri yüksek olmalıdır. Ağın k değerinin denetlenmesi önemli bir problemdir.

Bu çalışmada TDAlar için bir verimli konumlandırma ve k-bağlılık kontrol sisteminin tasarımı ve uygulanması anlatılmaktadır. Bu sistem IRIS düğümleri üzerinde TinyOS işletim sistemi için tasarlanıp, nesC ve Java dillerinde uygulanmıştır. Uygulanan sistemde TDAda bulunan düğümler kendi koordinatlarını diğer kök düğümlerin yardımıyla bulabilirler. Her düğüm koordinatını değiştirdiği zaman sistem otomatik olarak yeni konumunu hesaplar. Tasarlanan Java tabanlı arayüz sayesinde düğümlerin güncel konumları bulunur ve ağın güncel bağlılık değeri kullanıcıya sunulur.

İlerleyen bölümler şu şekilde organize edilmiştir. Bölüm 2'de kullanılan mesafe ölçme yöntemi, konumlandırma yöntemi ve k-bağlılık denetleme yöntemi anlatılmıştır. Bölüm 3'de üzerinde çalışılan sistemin, tasarlanan TinyOS bileşenlerinin ve Java arayüzünün detayları verilmiştir. Bölüm 4'de sonuçlar değerlendirilmiştir.

2. KULLANILAN YÖNTEMLER (USED MATERIALS AND METHODS)

Bu bölümde tasarladığımız sistemde kullandığımız mesafe ölçme yöntemi, konumlandırma yöntemi ve bağlılık denetleme yönteminin açıklamaları verilmiştir.

2.1. Mesafe Ölçme Yöntemi (Distance Estimation Method)

TDAda düğümlerin arasındaki mesafeyi ölçmek için farklı yöntemler sunulmuştur ki bu yöntemlerin arasında Geliş Zamanına Göre Ölçme (Time of Arrival (TOA)) [7], Geliş Zamanı Farkına Göre Ölçme (Time Difference of Arrival (TDOA)) [8] ve Alınan Sinyal Gücüne Göre Ölçme (Received Signal Strength (RSS)) [9] bilinen yöntemlerdir. TOA yönteminde düğümlerin arasında zaman senkronizasyonu yapılması gereklidir. Zaman senkronizasyonu başlı başına önemli bir sorundur. TDOA yönteminde en az 2 alıcı/verici gereklidir ve Cricket gibi özel düğüm tasarımları bu donanıma sahiptir [10]. Bu sebeplerden dolayı bu çalışmada RSS yöntemi kullanılmıştır.

İki düğümün arasında mesafe ölçmenin basit yollarından birisi alınan sinyallerin gücüne bakmaktır. Radyo sinyalleri havada hareket ederken güçleri azalır ve bu yüzden göndericiye yakın bir düğümde alınan sinyal

gücü, uzaktaki düğümlere göre daha fazladır. Düğümlerde alınan sinyal gücü RSS olarak isimlendirirler ve dBm birimiyle ölçülür.

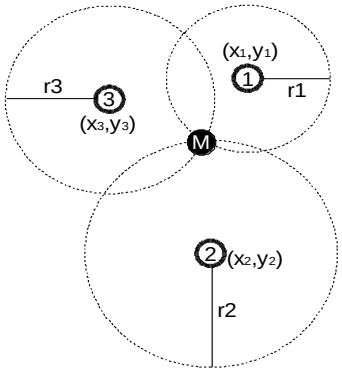
Yapılan denemelere göre genel olarak RSS değeri alıcı ve göndericinin uzaklığı ve aralarında bulunan engellere bağlı -40 dBm ve -140 dBm arasında olmaktadır. Eğer iki düğüm birbirine çok yakın olup, aralarında hiçbir engel olmazsa RSS değeri yaklaşık -40 dBm ve eğer iki düğümün arasındaki mesafe çok uzak veya aralarında bir nesne olursa RSS değeri yaklaşık -140 dBm olur. Genel olarak -140 dBm'den daha düşük alınan sinyaller parazit olarak gözardı edilir. Alınan RSS değerini kullanarak aşağıdaki formülden düğümlerin arasındaki mesafeyi tahmin edebiliriz [11]:

$$d = d_0 * 10^{\frac{RSS_0 - RSS}{10n}}$$

Yukarıdaki formülde *d* düğümlerin arasındaki mesafe, d_0 referans mesafe (örneğin 1 m), RSS_0, iki düğümün arasındaki mesafe d_0'iken alınan sinyal gücü, *RSS* gelen sinyalin gücü ve *n* sinyalin yolda kaybettiği güç oranı olmaktadır. *n* değeri farklı düğümlerde, antenin donanımı ve gücüne göre değişir.

2.2. Konumlandırma Yöntemi (Localization Method)

Trilaterasyon yönteminde konumlandırmanın yapılabilmesi için konumu belli olmayan bir düğümün, konumu bilen en az 3 adet kök düğümden, konum bilgilerini ve aralarındaki mesafeleri alması gerekmektedir.



Şekil 2. Trilaterasyon yöntemi
(Trilateration method)

Şekil 2'de görüldüğü gibi üç çemberin kesişimi bir nokta olabilir ve böylece konumu belli olmayan düğümün koordinatları çember denkleminden bulunabilir. Her bir kök düğümün çemberi için aşağıdaki denklemi yazabiliriz:

$$(x_i - x)^2 - (y_i - y)^2 = r_i^2 \quad for \quad i = 1,2,3 \qquad (1)$$

Denklem 1'de (x, y) konumu belli olmayan düğümün koordinatı ve (x_i, y_i), i nolu kök düğümün koordinatlarıdır. Eğer 3 nolu kök düğümün denklemini 1 ve 2 nolu kök düğümlerin denklemlerinden çıkarırsak, 2 ve 3 nolu denklemler elde edilir:

$$(x_1 - x)^2 - (x_3 - x)^2 + (y_1 - y)^2 - (y_3 - y)^2 = r_1^2 - r_3^2 \qquad (2)$$

$$(x_2 - x)^2 - (x_3 - x)^2 + (y_2 - y)^2 - (y_3 - y)^2 = r_2^2 - r_3^2 \qquad (3)$$

2 ve 3 nolu denklemleri (x, y) değişkenlerine göre düzenlersek aşağıdaki denklemler elde edilir:

$$2(x_3 - x_1)x + 2(y_3 - y_1)y = (r_1^2 - r_3^2) - (x_1^2 - x_3^2) - (y_1^2 - y_3^2) \qquad (4)$$

$$2(x_3 - x_2)x + 2(y_3 - y_2)y = (r_2^2 - r_3^2) - (x_2^2 - x_3^2) - (y_2^2 - y_3^2) \qquad (5)$$

4 ve 5 nolu denklemleri matris şeklinde yazabiliriz:

$$2\begin{bmatrix} x_3 - x_1 & y_3 - y_1 \\ x_3 - x_2 & y_3 - y_2 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} (r_1^2 - r_3^2) - (x_1^2 - x_3^2) - (y_1^2 - y_3^2) \\ (r_2^2 - r_3^2) - (x_2^2 - x_3^2) - (y_2^2 - y_3^2) \end{bmatrix} \qquad (6)$$

Böylece 6 nolu denklemden *x* ve *y* değişkenlerinin değeri diğer düğümlerin konumu ve mesafelerinden elde edilir. Örneğin bir TDAda kök düğümlerin koordinatlarının $(x_1, y_1) = (2,1)$, $(x_2, y_2) = (5,4)$, $(x_3, y_3) = (8,2)$ olduğunu varsayalım. Eğer koordinatı belli olmayan bir düğümün mesafesi bu kök düğümlere sırayla $r_1 = \sqrt{10}$, $r_2 = 2$ ve $r_3 = 3$ uzaktan olursa, bu düğümün koordinatı aşağıdaki şekilde bulunabilir:

$$2\begin{bmatrix} 6 & 1 \\ 3 & -2 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 - (-60) - (-3) \\ -5 - (-39) - 12 \end{bmatrix} \qquad (7)$$

$$\Rightarrow 6x + y = \frac{64}{2}\ ,\ 3x - 2y = \frac{22}{2} \quad \Rightarrow \quad (x, y) = (5,2)$$

Yukarıdaki denklemler bir doğrusal denklem sistemini oluşturmaktalardır. Doğrusal denklem sistemlerini çözebilen algoritmalardan biri Gauss Jordan eleme yöntemidir. Genel olarak Gauss Jordan algoritması iki aşamadan oluşmaktadır. İleri yok etme (forward elimination) aşamasında $i > 1$ inci satırın tüm $j < i$ inci sütunlularındaki değerler bir önceki satırın değerlerinin katlarıyla toplanıp, bir üst üçgen matris elde edilir. İkinci aşamada elde edilen üst üçgen matrisinde geri yer değiştirme (backward substitution) yöntemiyle son satırdan başlayarak her değişkenin değeri bir önceki satırın değişkenlerinin aracıyla bulunur. Üst üçgen matrisinde son satırda sadece bir değer bulunduğu için, son değişkenin değeri direkt olarak bulunabilir. İleri yok etme yöntemi aşağıdaki algoritmada gösterilmiştir. İleri yok etme algoritmasında *G* matrisin referansı, *n* ise satır sayısıdır.

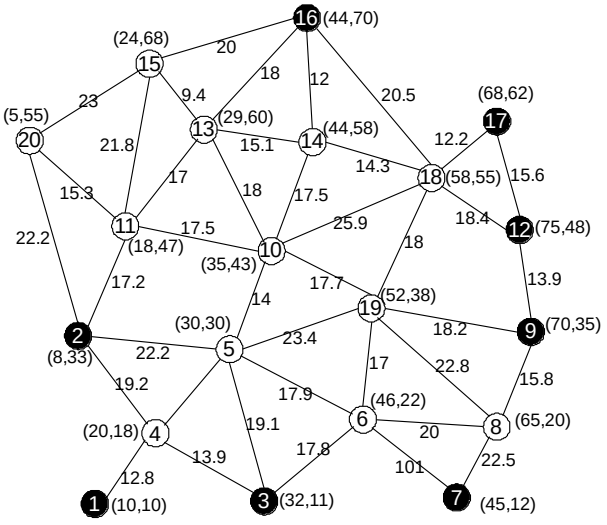
1: ***Algorithm Forward Elimination (G , n)***
2: ***Begin***
3: *for k=1 to n-1 do*
4: *for i=k to n do*
5: $G_{ik} = G_{ik}/G_{kk}$
6: *for j=k+1 to n+1 do*
7: $G_{ij} = G_{ij} - G_{ik} * G_{kj}$
8: ***End***

Yukarıdaki algoritma *G* matrisini bir üst üçgen matrisine çevirir ve ondan sonra değişkenlerin değeri aşağıdaki geri yer değiştirme algoritmasıyla bulunur. Geri yer değiştirme algoritmasında G matrisin referansı, *n* satır sayısı ve *x* yerine konulacak değişkendir.

1: ***Algorithm Backward Substitution (G , n , x)***
2: ***Begin***
3: *for i=n to 1 do*
4: $x_i = G_{in+1}$
4: *for j=i+1 to n do*
5: $x_i = x_i - G_{ij} * x_j$
6: $x_i = x_i/G_{ii}$
7: ***End***

Geri yer değiştirme algoritmasında ilk olarak en son değişkenin değeri matrisin en son satırından bulunur ve böylece her değişkenin değeri bulunduktan sonra bir önceki satırdaki denklemde çözülebilir.

Şekil 3'de örnek bir TDA bulunmaktadır. Bu şekilde, kök düğümler siyah ve konumlarını bilmeyen düğümler beyaz renk ile gösterilmektedir. Kök düğümler, uygulamaya başlarken, kendi koordinatlarını tüm komşularına gönderirler. Beyaz düğümler, kendi koordinatlarını, siyah düğümlerden gelen mesajları kullanarak bulurlar ve buldukları koordinatları komşularına gönderirler. Böylece tüm beyaz düğümler kendi koordinatlarını bulabilirler.

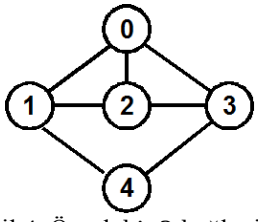


Şekil 3. Örnek bir duyarga ağı
(A sample wireless sensor network)

Şekil 3'de kenarların üzerindeki sayılar düğümlerin arasındaki mesafelerdir. Ayrıca tüm düğümlerin koordinatları onların yanında yazılmıştır. Bir siyah düğüm, kendi koordinatını hesaplamak için en az 3 komşusundan mesaj almalıdır. Bu nedenle Şekil 3'de, ilk aşamada, sadece 4 ve 18 numaralı düğümler kendi koordinatlarını hesaplayabilecekler. 4 numaralı düğüm kendi koordinatını hesaplandıktan sonra, bu koordinatı komşularına gönderir. Bu durumda, 5 numaralı düğüm 3 komşusundan mesaj alıp, kendi koordinatını hesaplar ve elde edilen bilgiyi yeni bir mesajda komşularına gönderir. Böylece sırayla 6, 8, 19, 10, 14, 13, 11, 15 ve 20 numaralı düğümler kendi koordinatlarını elde edebilirler.

2.3. k-Bağlılığı Denetleme Yöntemi (k-Connectivity Detection Method)

Bir ağın *k*–bağlılık değerini bulmak, o ağın güvenilirliğini belirleyen ana yollarından biridir ve bu yüzden *k* değerini denetleyen çeşitli algoritmalar sunulmuştur. Bir *k*-bağlı ağda aktif düğümlerin arasındaki bağları tamamen kesilmesi için en az *k* düğümün bozulması gerekmektedir. Örneğin Şekil 4'de görünen çizgenin *k* değeri 2 olduğundan dolayı, bu çizgenin bağlılığını bozmak için en az 2 düğümün silinmesi gerekmektedir. *k*-bağlılık denetleme algoritmalarının temel amacı mevcut bir ağın *k* değerini bulmaktır.



Şekil 4. Örnek bir 2-bağlı çizge
(A sample 2-connected graph)

Bu güne kadar yönsüz çizgelerde *k* değerini bulan asimptotik olarak zaman karmaşıklığı en iyi algoritma Henzinger tarafından sunulmuştur [12]. Bu sebeple bu çalışma bu algoritma üzerinden yapılmıştır. Henzinger, Hao [13] ve Even'in [14] minimum kesim ve *k* denetleme algoritmalarını birleştirerek, $O(kn^3)$ karmaşıklığına sahip bir denetim algoritması sunmuştur. *Henzinger-K-Detection* algoritması bu algoritmanın basamaklarını göstermektedir.

1: ***Algorithm Henzinger-K-Detection (G)***
2: ***Begin***
3: *i=0;*
4: *k=n-1;*
5: *repeat*
6: *i=i+1;*
7: *k=min(k, minCut(x_i ,G), minCut(x_i ,G^R));*
8: *until i>k*
9: ***End***

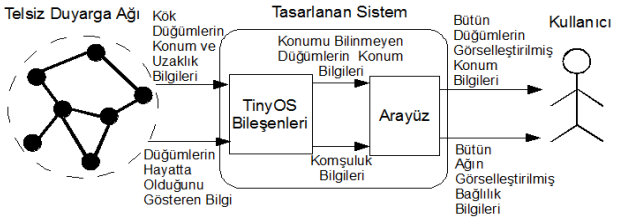
Henzinger-K-Detection algoritmasında *minCut* fonksiyonu x_i düğümünü *G*'den koparan en düşük düğümlerin kümesini vermektedir. Hao'nun algoritmasını kullanarak bu küme $O(n^3)$ karmaşıklığıyla bulunur. Herhangi bir k bağlı çizgede en fazla k düğümü

kontrol ettikten sonra, o çizgeyi bölen en düşük düğümlerin kümesi elde edilir. Eğer $C = \{c_1, c_2, \dots c_k\}$ bir çizgeyi bölen düğümler kümesinin en küçüğü ise, herhangi $v \notin C$ düğümünün *minCut* kümesi C'ye eşit olur. C kümesinin boyutu k olduğundan dolayı, çizgede en fazla $k+1$ düğümün *minCut* kümesini bulmak, k değerini belirler. Repeat-until döngüsünün en yüksek tekrarı k olduğu için, algoritmanın zaman karmaşıklığı $O(kn^3)$ olmaktadır.

3. TASARLANAN KONUMLANDIRMA VE *k*-BAĞLILIK DENETLEME SİSTEMİ (DEVELOPED LOCALIZATION AND *k*-CONNECTIVITY DETECTION SYSTEM)

TDAlar için tasarlanan konumlandırma ve k-bağlılık denetleme sistemi, iki temel bileşenden oluşmaktadır. İlk bileşen TinyOS işletim sistemini çalıştıran IRIS duyarga düğümleri için nesC dilinde hazırlanmıştır. Bu bileşen aracıyla düğümler kendi komşularını bulup, onlardan gelen mesajların sinyal gücü ve koordinatları üzerinden kendi koordinatlarını hesaplayabilirler. İkinci bileşen bir Java tabanlı arayüz yazılımı olarak, düğümlerin arasında gönderilen mesajları, ağda bulunan çıkış düğümü aracıyla algılayıp, düğümlerin güncel koordinatları ve ağın bağlantı durumunu denetler. Düğümlerin hareket etmesi durumunda değişen koordinatlar ve bağlantılar bu arayüz yazılımı tarafından kullanıcıya duyurulur. Bu bölümde ilk olarak üzerinde çalışılan donanım ve işletim sistemi anlatılacak, daha sonra tasarlanan TinyOS bileşenleri detaylı olarak açıklanacak, en son tasarlanan arayüz yazılımı incelenecektir.

Şekil 5'de tasarlanan sistemin genel yapısı gösterilmektedir. Bu şekilde görüldüğü gibi TDAda kök düğümlerin konum ve uzaklık bilgileri ve ayrıca diğer düğümlerin hayatta olduğu bilgileri çıkış düğümü tarafından tasarlanan sisteme aktarılır. Bu sistem, gelen bilgileri girdi olarak alır, TinyOS bileşenlerini kullanır ve tüm düğümlerin koordinatları ve ağın bağlılık bilgileri olarak kullanıcıya arayüz üzerinden sunar.



Şekil 5. Tasarlanan sistemin genel yapısı
(General structure of designed system)

3.1 Üzerinde Çalışılan Donanım ve İşletim Sistemi (Underlying Hardware And Communication System)

Tasarlanan sistem IRIS düğümleri üzerinde çalıştırılmıştır. Şekil 6'da örnek bir IRIS düğümü gösterilmektedir. Bu düğümlerin özellikleri Tablo 1'de gösterilmektedir.

Şekil 6. Bir IRIS düğümü
(An IRIS node)

Tablo 1. IRIS cihazların özellikleri
(Main properties of IRIS nodes)

İşletim sistemi	TinyOS
Radyo Arayüzü	2.4 GHz, IEEE 802.15.4
İletim Hızı	250 kbps
İletim Gücü	3 dBm
Bellek	128 kB flash, 8 kB RAM

IRIS duyarga düğümleri üzerinde TinyOS işletim sistemi kullanılmıştır. TinyOS, TDAların düğümleri için düşük güç kullanarak TDAlar üzerinde uygulamaları daha kolay geliştirmeyi amaçlayan bir işletim sistemidir. TinyOS farklı bileşenlerden oluşmaktadır. Bu bileşenler uygulamalarda farklı amaçlar için kullanmaktalardır. Her uygulama kendi ihtiyaçlarına göre gereken bileşenleri programa ekleyerek daha az kaynak kullanarak amaçlarını gerçekleştirebilir. Genel olarak TinyOS işletim sistemi bir telsiz duyarga düğümünün bileşenlerine çeşitli servisler sunmaktadır.

Bir telsiz duyarga düğümü aynı zamanda farklı işlemler yapabileceğinden dolayı, bu düğümlerin işletim sistemi verimli bir şekilde eşzamanlı işlemleri desteklemelidir. Örneğin bir düğüm bir hesaplama yaparken, duyarga cihazından bir veri alma zorunda kalıp, aynı zamanda telsiz radyo üzerinden bir diğer düğümle iletişim kurabilir. Bu nedenle genel olarak TinyOS olay tabanlı bir işletim sistemidir. Uygulamalar her bileşenin farklı olayları için ayrı fonksiyonlar tanımlayabilirler ve böylece o olayın gerçekleşmesinde tanımlanan fonksiyon çalıştırılabilir. Örneğin cihazın açılması, radyo cihazından mesaj gelme, duyargadan bilgi alma ve zamanlayıcı olayları, yaygın olarak kullanılan olaylardandır.

TinyOS işletim sisteminin uygulamaları C dilinin gömülü sistemler için geliştirilmiş versiyonu, nesC diliyle geliştirilmektedir. nesC dilinde her uygulama çeşitli alt yapı bileşenlere bölünerek geliştirilir. Örneğin her düğümün radyo cihazı, ışıkları ve duyargaları farklı bileşenlerle kontrol edilmektedir. Bu çalışmada tasarlanan sistem nesC diliyle yazılmıştır.

TinyOS işletim sistemine hazırlanan uygulama, TOSSIM benzetim ortamında az sayıda değişiklerle çalıştırılabilir. TOSSIM, TinyOS işletim sistemine özel bir benzetim ortamıdır. nesC dilinde yazılan programlar TOSSIM benzetim ortamında, sanal topolojiler üzerinde çalıştırılabilir. Hazırlanan sistemde kullanıcı sanal bir topoloji oluşturarak, yazdığı kodunu, gerçek düğümler olmadan TOSSIM benzetim ortamı aracıyla çalıştırabilir.

3.2 *Tasarlanan TinyOS Bileşenleri (Developed TinyOS Components)*

Uygulanan sistemde düğümlerin arasında *Hello* ve *POS* mesajları gönderilir. *Hello* mesajları aracıyla düğümler kendi komşularını bulurlar. *POS* mesajları kök düğümler tarafından gönderilir ve diğer düğümler bu mesajları kullanarak kendi koordinatlarını hesaplarlar. *Hello* mesajı sayesinde düğümler komşuluk listelerini oluşturur. *POS* mesajının içinde göndericinin koordinatı bulunmaktadır. Önerilen sistemde gönderilen mesajların yapısı aşağıdaki gibidir.

```
typedef nx_struct MESSAGE {
   nx_uint8_t type;
   nx_uint8_t sender;
   nx_int16_t data;
   nx_int16_t X;
   nx_int16_t Y;
}    MESSAGE_t;
```

Gönderilen tüm mesajlarda *type* mesajın tipi, *sender* mesajı gönderdiği düğüm, *data* mesajın tipine bağlı olarak bir bilgi, *X* göndericinin *x* koordinatı ve *Y* göndericinin y koordinatıdır.
TinyOS işletim sisteminde her bileşen farklı arayüzlerini kullanarak düğümün üzerinde bulunan cihazlardan yararlanabilir. Aşağıdaki kodda uyguladığımız TinyOS programında kullanılan arayüzleri gösterilmektedir.

```
module Algorithm @safe() {
 uses {
   interface Leds;
   interface Boot;
   interface Receive;
   interface AMSend;
   interface Timer<TMilli> as BroadcastTimer;
   interface SplitControl as AMControl;
   interface Packet;
   interface Random;
   interface Timer<TMilli> as Sender;
   interface PacketField<uint8_t> as PacketRSSI;
  }
}
```

Yukarıdaki arayüzleri düğümlerin radyo cihazı ve ışıklarının kullanmasını sağlayıp, aynı zamanda mesaj üretme, mesaj gönderme ve zamanlama işlemlerini de gerçekleştirmektelerdir. Bu arayüzler TinyOS işletim sisteminin uygun bileşenlerine bağlanarak cihazın donanımlar ve üzerinde çalışan yazılımlarının arasında bir köprü oluştururlar. Kullanılan TinyOS bileşenleri aşağıdaki kodlarda gösterilmektedir.

```
configuration AlgorithmApp {}
implementation {
 components MainC,Algorithm as App, LedsC,RandomC;
 components new AMSenderC(6);
 components new AMReceiverC(6);
 components new TimerMilliC() as T1;
 components new TimerMilliC() as T3;
 components RF230ActiveMessageC;
 components ActiveMessageC;
 App.Boot -> MainC.Boot;
 App.Receive -> AMReceiverC;
 App.AMSend -> AMSenderC;
 App.AMControl -> ActiveMessageC;
 App.Leds -> LedsC;
 App.Packet -> AMSenderC;
 App.BroadcastTimer->T1;
 App.Sender->T3;
 App.Random->RandomC;
 App.PacketRSSI -> RF230ActiveMessageC.PacketRSSI;
}
```

MainC.Boot bileşeni *Application.Boot* arayüzüne bağlanıp, düğüm çalıştırıldığı zaman *boot* fonksiyonunun çağırılmasına sebep olur. *Boot* fonksiyonu çağırıldığı zaman *Leds* bileşenin aracıyla düğümün üzerinde ışıkları açar ve *AMControl* bileşeninin *startDone* fonksiyonunu çağırarak cihazın radyo alıcı-göndericisini başlatır.

```
 event void Boot.booted() {
     call Leds.set(1);
     call AMControl.start();
 }
```

Yukarıdaki kodda *AMControl* bileşeninin *start* fonksiyonu çağrılmaktadır. Bu fonksiyon cihazın radyo alıcı-göndericisini çalıştırır. Cihazın radyo alıcı-göndericisinin başlatılması sona erdikten sonra aşağıda verilen *AMControl.startDone* fonksiyonu çağrılacaktır.

```
 event void AMControl.startDone(error_t err) {
     if (err != SUCCESS)  call AMControl.start();
     else {
          call Leds.set(0);
          call BroadcastTimer.startPeriodic(1000);
     }
}
```

Eğer radyo alıcı-gönderici başarıyla çalışmaya başlamazsa bu cihaz yeniden başlatılır. Aksi halde düğümün kırmızı ışığı açılarak *BroadcastTimer* isimli bir zamanlama bileşeni başlatılır. *startPeriodic* fonksiyonu bir zamanlayıcı planlayarak, her saniyede bir kere *fire* fonksiyonunu çağırır. Aşağıdaki kodda görüldüğü gibi *fire* fonksiyonu içinde pozisyonu belli olan düğümlerin koordinatları diğer düğümlere gönderilir.

```
event void BroadcastTimer.fired() {
 if(PosDetected)
    send(POS,mobCount,TOS_NODE_ID);
}
```

Yukarıda kullanılan *send* fonksiyonu mesajın içinde gönderilmesi gereken bilgileri aldıktan sonra, bir mesaj üretip, düğümün koordinatlarını mesaja ekleyerek, mesajı kendi komşularına gönderir.

```
void send(uint8_t typ,int16_t Data,uint8_t sender){
   int i=0;
   MESSAGE_t* out_msg;
   out_packet_size=sizeof(MESSAGE_t)+4*(Data-1);
   out_msg= (MESSAGE_t*)call Packet.getPayload(
```

```
    &out_packet,out_packet_size);
    out_msg->type=typ;
    out_msg->sender=sender;
    out_msg->data=Data;
    out_msg->X=myX;
    out_msg->Y=myY;
    for(i=0;i<Data;i++){
            out_msg->tag[i*2]=mobilesID[i];
            out_msg->tag[i*2+1]=mobilesDist[i];
    }
    i=call Random.rand16()%500;
    call Sender.startOneShot(i);
}
```

Düğümlerin arasında gönderilen mesajların çakışma ihtimalini azaltmak için *send* fonksiyonunda her düğüm 0 ve 500 milisaniye arasında rasgele bir sayı üretir ve o kadar bekledikten sonra mesajını gönderir. Bekleme işlemi bir diğer *Timer* bileşenin yardımıyla uygulanmaktadır. *send* fonksiyonunda *Sender* zamanlama bileşeninin *startOneShot* fonksiyonu çağrılarak bu bileşenin *fire* fonksiyonu bir rasgele zamandan sonra çağrılır ve böylece mesaj gönderme işlemi bu fonksiyonun içinde gerçekleştirilir.

```
event void Sender.fired() {
    if(radioBusy==TRUE){
            return;
    }
    if (call AMSend.send(AM_BROADCAST_ADDR,
            &out_packet, out_packet_size) == SUCCESS){
                radioBusy = TRUE;
                call Leds.led0Toggle();
    }
}
event message_t* Receive.receive(message_t* bufPtr,void* pkt,
uint8_t len){
      MESSAGE_t* payLoad;
      payLoad = (MESSAGE_t*) pkt;
      if(payLoad->type==POS){
            if(call PacketRSSI.isSet(bufPtr)){
                int d=call PacketRSSI.get(bufPtr);
                d=rssiToDistance(d);
               addToList(payLoad->sender,d, pkt->X, pkt->Y);
            }
       }
       else addToNeıghbors(payLoad->sender);
       return bufPtr;
  }
```

receive fonksiyonunda eğer gelen mesajın tipi *POS* ise mesajın RSSI değeri *PacketRSSI* bileşenin yardımıyla elde edilir ve dBm cinsinden olan değer mesafe cinsine çeviriler. Ondan sonra *addToList* fonksiyonu aracıyla göndericinin id'si, koordinatı ve mesafesi bir tabloya eklenir. Bu tablo düğümün pozisyonunu bulmak için trilaterasyon algoritması tarafından kullanılır. Eğer gelen mesajın tipi *POS* değilse mesajın göndericisinin id'si düğümün komşularının listesine eklenir.

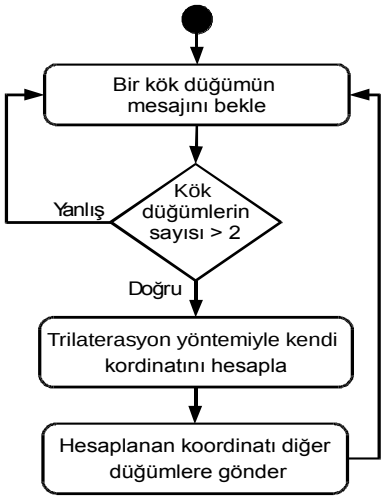
Eğer yeni gelen mesajın göndericisinin bilgileri önceden tabloda bulunursa, *addToList* fonksiyonunda eski bilgiler yeni bilgilerle değişilir. Aksi takdirde tabloya yenir bir satır eklenir.

```
void addToList(int16_t id,int16_t dist,int x, int y){
    int i=0;
    for(i=0;i<mobCount;i++)
        if(mobilesID[i]==id) break;
    nodeID[i]=id;
   nodeDist[i]=dist;
   nodeX[i]=x;
   nodeY[i]=y;
   if(i==mobCount)
     mobCount++;

}
```

Tabloda en az 3 satır bulunduktan sonra düğümün koordinatı aşağıdaki fonksiyonda trilaterasyon yöntemini kullanarak hesaplanır ve diğer düğümlere gönderilir. Trilaterasyon algoritmasında elde edilen tablodaki değişkenlerin değeri Gauss Jordan yöntemiyle hesaplanmaktadır.

```
bool GSolve(double a[][3],uint16_t n,double *x)
{
    int i, j, k, max;
    double t;
    for (i = 0; i < n; ++i) {
       max = i;
       for (j = i + 1; j < n; ++j)
          if (a[j][i] > a[max][i])
             max = j;
       for (j = 0; j < n + 1; ++j) {
          t = a[max][j];
          a[max][j] = a[i][j];
          a[i][j] = t;
       }
       for (j = n; j >= i; --j)
          for (k = i + 1; k < n; ++k)
             a[k][j] -= a[k][i]/a[i][i] * a[i][j];
  }
  for (i = n - 1; i >= 0; --i) {
      a[i][n] = a[i][n] / a[i][i];
      a[i][i] = 1;
      for (j = i - 1; j >= 0; --j) {
         a[j][n] -= a[j][i] * a[i][n];
         a[j][i] = 0;
      }
  }
  for (j=0;j<n;j++)
     x[j] = a[j][n]/2;
  return(TRUE);
}
```

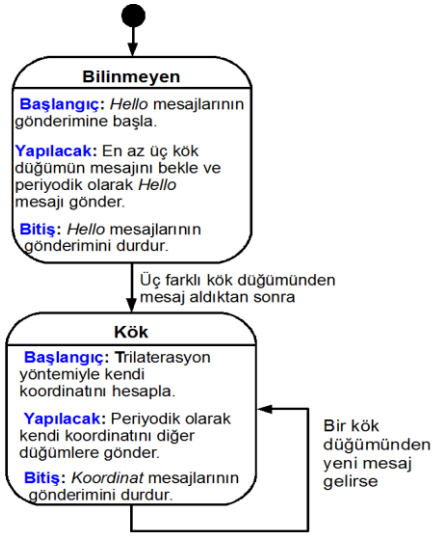
Pozisyonu belli olmayan düğümlerin üzerinde çalışan programın temel algoritması Şekil 7'de gösterilmektedir. Her düğüm ilk başta en az 3 kök düğümü tarafından *POS* mesajının gelmesini bekler. 3 *POS* mesajı geldikten sonra düğüm kendi koordinatını trilaterasyon yöntemiyle hesaplar ve koordinatını diğer düğümlere gönderir. Bu işlemden sonra döngünün başına dönerek yeni *POS* mesajlarının gelmesini bekler ve gelen her mesajın bilgilerine göre kendi koordinatını güncelleyip, yeni koordinatını diğer komşularına gönderir.



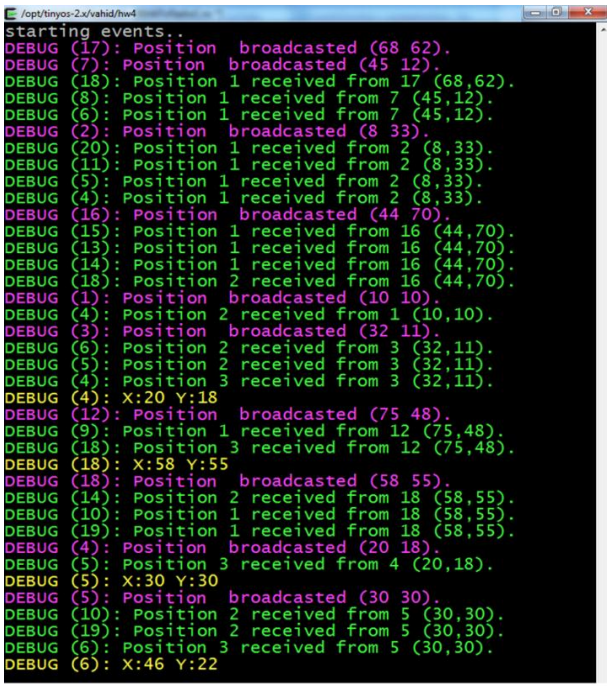
Şekil 7. Düğümlerin temel algoritması
(Base algorithm of nodes)

Bir Bilinmeyen durumunda bulunan düğüm en az 3 farklı kök düğümünden mesaj aldıktan sonra kendi koordinatını hesaplayıp ve kök durumuna geçer (Şekil 8). Kök durumundaki düğümler sürekli olarak diğer düğümlerden gelen mesajların yardımıyla kendi koordinatlarını güncelleyip, yeni koordinatlarını diğer düğümlere gönderirler.

Söz konusu olan programı gerçek düğümlerin üzerinde çalıştırılmadan önce benzetim ortamında da çalıştırabiliriz. TOSSIM benzetim ortamında bir benzetimi başlatmak için bir driver programı, C++ veya Python dilinde yazılabilir. Bu driver programı sanal bir topoloji oluşturarak, TinyOS programını çalıştırır ve topolojideki düğümlerin arasında, mesaj gönderimlerini gerçekleştirir. Şekil 3'deki topoloji için yapılan benzetimin çıktısı Şekil 9'da gösterilmektedir. Şekil 9'daki benzetim başlangıcında, 17 ve 7 numaralı düğümler, kök düğüm olarak, kendi koordinatlarını komşularına gönderirler.



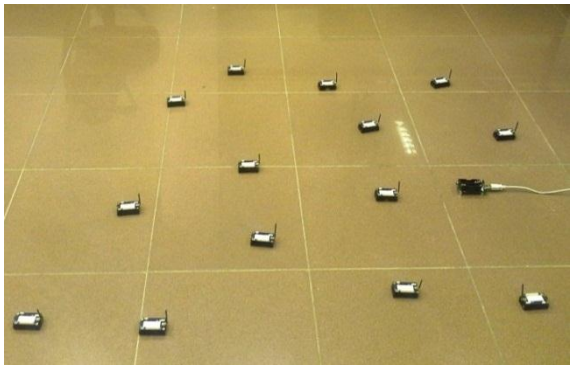
Şekil 8. Düğümlerin durum diyagramı
(State diagram of nodes)



Şekil 9. TOSSIM benzetim ortamının örnek çıktısı
(Sample screenshot of TOSSIM simulator)

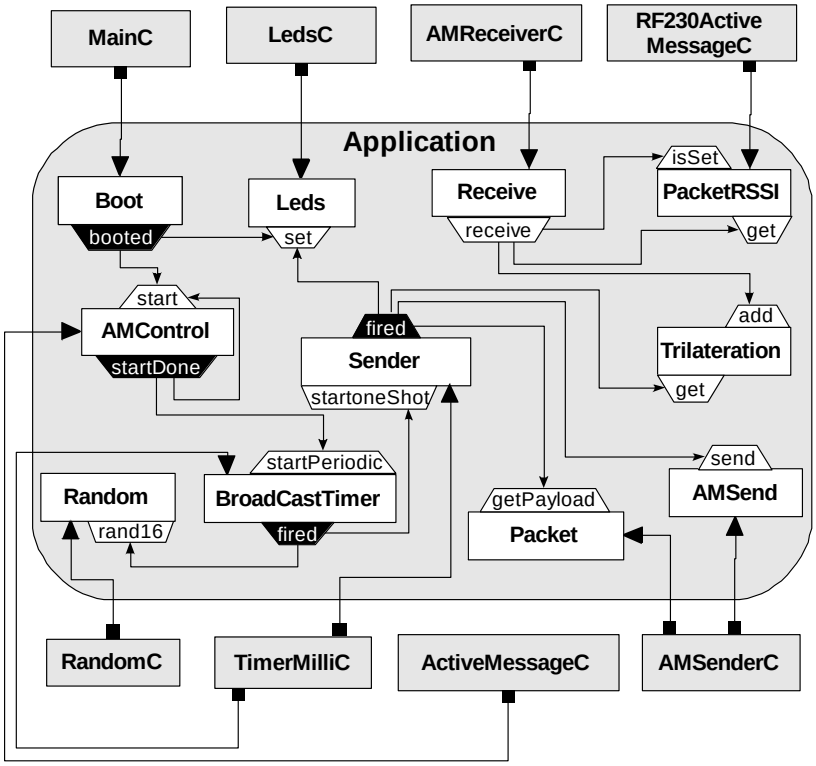
Şekil 9'da 17 nolu düğümün koordinatını 18 nolu düğüm almıştır. 7 nolu düğümün koordinatını 8 ve 6 nolu düğümler almıştır. 2 nolu düğümün komşularına gönderdiği koordinat, 20, 11, 5 ve 4 nolu düğümler tarafından alınmıştır. 16 nolu düğümün komşularına gönderdiği koordinat, 15, 13, 14 ve 18 nolu düğümler tarafından alınmıştır. 1 nolu düğümün sadece bir komşusu olduğu için komşularına gönderdiği koordinat 4 nolu düğüm tarafından alınmıştır. Böylece 4 nolu düğüm ikinci mesajını almıştır. 3 nolu düğüm kendi koordinatını komşularına gönderir, 6 ve 5 nolu düğümler ikinci mesajlarını alacak, 4 nolu düğüm üçüncü mesajını alacaktır. 4 nolu düğüm üçüncü mesajını aldıktan sonra, kendi koordinatını bulur. Daha sonra 12 nolu düğüm kendi koordinatını komşularına gönderir. Geri kalan işlemler Şekil 8'de gösterildiği gibi benzer bir şekilde devam eder.

Uygulanan sistem, 15 adet MEMSIC-CROSSBOW IRIS düğümlerinin üzerinde test edilmiştir. Tüm kök düğümlerin koordinatları statik olarak programın içinde tanımlanmıştır. Sink düğümü bilgisayara bağlanarak, düğümlerin arasındaki iletilen mesajları dinleyip, bilgisayarda çalışan bir sonraki bölümde anlatılan Java uygulamasına gönderir. Şekil 10'da kullanılan düğümler ve sink düğümü gösterilmiştir.

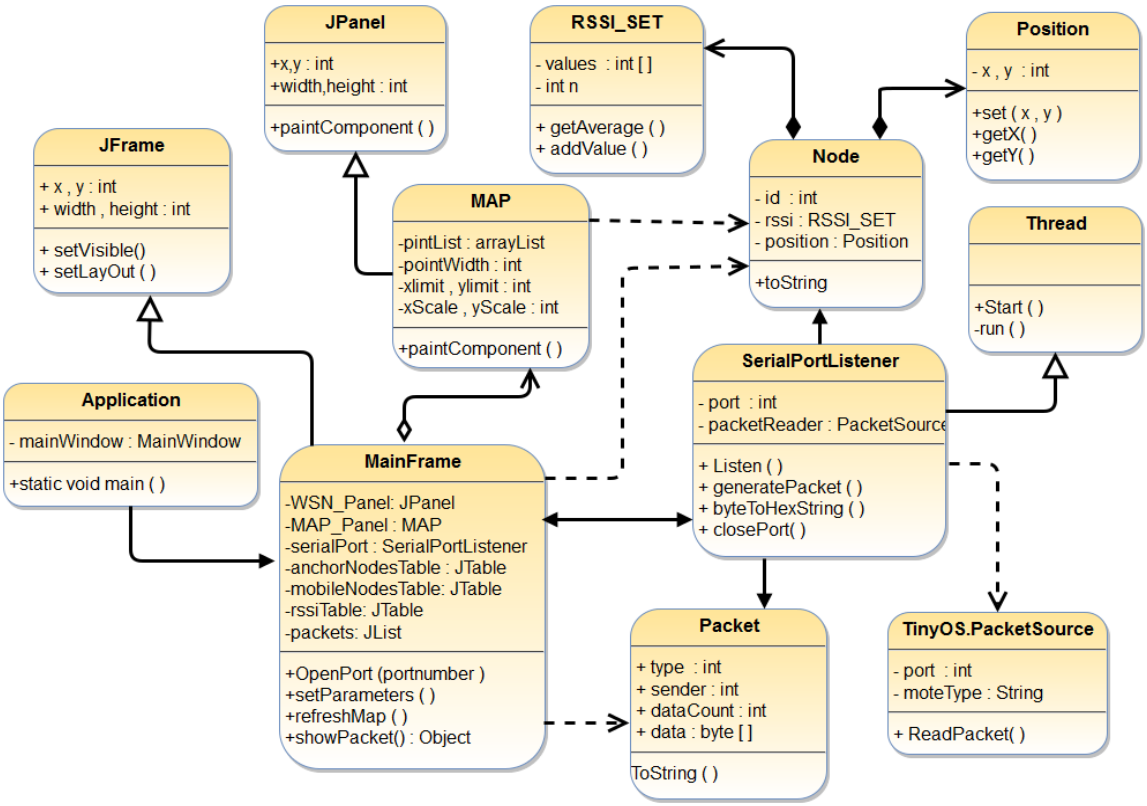


Şekil 10. IRIS düğümleri ve çıkış düğümü
(IRIS nodes and a sink node)

Uygulanan sistemin TinyOS bileşenlerinin ve onların arasında yapılan bağlantılar Şekil 11'de gösterilmektedir. Bu şekilde bileşenler gri renginde dikdörtgenlerle gösterilmektedir. Konumlandırma sistemine uyguladığımız *Application* isimli bileşen şeklin ortasında çizilmiştir. *Application* bileşeninin etrafında görünen diğer gri dikdörtgenlerin hepsi TinyOS işletim sisteminde hazır bulunan bileşenlerdir ki *Application*'da kullanılan arayüzlere bağlanarak, uygulamanın düğüm üzerinde çalışmasını sağlamaktalardır. Düğümlere gelen mesajlar *AMReceive* bileşeninin tarafından *Receive.receive* fonksiyonuna gönderilir. Bu fonksiyonda gelen mesajın tipine göre mesajın RSSI değeri *PacketRSSI* arayüzü aracıyla bulunur. IRIS düğümlerde gelen mesajların RSSI değeri *RF230ActiveMessage* bileşeni tarafından bulunmaktadır. Bu nedenle *PacketRSSI* arayüzü bu bileşene bağlanmıştır. *PacketRSSI*'da *isSet* ve *get* fonksiyonları bulunmaktadır. *isSet* gelen mesajda RSSI değerinin mevcut olmasını kontrol eder ve *get* fonksiyonu gelen mesajın RSSI değerini elde eder. Gelen *POS* mesajlarının RSSI değerleri elde edildikten sonra eğer en az 3 adet kök düğüm mesajı gelmişse *Trilateration.get* fonksiyonu çalıştırılır ve düğümün koordinatları hesaplanır.



Şekil 11. Uygulanan TinyOS programının bileşenleri
(Main components of implemented application)

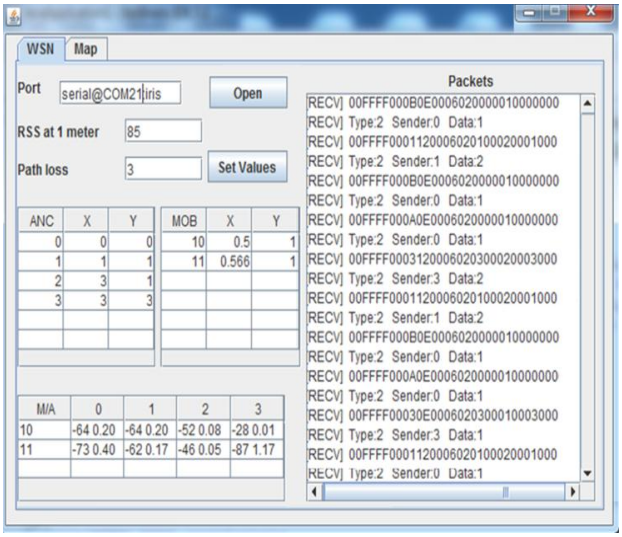


Şekil 12. Uygulanan Java arayüzünün sınıf diyagramı
(Main objects of implemented java interface)

3.3 Tasarlanan Arayüzü *(Developed User Interfaces)*

Düğümlerin gönderdiği mesajları, bulduğu koordinatları ve ağın bağlılığını görmek için bir Java uygulaması geliştirilmiştir. Bu yazılım bilgisayara bağlanan çıkış düğümüyle USB port üzerinden iletişim kurarak ağda gönderilen tüm paketleri algılayıp, toplanan bilgileri uygun arayüzlerinde göstermektedir. Bu yazılımın sınıf diyagramı Şekil 12'de gösterilmektedir. Bu şekilde *MainFrame* sınıfı uygulamanın arayüzüdür. Bu sınıfın örnek çıktısı Şekil 13'de gösterilmektedir. Pencerenin sağ tarafında bulunan kutuda ağda gönderilen tüm paketler gösterilir. Bu kutuda paketlerin boyutları ve ayrıca içindeki veriler gösterilir. Sol tarafta bulunan tablolarda kök ve bilinmeyen düğümlerin id'leri, koordinatları ve birbirinden aldıkları RSSI değerleri gösterilmektedir. Kullanıcı girdi kutuları aracıyla RSSI değerini mesafeye çevirme formülünde istediği değişiklikleri yapabilir. Pencerenin diğer sekmelerinde düğümlerin koordinatları ve aralarındaki bağlantılar harita ve çizge şekillerinde görünebilir.

MainFrame sınıfı ağda gönderilen paketleri algılamak için *PortListener* isimli bir sınıfı kullanmaktadır. *PortListener* sınıfı çıkış düğümünün port numarasını alır ve bu porttan gelen tüm verileri hem bayt ve hem paket olarak *MainFrame* sınıfına teslim eder. Bu sınıf *Thread* sınıfını genişleterek bir *Thread* olarak çalışır ve bu yüzden yazılım aynı zamanda hem gelen paketleri ve hem de kullanıcının komutlarını algılayabilir. Çıkış düğümü tarafından gönderilen paketler, *TinyOS.PacketSource* sınıfı aracıyla porttan okunur ve gelen paketin bilgilerine göre bir *Node* veya bir *Packet* nesnesi üretilir. Bu nesneler *MainFrame* sınıfına gönderildikten sonra gerekli tablolar güncellenir.



Şekil 13. Ağda gönderilen paketler
(Sent messages in the network)

```
public class PortListener   extends   Thread {
  private PacketSource reader;
  private Frame1 window;
  private static final int AM_TYPE = 6;

  public PortListener(Frame1 frm, String[] args) {
      window = frm;
      String source = null;
      try {
         if (args.length == 2 && args[0].equals("-comm")) {
            source = args[1];
         }
         else if (args.length > 0) {  System.exit(2);   }
         if (source == null) {
            reader = BuildSource.makePacketSource();
         } else {
            reader = BuildSource.makePacketSource(source);
         }
         if (reader == null) {
            System.exit(2);
         }
         reader.open(PrintStreamMessenger.err);
      } catch (Exception ex) {
          window.display(ex.toString());
      }
  }

  public void Close() {
        this.stop();
        reader.close();
  }
  public void run() {
      try {
         Packet p;
         while (true) {
             byte[] packet = reader.readPacket();
             p = generatePacket(packet);
             window.display("[RECV]  " + p.toString());
             window.processMessage(p);
           }
         }
        catch (Exception e) {
             window.display(e.toString());
        }
   }
}
```

PortListener sınıfında, *PacketSource* sınıfından bir referans tanımlanmıştır. Gelen parametrelerde USB portunun numarası ve cihazın adı bulunmaktadır. Bu port *reader.open* fonksiyonunu çağırarak açılır. USB portu açıldıktan sonra *PortListener* nesnesi bir *Thread* olarak çalışmaya başlar. Bu sınıfın *run* fonksiyonunda sonsuz bir döngü içinde *reader.readPacket()* fonksiyonundan gelen mesaj, porttan bayt dizisi şeklinde alınır. Gelen mesaj *generatePacket* fonksiyonuyla *Packet* şekline dönüştürülür ve metin olarak *display* fonksiyonuyla ekrana yazılır. Ayrıca *proccessMessage* fonksiyonu, mesajı gönderen düğümün bilgilerini, tablolara ekledikten sonra, düğümü bir iki boyutlu alanda ve uygun koordinatta çizer.

Packet sınıfında düğümlerin bilgileri *MainFrame* sınıfına gönderilir. Her *Packet* nesnesinde gönderici düğümün id'si, düğümün koordinatı, mesajın tipi ve gereken ekstra bilgiler tutulmaktadır. Bu sınıfın yapısı aşağıdaki gibidir:

```
public class Packet {
   public byte type;
   public byte sender;
   public int data;
   public int X,Y;
   public Packet(){}
```

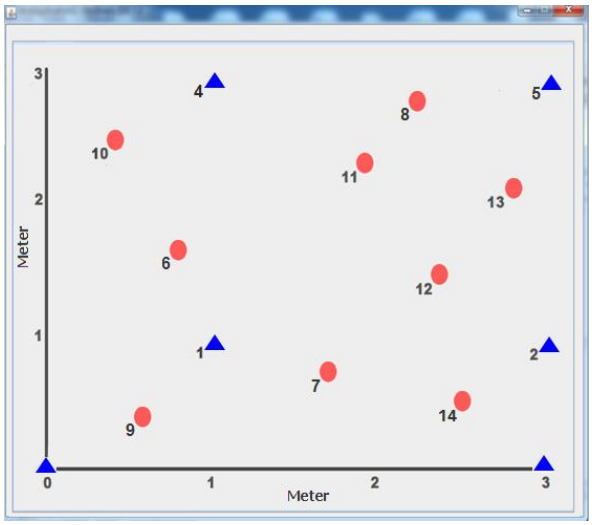
```
    public Packet(byte type,byte sender,int data,int x,int y){
        this.type=type;
        this.sender=sender;
        this.data=data;
        this.X=x;
        this.Y=y;
    }
}
```

MainFrame sınıfında mesajları işlemek için *processMessage* metodu yazılmıştır. Bu metod düğümlerin koordinatlarını alanda göstermek için refreshMap metodunu çağırır. Bu metodlar aşağıda verilmiştir.

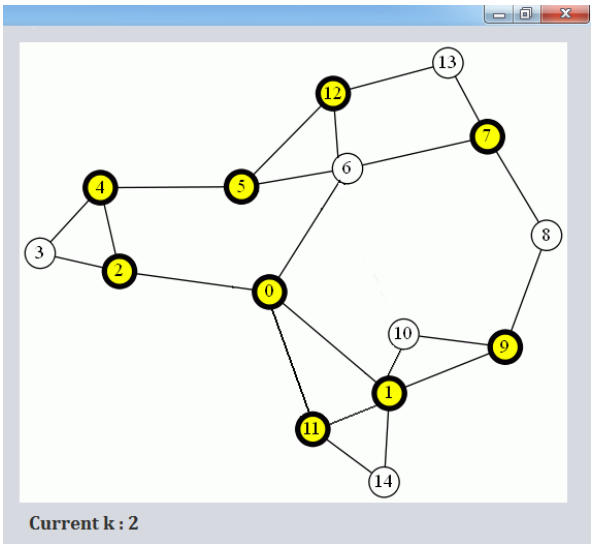
```
public class MainFrame extends javax.swing.JFrame {

    PortListener serialPortGateWay;
    float RSSIAtOneMeter, pathloss;
    DrawGraph myMap;
    private HashMap RSSITable;
    public Frame1() {
        initComponents();
        RSSIAtOneMeter=Float.parseFloat(jTextField3.getText());
        pathloss=Float.parseFloat(jTextField4.getText());
        RSSITable=new HashMap();
    }

    synchronized void processMessage(Packet pkt) {
        if (pkt.type == 2) {
            if (jTable1.getValueAt(pkt.sender,0) == null) {
                jTable1.setValueAt(pkt.sender, pkt.sender, 0);
                jTable1.setValueAt(pkt.X, pkt.sender, 1);
                jTable1.setValueAt(pkt.Y, pkt.sender, 2);
                jTable1.setValueAt(pkt.Z, pkt.sender, 3);
            }
            refreshMap ();
        }
    }

    public void refreshMap(){
        if (jTabbedPane1.getSelectedIndex() == 1) {
            jTabbedPane1.getSelectedComponent().repaint();
            this.pack();
        }
    }

    public void  refreshGraph (JLabel label, short Mat[][]) {
        int i, j;
        BeginDraw(false);
         for (i = 0; i < Mat.length; i++)
            for (j = i + 1; j < Mat.length; j++)
                if (Mat[i][j] == 1)
                    graph.add(i + " -> " + j + ";");
         EndDraw(label);
    }
}
```

refreshGraph metodu tablolarda bulunan düğümleri çizge olarak kullanıcıya sunar. Bu metod düğümleri çizmek için *GraphViz* kütüphanesini kullanır. Bu metodun örnek çıktısı Şekil 14'de gösterilmektedir. Ağın *k*-bağlılık değerini ve ağdaki kritik düğümleri bulmak için Henzinger'in algoritması belli zaman aralıkları içinde çalıştırılır. Aşağıda bu işlemin kodu verilmiştir.

```
class Henzinger_K_Finder {
    private short[][] mat;
    public Henzınger_K_Finder (short[][] m) {
        mat = m;
    }
    public int Find_K() {
        int i=0, k=m.length()-1;
        do{
            i++;
            t=MinCut.find(m,i);
            if(t<k)    k=t;
        }while(i>k);
        return k;
    }
}
```



Şekil 14. Düğümlerin koordinatları
(Coordinates of nodes)



Şekil 15. Ağın *k*-bağlı çizge olarak görüntüsü
(Monitoring network as *k*-connected graph)

Şekil 14'de haritanın *X* ve *Y* eksenlerinin limiti gelen paketlerin bilgilerinin üzerinden seçilir ve düğümler kendi

koordinatlarından çizilirler. Bu haritada kök düğümler mavi üçgenlerle ve koordinatlarını sonradan hesaplayan düğümler kırmızı dairelerle çizilmektedir. Koordinatı hesaplanmayan düğümler bu haritada çizilmemektedir ama bu düğümlerin id'leri diğer tablolarda görünebilir.

Şekil 15'de düğümlerin gönderdiği *Hello* mesajlarının üzerinden çizilen bağlantılar gösterilmektedir. Bu çizgede sarı ve daha kalın yuvarlaklarla çizilen düğümler bulunan k değeri için kritik düğümlerdir. Bu kritik düğümlerin bozulması veya hareket etmesi ağın k değerini 1 düşürebilir. Bu çizge çıktısında düğümler rasgele pozisyonlarda çizilmektelerdir. Bu çizge sayesinde kullanıcıya ağdaki kritik düğümler ve ağın hata toleransı görsel olarak sunulur.

3.4 Konumlandırmanın Maliyeti ve Düğümlerin Enerji Tüketimi (Localization Overhead and Energy Consumption)

Tasarlanan sistemde kök düğümler, periyodik olarak kordinatlarını diğer düğümlere göndermektelerdir. Konumunu bilmeyen her düğüm yeterli sayıda kök düğümden mesaj aldıktan sonra kendi kordinatlarını hesaplayabilir veya daha önce hesapladığı kordinatı düzeltir. Düğümler, carrier sense multiple access (CSMA) medium access layer (MAC) protokolünü kullanarak, paketlerini göndermeden önce 0 ile 500 ms arasında bir rasgele sayı çekip, bu sayı kadar bekledikten sonra paketlerini gönderirler. Bu protokol gönderilen paketlerin arasında bir çakışma ihtimaline yol açabilir. Ancak düğümler periyodik olarak mesajlarını rasgele zamanlarda gönderdiklerinden dolayı, ilerleyen periyotlarda yeni mesajlar alındığından düğümlerin konumlandırması gerçekleşir. Kök düğümlerde mesaj gönderimlerinin periyotları, düğümlerin enerji tüketimi ve konumlandırmanın hassasiyetini doğrudan etkilemektedir. Kısa gönderim periyotları, konum mesajlarının çakışma ve düğümlerin yer değiştirme etkisini azaltır ve hızlı hareket modellerinde bile daha güncel ve doğru konumlandırmaya yardımcı olur. Ancak, kısa süreli aralıkla mesaj gönderme, düğümlerin hızlı enerji tüketimine sebep olabilir.

Konumlandırma maliyetini periyodik olarak incelemek için düğümlerin enerji tüketimlerini kabaca hesaplayabiliriz. Duyarga düğümleri pille çalıştıklarından dolayı enerji tüketimleri önemli bir faktördür. Enerji tüketimini belirleyen en önemli etkenin radyo haberleşmesi olduğu bilinmektedir [8]. IRIS düğümlerinin veri sayfasına göre [28], 3.3 v güç kaynağıyla beslenen bu düğümler, 3 dBm radyo gücüyle çalışma durumunda, mesaj gönderirken 17 mA ve mesaj alırken 16 mA akı almaktalardır. Bu düğümlerin iletişim oranı 250 kbps (31.25 kBps) olduğu için 8 baytlık konumlandırma mesajlarını yaklaşık 0.25 ms'de gönderilebilirler. Böylece, enerji tüketiminin genel ilişkisi $E = V * I$'yi dikkate alarak, bir konumlandırma mesajı, gönderici tarafta $3300 * 17 * 0.00025 = 14$ mj ve alıcı tarafta $3300 * 16 * 0.00025 = 13.2$ mj enerji tüketir. Dolayısıyla d adet komşusu olan bir düğüm her periyotta yaklaşık $14 + (d * 13.2)$ mj enerji tüketecektir. Bu sebeple, mesaj gönderimlerinin periyotları, konumlandırmanın hassasiyeti ve aynı zamanda ağın toplam enerji tüketiminde önemli bir etkisi olduğundan dolayı bu sürelerin dengeli bir şekilde seçilmesi gerekmektedir. Uygulanan sistemde, kök düğümler her 20 s'de bir konum mesajı göndermektelerdir. Düğümlerin hızlı hareket ettiği başka bir ağ için konumlandırma kalitesi çok önemli bir parametre ise periyot değeri düşük tutulmalıdır.

4. SONUÇLAR (CONCLUSIONS)

Bu çalışmada TDAlar için bir konumlandırma ve k-bağlılık denetleme sisteminin tasarımı ve uygulaması anlatılmıştır. Uygulanan sistemde düğümler ağda bulunan kök düğümlerin yardımıyla kendi koordinatlarını bulup, diğer düğümlerin konumlandırmasında yardımcı olur. Ayrıca kullanıcı hazırlanan Java yazılımıyla düğümlerin güncel pozisyonları ve ağın bağlantı durumunu öğrenebilir. Geliştirilen sistemde düğümlerin arasında mesafe ölçme işlemi, alınan mesajların RSSI değerinden ve lokalizasyon işlemi Trilaterasyon algoritmasıyla gerçekleştirilmektedir. Ağın k değeri gelen güncel mesajlar ve Henzinger algoritmasıyla sürekli kontrol edilir. Uygulanan tüm yazılım bileşenleri nesne tabanlı yöntemiyle geliştirildiği için, önerilen sistem bir altyapı olarak farklı amaçlara göre özelleştirilip, sisteme yeni işlevler eklenebilir.

TEŞEKKÜR (ACKNOWLEDGMENTS)

Yazarlar 113E470 numaralı proje kapsamında verilen araştırma desteğinden dolayı TÜBİTAK'a teşekkür eder.

KAYNAKLAR (REFERENCES)

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam and E. Cayirci, "A Survey on Sensor Networks", IEEE Communications Magazine, 40(8):102-114, 2002.

[2] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey", Comput. Netw. 52, 12 (August 2008), 2292-2330.

[3] E. Biagioni and K. Bridges, "The Application of Remote Sensor Technology to Assist the Recovery of Rare and Endangered Species", Special issue on Distributed Sensor Networks for the International Journal of High Performance Computing Applications, 16(3): 315-324, 2002.

[4] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson, "Wireless Sensor Networks for Habitat Monitoring", Proceedings of ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'02), Atlanta, 88-97, 2002.

[5] S. Dilek, and S. Özdemir, "Sağlık Hizmetleri Sektöründe Kablosuz Algılayıcı Ağları", internatıonal journal of informatıcs technologies, 7.2, 2014.

[6] T. Alhmiedat and S. H. Yang, "A Survey: Localization and Tracking Mobile Targets through Wireless Sensors Network", PGNet, 2007.

[7] A. Jagoe, "Mobile location services: The definitive guide", Prentice Hall Professional, 1, 2003.

[8] H. Karl, A. Willig, Protocols and architectures for wireless sensor networks, Wiley 2005, ISBN 978-0-470-09510-2.

[9] J. C. Liberti and T. S. Rappaport, "Smart Antennas for Wireless Communications: IS-95 and Third Generation CDMA Applications", Upper Saddle River, Prentice- Hall, 1990.

[10] N. B. Priyantha, A. K. Miu, H. Balakrishnan and S. Teller, "The cricket compass for context-aware mobile applications", Proceedings of the 7th annual international conference on Mobile computing and networking, 1-14, ACM, 2001.

[11] J. Kang, D. Kim, Y. Kim, "RSS self-calibration protocol for WSN localization", The 2nd International Symposium on Wireless Pervasive Computing ISWPC '07, Puerto Rico, 2007.

[12] M. Henzinger and G. N. Harold, "Computing vertex connectivity: New bounds from old techniques", Journal of Algorithms, 34.2: 222-250, 2000.

[13] A .J. Hao and J. B. Orlin, "A faster algorithm for finding the minimum cut in a directed graph", Journal of Algorithms, 17: 424-446, 1994.

[14] S. Even, "An algorithm for determining whether the connectivity of a graph is at least k", SIAM Journal on Computing, 4(3): 393-396, 1975.

[15] K. Langendoen, and N. Reijers, "Distributed Localization in Wireless Sensor Networks: A Quantative Comparison", Elsevier Computer Networks, 43: 499-518, 2003.

[16] H. Lee, H. Dong, and H. Aghajan, "Robot-assisted Localization Techniques for Wireless Image Sensor Networks", Proceedings of IEEE Conf. on Sensor, Mesh, and Ad Hoc Communications and Networks (SECON), 1: 383-392, 2006.

[17] V. Ateş and M. A. Akcayol, "Kablosuz Ağlarda Tahmine Dayalı Hücreler Arası Geçiş Algoritmaları", International journal of informatics technologies, 3.3, 2010.

[18] D. Niculescu, and B. Nath, "Ad-hoc positioning system", Global Telecommunications Conference 2001, GLOBECOM'01, IEEE, 5: 2926-2931, 2001.

[19] P. N. Pathirana, N. Bulusu, A. V. Savkin, and S. Jha, "Node Localization Using Mobile Robots in Delay-Tolerant Sensor Networks", IEEE Transactions on Mobile Computing, 4(3):285-96, 2005.

[20] C. Savarese, K. Langendoen, and J. Rabaey, "Robust positioning algorithms for distributed ad-hoc wireless sensor networks", Proceedings of USENIX Technical Annual Conference, 317–328, 2002.

[21] A. Savvides, H. Park, and M. Srivastava, "The Bits and Flops of the N-hop Multilateration Primitive for Node Localization Problems", Proceedings of First ACM International Workshop on Wireless Sensor Networks and Application (WSNA), 112–121, 2002.

[22] Y. Shang, M. P. J. Fromherz, W. Ruml and Y. Zhang, "Localization from Mere Connectivity", Proceedings of International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc), 201-212, 2003.

[23] L. Chen, R. Yan, and Z. Ma, "TinyOS-based localization system design using accelerometer", 15th IEEE International Conference on Communication Technology (ICCT), IEEE, 511-518, 2013.

[24] C. Chen, "Design of a child localization system on RFID and wireless sensor networks", Journal of Sensors, 2010.

[25] N.P. Pathirana, B. Nirupama, V.S. Andrey and J. Sanjay, "Node localization using mobile robots in delay-tolerant sensor networks", IEEE Transactions on Mobile Computing, 4(3): 285-29, 2005.

[26] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler and K. Pister, "System Architecture Directions for Networked Sensors", Proceedings of the ACM ninth international conference on Architectural support for programming, 93-104, 2000.

[27] L. Doherty, K. S. J. Pister and L. E. Ghaoui, "Convex Position Estimation in Wireless Sensor Networks", Proceedings of IEEE INFOCOM, 3: 1655-1663, 2001.

[28] M. Inc, Iris datasheet, http://www.memsic.com/userfiles/files/datasheets/wsn/iris datasheet.pdf.