國立交通大學

資訊工程學系

碩士論文

以無線感測器網路實作室內安全監控以及
緊急逃生導引系統

Using Wireless Sensor Networks for Indoor Security Monitoring and Emergency Navigating System

研究生：蔡岳洋
指導教授：曾煜棋　教授

中華民國九十四年六月

以無線感測器網路實作室內安全監控以及緊急逃生導引系統

Using Wireless Sensor Networks for Indoor Security Monitoring and Emergency Navigating System

研 究 生：蔡岳洋　　Student：Yuen-Yung Tsai

指導教授：曾煜棋　　Advisor：Yu-Chee Tseng

國 立 交 通 大 學

資 訊 工 程 系

碩 士 論 文

A Thesis

Submitted to Department of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer and Information Science

June 2005

Hsinchu, Taiwan, Republic of China

中華民國九十四年六月

以無線感測器網路實作室內安全監控以及緊急逃生導引系統

學生：蔡岳洋

指導教授：曾煜棋

國立交通大學資訊工程學系（研究所）碩士班

摘　　要

近年來無線感測器網路被各研究單位廣泛地討論，無線感測器能夠提供人們多樣化之應用與服務，本論文利用無線感測器網路實作一套室內安全監控以及緊急急難導引系統，本系統擁有簡易網路建置方式、可靠拓樸重建以及安全導引人員逃生三項特點。在網路部署完成後，我的系統可以用於監控室內環境狀態，而當網路有緊急事件發生時，網路拓樸可能會因為被破壞性的事件而發生變化，我的系統能夠預先反應並且重建網路拓樸，同時感測器也會依據急難時感測到的資訊來提供安全的逃生路徑協助室內人員逃離災難現場，保障位處於災難現場人員的人身安全。

關鍵詞：無線感測器網路、簡易網路建置、可靠拓樸重建、安全導引

Using Wireless Sensor Networks for Indoor Security Monitoring and Emergency Navigating System

student：蔡岳洋

Advisors：Dr. Yu-Chee Tseng

Department（Institute）of CSIE
National Chiao Tung University

ABSTRACT

In the recent year, wireless sensor network has been widely discussed by many researchers. Wireless sensor network can provide many kinds of applications and services. In my paper, I use wireless sensor networks to implement an indoor security monitoring and emergency navigating system. It has a simple way to deploy the sensor networks, reliable topology reconstruction and navigation. After deployment, my system can be used to monitor the indoor environment. When there are emergencies in the network, the network topology may be destroyed, and my system can react to this situation previously. At the same time, the sensors will provide some navigation instructions for users to escape from the dangerous scene according to the sensed information. Hence, protect the peoples' lives .

Keywords: Wireless Sensor Network, simple Network Deployment, Reliable Topology Reconstruction, Navigation

誌謝

首先，我要感謝我的指導教授曾煜棋老師。由於老師提供了相當優良的研究環境以及給予我適切的指導，我才能有此研究與發明。再來，我要感謝我的指導學長潘孟鉉。沒有他的幫助與建議，我將無法將此論文完成。最後，我要感謝 HSCC 實驗室的每個成員，他們都曾給予我許多的意見，與我討論並在很多方面給我幫助。

蔡岳洋 2005 年於交大資工

目錄

圖目錄

第一章、簡介

第一節、研究背景

隨著微機電整合、無線網路及嵌入式處理技術的迅速提升，微小的電子裝置可以內嵌精密感測、計算及通訊等多樣化功能。近年來眾多研究單位以及產業界人士相當看好這類微小裝置的未來前景以及其所能夠發展之應用，而其中當屬微型無線感測器(Wireless Sensor)最受大家注目，微型無線感測器能部署於環境當中形成無線感測網路(Wireless Sensor Network)，傳統的 IEEE 802.11 無線隨意網路(Wireless Ad Hoc Network)可說是無線感測器網路的前身，它們皆為無基礎設施(Infrastructure-less)的網路型態，具有許多相通的特性，但無線感測器網路的建構條件更為嚴苛，使得許多現今通行於無線隨意網路上的協定無法直接套用於無線感測器網路上，因此在設計無線感測器網路上至今仍存在著許多嶄新的挑戰，歸納兩種網路主要的差異如下：

- 網路範疇不盡相同：無線感測器網路的節點數量通常較無線隨意網路超出數十倍甚至百倍，而且無線感測器網路的節點密度普遍較高，這點將使得許多利用廣播通訊方式為基礎所發展出來的隨意網路協定將不再適用於感測網路。
- 缺乏共同識別碼：由於無線感測器網路的節點數量龐大，因此每一個網路節點或許沒有唯一性參考指標，用來作為網路中的共通識別碼(identifier)，這點使得無線感測器網路的繞徑協定設計不同於無線隨意網路。
- 裝置能力明顯不同：感測器的硬體設計以成本、體積與耗電為主要考量，因此在無線感測器網路中，裝置節點本身所配備的電源、記憶體以及運算能力均受到極大的限制，這些限制相對於無線隨意網路中的行動主機來說，受限制的幅度則輕緩許多，因此無線感測器網路是被歸類為一個以任務導向的應用網路型態。

即使無線感測器網路有諸多限制，但我們還是可以在這一個網路平台上延伸出多樣化的應用。其一主要之應用為透過感測器在其各自的感應範圍內偵測週遭環境或特定目標，並且將所收集到的資訊經由無線傳輸之方式回報給網路監控者，網路監控者得知環

境中的狀態後能夠利用這些訊息來自我調整與維護或提供所設計之進階系統服務，如美國加州柏克萊大學所建構之野鳥生態監測系統[7]就是屬於此類應用。國內外的研究單位也有許多其他之無線感測器網路的相關應用正持續地在發展中，其應用範疇相當地廣大，涵蓋了商業、軍事、教育與工業等諸多方面，在醫療方面，可應用在特殊疾病病人的監控，可防止疾病的擴散且有效的進行控管，而在軍事上，可將感測器散佈在敵方之中，可監測敵人行動進而指揮部隊行動。

除了上述之應用外，在未來無線感測網路之應用也可深入人類日常生活之中。例如我們可以利用無線感測器網路來調控室內燈光，當感測器偵測到有人在房間時，會自動將房間的電燈打開，當房間裡頭沒有人的時候可以將房內燈光調暗或是關閉。另一個廣為大家討論的應用為利用感測器網路來偵測某個展覽會場參觀者的位置，藉此提供位置感知的服務(Location Aware Services)，當參觀者走到某個展品前，參觀者手上的 PDA 即可秀出該展品的資訊解說。當然無線感測器網路還有很多未開發的應用環境，由小而精巧的無線感測器所建構之網路具有無限的想像空間。

第二節、系統概述

在本篇論文裡，我使用無線感測器網路之概念來建製用於室內安全偵測以及緊急急難情況之應用、提出了幾項新穎的設計方法，並且使用 MICAz Mote 無線感測器平台來實作，本系統有下列之特色：

- 簡易的網路建置：我們將使用 JAVA 語言來實作網路建置以及蒐集回報資訊介面，網路建置者將建築物架構圖讀入介面中來規劃無線感測器擺放之位置，並且事先規劃可以行走之路徑作為爾後緊急急難發生時建築物內人員逃生路徑。

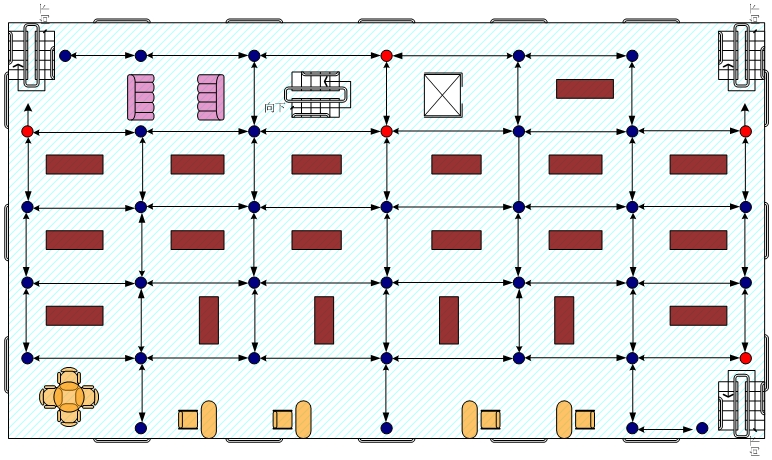


圖 1：網路建置功能

- 安全監控以及維護：當感測器部署完後，系統將會由監控室發出 Beacon 生成展延樹(Spanning tree)，在網路無緊急事件發生時，無線感測器使用展延樹定期回報所監控之區域之情況。監控室亦能追蹤回報之狀況來偵測有無節點發生故障，監控人員及能夠即時維護，保持整個網路正常運作。

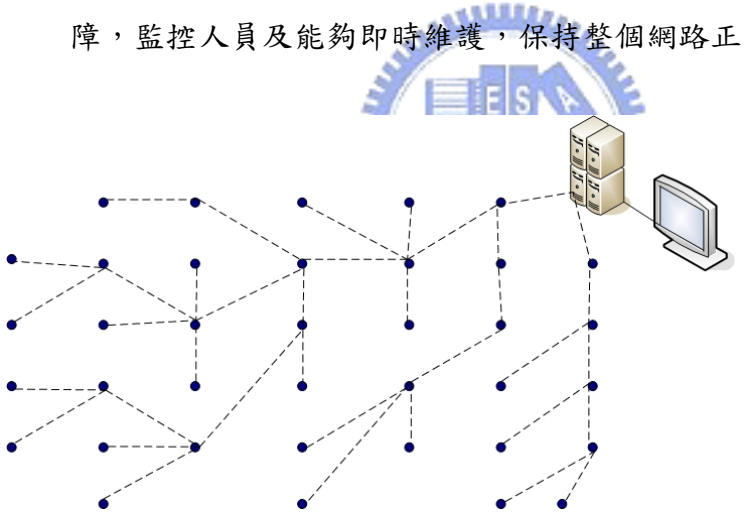


圖 2：回報拓樸

- 緊急逃生導引：當網路上的感測器偵測到有緊急事件發生時，感測器發佈緊急訊息封包來通報網路中其他感測器，感測器收到緊急訊息封包後將會閃爍燈號來指引建築物內部人員逃生，所指引的逃生方向將會避開緊急事件發生的位置以及其鄰近區域，保障人員能夠安全地逃離急難現場。

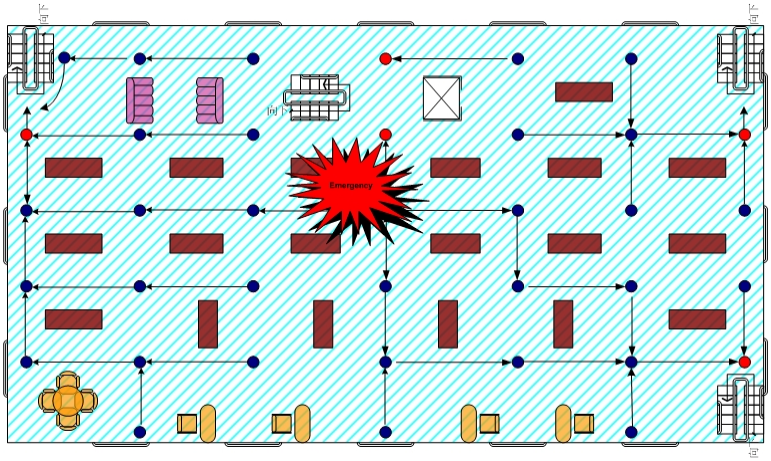


圖 3：緊急導引

■

我們將在下面的章節針對設計方法做詳盡之介紹，我們期許我們所提出之應用平台可以在不久的將來被廣泛的使用，造福廣大民眾。

第三節、動機

根據行政院內政部消防署的統計民國九十二年全台共發生 8642 場火災，造成 228 人死亡以及 767 人受傷，當火災這類的緊急事件發生時，人們往往因為驚慌而不知道該往何處逃生，最常見的錯誤逃生方式有(1)原路逃生：這是因為人們進入了一個不甚熟悉的建築物，因此當緊急情況發生時就選擇循原路逃離、(2)向著光源逃生：向著光源走是生物的基本本能，但是在火場內向著光源走也許意味著進入一個更大的火警區域、(3)盲目跟隨：跟著大眾一起逃生並不一定意味著會找到安全的逃生出口、(4)冒險跳樓等，因為這些錯誤的逃生方式而造成無謂之人員傷亡，也間接造成救護人員救災的危險性。

有鑑於此，本論文研究的主要目的為利用無線感測器網路之特性實作一室內安全監控以及緊急逃生系統之系統雛形，在平時，系統中無線感測器監控建築物有無異常之事件發生，例如溫度過高、瓦斯毒氣外洩等事件，當所監控的網路有緊急事件發生時，無線感測器會自動的標示逃生路徑，導引人們遠離危險區域通往安全的出口。除此之外無線感測器也會迅速的通報室內監控中心有緊急狀況發生，再由室內監控中心進一步回報該地的救援機構請求支援。

第三節、論文重要貢獻

為了達到上述之研究目的之功能，本論文研究最重要的課題為：

- 該如何導引民眾逃離災難現場
- 該如何快速反應緊急狀況發生時網路拓樸之變化

因此我們設計了分散式緊急逃生導引演算法(Distributed Emergency Navigation Algorithm)來解決上述的兩個課題。當有緊急情況發生時，分散式緊急逃生導引演算法會依照所感測到的災難程度來標示人員逃生的路徑，例如火災現場，該演算法會將人員導引至非被祝融侵蝕的安全出口，這一個演算法的難處在於如何分散式控制感測器導引人員避開危險區域至最近的逃生出口，而選用分散式的演算法主要是因為當遭遇緊急狀況時感測器可以直接的反應，不需要透過監控中心來指示每個感測器顯示逃生方位，縮短指示人員逃生之時間。

除了上述的演算法外，為了方便網路建置者部署整個建築物，我亦提供圖形化界面的網路建置工具，這一網路建置工具也可以讓網路建置者組態整個網路，當網路建置者在地圖上規劃好網路的架構，之後只需要將感測器置於相對應地點，整個網路也就可以開始運作，並且這一工具也可以用於監控感測器的狀態以及顯示所收集之感測數值。

總結我的貢獻有下列：1. 簡易之感測器部署方式，2. 室內安全監控，3. 安全導引逃生，我相信這一個室內安全監控以及緊急逃生系統的設計，可以造福一般大眾，帶給人們生活更多安全保障。

第二章、系統設計

本論文研究將研製一室內安全監控以及緊急逃生系統，為了達到此目的，研究面臨之主要的課題，條列如下：

- 室內感測器網路建置工具(Network Deployment Tool)
- 分散式緊急逃生導引演算法(Distributed Emergency Navigation Algorithm)

在這一個章節我們將針對上述的二個課題作一個詳盡的介紹。

第一節、室內感測器網路建置

為了使這一套室內緊急逃生導引系統發揮其功能，最重要的一件工作為網路該如何建置。在室內的環境中，常常有許多障礙物存在，像是隔間牆壁等，在導引人員逃生時應該要避免引導人們通過障礙物到達安全出口處。因此使用無線電波的連線(communication link)作為導引的路徑並不合適，因為無線電波可以繞射穿越牆壁，達到跟另一節點通訊的目的，因此無線感應器們的通訊連線並不等於人們可以逃生的路徑，如圖 4。在[5]中，作者設計了一套方法，可以在佈置的同時自動的找到邏輯連線。作者認為，感測器通常是循序佈建的，因此若感測器在佈置完成後短時間內收到其它感測器的封包，則它會視此感測器距離很近，因此與它建立邏輯連線。作者同時提到一種例外，就是當感測器佈建的線路呈現交叉或是繞圈的情況時，因為較晚佈建的感測器並不會視許久之前佈建的感測器為鄰居，造成斷路。針對這種情況，作者提出了修改感測器，在上面增加按鈕來設定感測器的方式來解決問題。

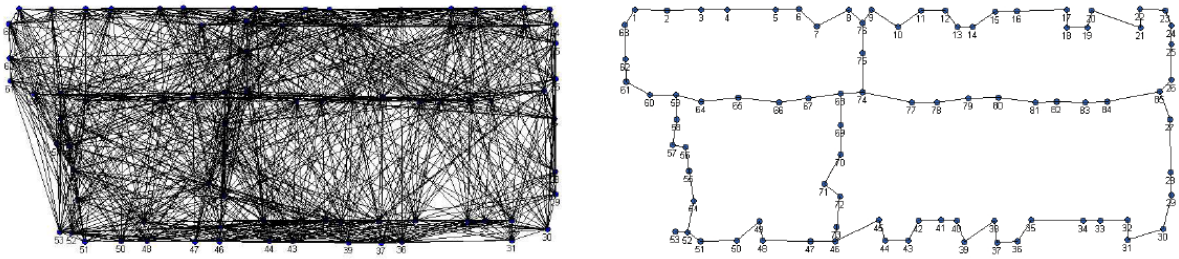


圖 4：通訊連線與邏輯連線

本論文認為[5]的方法，並不適用在我的實作上，主要有兩個問題：第一，需要在網路上新增節點時會無法正確偵測鄰居節點。這問題是由於新增的節點並不是循序增加的。第二，此方法無法得知鄰居的方位。在我們的導引系統中，最重要的就是要顯示逃生方向給人們看，因此必需要知道鄰居的方位。因此在我的實作中，我採用在監控端的電腦規劃邏輯連線的這種作法。我實作出了一個網路部署介面，當管理者決定佈建一個網路時，他首先將建築物平面圖載入此程式，接著在程式中規劃感測器佈建的位置。每當管理者在程式中新增一個感測器節點，程式就會附予此感測器一個獨一無二的識別號碼(ID)。每佈建一個新點，此識別號碼就會累加。在規劃完感測器的位置後，管理者還需要替每個感測器設定一個角色。感測器節點有三種不同的角色。第一種為一般的感測器，負責監測環境、回報數據以及導引；第二種為出口感測器，它佈建在出口的位置上，它除了有一般感測器的功能外，還要負責初始化導引功能；第三種為基地台，基地台不負責監測環境、回報或是導引，它主要的功能是做為電腦端與感測器網路端的閘道器(gateway)。最後，管理員決定感測器間的邏輯連線。這時程式會根據他所設計的連線計算出每個感測器的鄰居節點的方位。當所有的邏輯連線規畫完成後，管理者便依照規劃完成的網路圖到建築物的相對地方佈建感測器。當感測器佈建完成，管理者就從電腦端下達一個初始化網路的動作。流程如下：

- 節點建立 communication link table：在感測器開啟之後，它會定期的搜尋可以連節的網路節點。搜尋的方法採用 two-way handshaking 的協定。採用 two-way handshaking 的原因在於，每個感測器都是獨一無二的，它們的傳輸能力都不盡相同，我們不能將聽到的鄰居節點都當成自己的鄰居。舉例來說，A 節點的傳輸能力比 B 節點的傳輸能力強，若是 B 節點聽到 A 節點的封包後就將它當成自己的鄰居，那麼當 B 節點發送封包給 A 節點時，A 節點是聽不到的。這

就是所謂的不對稱連線(asymmetric link)。因此為了解決這個問題，採用 two-way handshaking 是必需的。另外，在實作上，我們還加入了偵測訊號強度的機制。當接收到的訊號強度低於一定的值時，我們就不會將此節點當成是鄰居節點，因此增強了傳輸的成功率。接下來就是詳細的步驟：

- 感測器隨意地後退一時間點(Backoff)，等待發送 HELLO 封包給它所有的鄰居。
- 當鄰居們聽到 HELLO 封包後，會檢查其訊號強度，若小於一定的值則忽略此封包。否則它們將會回應 ACK 封包告知 HELLO 封包的發送者它們的存在。
- 感測器接收到鄰居傳來的 ACK 封包後同樣也會檢查訊號強度，若訊號強度夠大的話則將鄰居資訊儲存到自身的 communication link Table 中。感測器們的 Neighboring Table 格式如下：

```
typedef struct TableEntry {
        uint8_t id;
        uint8_t goodness;
        uint8_t rssi;
} LinkTableEntry;
LinkTableEntry LinkTable[MAX_LINK];
```

圖 5：Neighboring Table

- 建立展延樹：當初始化開始時，基地台端首先泛洪(Flood) Beacon 封包給網路上所有的感測器。封包中帶有跳躍數(Hop Count)及序號(Sequence number)兩樣資訊。跳躍數代表節點距離根節點的距離。序號代表這是第幾個 Beacon 封包。管理者可以定期發送封包來初始化網路。當感測器收到 Beacon 封包時，會比對發送端識別號碼與本身的 communication link table。若感測器發現發送端不在 table 中則忽略這個封包。因為這個封包是由不對稱連線送過來的。若此封包是由對稱連線送過來的，它會先將封包的跳躍數加 1。接著比對序號及跳躍數。若是新的序號則表示需要重設母節點；若是舊的序號則再進一步比對跳躍數與本身的高度值，若本身高度值大於跳躍數表示需要更新母節點。若需要重設或是更新母節點，則此感測器就將發送端設為母節點，同時更新高度為跳躍數。若感測器有更改過母節點則它會發送一個註冊的封包到電腦端，這個封包

包含了鄰居節點、鄰居節點訊號強度、本身高度值以及母節點識別號碼，如圖 7。接著將這個封包重新廣播到網路上。在感測器初始的時候，高度為無限大。Beacon 格式如圖 6，在 TOSPacket 這個封包中有一個 bit 叫 Beacon，若此 bit 為 1 則代表為 Beacon 封包：

```
typedef struct packet {
    uint8_t UpStream:1,
            DownStream:1,
            Broadcast:1,
            Beacon:1,
            Reserved:4;
    uint8_t src;
    uint8_t org;
    uint8_t dst;
    uint8_t hop;
    uint8_t seq;
    uint8_t len;
    uint8_t data[ PACKET_DATA_LENGTH ];
} __attribute__ ((packed)) TOSPacket;
```

圖 6：Beacon

```
typedef struct registration {
    uint8_t parent;
    uint8_t link_count;
    uint8_t link[MAX_LINK];
    uint8_t rssi[MAX_LINK];
} __attribute__ ((packed)) Registration;
```

圖 7：註冊封包

- 基地台端建立源路由(Source routing)：建立源路由之目的在於網路監控者有可能會更改某些感測器的設定，例如監控者可以加快位處於廚房附近感測器的回報速率。此時則需要直接針對某些節點發佈指示，因此建立原路由是一件必要的工作。建立源路由的方法如下：
  - 電腦端會將註冊封包所傳回的母節點及鄰居記錄在電腦，如圖 8 所示。
  - 電腦端從目的端節點順著母節點往上找到電腦端，將這條路徑反過來即為源路由的路徑。例如要找 5 的源路由，可以從表中查到 5->6->0 再把它反過來成為 0->6->5 就是了。

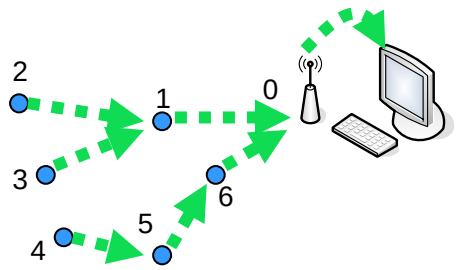


圖 8：鄰居表

ID	Parent	Neighbor
0	7e	1, 6, 7e
1	0	0, 2, 3, 6
2	1	1, 3
3	1	1, 2, 4
4	5	3, 5
5	6	4, 6
6	0	0, 1, 5

- 基地台端發送邏輯鄰居資訊：當基地台端建立完源路由後，基地台隨即將方才在部署感測器時所建立之邏輯鄰居資訊傳遞給感測器，感測器會將邏輯鄰居的方位記錄於 NavigationLink Table 中，當有緊急災難發生時感測器使用此資訊來指示使用者安全的逃生方向。NavigationLink Table 格式如下：

```
struct LinkEntry {
    bool        valid;
    uint8_t id;
    uint8_t     dir;
    uint8_t     role;
    float       Weight;
} LinkTable[MAX_NAVILINK];
```

圖 9：NavigationLink Table

- 決定預設逃生路徑：當所有的感測器接收到邏輯鄰居資訊後，基地台端會要求出口感測器發佈初始導引路徑封包(Initial navigation packet)，這一封包只在邏輯連線上傳送，如果感測器收到非屬於其邏輯連線之感測器所傳送的初始逃生路徑封包而會將它丟棄，網路上之感測器利用封包 Hop Count 來判斷他距離哪一個出口處較近，將它的預設路徑指定到傳送該初始逃生路徑封包的節點。而距離最近出口 Hop Count 也會被視為是該節點的預設權重值 (Default weight)，權重值的用處將在第二節做更詳盡的介紹，初始導引封包的格式如圖 10 所示。

```
enum {
        INIT = 0,
        EMG = 1
};

typedef struct guide {
    uint8_t type;
    uint8_t src;
    bool    tag;
    uint8_t emg;
    uint8_t hop;
    float   weight;
} __attribute__ ((packed)) NaviInfo;
```

圖 10：導引封包格式

完成初始化的動作後，網路隨即進入正常的運作模式，在正常的運作模式底下，感測器們會定時的回報所偵測到的事件，而網路監控者可以透過監控介面來得知所監測網路的狀況，並且監控者也可以針對感測器下達所設計之指令。除此之外，無線感測器也會定期的發送 HELLO 封包，來隨時注意連線的狀態有無發生變化，如果連線狀態發生變化則會即時通報給網路監控者。

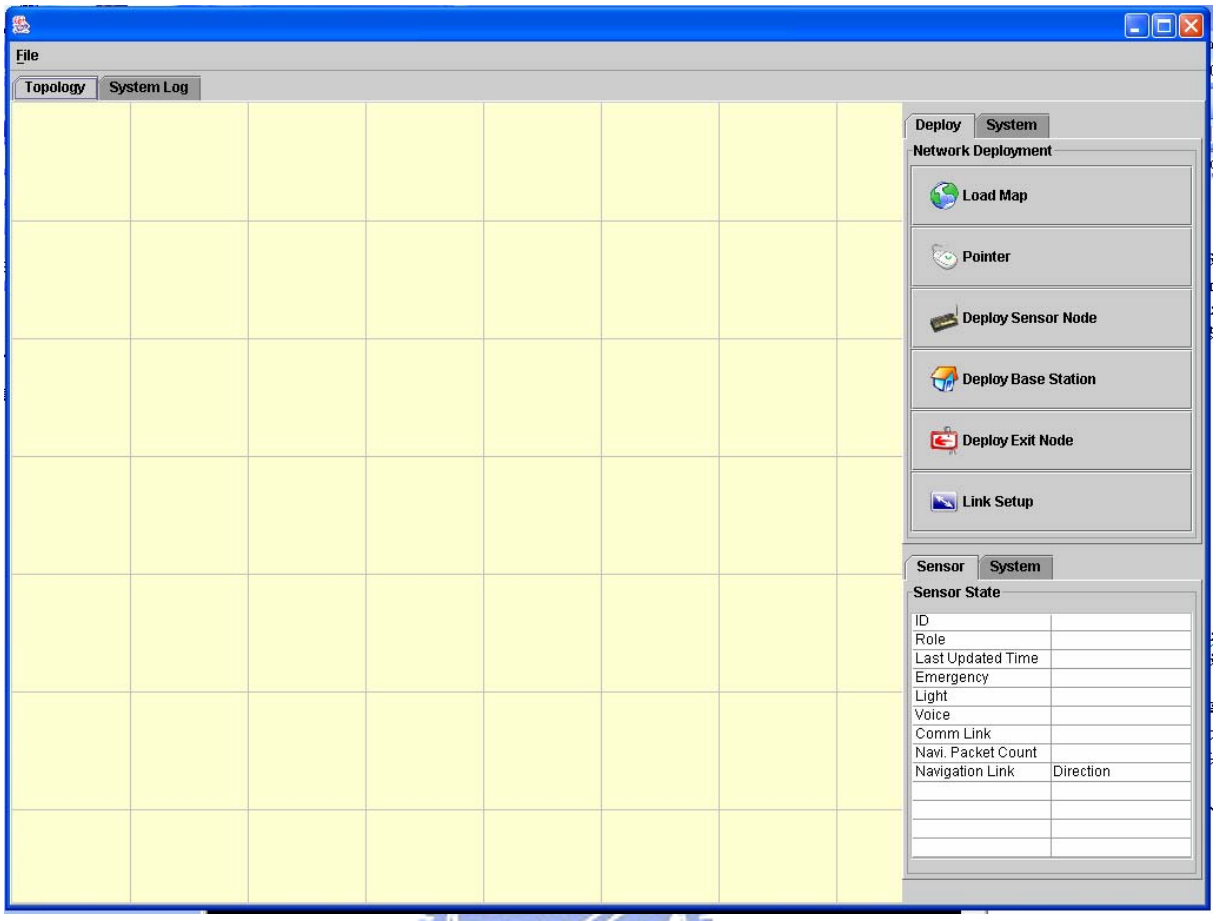


圖 11：網路建置與管理介面圖

第二節、分散式緊急逃生導引演算法

2-1、相關研究

當室內發生緊急急難時，人員的逃生除了要避免經過災難發生處外，也要注意不要接近災難點逃生。在論文裡，我設計了一個能夠有效的導引人員安全逃生演算法。在一篇文獻[3]裡頭，該作者也提出了一套用於導引人員逃離危險區域的演算法，[3]的作者使用吸引力以及排斥力的概念來設計演算法，在的演算法中，當緊急事件發生時發生急難的地點會產生一個排斥力來排斥逃難者接近它，而出口則製造吸引力來吸引逃難者走向它，作者提出一個分散式嘗錯(Heuristic)的演算法來達成其目的。在此演算法中，單一災難施加在節點上的排斥力是以距離的倒數來估算。而出口的吸引力則是以 Hop

Count(Hop count)來估算。每個感測器都會取吸引力最大且排斥力最小的點來做為引導的方向。

由以上的說明可以發現，[3]的作法導引路徑會朝向最近的出口，此做法在單一出口的情況下雖可適用，但在多個出口的情況下往往得不到好的效果，因為前往最近出口的路線未必是最安全的路線，圖 12 就是一個例子。並且因為使用嘗錯的方式來找尋逃生路徑會使用過多的封包交換才能使得該嘗錯演算法收斂。

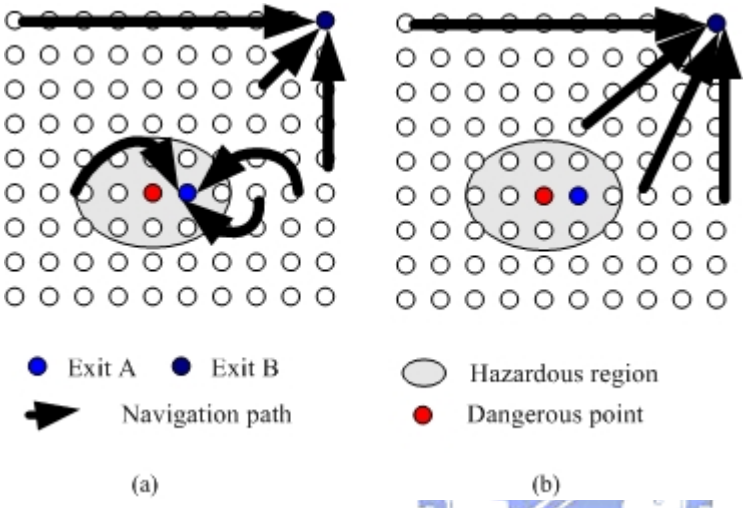


圖 12：不適當的逃生路徑 v.s.較佳的逃生路徑

為了解決[3]所提出之演算法的缺點，我參考了在[4]中所提出想法概念來設計這一分散式緊急逃生導引演算法，在[4]中，作者設計一個分散式多路徑無線隨意網路繞徑演算法(Multipath Ad Hoc Routing Protocol)，作者的概念為當要建立一條由 Source 到某一 Destination 的路由路徑時，由 Source 端發出一個要求建立路由的封包，當 Destination 收到時，則回覆回應封包，此時收到回應封包的節點會自動將自己指定一個權重值(Weight)，當回應封包被轉送時，收到的節點則自動將權重值累加，直到 Source 收到為止，Source 會被指定擁有在這一條路徑最大的權重值，而封包繞送的路徑即由權重高的節點送至權重低的節點，這一個概念類似於大自然河流的流向，河流的流向總是由地勢較高處流向地勢較低窪處，如果多處較低窪的流向河水也會因此而分流。然而無線隨意網路也會因為網路節點的移動性而使得連線中斷，此時繞徑到某個節點的資料也會因此而無法送出，這類似於河流流至一個低窪處但是卻沒能繼續流出而形成湖泊，在[4]的演

算法中，作者使用 Partial Link Reversal 的概念來解決這一個問題，當有連線中斷的情況發生時節點會提升權重值來找出比其權重更為低的路徑來幫忙轉送封包，這概念如圖 13 圖 13 所示：

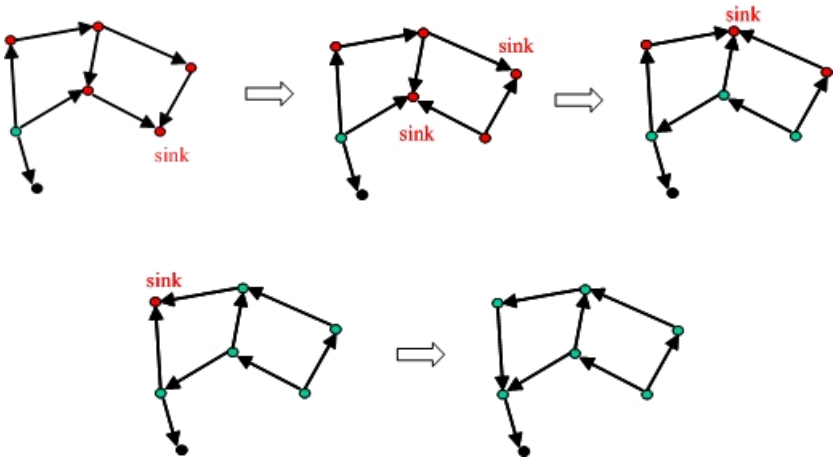


圖 13：Partial Link Reversal

2-2、導引演算法

在我所設計的演算法中也使用了如上所述的觀念。每個網路的節點都會給予一個高度值。當有災難發生時，感測器節點會依照這些災難來適當的調整本身的高度值。導引的方向就從高度值較高的點往高度值較低的點前進。演算法分成兩個階段：初始化階段(initialization phase)和導引階段(navigation phase)。在初始化階段，每個感測器節點會被附予一個初始的高度值。出口節點的高度值最低。其餘節點根據離出口的 Hop Count 不同而得到不同的高度值。當有感測器偵測到緊急災難時，演算法就進入導引階段。在導引階段中，臨近災難點的網路節點會提升本身的高度值來形成危險區域。我的演算法會盡量避免導引使用者穿過危險區。另外，在提升高度的同時，有可能會造成某些區域被包圍住，這時候就要使用如[4]所述的 partial link reversal 來解決。

一、初始化階段

初始化階段的目的在於給予每個節點一個初始的高度值。位於出口處的節點我們稱為出口節點，這些節點廣播初始化封包來啟動這個階段。初始化封包由三部份組成：發送者 ID、出口節點 ID 以及 Hop Count。最初的初始化封包其 Hop Count 為 0。當其餘的

感測器收到初始化封包後，會比對本身的高度值和 Hop Count，取較小的值當做是高度值。接著將 Hop Count 加 1 再重新廣播出去。同時它也會維護一個 NavigationLink Table，內容包含鄰居節點的高度和 ID。在這個階段必需注意兩點：

- 這邊的鄰居指的是 navigation link 上的鄰居，而不是實際 communication 上的鄰居，以後提到的鄰居也是這個意思。
- 為了因應網路拓樸的改變，出口節點每隔一段時間就必需廣播初始化封包來修正路線。

二、導引階段

當偵測到緊急狀況時，導引階段就會開始，其目的是讓每個感測器選出一個高度最小的鄰居當作它的逃生方向。以下，我先介紹一些術語。

D：一個常數。若一感測器與任何急難發生點的距離均小於或等於此一常數，則稱此感測器位在危險區域。以下將使用 Hop Count 來計算距離。

A_{emg}： 指定給偵測到緊急狀況之感測器的一個極大常數(maximum weight)。

Ai：感測器 i 的高度。

I_i：感測器i在初始化階段得到的高度。

$e_{i,j}$：從一個偵測到緊急狀況的感測器i到另一感測器j之間的Hop Count。

EMG 封包：EMG 封包內包含有五個欄位：(1) 事件發生的序列數，(2) 發現緊急狀況的感測器 ID，(3) 傳送者的 ID，(4) 傳送者的高度，(5) 從傳送者到發生緊急狀況之感測器的 Hop Count。EMG 封包可回傳緊急狀況的資料，並幫助感測器調整它們的高度，以達到導引的目的。

當感測器x偵測到緊急狀況時，它會把它自己的高度加上A_{emg}，並廣播一個EMG封包EMG(seq, x, x, A_{emg}, 0)，使此封包傳送的區域擴及整個網路。以下的規則簡單介紹了當感測器y收到感測器w傳送之EMG封包EMG(seq, x, w, A_w, h)後所採取的動作。

1. 首先，感測器 y 透過檢查(seq, x)的方式判斷這是不是新的緊急事件。

   (1) 若是新的緊急事件，則y會記錄這個事件並將$e_{x,y}$設成h+1。

   (2) 反之，則y檢查h+1 是否小於ex,y；若是則y把$e_{x,y}$設成h+1。

   接下來，y把w的高度A_w存在y的鄰居表。如果w=x且x是一個出口感測器，A必須清除它的鄰居表中對應於x的flag is_exit，以避免引導使用者到這個災難點。

2. y依照以下的方法調整高度值。如果$e_{x,y}$在步驟一曾經改變而且$e_{x,y} \le D$，y會將自己視在危險區域內，並以下面的方程式來調整高度：

$$A_y = \max\left\{A_y, A_{emg} \times \frac{1}{(h+1)^2} + I_y\right\} \qquad \text{方程式 1}$$

   在我的設計中，危險區域內的點高度調整的量是跟距離災難點的平方成反比。而考慮I_y的值是因為，同樣在危險區域內的點，離最近出口的距離仍然會不一樣，危險程度也會不一樣。同時我使用了 max來取得在多個災難中所能調整的最高高度。

3. y 接著比較自己與週圍鄰居的高度，若 y 不是出口節點且為週圍區域高度最小的點，y 必需依照以下方程式調整高度：

$$A_y = STA(A_{N_y}) \times \frac{1}{|N_y|} + \min\left\{A_{N_y}\right\} + \delta \qquad \text{方程式 2}$$

   N_y：鄰居的個數。N_y的個數可以代表逃生時可能的路徑。鄰居越少的點表示它的選擇有限因此必需更快的調整高度。

   STA()：鄰居高度的標準差。使用標準差可以快速的增加高度來反應緊急狀況。所以可以較快收斂。

   δ：一個小數值的常數。

4. y會檢查是否符合以下情況，若是符合的話它會重新廣播一個 EMG(seq,y,A_y,$e_{x,y}$+1)：

(1) 收到新的 EMG。

(2) y 在上面的步驟中曾經改變自己的高度。

以上的步驟一到三遵照 partial link reversal 的概念去調整高度。不使用 full link reversal 的做法是因為這個作法可以會使危險區外的點穿過危險區內，而不考慮是否有更安全的路線。相反的，partial link reversal 可以幫助導引使用者繞過危險區域。這部份會在模擬實驗和實作中驗證。

最後，我們來討論兩種特殊的情況：第一種情況為危險區內有出口，另一種為危險區將網路分割成兩部份，一部份有出口，另一部份沒有出口。

危險區內有出口時：主要有兩種作法；一種是導引到該危險區內的出口，另一種是無論如何先走出危險區再找其它的出口。在我的設計中，採用混合的作法；若是出口在一個 hop 的距離內則導引到該出口，不然的話就依照高度值去導引。這又分二種情況：危險區外有出口，在這種情況下會導引到危險區外的出口。危險區內的出口是唯一的出口，在這種情況下會經過多次的高度值計算，最後會導引回危險區內的出口。

危險區分割網路時：在這種情況下，網路被分成兩塊，一塊有出口，一塊沒有出口。沒有出口那邊的感測器在經過反覆的提升高度值之後會高於危險區邊緣的高度值，最後穿越危險區抵達出口。

由以上的討論可以得到兩個導引的規則：

- 若 y 在危險區內且鄰居就是出口，則導引到該出口。
- y 導引使用者到高度值最小的鄰居。

理論 1 假設最少有一個出口感測器不在發生災難的位置，演算法可以為除了出口以外的其它感測器在有限次數以內找到逃生路徑。

證明 一個感測器找不到出口只有在它是區域最小高度的點時才可能發生。又我們的演

算法在調整區域最小的點時會提升它的高度最少δ，δ是一個大於零的值。所以我們可以知道，此演算法會漸漸的減少區域最小的點的數量。最後所有的點都將找到路徑。

三、權重值A_{emg}的選取

權重值的大小會影響導引的正確性。太小的權重值會導致危險區域的邊緣低於感測器的初始高度。假設我們在初始階段所得到的最高初始高度是MAX_{ini},那麼A_{emg}最少要比$MAX_{ini}X(D+1)^2$高才能維持演算法的正確性。

以上就是整個導引協定運作的流程。在第三章第一節中，我會將我的演算法與所提出的方法作一個模擬的比較，由模擬的結果得知我的演算法可以節省緊急情況時封包的交換以及我們能提供一較佳的導引路徑保障人員的安全。

第三章、模擬與實作

第一節、模擬結果比較

這個小節我們要做模擬的比較分析。在這個模疑中我們考慮 4x5 和 10x10 兩個格狀網路。每個感測器都可以跟它東、西、南、北四個方向的感測器做溝通。A_{emg}是 200 而δ值則是 0.1。主要的比較對象是[3]的演算法。效能比較項目為封包量和收斂時間。在這邊我的演算法的封包量並沒有包括初始化階段。我使用了IEEE 802.15.4 中的unslotted CSMA/CA協定來模擬傳輸，它的傳輸速率可以到達每秒 20k個位元。收斂時間的單位是 ms。

模擬結果顯示在圖 14。由於網路很小，所以逃生導引的路徑幾乎是一樣的。但是，我的演算法的封包量要比[3]的小的多，而且收斂時間是[3]的四分之一到二分之一。在我的演算法中，EMG 封包會廣播到整個網路上，而所有的感測器都只會視區域的狀況去選擇逃生方向。而在[3]，當引力值改變時，感測器需要再次廣播，因而增加了封包量。

10x10 網路的結果顯示在圖 15。在狀況一的情形下，位於網路中間的感測器偵測到了危險的事件發生。但是，這個危險的事件並不會改變週圍感測器的相對高度。因此，我的演算法只需要很少量的封包交換即可快速收斂。狀況二，感測器的佈置和狀況一一樣，但這次是由出口感測器 A 偵測到危險事件。雖然兩個演算法都會計算出一樣的導引路線，但[3]會產生很多的封包交換，因為在 A 附近的點會先被 A 吸引，然後再被 A 排斥。在狀況三，在出口 A 附近有一個危險事件被偵測到。如圖所示，因為[3]並沒有危險區域的概念，因此它會導引一部份的使用者經過危險的區域，但這並不是我們想要的導引效果。狀況四，有一部份的感測器被危險區域包圍住。在這個狀況下，導引路徑無可避免的必需要穿過危險區域。但我的演算法仍然可以帶領使用者盡量遠離危險的地方。狀況五，我在左下角的位置又多增加了一個出口。[3]的演算法指引使用者直接穿過危險區域到達出口，但是這個問題在我的演算法中不會發生。狀況六顯示一個網路幾乎要被危險區域給分割成兩部份。我們再一次的看到我的演算法可以找出比[3]更安全的路徑來逃生。

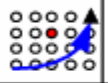
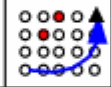
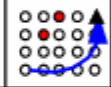
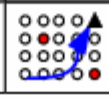
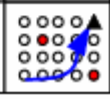
No	Method of Li et al.		Our method ($D = 1$)		No	Method of Li et al.		Our method ($D = 1$)	
	Path	pkt. count/ cnvg. time	Path	pkt. count/ cnvg. time		Path	pkt. count/ cnvg. time	Path	pkt. count/ cnvg. time
1		99 / 178.1		25 / 28.87	6		112 / 217.76		39 / 52.49
2		128 / 188.53		19 / 29.18	7		90 / 184.81		38 / 52.392
3		121 / 190.53		36 / 54.94	8		150 / 341.16		37 / 48.212
4		94 / 173.68		38 / 55.22	9		109 / 194.45		36 / 48.99
5		137 / 255.07		40 / 53.11	10		115 / 210.1		38 / 50.02

圖 14：封包交換個數

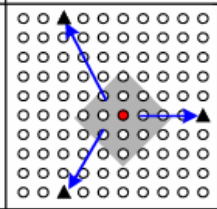
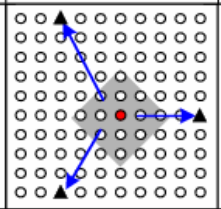
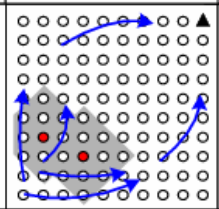
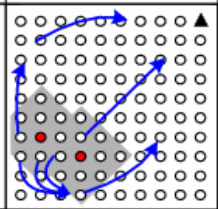
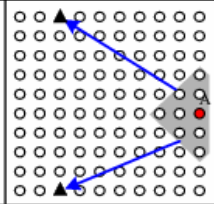
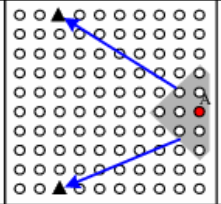
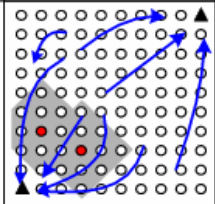
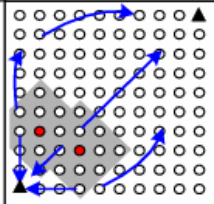
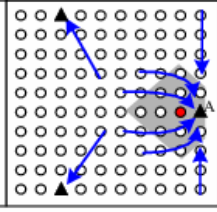
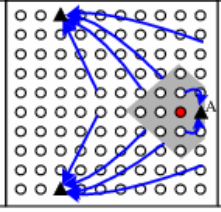
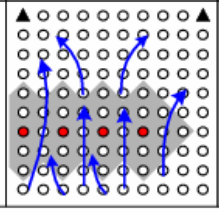
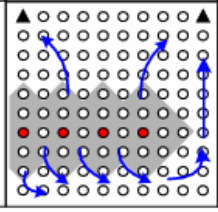
No	Method of Li et al.		Our method (D=2)		No	Method of Li et al.		Our method (D=2)	
	Path	pkt. count/ cnvg. time	Path	pkt. count/ cnvg. time		Path	pkt. count/ cnvg. time	Path	pkt. count/ cnvg. time
1		660 / 350.62		100 / 59.95	4		979 / 468.83		252 / 78..99
2		1215 / 530.81		130 / 87.5	5		731 / 372.0		264 / 107.15
3		742 / 442.52		137 / 87.89	6		1254 / 350.1		408 / 67.29

圖 15：導引效果比較

D 在導引演算法中是一個重要的參數。它是用來決定危險區域的大小。D 的值代表著災難所影響的範圍，它同時也會影響導引的結果和系統的效能。在一個小的網路下，

將 D 值訂得太大並沒有意義，因為危險區域可能會含蓋住整個網路。而一個太小的 D 值則是會造成導引路徑太過於靠近災難源。相反的，一個大一點的 D 值會使得路徑更為安全，但同時也會降低系統的效能。圖 16 顯示了不同的 D 值對於一個 10x10 格狀網路的影響。

圖 17 展示了δ和A_{emg}對逃生路徑和封包交換量的影響。我們可以發現到，使用一個較小的A_{emg}值和一個較大的δ值會導致人們穿過危險區域來逃生。這是因為一個較大的δ值會快速的提升一個處在區域最小值的點的高度，使它高於危險區域的點。因此我建議網路建置者使用一個較大的A_{emg}和一個較小的δ值。

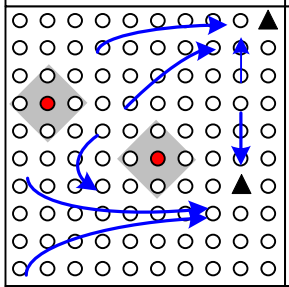
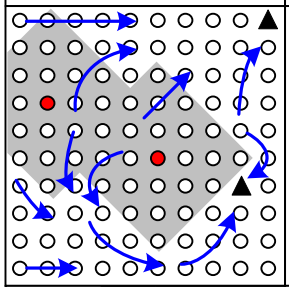
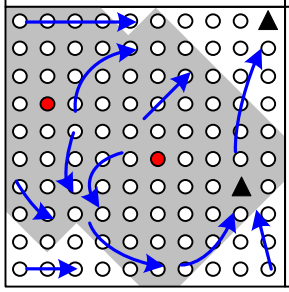
Network(10x10)	*D*	Pkt. count
	1	204
	3	211
	5	398

圖 16：D 值對網路的影響

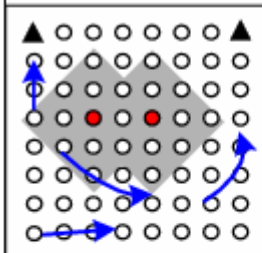
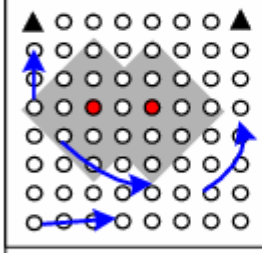
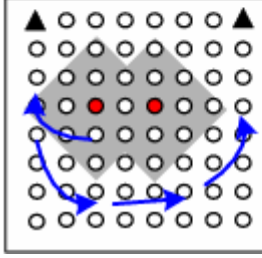
Path	A_{emg}: 100	A_{emg}: 300
	δ:0.1	δ:0.1
	Pkt. Count 184	Pkt. Count 187
	δ:1.1	δ:7.9
	Pkt. Count 148	Pkt. Count 148
	δ:1.2	δ:8.0
	Pkt. Count 155	Pkt. Count 148

圖 17：δ和A_{emg}對逃生路徑和封包交換量的影響

第二節、實作

2-1、軟硬體系統

在這一個項目裡頭我將簡介本論文所使用之軟硬體系統。我使用 Crossbow 公司所發展之 MICAz 無線感應器為硬體平台，並輔以專為 MICAz 設計之微型作業系統 TinyOS 來開發這套系統。下列我將對 MICAz 以及 TinyOS 做一個簡單的介紹。

一、無線感測器(MICAz Mote)

MICA 系列產品原由美國加州柏克萊大學(UC Berkeley)所研發的微塵感測器計畫所開發，MICAz 為目前該計畫所產出之新產品，MICAz 採用 ATMEL 公司所製造的晶片 ATMega128L 做為中央處理器，系統具有 8K 位元祖的可程式快閃記憶體與 512 位元組的隨機存取記憶體，採用精簡指令集設計的哈佛架構(Harvard Architecture)，提供 16 位元的定址轉換空間；系統針對嵌入式應用提供三種省電模式：Idle、Power-down 以及

Power-save。當系統處於 Idle 模式時，會自動停止執行所有的工作排程；當系統處於 Power-down 模式時，僅剩下非同步(Asynchronous)中斷和基礎的周邊設備尚可正常工作；倘若系統處於 Power-save 模式時，大致上運作行為與 Power-down 模式無異，但會多一組非同步計時器，以維持週期性的常態工作。

微塵感測器的射頻電路採用 Chipcon 公司的 CC2420 通訊模組製成，這個模組符合 IEEE 802.15.4 規格，其操作頻率為 2.4GHz 之 ISM Band，支援短距離的無線通訊功能，其最高傳輸率可達到每秒 250 K 位元組，使用 3 伏特的操作電壓，可調整訊號強弱與敏感度，亦支援睡眠管理模式的設計。溫度感測單元則是 Analog Device 的 AD7418 晶片，為內建 A/D 轉換器、溫度感測器、時脈震盪器的整合晶片，利用 I2C 傳輸介面將數位訊號輸入至中央處理器。此外，系統裡有一個 EEPROM 記憶單元，中央處理器會透過 SPI 介面與八位元的協同處理器 ATMEL 晶片連接，由協同處理器去管理大小為 256K 位元組的 EEPROM 模組 24LC256。這裡值得一提的是，每項元件都具有低耗電的設計，而整體運作在全效模式下所使用的電壓為 3 伏特，以及 19.5 毫安培的電流量，粗略估計為 60 微瓦，當 LED 顯示燈全部開啟時，則增至 100 微瓦，如果系統在 Idle 模式下，則使得耗電量下降到 3.1 安培。目前這一套系統，透過業界的 Crossbow Technology 公司負責生產、製造，同時這公司也持續地以專業的微機電整合技術，提供更多樣的周邊硬體，諸如壓力計、液晶顯示器等等，使得無線感測網路的應用更為廣泛。MICAz 的硬體以及其感應板如圖 18 所示：

圖 18：MICAz 的硬體以及其感應板

二、微型作業系統(TinyOS)

微型作業系統 TinyOS 是一套專為小型記憶體的通訊裝置所設計之作業系統，最早的 1.0 版本是由柏克萊大學於 2002 年 10 月份所發表釋出，並宣告原始碼的版權為大眾公有版權(General Public License)，發行至今透過廣大的社群共同參與並互相討論，已經新增了許多功能，目前最新的版本為 1.1.11，且能夠支援 Linux、WindowsXP 等數種電腦作業平台。TinyOS 的架構是建構於元件基礎(component-based)和事件驅動(event-driven)兩大觀念，在開發的過程中，所有的程序在設計時皆需要從元件與事件的角度上來進行思考。TinyOS 利用一個以元件為基礎的模型，來描述硬體與網路組成元件，每一個硬體裝置被描述成一個元件，統稱為 Graph of Components，加上一個排程器(Tiny Scheduler)而建構成整個系統。開發者可透過 NesC 程式語言來開發自訂的通訊演算規則，這是專為需要透過 TinyOS 平台來操作的嵌入式設備所設計的一套程式語言，它的語法類似於 C 語言，讓開發者可以迅速地適應它，進而迅速地專注於事件與系統元件的處理；TinyOS 本身亦提供了一套完整現成的函式介面給撰寫 NesC 的開發者所使用，可降低開發時的複雜度，大幅提增系統的開發期程。TinyOS 亦提供便利的封包架構，系統開發者可以將其所設計的封包包裹在 data 的欄位，感測器收到這一封包可以對這一個封包 data 的內容進行解析，並且做出相對應的動作。

Declaration	*Structure of TOS_Msg*
1.	typedef struct TOS_Msg
2.	uint8_t length;
3.	uint8_t fcfhi;
4.	uint8_t fcflo;
5.	uint8_t dsn;
6.	uint16_t destpan;
7.	uint16_t addr;
8.	uint8_t type;
9.	uint8_t group;
10.	int8_t data[TOSH_DATA_LENGTH];
11.	uint8_t strength;
12.	uint8_t lqi;
13.	bool crc;
14.	uint8_t ack;
15.	uint16_t time;
16.	} TOS_Msg;

圖 19：TinyOS 封包格式

三、導引方向指示版

在我們的實作中，最重要的就是能夠讓使用者輕易的知道要往何處逃生，但受限於 MicaZ Mote 上僅提供三顆燈號，因此我自己設計了一塊方向指示版(圖 21)。這塊方向指示版的輸入訊號是從 MicaZ Mote 上的 HIROSE SOCKET 所拉出來的三條代表 MicaZ 上燈號的訊號線。其中二條訊號線配合一個 10x10 的 LED 矩陣可以顯示出四個方位。另一條訊號線配合一顆紅色的 LED 用來代表緊急疏散的命令。平時 LED 矩陣上會顯示一組預設的逃生方向(圖 22)，就如同舊式的導引燈號一樣(圖 20)。一但網路上有一顆感測器感測到危險就會立刻把紅色的 LED 亮起，同時啟動緊急導引協定，於是人們就可以照著 LED 矩陣上所指示的方向逃生(圖 23)。

圖 20：舊式導引裝置

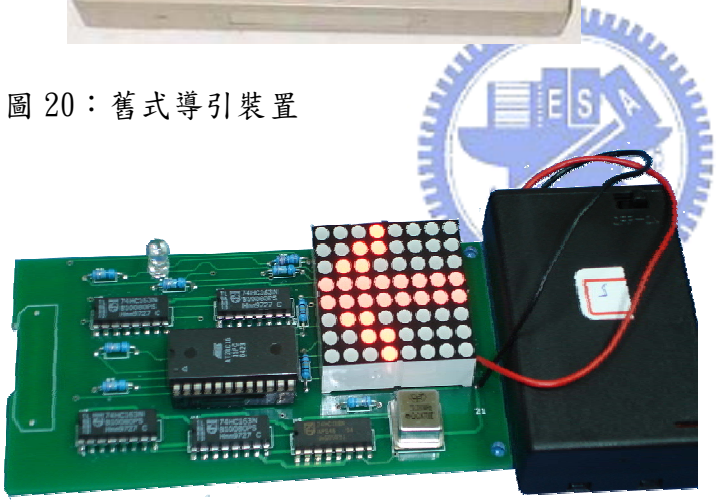


圖 21：導引方向指示版

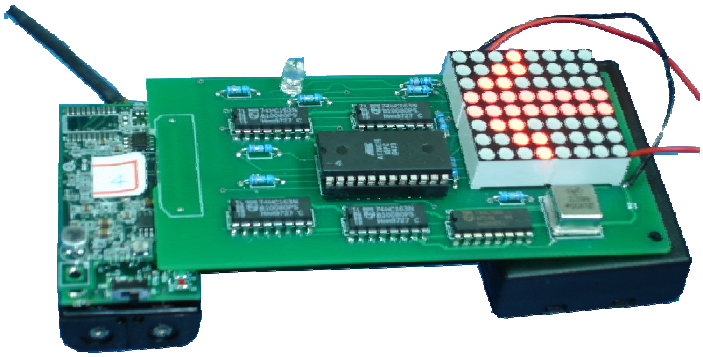


圖 22：預設導引方向

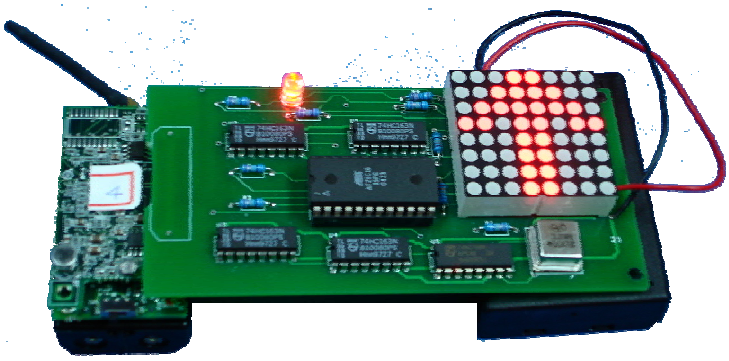


圖 23：緊急疏散指示燈以及導引方向

2-2、程式架構與實作心得

由上一項可以知道，TinyOS 實際上跟一般的作業系統很不一樣，它最主要的部份只有驅動程式以及排程器，在上面開發出來的應用程式會跟整個作業系統做一個很緊密的結合。圖 24 就是整個感測器端的程式架構。因為作業系統的支援有限，因此很多東西都必需自己處理，最重要的就是要自己負責佇列封包。佇列封包的目的在於，當應用程式端短時間內發送封包的速度比實際傳送快時，就必需要將這些較快的封包存在一個佇列中，然後將它一個一個的送出去。在圖 24 中可以看到，TinyOS 提供了一個函式庫稱為 QueuedSend，它的功能是佇列所有封包的指標，然後將這些封包一一的送出，而非佇列封包本身。因此，這個儲存封包的任務就必需由應用程式端來負責。另外，TinyOS 並沒有提供數學函式庫，因此在實作演算法時就必需要做一點修正。在我的演算法中，調整區域最小值的方法時是以標準差來當做基準，但是在標準差的計算中會使用到開根號，因此我們在實作中將標準差修正為平均值跟本身高度的差異。除此之外，由於硬體上資料記憶體(data memory)的大小僅有 4KB，而這 4KB 是必需跟作業系統共用，因此在設計佇列或是陣列時必需要特別注意到大小，否則很容易就會使用超過這個數量的記憶體。尤其是封包佇列，大小最多只能開到 16 個左右，這對於程式的穩定性來說是一個很大的考驗。總結來說，實作這個系統有以下的困難：

- 作業系統缺乏佇列封包的支援。
- 缺乏數學函式庫。
- 資料記憶體過小。
- 網路傳輸不穩。

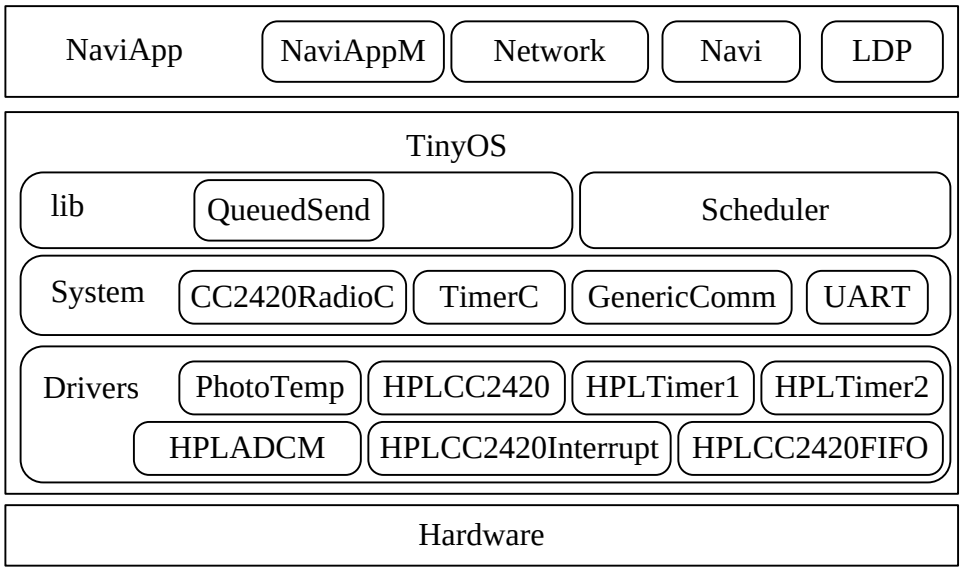


圖 24：系統架構

2-3、實驗設定

實驗採用 21 顆感測器，其中 1 顆為基地台，只用來作為感測器網路與電腦端間的橋接器。另外的 20 顆以 5x4 的格狀方式密集排列。其中最右上角的感測器設定為出口。所有的感測器其傳輸距離均開到最大，所有的點皆為全完連接(fully connected)，因此在此情況下，其干擾應最為嚴重。在邏輯連線的設定上，不考慮對角線的連線，每個點最多有 4 個鄰居。另外，在演算法參數的設定上，Max Weight 我們設定為 65535，而 safety factor 則是設定為 1。

下列三張圖為網路佈置以及實際運作的示意圖。圖 25 為平時的導引路線。圖 26 及圖 27 則是發生災難時的情形。在發生災難的點我們會亮起紅燈，並且可以從箭頭看出導引路徑變化的情形。

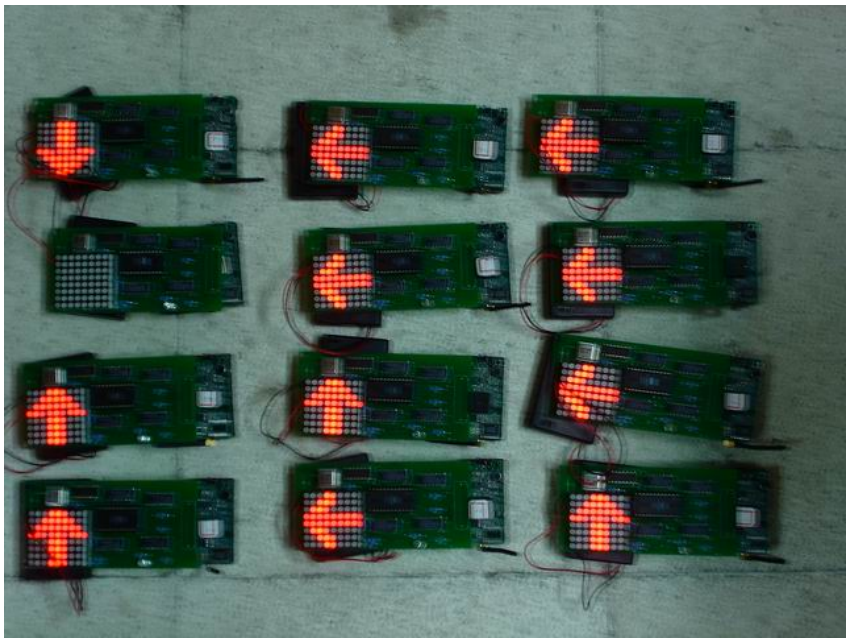


圖 25：經過初始化階段後的導引路徑。當沒有災難發生時，所有逃生方向都是走最近的路到出口。

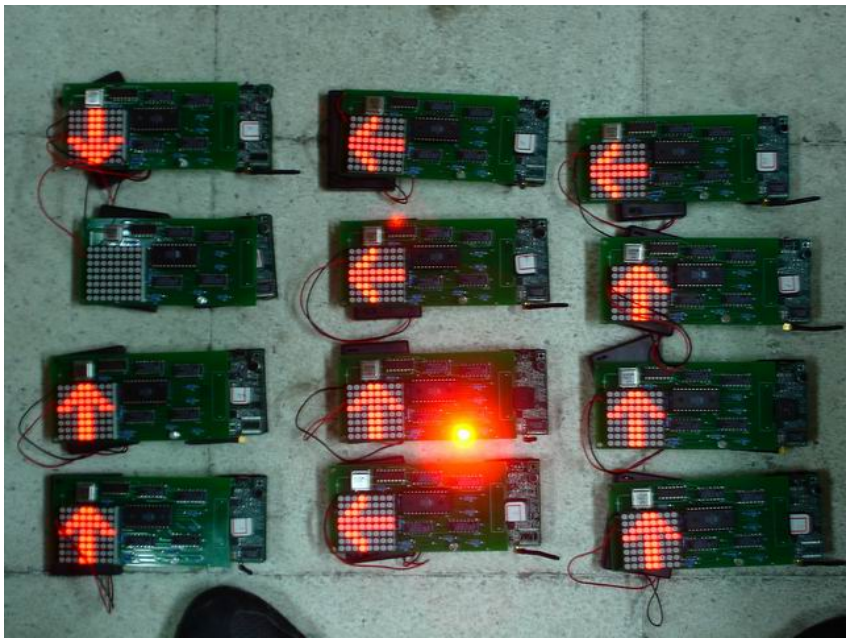


圖 26：發生一個災難時的導引路線。

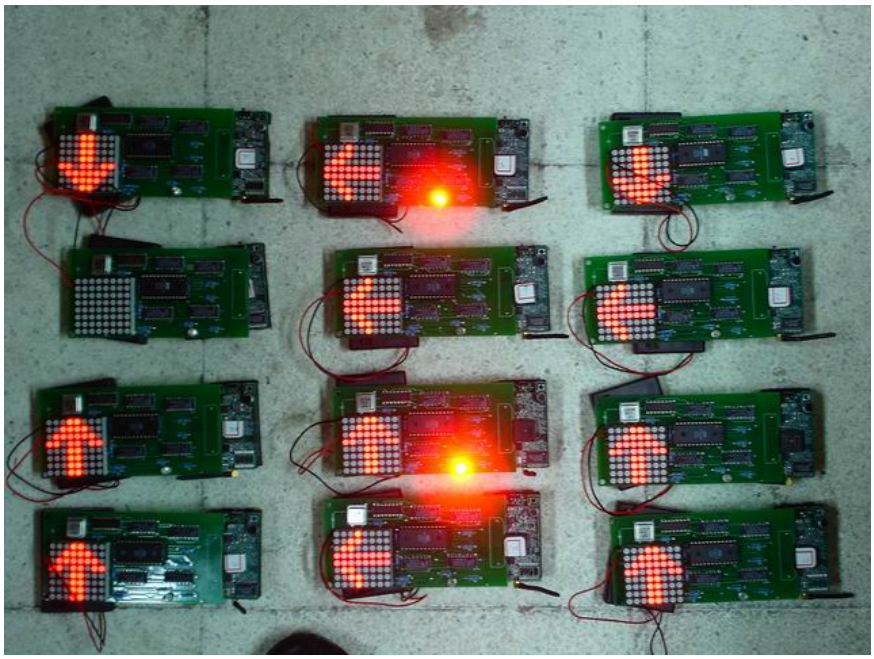


圖 27：發生兩個災難時的導引路線。

第三節、實驗結果

實驗結果如圖 28 所示。Simulation 表示在模擬時所得到的結果。Experiment 表示實際上跑出來的結果。Pkt. Count 裡斜線左邊表示模擬時得到的封包數，右邊表示實際上發送的封包數。

首先要看的是實作上的正確性。No1、No2、No3 的狀況中，可以看到最下一排先往右再往上最後到達出口，最左兩排則是先往下再往右最後再往上到達出口，確實有繞過危險區域。No4、No6 由於一個災難在上一個災難在下，所以會看到最下一排會先往上走再往右走再往上走，跟模擬結果相差無幾。

另外要注意的一點是封包的數量，由圖表中可以看到，實作收到的封包稍微多了一點。這是由於實際傳輸到網路上會先經過亂數時間的延遲(random backoff)以及介質的競爭，所以收到的封包順序不會像模擬那樣完美，在計算權重值的時候可能會多調整一到兩次，因此封包才會多了一點。但是，總結來說，這個實作確實有達到二點要求：

1. 導引路線安全性的確保。
2. 導引路線必需能很快計算出來。

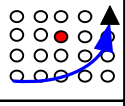
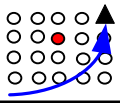
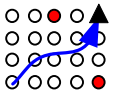
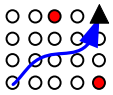
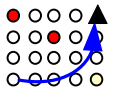
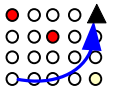
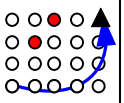
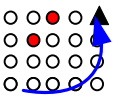
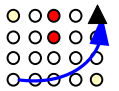
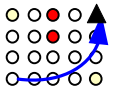
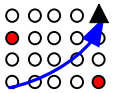
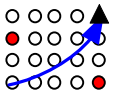
No	Simulation	Experiment	Pkt. count	No	Simulation	Experiment	Pkt. count
1			25/26	4			39/47
2			36/40	5			38/44
3			40/44	6			36/40

圖 28：實驗結果

第四章、結論

無線感測器網路的研究在最近非常的興盛。然而大部份的研究都使用模擬來驗證正確性。相反的，在本論文中我們使用了 MicaZ Mote 實作出了一套室內安全偵測以及緊急急難逃生系統。在這個實作中，我們可以發現實際上硬體的狀況不是完美的，比如有不對稱連線，或是感測器計算能力及儲存能力的問題。但是大部份使用模擬實驗來驗證的協定並沒有考慮這些因素，這使得實作這些演算法或協定成了不可能的任務。另外，大部份的論文都會有許多的假設，比如每個感測器會知道自己的位置，或是感測器知道彼此間的相互位置。然而在現實上，這些假設往往都不可能成立，因此要實作一個系統，我們必需要針對這些假設去尋找解決辦法。在我的實作中，我利用了電腦端的程式來解決感測器佈置及位置偵測的問題；以及在演算法中加入額外的協定(Handshaking protocol)來解決不對稱連線的問題。總結來說，我們的系統有下列的幾項優點：

- 簡易的網路佈置方式
- 安全導引人員逃生

我們提出的系統架構以及概念想法可以直接應用於百貨公司、商業大廈、展覽場地等人潮聚集的大型場所，在未來我們也將會繼續研究該如何擴張我們的系統提供更好的服務：

- 加入感測器間可以自動的偵測相互位置(Network Position)的功能來簡化網路佈置流程。
- 研究對於不同的緊急急難狀況是不是需要修改原始的導引演算法。
- 將原本平面導引的功能延伸至 3D 立體空間的導引。
- 提供負載平衡(Load balance)的拓樸重建演算法。

最後我們深切地期盼在不久的將來這一套概念系統可以被廣泛的採用，讓社會大眾的人身安全獲得更大保障。

參考文獻

[1]TinyOS Community Forum. An open-source OS for the networked sensor regime. web site: http://www.tinyos.net/.

[2]Inertial, MEMS Gyro, FAA Certified AHRS, and Smart Dust Wireless Sensors from Crossbow, web site: http://www.xbow.com/.

[3]Q. Li, M. DeRosa, and D. Rus, “Distributed algorithm for guiding navigation across a sensor network,” in Proc. of ACM MobiHOC, 2003.

[4]V. D. Park and M.S. Corson, “A highly adaptive distributed routing algorithm for mobile wireless network,” in Proc. of IEEE Infocom, 1997.

[5]X. Wang, F. Silva, and J. Heidemann, “Infrastructureless location aware configuration for sensor networks,” in Proc. of IEEE WMCSA, 2004.

[6]T. Y. Chiang, “Design and Implementation for an In-Building Fire Emergency System based on Wireless Sensor Network,” Master’s thesis, National Chiao Tung University, Hsin-Chu, Taiwan, January 2005.

[7]Habitat Monitoring on Great Duck Island. http://www.greatduckisland.net/technology.php

[8]Design and Construction of a Wildfire Instrumentation System Using Networked Sensors. http://firebug.sourceforge.net

[9]P. Bahl and V. N. Padmanabhan, “RADAR: An In-building RF-based user location and tracking system,” in Proc. of IEEE Infocom, 2000.

[10]K. Romer and F. Mattern, “The Design Space of Wireless Sensor Networks,” in IEEE Wireless Communications, Dec. 2004.

[11]C. Busch and S. Tirthapura, “Analysis of Link Reversal Routing Algorithms,” in Proc. of ACM symposium on Parallel Algorithms and Architectures, 2003.

附錄、緊急導引程式(NaviM.nc)

```
includes AM;
includes Navi;
#define  MAX_QUEUE_SIZE 16
module NaviM {
    provides {
        interface StdControl;
        interface NaviControl;
    }
    uses {
        interface SendMsg as NaviSend;
        interface ReceiveMsg as NaviReceive;
        interface Leds;
        interface Timer;
        interface Timer as NaviTimer;
    }
}
implementation {
/*********************************************/
/* Navigation Link Table:                    */
/*  Records of the navigation links including  */
/*  the direction, the role and the weight of  */
/*  each navigation link                       */
/*********************************************/
    struct LinkEntry {
        bool     valid;
        uint8_t  id;
        uint8_t      dir;
        uint8_t      role;
        float        Weight;
    } LinkTable[MAX_NAVILINK];
/************************************/
/* Emergency Event Pool:            */
/*  Records of the emergency events */
/************************************/
    struct EmgEntry {
        uint8_t  id;
        uint8_t      hop;
        uint8_t      rcvd;
    } EmgPool[MAX_EMG];
    uint8_t  LinkCount;         //navigation link count
    uint8_t  EmgCount;     //emergency event count
    uint8_t  SafetyFactor; //the radius of the razardous region

/*********************************************/
/* The role of this sensor: The value could be */
/*  0: base station node                     */
/*  1: exit sensor                           */
/*  2: normal sensor                         */
/*********************************************/
    uint8_t  Role;
    float    BaseWeight;   //initial weight
    float    Weight;       //actual weight
    //Internal Buffers
    TOS_Msg  gInitBuffer;  //buffer of the initialization packet
```

```
    TOS_Msg   gEmgBuffer;    //buffer of the EMG packet
    TOS_Msg   gFwdBuffer[MAX_QUEUE_SIZE];  //Forwarding buffer
    TOS_Msg   *gFwdBufPtr[MAX_QUEUE_SIZE]; //Pointers of the forwarding buffer
    uint8_t   iFwdBufTail,iFwdBufHead;
    bool gfInitBusy;
    bool gfEmgBusy;
    int       guide_index;
    uint16_t  emg_packet_count;  //emergency packet counts is used for statistic report
    // for periodical report emergency infomation
    bool      hasEmgNode;
/*******************************************************/
/*   The exit sensor will send the initialization packet */
/* in this function.                                     */
/*******************************************************/
    task void initialization(){
        NaviInfo  *naviInfo = (NaviInfo*)gInitBuffer.data;
        if( gfInitBusy ) return;
        Weight = BaseWeight = 0;
        gfInitBusy = TRUE;
        naviInfo->type = INIT;
        naviInfo->src = (uint8_t)TOS_LOCAL_ADDRESS;
        naviInfo->hop = 0;
        if( (call NaviSend.send(TOS_BCAST_ADDR,sizeof(NaviInfo),&gInitBuffer))!=SUCCESS){
            gfInitBusy = FALSE;
        }
    }
    void regularReport() {
        hasEmgNode = TRUE;
        call NaviTimer.start(TIMER_REPEAT, NAVI_INTERVAL);
    }
/*******************************************************/
/*   Periodically broadcast the altituide in order to */
/* recover the packet loss.                            */
/*******************************************************/
    task void emergencyReport() {
        NaviInfo  *naviInfo = (NaviInfo*)gEmgBuffer.data;
        if( gfEmgBusy || EmgCount==0) return;
        gfEmgBusy = TRUE;
        naviInfo->type = EMG;
        naviInfo->src = (uint8_t)TOS_LOCAL_ADDRESS;
        naviInfo->hop = EmgPool[EmgCount-1].hop;
        naviInfo->emg = EmgPool[EmgCount-1].id;
        naviInfo->weight = Weight;
        naviInfo->tag = TRUE;
        if( (call NaviSend.send(TOS_BCAST_ADDR,sizeof(NaviInfo),&gEmgBuffer))!=SUCCESS){
            gfEmgBusy = FALSE;
        }
    }
/*******************************************************/
/*   On detecting an emergency event, the sensor will */
/* raise its weight and send the EMG packet.           */
/*******************************************************/
    task void emergency(){
        NaviInfo  *naviInfo = (NaviInfo*)gEmgBuffer.data;
        if( gfEmgBusy ) return;
        call Leds.redOn();
```

```
        Weight = BaseWeight + MAX_WEIGHT;
        gfEmgBusy = TRUE;
        naviInfo->type = EMG;
        naviInfo->src = (uint8_t)TOS_LOCAL_ADDRESS;
        naviInfo->hop = 0;
        naviInfo->emg = (uint8_t)TOS_LOCAL_ADDRESS;
        naviInfo->weight = Weight;
        naviInfo->tag = TRUE;
        if( (call NaviSend.send(TOS_BCAST_ADDR,sizeof(NaviInfo),&gEmgBuffer))!=SUCCESS){
            gfEmgBusy = FALSE;
        }
    }
    static void showDirection(uint8_t dir){
        if(Role == ROLE_EXIT )return;
        switch(dir){
        case 0:
            call Leds.greenOff();
            call Leds.yellowOff();
            break;
        case 1:
            call Leds.greenOff();
            call Leds.yellowOn();
            break;
        case 2:
            call Leds.greenOn();
            call Leds.yellowOff();
            break;
        case 3:
            call Leds.greenOn();
            call Leds.yellowOn();
            break;
        }
    }
/*************************************************************/
/*   Find the index of the given id in the Emergency Pool. */
/* If there is no match, return -1.                          */
/*************************************************************/
    static int getEmgIndex(uint8_t id){
        int  i;
        for(i=0;i<EmgCount;i++){
            if(EmgPool[i].id == id)return i;
        }
        return -1;
    }
    static int newEmgEntry(uint8_t id,uint8_t hop){
        int  new_index = -1;
        if(EmgCount==MAX_EMG)return -1;
        EmgPool[EmgCount].id = id;
        EmgPool[EmgCount].hop = hop;
        new_index = EmgCount;
        EmgCount++;
        return new_index;
    }
    static int getMinWeightLink(){
        int  link_index = -1;
        int  i;
```

```
        float     min_Weight = MAX_WEIGHT;
        for(i=0;i<LinkCount;i++){
            if(LinkTable[i].Weight < min_Weight ){
                link_index = i;
                min_Weight = LinkTable[i].Weight;
            }
        }
        return link_index;
    }
    static int getExitFromLink(){
        int  link_index = -1;
        int  i;
        for(i=0;i<LinkCount;i++){
            if(LinkTable[i].role == ROLE_EXIT){
                link_index = i;
                break;
            }
        }
        return link_index;
    }
    static float getAvgWeight(){
        int  i;
        float     sum=0;
        for(i=0;i<LinkCount;i++){
            if(LinkTable[i].valid){
                sum+=LinkTable[i].Weight;
            }
        }
        sum/=LinkCount;
        return sum;
    }
/*****************************************/
/*   Find the index of the given id in the */
/* navigation link table. If we can't find */
/* this id in the table, the returned      */
/* value will be -1.                       */
/*****************************************/
    static int getLinkIndex(uint8_t id){
        int  i;
        for(i=0;i<LinkCount;i++){
            if(LinkTable[i].id == id )return i;
        }
        return -1;
    }
/*************************************************/
/*   Forwarding the initialization packets and the */
/* EMG packets.                                    */
/*************************************************/
    static TOS_MsgPtr forward(TOS_MsgPtr pMsg){
        TOS_MsgPtr     pNewBuf = pMsg;
        NaviInfo  *navi = (NaviInfo*)pMsg->data;
        if(((iFwdBufHead+1)%MAX_QUEUE_SIZE)==iFwdBufTail)return pNewBuf;
        navi->src = (uint8_t)TOS_LOCAL_ADDRESS;
        if((call NaviSend.send(TOS_BCAST_ADDR,sizeof(NaviInfo),pMsg))==SUCCESS){
            pNewBuf = gFwdBufPtr[iFwdBufHead];
            gFwdBufPtr[iFwdBufHead] = pMsg;
```

```
            iFwdBufHead++;iFwdBufHead%=MAX_QUEUE_SIZE;
        }
        return pNewBuf;
    }
    static void initialize(){
        int  i;
        for(i=0;i<MAX_NAVILINK;i++){
            LinkTable[i].valid = FALSE;
            LinkTable[i].role = ROLE_SENSOR;
            LinkTable[i].Weight = MAX_WEIGHT;
        }
        for(i=0;i<MAX_EMG;i++){
            EmgPool[i].rcvd = 0;
            EmgPool[i].hop = 0;
        }
        for(i=0;i<MAX_QUEUE_SIZE;i++){
            gFwdBufPtr[i] = &gFwdBuffer[i];
        }
        iFwdBufHead = iFwdBufTail = 0;
        gfInitBusy = FALSE;
        gfEmgBusy = FALSE;
        Weight = BaseWeight = MAX_WEIGHT;
        LinkCount = 0;
        EmgCount = 0;
        SafetyFactor = 0;
        guide_index = -1;
        emg_packet_count = 0;

        hasEmgNode = FALSE;

        call Leds.init();
        call Leds.redOff();call Leds.greenOff();call Leds.yellowOff();
    }
    command result_t StdControl.init(){
        initialize();
        return SUCCESS;
    }
    command result_t StdControl.start(){
        return SUCCESS;
    }
    command result_t StdControl.stop(){
        return SUCCESS;
    }
//Setup the safety factor of this network
    command result_t NaviControl.setSafetyFactor(uint8_t safety_factor){
        SafetyFactor = safety_factor;
        return SUCCESS;
    }
//Setup the role of this sensor
    command result_t NaviControl.setRole(uint8_t role){
        Role = role;
        return SUCCESS;
    }
//Setup the navigation links
    command result_t NaviControl.setLink(uint8_t link_count,uint8_t *links,uint8_t *dir){
        int  i;
```

```
            LinkCount = link_count;
            for(i=0;i<LinkCount;i++){
                LinkTable[i].valid = FALSE;
                LinkTable[i].id = links[i];
                LinkTable[i].dir = dir[i];
            }
            return SUCCESS;
        }

        command result_t NaviControl.reset(){
            initialize();
            call NaviTimer.stop();
            hasEmgNode = FALSE;
            return SUCCESS;
        }
    //Tell the exit node to start initialization phase.
        command result_t NaviControl.init(){
            post initialization();
            return SUCCESS;
        }
        command result_t NaviControl.emergency(){
            post emergency();
            return SUCCESS;
        }
        command uint8_t NaviControl.getNaviDestination(){
            if(guide_index==-1)return 255;
            return LinkTable[guide_index].id;
        }
        command float NaviControl.getNaviDstWeight(){
            if(guide_index==-1)return -1;
            return LinkTable[guide_index].Weight;
        }
        command float NaviControl.getWeight(){
            return Weight;
        }
        command uint16_t NaviControl.getPacketCount(){
            return emg_packet_count;
        }
        command result_t NaviControl.setEmg() {
            newEmgEntry(TOS_LOCAL_ADDRESS, 0);
        }
        event result_t Timer.fired(){
            return SUCCESS;
        }
    /*************************************************************/
    /*   We do initial phase and navigation phase here.  The base */
    /* station node does not join any navigation. It only        */
    /* functions as a gateway between the sensor network and the  */
    /* computer.                                                 */
    /*   Since the communication link is not necessarily the     */
    /* navigation link, we compare the sender id with the        */
    /* navigation link table first.                              */
    /*************************************************************/
        event TOS_MsgPtr NaviReceive.receive(TOS_MsgPtr pMsg){
            NaviInfo  *navi = (NaviInfo*)pMsg->data;
            int       navi_index = getLinkIndex(navi->src);  //Get the index of the sender id
```

```
        int       emg_index,min_Weight_link;
        bool new_emg = FALSE;
        bool changed = FALSE;
        float     ratio,tmp_Weight;
//The base station doesn't join any navigation.
        if(Role==ROLE_BASE)return pMsg;
//If the index is -1, the sensor is not our navigation link.
        if( navi_index == -1 ) return pMsg;
        if( navi->type == INIT ){ //initialization phase
            if( !LinkTable[navi_index].valid ){ //the first initial packet from this sender
                if( navi->hop == 0 )
                    LinkTable[navi_index].role = ROLE_EXIT;
                else LinkTable[navi_index].role = ROLE_SENSOR; //default role is normal
sensor
                LinkTable[navi_index].valid = TRUE;
                LinkTable[navi_index].Weight = navi->hop;
            }
            else if( LinkTable[navi_index].Weight > navi->hop ){
                LinkTable[navi_index].Weight = navi->hop;
            }
            navi->hop++;
            if( navi->hop < BaseWeight ){
                Weight = BaseWeight = navi->hop;
                guide_index = navi_index;
                showDirection(LinkTable[guide_index].dir);
                pMsg = forward(pMsg);
            }
        }
        else if( navi->type == EMG ){     //navigation phase
            if( (Weight-BaseWeight)>MAX_WEIGHT)return pMsg;

            emg_index = getEmgIndex(navi->emg);   //Try to find the same emergency event
from Pool
            LinkTable[navi_index].Weight = navi->weight;
            if( emg_index == -1 ){  //It's a new event.
                new_emg = TRUE;
                emg_index = newEmgEntry(navi->emg,(navi->hop+1)); //Log this event
            }
            else { //We received this event before.

    /*************************************************************************/
    /*   Compare the hop count of this packet with the hop count of this event. */
    /* Update the hop count of this event if the new hop count is smaller.      */
    /*************************************************************************/
                if( (navi->hop+1) < EmgPool[emg_index].hop ){
                    EmgPool[emg_index].hop = navi->hop+1;
                }
                EmgPool[emg_index].rcvd++;
            }
            if( (navi->hop+1) <= SafetyFactor ){ // The node is  in the hazardous region.
                //Raise the altitude according to the distance from the emergency node.
                ratio = (float)1/(EmgPool[emg_index].hop+1);
                tmp_Weight = MAX_WEIGHT*ratio*ratio + BaseWeight;
                if( Weight < tmp_Weight && Weight != 0 ){
                    Weight = tmp_Weight;
                    changed = TRUE;
```

```
                }
            }
            min_Weight_link = getMinWeightLink(); //Find the navigation link with the lowest
altitude
            if(min_Weight_link<0)return pMsg;
            //This node has the local minimum altitude and it's not a exit.
            if( LinkTable[min_Weight_link].Weight >= Weight && Weight != 0 ){
                //Raise the altitude according to the navigation link count.
                ratio = (float)1/LinkCount;
                Weight =
LinkTable[min_Weight_link].Weight+ratio*(getAvgWeight()-LinkTable[min_Weight_link].Weight)
+0.1;
                changed = TRUE;
            }

            if( changed || new_emg ){ //If this is a new emergency or we have changed our
altitude, reboradcast EMG.
                navi->weight = Weight;
                navi->hop = EmgPool[emg_index].hop;
                emg_packet_count++;
                pMsg = forward(pMsg);
            }
            guide_index = min_Weight_link;
            if((min_Weight_link=getExitFromLink())!=-1){ //If there is an exit next to this
node, guide people to it.
                guide_index = min_Weight_link;
            }
            showDirection(LinkTable[guide_index].dir);

            if (!hasEmgNode) { //Periodically broadcast our weight.
                regularReport();
            }
        }
        return pMsg;
    }
    event result_t NaviSend.sendDone(TOS_MsgPtr pMsg,result_t success){
        if( pMsg == &gInitBuffer ){
            gfInitBusy = FALSE;
        }
        else if( pMsg == &gEmgBuffer ){
            gfEmgBusy = FALSE;
        }
        else if( pMsg == gFwdBufPtr[iFwdBufTail] ){
            iFwdBufTail++;iFwdBufTail%=MAX_QUEUE_SIZE;
        }
        return SUCCESS;
    }
    event result_t NaviTimer.fired() {
        post emergencyReport();
        return SUCCESS;
    }
}
```