การปรับปรุงความแม่นยำของระบบค้นหาตำแหน่งภายในอาคาร

An Accuracy Improvement of Indoor Local Positioning System

คำนำ

ความเป็นมาและความสำคัญของปัญหา

เนื่องจากปัจจุบันระบบระบบค้นหาตำแหน่งภายนอกอาคารเช่น Global Positioning System (GPS) นั้นไม่เหมาะสมกับการใช้ในอาคารเนื่องจากสัญญาณที่ส่งจากดาวเทียมนั้นไม่สามารถทะลุผ่านสิ่งกีดขวางเข้ามาภายในอาคารได้รวมถึงระบบมีราคาแพงและมีความแม่นยำไม่สูงมากนักดังนั้นงานวิจัยนี้จึงได้นำเอาอุปกรณ์ตรวจจับแบบไร้สายมาประยุกต์ใช้งานซึ่งปัจจุบันได้มีการนำเอาระบบเครือข่ายตรวจจับแบบไร้สายมาใช้งานในหลากหลายรูปแบบเช่นการตรวจวัดคุณสมบัติต่างๆของสิ่งแวดล้อมเพื่อการศึกษาและสร้างองค์ความรู้ใหม่ๆ การตรวจจับความเคลื่อนไหวของสิ่งมีชีวิตและยานพาหนะเพื่อทางการทหาร, การตรวจวัดอุณหภูมิและความชื้นเพื่อควบคุมคุณภาพของผลผลิตทางการเกษตร, การตรวจวัดข้อมูลทางการแพทย์เช่นอัตราการเต้นของหัวใจ, การตรวจสอบโครงสร้างของสิ่งปลูกสร้างเช่นสะพานหรืออาคารเป็นต้นในงานวิจัยนี้จึงมีแนวคิดที่จะนำเอาอุปกรณ์ตรวจจับแบบไร้สายมาประยุกต์เพื่อใช้ในการระบุตำแหน่งของวัตถุภายในอาคาร (Local Positioning System: LPS) ซึ่งสามารถนำไปใช้ประโยชน์ได้หลากหลายเช่นระบบนำทางของหุ่นยนต์, ระบบคลังสินค้า หรือระบบรักษาความปลอดภัยเป็นต้น

วัตถุประสงค์

1. เพื่อศึกษาถึงงานวิจัยทางด้านระบบค้นหาตำแหน่งภายในอาคาร
2. เพื่อศึกษาถึงรูปแบบการเชื่อมต่อของอุปกรณ์
3. เพื่อศึกษาถึงความแม่นยำ, ข้อดีและข้อเสียของระบบที่ได้ออกแบบ
4. เพื่อเผยแพร่ผลการศึกษาให้เป็นองค์ความรู้แก่ผู้สนใจ

การตรวจเอกสาร

ความรู้พื้นฐานเรื่องระบบเครือข่ายตรวจจับแบบไร้สาย

1. ระบบเครือข่ายตรวจจับแบบไร้สายคืออะไร

ระบบเครือข่ายตรวจจับแบบไร้สายคือการนำเอาอุปกรณ์ตรวจจับ (Sensor) จำนวนมากมาเพื่อตรวจวัดค่าบางจากสิ่งแวดล้อมรอบตัวเราเช่น อุณหภูมิหรือความชื้นเป็นต้นซึ่งการเชื่อมต่อกันของอุปกรณ์ตรวจจับนั้นจะเชื่อมต่อกันแบบไร้สายโดยผ่านคลื่นวิทยุเป็นหลักซึ่งอุปกรณ์ Wireless Sensor Network (WSN) หรือ Mote นั้นประกอบด้วยส่วนหลักๆได้แก่ ส่วนตรวจจับ, ซีพียู และส่วนรับส่งแบบไร้สาย โดยเครือข่ายตรวจจับแบบไร้สายจะประกอบด้วย Mote จำนวนมากส่งข้อมูลผ่านด้วยกันเองจนถึงปลายทาง

การนำเอาเครือข่ายตรวจจับแบบไร้สายไปประยุกต์ใช้งานนั้นมีหลากหลายรูปแบบเช่นการติดตามลักษณะของสภาพแวดล้อมเช่นอุณหภูมิเพื่อนำข้อมูลที่ได้ไปศึกษาหรือควบคุมส่วนอื่นๆที่เกี่ยวข้องต่อไปโดยอัตโนมัติโดยไม่ต้องมีมนุษย์ไปทำการควบคุม, การตรวจจับควันไฟเพื่อส่งตำแหน่งของจุดที่เกิดเหตุเพื่อให้ผู้ที่เกี่ยวข้องสามารถแก้ปัญหาได้ทันเวลา (Byungrak., 2006) หรือการนำมาใช้ในการติดตามข้อมูลทางการแพทย์เช่นการติดตั้งอุปกรณ์ตรวจวัดอัตราการเต้นของหัวใจแล้วทำการส่งข้อมูลดังกล่าวให้แก่แพทย์เมื่อพบสิ่งผิดปกติเพื่อให้สามารถให้ความช่วยเหลือได้ทันเวลา(Aleksandar., 2006)

webC Mote

Shockfish

Telos

Mica2

Rene

EyesIFX

ภาพที่ 1 แสดงตัวอย่างของ Mote

ปัจจุบัน Mote ได้มีการพัฒนาหลากหลายรูปแบบ ภาพที่ 1 แสดงภาพตัวอย่างของ Mote ที่ได้รับการพัฒนาซึ่งการออกแบบ Mote นั้นขึ้นอยู่กับการประยุกต์ใช้งานเป็นหลักซึ่งจะมีข้อจำกัดที่เหมือนกันในเรื่องของพลังงานดังนั้นการออกแบบ Mote จึงต้องคำนึงถึงเรื่องของพลังงานมากที่สุดเนื่องจากสภาพแวดล้อมในความเป็นจริงนั้น Mote อาจถูกติดตั้งอยู่ในสถานที่ๆยากต่อการเปลี่ยนแบตเตอรี่รวมถึงการออกแบบระบบปฏิบัติการซึ่งควรออกแบบให้ใช้พลังงานให้น้อยที่สุดซึ่งโดยส่วนใหญ่แล้วจะให้ซีพียูอยู่ในโหมดประหยัดพลังงานโดยจะออกจากโหมดประหยัดพลังงานเฉพาะเมื่อต้องการอ่านข้อมูลหรือส่งข้อมูลที่จะเป็นเท่านั้น

ประสิทธิภาพของ Mote นั้นขึ้นอยู่กับปัจจัยหลายประการเช่นความเร็วของซีพียู, ความจุของหน่วยความจำ, อัตราการสิ้นเปลืองพลังงาน, ขนาด, ความเร็วในการสื่อสารข้อมูล และอื่นๆ ซึ่งในปัจจุบันอุปกรณ์ที่ประกอบเป็น Mote นั้นได้มีการพัฒนาไปอย่างต่อเนื่องจึงทำให้ประสิทธิ์ภาพของ Mote นั้นดีขึ้นในทุกๆด้านรวมถึงราคาที่ถูกลงจึงมีความเป็นไปได้สูงที่ความนิยมในการใช้ระบบเครือข่ายตรวจจับแบบไร้สายในอนาคตจะมีสูงขึ้นตามไปด้วย

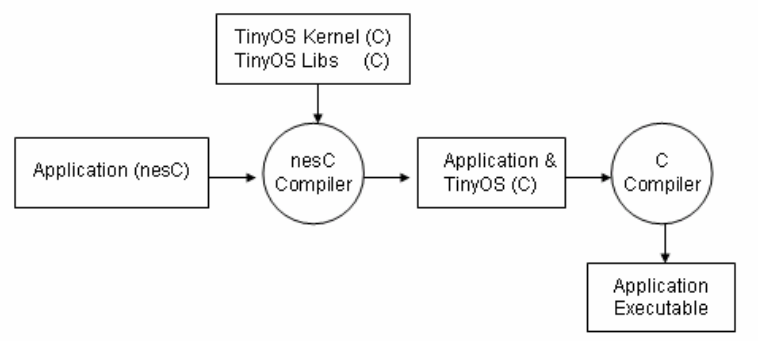
เนื่องจากข้อจำกัดในเรื่องของพลังงานที่มีอย่างจำกัดการส่งข้อมูลระหว่าง Mote ในระบบเครือข่ายตรวจจับแบบไร้สายนั้นจึงส่งข้อมูลในลักษณะมัลติฮอปคือการส่งข้อมูลให้ Mote ที่อยู่ติดกันเท่านั้นแล้วทำการส่งต่อไปยัง Mote อื่นๆจนถึงปลายทางซึ่งเป็นวิธีที่มีประสิทธิภาพและใช้พลังงานน้อยกว่า

2. ระบบปฏิบัติการไทนี่ (TinyOS)

นอกเหนือจากอุปกรณ์ที่ต้องประกอบรวมกันเป็น Mote แล้วสิ่งที่สำคัญที่สุดอีกประการหนึ่งซึ่งจะขาดไม่ได้นั่นคือระบบปฏิบัติการซึ่งทำหน้าที่เป็นเสมือนสมองของ Mote ซึ่งระบบปฏิบัติการที่เป็นที่นิยมและใช้งานอย่างแพร่หลายระบบปฏิบัติการหนึ่งคือระบบปฏิบัติการไทนี่ คือระบบปฏิบัติการ (OS) ที่ออกแบบเพื่อใช้งานกับระบบเครือข่ายตรวจจับแบบไร้สายโดยเฉพาะ[www.tinyos.net] ซึ่งเป็นระบบปฏิบัติการขนาดเล็กที่ถูกพัฒนาขึ้นมาพร้อมกับความสามารถของการจัดการงาน (Task)และมีความสามารถในการเชื่อมต่อแบบไร้สายสามารถรันได้บนซีพียูหลายตระกูลและมีรูปแบบการทำงานเป็นระบบปฏิบัติการแบบตอบสนองต่อเหตุการณ์

(Event-based execution) เช่นเมื่อระบบกำลังทำการสั่งงานให้อ่านค่าจากเซนเซอร์ แล้วมีเหตุการณ์ Timer.fired (เหตุการณ์เมื่อตัวนับเวลาถึงกำหนด) ซึ่งเกิดจากการอินเทอร์รัพท์จากการตัวนับเวลาก็จะรันโปรแกรมในส่วนนั้นทันทีซึ่งช่วยให้ประหยัดพลังงานได้เป็นอย่างดีเนื่องจากในช่วงเวลาที่ซีพียูไม่ได้ประมวลผลนั้นจะเข้าสู่โหมดประหยัดพลังงานโดยอัตโนมัติซึ่งทำให้การใช้พลังงานโดยรวมของระบบลดลง ส่วนการจัดการงานจะเป็นแบบเข้าก่อนออกก่อน (FIFO) ทำให้งานทั้งหมดสามารถทำงานได้พร้อม ๆ กับความสามารถในการตอบสนองต่อเหตุการณ์ต่างๆได้เป็นอย่างดี

ภาษาที่ใช้ในการพัฒนาโปรแกรมบนระบบปฏิบัติระบบปฏิบัติการไทนี่ มีชื่อว่า (Network-Embedded-Systems-C: NesC) ซึ่งมีลักษณะของไวยกรณ์คล้ายคลึงกับภาษาซีทั่วไป



ภาพที่ 2 ขั้นตอนของการพัฒนาโปรแกรม

จากภาพที่ 2 แสดงขั้นตอนของการพัฒนาโปรแกรมด้วยระบบปฏิบัติการไทนี่ เริ่มจากโปรแกรมที่เขียนขึ้นด้วยภาษาเนสซีผ่านการแปลโดยตัวแปลภาษาของระบบปฏิบัติการไทนี่ ซึ่งในการแปลนั้นอาจจำเป็นต้องใช้กลุ่มของฟังก์ชันมาตรฐาน (Library) และข้อมูลที่จำเป็นมาประกอบในการแปลด้วยซึ่งหลังจากการแปลแล้วก็จะได้เป็นไฟล์ภาษาซีมาตรฐาน (ANSI C) ซึ่งจะต้องถูกแปลเป็นภาษาที่สามารถนำมารันบนซีพียูได้อีกครั้งหนึ่ง

3. รูปแบบของภาษาเนสซี

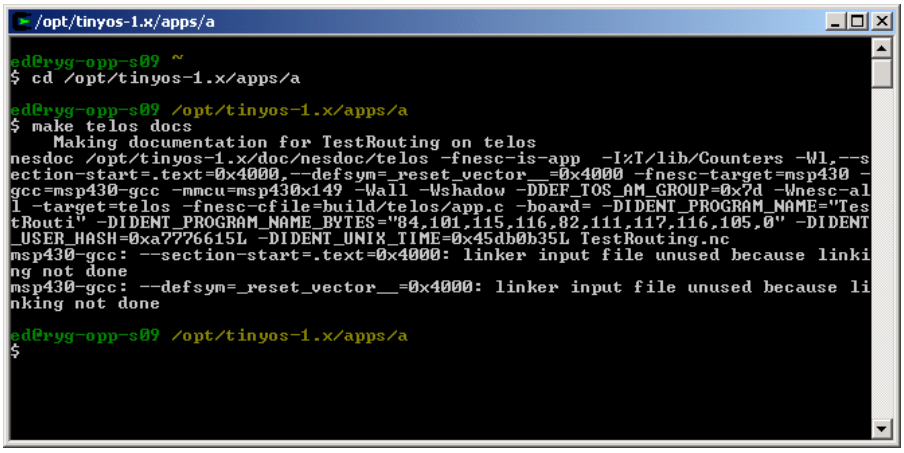
ภาษาเนสซี [Philip Levis,2006] เป็นการเขียนโปรแกรมมีโครงสร้างของภาษาคล้ายกับภาษาซี [Philip Levis,2006] องค์ประกอบของภาษาเนสซีมี 3 ส่วนหลักได้แก่กลุ่มของมอดูล, คอนฟิกูเรชั่นและอินเทอร์เฟสซึ่งมอดูลนั้นเป็นส่วนที่เก็บรหัส (Source code) ที่รันและคอนฟิกูเรชั่นคือส่วนที่เชื่อมต่อมอดูลเข้าด้วยกันโดยผ่านทางอินเทอร์เฟสซึ่งมอดูลจะถูกเชื่อมต่อกันด้วยไฟล์เชื่อมต่อ (Interface) ทุกไฟล์ของระบบปฏิบัติการไทนี่ จะมีนามสกุลเป็น .nc โดยมีหลักในการตั้งชื่อของไฟล์ตามตารางที่ 1 ดังนี้

ตารางที่ 1 แสดงรูปแบบของการตั้งชื่อไฟล์ต่างๆของระบบปฏิบัติการไทนี่

ประเภท	รูปแบบการตั้งชื่อ	ตัวอย่าง
อินเทอร์เฟส	ประกอบด้วยคำที่ขึ้นต้นของ	ADC
	อักษรตัวแรกเป็นตัวพิมพ์ใหญ่	Range
		SendMsg
		BombillaLocks
คอมโพเน้นท์	ประกอบด้วยคำที่ขึ้นต้นของ	Counter
	อักษรตัวแรกเป็นตัวพิมพ์ใหญ่	IntToRfm
	และตัวอักษรตัวสุดท้ายเป็น C	IntToRfmM
	หากเป็นคอนฟิกูเรชั่นและ	TimerC
	ตัวอักษรตัวสุดท้ายเป็น M หาก	TimerM
	เป็นมอดูล	UARTM
ไฟล์	นามสกุลของทุกไฟล์ที่เขียน	Counter.nc
	ด้วยภาษาเนสซีจะเป็น .nc	IntToRfm.nc
		IntToRfmM.nc
		TimerC.nc
แอพพลิเคชั่น	ประกอบด้วยคำที่มีความหมาย	CntToRfm
	ตามชื่อของระบบ	Chirp
		DemoTracking
		TestTinyAlloc

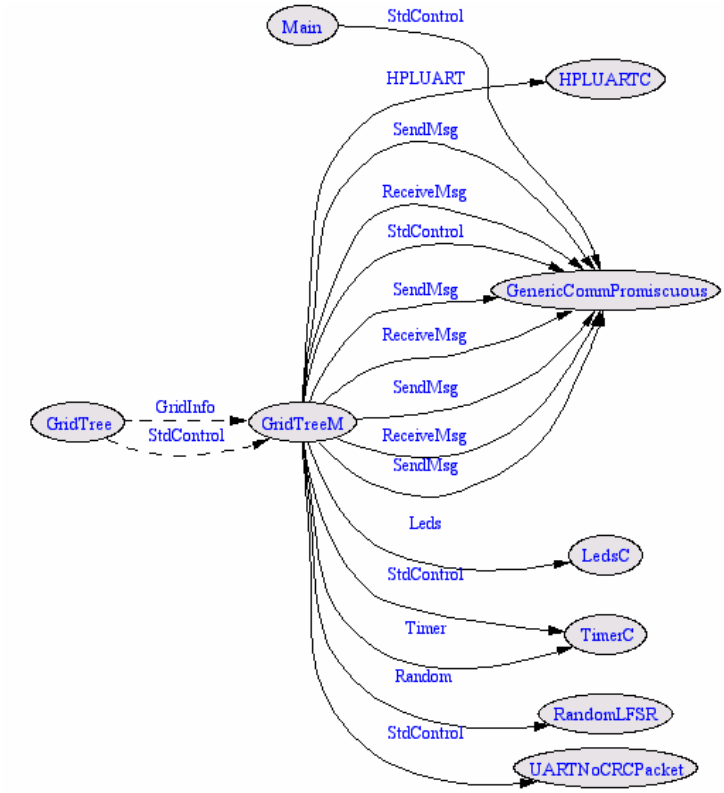
3.1 กราฟคอมโนเน้นท์

การพัฒนาโปรแกรมด้วยระบบปฏิบัติการไทนี่นั้นสิ่งที่สำคัญอีกประการหนึ่งคือการตรวจสอบการเชื่อมต่อของแต่ละคอมโนเน้นท์ (Wiring) เข้าด้วยกันนั้นถูกต้องตามที่ได้ออกแบบไว้หรือไม่ ซึ่งในระบบระบบปฏิบัติการไทนี่ได้มีเครื่องมือในการสร้างกราฟแสดงการเชื่อมต่อของแต่ละคอมโพเน้นท์โดยใช้คำสั่ง make <platform> docs



ภาพที่ 3 แสดงตัวอย่างการสร้างกราฟคอมโพเน้นท์

จากภาพที่ 3 แสดงตัวอย่างวิธีการสร้างกราฟคอมโพเน้นท์ซึ่งหลังจากที่ได้สร้างกราฟคอมโพเน้นท์แล้วจะได้รูปภาพซึ่งประกอบด้วยคอมโพเน้นท์ที่เชื่อมต่อกันโดยสามารถเลือกดูคอมโพเน้นท์ต่างๆหรือส่วนเชื่อมต่อ (Interface) เพื่อดูในรายละเอียดได้



ภาพที่ 4 แสดงตัวอย่างของกราฟคอมโพเน้นท์

ภาพที่ 4 แสดงตัวอย่างของกราฟคอมโพเน้นท์ที่ได้ (บางส่วน) โดยไฟล์ที่ได้จะอยู่ในรูปแบบของเว็บเพจหรือ HyperText Markup Language (HTML) ซึ่งแต่ละคอมโพเน้นท์สามารถลิ้งค์เพื่อดูรายละเอียดได้ว่าภายในคอมโพเน้นท์นั้นๆประกอบด้วยส่วนที่อินเทอร์เฟสโพรไวด์, อินเทอร์เฟสยูสและความหมายของแต่ละส่วน

4. งานที่เกี่ยวข้อง

ตารางที่ 2 แสดงการเปรียบเทียบระบบค้นหาตำแหน่ง

ชื่อระบบ	รูปแบบ	รูปแบบ	ความแม่นยำ	ประเภท
GPS	แยกศูนย์	สัญญาณดาวเทียม	1-5 เมตร	ภายนอก
E911/E112	รวมศูนย์	ระบบเครือข่ายโทรศัพท์	100 เมตร	ภายนอก
ActiveBadge	รวมศูนย์	อินฟราเรด	1 เมตร	ภายใน
ActiveBAT	รวมศูนย์	คลื่นวิทยุและคลื่นอัลตราโซนิค	5-10 ซ.ม.	ภายใน
Cricket	แยกศูนย์	คลื่นวิทยุและคลื่นอัลตราโซนิค	15 เซนติเมตร	ภายใน
SpotON	รวมศูนย์	คลื่นวิทยุ	1 เมตร	ภายใน
Smart Floor	รวมศูนย์	น้ำหนักและทิศทาง		ภายใน
Pinpoint 3D	รวมศูนย์	คลื่นวิทยุ	1-3 เมตร	ภายใน
RemoteEyes	รวมศูนย์	อินฟราเรดและกล้อง	1-3 เมตร	ภายใน
RADAR	Client	เครือข่าย Wifi	3-5 เมตร	ภายใน

จากตารางที่ 2 แสดงระบบค้นหาตำแหน่งภายในอาคารและภายนอกอาคารโดยมีรายละเอียดดังนี้

ระบบ Global Positioning System (GPS) เป็นระบบแบบแยกศูนย์ (Decentralize System) ใช้สำหรับภายนอกอาคารโดยหลักการของการรับสัญญาณจากสัญญาณจากดาวเทียมและวัดระยะ

เวลาจากเครื่องส่งสัญญาณจากดาวเทียมกับเครื่องรับสัญญาณอย่างน้อยสี่ดวงถึงจะคำนวณตำแหน่งลักษณะของ 3 มิติได้

E911/E112 เป็นระบบแบบรวมศูนย์ (Centralize System) สำหรับภายนอกอาคารใช้หลักการของการนับเวลาในการเดินทางไปและกลับของคลื่นวิทยุ (โทรศัพท์เคลื่อนที่) 3 สถานีฐานแล้วนำมาคำนวณหาตำแหน่งของเครื่องลูกข่ายโดยใช้สมการสามเหลี่ยม

ระบบ ActiveBadge [Roy Want., 1992] เป็นระบบแบบรวมศูนย์ (Centralize System) โดยใช้หลักการวัดเวลาในการเดินทางของคลื่นอัลตราโซนิคคลื่นอินฟราเรดที่ติดอยู่กับวัตถุทำการส่งสัญญาณออกมาทุกๆ 10 วินาทีเพื่อทำการเข้าจังหวะสัญญาณนาฬิกา (Synchronize) แล้วจึงส่งคลื่นอัลตราโซนิคเพื่อสถานีฐานได้รับก็จะทำการคำนวณเวลาในการเดินทางของคลื่นอัลตราโซนิคแล้วนำมาคำนวณหาตำแหน่งของวัตถุ

ระบบ ActiveBAT เป็นระบบแบบรวมศูนย์ (Centralize System) ใช้หลักการวัดความแตกต่างเวลาในการเดินทางของคลื่นวิทยุและคลื่นอัลตราโซนิคที่เดินทางจากวัตถุมาถึงสถานีฐานที่ติดตั้งเป็นโครงข่ายอยู่บนเพดานจำนวนอย่างน้อย 3 สถานีฐาน

ระบบ Cricket Version 1 [Nissanka B., 2000] เป็นระบบแบบแยกศูนย์ (Decentralize System) ใช้หลักการวัดความแตกต่างเวลาในการเดินทางของคลื่นวิทยุและคลื่นอัลตราโซนิคที่เดินทางจากสถานีฐานมาถึงวัตถุจำนวน 3 สถานีฐานแล้วนำมาสร้างสมการสามเหลี่ยมเพื่อคำนวณหาตำแหน่งของวัตถุ ซึ่งผลลัพธ์ที่ได้มีความแม่นยำประมาณ 15 เซนติเมตร

ระบบ Cricket Version 2 [Nissanka B., 2003] เป็นระบบแบบแยกศูนย์ (Decentralize System) ใช้เป็นการปรับปรุงความแม่นยำจาก Version 1 โดยทำการเพิ่มส่วนของการวัดความต่างของเฟสของคลื่นอัลตราโซนิคที่เดินทางมาถึงตัวรับคลื่นจำนวน 5 ตัวบนวัตถุและใช้ระบบปฏิบัติการไทนี่ซึ่งประกอบด้วยส่วนของ การรับ-ส่งคลื่นวิทยุ และส่วนของการรับ-ส่งคลื่นอัลตราโซนิค ซึ่งอุปกรณ์จะแบ่งออกเป็น 2 ประเภทคือส่วนที่ทำหน้าที่เป็น Beacon ซึ่งทำหน้าที่เสมือนเป็นจุดอ้างอิงติดตั้งอยู่ในตำแหน่งที่แน่นอนบนเพดานและอุปกรณ์รับสัญญาณซึ่งจะ

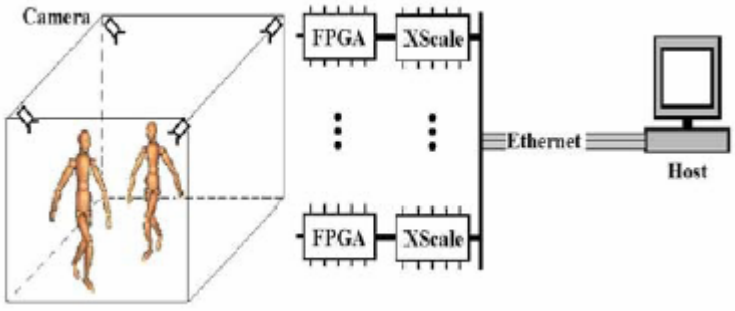
ติดอยู่กับวัตถุที่ต้องการหาพิกัดตำแหน่งโดยอุปกรณ์ในส่วนที่ทำหน้าที่เป็น Beacon จะส่งสัญญาณคลื่นวิทยุซึ่งประกอบด้วยข้อมูลพิกัดตำแหน่งของอุปกรณ์นั้นมาเป็นระยะเวลาคงที่ค่าหนึ่งแล้วตามด้วยการส่งคลื่นอัลตราโซนิค และส่วนของอุปกรณ์ที่ทำหน้าที่เป็น Listener จะทำการคำนวณระยะห่างระหว่าง Beacon และ Listener เพื่อหาตำแหน่งต่อไป

ระบบ SpotON [Hightower., 2000] เป็นระบบแบบรวมศูนย์ (Centralize System) ทำการวัดความเข้มของสัญญาณคลื่นวิทยุ (Receive Signal Strength Indicator: RSSI) ที่ได้รับจาก RFID ที่ 3 สถานีฐานแล้วนำค่าที่วัดได้มาทำการสร้างสมการวงกลม 3 วงเพื่อหาตำแหน่งของวัตถุ ซึ่งผลลัพธ์ที่ได้มีความแม่นยำประมาณ 1 ตารางเมตร

ระบบ SmartFloor [Jeffrey., 2000] เป็นระบบแบบรวมศูนย์ (Centralize System) เป็นระบบใช้เทคนิคการวัดน้ำหนักและระยะเวลาของการกดที่ตัวตรวจจับเพื่อนำข้อมูลที่ได้ไปทำการค้นหาในฐานข้อมูลว่าเป็นบุคคลใดโดยค่าที่ทำการวัดได้แก่น้ำหนักของการกดที่เท้าซ้ายและขวา, ระยะเวลาของการกด โดยจะทำการเก็บข้อมูลไว้ก่อนหน้านี้ด้วยข้อมูลในหลายตัวแปรเช่นการเปลี่ยนชนิดของรองเท้าเป็นต้น

ระบบ Pinpoint 3DiD เป็นระบบค้นหาตำแหน่งโดยเป็นผลิตภัณฑ์ในเชิงพาณิชย์ ใช้หลักการของการวัดเวลาในการเดินทางของคลื่นในระบบเครือข่ายแบบไร้สายมาตรฐาน 802.11 ซึ่งผลลัพธ์ที่ได้มีความแม่นยำประมาณ 1-3 เมตร

RemoteEyes: [Shen., 2004]



ภาพที่ 5 ระบบ RemoteEyes

จากภาพที่ 5 ระบบจะประกอบด้วยกล้องชนิดขาว-ดำและอุปกรณ์ส่งสัญญาณอินฟราเรดที่ติดตั้งอยู่ที่วัตถุและส่งสัญญาณที่ประกอบด้วยข้อมูลรหัสของวัตถุขนาด 8 บิต ระบบสามารถหาตำแหน่งของวัตถุโดยใช้สัญญาณที่ได้จากกล้องที่ความเร็ว 60 Frame per Second (FPS) มาทำการหาตำแหน่งของวัตถุโดยบริเวณที่รับสัญญาณอินฟราเรดได้จะเป็นจุดสีขาวและพัฒนาบน Field Programmable Gate Array (FPGA)

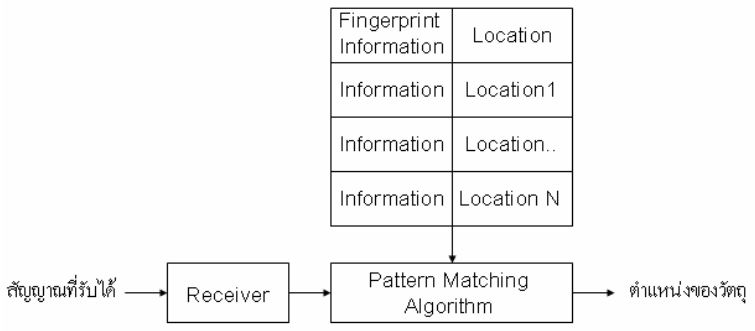
RADAR: [Paramvir., 2000] เป็นระบบแบบรวมศูนย์ (Centralize System) ทำการวัด RSSI ในระบบเครือข่ายแบบไร้สาย (Wireless LAN) มาตรฐาน 802.11b โดยใช้วิธีเทียบเคียงจากฐานข้อมูล (Database Correlation Method: DCM) และใช้วิธีในการค้นหาด้วย Nearest Neighbor(s) in Signal Space (NNSS) ซึ่งผลลัพธ์ที่ได้มีความแม่นยำประมาณ 3-5 เมตร

5. วิธีการหาตำแหน่ง

วิธีเทียบเคียงจากฐานข้อมูล (Database Correlation Method: DCM)

เป็นวิธีการคำนวณตำแหน่งโดยใช้หลักการคือในแต่ละตำแหน่งของวัตถุที่ส่งสัญญาณไปที่โหนดจะให้ค่าที่เป็นเอกลักษณ์ซึ่งเป็นค่าลักษณะการเปลี่ยนแปลงของกำลังของสัญญาณ (RSSI) ที่แตกต่างกันเนื่องจากการเดินทางจากหลายเส้นทางของสัญญาณจากวัตถุไปที่โหนดซึ่งแต่ละตำแหน่งของวัตถุนั่นจะให้ค่าที่เป็นเอกลักษณ์ที่แตกต่างกันทำให้ทราบตำแหน่งของวัตถุได้เมื่อทราบค่าที่เป็นเอกลักษณ์ที่ทำการวัดเก็บไว้ในฐานข้อมูลก่อนหน้านี้แล้ว เมื่อต้องการค้นหาตำแหน่งก็เพียงรับค่าสัญญาณแล้วทำการเปรียบเทียบค่าที่วัดได้กับค่าที่เก็บไว้ในฐานข้อมูลเมื่อทราบ

ข้อดีของระบบนี้ได้แก่การที่ต้องการแค่เพียง 1 โหนดเพื่อรับสัญญาณจากวัตถุมาทำการเปรียบเทียบกับค่าในฐานข้อมูลแต่ข้อเสียของระบบนี้ได้แก่สภาพแวดล้อมที่ทำการเก็บข้อมูลนั้นต้องเป็นสภาพแวดล้อมที่มีการเปลี่ยนแปลงค่อนข้างน้อยเนื่องจากค่าที่วัดได้นั้นอาจจะไม่ตรงกับสภาพแวดล้อมที่เปลี่ยนแปลงไป ทำให้ต้องมีการปรับปรุงฐานข้อมูลอย่างสม่ำเสมอทำให้เพิ่มค่าใช้จ่ายในส่วนนี้ พิจารณาจากภาพที่ 6 แสดงวิธีการค้นหาตำแหน่งของด้วยวิธี DCM



ภาพที่ 6 แสดงวิธีการค้นหาตำแหน่งของด้วยวิธี DCM

วิธีใช้ตัวนับตามระยะเวลา (Time of Arrival: TOA)

สำหรับวิธีการ คำนวณตำแหน่งแบบวิธีใช้ตัวนับตามระยะเวลาคือการหาระยะห่างระหว่างวัตถุและโหนดหลักการคือเราสามารถหาระยะห่าง R_i ซึ่งเป็นระยะทางระหว่างวัตถุกับ โหนด ที่ i จากการคำนวณตามสมการที่ (1.1)

$$R_i = ct_i \tag{1.1}$$

เมื่อ c คือค่าความเร็วแสงมีค่าเท่ากับ $3*10^8$ เมตรต่อวินาที

และ t_i คือระยะเวลาในการเดินทางสัญญาณ (TOA) ระหว่างวัตถุไปที่โหนด i แต่วิธีการนี้ต้องมีวงจรในการสร้างความสอดคล้องของสัญญาณนาฬิการะหว่างวัตถุกับโหนดจึงจะทำให้สามารถวัดค่า t_i ได้ พิจารณาจากสมการมาตรฐานของวงกลมที่รัศมี R คือ

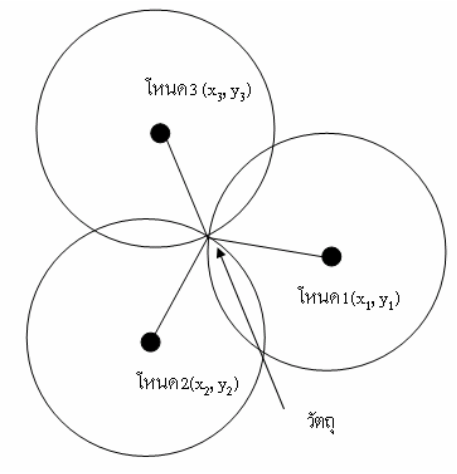
$$R_i = \sqrt{(X_i - x)^2 + (Y_i - y)^2 + (Z_i - z)^2} \quad \text{โดย } i = 1,2,3,...N \tag{1.2}$$

เมื่อ (X_i, Y_i, Z_i) เป็นพิกัดของโหนด i

และ (x, y, z) เป็นพิกัดของตำแหน่งของวัตถุที่ต้องการทราบตำแหน่ง

พิจารณาจากภาพที่ 7 แสดงการหาตำแหน่งของวัตถุด้วยวิธีนี้ซึ่งข้อเสียของวิธีการค้นหาตำแหน่งด้วยวิธีนี้ได้แก่การต้องการความสอดคล้องกันของสัญญาณนาฬิการะหว่างวัตถุและโหนด

จะพบว่าเมื่อนับระยะเวลาของสัญญาณจากวัตถุไปที่โหนดผิดพลาด 1 ไมโครวินาทีจะให้วัดระยะทาง ผิดพลาดถึง 300 เมตร



ภาพที่ 7 แสดงการหาตำแหน่งของวัตถุด้วยวิธีใช้ตัวนับตามระยะเวลา

วิธีการวัดมุมจากทิศทางของสัญญาณ (Angle of Arrival: AOA)

เป็นวิธีที่ใช้ค่ามุมของสัญญาณที่ส่งมาจากวัตถุโดยเราสามารถค้นหาตำแหน่งของวัตถุจากค่ามุมที่ได้รับที่โหนดจำนวน 2 มุมในแต่ละโหนดและเมื่อนำข้อมูลที่ได้รับมาสร้างสมการเส้นตรงทำให้เราสามารถหาตำแหน่งของวัตถุได้ พิจารณาจาก ภาพที่ 6 แสดงการหาตำแหน่งด้วยวิธี AOA

$$(y - y_i) = \tan(\theta_i)(x - x_i) \tag{1.12}$$

$$(y - y_j) = \tan(\theta_j)(x - x_j) \tag{1.13}$$

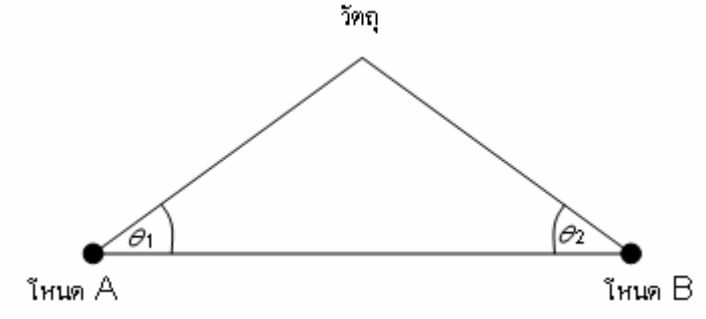
โดยที่ $\theta_i = \tan^{-1}\left(\frac{|y_i - y|}{|x_i - x|}\right)$ *และ* $\theta_j = \tan^{-1}\left(\frac{|y_j - y|}{|x_j - x|}\right)$

และ

(x, y) เมื่อ เป็น พิกัดที่ต้องการทราบของวัตถุที่ต้องการทราบตำแหน่ง

(x_i, y_i) เป็น พิกัดที่ต้องการทราบของ โหนด ตามแนวแกน x และ y จุดที่ i

(x_j, y_j) เป็น พิกัดที่ต้องการทราบของ โหนด ตามแนวแกน x และ y จุดที่ j



ภาพที่ 8 แสดงการหาตำแหน่งด้วยวิธี AOA

จากภาพที่ 8 แสดงวิธีการวัดมุมจากทิศทางของสัญญาณคือต้องการจำนวนโหนดจำนวนเพียง 2 โหนดเพื่อการคำนวณตำแหน่งซึ่งน้อยกว่าวิธีการคำนวณตำแหน่งด้วยวิธีใช้ตัวนับตามระยะเวลาและวิธีใช้ความแตกต่างของตัวนับตามระยะเวลาซึ่งต้องมีจำนวนโหนดอย่างน้อยจำนวน 3 โหนดขึ้นไปแต่ข้อเสียของการค้นหาตำแหน่งด้วยวิธีนี้คือการเพิ่มค่าใช้จ่ายในการปรับแต่งเสาอากาศเพื่อให้วัดค่าได้อย่างถูกต้องอย่างสม่ำเสมอรวมทั้งให้ค่าความถูกต้องค่อนข้างต่ำในสภาพแวดล้อมในชุมชนเมืองที่มีสิ่งปลูกสร้างจำนวนมากทำให้เกิดเส้นทางของสัญญาณหรือการขาดเส้นทางในแนวเส้นสายตา (Line of Sign: LOS) ส่งผลต่อการวัดมุมที่ผิดพลาดมากยิ่งขึ้น

วิธีใช้ความแตกต่างของตัวนับตามระยะเวลา (Time Difference of Arrival: TDOA)

สำหรับวิธีใช้ความแตกต่างของตัวนับตามระยะเวลานั้นเพื่อแก้ปัญหาด้านความถูกต้องของเวลาที่ต้องสอดคล้องกันระหว่างวัตถุและโหนดโดยเนื่องจากสัญญาณเคลื่อนที่ด้วยความเร็วแสงทำให้พบว่าในกรณีที่วัดเวลาผิดพลาดเพียง 1 ไมโครวินาทีทำให้เกิดข้อผิดพลาดในการวัดระยะทางถึง 300 เมตรดังนั้นเพื่อแก้ปัญหาเรื่องการสอดคล้องของสัญญาณนาฬิกาทำเกิดแนวคิดแทนที่จะทำการนับเวลาที่มาถึงโหนดเราจะทำการนับเวลาที่แตกต่างกันที่รับได้จากโหนดจำนวน 2 คู่เพื่อนำข้อมูลมาสร้างสมการ Hyperbolic [Y.T.Chan., 1994] เพื่อหาตำแหน่งของวัตถุโดยพิจารณาการหาค่าตำแหน่งของโหนดได้ดังนี้

ใช้หลักการของการนับเวลาเวลาที่แตกต่างกันที่นับได้จากคู่ของสัญญาณจากวัตถุไปที่โหนดตำแหน่งที่ i และตำแหน่งที่ j เพื่อสร้างของสมการ Hyperbolic พิจารณาความสัมพันธ์ของผลต่างของระยะทางระหว่าง โหนด ตำแหน่งที่ i และตำแหน่งที่ j สมการที่ (1.3) ดังนี้

$$R_{i,j} = ct_{i,j} = R_i - R_j \tag{1.3}$$

โดยที่ $R_{i,j}$ เป็นผลต่างของระยะทางระหว่าง โหนด ตำแหน่งที่ i และตำแหน่งที่ j และ $t_{i,j}$ เป็นความแตกต่างของเวลาของสัญญาณจากวัตถุไปที่ โหนด ตำแหน่งที่ i และตำแหน่งที่ j และ c คือค่าความเร็วแสงมีค่าเท่ากับ $3*10^8$ เมตรต่อวินาที

และ R_i คือระยะห่างระหว่าง โหนด ที่ตำแหน่งที่พิกัด (x, y) กับวัตถุซึ่งมีพิกัดเป็น (X_i, Y_i) ตามสมการที่ (1.4) ได้ดังนี้

$$R_i = \sqrt{(X_i - x)^2 + (Y_i - y)^2} = \sqrt{X_i^2 + Y_i^2 - 2X_i x - 2Y_i y + x^2 + y^2} \tag{1.4}$$

และสามารถหาความสัมพันธ์ของผลต่างของระยะทางได้จากวัตถุไปที่ โหนด ที่ i, j โดยมีความสัมพันธ์ตามสมการที่ (1.5) ดังนี้

$$R_{i,j} = \sqrt{(X_i - x)^2 + (Y_i - y)^2} - \sqrt{(X_j - x)^2 + (Y_j - y)^2} \tag{1.5}$$

โดยที่ $R_{i,j}$ เป็นผลต่างของระยะทางระหว่างวัตถุตำแหน่งที่ i และตำแหน่งที่ j และ (X_i, Y_i) และ (X_j, Y_j) เป็นพิกัดของ โหนด ที่ i และ j และ (x, y) เป็นตำแหน่งของวัตถุที่ต้องการทราบตำแหน่ง

เราสามารถหาคำตอบของสมการได้เมื่อจำนวนสมการเท่ากับจำนวนตัวแปรที่ไม่ทราบค่า เนื่องจากสมการที่ (1.5) ทำการแก้สมการค่อนข้างยากทำให้ต้องจัดรูปแบบของสมการใหม่ดังนี้จาก

$$R_i^2 = (R_{i,j} + R_1)^2 \tag{1.6}$$

ทำให้ทำการจัดรูปของสมการที่ (1.5) เขียนได้เป็น

$$R_{i,1}^2 + 2R_{i,1}R_1 + R_1^2 = X_i^2 + Y_i^2 - 2X_i x - 2Y_i y + x^2 + y^2 \tag{1.7}$$

ทำการลบค่าของสมการ (1.6) ที่ $i=1$ ออกจากสมการที่ (1.7) จะได้เป็น

$$R_{i,1}^2 + 2R_{i,1}R_1 = X_i^2 + Y_i^2 - 2X_{i,1}x - 2Y_{i,1}y \tag{1.8}$$

เมื่อ $X_{i,1}$ และ $Y_{i,1}$ มีค่าเท่ากับ $(X_i - X_1)$ และ $(Y_i - Y_1)$ ตามลำดับ

สมการที่ (1.8) สามารถหาคำตอบของสมการได้ดังนี้

$$\begin{bmatrix} x \\ y \end{bmatrix} = -\begin{bmatrix} X_{2,1}Y_{2,1} \\ X_{3,1}Y_{3,1} \end{bmatrix}^{-1} \times \left\{ \begin{bmatrix} R_{2,1} \\ R_{3,1} \end{bmatrix} R_1 + \frac{1}{2}\begin{bmatrix} R_{2,1}^2 - K_2 + K_1 \\ R_{3,1}^2 - K_3 + K_1 \end{bmatrix} \right\} \tag{1.9}$$

เมื่อ

$$K_1 = X_1^2 + Y_1^2$$
$$K_2 = X_2^2 + Y_2^2$$
$$K_3 = X_3^2 + Y_3^2$$

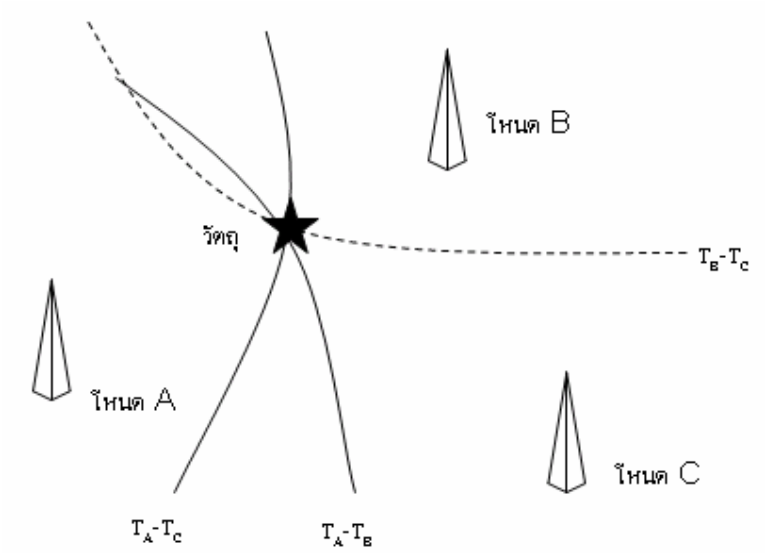
เราสามารถทำการหาค่า x และ y ได้ในเทอมของ R_1 ได้ โดยการแทนค่าของ x และ y ในสมการที่ (1.9) ไปแทนค่าในสมการที่ (1.8) และจัดรูปแบบสมการที่ (1.8) ในเทอมของ R_1 ในรูปแบบของสมการ Quadratic ดังนี้

$$aR_1^2 + bR_1 + c = 0 \tag{1.10}$$

สามารถหาคำตอบของสมการตามสูตรได้เป็น

$$R_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \tag{1.11}$$

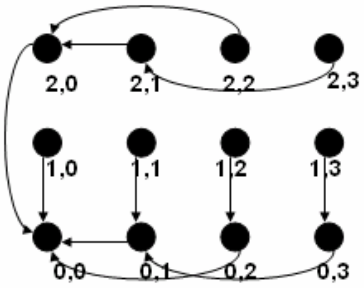
เลือกใช้รากของสมการเพียงค่าเดียวเนื่องจาก R_1 เป็นระยะห่างระหว่างวัตถุและโหนด ที่ 1 ทำให้รากของสมการมีค่าเป็นบวกเพียงค่าเดียวแล้วนำค่าที่ได้ไปแทนค่าในสมการ (1.9) พิจารณาภาพที่ 9 แสดงการคำนวณหาตำแหน่งด้วยวิธี TDOA



ภาพที่ 9 แสดงวิธีใช้ความแตกต่างของตัวนับตามระยะเวลา

โปรโตคอลการสื่อสารแบบมัลติฮอป

ในงานวิจัยนี้ใช้โปรโตคอลการสื่อสารแบบมัลติฮอปแบบกริดซึ่งเป็นโปรโตคอลที่ต้องทราบเส้นทางการส่งข้อมูลล่วงหน้า (Table Driven) โหนดจะถูกติดตั้งในลักษณะเป็นแบบกริดมีขนาด MxN ซึ่งจะมีชื่อระบุตำแหน่งเป็น (i, j) โดย i=0 ถึง M-1 และ j=0 ถึง N-1 และโหนดหลัก (Root Node) อยู่ที่ตำแหน่ง 0, 0 การแบ่งโหนดจะแบ่งออกเป็น 2 กลุ่มๆแรกคือ Low-neighbors คือกลุ่มของโหนดที่มีระยะห่างไปทางด้านที่มีตำแหน่งต่ำกว่าโหนดตนเองและกลุ่มที่สองคือ High-neighbors คือกลุ่มของโหนดที่มีระยะห่างไปทางด้านที่มีตำแหน่งสูงกว่าโหนดตนเองแต่ละโหนดจะพิจารณาโหนดข้างเคียงจาก tree hop คือจำนวน Hop (H) ที่ห่างจากตนเองจากภาพที่ 10



ภาพที่ 10 แสดงเส้นทางการส่งข้อมูลที่ค่า H=2

โหนดอยู่ในตำแหน่งกึ่งกลางของพื้นที่กริดโหนดนั้นก็จะมีทั้ง Low-neighbors (i, j-H) และ (i-H, j) และ High-neighbors (i, j+H) และ (i+H, j) ดังนั้นโหนด (i, j-H), (i-H, j), (i, j+H) และ (i+H, j) ก็คือโหนดที่มีระยะห่างจากโหนดนั้นๆเท่ากับ H ในทิศเหนือ, ใต้, ตะวันออก และตะวันตก ตามลำดับ

หากโหนดอยู่ที่ตำแหน่งขอบของพื้นที่กริดโหนดนั้นก็จะไม่มีโหนดข้างเคียงทิศใดทิศหนึ่ง เช่นโหนดแต่จะพิจาณาให้โหนด (i, max(j-H,0)),(max(i-H,0), j),(i, min(j+H, N-1)) หรือ (min(i+H, M-1),j) เป็นโหนดข้างเคียงแทนโดยทุกโหนดจะทำการติดต่อเฉพาะโหนดที่เป็นโหนดข้างเคียงเท่านั้นคือหากโหนด (i, j) ได้รับข้อความจากโหนดที่ไม่ใช่โหนดข้างเคียงก็จะไม่สนใจข้อความนั้น

การทำ Spanning tree เมื่อโหนดต้องการส่งข้อมูลโหนดนั้นก็จะส่งข้อมูลไปที่โหนดปลายโดยการส่งแบบกระจาย (Broadcast) ไปทุกโหนดซึ่งทุกโหนดที่ได้รับข้อความจะทำเช่นนี้ทุกโหนดจนข้อความนั้นไปถึงโหนดหลักซึ่งข้อความนั้นประกอบด้วยฟิลด์

data (x, y, t)

เมื่อ x, y คือชื่อโหนดปลายทางและ t คือข้อความที่ต้องการส่ง

ซึ่งโหนดที่ได้รับข้อความ data(u, v, t) ก็จะทำการพิจารณาว่าจะนำไปประมวลผลต่อไปหรือไม่สนใจข้อความนั้นซึ่งจะพิจาณาว่าตนเองนั้นเป็นโหนดปลายทางหรือไม่ ($u \neq i$ หรือ $v \neq j$) หรือตนเองมีข้อมูลที่อยู่ของโหนดปลายทางหรือไม่หากโหนดตนเองนั้นเป็นโหนดปลายทาง ($u = i$ และ $v = j$) และมีที่อยู่ของโหนดปลายทาง (x, y) และตำแหน่งของตนเองนั้นไม่ได้เป็นโหนดหลักก็จะทำการส่งต่อข้อความไปที่โหนดปลายทางโดยการ Broadcast ข้อความ (data(x, y, t)) นั้นออกไป แต่หากตนเองเป็นโหนดปลายทางก็จะนำข้อมูลมาประมวลผลต่อไป

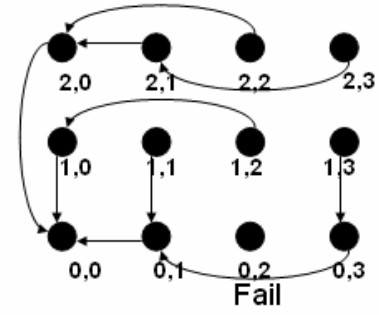
การเลือกที่อยู่ของโหนดปลายทางคือการสร้างเส้นทางจากโหนดใดๆไปที่โหนดหลักโดยทุกโหนดจะเลือก Low-neighbors จำนวนหนึ่งโหนดเป็นที่อยู่ของโหนดปลายทางดังนั้นการส่งข้อมูลจะมีระยะทาง (H) สั้นลงทุกๆครั้งที่มีการส่งข้อมูล หากทุก Low-neighbors นั้นใช้งานไม่ได้โหนดนั้นก็จะเลือกเอา High-neighbors จำนวนหนึ่งโหนดมาเป็นที่อยู่ของโหนดลายทาง ซึ่งหากโหนดนั้นเป็น High-neighbors ก็จะมีการเก็บค่า inversion ดังนั้นทุกโหนดจำเป็นต้องทำการรักษาข้อมูลที่อยู่ของโหนดปลายทางและค่า inversion ด้วย

การรักษาเส้นทางการส่งข้อมูลเริ่มจากโหนดนั้นมีข้อมูลที่อยู่ปลายทางโหนดนั้นจะส่งข้อความเพื่อการเชื่อมต่อแบบ Broadcast ทุกๆช่วงเวลาที่กำหนด หากโหนดนั้นไม่ได้รับข้อความเพื่อการเชื่อมต่อเป็นเวลาหนึ่งโหนดนั้นก็จะพิจาณาว่าโหนดตนเองนั้นไม่มีข้อมูลที่อยู่ปลายทาง ข้อความเพื่อการเชื่อมต่อนั้นประกอบด้วยสามฟิลด์ได้แก่

connected (i, j, c)

เมื่อ (i, j) คือตำแหน่งของโหนดที่ทำการส่งข้อความเพื่อการเชื่อมต่อที่ Broadcast ออกไป และ c คือ inversion

หากโหนดตนเองเป็นโหนดหลัก (0,0) ก็จะทำการส่งข้อความเพื่อการเชื่อมต่อทุกๆช่วงเวลาหากโหนดอื่นๆได้รับข้อความนี้และมีที่อยู่ของโหนดปลายทางก็จะทำการส่งข้อความเพื่อการเชื่อมต่อ (connected(i, j, c)) แบบ Broadcast ทุกๆช่วงเวลา



ภาพที่ 11 แสดงเส้นทางการส่งข้อมูลเมื่อโหนด 0, 2 ใช้งานไม่ได้ (H=2)

จากภาพที่ 11 แสดงตัวอย่างของเส้นทางการส่งข้อมูลเมื่อโหนดตำแหน่ง 0, 2 ใช้งานไม่ได้ โหนดข้างเคียงจะเปลี่ยนที่อยู่โหนดปลายทางไปที่โหนดในกลุ่มของ Low-neighbors แทน

การคำนวณค่า Inversion หากโหนด (i, j) ได้รับข้อความเพื่อการเชื่อมต่อ connected(x, y, d) และเลือกโหนด (x, y) เป็นที่อยู่โหนดปลายทางแล้วก็จะทำการคำนวณค่า inversion โดยหากโหนด (x, y) นั้นเป็น Low-neighbors จะทำการกำหนด inversion เป็น d แต่หากโหนด (x, y) นั้นเป็น High-neighbors จะทำการกำหนด inversion เป็น d+1

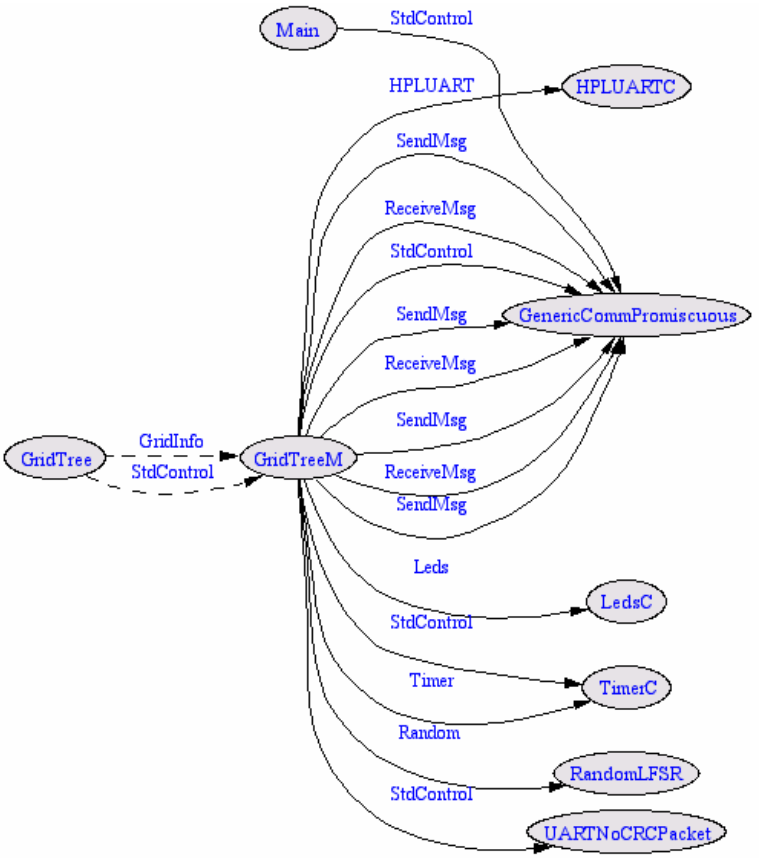
การค้นหาที่อยู่โหนดปลายทางเริ่มจากทุกโหนดไม่มีที่อยู่ของโหนดปลายทางและได้รับข้อความเพื่อการเชื่อมต่อ connected(x, y, d) และ (x, y) เป็นโหนดข้างเคียงก็จะทำการคำนวณค่า inversion โดยหากค่าที่คำนวณได้มีค่าน้อยเดิมก็จะเลือกโหนด (x, y) เป็นให้เป็นที่อยู่โหนดปลายทาง

การรักษาข้อมูลที่อยู่ของโหนดปลายทางเมื่อโหนด (i, j) มีที่อยู่โหนดปลายทาง (x, y) แล้วแต่ได้รับข้อความการเชื่อมต่อ connected(x, y, d) โหนดก็จะคำนวณค่า inversion จาก d หากค่าที่คำนวณได้น้อยกว่าหรือเท่ากับ cmax ก็จะเลือกเอาโหนด (x, y) นั้นเป็นโหนดปลายทางและปรับปรุงค่า inversion แต่หากคำนวณแล้วค่า inversion มากกว่าเดิมก็จะถือว่าไม่มีข้อมูลที่อยู่โหนดปลายทาง

การหายไปของที่อยู่โหนดปลายทางสาเหตุที่ทำให้ที่อยู่ของโหนดนั้นหายไปมี 2 สาเหตุ ได้แก่โหนด (i, j) ได้รับข้อความเชื่อมต่อเป็น connected(x, y, cmax) และ (x, y) นั้นมาจาก High-neighbors ซึ่งจะทำให้ค่า inversion มีค่ามากกว่า cmax และอีกสาเหตุหนึ่งคือโหนด (i, j) ไม่ได้รับข้อความเพื่อการเชื่อมต่อใดๆเป็นระยะเวลานานช่วงเวลาหนึ่ง

การเปลี่ยนข้อมูลที่อยู่ของโหนดปลายทางเมื่อโหนด (i, j) มีข้อมูลที่อยู่โหนดปลายทาง (x, y) แต่ได้รับ connected(u, v, f) เมื่อ $u \neq x$ และ $v \neq y$ แต่ u, v นั้นเป็นโหนดข้างเคียงก็จะทำการคำนวณค่า inversion จาก f หากค่า f น้อยกว่ามันก็จะถือเอา u, v เป็นที่อยู่โหนดปลายทาง

ในการคำนวณหาเส้นทางนั้นจะมีโปรแกรม 2 ส่วนคือการคำนวณหาที่อยู่ของโหนดปลายทางและโปรแกรมสำหรับเก็บข้อมูลระยะห่างระหว่างวัตถุกับโหนดแล้วส่งให้กับโหนดหลัก ในส่วนโปรแแกรมแรกนั้นแบ่งการทำงานออกเป็นอีกสองส่วนคือส่วนแรกคำนวณเวลาเมื่อไม่ได้รับข้อความเชื่อมต่อในช่วงเวลาหนึ่งและการส่งข้อความเพื่อการเชื่อมต่ออีกส่วนหนึ่งคือการรับข้อความเชื่อมต่อและจะพิจารณาข้อความนั้นว่าจะเป็นโหนดปลายทางได้หรือไม่ส่วนโปรแกรมที่สองนั้นจะคำนวณระยะห่างระหว่างวัตถุกับโหนดตนเองแล้วส่งให้กับโหนดหลักซึ่งโหนดอื่นๆที่ได้รับจะทราบว่าเป็นข้อความเพื่อส่งข้อมูลให้โหนดหลักก็จะไม่นำมาพิจารณาเพื่อเลือกให้เป็นที่อยู่โหนดปลายทาง

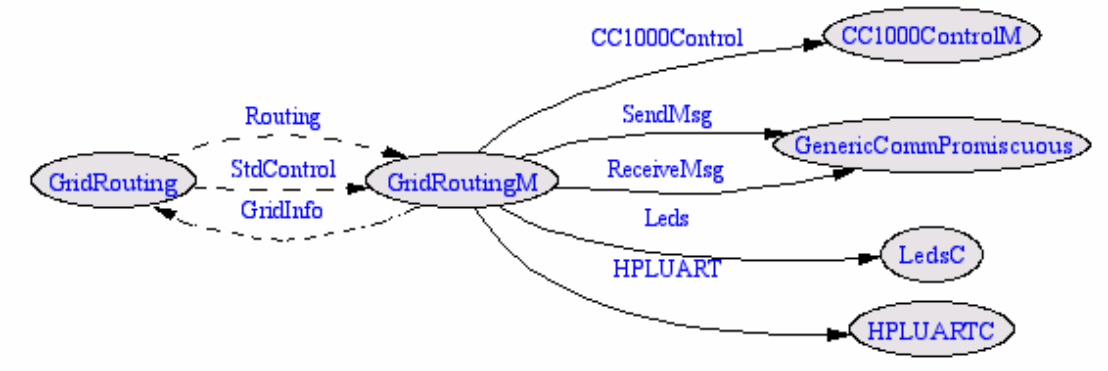


ภาพที่ 12 แสดงคอนฟิกูเรชั่นที่ทำหน้าที่ค้นหาเส้นทาง

จากภาพที่ 12 แสดงคอนฟิกูเรชั่นที่ทำหน้าที่ค้นหาเส้นทางซึ่งประกอบด้วยมอดูลต่างๆ ได้แก่

1. GridTree ทำหน้าที่เชื่อมต่อมอดูลต่างๆเข้าไว้ด้วยกัน
2. GridTreeM ทำหน้าที่รักษาเส้นทางการเชื่อมต่อของโหนด
3. RandomLFSR ทำหน้าที่สร้างค่าสุ่มเพื่อใช้ในการรอใช้ช่องสัญญาณในกรณีที่ช่องสัญญาณไม่ว่าง

และส่วนอื่นๆดังที่อธิบายแล้วข้างต้น



ภาพที่ 13 แสดงคอนฟิกูเรชั่นที่ทำหน้าที่รับส่งข้อมูล

จากภาพที่ 13 แสดงคอนฟิกูเรชั่นที่ทำหน้าที่ควบคุมการสื่อสารแบบมัลติฮอปซึ่งประกอบด้วยมอดูลต่างๆได้แก่

1. GridRouting ทำหน้าที่เชื่อมต่อมอดูลต่างๆเข้าไว้ด้วยกัน
2. GridRoutingM ทำหน้าที่พิจารณาหาที่อยู่ของโหนดปลายทางและส่งข้อมูลให้โหนดหลัก
3. CC1000ControlM ทำหน้าที่กำหนดค่าเริ่มต้นให้กับ RF Module เช่นค่าอัตราในการส่งข้อมูล (BAUDRATE), ความถี่ใช้งานและกำลังส่ง

และส่วนอื่นๆดังที่อธิบายแล้วข้างต้น

อุปกรณ์และวิธีการ

อุปกรณ์

ฮาร์ดแวร์

1. เครื่องคอมพิวเตอร์ 4 ชุดประกอบด้วยอุปกรณ์ดังต่อไปนี้
   - ซีพียู (CPU) เพนเทียม IV ความเร็ว 2.4 MHz
   - หน่วยความจำหลัก 2 GB

- ฮาร์ดดิสก์ขนาด 80 GB

2. WSN Board
3. อุปกรณ์รับ-ส่งคลื่นอัลตราโซนิค

ซอฟต์แวร์

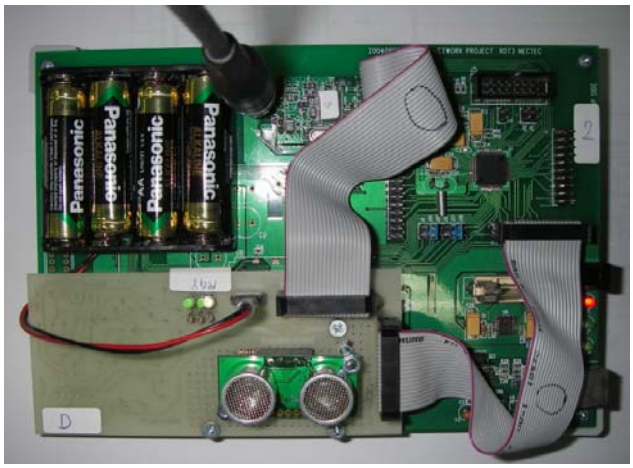
1. ระบบปฏิบัติการวินโดวส์
2. คอมไพเลอร์แบบ GNU C, C++
3. ระบบปฏิบัติการไทนี่ Version 1.1.7
4. โปรแกรม Virtual COM port (VCP) driver

วิธีการ

วิธีการของงานวิจัยนี้ใช้หลักการของการนับระยะเวลาในการเดินทางที่แตกต่างกันของคลื่นอัลตราโซนิคกับคลื่นวิทยุซึ่งความเร็วที่แตกต่างกันนั้นจะแปรผันตรงตามระยะทาง โดยติดตั้งอุปกรณ์ส่งสัญญาณอัลตราโซนิคและอุปกรณ์ส่งคลื่นวิทยุไว้ที่อุปกรณ์ที่ต้องการค้นหาตำแหน่งให้ทำการส่งสัญญาณมาที่โหนดที่ติดตั้งระบบปฏิบัติการไทนี่ซึ่งติดตั้งอยู่บนเพดานและทำการส่งค่าที่นับได้มาที่ศูนย์กลางเพื่อทำการคำนวณตำแหน่งของวัตถุต่อไป

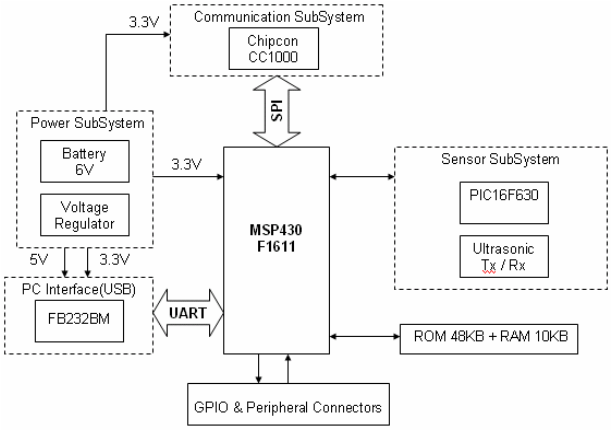
ตำแหน่งของการติดตั้งโหนดนั้นจำเป็นที่จะต้องระบุตำแหน่งให้ถูกต้องและแม่นยำ เนื่องจากต้องนำตำแหน่งของโหนดมาเพื่อประกอบการคำนวณหาตำแหน่งของวัตถุด้วยโดยอาจนำเอาตำแหน่งดังกล่าวมาประกอบกับแบบห้องหรือแบบอาคารเพื่อประกอบการแสดงผลของตำแหน่งของวัตถุนั้นๆ โดยการวัดระยะห่างระหว่างวัตถุกับโหนดที่ทราบตำแหน่งโดยที่วัตถุส่งสัญญาณอัลตราโซนิคนาน 1 มิลลิวินาทีไปพร้อมกับคลื่นวิทยุที่ประกอบด้วยข้อมูลของวัตถุได้แก่ รหัสประจำตัวของวัตถุนั้นและชนิดของข้อความที่ส่งเมื่อโหนดได้รับสัญญาณแล้วก็จะทำการวัดความแตกต่างของทั้งสองสัญญาณเนื่องจากสัญญาณซึ่งอัลตราโซนิคเดินทางด้วยความเร็วประมาณ 345 เมตรต่อวินาทีที่อุณหภูมิ 25 องศาเซลเซียสส่วนคลื่นวิทยุนั้นเดินทางด้วยความเร็วประมาณ $3*10^8$ เมตรต่อวินาที

การออกแบบอุปกรณ์ที่ใช้



ภาพที่ 14 อุปกรณ์ที่ใช้ในงานวิจัย

ภาพที่ 14 แสดงภาพของอุปกรณ์ที่ใช้ในงานวิจัยซึ่งได้ติดตั้งระบบปฏิบัติการไทนี่ซึ่งทำหน้าที่เป็นโหนดที่ติดตั้งบนเพดานและติดตั้งอยู่ที่วัตถุ



ภาพที่ 15 แผนผังของอุปกรณ์ที่ใช้ในงานวิจัย

ภาพที่ 15 แสดงแผนผังของอุปกรณ์ประกอบด้วยหน่วยประมวลผล (ซีพียู) แบบประหยัดพลังงานเบอร์ MSP430 [Texas Instrument., 2006] ของบริษัทเท็กซัสอินสตรูเม้นท์ จำกัด, ชิปรับส่งคลื่นวิทยุที่ความถี่ 433 MHz เบอร์ CC1000 [Texas Instrument., 2003] ของบริษัทเท็กซัสอินสตรูเม้นท์ จำกัด, ชิปควบคุมการรับส่งข้อมูลระหว่างอุปกรณ์กับเครื่องคอมพิวเตอร์ (PC) เบอร์

FT232BM [www.ftdichip.com] ของบริษัท Future Technology Devices จำกัดและส่วนรับส่งคลื่นอัลตราโซนิคควบคุมด้วยซีพียูเบอร์ PIC16F630 โดยใช้มอดูล SRF05 [www.robot-electronics.co.uk]

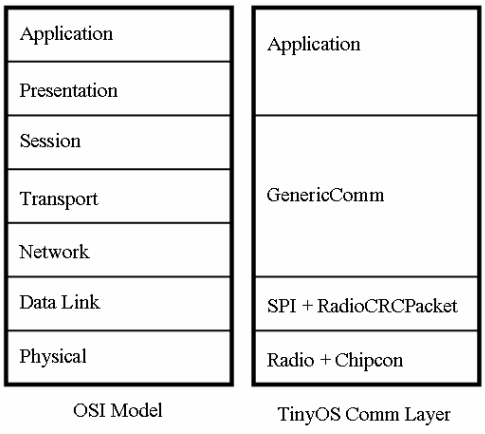
สิ่งที่สำคัญประการหนึ่งของระบบเครือข่ายตรวจจับแบบไร้สายคือพลังงานในงานวิจัยนี้ใช้อุปกรณ์ที่ประกอบด้วยซีพียูแบบประหยัดพลังงานเบอร์ MSP430F1611 ซึ่งใช้พลังงานน้อยกว่าเมื่อเทียบกับซีพียูตระกูลอื่นๆ

ตารางที่ 3 เปรียบเทียบพลังงานที่ซีพียูใช้ในแต่ละแพลตฟอร์ม

Platform	EnOcean	Crossbow	Telos	EYEIFX
MCU	PIC18F452	ATMEL128L	TIMSP430	TIMSP430
Current Active Mode (mA)	13.40	8.90	1.70	1.40
Current Sleep Mode (mA)	8.00	27.70	3.30	2.90
Supply Voltage (V)	4.75	2.70	1.80	2.10

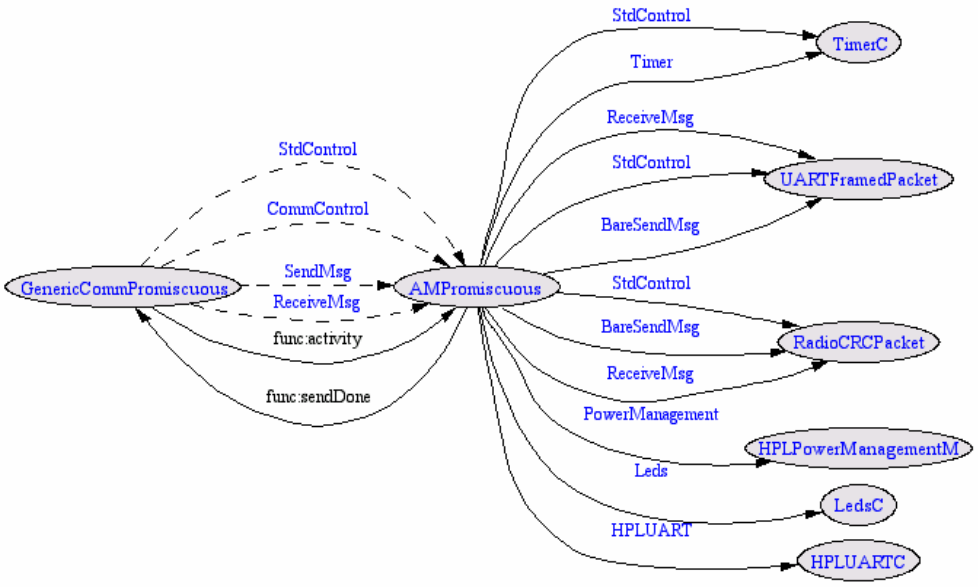
มอดูลการรับส่งข้อมูล

การรับส่งข้อมูลแบ่งออกเป็น 2 ส่วนคือการรับส่งข้อมูลผ่านคลื่นวิทยุและการรับส่งข้อมูลผ่านช่องต่ออนุกรม



ภาพที่ 16 เปรียบเทียบระบบปฏิบัติการไทนี่ Comm Layer กับ OSI Model

ภาพที่ 16 แสดงการเปรียบเทียบหน้าที่ของแต่ละมอดูลสำหรับการรับส่งข้อมูลกับ Open System Interconnection Model หรือ OSI Model



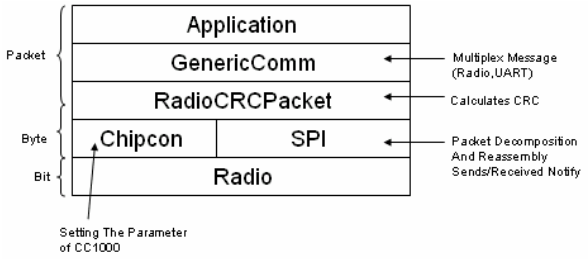
ภาพที่ 17 แสดงคอนฟิกูเรชั่นที่ทำหน้าที่รับส่งข้อมูล

จากภาพที่ 17 แสดงคอนฟิกูเรชั่นที่ทำหน้าที่รับส่งข้อมูลซึ่งประกอบด้วยมอดูลต่างๆได้แก่

1. GenericCommPromiscuous ทำหน้าที่ควบคุมการรับส่งข้อมูล
2. AMPromiscuous ทำหน้าที่พิจารณาว่าข้อมูลปีนข้อมูลสำหรับส่งไปที่ช่องต่ออนุกรมหรือส่งผ่านคลื่นวิทยุ

3.	TimerC	ทำหน้าที่ควบคุมเกี่ยวกับการตั้งเวลา
4.	UARTFramedPacket	ทำหน้าที่รับส่งข้อมูลผ่านช่องต่อแบบอนุกรม
5.	RadioCRCPacket	ทำหน้าที่รับส่งข้อมูลผ่านคลื่นวิทยุ
6.	HPLPowerManagementM	ทำหน้าที่ควบคุมพลังงาน
7.	LedsC	ทำหน้าที่แสดงผลผ่าน Led
8.	HPLUARTC	ทำหน้าที่กำหนดรูปแบบการรับส่งข้อมูลผ่านช่องต่อแบบอนุกรม

มอดูลการรับส่งคลื่นวิทยุ

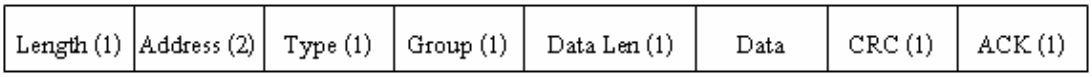


ภาพที่ 18 ลำดับชั้นของมอดูลรับส่งคลื่นวิทยุ

งานวิจัยนี้ได้ปรับปรุงมอดูลการควบคุมการรับส่งคลื่นวิทยุจากมาตรฐานเดิม [Jaein., 2004] เป็นดังภาพที่ 18 เพื่อใช้ในการทดลองให้ใช้งานร่วมกับอุปกรณ์สำหรับงานวิจัยนี้โดยแบ่งหน้าที่การทำงานของแต่มอดูลไว้ดังนี้

ตารางที่ 4 แสดงมอดูลที่เกี่ยวข้องกับการรับส่งข้อมูลผ่านคลื่นวิทยุ

ชื่อมอดูล	หน้าที่
Application	ทำหน้าที่ตรวจวัดระยะทาง,การสร้าง Spanning tree และรับส่งข้อมูลกับ PC
GenericComm	ทำหน้าที่ตรวจสอบ Field Type1 ว่าเป็น Packetสำหรับการติดต่อผ่านคลื่นวิทยุหรือเป็น Packet สำหรับการติดต่อผ่าน USB Port
RadioCRCPacket	ทำหน้าที่คำนวณ CRC หากการคำนวณไม่ผ่านก็จะทำการ Drop Packet
Chipcon	ทำหน้าที่กำหนดหรือเปลี่ยนแปลงค่าเริ่มต้นสำหรับการรับส่งคลื่นวิทยุในส่วนของการเริ่มต้นใช้งานและปรับเปลี่ยนค่าบางอย่างเมื่อกำลังทำงาน
SPI	ทำหน้าที่รวบรวมข้อมูลที่ได้แบบบิตจากส่วนรับส่งคลื่นวิทยุให้อยู่ในรูปแบบไบต์เพื่อส่งให้ส่วนอื่นทำการรวบรวมให้อยู่ในรูปแบบ Packet ต่อไป
Radio	เป็นส่วนเชื่อมต่อกับชิปรับส่งคลื่นวิทยุ



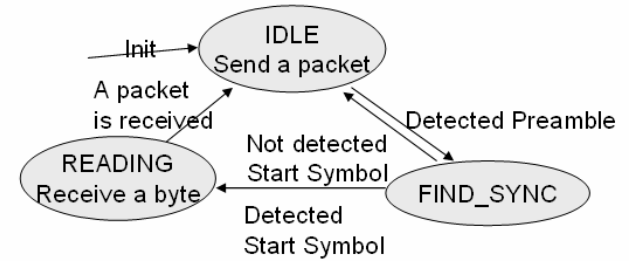
ภาพที่ 19 สตรีมข้อมูลของ Active Message (AM)

ข้อมูลจากมอดูล Application และ GenericComm จะใช้ AM (ภาพที่ 19) ในการติดต่อสื่อสารระหว่างกันโดยฟิลด์ Length คือความยาวของ ทั้งหมด AM, ฟิลด์Address คือที่อยู่ปลายทางของโหนดที่ต้องการส่งข้อมูลทั้งโหนดในเครือข่ายรวมถึงการส่งผ่านช่องต่ออนุกรมด้วย, ฟิลด์ Type คือฟิลด์ที่ระบุประเภทของข้อมูลว่าเป็นข้อมูลที่ส่งมาจากวัตถุ (Beacon) หรือการรับส่งข้อมูลปกติ, ฟิลด์ Group คือกลุ่มของข้อมูลใช้ในกรณีที่ในเครือข่ายนั้นมีหลายกลุ่ม(0x7D), ฟิลด์ Data Len เก็บขนาดของข้อมูลในฟิลด์ Data, ฟิลด์ CRC เก็บผลการตรวจสอบว่าข้อความที่รับมานั้นถูกต้องหรือไม่และฟิลด์ Ack ระบุว่าต้องการให้โหนดปลานทางนั้นมีการตอบกลับในกรณีที่รับข้อความนั้นได้หรือไม่

Preamble (18)	Sync (2)	Data

ภาพที่ 20 สตรีมข้อมูลของมอดูล Radio

ภาพที่ 20 แสดงสตรีมของข้อมูลที่รับส่งผ่านคลื่นวิทยุประกอบด้วยฟิลด์ Preamble ทำหน้าที่เพื่อเข้าจังหวะเพื่อให้ฝั่งรับสัญญาณเริ่มต้นหาฟิลด์ Sync ต่อไป, ฟิลด์ Sync เป็นฟิลด์ที่ระบุจุดเริ่มต้นของข้อมูล



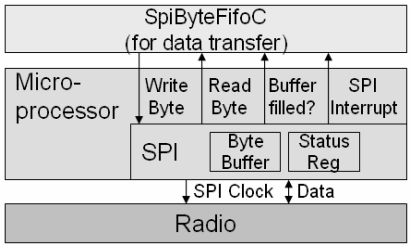
ภาพที่ 21 State diagram ของส่วนรับ-ส่งคลื่นวิทยุ

ภาพที่ 21 เริ่มจากซีพียูอยู่ในสถานะเริ่มต้นซีพียูจะถูกอินเตอรรัพท์จาก RF Module ทุกๆ 8 ลูกคลื่นสัญญาณนาฬิกาเนื่องจากซีพียูถูกกำหนดให้อยู่ใน Slave Mode หลังจาก RF Module ได้รับข้อมูลผ่านทาง SPI ครบ 8 บิตแล้วก็จะทำการอินเตอรรัพท์ซีพียูหลังจากนั้นซีพียูก็ทำการเปรียบเทียบข้อมูลที่รับว่าเป็นรูปแบบของ Preamble หรือไม่หากใช่ก็จะทำการหา Sync Byte ต่อไปหรือไม่ก็จะเข้าสู่สภาวะเริ่มต้นต่อไปก็จะทำการรับข้อมูลถัดมาเก็บไว้ในบัฟเฟอร์ภาครับพร้อมส่งสัญญาณให้เลเยอร์บนทราบว่าข้อมูลถัดจากนี้เป็นข้อมูลที่ต้องนำไปประมวลผลต่อไปและเข้าสู่สถานะ READING แต่หากไม่ตรงก็จะเข้าสู่สภาวะเริ่มต้นต่อไป ซึ่งในสถานะ READING นั้นก็จะทำการรวบรวมข้อมูลที่ได้รับมาใส่ลงในรูปแบบของ AM (ภาพที่19) และหลังจากได้รับข้อมูลจนครบแล้วก็จะส่งให้กับเลเยอร์บนทำการประมวลผลต่อไป

การเชื่อมต่อระหว่างซีพียูกับมอดูลรับส่งคลื่นวิทยุ

การเชื่อมต่อจะกระทำผ่านการเชื่อมต่อแบบอนุกรม (Serial Peripheral Interface: SPI) โดยซีพียูจะเขียนและอ่านข้อมูลผ่านทางบัฟเฟอร์ของซีพียูซึ่งต่อเชื่อมกับ RF Module ผ่านทางขา

PALE, PCLK และ PDATA ของซีพียู (MSP430F1611) ซึ่งซีพียูดังกล่าวนั้นมี SPI 2 ช่องต่อคือ SPI0 และ SPI1 โดย SPI0 นั้นต่อเชื่อมกับชิปรับส่งคลื่นวิทยุดังภาพที่ 22



ภาพที่ 22 แผนผังของการเชื่อมต่อระหว่างซีพียูกับชิปรับส่งคลื่นวิทยุ

งานวิจัยนี้ได้พัฒนาส่วนของการเชื่อมต่อระหว่างซีพียูกับชิปรับส่งคลื่นวิทยุคือไฟล์ HPLUST0M.nc ซึ่งมีรหัส (Source code) ตามภาคผนวกที่ 1

ไฟล์ CC1000RadioIntM.nc ทำหน้าที่รับข้อมูลจาก RF Module และมีตัวแปรเพื่อเก็บสถานะของการประมวลผลขนาด 1 ไบท์เพื่อให้ทราบว่าการรับส่งคลื่นวิทยุอยู่ในสถานะใดบ้าง โดยแบ่งออกเป็นดังนี้

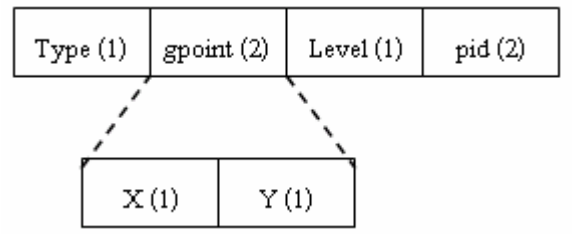
1. TX_STATE คือสถานะกำลังส่งข้อมูล
แบ่งออกเป็น
   - 1.1 TXSTATE_PREAMBLE คือสถานะส่ง Preamble
   - 1.2 TXSTATE_SYNC คือสถานะส่ง Sync byte
   - 1.3 TXSTATE_DATA คือสถานะส่ง Data
   - 1.4 TXSTATE_CRC คือสถานะส่ง CRC
   - 1.5 TXSTATE_FLUSH คือสถานะ Drop packet
   - 1.6 TXSTATE_WAIT_FOR_ACK คือสถานะรอ Acknowledge
   - 1.7 TXSTATE_READ_ACK คือสถานะตรวจสอบ Acknowledge
   - 1.8 TXSTATE_DONE คือสถานะส่งข้อมูลเรียบร้อยแล้ว
2. DISABLED_STATE คือสถานะถูกปิดการใช้งาน
3. IDLE_STATE คือสถานะรับข้อมูลเพื่อหา Sync byte

4. PRETX_STATE คือสถานะเตรียมส่งข้อมูล
5. SYNC_STATE คือสถานะพบ Sync byte แล้ว
6. RX_STATE คือสถานะกำลังรับข้อมูล
7. SENDING_ACK คือสถานะกำลังส่ง Acknowledge
8. POWER_DOWN_STATE คือสถานะโหมดประหยัดพลังงาน
9. NULL_STATE คือสถานะว่าง

ไฟล์ CC1000ControlM.nc ทำหน้าที่กำหนดค่าเริ่มต้นให้กับ RF Module เช่น ค่า BAUDRATE, ความถี่ใช้งาน, กำลังส่ง เป็นต้น ซึ่งจะถูกเรียกใช้เพียงครั้งเดียวหลังจากรีเซต

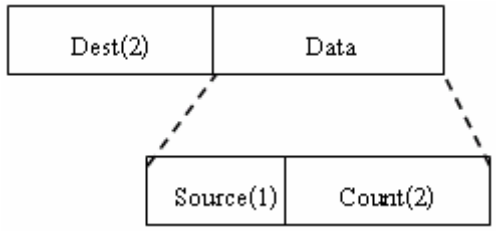
มอดูลควบคุมการสื่อสารแบบมัลติฮอป

มอดูลที่สำคัญอีกส่วนหนึ่งคือมอดูสำหรับการควบคุมการส่งข้อมูลผ่านเครือข่ายแบบมัลติฮอปซึ่งมีสตรีมข้อมูลสำหรับการหาเส้นทางดังภาพที่ 17



ภาพที่ 23 สตรีมข้อมูลสำหรับการค้นหาเส้นทาง

จากภาพที่ 23 ประกอบด้วยฟิลด์ Type ทำหน้าที่ระบุว่าเป็นสตรีมข้อมูลประเภทยืนยันการเชื่อมต่อ, ฟิลด์ gpoint เป็นข้อมูลตำแหน่งของโหนดตนเอง (x, y) , ฟิลด์ Level เป็นข้อมูลระดับตำแหน่ง (Level) ของโหนดตนเองเพื่อใช้ประกอบในการพิจารณาเลือกที่อยู่โหนดปลายทางของผู้ที่รับข้อมูลได้และฟิลด์ pid เป็นข้อมูลของที่อยู่โหนดปลายทาง



ภาพที่ 24 สตรีมข้อมูลสำหรับการส่งผลการวัดระยะทาง

จากภาพที่ 24 แสดงสตรีมข้อมูลของผลที่ได้จากการวัดระยะทางระหว่างวัตถุกับโหนดซึ่งประกอบด้วยฟิลด์ Dest เป็นข้อมูลของโหนดปลายทาง (x, y), ฟิลด์ Data ซึ่งประกอบ 2 ฟิลด์คือ ฟิลด์ Source เป็นข้อมูลของตำแหน่งของโหนดตนเอง (โหนดที่ทำการวัดระยะทาง) และฟิลด์ Count เป็นข้อมูลระยะทางระหว่างโหนดตนเองกับวัตถุ

มอดูลการวัดระยะทางระหว่างโหนดและวัตถุ

จากหลักการวัดความเร็วที่แตกต่างกันระหว่างคลื่นวิทยุกับคลื่นอัลตราโซนิคโดยการวัดระยะทางนั้นเริ่มจากการรับคลื่นวิทยุก่อนแล้วจึงรับสัญญาณอัลตราโซนิคซึ่งจะใช้ภาคแปลงสัญญาณอะนาล็อกเป็นดิจิตอลของซีพียูโดยทำการเปรียบเทียบสัญญาณอัลตราโซนิคกับค่าแรงดันคงที่ค่าหนึ่งแล้วซีพียูก็จะหยุดนับสัญญาณนาฬิกาแล้วนำค่าที่ได้ส่งให้กับ โหนดหลัก ต่อไปในส่วนของการส่งสัญญาณอัลตราโซนิคนั้นจะส่งออกจากวัตถุทุกๆ 10 วินาทีนานครั้งละ 1 มิลลิวินาทีที่ความถี่ 40 KHz พร้อมกับการส่งคลื่นวิทยุซึ่งในสตรีมข้อมูลนั้นประกอบด้วยข้อมูลรหัสวัตถุและฟิลด์ที่บอกว่าเป็นสตรีมข้อมูลประเภท Beacon ที่ส่งมาจากวัตถุโดยกำหนดที่อยู่ของโหนดปลายทางเป็นแบบกระจาย (0xFFFF) ซึ่งทำให้ทุก โหนด ที่รับ Packet และรับสัญญาณอัลตราโซนิคได้ก็จะทำการส่งข้อมูลไปที่โหนดหลักเพื่อคำนวณหาตำแหน่งของวัตถุต่อไป

ในงานวิจัยนี้ได้พัฒนามอดูลสำหรับการวัดระยะทางระหว่างโหนดกับวัตถุโดยใช้ชื่อว่า TestRoutingM.nc ซึ่งได้รวมเอาการทำงานในส่วนของการวัดระยะทางและส่วนของการส่งข้อมูลโดยมี Interface ที่ต้องการใช้ (uses interface) ดังนี้

1.	Leds	ทำหน้าที่ส่งข้อมูลเพื่อแสดงผลทาง Led
2.	Routing	ทำหน้าที่รับส่งข้อมูลที่วัดได้
3.	RoutingControl	ทำหน้าควบคุมการรับส่งข้อมูล
4.	ObjBeacon	ควบคุมเวลาการส่งสัญญาณ Beacon
5.	Timer1	ควบคุมเวลาการส่งค่าที่วัดระยะทางได้
6.	Timer2	ควบคุมการนับสัญญานาฬิกาสำหรับ
7.	UARTControl	ควบคุมการรับส่งข้อมูลผ่านทาง USB Port
8.	UARTSend	ทำหน้าที่ส่งข้อมูลผ่านทาง USB Port
9.	RadioReceiveCoord	ทำหน้าที่แจ้งเมื่อตรวจจับ Field SyncByte ได้
10.	RadioSendCoord	ทำหน้าที่แจ้งเมื่อส่งสัญญาณถึงฟิลด์ SyncByte
11.	ReceiveBeacon	แจ้งเมื่อรับข้อมูล Beacon ได้แล้ว
12.	SendBeacon	ควบคุมการส่งสัญญาณ Beacon
13.	GetRxBitOffset	เก็บค่าชดเชยเมื่อตรวจจับ Field SynByte ได้แล้ว

และมี Interface ที่รองรับ (provides interface) ดังนี้

1.	StdControl	ควบคุมการเริ่มต้นใช้งานของ Module นี้

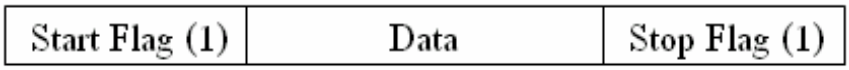
โดยมีรหัส (Source code) ตามภาคผนวกที่ 2

การตรวจจับสัญญาณอัลตราโซนิคนั้นได้ทดสอบ 2 วิธีได้แก่การใช้อินเทอร์รัพท์โดยตรวจจับจากระดับแรงดันที่ขาของซีพียูและการใช้อินเทอร์รัพท์จากภาคแปลงสัญญาณอนาล็อกเป็นดิจิตอล (Analog to Digital Converter: ADC) พบว่าการตรวจจับสัญญาณด้วยวิธีที่2 ได้ความแม่นยำมากกว่าวิธีแรกส่วนหนึ่งนั้นเนื่องมาจากซีพียูได้ถูกกำหนดลำดับความสำคัญ (Priority) ของการตอบสนองอินเทอร์รัพท์ให้กับส่วน ADC สูงกว่าส่วนอื่นๆ

การรับส่งคลื่นอัลตราโซนิคถูกควบคุมจากซีพียู PIC16F630 ซึ่งทำหน้าที่รับคำสั่งจากซีพียูหลักแล้วทำการสร้างคลื่นความถี่ขนาด 40 KHz มีระยะเวลา 1 มิลลิวินาทีและรับสัญญาณคลื่นอัลตราโซนิคเพื่อแปลงเป็นแรงดันส่งให้ซีพียูหลักต่อไป

มอดูลการเชื่อมต่อกับ PC

ในส่วนของการเชื่อมต่อระหว่างโหนดหลักกับ PC นั้น PC ต้องติดตั้งโปรแกรม Virtual COM port (VCP) driver ซึ่งทำหน้าที่แปลงรูปแบบข้อมูลระหว่าง Universal Serial Bus (USB) กับช่องต่อแบบอนุกรม (Serial Port)



ภาพที่ 25 รูปแบบของสตรีมข้อมูลระหว่าง PC กับโหนดหลัก

จากภาพที่ 25 Start Flag และ Stop Flag คือ 0x7E (01111110B) เมื่อ PC ได้รับข้อมูลจากโหนดหลักแล้วจะนำมาตรวจสอบตำแหน่งต่างๆของฟิลด์โดยดึงข้อมูลที่ใช้ในคำนวณหาตำแหน่งของวัตถุได้แก่หมายเลขของโหนดและค่าที่วัดได้โดยหาจุดเริ่มต้นและสิ้นสุดของสตรีมข้อมูลจาก Start Flagและ Stop Flag โดยฟิลด์ Data นั้นประกอบด้วยฟิลด์ของ AM

งานวิจัยนี้เลือกใช้วิธี TDOA (Time Difference of Arrival) คือการตรวจวัดเวลาในการเดินทางของคลื่นอัลตราโซนิคที่เดินทางมาถึง Sensor โหนด แต่ละตัวแล้วนำมาคำนวณด้วยสมการไฮเปอร์โบลิคจาก Sensor โหนด อย่างน้อย 3 ตัว ข้อดีของระบบ

1. ระบบจะทำการค้นหาตำแหน่งของวัตถุได้พื้นที่มากกว่า เนื่องจากระบบนี้เป็น Multi Hop
2. พลังงานที่ใช้ในระบบน้อยกว่าเนื่องจากเป็น Multi Hop
3. ทำการ Synchronize ระหว่าง โหนด กับวัตถุเฉพาะจุดที่วัตถุอยู่เท่านั้นไม่ต้องทำการ Synchronize ทุก โหนด จึงเกิดความผิดพลาดน้อยกว่าเมื่อระบบมีขนาดใหญ่

การคำนวณหาตำแหน่ง

การทดสอบความแม่นยำของผลการคำนวณตำแหน่งนั้นได้ทำการทดสอบโดยการกำหนดค่าที่จำเป็นต้องใช้ในการคำนวณหาตำแหน่งได้แก่ตำแหน่งของโหนดที่ติดตั้งบนเพดานและค่าที่แต่ละโหนดวัดได้ซึ่งได้มาจากการคำนวณโดยเริ่มจากกำหนดตำแหน่งของวัตถุไว้ที่ตำแหน่ง X=60 เซนติเมตรและ Y=60 เซนติเมตรและ Z=230 เซนติเมตรแล้วทำการคำนวณระยะทางระหว่างวัตถุกับโหนดทั้ง 3 โหนดจากสมการสามเหลี่ยมโดยให้ D_i คือระยะทางระหว่างวัตถุกับ Node ที่ i

$$D_i = \sqrt{(230\text{^}2) + (\sqrt{(60\text{^}2) + (60\text{^}2)} = 245.15$$

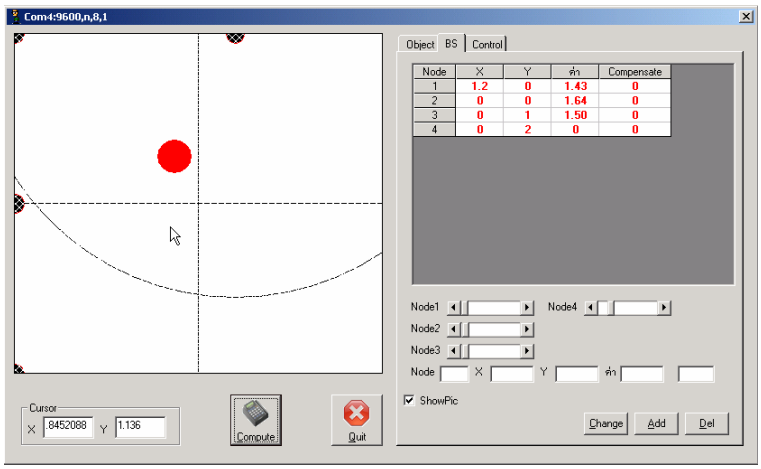
แล้วนำค่า D_i ที่ได้มาคำนวณด้วยสมการที่เขียนไว้จะได้คำตอบซึ่งเป็นตำแหน่งของวัตถุเท่ากับ X=60 เซนติเมตรและ Y=60 เซนติเมตร และทำการทดสอบโดยการเปลี่ยนตำแหน่งของวัตถุจะได้ผลลัพธ์ตามตารางด้านล่าง

ตารางที่ 5 แสดงผลลัพธ์ของการคำนวณตำแหน่ง

ครั้งที่	X (วัตถุ)	Y (วัตถุ)	X (ผลลัพธ์)	Y (ผลลัพธ์)
1	60.00	60.00	60.00	60.00
2	60.00	70.00	60.00	69.99
3	70.00	70.00	70.00	70.00
4	80.00	70.00	80.00	69.99
5	100.00	120.00	100.00	120.00
6	150.00	120.00	150.01	120.01

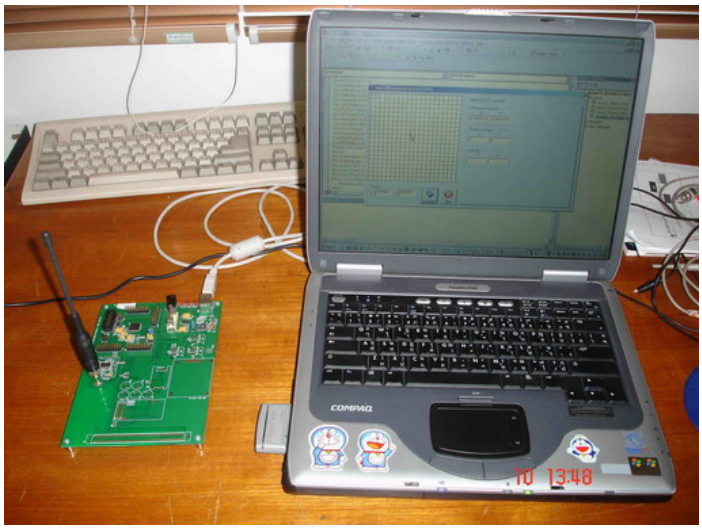
จากตารางที่ 5 จะเห็นได้ว่าสมการที่นำมาใช้มีความถูกต้องแม่นยำซึ่งขึ้นอยู่กับข้อมูลระยะทางระหว่างวัตถุกับโหนดที่ป้อนเข้าไป

งานวิจัยนี้ออกแบบระบบทดสอบโดยติดตั้งโหนดไว้ที่เพดานเป็นลักษณะแบบตาข่ายเพื่อรับสัญญาณอัลตราโซนิคจากวัตถุผลแล้วส่งข้อมูลไปที่โหนดหลักเพื่อคำนวณหาตำแหน่งของวัตถุร่วมกับข้อมูลตำแหน่งของโหนดที่รับสัญญาณได้



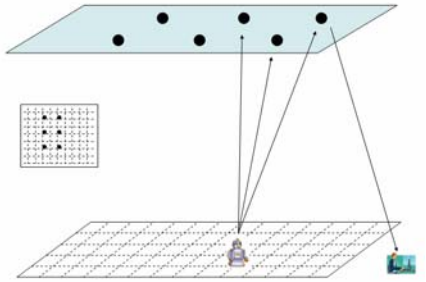
ภาพที่ 26 แสดงหน้าจอของระบบที่ทดลอง

จากภาพที่ 26 แสดงตัวอย่างหน้าจอที่ PC ที่ทำการคำนวณหาตำแหน่งของวัตถุโดยจะแสดงตำแหน่งของโหนดที่ติดตั้งบนเพดานและตำแหน่งของวัตถุที่คำนวณได้



ภาพที่ 27 แสดงโหนดหลักกับ PC

ภาพที่ 27 แสดงการเชื่อมต่อระหว่างโหนดหลักซึ่งทำหน้าที่รวบรวมข้อมูลจากโหนดต่างๆ และทำการส่งข้อมูลให้ PC ผ่านทางช่องต่อ USB เพื่อให้ PC คำนวณตำแหน่งของวัตถุต่อไป



ภาพที่ 28 หลักการทำงานของระบบ

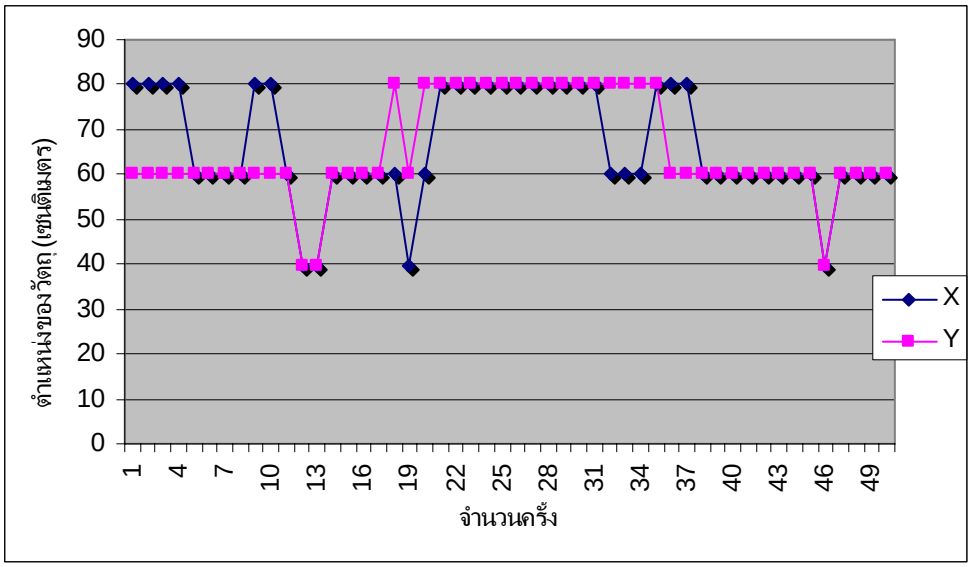
ภาพที่ 29 แสดงรูปแบบการติดตั้งโหนดจากห้องทดลอง

จากภาพที่ 29 คือภาพจากห้องทดลองแสดงการติดตั้งโหนดบนเพดานในลักษณะกริดและวัตถุซึ่งอยู่ด้านล่างจะส่งสัญญาณคลื่นวิทยุและคลื่นอัลตราโซนิคไปที่โหนดบนเพดาน

ผลและวิจารณ์

ผลจากการทดลอง

จากผลห้องทดลองซึ่งได้ทดสอบจำนวน 50 ครั้งโดยติดตั้งโหนดที่ตำแหน่ง (0, 0), (120, 0), (0, 120) และ (0, 240) เซนติเมตรโดยวางวัตถุไว้ที่ X = 60 เซนติเมตรและ Y = 60 เซนติเมตร และ Z = 210 เซนติเมตตามลำดับ



ภาพที่ 30 ผลการระบุตำแหน่งจากการทดลอง

จากภาพที่ 30 พบว่าค่าความผิดพลาดจะอยู่ในช่วงประมาณ ± 20 เซ็นติเมตร

ผลการจำลองการทำงาน

จากการจำลองการทำงานแสดงให้เห็นว่าการเพิ่มโหนดที่รับสัญญาณจากวัตถุจะเป็นการเพิ่มความแม่นยำของการระบุตำแหน่งของวัตถุโดยงานวิจัยนี้ทดสอบโดยการกำหนดตำแหน่งของโหนดไว้ 16 โหนดโดยวางไว้ที่ตำแหน่งต่างๆดังนี้

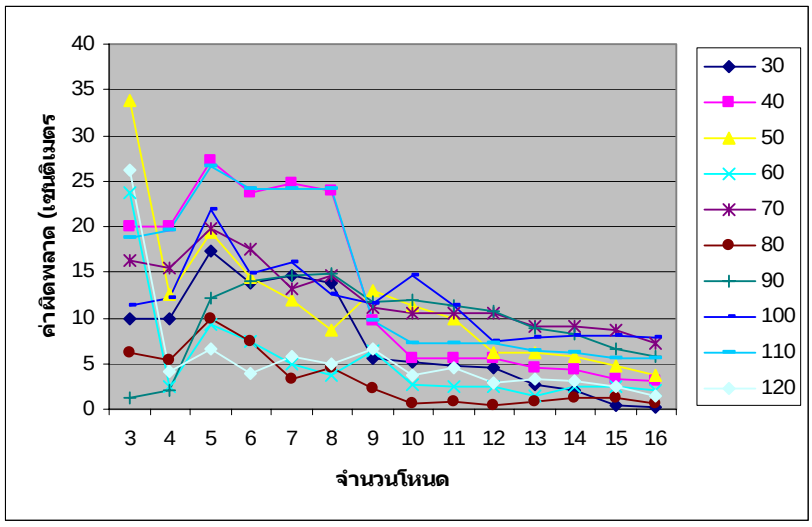
ตารางที่ 6 แสดงตำแหน่งของโหนด

โหนด ที่	X (cm.)	Y (cm.)
โหนด 1	0	0
โหนด 2	40	0
โหนด 3	0	40
โหนด 4	40	40
โหนด 5	0	80
โหนด 6	40	80
โหนด 7	0	120
โหนด 8	40	120
โหนด 9	80	0
โหนด 10	120	0
โหนด 11	80	40
โหนด 12	120	40
โหนด 13	80	80
โหนด 14	120	80
โหนด 15	80	120
โหนด 16	120	120

ตารางที่ 6 แสดงตำแหน่งของโหนดที่ติดตั้งเพื่อจำลองการทำงานและนำข้อมูลการวัดระยะทางของวัตถุกับโหนดจากผลการทดลองจริงโดยจำลองข้อมูลของการวัดระยะทางของแต่ละโหนดด้วยวิธี Least Square Method (LSM) ได้สมการดังนี้

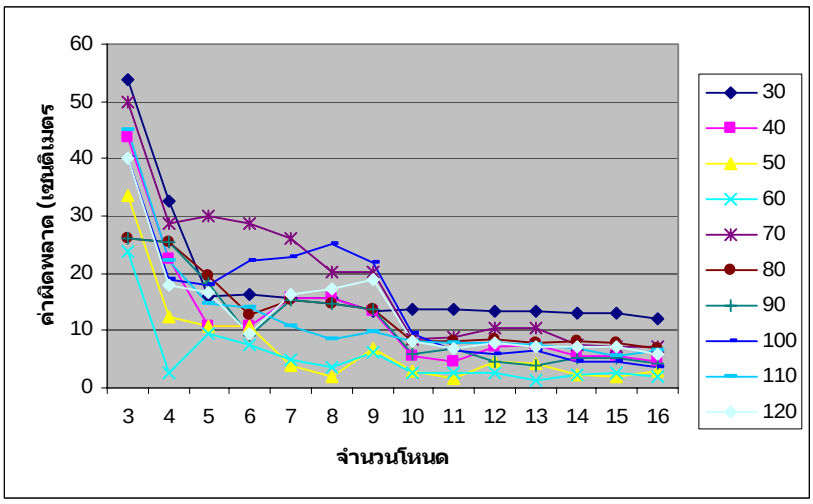
$$V = 2.67+9.55X$$

โดยที่ V คือค่าที่วัดได้และ X คือระยะห่างระหว่างโหนดกับวัตถุแล้วนำมาคำนวณเพื่อหาตำแหน่งของวัตถุและได้ผลตามรูปด้านล่าง



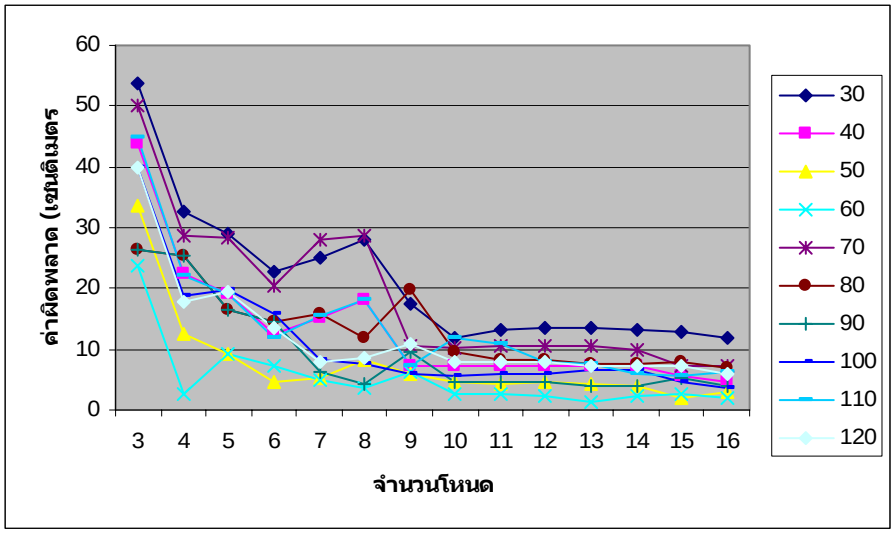
ภาพที่ 31 ผลการระบุตำแหน่งของวัตถุที่ตำแหน่งต่างๆ

จากภาพที่ 31 แสดงค่าความผิดพลาดของการระบุตำแหน่งที่จำนวนโหนดต่างๆและตำแหน่งของวัตถุต่างๆกันได้แก่ X, Y= 30 เซนติเมตร ถึง 120 เซนติเมตรที่ความสูง (Z) = 230 เซนติเมตรแกน X แสดงจำนวน โหนด ที่ใช้และแกน Y แสดงค่าความผิดพลาดของการระบุตำแหน่ง



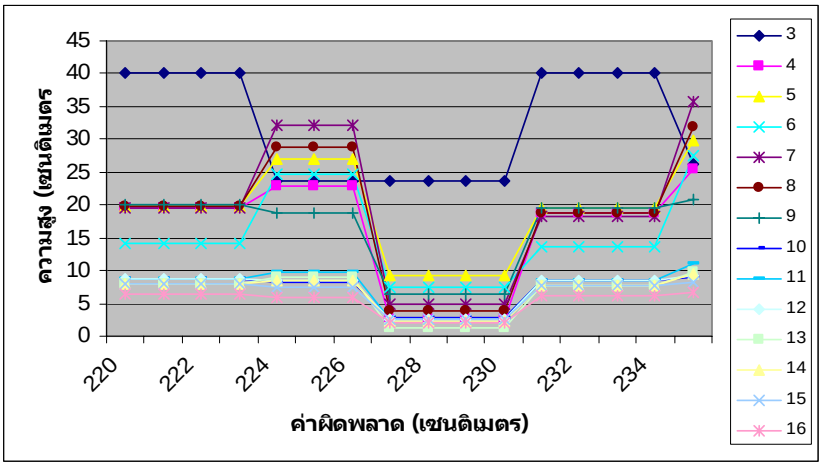
ภาพที่ 32 ผลการระบุตำแหน่งเมื่อคงตำแหน่งของแกน X

จากภาพที่ 32 แสดงค่าความผิดพลาดของการระบุตำแหน่งที่จำนวนโหนดต่างๆและตำแหน่งของวัตถุต่างๆกันโดยคงตำแหน่งแกน X ของวัตถุไว้ที่ 60 เซนติเมตรและเปลี่ยนแปลงตำแหน่งตามแนวแกน Y ตั้งแต่ 30 เซนติเมตรถึง 130 เซนติเมตรที่ความสูง (Z) = 230 เซนติเมตร แกน X แสดงจำนวน โหนด ที่ใช้และแกน Y แสดงค่าความผิดพลาดของการระบุตำแหน่ง



ภาพที่ 33 ผลการระบุตำแหน่งเมื่อคงตำแหน่งของแกน Y

จากภาพที่ 33 แสดงค่าความผิดพลาดของการระบุตำแหน่งที่จำนวนโหนดต่างๆและตำแหน่งของวัตถุต่างๆกันโดยคงตำแหน่งแกน Y ของวัตถุไว้ที่ 60 เซนติเมตรและเปลี่ยนแปลงตำแหน่งตามแนวแกน Y ตั้งแต่ 30 เซนติเมตรถึง 130 เซนติเมตรที่ความสูง (Z) = 230 เซนติเมตร แกน X แสดงจำนวน โหนด ที่ใช้และแกน Y แสดงค่าความผิดพลาดของการระบุตำแหน่ง



ภาพที่ 34 ผลการระบุตำแหน่งเมื่อเปลี่ยนแปลงแกน Z

จากภาพที่ 34 แสดงค่าความผิดพลาดของการระบุตำแหน่งที่ความสูง (แกน Z) ที่ระยะต่างๆ โดยคงตำแหน่งของวัตถุไว้ที่ X, Y=60 เซนติเมตรและเปลี่ยนแปลงตำแหน่งตามแนวแกน Z ตั้งแต่ 220 เซนติเมตร ถึง 235 เซนติเมตร แกน X แสดงจำนวนระยะความสูงและแกน Y แสดงค่าความผิดพลาดของการระบุตำแหน่ง

งานวิจัยนี้ยังได้ทำการทดสอบค่าที่ได้จากการวัดระยะทางเมื่อมีการส่ง Connected Packet (ข้อความเพื่อเชื่อมต่อ) โดยเปลี่ยนช่วงเวลาในการส่งข้อความตั้งแต่ 1 ถึง 5 วินาที พบว่าค่าที่วัดได้ไม่มีความเปลี่ยนแปลงมากนักจึงสรุปได้ว่าการทำงานในส่วนของการหาที่อยู่ของโหนดปลายทางนั้นไม่มีผลกระทบต่อการวัดระยะทางตามตารางที่ 7

ตารางที่ 7 ค่าที่โหนดวัดได้

ระยะทาง (ซ.ม.)	ครั้งที่ 1	ครั้งที่ 2	ครั้งที่ 3	ครั้งที่ 4	ครั้งที่ 5
105	12.56	12.60	12.63	12.72	12.71
110	13.23	13.12	13.17	13.10	13.08
115	13.56	13.57	13.19	13.24	13.21
120	14.34	14.30	14.33	14.33	14.33
125	14.83	14.78	14.79	14.81	14.79
130	15.28	15.35	15.24	15.28	15.23
180	20.47	20.55	20.54	20.46	20.54
185	20.94	21.01	21.25	21.18	21.05
190	21.44	21.45	21.47	21.46	21.43

สรุปและข้อเสนอแนะ

สรุป

จากผลการทดลองสรุปได้ว่าปัจจัยที่มีผลต่อความแม่นยำนั้นขึ้นอยู่กับหลายปัจจัยเช่นเวลาในการตอบสนองอินเทอร์รัพท์เมื่อรับสัญญาณอัลตราโซนิคได้ซึ่งจากการทดลองพบว่าการใช้อินเทอร์รัพท์จากภาคแปลงสัญญาณอนาล็อกเป็นดิจิตอล (Analog to Digital Converter: ADC) มีการตอบสนองดีกว่าการใช้อินเทอร์รัพท์แบบอื่นส่วนหนึ่งนั้นเนื่องมาจากซีพียูได้กำหนดความสำคัญของการตอบสนองให้กับภาค ADC สูงกว่าและอีกปัจจัยหนึ่งคือข้อมูลที่วัดระยะทางของแต่ละโหนดนั้นอาจมีความผิดพลาดซึ่งเกิดจากการส่งผ่านคลื่นวิทยุรวมถึงการได้จึงทำให้การคำนวณตำแหน่งของวัตถุนั้นผิดพลาดด้วยส่วนความถี่ของการทำ Spanning tree นั้นไม่มีผลกับความแม่นยำของระบบและความแม่นยำจากการจำลองนั้นจะมีความแม่นยำสูงที่สุดอยู่ที่ความสูง 227 เซนติเมตรถึง 230 เซนติเมตร (จากภาพที่ 34) และการเพิ่มโหนดจะเป็นการเพิ่มความแม่นยำของระบบได้มากขึ้นจากรูป 31 ถึงภาพที่ 34

ตารางที่ 8 แสดงค่าที่ระยะทางต่างๆ

ระยะทาง(ซม.)	ค่าที่วัดได้
105	13
110	13
115	14
120	14
125	15

จากตารางที่ 8 แสดงผลที่ได้จากการทดลองเมื่อวางโหนดและวัตถุห่างกันที่ระยะต่างๆซึ่งจะเห็นได้ว่าความแม่นยำนั้นจะอยู่ที่ประมาณ 10 เซนติเมตร โดยสังเกตได้จากค่าที่วัดได้จะเปลี่ยนแปลงทุกๆระยะ 10 เซนติเมตร

ข้อเสนอแนะ

งานวิจัยนี้สามารถนำไปพัฒนาเพื่อปรับปรุงความแม่นยำได้โดย

ติดตั้งอุปกรณ์วัดอุณหภูมิหรือวัดความชื้นเพื่อชดเชยการวัดระยะทางซึ่งมีผลกับความเร็วในการเดินทางของคลื่นอัลตราโซนิคซึ่งค่าความเปลี่ยนแปลงของอุณหภูมิ 1 องศาเซลเซียสนั้นจะทำให้ความเร็วในการเดินทางของคลื่นอัลตราโซนิดนั้นเปลี่ยนแปลงไป 0.17% หรือ 3.4 มิลลิเมตร ที่ระยะทาง 2 เมตร

การเพิ่มความเร็วของสัญญาณนาฬิกาที่โหนดนั้นทำให้สามารถส่งข้อมูลด้วยอัตราความเร็วที่สูงมากขึ้นทำให้ค่าที่วัดระยะทางนั้นละเอียดมากขึ้น

เพิ่มหน่วยความจำของซีพียูซึ่งจะทำให้สามารถพัฒนาโปรแกรมในส่วนของบัฟเฟอร์สำหรับการรับและส่งข้อมูลผ่านคลื่นวิทยุเนื่องจากหากอัตราความเร็วในการส่งน้อยนั้นทำให้ต้องใช้เวลาในการส่งสัญญาณนานจึงทำให้ช่องสัญญาณไม่ว่างโหนดจึงต้องรอ (Back off) ซึ่งหากโหนดนั้นจำเป็นต้องส่งข้อมูลอีกแต่ไม่มีบัฟเฟอร์รองรับก็จะทำให้ข้อมูลที่ต้องการส่งก่อนหน้านี้สูญหายได้

จากการวิเคราะห์พบว่าการเพิ่มความเร็วของสัญญาณนาฬิกาที่โหนดนั้นจะช่วยเพิ่มความแม่นยำของระบบได้ซึ่งการเพิ่มสัญญาณนาฬิกาซีพียูสามารถเลือกใช้สัญญาณนาฬิกาได้จาก 3 แหล่งได้แก่

1. Crystal หรือ Resonator
2. แหล่งกำเนิดความถี่จากภายนอก
3. Digital Control Oscillator (DCO) โดยกำหนดจากตัวต้านทานและตัวเก็บประจุ

ซึ่งการเพิ่มความเร็วของสัญญาณนาฬิกานั้นจำเป็นต้องมีการแก้ไขบางโมดูลที่จำเป็นได้แก่

1. โมดูล Timer ซึ่งเป็นโมดูลพื้นฐานทำหน้าที่ควบคุมฐานเวลาและใช้สำหรับโมดูลอื่นเพื่อประกอบการทำงาน

2. โมดูล Analog to Digital Converter (ADC) ทำหน้าที่แปลงสัญญาณอนาล็อกเป็นสัญญาณดิจิตอลซึ่งจะใช้โมดูล Timer เป็นฐานเวลาในการแปลงสัญญาณและสุ่มสัญญาณ

3. โมดูล Serial Program Interface (SPI) ทำหน้าที่เชื่อมต่อข้อมูลกับอุปกรณ์ภายนอกซึ่งจะใช้โมดูล Timer เป็นฐานเวลาในการกำหนดอัตราความเร็วในการรับและส่งข้อมูล

อุปสรรค

รูปแบบของสัญญาณที่ส่งออกมาจากอุปกรณ์ส่งสัญญาณอัลตราโซนิคยังไม่ครอบคลุมพื้นที่โหนดทั้งหมด (อย่างน้อย 3 โหนด) ทำให้ต้องติดตั้งโหนดให้ชิดกันมากขึ้นและการระบุตำแหน่งของโหนดที่มีแม่นยำสูงนั้นทำได้ยากซึ่งต้องระบุโดยใช้จุดส่งสัญญาณอัลตราโซนิคเป็นจุดวัด

ด้านการดีบัก (Debug) โปรแกรมนั้นทำได้ยากเนื่องจะสามารถดูสถานะการทำงานได้เพียงจากแอลอีดีหรือผ่านทางช่องต่อแบบอนุกรมซึ่งหากต้องการทดสอบการทำงานในส่วนของการเชื่อมต่อผ่านช่องต่อแบบอนุกรมเองนั้นจะทำได้ยากหรือต้องทำการดีบักผ่าน Led เท่านั้นซึ่งการทดสอบบนแพลทฟอร์มพีซีหรือ Tossim นั้นทำไม่ได้ในบางส่วนเช่นการทดสอบรับส่งข้อมูลผ่านคลื่นวิทยุหรือช่องต่ออนุกรมเป็นต้นซึ่ง Tossim นั้นจะทำได้เพียงดูข้อมูลว่าการรับส่งนั้นถูกต้องหรือไม่เท่านั้น

เอกสารและสิ่งอ้างอิง

Aleksandar Milenkovic, Chris Otto, Emil Jovanov. 2006. **Wireless sensor networks for personal health monitoring: Issues and an implementation.** Available online at http://www.sciencedirect.com

Byungrak Son, Yong-sork Her, and Jung-Gyu Kim. A Design and Implementation of Forest-Fires Surveillance System based on Wireless Sensor Networks for South Korea Mountains. **IJCSNS International Journal of Computer Science and Network Security, VOL.6 No.9B, September 2006.**

Philip Levis. 2006. TinyOS Programming.

Roy Want, Andy Hopper, Veronica Falcão, Jonathan Gibbons. 2003. The Active Badge Location System. **Olivetti Research Ltd. (ORL) Cambridge, England.**

Nissanka B. Priyantha, Anit Chakraborty, Hari Balakrishnan. 2000. The Cricket Location-Support System. **6th ACM International Conference on Mobile Computing and Networking (ACM MOBICOM), Boston, MA, August 2000.**

Texas Instruments Incorporated. 2006. **MSP430x1xx Family.** Available online at http://www.ti.com/msp430

Texas Instruments Incorporated. 2006. **Single Chip Very Low Power RF Transceiver.** Available online at http://www.ti.com/msp430

Future Technology Devices Intl. Ltd. 2002. **FT232BM USB UART.** Available online at http://www.ftdichip.com

Y.T. Chan. 1994. Simple and Efficient Estimator for Hyperbolic Location. **IEEE TRANSACTION ON SIGNAL PROCESSING, VOL.42, NO. 8, AUGUST 1994.**

David Gay, Philip Levis, Robert von Behren. **The nesC Language A Holistic Approach to Networked Embedded Systems.**

Mohamed G.Gouda, Yong-ri Choi, Anis Arora, Vinayak Naik. **The Logical Grid Routing Protocol in Sensor Network.**

Jaein Jeong, Sukun Kim. **CS262A Advanced Topics in Computer Systems DOT3 Radio Stack.** Available online at http:// cs.berkeley.edu

Hans-Jorg Korber, Housam Wattar. **Embedding a Microchip PIC18F452 based commercial platform into TinyOS**

ภาคผนวก

ภาคผนวก ก โมดูล HPLUST0M.nc

```
module HPLUST0M {
  provides interface HPLUSARTControl as USARTControl;
  provides interface HPLUSARTFeedback as USARTData;
}
implementation
{
  MSP430REG_NORACE(ME1);
  MSP430REG_NORACE(IFG1);
  MSP430REG_NORACE(U0TCTL);
  uint16_t l_br;
  uint8_t l_mctl;
  uint8_t l_ssel;
  TOSH_SIGNAL(UST0RX_VECTOR) {
    uint8_t temp = U0RXBUF;
    signal USARTData.rxDone(temp);
  }
  TOSH_SIGNAL(UST0TX_VECTOR) {
    signal USARTData.txDone();
  }
  async command void USARTControl.enableUART() {
     ME1 |= (UTXE0 | URXE0);   // USART0 UART module enable
  }
  async command void USARTControl.disableUART() {
     ME1 &= ~(UTXE0 | URXE0);   // USART0 UART module enable
     TOSH_SEL_UTXD0_IOFUNC();
     TOSH_SEL_URXD0_IOFUNC();
```

```
}
async command void USARTControl.enableUARTTx() {
   ME1 |= UTXE0;   // USART0 UART Tx module enable
}
async command void USARTControl.disableUARTTx() {
   ME1 &= ~UTXE0;   // USART0 UART Tx module enable
   TOSH_SEL_UTXD0_IOFUNC();
}
async command void USARTControl.enableUARTRx() {
   ME1 |= URXE0;   // USART0 UART Rx module enable
}
async command void USARTControl.disableUARTRx() {
   ME1 &= ~URXE0;  // USART0 UART Rx module disable
   TOSH_SEL_URXD0_IOFUNC();
}
async command void USARTControl.enableSPI() {
   ME1 |= USPIE0;   // USART0 SPI module enable
}
async command void USARTControl.disableSPI() {
   ME1 &= ~USPIE0;   // USART0 SPI module disable
   TOSH_SEL_SIMO0_IOFUNC();
   TOSH_SEL_SOMI0_IOFUNC();
   TOSH_SEL_UCLK0_IOFUNC();
}
async command void USARTControl.setModeSPI() {
  atomic {
   TOSH_SEL_SIMO0_MODFUNC();
   TOSH_SEL_SOMI0_MODFUNC();
```

```
TOSH_SEL_UCLK0_MODFUNC();
IE1 &= ~(UTXIE0 | URXIE0);  // interrupt disable
U0CTL |= SWRST;
U0CTL |= CHAR | SYNC | MM;  // 8-bit char, SPI-mode, USART as master
U0CTL &= ~(0x20);
U0TCTL = STC ;     // 3-pin
U0TCTL |= CKPH;    // half-cycle delayed UCLK
if (l_ssel & 0x80) {
  U0TCTL &= ~(SSEL_0 | SSEL_1 | SSEL_2 | SSEL_3);
  U0TCTL |= (l_ssel & 0x7F);
}
else {
  U0TCTL &= ~(SSEL_0 | SSEL_1 | SSEL_2 | SSEL_3);
  U0TCTL |= SSEL_SMCLK; // use SMCLK, assuming 1MHz
}
if (l_br != 0) {
  U0BR0 = l_br & 0x0FF;
  U0BR1 = (l_br >> 8) & 0x0FF;
}
else {
  U0BR0 = 0x02;   // as fast as possible
  U0BR1 = 0x00;
}
U0MCTL = 0;
ME1 &= ~(UTXE0 | URXE0); //USART UART module disable
ME1 |= USPIE0;   // USART SPI module enable
UCTL0 &= ~SWRST;
IFG1 &= ~(UTXIFG0 | URXIFG0);
```

```
    IE1 &= ~(UTXIE0 | URXIE0);  // interrupt disabled
  }
  return;
}
void setUARTModeCommon() {
  atomic {
    UCTL0 = SWRST;
    UCTL0 |= CHAR;  // 8-bit char, UART-mode
    U0RCTL &= ~URXEIE;  // even erroneous characters trigger interrupts
    UCTL0 = SWRST;
    UCTL0 |= CHAR;  // 8-bit char, UART-mode
    if (l_ssel & 0x80) {
      U0TCTL &= ~(SSEL_0 | SSEL_1 | SSEL_2 | SSEL_3);
      U0TCTL |= (l_ssel & 0x7F);
    }
    else {
      U0TCTL &= ~(SSEL_0 | SSEL_1 | SSEL_2 | SSEL_3);
      U0TCTL |= SSEL_ACLK; // use ACLK, assuming 32khz
    }
    if ((l_mctl != 0) || (l_br != 0)) {
      U0BR0 = l_br & 0x0FF;
      U0BR1 = (l_br >> 8) & 0x0FF;
      U0MCTL = l_mctl;
    }
    else {
      U0BR0 = 0x03;   // 9600 baud
      U0BR1 = 0x00;
      U0MCTL = 0x4A;
```

```
    }
    ME1 &= ~USPIE0;   // USART0 SPI module disable
    ME1 |= (UTXE0 | URXE0); //USART0 UART module enable;
    U0CTL &= ~SWRST;
    IFG1 &= ~(UTXIFG0 | URXIFG0);
    IE1 &= ~(UTXIE0 | URXIE0);  // interrupt disabled
  }
  return;
}
async command void USARTControl.setModeUART_TX() {
  atomic {
    TOSH_SEL_UTXD0_MODFUNC();
    TOSH_SEL_URXD0_IOFUNC();
  }
  setUARTModeCommon();
  return;
}
async command void USARTControl.setModeUART_RX() {
  atomic {
    TOSH_SEL_UTXD0_IOFUNC();
    TOSH_SEL_URXD0_MODFUNC();
  }
  setUARTModeCommon();
  return;
}
async command void USARTControl.setModeUART() {
  atomic {
    TOSH_SEL_UTXD0_MODFUNC();
```

```
    TOSH_SEL_URXD0_MODFUNC();
    setUARTModeCommon();
  }
  return;
}
async command void USARTControl.setClockSource(uint8_t source) {
    atomic {
      l_ssel = source | 0x80;
      U0TCTL &= ~(SSEL_0 | SSEL_1 | SSEL_2 | SSEL_3);
      U0TCTL |= (l_ssel & 0x7F);
    }
}
async command void USARTControl.setClockRate(uint16_t baudrate, uint8_t mctl) {
  atomic {
    l_br = baudrate;
    l_mctl = mctl;
    U0BR0 = baudrate & 0x0FF;
    U0BR1 = (baudrate >> 8) & 0x0FF;
    U0MCTL = mctl;
  }
}
async command result_t USARTControl.isTxIntrPending(){
  if (IFG1 & UTXIFG0){
    IFG1 &= ~UTXIFG0;
    return SUCCESS;
  }
  return FAIL;
}
```

```
async command result_t USARTControl.isTxEmpty(){
  if (U0TCTL & TXEPT) {
    return SUCCESS;
  }
  return FAIL;
}
async command result_t USARTControl.isRxIntrPending(){
  if (IFG1 & URXIFG0){
    IFG1 &= ~URXIFG0;
    return SUCCESS;
  }
  return FAIL;
}
async command result_t USARTControl.disableRxIntr(){
  atomic IE1 &= ~URXIE0;
  return SUCCESS;
}
async command result_t USARTControl.disableTxIntr(){
  atomic IE1 &= ~UTXIE0;
  return SUCCESS;
}
async command result_t USARTControl.enableRxIntr(){
  atomic {
    IFG1 &= ~URXIFG0;
    IE1 |= URXIE0;
  }
  return SUCCESS;
}
```

```
  async command result_t USARTControl.enableTxIntr(){
    atomic {
      IFG1 &= ~UTXIFG0;
      IE1 |= UTXIE0;
    }
    return SUCCESS;
  }
  async command result_t USARTControl.tx(uint8_t data){
    atomic U0TXBUF = data;
    return SUCCESS;
  }
  async command uint8_t USARTControl.rx(){
    uint8_t value;
    atomic value = U0RXBUF;
    return value;
  }
  default async event result_t USARTData.txDone() { return SUCCESS; }
  default async event result_t USARTData.rxDone(uint8_t data) { return SUCCESS; }
}
```

ภาคผนวก ข โมดูล TestRoutingM.nc

```
#ifndef DeadThreshold
#define DeadThreshold 3
#endif
includes GridTreeMsg;
module TestRoutingM
{
 provides{
        interface StdControl;
 }
 uses{
        interface Leds;
        interface Routing;
        interface StdControl as RoutingControl;
        interface Timer;
        interface Timer as ObjBeacon;
        interface Timer as Timer1;                      //detec timeout
        interface TimerJiffy as Timer2;                 //counter
        interface StdControl as UARTControl;
        interface SendMsg as UARTSend;


        interface RadioCoordinator as RadioReceiveCoord;
        interface RadioCoordinator as RadioSendCoord;

        interface ReceiveMsg as ReceiveBeacon;
    interface SendMsg as SendBeacon;
    async command uint8_t GetRxBitOffset();
```

```
  }
}

implementation
{
  MSP430REG_NORACE(CACTL1);
  MSP430REG_NORACE(CACTL2);
  TOS_Msg outMsgBuf;
  int8_t pending;
  uint16_t tick;
  uint16_t count;
  uint16_t val;
  uint8_t compensation = 0;
  bool UsRecved;
  uint16_t pendingDeadCount;
  command result_t StdControl.init(){
    pending = FALSE;
            tick = 0;
            count = 0;
            pendingDeadCount = 0;
    call RoutingControl.init();
    call UARTControl.init();
    return SUCCESS;
  }
  command result_t StdControl.start(){
            result_t ok1, ok3;
            ok1 = call RoutingControl.start();
            // if i am object(TOS_LOCAL_ADDR > 90) then start Beacon
```

```
        if ((TOS_LOCAL_ADDRESS != 0) && (TOS_LOCAL_ADDRESS > 90)){          //if
is object
                call ObjBeacon.start(TIMER_REPEAT,BEACON_INTERVAL);
        ok3 = call UARTControl.start();
        P2SEL |= 0x08;                  // Use as function
        CACTL2 = P2CA0;                         // Use as Comparator
        UsRecved = FALSE;               //TRUE FALSE;
        atomic   val = 0;
        return rcombine(ok1,ok3);
 }
 command result_t StdControl.stop(){
        result_t ok1, ok2, ok3;
        ok1 = call RoutingControl.stop();
        ok2 = call Timer.stop();
        ok3 = call UARTControl.stop();
        call ObjBeacon.stop();
        call Timer1.stop();
        call Timer2.stop();
        return rcombine3(ok1,ok2,ok3);
 }
 event result_t Timer1.fired()              //time out 100 ms
 {
  uint16_t val1;
  uint8_t          comp;
  AppMsg *msg = (AppMsg *)(outMsgBuf.data+HEADER_LEN);
  if (UsRecved) {
        atomic comp = compensation;
        val1 = val + comp;
```

```
        msg->src = TOS_LOCAL_ADDRESS;
        msg->count =  val1;// + comp;
                if(call Routing.send(&outMsgBuf)){
        }
  }
  atomic val = 0;
  call Timer2.stop();                                   //30.5us
  return SUCCESS;
}
event result_t ObjBeacon.fired(){
        struct Beacon *msg = (struct Beacon *)outMsgBuf.data;
        msg->src = TOS_LOCAL_ADDRESS;
        if (call SendBeacon.send(TOS_BCAST_ADDR, sizeof(struct Beacon),&outMsgBuf)){
                call Leds.redToggle();
        }
        return SUCCESS;
}
event result_t Timer.fired(){
  AppMsg *msg = (AppMsg *)(outMsgBuf.data+HEADER_LEN);
        tick++;
        if(TOS_LOCAL_ADDRESS != 0){
                // generate random messages to send
                msg->src = TOS_LOCAL_ADDRESS;
                msg->count = count;
                if(!pending){
                        pending = TRUE;
                        if(call Routing.send(&outMsgBuf)){
                                call Leds.redToggle();
```

```
                        }
                        else pending = FALSE;
                }
                else { //Hongwei:
                        pendingDeadCount++;
                        if (pendingDeadCount > DeadThreshold)
                                pending = FALSE;
                }
        }
        return SUCCESS;
}
event result_t Routing.sendDone(TOS_MsgPtr pmsg, result_t success)
{
        if(success == SUCCESS){
                count = count + 1;//++;
        }
                pending = FALSE;
        return success;
}
event TOS_MsgPtr Routing.receive(TOS_MsgPtr pmsg)
{
    AppMsg *msg = (AppMsg *)(pmsg->data+HEADER_LEN);
        struct ReportedMsg *cmsg = (struct ReportedMsg *)(outMsgBuf.data);
        if(TOS_LOCAL_ADDRESS != 0){
        }
        else{
                cmsg->src = msg->src;
                cmsg->count = msg->count;
```

```
                cmsg->type = msg->type;
                outMsgBuf.addr = TOS_UART_ADDR;
                if(!pending){
                        pending = TRUE;
                        if(call UARTSend.send(TOS_UART_ADDR, sizeof(struct
ReportedMsg),&outMsgBuf))
{
                        }
                        else
{
                                pending = FALSE;
                        }
                }
        else {
                        pendingDeadCount++;
                        if (pendingDeadCount > DeadThreshold)
                                pending = FALSE;
                }
        }
        return pmsg;
 }
 event result_t UARTSend.sendDone(TOS_MsgPtr pmsg, result_t success)
 {
                pending = FALSE;
        return success;
 }
 event result_t SendBeacon.sendDone(TOS_MsgPtr msg, result_t success) {
        return SUCCESS;
```

```
}

event TOS_MsgPtr ReceiveBeacon.receive(TOS_MsgPtr pmsg){
        return pmsg;

}
async event void RadioSendCoord.startSymbol(uint8_t bitsPerBlock, uint8_t offset,
TOS_MsgPtr msgBuff) {}
async event void RadioSendCoord.blockTimer() {}
async event void RadioSendCoord.byte(TOS_MsgPtr msg, uint8_t cnt)
{
    if (cnt == (offsetof(struct TOS_Msg,data) + 1) &&
        (TOS_LOCAL_ADDRESS !=0) &&
        (TOS_LOCAL_ADDRESS >90)&&
      (msg->type == AM_BEACON)){
                atomic {
                        TOSH_SET_USOUT_PIN();
                        TOSH_uwait(20);
                        TOSH_CLR_USOUT_PIN();
                }
            }
}
event result_t  Timer2.fired()
{
 atomic val++;
 return SUCCESS;
}
task void startCollect(){
```

```
        atomic {
                compensation = call GetRxBitOffset();
                call Leds.redToggle();
                call Timer2.setPeriodic(1);                     //30.5us
                call Timer1.start(TIMER_ONE_SHOT,100);
                CACTL1 = (CAON | CAREF_050 | CARSEL | CAIE);    //ComparatorA
                UsRecved = FALSE;
        }
 }
 async event void RadioReceiveCoord.startSymbol(uint8_t bitsPerBlock, uint8_t offset,
TOS_MsgPtr msgBuff) {};
 async event void RadioReceiveCoord.blockTimer() {};
 async event void RadioReceiveCoord.byte(TOS_MsgPtr msg, uint8_t cnt)
 {
    if (cnt == (offsetof(struct TOS_Msg,type) +
            sizeof(((struct TOS_Msg*)0)->type)) &&
        (msg->type == AM_BEACON) &&
        (TOS_LOCAL_ADDRESS !=0) &&
        (TOS_LOCAL_ADDRESS <=91))
                        post startCollect();
 }
  task void endCollect(){
        atomic{
        CACTL1 = 0x0000;                //ComparatorA Off
        call Timer2.stop();
        UsRecved = TRUE;
        }
  }
```

```
    TOSH_INTERRUPT(COMPARATORA_VECTOR) {          //Recv From US
        post endCollect();
        call Leds.greenToggle();
    }
}
```

ประวัติการศึกษา และการทำงาน

ชื่อ	นายสนธยา วิไลจิตต์
วัน เดือน ปี ที่เกิด	วันที่ 30 มิถุนายน 2515
สถานที่เกิด	สระบุรี
ประวัติการศึกษา	ปริญญาตรีวิศวกรรมไฟฟ้าโทรคมนาคม คณะวิศวกรรมศาสตร์ มหาวิทยาลัยศรีปทุม
ตำแหน่งหน้าที่การงานปัจจุบัน	เจ้าหน้าที่วิเคราะห์ระบบคอมพิวเตอร์ 4
สถานที่ทำงานปัจจุบัน	บริษัท ท่าอากาศยานไทย จำกัด (มหาชน) สาขาท่าอากาศยานสุวรรณภูมิ
ผลงานดีเด่นและรางวัลทางวิชาการ	
ทุนการศึกษาที่ได้รับ	