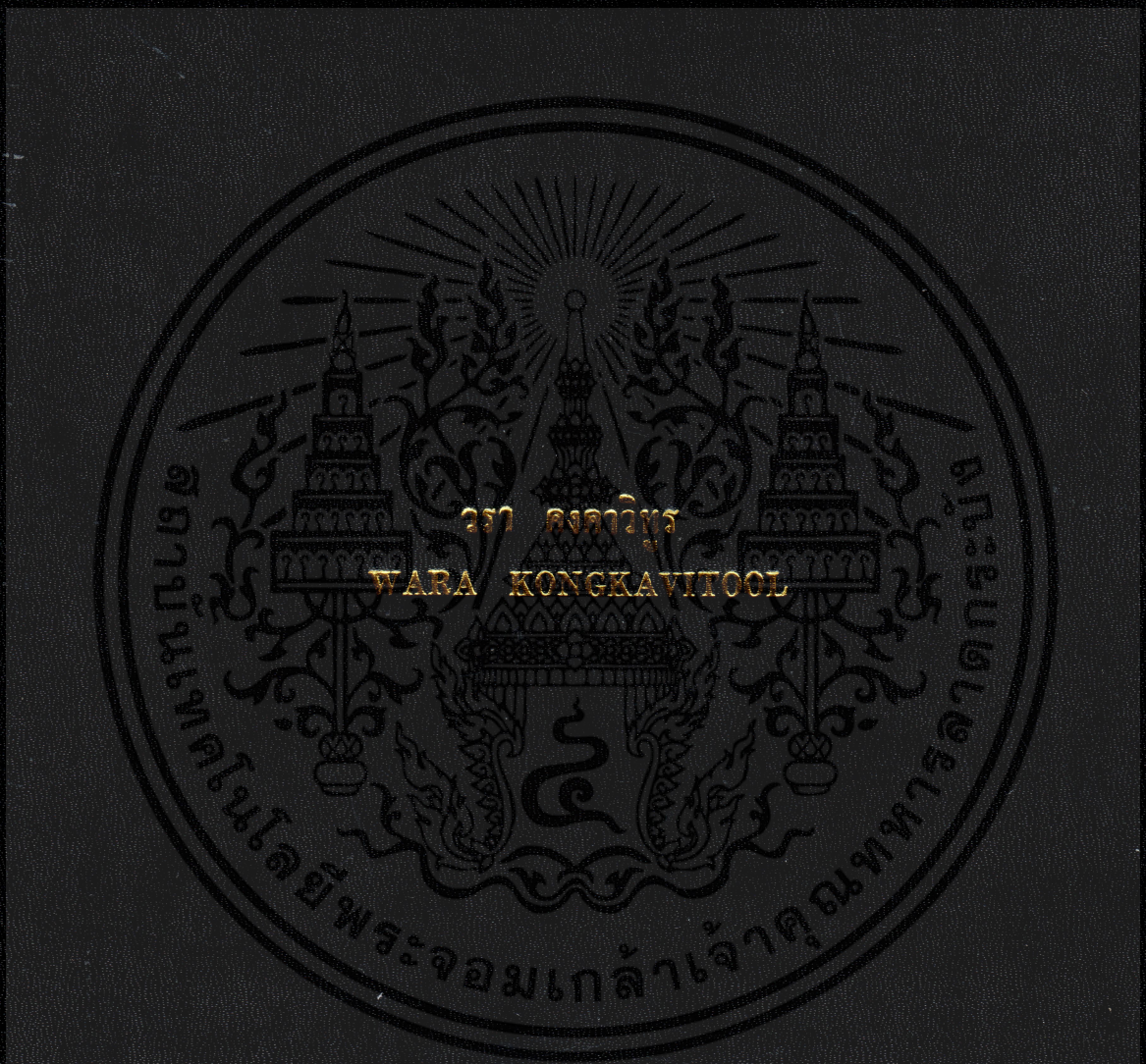
การรู้จำเสียงสระเสียงเดี่ยวในภาษาไทยโดยการใช้สเปกตรัมแอลพีซีบนสเกลบาร์ค

UNMIED VOWELS RECOGNITION IN THAI SPOKEN LANGUAGE BY USING LPC SPECTRUM ON THE BARK SCALE



วรา คงคาวิฑูร
WARA KONGKAVITOOL

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมไฟฟ้า
บัณฑิตวิทยาลัย
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ. 2544

ISBN 974-648-117-7

การรู้จำเสียงสระเสียงเดี่ยวในภาษาไทยโดยการใช้สเปกตรัมแอลพีซีบนสเกลบาร์ค

UNMIXED VOWELS RECOGNITION IN THAI SPOKEN LANGUAGE BY USING LPC SPECTRUM ON THE BARK SCALE

วรา คงคาวิทูร

WARA KONGKAVITOOL

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชาวิศวกรรมไฟฟ้า

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ.2544

ISBN: 974-648-117-7

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองวิทยานิพนธ์

หัวข้อวิทยานิพนธ์ การรู้จำเสียงสระเสียงเดี่ยวในภาษาไทยโดยการใช้สเปกตรัมแอลพีซีบนสเกลบาร์ค
UNMIXED VOWELS RECOGNITION IN THAI SPOKEN LANGUAGE BY USING LPC SPECTRUM ON THE BARK SCALE

ชื่อนักศึกษา นายวรา คงคาวิทูร

รหัสประจำตัว 41061036

ปริญญา วิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชา วิศวกรรมไฟฟ้า

อาจารย์ผู้ควบคุมวิทยานิพนธ์ ผศ.ดร.ไกรสิน ส่งวัฒนา

คณะกรรมการสอบวิทยานิพนธ์		ลายมือชื่อ
รศ.ดร.ยุทธพงษ์	รังสรรค์เสรี	
ผศ.ดร.ปัญญา	ฐิติมัชฌิมา	
รศ.ดร.ฟูศักดิ์	ชีวสุวิทย์	
รศ.ดร.สุวิพล	สิทธิชีวภาค	
ผศ.ดร.ไกรสิน	ส่งวัฒนา	

วัน/เดือนปี ที่สอบ 6 มีนาคม 2544 เวลา 12.00-13.00 น.

สถานที่สอบ ณ อาคาร 12 ชั้น ชั้น 4 (ห้อง E12-404)

บัณฑิตวิทยาลัยรับรองแล้ว

(รศ.ดร.บุญวัฒน์ อัตชู)

คณบดีบัณฑิตวิทยาลัย

วันที่ 26 **เดือน** มีนาคม **พ.ศ.** 2544

หัวข้อวิทยานิพนธ์	การรู้จำเสียงสระเสียงเดี่ยวในภาษาไทยโดยการใช้สเปกตรัม แอลพีซีบนสเกลบาร์ค
นักศึกษา	นายวรา คงคาวิทูร
รหัสประจำตัว	41061036
ปริญญา	มหาบัณฑิต
สาขาวิชา	วิศวกรรมไฟฟ้า
พ.ศ.	2544
อาจารย์ผู้ควบคุมวิทยานิพนธ์	ผศ.ดร. ไกรสิน ส่งวัฒนา

บทคัดย่อ

วิทยานิพนธ์นี้เสนอการสร้างแบบอ้างอิงของเสียงสระเสียงเดี่ยวในภาษาไทยจากสเปกตรัมแอลพีซีของสัญญาณเสียงบนสเกลบาร์ค ซึ่งทำให้สามารถแบ่งแยกเสียงสระดังกล่าวให้แตกต่างกันได้ 9 แบบ โดยในขั้นตอนแรกจะทำการหาสเปกตรัมแอลพีซีของเสียงสระแต่ละเสียง โดยสามารถหาได้จากสัมประสิทธิ์ของการทำ Linear Predictive Coding (LPC) แล้วจึงทำการแปลงสเปกตรัมแอลพีซีที่ได้ให้ไปอยู่ในสเกลบาร์คและทำการคำนวณความเข้มสัญญาณในแต่ละแถบความถี่วิกฤตบนสเกลบาร์คนั้น หลังจากนั้นจึงนำความเข้มสัญญาณในแต่ละความถี่วิกฤตที่ได้ไปทำการสร้างแบบอ้างอิงโดยการใช้โครงสร้างข้อมูลแบบ kd-tree และทดสอบประสิทธิภาพของแบบอ้างอิงโดยการรู้จำจะอาศัยเทคนิค k-Nearest Neighbor ในการตัดสินใจ

งานวิจัยนี้ได้ทำการหาสเปกตรัมแอลพีซีของเสียงสระเสียงเดี่ยวในภาษาไทยทั้ง 9 เสียง ซึ่งการใช้สเปกตรัมแอลพีซีจะสามารถกำจัดความแตกต่างของเสียงที่มีความถี่มูลฐานแตกต่างกันได้ ทำให้ความแตกต่างความถี่ของเสียงผู้หญิงและผู้ชายลดลงซึ่งส่งผลให้สามารถสร้างแบบอ้างอิงที่ใช้ร่วมกันทั้งผู้หญิงและผู้ชายได้ โดยในการทดสอบประสิทธิภาพในการรู้จำพบว่าให้ความถูกต้องเฉลี่ยมากกว่า 91 เปอร์เซ็นต์สำหรับแบบอ้างอิงชนิดไม่ขึ้นกับเพศ 93 เปอร์เซ็นต์สำหรับแบบอ้างอิงชนิดขึ้นกับเพศ และ 95 เปอร์เซ็นต์สำหรับแบบอ้างอิงชนิดขึ้นกับผู้พูด

Thesis Title	Unmixed Vowels Recognition in Thai spoken language by using LPC Spectrum on the Bark scale
Student	Mr. Wara Kongkavitool
Student ID	41061036
Degree	Master
Programme	Electrical Engineering
Year	2001
Thesis Advisor	Assist.Prof.Dr. Kraisin Songwatana

ABSTRACT

This thesis presents Recognition Model for nine Unmixed Vowels in Thai Spoken language by recognizing the respective LPC spectrum on the Bark scale. First, the LPC spectrum of each vowel is calculated from the speech signal, followed by applying Bark Transform to LPC spectrum and calculation of critical band on the Bark scale. Based on these critical band intensities we determine reference model by using kd-tree data structure.The reference models of 9 unmixed vowels are then tested by applying unknown sounds to the recognition procedure that use k-Nearest Neighbor technique for decisions.

The resultant reference models have the properties of fundamental frequency independent and makes possible gender independent unmixed vowels recognition. The experimental results showed the average recognition accuracy more than 91 percent for gender independent recognition. An accuracy of 93 percent is obtained for gender dependent models and 95 percent for speaker dependent model.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลงได้ด้วยความช่วยเหลืออย่างดียิ่งจากหลายๆฝ่าย ซึ่งผู้จัดทำใคร่ขอขอบคุณทุกๆท่านที่มีส่วนร่วมสนับสนุน ช่วยเหลือและแนะนำในทุกๆด้าน

ขอขอบพระคุณ ผศ.ดร ไกรสิน ส่งวัฒนา อาจารย์ที่ปรึกษาวิทยานิพนธ์ ที่ได้กรุณาเสียสละเวลา ให้คำปรึกษา และข้อเสนอแนะที่เป็นประโยชน์ตลอดจนห้องทดลองและการทำงาน ให้การทำวิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงได้ด้วยดี

ขอกราบขอบพระคุณบิดา มารดา ผู้ให้โอกาสและคอยให้กำลังใจเสมอมา

ขอขอบคุณพี่ๆและเพื่อนๆทั้งในและนอกห้องวิจัย ที่ช่วยเหลือให้คำปรึกษาและกำลังใจมาโดยตลอด

ขอขอบคุณเจ้าของเสียงที่ใช้ในการทดลองทุกๆท่านที่ให้ความร่วมมือและอดทนในการเก็บข้อมูลเพื่อทำการวิจัยหลายครั้งหลายคราวเป็นอย่างดี

สุดท้ายขอขอบคุณบัณฑิตวิทยาลัย ที่ได้ให้ทุนสนับสนุนการทำวิทยานิพนธ์ครั้งนี้

คุณค่าและประโยชน์อันพึงมีจากวิทยานิพนธ์ฉบับนี้ ผู้วิจัยขอมอบแด่ผู้มีพระคุณทุกท่าน

วรา คงคาวิทูร

สารบัญ

สารบัญ(ต่อ)

สารบัญ(ต่อ)

สารบัญตาราง

สารบัญรูป

สารบัญรูป(ต่อ)

สารบัญรูป(ต่อ)

บทที่ 1
บทนำ

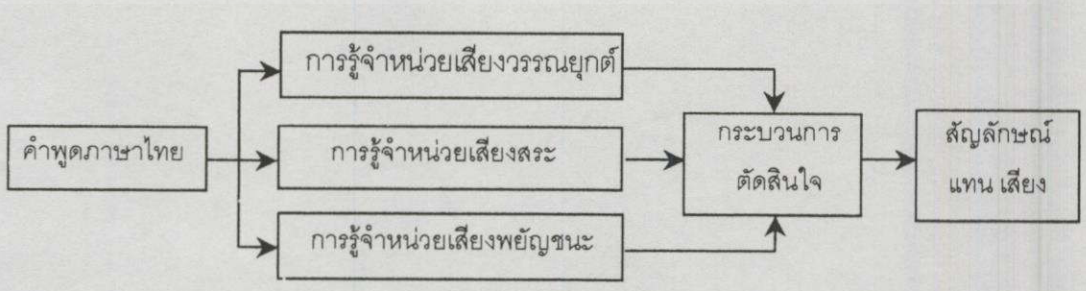
1.1 กล่าวนำ

ปัจจุบันเทคโนโลยีทางด้านคอมพิวเตอร์ได้ถูกพัฒนาให้มีขีดความสามารถมากขึ้น และนำมาใช้ร่วมกับเทคโนโลยีในด้านต่างๆ เพื่อทำให้การประมวลผลเป็นไปอย่างรวดเร็วและถูกต้อง ซึ่งโดยปกติการติดต่อสื่อสารระหว่างเครื่องคอมพิวเตอร์กับมนุษย์จะทำได้โดยการป้อนคำสั่งผ่านทางคีย์บอร์ดและเมาส์ ในขณะที่ได้มีความต้องการที่จะหาวิธีการอื่นที่สะดวกและเป็นธรรมชาติมากกว่า การพัฒนาให้คอมพิวเตอร์สามารถรับรู้คำสั่งจากเสียงพูดของมนุษย์ได้ จึงเป็นอีกเทคโนโลยีหนึ่งที่น่าสนใจซึ่งจะทำให้การติดต่อสื่อสารระหว่างมนุษย์กับคอมพิวเตอร์ทำได้ง่ายและสะดวกขึ้น

จากความต้องการให้เครื่องคอมพิวเตอร์สามารถรับรู้เสียงพูดได้ จึงทำให้เกิดศาสตร์แขนงหนึ่งเรียกว่า "Speech Recognition" แต่เนื่องจากการพูดของมนุษย์มีความซับซ้อน และมีความแตกต่างกันในแต่ละบุคคลจึงทำให้การพัฒนาเป็นไปอย่างล่าช้า โดยสามารถแบ่งวิธีการรับรู้เสียงพูดออกได้เป็น 2 วิธี คือ

1. พิจารณาทั้งหน่วยภาษาที่เปล่งเสียงออกมาทั้งชุด มีทั้งระบบการรู้จำคำเดี่ยว[1]-[2] (Isolated word Recognition) และระบบรู้จำคำพูดต่อเนื่อง (Continuous word Recognition) ซึ่งข้อดีของระบบเหล่านี้คือ ง่าย เนื่องจากมีการหลีกเลี่ยงผลกระทบอันเนื่องจากฐานของเสียงภายในคำหรือกลุ่มคำนั้น แต่ข้อเสีย คือสามารถรู้จำคำได้ในจำนวนคำที่จำกัด เนื่องจากต้องใช้เนื้อที่จำนวนมากในการจัดเก็บแบบจำลองอ้างอิง และต้องใช้เวลาในการคำนวณเพื่อเปรียบเทียบมากตามจำนวนของแบบจำลองอ้างอิงที่มีอยู่

2. พิจารณาโดยการแยกแยะรายละเอียดของหน่วยเสียง (Phonetic Recognition)[3]-[5] วิธีนี้จะพิจารณาลักษณะของหน่วยเสียงที่มีขนาดเล็กลงไป เช่น หน่วยเสียงพยัญชนะ หน่วยเสียงสระ และหน่วยเสียงวรรณยุกต์ ดังแสดงในรูปที่ 1.1 โดยจะใช้หน่วยเสียงย่อยเหล่านี้เป็นหลักในการรู้จำเสียงพูด ซึ่งวิธีนี้เหมาะสำหรับการพัฒนาไปสู่ระบบการรู้จำคำจำนวนมาก



รูปที่ 1.1 ส่วนประกอบของระบบการรู้จำภาษาไทยโดยวิธีแยกจำลักษณะของหน่วยเสียง

ด้วยเหตุนี้ เพื่อพัฒนาไปสู่ระบบการรู้จำเสียงพูดภาษาไทยทั้งภาษาซึ่งมีคำจำนวนมาก วิทยานิพนธ์นี้จึงทำการพัฒนาระบบการรู้จำแบบแยกแยะหน่วยเสียง โดยมุ่งหวังที่จะสร้างระบบการรู้จำหน่วยเสียงสระซึ่งเป็นส่วนสำคัญของคำ เนื่องจากหน่วยเสียงสระเป็นส่วนที่มากที่สุดหากมองในเชิงของเวลา และเนื่องจากในภาษาไทยมีสระมากทั้งสระเดี่ยว สระผสม สระเกิน[6] และยังแบ่งเป็นสระเสียงสั้นและสระเสียงยาว แต่องประกอบของเสียงสระทั้งหมดนั้นก็เกิดจากส่วนย่อยที่สุดของเสียงสระก็คือ เสียงสระเสียงเดี่ยวดังนั้นการรู้จำเสียงสระสำหรับภาษาไทยทั้งหมดนั้นก็จะต้องเริ่มต้นจากการรู้จำเสียงสระเสียงเดี่ยวซึ่งเป็นงานวิจัยในวิทยานิพนธ์นี้

1.2 วัตถุประสงค์ในการทำวิทยานิพนธ์

วิทยานิพนธ์ฉบับนี้เป็นการสร้างแบบจำลองการรู้จำเสียงสระเสียงเดี่ยวของภาษาไทยคำโดด (Monosyllabic) หรือคำพยางค์เดียว โดยใช้การหาสเปกตรัมแอลพีซีบนสเกลบาร์กแล้วนำมาสร้างแบบอ้างอิงแล้วใช้การตัดสินใจโดยอาศัยเทคนิค K-Nearest Neighbor โดยมีวัตถุประสงค์ดังนี้

1. เพื่อศึกษาลักษณะของเสียงสระที่แตกต่างกัน
2. เพื่อศึกษาและออกแบบระบบการรู้จำเสียงสระเสียงเดี่ยวในภาษาไทย โดยมุ่งเน้นให้แบบจำลองนี้ขึ้นนี้สามารถรู้จำเสียงพูดแบบต่างบุคคลและสามารถรู้จำเสียงสระได้ในคำที่ใช้พยัญชนะหรือวรรณยุกต์แตกต่างกันออกไปจากคำต้นแบบได้โดยระบบที่พัฒนาขึ้นนี้สามารถใช้ร่วนกันได้ทั้งผู้ชายและผู้หญิง
3. เพื่อหาสัมประสิทธิที่เหมาะสมที่สุดของการคำนวณสเปกตรัมแอลพีซี และในการทำ K-Nearest Neighbor เพื่อให้ได้อัตราความถูกต้องในการรู้จำมากที่สุด
4. เพื่อเป็นองประกอบในการพัฒนาไปสู่ระบบการรู้จำคำพูดภาษาไทยทั้งระบบ

1.3 ข้อกำหนดในการทำวิทยานิพนธ์

1. งานวิจัยนี้มุ่งศึกษาพัฒนาอัลกอริธึมบนเครื่องคอมพิวเตอร์ส่วนบุคคล โดยใช้โปรแกรมภาษา Borland C++ Builder โดยมีอุปกรณ์เพิ่มเติมได้แก่ การ์ดเสียง (Sound Blaster AWE64) ไมโครโฟน และลำโพง
2. ข้อมูลเสียงที่ใช้ในวิทยานิพนธ์ฉบับนี้ เป็นเสียงที่มีสำเนียงภาคกลางเท่านั้น
3. ในการทดลองได้ทำการเก็บตัวอย่างเสียงจากผู้ออกเสียง 20 คน ที่เป็นชาย 10 คน และหญิง 10 คน โดยคำที่ใช้ประกอบด้วย คำพยางค์เดียวที่เกิดจากเสียงสระเสียงเดี่ยว 18 เสียง(เสียงสั้น 9 เสียงและเสียงยาว 9 เสียง) ผสมกับเสียงพยัญชนะ 21 เสียง และเสียงวรรณยุกต์อีก 5 ระดับเสียง รวมเป็น 37,800 คำ

1.4 โครงประกอบของวิทยานิพนธ์

แบ่งออกเป็น 7 บทดังนี้

บทที่ 1 ดังได้กล่าวมาแล้วข้างต้น

บทที่ 2 กล่าวถึงระบบการพูดของมนุษย์ในด้านองค์ประกอบของสรีระและหน้าที่ของอวัยวะต่างๆ ในการเปล่งเสียงพูด และหน่วยเสียงที่ประกอบกันขึ้นเป็นพยางค์ของคำในภาษาไทย

บทที่ 3 กล่าวถึงการดึงเอาลักษณะพารามิเตอร์ที่ต้องการออกมาจากไฟล์เสียง ซึ่งก็คือสเปกตรัมแอลพีซี

บทที่ 4 กล่าวถึงขั้นตอนและทฤษฎีในการแปลงสเปกตรัมแอลพีซีให้มาอยู่ในสเกลบาร์กและการหาเวคเตอร์ตัวแทนหน่วยเสียง

บทที่ 5 กล่าวถึงทฤษฎีการสร้างแบบอ้างอิงโดยใช้โครงสร้างข้อมูลแบบ KD-Tree และการตัดสินใจโดยใช้เทคนิค K-Nearest Neighbor

บทที่ 6 เป็นขั้นตอนในการทดลอง

บทที่ 7 เป็นบทสรุปเกี่ยวกับการทดลองทั้งหมดที่ทำมา พร้อมทั้งข้อสังเกต ปัญหาที่พบในการทดลอง และข้อเสนอแนะสำหรับผู้ที่ทำการวิจัย และพัฒนาระบบการรู้จำเสียงพูดต่อไป

ภาคผนวก

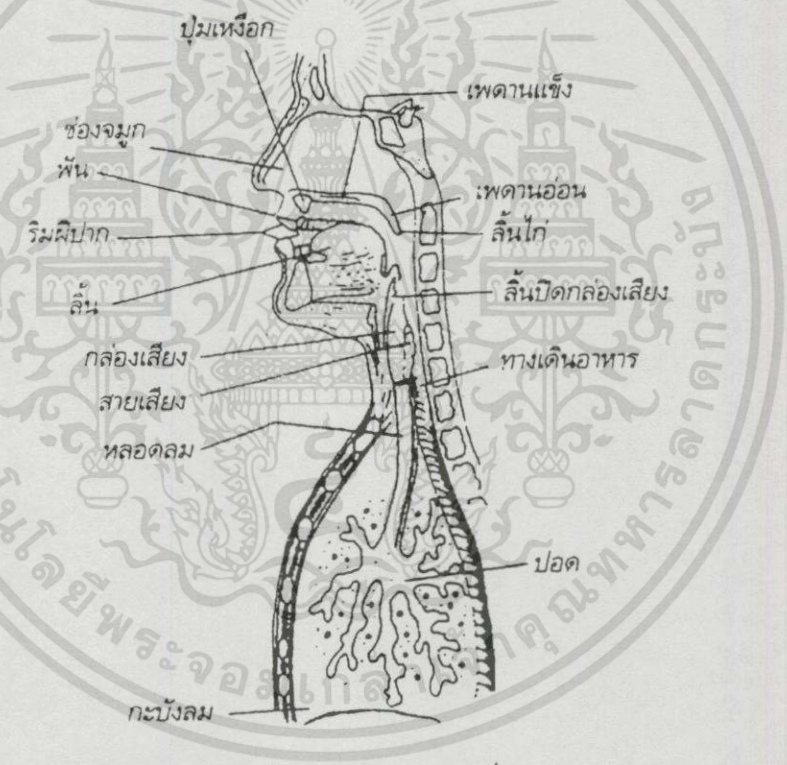
ภาคผนวก ก

ภาคผนวก ข

บทที่ 2
ระบบเสียงในภาษาไทย

2.1 ทฤษฎีการสร้างเสียงพูด

การพูดของมนุษย์มิใช่อาการที่เกิดเฉพาะที่ปากเท่านั้น หากเริ่มจากลมหายใจเข้าของมนุษย์เองที่นำลมเข้าสู่ปอด จากนั้นจะใช้ลมจากปอดซึ่งก็คือลมหายใจออก มาทำให้เกิดเสียงพูด โดยลมจะถูกบังคับให้ผ่านอวัยวะต่างๆที่สำคัญ คือ เส้นเสียงซึ่งอยู่ในช่องของหลอดลม หรือบริเวณที่เรียกว่าลูกกระเดือก แล้วลมนั้นจะผ่านต่อออกมาทางช่องปากหรือช่องจมูก อวัยวะในช่องปากก็จะคัดแปลงให้เป็นเสียงต่างๆตามที่ผู้พูดต้องการ [7]



รูปที่ 2.1 ภาพตัดขวางแสดงอวัยวะในระบบการพูดของมนุษย์

2.1.1 อวัยวะที่ใช้ในการออกเสียงพูด

อวัยวะส่วนที่มีหน้าที่โดยตรงในการออกเสียงพูด ดังแสดงในรูปที่ 2.1 มีดังนี้คือ

1. ริมฝีปาก เป็นอวัยวะส่วนที่เคลื่อนไหวได้มาก และทำให้เสียงแตกต่างกันได้มาก เราอาจจะบังคับริมฝีปากให้อยู่ชิดกัน ห่างกัน ยื่นออก หรือห่อกลม ฯลฯ ลักษณะริมฝีปากต่างๆนี้ล้วนแต่มีอิทธิพลต่อการออกเสียง และการทำให้เสียงแตกต่างกันไปทั้งสิ้น

2. ฟัน เป็นอวัยวะที่เกิดของเสียงหลายชนิด เช่นเมื่อฟันบนกดลงบนริมฝีปากล่าง หรือฟันล่าง ลมที่ผ่านออกมาโดยแรงจะลอดช่องที่พอผ่านได้ออกมา ทำให้เกิดเป็นเสียงชนิดที่เรียกว่าเสียงเสียดแทรก เป็นต้น

3. ปุ่มเหงือก เป็นส่วนนูนออกมาอยู่หลังฟันด้านบน ถ้าเอาลิ้นแตะดูจะรู้สึกว่ามีลักษณะเป็นคลื่น

4. เพดานแข็ง หรือ เพดานปาก คือ ส่วนเฉพาะที่โค้งเป็นกระดูกแข็ง

5. เพดานอ่อน คือ ส่วนของเพดานที่อยู่ต่อจากเพดานแข็งไปข้างในมีลักษณะเป็นกระดูกอ่อนที่ขยับขึ้น-ลงได้ เวลาหายใจเพดานอ่อนและลิ้นไก่ซึ่งอยู่ตอนปลายจะลดระดับลงมาเปิดช่องให้ลมออกไปทางจมูก เวลาพูดส่วนใหญ่ปลายเพดานอ่อนและลิ้นไก่จะถูกยกขึ้นไปจรดกับหลังคอ นอกจากเวลาออกเสียงนาสิกเท่านั้นที่เพดานอ่อนจะลดระดับลงมา เพื่อให้ลมออกทางช่องจมูก

6. ลิ้นไก่ เป็นก้อนเนื้อเล็กๆอยู่ต่อปลายเพดานตรงกลางปาก อวัยวะส่วนนี้สั่นรัวได้

7. ลิ้น เป็นส่วนที่เคลื่อนไหวมากที่สุดในการออกเสียงพูด จึงต้องแบ่งออกเป็น 3 ส่วน ตามหน้าที่ในการออกเสียง คือ

   7.1) ปลายลิ้น คือ ส่วนปลายลิ้นซึ่งสามารถยกขึ้นไปแตะอวัยวะส่วนต่างๆ ในปากตอนบนได้โดยง่าย

   7.2) หน้าลิ้น คือ ลิ้นที่อยู่ตรงข้ามกับเพดานแข็ง

   7.3) หลังลิ้น คือ ส่วนของลิ้นที่อยู่ตรงข้ามกับเพดานอ่อน

8. แผ่นเนื้อปากหลอดลมเป็นก้อนเนื้อเล็กๆคล้ายลิ้นไก่ อยู่ต่อโคนลิ้นลงไปในคอ มีหน้าที่ปิดช่องลมเมื่อรับประทานอาหาร และเปิดช่องลมเมื่อพูด

9. กรวยคอ หมายถึง โพรงคอที่อยู่ถัดจากปากลงไปจนถึงเส้นเสียง

10. เส้นเสียง หรือ สายเสียง เป็นอวัยวะสำคัญที่เกิดของเสียง เส้นเสียงมีลักษณะเป็นกล้ามเนื้อ 2 แผ่นปิดขวาง อยู่บริเวณปากช่องหลอดลมจากด้านหลังมาด้านหน้า ระหว่างเส้นเสียงจะมีช่องว่าง ซึ่งเป็นทางผ่านให้ลมเข้าถึงปอดและออกมาจากปอดได้ ช่องว่างนี้เรียกว่า ช่องระหว่างเส้นเสียง (Glottis) เส้นเสียงทั้งสองสามารถดึงออกให้ห่างจากกันหรือดึงเข้าหากันได้ ซึ่งเส้นเสียงนี้เป็นส่วนสำคัญที่ทำให้เกิดเสียงพูดขึ้นในภาษา

11. ช่องจมูก หมายถึง โพรงในช่องจมูก ซึ่งอยู่เหนือลิ้นไก่ขึ้นไป เป็นช่องที่ลมซึ่งผ่านเส้นเสียงขึ้นมาจะผ่านออกไปทางจมูกได้เมื่อเวลาหายใจและเวลาออกเสียงนาสิก ในเวลาที่พูดเสียงอื่นลิ้นไก่จะถูกยกขึ้นไปปิดช่องจมูก เพื่อให้ลมออกมาทางปาก

12. เส้นเสียงปลอม เป็นอวัยวะที่มีลักษณะเหมือนเส้นเสียงแต่อยู่เหนือเส้นเสียงขึ้นไป เส้นเสียงปลอมนี้เข้าใจกันว่าจะดึงเข้าหากันเมื่อเวลาพูดเสียงกระซิบ

2.1.2 ลักษณะร่วมของเสียงพูด

เสียงที่ใช้ในภาษาพูดนั้นจะมีลักษณะที่สำคัญบางประการร่วมกัน ซึ่งเรียกได้ว่าเป็นลักษณะร่วมของเสียงพูด ลักษณะที่กล่าวถึงนี้มีอยู่หลายประการ [8] คือ

1. ความก้อง หรือ ไม่ก้องของเสียง

เสียงก้อง หรือ เสียงโฆษะ (Voice)

คือเสียงที่เกิดในขณะที่เส้นเสียงเกิดการตึงตัวหรือเรียกว่าเส้นเสียงปิด เมื่อมีแรงดันให้อากาศไหลผ่านกล่องเสียงในขณะที่เส้นเสียงปิดจะเกิดการสั่นสะบัดของเส้นเสียง เป็นผลให้สัญญาณเสียงที่ได้ (speech waveform) มีลักษณะเป็นคาบ (quasi-periodic) ซึ่งสามารถเรียกความถี่ในการปิด-เปิดของเส้นเสียงนี้ว่า "ความถี่มูลฐาน" (Fundamental Frequency: F_0) ตัวอย่างของเสียงก้องได้แก่ เสียงสระต่างๆ และเสียงพยัญชนะเช่น บ ด ที่เกิดจากการเปล่งเสียงออกทางปาก หรือเสียงพยัญชนะ ม น ง ที่เกิดจากการเปล่งเสียงออกทางจมูก

เสียงไม่ก้อง หรือเสียงอโฆษะ (Unvoiced หรือ voiceless)

คือเสียงที่เกิดในขณะที่เส้นเสียงคลายจากการตึงหรือเรียกว่าเส้นเสียงเปิด เมื่อมีแรงดันให้อากาศไหลผ่านกล่องเสียงในขณะที่เส้นเสียงเปิด อากาศที่ไหลผ่านอย่างรวดเร็วจะเกิดการไหลวนและปั่นป่วนทำให้เกิดเสียงที่มีลักษณะเป็นเสียงของสัญญาณรบกวน (Noise) ซึ่งไม่เป็นคาบ ตัวอย่างของเสียงไม่ก้องได้แก่ เสียงพยัญชนะ ฟ ซ ส ฯลฯ หรือเกิดจากการสร้างแรงดันอากาศหลังตำแหน่งปิดกั้นของช่องทางเดินเสียง และเมื่อการปิดกั้นนี้ถูกเปิดออก อากาศจะถูกปล่อยออกมาอย่างทันทีทันใดเกิดเป็นเสียงที่เรียกว่าเสียงระเบิด (Plosive Sound) เช่น การเปล่งเสียงเริ่มแรกของพยัญชนะต้นของคำต่างๆ

2. ความยาวของเสียง (Length)

หมายถึง การที่เสียงใดเสียงหนึ่งเปล่งออกมาได้นานเท่าใด เสียงพูดบางเสียงอาจจะเปล่งออกมาได้ติดต่อกันได้นาน เช่น เสียงสระ เสียงพยัญชนะนาสิก หรือ เสียงพยัญชนะเสียดแทรก

ในภาษาไทย เสียงพูดที่มีความยาว-สั้น ก็มีเพียงเสียงสระเท่านั้น เช่น อะ อิ อุ เป็นเสียงสั้น อา อี อู เป็นเสียงยาวเป็นต้น

3. ระดับเสียงสูง-ต่ำ (Pitch)

เสียงพูดจะมีระดับ สูง หรือ ต่ำ อยู่ที่ความถี่ของเสียง (Fundamental frequency) ถ้าความถี่ต่ำเสียงก็จะต่ำ อวัยวะส่วนที่ทำให้เสียงมีระดับ สูง-ต่ำ คือเส้นเสียง ดังนั้นระดับเสียงสูง-ต่ำก็คืออัตราการสั่นสะบัดของเส้นเสียงนั่นเอง

ในการพูดเสียงที่มีระดับสูง-ต่ำได้คือเสียงก้องเท่านั้นเพราะมีการสั่นสะเทือนของเส้นเสียงที่ทำให้เกิดมีความถี่ระดับต่างๆได้ ในภาษาไทยระดับเสียง สูง-ต่ำ ของคำเราเรียกว่า "วรรณยุกต์"

4. ความดัง (Loudness)

ความดังขึ้นอยู่กับปริมาณของลม ที่ผู้พูดเปล่งเสียงออกมาในช่วงเวลาหนึ่งๆ

5. การลงน้ำหนัก (Stress)

หมายถึง การออกเสียงพยางค์ใดพยางค์หนึ่งให้ดังเน้นมากหรือน้อยกว่าพยางค์อื่นที่อยู่ข้างเคียง (เพื่อต้องการเรียกร้องความสนใจเป็นพิเศษ หรือแสดงอารมณ์อย่างใดอย่างหนึ่ง)

6. ช่วงต่อของเสียง (Juncture)

หมายถึงช่วงระยะที่ผู้พูดเปล่งเสียงหนึ่งแล้วต่อไปเปล่งอีกเสียงหนึ่งซึ่งเรียงกันมาเป็นลำดับ เสียงที่ประกอบกันเข้าเป็นพยางค์จะมีช่วงต่อของเสียงแนบสนิทจนไม่เห็นร่องรอย (close juncture) แต่ถ้าเสียงปรากฏอยู่คนละพยางค์หรือคนละคำ จะมีช่วงต่อ "ห่าง" จนสังเกตได้ชัด (open juncture) ดังนั้นช่วงต่อของเสียง โดยเพาะช่วงต่อห่างจะมีความสำคัญมากในการแบ่งคำในภาษา

2.2 หน่วยเสียงสำคัญในภาษาไทย

"หน่วยเสียง" (phoneme) เป็นหน่วยเล็กที่สุดของภาษา หน่วยดังกล่าวได้แก่เสียงสำคัญๆ ในภาษาใดภาษาหนึ่ง ซึ่งทำหน้าที่ให้ความหมายของคำที่ใช้ในภาษานั้น และทำให้ความหมายของคำนั้นๆมีความหมายแตกต่างจากคำอื่นๆ หน่วยเสียงสำคัญในภาษาไทยมี 3 ประเภทใหญ่ๆคือ เสียงพยัญชนะ เสียงสระ และเสียงวรรณยุกต์ หน่วยเสียงทั้ง 3 นี่เองที่ประกอบกันเข้าเป็นคำที่ใช้ในภาษาไทย

เสียงพูดของมนุษย์ซึ่งมีความแตกต่างกันมากมายนั้นถ้าเราพิจารณาอย่างกว้างๆจะพบว่าสามารถแบ่งออกเป็น 2 ประเภทใหญ่ คือ

1. เสียงเรียง (segmental sound) เป็นหน่วยเสียงที่สามารถแยกออกจากเสียงอื่นได้โดยเด็ดขาด เพราะมีลักษณะเด่นเฉพาะตัว ในภาษาไทยได้แก่เสียงสระ และเสียงพยัญชนะ
2. เสียงซ้อน (supra-segmental feature) เป็นเสียงที่ทำหน้าที่เป็นส่วนประกอบของเสียงอื่นเพราะไม่สามารถแยกเปล่งเสียงได้ตามลำพัง ในภาษาไทยได้แก่เสียงวรรณยุกต์และทำนองเสียง เป็นต้น

2.3 หน่วยเสียงสระ

2.3.1 ลักษณะของเสียงสระ

ลักษณะสำคัญของเสียงสระก็คือ "เป็นเสียงก้องที่เปล่งเสียงออกมาโดยให้ลมออกทางช่องปากโดยไม่ถูกลิ้นกักหรือขัดขวาง" ดังนั้นเวลาเราออกเสียงสระจะออกเสียงได้สะดวกและออกเสียงได้นาน ทั้งนี้เพราะคุณสมบัติของเสียงสระมีความดังเด่นกว่าเสียงอื่นๆที่เรียงอยู่ข้างเสมอ อวัยวะที่เกี่ยวข้องกับการออกเสียงสระได้แก่ ลิ้น กับริมฝีปาก ถ้าลิ้นส่วนใดทำหน้าที่เพียงส่วนเดียว เสียงที่เกิดขึ้นก็จะมีเพียงเสียงเดียว เสียงเช่นนี้เรียกว่า "สระเดี่ยว" แต่ถ้าลิ้นส่วนอื่นทำหน้าที่ร่วมด้วยเสียงสระนั้นเรียงว่า "สระประสม"

สำหรับภาษาไทยมีหน่วยเสียงสระทั้งหมด 24 หน่วยเสียง แยกออกเป็นสระเดี่ยว 18 หน่วยเสียง และสระประสม 6 หน่วยเสียง [9]

สระเดี่ยว

เสียงสระเดี่ยว 18 หน่วยเสียง พิจารณาการเกิดเสียงได้เป็น 2 กรณีใหญ่ๆ คือ

1. การเกิดจากส่วนต่างๆของลิ้นหมาย ถึง ลมผ่านส่วนหน้า ส่วนกลาง หรือส่วนหลังของลิ้น
2. การเกิดจากลมผ่านลิ้นในขณะที่ลิ้นอยู่ในระดับ สูง กลาง หรือ ต่ำ

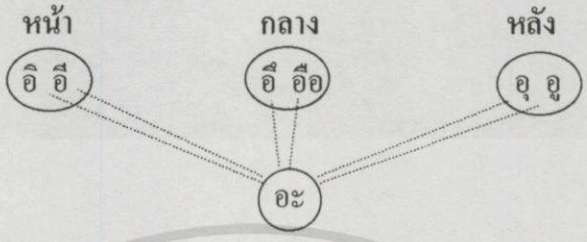
สระ	ระดับลิ้น	หน่วยเสียง	สัญลักษณ์
หน้า	สูง	อิ อี	“i”,“i:”
	กลาง	เอะ เอ	“e”,“e:”
	ต่ำ	แอะ แอ	“ɛ”,“ɛ:”
กลาง	สูง	อึ ออ	“ɯ”,“ɯ:”
	กลาง	เออะ เออ	“ɤ”,“ɤ:”
	ต่ำ	อะ อา	“a”,“a:”
หลัง	สูง	อุ อู	“u”,“u:”
	กลาง	โอะ โอ	“o”,o:”
	ต่ำ	เอาะ ออ	“ɔ”,“ɔ:”

นอกจากนี้ หน่วยเสียงสระเดี่ยว 18 หน่วย สามารถแบ่งตามความสั้น-ยาวของการออกเสียงได้เป็น

- สระเดี่ยวเสียงสั้น 9 หน่วย ได้แก่ อะ อิ อึ อุ เอะ แอะ โอะ เอาะ เออะ
- สระเดี่ยวเสียงยาว 9 หน่วย ได้แก่ อา อี ออ อู เอ แอ โอ ออ เออ

สระประสม

เสียงสระประสม 6 หน่วยเสียง เกิดจากลมผ่านกระทบลิ้น 2 ส่วนคือส่วนบนและส่วนล่าง ซึ่งในขณะออกเสียงลิ้นจะอยู่ในระดับสูงแล้วลดลงต่ำ โดยเสียงหลังเป็นเสียงสระ อะ เสมอดังแผนผังดังนี้



เสียงสระประสม 6 หน่วยเสียงได้แก่ เอียะ (อิ+อะ) เอีย (อี+อะ) เอือะ (อึ+อะ) เอือ (ออ+อะ) อัวะ (อุ+อะ) อัว (อู+อะ)

2.3.2 หน้าที่ของหน่วยเสียงสระในภาษาไทย

หน่วยเสียงสระในภาษาไทยทั้ง 24 หน่วยเสียงนี้ ทำหน้าที่เป็นแกนกลางของพยางค์หรือคำ กล่าวคือ คำ ทุกคำในภาษาไทยจะต้องมีเสียงสระอยู่ด้วย และเสียงสระในภาษาไทยจะสามารถเกิดกับเสียงพยัญชนะต้นได้ทุกเสียงและสามารถเกิดกับหน่วยเสียงวรรณยุกต์ได้ทุกหน่วย แต่ไม่สามารถเกิดกับหน่วยเสียงพยัญชนะสะกดได้ทุกหน่วย หน่วยเสียงสระที่ทำให้เกิดคำหรือพยางค์ใช้ได้มากที่สุดในภาษามักเป็นหน่วยเสียงสระยาว

2.4 หน่วยเสียงพยัญชนะ

เสียงพยัญชนะในภาษาไทยมีทั้งหมด 21 หน่วยเสียง(44 รูป) ดังแสดงในตารางที่ 2.1 หน่วยเสียงพยัญชนะออกเสียงได้ไม่สะดวกเท่าหน่วยเสียงสระ เพราะเวลาออกเสียงลมหายใจที่พุ่งออกมาจากหลอดลมจะถูกขัดขวางตามส่วนต่างๆของปาก เสียงพยัญชนะจึงออกเสียงให้ยาวนานอย่างเสียงสระไม่ได้ และเสียงพยัญชนะก็ไม่ใช่เสียงก้องเสมอไป

2.4.1 ลักษณะของเสียงพยัญชนะ

หน่วยเสียงพยัญชนะ 21 หน่วยเสียงนี้จำแนกเป็น เสียงก้อง เสียงไม่ก้อง เสียงหนัก เสียงเบา และลักษณะการเกิดเสียง ดังนี้

เสียงก้อง (โฆษะ) มี 9 หน่วยเสียง คือ /ง/ /ย/ /บ/ /ด/ /ม/ /น/ /ร/ /ล/ /ว/

เสียงไม่ก้อง (อโฆษะ) มี 12 หน่วยเสียง คือ /ก/ /ค/ /จ/ /ช/ /ซ/ /ท/ /ต/ /ป/ /พ/ /ฟ/ /อ/ /ฮ/

เสียงหนัก (ธนิต) มี 4 หน่วยเสียง คือ /ค/ /ช/ /ท/ /พ/

เสียงเบา (สถิล) มี 4 หน่วยเสียง คือ /ก/ /จ/ /ต/ /ป/

ตารางที่ 2.1 เสียงพยัญชนะในภาษาไทย

ลำดับที่	อักษรไทยใช้แทนหน่วยเสียง	แทน สัญลักษณ์หน่วยเสียง	
		แบบสากล	แบบไทย
1.	ก	k	ก
2.	ข ฃ ค ฅ ฆ	kh	ค
3.	ง	η	ง
4.	จ (จร_*)	c	จ
5.	ฉ ช ฌ	ch	ช
6.	ญ ย (หย_*) (หญ_*)	j	ย
7.	ซ ศ ษ ส (ทร_*)	s	ซ
8.	ฐ ฑ ฒ ถ ท ธ (ทร_*)	th	ท
9.	บ	b	บ
10.	ฎ ด (ฑ_*)	d	ด
11.	ฏ ต	t	ต
12.	ป	p	ป
13.	ผ พ ภ	ph	พ
14.	ฝ ฟ	f	ฟ
15.	ม (หม_*)	m	ม
16.	น ณ (หน_*)	n	น
17.	ร	r	ร
18.	ล ฬ (หล_*)	l	ล
19.	ว (หว_*)	w	ว
20.	อ	?	อ
21.	ฮ ห	h	ฮ
อักษร 44 รูป		21 หน่วยเสียง	

<u>หมายเหตุ</u> (*) หมายถึง พยัญชนะที่อยู่ในตำแหน่งพยัญชนะต้นแล้วออกเสียงเช่นเดียวกับเสียงพยัญชนะที่อยู่ในลำดับนั้นๆ

ลักษณะของเสียงพยัญชนะ จำแนกตามลักษณะรูปเสียง (classification by form)

1. เสียงกัก หรือ เสียงหยุด (Stop)

เป็นเสียงที่เมื่อผ่านกล่องเสียงเข้ามาถึงช่องปากแล้ว ในปากจะมีฐานกรณ์แห่งใดแห่งหนึ่งกั้นเสียงนี้ไว้ไม่ให้ออกจากปากแต่การกั้นเป็นเพียงชั่วระยะเวลาอันสั้นเท่านั้น แล้วฐานกรณ์ที่กั้นนั้นจะเปิดออก อากาศที่ถูกกักไว้จะถูกปล่อยออกมา เนื่องจากอากาศถูกกั้นไว้เมื่อถูกปล่อยออกมาจึงออกมาในลักษณะระเบิด บางทีจึงเรียกเสียงประเภทนี้ว่าเป็นเสียงระเบิด (plosive sound) มี 9 หน่วยเสียง คือ /ป/ /พ/ /บ/ /ต/ /ท/ /ด/ /ก/ /ค/ /อ/ เสียงพยัญชนะสะกดทุกเสียงในภาษาจะมีลักษณะเป็นเสียงระเบิด หรือเสียงกัก

2. เสียงเสียดแทรก (Fricative)

เป็นเสียงที่เมื่ออากาศผ่านขึ้นมาจากปอดผ่านกล่องเสียงเข้ามาถึงช่องปากแล้วในปากจะมีฐานกรณ์แห่งใดแห่งหนึ่งกั้นอากาศนี้ไว้ แต่การกั้นนี้ไม่สนิทมิดชิดเหมือนเสียงหยุด ยังมีช่องให้อากาศเล็ดลอดแทรกออกมาได้ทำให้เกิดเสียงขณะแทรกออกมา มี 3 หน่วยเสียงคือ /ซ/ /ฟ/ /ฮ/

3. เสียงกึ่งเสียดแทรก (Affricate)

เป็นเสียงในช่องปากที่มีคุณสมบัติเหมือนกับเริ่มต้นด้วยเสียงหยุดและตามด้วยเสียงแทรก มี 2 หน่วยเสียง คือ /จ/ และ /ช/

4. เสียงนาสิก (Nasal)

เป็นเสียงที่เมื่ออากาศผ่านกล่องเสียง ผ่านช่องคอแล้วก็เข้าสู่ช่องจมูก โดยที่ช่องปากมีฐานกรณ์กั้นไว้สนิทไม่ให้อากาศออกทางช่องปาก เสียงที่อากาศผ่านออกมาทางช่องจมูก มี 3 หน่วยเสียง คือ /ม/ /น/ /ง/

5. เสียงข้าง (Lateral)

เป็นเสียงที่อากาศในช่องปากออกสู่ภายนอกปากโดยผ่านทางข้างๆลิ้น มีหน่วยเสียงเดียว คือ /ล/

6. เสียงรัว (Trill)

คือเสียงที่เมื่ออากาศเข้ามาอยู่ในช่องปากแล้วมีการกระดกปลายลิ้นรัวเพดานหลายๆครั้ง มีหน่วยเสียงเดียว คือ /ร/

7. เสียงครึ่งสระ (Semi-Vowel)

คำอธิบายทั่วๆไปของเสียงประเภทนี้ไม่ค่อยชัดแจ้งนัก มักกล่าวว่าตำแหน่งลิ้นเมื่อเริ่มต้นเสียงต่างไปจากตำแหน่งในตอนท้ายๆเสียง เสียงประเภทนี้บางครั้งก็เรียกว่าเสียงครึ่งสระ เพราะมีตำแหน่งลิ้นและปากคล้ายเสียงสระ มี 2 หน่วยเสียง คือ เสียง /ว/ และ /ย/

2.4.2 หน้าที่ของหน่วยเสียงพยัญชนะในภาษาไทย

เสียงพยัญชนะในภาษาไทย 21 หน่วยเสียงนี้สามารถทำหน้าที่ได้ดังนี้

1. เป็นพยัญชนะต้นของพยางค์ คือสามารถนำหน้าเสียงสระในพยางค์หนึ่งๆได้ ในตำแหน่งนี้เสียงพยัญชนะสามารถเกิดได้หน่วยเดียว หรือ สองหน่วยดังนี้

- เกิดได้ หน่วยเดียว คือ ทำหน้าที่เป็นพยัญชนะต้นเดี่ยว หน่วยเสียงทั้ง 21 หน่วยเสียงนี้สามารถทำหน้าที่เป็นพยัญชนะต้นเดี่ยวได้ทั้งสิ้น
- เกิดได้ สองหน่วย คือ ทำหน้าที่เป็นพยัญชนะต้นควบ โดยหน่วยเสียงแรกเป็น /ก/ /ค/ /ต/ /ป/ และ /พ/ กับหน่วยเสียงที่สองเป็น /ร/ /ล/ หรือ/ว/

2. เป็นพยัญชนะสะกดของพยางค์ ในตำแหน่งนี้เสียงพยัญชนะในภาษาไทยสามารถเกิดได้ 9 หน่วยเสียง คือ /ป/ (แม่กบ) /ต/ (แม่กด) /ก/ (แม่กก) /ม/ (แม่กม) /ง/ (แม่กง) /น/ (แม่กน) /ย/ (แม่เกย) /ว/ (แม่เกอว) และ ไม่มีเสียงพยัญชนะสะกด (แม่กา)

2.5 หน่วยเสียงวรรณยุกต์

เสียงวรรณยุกต์ คือ ระดับเสียงสูง-ต่ำ ของคำในภาษาไทย เช่นเดียวกับภาษาจีน และภาษาอื่นๆ ที่เป็นภาษาคำโดดซึ่งมีการกำหนดเสียงสูงต่ำไว้ตายตัวในคำแต่ละคำ ถ้าออกเสียงสูง-ต่ำผิดไปความหมายย่อมผิดตามไปด้วย

ในภาษาไทยหน่วยเสียงวรรณยุกต์เป็นหน่วยเสียงสำคัญ ที่ทำให้คำที่มีส่วนประกอบแวดล้อมอื่นๆเหมือนกัน คือมี เสียงพยัญชนะต้น สระ และพยัญชนะสะกดอย่างเดียวกันมีความหมายต่างกัน ดังนั้นอาจกล่าวได้ว่าหน้าที่ของหน่วยเสียงวรรณยุกต์ก็คือ การทำให้เกิดคำขึ้นใช้ในภาษามากขึ้นและเป็นวิธีการสร้างคำขึ้นใช้เพิ่มขึ้นในภาษาเป็นวิธีแรก ทั้งนี้เพราะถ้าเราเปลี่ยนเสียงวรรณยุกต์ก็จะทำให้คำเกิดความหมายเพิ่มขึ้นใหม่นั่นเอง

เสียง สูง-ต่ำ ในภาษาพูด เกิดจากการสั่นสะเทือนของเส้นเสียงในอัตราต่างๆกัน โดยเสียงที่เปล่งออกมาในขณะที่เส้นเสียงสั่นนั้นจะต้องเป็นเสียงก้อง ดังนั้นหน่วยเสียงวรรณยุกต์ในภาษาไทยจึงจัดเป็นหน่วยเสียงซ้อน สัทอักษรที่ใช้จึงเป็นรูปเครื่องหมายเขียนซ้อนข้างบนหน่วยเสียงสระ(ซึ่งเป็นเสียงก้อง) ซึ่งมีรูปวรรณยุกต์อยู่ 4 รูป แทนเสียงวรรณยุกต์ทั้งหมด 5 หน่วยเสียง โดยเสียงสามัญไม่มีรูปวรรณยุกต์

2.5.1 ลักษณะของเสียงวรรณยุกต์

สามารถแบ่งออกตามลักษณะระดับเสียงได้เป็น 2 กลุ่มใหญ่ๆ คือ

1. กลุ่มวรรณยุกต์ระดับ (Level tone) มี 3 หน่วยเสียง คือ

1.1 หน่วยเสียงวรรณยุกต์ระดับต่ำ (Low tone) แทนด้วยสัญลักษณ์ /ˍ/

คือ เสียงวรรณยุกต์เอก หน่วยเสียงนี้จะปรากฏในพยางค์ของภาษาไทยได้ทุกแบบ

1.2 หน่วยเสียงวรรณยุกต์ระดับกลาง (Mid tone) ไม่มีสัญญลักษณ์

คือ เสียงวรรณยุกต์สามัญ หน่วยเสียงนี้จะไม่ปรากฏในพยางค์ที่มีตัวสะกดเป็นพยัญชนะกัก (พยางค์คำตาย)

1.3 หน่วยเสียงวรรณยุกต์ระดับสูง (High tone) แทนด้วยสัญลักษณ์ /ˊ/

คือ เสียงวรรณยุกต์ตรี หน่วยเสียงนี้จะไม่ปรากฏในพยางค์ที่ประสมด้วยสระเสียงยาว ซึ่งมีตัวสะกดเป็นเสียงกัก

2. กลุ่มวรรณยุกต์เปลี่ยนระดับ **(Contour tone)** มี 2 หน่วยเสียง คือ

2.1 หน่วยเสียงวรรณยุกต์เปลี่ยนตก (Falling tone) แทนด้วยสัญลักษณ์ /ˆ/

คือ เสียงวรรณยุกต์โท หน่วยเสียงนี้จะไม่ปรากฏในพยางค์ที่มีสระเสียงสั้น และมีเสียงพยัญชนะสะกดเป็นพยัญชนะกัก

2.2 หน่วยเสียงวรรณยุกต์เปลี่ยนขึ้น (Rising tone) แทนด้วยสัญลักษณ์ /ˇ/

คือ เสียงวรรณยุกต์จัตวา หน่วยเสียงนี้จะไม่ปรากฏในพยางค์ที่มีเสียงพยัญชนะสะกดเป็นเสียงกักเลย

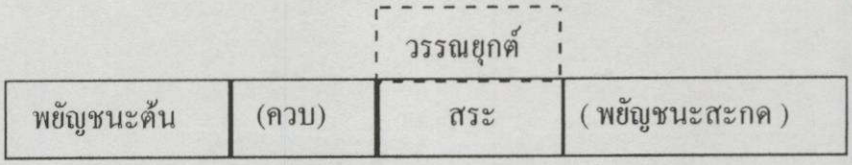
2.6 ลักษณะพยางค์ของคำไทย

2.6.1 คำจำกัดความของพยางค์และคำในภาษาไทย

ศ.ดร. กาญจนา นาคสกุล ได้ให้ความหมายของพยางค์ในระบบเสียงภาษาไทยว่า "พยางค์หมายถึง จำนวนเสียงที่ดังเด่นซึ่งปรากฏในกลุ่มเสียงที่เรียงกันเป็นคำพูด ส่วนเสียงอื่นๆที่อยู่ข้างเคียงก็จะประกอบกันเข้าเป็นส่วนของพยางค์" เสียงที่ดังเด่นในกลุ่มเสียงก็คือเสียงสระ ซึ่งมีลักษณะประจำตัวก็คือเป็นเสียงก้องซึ่งดังเด่นกว่าเสียงอื่นๆ ดังนั้นเสียงสระจึงมักเป็นเสียงที่ทำให้เกิดพยางค์ ถ้ามีเสียงสระเด่นอยู่กี่เสียง พยางค์ก็จะมีจำนวนเท่านั้นด้วย

พยางค์ที่เปล่งออกมาครั้งหนึ่งๆ อาจมีความหมายหรือไม่ก็ได้ แต่เมื่อใดพยางค์ที่ประกอบขึ้นจาก เสียงสระ พยัญชนะ และวรรณยุกต์ เป็นอย่างน้อยที่สุด และกลุ่มเสียงเหล่านี้มีความหมายและสามารถปรากฏได้โดยลำพัง พยางค์นั้นๆก็จะกลายเป็นคำในภาษา

คำในภาษาไทยส่วนใหญ่จะเป็นคำพยางค์เดียว ซึ่งเป็นคำพื้นฐาน (Base words) ของภาษา ภาษาไทยจึงจัดอยู่ในตระกูลภาษาคำโดด หรือ คำพยางค์เดียว (Monosyllabic language) หน่วยเสียงที่ประกอบกันเข้าเป็นพยางค์จะต้องมีอย่างน้อย 3 หน่วย คือ หน่วยเสียงพยัญชนะต้น 1 หน่วย หน่วยเสียงสระ 1 หน่วย และ หน่วยเสียงวรรณยุกต์ 1 หน่วย และมีหน่วยเสียงอย่างมากไม่เกิน 5 หน่วย คือเพิ่มหน่วยเสียงพยัญชนะต้นที่เป็นเสียงควบกล้ำอีก 1 หน่วย และหน่วยเสียงพยัญชนะสะกดอีก 1 หน่วย โดยมีองค์ประกอบของหน่วยเสียงต่างๆในพยางค์ แสดงได้ดังรูปที่ 2.2



รูปที่ 2.2 องค์ประกอบของพยางค์ในภาษาไทย

2.6.2 ลักษณะโครงสร้างของคำพยางค์เดียวต่อการผันของเสียงวรรณยุกต์

เราต้องตระหนักเสมอว่ารูปวรรณยุกต์ในคำภาษาไทยบางครั้งไม่แสดงเสียงให้เห็นในการเขียนเสมอไป ทั้งนี้การกำหนดเสียงวรรณยุกต์ขึ้นอยู่กับลักษณะของพยางค์ว่าเป็นคำเป็น หรือ คำตาย

ลักษณะโครงสร้างของคำพยางค์เดียวในภาษาไทยมี 5 แบบ ซึ่งลักษณะโครงสร้างที่ต่างกันของพยางค์จะมีผลต่อการผันของเสียงวรรณยุกต์ ดังแสดงในตารางที่ 2.2

ตารางที่ 2.2 แสดงลักษณะของคำพยางค์เดียวในภาษาไทย

เสียงวรรณยุกต์ / โครงสร้างพยางค์	สามัญ	เอก	โท	ตรี	จัตวา
1. พ (พ) ส ส $^{0-4}$	+	+	+	+	+
2. พ (พ) ส น $^{0-4}$	+	+	+	+	+
3. พ (พ) ส ส น $^{0-4}$	+	+	+	+	+
4. พ (พ) ส ก ^{1,3}	−	+	−	+	−
5. พ (พ) ส ส ก ^{1,2}	−	+	+	−	−

หมายเหตุ + หมายถึงโครงสร้างพยางค์สามารถผันระดับเสียงวรรณยุกต์นั้นได้

− หมายถึงโครงสร้างพยางค์ไม่สามารถผันระดับเสียงวรรณยุกต์นั้นได้

เมื่อกำหนดให้

พ แทนหน่วยเสียงพยัญชนะต้น 1 หน่วย

พพ แทนหน่วยเสียงพยัญชนะต้น 2 หน่วยควบกัน หรือพยัญชนะต้นควบ โดยหน่วยเสียงที่ 2 คือ ร / r / ล / l / หรือ ว / w /

ส แทนหน่วยเสียงสระเดี่ยวสั้น

สส แทนหน่วยเสียงสระเดี่ยวยาว และหน่วยเสียงสระประสม

น แทนหน่วยเสียงพยัญชนะสะกดที่เป็นพยัญชนะนาสิก / m, n, ŋ / และครึ่งสระ / j, w /

ก แทนหน่วยเสียงสะกดที่เป็นพยัญชนะกัก / p, t, k, ? /

0 แทนหน่วยเสียงวรรณยุกต์ สามัญ

1 แทนหน่วยเสียงวรรณยุกต์ เอก

2 แทนหน่วยเสียงวรรณยุกต์ โท

3 แทนหน่วยเสียงวรรณยุกต์ ตรี

4 แทนหน่วยเสียงวรรณยุกต์ จัตวา

และจากจำนวนอักษร 44 รูป ในภาษาไทยได้แบ่งเพื่อสะดวกต่อการผันเสียงวรรณยุกต์ เป็นอักษรไตรยางค์ ดังได้แสดงไว้ในตารางที่ 2.3

ตารางที่ 2.3 อักษรไตรยางค์

อักษรไตรยางค์		รูปวรรณยุกต์			
		เอก	โท	ตรี	จัตวา
อักษรสูง	ข ฃ ฉ ถ ฐ ผ ฝ ศ ษ ส ห	+	+	−	−
อักษรกลาง	ก จ ด ฎ ต ฏ บ ป อ	+	+	+	+
อักษรต่ำ-คู่	ค ฆ ฅ ช ฌ ฑ ท ธ ฒ พ ภ ฟ ซ ฮ	+	+	−	−
ต่ำ-เดี่ยว	ม น ง ณ ย ญ ร ล ฬ ว	+	+	−	−

อักษรสูง 11 ตัว ผันวรรณยุกต์ได้ 3 เสียง เช่น ขา ข่า ข้า

อักษรกลาง 9 ตัว ผันวรรณยุกต์ได้ครบทั้ง 5 เสียง เช่น จา จ่า จ้า จ๊า จ๋า

อักษรต่ำ 24 ตัว ผันวรรณยุกต์ได้ 3 เสียง เช่น ทา ท่า ท้า

การผันอักษรต่ำนี้มีข้อสังเกต คือถ้ามีรูปวรรณยุกต์เอกจะเป็นเสียงวรรณยุกต์โท ถ้ารูปวรรณยุกต์โทจะเป็นเสียงวรรณยุกต์ตรี นอกจากนี้อักษรต่ำยังแบ่งออกเป็นอักษรต่ำคู่ 14 ตัวและอักษรต่ำเดี่ยวอีก 10 ตัว เพื่อประโยชน์ในการผันเสียงวรรณยุกต์คือ เมื่อนำคำที่เป็นอักษรต่ำคู่มาผันร่วมกับคำที่เป็นอักษรสูงจะเกื้อกูลกันทำให้การผันเสียงวรรณยุกต์ทำได้ครบทั้ง 5 เสียง ดังตัวอย่างในตารางที่ 2.4

ตารางที่ 2.4 ตัวอย่างการผันเสียงอักษรต่ำคู่ กับอักษรสูง

เสียงวรรณยุกต์				
สามัญ	เอก	โท	ตรี	จัตวา
คา	ข่า	ค่า ข้า	ค้า	ขา

อักษรต่ำที่ใช้คู่กับอักษรสูง แล้วทำให้ผันเสียงวรรณยุกต์ได้ครบทั้ง 5 เสียง มี 7 คู่ ดังตารางที่ 2.5

ตารางที่ 2.5 การจับคู่ในการผันเสียงวรรณยุกต์

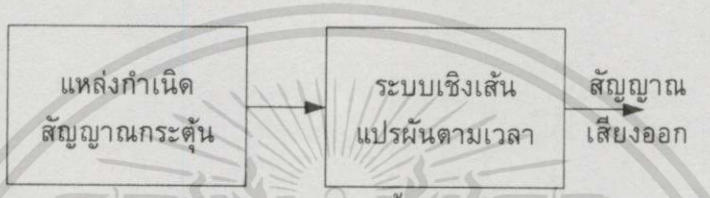
คู่ที่	อักษรสูง	อักษรต่ำ
1	ข ฃ	ค ฅ ฆ
2	ฉ	ช ฌ
3	ถ ฐ	ท ฑ ธ ฒ
4	ผ	พ ภ
5	ฝ	ฟ
6	ศ ษ ส	ซ
7	ห	ฮ
11 ตัว		14 ตัว

ส่วนอักษรต่ำเดี่ยวอีก 10 ตัวนั้นการที่จะทำให้ผันเสียงได้ครบนั้นจะนำตัว "ห" มาช่วยก็จะทำให้ผันเสียงวรรณยุกต์ได้ครบ เช่น นา หน่า น่า น้า หนา

บทที่ 3
การคำนวณสเปกตรัมแอลพีซี

3.1 แบบจำลองระบบกำเนิดเสียงพูด

เราสามารถแสดงบล็อกไดอะแกรมจำลองระบบกำเนิดเสียงเบื้องต้นดังรูปที่ 3.1 จากรูปมีการแยกภาคแหล่งกำเนิดสัญญาณกระตุ้นออกจากส่วนกำทอนเสียงซึ่งแทนด้วยระบบเชิงเส้นแปรผันตามเวลา[10]



รูปที่ 3.1 บล็อกไดอะแกรมจำลองระบบกำเนิดเสียงเบื้องต้น

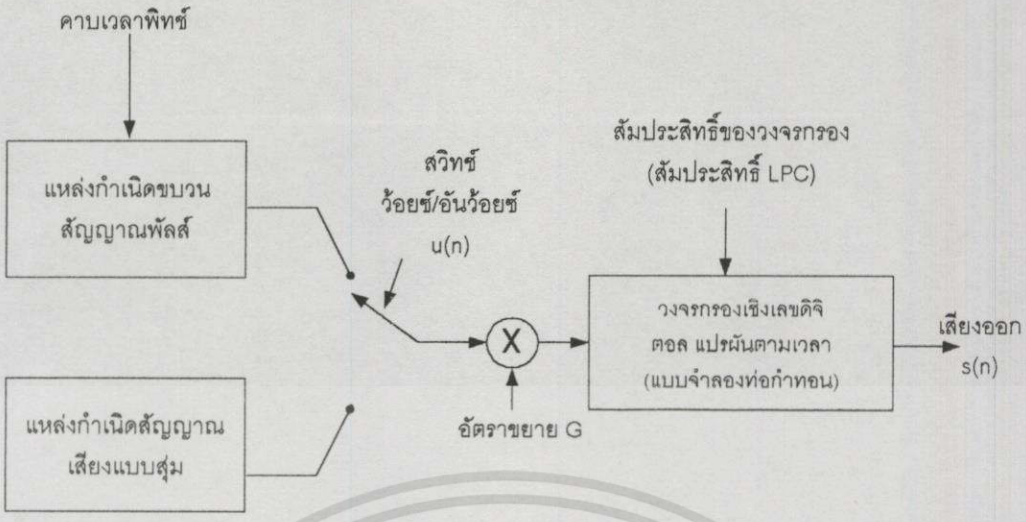
จากรูปที่ 3.1 แหล่งกำเนิดสัญญาณกระตุ้นทำหน้าที่แทนการทำงานของปอดและกล่องเสียง ส่วนนี้จะผลิตขบวนพัลส์ที่มีคาบเวลาพิทช์ขณะเปล่งเสียงว้อยซ์ และให้กำเนิดเสียงซึ่งคล้ายเสียงรบกวนขณะเปล่งเสียงอันว้อยซ์

ส่วนที่สองเป็นท่อกำทอนเสียง จะแทนการทำงานของช่องปากและโพรงจมูก ทำหน้าที่เสมือนตัวกรองสัญญาณ (Filter) ที่ยอมให้ความถี่ฟอร์แมนท์ผ่านได้ ซึ่งสามารถแทนด้วยระบบเชิงเส้นแปรผันตามเวลา (time-varying linear system)

จากแบบจำลองรูปที่ 3.1 ประกอบกับการใช้เทคนิคการประมวลสัญญาณเชิงเลข (digital signal processing) สามารถแสดงเป็นแบบจำลองระบบกำเนิดสัญญาณเสียงพูดเชิงเลขดังรูปที่ 3.2

แหล่งกำเนิดสัญญาณกระตุ้นแบ่งเป็น 2 ชนิด ตามประเภทของเสียง โดยชนิดแรกเป็นแหล่งกำเนิดเสียงว้อยซ์ ทำหน้าที่ผลิตขบวนสัญญาณพัลส์ด้วยคาบเวลาคงที่ และชนิดที่สองเป็นแหล่งกำเนิดสัญญาณเสียงรบกวนแบบสุ่ม ในกรณีที่กำเนิดเสียงอันว้อยซ์

แบบจำลองอวัยวะกำทอนเสียง เป็นวงจรกรองเชิงดิจิตอลซึ่งเป็นชนิดที่มีแต่โพลทั้งหมด (all-pole digital filter) หรือ วงจรกรองเชิงเลขแบบป้อนกลับ (recursive digital filter)



รูปที่ 3.2 แบบจำลองระบบสร้างสัญญาณเสียงพูด

การสร้างระบบเสียงพูดจำเป็นต้องวิเคราะห์หาค่าตัวแปรหรือพารามิเตอร์ที่ต้องใช้ตามแบบจำลองและระบบประมวลสัญญาณเสียง ซึ่งประกอบไปด้วย

- ดัชนีหรือพารามิเตอร์บอกชนิดของเสียงว่าเป็นเสียงว้อยซ์หรืออันว้อยซ์
- คาบเวลาพิทช์ของขบวนพัลซ์ $u(n)$ ในกรณีเสียงว้อยซ์
- อัตราการขยาย G :gain ของวงจรกรองหรือค่ารากกำลังสองเฉลี่ยของเสียง
- พารามิเตอร์ของแบบจำลองท่อกำทอนหรือสัมประสิทธิ์ของวงจรกรอง
- อัตราการสุ่มตัวอย่างสัญญาณเสียงหรืออัตราเร็วข้อมูล

3.2 หลักการของการประมาณเชิงเส้น

เนื่องจากสัญญาณเสียงมีพารามิเตอร์ที่สำคัญต่างๆดังที่กล่าวมาแล้ว การวิเคราะห์การเข้ารหัสโดยการประมาณเชิงเส้น (Linear predictive coding: LPC) จึงถูกใช้ในการหาค่าพารามิเตอร์พื้นฐานต่างๆที่กล่าวมา เพื่อใช้สำหรับเข้ารหัสสัญญาณเสียงเพื่อการรู้จำเสียง หรือการส่งสัญญาณเสียงด้วยความเร็วบิตต่ำ ด้วยเหตุที่ว่าวิธีนี้เป็นการวิเคราะห์พารามิเตอร์ต่างๆ ที่ให้ความแม่นยำสูง และสามารถย่อข้อมูลได้อย่างมีประสิทธิภาพ

หลักการพื้นฐานของการประมาณเชิงเส้น[11-12] คือ การประมาณค่าสัญญาณจากผลรวมเชิงเส้นของสัญญาณก่อนหน้านี้

สมมติว่าสัญญาณเดิมเป็น $s(n)$ การประมาณค่าสัญญาณเป็น $s'(n)$ ดังนั้นเราสามารถอธิบายการประมาณเชิงเส้นได้ด้วยสมการต่อไปนี้

$$s'(n) = \sum_{k=1}^{p} \alpha_k s(n-k) \tag{3.1}$$

เมื่อ α_k เป็นค่าคงที่ เรียกวิธีการนี้ว่าการประมาณเชิงเส้นอันดับ p โดยมีเงื่อนไขว่า ค่า α_k ที่ใช้ในการประมาณจะต้องทำให้ผลรวมของกำลังสองของความคลาดเคลื่อน $\{s(n)-s'(n)\}^2$ มีค่าน้อยที่สุด นั่นคือ $\sum e^2(n) = \sum \{s(n)-s'(n)\}^2$ มีค่าต่ำที่สุด ซึ่งการประมาณเชิงเส้นนี้มีหลายวิธี ได้แก่[13]

- วิธีโควาเรียนซ์ (Covariance Method)
- วิธีออโตคอร์รีเลชัน (Autocorrelation Method)
- วิธีแลตทิส (Lattice Method)
- วิธีวงจรกรองผกผัน (Inverse Filter Method)
- วิธีการประมาณสเปกตรัม (Spectrum Estimation Method)

และอื่นๆอีกหลายวิธี แต่ในที่นี้จะเลือกใช้วิธีออโตคอร์รีเลชัน หรือ วิธีอัตตสัมพันธ์

จากหลักการพื้นฐานของการประมาณเชิงเส้น และรูปที่ 3.2 ซึ่งเป็นแบบจำลองระบบสร้างสัญญาณเสียง เราสามารถเขียนสมการได้เป็น

$$s(n) = G \cdot u(n) + \sum_{k=1}^{p} a_k s(n-k) \tag{3.2}$$

โดยที่ a_k คือสัมประสิทธิ์ของวงจรกรองเชิงเลข

ส่วนการประมาณเชิงเส้นโดยใช้สัมประสิทธิ์ $\{\alpha_k\}$ คือ

$$s'(n) = \sum_{k=1}^{p} \alpha_k s(n-k) \tag{3.3}$$

ดังนั้นความคลาดเคลื่อนคือ

$$\begin{aligned} e(n) &= s(n) - s'(n) \\ &= s(n) - \sum_{k=1}^{p} \alpha_k s(n-k) \end{aligned} \tag{3.4}$$

ฟังชันถ่ายโอนระหว่าง $e(n)$ และ $s(n)$ คือ

$$A(z) = \frac{E(z)}{S(z)} = 1 - \sum_{k=1}^{p} \alpha_k z^{-k} \tag{3.5}$$

จากสมการที่ 3.2 ถึง 3.5 จะเห็นได้ว่าถ้า $\{\alpha_k\} = \{a_k\}$ แล้ว

$$e(n) = G \cdot u(n) \tag{3.6}$$

ในการประมวลสัญญาณจะมีการแบ่งสัญญาณออกเป็นช่วงสั้นๆ[13] (Windowing) ช่วงละประมาณ 20-40 มิลลิวินาที ซึ่งจะได้มีการอธิบายต่อไป ดังนั้นค่าผลรวมของกำลังสองของความคลาดเคลื่อน

$$\begin{aligned} E &= \sum_m e_n^2(m) \\ &= \sum_m [s_n(m) - s'_n(m)]^2 \\ &= \sum_m [s_n(m) - \sum_{k=1}^{p} \alpha_k s_n(m-k)]^2 \end{aligned} \tag{3.7}$$

โดยที่ n คือช่วงที่ n ของสัญญาณที่ใช้คำนวณ เพราะฉะนั้นเพื่อให้ได้ค่า E_n ต่ำที่สุดจะต้องมีเงื่อนไขว่า

$$\frac{\partial E_n}{\partial \alpha_i} = 0 \text{ เมื่อ } i = 1,2,\ldots,p$$

จากสมการที่ 3.7

$$\begin{aligned} \frac{\partial E_n}{\partial \alpha_i} &= -2 s_n(m-i) \sum_m [s_n(m) - \sum_{k=1}^{p} \alpha_k s_n(m-k)] \\ &= -2[\sum_m s_n(m) s_n(m-i) - \sum_{k=1}^{p} \sum_m s_n(m-k) s_n(m-i)] \end{aligned} \quad \text{เมื่อ } i = 1,2,3,\ldots,p$$

$\frac{\partial E_n}{\partial \alpha_i} = 0$ ก็ต่อเมื่อ

$$\sum_{k=1}^{p} \alpha_k \sum_m s_n(m-k) s_n(m-i) = \sum_m s_n(m) s_n(m-i) \text{ เมื่อ } i = 1,2,3,\ldots,p \tag{3.8}$$

ถ้าเรากำหนดให้ $\phi_n(i,k) = \sum s_n(m-k)s_n(m-i)$ เพราะฉะนั้น

$$\sum_{k=1}^{p} \alpha_k \phi_n(i,k) = \phi_n(i,0) \tag{3.9}$$

จากสมการที่ 3.7-3.9 จะได้ว่า

$$E_n = \sum_m s_n^2(m) - \sum_{k=1}^{p} \alpha_k \sum_m s_n(m)s_n(m-i)$$

และจาก $\phi_n(i,k) = \sum s_n(m-k)s_n(m-i)$

$$E_n = \phi_n(0,0) - \sum_{k=1}^{p} \alpha_k \phi_n(0,k) \tag{3.10}$$

สมมติว่าใน 1 เฟรมของสัญญาณที่ตัดมาคำนวณมี N ตัวอย่างคือ $s_n(0), s_n(1), s_n(2), \ldots, s_n(N-1)$ ในที่นี้เราให้ $s_n(m) = 0$ เมื่อ $m < 0$ หรือ $m > N-1$ เพราะฉะนั้น

$$\begin{aligned}\phi_n(i,k) &= \sum_m s_n(m-k)s_n(m-i) \\ &= \sum_{m=0}^{N-1-(i-k)} s_n(m)s_n(m+i-k) \qquad 0 \le k \le p, k \le i \le p\end{aligned}$$

ให้

$$R_n(k) = \sum_{m=0}^{N-i-k} s_n(m)s_n(m+k) \tag{3.11}$$

$$\therefore \phi_n(i,k) = R_n(|i-k|) \text{ เมื่อ } i = 1,2,\ldots,p; k = 0,1,2,\ldots,p$$

จากสมการที่ 3.9 จะได้ว่า

$$\sum_{k=1}^{p} \alpha_k R_n(|i-k|) = R_n(i) \text{ เมื่อ } i = 1,2,\ldots,p \tag{3.12}$$

และจากสมการที่ 3.10

$$E_n = R_n(0) - \sum_{k=1}^{p} \alpha_k R_n(k) \tag{3.13}$$

จากสมการที่ 3.12 เขียนให้อยู่ในรูปเมตริกซ์ได้เป็น

$$\begin{bmatrix} R_n(0) & R_n(1) & \cdots & R_n(p-1) \\ R_n(1) & R_n(0) & \cdots & R_n(p-2) \\ \vdots & \vdots & \ddots & \vdots \\ R_n(p-1) & R_n(p-2) & \cdots & R_n(0) \end{bmatrix} \cdot \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_p \end{bmatrix} = \begin{bmatrix} R_n(1) \\ R_n(2) \\ \vdots \\ R_n(p) \end{bmatrix} \tag{3.14}$$

หรือ

$$R_n \cdot \alpha = r_n \tag{3.15}$$

เมื่อ $R_n = \begin{bmatrix} R_n(0) & R_n(1) & \cdots & R_n(p-1) \\ R_n(1) & R_n(0) & \cdots & R_n(p-2) \\ \vdots & \vdots & \ddots & \vdots \\ R_n(p-1) & R_n(p-2) & \cdots & R_n(0) \end{bmatrix}, \alpha = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_p \end{bmatrix}$ และ $r_n = \begin{bmatrix} R_n(1) \\ R_n(2) \\ \vdots \\ R_n(p) \end{bmatrix}$

จากสมการที่ 3.2 และ 3.4

$$G \cdot u(n) = s(n) - \sum_{k=1}^{p} a_k s(n-k)$$

$$e(n) = s(n) - \sum_{k=1}^{p} \alpha_k s(n-k)$$

เมื่อ $a_k = \alpha_k$ จะได้ว่า

$$e(n) = G \cdot u(n)$$

$$\therefore E_n = \sum_{m=0}^{N-1} e^2(m) = G \sum_{m=0}^{N-1} u^2(m) \tag{3.16}$$

$$= R_n(0) - \sum_{k=1}^{p} \alpha_k R_n(k)$$

จากสมการที่ 3.15 เราสามารถหา α โดยตรงดังนี้

$$\alpha = R_n^{-1} \cdot r_n \tag{3.17}$$

และจากสมการที่ 3.16 เราสามารถหาค่า G โดยตรงจาก

$$G = \frac{R_n(0) - \sum_{k=1}^{p} \alpha_k R_n(k)}{\sum_{m=0}^{N-1} u^2(m)} \qquad (3.18)$$

3.3 การคำนวณค่าสัมประสิทธิ์ของ LPC และอัตราการขยาย

ในการหาค่าสัมประสิทธิ์ LPC (α) และอัตราการขยาย (G) [13] เราจึงต้องมีการเตรียมข้อมูลในการวิเคราะห์ก่อน ซึ่งขั้นตอนในการเตรียมสัญญาณเข้าเป็นดังรูปที่ 3.3

รูปที่ 3.3 แสดงขั้นตอนการเตรียมสัญญาณในการวิเคราะห์

การวิเคราะห์สามารถแบ่งเป็นขั้นตอนต่างๆดังนี้[13]

- การพรีเอมฟาซิส (preemphasis)
- การแบ่งช่วงสัญญาณ (frame blocking)
- การวินโดว์ (windowing)
- การหาออโตคอร์รีเลชัน (autocorrelation analysis)
- การหาสัมประสิทธิ์ α และอัตราการขยาย G
-

3.3.1 การพรีเอมฟาซิส

เนื่องจากสัญญาณเสียงพูดของมนุษย์จะมีองประกอบส่วนใหญ่อยู่บริเวณที่ความถี่ต่ำ เมื่อเทียบกับแถบความถี่ปฏิบัติงานไม่เกิน 5.5 กิโลเฮิรตซ์ ดังนั้นเพื่อให้อัตราส่วนสัญญาณเสียงต่อ

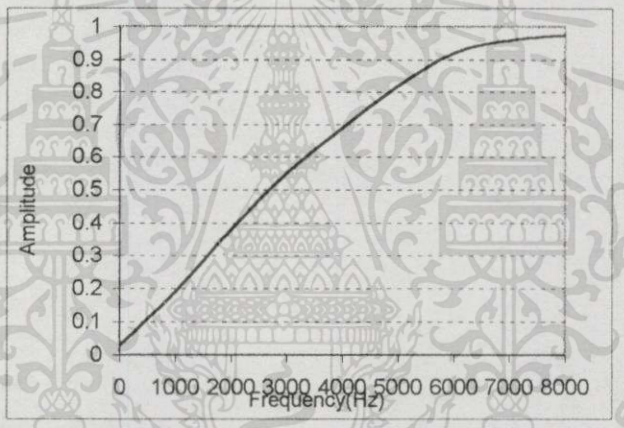
สัญญาณรบกวน (signal to noise ratio: SNR) มีค่าค่อนข้างคงที่ตลอดช่วงความถี่ปฏิบัติงานนี้ เราจึงต้องมีการพรีเอมฟาซีส โดยเน้นความถี่สูงให้มีขนาดสูงขึ้น นั่นคือ การพรีเอมฟาซีสก็คือการกรองสัญญาณด้วยวงจรความถี่สูงผ่าน (high pass filter) ซึ่งมักนิยมใช้วงจรกรองอันดับหนึ่ง มีฟังชันการถ่ายโอนเป็น

$$H(z) = 1 - a \cdot z^{-1} \text{ โดยที่ } 0.9 < a < 1$$

เมื่อเปรียบเทียบกันรูปที่ 3.3 เราจะได้ว่า

$$s'(n) = s(n) - a \cdot s(n-1)$$

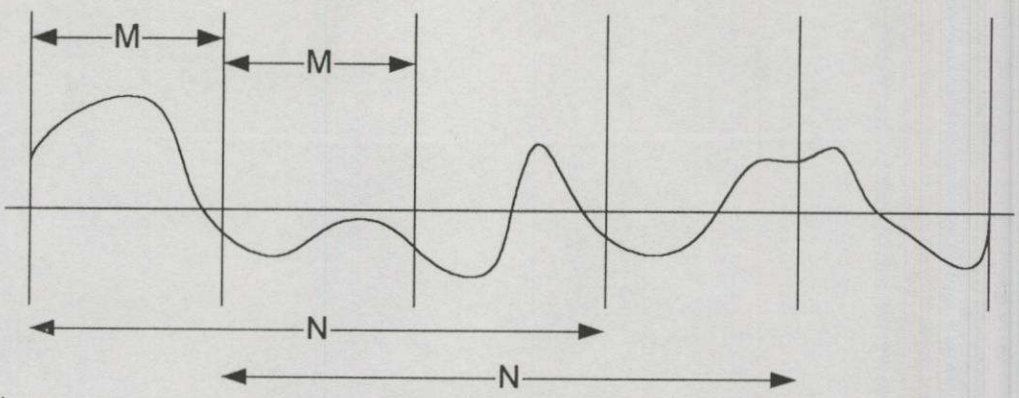
ยิ่งค่า a ใกล้ 1 เท่าใดความถี่สูงก็จะยิ่งถูกขยายมากขึ้นเท่านั้น ค่า a ที่นิยมสำหรับใช้ในการหาพารามิเตอร์ของ LPC คือ 15/16=0.9375 [13] เมื่อนำฟังชันถ่ายโอนมาพล็อตกราฟของขนาดเทียบกับความถี่จะได้ดังรูปที่ 3.4



รูปที่ 3.4 แสดงขนาดสเปกตรัมของฟังชันถ่ายโอนของการพรีเอมฟาซีส

3.3.2 การแบ่งช่วงสัญญาณ

สัญญาณที่ผ่านการพรีเอมฟาซีสแล้วจะถูกตัดแบ่งออกเป็นช่วงๆ หรือเฟรม ช่วงละ N ตัวอย่างสัญญาณ การวิเคราะห์จะวิเคราะห์ทีละช่วงของแต่ละ N ตัวอย่างสัญญาณ ดังรูปที่ 3.5



รูปที่ 3.5 แสดงการแบ่งช่วงของสัญญาณที่ใช้ในการวิเคราะห์

โดยช่วงในการวิเคราะห์แต่ละช่วงจะถูกเลื่อนไปเป็นระยะ M ช่วงสัญญาณ จะเห็นได้ว่า ถ้าค่า M โตกว่าค่า N ในการเลื่อนของช่วงในการวิเคราะห์จะทำให้บางสัญญาณไม่ถูกใช้ในการวิเคราะห์ ก็จะเป็นการสูญเสียส่วนหนึ่งที่ทำให้ผลที่ได้ไม่ถูกต้องเท่าที่ควร ถ้าค่า M เล็กกว่า N ก็จะทำให้ตัวอย่างสัญญาณทุกตัวถูกนำมาวิเคราะห์ ยิ่งค่า M เล็กเท่าใดความแม่นยำในการวิเคราะห์ก็จะยิ่งสูงขึ้นเท่านั้น แต่ก็จะทำให้การคำนวณช้าลง ดังนั้นจึงมีหลักการในการกำหนดขนาดของช่วงในการวิเคราะห์คือ

- ช่วงในการวิเคราะห์จะต้องสั้นพอที่คุณสมบัติของเสียงที่เราสนใจจะวิเคราะห์นั้น ยังไม่เปลี่ยนแปลงในวินโดว์
- ช่วงในการวิเคราะห์จะต้องยาวพอที่จำนวนตัวอย่างสัญญาณในวินโดว์ สามารถนำมาคำนวณหาคุณสมบัติที่ต้องการได้
- ช่วงในการวิเคราะห์ที่ติดกัน ไม่ควรจะสั้นจนกระโดดข้ามข้อมูลบางส่วนไป แต่ควรเลื่อนวินโดว์ให้น้อยกว่าขนาดของเฟรม

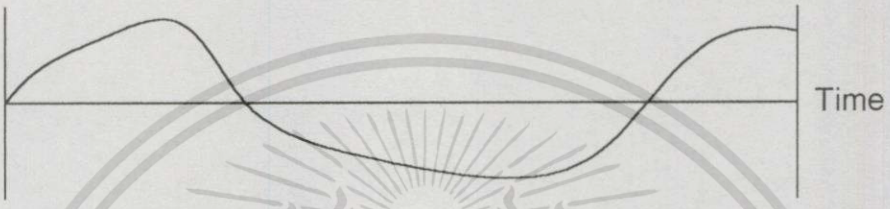
เนื่องจากเราใช้ความถี่ในการสุ่มสัญญาณ 11.025 กิโลเฮิรตซ์ ในการวิเคราะห์นี้ เราเลือกใช้ค่า N=300 และค่า M=100 นั่นคือช่วงในการวิเคราะห์คือ 27.21 มิลลิวินาที และระยะในการเลื่อนเฟรมประมาณ 9 มิลลิวินาที

3.3.3 การวินโดว์

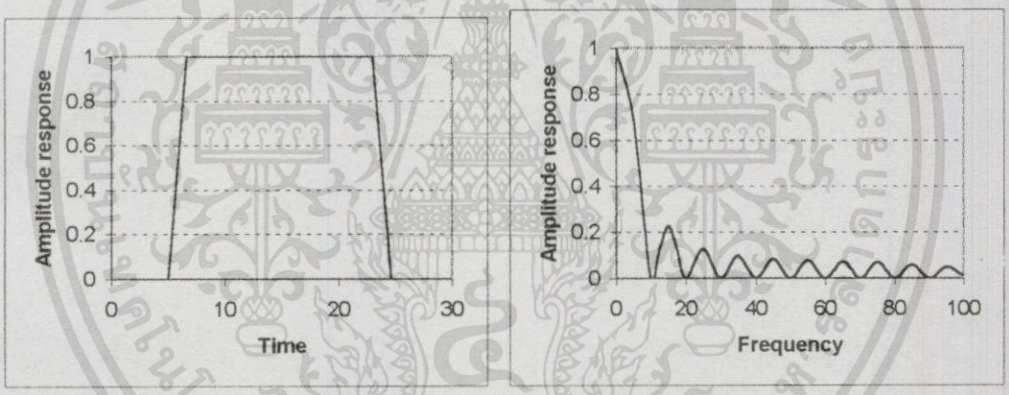
พิจารณาช่วงสัญญาณ N ตัวอย่างสัญญาณของช่วงใดๆที่ตัดมาวิเคราะห์(รูปที่3.6) จะเห็นว่าที่ขอบของเฟรมที่ตัดมานี้มีความไม่ต่อเนื่องของสัญญาณ ถ้ามองในโดเมนความถี่ ก็จะมีความถี่สูงเกิดขึ้น ดังนั้นเพื่อที่จะลดองค์ประกอบของความถี่สูงเหล่านี้ เราจะคูณสัญญาณด้วยฟังชันวินโดว์เพื่อลดความไม่ต่อเนื่องของสัญญาณที่ขอบและไม่ทำให้สเปกตรัมของสัญญาณในช่วงความถี่ต่ำเปลี่ยนไปมากนักในที่นี้จะใช้ฟังชันวินโดว์แฮมมิ่ง[10][12][13] (Hamming window function) ซึ่งนิยามโดยสมการดังนี้คือ

$$w(n) = 0.54 - 0.46\cos(2\pi n/(N-1)) \text{ เมื่อ } n = 0,1,2,...,N-1 \qquad (3.19)$$

รูปที่ 3.7 และ 3.8 แสดงองค์ประกอบทางเวลาและทางความถี่ของฟังชันวินโดว์แบบสี่เหลี่ยม และแฮมมิ่ง ตามลำดับ จะเห็นว่าสเปกตรัมของวินโดว์แฮมมิ่งมีริพเพิ่ล (ripple) น้อยกว่าของวินโดว์สี่เหลี่ยม

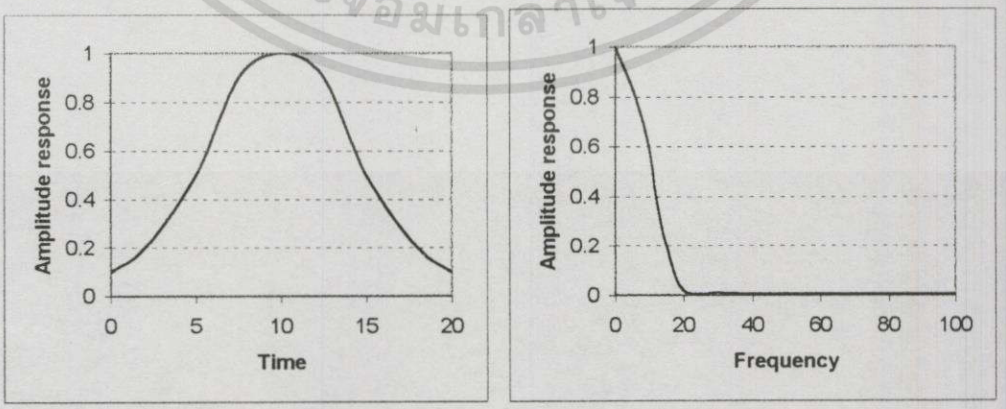


รูปที่ 3.6 ส่วนของสัญญาณที่ตัดมาวิเคราะห์



(ก) (ข)

รูปที่ 3.7 แสดงวินโดว์แบบสี่เหลี่ยม (ก)ในโดเมนเวลา (ข)ในโดเมนความถี่



(ก) (ข)

รูปที่ 3.8 วินโดว์แบบแฮมมิ่ง (ก)ในโดเมนเวลา (ข)ในโดเมนความถี่

ดังนั้นจากรูปที่ 3.3 เราจะได้ว่า

$$x'(n) = x(n) \cdot w(n) \tag{3.20}$$

3.3.4 การคำนวณออโตคอร์รีเลชัน

จากสมการที่ 3.11 และ 3.20 จะได้ว่า

$$R_n(k) = \sum_{m=0}^{N-1-k} x'(m)x'(m+k) \tag{3.21}$$

3.3.5 การหาสัมประสิทธิ์ α และอัตราการขยาย G

เมื่อได้ค่า $R_n(0), R_n(1), R_n(2), \ldots, R_n(p)$ แล้วก็สามารถหาค่า α และค่า G ได้จากสมการ

$$\alpha = R_n^{-1} \cdot r_n \tag{3.22}$$

$$G = \frac{R_n(0) - \sum_{k=1}^{p} \alpha_k R_n(k)}{\sum_{m=0}^{N-1} u^2(m)} \tag{3.23}$$

3.3.6 การคำนวณสเปกตรัมแอลพีซี

จากบล็อกไดอะแกรมในรูปที่ 3.2 แสดงให้เห็นว่าสเปกตรัมแอลพีซีก็คือ ฟังก์ชันถ่ายโอนของอวัยวะกำทอนเสียงดังนั้น จากสมการที่ 3.2 ทำการแปลงให้อยู่ใน Z โดเมนได้เป็น

$$S(z) = \sum_{k=1}^{p} a_k z^{-1} S(z) + GU(z)$$

ซึ่งจัดรูปให้เป็นฟังก์ชันถ่ายโอนของอวัยวะกำทอนเสียงทำให้สามารถคำนวณ สเปกตรัมแอลพีซีได้ดังนี้

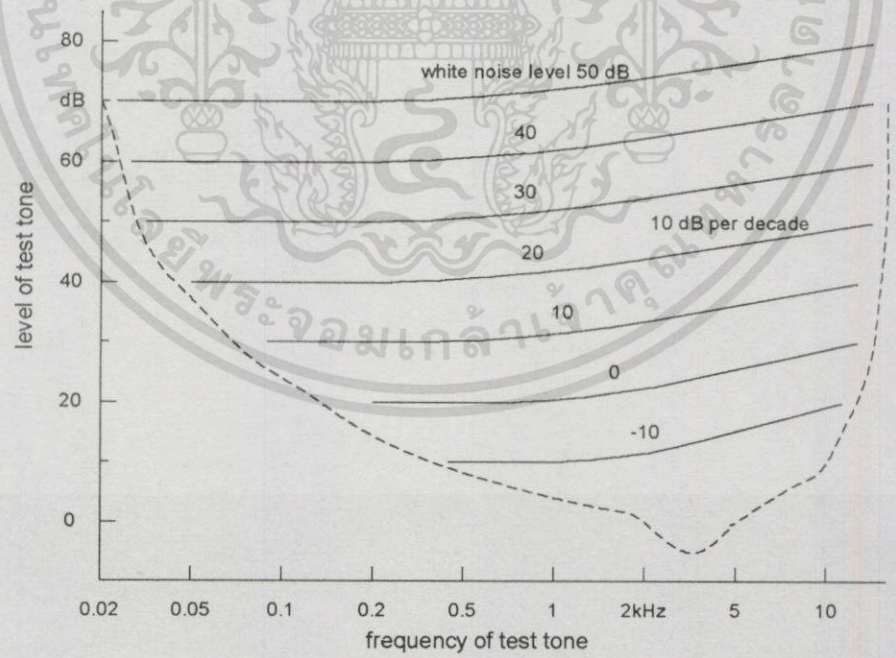
$$H(z) = \frac{S(z)}{GU(z)} = \frac{1}{1 - \sum_{k=1}^{p} a_k z^{-k}}$$

บทที่ 4
แถบความถี่วิกฤต และความเข้มสัญญาณในแถบความถี่วิกฤต

4.1 กล่าวนำ

ในบทนี้จะกล่าวถึงคำจำกัดความของแถบความถี่วิกฤต(Critical Band), วิธีในการหาความกว้างของแถบความถี่วิกฤต(Critical Bandwidth) และรวมถึงการกำหนดสเกลบาร์กที่ใช้แถบความถี่วิกฤตเป็นเกณฑ์ (Critical Band Rate Scale หรือ Bark scale)

คำจำกัดความของแถบความถี่วิกฤตนั้นได้ถูกกล่าวขึ้นโดย Fletcher [14] โดยที่เขาได้ตั้งสมมติฐานไว้ว่าในกรณีที่มีสัญญาณรบกวน(Noise) ทับซ้อนจนสามารถบดบัง(Mask) สัญญาณโทนทดสอบ(Test Tone) ได้นั้น ส่วนของสัญญาณรบกวนที่มีผลในการบดบังสัญญาณโทนทดสอบจริงๆจะเป็นเพียงส่วนที่มีความถี่ใกล้เคียงกับสัญญาณโทนทดสอบเท่านั้น ในการที่จะทำการหาค่าสัมประสิทธิ์ต่างๆของแถบความถี่วิกฤตได้นั้นยังต้องอาศัยสมมติฐานที่ว่า การบดบังสัญญาณจะเป็นผลก็ต่อเมื่อกำลังงานของสัญญาณรบกวนในแถบสัญญาณเดียวกับสัญญาณโทนทดสอบนั้นเท่ากับกำลังงานของสัญญาณโทนทดสอบ และถือว่ากำลังงานของสัญญาณรบกวนที่อยู่นอกเหนือจากแถบสัญญาณที่ใกล้กับสัญญาณโทนทดสอบไม่มีผลต่อการบดบังสัญญาณ



รูปที่ 4.1 ระดับของสัญญาณโทนทดสอบที่ถูกบดบังโดยสัญญาณ White Noise ที่ระดับสัญญาณต่างๆ และเส้นประเป็นระดับ Threshold in quiet

รูปที่ 4.1 เป็นรูปที่แสดงการบดบังสัญญาณโทนทดสอบโดยใช้สัญญาณรบกวนแถบความถี่กว้าง(Broadband Noise) ซึ่งในที่นี้ใช้ White Noise ที่ระดับความเข้มสัญญาณต่างๆ โดยเส้นประเป็นระดับ Threshold in Quiet ซึ่งหมายถึงระดับของสัญญาณโทนที่หูของมนุษย์เริ่มที่จะได้ยินเมื่อไม่มีสัญญาณรบกวนใดๆ เช่นหากมีสัญญาณโทนที่ความถี่ 2 kHz โดยไม่มีสัญญาณรบกวนใดๆ หูเราสามารถได้ยินสัญญาณโทนนั้นได้ก็ต่อเมื่อ สัญญาณมีระดับความเข้มประมาณ 0 dB ($10^{-12} w/m^2$) ขึ้นไป ส่วนเส้นทึบเป็นเส้นที่แสดงระดับ Masking Threshold สำหรับความเข้มสัญญาณ White Noise ที่ระดับต่างๆซึ่ง Threshold นี้บ่งบอกว่าหากสัญญาณโทนมีระดับความเข้มน้อยกว่า Threshold นี้ เราจะไม่สามารถได้ยินสัญญาณโทนนี้ได้ เช่น สัญญาณโทนที่ 2 kHz ขณะที่มีสัญญาณ White Noise ความเข้มสัญญาณ 20 dB เราจะสามารถได้ยินโทนนี้ก็ต่อเมื่อโทนนี้มีระดับความเข้มสัญญาณมากกว่าประมาณ 40 dB ซึ่งจะเห็นได้ว่าในการบดบังสัญญาณโดย White Noise นั้นจะมีระดับ Threshold ที่ไม่เป็นอิสระกับความถี่ทั้งๆที่ตัว White Noise เองนั้นความเข้มสัญญาณของมันไม่ขึ้นกับความถี่ แต่จะเห็นได้ว่าระดับการบดบังสัญญาณจะมีลักษณะที่ไม่ขึ้นกับความถี่ในช่วงที่สัญญาณมีความถี่ต่ำกว่า 500 Hz ส่วนที่ความถี่ประมาณ 1 kHz ขึ้นไปนั้นจะมีอัตราการเปลี่ยนแปลงอยู่ที่ประมาณ 10 dB/decade

ระบบการรับฟังเสียงของคนเรามีระดับของการถูกบดบังสัญญาณที่ไม่ขึ้นกับความถี่ในช่วงที่ต่ำกว่า 500 Hz ในช่วงความถี่นี้แถบความถี่วิกฤตจะมีความกว้างเท่าๆกันด้วย สำหรับความถี่ที่สูงกว่านี้นั้นจะมีระดับการบดบังสัญญาณเพิ่มขึ้นประมาณ 10 dB/decade นั่นก็หมายความว่าระดับความเข้มของสัญญาณนั้นเพิ่มขึ้นตามความถี่ ดังนั้นความกว้างแถบความถี่ของช่วงสัญญาณนี้เพิ่มขึ้น 10 dB/decade ด้วยเช่นกัน

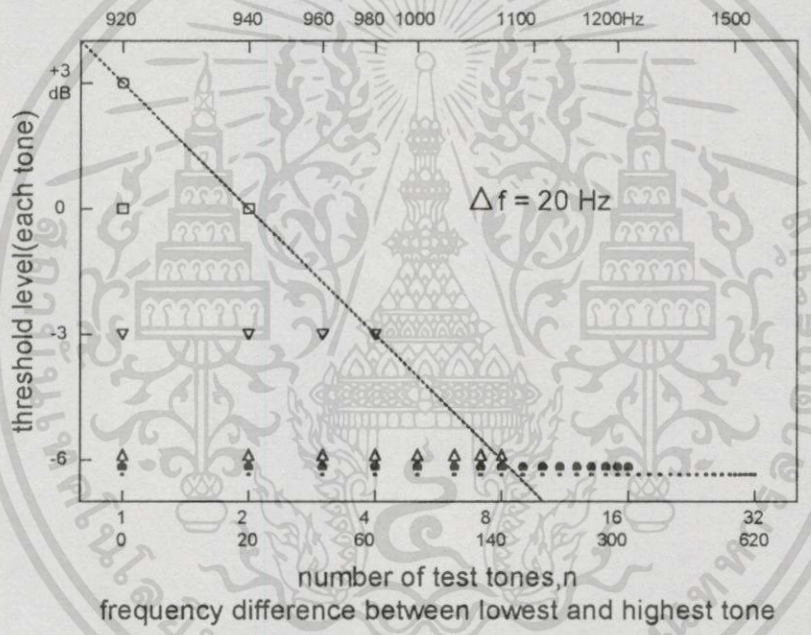
หากเราเชื่อว่าสมมติฐานของ Fletcher นั้นถูกต้องแสดงว่าเราต้องสามารถได้ยินเสียงสัญญาณโทนในขณะที่สัญญาณรบกวนที่มีแถบสัญญาณใกล้เคียงกับแถบความถี่วิกฤตที่มีความถี่ของสัญญาณโทนเป็นศูนย์กลางนั้นมีกำลังเท่ากับสัญญาณโทน ซึ่งจะทำให้เราสามารถที่จะคาดคะเนความกว้างของแถบความถี่วิกฤตได้โดย ที่ความถี่ต่ำกว่า 500 Hz นั้นระดับความเข้มของโทนที่ถูกบดบังสัญญาณจะสูงกว่าระดับความหนาแน่นของ White Noise ที่บดบังสัญญาณโทนนั้นอยู่ประมาณ 17 dB ซึ่งจากสมมติฐานนี้ เราสามารถหาความกว้างแถบความถี่วิกฤตได้เป็น $10^{17/10}$ หรือประมาณ 50 Hz

แต่ในความเป็นจริงแล้ว เราสามารถได้ยินเสียงโทนในขณะที่กำลังของเสียงโทนเป็นเพียงครึ่งหนึ่งหรือเพียงหนึ่งในสี่ของกำลังของสัญญาณรบกวนในแถบความถี่ที่ใกล้กับความถี่ของเสียงโทนนั้นเท่านั้น[14]-[15] และการศึกษาโดยใช้ข้อมูลอื่นๆเพิ่มเติม เราสามารถคาดคะเนความกว้างของแถบความถี่วิกฤตได้ใกล้เคียงยิ่งขึ้น ซึ่งในย่านความถี่ต่ำกว่า 500 Hz นั้นความกว้างของแถบ

ความถี่วิกฤตจะมีค่าประมาณ 100 Hz ส่วนที่ความถี่สูงกว่า 500 Hz นั้นจะมีความกว้างของแถบความถี่วิกฤตประมาณ 20 เปอร์เซ็น ของความถี่ศูนย์กลางนั้นๆ

4.2 วิธีการในการหาความกว้างแถบความถี่วิกฤต

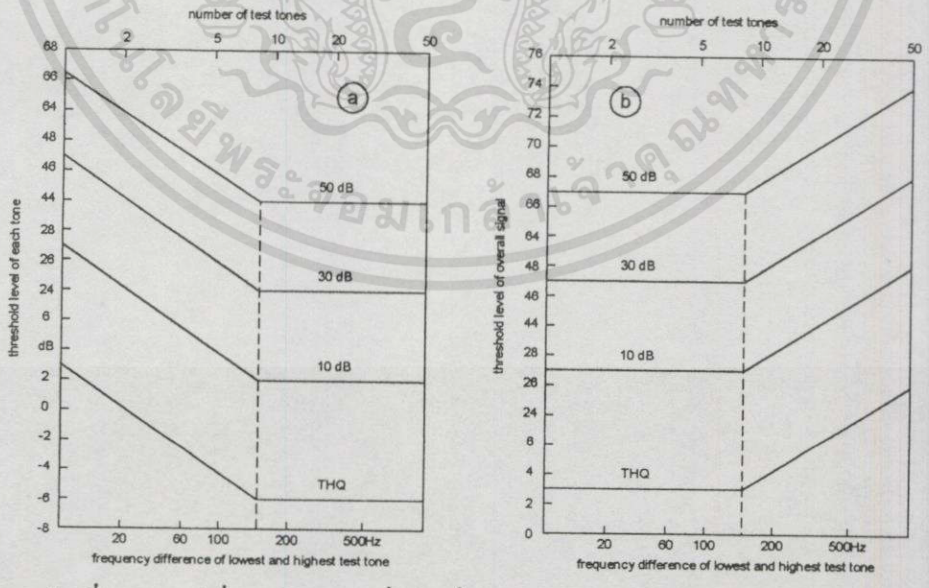
วิธีการแรกที่ใช้ในการหาความกว้างของแถบความถี่วิกฤตคือ การวัดจากระดับ Threshold โดยตรงซึ่งมีลักษณะเช่นเดียวกับวิธีการอื่นๆที่ใช้การวัดระดับสัญญาณโดยตรงคือ ความกว้างแถบความถี่ หรือระดับสัญญาณที่มีความเกี่ยวข้องกับแถบกว้างความถี่ จะต้องเป็นตัวแปรซึ่งในกรณีนี้ Threshold in Quiet ของสัญญาณโทนความถี่ต่างๆซึ่งมีแอมปลิจูดเท่ากันถูกใช้ในการประมาณความกว้างแถบความถี่วิกฤตที่อยู่ในช่วงประมาณ 1KHz



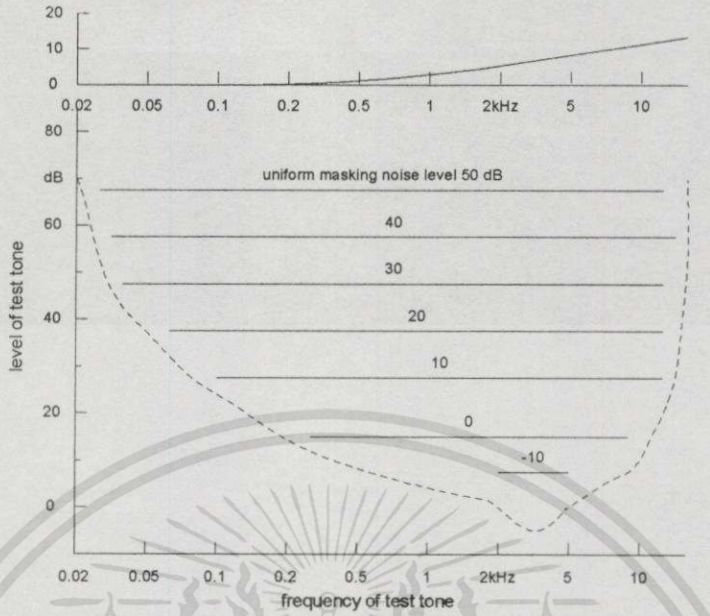
รูปที่ 4.2 ความสัมพันธ์ระหว่างระดับ Threshold in Quiet และจำนวนสัญญาณโทนทดสอบโดยจุดที่มีลูกศรอยู่เป็นจุดที่ใช้ในการคาดคะเนความกว้างแถบความถี่วิกฤต

รูปที่ 4.2 เป็นระดับ Threshold in Quiet ของแต่ละโทนเมื่อเปลี่ยนจำนวนของสัญญาณโทนความถี่ต่างๆและความแตกต่างระหว่างความถี่สูงสุดและความถี่ต่ำสุดของโทน จากรูปเป็นการวัดระดับของ Threshold in Quiet โดยอาศัยวิธีการ Tracking สัญญาณโดยเริ่มต้นที่สัญญาณโทนที่ความถี่ 920 Hz เพียงสัญญาณเดียวจะได้ Threshold in Quiet ที่ +3dB เมื่อเพิ่มสัญญาณโทนอีกค่าก็คือ 940 Hz แล้วทำการ Tracking เพื่อวัดระดับ Threshold in Quiet ของสัญญาณที่มีโทนทั้งสองรวมอยู่ก็จะได้ระดับ Threshold in Quiet ของแต่ละสัญญาณที่ 0 dB และเมื่อทำตามลักษณะนี้ไปเรื่อยๆ โดยการเพิ่มจำนวนสัญญาณโทนที่ละสองเท่าแล้วทำการ Tracking เพื่อวัดระดับ Threshold in

Quiet จะได้ –3dB และ –6dB สำหรับจำนวนโทน 4 และ 8 โทนตามลำดับซึ่งแสดงให้เห็นว่าระดับ Threshold in Quiet ของแต่ละโทนจะลดลงเมื่อมีการรวมกันของสัญญาณโทนมากขึ้น แต่เมื่อเพิ่มจำนวนโทนจนไปถึงจุดหนึ่ง ระดับ Threshold in Quiet ของแต่ละโทนก็จะไม่ลดลงอีกดังที่สังเกตุได้ชัดในรูปที่ 4.2 ที่จำนวนโทน 16 และ 32 โทนก็จะไม่มีความแตกต่างกันของระดับ Threshold in Quiet ของแต่ละโทน กับที่ระดับ Threshold in Quiet ของการใช้จำนวนโทนเท่ากับ 8 มากนัก ซึ่งค่านี้เองเป็นค่าที่เราใช้ในการวัดความกว้างแถบความถี่วิกฤต เป็นที่น่าสังเกตุว่าช่วงของจำนวนโทนตั้งแต่ 1 ถึง 8 นั้นจะมีระดับ Threshold in Quiet ลดลง 3 dB ทุกๆการเพิ่มขึ้นสองเท่าของจำนวนโทน หมายความว่าที่ระดับ Threshold in Quiet ค่าใดค่าหนึ่งนั้น ระดับแรงดันเสียงรวมทุกโทนค่อนข้างคงที่ไม่ขึ้นกับจำนวนโทน แต่กฎนี้ใช้ได้กับเฉพาะจำนวนโทนที่ไม่เกิน 8 เท่านั้นถ้ามากกว่านี้ระดับ Threshold in Quiet ของแต่ละโทนไม่ได้ลดลงตามการเพิ่มขึ้นของจำนวนโทนจึงทำให้ ระดับแรงดันเสียงโดยรวมเพิ่มขึ้น ซึ่งหมายความว่าระดับ Threshold In Quiet ของระดับการได้ยินเสียงของเราสามารถประมาณได้โดยความเข้มของเสียงโทนทั้งหมด ตราบใดที่โทนต่างๆอยู่ในแถบความถี่วิกฤต ส่วนที่อยู่นอกเหนือแถบกว้างความถี่วิกฤตนี้จะไม่นำมามีผลกับค่า Threshold In Quiet หรืออีกนัยหนึ่งคือ สัญญาณโทนความถี่ 920 Hz ได้ก็ต่อเมื่อมันมีความเข้ม +3 dB ขึ้นไป หรือสัญญาโดๆที่มีความถี่อยู่ในแถบความถี่วิกฤตที่มีความถี่ศูนย์กลางที่ 920 Hz แล้วมีความเข้มสัญญาณรวมกัน 3 dB ขึ้นไป การคำนวณความกว้างแถบความถี่วิกฤตโดยวิธีนี้สามารถคำนวณได้จากจำนวนของโทน และระยะห่างระหว่างโทนซึ่งอยู่ในรูปจะได้ค่า 140 Hz ซึ่งมาจาก (8-1)*20=140 Hz เป็นความกว้างแถบความถี่วิกฤตสำหรับความถี่ศูนย์กลางที่ 920 Hz



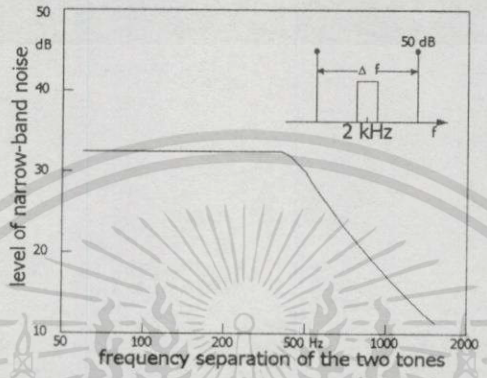
รูปที่ 4.3 ผลต่อเนื่องจากรูปที่ 4.2 โดยการเพิ่มผลที่ได้จากการใช้ Uniform Masking Noise ด้านซ้ายเป็นระดับ Threshold ของแต่ละโทนส่วนด้านขวาเป็นของสัญญาณรวมทุกโทน



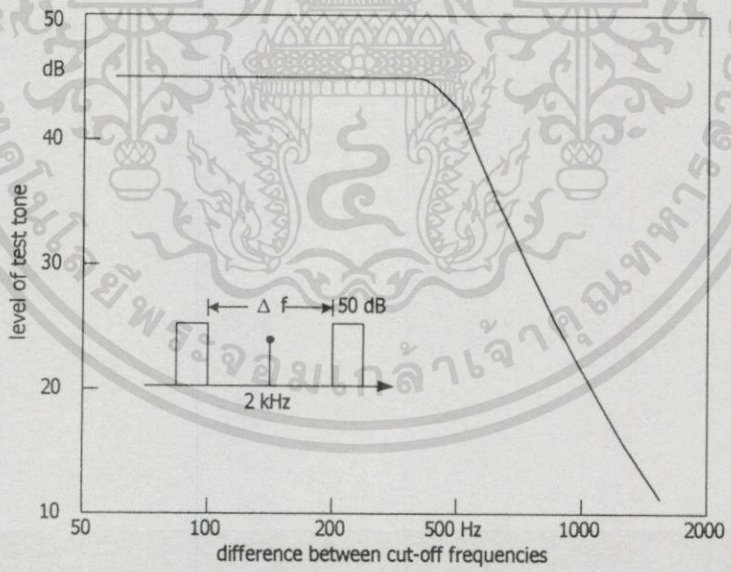
รูปที่ 4.4 ระดับของสัญญาณโทนทดสอบที่ถูกบดบังโดย Uniform Masking Noise ที่ระดับสัญญาณต่างๆ โดยกราฟด้านบนเป็นค่าการลดทอนที่ให้กับ White Noise เพื่อสร้าง Uniform Masking Noise

การหาค่าความกว้างแถบความถี่วิกฤตโดยใช้ Threshold In Quiet นี้จะให้คำตอบที่ถูกต้องก็ต่อเมื่ออยู่ในย่านความถี่ซึ่งระดับ Threshold In Quiet ไม่ขึ้นกับความถี่ ซึ่งช่วงความถี่นั้นอยู่ในช่วงที่ไม่เกิน 500 Hz ถึง 2KHz เท่านั้นอย่างไรก็ตามหากดูในรูปที่4.4 จะเห็นได้ว่า Uniform Masking Noise จะสามารถบดบังสัญญาณซึ่งจะไม่ขึ้นกับความถี่ของโทน ถ้าหากว่าปรากฏการณ์ที่เกิดขึ้นซึ่งอธิบายในรูปที่ 4.2 นั้นเกิดขึ้นไม่ใช่เพียงแค่ที่ Threshold In Quiet เท่านั้น แต่ยังเกิดขึ้นกับระดับของการบดบังสัญญาณโดย Uniform Masking Noise แล้วหมายความว่ามันเป็นไปได้ที่จะเกิดปรากฏการณ์นี้ตลอดช่วงความถี่ของการได้ยินของมนุษย์ได้ด้วย ผลที่ได้จากการวัดจะใกล้เคียงกับรูปที่ 4.2 สำหรับความถี่ของสัญญาณโทนที่ใกล้เคียงกับ 920 Hz ส่วนการใช้ Uniform Masking Noise ที่ระดับต่างๆแสดงในรูปที่ 4.3 จำนวนสัญญาณโทนซึ่งมีระดับสัญญาณเท่ากันจะเป็นค่าในแกนด้านบน ส่วนความกว้างแถบสัญญาณที่ได้จากการเปลี่ยนแปลงจำนวนโทนแสดงอยู่ในแกนด้านล่าง ผลแสดงให้เห็นชัดว่าวิธีที่กล่าวผ่านมาทั้งสองวิธีนั้น ระดับ Threshold ของแต่ละโทนจะลดลงในช่วงที่ความแตกต่างของความถี่มีค่าน้อยกว่าค่าๆหนึ่ง ส่วนที่ความแตกต่างของความถี่มีค่าสูงกว่าค่านั้นระดับสัญญาณของแต่ละโทนจะคงที่ ในรูปที่ 4.3a นั้นจะแสดงระดับ Threshold ของแต่ละโทนในขณะที่รูปที่ 4.3b จะแสดงเปรียบเทียบกับระดับแรงดันเสียงรวมทุกโทนแทนที่จะแยกออกสำหรับแต่ละโทน และจะเห็นได้ว่าระดับแรงดันเสียงรวมนั้นจะคงที่ที่ระดับ Threshold in Quiet หรือ Masking Threshold ในบริเวณที่ความแตกต่างของความถี่น้อยกว่าความกว้างแถบความถี่วิกฤต ระดับ

แรงดันของเสียงรวมจะเพิ่มขึ้นเมื่อความแตกต่างของความถี่มากกว่าความกว้างแถบความถี่วิกฤต ซึ่งแสดงให้เห็นว่าองค์ประกอบของเสียงส่วนที่เกินจากแถบความถี่วิกฤตนั้นจะไม่ถูกนำมามีผลในการวัด Threshold in Quiet หรือระดับการบดบังสัญญาณโดย Uniform Masking Noise จึงสามารถกล่าวได้ว่าสิ่งที่มีผลต่อ Threshold in Quiet และระดับของการบดบังสัญญาณคือ ความเข้มของสัญญาณในแถบความถี่วิกฤตนั้นๆเท่านั้น



รูปที่ 4.5 ระดับ Threshold ของสัญญาณรบกวนแถบความถี่แคบที่มีความถี่ศูนย์กลางอยู่ระหว่างสองโทนที่ใช้ในการบดบังสัญญาณ โดยแสดงเปรียบเทียบกับความแตกต่างของความถี่ของสองโทน



รูปที่ 4.6 ระดับ Threshold ที่สัญญาณโทนทดสอบถูกบดบังโดยสัญญาณรบกวนสองแถบ (Bandpass Noise) โดยเปรียบเทียบกับความแตกต่างความถี่ของ Cutoff ของทั้งสองแถบ

อีกวิธีที่ใช้ในการประมาณความกว้างแถบความถี่วิกฤตคือการใช้สัญญาณโทนสองความถี่ที่มีระดับสัญญาณเท่ากันทำหน้าที่เป็น Masker และให้สัญญาณรบกวนแถบสัญญาณแคบทำหน้าที่

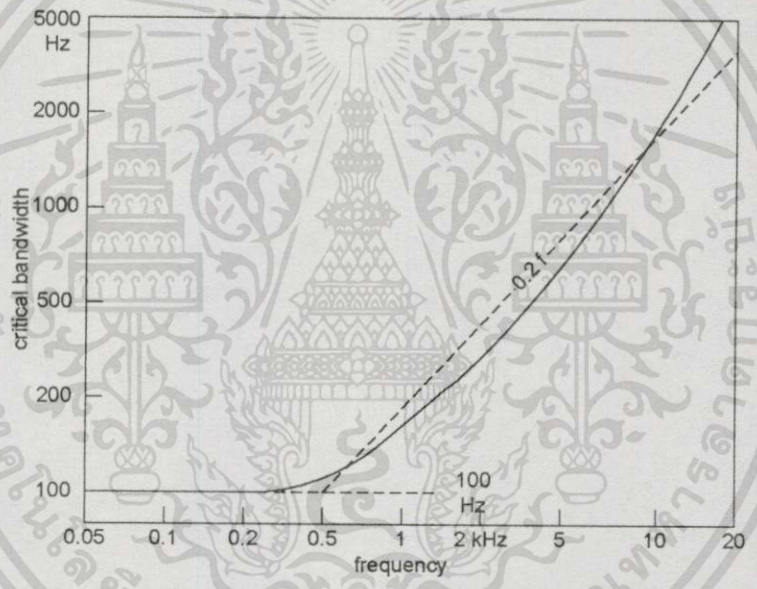
เป็นสัญญาณทดสอบ โดยจะทำการวัด Threshold ของการที่สัญญาณทดสอบถูกบดบังเทียบกับความแตกต่างของความถี่ของสัญญาณโทนทั้งสองซึ่งมีสัญญาณทดสอบอยู่ที่ความถี่กึ่งกลาง ดังนั้นช่วงความถี่ของสัญญาณทดสอบจะต้องแคบกว่าความกว้างแถบสัญญาณความถี่วิกฤต ในรูปที่ 4.5 ภาพเล็กแสดงสัญญาณโทนทั้งสองที่ใช้ในการบดบังสัญญาณ และสัญญาณรบกวนแถบสัญญาณแคบที่ใช้เป็นเป็นสัญญาณทดสอบในแกนความถี่ และในรูปใหญ่เป็นข้อมูลระดับ threshold เมื่อถูกบดบังด้วยสัญญาณโทนที่มีระดับ 50 dB สองค่าและสัญญาณทดสอบมีความถี่ศูนย์กลางที่ 2 kHz โดยระดับ Threshold ของสัญญาณรบกวนแถบความถี่แคบที่ถูกบดบังสัญญาณโดยสัญญาณโทนทั้งสองนั้นจะแสดงเปรียบเทียบกับความแตกต่างความถี่ของสัญญาณโทนทั้งสอง สำหรับที่ความแตกต่างของความถี่น้อยๆนั้นระดับ threshold จะไม่ขึ้นกับความแตกต่างของความถี่ แต่เมื่อความแตกต่างของความถี่เกินกว่าระดับหนึ่งไปแล้วระดับ Threshold จะลดลงซึ่งจุดที่ความแตกต่างความถี่ค่านั้นก็คือ ความกว้างแถบความถี่วิกฤต เมื่อทำการวัดโดยเปลี่ยนค่าความถี่ศูนย์กลางของสัญญาณทดสอบไปเรื่อยๆก็จะทำให้เราได้ความสัมพันธ์ระหว่างความกว้างแถบความถี่วิกฤตกับความถี่

อีกวิธีการหนึ่งที่คล้ายกันจะแสดงในรูปที่ 4.6 เราจะใช้สัญญาณรบกวนแถบสัญญาณแคบสองช่วงความถี่มาเป็น Masker แล้วใช้สัญญาณโทนเป็นสัญญาณทดสอบโดยให้ความแตกต่างระหว่าง Higher และ Lower cutoff ของสัญญาณรบกวนแถบความถี่แคบทั้งสองเป็นความแตกต่างความถี่และให้มันเปลี่ยนค่าไปเรื่อยๆขณะที่ทำการวัดระดับ Threshold โดยสัญญาณทดสอบเป็นสัญญาณโทนความถี่ 2 kHz และระดับของสัญญาณรบกวนทั้งสองแถบที่ใช้เป็น Masker มีระดับ 50 dB และสัญญาณรบกวนแถบความถี่แคบที่มีแถบความถี่กว้าง 200 Hz ซึ่งจะเห็นได้ว่าได้ผลเช่นเดียวกันกับที่ได้จากวิธีการก่อนหน้านี้

นอกจากนี้ยังมีวิธีการอื่นๆอีกที่ใช้ในการหาความกว้างแถบความถี่วิกฤต เช่นการใช้การตรวจสอบการเปลี่ยนเฟส, การวัดระดับความดังของเสียง, และการวัดโดยใช้ Short Impulse เป็นต้น ซึ่งวิธีการทั้งหมดนี้จะให้ผลที่ใกล้เคียงกัน

4.3 Critical Band Rate Scale หรือ Bark scale

มีวิธีการหลายอย่างที่ใช้ในการประมาณค่าความกว้างแถบความถี่วิกฤตซึ่งได้กล่าวผ่านมาในหัวข้อที่แล้ว ซึ่งแถบความถี่วิกฤตแรกที่อยู่ในช่วงความถี่ต่ำที่สุดของระบบการได้ยินของมนุษย์จะอยู่ที่ประมาณ 20-100 Hz แต่จะเป็นการง่ายกว่าหากเรารวมเอาช่วงความถี่ 0-20 Hz ที่เราไม่สามารถได้ยินได้รวมเข้าไว้ด้วย และกำหนดให้แถบความถี่วิกฤตแรกของการได้ยินของมนุษย์เป็น 0-100 Hz ในรูปที่ 4.7 เป็นข้อมูลที่ได้จากการหาค่าเฉลี่ยจากการใช้วิธีการต่างๆกับกลุ่มตัวอย่างจำนวนมาก แสดงให้เห็นว่าที่ความถี่ต่ำกว่า 500 Hz นั้นแถบความถี่วิกฤตจะมีความกว้างประมาณ 100 Hz ส่วนที่ความถี่สูงกว่านั้นแถบความถี่วิกฤตจะกว้างประมาณ 20 เปอร์เซ็นต์ของความถี่ศูนย์กลาง ส่วนค่าที่ได้จากผลการทดลองจริงๆนั้นอยู่ในตารางที่ 4.1 ที่แสดงทั้งของเขตบนและล่างของแถบความถี่วิกฤต



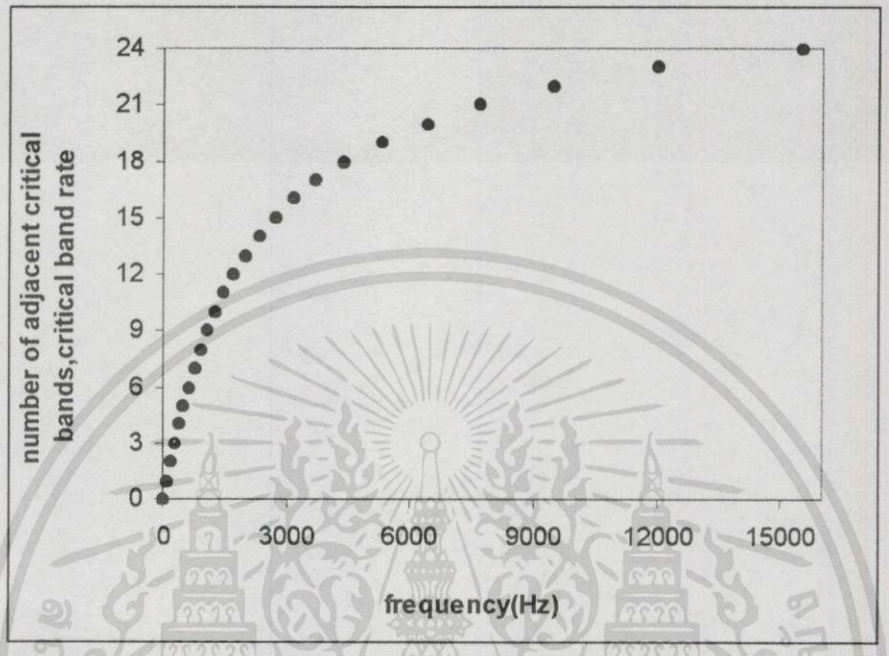
รูปที่ 4.7 ความกว้างแถบความถี่วิกฤตเทียบกับความถี่ และเส้นประแสดงการประมาณ

ตารางที่ 4.1 ขอบเขตและความถี่ศูนย์กลาง Critical Band Rate scale[14]

Z Bark	f_{Lower} / f_{Upper} Hz	f_c Hz	f_c Bark	Bandwidth Hz	Z Bark	f_{Lower} / f_{Upper} Hz	f_c Hz	f_c Bark	Bandwidth Hz
	0					1720			
0		50	0.5	100	12		1850	12.5	280
	100					2000			
1		150	1.5	100	13		2150	13.5	320
	200					2320			
2		250	2.5	100	14		2500	14.5	380
	300					2700			
3		350	3.5	100	15		2900	15.5	450
	400					3150			
4		450	4.5	110	16		3400	16.5	550
	510					3700			
5		570	5.5	120	17		4000	17.5	700
	630					4400			
6		700	6.5	140	18		4800	18.5	900
	770					5300			
7		840	7.5	150	19		5800	19.5	1100
	920					6400			
8		1000	8.5	160	20		7000	20.5	1300
	1080					7700			
9		1170	9.5	190	21		8500	21.5	1800
	1270					9500			
10		1370	10.5	210	22		10500	22.5	2500
	1480					12000			
11		1600	11.5	240	23		13500	23.5	3500
	1720					15500			

ความหมายของแถบความถี่วิกฤตนั้นมีความสำคัญต่อการอธิบายลักษณะในการได้ยินของมนุษย์อย่างมาก โดยจะถูกใช้ในแบบจำลองต่างๆในรูปแบบของสเกลที่เรียกว่า Critical Band Rate Scale หรือรู้จักกันในชื่อของ Bark scale ซึ่งสเกลนี้อ้างอิงจากหลักความจริงที่ว่าระบบการรับฟังเสียงของคนเรานั้นจะทำการแบ่งการวิเคราะห์เสียงออกเป็นส่วนต่างๆตามแถบความถี่วิกฤต และถ้าเรายึดเอาแถบความถี่วิกฤตแรกเป็นจุดอ้างอิงแล้วทำการคำนวณแถบความถี่วิกฤตต่อมาเรื่อยๆโดยให้ขอบเขตบนของแถบความถี่วิกฤตปัจจุบันเป็นขอบเขตล่างของแถบความถี่วิกฤตถัดไปก็จะทำให้

ได้สเกลที่เรียกว่า Bark scale ซึ่งจากตารางที่ 1 จะเห็นได้ว่าช่วงความถี่ถึง 16 kHz นั้นจะถูกแบ่งออกเป็นแถบความถี่วิกฤต 24 แถบ ส่วนในรูปที่ 4.8 เป็นแผนภูมิที่นำข้อมูลในตารางที่ 4.1 มาแสดงให้เห็น ซึ่งใน Bark scale นั้นอันดับของ Critical Band ถูกเรียกเป็น "Bark"



รูปที่ 4.8 ความสัมพันธ์ระหว่างลำดับของ Critical Band กับความถี่

ซึ่งความสัมพันธ์ระหว่าง Bark scale และ สเกลความถี่มีการประมาณการแปลงระหว่างความถี่ Hz และ Bark ตลอดช่วงความถี่เสียงที่เราสามารถได้ยินดังต่อไปนี้[14]

$$z = 13\arctan(0.76f) + 3.5\arctan(\frac{f}{7.5})^2 \text{ Bark} \qquad (4.1)$$

และ

$$\Delta f_G = 25 + 75[1 + 1.4f^2]^{0.69} \text{ Hz} \qquad (4.2)$$

โดยที่ f เป็นความถี่ในหน่วย kHz

4.3 Critical Band Intensity

ความสามารถในการรับรู้เสียงในย่านความถี่ต่างๆของระบบการได้ยินเสียงของมนุษย์นั้นสามารถอธิบายอย่างคร่าวๆได้โดยความเข้มของสัญญาณเสียงที่ถูกคำนวณโดยแบ่งเป็นย่านความถี่

ตามแถบความถี่วิกฤตซึ่งเรียกค่านี้ว่าเป็น Critical Band Intensity โดยค่า Critical Band Intensity สามารถคำนวณได้โดยใช้สมการดังนี้

$$I_G(f) = \int_{f-0.5\Delta f_G(f)}^{f+0.5\Delta f_G(f)} \frac{dI}{df} df \tag{4.3}$$

และสามารถเขียนได้เป็น

$$I_G(z) = \int_{z-0.5Bark}^{z+0.5Bark} \frac{dI}{dz} dz \tag{4.4}$$

สำหรับการแสดงค่าในสเกล logarithm นั้นเราใช้ I_0 เป็นค่าอ้างอิงดังนั้น Critical Band Level(L_G) สามารถหาได้จาก

$$L_G = 10 \cdot \log \frac{I_G}{I_0} dB \tag{4.5}$$

จากสมการเห็นได้ว่า Critical Band Intensity นั้นเป็นค่าความเข้มส่วนหนึ่งของเสียงในช่วงความถี่ของแถบความถี่วิกฤตและการแปลงจากสเกลความถี่ให้เป็น Bark scale นั้นคือการเปลี่ยนจากขนาด Window ที่ขึ้นกับความถี่เป็นขนาด Window ที่มีขนาดเดียวคือ 1 Bark ไม่ขึ้นกับช่วงความถี่ใดๆ

บทที่ 5

การหา k-Nearest Neighbor โดยใช้ kd-tree

ในบทนี้จะกล่าวถึงคำจำกัดความของ Nearest Neighbor[16], โครงสร้างข้อมูลแบบ kd-tree [17], k-Nearest Neighbor[18] และวิธีในการสร้างแบบอ้างอิงให้อยู่ในรูปแบบ kd-tree[19] รวมถึงวิธีการในการจัดกลุ่มเวคเตอร์ใดๆเข้ากลุ่มโดยวิธี k-Nearest Neighbor

5.1 ความหมายของ Nearest Neighbor

กำหนดให้มีสเปสแบบหลายมิติ (Multi-dimensional Space) ให้เวคเตอร์ข้อมูลสมาชิก (Exemplar) เป็นสมาชิกในเซตของเวคเตอร์ข้อมูลสมาชิก E และให้เซตของเวคเตอร์ข้อมูลสมาชิกประกอบไปด้วยเวคเตอร์ข้อมูลสมาชิกจำนวนจำกัดจำนวนหนึ่ง

E เป็นเซตของเวคเตอร์ข้อมูลสมาชิกและ $\underline{d}$ เป็นเวคเตอร์เป้าหมายที่สนใจและต้องการหา Nearest Neighbor ถ้าให้ Nearest Neighbor ของ $\underline{d}$ คือ $\underline{d'}$ ซึ่ง $\underline{d'} \in E$ หมายความว่าไม่มีจุดใดๆใน E ที่ใกล้กับ $\underline{d}$ มากกว่า $\underline{d'}$ หรือ $None-nearer(E,d,d')$ ก็ต่อเมื่อ

$$None-nearer(E,\underline{d},\underline{d'}) \Leftrightarrow \forall \underline{d''} \in E \ |\underline{d}-\underline{d'}| \le |\underline{d}-\underline{d''}| \qquad (5.1)$$

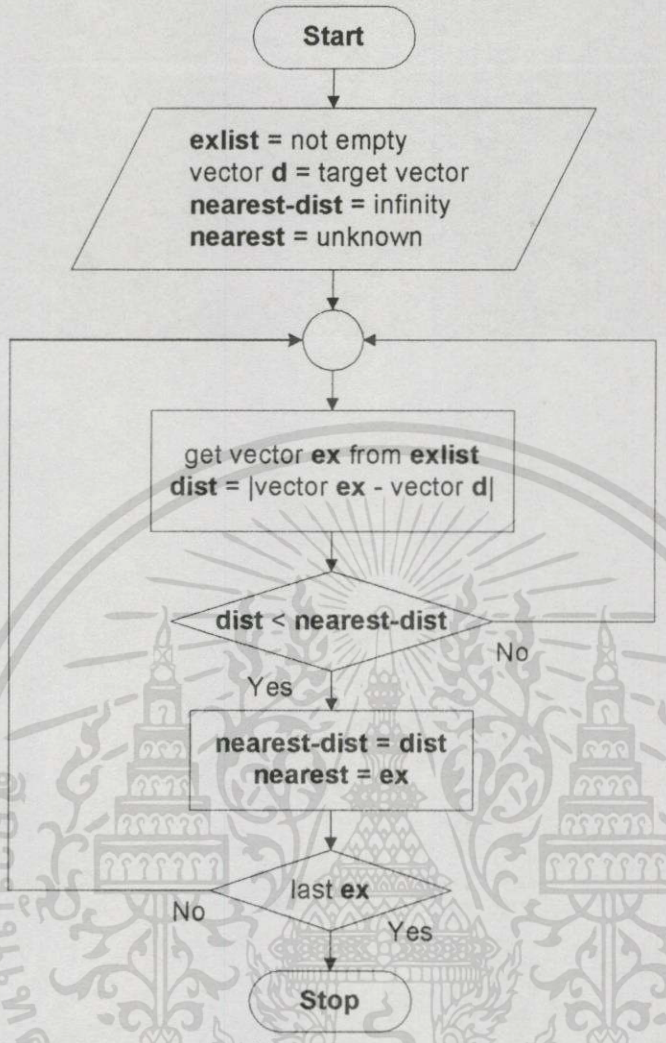
ในสมการที่5.1 ระยะห่างที่ใช้คือระยะทางแบบ Euclidean ซึ่งสามารถหาได้โดย

$$|\underline{d}-\underline{d'}| = \sqrt{\sum_{i=1}^{i=k_d}(d_i - d_i')^2} \qquad (5.2)$$

โดยที่ d_i เป็นองค์ประกอบในมิติหนึ่งของเวคเตอร์ $\underline{d}$

5.2 การหา Nearest Neighbor โดยตรง

วิธีการนี้ทำโดยการแจกแจงเซตของเวคเตอร์ข้อมูลสมาชิกออกเป็นลำดับรายการของเวคเตอร์ข้อมูลสมาชิกทั้งหมด ในรูปที่ 5.1 เป็นโฟลวชาร์ทแสดงขั้นตอนในการหา Nearest Neighbor โดยตรงโดยการตรวจดูเวคเตอร์ข้อมูลสมาชิกในรายการทั้งหมดซึ่งจะทำให้วิธีการนี้มีความซับซ้อนในการคำนวณเป็น $O(N)$ โดยที่ N เป็นขนาดของ E แต่ถ้าเราทำการจัดโครงสร้างของเซตของเวคเตอร์ข้อมูลสมาชิกให้มีประสิทธิภาพมากขึ้น เราก็สามารถที่จะหลีกเลี่ยงจากการที่จะต้องคำนวณหาระยะทางจากทุกเวคเตอร์ข้อมูลสมาชิกออกไปได้



exlist = เซตของเวคเตอร์ข้อมูลสมาชิก

vector **d** = เวคเตอร์ที่ต้องการหา Nearest Neighbor

vector **ex** = เวคเตอร์ข้อมูลสมาชิก

dist = ระยะห่างระหว่างเวคเตอร์ ex และ เวคเตอร์ d

nearest = เวคเตอร์ที่เป็น Nearest Neighbor

nearest-dist = ระยะห่างระหว่างเวคเตอร์ d กับเวคเตอร์ที่เป็น Nearest Neighbor

รูปที่ 5.1 โฟลวชาร์ทการคำนวณหา Nearest Neighbor โดยตรง

5.3 kd-tree

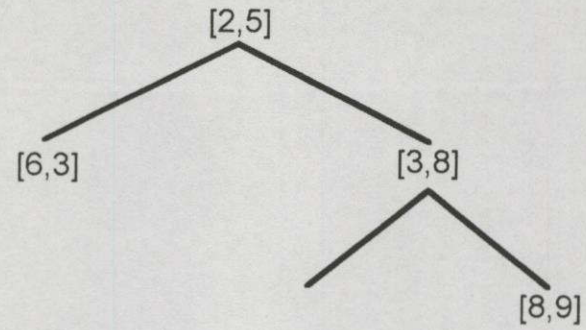
kd-tree เป็นโครงสร้างข้อมูลที่ออกแบบมาเพื่อใช้ในการเก็บข้อมูลเวคเตอร์ k มิติที่จะทำให้การค้นหาข้อมูลเป็นไปอย่างมีประสิทธิภาพ kd-tree เป็น Binary tree แบบหนึ่ง ความหมายของแต่ละส่วนประกอบในโหนดของ kd-tree แสดงในตารางที่ 5.1

ตารางที่ 5.1 องค์ประกอบต่างๆภายในโหนด

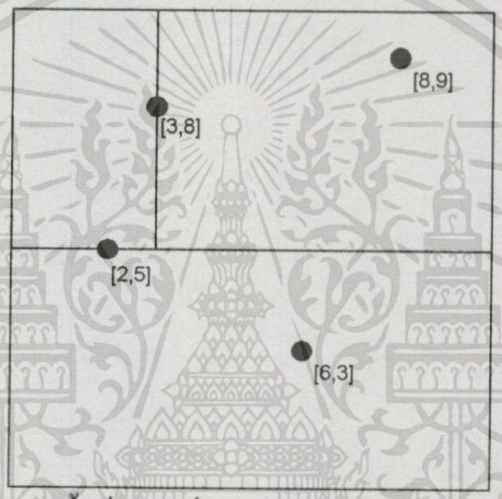
Field Name:	**Field Type**	Description
dom	domain-vector	A point from $k_d - d$ space
split	integer	The splitting dimension
left	kd-tree	A kd-Tree representing those points to the left of the splitting plane
right	kd-tree	A kd-Tree representing those points to the right of the splitting plane

เซตของเวคเตอร์ข้อมูลสมาชิก E จะถูกแสดงโดยโหนดต่างๆใน kd-tree ซึ่งแต่ละโหนดจะประกอบด้วยเวคเตอร์ข้อมูลสมาชิกหนึ่งตัว จากตารางที่ 5.1 ข้อมูลหนึ่งที่อยู่ในโหนดคือ dom ซึ่งจะเป็นตัวบอก domain-vector หรือพิกัดในสเปสของเวคเตอร์ข้อมูลสมาชิก โดยจะทำหน้าที่เป็นหลักในการแบ่งสเปสออกเป็นส่วนย่อยๆสองส่วนตามระนาบหลายมิติที่ใช้ในการแบ่งสเปส (Splitting Hyperplane) จุดข้อมูลทั้งหมดที่อยู่ทางสเปสย่อยด้านซ้ายจะแสดงโดย Tree ย่อยด้ายซ้าย ส่วนจุดข้อมูลที่อยู่ด้านขวาก็จะแสดงโดย Tree ย่อยที่อยู่ด้านขวาซึ่งระนาบหลายมิติที่ใช้ในการแบ่งสเปสคือ ระนาบที่ผ่านจุด dom และตั้งฉากกับทิศทางที่กำหนดโดยค่า split ในโหนด ถ้าให้ i เป็นค่าของการ split ในโหนดแล้ว จุดใดๆจะอยู่ทางซ้ายของ dom ก็ต่อเมื่อมิติที่ i ของมันมีค่าน้อยกว่าหรือเท่ากับมิติที่ i ของ dom ส่วนเวคเตอร์สมาชิกที่เหลือก็จะอยู่ทางขวา

รูปที่ 5.2 แสดง kd-tree ซึ่งมี dom 4 ค่าคือ (2,5),(3,8),(6,3) และ (8,9) โดยที่โหนดราก (Root) มีค่า dom เป็น (2,5) จะแบ่งสเปสในแกน y เป็นสองสเปสย่อย โดยที่จุด (3,8) อยู่ใน สเปสย่อย ซึ่ง $\{(x, y) \mid y \leq 5\}$ และอยู่ใน Tree ย่อยด้านซ้าย ในรูปที่ 5.3 แสดงรูปในลักษณะสองมิติให้เห็นว่าโหนดต่างๆแบ่งส่วนสเปสอย่างไร



รูปที่ 5.2 โครงสร้างข้อมูลสองมิติซึ่งประกอบไปด้วยเวคเตอร์สมาชิก 4 ตัว โหนด(2,5) ทำหน้าที่แบ่งระนาบ y=5 และโหนด (3,8) ทำหน้าที่แบ่งระนาบ x=3



รูปที่ 5.3 แผนภาพแสดงการแบ่งพื้นที่ในรูปที่ 5.2

5.4 การสร้าง kd-tree

เราสามารถสร้าง kd-tree ได้จากเซตของเวคเตอร์ข้อมูลสมาชิก E ได้จากโฟลวชาร์ทในรูปที่ 5.4 และวิธีการในการเลือก Pivot นั้นจะทำการเลือกเอาโดเมนเวคเตอร์ที่เหมาะสมในเซตเพื่อใช้เป็นรากของ tree ซึ่งวิธีการในการเลือก pivot นั้นจะกล่าวถึงโดยละเอียดในหัวข้อถัดไป แต่ไม่ว่าจะเลือกเวคเตอร์ข้อมูลสมาชิกใดเป็นราก ก็จะไม่มีผลต่อความถูกต้องของ kd-tree แต่จะมีผลความลึกและลักษณะของ tree เท่านั้น โครงสร้างข้อมูล kd-tree จะประกอบไปด้วย root, split, kdleft,และ kdright โดยที่

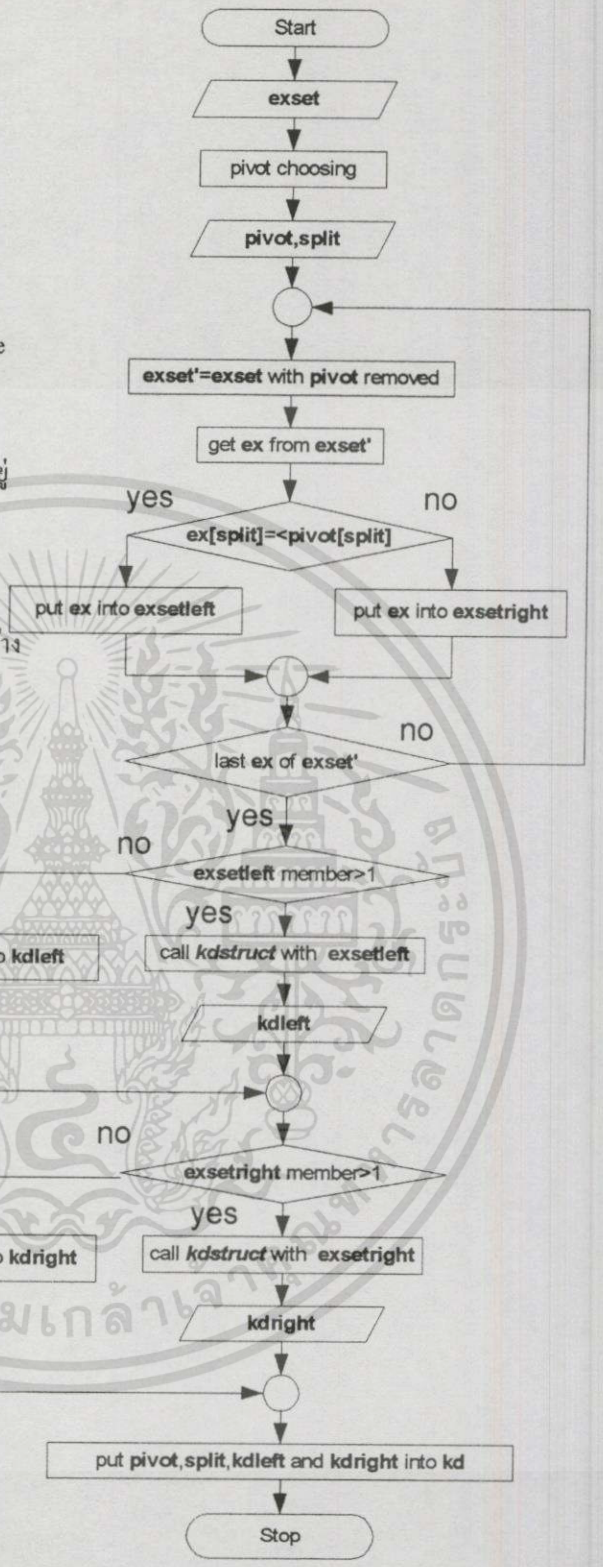
root คือ เวคเตอร์ที่ใช้เป็นรากของโครงสร้างข้อมูลหรือเป็น pivot แรกในการแบ่งสเปส

split คือ ค่าที่บอกว่าจะใช้มิติใดของเวคเตอร์ที่เลือกเป็น pivot เป็นตัวแบ่งสเปส

kdleft, kdright คือ kd-tree ย่อยภายในที่อยู่ทางด้านซ้ายและขวาของ splitting plane

kdstruct procedure
input : **exset (not empty)**
output : **kd**

exset : เซตของเวคเตอร์ทั้งหมดในสเปส
kd : โครงสร้างข้อมูล **kdtree**
pivot : เวคเตอร์ที่ถูกเลือกให้เป็นจุดแบ่งสเปส
split : มิติของ **pivot** ที่ถูกใช้เป็น splitting plane
exset' : เซตของเวคเตอร์ทั้งหมดยกเว้น **pivot**
ex : เวคเตอร์ในสเปส
exsetleft : เซตของวคเตอร์ทั้งหมดในสเปสที่อยู่ทางซ้ายของ splitting plane
exsetright : เซตของวคเตอร์ทั้งหมดในสเปสที่อยู่ทางขวาของ splitting plane
kdleft : องค์ประกอบหนึ่งของ kd ซึ่งมีโครงสร้างเช่นเดียวกับ kd
kdright : องค์ประกอบหนึ่งของ kd ซึ่งมีโครงสร้าง เช่นเดียวกับ kd



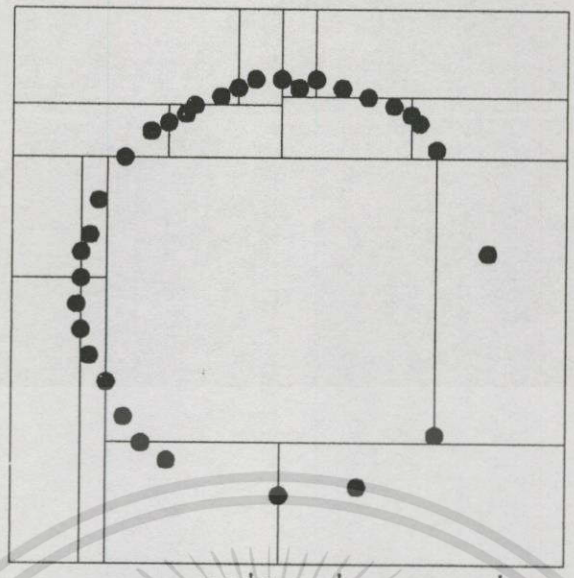
รูปที่ 5.4 โฟลวชาร์ทการคำนวณการสร้าง kd-tree

5.5 การเลือก Pivot จากเซตของเวคเตอร์ข้อมูลสมาชิก

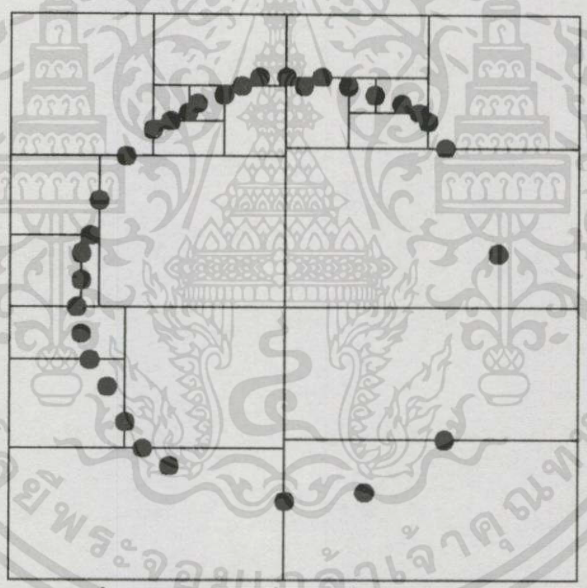
วิธีการในการสร้าง Tree ในหัวข้อที่ผ่านมานั้นต้องการค่า Pivot และ Splitting plane เพื่อที่จะสร้างรากของ kd-tree ซึ่งในการสร้าง Tree นั้นควรจะให้มีความสมดุลรวมทั้งรูปร่างของรูปทรงหลายมิติ (Hyper Region:hr) ของโหนดกิ่งก็ควรมีขนาดในมิติต่างๆที่เท่าๆกันโดยในเรื่องของความสมดุลย์นั้นมีความสำคัญเนื่องจากหากว่า Tree มีลักษณะที่ไม่สมดุลอาจทำให้ความซับซ้อนของการคำนวณในการเข้าถึงโหนดกิ่งสุดท้ายนั้นสูงขึ้นถึง $O(N)$ แทนที่จะเป็น $O\log(N)$ ก็เป็นได้ ส่วนเงื่อนไขที่ต้องการให้ทุกๆมิติของรูปทรงหลายมิติที่แบ่งได้มีขนาดที่เท่าๆกันนั้นทำเพื่อให้เกิดโอกาสในการสิ้นสุดการค้นหามากที่สุด รูปที่ 5.5 เป็น Tree ที่มีความสมดุล ซึ่งโหนดกิ่งมีลักษณะไม่เป็นจัตุรัส แต่ในรูปที่ 5.6 เป็นจุดข้อมูลชุดเดียวกันแต่มีการทำให้บริเวณโหนดกิ่งมีความเป็นจตุรัสมากขึ้น

วิธีในการหา Pivot แบบหนึ่งซึ่งสามารถทำให้ได้ Tree ที่สมดุลคือการหา Splitting Dimension ที่มี Variance สูงสุดและให้ Pivot เป็นจุดที่เป็น Median ของจุดข้อมูลที่จะทำการแบ่ง การใช้วิธีนี้จะทำให้พื้นที่ที่แบ่งออกใกล้เคียงจัตุรัส เนื่องจากเป็นการแบ่งทีละมิติ แต่วิธีการนี้จะให้ผลดีในกรณีที่จุดมีการกระจายทั่วถึงกันหมด ถ้าหากมีข้อมูลมีการกระจายไปด้านใดด้านหนึ่งอย่างมากก็จะทำให้รูปทรงที่แบ่งได้มีขนาดที่แตกต่างอย่างมากในบางมิติ ซึ่งจะทำให้ต่างจากลักษณะจัตุรัสที่ต้องการมากขึ้นซึ่งจะเห็นได้จากรูปที่ 5.6

เพื่อเป็นการหลีกเลี่ยงไม่ให้เกิดรูปทรงหลายมิติที่มีลักษณะนี้ เราสามารถเลือก Pivot ที่แบ่งเวคเตอร์ข้อมูลสมาชิกบริเวณกึ่งกลางของมิติที่มีการกระจายมากที่สุด จะเห็นได้ในรูปที่ 5.6 ว่าเราจะได้การแบ่งรูปทรงซึ่งมีรูปร่างใกล้เคียงจัตุรัสมากกว่าโดยที่เราจะสูญเสียความสมดุลย์ของ Tree ไปบ้าง นั่นหมายความว่าบริเวณพื้นที่ว่างเปล่าขนาดใหญ่จะแบ่งเป็นรูปทรงหลายมิติเพียงไม่กี่ส่วน ลักษณะเช่นนี้ทำให้จำนวนของโหนดกิ่งที่จะต้องค้นหาในกรณีที่ในพื้นที่มี Nearer Neighbor อยู่ มีจำนวนน้อยลงกว่าแบบปกติ



รูปที่ 5.5 การเลือก Pivot จาก Median ของมิติที่มีการเบี่ยงเบนสูงสุดซึ่งจะทำให้ Tree ที่ได้มีความสมดุลมากที่สุด

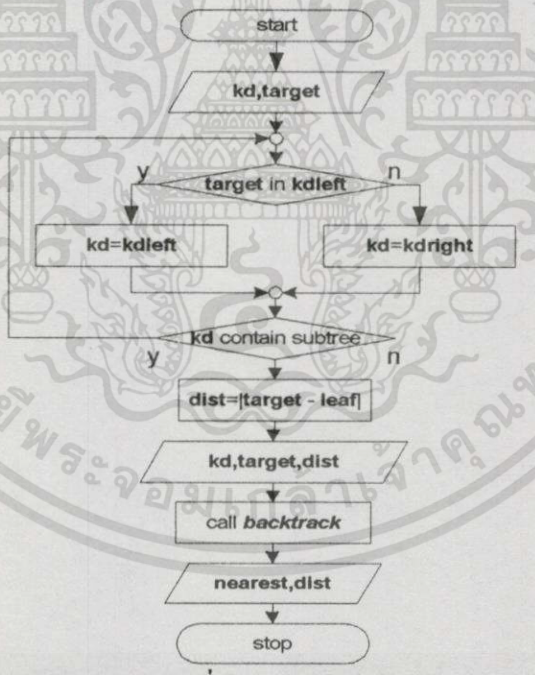


รูปที่ 5.6 การเลือก Pivot จากจุดกึ่งกลางของมิติที่กว้างที่สุดทำให้ได้รูปทรงที่มีลักษณะใกล้เคียงจัตุรัสมากขึ้นแต่สูญเสียสมดุลย์บางส่วนไป

วิธีการในการเลือก Pivot นี้เริ่มต้นโดยเลือก Splitting Dimension จากมิติที่ยาวที่สุดใน รูปทรงหลายมิติปัจจุบันแล้วเลือกจุด Pivot จากจุดที่ใกล้กับจุดกึ่งกลางของรูปทรงหลายมิติในมิตินั้นมากที่สุด บางครั้ง Pivot อาจทำให้เกิดโหนดที่สูญเสียความสมดุลซึ่งเป็นกรณีที่ไม่ค่อยดี เนื่องจากจะทำให้เกิดโหนดกิ่งว่างเปล่าขนาดใหญ่ขึ้นและถ้าหากเกิดโหนดว่างเปล่าขนาดใหญ่จนสูญเสียความสมดุลย์มากเกินไปก็จะเห็นกลับมาใช้การเลือก Pivot แบบที่เลือกจาก Median

5.6 Nearest Neighbor Search ใน kd-tree

ในหัวข้อนี้จะอธิบายถึงวิธีการในการหา Nearest Neighbor บนโครงสร้างข้อมูลแบบ kd-tree ในตัวอย่างเริ่มต้นด้วยเวคเตอร์เป้าหมายที่ต้องการหา Nearest Neighbor อยู่ในโหนดกิ่งดังในรูปที่ 5.8 ที่แสดงโดยเครื่องหมาย **X** และโหนดกิ่งที่เวคเตอร์เป้าหมายอยู่เป็นจุดสีดำซึ่งโหนดนี้ยังไม่สามารถสรุปได้ว่าเป็น Nearest Neighbor แต่เราสามารถบอกได้ว่าโหนดใดๆก็ตามที่จะอยู่ใกล้จุดเป้าหมายมากกว่าโหนดนี้จะต้องอยู่ภายในวงกลมซึ่งมีจุดศูนย์กลางอยู่ที่จุดเป้าหมายและผ่านโหนดกิ่งนี้ และเมื่อย้อนกลับไปดูที่โหนดแม่ของโหนดกิ่งนี้ซึ่งในรูปที่ 5.9 เป็นจุดสีดำแล้วดูว่าจะมีโหนดลูกอื่นของโหนดแม่นี้ที่อยู่ใกล้จุดเป้าหมายมากกว่าหรือไม่ ซึ่งในรูปที่ 5.9 นี้จะเห็นว่าไม่มี เนื่องจาก วงกลมที่เป็นขอบเขตของจุดที่ใกล้กว่าไม่มาซ้อนทับกับสเปสของโหนดลูกอื่นของโหนดแม่นี้ ถ้าหากไม่มีจุดที่ใกล้กว่าในลักษณะนี้เราสามารถที่จะทำในระดับชั้นถัดไปได้ทันที คือต้องตรวจสอบโหนดแม่ถัดไปซึ่งอยู่เหนือขึ้นไป เนื่องจากมันมีส่วนที่ทับซ้อนกันกับวงกลมที่กำหนดขอบเขตสูงสุด โดย รูปที่ 5.7 เป็นโฟลวชาร์ทแสดงขั้นตอนในการคำนวณหา Nearest Neighbor ของเวคเตอร์เป้าหมายในโครงสร้างข้อมูลแบบ kd-tree



โดยที่ kd : โครงสร้างข้อมูลแบบ kd-tree ที่ต้องการเข้าค้นหา

target : เวคเตอร์เป้าหมายที่ต้องการการค้นหา Nearest Neighbor

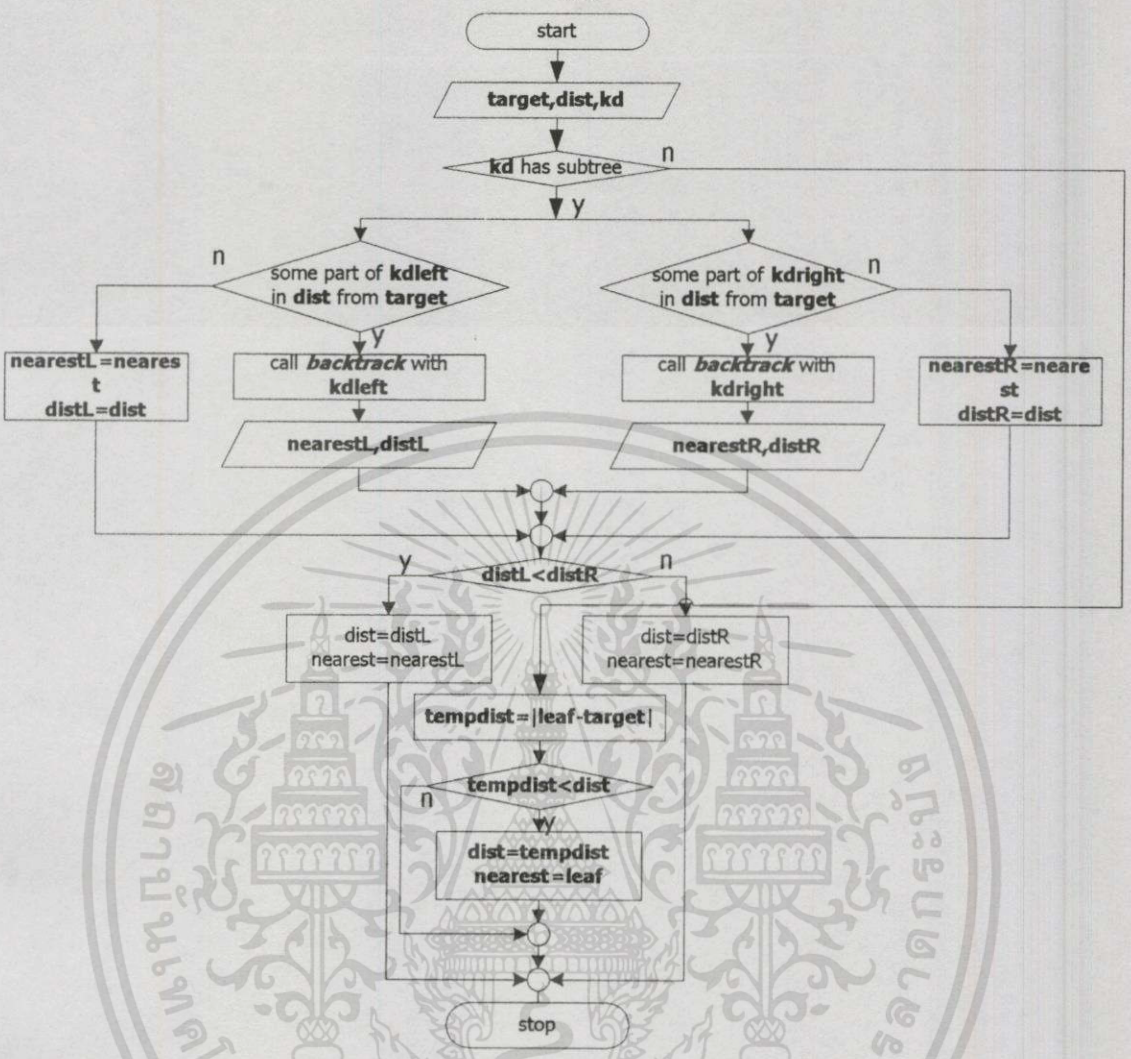
dist : ระยะห่างระหว่างโหนดกิ่งที่เวคเตอร์เป้าหมายอยู่กับเวคเตอร์เป้าหมายเอง

nearest : เวคเตอร์ที่เป็น Nearest Neighbor ของเวคเตอร์เป้าหมาย

(a) Main procedure

รูปที่ 5.7 โฟลวชาร์ทการคำนวณหา Nearest Neighbor ใน kd-tree

(b)



โดยที่ distL : ระยะห่างระหว่าง Nearest Neighbor บน kdleft กับเวคเตอร์เป้าหมาย

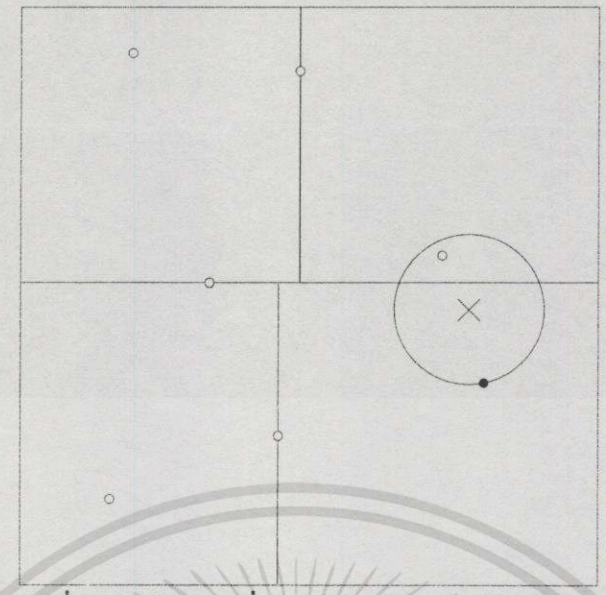
nearestL : โหนดกิ่งที่เป็น Nearest Neighbor บน kdleft กับเวคเตอร์เป้าหมาย

distR : ระยะห่างระหว่าง Nearest Neighbor บน kdright กับเวคเตอร์เป้าหมาย

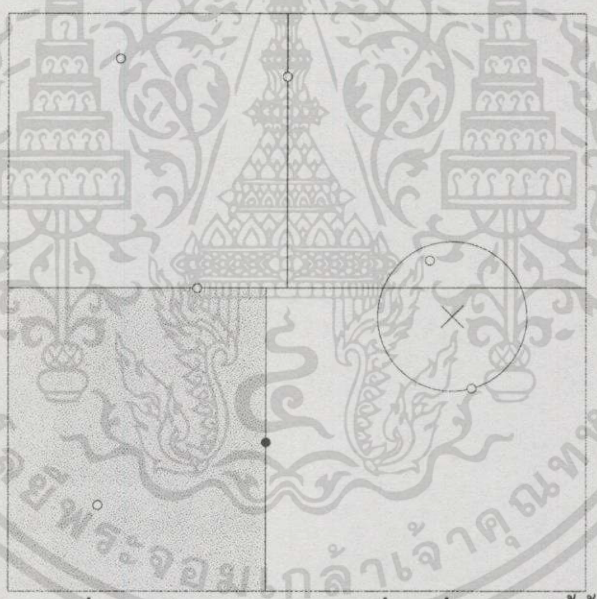
nearestR : โหนดกิ่งที่เป็น Nearest Neighbor บน kdright กับเวคเตอร์เป้าหมาย

(c) Subroutine *backtrack*

รูปที่ 5.7 (ต่อ)



รูปที่ 5.8 จุดสีดำเป็นโหนดซึ่งมีจุดเป้าหมายซึ่งแสดงโดยกากบาทเป็นสมาชิก และ nearer neighbor ต้องอยู่ภายในวงกลมนี้



รูปที่ 5.9 จุดสีดำเป็นโหนดแม่ที่อยู่ใกล้เวคเตอร์เป้าหมายที่สุด ซึ่งในกรณีนี้พื้นที่แรเงาซึ่งเป็นส่วนของโหนดลูกอีกอันหนึ่งไม่จำเป็นต้องเข้าไปทำการค้นหา

ในการที่จะตรวจสอบว่ารูปทรงหลายมิติ (Hyperrectangle: hr) มีการทับซ้อนกับทรงกลมหลายมิติ (Hypersphere) รัศมี r ซึ่งมีจุดศูนย์กลางที่ t หรือไม่ ต้องเริ่มต้นด้วยการหาจุด p บน hr ที่ใกล้กับจุด t มากที่สุด โดยการหาว่าจุด p ที่ใกล้กับจุด t ที่สุดอยู่ที่ใดทำโดย ให้ $hr_i^{\min}$ เป็นค่าต่ำสุดของมิติที่ i ของ hr และ $hr_i^{\max}$ เป็นค่ามากที่สุดของมิติที่ i ของ hr และ p_i เป็นมิติที่ i ของ p ซึ่งจะมีค่าตามเงื่อนไขต่อไปนี้

$$p_i = \begin{cases} hr_i^{\min} & \text{if } t_1 \le hr_i^{\min} \\ t_i & \text{if } hr_i^{\min} < t_i < hr_i^{\max} \\ hr_i^{\min} & \text{if } t_i \ge hr_i^{\max} \end{cases} \qquad (5.6)$$

รูปทรงหลายมิติและทรงกลมหลายมิติจะมีการทับซ้อนกันต่อเมื่อระยะห่างระหว่าง p กับ t น้อยกว่าหรือเท่ากับ r

การค้นหาที่ใช้จะเป็นทางด้านลึกก่อน(Depth first search) โดยจะค้นหาไปยังโหนดลูกที่มีเป้าหมายอยู่ จากโฟลวชาร์ทในรูปที่ 5.7 เริ่มต้นจาก kd-tree(kd) และ ระบุเวคเตอร์เป้าหมาย(target vector) ที่ต้องการหา Nearest Neighbor เริ่มด้วยทำการตรวจสอบว่าเวคเตอร์เป้าหมายอยู่ในโครงสร้างส่วนซ้าย(kdleft) หรือส่วนขวา(kdright) โดยตรวจสอบจากค่า pivot และ split แล้วจึงตรวจสอบว่ามีโครงสร้างย่อยภายใน kdleft หรือ kdright นี้หรือไม่หากมีก็จะต้องตรวจสอบลึกลงไปในโครงสร้างจนกระทั่งรู้ว่าเวคเตอร์เป้าหมายอยู่ในโหนดกิ่งใด ซึ่งสิ่งที่จะได้คือโหนดกิ่งเป็น Nearest Neighbor เบื้องต้นและได้ระยะห่าง(dist) จากโหนดกิ่งไปยังเวคเตอร์เป้าหมาย ถัดไปก็เป็นการตรวจสอบย้อนกลับไปยังโหนดอื่นๆ ว่ามีส่วนใดบ้างที่อยู่ในทรงกลมหลายมิติที่มีเวคเตอร์เป้าหมายเป็นศูนย์กลางและมีรัศมีเท่ากับระยะห่างจาก Nearest Neighbor เบื้องต้นไปยังเวคเตอร์เป้าหมาย หากพบว่าไม่มีโหนดกิ่งอื่นใดอยู่ในรัศมีก็จะไม่ต้องทำการค้นหาในบริเวณอื่น แต่ถ้าหากว่ามีก็จะทำการค้นหาเพื่อตรวจสอบว่ามีโหนดกิ่งใดที่มีระยะห่างต่ำกว่าที่ได้ก่อนหน้านี้หรือไม่

ปัญหาเบื้องต้นสำหรับการค้นหาใน kd-tree คือหากมี kd-tree ที่มีโหนดจำนวน N โหนด จะต้องค้นหาไปกี่โหนดจึงจะสามารถพบ Nearest Neighbor โดยใช้วิธีการในหัวข้อที่ผ่านมา เห็นได้ชัดว่าจะต้องมีการค้นหาอย่างน้อย $O(\log N)$ ครั้ง เนื่องจากต้องมีการค้นหาเข้าไปยังโหนดกิ่งอย่างน้อยหนึ่งกิ่ง และแต่ละโหนดนั้นจะเข้าถึงเพียงครั้งเดียวเท่านั้น

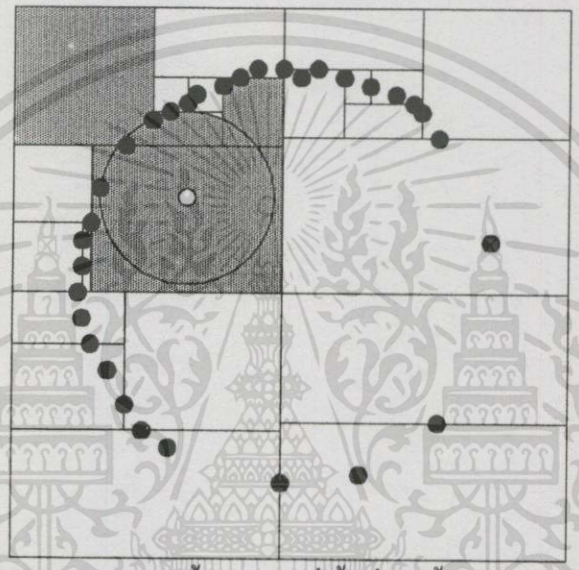
ในรูปที่ 5.10 จะเห็นได้ชัดว่าการค้นหาจะเกิดขึ้นเพียงไม่กี่โหนดเท่านั้น หากดูในรูปจะเห็นว่าเฉพาะส่วนที่แรเงาเท่านั้นที่จะมีการค้นหา สิ่งทีสำคัญในการค้นหาคือจำนวนของการค้นหาในกรณีที่เลวร้ายที่สุด และจำนวนครั้งที่คาดว่าจะต้องเกิดขึ้น สำหรับในกรณีที่เลวร้ายที่สุดนั้นสามารถเกิดขึ้นได้ในกรณีของการกระจายของจุดซึ่งทำให้จำเป็นต้องทำการค้นหาในทุกๆโหนด ในรูปที่ 5.11 Tree ลักษณะสองมิติซึ่งจุดต่างๆกระจายอยู่บริเวณขอบของวงกลมซึ่งทำให้มีความจำเป็นที่จะต้องค้นหาในทุกๆกิ่ง

สำหรับการคำนวนการประมาณจำนวนครั้งที่ต้องใช้ในการค้นหาข้อนค่างเป็นเรื่องยาก เนื่องจากการวิเคราะห์จะขึ้นอยู่กับลักษณะการกระจายของจุดใน kd-tree และตำแหน่งของจุดเป้าหมายด้วย

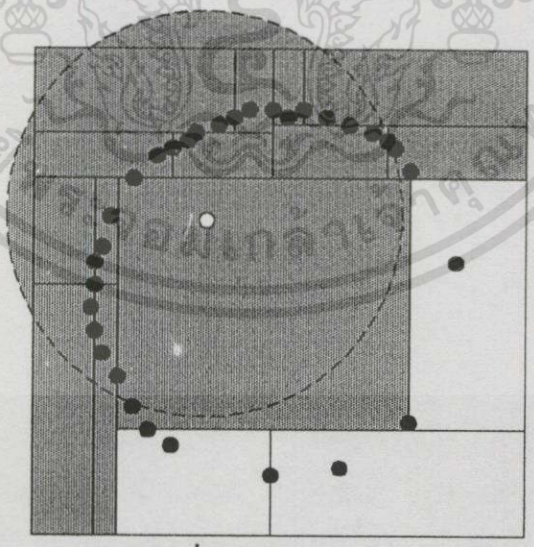
ในหัวข้อนี้เราจึงสนใจที่จำนวนของรูปทรงหลายมิติที่จำเป็นต้องเข้าไปค้นหาแทนโดยที่จะต้องเป็นรูปทรงหลายมิติที่มีส่วนซ้อนทับกับทรงกลมหลายมิติซึ่งมีจุดศูนย์กลางอยู่ที่จุดเป้าหมายซึ่ง

มีพื้นผิวผ่านจุดที่ใกล้ที่สุดในขั้นต้น ตัวอย่างในรูปที่ 5.10 มีรูปทรงหลายมิติเพียงสองส่วนที่มีส่วนซ้อนทับกับทรงกลมหลายมิติเท่านั้น เห็นได้ว่าจำนวนรูปทรงหลายมิติที่สนใจนั้นจะไม่ขึ้นกับค่า N และจำนวนของเวคเตอร์ข้อมูลสมาชิก ส่วนเวลาที่ใช้ในการค้นหาจะมีลักษณะเป็น logarithm เนื่องจากการค้นหาเข้าไปจากรากของ Tree ไปยังกิ่งมีลักษณะเป็น logarithm (ในกรณีที่ Tree นั้นมีความสมดุล) แต่จำนวนครั้งที่ใช้ในการ Back Tracking ต้องเป็นค่าคงที่

ถึงแม้ว่าจำนวนรูปทรงหลายมิติที่สนใจจะไม่ขึ้นอยู่กับจำนวน N แต่มันยังคงขึ้นอยู่กับค่าของ k ซึ่งก็คือจำนวนมิติของเวคเตอร์สมาชิก



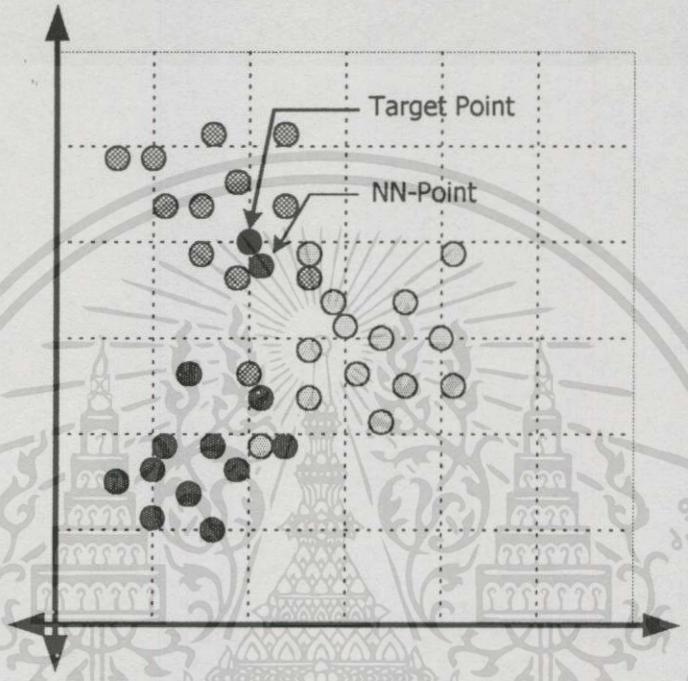
รูปที่ 5.10 โดยปกติแล้วการค้นหาจะเกิดขึ้นเพียงไม่กี่พื้นที่เท่านั้น



รูปที่ 5.11 การกระจายของเวคเตอร์สมาชิกที่ทำให้ต้องทำการค้นหาในหลายพื้นที่

5.7 k-Nearest Neighbor

การประยุกต์ที่นิยมมากอย่างหนึ่งของการทำ Nearest Neighbor คือการทำ k-Nearest Neighbor เราสามารถใช้วิธีนี้ในการจัดกลุ่มให้กับเวคเตอร์ใดๆเข้ากับกลุ่มที่ปรากฏมากที่สุดของเวคเตอร์ที่ใกล้กับเวคเตอร์เป้าหมายนั้นมากที่สุดจำนวน k ตัว หรืออีกนัยหนึ่งก็คือการตัดสินใจในการจัดกลุ่มให้กับเวคเตอร์ใดๆจะทำโดยการโหวตของเวคเตอร์ที่ใกล้ที่สุดจำนวน k ตัว



รูปที่ 5.12 การจัดกลุ่มโดยอาศัยการตัดสินใจของ k-Nearest Neighbor

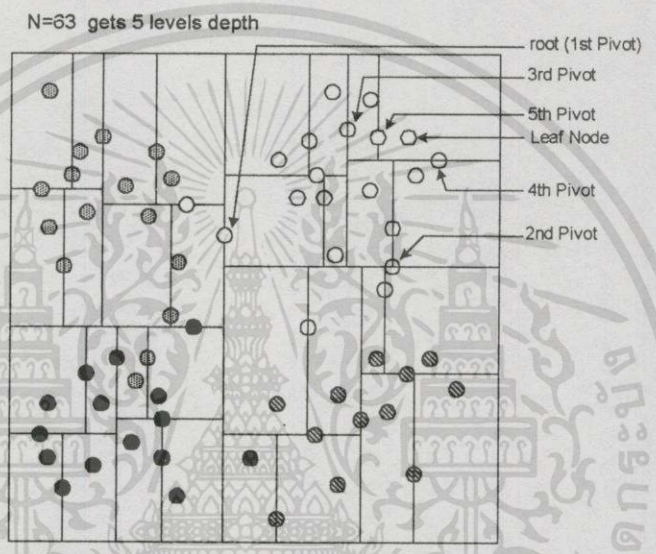
จากรูปที่ 5.12 จะเห็นได้ว่าหากอาศัยการตัดสินใจของ Nearest Neighbor เพียงค่าเดียวแล้วอาจได้ผลที่ผิดพลาดได้ เนื่องจาก Nearest Neighbor นั้นอาจเป็นค่าที่เกิดจากการเบี่ยงเบนอย่างมากก็เป็นได้ แต่ถ้าหากใช้การตัดสินใจของ Nearest Neighbor ในลำดับถัดมาด้วยก็จะช่วยให้มีความถูกต้องในการตัดสินใจมากขึ้น

5.8 Approximate k-Nearest Neighbor

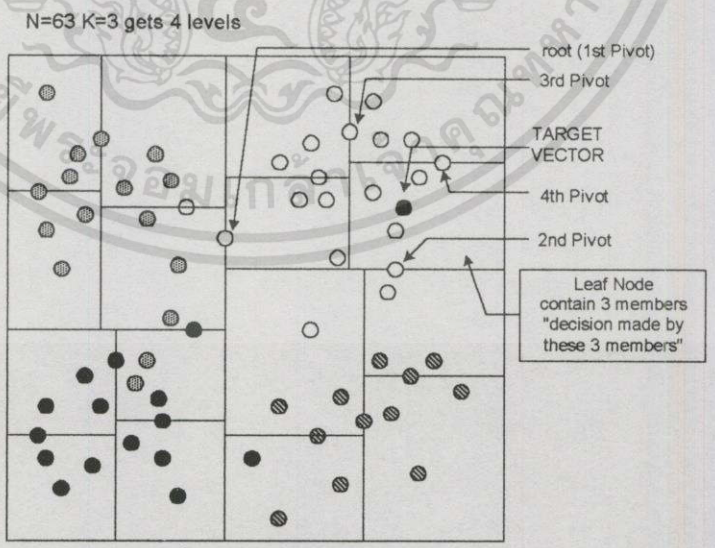
โครงสร้างของ kd-tree นั้น ภายในรูปทรงหลายมิติย่อยๆที่แบ่งได้จะมีเวคเตอร์ข้อมูลสมาชิกอยู่ได้เพียงตัวเดียว และในการค้นหา Nearest Neighbor สำหรับเวคเตอร์เป้าหมายใดๆ จะต้องมีการค้นหาไปยังโหนดกิ่งหรือ Hyper Region ที่เวคเตอร์เป้าหมายอยู่ก่อน ซึ่งต้องอาศัยการคำนวณ $O(\log n)$ ครั้งและต้องมีการค้นหา hr ที่อยู่โดยรอบในรัศมี ซึ่งทำให้ไม่สามารถระบุความซับซ้อนในการคำนวณได้

แต่ถ้าหากว่าชุดข้อมูลที่ใช้ในการสร้าง kd-tree มีการจัดกลุ่มและการกระจายตัวที่ดีแล้ว เราสามารถที่จะประมาณได้ว่า Nearest Neighbor ของเวคเตอร์เป้าหมายคือ เวคเตอร์ข้อมูลสมาชิกที่อยู่ใน hr เดียวกับที่เวคเตอร์เป้าหมายอยู่ เรียกการหา Nearest Neighbor แบบนี้ว่า Approximate Nearest Neighbor ซึ่งจะมีความซับซ้อนในการคำนวณที่แน่นอนคือ $O(\log n)$

และแทนที่ภายใน hr ย่อยหนึ่งๆจะมีเวคเตอร์สมาชิกอยู่เพียงเวคเตอร์เดียวดังรูปที่ 5.13 แต่อนุญาติให้มีเวคเตอร์สมาชิกจำนวน K ตัวดังรูปที่ 5.14 เราสามารถใช้ประโยชน์จาก hr ที่มีเวคเตอร์สมาชิก K ตัวนี้ในการประมาณหา k-Nearest Neighbor ของเวคเตอร์เป้าหมายได้ การประมาณนี้เรียกว่า Approximate k-Nearest Neighbor



รูปที่ 5.13 การจัดข้อมูลโดยใช้โครงสร้างข้อมูลแบบ kd-tree ปกติ



รูปที่ 5.14 โครงสร้างข้อมูลแบบ kd-tree เมื่อนำมาประยุกต์ใช้กับ Approximate k-NN

บทที่ 6

การทดลอง และ ผลการทดลอง

6.1 กล่าวนำ

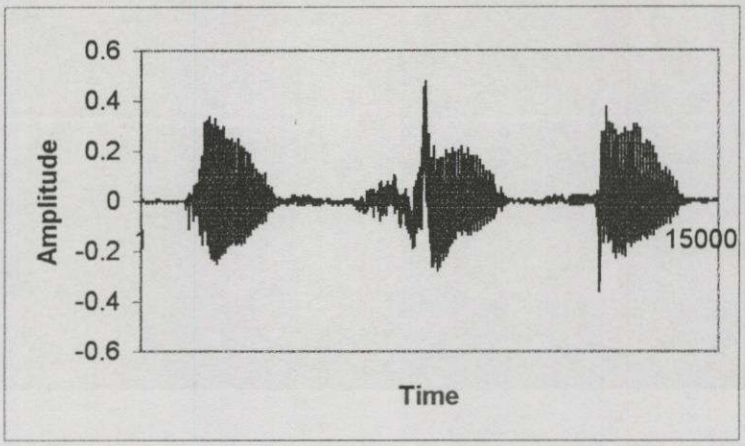
เนื้อหาในบทนี้จะกล่าวถึงวิธีการทดลองและผลการทดลองที่ได้ในขั้นตอนต่างๆของการรู้จำเสียงสระเสียงเดี่ยวของคำพยางค์เดียวในภาษาไทยโดยจะแบ่งเป็นสามส่วนใหญ่ๆดังนี้

- ส่วนที่ 1 เป็นขั้นตอนในการวิเคราะห์และพัฒนาอัลกอริทึมในการสร้างแบบอ้างอิงสำหรับการรู้จำเสียงสระเสียงเดี่ยวสำหรับภาษาไทย
- ส่วนที่ 2 เป็นขั้นตอนในการสร้างแบบอ้างอิง
- ส่วนที่ 3 เป็นขั้นตอนในการทดสอบแบบอ้างอิงที่สร้างขึ้น

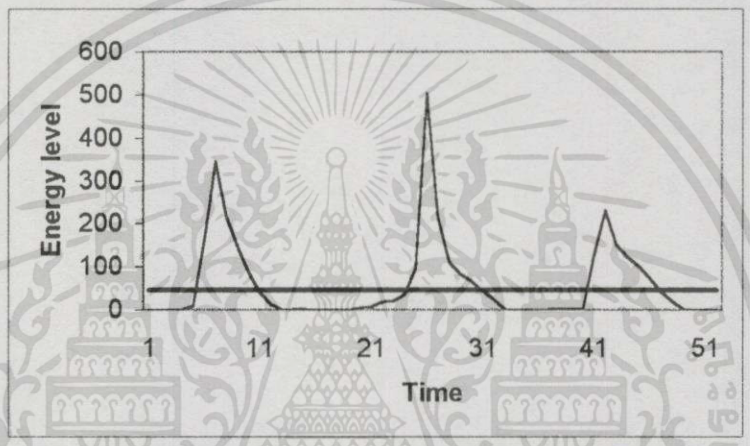
6.2 การกำหนดของเขตของพยางค์

สัญญาณเสียงพูดที่ใช้ในวิทยานิพนธ์นี้เป็นคำพยางค์เดียวที่ได้จากการเก็บตัวอย่างข้อมูลเสียงโดยใช้เครื่องคอมพิวเตอร์ส่วนบุคคล และ การ์ดเสียง(Sound Blaster AWE64) ซึ่งข้อมูลจะถูกบันทึกอยู่ในรูปแบบของไฟล์ข้อมูล ".wav" ซึ่งข้อมูล 1 ตัวอย่างของเสียงจะถูกแทนด้วยข้อมูลขนาด 8 บิท โดยใช้ความถี่ในการแซมปลิ้ง เสียงเท่ากับ 11.025 kHz และไฟล์ข้อมูล ".wav" นี้จะถูกใช้เป็นข้อมูลอินพุทสำหรับการคำนวณของโปรแกรมที่เขียนขึ้น โดยในวิทยานิพนธ์นี้เลือกใช้โปรแกรม Borland C++ Builder ในการสร้างและพัฒนาโปรแกรมต่างๆที่ใช้ในการรู้จำเสียงพูด

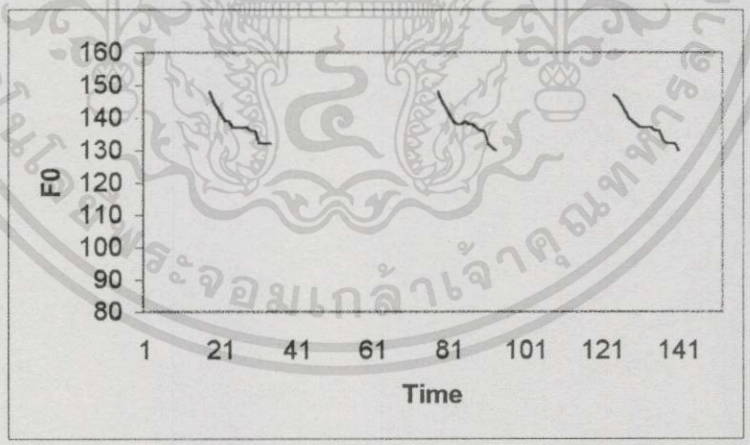
จากที่กล่าวมาแล้วว่าหน่วยเสียงสระมีลักษณะเป็นเสียงก้อง (Voice) และมีลักษณะสัญญาณเป็นคาบ ซึ่งเป็นลักษณะที่เห็นได้ชัดในโดเมนของเวลา ดังนั้นการกำหนดขอบเขตของคำเพื่อที่จะวิเคราะห์หน่วยเสียงสระ จึงกำหนดขอบเขตภายในช่วงที่มีความเป็นคาบ[20]ทั้งหมดของคำหรือพยางค์นั้นๆ รวมถึงการใช้ค่าระดับพลังงานร่วมด้วย โดยเริ่มต้นจากสัญญาณเสียงที่ถูกเก็บมาในลักษณะที่ให้ผู้พูดคำโดดโดยแต่ละคำมีช่วงห่างกันเล็กน้อยดังรูปที่ 6.1(a) และจึงทำการคำนวณค่าพลังงานของเสียงได้ดังรูปที่ 6.1(b) แล้วนำค่าเฉลี่ยของพลังงานทั้งหมดที่แสดงโดยเส้นแนวนอนในรูปที่ 6.1(b) เป็นค่าในการกำหนดขอบเขตของคำเบื้องต้น แล้วจึงคำนวณพิทช์ของสัญญาณเสียง ได้ดังรูปที่ 6.1(c) แล้วใช้บริเวณที่มีค่าพิทช์เป็นขอบเขตของเสียงอีกครั้งหนึ่ง ซึ่งขอบเขตของเสียงที่จะนำไปใช้ในการวิเคราะห์นั้นจะใช้เฉพาะส่วนที่อยู่ในทั้งสองขอบเขตซึ่งแสดงในรูปที่ 6.1(d) เส้นขอบเขตสีเข้มเป็นขอบเขตที่ได้จากการใช้ค่าพลังงาน ส่วนเส้นสีอ่อนเป็นขอบเขตที่ได้จากการมีความถี่มูลฐาน และจะทำให้ได้มาซึ่งส่วนของเสียงสระที่จะถูกตัดออกมาเพื่อใช้ในการวิเคราะห์ดังรูปที่ 6.1(e)



(a) สัญญาณเสียง "กิ" "สิ" และ "บิ"

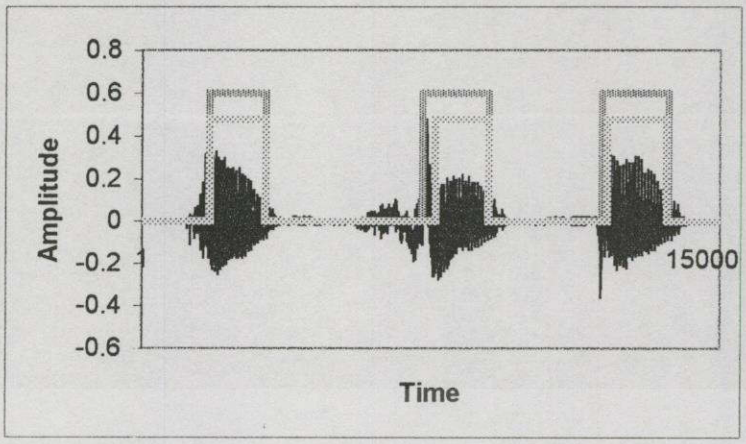


(b) ค่าพลังงานและค่าเฉลี่ยพลังงานที่ใช้เป็นระดับในการตัดขอบเขต

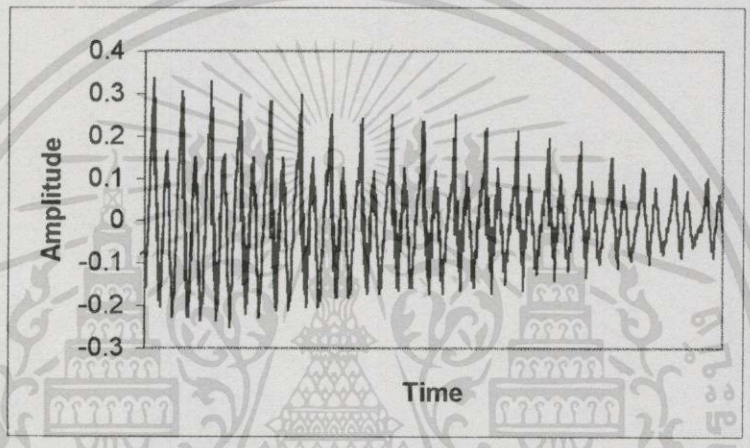


(c) ความถี่มูลฐาน หรือ พิทช์ ของสัญญาณเสียง

รูปที่ 6.1 การตัดคำโดยใช้ค่าพลังงานและความถี่มูลฐาน



(d) ขอบเขตของทั้งสองวิธี โดยบริเวณที่ใช้คือบริเวณที่มีการทับซ้อนกันของทั้งสองวิธี



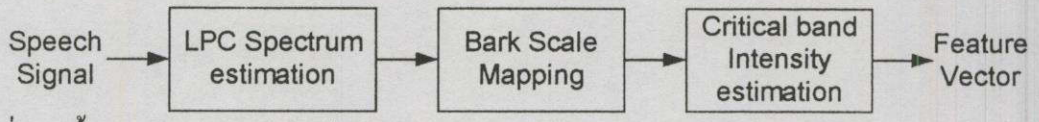
(e) สัญญาณเสียง "ก" ที่ตัดออกมาได้

รูปที่ 6.1 (ต่อ)

6.3 ขั้นตอนในการวิเคราะห์ และ พัฒนาอัลกอริทึมในการสร้างแบบอ้างอิง

แบ่งออกเป็น 3 ขั้นตอนตามรูปที่ 6.2 ดังนี้

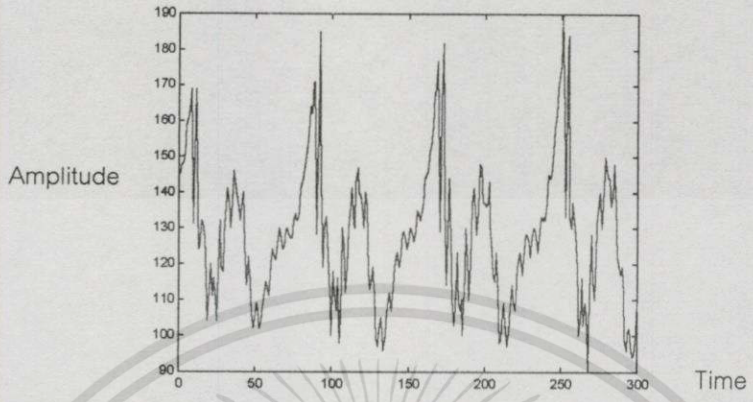
1. การหาสเปกตรัม LPC
2. การแปลงจากสเกลความถี่ ไปเป็นสเกลบาร์ก หรือ Critical Band Rate scale และการคำนวณ Critical Band Intensity
3. ใช้ Critical Band Intensity เป็น Feature Vector ในการสร้างแบบอ้างอิง โดยใช้โครงสร้างข้อมูลแบบ kd-tree



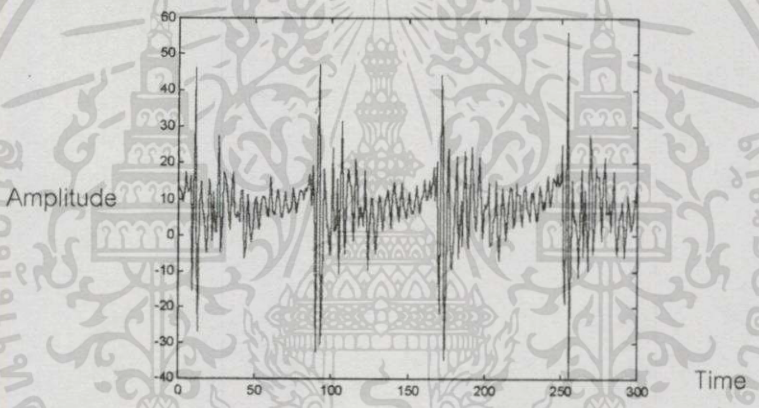
รูปที่ 6.2 ขั้นตอนในการวิเคราะห์

6.3.1 การหาสเปกตรัมแอลพีซี

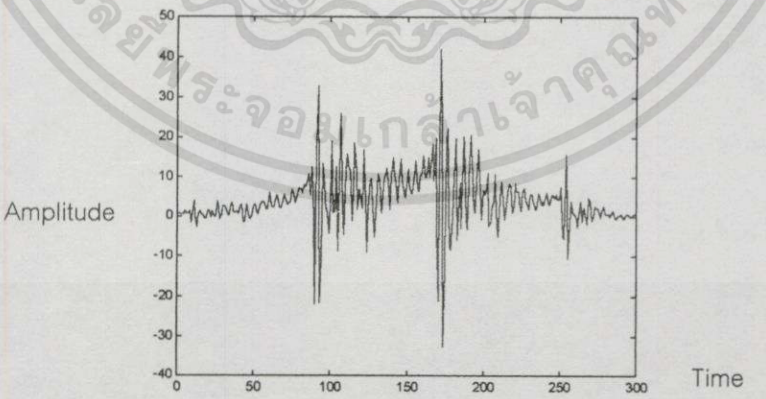
การหาสเปกตรัมแอลพีซีมีขั้นตอนดังที่กล่าวมาแล้วในบทที่ 3 โดยรูปที่ 6.3 (a)-(e) เป็นตัวอย่างสัญญาณที่เข้ามาและสัญญาณในขั้นตอนต่างๆจนถึงสเปกตรัมแอลพีซีที่ได้



(a) สัญญาณที่ถูกจัดแบ่งเป็นเฟรม (สัญญาณเฟรมแรกของรูปที่ 6.1b)

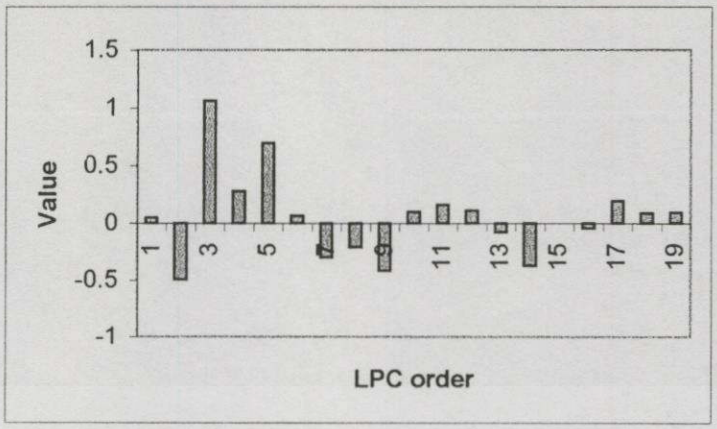


(b) สัญญาณเสียงที่ผ่านการปรีเอมฟาซีส

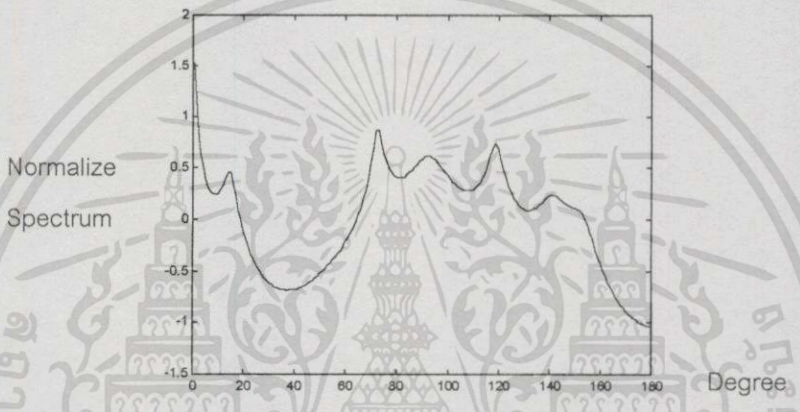


(c) สัญญาณที่ผ่านฟังชันวินโดว์แบบแฮมมิ่ง

รูปที่ 6.3 สัญญาณในขั้นตอนต่างๆในการคำนวณสเปกตรัมแอลพีซี



(d) ค่าสัมประสิทธิ์ที่ได้จากการคำนวณแอลพีซีอันดับ 19



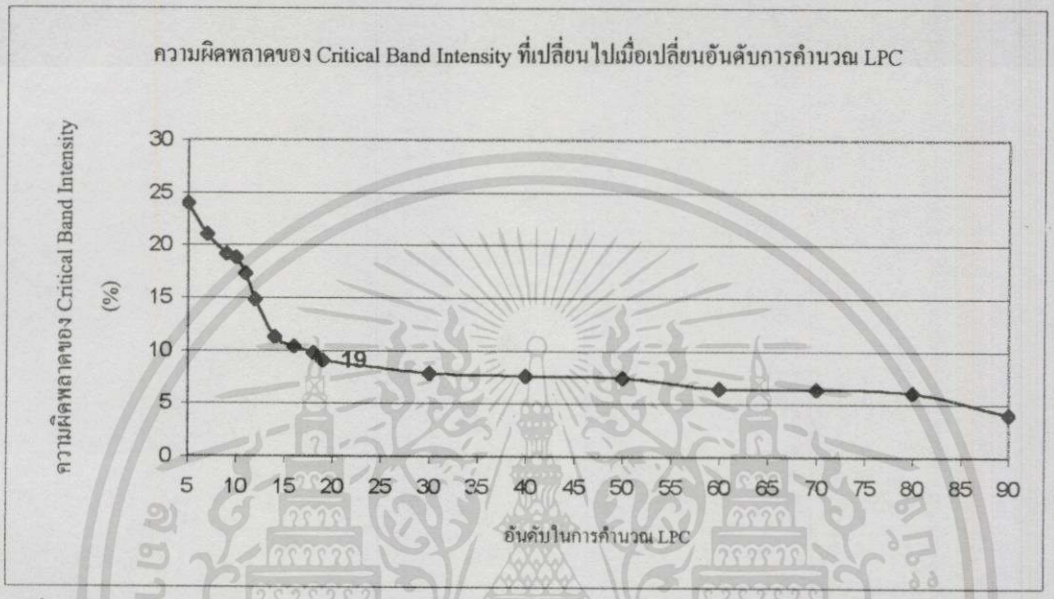
(e) สเปกตรัมแอลพีซีที่ได้

รูปที่ 6.3 (ต่อ)

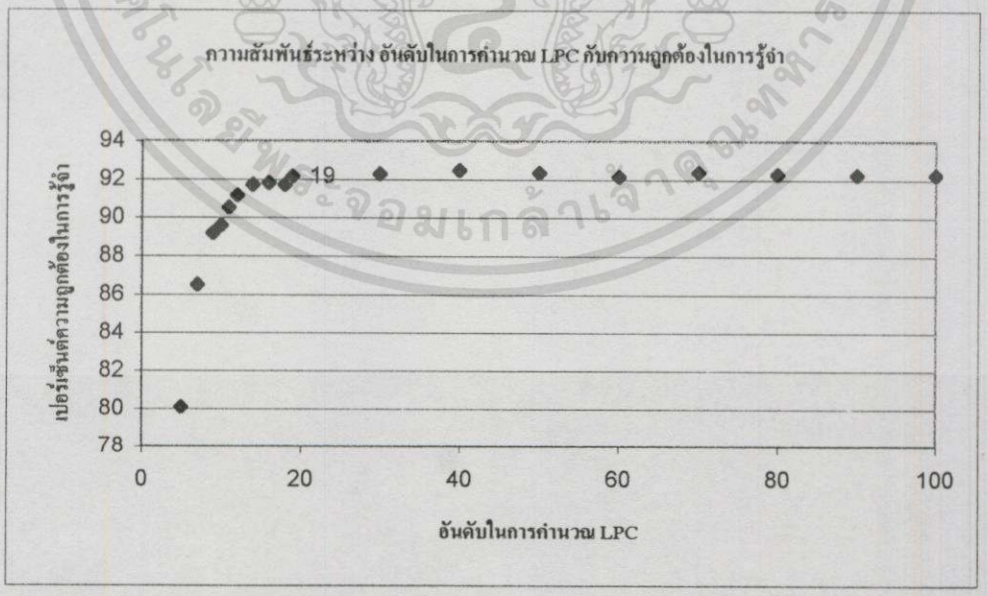
จากรูปที่ 6.3(a) เป็นสัญญาณเสียงที่เราต้องการจะทำการหาสเปกตรัมแอลพีซีโดยเริ่มต้นจากทำการแบ่งช่วงสัญญาณออกเป็นเฟรมละ 300 samples และการวิเคราะห์แต่ละเฟรมจะขยับไปครั้งละ 100 samples ตามรูปที่ 6.3(a) หลังจากนั้นเป็นการปรีเอมฟาซีสให้ความถี่สูงมีขนาดสูงขึ้น ซึ่งจะได้เป็นรูปที่ 6.3(b) แล้วจึงจะทำการวินโดว์เพื่อลดความไม่ต่อเนื่องของสัญญาณที่บริเวณขอบโดยการใช้แฮมมิ่งวินโดว์ซึ่งจะได้ผลดังรูปที่ 6.3(c) แล้วจึงนำสัญญาณที่ผ่านการวินโดว์ แล้วไปทำการคำนวณเพื่อหาสัมประสิทธิ์แอลพีซี ตามที่กล่าวมาแล้วในบทที่ 3 จะได้สัมประสิทธิ์แอลพีซี ในรูปที่ 6.3(d) แล้วจึงนำค่านี้ไปทำการคำนวณหาสเปกตรัมแอลพีซี ได้ผลดังรูปที่ 6.3(e)

สำหรับอันดับในการคำนวณแอลพีซีนั้น หากใช้อันดับต่ำก็จะทำให้การคำนวณเร็วขึ้น แต่ก็จะทำให้สเปกตรัมแอลพีซี แสดงรายละเอียดน้อยลง และส่งผลให้การคำนวณ Critical Band Intensity ผิดพลาดด้วย แต่ถ้าใช้การคำนวณแอลพีซีที่อันดับสูงมากก็จะทำให้การคำนวณซับซ้อนเกินความจำเป็น ดังนั้นจึงได้ทำการทดลองเพื่อหาความสัมพันธ์ระหว่าง ความผิดพลาดของการคำนวณ Critical Band Intensity กับอันดับของการคำนวณแอลพีซี ในรูปที่ 6.4 และความถูกต้องใน

การรู้จำกับอันดับที่ใช้ในการคำนวณแอลพีซีในรูปที่ 6.5 ซึ่งผลทั้งหมดนี้เป็นค่าเฉลี่ยจากตัวอย่างเสียงของผู้ชาย 5 คนและผู้หญิง 5 คน ออกเสียงสระเสียงเดี่ยวทั้งเสียงสั้นและเสียงยาว 18 เสียง ผสมกับเสียงพยัญชนะ 21 เสียง และผันเสียงวรรณยุกต์อีก 5 เสียง รวมเป็นเสียงตัวอย่างทั้งหมด 10x18x21x5=18,900 เสียง ซึ่งเห็นได้ว่าทั้งสองรูปให้ผลที่สอดคล้องกันคือการคำนวณแอลพีซีที่ก่อให้เกิดประสิทธิภาพสูงสุดควรอยู่ที่อันดับที่ 19



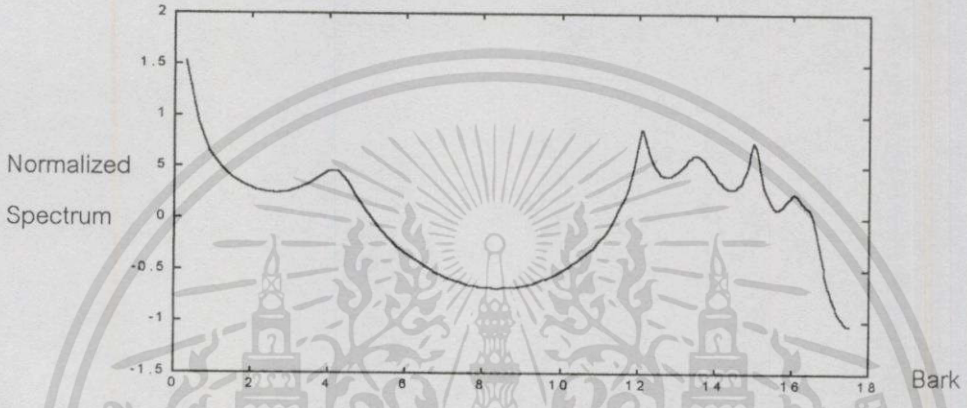
รูปที่ 6.4 ความสัมพันธ์ระหว่างอันดับที่ใช้ในการคำนวณแอลพีซีและความผิดพลาดที่เกิดขึ้นในการคำนวณ Critical Band Intensity



รูปที่ 6.5 ความสัมพันธ์ระหว่างอันดับในการคำนวณแอลพีซีและความถูกต้องในการรู้จำ

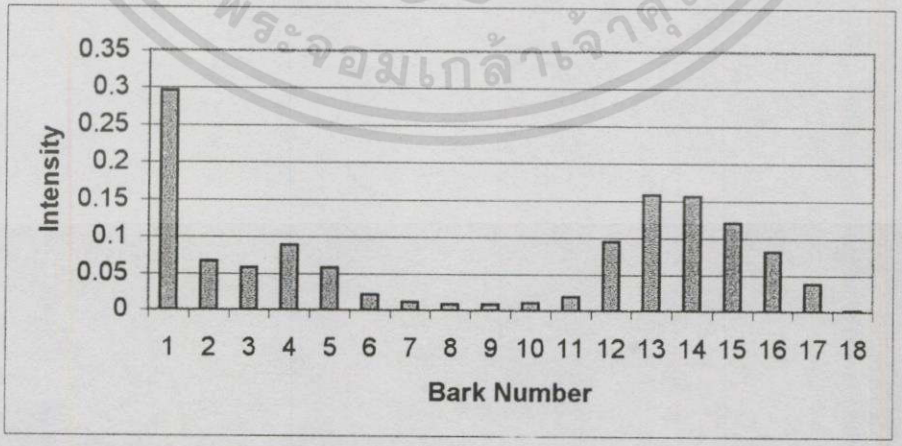
6.3.2 การแปลงสเปกตรัมแอลพีซีไปบนสเกลบาร์ก

จากที่กล่าวผ่านมาในบทที่ 4 สเกลบาร์กหรือ Critical Band Rate scale นั้นช่วยให้สามารถแสดงคุณสมบัติของการตอบสนองต่อความถี่ต่างๆของการได้ยินของมนุษย์ได้เป็นเชิงเส้นตลอดช่วงความถี่ที่มนุษย์สามารถได้ยิน ดังนั้นเราจึงทำการแสดงสเปกตรัมแอลพีซีบนสเกลบาร์ก และใช้ความเข้มของสัญญาณในแต่ละแถบความถี่วิกฤตเป็นองค์ประกอบของเวคเตอร์แทนหน่วยเสียง ซึ่งจากสเปกตรัมในโดเมนความถี่จากรูปที่ 6.3(e) นั้นเราสามารถแสดงบนสเกลบาร์กได้ดังรูปที่ 6.6



รูปที่ 6.6 สเปกตรัมแอลพีซีของรูปที่ 6.3(e) บนสเกลบาร์ก

เมื่อเทียบรูปที่ 6.6 และรูปที่ 6.3(e) จะเห็นได้ว่าในช่วงสัญญาณที่อยู่ในช่วงความถี่ 5.5 kHz นั้นจะอยู่ใน บาร์กที่ 18 ซึ่งแสดงว่าจำนวณแถบความถี่ที่ที่สามารถนำมาใช้ในการรู้จำมีทั้งหมด 18 แถบความถี่ หลังจากนั้นเราทำการคำนวณความเข้มในแต่ละแถบความถี่วิกฤตทั้ง 18 แถบความถี่วิกฤต ได้ตามรูปที่ 6.7

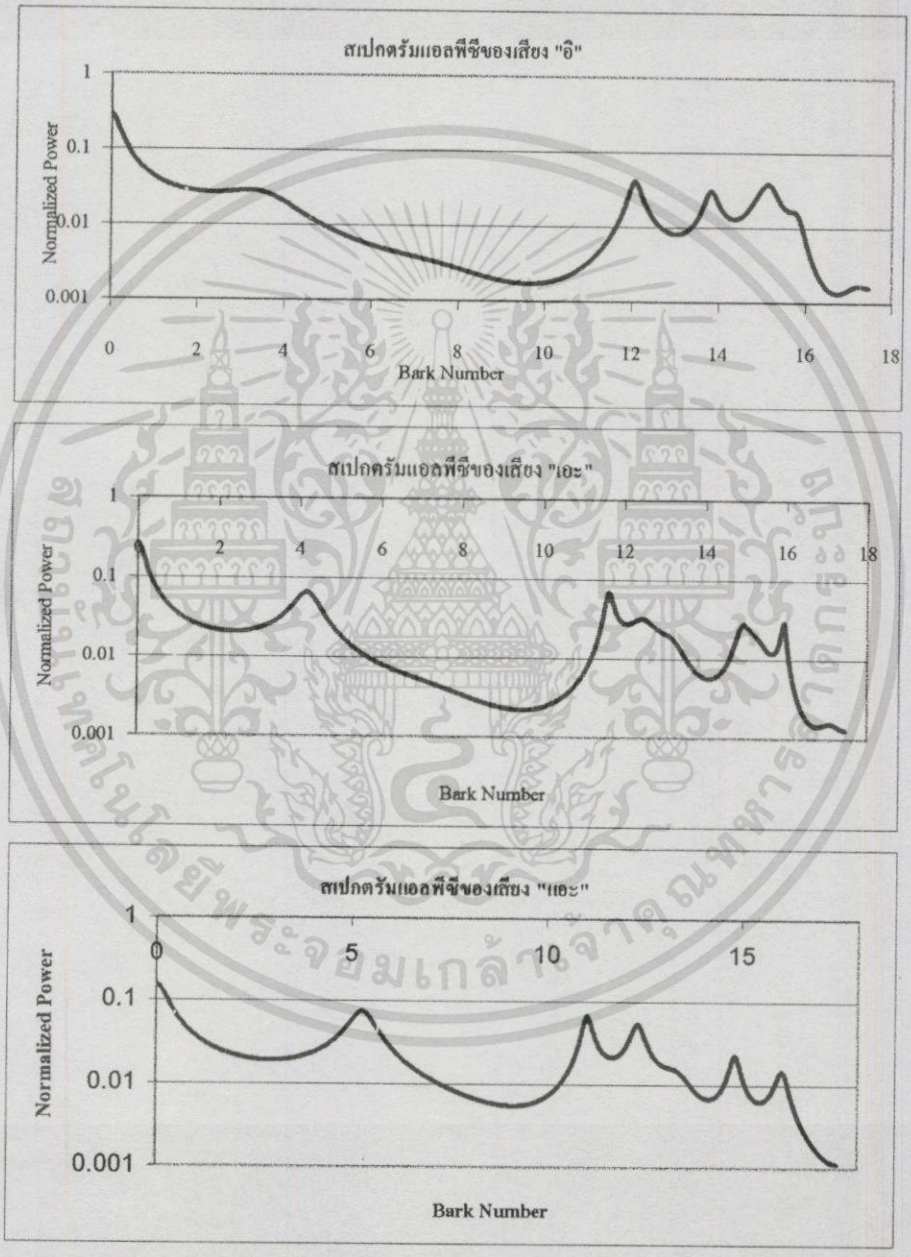


รูปที่ 6.7 ความเข้มสัญญาณในแต่ละแถบความถี่วิกฤต

ความเข้มสัญญาณในแถบความถี่วิกฤตทั้ง 18 ค่าสำหรับสัญญาณเสียงแต่ละเฟรมนั้นเราสามารถนำไปใช้เป็นองค์ประกอบของ Feature Vector เพื่อที่จะใช้ในการสร้างแบบอ้างอิง และทดสอบแบบอ้างอิง

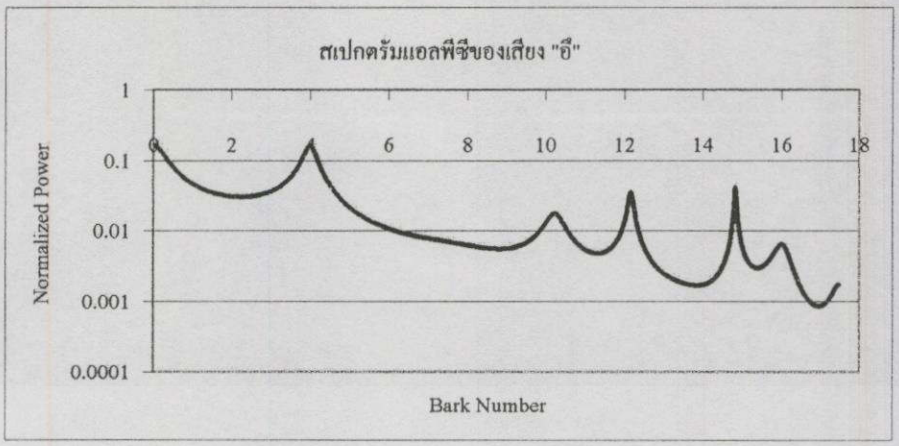
6.3.3 การสร้างและทดสอบแบบอ้างอิงที่ใช้โครงสร้างข้อมูลแบบ kd-tree

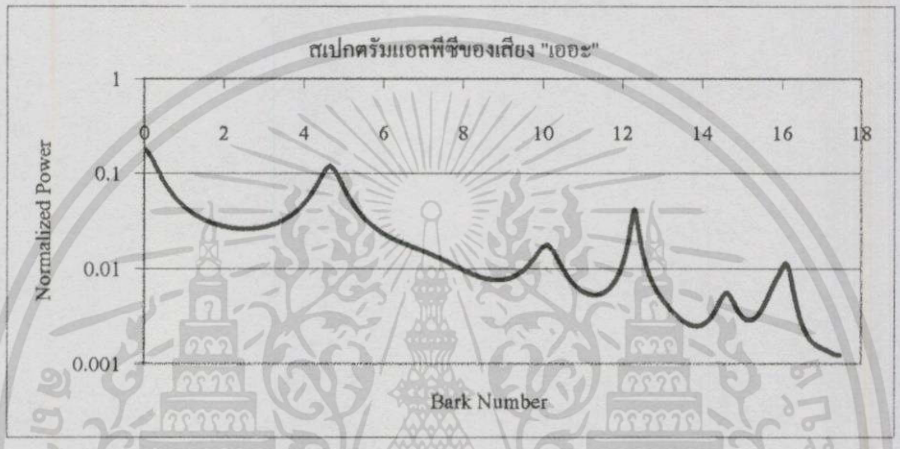
จากหัวข้อที่ผ่านมาเราสามารถคำนวณสเปกตรัมแอลพีซี โดยใช้การคำนวณแอลพีซี อันดับที่ 19 ซึ่งตัวอย่างสเปกตรัมแอลพีซี ของแต่ละเสียงสระจะมีลักษณะดังนี้

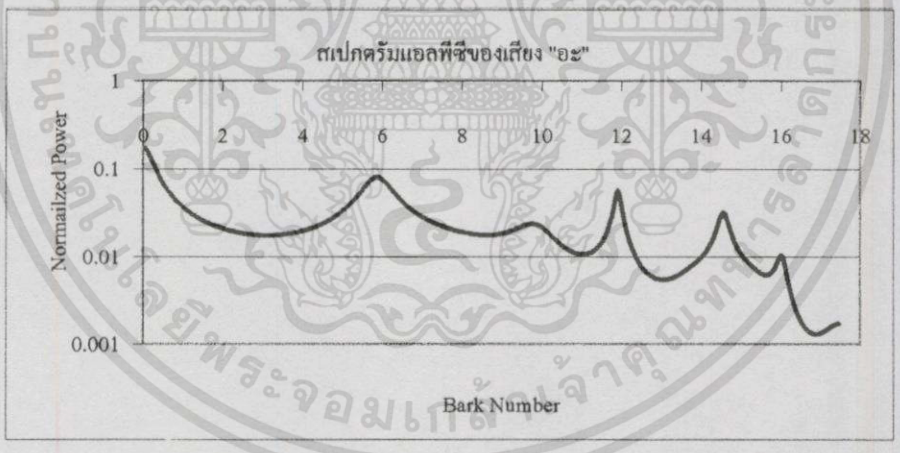


(a) สระหน้า

รูปที่ 6.8 สเปกตรัมแอลพีซีของเสียงสระทั้ง 9 เสียง

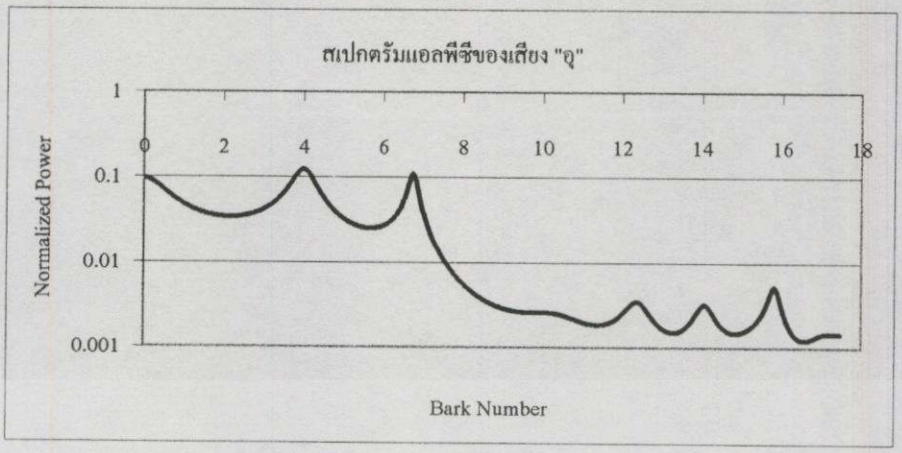


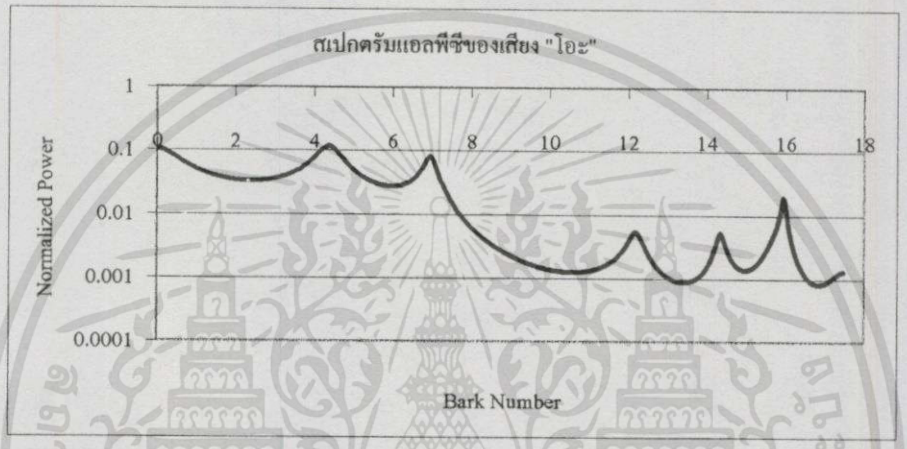


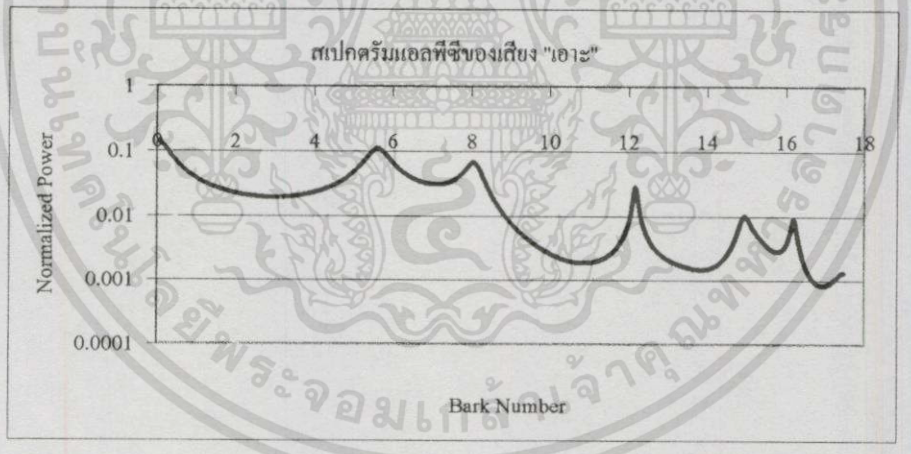


(b) สระกลาง

รูปที่ 6.8 (ต่อ)



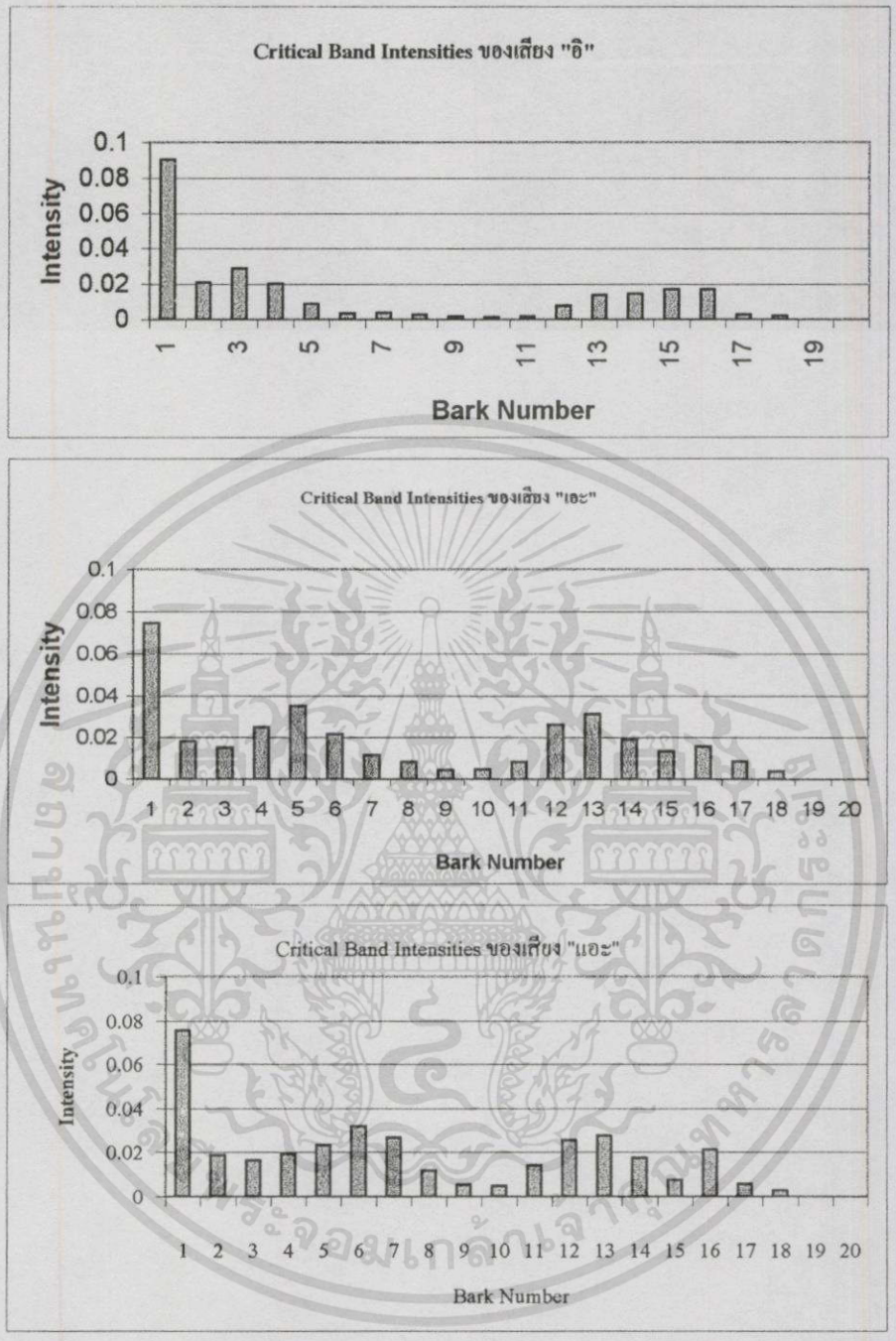




(c) สระหลัง

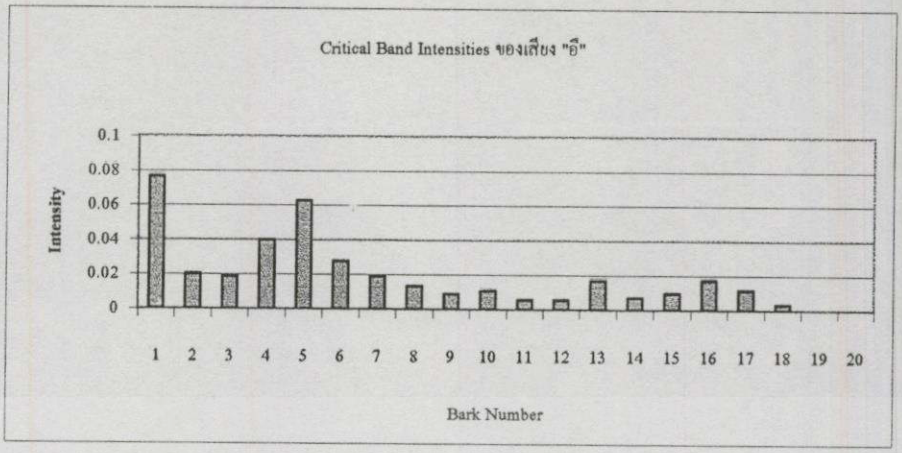
รูปที่ 6.8 (ต่อ)

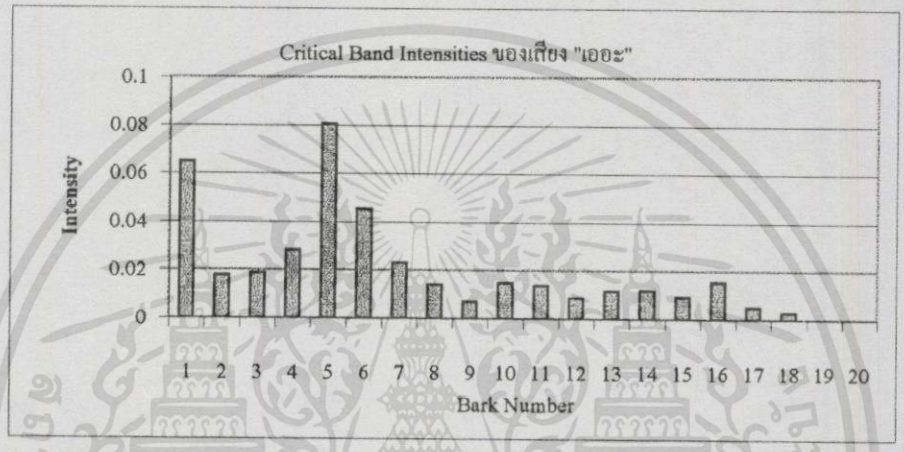
จากรูปที่ 6.8(a) **เป็นสเปกตรัมแอลพีซีของเสียงสระ “อิ”, “เอะ” และ “แอะ” เป็นกลุ่มของสระหน้า ส่วนรูปที่** 6.8(b) **เป็นเป็นสเปกตรัมแอลพีซีของเสียงสระ “อึ”, “เออะ” และ “อะ” เป็นกลุ่มของสระกลาง และรูปที่** 6.8(c) **เป็นสเปกตรัมแอลพีซีของเสียงสระ “อุ”, “โอะ” และ “เอาะ” เป็นกลุ่มของสระหลัง ซึ่งจะเห็นได้ชัดว่ามีความแตกต่างระหว่างกลุ่มเสียงสระมากกว่าภายในกลุ่มเสียงสระเดียวกัน แต่ภายในกลุ่มเสียงสระเดียวกันก็ยังคงมีความแตกต่างในรายละเอียดของแต่ละเสียงสระ และเมื่อเรานำสเปกตรัมแอลพีซีนี้ มาคำนวณหา** Critical Band Intensity **ได้ดังรูปที่** 6.9

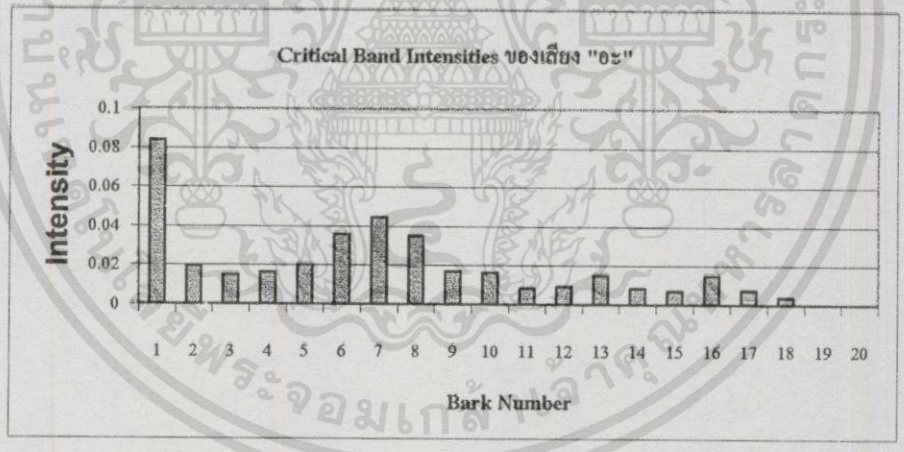


(a) สระหน้า

รูปที่ 6.9 Critical Band Intensity ทั้ง 18 ค่าของสเปกตรัมแอลพีซีของเสียงสระทั้ง 9 เสียง

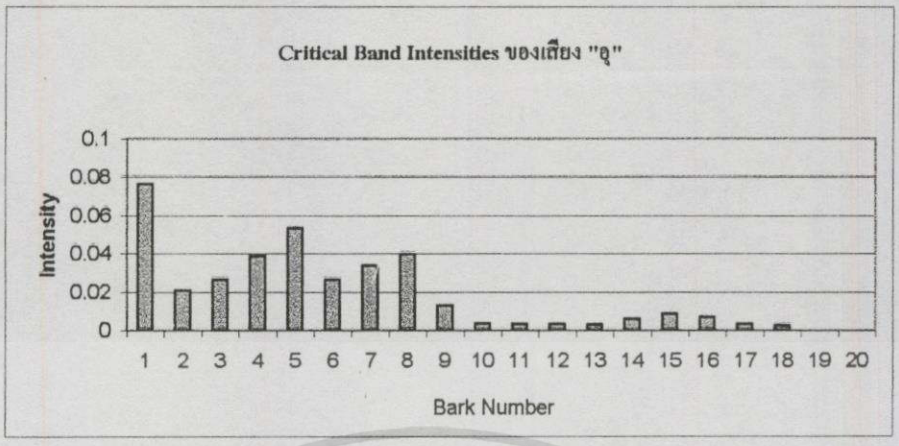


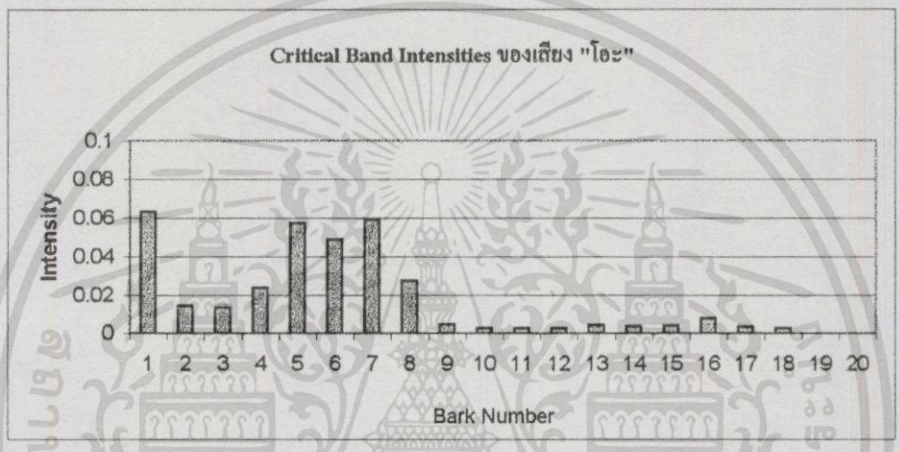


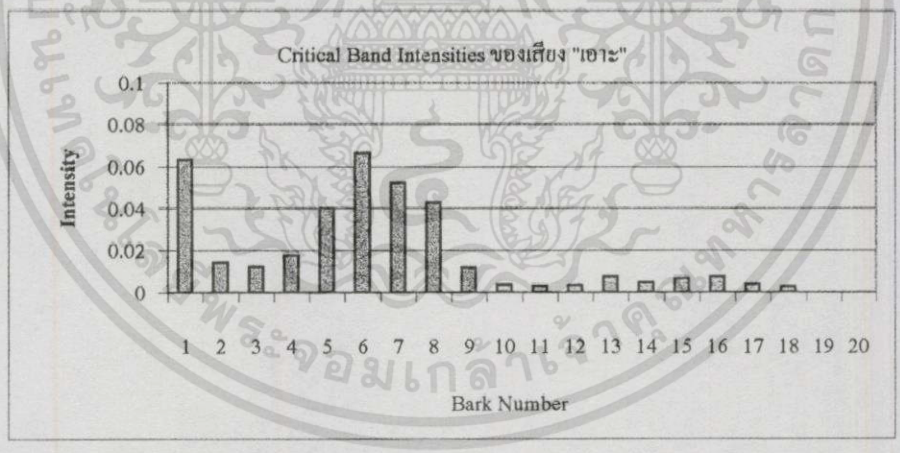


(b) สระกลาง

รูปที่ 6.9 (ต่อ)







(c) สระหลัง

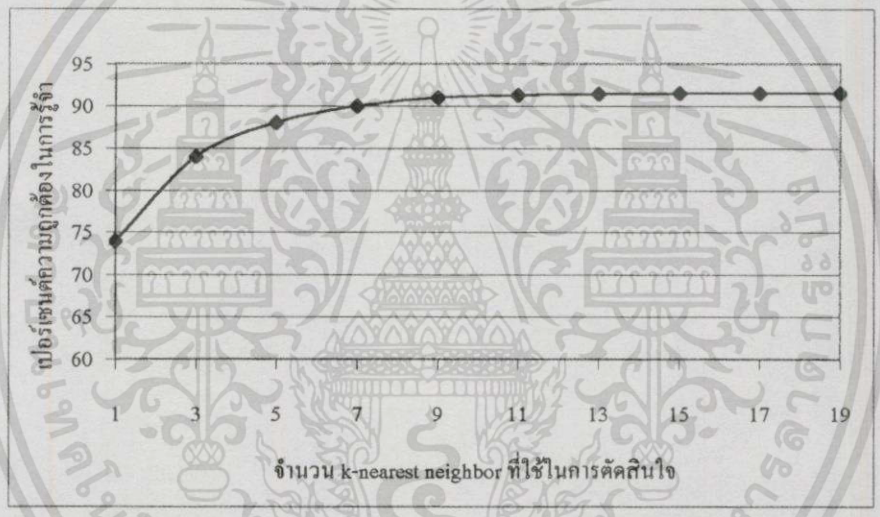
รูปที่ 6.9 (ต่อ)

Critical Band Intensity ซึ่งแสดงเป็นตัวอย่างในรูปที่ 6.9 นี้เป็นสิ่งที่เราใช้เป็นตัวแทนของแต่ละเฟรมของสัญญาณเสียงสระในการสร้างแบบอ้างอิงและรู้จำ ซึ่งเราสามารถสังเกตุเห็นได้ว่า มีความแตกต่างกันระหว่างเสียงสระซึ่งจะทำให้สามารถแยกแยะเสียงสระทั้งหมดออกจากกันได้

ในหัวข้อนี้เราจึงนำความแตกต่างของ Critical Band Intensity 14 ค่าของเสียงสระแต่ละเสียงมาใช้เป็นเวคเตอร์ 14 มิติ สร้างเป็นแบบอ้างอิงโดยใช้โครงสร้างข้อมูลแบบ kd-tree และ

ทดสอบแบบอ้างอิงด้วยการค้นหา k-Nearest Neighbor ใน kd-tree ดังที่กล่าวผ่านมาแล้วในบทที่ 5 โดยที่ตัวอย่างเสียงที่ใช้ทั้งหมดบันทึกจากผู้พูดที่เป็นชาย 10 คน และ หญิง 10 คน โดยผู้พูดมีอายุระหว่าง 19-27 ปี ซึ่งคำที่ใช้ในการเก็บตัวอย่างเสียงพูดเกิดจากการผสมคำจาก สระเสียงเดี่ยวทั้งสั้นและยาวจำนวน 18 เสียง และ เสียงวรรณยุกต์จำนวน 5 เสียง กับ เสียงพยัญชนะอีก 21 เสียง รวมทั้งสิ้นเป็น 20x18x5x21=37,800 เสียง

สำหรับการทดสอบแบบอ้างอิงโดยใช้ การค้นหา k-Nearest Neighbor จาก kd-tree นั้นหากใช้ค่า k ที่น้อยเกินไปก็อาจทำให้ผลการตัดสินใจของ k-Nearest Neighbor ผิดพลาดได้มากกว่าการใช้ค่า k ที่เหมาะสม ดังนั้นจึงทำการทดลองเพื่อหาค่า k ที่เหมาะสมโดยใช้ตัวอย่างเสียงทั้งหมด 37,800 เสียงในการทดสอบโดยผลที่ได้ในรูปที่ 6.10 แสดงให้เห็นว่าค่า k=9 สำหรับการตัดสินใจโดยวิธี k-Nearest Neighbor สำหรับกรณีนี้มีประสิทธิภาพสูงสุด



รูปที่ 6.10 ความสัมพันธ์ระหว่างจำนวนของ k-Nearest Neighbor ที่ใช้ในการตัดสินใจกับความถูกต้องในการรู้จำที่ได้รับ

เมื่อได้พารามิเตอร์ต่างๆที่ทำให้เกิดประสิทธิภาพสูงสุดในการรู้จำแล้วจึงทำการสร้างและทดสอบแบบอ้างอิง 3 แบบคือ

1. แบบอ้างอิงสำหรับผู้พูดคนเดียว (Speaker Dependent Reference set) ใช้เสียงตัวอย่างจากผู้พูดคนเดียวจำนวน 1,890 เสียง ในการสร้างแบบอ้างอิง และ ทดสอบ ซึ่งจะทำทั้งเสียงผู้ชายและผู้หญิงอย่างละ 1 แบบอ้างอิง โดยรายละเอียดผลการทดสอบความถูกต้องในการรู้จำแสดงในตารางที่ 6.1
2. แบบอ้างอิงสำหรับผู้พูดเพศเดียวกัน (Gender Dependent Reference set) ใช้เสียงตัวอย่างจากผู้พูดที่เป็นผู้ชายทั้งหมดในการสร้างแบบอ้างอิงสำหรับเพศชาย และ เสียงตัวอย่างจากผู้พูดที่เป็นหญิงทั้งหมดในการสร้างแบบอ้างอิงสำหรับเพศหญิง ดังนั้นจะมีตัวอย่างเสียง

แบบอ้างอิงละ 18,900 เสียง โดยผลการทดสอบความถูกต้องในการรู้จำแสดงในตารางที่ 6.2

3. แบบอ้างอิงสำหรับผู้พูดหลายคน (Speaker Independent Reference set) ใช้เสียงตัวอย่างจากผู้พูดทั้งหมด 37,800 เสียงในการสร้างแบบจำลอง โดยผลการทดสอบความถูกต้องในการรู้จำแสดงในตารางที่ 6.3

ตารางที่ 6.1 ผลการทดสอบแบบอ้างอิงสำหรับผู้พูดคนเดียว

แบบอ้างอิง		สระหน้า			สระกลาง			สระหลัง			ความถูกต้อง
แบบทดสอบ		"อิ"	"เอะ"	"แอะ"	"อึ"	"เออะ"	"อะ"	"อุ"	"โอะ"	"เอาะ"	
สระหน้า	"อิ"	410	5	5	0	0	0	0	0	0	97.56
	"เอะ"	8	408	4	0	0	0	0	0	0	97.26
	"แอะ"	8	4	404	2	2	0	0	0	0	96.31
สระกลาง	"อึ"	2	0	0	401	10	6	0	0	2	95.36
	"เออะ"	0	0	0	9	405	6	0	0	0	96.49
	"อะ"	0	0	2	7	12	394	4	0	2	93.81
สระหลัง	"อุ"	0	0	0	0	0	2	401	11	7	95.42
	"โอะ"	0	0	0	0	4	0	13	396	7	94.40
	"เอาะ"	0	2	2	4	0	0	4	12	398	94.65
										เฉลี่ย	95.69

(a) ผู้พูดเพศชาย

แบบอ้างอิง		สระหน้า			สระกลาง			สระหลัง			ความถูกต้อง
แบบทดสอบ		"อิ"	"เอะ"	"แอะ"	"อึ"	"เออะ"	"อะ"	"อุ"	"โอะ"	"เอาะ"	
สระหน้า	"อิ"	415	0	5	0	0	0	0	0	0	98.75
	"เอะ"	11	403	4	4	0	0	0	0	0	95.83
	"แอะ"	11	5	401	2	2	0	0	0	0	95.42
สระกลาง	"อึ"	2	0	0	408	7	2	0	0	2	97.08
	"เออะ"	0	0	0	14	397	4	5	0	0	94.58
	"อะ"	0	0	5	9	14	387	4	0	2	92.08
สระหลัง	"อุ"	0	0	0	0	0	2	411	4	4	97.92
	"โอะ"	0	0	0	0	4	0	11	404	2	96.25
	"เอาะ"	0	2	2	9	0	4	4	16	385	91.67
										เฉลี่ย	95.51

(b) ผู้พูดเพศหญิง

จากผลการทดสอบแบบอ้างอิงสำหรับผู้พูดคนเดียวในตารางที่ 6.1(a),(b) ได้ความถูกต้องในการรู้จำเฉลี่ยประมาณ 95 เปอร์เซ็นต์ ซึ่งสูงที่สุดใน 3 แบบอ้างอิงที่สร้างและรายละเอียดในตารางนั้นแถวแสดงถึงตัวอย่างเสียงที่นำไปใช้เป็นสัญญาณเสียงที่ไม่ทราบค่าในการทดสอบ ส่วน

คอลัมน์เป็นผลการรู้จำที่ได้จากแบบอ้างอิง และตัวเลขในช่องคือจำนวนของสัญญาณเสียงที่นำเข้าทดสอบถูกรู้จำว่าเป็นเสียงสระใด ซึ่งเราสามารถเห็นได้อย่างชัดเจนว่า ความผิดพลาดในการรู้จำที่เกิดขึ้นนั้นส่วนใหญ่เกิดภายในกลุ่มสระ หน้า กลาง และ สระหลัง ซึ่งมีความคล้ายคลึงกันมากกว่าความผิดพลาดที่เกิดขึ้นข้ามกลุ่มของสระมาก

ตารางที่ 6.2 ผลการทดสอบแบบอ้างอิงสำหรับผู้พูดเพศเดียวกัน

แบบอ้างอิง		สระหน้า			สระกลาง			สระหลัง			ความถูกต้อง
แบบทดสอบ		"อิ"	"เอะ"	"แอะ"	"อึ"	"เออะ"	"อะ"	"อุ"	"โอะ"	"เอาะ"	
สระหน้า	"อิ"	2020	20	34	0	9	0	18	0	0	96.18
	"เอะ"	28	2005	49	0	18	0	0	0	0	95.50
	"แอะ"	53	30	2003	5	9	0	0	0	0	95.40
สระกลาง	"อึ"	9	9	0	1951	69	36	0	9	18	92.92
	"เออะ"	0	0	9	49	1990	52	0	0	0	94.77
	"อะ"	0	0	9	40	81	1927	35	0	9	91.75
สระหลัง	"อุ"	0	0	0	0	0	9	1991	66	35	94.79
	"โอะ"	0	0	0	0	44	14	83	1893	66	90.15
	"เอาะ"	0	9	9	18	0	0	36	56	1973	93.96
										เฉลี่ย	93.94

(c) ผู้พูดเพศชาย

แบบอ้างอิง		สระหน้า			สระกลาง			สระหลัง			ความถูกต้อง
แบบทดสอบ		"อิ"	"เอะ"	"แอะ"	"อึ"	"เออะ"	"อะ"	"อุ"	"โอะ"	"เอาะ"	
สระหน้า	"อิ"	2001	33	47	20	0	0	0	0	0	95.30
	"เอะ"	80	1928	54	21	18	0	0	0	0	91.80
	"แอะ"	73	41	1965	12	9	0	0	0	0	93.58
สระกลาง	"อึ"	9	0	2	1991	46	44	0	0	9	94.79
	"เออะ"	0	0	0	100	1961	23	17	0	0	93.38
	"อะ"	0	0	47	51	83	1893	18	0	9	90.15
สระหลัง	"อุ"	0	0	0	0	26	9	1975	70	21	94.05
	"โอะ"	0	0	0	0	18	26	71	1965	20	93.58
	"เอาะ"	0	9	9	44	0	18	36	99	1886	89.82
										เฉลี่ย	92.94

(d) ผู้พูดเพศหญิง

ผลการทดลองในตารางที่ 6.2(a),(b) นี้แสดงในลักษณะเดียวกับตารางที่ 6.1 โดยได้ความถูกต้องในการรู้จำเฉลี่ยของการใช้แบบอ้างอิงสำหรับผู้พูดเพศเดียวกัน ประมาณ 93 เปอร์เซ็นต์ซึ่งต่ำกว่าผลที่ได้จากการทดลองแรกเพียงประมาณ 2 เปอร์เซ็นต์เท่านั้น ส่วนความผิดพลาดในการรู้

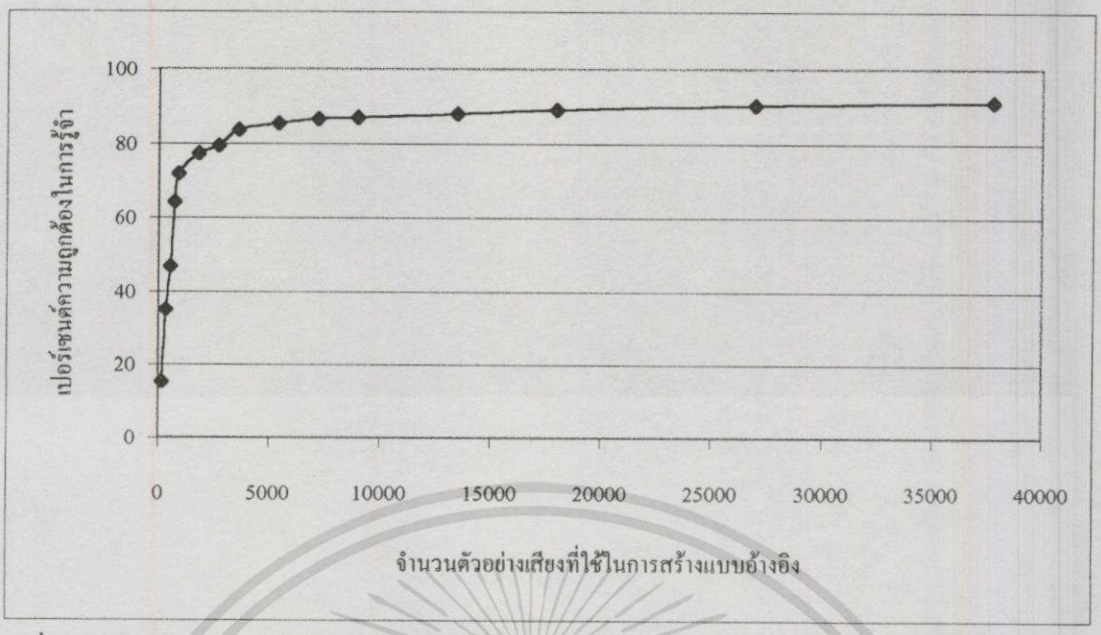
จำที่เกิดขึ้นส่วนใหญ่ก็ยังคงอยู่ในกลุ่มของสระ มากกว่าข้ามกลุ่มสระ แต่มีบางสระที่เริ่มมีการข้ามกลุ่มของสระที่เห็นได้ชัด

ตารางที่ 6.3 ผลการทดสอบแบบอ้างอิงสำหรับหลายผู้พูด

แบบอ้างอิง		สระหน้า			สระกลาง			สระหลัง			ความถูกต้อง
แบบทดสอบ		"อิ"	"เอะ"	"แอะ"	"อึ"	"เออะ"	"อะ"	"อุ"	"โอะ"	"เอาะ"	
สระหน้า	"อิ"	4048	21	74	2	18	0	31	7	0	96.38
	"เอะ"	231	3844	86	0	39	0	0	0	0	91.52
	"แอะ"	123	263	3725	18	72	0	0	0	0	88.68
สระกลาง	"อึ"	18	18	0	3541	410	89	0	18	108	84.30
	"เออะ"	0	0	30	151	3952	68	0	0	0	94.10
	"อะ"	0	0	18	76	175	3839	75	0	18	91.40
สระหลัง	"อุ"	0	0	5	0	0	19	3986	130	61	94.90
	"โอะ"	0	0	4	0	150	18	236	3678	115	87.57
	"เอาะ"	0	18	18	35	0	70	49	161	3850	91.67
										เฉลี่ย	91.17

ตารางสุดท้ายนี้สำหรับแบบอ้างอิงแบบหลายผู้พูดซึ่งได้ความถูกต้องเฉลี่ยประมาณ 91 % ซึ่งต่ำกว่าแบบอ้างอิงแรกอยู่ 4 เปอร์เซ็นต์ เนื่องจากเป็นแบบอ้างอิงสำหรับผู้พูดถึง 20 คน รายละเอียดในตารางแสดงให้เห็นว่าเริ่มมีความผิดพลาดเกิดขึ้นข้ามกลุ่มของสระมากขึ้นในบางสระ

สำหรับจำนวนตัวอย่างเสียงที่ใช้สำหรับแบบอ้างอิงหลายผู้พูดนั้นจำนวนตัวอย่างเสียงที่ใช้สร้างแบบอ้างอิงมีความสำคัญมาก เนื่องจากหากมีตัวอย่างเสียงอยู่ในแบบอ้างอิงน้อยเกินไปการใช้ k-Nearest Neighbor ในการตัดสินใจอาจเกิดความผิดพลาดได้มาก ดังนั้นจึงทำการทดลองเพื่อแสดงให้เห็นจำนวนตัวอย่างเสียงที่ควรใช้ในการสร้างแบบจำลองเพื่อให้ได้ประสิทธิภาพสูง ดังในรูปที่ 6.11 เป็นการทดลองสร้างแบบอ้างอิงด้วยจำนวนตัวอย่างเสียงต่างๆกันโดยแต่ละผู้พูดใช้จำนวนเสียงเท่าๆกัน แสดงให้เห็นว่าหากใช้จำนวนตัวอย่างเสียงน้อยเกินไปจะทำให้ความถูกต้องในการรู้จำลดลงอย่างมาก และสำหรับผู้พูดจำนวนหนึ่งก็มีจำนวนตัวอย่างเสียงที่พอเพียงในการสร้างแบบจำลองที่มีประสิทธิภาพได้



รูปที่ 6.11 ความสัมพันธ์ระหว่างจำนวนตัวอย่างเสียงที่ใช้ในการสร้างแบบอ้างอิงกับเปอร์เซนต์ความถูกต้องในการรู้จำ

บทที่ 7
สรุปผล และ ข้อเสนอแนะ

วิทยานิพนธ์นี้เป็นการเสนอวิธีการในการรู้จำเสียงสระเสียงเดี่ยวในภาษาไทย ในโดเมนของเวลานั้นเนื่องจากเสียงสระเป็นหน่วยเสียงก้อง ซึ่งมีลักษณะสำคัญคือความเป็นคาบดังนั้นจึงทำการวิเคราะห์เฉพาะส่วนที่มีความถี่มูลฐานชัดเจน โดยในการวิเคราะห์จะทำการวิเคราะห์ในโดเมนความถี่ จากความเข้มในแถบความถี่วิกฤตของสเปกตรัมแอลพีซี การสร้างแบบอ้างอิงจะใช้โครงสร้างข้อมูลแบบ kd-tree และการรู้จำจะใช้การตัดสินใจจาก k-Nearest Neighbor

7.1 การทดลอง

สัญญาณเสียงที่เป็นข้อมูลดิบที่ใช้ในการวิเคราะห์จะถูกนำมาตัดนำเอาเฉพาะส่วนของเสียงสระมาใช้เท่านั้นโดยวิธีการกำหนดขอบเขตของเสียงสระนั้นใช้ทั้งค่าพลังงานเฉลี่ยร่วมกับการปรากฏของความถี่มูลฐาน จากนั้นจะนำส่วนของสัญญาณเสียงสระนั้นไปทำการหาสเปกตรัมแอลพีซีโดยอันดับในการคำนวณแอลพีซีที่ได้ทดลองและพบว่าได้ประสิทธิภาพสูงสุดคือ อันดับที่ 19

หลังจากนั้นจึงนำสเปกตรัมแอลพีซีไปแสดงบนสเกลบาร์ก และทำการคำนวณหาความเข้มสัญญาณในแต่ละความถี่วิกฤตบนสเกลบาร์ก แล้วจึงนำความเข้มสัญญาณของทุกแถบความถี่วิกฤตซึ่งในที่นี้มีจำนวน 18 แถบความถี่วิกฤตมาใช้เป็น Feature Vector

ในการสร้างแบบอ้างอิงจะใช้โครงสร้างข้อมูลแบบ kd-tree ซึ่งมีจำนวนมิติเท่ากับ 18 ตามจำนวนแถบสัญญาณที่มี และในการตัดสินว่าเวคเตอร์ที่ถูกนำเข้ามาทดสอบควรเป็นเวคเตอร์แทนเสียงใดนั้นใช้การตัดสินใจจาก k-Nearest Neighbor ซึ่งค่า K ที่ทดสอบแล้วว่าจะให้ผลดีที่สุดคือ K=9

ส่วนในการสร้างแบบอ้างอิงนั้นได้แบ่งเป็น 3 แบบ คือ แบบอ้างอิงสำหรับผู้พูดคนเดียว แบบอ้างอิงสำหรับผู้พูดเพศเดียวกัน และ แบบอ้างอิงสำหรับผู้พูดหลายคน ซึ่งให้ค่าความถูกต้องในการรู้จำเฉลี่ยประมาณ 95 , 93 และ 91 เปอร์เซ็นต์ตามลำดับ โดยที่ความผิดพลาดที่เกิดขึ้นส่วนใหญ่เกิดขึ้นภายในกลุ่มสระเดียวกัน ซึ่งชัดเจนว่าเสียงสระในกลุ่มเดียวกันจะมีความคล้าย คลึงกันมากกว่าจึงทำให้ความผิดพลาดในการรู้จำไปยังสระในกลุ่มเดียวกันมากกว่าข้ามกลุ่ม

7.2 ข้อสังเกต ปัญหาที่พบในการทดลอง

- การเลือกอันดับที่ใช้การคำนวณหาสัมประสิทธิ์แอลพีซีเพื่อใช้ในการหาสเปกตรัมเพื่อให้เกิดความคลาดเคลื่อนของสเปกตรัมน้อยที่สุดและให้ได้ความถูกต้องในการรู้จำสูงที่สุด ในขณะที่ไม่ก่อให้เกิดความซับซ้อนในการคำนวณมากเกินไปนั้น จะขึ้นอยู่กับกลุ่มตัวอย่างเสียงอยู่มากในลักษณะที่เสียงของผู้ชายโดยทั่วไปมีความถี่ค่อนข้างต่ำจึงต้องการอันดับของการคำนวณที่ต่ำกว่าในขณะที่เสียงของผู้หญิงซึ่งส่วนใหญ่ต้องการอันดับในการคำนวณที่สูงกว่า แต่จากการทดลองแล้วโดยรวมอันดับที่ 19 สามารถครอบคลุมทั้งเสียงผู้ชายและเสียงผู้หญิง
- สำหรับการใช้ประโยชน์จากความเข้มสัญญาณในแถบความถี่วิกฤตนั้น จริงๆแล้วจะเป็นการดีกว่าหากเราสามารถรู้ได้ว่าสำหรับเสียงสระแต่ละเสียงนั้น ความถี่ศูนย์กลางใดที่มีประโยชน์ในการแยกแยะเสียงสระแต่ละเสียงออกจากกัน แทนที่จะใช้ความถี่ศูนย์กลางตามสเกลบาร์ก และรวมถึงอาจทำให้สามารถลดจำนวนแถบสัญญาณที่จะใช้ลงซึ่งจะทำให้การซับซ้อนในการคำนวณลดลงด้วย
- การสร้างแบบอ้างอิงโดยใช้โครงสร้างข้อมูลแบบ kd-tree นั้นเป็นเพียงวิธีหนึ่งในการสร้างฐานข้อมูลที่สามารถเข้าค้นหาข้อมูลได้โดยเร็ว และวิธีที่ใช้ในการเลือก pivot ก็ยังมีอีกหลายวิธีด้วยเช่นกัน
- การตัดสินใจในการรู้จำที่กระทำโดยการลงคะแนนของ Nearest Neighbor จำนวน k ตัวนั้น จำนวน k ที่ก่อให้เกิดประโยชน์ที่สุดนั้นไม่จำเป็นต้องเท่ากับ 9 เสมอไป แต่มันขึ้นกับขนาดของฐานข้อมูล ถ้าหากฐานข้อมูลมีขนาดเล็กการใช้ค่า k มากก็อาจก่อให้เกิดความสับสนในการตัดสินใจได้

7.3 ข้อเสนอแนะ

เสียงสระในภาษาไทยนั้นมีอยู่ 24 เสียง โดยแบ่งเป็นสระเสียงเดี่ยว เสียงสั้นและเสียงยาวอย่างละ 9 เสียง และยังมีสระผสมอีก 6 เสียง ซึ่งในที่นี้ทำการรู้จำเฉพาะว่าเป็นเสียงสระเสียงเดี่ยวเสียงเดี่ยวเสียงใดเท่านั้น แต่ยังไม่สามารถระบุได้ว่าเป็นเสียงสั้นหรือเสียงยาว และเสียงสระผสมซึ่งเกิดจากการการรวมกันของสระเดี่ยวสองเสียงนั้น ซึ่งมีแนวทางการพัฒนาดังนี้

- ความแตกต่างของเสียงสระเสียงสั้นและเสียงยาวนั้นที่เห็นได้ชัดที่สุดคือ ระยะเวลาในการออกเสียงแต่ความแตกต่างนี้ไม่ทำให้สามารถแยกแยะเสียงสั้นและยาวระหว่างบุคคลได้ ลักษณะอีกอย่างคือการหยุดของเสียงโดยที่เสียงสั้น จะมีลักษณะการหยุดของเสียงแบบทันทีทันใด ในขณะที่เสียงยาว จะค่อยๆลดระดับเสียงลงอย่างช้าๆเมื่อเปรียบเทียบกับเสียงสั้น

- เสียงสระผสมนั้นเกิดจากการผสมของเสียงสระเพียง 3 เสียงคือ “อิ” , “อึ” และ “อุ” ซึ่งจะอยู่ในตอนหน้าของเสียง กับเสียง “อะ” ซึ่งจะอยู่ในตอนหลังเท่านั้น
- สิ่งที่น่าสนใจอีกอย่างคือการศึกษาหาจำนวนของ training set ที่จำเป็นสำหรับแบบอ้างอิงแต่กรณีที่จะทำให้เกิดประสิทธิภาพสูงสุด
- ความคล้ายคลึงกันของสระในกลุ่มสระเดียวกัน เป็นต้นเหตุของความผิดพลาดส่วนใหญ่ที่เกิดขึ้นในการรู้จำ ดังนั้นหากทำการแยกแยะกลุ่มสระให้ได้ชั้นหนึ่งก่อน คือเป็นสระหน้า สระกลาง หรือ สระหลัง แล้วจึงทำการแยกแยะสระในแต่ละกลุ่มซึ่งมีเพียงสามสระที่แตกต่างกันอีกชั้นหนึ่งก็อาจทำให้ได้ความถูกต้องในการรู้จำที่สูงขึ้น

เอกสารอ้างอิง

1. ชวดี อิศรปรีดา และคณะ. "การรู้จำเสียงพูด." ปริญญานิพนธ์วิศวกรรมศาสตรบัณฑิต สาขาวิชาวิศวกรรมโทรคมนาคม, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง. 2538.
2. กรุณา แก้วสมศรี และคณะ. "การต่อหมายเลขโทรศัพท์โดยใช้เสียง" ปริญญานิพนธ์วิศวกรรมศาสตรบัณฑิต สาขาวิชาวิศวกรรมโทรคมนาคม, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง. 2539.
3. ธันวา ศรีประโมง. "การวิเคราะห์เสียงพูดภาษาไทยในแกนความถี่ฮาร์โมนิค" วิทยานิพนธ์วิศวกรรมศาสตรมหาบัณฑิต สาขาวิชาวิศวกรรมไฟฟ้า บัณฑิตวิทยาลัย, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง. 2537.
4. ณัฐกร ทับทอง. "การรู้จำคำพูดภาษาไทย โดยใช้ลักษณะบ่งความต่างของหน่วยเสียง" วิทยานิพนธ์วิทยาศาสตรมหาบัณฑิต สาขาวิชาวิศวกรรมคอมพิวเตอร์ บัณฑิตวิทยาลัย, จุฬาลงกรณ์มหาวิทยาลัย. 2538.
5. ทัศเวท วีระวัฒน์. "การรู้จำเสียงคำไทยเฉพาะบุคคล" วิทยานิพนธ์วิศวกรรมศาสตรมหาบัณฑิต สาขาวิชาวิศวกรรมไฟฟ้า บัณฑิตวิทยาลัย, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง. 2541.
6. อมร ทวีศักดิ์, สัทศาสตร์, สถาบันวิจัยภาษาและวัฒนธรรมเพื่อพัฒนาชนบท มหาวิทยาลัยมหิดล 1999
7. วิไลวรรณ ขนิษฐานันท์. ภาษาและภาษาศาสตร์. พิมพ์ครั้งที่ 5. กรุงเทพฯ : สำนักพิมพ์มหาวิทยาลัยธรรมศาสตร์. 2533.
8. ราตรี ธันวารชร. **การศึกษาภาษาไทยตามแนวภาษาศาสตร์ เล่ม๑ เสียงและระบบเสียงในภาษาไทย.** กรุงเทพฯ : คณะศิลปศาสตร์ มหาวิทยาลัยธรรมศาสตร์. 2537.
9. ชาคริต อนันทราวัน. **หลักภาษาไทย.** กรุงเทพฯ : โอเดียนสโตร์. 2524.
10. สุนทร อรอินทร์, อัฐ เครือฟัก, "การประมวลผลเสียงพูดโดยการประมาณเชิงเส้น", วิทยานิพนธ์ภาควิชาวิศวกรรมไฟฟ้า สาขาวิชาวิศวกรรมโทรคมนาคม คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง 2538
11. Rabiner L.R., Schafer R.W. **Digital Processing of Speech Signals.** New Jersey : Prentice-Hall, Inc. 1978.
12. Thomas W. **Voice and Speech Processing.** Newyork : McGraw-Hill, Inc. 1987.

13. Rabiner L.R., Juang B.H. **Fundamentals of Speech Recognition.** New Jersey : Prentice Hall, Inc. 1993.

14. E.Zwicker, H.Fastl, **Psychoacoustics: Facts and Models Second Edition**, Springer, 1999

15. David M. Haward, James Angus, **Acoustics and Psychoacoustics**, Prentice Hall, 1996

16. Morton Nadler, Eric P. Smith, **Pattern Recognition Engineering**, John Wiley&Sons, 1993.

17. James A. McHugh, **Alogirthmic Graph Theory.** New Jersey : Prentice-Hall International, Inc. 1990.

18. Richard O. Duda, Peter E. Hart, **Pattern Classification and Scene Analysis.** New York : John Wiley & Sons.

19. Andrew Moore "Efficient Memory-based Learning for Robot Control" PhD. Thesis; Technical Report No. 209, Computer Laboratory, University of Cambridge. 1991.

20. จิตรลดา จารุมิศร์. "การออกแบบแบบจำลองในการรู้จำเสียงวรรณยุกต์สำหรับภาษาไทยโดยใช้เทคนิคการควอนไตซ์พิทช์ และ Hidden Makov Modelling" วิทยานิพนธ์วิศวกรรมศาสตร์มหาบัณฑิต สาขาวิชาวิศวกรรมไฟฟ้า บัณฑิตวิทยาลัย, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง. 2542.

21. William H. Press, Brain P. Flannery, Saul A. Teukolsky, William T. Vetterling, **Numerical Recipes in C.** New York : Cambridge University Press. 1988

22. Inge Gavat, Matei Zirra, "Fuzzy Models for Recognition of Spoken Vowels in Romanian Language", Proceeding of the IEEE International Conference on Fuzzy Systems, 1996.

ภาคผนวก

ภาคผนวก ก
โปรแกรมที่พัฒนาขึ้นในการวิจัย

1. โปรแกรมการตัดแบ่งสัญญาณเสียงที่ได้ออกเป็นเสียงสระโดด

1.1 โปรแกรมหลัก

```
bool SegmentWAV(char    inputWAVfilename[100]
                ,char   inputENGfilename[100]
                ,int    percent
                ,int    minlen
                ,bool   txt)
{
        FILE        *inputWAVfile;
        FILE        *inputENGfile;
        FILE        *outputfile;
        FILE        *outputtextfile;
        Char        outputWAVfilename[100];
        Char        outputTXTfilename[100];
        Int         temp,tempwav[frame];
        Float       average;
        Int         length;
        Float       threshold;
        Char        sound[maxlen]="";
        Int         voiceno;
        Char        voicenotxt[4];

        if((inputWAVfile=fopen(inputWAVfilename,"rb"))==NULL)
        {
                MessageBox(0,"Can't open input WAV file",inputWAVfilename,MB_OK);
                return(false);
        }
```

```
    {
        MessageBox(0,"Can't open input ENG file",inputENGfilename,MB_OK);
        return(false);
    }
    if (txt)
    {
        strcpy(outputTXTfilename,inputWAVfilename);
        outputTXTfilename[AnsiPos(".",outputTXTfilename)-1]='\0';
        strcat(outputTXTfilename,".xls");
        if((outputtextfile=fopen(outputTXTfilename,"wt"))==NULL)
        {
            MessageBox(0,"Can't create output text
            file",outputTXTfilename,MB_OK);
            return(false);
        }
    }
    average = 0;
    length = 0;
    while(!feof(inputENGfile))
    {
        fread(&temp,sizeof(int),1,inputENGfile);
        average += temp;
        length++;
    }
    rewind(inputENGfile);
    average/=length;
    threshold = average*percent/100;
    fseek(inputWAVfile,60,SEEK_SET);
    voiceno = 1;
    while(!feof(inputENGfile))
    {
```

```
fread(&temp,sizeof(int),1,inputENGfile);
fread(tempwav,sizeof(char),frame,inputWAVfile);
if (temp > threshold)
{
        if(txt)
                fprintf(outputtextfile,"0\n");
        if((outputfile=fopen("tempwav.dat","wb"))==NULL)
        {
                MessageBox(0,"Can't create output
                file","tempwav.dat",MB_OK);
                return(false);
        }
        length = 0;
        while (temp > threshold && !feof(inputENGfile))
        {
                fwrite(tempwav,sizeof(char),frame,outputfile);
                fread(&temp,sizeof(int),1,inputENGfile);
                fread(tempwav,sizeof(char),frame,inputWAVfile);
                length++;
                if(txt)
                        fprintf(outputtextfile,"0\n");
        }
        fclose(outputfile);
        if (length*frame > minlen)
        {
                strcpy(outputWAVfilename,inputWAVfilename);
                outputWAVfilename[AnsiPos(".",outputWAVfilename)-
                1]='\0';
                itoa(voiceno,voicenotxt,10);
                if (voiceno < 10)
                {
```

```
                    strcat(outputWAVfilename,"-00");
                    strcat(outputWAVfilename,voicenotxt);
                }
                if (voiceno >= 10 && voiceno <100)
                {
                    strcat(outputWAVfilename,"-0");
                    strcat(outputWAVfilename,voicenotxt);
                }
                if (voiceno >= 100)
                {
                    strcat(outputWAVfilename,"-");
                    strcat(outputWAVfilename,voicenotxt);
                }
                strcat(outputWAVfilename,".wav");
                WriteDataToFile("tempwav.dat",outputWAVfilename,false);
                voiceno++;
            }
        }
        else
        if(txt)
            fprintf(outputtextfile,"%d\n",(int)threshold);
    }
    fclose(inputWAVfile);
    fclose(inputENGfile);
    fclose(outputfile);
    if(txt)
        fclose(outputtextfile);
    return(true);
}
```

1.2 โปรแกรมย่อยในการคำนวนค่าพลังงาน

```
bool Energy( char   inputfilename[100]
            ,char   outputfilename[100]
            ,char   outputtextfilename[100]
            ,bool   txt)
{
        FILE            *inputfile;
        FILE            *outputfile;
        FILE            *outputtextfile;
        Int             energyframe[frame];
        Int             i;
        Int             temp;
        Float           energy;
        Int             energyint;

        if((inputfile=fopen(inputfilename,"rb"))==NULL)
        {
                MessageBox(0,"Can't open input file",inputfilename,MB_OK);
                return(false);
        }
        if((outputfile=fopen("temp.sqr","wb"))==NULL)
        {
                MessageBox(0,"Can't create output file","temp.sqr",MB_OK);
                return(false);
        }
        fseek(inputfile,60,SEEK_SET);
        while(!feof(inputfile))
        {
                temp = fgetc(inputfile);
                temp -= 128;
```

```
            fwrite(&temp,sizeof(int),1,outputfile);
    }
    fclose(inputfile);
    fclose(outputfile);
    if((inputfile=fopen("temp.sqr","rb"))==NULL)
    {
            MessageBox(0,"Can't open input file","temp.sqr",MB_OK);
            return(false);
    }
    if((outputfile=fopen(outputfilename,"wb"))==NULL)
    {
            MessageBox(0,"Can't create output file",outputfilename,MB_OK);
            return(false);
    }
    if (txt)
            if((outputtextfile=fopen(outputtextfilename,"wt"))==NULL)
            {
                    MessageBox(0,"Can't create output text
                    file",outputtextfilename,MB_OK);
                    return(false);
            }

while(!feof(inputfile))
{
            fread(energyframe,sizeof(int),frame,inputfile);
            energy = 0;
            for (i=0;i<frame;i++)
                    energy+=energyframe[i];
            energy/=frame;
            if (energy<0)
                    energy = 0;
```

```
                energy = 0;
            energyint = (int)energy;
            fwrite(&energyint,sizeof(int),1,outputfile);
            if (txt)
                fprintf(outputtextfile,"%d\n",(int)energy);
        }
        fclose(inputfile);
        fclose(outputfile);
        if (txt)
            fclose(outputtextfile);
        return(true);
}
```

1.3 โปรแกรมย่อยในการคำนวนค่าความถี่มูลฐาน

```
int Pitch(char   inputfilename[100]
        ,char   digitalfilename[100]
        ,char   autocorfilename[100]
        ,char   clipfilename[100]
        ,char   pitchfilename[100]
        ,int    framelength
        ,int    shiftlength
        ,int    percentpitch
        ,int    percentclip
        ,int    lag
        ,bool   txt)
{
        FILE            *inputfile;
        FILE            *digitalfile;
        FILE            *autocorfile;
        FILE            *clipfile;
        FILE            *pitchfile;
```

```
FILE        *autocortextfile;
FILE        *cliptextfile;
FILE        *pitchtextfile;
char        digitaltextfilename[100];
char        autocortextfilename[100];
char        cliptextfilename[100];
char        pitchtextfilename[100];
int         *pitch;
int         count;
int         round2;
int         number_sample;
int         c;
int         temp;
int         round;
int         i;

if (txt)
{
        strcpy(digitaltextfilename,digitalfilename);
        digitaltextfilename[AnsiPos(".",digitaltextfilename)-1]='\0';
        strcat(digitaltextfilename,"-tmp.xls");
        strcpy(autocortextfilename,autocorfilename);
        autocortextfilename[AnsiPos(".",autocortextfilename)-1]='\0';
        strcat(autocortextfilename,"-tmp.xls");
        strcpy(cliptextfilename,clipfilename);
        cliptextfilename[AnsiPos(".",cliptextfilename)-1]='\0';
        strcat(cliptextfilename,"-tmp.xls");
        strcpy(pitchtextfilename,pitchfilename);
        pitchtextfilename[AnsiPos(".",pitchtextfilename)-1]='\0';
        strcat(pitchtextfilename,"-pth.xls");
}
```

```
if ((inputfile = fopen(inputfilename,"rb")) == NULL )
{
        MessageBox(0,"Can't open inputfile",inputfilename,MB_OK);
        return(1);
}
if ((digitalfile = fopen(digitalfilename,"wb")) == NULL )
{
        MessageBox(0,"Can't create digitalfile",digitalfilename,MB_OK);
        return(1);
}
if ((clipfile = fopen(clipfilename,"wb")) == NULL )
{
        MessageBox(0,"Can't create clipfile",clipfilename,MB_OK);
        return(1);
}
if ((pitchfile = fopen(pitchfilename,"wb")) == NULL )
{
        MessageBox(0,"Can't create pitchfile",pitchfilename,MB_OK);
        return(1);
}
if (txt)
{
        if ((digitaltextfile = fopen(digitaltextfilename,"wt")) == NULL )
        {
                MessageBox(0,"Can't create
                digitaltextfile",digitaltextfilename,MB_OK);
                return(1);
        }
        if ((autocortextfile = fopen(autocortextfilename,"wt")) == NULL )
        {
```

```
                MessageBox(0,"Can't create
                autocortextfile",autocortextfilename,MB_OK);
                return(1);
        }
        if ((cliptextfile = fopen(cliptextfilename,"wt")) == NULL )
        {
                MessageBox(0,"Can't create
                cliptextfile",cliptextfilename,MB_OK);
                return(1);
        }
        if ((pitchtextfile = fopen(pitchtextfilename,"wt")) == NULL )
        {
                MessageBox(0,"Can't create
                pitchtextfile",pitchtextfilename,MB_OK);
                return(1);
        }
}
number_sample = 0;
fseek(inputfile,44L,SEEK_CUR);
number_sample=-1;
while (!feof(inputfile))
{
        temp = fgetc(inputfile);
        number_sample++;
}
round = (number_sample/(shiftlength))-((framelength-shiftlength)/shiftlength);
rewind(inputfile);
fseek(inputfile,44L,SEEK_CUR);

for (i = 1;i <= number_sample-3;i++)
{
```

```
        int     x_n,x_n1,x_n2,x_n3;
        float   y_n,y_n1,y_n2,y_n3;
        if (i == 1)
        {
                y_n  = 0;
                y_n1 = 128;
                y_n2 = 128;
                y_n3 = 128;
                x_n3 = (int) fgetc(inputfile);
                x_n2 = (int) fgetc(inputfile);
                x_n1 = (int) fgetc(inputfile);
                x_n  = (int) fgetc(inputfile);
        }
        else
        {
                y_n3 = y_n2;
                y_n2 = y_n1;
                y_n1 = y_n;
                x_n3 = x_n2;
                x_n2 = x_n1;
                x_n1 = x_n;
                x_n = (int) fgetc(inputfile);
        }
        y_n = (2.13806*y_n1)-(1.802663*(y_n2))+(0.545353*(y_n3))+(0.085292*
        (x_n))-(0.0257159*(x_n1))-(0.0257159*(x_n2))+(0.085292*(x_n3));
        fwrite(&y_n,sizeof(float),1,digitalfile);
        if (txt)
                fprintf(digitaltextfile,"%d\n",y_n);
}
fclose(inputfile);
fclose(digitalfile);
```

```
if ((digitalfile = fopen(digitalfilename,"rb")) == NULL )
{
        MessageBox(0,"Can't open digitalfile",digitalfilename,MB_OK);
        return(1);
}
char    *clip;
float   *wave;
float   *temp2;
if ((clip = (char*)calloc(framelength,sizeof(char)))==NULL)
{
        MessageBox(0,"Can't allocate memory for clip","clip",MB_OK);
        return(1);
}
if ((wave = (float*)calloc(framelength,sizeof(float)))==NULL)
{
        MessageBox(0,"Can't allocate memory for wave","wave",MB_OK);
        return(1);
}
if ((temp2 = (float*)calloc(framelength,sizeof(float)))==NULL)
{
        MessageBox(0,"Can't allocate memory for temp2","temp2",MB_OK);
        return(1);
}
for (int d=0;d<round;d++)
{
        char    i1;
        float   firstmax,lastmax,minofmax;
        int     threshold_clip,upperthreshold,lowerthreshold;
        if (d==0)
                for (int j=0;j<framelength;j++)
                {
```

```
                fread(&temp2[j],sizeof(float),1,digitalfile);
                wave[j] = temp2[j];
            }
        else
        {
            for (int j=0;j<framelength-shiftlength;j++)
            {
                temp2[j] = wave[j+shiftlength];
                wave[j] = temp2[j];
            }
            for (int j=0;j<shiftlength;j++)
            {
                fread(&wave[j+(framelength-shiftlength)],sizeof
                (float),1,digitalfile);
                temp2[j+(framelength-shiftlength)] = wave[j+
                (framelength-shiftlength)];
            }
        }
    for (int j=0;j<framelength;j++)
        temp2[j] = abs(temp2[j] - 128);
    firstmax = 0;
    lastmax = 0;
    for (int j=0;j<shiftlength;j++)
    {
        if (firstmax < temp2[j])
            firstmax = temp2[j];
        if (lastmax < temp2[j+(framelength-shiftlength)])
            lastmax = temp2[j+(framelength-shiftlength)];
    }
    if (firstmax < lastmax)
        minofmax = firstmax;
```

```
            if (lastmax < firstmax)
                    minofmax = lastmax;
            if (firstmax == lastmax)
                    minofmax = firstmax;
            threshold_clip = (minofmax * percentclip)/100;
            upperthreshold = 128 + threshold_clip;
            lowerthreshold = 128 - threshold_clip;
            for (c=0;c<framelength;c++)
            {
                    if (wave[c] >= upperthreshold)
                            i1 = 1;
                    if (wave[c] <= lowerthreshold)
                            i1 =-1;
                    if ((wave[c] > lowerthreshold) && (wave[c] < upperthreshold))
                            i1 = 0;
                    clip[c] = i1;
                    if (txt)
                            fprintf(cliptextfile,"%d\n",clip[c]);
            }
            fwrite(clip,sizeof(char),framelength,clipfile);
    }
    free(clip);
    free(wave);
    free(temp2);
    fclose(digitalfile);
    fclose(clipfile);

    if ((clipfile=fopen(clipfilename,"rb"))==NULL)
    {
            MessageBox(0,"Can't open clipfile",clipfilename,MB_OK);
            return(1);
```

```
    }
    if ((autocorfile=fopen(autocorfilename,"wb"))==NULL)
    {
            MessageBox(0,"Can't create autocorfile",autocorfilename,MB_OK);
            return(1);
    }
    count = -1;
    while (!feof(clipfile))
    {
            temp=fgetc(clipfile);
            count = count++;
    }
    char    *x1;
    int     y1,y2,autotemp,Autocor;
    int     *autocor;
    int     *Autok;
    int     Nextpitch;
    int     round1 = count/framelength;

    if ((x1 = (char*)calloc(framelength,sizeof(char)))==NULL)
    {
            MessageBox(0,"Can't allocate memory for x1","x1",MB_OK);
            return(1);
    }
    if ((autocor = (int*)calloc(lag,sizeof(int)))==NULL)
    {
            MessageBox(0,"Can't allocate memory for autocor","autocor",MB_OK);
            return(1);
    }
    if ((Autok = (int*)calloc(lag,sizeof(int)))==NULL)
    {
```

```
            MessageBox(0,"Can't allocate memory for AutoK","AutoK",MB_OK);
            return(1);
    }
    rewind(clipfile);
    for (int j=1;j<=round1;j++)
    {
            fread(x1,sizeof(char),framelength,clipfile);
            for (int k=0;k<lag;k++)
            {
                    Autocor = 0;
                    for (int m=0;m<framelength-k-1;m++)
                    {
                            y1 = x1[m];
                            y2 = x1[m+k];
                            autotemp = y1*y2;
                            Autocor = Autocor+autotemp;
                    }
                    autocor[k]=Autocor;
                    if (txt)
                            fprintf(autocortextfile,"%d\n",autocor[k]);
            }
            fwrite(autocor,sizeof(int),lag,autocorfile);
    }
    free(x1);
    free(autocor);
    fclose(clipfile);
    fclose(autocorfile);

    if ((autocorfile = fopen(autocorfilename,"rb")) == NULL )
    {
            MessageBox(0,"Can't open autocorfile",autocorfilename,MB_OK);
```

```
            return(1);
    }
    count = -1;
    while (!feof(autocorfile))
    {
            temp = fgetc(autocorfile);
            count = count++;
    }
    count /= 4;
    round2 = (count/lag);
    rewind(autocorfile);
    pitch=(int*)calloc(sizeof(int),round2);
    int lastk,examinepitch,pitchfrequency;
    for (int n=1;n<=round2;n++)
    {
            PITCH:
            for (int j=0;j<lag;j++)
                    fread(&Autok[j],sizeof(int),1,autocorfile);
            Nextpitch = (Autok[0] * percentpitch)/100;
            for (int k=0;k<lag;k++)
            {
                    int flag = 0;
                    if (k == 0)
                    {
                            lastk = 0;
                            examinepitch = 0;
                            k = 30;
                    }
                    if (abs(Autok[k]) >= Nextpitch)
                    {
                            if (k < 220)
```

```
                    for (c=1;c<=30;c++)
                        if (abs(Autok[k]) < abs(Autok[k+c]))
                            flag = 1;
                if (flag == 0)
                {
                    Nextpitch = (Autok[k]*percentpitch)/100;
                    if (Nextpitch == 0)
                        Nextpitch = 1;
                    examinepitch = k - lastk;
                    lastk = k;
                    pitchfrequency = 11025/examinepitch;
                    k = lag;
                }
            }
        }
        if (examinepitch != 0)
            pitchfrequency = 11025/examinepitch;
        if (examinepitch == 0)
        {
            percentpitch = percentpitch - 5;
            goto PITCH;
        }
        pitch[n-1]=pitchfrequency;
        if (txt)
            fprintf(pitchtextfile,"%d\n",pitch[n-1]);
}
fwrite(pitch,sizeof(int),round2,pitchfile);
free(pitch);
fclose(autocorfile);
fclose(pitchfile);
if (txt)
```

```
        if (txt)
        {
                fclose(digitaltextfile);
                fclose(cliptextfile);
                fclose(autocortextfile);
                fclose(pitchtextfile);
        }
        return(0);
}
```

2. โปรแกรมคำนวณความเข้มสัญญาณในแถบความถี่วิกฤตจากไฟล์เสียง

```
bool   WAV2IB(   char      inputfilename[128]
                ,char      outputpath[128]
                ,bool      path
                ,char      addtext[128]
                ,int       order
                ,float     adapt
                ,float     res
                ,bool      ibtxt
                ,bool      hztxt
                ,bool      lpctxt)
{
//************************
//Determine Bark Band Edge
//************************
        float   bandEdge[24];
        float   bark;
        int     barkno=1;

        bandEdge[0]=0;
        for (int fno=0;fno<5514;fno++)
```

```
                bark = 6*log(((fno*2*M_PI)/(1200*M_PI*adapt))+sqrt(pow((fno*2*M_PI)/
                (1200*M_PI*adapt),2)+1));
                if (bark > barkno)
                {
                        bandEdge[barkno]=fno/(5512.5/180);
                        barkno++;
                }
        }
        bandEdge[barkno]=180;
//*************************************
//Bark Band Edge is in "bandEdge" in 'w'
//*************************************
//***
//Hz
//***
        complex         <double>*hz;
        float           *abshz;

        if((hz=(complex<double>*)calloc(180/res+1,sizeof(complex<double>)))==NULL)
        {
                MessageBox(0,"Can not allocate memory for hz", "Error!", MB_OK);
                return (false);
        }
        if((abshz=(float*)calloc(180/res+1,sizeof(float)))==NULL)
        {
                MessageBox(0,"Can not allocate memory for abshz", "Error!", MB_OK);
                return (false);
        }
//*********
//LPC PHASE
//*********
```

```
char    outputfilename[128]="";
char    outputibtextfilename[128]="";
char    outputhztextfilename[128]="";
char    outputlpctextfilename[128]="";
FILE    *inputfile;
FILE    *outputfile;
char    outputtext1[128]="pre.xls";
char    outputtext2[128]="win.xls";
char    outputtext3[128]="sig.xls";
FILE    *outputibtextfile;
FILE    *outputhztextfile;
FILE    *outputlpctextfile;
int     length = -1;
int     number_of_frame;
int     nf, fl, i;
float   *signal;
float   *preemphasis;
float   *windowed;
float   R[MAXP+1],A[MAXP][MAXP],X[MAXP][MAXP];
float   alpha[MAXP+1],phi[MAXP+1];
char    temp_char;
int     temp;
float   q,gain;


if (!path)
{
        for (i=0;i<AnsiPos(".",inputfilename)-1;i++)
                outputfilename[i]=inputfilename[i];
        strcat(outputfilename,"_");
        strcat(outputfilename,addtext);
```

```
            strcpy(outputibtextfilename,outputfilename);
            strcpy(outputhztextfilename,outputfilename);
            strcpy(outputlpctextfilename,outputfilename);
            strcat(outputfilename,".IB");
            strcat(outputibtextfilename,"-IB.xls");
            strcat(outputhztextfilename,"-HZ.xls");
            strcat(outputlpctextfilename,"-LPC.xls");
}
else
{
            int     namelength = strlen(inputfilename);
            int     startname;
            char    tempname[128]="";
            int     j=0;

            for (i=namelength;i>0 && inputfilename[i]!='\\';i--)
                    startname = i;
            for (i=startname;i<namelength;i++)
            {
                    tempname[j]=inputfilename[i];
                    j++;
            }
            for (i=0;i<AnsiPos(".",tempname)-1;i++)
                    outputfilename[i]=tempname[i];
            strcpy(tempname,outputfilename);
            strcpy(outputfilename,outputpath);
            strcat(outputfilename,"\\");
            strcat(outputfilename,tempname);
            strcat(outputfilename,"_");
            strcat(outputfilename,addtext);
            strcpy(outputibtextfilename,outputfilename);
```

```
            strcpy(outputhztextfilename,outputfilename);
            strcpy(outputlpctextfilename,outputfilename);
            strcat(outputfilename,".IB");
            strcat(outputibtextfilename,"-IB.xls");
            strcat(outputhztextfilename,"-HZ.xls");
            strcat(outputlpctextfilename,"-LPC.xls");
    }
    if ((inputfile=fopen(inputfilename,"rb"))==NULL)
    {
            MessageBox(0,"Can't open inputfile",inputfilename,MB_OK);
            return(false);
    }
    if ((outputfile=fopen(outputfilename,"wb"))==NULL)
    {
            MessageBox(0,"Can't create outputfile",outputfilename,MB_OK);
            return(false);
    }
    if (ibtxt)
            if ((outputibtextfile=fopen(outputibtextfilename,"wt"))==NULL)
            {
                    MessageBox(0,"Can't create output
                    textfile",outputibtextfilename,MB_OK);
                    return(false);
            }
    if (hztxt)
            if ((outputhztextfile=fopen(outputhztextfilename,"wt"))==NULL)
            {
                    MessageBox(0,"Can't create output
                    textfile",outputhztextfilename,MB_OK);
                    return(false);
            }
```

```
        if (lpctxt)
                if ((outputlpctextfile=fopen(outputlpctextfilename,"wt"))==NULL)
                {
                        MessageBox(0,"Can't create output
                        textfile",outputlpctextfilename,MB_OK);
                return(false);
 }
 if ((signal=(float*)calloc(FRAME+1,sizeof(float)))==NULL)
 {
         MessageBox(0,"Can't allocate memory for signal","signal",MB_OK);
        return(false);
 }
 if ((preemphasis=(float*)calloc(FRAME+1,sizeof(float)))==NULL)
 {
        MessageBox(0,"Can't allocate memory for
        preemphasis","preemphasis",MB_OK);
        return(false);
}
if ((windowed=(float*)calloc(FRAME+1,sizeof(float)))==NULL)
{
        MessageBox(0,"Can't allocate memory for
        windowed","windowed",MB_OK);
        return(false);
}

while (!feof(inputfile))
{
        temp = fgetc(inputfile);
        length++;
}
rewind(inputfile);
```

```
length -= 60;  /* bypass the header of wave file */
fseek(inputfile,60,SEEK_SET); /* set pointer to first data */
number_of_frame = (length-FRAME+SHIFT)/SHIFT;
fread(&temp_char,sizeof(char),1,inputfile);
signal[0]=(unsigned char)temp_char;
int called = FIRST_CALL;
for (nf=0;nf<number_of_frame;nf++)
{
        //*******************************
        //Start LPC Process for each Frame
        //*******************************
        //LPC Step 1.
        //Preemphasis and window
        if (called == FIRST_CALL)
        {
            for ( fl=1;fl<=FRAME;fl++ )
            {
                fread(&temp_char,sizeof(char),1,inputfile);
                signal[fl]=(unsigned char)temp_char;
                preemphasis[fl] = (signal[fl] -  ADAPTIVE *(float)signal[fl-
                1]);
                windowed   [fl] = preemphasis[fl]*(0.54-0.46*cos(2*M_PI*
                (fl-1)/(FRAME-1)));
            }
            called = NEXT_CALL;
        }
        else
        {
            for (fl=1;fl<=(FRAME - SHIFT);fl++)
            {
                signal    [fl] = signal[fl+SHIFT];
```

```
                preemphasis[fl] = preemphasis[fl+SHIFT];
                windowed   [fl] = preemphasis[fl]*(0.54-0.46*cos(2*M_PI*
                (fl-1)/(FRAME-1)));
        }
        for (fl=(FRAME - SHIFT+1);fl<=FRAME;fl++)
        {
                fread(&temp_char,sizeof(char),1,inputfile);
                signal[fl]=(unsigned char)temp_char;
                preemphasis[fl] = (signal[fl] - ADAPTIVE * (float)signal[fl-
                1]);
                windowed   [fl] = preemphasis[fl]*(0.54-0.46*cos(2*M_PI*
                (fl-1)/(FRAME-1)));
        }
}
//LPC Step 2.
//AutoCorrelation
float   alk,akk,maxvalue,tempfloat;
int     m,i,j,k,l,maxrow;
float   matrix_temp[MAXP][MAXP];

for (k=0;k<=order;k++)
{
        R[k] = 0;
        for (m=0;m<=FRAME-1-k;m++)
        R[k] += windowed[m+1] * windowed[m+k+1];
}
for (i=0;i<order;i++)
{
        m = 0;
        for (k=i;k<order;k++)
        {
```

```
                A[i][k] = R[m++];
                A[k][i] = A[i][k];
            }
        }
        for (i=0;i<order;i++)
            for (j=0;j<order;j++)
                matrix_temp[i][j] = A[i][j];
        for (i=0;i<order;i++)
            for (j=0;j<order;j++)
                if (i==j)
                    X[i][j] = 1;
                else
                    X[i][j]=0;
        for (k=0;k<order;k++)
        {
            maxrow = k;
            maxvalue = fabs(matrix_temp[k][k]);
            for (i=k+1;i<order;i++)
            {
                if (maxvalue < fabs(matrix_temp[i][k]))
                {
                    maxvalue = fabs(matrix_temp[i][k]);
                    maxrow   = i;
                }
            }
            if (maxrow != k)
            {
                for (j=0;j<order;j++)
                {
                    tempfloat          = matrix_temp[k][j];
                    matrix_temp[k][j]  = matrix_temp[maxrow][j];
```

```
                    matrix_temp[maxrow][j] =  tempfloat;
                }
                for (j=0;j<order;j++)
                {
                    tempfloat          = X[k][j];
                    X[k][j]     = X[maxrow][j];
                    X[maxrow][j] =  tempfloat;
                }
            }
            akk = matrix_temp[k][k];
            for (j=k;j<order;j++)
                matrix_temp[k][j] /= akk;
            for (j=0;j<order;j++)
                X[k][j] /= akk;
            akk = matrix_temp[k][k];
            for (l=0;l<order;l++)
            {
                alk = matrix_temp[l][k];
                if (k != l)
                {
                    for (m=k;m<order;m++)
                        matrix_temp[l][m]  -= matrix_temp[k]
                        [m]/akk*alk;
                    for (m=0;m<order;m++)
                        X[l][m] -= X[k][m]/akk*alk;
                }
            }
        }
    for (i=1;i<=order;i++)
        phi[i] = R[i];
    //LPC Step 3.
```

```
//Determine Coffecients
for (i=0;i<order;i++)
{
        alpha[i]=0;
        for (k=0;k<order;k++)
                alpha[i] += X[i][k]*phi[k+1];
        if(lpctxt)
                fprintf(outputlpctextfile,"%10.7f\t",alpha[i]);
}
if(lpctxt)
        fprintf(outputlpctextfile,"\n");
q = (float)R[0];
for (i=0;i<order;i++)
        q -= ((float)alpha[i]*(float)R[i+1]);
gain = sqrt(q);
//***********************************
//End of LPC Process for each frame
//LPC Coefficients are in "alpha"
//***********************************

//*****************************
//Determine H(z) for each frame
//*****************************
        float   x,y;
        int     w_no=0;
        float   w;
        complex         <double>z;
        complex         <double>zsum;
        complex         <double>p;

        for (w=0;w<180;w+=res)
```

```
{
        x=r*cos(w*M_PI/180);
        y=r*sin(w*M_PI/180);
        z=complex<double>(x,y);
        zsum=complex<double>(0,0);
        for (k=1;k<=order;k++)
        {
            p=pow(z,-1*k);
            zsum=zsum+(complex<double>(alpha[k-1],0)*p);
        }
        hz[w_no]=complex<double>(1,0)/(complex<double>(1,0)-zsum);
        w_no++;
}
//Absolute Value
for (int i=0;i<180/res;i++)
{
        abshz[i]=(float)abs(hz[i]);
        if(hztxt)
            fprintf(outputhztextfile,"%10.7f\n",abshz[i]);
}
if(hztxt)
        fprintf(outputhztextfile,"\n");
```

```
//*****************************
//H(z) is sampled followed "res"
//and its absolute is normalized
//"abshz" size 180/res
//*****************************
```

```
//*********************
//Determine IB from H(z)
//**********************
                float   IB[24];
                float   ifloat;
                float   rem;

                barkno=1;
                for (int i=0;i<24;i++)
                        IB[i]=0;
                for (int i=1;i<180/res;i++)
                {
                        ifloat = i*res;
                        if (ifloat<=bandEdge[barkno])
                          IB[barkno-1]+=(abshz[i]+abshz[i-1])*res*0.5;
                        else
                        {
                          rem = res-(ifloat-bandEdge[barkno]);
                          IB[barkno-1]+=0.5*rem*(abshz[i-1]+((((res-rem)/res)*
                          (abshz[i-1]-abshz[i]))+abshz[i]));
                          barkno++;
                          IB[barkno-1]+=(((abshz[i]+abshz[i-1])*res)-(rem*(abshz[i-
                          1]+((((res-rem)/res)*(abshz[i-1]-abshz[i]))+abshz
                          [i])))*0.5;
                        }
                }
//***************************
//Normalize Bark Intensities
//***************************
                int     aj;
                float   a=0;
```

```
                for (aj=0;aj<24;aj++)
                        a+=pow(IB[aj],2);
                a = sqrt(a);
                a = 1/a;
                for (aj=0;aj<24;aj++)
                        IB[aj]*=a;
//***************************
//Write Bark Intensities "IB"
//***************************
                fwrite(IB,sizeof(float),24,outputfile);
                for (int i=0;i<24 && ibtxt;i++)
                        fprintf(outputibtextfile,"%10.7f\t",IB[i]);
                if (ibtxt)
                        fprintf(outputibtextfile,"\n");
        }
        free(signal);
        free(preemphasis);
        free(windowed);
        fclose(inputfile);
        fclose(outputfile);
        if (ibtxt)
                fclose(outputibtextfile);
        if (hztxt)
                fclose(outputhztextfile);
        if (lpctxt)
        {
                fclose(outputlpctextfile);
                fclose(outputfile1);
                fclose(outputfile2);
                fclose(outputfile3);
```

```
            fclose(outputfile3);
        }
    //HZ
        free(hz);
        free(abshz);
        return(true);
}
```

3. โปรแกรมสร้างและค้นหาข้อมูลใน Kd-Tree

3.1 โปรแกรมหลัก

```
bool   newann( char   trainfilename[128]
             ,char   testfilename[128]
             ,char   resultfilename[128]
             ,int    vectorspermodel
             ,int    numofmodel
             ,int    order
             ,int    dimension
             ,float  errorbound)
{
    int         n_pts;                  // actual number of data points
    int         m_pts;
    ANNpointArray   data_pts;           // data points
    ANNpoint        query_pt;           // query point
    ANNidxArray     nn_idx;             // near neighbor indices
    ANNdistArray    dists;              // near neighbor distances
    ANNkd_tree      *the_tree;          // search structure
    int     *modelstat;
    int     expectedmodel;
    FILE    *trainfile;
    FILE    *testfile;
    FILE    *resultfile;
```

```
int     length = -1;
int     temp;
int     round;
nt      k;
int     dim;
float   eps = errorbound;
int     result;
float   *tempfloat;
int     actualmodel;
int     correct = 0;
int     lastmodel = 0;
bool    changemodel;
char    conclusionfilename[128]="conclusion.xls";
int     *resultstat;

if ((trainfile=fopen(trainfilename,"rb"))==NULL)
{
        MessageBox(0,"Can't open trainfile",trainfilename,MB_OK);
        return(false);
}
if ((testfile=fopen(testfilename,"rb"))==NULL)
{
        MessageBox(0,"Can't open testfile",testfilename,MB_OK);
        return(false);
}
if ((resultfile=fopen(resultfilename,"at"))==NULL)
{
        MessageBox(0,"Can't open resultfile",resultfilename,MB_OK);
        return(false);
}
if ((conclusionfile=fopen(conclusionfilename,"at"))==NULL)
```

```
{
        MessageBox(0,"Can't open conclusionfile",conclusionfilename,MB_OK);
        return(false);
}
while (!feof(trainfile))
{
        temp = fgetc(trainfile);
        length++;
}
length/=4;
length/=dimension;
n_pts = length;
m_pts = length;
rewind(trainfile);
while (!feof(testfile))
{
        temp = fgetc(testfile);
        length++;
}
length/=4;
length/=dimension;
round = length;
rewind(testfile);
k = order+1;
dim = dimension;

if ((modelstat=(int*)calloc(numofmodel,sizeof(int)))==NULL)
{
        MessageBox(0,"Can't allocate memory for modelstat","data",MB_OK);
        return(false);
}
```

```
if ((tempfloat=(float*)calloc(dim,sizeof(float)))==NULL)
{
        MessageBox(0,"Can't allocate memory for
        tempfloat","tempfloat",MB_OK);
        return(false);
}
if ((resultstat=(int*)calloc(numofmodel,sizeof(int)))==NULL)
{
        MessageBox(0,"Can't allocate memory for resultstat","resultstat",MB_OK);
        return(false);
}
query_pt = annAllocPt(dim);                    // allocate query point
data_pts = annAllocPts(m_pts, dim);            // allocate data points
nn_idx = new ANNidx[k];                        // allocate near neigh indices
dists = new ANNdist[k];                        // allocate near neighbor dists
// read data points
for (n_pts=0;n_pts<m_pts;n_pts++)
{
        fread(tempfloat,sizeof(float),dim,trainfile);
        for (int d=0;d<dim;d++)
                data_pts[n_pts][d]=(double)tempfloat[d];
}
rewind(trainfile);
the_tree = new ANNkd_tree(                     // build search structure
                        data_pts,              // the data points
                        n_pts,                 // number of points
                        dim);                  // dimension of space
for (int i=0;i<numofmodel;i++)
        resultstat[i]=0;
fprintf(conclusionfile,"%s %s K=%d\n",trainfilename,testfilename,order);
for (int round_no=0;round_no<round;round_no++)
```

```
{
        if (round_no!=(vectorspermodel*(lastmodel+1))-1)
                changemodel = false;
        else
        {
                changemodel = true;
                lastmodel++;
        }
        fread(tempfloat,sizeof(float),dim,testfile);
        for (int d=0;d<dim;d++)
                query_pt[d]=tempfloat[d];
        the_tree->annkSearch(           // search
                        query_pt,       // query point
                        k,              // number of near neighbors
                        nn_idx,         // nearest neighbors (returned)
                        dists,          // distance (returned)
                        eps);           // error bound
        for (int modelno=0;modelno<numofmodel;modelno++)
                modelstat[modelno]=0;
        for (int kno=1;kno<k;kno++)
        {
                for (int modelno=0;modelno<numofmodel;modelno++)
                {
                        if (nn_idx[kno]>=vectorspermodel*modelno && nn_idx
                        [kno]<vectorspermodel*(modelno+1))
                                modelstat[modelno]+=1;
                }
        }
        result = 0;
        for (int modelno=0;modelno<numofmodel;modelno++)
        {
```

```
            if (round_no>=vectorspermodel*modelno && round_no
            <vectorspermodel*(modelno+1))
                actualmodel = modelno;
    }
    for (int modelno=0;modelno<numofmodel;modelno++)
            if (modelstat[modelno]>modelstat[0])
            {
                temp = modelstat[0];
                modelstat[0] = modelstat[modelno];
                modelstat[modelno] = temp;
                result = modelno;
            }
    if (actualmodel == result)
    {
            correct++;
            fprintf(resultfile,"%d\t%d\t1\t\tDetail:\t",actualmodel,result);
    }
    else
            fprintf(resultfile,"%d\t%d\t0\t\tDetail:\t",actualmodel,result);
    for (int m=0;m<numofmodel;m++)
    {
            if (m == result)
            {
                fprintf(resultfile,"1\t");
                resultstat[m]++;
            }
            else
                fprintf(resultfile,"0\t");
    }
    if(changemodel)
    {
```

```
                {
                        fprintf(resultfile,"\nModel#%d\n",lastmodel);
                        fprintf(conclusionfile,"Model#%d\t",lastmodel);
                        for (int j=0;j<numofmodel;j++)
                        {
                                fprintf(conclusionfile,"%d\t",resultstat[j]);
                                resultstat[j]=0;
                        }
                        fprintf(conclusionfile,"\n");
                }
                fprintf(resultfile,"\n");
        }
        fclose(trainfile);
        fclose(testfile);
        fclose(resultfile);
        fclose(conclusionfile);
        free(resultstat);
        free(modelstat);
        free(tempfloat);
        delete the_tree;
        delete nn_idx;
        delete dists;
        delete data_pts;
        delete query_pt;
        return(true);
}
```

3.2 โปรแกรมย่อยในการสร้าง Kd-Tree จากเซตข้อมูล

```
ANNkd_tree::ANNkd_tree(                 // construct from point array
  ANNpointArray     pa,                 // point array (with at least n pts)
  int               n,                  // number of points
```

```
    int                 bs,             // bucket size
    ANNsplitRule        split)          // splitting method
{
    SkeletonTree(n, dd, bs);            // set up the basic stuff
    pts = pa;                           // where the points are
    if (n == 0) return;                 // no points--no sweat
    ANNorthRect bnd_box(dd);            // bounding box for points
    annEnclRect(pa, pidx, n, dd, bnd_box);// construct bounding rectangle
                                        // copy to tree structure
    bnd_box_lo = annCopyPt(dd, bnd_box.lo);
    nd_box_hi = annCopyPt(dd, bnd_box.hi);
    switch (split) {                    // build by rule
    case ANN_KD_STD:                    // standard kd-splitting rule
        root = rkd_tree(pa, pidx, n, dd, bs, bnd_box, kd_split);
        break;
    case ANN_KD_MIDPT:                  // midpoint split
        root = rkd_tree(pa, pidx, n, dd, bs, bnd_box, midpt_split);
        break;
    case ANN_KD_FAIR:                   // fair split
        root = rkd_tree(pa, pidx, n, dd, bs, bnd_box, fair_split);
        break;
    case ANN_KD_SUGGEST:                // best (in our opinion)
    case ANN_KD_SL_MIDPT:               // sliding midpoint split
        root = rkd_tree(pa, pidx, n, dd, bs, bnd_box, sl_midpt_split);
        break;
    case ANN_KD_SL_FAIR:                // sliding fair split
        root = rkd_tree(pa, pidx, n, dd, bs, bnd_box, sl_fair_split);
        break;
    default:
        annError("Illegal splitting method", ANNabort);
    }
```

```
    }

    void ANNkd_tree::SkeletonTree(      // construct skeleton tree
            int n,                      // number of points
            int dd,                     // dimension
            int bs)                     // bucket size
    {
            dim = dd;                   // initialize basic elements
            n_pts = n;
            bkt_size = bs;
            root = NULL;                // no associated tree yet
            pts = NULL;                 // no associated points yet
            pidx = new int[n];          // allocate space for point indices
            for (int i = 0; i < n; i++)
                    pidx[i] = i;        // initially identity
            bnd_box_lo = bnd_box_hi = NULL;     // bounding box is nonexistent
            if (KD_TRIVIAL == NULL)             // no trivial leaf node yet?
            KD_TRIVIAL = new ANNkd_leaf(0, IDX_TRIVIAL);      // allocate it
    }
    ANNbool ANNorthRect::inside(int dim, ANNpoint p)
    {
            for (int i = 0; i < dim; i++)
                    if (p[i] < lo[i] || p[i] > hi[i])
                            return ANNfalse;
            return ANNtrue;
    }
    void annEnclRect(
                    ANNpointArray       pa,     // point array
                    ANNidxArray         pidx,   // point indices
                    int                 n,      // number of points
                    int                 dim,    // dimension
```

```
                ANNorthRect         &bnds)// bounding cube (returned)
{
    for (int d = 0; d < dim; d++)
    {       // find smallest enclosing rectangle
        ANNcoord lo_bnd = PA(0,d);  // lower bound on dimension d
        ANNcoord hi_bnd = PA(0,d);  // upper bound on dimension d
        for (int i = 0; i < n; i++)
        {
            if (PA(i,d) < lo_bnd)
                lo_bnd = PA(i,d);
            else
                if (PA(i,d) > hi_bnd)
                    hi_bnd = PA(i,d);
        }
        bnds.lo[d] = lo_bnd;
        bnds.hi[d] = hi_bnd;
    }
}

ANNpoint annCopyPt(int dim, ANNpoint source)        // copy point
{
    ANNpoint p = new ANNcoord[dim];
    for (int i = 0; i < dim; i++)
        p[i] = source[i];
    return p;
}
ANNkd_ptr rkd_tree(                 // recursive construction of kd-tree
  ANNpointArray     pa,             // point array
  ANNidxArray       pidx,           // point indices to store in subtree
  int               n,              // number of points
  int               dim,            // dimension of space
```

```
int                 bsp,        // bucket space
ANNorthRect         &bnd_box,   // bounding box for current node
ANNkd_splitter      splitter)   // splitting routine
{
    if (n <= bsp)
    {                                   // n small, make a leaf node
        if (n == 0)                     // empty leaf node
            return KD_TRIVIAL;          // return (canonical) empty leaf
        else                            // construct the node and return
            return new ANNkd_leaf(n, pidx);
    }
    else
    {                                   // n large, make a splitting node
        int cd;                         // cutting dimension
        ANNcoord cv;                    // cutting value
        int n_lo;                       // number on low side of cut
        ANNkd_node *lo, *hi;            // low and high children
                                        // invoke splitting procedure
        (*splitter)(pa, pidx, bnd_box, n, dim, cd, cv, n_lo);
        ANNcoord lv = bnd_box.lo[cd];   // save bounds for cutting
        dimension
        ANNcoord hv = bnd_box.hi[cd];
        bnd_box.hi[cd] = cv;            // modify bounds for left subtree
        lo = rkd_tree(pa, pidx, n_lo,dim, bsp, bnd_box, splitter);
        bnd_box.hi[cd] = hv;            // restore bounds

        bnd_box.lo[cd] = cv;            // modify bounds for right subtree
        hi = rkd_treepa, pidx + n_lo, n-n_lo,// ...from pidx[n_lo..n-1]
        dim, bsp, bnd_box, splitter);
        bnd_box.lo[cd] = lv;            // restore bounds
        ANNkd_split *ptr = new ANNkd_split(cd, cv, lv, hv, lo, hi);
```

```
            return ptr;                    // return pointer to this node
        }
    }
    void kd_split(
        ANNpointArray       pa,            // point array (permuted on return)
        ANNidxArray         pidx,          // point indices
        const ANNorthRect   &bnds,         // bounding rectangle for cell
        int                 n,             // number of points
        int                 dim,           // dimension of space
        int                 &cut_dim,      // cutting dimension (returned)
        ANNcoord            &cut_val,      // cutting value (returned)
        int                 &n_lo)         // num of points on low side (returned)
    {
                                           // find dimension of maximum spread
        cut_dim = annMaxSpread(pa, pidx, n, dim);
        n_lo = n/2;                        // median rank
                                           // split about median
        annMedianSplit(pa, pidx, n, cut_dim, cut_val, n_lo);
    }

    int annMaxSpread(                      // compute dimension of max spread
            ANNpointArray   pa,            // point array
            ANNidxArray     pidx,          // point indices
            int             n,             // number of points
            int             dim)           // dimension of space
    {
        int max_dim = 0;                   // dimension of max spread
        ANNcoord max_spr = 0;              // amount of max spread
        if (n == 0) return max_dim;        // no points, who cares?
        for (int d = 0; d < dim; d++)
        {
```

```
            ANNcoord spr = annSpread(pa, pidx, n, d);
            if (spr > max_spr)
            {
                max_spr = spr;
                max_dim = d;
            }
        }
        return max_dim;
}


void annMedianSplit(
        ANNpointArray   pa,         // points to split
        ANNidxArray     pidx,       // point indices
        int             n,          // number of points
        int             d,          // dimension along which to split
        ANNcoord        &cv,        // cutting value
        int             n_lo)       // split into n_lo and n-n_lo
{
        int l = 0;                  // left end of current subarray
        int r = n-1;                // right end of current subarray
        while (l < r)
        {
            register int i = (r+l)/2;   // select middle as pivot
            register int k;

            if (PA(i,d) > PA(r,d))          // make sure last > pivot
                PASWAP(i,r)
            PASWAP(l,i);                    // move pivot to first position
            ANNcoord c = PA(l,d);           // pivot value
            i = l;
```

```
        k = r;
        for(;;)
        {
                while (PA(++i,d) < c) ;
                while (PA(--k,d) > c) ;
                if (i < k) PASWAP(i,k) else break;
        }
        PASWAP(l,k);                    // pivot winds up in location k
        if (k > n_lo)
                r = k-1; // recurse on proper subarray
        else
                if (k < n_lo)
                        l = k+1;
                else
                        break;          // got the median exactly
}
if (n_lo > 0)
{                                       // search for next smaller item
        ANNcoord c = PA(0,d);           // candidate for max
        int k = 0;                      // candidate's index
        for (int i = 1; i < n_lo; i++)
        {
                if (PA(i,d) > c)
                {
                        c = PA(i,d);
                        k = i;
                }
        }
        PASWAP(n_lo-1, k);      // max among pa[0..n_lo-1] to pa[n_lo-1]
}
cv = (PA(n_lo-1,d) + PA(n_lo,d))/2.0;
```

```
        cv = (PA(n_lo-1,d) + PA(n_lo,d))/2.0;
}
```

3.3 โปรแกรมย่อยในการค้นหาข้อมูลใน Kd-Tree

```
void ANNkd_tree::annkSearch(
        ANNpoint        q,          // the query point
        int             k,          // number of near neighbors to return
        ANNidxArray     nn_idx,     // nearest neighbor indices (returned)
        ANNdistArray    dd,         // the approximate nearest neighbor
        double          eps)        // the error bound
{
        ANNkdDim = dim;             // copy arguments to static equivs
        ANNkdQ = q;
        ANNkdPts = pts;
        ANNptsVisited = 0;          // initialize count of points visited
        if (k > n_pts)
            annError("Requesting more near neighbors than data points", ANNabort);
        ANNkdMaxErr = ANN_POW(1.0 + eps);
        FLOP(2)                     // increment floating op count
        ANNkdPointMK = new ANNmin_k(k);  // create set for closest k points
                                    // search starting at the root
        root->ann_search(annBoxDistance(q, bnd_box_lo, bnd_box_hi, dim));
        for (int i = 0; i < k; i++)
        {
            dd[i] = ANNkdPointMK->ith_smallest_key(i);
            nn_idx[i] = ANNkdPointMK->ith_smallest_info(i);
        }
        delete ANNkdPointMK;        // deallocate closest point set
}

void ANNkd_split::ann_search(ANNdist box_dist)
```

```
        if (ANNmaxPtsVisited && ANNptsVisited > ANNmaxPtsVisited) return;
        ANNcoord cut_diff = ANNkdQ[cut_dim] - cut_val;
        if (cut_diff < 0)
        {
                child[LO]->ann_search(box_dist);// visit closer child first
                ANNcoord box_diff = cd_bnds[LO] - ANNkdQ[cut_dim];
                if (box_diff < 0)          // within bounds - ignore
                        box_diff = 0;
                box_dist = (ANNdist) ANN_SUM(box_dist,
                ANN_DIFF(ANN_POW(box_diff), ANN_POW(cut_diff)));
                if (box_dist * ANNkdMaxErr < ANNkdPointMK->max_key())
                        child[HI]->ann_search(box_dist);
        }
        else {                              // right of cutting plane
                child[HI]->ann_search(box_dist);// visit closer child first
                ANNcoord box_diff = ANNkdQ[cut_dim] - cd_bnds[HI];
                if (box_diff < 0)          // within bounds - ignore
                        box_diff = 0;
                box_dist = (ANNdist) ANN_SUM(box_dist,
                ANN_DIFF(ANN_POW(box_diff), ANN_POW(cut_diff)));
                if (box_dist * ANNkdMaxErr < ANNkdPointMK->max_key())
                        child[LO]->ann_search(box_dist);
        }
        FLOP(10)                            // increment floating ops
        SPL(1)                              // one more splitting node visited
}
void ANNkd_leaf::ann_search(ANNdist box_dist)
{
        register ANNdist dist;          // distance to data point
        register ANNcoord* pp;          // data coordinate pointer
        register ANNcoord* qq;          // query coordinate pointer
```

```
        register ANNdist min_dist;          // distance to k-th closest point
        register ANNcoord t;
        register int d;
        min_dist = ANNkdPointMK->max_key();     // k-th smallest distance so far
        for (int i = 0; i < n_pts; i++)
        {       // check points in bucket
                pp = ANNkdPts[bkt[i]];      // first coord of next data point
                qq = ANNkdQ;                        // first coord of query point
                dist = 0;
                for(d = 0; d < ANNkdDim; d++)
                {
                        COORD(1)                    // one more coordinate hit
                        FLOP(4)                     // increment floating ops
                        t = *(qq++) - *(pp++);  // compute length and adv coordinate
                        if( (dist = ANN_SUM(dist, ANN_POW(t))) > min_dist)
                                break;
                }
        if (d >= ANNkdDim &&ANN_ALLOW_SELF_MATCH || dist!=0))
        {
                ANNkdPointMK->insert(dist, bkt[i]);
                min_dist = ANNkdPointMK->max_key();
        }
        LEAF(1)                             // one more leaf node visited
        PTS(n_pts)                          // increment points visited
        ANNptsVisited += n_pts;             // increment number of points visited
}
ANNdist annBoxDistance(                     // compute distance from point to box
        const ANNpoint      q,              // the point
        const ANNpoint      lo,             // low point of box
        const ANNpoint      hi,             // high point of box
        int                 dim)            // dimension of space
```

```
{
        register ANNdist dist = 0.0;     // sum of squared distances
        register ANNdist t;
        for (register int d = 0; d < dim; d++)
        {
                if (q[d] < lo[d]) {                    // q is left of box
                        t = ANNdist(lo[d]) - ANNdist(q[d]);
                dist = ANN_SUM(dist, ANN_POW(t));
        }
        else
                if (q[d] > hi[d])
                {
                        t = ANNdist(q[d]) - ANNdist(hi[d]);
                        dist = ANN_SUM(dist, ANN_POW(t));
                }
    }
    FLOP(4*dim)                                // increment floating op count
    return dist;
}
```

ภาคผนวก ข

ผลงานวิจัยที่ได้รับการตีพิมพ์

ผลงานวิจัยเรื่อง "Unmixed Vowels Recognition in Thai spoken language using Vocal Tract Transfer functions on the Bark scale" ได้นำเสนอและตีพิมพ์ในงานประชุมวิชาการนานาชาติ Wireless Personal Multimedia Communications ครั้งที่ 3 (WPMC'00) ซึ่งจัดประชุมโดย สถาบันเทคโนโลยีแห่งเอเชีย (AIT)

Unmixed Vowels Recognition in Thai spoken language using Vocal Tract Transfer functions on the Bark scale

K.Songwatana and W.Kongkavitool

Faculty of Engineering ,King Mongkut's Institute of Technology, Ladkrabang
3-2 Chalongkrung Road, Ladkrabang, Bangkok 10520
phone:662 3269967;email:kskraisi@kmitl.ac.th

Abstract **- In this paper we address the problem of unmixed vowels recognition in Thai spoken language. The vowels to be recognized are "i", "e", "ɛ", "ɯ", "ɣ", "a", "u", "o", and "ɔ". Each vowel is represented by a 18-dimensional vector of Critical Band Intensities of the vocal tract transfer function. Using the K-Nearest Neighbor rule on 21,600 utterances from 5 males and 5 females, the success rate of 92 % for speaker independent was achieved. Using 2,160 utterance from one single Thai subject, more than 96% accuracy for speaker dependent has been achieved**

Keywords: **Vowel Recognition, Thai Speech Recognition, Bark Scale.**

1. INTRODUCTION

There are 24 vowels on Thai spoken language. The 24 vowels consist of 9 short unmixed utterances, ("i", "e", "ɛ", "ɯ", "ɣ", "a", "u", "o", and "ɔ") 9 long unmixed utterances, ("i:", "e:", "ɛ:", "ɯ:", "ɣ:", "a:", "u:", "o:", and "ɔ:") and 6 mixed utterances ("ia", "ɯa", "ua", "ia:", "ɯa:", "ua:"). 9 utterances from the unmixed vowel form the bases to compose the rest of the vowels. The long unmixed vowels are repetition of the same vowels over a longer duration with gradual release of vocal cord input. The short unmixed vowels have shorter duration and more immediate stop of vocal cord input. The mixed vowels use 2 basic vowels, one after another to form a new vowel. Therefore to recognize Thai spoken vowels, the unmixed vowels have to be first recognized.

This paper presents recognition method for Thai unmixed vowels based on the vocal tract transfer functions that can be directly derived from Linear Predictive Coding coefficients. From the human auditory system, we can estimate the frequency selectivity of the hearing system by approximating its Critical Band Intensities. The vocal tract transfer functions are mapped onto the Critical Band Rate Scale(Bark Scale) and the respective Critical Band Intensities are used as feature vectors. Based on these feature vectors, a K-Nearest Neighbor rule is applied to classify the unmixed vowels.

2. FEATURE EXTRACTION

The speech producing block diagram in Figure 1 shows that the difference between each spoken vowel is coming from the vocal tract transfer function (H(z)). Therefore we can recognize the vowel by recognizing the vocal tract transfer function.

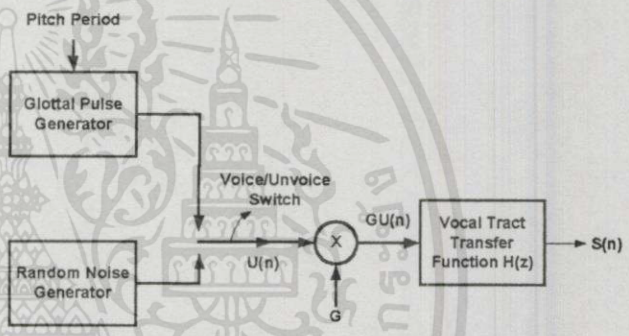


Figure 1. Speech Production Block diagram.

2.1 Obtaining the vocal tract transfer function from Linear Predictive Coding coefficients

In LPC, a given speech sample at time n, s(n), can be approximated as a linear combination of the past p samples[2]

$$s(n) \approx \sum_{k=1}^{p} a_k s(n-k) \qquad (1)$$

where a_k is the LPC coefficient of the k^{th} delay sample. We can convert (1) to an equality by including an excitation term, GU(n), according to Figure 1.

$$s(n) = \sum_{k=1}^{p} a_k s(n-k) + GU(n) \qquad (2)$$

where U(n) is a normalized excitation and G is the gain of the excitation. By expressing (6) in the z-domain we get the relation

$$S(z)=\sum_{k=1}^{p} a_k z^{-1} S(z)+GU(z) \quad (3)$$

which leads to the predicted vocal tract transfer function

$$H(z)=\frac{S(z)}{GU(z)}=\frac{1}{1-\sum_{k=1}^{p} a_k z^{-k}} \quad (4)$$

2.2 Bark Scale or Critical Band Rate Scale

The Bark Scale or Critical Band Rate Scale is a psychoacoustics based spectral measure that best fit the human hearing. The spectral energy over the Bark Scale represents a close correspondence with the spectral information processing in the human ear.[3] The following expression have proven useful to describe the dependence of Critical Band Rate on frequency over the whole auditory frequency range :

$$B(Bark) = 13\arctan(0.76f / KHz) + 3.5\arctan(f / 7.5KHz)^2 \quad (5)$$

where f is frequency in Hz

2.3 Critical Band Intensities as feature vectors

The frequency selectivity of human hearing system can be approximated by subdividing the Intensity of sound into parts that fall into Critical Bands. The Critical Band Intensity, I_B, can be calculated by the following equation

$$I_B = \int_{B-0.5Bark}^{B+0.5Bark} \frac{dI}{dB} dB \quad (6)$$

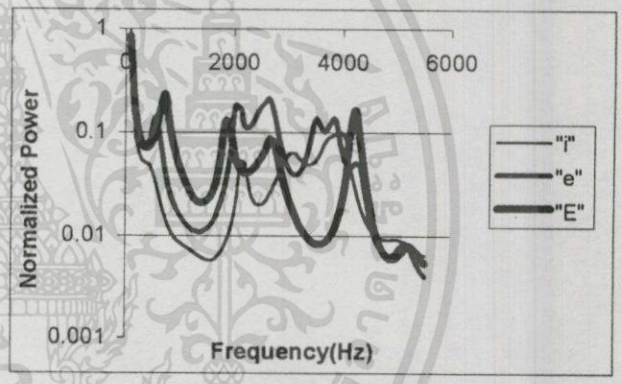
In this paper, the speech is sampled at a rate of 11.025 KHz. The highest frequency of speech sampled is 5.50125 KHz which falls into the 18th Bark. Therefore, 18 Critical Band Intensities(1 to 18) are used as components of the feature vectors.

3. K-NEAREST NEIGHBOR CLASSIFICATION

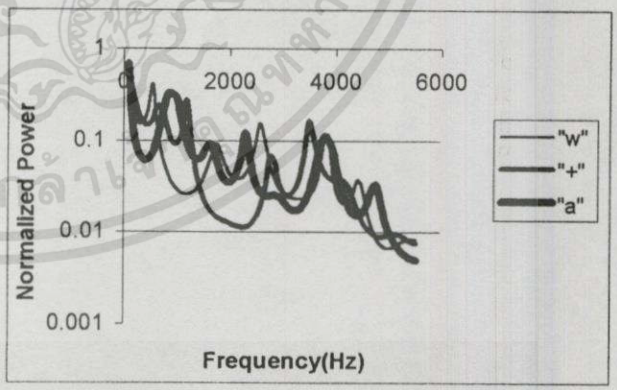
The K-NN algorithm [4] is a well-known algorithm for pattern classification. The set of 18-dimensional feature vectors of Critical Band Intensities from N training utterances $I=\{I_1, I_2, ..., I_N\}$, belong to one of the 9 classes of unmixed vowels. Each class is described by a set of 18-dimensional feature vectors $I_i^r = [i_{i1}^r, i_{i2}^r, ..., i_{i18}^r]$, i=1...N. An unknown feature vector of 18-dimensional Critical Band Intensities, $I_{Unknown}=[i_1, i_2, ..., i_{18}]$, must be classified into one of the 9 classes by computing the distance $d(I_{Unknown}, I_i)$ between the unknown vector $I_{Unknown}$ and all vector I_i for i=1...N. The first k=9 vectors [5] with the least distance from the unknown are selected. The decision is then made to classify the unknown vector to be in the class (unmixed vowel) in which the largest of the 9 selected vectors belong to.

4. EXPERIMENTS

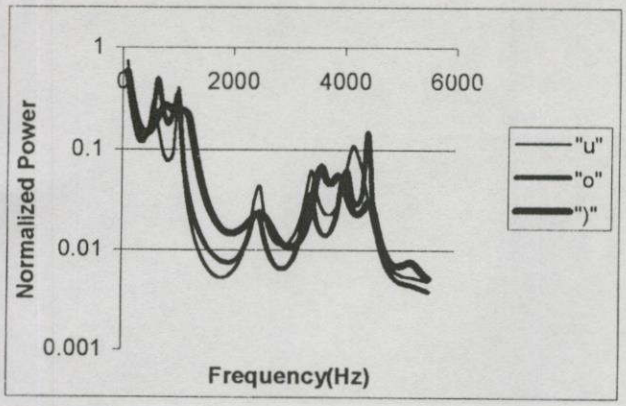
We have collected 21,600 speech samples from 5 men and 5 women. In the feature extraction process, we have obtained the vocal tract transfer function of each frame in all speech samples using the Linear Predictive Coding of order 19[7]. Samples of Transfer Function for all 9 vowels are presented in the group of "front vowels" ("i", "e", "ɛ"), "middle vowels" ("ω", "ɣ", "a") and "back vowels" ("u", "o", "ɔ") [9] in Figure 2(a), 2 (b), and 2(c) respectively.



(a) Front vowels



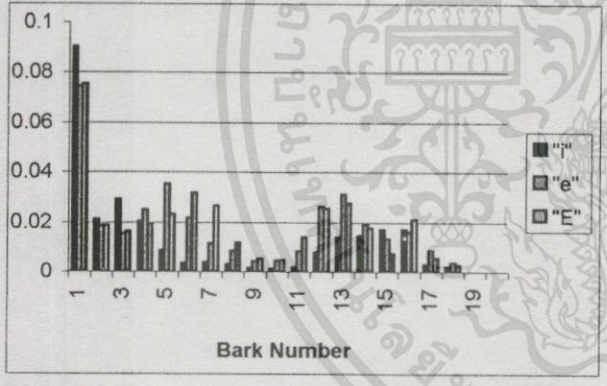
(b) Middle vowels



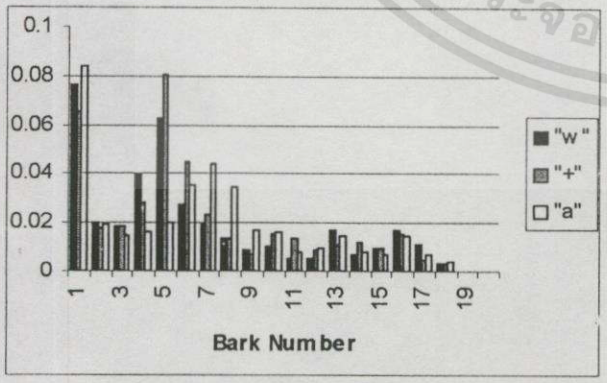
(c) Back vowels

Figure 2. Examples of vocal tract transfer functions

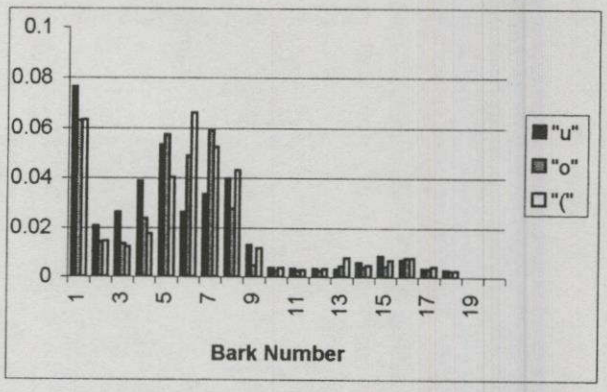
The 'front', 'middle', and 'back' vowels indicate the position of the tongue in the mouth cavity while the vowel is being created. Figure 2 indicates that there maybe enough differences in vocal tract transfer functions for pattern classification. We then map the vocal tract transfer functions to the Bark Scale and calculate the Critical Band Intensity of each Bark by equation (6). The resultants 18 Critical Band Intensities are used as 18-dimensional feature vectors. The examples for the 'front', 'middle', and 'back' vowels are shown in Figure 3(a), (b), and (c).



(a) Front vowels



(b) Middle vowels



(c) Back vowels

Figure 3 Examples of Critical Band Intensities

Large amount of feature vectors obtained from 21,600 speech samples have been divided into 2 sets, 10,800 samples of training set and 10,800 samples of test set. The training set is used to create 2 types of reference models: Speaker Dependent and Speaker Independent.

For the speaker dependent, 2 experiments have been carried out for 1 man and 1 woman. 1,080 speech samples from 1 person are used as training data for create reference model by the K-NN method as mentioned earlier. The rest 1,080 speech samples are used as test set. The recognition results are shown in Table 1(a) and (b).

Table 1. Speaker Dependent Recognition Result

Test	Model	Front			Middle			Back			(%) Accuracy
		"ɪ"	"e"	"ɛ"	"ɯ"	"ɣ"	"a"	"u"	"o"	"ɔ"	
Front	"ɪ"	236	1	3	0	0	0	0	0	0	98.33
	"e"	3	235	2	0	0	0	0	0	0	97.92
	"ɛ"	4	1	233	1	1	0	0	0	0	97.08
Middle	"ɯ"	1	0	0	230	6	2	0	0	1	95.83
	"ɣ"	0	0	0	5	233	2	0	0	0	97.08
	"a"	0	0	1	4	5	227	2	0	1	94.58
Back	"u"	0	0	0	0	0	1	231	6	2	96.25
	"o"	0	0	0	0	2	0	6	228	4	95.00
	"ɔ"	0	1	1	2	0	0	2	5	229	95.42
										Average	96.39

(a) man

Test	Model	Front			Middle			Back			(%) Accuracy
		"ɪ"	"e"	"ɛ"	"ɯ"	"ɣ"	"a"	"u"	"o"	"ɔ"	
Front	"ɪ"	237	0	3	0	0	0	0	0	0	98.75
	"e"	6	230	2	2	0	0	0	0	0	95.83
	"ɛ"	6	3	229	1	1	0	0	0	0	95.42
Middle	"ɯ"	1	0	0	233	4	1	0	0	1	97.08
	"ɣ"	0	0	0	8	227	2	3	0	0	94.58
	"a"	0	0	3	5	8	221	2	0	1	92.08
Back	"u"	0	0	0	0	0	1	235	2	2	97.92
	"o"	0	0	0	0	2	0	6	231	1	96.25
	"ɔ"	0	1	1	5	0	2	2	9	220	91.67
										Average	95.51

(b) woman

In Table 1(a) and (b), the column represents the test utterances in 9 unmixed vowels, the row represents 9 reference models. The tables provide the number of test utterance, in each unmixed vowel, to be classify as

one of the unmixed vowel by the K-NN method, where k=9. Last column shows an average accuracy of 96.39% and 95.51% for the man and the woman models respectively. The error in recognition almost always occurs in its group of "front vowels", "middle vowels", and "back vowels". This is because of the similarity in vocal tract transfer function.

We also have created 2 gender-dependent models for men and women. By combining training sets of the same gender, the gender dependent reference models are created using the training set. Similarly the test data used are from the same gender. The results show in Table 2(a) and (b) for male and female respectively.

Table 2. Gender Dependent Recognition Result

	Model	Front			Middle			Back			(%)
Test		"i"	"e"	"ɛ"	"ɯ"	"ɣ"	"a"	"u"	"o"	"ɔ"	Accuracy
Front	"i"	1166	5	14	0	5	0	10	0	0	97.17
	"e"	13	1157	20	0	10	0	0	0	0	96.42
	"ɛ"	30	5	1157	3	5	0	0	0	0	96.42
Middle	"ɯ"	5	5	0	1127	33	15	0	5	10	93.92
	"ɣ"	0	0	5	25	1148	22	0	0	0	95.67
	"a"	0	0	5	20	37	1113	20	0	5	92.75
Back	"u"	0	0	0	0	0	5	1150	30	15	95.83
	"o"	0	0	0	0	25	5	40	1095	35	91.25
	"ɔ"	0	5	5	10	0	0	16	25	1139	94.92
										Average	94.93

(a) Male

	Model	Front			Middle			Back			(%)
Test		"i"	"e"	"ɛ"	"ɯ"	"ɣ"	"a"	"u"	"o"	"ɔ"	Accuracy
Front	"i"	1155	10	25	10	0	0	0	0	0	96.25
	"e"	40	1113	25	12	10	0	0	0	0	92.75
	"ɛ"	40	15	1135	5	5	0	0	0	0	94.58
Middle	"ɯ"	5	0	0	1150	20	20	0	0	5	95.83
	"ɣ"	0	0	0	50	1132	10	8	0	0	94.33
	"a"	0	0	25	25	40	1095	10	0	5	91.25
Back	"u"	0	0	0	0	15	5	1140	30	10	95.00
	"o"	0	0	0	0	10	15	35	1135	5	94.58
	"ɔ"	0	5	5	25	0	10	19	45	1091	90.92
										Average	93.94

(b) Female

The results show an average accuracy of 94.93% for male and 93.94% for female, a little lower accuracy than speaker dependent.

For speaker independent, all of the 10,800 utterances have been used as a training set and the rest 10,800 utterances have been used as a test set. The result is shown in Table 3.

Table 3. Speaker Independent Recognition Result

	Model	Front			Middle			Back			(%)
Test		"i"	"e"	"ɛ"	"ɯ"	"ɣ"	"a"	"u"	"o"	"ɔ"	Accuracy
Front	"i"	2337	5	28	1	10	0	15	4	0	97.38
	"e"	120	2220	40	0	20	0	0	0	0	92.50
	"ɛ"	60	137	2153	10	40	0	0	0	0	89.71
Middle	"ɯ"	10	10	0	2046	220	44	0	10	60	85.25
	"ɣ"	0	0	17	66	2284	33	0	0	0	95.17
	"a"	0	0	10	40	80	2217	43	0	10	92.38
Back	"u"	0	0	3	0	0	10	2300	57	30	95.83
	"o"	0	0	2	0	80	10	120	2128	60	88.67
	"ɔ"	0	10	10	20	0	40	22	75	2223	92.63
										Average	92.17

The result in Table 3 shows that the speaker independent model provides a lower accuracy rate than speaker-dependent and gender-dependent models. Also, some errors occur across the groups of vowel. This is due to some differences in spectral bands of male and female vocal-tract transfer functions. The average accuracy of 92.17% is achieved.

5. CONCLUSION

By mapping the vocal tract transfer function onto the Critical Band Rate scale and using the K-NN pattern classification process, speaker dependent and speaker independent recognition models can be created to classify the 9 basic unmixed vowels in Thai spoken language. Training samples taken from 5 males and 5 females Thai speaker are used to create the models. Test samples from the same people are taken and used to test the resultant models.

The result shows that the model created from the vocal tract transfer function on the Bark Scale can be used to recognized the unmixed vowel with accuracy up to 96% for speaker dependent and 92% for speaker independent.

REFERENCES

[1] K.Songwatana, I.Arungsrisangchai, C.Charumit, "Tone Recognition Model for Thai language using Pitch Quantization and Hidden Makov Model techniques", *National Conference on Computer Technology and Computer Engineering*, Kasetsart University, Thailand, 1998

[2] Lawrence Rabiner, Biing-Hwang Juang, *Fundamentals of Speech Recognition*, Prentice Hall, New York, 1993.

[3] E.Zwicker, H.Fastl, *Psychoacoustics: Facts and Models* Second Edition, Springer, 1999

[4] Morton Nadler, Eric P. Smith, *Pattern Recognition Engineering*, John Wiley&Sons, 1993.

[5] Inge Gavat, Matei Zirra, "Fuzzy Models for Recognition of Spoken Vowels in Romanian Language", *Fuzzy Systems, 1996., Proceedings of the Fifth IEEE International Conference on* Volume: 3 , 1996 , Page(s): 1552 -1556 vol.3.

[6] David M. Haward, James Angus, *Acoustics and Psychoacoustics*, Prentice Hall, 1996

[7] S.Oraintara, A.Kruafak, "Voice Processing by Linear Prediction", Thesis, Telecommunication Dept, King Mongkhut's Institute of Technology Ladkrabang, Thailand, 1995

[8] Stefan Berchtold, Bernhard Ertl, Daniel A. Kein, Hans-Peter Kriegel, Thomas Seidl, "Fast Nearest Neighbor Search in High Dimensional space", *Data Engineering, 1998. Proceedings., 14th International Conference on* , 1998 , Page(s): 209 -218

[9] A.Thaweesak, *Phonetics*, Language and Culture Research Institute, Mahidol University, Bangkok 1999

ประวัติผู้เขียน

นายวรา คงคาวิทูร เกิดเมื่อวันที่ 5 กุมภาพันธ์ พ.ศ.2520 ที่กรุงเทพมหานคร สำเร็จการศึกษาระดับปริญญาตรี วิศวกรรมศาสตร์บัณฑิต สาขาวิชาวิศวกรรมโทรคมนาคม สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง เมื่อปี พ.ศ.2541