ระบบสั่งงานด้วยเสียงพูด

Voice-to-Command System

โดย

นางสาวหรรษธร ธีรรัตน์รักษ์

นายกรกช เรืองฉาย

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2549

ระบบสั่งงานด้วยเสียงพูด

Voice-to-Command System

โดย

นางสาวหรรษธร ธีรรัตน์รักษ์

นายกรกช เรืองฉาย

อาจารย์ที่ปรึกษา

อาจารย์ชินภัทร นันทจิวากรชัย

ปริญญานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2549

ปริญญานิพนธ์ปีการศึกษา 2549

ภาควิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ระบบสั่งงานด้วยเสียงพูด

Voice-to-Command System

ผู้จัดทำ 1. นางสาวหรรษธร ธีรรัตน์รักษ์ 46010900

2. นายกรกช เรืองฉาย 46010008

.....อาจารย์ที่ปรึกษา

(อาจารย์ชินภัทร นันทจิวากรชัย)

ระบบสั่งงานด้วยเสียง

น.ส. หรรษธร ธีรรัตน์รักษ์ รหัส 46010900
นาย กรกช เรืองฉาย รหัส 46010008
อ.ชินภัทร นันทจิวากรชัย อาจารย์ที่ปรึกษา
ปีการศึกษา 2549

บทคัดย่อ

ปัจจุบัน การสั่งงานควบคุมเครื่องจักรกลขนาดเล็กที่มีฟังก์ชั่นการทำงานหลายอย่าง มักจะต้องใช้ปุ่มหรือคันบังคับที่มีจำนวนมากตามฟังก์ชั่นการทำงานเครื่องจักรนั้นๆ ทำให้ต้องฝึกฝนบุคคลากรเพื่อที่จะเรียนรู้ในการสั่งงานควบคุม และต้องใช้ระยะเวลาในการสร้างความคุ้นเคยกับการใช้ปุ่มหรือคันบังคับที่มีจำนวนมากนั้นอีกด้วย

โครงงานนี้จึงนำเสนอแนวทางการศึกษาการควบคุมเครื่องจักรกลขนาดเล็ก ด้วยการสั่งงานด้วยเสียงพูดภาษาไทยพยางค์เดียว โดยระบบในโครงงานนี้จะใช้คอมพิวเตอร์ในการประมวลผลกระบวนการรู้จำคำสั่งเสียง และส่งผลลัพธ์ออกไปควบคุมโมเดลจำลองของเครื่องจักร เพื่อให้โมเดลดังกล่าวทำงานตามคำสั่งที่ต้องการ

Voice-to-Command System

Miss Hassatorn Theeraratrak 46010900

Mr. Korakot Ruangchai 46010008

Mr.Chinnapat Nantajiwakornchai Advisor

Educational Year 2549

Abstract

Nowadays, there are many functions to control the machineries by using many joysticks and buttons. Therefore, it is necessary for workers to be trained before using the machineries.

This project presents Machinery Controlled with Speech Recognition using Thai speech with 1 syllable to reduce complex functions of machineries control. The system uses personal computer for speech recognition processing. The PC will process and send output to control the machinery model.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงได้เป็นอย่างดี เนื่องจากคำแนะนำ และความช่วยเหลือจากหลายฝ่ายด้วยกัน โดยเฉพาะอย่างยิ่งอาจารย์ที่ปรึกษาที่คอยให้คำปรึกษา และความช่วยเหลือเสมอมา ซึ่งก็คืออาจารย์ ชินภัทร นันทจิวากรชัย ขอขอบพระคุณเป็นอย่างสูง

นอกเหนือจากนี้ต้องขอขอบพระคุณอาจารย์ทุกท่านในภาควิชาวิศวกรรมอิเล็กทรอนิกส์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบังเป็นอย่างยิ่งที่ได้ช่วยประสิทธิ์ประสาทวิชาความรู้ให้แก่คณะผู้จัดทำ และขอขอบคุณเพื่อนๆ ทุกคนที่ให้ความร่วมมือในการอัดเสียงเพื่อบันทึกผลการทดลองและยังคอยเป็นกำลังใจให้กัน

สุดท้ายต้องขอขอบคุณบุคคลที่สำคัญที่สุดในชีวิตที่ทำให้ข้าพเจ้ามีวันนี้ นั่นคือ บิดา มารดา และบุคคลในครอบครัว อันเป็นที่เคารพรัก ซึ่งได้เลี้ยงดู คอยให้คำสั่งสอนข้าพเจ้ามาเป็นอย่างดี พร้อมให้โอกาสในการศึกษาอย่างเต็มที่ และยังให้กำลังใจ ความรักเสมอมา ข้าพเจ้าขอกราบขอบพระคุณมา ณ ที่นี้ด้วย

หรรษธร ธีรรัตน์รักษ์
กรกช เรืองฉาย

สารบัญ

สารบัญรูป

สารบัญตาราง

บทที่ 1
บทนำ

1.1 ความเป็นมาของโครงงาน

ในปัจจุบัน ระบบรู้จำเสียงพูดได้เข้ามามีบทบาทในชีวิตประจำวันของมนุษย์ในหลายทาง เช่น การสั่งงานเครื่องใช้ในชีวิตประจำวันด้วยเสียงพูด ที่สามารถอำนวยความสะดวกให้กับผู้ใช้งานทั่วไปได้มากขึ้น รวมทั้งอำนวยความสะดวกต่อผู้พิการ ที่ปกติไม่สามารถใช้งานอุปกรณ์และเครื่องมือเครื่องใช้ได้สะดวกนัก หรือแม้แต่ระบบรักษาความปลอดภัยโดยใช้เสียงพูด เนื่องจากเสียงพูดของแต่ละคนมีเอกลักษณ์เป็นของตนเอง

โครงงานนี้จึงนำเสนอแนวทางการประยุกต์ใช้งานระบบรู้จำเสียงพูด กับเครื่องจักรกลขนาดเล็กที่มีความยุ่งยากซับซ้อนในการควบคุม เช่น รถโฟล์คลิฟท์ในห้างสรรพสินค้า โดยสั่งงานอุปกรณ์ดังกล่าวด้วยเสียงพูดภาษาไทยพยางค์เดียว โดยระบบในโครงงานนี้จะใช้วิธีการที่เรียกว่า เมลฟรีเควนซี่เซปสตรัลโคเอฟฟิเซี่ยน (Mel Frequency Cepstral Coefficient หรือ MFCC) เพื่อทำการหาลักษณะเด่นของคำสั่งเสียงพูดแต่ละคำ และใช้เทคนิคการรู้จำเสียงพูดที่เรียกว่า ฮิดเด็นมาร์คอฟโมเดล (Hidden Markov Model หรือ HMM) ซึ่งจะต้องทำการเก็บข้อมูลเสียงต้นแบบไว้ก่อน และจึงนำคำสั่งเสียงมาเปรียบเทียบกับข้อมูลเสียงต้นแบบ ระบบจึงจะทำการตัดสินใจ ว่าสัญญาณเสียงที่รับเข้ามานั้นเป็นคำใด และนำไปสู่ควบคุมการทำงานของอุปกรณ์ต่างๆ ตามฟังก์ชั่นการทำงานที่กำหนดไว้

1.2 วัตถุประสงค์

1) เพื่อศึกษากระบวนการหาค่าลักษณะสำคัญของเสียงพูด
2) เพื่อศึกษากระบวนการรู้จำเสียงพูด โดยใช้แบบจำลองฮิดเด็นมาร์คอฟ
3) เพื่อให้ระบบสามารถรู้จำคำสั่งเสียงพูดภาษาไทยพยางค์เดียวที่กำหนด
4) เพื่อเป็นพื้นฐานในการนำระบบรู้จำเสียงพูดไปประยุกต์ใช้งานกับฮาร์ดแวร์ที่มีความซับซ้อนในการควบคุม

1.3 ขอบเขตของโครงงาน

โครงงานนี้แบ่งออกเป็น 2 ส่วน คือ

1) ศึกษากระบวนการรู้จำเสียงพูด โดยทำการหาลักษณะสำคัญของคำสั่งเสียงแต่ละคำ และให้ระบบเรียนรู้คำสั่งดังกล่าว โดยกระบวนการทั้งหมดนี้ใช้คอมพิวเตอร์เป็นตัวประมวลผล
2) นำกระบวนการรู้จำเสียงพูดที่ได้ศึกษา ไปควบคุมการทำงานของฮาร์ดแวร์ภายนอกคอมพิวเตอร์

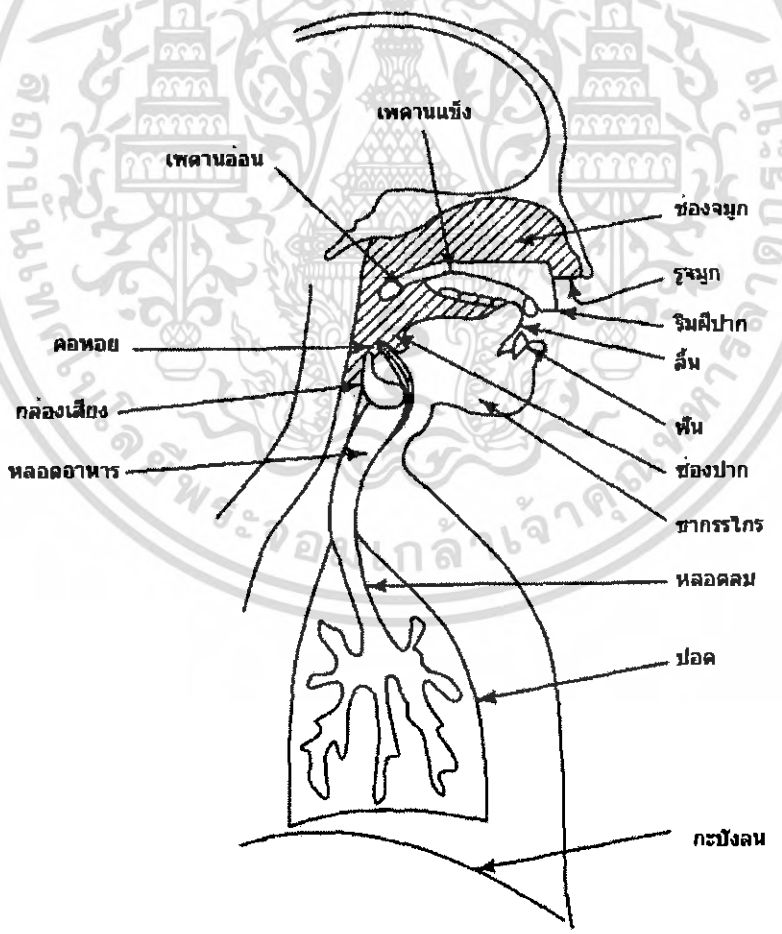
1.4 ประโยชน์ที่คาดว่าจะได้รับ

1) ได้เรียนรู้ขั้นตอนต่าง ๆ ในกระบวนการรู้จำเสียงพูด
2) ได้เรียนรู้การใช้งานโปรแกรมแมทแล็ปในการประมวลผลสัญญาณเสียงพูด
3) สามารถประยุกต์ใช้งานระบบรู้จำเสียงพูดกับฮาร์ดแวร์ที่มีความซับซ้อนในการควบคุม

บทที่ 2
ระบบการเปล่งเสียงของมนุษย์

มนุษย์สามารถเปล่งเสียงพูดได้ด้วยอวัยวะที่ใช้ในการออกเสียง (Organ of Speech) อวัยวะดังกล่าวมีอยู่หลายส่วน แต่ละส่วนสามารถทำให้เกิดเสียงพูดที่แตกต่างกันออกไปได้ ดังรูปที่ 2.1 อวัยวะที่ใช้ในการออกเสียงแบ่งได้เป็น 2 ประเภท คือ

1. อวัยวะที่ใช้ในการกระทำอาการ (Articulator) หมายถึง อวัยวะส่วนที่เคลื่อนไหว เพื่อผลักลมไปยังส่วนต่าง ๆ อวัยวะที่สำคัญในส่วนนี้ คือ ลิ้น ซึ่งเป็นส่วนที่เคลื่อนไหวได้มากที่สุด
2. อวัยวะที่เป็นตำแหน่งที่ทำให้เกิดเสียงต่าง ๆ (Point of Articulator) หมายถึง ตำแหน่งที่เกิดของเสียงต่าง ๆ เช่น ริมฝีปาก ฟัน เพดานส่วนต่าง ๆ เป็นต้น



รูปที่ 2.1 อวัยวะที่ใช้ในการออกเสียง

2.1. อวัยวะที่ใช้ในการออกเสียง (Organ of Speech)

1. ปอดและกะบังลม ทำหน้าที่สำคัญในการหายใจ และเป็นต้นกำเนิดการไหลของอากาศในกระบวนการผลิดเสียง
2. หลอดลม ทำหน้าที่นำอากาศจากปอดผ่านกล่องเสียง และเป็นอวัยวะที่อยู่ด้านหน้าของหลอดอาหาร
3. กล่องเสียง ทำหน้าที่เป็นทางเดินอากาศเวลาหายใจ และเป็นดัวผลิดพัลส์ของอากาศขณะเปล่งเสียง ประกอบด้วยเส้นเสียง (Vocal cords) และช่องสายเสียง (Glottis)
4. ช่องปากและส่วนของหลอดอาหารดอนต้น อวัยวะกลุ่มนี้อยู่ต่อจากกล่องเสียง อาจเรียกว่าอวัยวะกำทอนเสียง (Vocal tract) ทำหน้าที่กำทอนเสียงทั้งที่เกิดจากกล่องเสียงและเสียงที่เกิดภายในช่องปาก
5. โพรงจมูก เริ่มจากเพดานอ่อนจนถึงรูจมูกทั้งสอง ทำหน้าที่กำทอนเสียงร่วมกับช่องปากเมื่อมีการเปล่งเสียงที่ออกจากจมูก (Nasal sounds) เช่น เสียง ม, น, และ ง เป็นต้น

2.2 การเกิดของเสียง (Speech Production)

การเกิดของเสียงแบ่งได้เป็น 3 ขั้นตอนใหญ่ คือ

1. ขั้นเริ่มต้น (Initiation) ขั้นตอนนี้เป็นขั้นตอนที่ลมเริ่มถูกขับออกจากปอด
2. ขั้นตอนดัดแปลงลมที่เส้นเสียง (Phonation) เป็นขั้นดอนที่ลมจากปอดผ่านมายังหลอดลมและกล่องเสียง ซึ่งที่กล่องเสียงนี้ เส้นเสียงจะทำหน้าที่เป็นลิ้นเปิดปิด ทำให้เกิดเสียง 2 ชนิด คือ เสียงก้อง (Voiced) และเสียงไม่ก้อง (Voiceless) อวัยวะที่ใช้ในช่วงนี้ คือ ส่วนที่ต่อจากปอดขึ้นมาจนถึงกล่องเสียง
3. ขั้นดอนเปลี่ยนแปลงลักษณะเสียง (Articulator) ในขั้นตอนนี้ ลมที่ผ่านออกมาจากกล่องเสียง จะถูกแปลงให้เกิดเสียงในลักษณะต่าง ๆ อวัยวะที่ใช้ คือ ส่วนที่ต่อจากกล่องเสียงจนถึงริมฝีปาก

2.3 ประเภทของเสียง

เสียงที่เกิดจากขั้นตอนหัวข้อที่แล้ว จะถูกแบ่งออกเป็น 2 ประเภทใหญ่ ๆ คือ

1. เสียงก้อง (Voiced sound)

เกิดจากการออกเสียงในขณะที่เส้นเสียงถูกดึงเข้ามาใกล้กันจนเกือบปิดช่องทางลมสนิท ลมที่ดันขึ้นมาจากปอดจะทำให้เส้นเสียงสั่น ลมที่ออกมาไม่สะดวกเนื่องจากต้องบีบตัวผ่านช่องแคบเป็นจังหวะ จึงทำให้เกิดเป็นเสียงขึ้น

2. เสียงไม่ก้อง (Voiceless sound)

เป็นการออกเสียงในขณะที่เส้นเสียงยังเปิดกว้าง โดยเปิดช่องระหว่างเส้นเสียงหรือคอหอย ให้ลมหายใจผ่านเข้าออกสะดวกขึ้น หรืออาจกล่าวได้ว่า เป็นเสียงที่เกิดในช่องปากหรือโพรงจมูก โดยที่อวัยวะในช่องปาก ริมฝีปาก จะมาขวางการไหลของอากาศให้ผ่านได้เป็นช่องเล็ก ๆ อากาศจึงไหลผ่านได้อย่างรวดเร็ว

บทที่ 3
ทฤษฎีการรู้จำเสียงพูด

องค์ประกอบหลักของระบบรู้จำเสียงพูด ประกอบด้วยขั้นตอนดังต่อไปนี้

1. การประมวลผลสัญญาณเสียงเบื้องต้น (Preprocessing)
2. การสกัดค่าลักษณะสำคัญของเสียงพูด (Speech Feature Extraction)
3. เทคนิคการรู้จำ (Speech Recognition)

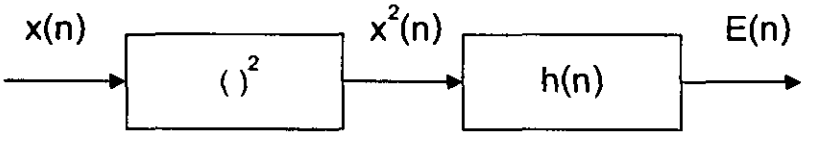
3.1. การประมวลผลสัญญาณเสียงเบื้องต้น (Preprocessing)

เป็นขั้นตอนที่จะทำให้สัญญาณเสียงที่จะนำไปใช้หรือรับเข้ามานั้น มีความสมบูรณ์มากที่สุด ขั้นตอนเหล่านี้ประกอบด้วย การหาจุดเริ่มต้นและจุดสิ้นสุดของสัญญาณ (End-point Detection) การแบ่งช่วงสัญญาณ (Frame Blocking) และการทำวินโดว์ (Windowing)

3.1.1 การหาจุดเริ่มต้นและจุดสิ้นสุดของสัญญาณ (End Point Detection)

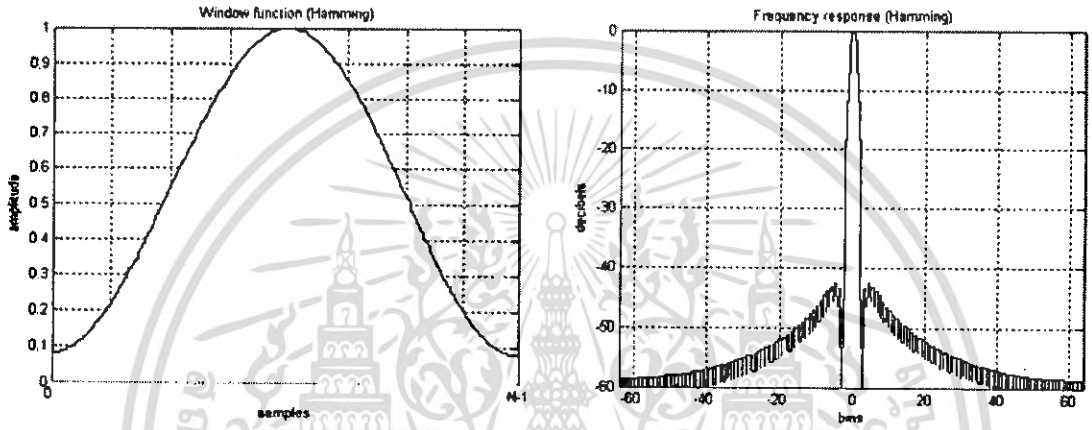
ขั้นตอนนี้เป็นขั้นตอนที่ใช้หาขอบเขตของสัญญาณ เพื่อนำเฉพาะสัญญาณเสียงในช่วงที่เป็นเสียงพูดมาประมวลผลเท่านั้น เนื่องจากสัญญาณเสียงที่รับเข้ามาจากไมโครโฟนประกอบด้วยส่วนของสัญญาณที่เป็นเสียงพูด กับส่วนที่ไม่ใช่เสียงพูดนำหน้าและต่อท้ายสัญญาณเสียงมาด้วย เช่น ช่วงที่ไม่มีเสียงหรือส่วนที่เกิดจากสิ่งแวดล้อม จึงต้องใช้กระบวนการนี้ตัดส่วนที่ไม่ต้องการออกไป

สัญญาณเสียงเป็นสัญญาณที่มีการเปลี่ยนแปลงตลอดเวลา และส่วนที่ไม่ใช่เสียงพูดจะมีขนาดสัญญาณเฉลี่ยต่ำกว่าส่วนที่เป็นเสียงพูด ดังนั้นในการหาจุดเริ่มต้นและจุดสิ้นสุดของสัญญาณ จะมีการตั้งค่าพลังงานขึ้นมาค่าหนึ่ง (Threshold) ซึ่งจะเป็นค่าที่ใช้ตัดสินใจแยกส่วนที่เป็นเสียงและไม่ใช่เสียงออกจากกัน และจะใช้วิธีการหาค่าพลังงานช่วงสั้น (Short Time Energy) มาใช้ในการหาค่าพลังงานช่วงที่ไม่ใช่เสียงพูด ดังแผนภาพในรูปที่ 3.1



รูปที่ 3.1 แผนภาพการหาค่าพลังงานช่วงสั้น (Short Time Energy)

ค่าพลังงานของเสียง หาได้จากการนำค่าขนาดของสัญญาณเสียงมายกกำลังสอง แล้วนำไปถ่วงน้ำหนักด้วย Window function ในที่นี้เลือกใช้ Hamming Window ดังแสดงในรูปที่ 3.2 ค่าพลังงานที่หาได้จะถูกนำมาเปรียบเทียบกับค่าพลังงานที่ได้ตั้งเอาไว้ สมการที่ใช้หาค่าพลังงานนี้แสดงในสมการที่ 1



รูปที่ 3.2 แสดงรูป Hamming Window และผลตอบสนองทางความถี่

สมการหาค่าพลังงานช่วงสั้น

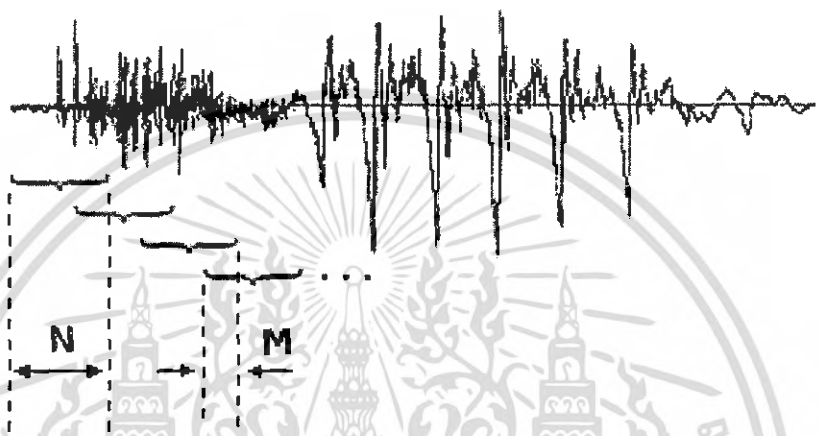
$$E_n = \sum_{m=-\infty}^{\infty} x^2(n) \cdot h(n-m) \quad (1)$$

ฟังก์ชันของ Hamming Window

$$h(n) = 0.54 - 0.46\cos\left(\frac{2\pi n}{(N-1)}\right) \quad (2)$$

3.1.2 การแบ่งช่วงสัญญาณ (Frame Blocking)

ข้อมูลเสียงซึ่งผ่านการตัดส่วนที่ไม่ต้องการออกไปแล้ว จะถูกแบ่งออกเป็นช่วงสัญญาณ (Block) ซึ่งแต่ละช่วงสัญญาณจะประกอบไปด้วยสัญญาณเสียง N สัญญาณ และแต่ละช่วงสัญญาณจะถูกวิเคราะห์โดยการเลื่อนสัญญาณไปครั้งละ M สัญญาณ ดังรูปที่ 3.3



รูปที่ 3.3 แสดงการแบ่งช่วงสัญญาณ

ถ้าค่า M มีค่าน้อยกว่าค่า N มากเท่าใด จะทำให้การวิเคราะห์สัญญาณมีความแม่นยำมากขึ้นเท่านั้น แต่ข้อเสียคือจะต้องใช้เวลาในการวิเคราะห์นานขึ้น และถ้าค่า M มีค่ามากกว่าค่า N จะทำให้สัญญาณบางส่วนไม่ถูกใช้ในการวิเคราะห์ ซึ่งทำให้เกิดการผิดพลาดในการวิเคราะห์สัญญาณส่วนต่อไปได้

3.1.3 การวินโดว์ (Windowing)

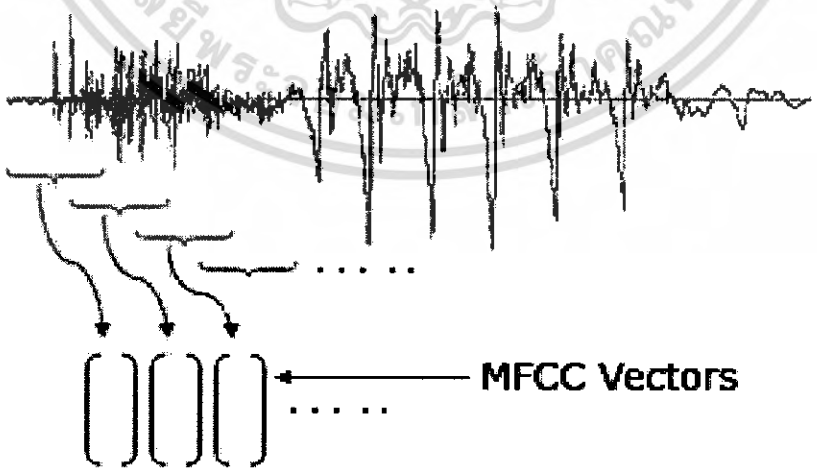
เนื่องจากเราใช้วิธีการตัดสัญญาณออกเป็นช่วง ๆ เพื่อนำมาหาสเป็กตรัม เท่ากับเป็นการหาสเป็กตรัมของบล็อกสัญญาณ จึงอาจทำให้เกิดความคลาดเคลื่อนในสเป็กตรัมที่ได้ วิธีที่จะลดความคลาดเคลื่อนสามารถทำได้โดยคูณสัญญาณแต่ละบล็อกด้วย Window Function ก่อนที่จะทำการหาสเป็กตรัม ในระบบนี้ใช้ Hamming Window เช่นเดียวกับขั้นตอนการหาจุดเริ่มต้นและจุดสิ้นสุดของสัญญาณ

3.2 การสกัดค่าลักษณะสำคัญของเสียงพูด (Speech Feature Extraction)

ในการรู้จำเสียงพูด ระบบไม่ได้เอาสัญญาณเสียงไปเปรียบเทียบกันโดยตรง แต่จะดึงเฉพาะลักษณะสำคัญของเสียงออกมา เราเรียกลักษณะสำคัญนี้ว่า Speech Feature คำพูดของคนเราย่อมจะมีลักษณะเด่นของแต่ละคำแตกต่างกันไป กรณีที่คนหลาย ๆ คนพูดคำพูดเดียวกัน อาจจะมีลักษณะเด่นของคำพูดที่ต่างกันบ้าง แต่จะต้องมีลักษณะเด่นบางอย่างร่วมกัน ขั้นตอนนี้เป็นการหาลักษณะเด่นที่สำคัญของเสียงพูดเพื่อเป็นตัวแทนของคำพูดแต่ละคำ การสกัดหาค่าลักษณะสำคัญของเสียงพูดสามารถทำได้หลายวิธี เช่น วิธีการประมาณพันธะเชิงเส้น (Linear Prediction Coefficient : หรือ LPC), วิธีการหาสัมประสิทธิ์เซปสตรัม (Cepstral Coefficient) และวิธีเวฟเล็ต (Wavelet) แต่ในระบบนี้จะใช้วิธีที่เรียกว่าการหาสัมประสิทธิ์เซปสตรัมที่คำนวณบนแกนความถี่แบบเมล (Mel Frequency Cepstral Coefficient หรือ MFCC)

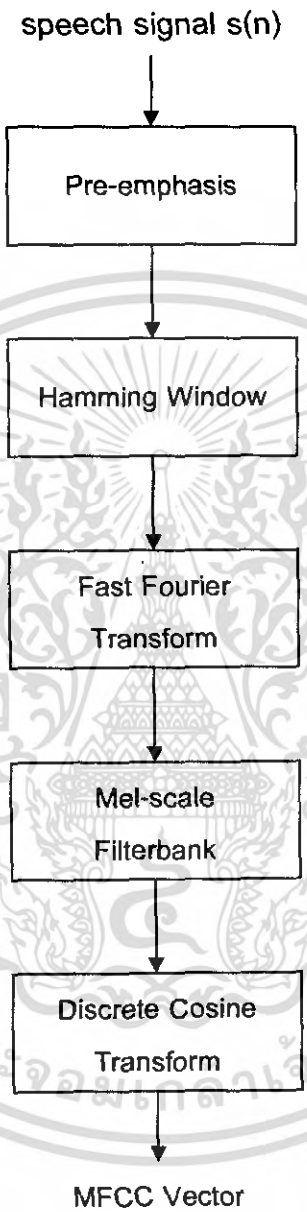
Mel Frequency Cepstral Coefficient (MFCC)

วิธีการหาสัมประสิทธิ์เซปสตรัมที่คำนวณบนแกนความถี่แบบเมล (Mel Frequency Cepstral Coefficient หรือ MFCC) เป็นการแปลงสัญญาณเสียงพูดให้อยู่ในรูปของกลุ่มเวคเตอร์ เช่น 1 เวคเตอร์ใช้แทนสัญญาณเสียงยาวประมาณ 20 มิลลิวินาที แต่ละเวคเตอร์ก็แทนสัญญาณเสียงที่ค่อย ๆ เลื่อนไปแบบ overlap เช่นเลื่อนไปทีละ 10 มิลลิวินาที ดังแสดงในรูปที่ 3.4 ดังนั้นหากมีเสียงที่ยาวซัก 1 วินาทีเข้ามา ก็จะแทนด้วยเวคเตอร์จำนวน 100 เวคเตอร์ เราเรียกเวคเตอร์นี้ว่า MFCC Vector



รูปที่ 3.4 แสดงสัญญาณเสียงที่ถูกเปลี่ยนเป็น MFCC Vectors

การเปลี่ยนสัญญาณเสียงพูดให้เป็นเวคเตอร์มีขั้นตอนดังบล็อกไดอะแกรมในรูปที่ 3.5



รูปที่ 3.5 บล็อกไดอะแกรมของกระบวนการ MFCC

3.2 1 กระบวนการ Pre-emphasis

สัญญาณเสียงที่ผ่านการประมวลผลเบื้องต้นแล้ว จะนำมาผ่านกระบวนการ Pre-emphasis ซึ่งเป็นการนำมาผ่าน FIR Filter ทำให้สัญญาณที่ได้มีความราบเรียบมากขึ้น ด้วยสมการที่ 3

$$x(n) = s(n) * p(n) \quad (3)$$

โดย p(n) เป็น FIR Filter ซึ่งมีฟังก์ชันตามสมการที่ 4

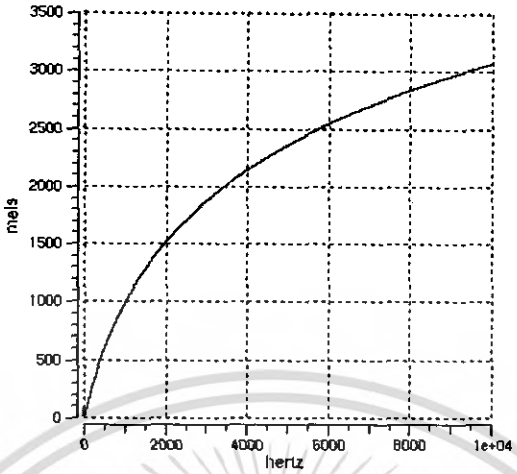
$$p(n) = 1 - \alpha(n-1) \quad (4)$$

โดยที่ α เป็นค่าถ่วงน้ำหนัก มีค่าเท่ากับ 0.95

3.2.2 กระบวนการ Mel-Frequency Scaling and Cepstrum

เนื่องจากระบบการได้ยินของมนุษย์จะมีลักษณะไม่เป็นเชิงเส้น จึงต้องมีการทำ Frequency Scaling เป็นการเปลี่ยนสเกลความถี่เชิงเส้นให้เป็นความถี่ในสเกลที่มนุษย์ได้ยิน นั่นคือความถี่ในสเกลแบบเมล (Mel Frequency Scale) ในช่วงความถี่ต่ำกว่า 1 kHz จะเป็นเชิงเส้น แต่ช่วงความถี่ที่สูงกว่านั้นจะเป็นแบบ logarithm โดยความถี่เมลกับความถี่เชิงเส้นมีความสัมพันธ์กันตามสมการที่ 5

$$Mel = 1127 \ln\left(1 + {}^{Freq}/_{700}\right) \quad (5)$$

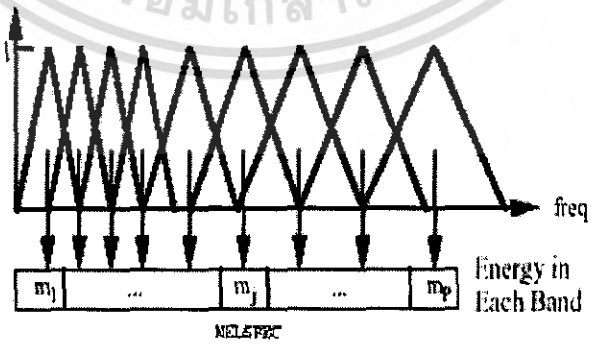


รูปที่ 3.6 กราฟความสัมพันธ์ระหว่างความถี่เชิงเส้น (Hertz) กับความถี่เมล (Mel)

สัญญาณเสียงที่ผ่านการสเกลทางความถี่แล้ว จะถูกนำมาแปลง Discrete Fourier Transform (DFT) เพื่อหาสเป็กตรัมของสัญญาณ ดังสมการที่ 6

$$Y(f) = \sum_{n=0}^{N-1} y(n) \cdot e^{-j2\pi n/N} \qquad (6)$$

สัญญาณที่ผ่านการแปลง DFT ของแต่ละวินโดว์จะถูกนำมาผ่าน Mel-frequency Filterbank ซึ่งเป็นกลุ่มของ Band Pass Filter โดยกำหนดให้ Mel-frequency Filterbank แทนด้วย Y_k ($k = 1, 2, 3, \ldots, K$) ซึ่งเป็น Triangular Filter ดังรูปที่ 3.7



รูปที่ 3.7 Mel-frequency Filterbank

โดยสัญญาณที่แปลง DFT แล้ว จะนำมาคูณกับค่า Filter Gain ดังสมการที่ 7

$$Y_k = \sum_{f=0}^{N-1} |Y(f)| \cdot H_k(f) \quad (7)$$

สุดท้ายก็จะนำเอา Mel-frequency Cepstrum ไปทำ Discrete Cosine Transform (DCT) เพื่อให้ได้เป็น MFCC Vectors ดังสมการที่ 8

$$c_i = \sqrt{\frac{2}{N}} \sum_{k=0}^{K} Y_k \cdot \cos\left(\frac{\pi i}{N}(k-0.5)\right) \quad (8)$$

3.3 เทคนิคการรู้จำ (Speech Recognition)

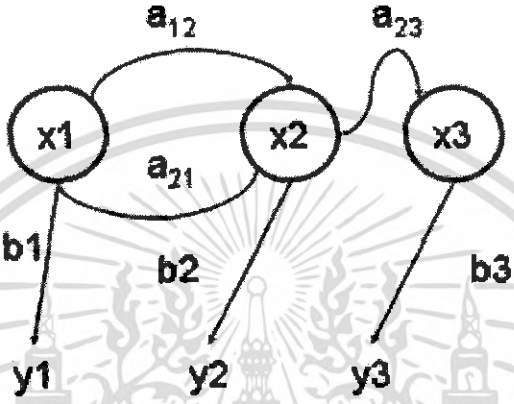
เมื่อได้ค่าลักษณะสำคัญของเสียงพูดซึ่งอยู่ในรูปของ MFCC Vector มาแล้ว ก็จะนำค่าที่ได้มาวิเคราะห์ว่าเป็นคำพูดใด ในระบบนี้ใช้เทคนิคการรู้จำเสียงพูดโดยใช้แบบจำลองฮิดเด็นมาร์คอฟ (Hidden Markov Model หรือ HMM)

แบบจำลองฮิดเด็นมาร์คอฟ (Hidden Markov Model : HMM)

แบบจำลองฮิดเด็นมาร์คอฟ คือ การเก็บรวบรวมของสถานะหลาย ๆ สถานะที่ถูกเชื่อมโยงโดยการเปลี่ยนสถานะ ในที่นี้สถานะ คือ ลักษณะของสัญญาณเสียงหรือเครื่องหมายที่หมายถึงสัญญาณเสียงนั้น ณ เวลาหนึ่ง ๆ ส่วนการเปลี่ยนสถานะ คือ การเลื่อนหรือเปลี่ยนแปลงของสัญญาณเสียงจากเวลาหนึ่งไปหาอีกเวลาหนึ่ง ในแบบจำลองฮิดเด็นมาร์คอฟมีค่าความน่าจะเป็นอยู่สองชนิด คือ ค่าความน่าจะเป็นที่การเปลี่ยนสถานะหนึ่งจะเกิดขึ้น (Transition probability) และค่าความน่าจะเป็นของเอาท์พุท (Output probability) เมื่อมีการเปลี่ยนสถานะหนึ่งเกิดขึ้น เอาท์พุทที่ว่านี้จะนำไปใช้ในการตัดสินใจว่าเสียงที่ได้ยินนั้นเป็นเสียงอะไร

รูปที่ 3.8 Left-to-Right Hidden Markov Model

กระบวนการในแบบจำลองฮิดเด็นมาร์คอฟ เป็นการหาค่าตัวแปรที่ซ่อนอยู่ (Hidden parameter) จากค่าตัวแปรที่เราสามารถทราบค่าได้ (Observable parameter) เราไม่สามารถมองเห็นแต่ละสถานะ (State) ของระบบได้โดยตรง แต่เราสามารถมองเห็นตัวแปรที่เป็นผลมาจากสถานะนั้น ๆ ได้ โดยแต่ละสถานะจะมีความน่าจะเป็นที่จะเป็นไปได้ของเอาท์พุทอยู่ ดังรูปที่ 3.9



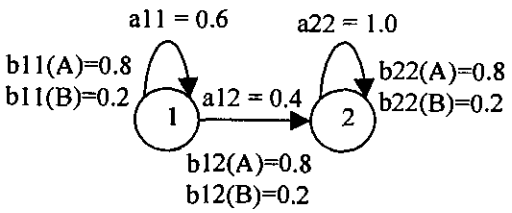
รูปที่ 3.9 แผนภาพการเปลี่ยนสถานะในกระบวนการ HMM

โดยที่ x คือ Hidden states

y คือ Observable outputs

a คือ ความน่าจะเป็นที่จะเกิดการเปลี่ยนสถานะ (Transition probability)

b คือ ความน่าจะเป็นของเอาท์พุท (Output probability)



รูปที่3.10 แบบจำลองฮิดเด็นมาร์คอฟ ที่มี 2 สถานะ และมี 2 เอาท์พุท

รูปที่ 3.10 แสดงถึงตัวอย่างของแบบจำลองฮิดเด็นมาร์คอฟที่มี 2 เอาท์พุตคือ A และ B และ 2 สถานะ คือ 1 และ 2 และมีการกำหนดค่าความน่าจะเป็นของ a และ b มาแล้ว ค่า a12=0.4 หมายความว่า เมื่อสถานะปัจจุบันเป็น 1 สถานะต่อไปเป็น 2 จะมีความน่าจะเป็นอยู่ 0.4 ส่วน b12(A) = 0.8 คือ เมื่อมีการเปลี่ยนสถานะจาก 1 ไป 2 ความน่าจะเป็นที่จะมีเอาท์พุทเป็น A คือ 0.8 (เมื่อมีการเปลี่ยนสถานะจะมีการสร้างเอาท์พุทใดเอาท์พุทหนึ่งออกมาเท่านั้น)

$$a_{ij} = P\left(X_{t+1} = j \middle| X_t = i\right) \tag{9}$$

$$b_{ij}(k) = P\left(Y_t = k \middle| X_t = i, X_{t+1} = j\right) \tag{10}$$

a_{ij} คือ ความน่าจะเป็นของสถานะถัดไป j เมื่อสถานะปัจจุบันเป็น i

$b_{ij}(k)$ คือ ความน่าจะเป็นของเอาท์พุทเป็น k เมื่อมีการเปลี่ยนสถานะจากสถานะ i ไป j

คุณสมบัติของ a และ b มีดังนี้

1. $a_{ij} \geq 0 \quad , \quad b_{ij}(k) \geq 0 \quad , \quad \forall i, j, k$
2. $\sum_j a_{ij} = 1$
3. $\sum_k b_{ij}(k) = 1$

สำหรับแบบจำลองฮิดเด็นมาร์คอฟลำดับที่ 1 จะต้องมีสมมุติฐานอยู่สองประการ คือ

1. ความน่าจะเป็นของสถานะถัดไปขึ้นอยู่กับสถานะปัจจุบันเท่านั้น
2. ความน่าจะเป็นของเอาท์พุทถัดไปขึ้นอยู่กับการเปลี่ยนสถานะปัจจุบันเท่านั้น ไม่ใช่การเปลี่ยนสถานะในอดีต

3.3.1 ปัญหาของแบบจำลองฮิดเด็นมาร์คอฟ

ปัญหาของแบบจำลองฮิดเด็นมาร์คอฟ มีอยู่สามข้อด้วยกัน คือ

1. Evaluation Problem

แบบจำลองสามารถสร้างหรือเข้าใจเสียงที่ได้ยินดีแค่ไหน เรียกว่า Evaluation Problem ปัญหานี้สามารถแก้ไขโดยใช้ Forward Algorithm ซึ่งจะทำให้แบบจำลองมีลักษณะเดียวกันกับเสียงที่ได้ยิน ซึ่งสามารถใช้ในการรู้จำเสียงพูดที่ไม่ติดต่อกันได้

2. Decoding Problem

ลำดับของสถานะในแบบจำลอง สร้างเสียงที่ได้ยินดีแค่ไหน เรียกว่า Decoding Problem ปัญหานี้สามารถแก้ไขโดยใช้ Viterbi Algorithm ทำให้สามารถหาลำดับของสถานะที่ดีที่สุดในการเปรียบเทียบกับเสียงที่ได้ยิน เพื่อที่จะนำมาใช้ในการรู้จำเสียงที่ติดต่อกันได้

3. Learning Problem

ตัวแปร (Parameter) ของแบบจำลองควรเป็นอย่างไร เพื่อที่จะทำให้มีความผิดพลาดในการสร้างเสียงหรือเข้าใจเสียงที่ได้ยินน้อยที่สุด เรียกว่า Learning Problem ปัญหานี้สามารถแก้ไขโดยใช้ Forward-Backward (Baum-Welch) Algorithm ซึ่งจะทำให้ระบบสามารถเรียนรู้ได้แบบอัตโนมัติจากค่าตัวแปรต่าง ๆ โดยการฝึกฝนจากชุดข้อมูลที่ป้อนเข้าระบบ

3.3.2 The Forward Algorithm

Evaluation Problem สามารถอธิบายได้อีกแง่มุมหนึ่ง คือ ให้โมเดลหนึ่งเป็น M กับตัวแปร {s} - a กลุ่มของสถานะ, {a}, {b} หาค่าความน่าจะเป็นที่โมเดลนั้นจะสร้างลำดับของเอาท์พุท y_1^T ปัญหานี้สามารถเขียนเป็นสมการได้ดังนี้

$$P(Y_1^T = y_1^T) = \sum_X P(X_1^{T+1} = x_1^{T+1}) P(Y_1^T = y_1^T \mid X_1^{T+1} = x_1^{T+1}) \quad (11)$$

จากสมมุติฐานข้อที่หนึ่ง ค่าแรกของสมการทางด้านขวาสามารถเขียนใหม่ได้ดังนี้

$$P(X_1^{T+1} = x_1^{T+1}) = \prod_{t=1}^{T} P(X_{t+1} = x_{t+1} \mid X_t = x_t) \quad (12)$$

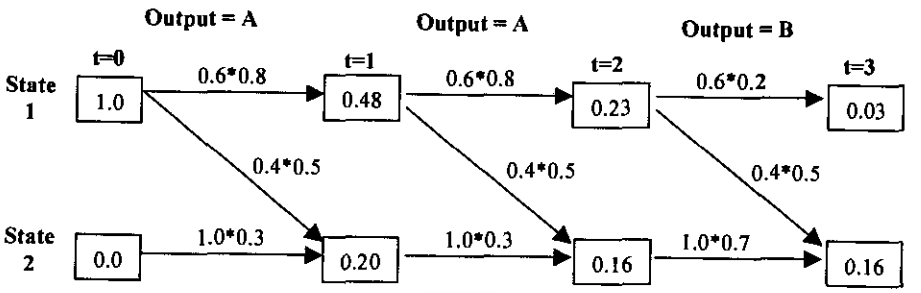
จากสมมุติฐานข้อที่สอง ค่าที่สองของสมการทางด้านขวาสามารถเขียนใหม่ได้ดังนี้

$$P(Y_1^T = y_1^T \mid X_1^{T+1} = x_1^{T+1}) = \prod_{t=1}^{T} P(Y_t = y_t \mid X_t = x_t, X_{t+1} = x_{t+1}) \quad (13)$$

เมื่อมีการแทนค่าดังกล่าวสมการสุดท้ายที่ได้คือ

$$\alpha_i(t) = \begin{cases} 0, t = 0 \cap i \neq S_1 \\ 1, t = 0 \cap i = S_1 \\ \sum_j \alpha_j(t-1) a_{ji} b_{ji}(y_t), t > 0 \end{cases} \quad (14)$$

$P(y|M)$ หรือ $\alpha_i(t)$ คือ ความน่าจะเป็นที่กระบวนการมาร์คอฟในสถานะ i ได้สร้างเอาท์พุท y_1^t



รูปที่ 3.11 การคำนวณ Trellis ตามรูปแบบ Forward Computation

รูปที่ 3.11 แสดงถึงการคำนวณของค่า α ที่เรียกว่า Trellis โดยการใช้ทฤษฎีของแบบจำลองฮิดเด็นมาร์คอฟ ตามรูปที่ 11 โดยสมมุติว่าลำดับการสังเกตเป็น A A B ตามลำดับ ได้เอาท์พุทเป็นลำดับของสถานะ 1 1 1 2

อย่างไรก็ตามในทฤษฎีการรู้จำเสียง ความน่าจะเป็นที่ควรพิจารณาคือ P(M|y) แต่ที่ผ่านมาได้คำนวณ P(y|M) จากกฎของเบย์

$$P(M \mid y) = \frac{P(y \mid M)P(M)}{P(y)} \qquad (15)$$

จะเห็นได้ว่า P(y) มีค่าคงที่สำหรับค่าอินพุตนั้นๆ P(M) คือความน่าจะเป็นที่มาจากแบบจำลองทางภาษา (Language Model) ดังนั้น สิ่งที่สำคัญที่ควรคำนึงถึงคือ P(M|y) จะเห็นได้ว่าเมื่อมีค่า P(y|M) สูงขึ้นก็จะทำให้ P(M|Y) สูงขึ้นเช่นกัน ดังนั้นใช้การคำนวณ P(y|M) แทนได้

3.3.3 The Viterbi Algorithm

จะเห็นได้ว่า Forward Algorithm ไม่มีการคำนวณลำดับของสถานะ แต่ก็เป็นไปได้ที่จะสร้างลำดับของสถานะที่มีความน่าจะเป็นสูงที่สุดเพื่อที่จะนำลำดับของสถานะนั้นมาใช้ในขณะที่กำลังสร้างลำดับการสังเกต ในการสร้างกระบวนการดังกล่าวจึงมีการดัดแปลงสมการของ Forward Algorithm ดังสมการด้านล่าง

$$v_i(t) = \begin{cases} 0, t = 0 \cap i \neq S_1 \\ 1, t = 0 \cap i = S_1 \\ \max_j v_j(t-1) a_{ji} b_{ji}(y_t), t > 0 \end{cases} \quad (16)$$

Viterbi Algorithm สามารถแสดงลำดับของสถานะที่มีความน่าจะเป็นสูงที่สุด โดยเลือกการเปลี่ยนสถานะที่มีความน่าจะเป็นมากที่สุดในแต่ละสถานะ

3.3.4 The Forward-Backward Algorithm

ปัญหานี้เป็นปัญหาที่ยากที่สุดเพราะว่ายังไม่มีวิธีการใดที่จะหาพารามิเตอร์ที่ดีที่สุดสำหรับแบบจำลองฮิดเด็นมาร์คอฟได้ ดังนั้นกระบวนการซ้ำซ้อนเช่น Forward-Backward (Baum-Welch) Algorithm จึงต้องนำมาใช้เพื่อหาค่าพารามิเตอร์ที่เหมาะสมที่สุดสำหรับแบบจำลอง $\alpha_i(t)$ เป็นความน่าจะเป็นที่ M ได้สร้าง y_1^t และอยู่ในสถานะ i ซึ่งเป็น Forward Algorithm $\beta_i(t)$ เป็นความน่าจะเป็นที่ M ในสถานะ i ได้สร้างเอาท์พุท y_{t+1}^T ซึ่งเป็น Backward Algorithm สมการของ $\beta_i(t)$ มีดังนี้

$$\beta_i(t) = \begin{cases} 0, t = T \cap i \neq S_F \\ 1, t = T \cap i = S_F \\ \sum_j \beta_j(t+1) a_{ij} b_{ij}(y_{t+1}), 0 \leq t < T \end{cases} \quad (17)$$

และยังมีการอธิบายถึงตัวแปรอีกตัวหนึ่งคือ $\gamma_{ij}(t)$ ที่เป็นความน่าจะเป็นของการเปลี่ยนสถานะหนึ่งที่จะนำไปสู่เอาท์พุท y_{t+1}^{T}

$$\gamma_{ij}(t) = P(X_t = i, X_{t+1} = j \mid y_1^T)$$
$$= \frac{\alpha_i(t-1)a_{ij}b_{ij}(y_t)\beta_j(t)}{\alpha_{S_F}(T)} \quad (18)$$

ดังนั้นเมื่อให้เอาท์พุทเป็น y_1^t ความน่าจะเป็นของจำนวนการเปลี่ยนสถานะจากสถานะ i ไปหาสถานะ j คือ $\sum_{t=1}^{T}\gamma_{ij}(t)$ และความน่าจะเป็นของจำนวนการเปลี่ยนสถานะจากสถานะ i ไปหาสถานะใดๆ คือ $\sum_{t=1}^{T}\sum_{k}\gamma_{ik}(t)$ ดังนั้นจึงเป็นไปได้ที่จะคำนวณความน่าจะเป็นของระบบที่มีการเปลี่ยนสถานะจาก i ไปหา j ซึ่งเป็นไปตามสมการด้านล่าง

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T}\gamma_{ij}(t)}{\sum_{t=1}^{T}\sum_{k}\gamma_{ik}(t)} \quad (19)$$

ในทำนองเดียวกัน อัตราส่วนระหว่างความถี่ที่ระบบจะมีเอาท์พุทเป็น k และความถี่ที่ระบบจะมีเอาท์พุทเป็นอะไรก็ได้ คือ

$$\bar{b}_{ij}(k) = \frac{\sum_{t:y_t=k}\gamma_{ij}(t)}{\sum_{t=1}^{T}\gamma_{ij}(t)} \quad (20)$$

บทที่ 4
โครงสร้างของระบบ

4.1 โครงสร้างโดยรวมของระบบ

รูปที่ 4.1 แผนภาพแสดงลำดับการทำงานโดยรวมของระบบสั่งงานด้วยเสียงพูด

จากรูปที่ 4.1 แสดงถึงโครงสร้างโดยรวมของระบบสั่งงานด้วยเสียงพูดที่ได้สร้างขึ้น โดยเริ่มจากระบบจะรอรับคำสั่งเสียงจากไมโครโฟนเข้ามาประมวลผลภายในคอมพิวเตอร์ ซึ่งในระบบนี้ใช้โปรแกรม MATLAB เวอร์ชั่น 2006a ในการเขียนโปรแกรมรู้จำเสียงพูดและประมวลผล เมื่อโปรแกรมประมวลผลเสร็จและรู้ว่าคำพูดที่รับเข้ามาเป็นคำสั่งเสียงใดแล้ว โปรแกรมจะทำการส่งผลลัพธ์ที่ได้จากการประมวลผลออกทางพอร์ตอนุกรม เพื่อไปควบคุมการทำงานของอุปกรณ์ภายนอกที่เราต้องการ

4.2 โครงสร้างของโปรแกรม

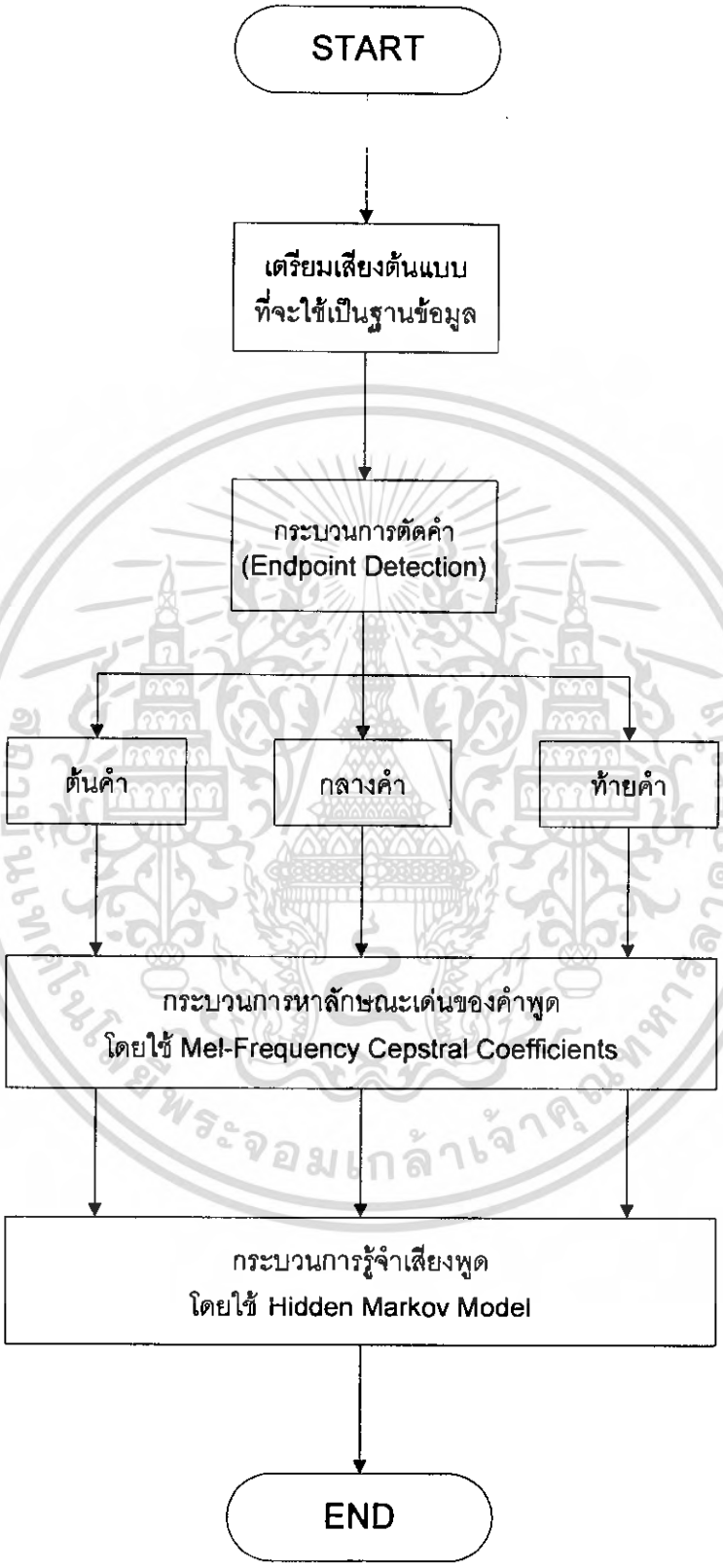
รูปที่ 4.2 แสดงกระบวนการประมวลผลคำสั่งเสียง

ในรูปที่ 4.2 แสดงถึงกระบวนการประมวลผลคำสั่งเสียง ซึ่งก่อนที่จะรับคำสั่งเสียงเข้ามาประมวลผลได้นั้น จะต้องมีการสร้างฐานข้อมูลเสียงตามกระบวนการรู้จำเสียงขึ้นมาก่อน เพื่อให้ระบบได้เรียนรู้คำสั่งเสียงพูดที่เราต้องการ เมื่อสร้างฐานข้อมูลเสียงพูดของแต่ละคำสั่งเสียงแล้ว จึงจะสามารถสั่งงานด้วยคำสั่งเสียงได้ ซึ่งรายละเอียดของกระบวนการสร้างฐานข้อมูลเพื่อให้ระบบเรียนรู้เสียงพูดและกระบวนการประมวลผลการรู้จำเสียงมีดังต่อไปนี้

4.2.1 กระบวนการสร้างฐานข้อมูลเพื่อให้ระบบรู้จำเสียงพูด

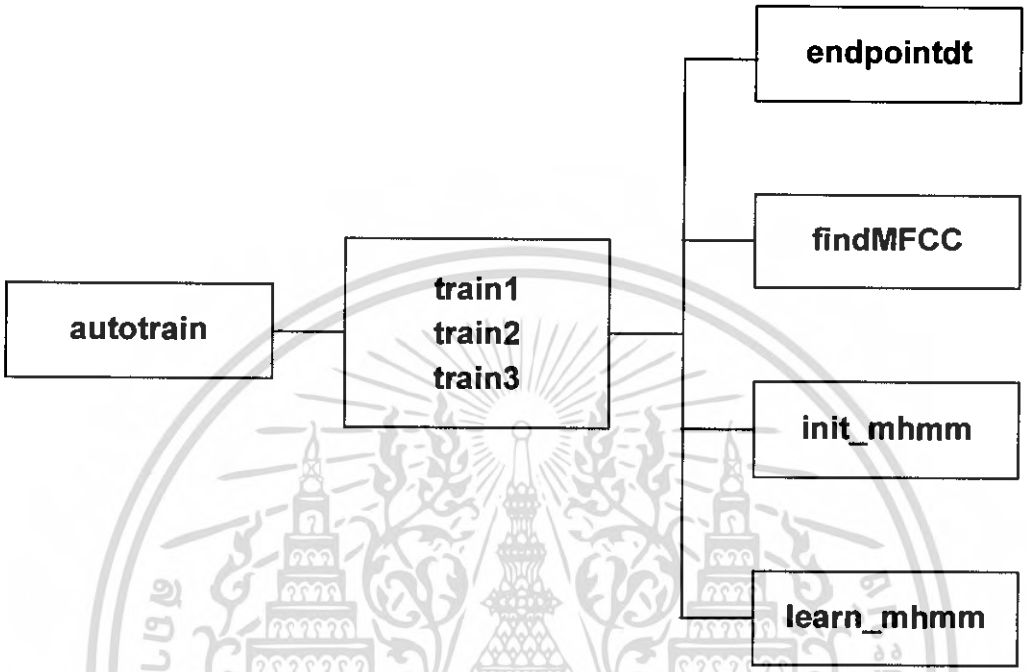
การที่ระบบจะสามารถแยกแยะได้ว่าเสียงพูดที่เราพูดเป็นคำสั่งเสียงใดได้นั้น เราจำเป็นจะต้องสอนให้ระบบรู้จักเสียงพูดที่เราต้องการเสียก่อน ซึ่งเรียกกระบวนการนี้ว่า กระบวนการรู้จำเสียงพูด (Speech Recognition)

ในขั้นต้น เราจะต้องเก็บข้อมูลเสียงต้นแบบเพื่อนำมาเป็นฐานข้อมูลให้กับระบบเสียก่อน เสียงต้นแบบนี้ก็คือคำสั่งเสียงที่เราต้องการให้ระบบเรียนรู้นั่นเอง จากนั้นเราจะนำเสียงต้นแบบทั้งหมดมาผ่านกระบวนการต่างๆ เพื่อสร้างแบบจำลองทางคณิตศาสตร์ที่เป็นตัวแทนของคำสั่งเสียงพูดแต่ละคำ โดยในระบบได้ใช้กระบวนการ เมลฟรีเควนซี่เซปสตรัลโคเอฟฟิเซี่ยน (Mel-Frequency Cepstral Coefficients : MFCC) ในการหาลักษณะสำคัญของคำพูดแต่ละคำ ซึ่งผลลัพธ์ที่ได้จะอยู่ในรูปของค่าเวคเตอร์ซึ่งใช้เป็นตัวแทนของคำพูดนั้นๆ จากนั้นจึงนำค่าดังกล่าวไปสร้างเป็นโมเดลเสียงโดยใช้ แบบจำลองฮิดเด็นมาร์คอฟ (Hidden Markov Model : HMM) ซึ่งกระบวนการต่างๆ ที่ได้กล่าวมานี้แสดงดังแผนภาพในรูปที่ 4.3



รูปที่ 4.3 กระบวนการสร้างฐานข้อมูลเพื่อให้ระบบรู้จำเสียงพูด

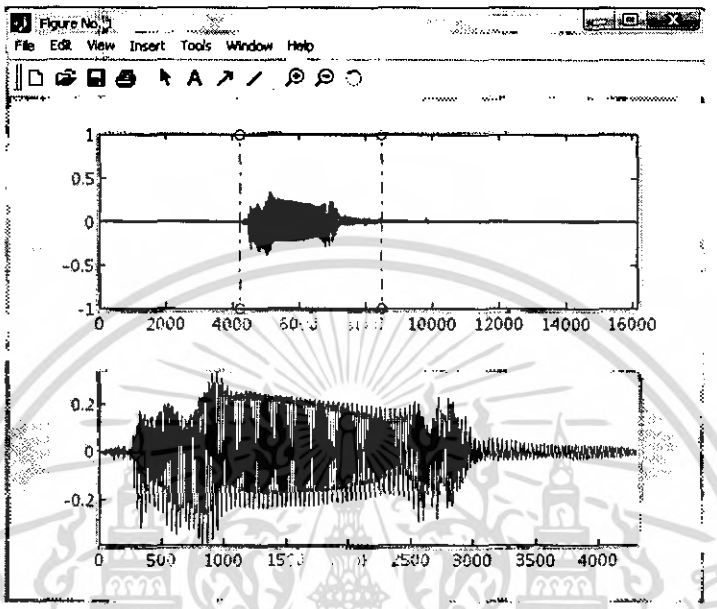
ในการสร้างฐานข้อมูลเพื่อที่ให้ระบบรู้จำเสียงพูดได้นั้น เราจะทำการประมวลผลไฟล์ autotrain.m โดยในไฟล์ดังกล่าว จะประกอบไปด้วยฟังก์ชั่นของกระบวนการต่าง ๆ ดังต่อไปนี้



รูปที่ 4.4 แสดงส่วนประกอบของไฟล์ autotrain.m

เพื่อให้กระบวนการรู้จำเสียงพูดในระบบนี้มีประสิทธิภาพมากขึ้นและใช้เวลาในการประมวลผลน้อยลง เราจึงทำการแบ่งคำพูดออกเป็นช่วงต่างๆ 3 ช่วง คือ ช่วงที่ออกเสียงพยัญชนะต้นซึ่งก็คือช่วงต้นของคำ ช่วงที่เป็นเสียงสระและวรรณยุกต์ซึ่งก็คือช่วงกลางคำ และช่วงที่เป็นเสียงตัวสะกดซึ่งก็คือช่วงท้ายของคำ (แสดงในรูปที่ 4.3) ซึ่งแต่ละช่วงของคำพูดนี้จะถูกนำไปผ่านกระบวนการรู้จำตามฟังก์ชั่นในไฟล์ train1, train2 และ train3 ตามลำดับ และแต่ละไฟล์จะประกอบไปด้วยฟังก์ชั่นย่อย ได้แก่ endpointdt, find_MFCC, init_mhmm และ learn_mhmm ซึ่งมีหน้าที่ต่างๆ ดังนี้

1. **ฟังก์ชั่น endpointdt :** เป็นฟังก์ชั่นที่ทำหน้าที่หาจุดเริ่มต้นและจุดสิ้นสุดของคำ โดยจะตัดสัญญาณรบกวนออกแล้วจะได้สัญญาณดังรูป 4.5



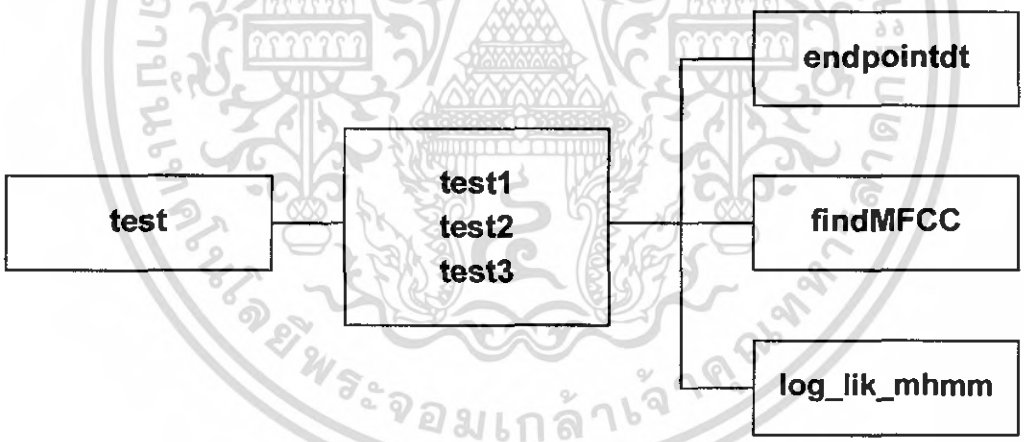
รูปที่ 4.5 แสดงการหาจุดเริ่มต้นและจุดสิ้นสุดของสัญญาณ

2. **ฟังก์ชั่น findMFCC :** เป็นฟังก์ชั่นที่เอาไว้แปลงสัญญาณเสียงให้เป็นค่าพารามิเตอร์เพื่อเอาไว้ใช้ในการวิเคราะห์และใช้ในกระบวนการต่างๆ เทคนิคในการหาค่าพารามิเตอร์นี้มีหลายวิธี เช่น ลิเนียร์พรีดิกชั่นโค้ดดิ้ง(Linear Prediction Coding หรือ LPC) , เมลฟรีเควนซี่เซปสตรัลโคเอฟฟิเชี่ยน(Mel-Frequency Cepstrum Coefficients หรือ MFCC) เราเลือกใช้วิธีเมลฟรีเควนซี่เซปสตรัลโคเอฟฟิเชี่ยน เพราะเป็นที่นิยมใช้กันอย่างกว้างขวางและมีความถูกต้องสูง
3. **ฟังก์ชั่น init_mhmm :** เป็นฟังก์ชั่นที่สร้างค่าพารามิเตอร์เริ่มต้นของแบบจำลอง เพื่อเอาไว้เปรียบเทียบคุณลักษณะของเสียง มีหลายวิธีเช่น ไดนามิกไทม์วาปปิ้ง(Dynamic Time Warpong หรือ DTW), เวคเตอร์ควอนไตเซชั่น(Vector Quantization หรือ VQ) , ฮิดเด็นมาร์คอฟโมเดล(Hidden Makrov Modeling หรือ HMM) เราเลือกใช้แบบจำลองฮิดเด็นมาร์คอฟเพราะเป็นวิธีที่มีความถูกต้องสูง
4. **ฟังก์ชั่น learn_mhmm :** เป็นฟังก์ชั่นที่ใช้ทำให้แบบจำลองเรียนรู้ อ๊อบเซอร์เวชั่นเวคเตอร์ (Observation Vector) ที่ได้จากกระบวนการเมลฟรีเควนซี่เซปสตรัลโคเอฟฟิเชี่ยน และจะคำนวณค่าความน่าจะเป็นของแต่ละสถานะ(state) ออกมา

4.2.2 การประมวลผลที่ได้จากกระบวนการรู้จำเสียง

เมื่อเราได้สร้างแบบจำลองของคำสั่งเสียงพูดแต่ละคำขึ้นมาแล้ว เราจะทำการสั่งระบบด้วยคำสั่งเสียงที่เราต้องการ โดยกระบวนการประมวลผลคำสั่งเสียงนี้จะเหมือนกับกระบวนการสร้างแบบจำลองที่ได้กล่าวมาแล้ว โดยระบบจะนำคำสั่งเสียงที่เราพูดเข้าทางไมโครโฟนไปประมวลผลตามขั้นตอนต่างๆ เริ่มจากกระบวนการตัดคำ ซึ่งจะตัดเสียงในส่วนที่เราไม่ต้องการในช่วงต้นและช่วงท้ายของการอัดเสียงออกไป ให้เหลือเพียงสัญญาณเสียงในช่วงที่เป็นคำพูดเท่านั้น จากนั้นจะนำสัญญาณดังกล่าวไปหาค่าพารามิเตอร์ซึ่งเป็นตัวแทนของเสียงด้วยกระบวนการเมลฟรีเควนซี่เซปสตรัลโคเอฟฟิเชี่ยน (Mel-Frequency Cepstrum Coefficients : MFCC) ซึ่งในการประมวลผลจะแยกคำพูดออกเป็นช่วงๆ 3 ช่วงเช่นเดิม เมื่อได้พารามิเตอร์ดังกล่าวมาแล้ว จึงนำเข้าสู่แบบจำลองของคำสั่งเสียงที่เราได้สร้างไว้แล้ว ในขั้นตอนนี้ระบบจะทำการคำนวณหาความน่าจะเป็นว่าคำพูดที่เราพูดนั้นเป็นคำใด โดยผลลัพธ์ของกระบวนการนี้จะเป็นความน่าจะเป็นสูงสุดของแบบจำลอง

ขั้นตอนทั้งหมดที่ได้กล่าวมาข้างต้นนี้ จะอยู่ในไฟล์ test.m โดยในไฟล์ดังกล่าวจะประกอบไปด้วยฟังก์ชั่นของกระบวนการต่างๆ ดังนี้



รูปที่ 4.6 แสดงส่วนประกอบของไฟล์ test.m

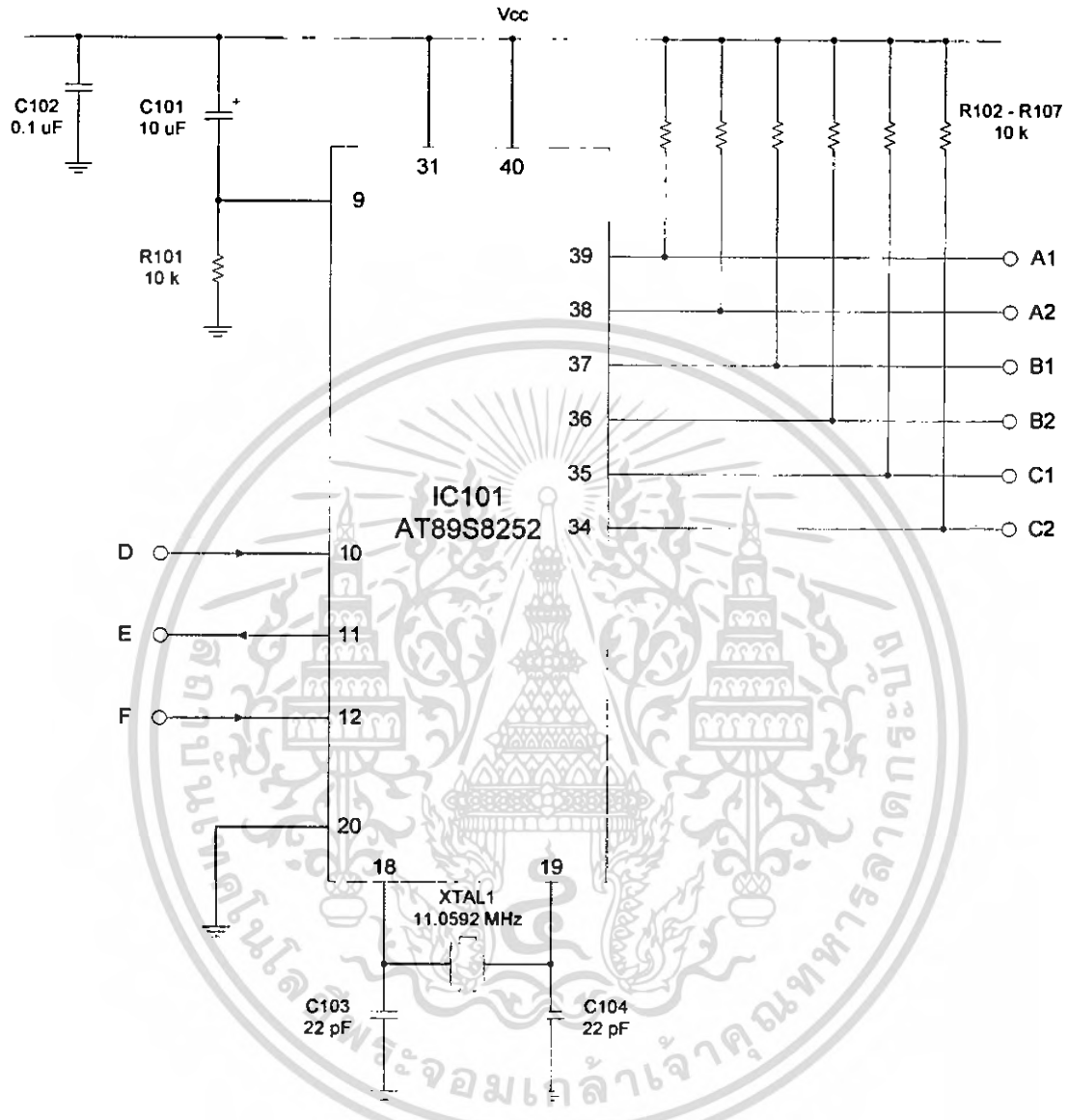
1. **ฟังก์ชั่น endpointdt :** มีหน้าที่เหมือนกับในส่วนของการวิเคราะห์หาคุณลักษณะของเสียง
2. **ฟังก์ชั่น findMFCC :** มีหน้าที่เหมือนกับในส่วนของการวิเคราะห์หาคุณลักษณะของเสียง
3. **ฟังก์ชั่น lok_lik_mhmm :** ทำหน้าที่คำนวณค่าความน่าจะเป็น ที่ได้จากแบบจำลองฮิดเดนมาร์คอฟ และในที่สุดก็จะได้ค่าความน่าจะเป็นของเสียงคำสั่งที่เปรียบเทียบกับรูปแบบเสียงอ้างอิงออกมา

บทที่ 5
การประยุกต์ใช้งาน

ระบบสั่งงานด้วยเสียงพูดที่ได้ทำนี้ สามารถนำไปใช้สั่งงานในระบบต่างๆ ได้หลากหลาย ในโครงงานนี้ขอเสนอแนวทางการประยุกต์ใช้งานกับรถโฟล์คลิฟท์ โดยจะจำลองการทำงานกับโมเดลรถขนาดเล็ก เพื่อให้เห็นของการสั่งงานรถโฟล์คลิฟท์ด้วยเสียงพูดว่าให้ผลเช่นไร

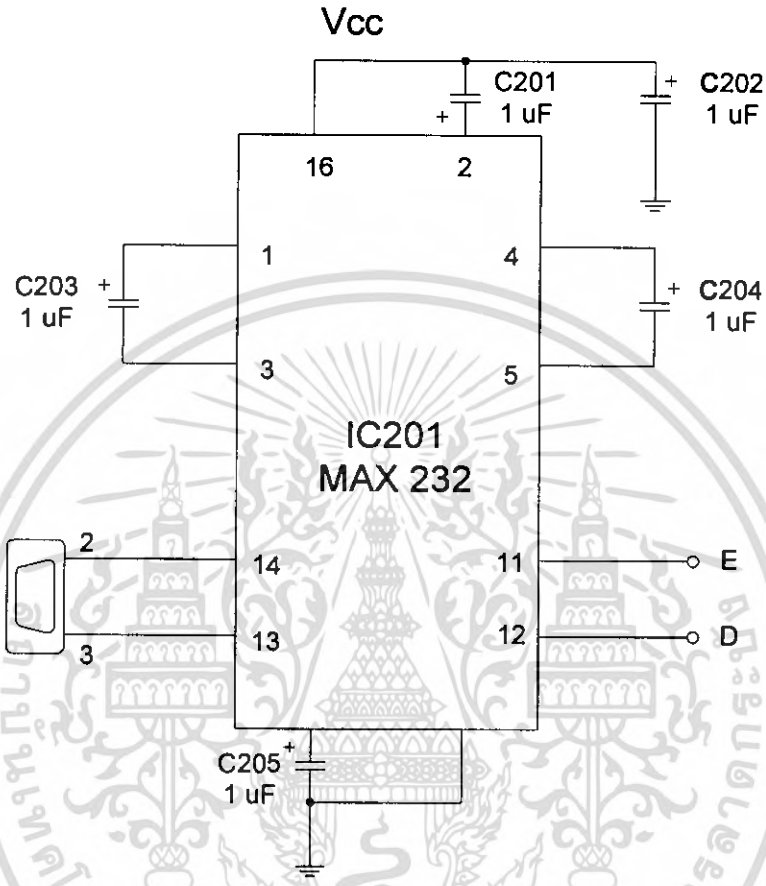
5.1 วงจรควบคุมโมเดลรถ

เนื่องจากระบบของเราใช้คอมพิวเตอร์ในการประมวลผลการรู้จำเสียงพูด ดังนั้นจึงต้องมีการรับ-ส่งข้อมูลระหว่างคอมพิวเตอร์กับวงจรภายนอกที่จะไปควบคุมมอเตอร์ของโมเดลรถ รายละเอียดของวงจรที่ใช้ควบคุมการรับ-ส่งข้อมูล และวงจรควบคุมมอเตอร์ มีดังต่อไปนี้



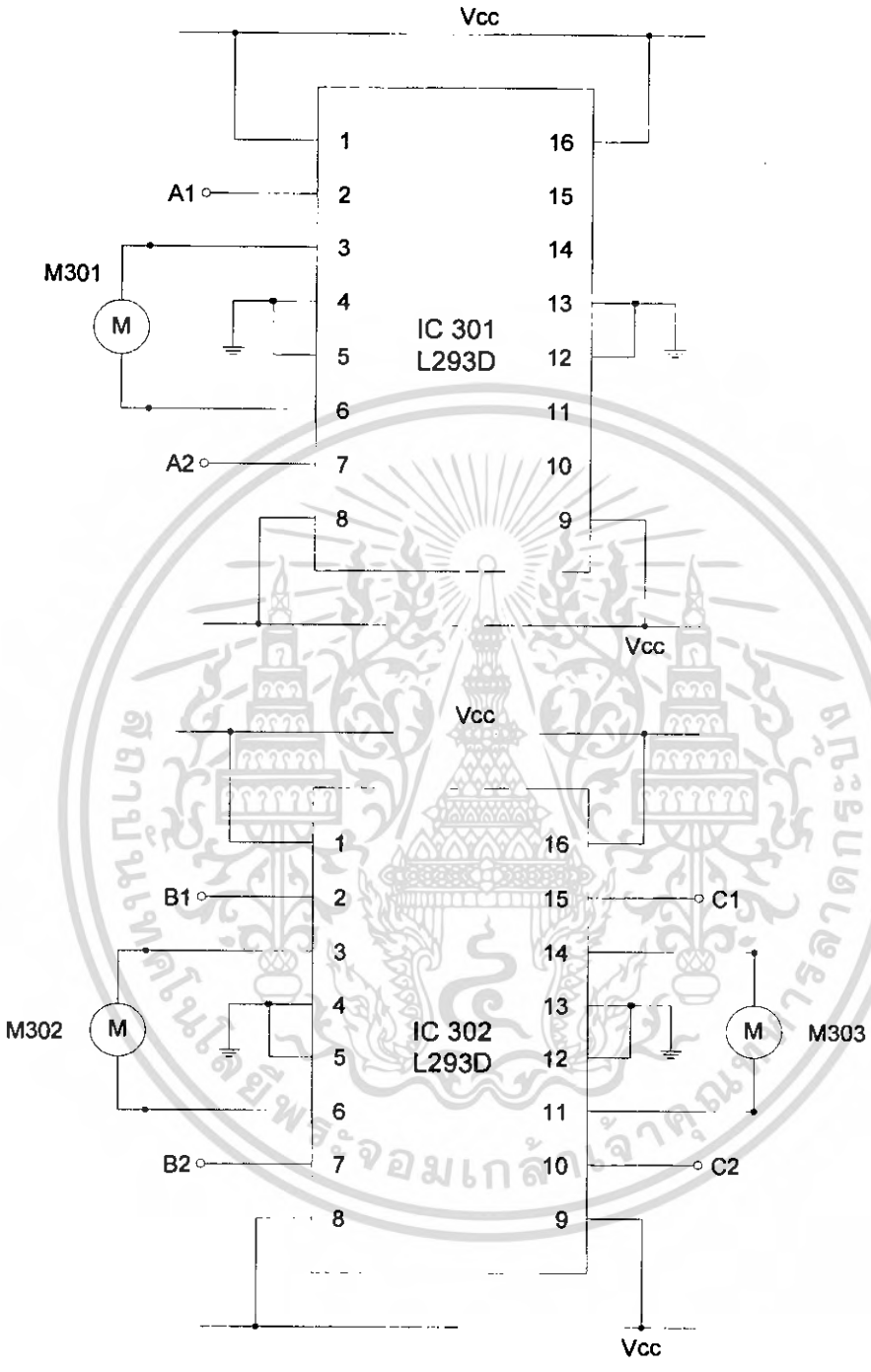
รูปที่ 5.1 วงจรไมโครคอนโทรลเลอร์

รูปที่ 5.1 แสดงวงจรควบคุม โดยใช้ไมโครคอนโทรลเลอร์เบอร์ AT89S8252 เป็นตัวควบคุมการรับ-ส่งข้อมูลระหว่างคอมพิวเตอร์กับวงจรภายนอก และเป็นตัวควบคุมการหมุนของมอเตอร์รถ



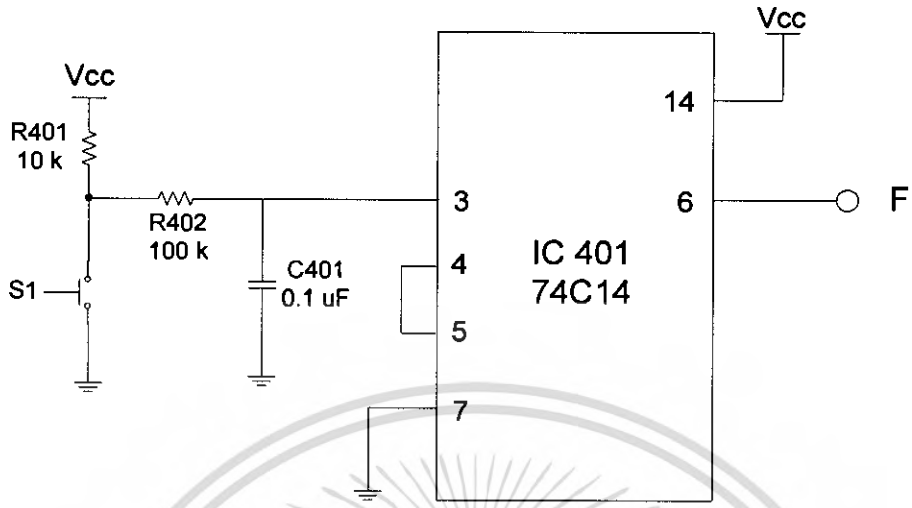
รูปที่ 5.2 แสดงวงจร MAX232

รูปที่ 5.2 แสดงวงจรในส่วนของ IC MAX232 ซึ่งเป็น IC ที่ใช้แปลงระดับแรงดันไฟระหว่างพอร์ตอนุกรมของคอมพิวเตอร์กับไมโครคอนโทรลเลอร์



รูปที่ 5.3 แสดงวงจร L293D

รูปที่ 5.3 แสดงวงจรขับมอเตอร์ไฟตรง โดยใช้ IC เบอร์ L293D เป็นตัวขับมอเตอร์



รูปที่ 5.4 แสดงวงจร Debounce Switch

และรูปที่ 5.4 แสดงวงจรสวิทช์ ซึ่งใช้สำหรับควบคุมการพูดคำสั่งเสียง โดยมี IC เบอร์ 74HC14 ซึ่งเป็น IC Schmitt Trigger ต่อไว้เพื่อป้องกันการ bounce ของสวิทช์

การทำงานเริ่มจาก เมื่อต้องการสั่งงานโมเดลรถด้วยเสียงพูด จะต้องทำการกดสวิทช์ S1 เมื่อสวิทช์ถูกกดจะทำให้เกิดการอินเตอร์รัปต์ขึ้นในไมโครคอนโทรลเลอร์ ซึ่งตามโปรแกรมที่เขียนไว้ในไมโครคอนโทรลเลอร์ (ดูภาคผนวก ข) กำหนดไว้ว่า เมื่อเกิดอินเตอร์รัปต์ให้ทำการส่งข้อมูลไปยังคอมพิวเตอร์เพื่อให้โปรแกรมรู้จำเสียงพูดทำงาน ดังนั้นเราจึงสามารถพูดสั่งงานทางไมโครโฟนได้ทันทีที่กดสวิทช์ S1 แล้ว

เมื่อเราพูดสั่งงานเรียบร้อยแล้ว โปรแกรมแม็ทแลปจะทำการประมวลผลว่า คำที่เราพูดนั้นเป็นคำสั่งเสียงใดในฐานข้อมูล เมื่อประมวลผลเสร็จแล้วจึงทำการส่งผลลัพธ์ซึ่งอยู่ในรูปแบบของรหัสแอสกี้ (ASCII) ออกมาให้ไมโครคอนโทรลเลอร์ ไมโครคอนโทรลเลอร์จะทำการตรวจสอบว่ารหัสแอสกี้ที่รับมานั้นตรงกับฟังก์ชั่นการทำงานใดในมอเตอร์ (ดูภาคผนวก ข) จากนั้นจึงทำการส่งลอจิกออกทางพอร์ต 0 เข้าสู่ IC ขับมอเตอร์เพื่อให้มอเตอร์ทำงานตามต้องการ

5.2 รายการอุปกรณ์

วงจรไมโครคอนโทรลเลอร์

ตัวต้านทาน 1/4 วัตต์ 1%

R101-107	10 kΩ	1 ตัว

ตัวเก็บประจุ

C101	10 μF อิเล็กทรอไลต์	1 ตัว
C102	0.1 μF เซรามิก	1 ตัว
C103-104	22 pF เซรามิก	2 ตัว

อุปกรณ์สารกึ่งตัวนำ

IC101	AT89S8252	1 ตัว
XTAL1	คริสตอล 11.0592 MHz	1 ตัว

วงจร MAX232

ตัวเก็บประจุ

C201-205	1 μF อิเล็กทรอไลต์	5 ตัว

อุปกรณ์สารกึ่งตัวนำ

IC201	MAX232	1 ตัว

อื่นๆ

หัวต่อ DB9 ตัวผู้	1 ตัว

วงจรขับมอเตอร์

อุปกรณ์สารกึ่งตัวนำ

IC301-302	L293D	2 ตัว

อื่นๆ

M301-303	DC Motor	3 ตัว

วงจรดีเบาซ์สวิทช์

ตัวต้านทาน 1/4 วัตต์ 1%

R401	10 kΩ	1 ตัว
R402	100 kΩ	1 ตัว

ตัวเก็บประจุ

C401	0.1 μF เซรามิก	1 ตัว

อุปกรณ์สารกึ่งตัวนำ

IC401	74HC14	1 ตัว

อื่น ๆ

S1	สวิทช์กดติดปล่อยดับ	1 ตัว

บทที่ 6
การทดลอง

6.1 การทดลอง

ในการทดลองสั่งงานด้วยเสียงนี้ จะใช้โปรแกรมแมทแล็ปในการประมวลผลเพื่อให้คอมพิวเตอร์เรียนรู้คำสั่งเสียงที่ได้บันทึกไว้ จากนั้นจะทำการสั่งงานด้วยเสียงโดยให้คอมพิวเตอร์ประมวลผลเสียงพูดที่สั่ง แล้วส่งผลลัพธ์ออกไปควบคุมโมเดลรถโฟล์คลิฟท์

คอมพิวเตอร์ที่ใช้ในการประมวลผลในระบบนี้ มีคุณสมบัติดังนี้

- หน่วยประมวลผล CPU Intel Pentium M 1.60 GHz
- หน่วยความจำ RAM 256 MB
- การ์ดเสียง Conexant AMC Audio
- ใช้โปรแกรม MATLAB 2006a ในการเขียนโปรแกรมเพื่อให้ระบบรู้จำเสียงพูด

ฐานข้อมูลเสียงที่ใช้ในระบบนี้ จะใช้เสียงจากคน 10 คน เป็นผู้ชาย 5 คน และผู้หญิง 5 คน คำสั่งเสียงที่ใช้เป็นฐานข้อมูลให้กับระบบจะถูกกำหนดไว้ตามตารางที่ 6.1 โดยจะทำการบันทึกเสียงลงในคอมพิวเตอร์จากคน 10 คน เป็นผู้ชาย 5 คน และผู้หญิง 5 คน แต่ละคนบันทึกคำสั่งเสียงแต่ละคำทั้งหมด 3 ครั้ง รวมเป็นไฟล์ฐานข้อมูลเสียงทั้งหมด 210 ไฟล์ ซึ่งไฟล์เสียงดังกล่าวมีคุณสมบัติดังนี้

- ใช้ความละเอียด (Audio Sample Size) ขนาด 16 บิต
- ใช้ความถี่ในการสุ่ม (Audio Sample Rate) 8000 เฮิรตซ์
- อัตราบิต (Bit Rate) 128 กิโลบิตต่อวินาที

คำสั่งที่	คำสั่งเสียง	ฟังก์ชั่นการทำงาน
1	ซ้าย	รถเลี้ยวซ้าย
2	ขวา	รถเลี้ยวขวา
3	หน้า	รถเดินหน้า
4	หลัง	รถถอยหลัง
5	ยก	แขนโฟล์คลิฟท์เลื่อนขึ้น
6	วาง	แขนโฟล์คลิฟท์เลื่อนลง
7	หยุด	หยุดการเคลื่อนไหว

ตาราง 6.1 แสดงคำสั่งที่ใช้ในการรู้จำเสียง

การทดสอบระบบ แบ่งออกเป็น 2 การทดลองดังนี้

1) การทดสอบความถูกต้องโดยรวมในการสั่งงานระบบ เปรียบเทียบระหว่างผู้สั่งงานที่มีเสียงอยู่ในฐานข้อมูล เป็นชาย 1 คน และหญิง 1 คน กับผู้สั่งงานที่มีเสียงอยู่นอกฐานข้อมูล เป็นชาย 1 คน และหญิง 1 คนเช่นกัน แต่ละคนพูดสั่งงานด้วยชุดคำสั่งตามตารางที่ 6.1 ทั้งหมด 10 ครั้ง
2) การทดสอบความถูกต้องในการสั่งงานแต่ละคำสั่งเสียง โดยเปรียบเทียบว่าคำสั่งเสียงใดที่สั่งงานได้ถูกต้องมากกว่าและน้อยกว่ากัน โดยผู้สั่งงานเป็นผู้ที่มีเสียงอยู่ในฐานข้อมูล เป็นชาย 1 คน และหญิง 1 คน แต่ละคนพูดสั่งงานด้วยชุดคำสั่งตามตารางที่ 6.1 ทั้งหมด 20 ครั้ง

6.2 ขั้นตอนการทดลอง

ผู้ทดสอบแต่ละคนทำตามขั้นตอนดังนี้

1) ให้ระบบเรียนรู้คำสั่งเสียงที่ได้สร้างเป็นฐานข้อมูลเสียงไว้ โดยใช้โปรแกรมแมทแล็ปประมวลผลไฟล์ที่ชื่อ autotrain.m
2) ทำการสั่งงานระบบด้วยคำสั่งในตารางที่ 6.1 โดยใช้โปรแกรมแมทแล็ปประมวลผลไฟล์ที่ชื่อ test.m
3) โปรแกรมแมทแล็ปจะส่งผลลัพธ์ที่ได้จากการประมวลผลออกมาทางพอร์ตอนุกรมของคอมพิวเตอร์ เข้าสู่วงจรควบคุมมอเตอร์ของโมเดลรถโฟล์คลิฟท์
4) บันทึกผลการทดลองลงในตาราง ถ้าโมเดลรถโฟล์คลิฟท์ทำงานถูกต้องตามคำสั่งเสียงให้แทนด้วยเครื่องหมายถูก (✓) ถ้าโมเดลรถโฟล์คลิฟท์ทำงานผิด ให้บันทึกฟังก์ชั่นการทำงานที่โมเดลรถโฟล์คลิฟท์ทำ

6.3 ผลการทดลอง

6.3.1 การทดสอบความถูกต้องโดยรวมในการสั่งงานระบบ

1) ผลการทดลองสั่งงานระบบโดยผู้ชายที่มีเสียงอยู่ในฐานข้อมูล

คำสั่งเสียง	ผลการทดลองสั่งงาน				
	ครั้งที่ 1	ครั้งที่ 2	ครั้งที่ 3	ครั้งที่ 4	ครั้งที่ 5
ซ้าย	✓	✓	✓	✓	✓
ขวา	✓	วาง	✓	✓	✓
หน้า	✓	✓	✓	✓	✓
หลัง	✓	✓	✓	✓	✓
ยก	✓	✓	✓	✓	✓
วาง	✓	✓	✓	✓	ขวา
หยุด	✓	✓	✓	✓	✓

คำสั่งเสียง	ผลการทดลองสั่งงาน				
	ครั้งที่ 6	ครั้งที่ 7	ครั้งที่ 8	ครั้งที่ 9	ครั้งที่ 10
ซ้าย	✓	✓	✓	✓	✓
ขวา	✓	✓	✓	✓	✓
หน้า	✓	✓	✓	✓	✓
หลัง	✓	ขวา	✓	✓	✓
ยก	✓	หยุด	✓	✓	✓
วาง	✓	✓	✓	✓	✓
หยุด	✓	✓	✓	✓	✓

ตาราง 6.2 แสดงผลการทดลองสั่งงานระบบโดยผู้ชายที่มีเสียงอยู่ในฐานข้อมูล

คิดเป็น % ความถูกต้อง เท่ากับ 94.29%

2) ผลการทดลองสั่งงานระบบโดยผู้หญิงที่มีเสียงอยู่ในฐานข้อมูล

คำสั่งเสียง	ผลการทดลองสั่งงาน				
	ครั้งที่ 1	ครั้งที่ 2	ครั้งที่ 3	ครั้งที่ 4	ครั้งที่ 5
ซ้าย	✓	✓	✓	✓	✓
ขวา	✓	✓	✓	✓	✓
หน้า	✓	✓	✓	✓	✓
หลัง	ซ้าย	✓	✓	✓	✓
ยก	✓	✓	✓	✓	✓
วาง	✓	✓	หยุด	✓	✓
หยุด	✓	✓	✓	✓	✓

คำสั่งเสียง	ผลการทดลองสั่งงาน				
	ครั้งที่ 6	ครั้งที่ 7	ครั้งที่ 8	ครั้งที่ 9	ครั้งที่ 10
ซ้าย	✓	✓	✓	✓	✓
ขวา	✓	หลัง	✓	✓	✓
หน้า	✓	✓	✓	✓	✓
หลัง	ซ้าย	✓	✓	✓	✓
ยก	✓	✓	✓	✓	✓
วาง	✓	✓	ขวา	หยุด	✓
หยุด	✓	✓	✓	✓	✓

ตาราง 6.3 แสดงผลการทดลองสั่งงานระบบโดยผู้หญิงที่มีเสียงอยู่ในฐานข้อมูล

คิดเป็น % ความถูกต้อง เท่ากับ 91.43%

3) ผลการทดลองสั่งงานระบบโดยผู้ชายที่มีเสียงอยู่นอกฐานข้อมูล

คำสั่งเสียง	ผลการทดลองสั่งงาน				
	ครั้งที่ 1	ครั้งที่ 2	ครั้งที่ 3	ครั้งที่ 4	ครั้งที่ 5
ซ้าย	✓	✓	✓	✓	✓
ขวา	วาง	วาง	✓	✓	หลัง
หน้า	✓	✓	✓	✓	✓
หลัง	วาง	✓	✓	✓	ขวา
ยก	✓	✓	หยุด	✓	✓
วาง	✓	ขวา	✓	✓	✓
หยุด	✓	✓	✓	ยก	✓

คำสั่งเสียง	ผลการทดลองสั่งงาน				
	ครั้งที่ 6	ครั้งที่ 7	ครั้งที่ 8	ครั้งที่ 9	ครั้งที่ 10
ซ้าย	✓	✓	✓	✓	✓
ขวา	✓	✓	วาง	วาง	✓
หน้า	ซ้าย	✓	✓	✓	✓
หลัง	✓	ซ้าย	✓	✓	✓
ยก	✓	✓	✓	✓	✓
วาง	✓	ขวา	✓	✓	หลัง
หยุด	✓	✓	✓	✓	✓

ตาราง 6.4 ผลการทดลองสั่งงานระบบโดยผู้ชายที่มีเสียงอยู่นอกฐานข้อมูล

คิดเป็น % ความถูกต้อง เท่ากับ 80%

4) ผลการทดลองสั่งงานระบบโดยผู้หญิงที่มีเสียงอยู่นอกฐานข้อมูล

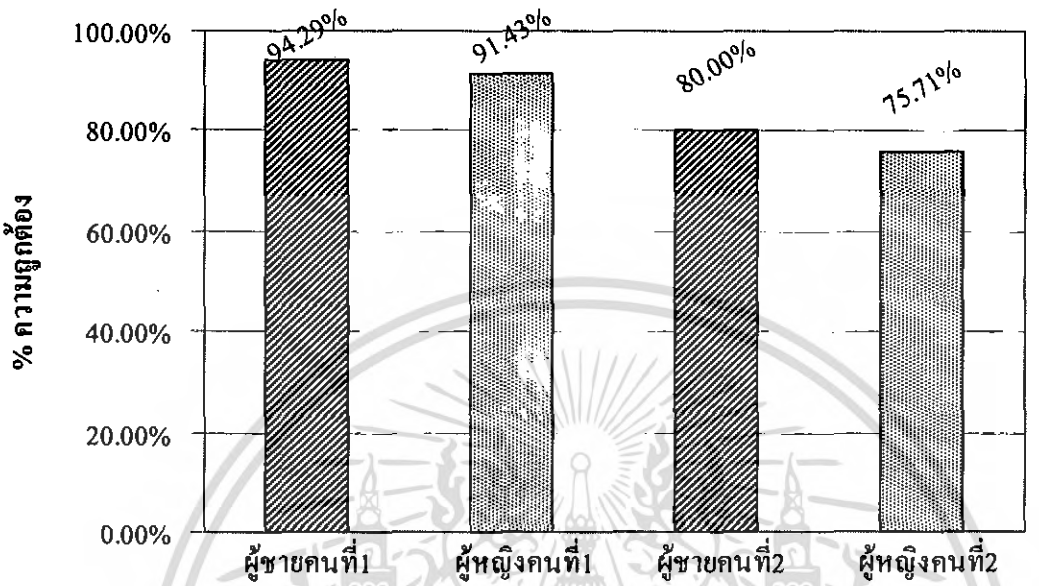
คำสั่งเสียง	ผลการทดลองสั่งงาน				
	ครั้งที่ 1	ครั้งที่ 2	ครั้งที่ 3	ครั้งที่ 4	ครั้งที่ 5
ซ้าย	✓	✓	✓	✓	✓
ขวา	หลัง	✓	✓	หน้า	หลัง
หน้า	✓	หยุด	✓	✓	✓
หลัง	ซ้าย	✓	ซ้าย	ซ้าย	ซ้าย
ยก	✓	✓	✓	✓	✓
วาง	หลัง	✓	หลัง	✓	✓
หยุด	✓	✓	✓	✓	✓

คำสั่งเสียง	ผลการทดลองสั่งงาน				
	ครั้งที่ 6	ครั้งที่ 7	ครั้งที่ 8	ครั้งที่ 9	ครั้งที่ 10
ซ้าย	✓	✓	✓	✓	✓
ขวา	✓	✓	หลัง	✓	✓
หน้า	✓	✓	✓	✓	✓
หลัง	ซ้าย	✓	✓	ขวา	ขวา
ยก	✓	✓	✓	✓	✓
วาง	หลัง	หยุด	ขวา	✓	✓
หยุด	✓	✓	✓	✓	✓

ตาราง 6.5 ผลการทดลองสั่งงานระบบโดยผู้หญิงที่มีเสียงอยู่นอกฐานข้อมูล

คิดเป็น % ความถูกต้อง เท่ากับ 75.71%

5) กราฟผลการทดสอบความถูกต้องโดยรวมในการสั่งงานระบบ



รูปที่ 6.1 กราฟการทดสอบความถูกต้องโดยรวมในการสั่งงานระบบ

6.3.2 การทดสอบความถูกต้องในการสั่งงานแต่ละคำสั่งเสียง

1) ผลการทดลองเปรียบเทียบความถูกต้องในการสั่งงานแต่ละคำสั่งเสียง โดยผู้ชายที่มีเสียงอยู่ในฐานข้อมูลเป็นผู้สั่งงาน

คำสั่ง / ครั้งที่	ซ้าย	ขวา	หน้า	หลัง
1	✓	✓	✓	✓
2	✓	วาง	✓	✓
3	✓	✓	✓	✓
4	✓	✓	✓	✓
5	✓	✓	✓	✓
6	✓	✓	✓	✓
7	✓	✓	✓	ขวา
8	✓	✓	✓	✓
9	✓	✓	✓	✓
10	✓	✓	✓	✓
11	✓	✓	✓	✓
12	✓	✓	✓	✓
13	✓	✓	✓	✓
14	✓	✓	✓	✓
15	✓	✓	✓	✓
16	✓	✓	✓	✓
17	✓	✓	✓	✓
18	✓	✓	✓	✓
19	✓	✓	✓	✓
20	✓	✓	✓	✓
% ความถูกต้อง	100%	95%	100%	95%

ตาราง 6.6 ผลการทดลองเปรียบเทียบความถูกต้องในการสั่งงานแต่ละคำสั่งเสียง โดยผู้ชายที่มีเสียงอยู่ในฐานข้อมูลเป็นผู้สั่งงาน

2) ผลการทดลองเปรียบเทียบความถูกต้องในการสั่งงานแต่ละคำสั่งเสียง โดยผู้ชายที่มีเสียงอยู่ในฐานข้อมูลเป็นผู้สั่งงาน (ต่อ)

ครั้งที่ \ คำสั่ง	ยก	วาง	หยุด
1	✓	✓	✓
2	✓	✓	✓
3	✓	✓	✓
4	✓	✓	✓
5	✓	ขวา	✓
6	✓	✓	✓
7	หยุด	✓	✓
8	✓	✓	✓
9	✓	✓	✓
10	✓	✓	✓
11	✓	✓	✓
12	✓	หยุด	✓
13	✓	✓	✓
14	✓	✓	✓
15	✓	✓	✓
16	✓	✓	✓
17	✓	ขวา	✓
18	✓	✓	✓
19	✓	✓	✓
20	✓	✓	✓
% ความถูกต้อง	95%	85%	100%

ตาราง 6.7 ผลการทดลองเปรียบเทียบความถูกต้องในการสั่งงานแต่ละคำสั่งเสียง โดยผู้ชายที่มีเสียงอยู่ในฐานข้อมูลเป็นผู้สั่งงาน (ต่อ)

3) ผลการทดลองเปรียบเทียบความถูกต้องในการสั่งงานแต่ละคำสั่งเสียง โดยผู้หญิงที่มีเสียงอยู่ในฐานข้อมูลเป็นผู้สั่งงาน

คำสั่ง / ครั้งที่	ซ้าย	ขวา	หน้า	หลัง
1	✓	✓	✓	ซ้าย
2	✓	✓	✓	✓
3	✓	✓	✓	✓
4	✓	✓	✓	✓
5	✓	✓	✓	✓
6	✓	✓	✓	ซ้าย
7	✓	หลัง	✓	✓
8	✓	✓	✓	✓
9	✓	✓	✓	✓
10	✓	✓	✓	✓
11	✓	✓	✓	✓
12	✓	✓	✓	✓
13	✓	✓	✓	✓
14	✓	หลัง	✓	✓
15	✓	✓	✓	✓
16	✓	✓	✓	✓
17	✓	✓	✓	✓
18	✓	✓	✓	✓
19	✓	✓	✓	✓
20	✓	✓	✓	✓
% ความถูกต้อง	100%	90%	100%	90%

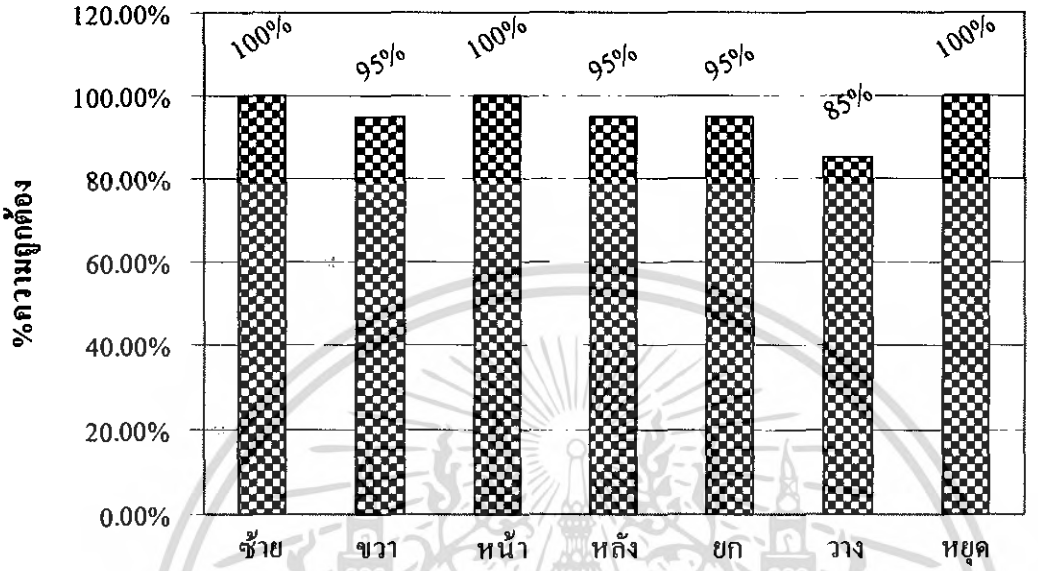
ตาราง 6.8 ผลการทดลองเปรียบเทียบความถูกต้องในการสั่งงานแต่ละคำสั่งเสียง โดยผู้หญิงที่มีเสียงอยู่ในฐานข้อมูลเป็นผู้สั่งงาน

4) การทดลองเปรียบเทียบความถูกต้องในการสั่งงานแต่ละคำสั่งเสียง โดยผู้หญิงที่มีเสียงอยู่ในฐานข้อมูลเป็นผู้สั่งงาน (ต่อ)

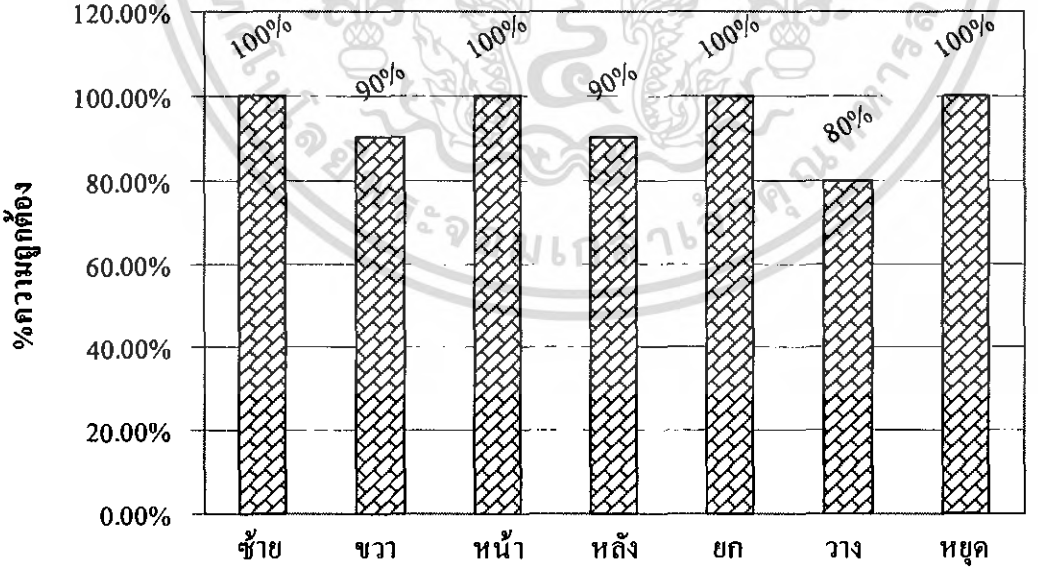
คำสั่ง / ครั้งที่	ยก	วาง	หยุด
1	✓	✓	✓
2	✓	✓	✓
3	✓	หยุด	✓
4	✓	✓	✓
5	✓	✓	✓
6	✓	✓	✓
7	✓	✓	✓
8	✓	ขวา	✓
9	✓	หยุด	✓
10	✓	✓	✓
11	✓	✓	✓
12	✓	✓	✓
13	✓	✓	✓
14	✓	✓	✓
15	✓	✓	✓
16	✓	✓	✓
17	✓	ขวา	✓
18	✓	✓	✓
19	✓	✓	✓
20	✓	✓	✓
% ความถูกต้อง	100%	80%	100%

ตาราง 6.9 ผลการทดลองเปรียบเทียบความถูกต้องในการสั่งงานแต่ละคำสั่งเสียง โดยผู้หญิงที่มีเสียงอยู่ในฐานข้อมูลเป็นผู้สั่งงาน (ต่อ)

5) กราฟผลการทดสอบความถูกต้องในการสั่งงานแต่ละคำสั่งเสียง



รูปที่ 6.2 กราฟการทดสอบความถูกต้องในการสั่งงานแต่ละคำสั่งเสียงของผู้ชายคนที่1



รูปที่ 6.3 กราฟการทดสอบความถูกต้องในการสั่งงานแต่ละคำสั่งเสียงของผู้หญิงคนที่1

บทที่ 7
สรุปผลการทดลอง

7.1 สรุปผลการทดลอง

จากการทดลองระบบรู้จำเสียงพูด โดยทดลองกับโมเดลรถโฟล์คลิฟท์ ได้ผลสรุปดังนี้

1. ในการทดลองเพื่อตรวจสอบความถูกต้องโดยรวมในการสั่งงานของระบบ เปรียบเทียบระหว่างผู้สั่งงานที่มีเสียงอยู่ในฐานข้อมูล เป็นชาย 1 คน และหญิง 1 คน กับผู้สั่งงานที่มีเสียงอยู่นอกฐานข้อมูล เป็นชาย 1 คน และหญิง 1 คนเช่นกัน แต่ละคนพูดสั่งงานด้วยชุดคำสั่งตามตารางที่ 6.1 ทั้งหมด 10 ครั้ง จากการทดลองพบว่า ผู้สั่งงานที่มีเสียงอยู่ในฐานข้อมูลทั้งผู้ชายและผู้หญิง เมื่อสั่งงานจะมีเปอร์เซ็นต์ความถูกต้องสูงกว่าผู้สั่งงานที่ไม่ได้มีเสียงอยู่ในฐานข้อมูล โดยมีเปอร์เซ็นต์ความถูกต้องของผู้ชาย และผู้หญิงเฉลี่ยเท่ากับ 94.29% และ 91.43% ตามลำดับ ส่วนเปอร์เซ็นต์ความถูกต้องของผู้สั่งงานที่มีเสียงอยู่นอกฐานข้อมูลของผู้ชายและผู้หญิงเฉลี่ยเท่ากับ 80.0% และ 75.71% ตามลำดับ

2. ในการทดลองเพื่อตรวจสอบความถูกต้องในการสั่งงานแต่ละคำสั่งเสียง โดยเปรียบเทียบว่าคำสั่งเสียงใดที่สั่งงานได้ถูกต้องมากกว่าและน้อยกว่ากัน โดยผู้สั่งงานเป็นผู้ที่มีเสียงอยู่ในฐานข้อมูล เป็นชาย 1 คน และหญิง 1 คน แต่ละคนพูดสั่งงานด้วยชุดคำสั่งตามตารางที่ 6.1 ทั้งหมด 20 ครั้ง จากการทดลองพบว่า เมื่อสั่งงานคำว่า ขวา หลัง และวาง จะให้เปอร์เซ็นต์ความถูกต้องน้อยกว่าคำสั่งเสียงอื่นๆ

7.2 วิจารณ์ผลการทดลอง

จากการทดลองพบว่าสิ่งที่มีผลต่อการรู้จำเสียงพูดของระบบนั้นมีหลายอย่าง ดังนี้

ผู้สั่งงานที่มีเสียงอยู่ในฐานข้อมูลไม่ว่าจะเป็นผู้ชายหรือผู้หญิง เมื่อสั่งงานจะมีเปอร์เซ็นต์ความถูกต้องมากว่าผู้สั่งงานที่มีเสียงอยู่นอกฐานข้อมูล เพราะถ้าเสียงของผู้สั่งงานมีความคล้ายคลึงกับเสียงในฐานข้อมูล จะทำให้ระบบสามารถแยกแยะคำสั่งเสียงเหล่านั้นได้ง่ายกว่าเสียงที่แตกต่างไปจากเสียงในฐานข้อมูล การพูดในแต่ละครั้งของแต่ละคน จะมีลักษณะแตกต่างกันไป ทั้งความสั้นยาวของคำพูด น้ำเสียง และระดับเสียงของผู้พูด ดังนั้นควรให้ระบบเรียนรู้ลักษณะเสียงที่ต่าง ๆ กัน เพื่อการใช้งานที่หลากหลายมากขึ้น

ความผิดพลาดของแต่ละคำสั่งเสียงอาจเกิดจากลักษณะคำพูดที่คล้ายคลึงกัน เช่น ขวากับวาง และวางกับหลัง จะเห็นว่าคำเหล่านี้จะมีพยัญชนะ สระ หรือตัวสะกดที่คล้ายกัน จึงอาจจะทำให้ระบบประมวลผลผิดพลาดได้ การออกเสียงของผู้ทำการทดสอบก็มีส่วนทำให้ผลการทดลองผิดพลาดเช่นกัน เช่นเสียงหายใจแรงก่อนที่จะพูด หรือการออกเสียงที่ไม่ชัด เป็นต้น

นอกจากนี้ สภาพแวดล้อมก็มีส่วนในการรู้จำเสียงพูดของระบบเช่นกัน โดยการทดลองในสภาพห้องเงียบจะให้ผลดีกว่าสภาพห้องที่มีเสียงรบกวน เนื่องจากในสภาพห้องที่มีเสียงรบกวนนั้นระบบอาจจะทำการประมวลผลเสียงรบกวนด้วย ยิ่งเสียงรบกวนมากความผิดพลาดยิ่งสูง

7.3 ปัญหาและแนวทางการพัฒนา

ในการทดลองให้ระบบรู้จำเสียงพูดจะพบว่าเกิดความผิดพลาดขึ้นเป็นส่วนน้อย ซึ่งอาจแก้ไขโดยเพิ่มจำนวนฐานข้อมูลให้มีความหลากหลายมากยิ่งขึ้น เมื่อสั่งงานระบบจะได้แยกแยะแต่ละคำสั่งเสียงได้ถูกต้องมากขึ้น นอกจากนี้ยังมีผลของอุปกรณ์บันทึกเสียง ดังนั้นเราจึงควรเลือกใช้อุปกรณ์บันทึกเสียงที่ดี เช่น ไมโครโฟน และซาวด์การ์ด เพื่อให้คุณภาพของเสียงที่จะนำไปประมวลผลมีคุณภาพดีขึ้น และถ้าต้องการให้มีความถูกต้องในการสั่งงานระบบมากขึ้น สามารถเพิ่มส่วนในการวิเคราะห์สัญญาณเสียงให้ละเอียดมากยิ่งขึ้น เพื่อแยกแยะคำที่คล้ายคลึงกัน แต่ต้องคำนึกถึงความเร็วในการประมวลผลด้วย

เราสามารถนำระบบรู้จำเสียงพูดนี้ไปพัฒนาใช้งานกับเครื่องจักรกลขนาดเล็กที่มีการใช้งานอยู่จริง โดยการสร้างวงจรอินเตอร์เฟสระหว่างคอมพิวเตอร์กับส่วนควบคุมการทำงานของเครื่องจักร และสามารถนำระบบรู้จำเสียงพูดที่ได้พัฒนาในคอมพิวเตอร์นี้ไปพัฒนาลงในชิปเดี่ยว (Single Chip) เพื่อความสะดวกในการใช้งาน

หนังสืออ้างอิง

1. **Hidden Markov Model.** from http://en.wikipedia.org/wiki/Hidden _Markov_model
2. Amarin Deemagarn, Asanee Kawtrakul. **Thai Connected Digit Speech Recognition Using Hidden Markov Models.** from http://www.isca-speech.org
3. Ben Milner, Xu Shao. **Speech Reconstruction from Mel-Frequency Cepstral Coefficients using a Source-Filter Model.** from http://www.dcs.shef.ac.uk/~xu
4. Li Tan, Montri Karnjanadecha. **Modified Mel-Frequency Cepstral Coefficients.** from http://fivedots.coe.psu.ac.th/~montri
5. จุฬารัตน์ ตันประเสริฐ. **การสร้างรหัสลับเสียงพูดภาษาไทย.** จาก http://micro.se-ed.com/content/mc188/MC188_75.asp
6. Speecy. **หลักการรู้จำเสียง.** จาก http://thaispeech.longdo.org/speechy/thaispeech1/index_html
7. รศ.ดร.สมศักดิ์ ชุ่มช่วย. **Digital Filter.** จาก http://www.kmitl.ac.th/~kchsomsa/
8. **Window function.** from http://en.wikipedia.org/wiki/Window_function
9. วรพจน์ กรแก้ววัฒนกุล, ชัยวัฒน์ ลิ้มพรจิตรวิไล. **เรียนรู้และปฏิบัติการไมโครคอนโทรลเลอร์ MCS-51.**

ภาคผนวก ก

Source Code ในโปรแกรมแมทแล็ป

1. energy.m ไฟล์สำหรับการหาช่วงต้นและช่วงท้ายของสัญญาณเสียง

```
%---------------------------------%
%Read file wave
%---------------------------------%
figure(2);
clf;
figure(1);
clf;
[signal,fs,nbits,opts] = wavread('C:\Documents and Settings\Nancy\Desktop\wave');

%---------------------------------%
% Preempazis speech signal
%---------------------------------%
preemphasis=filter([1,-0.95],1,signal);
subplot(3,1,1);
plot(preemphasis);
axis([0 length(preemphasis)  -1 1]);

%---------------------------------%
% Calculate energy
%---------------------------------%
signal=preemphasis';
winSizeMs=15;
winShiftMs=5;

winSize=round((fs*winSizeMs)/1000);
winShift=round((fs*winShiftMs)/1000);
mywindow=window(@hamming,winSize);
```

```
j=1;
for i=1:winShift:length(signal)-winSize
        energy(j)=(sum((abs(signal(i:i+winSize-1))).*mywindow'));
        j=j+1;
end

%---------------------------------%
% Calculate and Print thresdshold
%---------------------------------%
normal=400/max(energy);
energy=energy*normal;
thredshold=max(energy)*0.1;
thredshold1=max(energy)*0.2;
subplot(3,1,2);
plot(energy);
axis([0 length(energy) 0 max(energy)]);
linet=energy;
linet(1:length(linet))=thredshold1;
hold on;
plot(linet,'y');
plot(linet/2,'r');
```

```
%---------------------------------%
% Define start position
%---------------------------------%
leveluv=min(energy(1:5));
j=1;
while(energy(j)<thredshold1)
        j=j+1;
end
endpointl=floor(winShift*j);

while(energy(j)>max(energy)*0.02)                    %Threshold value
        j=j-1;
end
endpoints=floor(winShift*j);
fstart=j+2;

%---------------------------------%
% Define end position
%---------------------------------%
j=length(energy);
if energy(j)<thredshold/2
        while(energy(j)<thredshold)
                j=j-1;
        end
        while(energy(j)>(thredshold/2))
                j=j+1;
        end
end
endpointe=floor(winShift*(j+0.5));
signal=signal(endpoints:endpointe);
```

```
subplot(3,1,1);
hold on;
stem(endpointl, 1,'-.','y');
stem(endpointl,-1,'-.','y');
stem(endpoints, 1,'-.','r');
stem(endpoints,-1,'-.','r');
stem(endpointe, 1,'-.','r');
stem(endpointe,-1,'-.','r');

subplot(3,1,3);
hold on;
plot(signal);
axis([0 length(signal) min(signal) max(signal) ]);
figure(2);
plot(energy,'r');
axis([0 length(energy) 0 400]);
hold on;
```

2. wavetomfcc.m ไฟล์สำหรับการหา MFCC Vectors

```
function parameter= wavetomfcc(y, fs, FP)
%       wave2mfcc: Wave to MFCC (Mel-Frequency Cepstral Cofficient) conversion
%       Usage:
%       parameter = wave2mfcc(y, fs, FP)
%       parameter: MFCC and log energy, plus their delta value.
%       fs: sampling rate
%       FP: Parameters for deriving the speech features

if nargin<2; fs=8000; end
if nargin<3, FP=setFeatureParam(fs); end

y=double(y);    % Convert to double
y=y-mean(y);    % Shift to zero mean

% ----------First Step : pre-emphasis.
y = filter([1, -0.95], 1, y);

% ----------Second Step: frame blocking.
framedY = buffer2(y, FP.frameSize, FP.overlap);
filterBankParam = getTriFilterParam(FP.frameSize, fs, FP.filterNum, 0);    % Parameters for
                                                                           triangular filterbank
parameter = [];
for i = 1:size(framedY, 2),
        % ----------Third Step: hamming window.
        Wframe  = hamming(FP.frameSize).*framedY(:,i);
```

```
        % ----------Forth Step: fast fourier transform.
        fftMag = abs(fft(Wframe));
        halfIndex = floor((FP.frameSize+1)/2);
        fftMag = fftMag(1:halfIndex);
        fftMag = interp1(1:halfIndex,fftMag,1:1/FP.alpha:halfIndex)';          % VTLN
        fftMag = [fftMag;zeros(halfIndex-length(fftMag),1)];

        % ----------Fifth Step: triangular bandpass filter.
        tbfCoef = triBandFilter(fftMag, FP.filterNum, filterBankParam);

        % ----------Sixth Step: cosine transform. (Using DCT to get L-order MFCC parameters.)
        mfcc = melCepstrum(FP.mfccDim, FP.filterNum, tbfCoef);
        parameter = [parameter mfcc'];
end

if (FP.useEnergyFeature==1)
        energy = sum(framedY.^2)/FP.frameSize;
        logEnergy = 10*log10(eps+energy);
        parameter = [parameter; logEnergy];
end
```

```
% ----------compute delta energy and delta cepstrum
if (FP.useDelta>=1)
        deltaWindow = 2;
        paraDelta = deltaFunction(deltaWindow, parameter);
        parameter = [parameter; paraDelta];
end
if (FP.useDelta==2)
        paraDeltaDelta = deltaFunction(deltaWindow, paraDelta);
        parameter = [parameter; paraDeltaDelta];
end

% ----------Subfunction----------%

% ----------Triangular band-pass filters
function tbfCoef = triBandFilter(fftMag, P, filterBankParam)
fstart=filterBankParam(1,:);
fcenter=filterBankParam(2,:);
fstop=filterBankParam(3,:);

% Triangular bandpass filter.
for i=1:P
        for j = fstart(i):fcenter(i),
                filtmag(j) = (j-fstart(i))/(fcenter(i)-fstart(i));
        end
        for j = fcenter(i)+1:fstop(i),
                filtmag(j) = 1-(j-fcenter(i))/(fstop(i)-fcenter(i));
        end
        tbfCoef(i) = sum(fftMag(fstart(i):fstop(i)).*filtmag(fstart(i):fstop(i))');
end
tbfCoef=log(eps+tbfCoef.^2);
```

```
% ----------TBF coefficients to MFCC
function mfcc = melCepstrum(L, P, tbfCoef)
% -----DCT to find MFCC
for i = 1:L
        coef = cos((pi/P)*i*(linspace(1,P,P)-0.5))';
        mfcc(i) = sum(coef.*tbfCoef');
end

% ----------Delta function
function parameter = deltaFunction(deltaWindow,parameter)
% compute delta cepstrum and delta log energy.
rows  = size(parameter,1);
cols  = size(parameter,2);
temp  = [parameter(:,1)*ones(1,deltaWindow) parameter
parameter(:,end)*ones(1,deltaWindow)];
temp2 = zeros(rows,cols);
denominator = sum([1:deltaWindow].^2)*2;

for i = 1+deltaWindow : cols+deltaWindow,
        subtrahend = 0;
        minuend    = 0;
        for j = 1 : deltaWindow,
                subtrahend = subtrahend + temp(:,i+j)*j;
                minuend = minuend + temp(:,i-j)*(-j);
        end;
        temp2(:,i-deltaWindow) = (subtrahend + minuend)/denominator;
end;
parameter = temp2;
```

```
function filterBankParam = getTriFilterParam(frameSize, fs, filterNum, plotOpt)
% getTriParam: Get parameters of triangular filter bank
maxMelFreq = freq2mel(fs/2);
sideWidth=maxMelFreq/(filterNum+1);
index=0:filterNum-1;
filterBankParam=floor(mel2freq([index; index+1; index+2]*sideWidth)/fs*frameSize)+1;
filterBankParam(end, end)= frameSize/2;

% ----------Normal frequency to mel-scaled frequency conversion
function mel = freq2mel(freq)
mel = 2595*log10(1+freq/700);

% ----------Mel-scaled frequency to normal frequency conversion
function freq = mel2freq(mel)
freq = 700*(10.^(mel/2595)-1);

function out = buffer2(y, frameSize, overlap)
% BUFFER2 Frame blocking
% This is almost the same as "buffer" except that there is no leading zeros

if nargin<3, overlap=0; end
if nargin<2, frameSize=256; end
y = y(:);
step = frameSize-overlap;
frameCount = floor((length(y)-overlap)/step);
out = zeros(frameSize, frameCount);
for i=1:frameCount,
        startIndex = (i-1)*step+1;
        out(:, i) = y(startIndex:(startIndex+frameSize-1));
end
```

3. ฟังก์ชั่นต่างๆของไฟล์ autotrain.m

1) endpointdt

```
%---------------------------------%
%Read file wave
%---------------------------------%
function signal=endpointdt(signal,fs)

%---------------------------------%
% Preempazis speech signal
%---------------------------------%
preemphasis=filter([1,-0.95],1,signal);

%---------------------------------%
% Calculate energy
%---------------------------------%
signal=preemphasis';
winSizeMs=15;
winShiftMs=5;

winSize=round((fs*winSizeMs)/1000);
winShift=round((fs*winShiftMs)/1000);
mywindow=window(@hamming,winSize);
j=1;
for i=1:winShift:length(signal)-winSize
  energy(j)=(sum((abs(signal(i:i+winSize-1))).*mywindow'));
  zerocr(j)=sum(abs(sgnz(signal(i+1:i+winSize))-sgnz(signal(i:i+winSize-1)))/2)/winSize;
  bandcr(j)=sum(abs(sgnb(signal(i+1:i+winSize))-sgnb(signal(i:i+winSize-1))));
  j=j+1;
end
```

```
%----------------------------------%
% Calculate and Print thresdshold
%----------------------------------%
normal=400/max(energy);
energy=energy*normal;
 normal=150/max(zerocr);
 zerocr=zerocr*normal;
thredshold=max(energy)*0.1;
thredshold1=max(energy)*0.2;

linet=energy;
linet(1:length(linet))=thredshold1;

%----------------------------------%
% Define start position
%----------------------------------%
leveluv=min(energy(1:5));
levelt=max(energy)*0.02;
j=1;
if(leveluv>levelt) levelt=leveluv; end
while(energy(j)<thredshold1)
   j=j+1;
end
endpointl=floor(winShift*j);

while(energy(j)>levelt) %Threshold value
   j=j-1;
end
endpoints=floor(winShift*j);
fstart=j+2;
```

```
%---------------------------------%
% Define end position
%---------------------------------%
j=length(energy);
if energy(j)<thredshold/2
while(energy(j)<thredshold)
   j=j-1;
end

while(energy(j)>(thredshold/2))
   j=j+1;
end
end
endpointe=floor(winShift*(j+0.5));

signal=signal(endpoints:endpointe);
```

2) find_ MFCC

```
function [MFCC,DeltaMFCC,DeltaDeltaMFCC,MixMFCC] = findMFCC(s,num_MFCC)
w=512;
ow=100;
L=length(s);
b=1; m=1;
e=b+w-1;
while e<L
  x=s(b:e);
  f=abs(fft(x'.*hanning(w)));
  f=f(1:(w/2));
  f=f.^2;
  ef=log10(mel(f));
  MFCC(m,:)=dct(ef)*200;
  tDc(m)=b+w/2;
  m=m+1;
  b=b+ow;
  e=b+w-1;
end
MFCC = MFCC(:,1:num_MFCC) ;
```

```
[a,b] = size(MFCC) ;
DeltaMFCC = zeros(a,b) ;
DeltaMFCC(1,:) = zeros(1,b) ;
for i=2:a
   DeltaMFCC(i,:) = MFCC(i,:)-MFCC(i-1,:) ;
end
DeltaDeltaMFCC = zeros(a,b) ;
DeltaDeltaMFCC(1,:) = zeros(1,b) ;
DeltaDeltaMFCC(2,:) = zeros(1,b) ;
for i=3:a
  DeltaDeltaMFCC(i,:) = DeltaMFCC(i,:)-DeltaMFCC(i-1,:) ;
end
MixMFCC(:,1:b) = MFCC(:,:) ;
MixMFCC(:,b+1:2*b) = DeltaMFCC(:,:) ;
MixMFCC(:,2*b+1:3*b) = DeltaDeltaMFCC(:,:) ;
```

3) init_mhmm

```
function [init_state_prob, transmat, mixmat, mu, Sigma] =  init_mhmm(data, Q, M, cov_type,
                                                                left_right)
% INIT_MHMM Compute initial param. estimates for an HMM with mixture of Gaussian
outputs.
% Inputs:
% data(:,t,l) = observation vector at time t in sequence l
% Q = num. hidden states
% M = num. mixture components
% cov_type = 'full', 'diag' or 'spherical'
% left_right = 1 if the model is a left-to-right HMM, 0 otherwise
%
% Outputs:
% init_state_prob(i) = Pr(Q(1) = i)
% transmat(i,j) = Pr(Q(t+1)=j | Q(t)=i)
% mixmat(j,k) = Pr(M(t)=k | Q(t)=j) where M(t) is the mixture component at time t
% mu(:,j,k) = mean of Y(t) given Q(t)=j, M(t)=k
% Sigma(:,:,j,k) = cov. of Y(t) given Q(t)=j, M(t)=k

O = size(data, 1);
T = size(data, 2);
nex = size(data, 3);
data = reshape(data, [O T*nex]);
init_state_prob = normalise(ones(Q,1));
transmat = mk_stochastic(ones(Q,Q));
if M > 1
  mixmat = mk_stochastic(rand(Q,M));
else
  mixmat = ones(Q, 1);
```

```
end
```

```
if 0
  Sigma = repmat(eye(O), [1 1 Q M]);
  % Initialize each mean to a random data point
  indices = randperm(T);
  mu = reshape(data(:,indices(1:(Q*M))), [O Q M]);
end

% Initialize using K-means, where K = Q*M
% We should really segment the sequence uniformly into Q strips,
% and run M-means on each segment.
mix = gmm(O, Q*M, cov_type);
options = foptions;
max_iter = 5;
options(14) = max_iter;
mix = gmminit(mix, data', options);
mu = reshape(mix.centres', [O Q M]);
i = 1;
for q=1:Q
  for m=I:M
    switch cov_type
      case 'diag',
           Sigma(:,:,q,m) = diag(mix.covars(i,:));
      case 'full',
           Sigma(:,:,q,m) = mix.covars(:,:,i);
      case 'spherical',
           Sigma(:,:,q,m) = mix.covars(i) * eye(O);
    end
    i = i + 1;
  end
end
```

4) learn_mhmm

```
function [LL, init_state_prob, transmat, mu, Sigma, mixmat, mu_history] = learn_mhmm(data,
init_state_prob, transmat, mu, Sigma, mixmat, max_iter, thresh, verbose, cov_type, static2)
% LEARN_MHMM Compute the ML parameters of an HMM with mixtures of Gaussians output
    using EM.
%
% [LL, PRIOR, TRANSMAT, MU, SIGMA, MIXMAT, MU_HISTORY] =
    LEARN_MHMM(DATA, PRIOR0, TRANSMAT0,
% MU0, SIGMA0, MIXMAT0) computes the ML estimates of the following parameters,
% where, for each time t, Q(t) is the hidden state, M(t) is the mixture component, and Y(t) is
% the observation.
%   prior(i) = Pr(Q(1) = i),
%   transmat(i,j) = Pr(Q(t+1)=j | Q(t)=i)
%   mixmat(j,k) = Pr(M(t)=k | Q(t)=j)
%   mu(:,j,k) = E[Y(t) | Q(t)=j, M(t)=k ]
%   Sigma(:,:,j,k) = Cov[Y(t) | Q(t)=j, M(t)=k]
% PRIOR0 is the initial estimate of PRIOR, etc.
% To learn an HMM with a single Gaussian output, just set mixmat = ones(Q,1).
%
% DATA(:,t,l) is the observation vector at time t for sequence l. If the sequences are of
% different lengths, you can pass in a cell array, so DATA{l} is an O*T matrix.
%
% LL is the "learning curve": a vector of the log lik. values at each iteration.
% LL might go positive, since prob. densities can exceed 1, although this probably
% indicates that something has gone wrong e.g., a variance has collapsed to 0.
% MU_HISTORY(:,:,:,r) is MU at iteration r.
%
% There are several optional arguments, which should be passed in the following order
%  LEARN_MHMM(DATA, INIT_STATE_PROB, TRANSMAT, MU, SIGMA, MIXMAT, ...
```

```
%    MAX_ITER, THRESH, VERBOSE, COV_TYPE, STATIC)
% These have the following meanings
%
% max_iter = max. num EM steps to take (default 10)
% thresh = threshold for stopping EM (default 1e-4)
% verbose = 0 to suppress the display of the log lik at each iteration (Default 1).
% cov_type = 'full', 'diag' or 'spherical' (default 'full')
% static = 1 if we don't want to re-estimate prior and transmat, i.e., no dynamics (default = 0)
%
%learn_mhmm(data, init_state_prob, transmat, mu, Sigma, mixmat, max_iter, thresh, verbose,
   cov_type, static)

if nargin<7, max_iter = 10; end
if nargin<8, thresh = 1e-4; end
if nargin<9, verbose = 1; end
if nargin<10, cov_type = 'full'; end
if nargin<11, static2 = 0; end

if ~iscell(data)
  data = num2cell(data, [1 2]); % each elt of the 3rd dim gets its own cell
end
numex = length(data);

previous_loglik = -inf;
loglik = 0;
converged = 0;
iter = 1;
LL = [];
```

```
O = size(data{1},1);
Q = length(init_state_prob);
M = size(mixmat,2);
mu_history = zeros(O,Q,M,max_iter);


while (iter <= max_iter) & ~converged
 % E step
 [loglik, exp_num_trans, exp_num_visits1, postmix, m, ip, op] = ...
    ess_mhmm(init_state_prob, transmat, mixmat, mu, Sigma, data);
 [a,b] = size(postmix) ;
 status = 0 ;
 for j=1:a
    for k=1:b
        if postmix(j,k) <= 0
           status = 1 ;
        end
    end
 end
 if status == 1
  break ;
 end
 if verbose, fprintf('iteration %d, loglik = %f\n', iter, loglik); end
```

```
% M step
if ~static2
  init_state_prob = normalise(exp_num_visits1);
  transmat = mk_stochastic(exp_num_trans);
end
if M == 1
  mixmat = ones(Q,1);
else
  mixmat = mk_stochastic(postmix);
end

for j=1:Q
  for k=1:M
    mu(:,j,k) = m(:,j,k) / postmix(j,k);

    if cov_type(1) == 's'
        s2 = (1/O)*( (ip(j,k)/postmix(j,k)) - mu(:,j,k)'*mu(:,j,k) );
        Sigma(:,:,j,k) = s2 * eye(O);
    else
        %SS = op(:,:,j,k)/postmix(j,k) - mu(:,j,k)*mu(:,j,k)';
  SS = op(:,:,j,k)/postmix(j,k) ;
        if cov_type(1)=='d'
          SS = diag(diag(SS));
        end
        Sigma(:,:,j,k) = SS;
    end

  end
end
```

```
  mu_history(:,:,:,iter) = mu;
  ll_history(iter) = loglik;
  converged = em_converged(loglik, previous_loglik, thresh);
  previous_loglik = loglik;
  iter =  iter + 1;
  LL = [LL loglik];
end

function [loglik, exp_num_trans, exp_num_visits1, postmix, m, ip, op] = ...
    ess_mhmm(prior, transmat, mixmat, mu, Sigma, data)
% ESS_MHMM Compute the Expected Sufficient Statistics for a MOG Hidden Markov Model.
%
% Outputs:
% exp_num_trans(i,j)  = sum_l sum_{t=2}^T Pr(Q(t-1) = i, Q(t) = j| Obs(l))
% exp_num_visits1(i)  = sum_l Pr(Q(1)=i | Obs(l))
% postmix(i,k) = sum_l sum_t w(i,k,t) where w(i,k,t) = Pr(Q(t)=i, M(t)=k | Obs(l))  (posterior
mixing weights)
% m(:,i,k)  = sum_l sum_t w(i,k,l) * Obs(:,t,l)
% ip(i,k) = sum_l sum_t w(i,k,l) * Obs(:,t,l)' * Obs(:,t,l)
% op(:,:,i,k) = sum_l sum_t w(i,k,l) * Obs(:,t,l) * Obs(:,t,l)'
%
% where Obs(l) = Obs(:,:,l) = O_1 .. O_T for sequence l


verbose = 0;
```

```
%[O T numex] = size(data);
numex = length(data);
O = size(data{1},1);
Q = length(prior);
M = size(mixmat,2);
exp_num_trans = zeros(Q,Q);
exp_num_visits1 = zeros(Q,1);
postmix = zeros(Q,M);
m = zeros(O,Q,M);
op = zeros(O,O,Q,M);
ip = zeros(Q,M);
loglik = 0;
if verbose, fprintf(1, 'forwards-backwards example # '); end
for ex=1:numex
 if verbose, fprintf(1, '%d ', ex); end
 %obs = data(:,:,ex);
 obs = data{ex};
 T = size(obs,2);
 [B, B2] = mk_mhmm_obs_lik(obs, mu, Sigma, mixmat);
 [gamma, xit, current_loglik, gamma2] = forwards_backwards_mix(prior, transmat, B, B2,
mixmat);
 loglik = loglik + current_loglik;
 if verbose, fprintf(1, 'll at ex %d = %f\n', ex, loglik); end
```

```
exp_num_trans = exp_num_trans + sum(xit,3);
exp_num_visits1 = exp_num_visits1 + gamma(:,1);
postmix = postmix + sum(gamma2,3);
for j=1:Q
 for k=1:M
  w = reshape(gamma2(j,k,:), [1 T]); % w(t) = Pr(Q(t)=j, M(t)=k | obs)
  wobs = obs .* repmat(w, [O 1]); % wobs(:,t) = w(t) * obs(:,t)
  m(:,j,k) = m(:,j,k) + sum(wobs, 2); % m(:) = sum_t w(t) obs(:,t)
  %op(:,:,j,k) = op(:,:,j,k) + wobs * obs'; % op(:,:) = sum_t w(t) * obs(:,t) * obs(:,t)'
  ip(j,k) = ip(j,k) + sum(sum(wobs .* obs, 2)); % ip = sum_t w(t) * obs(:,t)' * obs(:,t)
 end
end
[d N] = size(obs);
for j=1:Q
  for k=1:M
    w = reshape(gamma2(j,k,:), [1 T]) ;
    new_mu(:,j,k) = m(:,j,k) / postmix(j,k) ;
    temp = new_mu(:,j,k)*ones(1,N);
    op(:,:,j,k) = op(:,:,j,k)+ (repmat(w, [O 1]).*(obs-temp))*(obs-temp)' ;
 end
 end
end
if verbose, fprintf(1, '\n'); end
```

4. test.m ไฟล์สำหรับการเทสต์สัญญาณเสียงทั้งคำ

```
%-----------------------------------------------%
            %Open Port
%-----------------------------------------------%
warning off MATLAB:serial:fscanf:unsuccessfulRead
C3=serial('com3');
fopen(C3);
fprintf(C3,'%s','S');
while 1==1
   input=fscanf(C3,'%c',1);
    if input=='C'



%-----------------------------------------------%
          %Speech Recognition System
%-----------------------------------------------%
%[s,fs,nbits,opts] = wavread...
   %('C:\Documents and Settings\admin\Desktop\pu');
s=wavrecord(8000,8000);
fs=8000;
test1_System;
test2_System;
test3_System;
```

```
%----------Parameter---------%
r1=(result(1)+result2(1)+result3(1))/3;
r2=(result(2)+result2(2)+result3(2))/3;
r3=(result(3)+result2(3)+result3(3))/3;
r4=(result(4)+result2(4)+result3(4))/3;
r5=(result(5)+result2(5)+result3(5))/3;
r6=(result(6)+result2(6)+result3(6))/3;
r7=(result(7)+result2(7)+result3(7))/3;
R=[r1 r2 r3 r5 r6];
RR=[r1 r2 r3 r4 r5 r6];
RRR=[r1 r2 r3 r4 r5 r6 r7];

%-----------------------------%
 c7=result(7);
 c4=result(4);
for n=1:5
   if max(R) == R(n)

     if n==1
        c(n)=result3(n);
        if c(n)>c7 | c(n) > result(3)
          fprintf(C3,'%s','L');
        elseif max(RR) == r4
            if c4>c(n)
               fprintf(C3,'%s','B');
            end
```

```
    elseif max(RRR) == r7
        if c7>c(n)
            fprintf(C3,'%s','S');
        end
    else fprintf(C3,'%s','S')
    end

elseif n==2
    c(n)=result(n);
    if result(6) > result(2) & result(6) > result3(2)  %Becouse R(2) close to R(6)
     fprintf(C3,'%s','D');
    elseif c(n)>c7 | c(n) > result2(3)
     fprintf(C3,'%s','R');
    elseif max(RR) == r4
        if c4>c(n)
            fprintf(C3,'%s','B');
        end
    elseif max(RRR) == r7
        if c7>c(n)
            fprintf(C3,'%s','S');
        end
    else fprintf(C3,'%s','S')
    end
```

```
elseif n==3
  c(n)=result(n);
  if c(n)>c7 | c(n) > result3(5)
   fprintf(C3,'%s','F');
  elseif max(RR) == r4
     if c4>c(n)
        fprintf(C3,'%s','B');
     end
  elseif max(RRR) == r7
     if c7>c(n)
        fprintf(C3,'%s','S');
     end
  else fprintf(C3,'%s','S')
  end

elseif n==4
  n=n+1;
  c(n)=result3(n);
  if c(n)>result3(7)
   fprintf(C3,'%s','U');
  elseif max(RRR) == r7
     if c7>c(n)
        fprintf(C3,'%s','S');
     end
  elseif max(RR) == r4
     if c4>c(n)
        fprintf(C3,'%s','B');
     end
  else fprintf(C3,'%s','S')
  end
```

```
    elseif n==5
      n=n+1;
      c(n)=result(n);
      if c(n)>c7 | c(n)>result3(7)
       fprintf(C3,'%s','D');
      elseif max(RR) == r4
         if c4>c(n)
            fprintf(C3,'%s','B');
         end
      elseif max(RRR) == r7
         if c7>c(n)
            fprintf(C3,'%s','S');
         end
      else fprintf(C3,'%s','S')
      end

    end
  end
end
```

5. ฟังก์ชั่นต่างๆของไฟล์ test.m

1) log_lik_mhmm

```
function loglik = log_lik_mhmm(data, prior, transmat, mixmat, mu, Sigma)
% LOG_LIK_MHMM Compute the log-likelihood of a dataset using a mixture of Gaussians
HMM
% loglik = log_lik_mhmm(data, prior, transmat, mixmat, mu, sigma)

[O T ncases] = size(data);
loglik = 0;
for m=1:ncases
  obslik = mk_mhmm_obs_lik(data(:,:,m), mu, Sigma, mixmat);
  [alpha, LL, xi] = forwards(prior, transmat, obslik);
  loglik = loglik + LL;
end
```

ภาคผนวก ข

Source Code ในไมโครคอนโทรลเลอร์

```
;*****************************************************************************
; Program        : Serial Port
; Controller     : AT89S8252
; XTAL           : 11.0592 MHz
;*****************************************************************************

                ORG             0000H
                SJMP            MAIN
                ORG             0003H
                JMP             SPEAK
                ORG             0030H

;*****************************************************************************
;Main Program : set up some involving registers
;*****************************************************************************

        MAIN:   CLR     A                       ;Clear Accumulator

                MOV     P0,#00H                 ;Clear port 0
                MOV     IE,#10000001B           ;Enable INT0 / Disable timer 1

                MOV     TMOD,#00100000B         ;Timer 1 mode 2
                MOV     TH1,#0FDH               ;Baud rate = 9600 bps
                MOV     TL1,TH1

                MOV     SCON,#01010000B         ;Serial port in mode 1
                MOV     TCON,#01000001B
```

```
;*******************************************************************************
;Index : output loop
;*******************************************************************************

INDEX: JNB   RI,INDEX
       MOV   A,SBUF
       CLR   RI

;*******************************************************************************
;Subroutine : Output Condition
;*******************************************************************************

LEFT:  CJNE  A,#'L',RIGHT
       MOV   P0,#00001000B
       AJMP  INDEX

RIGHT:CJNE   A,#'R',FRONT
       MOV   P0,#00000001B
       AJMP  INDEX

FRONT:CJNE   A,#'F',BACK
       MOV   P0,#000001001B
       AJMP  INDEX

BACK:  CJNE  A,#'B',UP
       MOV   P0,#00000110B
       AJMP  INDEX
```

```
UP:     CJNE   A,#'U',DOWN
        MOV    P0,#00010000B
        AJMP   INDEX

DOWN:CJNE   A,#'D',STOP
        MOV    P0,#00100000B
        AJMP   INDEX

STOP:  MOV    P0,#00H
        AJMP   INDEX

;*************************************************************************
;Interrupt service routine : Push-to-talk
;*************************************************************************

SPEAK : MOV  SBUF,#'C'
         JNB    TI,$
         CLR   TI
         RETI

END
```

ภาคผนวก ค

Datasheet

Microcontroller AT89S8252

Features

- Compatible with MCS®-51 Products
- 8K Bytes of In-System Reprogrammable Downloadable Flash Memory
  - SPI Serial Interface for Program Downloading
  - Endurance: 1,000 Write/Erase Cycles
- 2K Bytes EEPROM
  - Endurance: 100,000 Write/Erase Cycles
- 4V to 6V Operating Range
- Fully Static Operation: 0 Hz to 24 MHz
- Three-level Program Memory Lock
- 256 x 8-bit Internal RAM
- 32 Programmable I/O Lines
- Three 16-bit Timer/Counters
- Nine Interrupt Sources
- Programmable UART Serial Channel
- SPI Serial Interface
- Low-power Idle and Power-down Modes
- Interrupt Recovery from Power-down
- Programmable Watchdog Timer
- Dual Data Pointer
- Power-off Flag

Description

The AT89S8252 is a low-power, high-performance CMOS 8-bit microcontroller with 8K bytes of downloadable Flash programmable and erasable read-only memory and 2K bytes of EEPROM. The device is manufactured using Atmel's high-density nonvolatile memory technology and is compatible with the industry-standard 80C51 instruction set and pinout. The on-chip downloadable Flash allows the program memory to be reprogrammed in-System through an SPI serial interface or by a conventional nonvolatile memory programmer. By combining a versatile 8-bit CPU with downloadable Flash on a monolithic chip, the Atmel AT89S8252 is a powerful microcontroller, which provides a highly-flexible and cost-effective solution to many embedded control applications.

The AT89S8252 provides the following standard features: 8K bytes of downloadable Flash, 2K bytes of EEPROM, 256 bytes of RAM, 32 I/O lines, programmable watchdog timer, two data pointers, three 16-bit timer/counters, a six-vector two-level interrupt architecture, a full duplex serial port, on-chip oscillator, and clock circuitry. In addition, the AT89S8252 is designed with static logic for operation down to zero frequency and supports two software selectable power saving modes. The Idle Mode stops the CPU while allowing the RAM, timer/counters, serial port, and interrupt system to continue functioning. The Power-down mode saves the RAM contents but freezes the oscillator, disabling all other chip functions until the next external interrupt or hardware reset.

The downloadable Flash can be changed a single byte at a time and is accessible through the SPI serial interface. Holding RESET active forces the SPI bus into a serial programming interface and allows the program memory to be written to or read from unless lock bits have been activated.

ATMEL®

8-bit Microcontroller with 8K Bytes Flash

AT89S8252

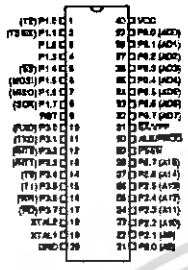
Not Recommended for New Designs. Use AT89S8253.

0401G–MICRO–3/06

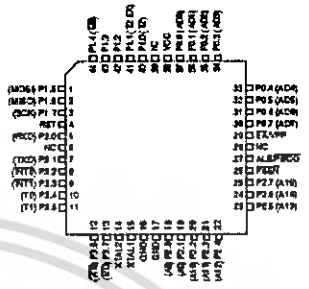
ATMEL

Pin Configurations

PDIP



TQFP



PLCC

Pin Description

VCC Supply voltage.

GND Ground.

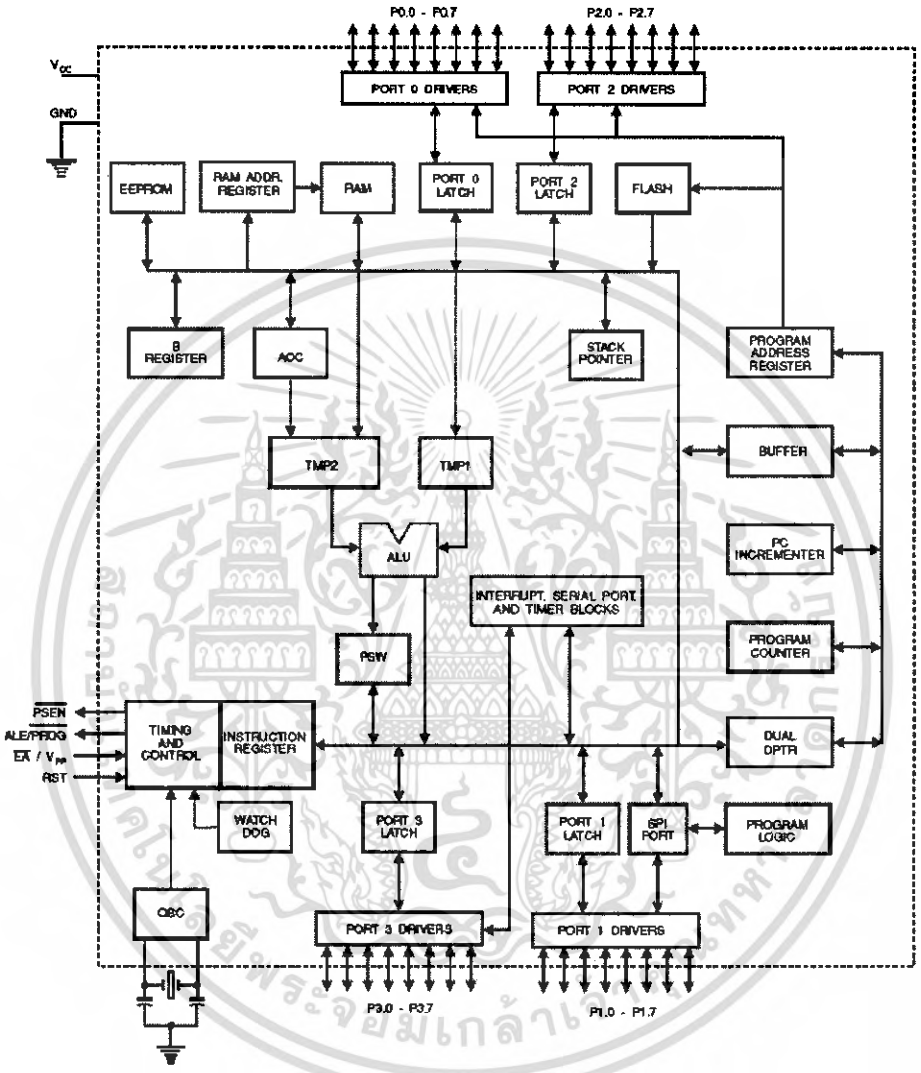
Port 0 Port 0 is an 8-bit open drain bi-didirectional I/O port. As an output port, each pin can sink eight TTL inputs. When 1s are written to port 0 pins, the pins can be used as high-impedance inputs.

Port 0 can also be configured to be the multiplexed low-order address/data bus during accesses to external program and data memory. In this mode, P0 has internal pull-ups.

Port 0 also receives the code bytes during Flash programming and outputs the code bytes during program verification. External pull-ups are required during program verification.

Port 1 Port 1 is an 8-bit bi-directional I/O port with internal pull-ups. The Port 1 output buffers can sink/source four TTL inputs. When 1s are written to Port 1 pins, they are pulled high by the internal pull-ups and can be used as inputs. As inputs, Port 1 pins that are externally being pulled low will source current (I_{IL}) because of the internal pull-ups.

Block Diagram



Some Port 1 pins provide additional functions. P1.0 and P1.1 can be configured to be the timer/counter 2 external count input (P1.0/T2) and the timer/counter 2 trigger input (P1.1/T2EX), respectively.

Furthermore, P1.4, P1.5, P1.6, and P1.7 can be configured as the SPI slave port select, data input/output and shift clock input/output pins as shown in the following table.

Port Pin	Alternate Functions
P1.0	T2 (external count input to Timer/Counter 2), clock-out
P1.1	T2EX (Timer/Counter 2 capture/reload trigger and direction control)
P1.4	$\overline{SS}$ (Slave port select input)
P1.5	MOSI (Master data output, slave data input pin for SPI channel)
P1.6	MISO (Master data input, slave data output pin for SPI channel)
P1.7	SCK (Master clock output, slave clock input pin for SPI channel)

Port 1 also receives the low-order address bytes during Flash programming and verification.

Port 2

Port 2 is an 8-bit bi-directional I/O port with internal pull-ups. The Port 2 output buffers can sink/source four TTL inputs. When 1s are written to Port 2 pins, they are pulled high by the internal pull-ups and can be used as inputs. As inputs, Port 2 pins that are externally being pulled low will source current (I_{IL}) because of the internal pull-ups.

Port 2 emits the high-order address byte during fetches from external program memory and during accesses to external data memory that use 16-bit addresses (MOVX @ DPTR). In this application, Port 2 uses strong internal pull-ups when emitting 1s. During accesses to external data memory that use 8-bit addresses (MOVX @ RI), Port 2 emits the contents of the P2 Special Function Register.

Port 2 also receives the high-order address bits and some control signals during Flash programming and verification.

Port 3

Port 3 is an 8-bit bi-directional I/O port with internal pull-ups. The Port 3 output buffers can sink/source four TTL inputs. When 1s are written to Port 3 pins, they are pulled high by the internal pull-ups and can be used as inputs. As inputs, Port 3 pins that are externally being pulled low will source current (I_{IL}) because of the pull-ups.

Port 3 receives some control signals for Flash programming and verification.

Port 3 also serves the functions of various special features of the AT89S8252, as shown in the following table.

Port Pin	Alternate Functions
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	$\overline{INT0}$ (external interrupt 0)
P3.3	$\overline{INT1}$ (external interrupt 1)
P3.4	T0 (timer 0 external input)
P3.5	T1 (timer 1 external input)
P3.6	$\overline{WR}$ (external data memory write strobe)
P3.7	$\overline{RD}$ (external data memory read strobe)

RST

Reset input. A high on this pin for two machine cycles while the oscillator is running resets the device.

ALE/$\overline{PROG}$

Address Latch Enable is an output pulse for latching the low byte of the address during accesses to external memory. This pin is also the program pulse input ($\overline{PROG}$) during Flash programming.

In normal operation, ALE is emitted at a constant rate of 1/6 the oscillator frequency and may be used for external timing or clocking purposes. Note, however, that one ALE pulse is skipped during each access to external data memory.

If desired, ALE operation can be disabled by setting bit 0 of SFR location 8EH. With the bit set, ALE is active only during a MOVX or MOVC instruction. Otherwise, the pin is weakly pulled high. Setting the ALE-disable bit has no effect if the microcontroller is in external execution mode.

$\overline{PSEN}$

Program Store Enable is the read strobe to external program memory.

When the AT89S8252 is executing code from external program memory, $\overline{PSEN}$ is activated twice each machine cycle, except that two $\overline{PSEN}$ activations are skipped during each access to external data memory.

$\overline{EA}$/VPP

External Access Enable. $\overline{EA}$ must be strapped to GND in order to enable the device to fetch code from external program memory locations starting at 0000H up to FFFFH. Note, however, that if lock bit 1 is programmed, $\overline{EA}$ will be internally latched on reset.

$\overline{EA}$ should be strapped to V_{CC} for internal program executions. This pin also receives the 12-volt programming enable voltage (V_{PP}) during Flash programming when 12-volt programming is selected.

XTAL1

Input to the inverting oscillator amplifier and input to the internal clock operating circuit.

XTAL2

Output from the inverting oscillator amplifier.

MAX232 Comunication Interface

19-4323; Rev 10; 8/01

MAXIM

+5V-Powered, Multichannel RS-232 Drivers/Receivers

General Description

The MAX220–MAX249 family of line drivers/receivers is intended for all EIA/TIA-232E and V.28/V.24 communications interfaces, particularly applications where ±12V is not available.

These parts are especially useful in battery-powered systems, since their low-power shutdown mode reduces power dissipation to less than 5µW. The MAX225, MAX233, MAX235, and MAX245/MAX246/MAX247 use no external components and are recommended for applications where printed circuit board space is critical.

Applications

Portable Computers
Low-Power Modems
Interface Translation
Battery-Powered RS-232 Systems
Multidrop RS-232 Networks

Features

Superior to Bipolar

- Operate from Single +5V Power Supply (+5V and +12V—MAX231/MAX239)
- Low-Power Receive Mode in Shutdown (MAX223/MAX242)
- Meet All EIA/TIA-232E and V.28 Specifications
- Multiple Drivers and Receivers
- 3-State Driver and Receiver Outputs
- Open-Line Detection (MAX243)

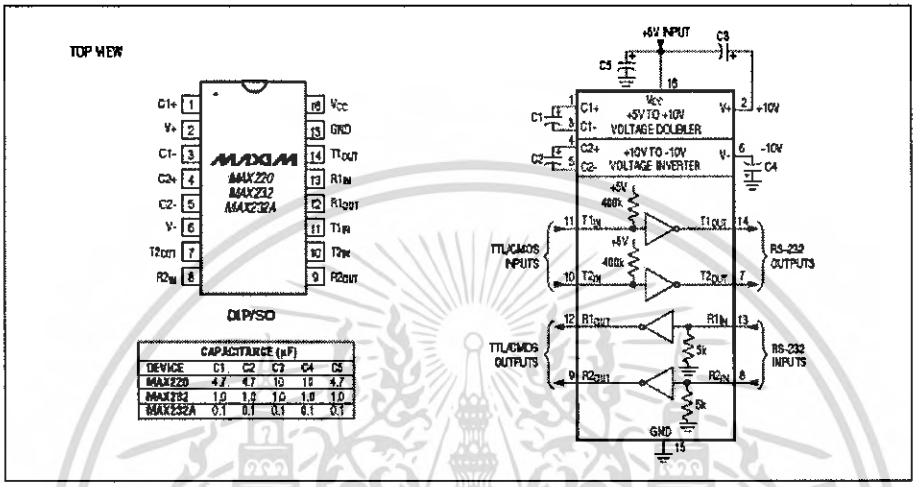
Ordering Information

PART	TEMP. RANGE	PIN-PACKAGE
MAX220CPE	0°C to +70°C	16 Plastic DIP
MAX220CSE	0°C to +70°C	16 Narrow SO
MAX220CWE	0°C to +70°C	16 Wide SO
MAX220C/D	0°C to +70°C	Dice*
MAX220EPE	-40°C to +85°C	16 Plastic DIP
MAX220ESE	-40°C to +85°C	16 Narrow SO
MAX220EWE	-40°C to +85°C	16 Wide SO
MAX220EJE	-40°C to +85°C	16 CERDIP
MAX220MJE	-55°C to +125°C	16 CERDIP

Ordering information continued at end of data sheet.
**Contact factory for dice specifications.*

Selection Table

Part Number	Power Supply (V)	No. of RS-232 Drivers/Rx	No. of Ext. Caps	Nominal Cap. Value (µF)	SHDN & Three-State	Rx Active in SHDN	Data Rate (kbps)	Features
MAX220	+5	2/2	4	0.1	No	—	120	Ultra-low-power, industry-standard pinout
MAX222	+5	2/2	4	0.1	Yes	—	200	Low-power shutdown
MAX223 (MAX213)	+5	4/5	4	1.0 (0.1)	Yes	✔	120	MAX241 and receivers active in shutdown
MAX225	+5	5/5	0	—	Yes	✔	120	Available in SO
MAX230 (MAX200)	+5	5/0	4	1.0 (0.1)	Yes	—	120	5 drivers with shutdown
MAX231 (MAX201)	+5 and +7.5 to +13.2	2/2	2	1.0 (0.1)	No	—	120	Standard +5/+12V or battery supplies; same functions as MAX232
MAX232 (MAX202)	+5	2/2	4	1.0 (0.1)	No	—	120 (64)	Industry standard
MAX232A	+5	2/2	4	0.1	No	—	200	Higher slew rate, small caps
MAX233 (MAX203)	+5	2/2	0	—	No	—	120	No external caps
MAX233A	+5	2/2	0	—	No	—	200	No external caps, high slew rate
MAX234 (MAX204)	+5	4/0	4	1.0 (0.1)	No	—	120	Replaces 1489
MAX235 (MAX205)	+5	5/5	0	—	Yes	—	120	No external caps
MAX236 (MAX206)	+5	4/3	4	1.0 (0.1)	Yes	—	120	Shutdown, three state
MAX237 (MAX207)	+5	5/3	4	1.0 (0.1)	No	—	120	Complements IBM PC serial port
MAX238 (MAX208)	+5	4/4	4	1.0 (0.1)	No	—	120	Replaces 1488 and 1489
MAX239 (MAX209)	+5 and +7.5 to +13.2	3/5	2	1.0 (0.1)	No	—	120	Standard +5/+12V or battery supplies; single-package solution for IBM PC serial port
MAX240	+5	5/5	4	1.0	Yes	—	120	DIP or flatpack package
MAX241 (MAX211)	+5	4/5	4	1.0 (0.1)	Yes	—	120	Complete IBM PC serial port
MAX242	+5	2/2	4	0.1	Yes	✔	200	Separate shutdown and enable
MAX243	+5	2/2	4	0.1	No	—	200	Open-line detection simplifies cabling
MAX244	+5	8/10	4	1.0	No	—	120	High slew rate
MAX245	+5	8/10	0	—	Yes	✔	120	High slew rate, int. caps, two shutdown modes
MAX246	+5	9/10	0	—	Yes	✔	120	High slew rate, int. caps, three shutdown modes
MAX247	+5	8/9	0	—	Yes	✔	120	High slew rate, int. caps, nine operating modes
MAX248	+5	8/8	4	1.0	Yes	✔	120	High slew rate, selective half-chip enables
MAX249	+5	6/10	4	1.0	Yes	✔	120	Available in quad flatpack package



CAPACITANCE (µF)					
DEVICE	C1	C2	C3	C4	C5
MAX220	4.7	4.7	10	10	4.7
MAX232	1.0	1.0	1.0	1.0	1.0
MAX232A	0.1	0.1	0.1	0.1	0.1

Figure 5. MAX220/MAX232/MAX232A Pin Configuration and Typical Operating Circuit

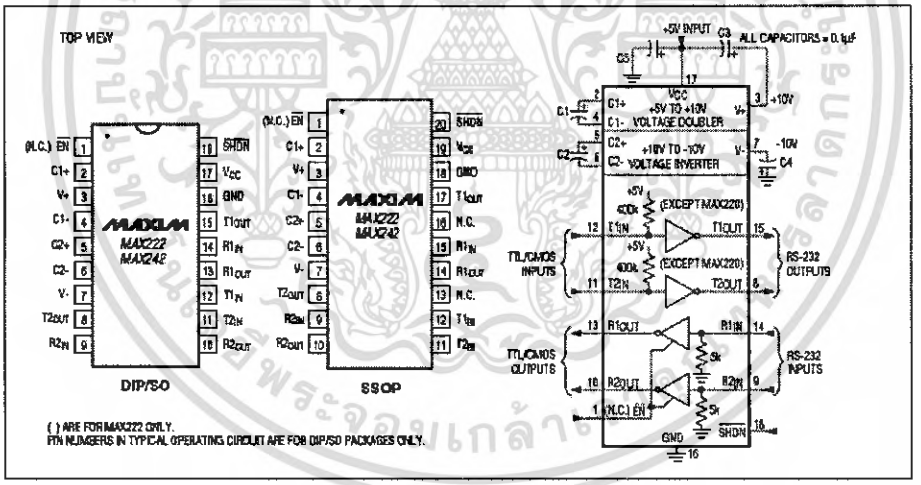


Figure 6. MAX222/MAX242 Pin Configurations and Typical Operating Circuit

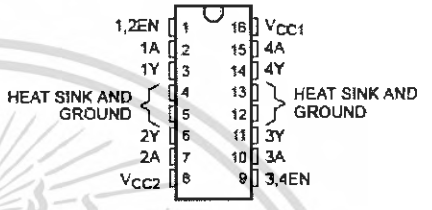
L293D Motor Driver

L293, L293D
QUADRUPLE HALF-H DRIVERS

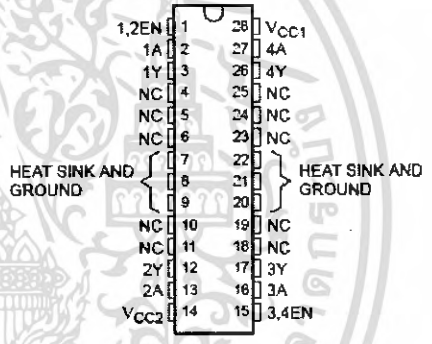
SLRS009C – SEPTEMBER 1986 – REVISED NOVEMBER 2004

- Featuring Unitrode L293 and L293D Products Now From Texas Instruments
- Wide Supply-Voltage Range: 4.5 V to 36 V
- Separate Input-Logic Supply
- Internal ESD Protection
- Thermal Shutdown
- High-Noise-Immunity Inputs
- Functionally Similar to SGS L293 and SGS L293D
- Output Current 1 A Per Channel (600 mA for L293D)
- Peak Output Current 2 A Per Channel (1.2 A for L293D)
- Output Clamp Diodes for Inductive Transient Suppression (L293D)

L293 . . . N OR NE PACKAGE
L293D . . . NE PACKAGE
(TOP VIEW)



L293 . . . DWP PACKAGE
(TOP VIEW)



description/ordering information

The L293 and L293D are quadruple high-current half-H drivers. The L293 is designed to provide bidirectional drive currents of up to 1 A at voltages from 4.5 V to 36 V. The L293D is designed to provide bidirectional drive currents of up to 600-mA at voltages from 4.5 V to 36 V. Both devices are designed to drive inductive loads such as relays, solenoids, dc and bipolar stepping motors, as well as other high-current/high-voltage loads in positive-supply applications.

All inputs are TTL compatible. Each output is a complete totem-pole drive circuit, with a Darlington transistor sink and a pseudo-Darlington source. Drivers are enabled in pairs, with drivers 1 and 2 enabled by 1,2EN and drivers 3 and 4 enabled by 3,4EN. When an enable input is high, the associated drivers are enabled, and their outputs are active and in phase with their inputs. When the enable input is low, those drivers are disabled, and their outputs are off and in the high-impedance state. With the proper data inputs, each pair of drivers forms a full-H (or bridge) reversible drive suitable for solenoid or motor applications.

ORDERING INFORMATION

T_A	PACKAGE†		ORDERABLE PART NUMBER	TOP-SIDE MARKING
0°C to 70°C	HSOP (DWP)	Tube of 20	L293DWP	L293DWP
	PDIP (N)	Tube of 25	L293N	L293N
	PDIP (NE)	Tube of 25	L293NE	L293NE
		Tube of 25	L293DNE	L293DNE

† Package drawings, standard packing quantities, thermal data, symbolization, and PCB design guidelines are available at www.ti.com/sc/package.

Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and disclaimers thereto appears at the end of this data sheet.

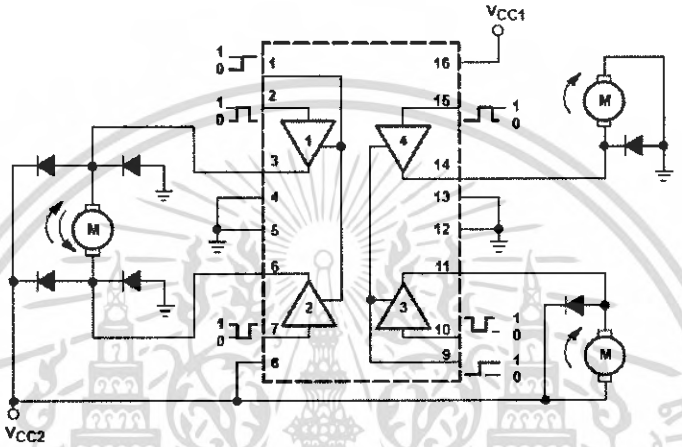
description/ordering information (continued)

On the L293, external high-speed output clamp diodes should be used for inductive transient suppression.

A V_{CC1} terminal, separate from V_{CC2}, is provided for the logic inputs to minimize device power dissipation.

The L293and L293D are characterized for operation from 0°C to 70°C.

block diagram



NOTE: Output diodes are internal in L293D.

FUNCTION TABLE
(each driver)

INPUTS†		OUTPUT
A	EN	Y
H	H	H
L	H	L
X	L	Z

H = high level, L = low level, X = irrelevant,
Z = high impedance (off)
† In the thermal shutdown mode, the output is in the high-impedance state, regardless of the input levels.

74HC14 Schmitt Trigger

PIN DESCRIPTION

PIN NO.	SYMBOL	NAME AND FUNCTION
1, 3, 5, 9, 11, 13	1A to 6A	data inputs
2, 4, 6, 8, 10, 12	1Y to 6Y	data outputs
7	GND	ground (0 V)
14	V_{CC}	positive supply voltage

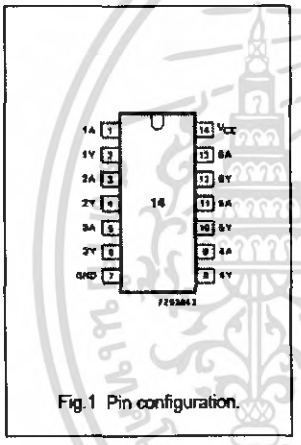


Fig.1 Pin configuration.

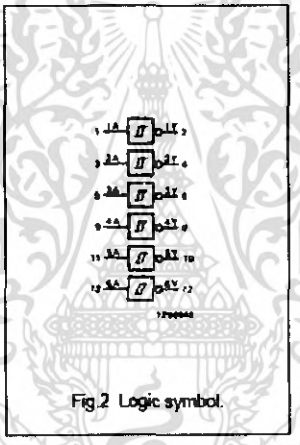


Fig.2 Logic symbol.

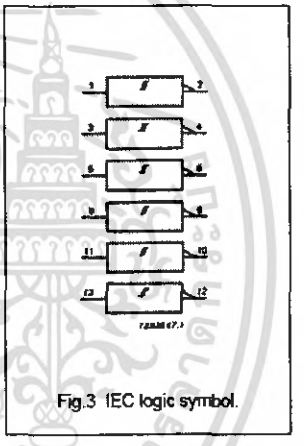


Fig.3 IEC logic symbol.

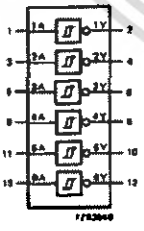


Fig.4 Functional diagram.

Fig.5 Logic diagram (one Schmitt trigger).

FUNCTION TABLE

INPUT	OUTPUT
nA	nY
L	H
H	L

Notes

1. *H = HIGH voltage level*
   L = LOW voltage level

APPLICATIONS

- Wave and pulse shapers
- Astable multivibrators
- Monostable multivibrators

Serial Port, RS-232