ระบบตรวจจับตัวอักษรจากภาพถ่าย

Text Detection from Images

โดย

นายพูลศักดิ์ จีรบุณย์

รหัส 45066073

อาจารย์ที่ปรึกษา

ดร.ธนารัตน์ ชลิดาพงศ์

รายงานนี้เป็นส่วนหนึ่งของวิชาโครงการพัฒนาระบบงาน
หลักสูตรวิทยาศาสตรมหาบัณฑิต สาขาวิชาเทคโนโลยีสารสนเทศ
ภาคเรียนที่ 2 ปีการศึกษา 2546
คณะเทคโนโลยีสารสนเทศ
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ชื่อหัวข้อ	ระบบตรวจจับตัวอักษรจากภาพถ่าย
นักศึกษา	นายพูลศักดิ์ จีรบุณย์
อาจารย์ที่ปรึกษา	ดร. ธนารัตน์ ชลิดาพงศ์
ระดับการศึกษา	วิทยาศาสตรมหาบัณฑิต สาขาวิชาเทคโนโลยีสารสนเทศ
แขนงวิชา	วิทยาการสารสนเทศ
ปีการศึกษา	2546

บทคัดย่อ

การตรวจจับตัวอักษรจากภาพถ่ายตามธรรมชาตินั้นมีประโยชน์ในงานหลายๆ ด้าน ดังนั้นจึงได้ทำการพัฒนาระบบตรวจจับตัวอักษรจากภาพถ่ายตามธรรมชาติ โดยที่ภาพจะถูกปรับปรุงภาพให้ดีขึ้นด้วยการใช้อัลกอริทึม Gaussian Filtering ซึ่งใช้ลดสัญญาณรบกวนให้ลดน้อยลง หลังจากนั้นจะถูกนำไปแปลงเป็นภาพ 2 ระดับ (Binary Image) โดยกระบวนการ Thresholding เสร็จแล้วจะถูกทำการตรวจจับเส้นขอบโดยใช้อัลกอริทึมในการหาเส้นขอบแล้วสุดท้ายจะเป็นการใช้เทคนิค Connected Component Analysis ร่วมกับกฎข้อบังคับทางด้านความสัมพันธ์โครงร่างเพื่อที่จะตรวจจับบริเวณที่เป็นตัวอักษรในภาพ

Title	Text Detection from Images
Student	Mr. Poonsak Jeeraboon
Advisor	Dr. Thanarat Chalidabhongse
Level of Study	Master of Science in Information Technology
Major	Information Science
Academic Year	2003

ABSTRACT

Text detection from images is useful in many applications. In this project, we develop a system for detecting texts embedded in the natural scenes. The image, which is captured by a digital camera, is initially passed through the Gaussian Filtering algorithm for smoothing the image as well as reducing noise. Thresholding technique is used for binarizing the image. Afterward, the edge detection algorithm is applied on the filtered image to find edges. Finally, the connected component technique and the layout relation constraints are performed on the image in order to detect the text region in the image. Results are also presented in this report.

กิตติกรรมประกาศ

โครงการพัฒนาระบบงานนี้ประสบความสำเร็จลุล่วงไปด้วยความร่วมมือและความช่วยเหลือจากบุคคลหลายท่าน โดยเฉพาะอย่างยิ่งนั้น ผู้เขียนขอขอบคุณ ดร.ธนารัตน์ ชลิดาพงศ์ ซึ่งเป็นผู้ให้ความช่วยเหลือคำแนะนำที่เป็นประโยชน์ในทุกๆ ด้านในโครงการพัฒนาระบบงานนี้

ผู้เขียนขอขอบคุณ คุณพ่อคุณแม่ที่เป็นกำลังใจให้เสมอมา คอยถามไถ่ความก้าวหน้าของโปรเจ็คอยู่ตลอด ผู้เขียนขอขอบคุณ คุณพนมพร สาคร (พี่หนู) และคุณอลงกต น้อยใจบุญ (กต) ที่คอยให้คำปรึกษาและข้อแนะนำเกี่ยวกับการใช้งานโปรแกรม Microsoft Visual C++ และ Microsoft Vision SDK รวมทั้งเพื่อนๆ ทุกคนที่คอยให้กำลังใจจนผู้เขียนสามารถพัฒนาระบบงานได้สำเร็จลุล่วงไปด้วยดี

ผู้เขียนหวังเป็นอย่างยิ่งว่าโครงการพัฒนาระบบงานนี้จะเป็นประโยชน์ต่อนิสิต นักศึกษา รวมทั้งผู้ค้นคว้าที่สนใจในเรื่องนี้ โดยถ้าหากว่าโครงการพัฒนาระบบงานนี้มีข้อผิดพลาดประการใดก็ตาม ผู้เขียนขอน้อมรับไว้เพื่อนำไปปรับปรุงให้สมบูรณ์ยิ่งขึ้น แต่ถ้าหากความดีที่ได้รับจากโครงการพัฒนาระบบงานนี้ ผู้เขียนขอมอบให้แก่บุพการี ครูบาอาจารย์ และผู้มีพระคุณทุกท่าน

พูลศักดิ์ จีรบุณย์

กุมภาพันธ์ 2547

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

หน้า

ตารางที่

สารบัญภาพ

หน้า

ภาพที่

สารบัญภาพ (ต่อ)

หน้า

ภาพที่

บทที่ 1
บทนำ

1.1 ความเป็นมาของปัญหา

ในปัจจุบันนี้คอมพิวเตอร์ได้เข้ามามีบทบาทกับการดำรงชีวิตของมนุษย์ในทุกๆ ด้านเป็นอย่างมาก ดังนั้นจึงได้มีการนำเอาระบบคอมพิวเตอร์เข้ามาประยุกต์ใช้ในงานต่างๆ กันอย่างมากขึ้น แต่อย่างไรก็ตามนั้น คอมพิวเตอร์นั้นก็เป็นเพียงเครื่องมือที่มนุษย์นั้นนำมาใช้งานตามวัตถุประสงค์ที่มนุษย์ต้องการจะให้เป็นเท่านั้น ซึ่งก็คือมนุษย์เป็นคนคิดค้นคอมพิวเตอร์ แล้วให้คอมพิวเตอร์ทำงานเพียงแค่ตามที่มนุษย์ควบคุมและสั่งการเท่านั้น

เทคโนโลยีของระบบคอมพิวเตอร์นั้นมีวิวัฒนาการก้าวหน้าไปอย่างรวดเร็ว จากเดิมที่มนุษย์ใช้คอมพิวเตอร์เป็นเพียงเครื่องมือให้ปฏิบัติภารกิจตามที่มนุษย์ควบคุมเท่านั้น แต่ทว่าเนื่องด้วยความก้าวหน้าทางด้านเทคโนโลยีคอมพิวเตอร์นั้น มีความพยายามที่จะทำให้คอมพิวเตอร์นั้นสามารถที่จะคิดและวินิจฉัยด้วยตนเอง รวมทั้งมีความฉลาดมีความคิดให้เหมือนหรือใกล้เคียงกับมนุษย์มากที่สุดเท่าที่จะเป็นไปได้ ซึ่งในปัจจุบันนี้มีการศึกษาค้นคว้ารวมทั้งวิจัยทางด้านปัญญาประดิษฐ์ (Artificial Intelligence) มากขึ้นตามลำดับ

ถ้าเราลองมองสิ่งต่างๆ รอบตัวเรา ไม่ว่าจะเป็นตามอาคารพาณิชย์, ป้ายข้อความแจ้งเตือน, ป้ายโฆษณาต่างๆ รวมถึงป้ายบอกทางตามท้องถนนที่เราใช้สัญจรไปมา เราจะพบว่าสิ่งหนึ่งที่เป็นสิ่งสำคัญที่ใช้ในการสื่อสารกับมนุษย์เพื่อให้เกิดความเข้าใจที่ตรงกัน นั่นก็คือตัวอักษร ซึ่งเมื่อเราได้อ่านตัวอักษรที่ปรากฏอยู่นั้น ก็จะทำให้เราเข้าใจถึงสิ่งที่ต้องการจะสื่อให้ได้รับรู้ ซึ่งมนุษย์เรานั้นมีความสามารถที่จะใช้ในการแยกแยะว่าสิ่งใดคือสิ่งที่เป็นตัวอักษร สิ่งใดไม่ใช่ตัวอักษร เหตุที่เป็นเช่นนี้ก็เพราะว่ามนุษย์เรามีความสามารถที่จะแยกแยะสิ่งเหล่านี้ได้ แต่มันไม่ใช่สำหรับระบบคอมพิวเตอร์ เราจึงเกิดความคิดที่ว่า จะทำอย่างไรที่จะทำให้เครื่องคอมพิวเตอร์มีความสามารถในการแยกแยะได้เหมือนหรือใกล้เคียงกับมนุษย์มากที่สุด ซึ่งสิ่งนี้เองเป็นที่มาของโครงการพัฒนาระบบตรวจจับตัวอักษรจากภาพถ่ายตามธรรมชาติ (Text Detection System from Natural Scenes)

การตรวจจับสกัดแยกตัวอักษรที่อยู่ในภาพถ่ายตามธรรมชาตินั้น สามารถนำมาประยุกต์ทำให้เกิดสิ่งที่มีประโยชน์ได้อย่างมาก ตัวอย่างหนึ่งของระบบที่ใช้การตรวจจับตัวอักษรจากภาพถ่ายตามธรรมชาติก็คือ ระบบตรวจจับตัวอักษรจากภาพถ่ายป้ายต่างๆ เป็นต้น ซึ่งสามารถนำเอาไป

ประยุกต์และพัฒนาต่อเป็นระบบที่มีประโยชน์อย่างมากมาย เช่น ระบบรู้จำป้าย (Sign Recognition) เป็นต้น โดยถึงแม้ว่าในปัจจุบัน เรามีเทคโนโลยีที่สามารถที่จะตรวจจับตัวอักษรบนภาพที่ได้จากการสแกนจากหนังสือได้ค่อนข้างที่จะมีประสิทธิภาพที่ดี แต่ว่าการที่เราจะตรวจจับส่วนที่เป็นตัวอักษรจากภาพถ่ายตามธรรมชาตินั้นเป็นสิ่งที่ยังคงต้องศึกษาและวิจัยต่อไปเนื่องจากว่าตัวอักษรในภาพนั้นอาจจะมีลักษณะที่เบนเอียงออกไป อีกทั้งยังคงมีความแตกต่างของลักษณะของแสงที่ตกกระทบ รวมถึงอาจมีสิ่งรอบข้างของจุดที่เราสนใจไม่ว่าจะเป็นวัตถุต่างๆ ที่ไม่ใช่ตัวอักษรปรากฏอยู่ในภาพ ซึ่งเป็นสิ่งที่ทำให้การที่จะตรวจจับส่วนที่เป็นตัวอักษร ทำได้ยากมากยิ่งขึ้นอีกด้วย

1.2 วัตถุประสงค์ของระบบ

พัฒนาระบบที่ใช้ในการตรวจจับตัวอักษรที่ปรากฏอยู่ในภาพถ่ายตามธรรมชาติ โดยนำภาพถ่ายตามธรรมชาติที่ได้จากกล้องถ่ายรูปดิจิตอล (Digital Camera) มาประมวลผลตรวจจับหาบริเวณที่เป็นตัวอักษร แล้วทำการสร้างกรอบล้อมรอบบริเวณที่เป็นตัวอักษรในภาพถ่ายตามธรรมชาติเพื่อแสดงถึงบริเวณที่เป็นตัวอักษร

1.3 ขอบเขตของระบบ

- พัฒนาสำหรับเครื่องไมโครคอมพิวเตอร์บนระบบปฏิบัติการ Windows
- ข้อมูลนำเข้า (Input) ที่ใช้ทดสอบระบบเป็นภาพถ่ายจากกล้องถ่ายรูปดิจิตอล (Digital Camera) รวมถึงกล้องถ่ายรูปดิจิตอลที่ติดตั้งอยู่ในโทรศัพท์เคลื่อนที่ (Mobile Phone) บางรุ่น

1.4 ขั้นตอนการพัฒนาระบบ

- ศึกษาทฤษฎีรวมถึงแนวทางที่ใช้ในการพัฒนาระบบตรวจจับตัวอักษรจากภาพถ่ายตามธรรมชาติ
- ออกแบบระบบตรวจจับตัวอักษรจากภาพถ่ายตามธรรมชาติ
- พัฒนาโปรแกรมระบบตรวจจับตัวอักษรจากภาพถ่ายตามธรรมชาติ
- ทดสอบและแก้ไขโปรแกรมระบบตรวจจับตัวอักษรจากภาพถ่ายตามธรรมชาติ
- สรุปผลและข้อเสนอแนะเพิ่มเติม

1.5 ประโยชน์ที่คาดว่าจะได้รับ

- โปรแกรมประยุกต์ที่สามารถวิเคราะห์ได้ว่าส่วนใดเป็นบริเวณที่เป็นตัวอักษรในภาพถ่ายตามธรรมชาติ
- สามารถนำระบบไปพัฒนาต่อเป็นโปรแกรมประยุกต์ที่มีประโยชน์มากมาย อาทิเช่น ระบบรู้จำป้าย เป็นต้น

บทที่ 2
ทฤษฎีและแนวคิดที่สำคัญ

2.1 การตรวจจับตัวอักษร (Text Detection)

ในปัจจุบันนี้ เราสามารถที่จะจำแนกวิธีการที่ใช้ในการตรวจจับตัวอักษรออกเป็นหลายประเภท ดังนี้

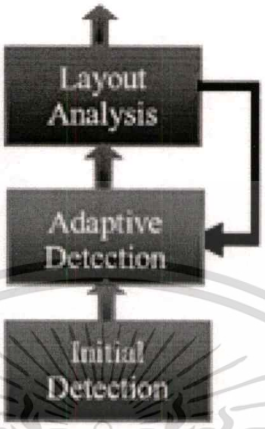
- วิธีฟิลเตอร์เส้นขอบ (Edge Filtering)
- วิธีแบ่งแยกพื้นผิว (Texture Segmentation)
- วิธีควอนไตซ์สี (Color Quantization)
- วิธีโครงข่ายประสาทเทียมและบูตสแตรป (Neural Networks and Bootstrapping)

โดยที่แต่ละวิธีนั้นก็มีข้อดีข้อเสียที่แตกต่างกันออกไป ซึ่งเกี่ยวข้องกับความเชื่อถือได้ (Reliability), ความแม่นยำเที่ยงตรง (Accuracy) รวมถึงความยากง่ายในการพัฒนาระบบขึ้นมาเพื่อใช้งาน

2.1.1 วิธี Adaptive Multi-resolution Algorithm (Gao and Yang. 2001)

วิธีนี้เป็นวิธีหนึ่งที่อาศัยหลักการของการฟิลเตอร์เส้นขอบ (Edge Filtering) มาใช้ในการตรวจจับหาส่วนที่เป็นตัวอักษร โดยที่การตรวจจับตัวอักษรด้วยวิธีการนี้จะมีลักษณะที่เป็นโครงสร้างอัลกอริทึมแบบเป็นลำดับชั้น (Hierarchical Algorithm Structure) โดยที่มีการเน้นในแต่ละลำดับชั้นที่ไม่เหมือนกัน โดยในลำดับชั้นแรกจะเป็นการตรวจจับบริเวณที่เป็นตัวอักษร (Text Cue) โดยการใช้การฟิลเตอร์เส้นขอบ (Edge Filtering) โดยบริเวณที่เป็นตัวอักษรนี้ ข้อมูลไม่ว่าจะเป็น ตำแหน่ง ขนาด และสี ข้อมูลเหล่านี้จะถูกใช้เป็นข้อมูลสำหรับอัลกอริทึมการแบ่งแยกสีแบบอะแด็ปทีฟ (Adaptive Color Segmentation Algorithm) ซึ่งเป็นกระบวนการทำงานในลำดับชั้นถัดมา ซึ่งจะเป็นการใช้อัลกอริทึม Gaussian Mixtures Color Modeling เพื่อสำหรับปรับสภาพที่แปรเปลี่ยนต่างๆ ในภาพ โดยที่ข้อมูลข้างต้น ไม่ว่าจะเป็น ตำแหน่ง ขนาด และสีนั้น จะช่วยลดความซับซ้อนในด้านการคำนวณสำหรับการทำ Color Segmentation ลงได้อย่างมากทีเดียว เนื่องจากทำให้เราคำนวณเพียงแค่บางส่วนของภาพโดยไม่ต้องทำการคำนวณภาพทั้งภาพนั่นเอง

การตรวจจับตัวอักษรด้วยวิธีนี้เป็นการใช้โครงสร้างอัลกอริทึมแบบเป็นลำดับชั้น ดังแสดงในภาพที่ 2.1 โดยที่มีการเน้นในแต่ละลำดับชั้นที่แตกต่างกันออกไป



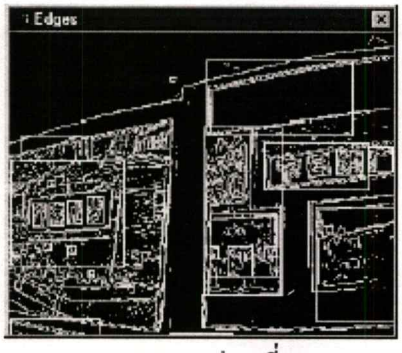
ภาพที่ 2.1 แสดงอัลกอริทึมในการตรวจจับตัวอักษร

อัลกอริทึมที่ใช้ในการตรวจจับตัวอักษรประกอบไปด้วย 3 ขั้นตอน คือ

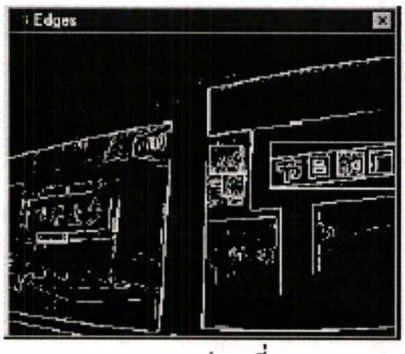
- อัลกอริทึมตรวจจับเส้นขอบแบบหลายมาตราส่วน (Multi-scale Edge Detection Algorithm)
- การค้นหาแบบอะแด็ปทีฟ (Adaptive Searching) และการทำ Color Modeling กับบริเวณใกล้ๆ กับส่วนที่เป็นตัวอักษร
- ทำการวิเคราะห์โครงร่าง (Layout Analysis) ของบริเวณตัวอักษรที่ตรวจจับได้

ในลำดับชั้นแรกของวิธีนี้จะทำการตรวจจับบริเวณที่เป็นตัวอักษร (Text Cues) โดยใช้เทคนิคฟิลเตอร์เส้นขอบ (Edge Filtering) ซึ่งจะใช้การตรวจจับหาเส้นขอบแบบหลายมาตราส่วน (Multi-scale Edge Detector) เพื่อเป็นการชดเชยค่าความแปรปรวนต่างๆ เช่น สัญญาณรบกวน (Noise) และคอนแทรสต์ (Contrast) ดังแสดงภาพตัวอย่างในภาพที่ 2.2

โดยจากภาพที่ 2.2 นั้น จะเห็นว่าป้ายทั้ง 2 ป้ายคือป้ายทางด้านซ้ายและป้ายทางด้านขวานั้นมีสภาพของแสงและคอนแทรสต์ที่แตกต่างกันออกไป ซึ่งสามารถตรวจจับได้ด้วย Edge Detector ในมาตราส่วน (Scale) ที่แตกต่างกันออกไป ซึ่งผลลัพธ์เมื่อทำการรวมทั้ง 2 มาตราส่วนก็จะทำให้สามารถตรวจจับป้ายทั้งด้านซ้ายและด้านขวาได้



มาตราส่วนที่ 1



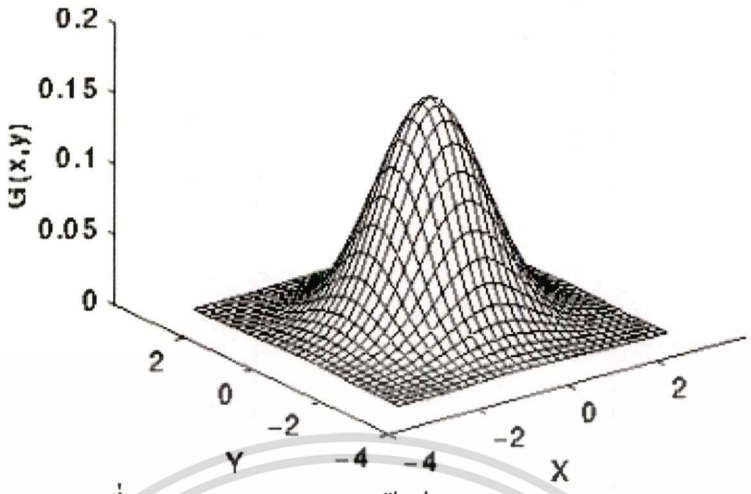
มาตราส่วนที่ 2

ผลลัพธ์เมื่อรวม 2 มาตราส่วนเข้าด้วยกัน

ภาพที่ 2.2 แสดงการตรวจจับเส้นขอบโดยใช้ Multi-scale Edge Detector

กระบวนการฟิลเตอร์เส้นขอบ (Edge Filtering) เริ่มต้นด้วยการปรับปรุงภาพด้วยการใช้ Gaussian Filter ก่อน แล้วจึงค่อยทำการหาเส้นขอบโดยใช้ 1st Derivative Edge Detection หลังจากนั้นจะใช้ Edge Clustering ร่วมกับข้อกำหนดต่างๆ เช่น อัตราส่วนของด้าน (Aspect Ratio), คู่ของขอบขึ้นและขอบลง เป็นต้น เพื่อใช้ในการหาบริเวณที่เป็นตัวอักษร (Text Cue) โดยผลลัพธ์ที่ได้จากเลเยอร์นี้จะเป็นตำแหน่ง, ขนาด และสีของบริเวณที่เป็นตัวอักษร ซึ่งผลลัพธ์นี้จะช่วยลดเวลาในการประมวลผลในเลเยอร์ที่ 2 ลงได้มาก โดยเราสามารถควบคุมมาตราส่วน (Scale) ของ Edge Detector ได้โดยทำการควบคุมความกว้างของ Gaussian Function ดังแสดงในภาพที่ 2.3 และค่า Threshold สำหรับการตรวจจับเส้นขอบ

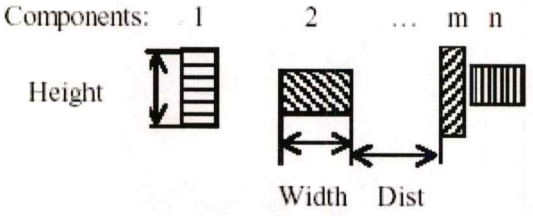
ขั้นตอนถัดมาจะเป็นการแยกแยะระหว่างบริเวณที่เป็นตัวอักษรกับบริเวณที่ไม่ใช่ตัวอักษรในบริเวณที่ตรวจจับมาได้จากเลเยอร์แรก โดยจะเป็นการใช้ Adaptive Searching เพื่อกำหนดบริเวณที่ตรวจจับ (Search Region) โดยการใช้ข้อมูลเบื้องต้นที่ได้จากกระบวนการทำงานในเลเยอร์แรก นอกจากนี้เราจะใช้รูปร่าง (Shape) และสี เป็นข้อกำหนดขอบเขตที่จะใช้แยกความแตกต่างของข้อความบนป้ายที่แตกต่างกันโดยคำนึงจากข้อสังเกตที่ว่า ข้อความบนป้ายเดียวกันนั้นมีแนวโน้มที่จะมีลักษณะต่างๆ ที่คล้ายคลึงกัน ไม่ว่าจะเป็น แบบตัวอักษร (Font), สี และขนาด เป็นต้น



ภาพที่ 2.3 แสดงการกระจายเก๊าส์ (Gaussian Distribution)

ขั้นตอนถัดไปจะเป็นการสกัดแยกตัวอักษร (Text Extraction) ออกจากบริเวณที่ตรวจจับ โดยที่การสกัดแยกตัวอักษรนั้นสามารถที่จะเป็นได้ทั้งในกรณีที่ง่ายและยาก ในกรณีที่ง่ายนั้น สีหรือค่าความเข้ม (Intensity) ของตัวอักษรในป้ายนั้นมีลักษณะเหมือนหรือคล้ายคลึงกัน คือไม่มีการเปลี่ยนแปลงอย่างมากมาย ดังนั้นในกรณีนี้เราสามารถที่จะใช้ข้อมูลต่างๆ เช่น สีซึ่งเป็นข้อมูลที่ได้มาในขั้นต้นแล้ว เป็นต้น สำหรับใช้ในการสกัดแยกตัวอักษรได้ แต่ในกรณีที่มีการเปลี่ยนแปลงอย่างมาก ไม่ว่าจะเป็นสีหรือค่าความเข้มบนข้อความเดียวกันนั้น ทำให้เราไม่สามารถที่จะใช้ข้อมูลเบื้องต้นดังกล่าวในการที่จะสกัดแยกตัวอักษรออกมาได้

ขั้นตอนสุดท้ายนั้นเป็นขั้นตอนของการวิเคราะห์ความสัมพันธ์ของโครงร่าง (Layout Relation Analysis) นั้นเป็นขั้นตอนที่จะเข้ามาช่วยแก้ปัญหาที่ว่า เราจะทราบได้อย่างไรว่าบริเวณข้อความนี้เป็นตัวอักษร หรือว่าเป็นเพียงแค่ส่วนย่อยของตัวอักษร โดยเราจะใช้เทคนิคการวิเคราะห์โครงร่าง ซึ่งจะเป็นการใช้ข้อกำหนดขอบเขตต่างๆ ในการจัดการกับปัญหาตรงจุดนี้ ซึ่งข้อกำหนดขอบเขตนั้นจะเป็นค่าคงที่ (Constant) โดยจะแทนด้วย *Cn* ซึ่งเป็นค่าคงที่ที่กำหนดขึ้นได้จากการทดลอง และแสดงสัญลักษณ์อื่นๆ ของความสัมพันธ์โครงร่าง (Layout Relation) ในภาพที่ 2.4 โดยมีตัวอย่างของข้อกำหนดขอบเขตต่างๆ ดังนี้



ภาพที่ 2.4 แสดงสัญลักษณ์ต่างๆ สำหรับ Layout Relation

- ตัวอักษรบนข้อความเดียวกันนั้น ควรจะอยู่ใกล้ๆ กับตัวอักษรอื่นๆ ที่อยู่ข้างเคียง

$$Dist(i, j) / max[Width(i), Width(j)] < C1$$

- อัตราส่วนของด้าน (Aspect Ratio) ของตัวอักษรแต่ละตัวนั้น จะมีขอบเขตที่แน่นอน

$$C2 < Aspect\ Ratio < C3$$

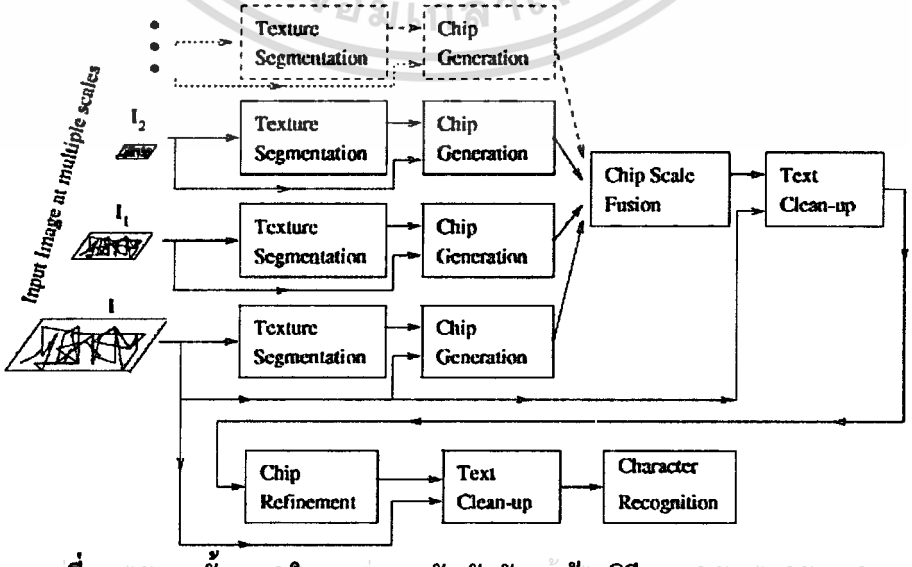
- หลังจากที่กำหนดให้แต่ละตัวอักษรเป็นแต่ละส่วน (Component) ได้แล้วนั้น ตัวอักษรแต่ละตัวนั้นควรที่จะจัดเรียงเป็นแนวเดียวกันกับตัวอักษรตัวอื่น

2.1.2 วิธี Texture Segmentation (Wu *et.al.* 1999)

การตรวจจับตัวอักษร (Text Detection) จากภาพโดยวิธีแบ่งแยกพื้นผิวนั้น อาศัยความที่ตัวอักษรนั้นมีคุณลักษณะเฉพาะของตัวมันเองเพื่อที่จะทำให้สามารถตรวจจับได้ เช่น การที่ตัวอักษรมีความถี่ที่แน่นอน และตัวอักษรจะมีลักษณะที่ปะติดปะต่อกัน ซึ่งก็คือตัวอักษรในประโยคเดียวกันมักจะมีความสูงใกล้เคียงกัน, ทิศทางและการเว้นวรรคที่ใกล้เคียงกัน

การตรวจจับตัวอักษรด้วยวิธีแบ่งแยกพื้นผิว มีขั้นตอนการทำงานหลักๆ อยู่ 5 ขั้นตอน (ภาพที่ 2.5) ดังนี้

- การแบ่งแยกพื้นผิว (Texture Segmentation)
- การตีกรอบตัวอักษร (Chip Generation)
- การรวมกรอบตัวอักษรจากหลายมาตราส่วน (Chip Scale Fusion)
- การฟื้นฟูตัวอักษร (Text Cleanup)
- การปรับปรุงกรอบตัวอักษร (Chip Refinement)



ภาพที่ 2.5 แสดงขั้นตอนในการตรวจจับตัวอักษรด้วยวิธี Texture Segmentation

วิธีการนี้มีข้อสมมติฐานที่ว่า คุณลักษณะตัวอักษรนั้นจะมีลักษณะของพื้นผิวที่แตกต่างเป็นเอกลักษณ์ โดยขั้นตอนของการแบ่งแยกพื้นผิวเริ่มต้นด้วยการปรับปรุงภาพด้วย Gaussian Filter แล้วหลังจากนั้นทำการตรวจหาเส้นขอบโดยใช้ 2nd Derivative Edge Detection เมื่อเสร็จแล้วก็นำมาผ่านการแปลงแบบไม่เป็นเชิงเส้น (Nonlinear Transformation) เพื่อที่จะสร้างเป็น Feature Vector ในแต่ละพิกเซล

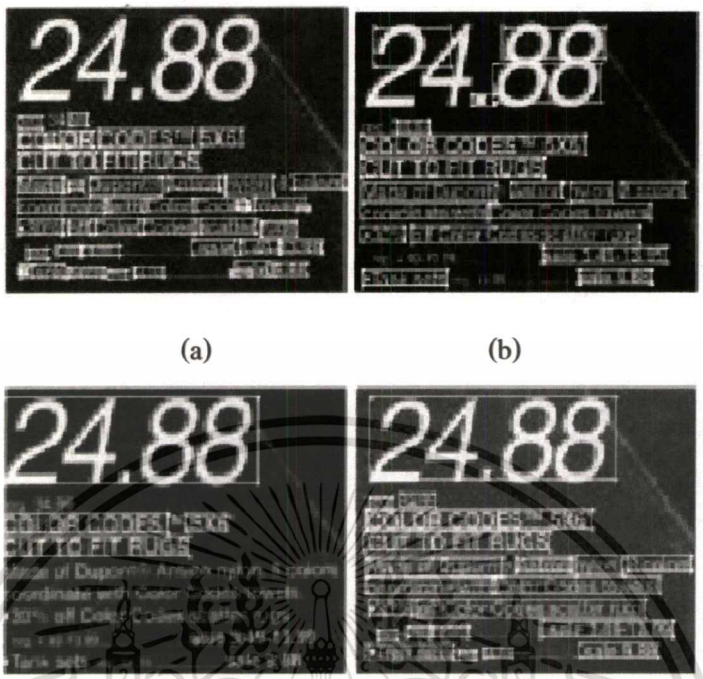
ขั้นตอนที่ 2 ของวิธีการนี้คือขั้นตอนการตีกรอบตัวอักษร (Chip Generation) โดยขั้นตอนนี้จะเป็นการหาข้อความตัวอักษรภายในหรือใกล้ๆ บริเวณที่ได้ทำการแบ่งแยกแล้วจากขั้นตอนแรก โดยในขั้นตอนที่ 2 นี้แบ่งการทำงานออกเป็น 5 ขั้นตอนย่อย คือ

- ***Stroke Generation***: เพื่อจัดกลุ่มของพิกเซลขอบ (Edge Pixels) โดยใช้หลักการ Connected Components
- ***Stroke Filtering***: เพื่อขจัด Stroke ที่ไม่น่าจะเกิดขึ้นของข้อความตัวอักษรในแนวนอน
- ***Stroke Aggregation***: เพื่อสร้างกรอบให้กับเส้นที่เชื่อมต่อกันที่มาจากข้อความเดียวกัน
- ***Chip Filtering***: เพื่อทำการขจัดกรอบตัวอักษรที่ไม่น่าจะเกิดขึ้น
- ***Chip Extension***: เพื่อที่จะทำให้กรอบตัวอักษรนั้นครอบคลุมส่วนที่เป็นตัวอักษรได้สมบูรณ์ยิ่งขึ้น

ขั้นตอนที่ 3 คือการรวมกรอบตัวอักษรจากหลายมาตราส่วน (Chip Scale Fusion) โดยขั้นตอนนี้นั้น ภาพที่จะถูกประมวลผล จะต้องถูกทำการแปลงภาพไปเป็นหลายๆ ความละเอียด (Resolution) เพื่อที่จะทำให้สามารถตรวจจับตัวอักษรได้ หากมีตัวอักษรหลายๆ ขนาดอยู่ในภาพเดียวกัน ดังแสดงในภาพที่ 2.6

ขั้นตอนที่ 4 เป็นการฟื้นฟูตัวอักษร (Text Cleanup) โดยใช้การทำภาพให้เป็น 2 ระดับ (Binarization) ซึ่งเป็นการใช้ Histogram-based Algorithm ในการหาค่า Threshold โดยอัตโนมัติ

ขั้นตอนสุดท้ายของวิธีการนี้คือการปรับปรุงกรอบตัวอักษร (Chip Refinement) ซึ่งจะเป็นขั้นตอนที่ใช้ในการกำจัดกรอบที่ไม่ใช่ตัวอักษรออกไป โดยการใช้อัลกอริทึมเดียวกันกับขั้นตอนการตีกรอบตัวอักษร (Chip Generation) แต่มีการเพิ่มกฎข้อบังคับ (Constraint) มากขึ้น



(a) (b)

(c) (d)

ภาพที่ 2.6 แสดงการรวมกรอบตัวอักษรจากหลายมาตราส่วน
(a) สร้างกรอบตัวอักษรจากภาพ Full Resolution
(b) สร้างกรอบตัวอักษรจากภาพ Half Resolution
(c) สร้างกรอบตัวอักษรจากภาพ 1/4 Resolution
(d) รวมกรอบตัวอักษรจากทั้ง 3 ระดับเข้าด้วยกัน

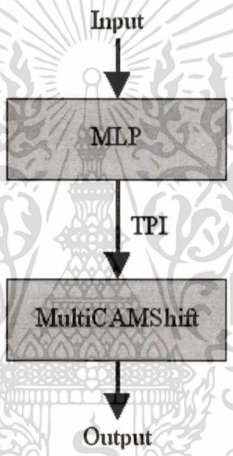
2.1.3 วิธี MultiCAMShift (Jung *et.al.* 2002)

วิธีนี้อาศัยวิธีโครงข่ายประสาทเทียมและบูตสแตรป (Neural Networks and Bootstrapping) โดยวิธีนี้นั้นเล็งเห็นว่าการตรวจจับตัวอักษรโดยใช้พื้นผิวในการแบ่งแยก (Texture-based) นั้นมีข้อเสียอยู่หลายประการ ได้แก่ ประการแรก คือมีความยากในการออกแบบตัวแยกพื้นผิว (Texture Classifier) สำหรับสภาพตัวอักษรที่แตกต่างกัน ประการถัดไป คือใช้เวลาในการประมวลผลค่อนข้างนานมาก และประการสุดท้าย คือวิธีการแบบ Texture-based มักจะต้องใช้กับรูปภาพที่มีความละเอียดสูง (High Resolution) นั่นเอง

วิธีการนี้ประกอบไปด้วยขั้นตอนหลักๆ 2 ขั้นตอน ดังภาพที่ 2.7 ซึ่งประกอบไปด้วย

- ใช้โครงข่ายประสาทเทียม (Neural Network) แบบ Multi-layer Perceptron (MLP)
- ใช้อัลกอริทึม Multiple Continuously Adaptive Mean Shift (MultiCAMShift)

การทำงานของการตรวจจับตัวอักษรเริ่มต้นการนำภาพเข้าสู่ MLP เพื่อใช้แยกแยะพื้นที่บริเวณที่เป็นตัวอักษรกับบริเวณที่ไม่ใช่ตัวอักษร โดยใช้เลเยอร์ซ่อน (Hidden Layer) ทั้งหมด 2 เลเยอร์ ที่ซึ่งเชื่อมต่อกันแบบทั่วถึงกันหมด (Fully Connected) ระหว่างเลเยอร์ที่ติดกัน โดยที่ผลลัพธ์ที่ได้จากขั้นตอนนี้เราจะเรียกว่า TPI (Text Probability Image) ซึ่งจะเป็นการแสดงความเป็นไปได้ว่าพิกเซลของภาพอินพุตที่สอดคล้องนั้นเป็นส่วนหนึ่งของตัวอักษรในภาพ ซึ่งค่าในแต่ละพิกเซลจะมีค่าอยู่ระหว่าง 0 กับ 1 โดยในขั้นตอนนี้ของ MLP นั้นจะเป็นการใช้เทคนิคบูตสแตรป (Bootstrap) ในการสอนระบบเพื่อให้สามารถแยกความแตกต่างระหว่างบริเวณที่เป็นตัวอักษรกับบริเวณที่ไม่ใช่ตัวอักษร



ภาพที่ 2.7 แสดงโครงสร้างการตรวจจับตัวอักษรด้วย MultiCAMShift

เพื่อเป็นการหลีกเลี่ยงการที่จะต้องประมวลผลทุกๆ พิกเซลในภาพนั้น MLP ถูกนำเข้ามาใช้เพื่อจัดการตรงจุดนี้เพื่อลดการประมวลผลให้น้อยลงในขั้นตอนของการตรวจจับตัวอักษร ซึ่งจากที่ได้กล่าวไปแล้วข้างต้นว่า ตรงจุดนี้เป็นการแก้ไขจุดอ่อนของวิธีการ Texture-based ซึ่งจะต้องใช้การประมวลผลภาพในทุกๆ พิกเซล ซึ่ง MLP นั้นจะให้ผลที่ดีมาก หากว่าในภาพนั้น มีบริเวณที่เป็นตัวอักษรไม่มากนัก

ผลลัพธ์ที่ได้จากขั้นตอน MLP นั้นก้อคือ TPI โดยที่ขั้นตอนถัดมาหลังจากที่ได้ TPI ออกมาแล้วก็คือ จะเป็นการใช้อัลกอริทึม MultiCAMShift กับ TPI นั้นๆ ทำให้สามารถช่วยลดเวลาในการประมวลผลลงได้อย่างมาก โดยที่สามารถแสดงอัลกอริทึม MultiCAMShift ได้ดังนี้

- กำหนดตำแหน่งตั้งต้น ($mean_x(i)_0$, $mean_y(i)_0$) และขนาด ($\lambda_x(i)_0$ * $\lambda_y(i)_0$) ของ Search Windows W_s
- **Do**

  สำหรับแต่ละ Search Window $W(i)$
- สร้าง TPI ภายใน $W(i)$ โดยใช้ MLP
- หาตำแหน่งและขนาดใหม่ของบริเวณที่เป็นตัวอักษรโดยใช้ Mean Shift Vector

  ทำการรวมกรอบ (Windows) ที่เหลื่อมล้ำกัน

  ทำการเพิ่มค่าการวนรอบ t

While ($\| mean_x(i)_t - mean_x(i)_{t+1} \| > \varepsilon_x$ **or** $\| mean_y(i)_t - mean_y(i)_{t+1} \| > \varepsilon_y$)
- ทำการฟิลเตอร์บริเวณที่เป็นตัวอักษรโดยใช้พื้นที่และ Aspect Ratio ของกรอบที่ล้อมบริเวณดังกล่าว

ในขั้นแรกนั้นจะใช้อัลกอริทึม MultiCAMShift ในการตรวจจับตัวอักษรไปบนแต่ละ Search Window โดยที่ไม่ได้คำนึงถึงว่าใน Search Window นั้นจะมีส่วนที่เป็นข้อความอยู่ด้วยหรือไม่ก็ตาม โดยที่ในแต่ละครั้งของการวนรอบที่ติดกัน (Consecutive Iterations) นั้นเราจะประมาณขนาด, ตำแหน่งและมุมเอียง ของบริเวณที่เป็นตัวอักษร แล้วทำการเปลี่ยนค่าพารามิเตอร์ของ Search Window โดยขึ้นอยู่กับค่าที่ประมาณไว้ แล้วหลังจากนั้นจะเป็นการทำ Window-merge Operation ซึ่งเป็นขั้นตอนที่ใช้ในการรวม Window โดยจะทำการกำจัดส่วนที่ซ้อนทับกัน (Overlapping Search Windows) ออกไป โดยถ้าหากว่าค่า Mean Shift มีค่ามากกว่าค่า Threshold ε_x หรือ ε_y ให้ทำการวนรอบการทำงานของอัลกอริทึมอีกครั้ง

สำหรับในแต่ละครั้งของการวนรอบการทำงานของอัลกอริทึม จะทำการรวม Window ที่ซ้อนทับกัน (Overlapping Window) เข้าด้วยกัน เพื่อที่จะช่วยลดเวลาในการประมวลผลอัลกอริทึม MultiCAMShift ให้น้อยลง

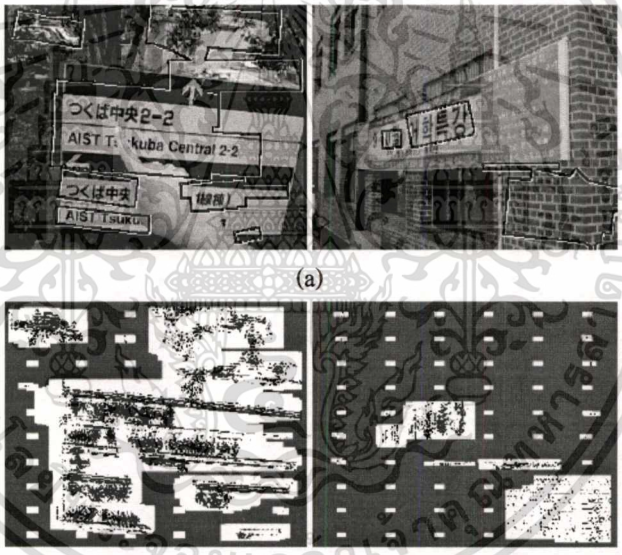
เมื่อมี Overlapping Window เกิดขึ้นนั้น เราจะทำการตรวจสอบ โดยไม่คำนึงถึงว่าเป็นข้อความเดียวกัน หรือว่าเป็นคนละข้อความกัน ซึ่งใช้ระดับของการซ้อนกัน (Degree of Overlap) ของบริเวณทั้งสองในการตรวจสอบ โดยกำหนดให้ D_α และ D_β เป็นบริเวณของบริเวณตัวอักษร α และ β ซึ่งเราจะนิยามระดับของการซ้อนกัน หรือ Degree of Overlap คือ

$$\Lambda(\alpha,\beta) = max[s(D_\alpha \cap D_\beta)/s(D_\alpha), s(D_\alpha \cap D_\beta)/s(D_\beta)]$$

โดยที่ $s(\lambda)$ คือจำนวนพิกเซลภายใน λ ซึ่ง α และ β จะถูกพิจารณาว่าเป็นข้อความเดียวกัน ถ้าหาก $T_0 <= \Lambda(\alpha,\beta)$ โดยที่ T_0 คือค่า Threshold ซึ่งจะตั้งไว้ที่ 0.9 มิฉะนั้นแล้ว α และ β จะถือว่าเป็นคนละข้อความกัน

Overlapping Window ที่มาจากข้อความเดียวกันจะถูกทำการรวมเข้าด้วยกัน และกระบวนการในการรวมเข้าด้วยกัน (Merging Process) นั้นจะวนซ้ำไปเรื่อยจนกระทั่งไม่มี Window ที่สามารถรวมกันได้อีก

หลังจากนำ TPI ที่ได้จากขั้นตอนแรก มาผ่านการประมวลผลอัลกอริทึม MultiCAMShift จะทำให้สามารถตรวจจับบริเวณที่เป็นตัวอักษรได้ ดังแสดงตัวอย่างในภาพที่ 2.8 (a) และในภาพที่ 2.8 (b) เป็นการแสดงให้เห็นถึง TPI ที่สอดคล้องกับบริเวณที่ตรวจจับได้



(a)

(b)

ภาพที่ 2.8 แสดงผลลัพธ์ที่ได้จากการตรวจจับตัวอักษร

(a) แสดงบริเวณตัวอักษรที่ตรวจจับได้

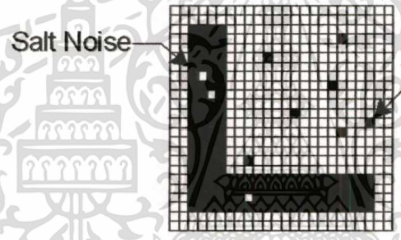
(b) แสดง TPI ที่สอดคล้องกับภาพที่ 2.8 (a)

จะเห็นได้ว่าวิธีการแบ่งแยกพื้นผิวด้วย MLP (Multi-layer Perceptron) นั้นมีความสามารถเพียงพอที่จะใช้ตรวจจับบริเวณที่เป็นตัวอักษรในสภาพหลายๆ อย่าง แต่ว่าอย่างไรก็ตามนั้น การที่จะนำเอาผลลัพธ์ที่ได้จาก MLP ไปใช้งานเลยนั้น อาจจะดูไม่เพียงพอนัก เพราะจากภาพที่ 2.8 นั้นจะเห็นว่ามี False Alarm เกิดขึ้นด้วยบางส่วน จึงมีความจำเป็นที่จะต้องขจัดสิ่งเหล่านี้ออกไปเสียก่อนเพื่อเพิ่มประสิทธิภาพให้สูงขึ้น

2.2 การปรับปรุงภาพ (Image Enhancement) (Shapiro and Stockman. 2001)

การปรับปรุงภาพเป็นกระบวนการที่ทำให้คุณภาพของภาพ (Image) ดีขึ้นก่อนที่จะนำไปวิเคราะห์ โดยเหตุผลที่ต้องมีการปรับปรุงภาพก่อนนั้น เป็นเพราะว่าเวลาที่เราได้ข้อมูล เช่นรูปภาพที่ได้จากกล้องถ่ายรูปดิจิตอลนั้น มักจะมีสัญญาณรบกวน (Noise) ปนมาด้วยเสมอ ซึ่งสัญญาณรบกวนนั้นอาจจะเกิดมาจากแสงสว่าง หรือไม่ก็เกิดสัญญาณรบกวนจากตัวกล้องถ่ายรูปเอง อีกทั้งสัญญาณรบกวนอาจจะยังสามารถเกิดจากปัจจัยอื่นๆ อีกมากมาย ดังนั้นจึงต้องมีการปรับปรุงภาพให้มีคุณภาพที่ดีขึ้นเสียก่อนที่จะนำเอาไปวิเคราะห์

เป้าหมายหลักของการปรับปรุงภาพคือการกำจัดสัญญาณรบกวนให้หมดไป หรือไม่ก้อทำให้เหลือน้อยที่สุด โดยสัญญาณรบกวนนั้นมีอยู่หลายลักษณะ เช่นสัญญาณรบกวนแบบเม็ดขาว (Salt Noise) และสัญญาณรบกวนแบบเม็ดดำ (Pepper Noise) ดังแสดงในภาพที่ 2.9



ภาพที่ 2.9 แสดงลักษณะสัญญาณรบกวนทั้งแบบ Salt Noise และ Pepper Noise

โดยปกติแล้วนั้น รูปภาพที่ได้มานั้นอาจจะถูกรบกวนโดยสัญญาณรบกวน (Noise) ได้หลายประการ เราสามารถที่จะกำจัดสัญญาณรบกวน (Noise) ออกได้โดยใช้ฟิลเตอร์ (Filter) โดยฟิลเตอร์แบ่งออกเป็น 2 ประเภท คือ

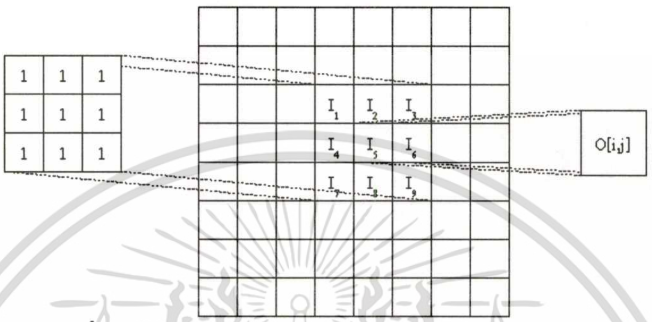
- ฟิลเตอร์ที่เป็นเชิงเส้น (Linear Filter) คือฟิลเตอร์ที่มีการให้น้ำหนัก (Weight) ในแต่ละพิกเซล
- ฟิลเตอร์ที่ไม่เป็นเชิงเส้น (Nonlinear Filter) คือฟิลเตอร์ที่ไม่มีการให้น้ำหนัก (Weight) ในแต่ละพิกเซล

ฟิลเตอร์ที่เป็นเชิงเส้น (Linear Filter) นั้นสามารถแบ่งออกเป็นได้ 2 ประเภทคือ

- ฟิลเตอร์ที่ไม่มีการผันแปรในเชิงพื้นที่ (Spatially Invariant Filters) คือฟิลเตอร์ที่ให้น้ำหนัก (Weight) เดียวกันหมดในทุกๆ ส่วนของรูปภาพ
- ฟิลเตอร์ที่มีการผันแปรในเชิงพื้นที่ (Spatially Varying Filters) คือฟิลเตอร์ที่มีการผันแปรน้ำหนัก (Weight) ในแต่ละส่วนของรูปภาพ

2.2.1 Mean Filter

Mean Filter เป็นฟิลเตอร์ที่เป็นเชิงเส้นที่ไม่มีการผันแปรในเชิงพื้นที่ (Spatially Invariant Linear Filter) โดยหลักการทำงานของ Mean Filter ก็คือจะใช้ Mask (หรือจะเรียกว่าเป็น Kernel, Convolution หรือ Structure Element ก็ได้) กระทำกับทุกๆ พิกเซลในรูปภาพ โดยจะเป็นการแทนค่าของพิกเซลด้วยค่าเฉลี่ยของทุกๆ พิกเซลที่กระทำกับ Mask อยู่ดังแสดงในภาพที่ 2.10



ภาพที่ 2.10 แสดงลักษณะการทำงานของ Mean Filter

การทำงานของ Mean Filter เป็นไปตามสมการ

$$O[i,j] = \frac{1}{M} \sum_{(k,l) in N} I[k,l]$$

โดยจากภาพที่ 2.10 นั้น ให้ความสำคัญกับทุกๆ Neighborhood เท่ากันหมด โดยใช้ Mask ขนาด 3x3 พิกเซลที่ซึ่งมี Weight เท่ากันหมดนั่นเอง โดยเมื่อพิจารณาจากสมการและภาพที่ 2.10 แล้วนั้น จะได้ว่า

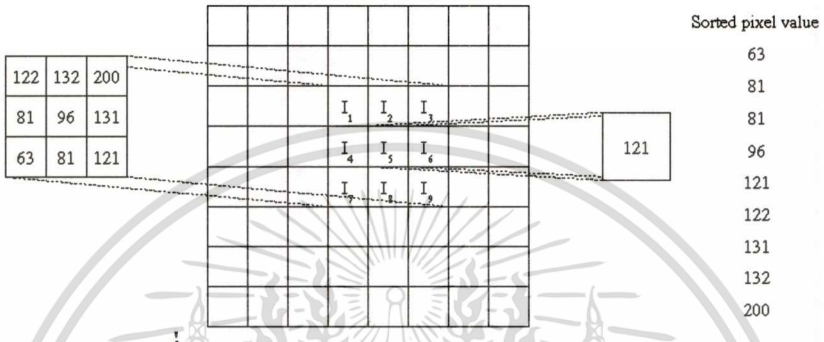
$$O[i,j] = \frac{1}{9}[(1 \times I_1) + (1 \times I_2) + ... + (1 \times I_9)]$$

เมื่อคำนวณได้ค่า O[i,j] ออกมาแล้วก็ทำการนำค่านั้นไปแทนที่ค่าเดิมของรูปภาพ ณ ตำแหน่ง O[i,j] นั่นเอง

ภาพที่ 2.11 แสดงภาพตัวอย่างก่อนและหลังจากนำมาผ่าน Mean Filter ด้วย Mask 5x5

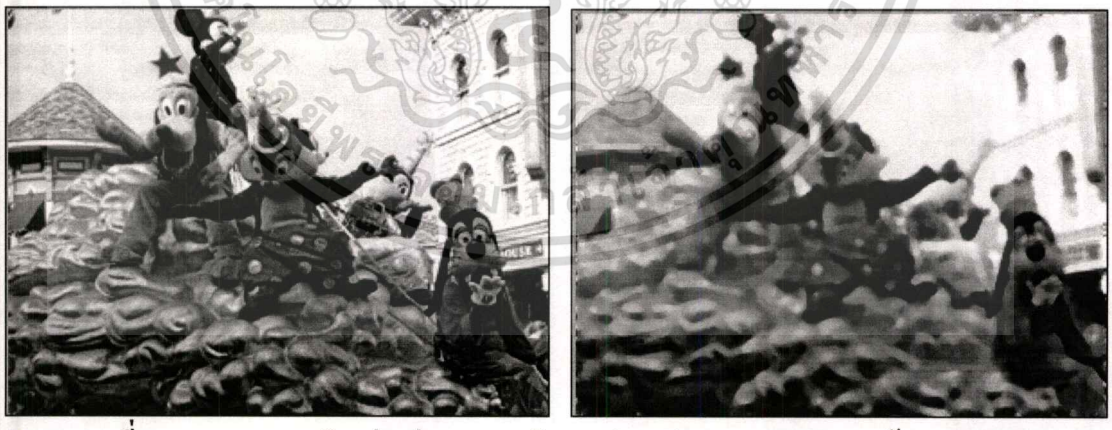
2.2.2 Median Filter

Median Filter แตกต่างจากการปรับปรุงภาพด้วย Mean Filter ตรงที่ Mean Filter จะเป็นการใช้ Mask กระทำกับทุกๆ พิกเซลในรูปภาพ โดยจะเป็นการแทนค่าของพิกเซลด้วยค่าเฉลี่ยของทุกๆ พิกเซลที่กระทำกับ Mask แต่สำหรับการใช้ Median Filter นั้นจะทำการแทนที่ค่าของพิกเซลแต่ละพิกเซลในรูปภาพด้วยค่ากลาง (Median) ของค่าในบริเวณรอบๆ



ภาพที่ 2.12 แสดงลักษณะการทำงานของ Median Filter

จากภาพที่ 2.12 ซึ่งแสดงลักษณะการทำงานของ Median Filter โดยจะเป็นการนำเอาค่าในแต่ละพิกเซลมาจัดเรียง (Sort) แล้วนำค่าที่เป็นค่ากลาง (Median) มาแทนที่ค่าเดิมในรูปภาพ ณ ตำแหน่งนั้น โดยจากรูปจะเห็นว่า เมื่อนำค่ามาเรียงกันแล้ว พบว่าค่ากลาง คือ 121 ดังนั้นค่าเดิมคือ 96 จะถูกแทนที่ด้วยค่ากลาง ซึ่งก็คือ 121

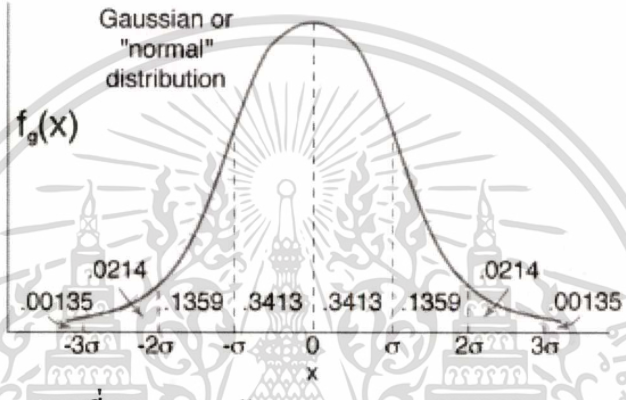


ภาพที่ 2.13 แสดงภาพตัวอย่างก่อนและหลังจากนำมาผ่าน Median Filter ด้วย Mask 5x5

สิ่งที่แตกต่างกันระหว่าง Mean Filter กับ Median Filter คือว่า Mean Filter นั้นพยายามที่จะลดความคมของส่วนที่เป็นขอบ แต่สำหรับ Median Filter นั้นไม่ได้เป็นเช่นนั้น ซึ่งตัว Median Filter นั้นมีประสิทธิภาพที่อยู่ในเกณฑ์ที่ดีมากสำหรับการจัดการกับสัญญาณรบกวนแบบ Salt Noise และ Pepper Noise

2.2.3 Gaussian Filter

Gaussian Filter เป็นฟิลเตอร์ที่เป็นเชิงเส้น (Linear Filter) ที่ซึ่งค่าน้ำหนัก (Weight) นั้นจะเป็นไปตามลักษณะของฟังชันก์เก๊าส์ (Gaussian Function) ซึ่งก็คือตัว Gaussian Filter นั้นจะให้ความสำคัญของแต่ละ Neighbor ไม่เท่ากัน โดยที่จะให้ความสำคัญกับบริเวณที่อยู่ตรงกลางมากที่สุด แล้วก็จะให้ความสำคัญหรือให้ Weight น้อยลงเรื่อยๆ ตามลักษณะของ Gaussian Function ดังแสดงในภาพที่ 2.14



ภาพที่ 2.14 แสดงลักษณะของ Gaussian Function

คุณสมบัติที่เด่นประการหนึ่งของ Gaussian Filter ก็คือสามารถที่จะจัดการกับสัญญาณรบกวนแบบ Gaussian Noise ได้ดีมาก

ภาพที่ 2.15 แสดงภาพตัวอย่างก่อนและหลังจากนำมาผ่าน Gaussian Filter ด้วย Mask 5x5

2.3 การตรวจจับเส้นขอบ (Edge Detection) (Shapiro and Stockman. 2001)

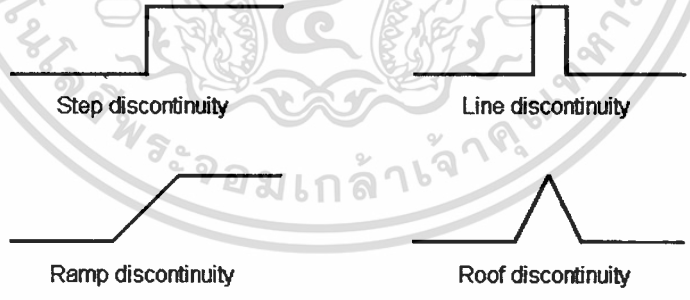
เส้นขอบ (Edge) หมายถึงการเปลี่ยนแปลงความเข้ม (Intensity) ในปริมาณที่มากของรูปภาพซึ่งอาจเรียกได้ว่าเป็นความไม่ต่อเนื่อง (Discontinuity) ก็ได้ โดยปกติแล้วนั้นจะเกิดขึ้นระหว่างขอบของบริเวณ 2 บริเวณที่แตกต่างกัน ดังนั้นการตรวจจับเส้นขอบ (Edge Detection) ก็จะเป็นกระบวนการในการตรวจจับการเปลี่ยนแปลงความเข้มในปริมาณที่มากในรูปภาพนั่นเอง

เราสามารถแบ่งลักษณะของเส้นขอบ (Edge) หรือความไม่ต่อเนื่อง (Discontinuity) ออกเป็น 2 ประเภท ดังนี้

- ความไม่ต่อเนื่องแบบขั้นบันได (Step Discontinuity)
- ความไม่ต่อเนื่องแบบเส้น (Line Discontinuity)

ความไม่ต่อเนื่องแบบขั้นบันได (Step Discontinuity) คือเป็นการเปลี่ยนแปลงค่าความเข้ม (Intensity) อย่างฉับพลันจากค่าหนึ่งไปเป็นอีกค่าหนึ่ง แต่สำหรับความไม่ต่อเนื่องแบบเส้น (Line Discontinuity) คือเป็นการเปลี่ยนแปลงค่าความเข้ม (Intensity) อย่างฉับพลันแล้วกลับคืนค่าสู่ค่าเดิมภายในช่วงระยะสั้นๆ

แต่เนื่องจากในความเป็นจริงนั้น การที่จะพบเส้นขอบหรือความไม่ต่อเนื่องที่เปลี่ยนแปลงฉับพลันขนาดนี้นั้นมีโอกาสพบน้อยมาก ส่วนมากจะพบเส้นขอบแบบขั้นบันได ในลักษณะของ Ramp Edge ส่วน Line Edge นั้นเรามักจะพบในลักษณะของ Roof Edge ดังแสดงในภาพที่ 2.16



ภาพที่ 2.16 แสดงลักษณะของความไม่ต่อเนื่อง (Discontinuity)

การตรวจจับเส้นขอบ (Edge Detection) แบ่งออกเป็น 2 ประเภท ดังนี้

- 1st Derivative Edge Detection หรือ Gradient
- 2nd Derivative Edge Detection หรือ Laplacian

Gradient เป็นการวัดการเปลี่ยนแปลงในฟังก์ชัน ซึ่งเขียนได้ดังสมการ

$$\nabla f = \frac{\Delta f(x)}{\Delta x}$$

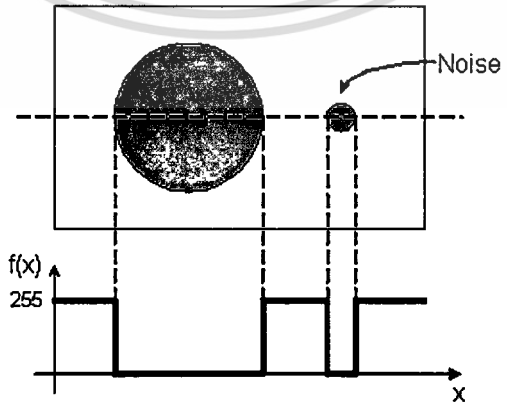
จากสมการ การหาค่า Gradient ก็คือการหาค่าการเปลี่ยนแปลงของฟังก์ชัน f(x) หรือการหาอนุพันธ์อันดับที่ 1 โดยที่ ∇f ก็คือ Gradient ของฟังก์ชัน f นั่นเอง เมื่อทำการหา Gradient มาได้แล้ว จุดสูงสุดของ f'(x) ก็จะถือได้ว่าเป็นจุดที่เป็นเส้นขอบ (Edge) นั่นเอง

แต่สำหรับการหา Laplacian เป็นการหาอนุพันธ์อันดับที่ 2 ของฟังก์ชัน (2^{nd} Derivative) ดังสมการ โดยที่เมื่อทำการหา Laplacian ได้แล้ว จุดที่ตัดจุดศูนย์ (Zero-crossing points) คือจุดที่เป็นเส้นขอบ

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

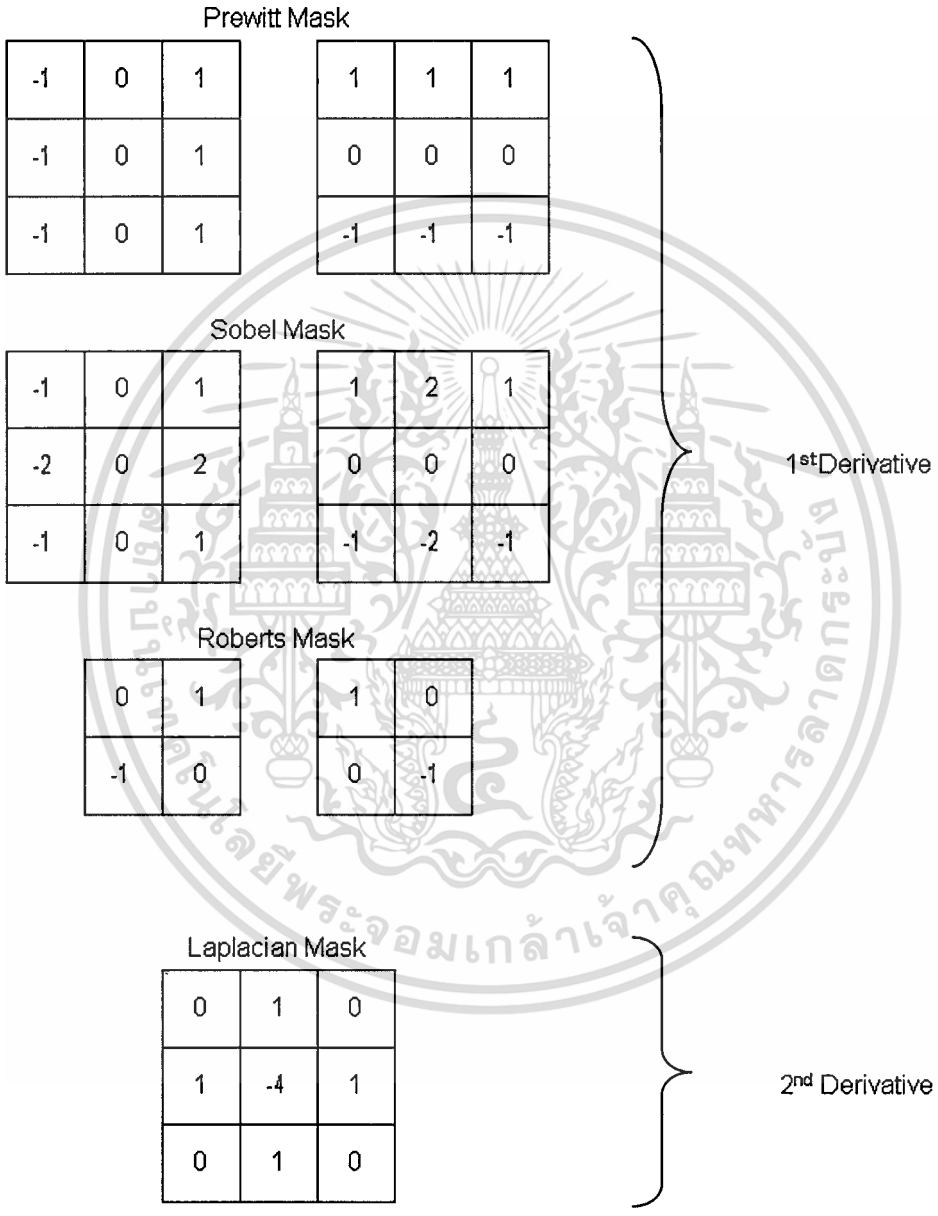
การหาเส้นขอบโดยการใช้ Gradient (1^{st} Derivative) นั้น จุดที่บ่งบอกว่าเป็นเส้นขอบ (Edge) จะกว้างเกินไป เพราะเราคิดว่าถ้าค่าอนุพันธ์ของฟังก์ชันสูงเกินกว่าค่าหนึ่ง เราจะพิจารณาว่าจุดนั้นเป็นเส้นขอบ ดังนั้นบริเวณที่ถูกพิจารณาว่าเป็นเส้นขอบจะมีจำนวนมาก แต่สำหรับวิธีการหาเส้นขอบโดยใช้ Laplacian (2^{nd} Derivative) นั้นดีกว่าตรงที่จุดที่จะถูกพิจารณาว่าเป็นเส้นขอบนั้นคือจุดที่ตัดจุดศูนย์ (Zero-crossing) เลยนั่นเอง ซึ่งสำหรับบริเวณที่เป็น 0 ติดๆ กันไปเรื่อยๆ ก็จะทำให้รู้ว่าบริเวณนั้นไม่เป็นส่วนที่เป็นเส้นขอบ

ในการหาเส้นขอบ (Edge Detection) นั้น ถ้าหากว่าภาพที่จะนำมาหาเส้นขอบมีสัญญาณรบกวนอยู่มาก ก็จะทำให้มีผลกระทบต่อการหาเส้นขอบ ดังนั้นการที่จะนำภาพมาผ่านกระบวนการปรับปรุงรูปภาพ (Image Enhancement) เพื่อทำกำจัดหรือลดสัญญาณรบกวนให้เหลือน้อยที่สุดเท่าที่จะเป็นไปได้ก่อนที่จะนำเข้าสู่ขั้นตอนของการหาเส้นขอบ จึงเป็นการช่วยให้การหาเส้นขอบมีประสิทธิภาพมากยิ่งขึ้น



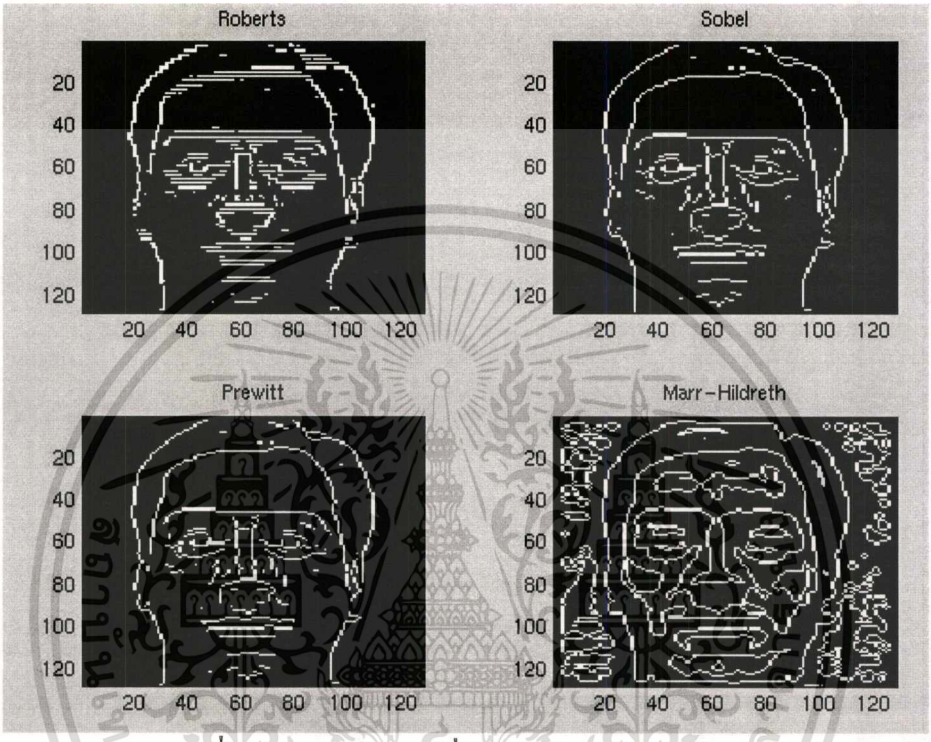
ภาพที่ 2.17 แสดงผลกระทบของสัญญาณรบกวนที่มีต่อการตรวจจับเส้นขอบ

หลักการหาเส้นขอบนั้นสามารถทำได้โดยการใช้ Mask หรือที่เรียกได้หลายอย่าง เช่น Kernel หรือ Structure Element เป็นต้น มากระทำกับภาพ โดยภาพที่ 2.18 เป็นการแสดงตัวอย่างของ Mask ที่ใช้ในการตรวจจับเส้นขอบ



ภาพที่ 2.18 แสดงตัวอย่างของ Mask ที่ใช้ในการตรวจจับเส้นขอบ

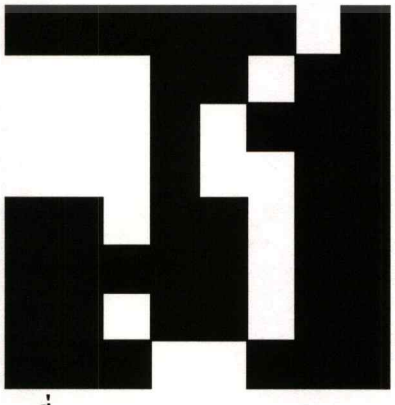
โดยจากภาพที่ 2.18 นั้น ทั้ง Prewitt Mask, Sobel Mask และ Roberts Mask เป็น Mask ที่ใช้ในการหาเส้นขอบแบบ 1st Derivative ทั้งแบบแนวดิ่ง (Vertical) และแนวนอน (Horizontal) ส่วน Laplacian Mask เป็น Mask ที่ใช้ในการเส้นขอบแบบ 2nd Derivative



ภาพที่ 2.19 แสดงผลลัพธ์ที่ได้จากการตรวจจับเส้นขอบ

2.4 หลักการ Connected Component

Connected Component เป็นหลักการที่ใช้ในการวิเคราะห์ภาพขาวดำ (Binary Image) ว่าในภาพดังกล่าวนั้นประกอบไปด้วยองค์ประกอบ (Component) ทั้งหมดเป็นจำนวนเท่าใด



0	0	0	0	0	0	1	0
1	1	1	0	0	1	0	0
1	1	1	0	1	0	0	0
1	1	1	0	1	1	0	0
0	0	1	0	0	1	0	0
0	0	0	0	0	1	0	0
0	0	1	0	0	1	0	0
0	0	0	1	1	0	0	0

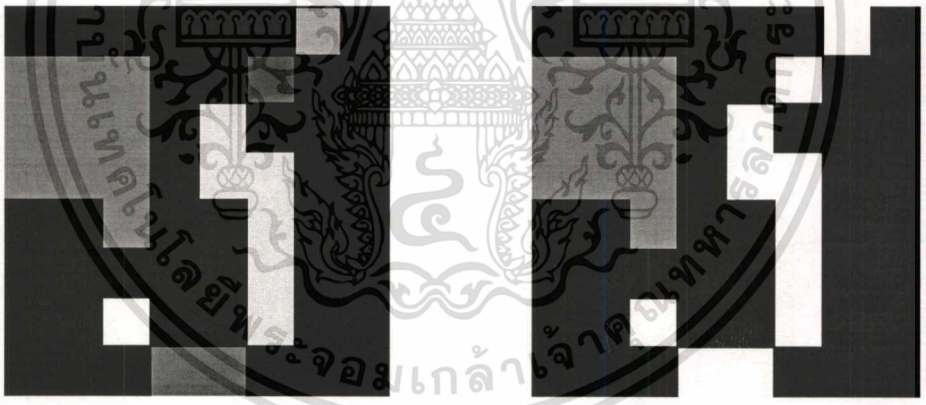
ภาพที่ 2.20 แสดงภาพ Binary Image และค่าของแต่ละพิกเซลที่สอดคล้องกับภาพ

โดยที่นิยามของหลักการ Connected Component มีอยู่ว่า แต่ละพิกเซลจะถือว่าเป็นองค์ประกอบ (Component) เดียวกันได้ ก็ต่อเมื่อพิกเซลจะต้องเชื่อมต่อถึงกัน ทุกคู่ของพิกเซลต้องมีค่าเดียวกัน และมีเส้นทางเชื่อมถึงกันได้

จากภาพที่ 2.20 จะเห็นได้ว่าเป็นภาพขาวดำ (Binary Image) ซึ่งแต่ละพิกเซลมีค่าได้คือ 0 (สีดำ) กับ 1 (สีขาว) ซึ่งในการที่จะพิจารณาถึงองค์ประกอบที่เชื่อมต่อถึงกัน (Connected Component) นั้น เราสามารถที่ใช้หลักการพิจารณาองค์ประกอบที่เชื่อมต่อถึงกันออกเป็น 2 กรณี ดังนี้

- 4-neighborhood Connected Component
- 8-neighborhood Connected Component

โดยที่ถ้าหากเป็น 4-neighborhood นั้น จะพิจารณาเพียงแค่พิกเซลที่อยู่ติดทางด้านซ้าย ขวา บน และล่าง ของพิกเซลที่เรากำลังพิจารณาอยู่เท่านั้น แต่ถ้าหากเป็น 8-neighborhood นั้นจะพิจารณาพิกเซลทุกๆ ด้านที่อยู่ติดกับพิกเซลที่เรากำลังพิจารณาอยู่ ซึ่งเราสามารถที่จะกำหนดแต่ละองค์ประกอบที่เชื่อมต่อถึงกันของภาพได้ดังแสดงในภาพที่ 2.21



0	0	0	0	0	0	6	0
1	1	1	0	0	5	0	0
1	1	1	0	4	0	0	0
1	1	1	0	4	4	0	0
0	0	1	0	0	4	0	0
0	0	0	0	0	4	0	0
0	0	2	0	0	4	0	0
0	0	0	3	3	0	0	0

4-neighborhood Connected Component

0	0	0	0	0	0	2	0
1	1	1	0	0	2	0	0
1	1	1	0	2	0	0	0
1	1	1	0	2	2	0	0
0	0	1	0	0	2	0	0
0	0	0	0	0	2	0	0
0	0	2	0	0	2	0	0
0	0	0	2	2	0	0	0

8-neighborhood Connected Component

ภาพที่ 2.21 แสดงการกำหนดองค์ประกอบที่เชื่อมต่อถึงกัน (Connected Component Labeling)

จากภาพที่ 2.21 จะเห็นว่าถ้าเรากำหนดแต่ละองค์ประกอบที่เชื่อมต่อถึงกันด้วย 4-neighborhood Connected Component จะพบว่าเราสามารถแบ่งองค์ประกอบออกได้ทั้งหมด 6 องค์ประกอบ แต่ถ้าหากเราใช้ 8-neighborhood Connected Component ในการกำหนดแต่ละองค์ประกอบที่เชื่อมต่อถึงกัน จะพบว่าเราจะแบ่งองค์ประกอบออกได้ทั้งหมดเพียงแค่ 2 องค์ประกอบเท่านั้น

การกำหนดองค์ประกอบที่เชื่อมต่อถึงกัน (Connected Component Labeling) สามารถทำได้ 2 วิธี คือ

- Recursive Algorithm
- Row-by-row 2 passes Algorithm

```
Compute the connected components of a binary image.

B is the original binary image.
LB will be the labeled connected component image.

        procedure recursive_connected_components(B, LB);
        {
        LB := negate(B);
        label := 0;
        find_components(LB, label);
        print(LB);
        }

        procedure find_components(LB, label);
        {
            for L := 0 to MaxRow
                for P := 0 to MaxCol
                    if LB[L,P] == -1 then
                        {
                        label := label + 1;
                        search(LB, label, L, P);
                        }
        }

        procedure search(LB, label, L, P);
        {
        LB[L,P] := label;
        Nset := neighbors(L, P);
        for each LB[L',P'] in Nset
            {
            if LB[L',P'] == -1
            then search(LB, label, L', P');
            }
        }
```

ภาพที่ 2.22 แสดง Recursive Algorithm สำหรับการกำหนดองค์ประกอบที่เชื่อมต่อถึงกัน

```
Compute the connected components of a binary image.

B is the original binary image.
LB will be the labeled connected component image.

    procedure classical_with_union-find(B, LB);
    {
    "Initialize structures."
    Initialize();
    "Pass 1 assigns initial labels to each row L of the image."
    for L := 0 to MaxRow
        {
        "Initialize all labels on line L to zero"
        for P := 0 to MaxCol
            LB[L,P] := 0;
        "Process line L."
        for P := 0 to MaxCol
            if B[L,P] == 1 then
                {
                A := prior_neighbors(L, P);
                if isempty(A)
                then {M := label; label := label + 1;};
                else M := min(labels(A));
                LB[L,P] := M;
                for X in labels(A) and X <> M
                    union(M, X, PARENT);
                }
        }
    "Pass 2 replaces Pass 1 labels with equivalence class labels."
    for L := 0 to MaxRow
        for P := 0 to MaxCol
            if B[L,P] == 1
            then LB[L,P] := find(LB[L,P], PARENT);
    };
```

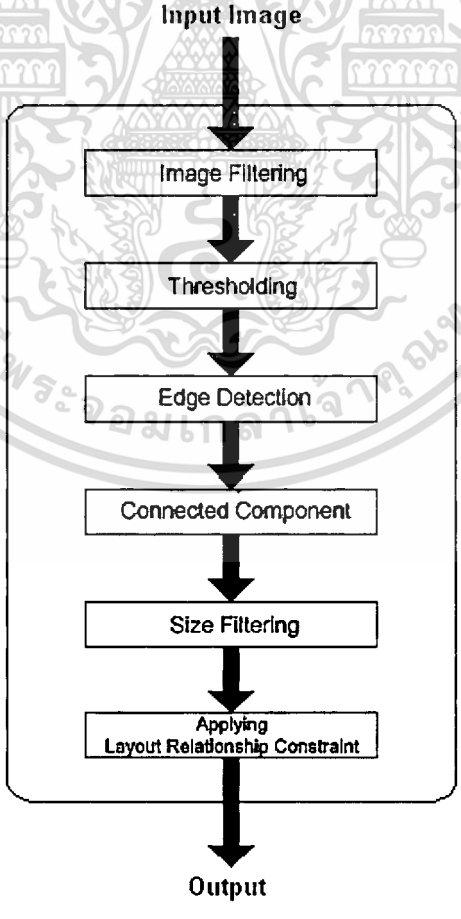
ภาพที่ 2.23 แสดง Row-by-row 2 passes Algorithm สำหรับการกำหนดองค์ประกอบที่เชื่อมต่อถึงกัน

โดยทั้ง 2 อัลกอริทึมที่ใช้สำหรับการกำหนดองค์ประกอบที่เชื่อมต่อถึงกันนั้นมีข้อสังเกตอยู่ที่ว่าวิธีการ Recursive Algorithm นั้นเป็นวิธีการที่จะใช้ทรัพยากรทางด้านหน่วยความจำค่อนข้างมาก ดังนั้นวิธีการนี้จะใช้ได้ดีมากหากว่ามีหน่วยความจำ (Memory) มากเพียงพอที่จะเก็บเซตของพิกเซลทั้งหมดไว้ได้ แต่สำหรับวิธีการ Row-by-row 2 passes Algorithm นั้นจะใช้ทรัพยากรทางด้านหน่วยความจำที่น้อยกว่าแบบแรก โดยวิธีการนี้มีหลักการทำงานคือ รอบแรกที่ทำการประมวลผลนั้น จะทำการกำหนดหมายเลขให้กับแต่ละองค์ประกอบไปทีละบรรทัด เสร็จแล้วการประมวลผลในรอบที่สองนั้น จะเป็นการแทนที่องค์ประกอบที่เชื่อมต่อถึงกันด้วยหมายเลขเดียว โดยสามารถแสดง Recursive Algorithm ในภาพที่ 2.22 และแสดง Row-by-row 2 passes Algorithm ในภาพที่ 2.23

บทที่ 3
ระบบตรวจจับตัวอักษรโดยใช้ความสัมพันธ์ขององค์ประกอบ

3.1 อัลกอริทึมที่ใช้ในการตรวจจับตัวอักษร

กระบวนการทำงานของระบบตรวจจับตัวอักษรนั้นประกอบไปด้วยขั้นตอน ดังแสดงในภาพที่ 3.1 อินพุตของระบบตรวจจับตัวอักษรคือ ภาพถ่ายที่ได้จากกล้องถ่ายรูปดิจิตอล โดยขั้นตอนแรกของกระบวนการทำงานของระบบก็คือ การนำภาพถ่ายนั้นมาผ่านขั้นตอนก่อนประมวลผล (Pre-processing) โดยการนำภาพมาผ่านฟิลเตอร์ เพื่อที่จะทำการลดสัญญาณรบกวน (Noise) ให้เหลือน้อยที่สุดเท่าที่จะเป็นไปได้ สำหรับระบบที่พัฒนาในโครงงานนี้ เราใช้ Gaussian Filter ในการกรองสัญญาณรบกวนออกจากภาพถ่ายอินพุต (รายละเอียด แสดงในหัวข้อ 2.2.3)



ภาพที่ 3.1 แสดงโครงสร้างของระบบตรวจจับตัวอักษร

หลังจากที่นำภาพมาผ่านฟิลเตอร์เพื่อลดสัญญาณรบกวนต่างๆ ลงแล้วนั้น เพื่อความง่ายในการที่จะนำภาพเอาไปประมวลผลในกระบวนการลำดับถัดๆ ไป จึงทำการแปลงภาพให้เหลือเป็นเพียงแค่ภาพ 2 ระดับ (Binary Image) โดยการใช้เทคนิค Thresholding ซึ่งผลลัพธ์ที่ได้จากขั้นตอนนี้ก็คือแต่ละพิกเซลในภาพนั้นจะเป็นไปได้เพียงแค่ ไม่เป็นสีขาวก็เป็นสีดำ ซึ่งสิ่งที่สำคัญประการหนึ่งของเทคนิคการทำ Thresholding ก็คือค่า Threshold ที่ตั้งไว้เพื่อใช้ในการแยกแยะที่จะระบุค่าในแต่ละพิกเซล ซึ่งมีค่าตั้งแต่ 0 ถึง 255 ให้เหลือเป็นค่า 0 หรือ 1 ซึ่งหากมีการปรับเปลี่ยนค่า Threshold ไปก็จะทำให้ผลลัพธ์ที่ได้จากขั้นตอนการทำ Thresholding นั้นแตกต่างกันออกไป โดยตัวระบบเลือกใช้ค่า 128 เป็นค่า Threshold สำหรับการทำ Thresholding

เมื่อผ่านขั้นตอนการแปลงภาพให้เป็น Binary Image แล้วนั้น ก็ทำการนำภาพที่ได้นั้นมาผ่านอัลกอริทึมที่ใช้ในการตรวจจับหาเส้นขอบ (Edge Detection) ซึ่งในการตรวจจับหาเส้นขอบนั้นหากภาพที่นำมาตรวจจับหาเส้นขอบมีสัญญาณรบกวน (Noise) ก็จะมีผลกระทบอย่างมากต่อการตรวจจับหาเส้นขอบ ดังนั้นการที่ทำการลดสัญญาณรบกวนโดยการใช้ฟิลเตอร์ ในขั้นแรก (Pre-processing) ก็จะเป็นการช่วยทำให้การตรวจจับเส้นขอบทำได้อย่างที่มีประสิทธิภาพมากยิ่งขึ้น ในการหาเส้นขอบ เราใช้อัลกอริทึมที่แสดงในหัวข้อ 2.3 โดยใช้ Laplacian Mask แสดงในภาพที่ 2.18

ขั้นตอนถัดไปเป็นการนำเอาภาพ 2 ระดับ (Binary Image) ที่ผ่านขั้นตอนการตรวจจับหาเส้นขอบมาทำการวิเคราะห์หาองค์ประกอบที่เชื่อมต่อถึงกัน (Connected Component) เพื่อที่จะวิเคราะห์และแยกแยะว่าในภาพนั้นมีองค์ประกอบทั้งหมดอยู่เท่าใด หลังจากนั้นก็จะเป็นการกรองด้วยขนาดขององค์ประกอบและใช้ข้อกำหนดกฎเกณฑ์ความสัมพันธ์ของโครงร่าง (Layout Relation Constraint) เพื่อที่จะใช้ช่วยในการวิเคราะห์ว่าองค์ประกอบใดเป็นส่วนที่เป็นตัวอักษรในภาพ โดยใช้กฎเกณฑ์ที่ได้จากการสังเกตการจัดวางและความสัมพันธ์ของตำแหน่งของตัวอักษรและข้อความ ดังนี้

- ตัวอักษรบนข้อความเดียวกันนั้น ควรจะอยู่ใกล้ๆ กับตัวอักษรอื่นๆ ที่อยู่ข้างเคียง คือ ระยะห่างระหว่างแต่ละองค์ประกอบ ควรที่จะมีขอบเขตอยู่ไม่เกินค่าๆ หนึ่ง
- อัตราส่วนของด้าน (Aspect Ratio) คืออัตราส่วนระหว่างความกว้างกับความสูงของตัวอักษรแต่ละตัวนั้น จะมีขอบเขตที่แน่นอน
- หลังจากที่กำหนดให้แต่ละตัวอักษรเป็นแต่ละส่วน (Component) ได้แล้วนั้น ตัวอักษรแต่ละตัวนั้นควรที่จะจัดเรียงเป็นแนวเดียวกันกับตัวอักษรตัวอื่น

หลังจากผ่านกระบวนการขั้นตอนข้างต้น รวมถึงใช้ข้อกำหนดกฎเกณฑ์ความสัมพันธ์ ดังที่ได้กล่าวมาทั้งหมดแล้วนั้น ก็จะทำการสร้างกรอบล้อมรอบบริเวณที่เป็นส่วนของข้อความที่เป็นตัวอักษรที่อยู่ในภาพได้

3.2 เครื่องมือที่ใช้ในการพัฒนาระบบ

Microsoft Vision System Development Kit หรือที่จักกันในนามของ MS Vision SDK ถูกพัฒนาขึ้นโดย Vision Technology Research Group เพื่อใช้ในการสนับสนุนการพัฒนาโปรแกรมทางด้าน Image Processing, Real-time Image Processing และ Computer Vision บนระบบปฏิบัติการ Microsoft Windows

MS Vision SDK มีคลาสและฟังก์ชั่นในภาษา C++ สำหรับการทำงานที่เกี่ยวข้องกับงานด้านภาพ ผู้เขียนโปรแกรมที่ติดตั้ง MS Vision SDK สามารถพัฒนาโปรแกรม โดยใช้ MS Visual C++ ในชุด Visual Studio ได้อย่างอิสระ ไม่ขึ้นกับอุปกรณ์ต่อเชื่อมที่นำสัญญาณเข้าเครื่องคอมพิวเตอร์ เพราะ MS Vision SDK อาศัย Windows Driver Model (WDM) ที่ทำหน้าที่เป็นตัวกลางในการติดต่อระหว่าง Application Software กับอุปกรณ์ต่อเชื่อม ดังนั้นผู้เขียนโปรแกรมจึงสามารถใช้อุปกรณ์ที่เป็นตัวนำข้อมูลภาพ ทั้งภาพนิ่งและภาพเคลื่อนไหวมาใช้ได้หลากหลายรูปแบบ เช่น กล้องถ่ายรูปดิจิตอล กล้องถ่ายวิดีโอดิจิตอล ที่ต่อกับอุปกรณ์คอมพิวเตอร์ผ่านทาง USB Port หรือผ่านทาง Port IEEE 1394 (Fire Wire) รวมทั้งกล้องวิดีโออนาล็อกที่ต่อผ่านทาง Capture card ที่ต่อกับคอมพิวเตอร์ผ่านทาง PCI bus เป็นต้น

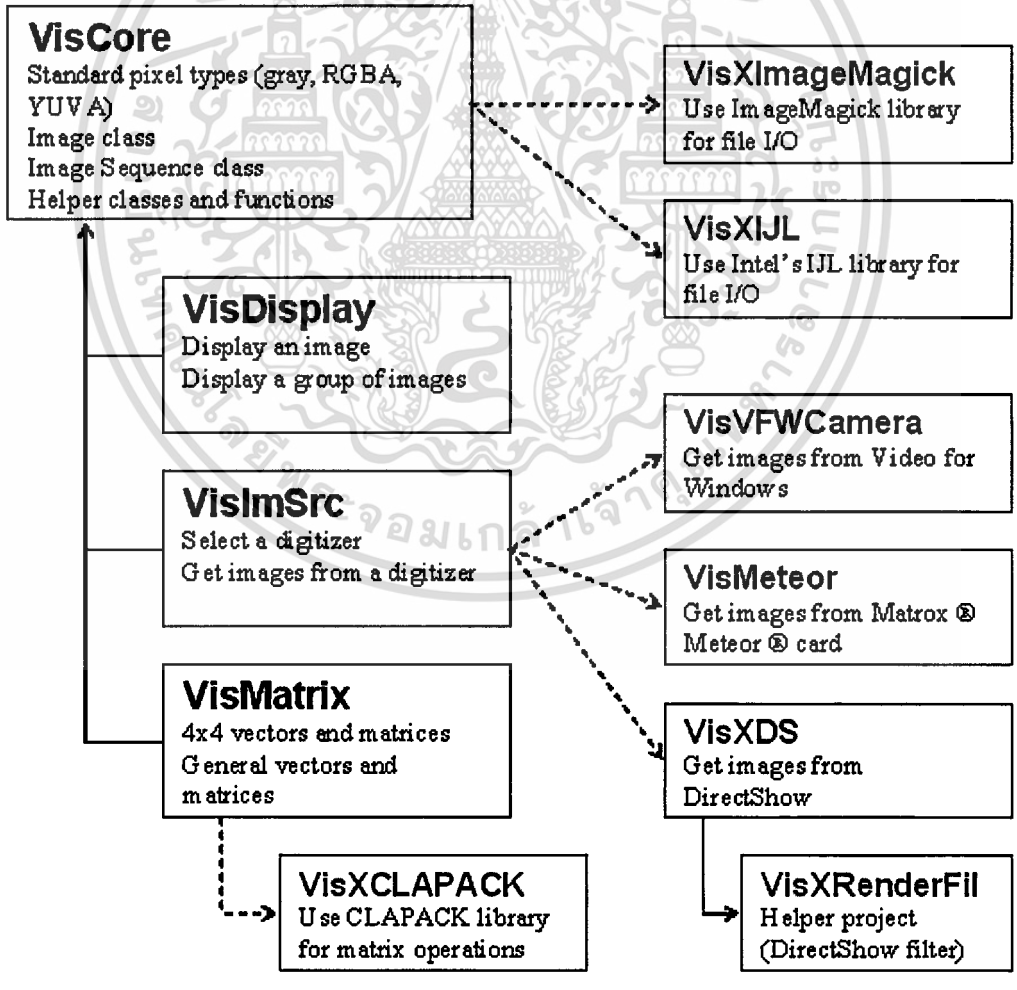
สรุปข้อดีและข้อเสียของ MS Vision SDK เมื่อเปรียบเทียบกับ ชุดพัฒนา Application Software ทางด้าน Image Processing อื่นๆ

ข้อดี

- สามารถประมวลผลได้รวดเร็วเหมาะสมกับงานแบบ real-time image processing
- การติดต่อระหว่าง Application กับระบบปฏิบัติการ MS Windows เป็นไปได้อย่างมีประสิทธิภาพ ในเรื่องของการใช้ image memory ร่วมกันระหว่าง processes ต่างๆ ภายใน Application รวมทั้งสามารถใช้งาน Graphic Device Interface
- ผู้ใช้งานสามารถสร้าง pixel data type ได้เอง (User-definable pixel) ซึ่งเหมาะสมกับงานวิจัยทางด้าน image processing
- ผู้เขียนโปรแกรมสามารถสร้าง Application โดยไม่ต้องคำนึงถึงอุปกรณ์ต่อเชื่อมที่ใช้ในการนำสัญญาณภาพเข้ามา ซึ่งรวมถึงสามารถต่อเชื่อมอุปกรณ์ใหม่ๆ ได้โดยไม่ต้องแก้ไขตัวโปรแกรม

<u>ข้อเสีย</u>

- ข้อมูลภาพทั้งหมดจะอยู่ในหน่วยความจำ (RAM) ทำให้ไม่สามารถรองรับแฟ้มข้อมูลภาพ (Image files) ชนิดที่มีขนาดใหญ่เกินกว่าที่จะอยู่ในหน่วยความจำได้หมด MS Vision SDK สามารถรองรับขนาดข้อมูลภาพได้ประมาณ 2 GB (แต่ถ้าเป็น MS Windows 9x จะสามารถรับได้ประมาณ 1 GB)
- MS Vision SDK ไม่ได้เตรียมฟังก์ชั่นสำหรับการทำ image processing ในระดับ high-level ไว้ให้ เช่น การทำงาน threshold, erosion, dilation เป็นต้น ผู้สร้างจะต้องเขียนฟังก์ชั่นเหล่านี้ไว้ใช้งานเอง หรืออาจใช้ library ของ Intel's Image Processing Library แต่ได้เตรียมฟังก์ชั่นพื้นฐานในระดับ low-level สำหรับการใช้งานด้าน image processing ไว้แล้ว



ภาพที่ 3.2 แสดงโครงสร้างหลักของ MS Vision SDK

ภาพที่ 3.2 แสดงโครงสร้างหลักของ MS Vision SDK ซึ่งประกอบไปด้วย header files ที่ใช้สำหรับการ include เพื่อให้ precomplier ทราบว่าจะมีการเรียกใช้ components หรือคลาสที่ได้ define ไว้มาใช้ในโปรแกรม โดยในที่นี้จะอธิบายในส่วนของ header file ที่สำคัญๆ ดังนี้คือ

- VisWin.h ซึ่งไม่ได้อยู่รวมกับ MS Vision SDK แต่จะอยู่ใน Microsoft Foundation Class (MFC) ซึ่งเป็น header file ที่ define macros พื้นฐาน และคลาสที่ต้องการใช้ใน MS Vision SDK
- VisCore.h ประกอบไปด้วยคลาสหลักๆ หลายคลาส เช่น คลาส CVisImage เป็นตัวที่เก็บข้อมูลภาพซึ่งใช้สำหรับการจัดเก็บ การนำเอาข้อมูลไปใช้ในการประมวลผล รวมทั้งการนำเสนอข้อมูลที่ผ่านการประมวลผลแล้ว คลาส CVisSequence ใช้สำหรับการจัดเก็บข้อมูลภาพในแบบต่อเนื่อง และการบันทึกภาพต่อเนื่องที่ผ่านการ Process แล้วในรูปแบบ AVI file และสุดท้ายฟังก์ชั่น และฟังก์ชั่น DisplayInHdc เป็นการนำเสนอข้อมูลภาพที่ผ่านการ process แล้วด้วย Windows GDI
- VisImSrc.h สนับสนุนการ capture ภาพจากอุปกรณ์เชื่อมต่อ โดยมี Class CVisImageSource และฟังก์ชั่น VisFIndImageSource ในการค้นหาแหล่งที่มาของสัญญาณภาพ
- VisDisplay.h ใช้ในการนำเสนอภาพที่ผ่านการ process แล้วเพื่อนำเสนอผ่านทางหน้าจอมอนิเตอร์ซึ่งมี Class CVisPane, CVisPaneArray รวมถึงฟังก์ชั่น VisDisplayImage

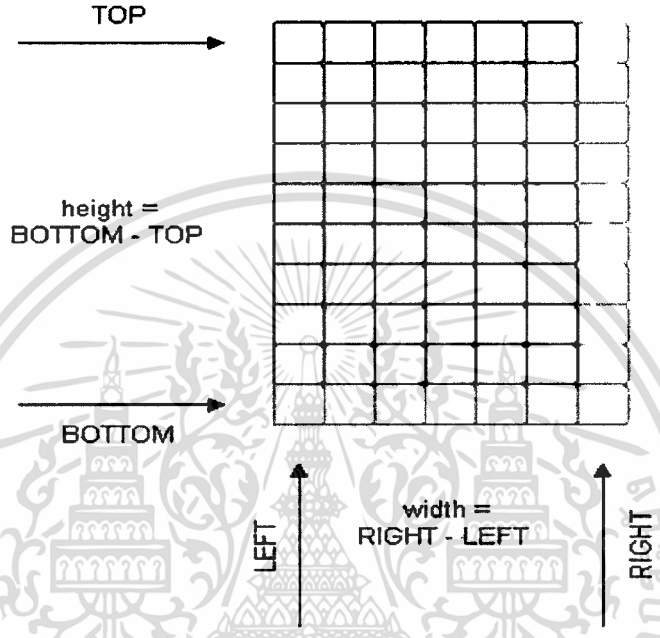
3.2.1 คลาส CVisImage

คลาส CVisImage เป็น คลาสหลักที่ใช้ในการทำงานของ MS Vision SDK คลาสนี้ถูกกำหนดไว้ใน Project "VisCore.h" คลาส CVisImage คล้ายคลึงกับ Windows bitmap header กล่าวคือ คลาส CVisImage หน้าที่เก็บคุณสมบัติหลายอย่างเกี่ยวกับภาพ อีกทั้งยังเก็บ pointer ที่ชี้ยังไปหน่วยความจำที่เก็บข้อมูลภาพจริง ข้อแตกต่างจาก bitmap คือ คลาสนี้ถูกเขียนโดยใช้ภาษา C++ และเขียนเป็นแบบ templates ที่สามารถเก็บและสามารถ process pixel format ชนิดต่างๆ กันได้

ข้อมูลภาพที่จัดเก็บใน Vision SDK จัดเก็บในรูป array ของ pixels ซึ่ง pixels ทั้งหมดจะมี data type ชนิดเดียวกันขึ้นอยู่กับผู้เขียนโปรแกรมว่าต้องการใช้ data type แบบใด pixel type (data type ของ pixel นั้นๆ) มีหลายแบบและมีทั้งแบบที่ MS Vision SDK กำหนดไว้แล้ว (Predefined Pixel and Image Types) และผู้เขียนโปรแกรมกำหนดเองภายหลัง (Adding a New Pixel Type)

MS Vision SDK ใช้โครงสร้างแบบ standard Windows RECT structure สำหรับการระบุข้อมูลเกี่ยวกับพื้นที่ของข้อมูลภาพ ดังแสดงในภาพที่ 3.3

จากภาพตำแหน่ง (0, 0) คือตำแหน่ง TOP, LEFT ในขณะที่ pixel สุดท้ายของภาพคือตำแหน่ง BOTTOM, RIGHT โดยค่า height สามารถหาได้จากการนำค่า BOTTOM ลบด้วยค่า TOP และค่า width หาได้จากการนำค่า RIGHT ลบด้วยค่า LEFT



ภาพที่ 3.3 แสดงโครงสร้างของการเก็บภาพใน Vision SDK

ผู้เขียนโปรแกรมสามารถอ้างถึงตำแหน่งของแต่ละ pixel ได้โดยตรง โดย MS Vision SDK จะกำหนดค่า x เป็นคอลัมน์ของภาพและค่า y เป็นแถวของภาพ โดยค่า x จะเริ่มต้นที่ 0 และเพิ่มค่าขึ้นทีละ 1 จากด้านซ้ายไปขวา ส่วนค่า y ก็จะเพิ่มค่าขึ้นทีละ 1 เหมือนค่า x ต่างกันตรงที่นับจากบนลงล่าง แต่จุด origin หรือจุด (0, 0) เป็นเพียงจุดสมมุติผู้เขียนสามารถกำหนดจุด origin ได้เองที่ตำแหน่งใดๆ ก็ได้ในภาพ ซึ่งทำให้การทำงานมีความยืดหยุ่นมากขึ้น

Object หรือ Instance ที่สร้างจากคลาส CVisImage นั้น ที่จริงแล้วไม่ได้เก็บข้อมูลของแต่ละ pixel จริงๆ แต่จะเก็บ pointer ที่ชี้ไปยัง block ข้อมูลจริงอีกทีหนึ่ง เพราะฉะนั้นผู้เขียนสามารถสร้าง objects หลายๆ objects โดยที่แต่ละ objects สามารถอ้างอิงไปที่ข้อมูลจริง โดยไม่คัดลอกข้อมูลไปยังที่ใหม่ โดยเมื่อไม่มีการใช้ object นั้นๆ แล้ว MS Vision SDK จะทำการตรวจสอบว่าถ้าไม่มีการอ้างอิงถึงข้อมูลจริงๆ อีกต่อไปแล้ว ก็จะทำการคืนหน่วยความจำส่วนนั้นไป

นอกจากนี้ในคลาส CVisImage ยังสามารถเลือกที่จะ Process เพียงแค่บางพื้นที่ของภาพเท่านั้น โดยใช้ method SubImage() ซึ่งสามารถกำหนดให้มีการ process เฉพาะตำแหน่งที่กำหนด

3.2.1.1 Pixel และ Image Type

CVisImage ถูกสร้างในแบบ template ในโครงสร้างของภาษา C++ ซึ่งเราสามารถกำหนดชนิดของ object ที่สร้างว่าจะให้เป็นแบบชนิดใดแล้วแต่ความต้องการโดยใช้คำสั่ง CVisImage <pixeltype>

ชนิดของพิกเซล (Pixel Types)

- Gray Scale Types แต่ละ pixel ประกอบไปด้วยค่าความสว่างของแต่ละ pixel ซึ่งจะมีระดับความสว่าง 256 ระดับ (0-255) ซึ่งใช้หน่วยความจำขนาด 8 บิต ตารางที่ 3.1 แสดงรายละเอียดของ pixel แต่ละชนิดในกลุ่มนี้

ตารางที่ 3.1 แสดงรายละเอียดของ pixel แต่ละชนิดในกลุ่ม Gray Scale Pixel Type

Name	Component Size	Data type	Pixel Size	Display	ImageSource
CVis(Gray)BytePixel	8 bits	Unsigned char	8 bits	Yes	Yes
CVis(Gray)CharPixel	8 bits	Signed char	8 bits	No	No
CVis(Gray)ShortPixel	16 bits	Signed short	16 bits	No	No
CVis(Gray)UShortPixel	16 bits	Unsigned short	16 bits	Yes	No
CVis(Gray)IntPixel	32 bits	Signed int	32 bits	No	No
CVis(Gray)LongPixel	32 bits	Signed long	32 bits	No	No
CVis(Gray)UIntPixel	32 bits	Unsigned int	32 bits	Yes	Yes
CVis(Gray)ULongPixel	32 bits	Unsigned long	32 bits	Yes	Yes
CVis(Gray)FloatPixel	32 bits	Float	32 bits	No	No
CVis(Gray)DoublePixel	64 bits	Double	64 bits	No	No

- RGBA Color Pixel Types จะประกอบไปด้วย component ย่อยๆ ภายในแต่ละ pixel จำนวน 4 ค่าได้แก่ ค่าความสว่างในย่านสีแดง (method R()) ค่าความสว่างในย่านสีน้ำเงิน (method B()) ค่าความสว่างในย่านสีเขียว (method G()) และค่าอัลฟ่า (method A()) ซึ่งเป็นค่าที่กำหนดระดับการ transparency ตารางที่ 3.2 แสดงรายละเอียดของแต่ละ pixel ภายในกลุ่ม RGBA Color pixel types โดยลำดับในการเก็บคือ B-G-R-A ซึ่งตรงกับค่า RGBQUAD ซึ่งใช้ใน 32-bits Windows Bitmaps

ตารางที่ 3.2 แสดงรายละเอียดของ pixel แต่ละชนิดในกลุ่ม RGBA Color Pixel Types

Name	Component Size	Data type	Pixel Size	Display	ImageSource
CVisRGBABytePixel	8 bits	Unsigned char	32 bits	Yes	Yes
CVisRGBACharPixel	8 bits	Signed char	32 bits	No	No
CVisRGBAShortPixel	16 bits	Signed short	64 bits	No	No
CVisRGBAUShortPixel	16 bits	Unsigned short	64 bits	No	No
CVisRGBAIntPixel	32 bits	Signed int	128 bits	No	No
CVisRGBALongPixel	32 bits	Signed long	128 bits	No	No
CVisRGBAUIntPixel	32 bits	Unsigned int	128 bits	No	No
CVisRGBAULongPixel	32 bits	Unsigned long	128 bits	No	No
CVisRGBAFloatPixel	32 bits	Float	128 bits	No	No
CVisRGBADoublePixel	64 bits	Double	256 bits	No	No

- YUVA Color Pixel Types คล้ายกลับกลุ่ม RGBA Color Pixel Types ที่ได้กล่าวมาแล้ว เพียงแต่ค่าความสว่างจัดเก็บในแบบ YUV component แทนที่จะเป็น RGB เท่านั้น ตารางที่ 3.3 แสดงรายละเอียดของ pixel แต่ละชนิดในกลุ่ม YUVA Color Pixel Types

ตารางที่ 3.3 แสดงรายละเอียดของ pixel แต่ละชนิดในกลุ่ม YUVA Color Pixel Types

Name	Component Size	Data type	Pixel Size	Display	ImageSource
CVisYUVABytePixel	8 bits	(Un)signed char	32 bits	Yes	Yes
CVisYUVACharPixel	8 bits	(Un)signed char	32 bits	Yes	Yes
CVisYUVAShortPixel	16 bits	(Un)signed short	64 bits	No	Yes
CVisYUVAIntPixel	32 bits	(Un)signed int	128 bits	No	No
CVisYUVALongPixel	32 bits	(Un)signed long	128 bits	No	No
CVisYUVAULongPixel	32 bits	(Un)signed long	128 bits	No	No
CVisYUVAUIntPixel	32 bits	(Un)signed int	128 bits	No	No
CVisYUVAFloatPixel	32 bits	Float	128 bits	No	No
CVisYUVADoublePixel	64 bits	Double	256 bits	No	No

3.2.1.2 การสร้าง CVisImage Object

การสร้าง CVisImage Object สามารถระบุขนาดของภาพได้โดยใช้ CVisImage constructor ในการกำหนดขนาดคอลัมน์ และแถวของภาพได้ เช่น

```
CVisByteImage image(100, 50);
```

คำสั่งข้างต้นเป็นการกำหนดขนาดของ Object ให้มีขนาด 100 คอลัมน์ และ 50 แถว หรือ กว้าง 100 pixels และสูง 50 pixels แต่ผู้เขียนโปรแกรมสามารถสร้าง object ก่อนแล้วระบุขนาดในภายหลังโดยใช้ method Allocate เช่น

```
CVisByteImage image;
image.Allocate(100, 50);
```

จากที่ได้กล่าวมาแล้วข้างต้นว่า CVisImage จะทำการเก็บ pointer ที่อ้างอิงถึงตำแหน่งของหน่วยความจำที่เก็บข้อมูลภาพจริง ในบางครั้งผู้เขียนโปรแกรมต้องการเก็บข้อมูลภาพชุดนั้นไปไว้ที่ตำแหน่งแยกกันต่างหาก method Copy() จะเป็นการสร้างหน่วยความจำที่ตำแหน่งใหม่และมีการสร้างชุดข้อมูลใหม่สำหรับเก็บข้อมูลเดิม โดยที่การแก้ไข การ process ที่ข้อมูลชุดนี้จะไม่มีผลกระทบกับข้อมูลต้นฉบับแต่อย่างใด แต่ method Copy() จะทำงานได้เฉพาะกับข้อมูลในชนิดเดียวกัน แต่ถ้าเป็นกรณี เช่น เป็น pixel type ต่างชนิดกันต้องใช้คำสั่งอื่น คือ method CopyPixelTo() ดังตัวอย่างต่อไปนี้

```
// assign imageFaceOriginal to the subimage in the original data.
CVisByteImage imageFaceOriginal = imageOriginal.SubImage(rectFace);

// fill imageFaceCopy with a new image data block containing just the subimage
CVisByteImage imageFaceCopy.Copy(imageFaceOriginal);

// convert and copy the data into an RGBA version of the face subimage
CVisRGBAByteImage imageFaceRGBA(rectFace);
imageFaceOriginal.CopyPixelsTo(imageFaceRGBA);
```

จากตัวอย่างข้างต้นมีการใช้ method Copy() และ method CopyPixelTo() จะเห็นได้ว่า Copy() สามารถใช้ได้เฉพาะ pixel ในแบบเดียวกัน ส่วน CopyPixelTo() สามารถใช้เปลี่ยนแปลง

ข้ามกันไปมาได้ซึ่งคุณลักษณะนี้มีประโยชน์มากในการทำขบวนการทาง image processing อย่าง การทำ binary threshold เป็นต้น

แต่ method CopyPixelTo() ไม่สามารถเปลี่ยน pixel จากกลุ่ม RGBA หรือ YUVA ไปเป็นกลุ่ม grayscale ได้ หรือแม้แต่ RGBA ไปเป็น YUVA ได้ แต่ MS Vision SDK ก็มีฟังก์ชั่นที่ใช้ในการแปลงค่าจาก RGBA ไปเป็น grayscale ได้คือ VisBrightnessFromRGBA ซึ่งมีสูตรในการคำนวณคือ

G = 0.299R + 0.587G + 0.114B

โดยที่ G คือ ค่าความสว่างในแถบ grayscale

R คือ ค่าความสว่างในย่านสีแดง

G คือ ค่าความสว่างในย่านสีเขียว

B คือ ค่าความสว่างในย่านสีน้ำเงิน

3.2.1.3 Operations พื้นฐานของคลาส CVisImage

method Pixel(x, y) เป็น method ในการเข้าถึงค่าความสว่างของแต่ละ pixel ในข้อมูลภาพทุกๆ ชนิด แต่ในบางครั้งการเขียนโปรแกรมต้องมีการเข้าถึงทุกๆ pixel ในภาพ method RowPointer(row) จึงมีความสะดวก และรวดเร็วกว่าในการทำงาน ดังตัวอย่างต่อไปนี้

```
float FindAverageGray(CVisFloatImage image)
{
        assert(image.NBands() == 1);
        float fltTotalIntensity = 0;
        for (int y = image.Top(); y < image.Bottom();y++)
        {
                float *pflt = image.RowPointer(y);
                for (int x = image.Left(); x < image.Right();x++)
                fltTotalIntensity += pflt[x];
        }
        return (fltTotalIntensity / image.NPoints());
}
```

จากตัวอย่างข้างต้น pflt เป็น pointer ที่รับค่าตำแหน่งของแถวของข้อมูลภาพที่แต่ละแถว เวลาต้องการเข้าถึงค่าในแต่ละ pixel จริงก็สามารถสั่งให้ pflt เลื่อนไปทีละตำแหน่งจนสุดแนวแกน x ได้

คลาส CVisByteImage และ CVisRGBABtyeImage มีความคล้ายกับ Windows Bitmaps เพราะฉะนั้น คุณสมบัติพื้นฐานบางอย่างจึงสามารถใช้ร่วมกันได้ อย่างเช่น การใช้ Windows GDI functions ในการแก้ไข การเสริมแต่งภาพได้โดยตรง การเรียกใช้ Windows GDI functions นั้นต้องมีการใช้ method Hdc เพื่อให้ได้ object ของคลาส HDC ซึ่งสามารถเรียกใช้ Windows GDI functions ได้ดังตัวอย่างต่อไปนี้

```
CVisRGBAByteImage image(100, 100);
HDC hdcImage = image.Hdc();
if (hdcImage != 0)
{
        // Note that we use (0, 0) to refer to the top-left corner of the
        // image when working with Windows GDI functions.
        RECT rect;
        rect.left = rect.top = 0;
        rect.right = image.Width();
        rect.bottom = image.Height();

        HBRUSH hBrush = CreateSolidBrush((COLORREF) 0xff0000);
        if (hBrush != 0)
        {
                FillRect(hdcImage, &rect, hBrush);
                DeleteObject(hBrush);
        }

        DrawText(hdcImage, "Hello World!", - 1, &rect,
                        DT_CENTER | DT_SINGLELINE | DT_VCENTER);
        image.DestroyHdc();
}
```

จากตัวอย่างข้างต้นเป็นการสร้าง object ของคลาส CVisRGBAByteImage ขนาดความกว้าง 100 คอลัมน์ และสูง 100 แถว (100 x 100 pixels) โดยให้สีพื้นภายในเป็นสีน้ำเงิน และเขียนตัวอักษร "Hello World!" ไว้ตรงกลาง สังเกตว่ามีการใช้ Windows GDI functions เช่น DrawText() โดยต้องมีการสร้าง object ในคลาส HDC โดยรับค่าจาก method Hdc()

3.2.2 การแสดงข้อมูลภาพออกจากจอภาพคอมพิวเตอร์

เมื่อมีการสร้าง object ในคลาส CVisImage และมีการ process ข้อมูลภายในแล้ว การจะแสดงภาพที่ได้สามารถทำได้ ดังนี้

3.2.2.1 แสดงผลโดยใช้ Windows HDC

ถ้า object ที่สร้างจากคลาส CVisImage มีคุณสมบัติเหมือนใน Windows bitmaps ผู้เขียนโปรแกรมสามารถใช้ method DisplayInHdc() สำหรับการแสดงภาพบน Windows ได้ โดยจะต้องมีการสร้าง object ของคลาส HDC จากนั้นผ่านค่า window's HWND ไปให้กับฟังก์ชั่น GetDC() โดยผู้เขียนโปรแกรมสามารถตรวจสอบผลลัพธ์ที่ได้ หากมีชนิดของข้อมูลภาพแตกต่างออกไป ผู้เขียนโปรแกรมสามารถใช้ method CopyPixelTo() สำหรับการเปลี่ยนแปลงชนิดของข้อมูลภาพได้

ชนิดของข้อมูลภาพที่ DisplayInHdc สนับสนุน

- CVisRGBAByteImage
- CvisByteImage
- CVisUShortImage (Displayed as RGB555)
- CvisUIntImage
- CVisYUVACharPixel (Displayed as gray scale)

3.2.2.2 แสดงผลโดยใช้คลาส CVisPane และ VisDisplayImage

คลาส CVisPane และ CVisPaneArray กำหนดอยู่ใน VisDisplay.h ผู้เขียนโปรแกรมสามารถ include VisDisplay.h และสามารถสร้าง object จากคลาส CVisPane และ CVisPaneArray โดยสามารถนำ object ที่สร้างจากคลาส CVisImage มาแสดงผลได้

3.2.3 การจัดการติดต่อกับอุปกรณ์ต่อพ่วง

Vision SDK สามารถติดต่อกับอุปกรณ์เชื่อมต่อจากภายนอกได้โดยที่ไม่ขึ้นกับอุปกรณ์ โดย MS Vision SDK ได้เตรียม VisImSrc DLL ไว้ให้ โดยผู้ใช้โปรแกรมสามารถเลือกอุปกรณ์เชื่อมต่อในขณะที่กำลังต่ออยู่กับคอมพิวเตอร์ได้

คลาส CVisImageSource ใน VisImSrc Project โดยสามารถใช้ฟังก์ชั่น VisFindImageSource() ซึ่ง return เป็น instance ของคลาส CVisImageSource ดังตัวอย่าง

CVisImageSource ImSrc = VisFindImageSource("");

จากตัวอย่างข้างต้น ImSrc จะทำการติดต่อกับอุปกรณ์เชื่อมต่อโดยจะเก็บข้อมูลภาพในลักษณะข้อมูลดิบเพื่อรอการ process ต่อไป ในกรณีต้องการติดต่อกับอุปกรณ์ที่เก็บภาพต่อเนื่อง เช่น กล้องถ่ายวิดีโอ สามารถใช้ method SetUseContinuousGrab() ได้

คลาส CVisSequence เป็นคลาสที่มีการใช้มากในการจัดการเกี่ยวกับ process ภาพต่อเนื่อง รวมถึงการอ่าน และการเขียนข้อมูลภาพที่เป็นภาพต่อเนื่อง เช่น AVI file คลาส CVisSequence อยู่ใน VisCore Project เหมือนกับคลาส CVisImage และมีลักษณะเป็น template เช่นเดียวกัน เพราะฉะนั้นจึงสามารถทำงานกับ data type ในชนิดต่างๆ กันได้ การทำงานของคลาส CVisSequence จะทำงานกับข้อมูลภาพที่มีลักษณะเป็นชุดโดยในแต่ละชุดจะต้องเก็บข้อมูลในประเภทเดียวกัน คลาส CVisSequence ใช้ Standard Template library deque class ในการจัดเก็บกลุ่มของข้อมูลภาพ

deque เป็นโครงสร้างข้อมูลที่เก็บ objects ไว้ภายใน และสามารถเข้าถึงข้อมูลโดยการใช้ดัชนี (index) สามารถเพิ่ม และสามารถลบข้อมูลออกจากตำแหน่งหัว หรือท้ายของ deque ได้ คุณสมบัติคล้ายในโครงสร้างข้อมูลแบบคิว

การใช้คลาส CVisSequence สามารถกำหนดขนาดของจำนวน objects ที่จะบรรจุไว้ใน deque ได้ โดยค่า default คือค่า 0 ซึ่งเวลาดึง object ไปใช้จะได้ object เป็นชุดล่าสุดที่เข้ามาใน deque การกำหนดว่าจะใช้ buffer เท่าไหร่ขึ้นกับชนิดของการทำงาน ตัวอย่างเช่น ถ้าต้องการ capture ภาพที่ได้สดๆ มาแสดงบนจอภาพคอมพิวเตอร์ควรตั้งไว้ในขนาดที่ไม่มาก คือ ประมาณ 0 ถึง 5 แต่ถ้าต้องการเก็บภาพจากส่วนอื่นของโปรแกรมไว้ก่อนทำการ process ก็ควรตั้งค่า buffer ขนาดปานกลางคือ ประมาณ 30 ถึง 200 และการตั้งค่า buffer ขนาดสูงคือ ประมาณมากกว่า 200 สำหรับการเก็บภาพแต่ละเฟรมเพื่อนำมาสร้างเป็นไฟล์วิดีโอ สำหรับ method ในการตั้งค่าขนาดของ buffer ของคลาส CVisSequence คือ SetLenghtMax() และ method อื่นๆ เช่น ReadStream(), InsertStream(), AppendStream() และ WriteStream() ในการจัดการเกี่ยวกับไฟล์ AVI ด้วย

การนำเอา object แต่ละ object ที่อยู่ในคลาส CVisSequence มาทำการ process สามารถใช้ method Pop() โดยผู้เขียนสามารถสร้าง object ในคลาส CVisImage มารับไป process ต่ออีกทางหนึ่ง ดังตัวอย่างต่อไปนี้

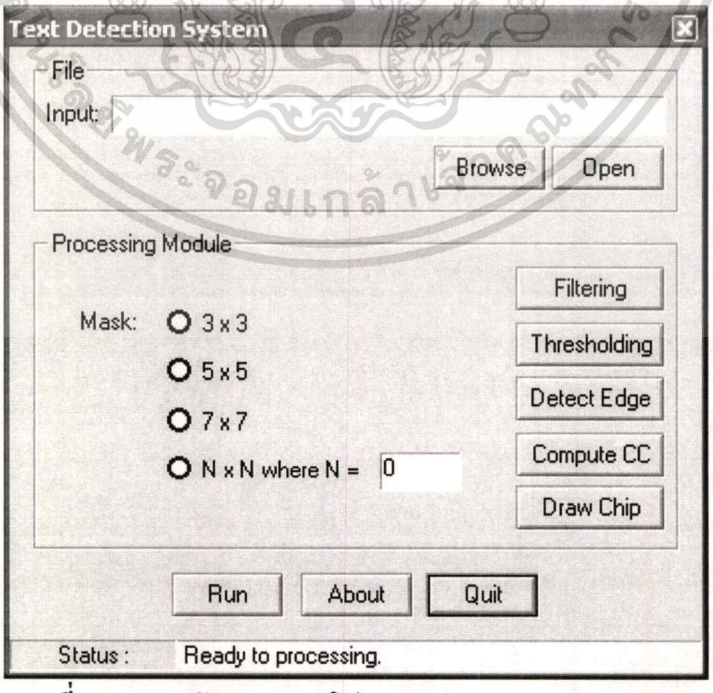
```
CVisImageSource imagesource = VisFindImageSource("");
if(imagesource.IsValid()) {
        CVisSequence<CVisRGBABytePixel> sequence;
        Sequence.ConnectToSource(imagesource, true, false);

        CVisRGBAByteImage imageT;
        if(sequence.Pop(image, 2000) {
                imageT.FWriteFile("out.bmp");
        }
        sequence.DisconnectFromSource();
}
```

จากตัวอย่างโปรแกรมข้างต้นมีการใช้ method ConnectToSource() เพื่อการเชื่อมต่อ object ในคลาส CVisSequence กับ Image Source จากนั้นจึงมีการใช้ method Pop() สำหรับการนำข้อมูลภาพภายใน deque ของ CVisSequence มาเก็บไว้ใน object ของคลาส CVisRGBAByteImage เพื่อทำการ process ต่อไป

3.3 ลักษณะของโปรแกรม Text Detection System

ระบบตรวจจับตัวอักษรจากภาพถ่ายตามธรรมชาติ (Text Detection System from Natural Scenes) นั้นจากที่ได้กล่าวไปแล้วข้างต้นคือ ประกอบไปด้วยขั้นตอนต่างๆ หลายขั้นตอน เริ่มตั้งแต่นำภาพมาเข้าสู่กระบวนการ Filtering โดยในที่นี้จะใช้ Gaussian Filter ในการลดสัญญาณรบกวน (Noise) ออกไปจากภาพให้ได้มากที่สุด หลักจากนั้นจะเป็นการแปลงภาพให้เป็น 2 ระดับ เพื่อทำให้การประมวลผลในขั้นถัดไป สามารถทำได้ง่ายขึ้น ถัดมาจะเป็นการหาเส้นขอบโดยจะเป็นการใช้ 2^{nd} Derivative Edge Detector หลังจากนั้นจะนำเอาเทคนิค Connected Component มาใช้เพื่อแยกแยะองค์ประกอบในภาพ สุดท้ายเมื่อเราสามารถแยกภาพออกเป็นแต่ละองค์ประกอบ (Component) ได้แล้วนั้น จะเป็นการนำเอาแต่ละองค์ประกอบมาวิเคราะห์เพื่อที่จะพิจารณาว่าเป็นบริเวณที่เป็นตัวอักษรหรือไม่ หากเป็นบริเวณที่เป็นตัวอักษร ก็จะทำการตีกรอบล้อมรอบบริเวณดังกล่าว

ระบบตรวจจับตัวอักษรจากภาพถ่ายตามธรรมชาติเป็นโปรแกรมที่ทำงานบนระบบปฏิบัติการวินโดวส์ ซึ่งตัวระบบตรวจจับตัวอักษรนั้นได้ใช้เครื่องมือในการพัฒนาระบบขึ้นด้วยโปรแกรม Microsoft Visual C++ ร่วมกับ Microsoft Vision SDK ซึ่งเป็นตัวช่วยในการพัฒนาโปรแกรมทางด้านการประมวลผลภาพ (Image Processing) โดยตัวระบบตรวจจับตัวอักษรนั้นเป็นโปรแกรมที่เป็นลักษณะ GUI (Graphic User Interface) ดังแสดงในภาพที่ 3.4



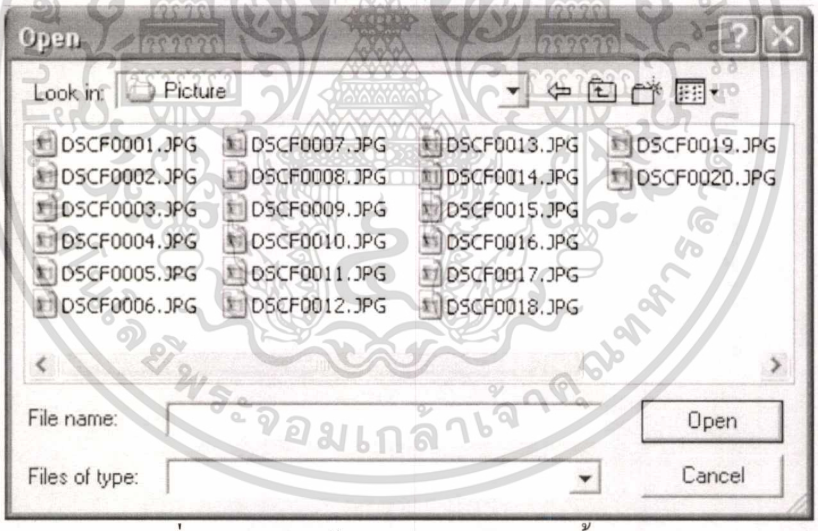
ภาพที่ 3.4 แสดงลักษณะของโปรแกรม Text Detection System

จากภาพที่ 3.4 จะเห็นว่าโปรแกรม Text Detection System นั้นจะประกอบไปด้วย 3 ส่วนหลักๆ ดังนี้

- File
- Processing Module
- Status

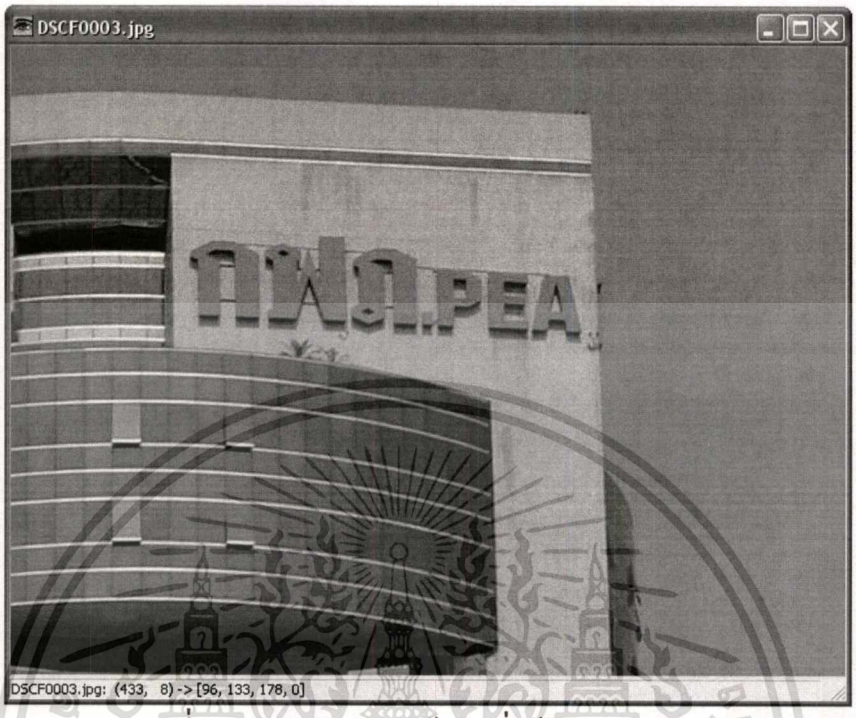
โดยส่วนประกอบหลักของโปรแกรม Text Detection System นั้นจะมีรายละเอียดต่างๆ ที่จะกล่าวต่อไปดังต่อไปนี้

ในส่วนของ "File" นั้นจะเป็นส่วนที่มีไว้สำหรับจัดการในเรื่องของการเลือกรูปภาพเพื่อที่จะเปิดขึ้นมาประมวลผลต่อไป โดยผู้ใช้งานสามารถที่จะเลือกภาพได้โดยการกดปุ่ม Browse ซึ่งเมื่อกดแล้วจะปรากฏหน้าต่างสำหรับเลือกภาพอินพุตที่ต้องการ โดยในภาพที่ 3.5 จะแสดงลักษณะหน้าต่างสำหรับเปิดภาพขึ้นมาแสดงเมื่อผู้ใช้กดปุ่ม Browse



ภาพที่ 3.5 แสดงหน้าต่างสำหรับเปิดภาพขึ้นมาแสดง

เมื่อผู้ใช้ทำการเลือกไฟล์ภาพที่ต้องการเสร็จแล้ว ให้กดปุ่ม Open เมื่อกดเสร็จ โปรแกรมจะกลับมาที่หน้าจอหลักของตัวระบบตรวจจับตัวอักษร หลังจากนั้นให้ผู้ใช้ทำการกดปุ่ม Open ที่ตัวโปรแกรมอีกครั้งเพื่อทำการเปิดภาพนั้นขึ้นมาแสดงบนหน้าจอ โดยที่รูปภาพนั้นจะถูกแสดงขึ้นมาในหน้าต่างใหม่ ดังแสดงในภาพที่ 3.6



ภาพที่ 3.6 แสดงตัวอย่างหน้าต่างที่ใช้ในการแสดงภาพ

ในส่วนถัดมาก็คือส่วนของ "Processing Module" โดยที่ส่วนนี้จะเป็นส่วนที่ใช้ในการประมวลผลภาพทั้งหมดที่ใช้ในระบบตรวจจับตัวอักษร ซึ่งจะเห็นว่าในส่วน Processing Module นี้ประกอบไปด้วยฟังก์ชันที่ใช้ในการทำงานหลายอย่าง ดังนี้

- **Filtering:** เป็นฟังก์ชันที่ใช้ในการลดสัญญาณรบกวน (Noise) ในภาพลงให้เหลือน้อยที่สุด โดยจะเป็นการใช้ Guassian Filter ในการจัดการกับสัญญาณรบกวนในภาพ โดยตัวระบบได้มีขนาดของ Mask ที่จะใช้ได้มา 4 รูปแบบคือ Mask ขนาด 3x3, 5x5, 7x7 และ Custom Mask คือผู้ใช้สามารถกำหนดขนาดของ Mask ได้เองตามต้องการ โดยถ้าหากผู้ใช้ไม่ได้มีการเลือกขนาดของ Mask ตัวโปรแกรมจะกำหนดค่า Default ของขนาด Mask เป็น 0 ซึ่งก็คือ โปรแกรมจะไม่มีการจัดการกับสัญญาณรบกวน
- **Thresholding:** เป็นฟังก์ชันที่ใช้ในการแปลงภาพให้เป็นภาพ 2 ระดับ (Binary Image) เพื่อความง่ายในการประมวลผลในลำดับถัดๆ ไป
- **Detect Edge:** เป็นฟังก์ชันที่ใช้ในการตรวจหาเส้นขอบ โดยที่ผลลัพธ์ที่ได้จากฟังก์ชันการทำงานนี้คือ บริเวณที่เป็นเส้นขอบจะถูกแทนด้วยสีขาว นอกนั้นจะแทนด้วยสีดำทั้งหมด

- **Compute CC:** เป็นฟังก์ชันที่ใช้ในการหาองค์ประกอบที่เชื่อมต่อถึงกัน (Connected Component) เพื่อใช้ในการแยกองค์ประกอบในภาพให้เป็นแต่ละองค์ประกอบ โดยที่ผลลัพธ์ที่ได้จากฟังก์ชันการทำงานนี้คือ แต่ละองค์ประกอบจะถูกแทนด้วยสีที่แตกต่างกันออกไป
- **Draw Chip:** เป็นฟังก์ชันที่ใช้ในการวิเคราะห์ถึงองค์ประกอบที่น่าจะเป็นบริเวณของข้อความ แล้วทำการตีกรอบล้อมรอบบริเวณที่น่าจะเป็นตัวอักษร โดยผลลัพธ์ที่ได้จากฟังก์ชันการทำงานนี้ก็คือ กรอบสี่เหลี่ยมล้อมรอบบริเวณที่เป็นตัวอักษร

ระบบตรวจจับตัวอักษรจะทำงานได้โดย เริ่มต้นเปิดภาพที่ต้องการจะตรวจจับตัวอักษรขึ้นมาแล้วจึงเริ่มดำเนินการตรวจจับ โดยเลือกขนาดของ Mask ที่จะใช้ในการลดสัญญาณรบกวนในภาพ โดยตัวระบบได้มีขนาดของ Mask ให้เลือกใช้อยู่ทั้งหมด 4 แบบ คือขนาด 3x3, 5x5, 7x7 และแบบสุดท้ายเป็นแบบที่ผู้ใช้งานสามารถกำหนดขนาดของ Mask เองได้ (ในลักษณะ NxN) ดังแสดงในภาพที่ 3.4 โดยที่ขนาดของ Mask ยิ่งมีขนาดใหญ่มากเท่าใด ภาพที่เป็นผลลัพธ์จากขั้นตอน Filtering นี้จะยิ่งมีความคมชัดน้อยลงเท่านั้น ทั้งนี้การเลือกใช้ขนาดของ Mask ก็มีความสำคัญเป็นอย่างมากต่อความถูกต้องในการที่จะตรวจจับบริเวณที่เป็นตัวอักษรของระบบ

เมื่อเลือกขนาดของ Mask เสร็จแล้ว ผู้ใช้จะต้องใช้ฟังก์ชัน “Filtering” เพื่อทำการลดสัญญาณรบกวนที่มีอยู่ในภาพ โดยผลลัพธ์ที่ได้จะเป็นภาพเฉดเทา (Grayscale) ที่มีลักษณะเบลอมากขึ้นกว่าเดิมเมื่อเทียบกับภาพต้นฉบับ

หลังจากที่ผ่านขั้นตอนการลดสัญญาณรบกวน (Filtering) ออกไปแล้ว ผู้ใช้จะต้องทำการใช้ฟังก์ชัน “Thresholding” เพื่อที่จะแปลงภาพให้เป็นภาพ 2 ระดับ (Binary Image) หลังจากนั้นจะต้องใช้ฟังก์ชัน “Detect Edge” เพื่อตรวจจับหาบริเวณที่เป็นเส้นขอบบนภาพ 2 ระดับนั้น ซึ่งผลลัพธ์ที่แสดงออกมา ถ้าเป็นสีขาว แสดงว่าบริเวณนั้นเป็นบริเวณที่เป็นเส้นขอบ

เมื่อทำการตรวจจับหาเส้นขอบเสร็จเรียบร้อยแล้ว จะต้องเรียกใช้ฟังก์ชัน “Compute CC” ซึ่งฟังก์ชันนี้จะทำการตรวจสอบภาพที่ผ่านการตรวจจับเส้นขอบแล้วนั้น ว่ามีกี่องค์ประกอบในภาพ โดยจะทำการกำหนดสีให้แต่ละองค์ประกอบเป็นสีที่แตกต่างกันออกไป

สุดท้ายเมื่อสามารถทำการแยกแยะให้เป็นองค์ประกอบที่แตกต่างกันได้แล้วนั้น ผู้ใช้จะต้องทำการเรียกใช้ฟังก์ชัน “Draw Chip” เพื่อใช้ในการสร้างกรอบล้อมรอบบริเวณที่เป็นตัวอักษร โดยที่ในฟังก์ชันการทำงานนี้จะมีการใช้การกรองขนาด (Size Filtering) ในการวิเคราะห์แต่ละองค์ประกอบ เช่น องค์ประกอบที่มีขนาดเล็กมากเกินไป ไม่น่าจะเป็นองค์ประกอบที่เป็นตัวอักษร หรือองค์ประกอบที่มีลักษณะเป็นแนวยาวทั้งในแนวระดับและแนวตั้ง ก็น่าจะเป็น

องค์ประกอบที่ไม่ใช่ตัวอักษร ซึ่งองค์ประกอบเหล่านี้จะถูกกรองออกไปจากภาพ หลังจากนั้นจะทำการวิเคราะห์โครงร่าง (Layout Analysis) เพื่อวิเคราะห์ถึงบริเวณที่น่าจะเป็นบริเวณของข้อความที่เป็นตัวอักษร เสร็จแล้วก็จะทำการสร้างกรอบล้อมรอบบริเวณดังกล่าว

จะเห็นว่าตัวระบบตรวจจับตัวอักษรนั้นมีขั้นตอนการประมวลผลอยู่หลายขั้นตอน ก่อนที่จะสามารถตรวจจับบริเวณที่เป็นส่วนของข้อความที่เป็นตัวอักษรได้ ทางผู้พัฒนาระบบจึงได้มีฟังก์ชันการทำงานพิเศษให้เลือกใช้ ซึ่งก็คือฟังก์ชัน "Run" ซึ่งผู้ใช้งานระบบสามารถที่จะตรวจจับส่วนที่เป็นตัวอักษรในภาพได้อย่างง่ายดาย เพียงแค่ทำการเปิดภาพที่ต้องการจะทดสอบขึ้นมา แล้วทำการเลือกขนาด Mask ตามต้องการ หลังจากเลือกขนาดของ Mask เสร็จ ก็ให้ทำการกดที่ปุ่ม "Run" ได้เลย ระบบจะทำการประมวลผลแล้วแสดงผลลัพธ์ออกมาบนหน้าจอให้กับผู้ใช้งานทันที เพียงแต่อาจจะใช้เวลาในการประมวลผล 3-4 วินาทีโดยประมาณในการประมวลผล

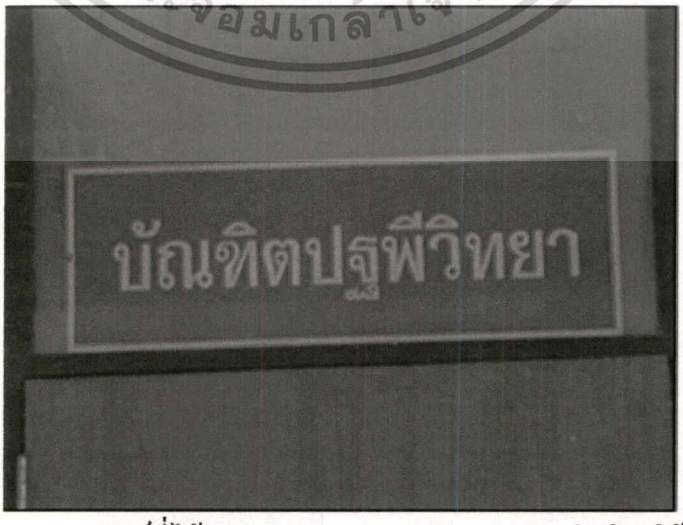
ในส่วนสุดท้ายของโปรแกรม Text Detection System คือส่วนของ "Status" โดยส่วนนี้จะเป็นส่วนที่ใช้แสดงสถานะของระบบ ซึ่งเมื่อผู้ใช้งานเริ่มต้นใช้งานระบบ Text Detection System ตรงส่วนของ Status จะแสดงสถานะเป็น "Ready to processing" ซึ่งหมายถึงระบบพร้อมที่จะประมวลผลตรวจจับตัวอักษรเป็นต้น

3.4 ขั้นตอนการทำงานของโปรแกรม Text Detection System

ขั้นตอนการทำงานของโปรแกรม Text Detection System เริ่มต้นด้วยการเปิดไฟล์ภาพที่ต้องการจะทดสอบขึ้นมา โดยตัวระบบสนับสนุนไฟล์ภาพทั้งที่อยู่ในรูปแบบ BMP และ JPG ซึ่งจะแสดงภาพตัวอย่างที่ใช้ในการทดสอบระบบ ดังแสดงในภาพที่ 3.7

ภาพที่ 3.7 แสดงภาพตัวอย่างที่ใช้ในการทดสอบระบบ

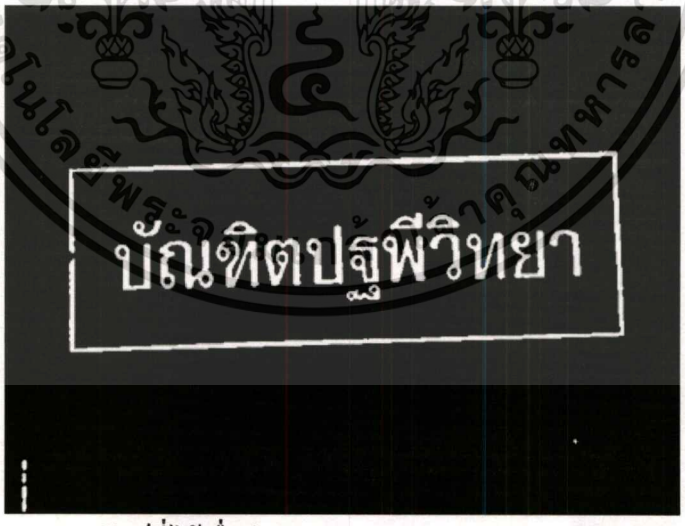
จากภาพที่ 3.7 จะเห็นว่าภาพตัวอย่างที่นำมาใช้ในการทดสอบระบบนั้นเป็นภาพของป้ายที่แสดงคำว่า "บัณฑิตปฐพีวิทยา" ซึ่งติดตั้งบริเวณเหนือบานประตูห้อง โดยที่ผลลัพธ์ที่ต้องการจะให้ระบบแสดงออกมา ก็คือการสร้างกรอบล้อมรอบคำว่าบัณฑิตปฐพีวิทยานั่นเอง เริ่มต้นการประมวลผลอันดับแรก โดยการเลือกขนาดของ Mask ที่ต้องการ ในที่นี้เลือกขนาดเป็น 5x5 หลังจากนั้นทำการกดที่ปุ่ม "Filtering" จะทำให้ได้ผลลัพธ์ดังแสดงในภาพที่ 3.8



ภาพที่ 3.8 แสดงผลลัพธ์ที่ได้จากการทำ Filtering กับภาพตัวอย่างโดยใช้ Mask 5x5

จากภาพที่ 3.8 จะเห็นว่าผลลัพธ์ที่ได้จากการทำ Filtering กับภาพตัวอย่างที่ใช้ในการทดสอบระบบ คือภาพที่มีลักษณะเบลอมากขึ้น โดยที่การเลือกใช้ขนาดของ Mask มีผลอย่างมากต่อผลลัพธ์ที่ได้ในขั้นตอนนี้ เนื่องจากหากเราใช้ Mask ขนาดใหญ่ขึ้น ก็จะทำให้ผลลัพธ์ที่ได้นั้นมี Noise เหลืออยู่ในจำนวนที่น้อยลงเมื่อเทียบกับการที่เราเลือกใช้ Mask ขนาดเล็ก แต่ข้อเสียของการเลือกใช้ Mask ขนาดใหญ่ขึ้นนั้นก็คือว่า จะทำให้รายละเอียดต่างๆ ที่อาจจะเป็นสิ่งที่สำคัญของภาพนั้นเลือนหายไปด้วยเช่นกัน ดังนั้นการที่จะเลือกใช้ขนาดของ Mask จึงเป็นสิ่งที่มีผลอย่างมากต่อการทำงานของระบบในขั้นตอนถัดไป แต่อย่างไรก็ตาม การลดสัญญาณรบกวนในภาพก่อนที่จะนำภาพเข้าสู่การประมวลผลในขั้นตอนถัดๆ ไป จะช่วยทำให้การประมวลผลในขั้นถัดไปทำได้ด้วยความรวดเร็วมากยิ่งขึ้น

หลังจากที่ได้ผ่านขั้นตอน Filtering ซึ่งช่วยลดสัญญาณรบกวน (Noise) ลงไปได้ระดับนึงแล้ว ขั้นตอนถัดมาที่จะต้องกระทำคือขั้นตอนการทำ Thresholding ซึ่งก็คือการแปลงภาพให้เหลือเป็นเพียงแค่ภาพ 2 ระดับ (Binary Image) เนื่องจากว่าระบบที่ใช้ในการตรวจจับตัวอักษรนั้น ไม่มีความจำเป็นจะต้องใช้ภาพสีหรือแม้แต่ภาพเฉดเทา (Grayscale Image) โดยที่ภาพที่มีเพียงแค่ 2 ระดับคือขาวกับดำ ก็เพียงพอต่อการทำงานของระบบแล้ว รวมทั้งยังทำให้การประมวลต่างๆ สามารถทำได้รวดเร็วและง่ายยิ่งขึ้น โดยที่ภาพที่ 3.9 เป็นภาพผลลัพธ์เมื่อผ่านขั้นตอนของการทำ Thresholding ซึ่งจะเห็นว่าภาพผลลัพธ์จะมีเพียงแค่ 2 ระดับ คือไม่เป็นสีขาวก็เป็นสีดำ



ภาพที่ 3.9 แสดงผลลัพธ์ที่ได้เมื่อผ่านกระบวนการทำ Thresholding กับภาพตัวอย่าง

จากภาพที่ 3.9 จะเห็นว่า แม้เราจะนำภาพเข้าผ่านกระบวนการ Filtering เพื่อทำการลดสัญญาณรบกวนออกไปแล้วก็ตาม แต่อย่างไรก็ตามจะเห็นว่าก็ยังคงมีเหลืออยู่บ้าง แต่ว่าสัญญาณรบกวนเหล่านี้จะถูกกรองอีกครั้งหนึ่งในขั้นตอนของการทำ Size Filtering ในขั้นตอนถัดๆ ไป

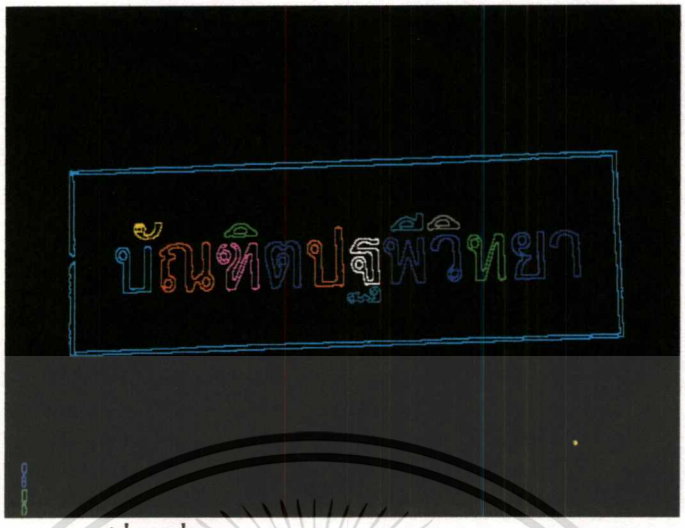
เมื่อผ่านกระบวนการทำ Thresholding เพื่อแปลงภาพให้เหลือเป็นภาพ 2 ระดับเสร็จเรียบร้อยแล้ว ขั้นตอนถัดมาคือการตรวจหาเส้นขอบ (Edge Detection) บนภาพที่ผ่านกระบวนการทำ Thresholding เรียบร้อยแล้ว ซึ่งขั้นตอนของการตรวจหาเส้นขอบนั้น จะทำให้เราสามารถกรองบริเวณที่ไม่เป็นขอบหรือความไม่ต่อเนื่อง (Discontinuity) ออกไป โดยรายละเอียดทางทฤษฎีเกี่ยวกับความไม่ต่อเนื่องนั้นได้อธิบายไว้แล้วในบทที่ 2 หัวข้อ 2.3

หลังจากนำภาพ 2 ระดับ (Binary Image) มาผ่านกระบวนการตรวจหาเส้นขอบ จะทำให้เราได้ผลลัพธ์ดังแสดงที่ภาพที่ 3.10

ภาพที่ 3.10 แสดงผลลัพธ์ที่ได้เมื่อผ่านกระบวนการทำ Edge Detection กับภาพตัวอย่าง

จากภาพที่ 3.10 ซึ่งเป็นผลลัพธ์เมื่อผ่านกระบวนการตรวจหาเส้นขอบ จะเห็นว่าในภาพที่ 3.9 บริเวณใดที่มีการเปลี่ยนแปลงระหว่างสีขาวกับสีดำ (ซึ่งก็คือบริเวณที่เป็นความไม่ต่อเนื่องหรือ Discontinuity นั่นเอง) จะถูกเปลี่ยนเป็นสีขาว ส่วนบริเวณอื่นๆ นอกจากนั้นจะถูกทำการแทนที่ด้วยสีดำทั้งหมด

เราจะพบว่าถ้าเราลองสังเกตที่ภาพที่ผ่านขั้นตอนการตรวจหาเส้นขอบมาแล้วนั้น จะเห็นว่าบริเวณต่างๆ จะถูกแบ่งแยกออกจากกัน เช่นบริเวณที่เป็นตัวอักษรนั้นก็จะแยกออกเป็นแต่ละตัว เพียงแต่ว่าแต่ละบริเวณนั้นๆ ยังคงมีค่าของพิกเซลที่เหมือนกันคือค่า 255 (สีขาว) ด้วยเหตุนี้เราจะต้องทำการแยกแยะแต่ละบริเวณออกจากกันให้เป็นองค์ประกอบเดี่ยวๆ เช่นแยกตัวอักษรออกเป็นตัวเดี่ยวๆ เพื่อที่จะทำให้เราสามารถวิเคราะห์ถึงแต่ละองค์ประกอบนั้นได้ ด้วยเหตุนี้จึงได้มีการนำเอาเทคนิค Connected Component มาใช้ในการแยกแยะองค์ประกอบออกจากกัน ซึ่งเมื่อเรานำภาพมาผ่านกระบวนการของเทคนิคการทำ Connected Component จะได้ผลลัพธ์ดังแสดงในภาพที่ 3.11



ภาพที่ 3.11 แสดงผลลัพธ์ที่ได้เมื่อผ่านเทคนิคการทำ Connected Component กับภาพตัวอย่าง

จากภาพที่ 3.11 เป็นผลลัพธ์ที่ได้จากการประยุกต์ใช้เทคนิค Connected Component เข้าไป โดยหลักการทำงานของขั้นตอนนี้คือ ตรวจหาองค์ประกอบที่เชื่อมต่อถึงกัน แล้วก็ทำการกำหนดค่าที่แตกต่างกันออกไปให้กับแต่ละองค์ประกอบ ซึ่งจากภาพที่ 3.11 จะเห็นว่าตัวอักษรแต่ละตัว, กรอบสี่เหลี่ยมที่ครอบบริเวณข้อความ และบานพับประตูจะถูกแทนที่ด้วยค่าที่แตกต่างกันออกไป ซึ่งจะช่วยให้เราสามารถที่จะวิเคราะห์แต่ละองค์ประกอบในขั้นตอนถัดๆ ไปได้

ขั้นตอนสุดท้ายในกระบวนการประมวลผลของระบบตรวจจับตัวอักษรนั้น จะเป็นการนำเอาแต่ละองค์ประกอบที่ซึ่งเรานั้นสามารถแยกแยะออกจากกันได้ด้วยเทคนิคของการทำ Connected Component ในขั้นตอนที่ผ่านมา แล้วนำเอาแต่ละองค์ประกอบนั้นมาวิเคราะห์ด้วยกฎความสัมพันธ์โครงร่าง (Layout Relation Analysis) โดยผลลัพธ์ที่ได้จะเป็นการตีกรอบล้อมรอบบริเวณที่เป็นส่วนที่เป็นตัวอักษร ดังแสดงในภาพที่ 3.12

ภาพที่ 3.12 แสดงผลลัพธ์ที่ได้จากการตรวจจับบริเวณที่เป็นตัวอักษร

จากภาพที่ 3.12 ซึ่งเป็นผลลัพธ์ที่ได้จากการตรวจจับบริเวณที่เป็นตัวอักษร โดยที่ขั้นตอนก่ารตีกรอบล้อมรอบบริเวณที่เป็นตัวอักษรนั้นจะเป็นการใช้การกรองขนาด (Size Filtering) กับภาพที่ผ่านกระบวนการทางเทคนิค Connected Component โดยขั้นตอนการกรองขนาดนี้ จะเป็นขั้นตอนที่ช่วยในการกรององค์ประกอบที่ถูกแยกออกจากกันแล้วด้วยขั้นตอนก่อนหน้านี้ที่ไม่น่าจะเป็นตัวอักษรออกไป เช่น องค์ประกอบที่มีขนาดเล็กมากๆ รวมถึงองค์ประกอบที่มีลักษณะเป็นแนวยาวทั้งในแนวระดับและแนวตั้ง จะถูกกรองออกไปเป็นต้น ทั้งนี้ยังมีการใช้การวิเคราะห์โครงร่าง (Layout Analysis) ของแต่ละองค์ประกอบเพื่อพิจารณาและวิเคราะห์ว่าองค์ประกอบใดน่าจะเป็นองค์ประกอบที่เป็นตัวอักษร แล้วจึงดำเนินการสร้างกรอบล้อมรอบบริเวณดังกล่าว

บทที่ 4

การทดสอบระบบและผลการทดลอง

4.1 อุปกรณ์ที่ใช้ในการทดสอบ

- เครื่องไมโครคอมพิวเตอร์ AMD AthlonXP 2000+ หน่วยความจำขนาด 256 MB
- ระบบปฏิบัติการ Microsoft Windows XP Professional with Service Pack 1
- โปรแกรม Microsoft Visual C++ 6.0 และ Microsoft Vision SDK 1.2
- กล้องถ่ายภาพดิจิตอลยี่ห้อ Fujifilm รุ่น 2600Zoom ความละเอียด 2.0 ล้านพิกเซล

การทดสอบระบบทำโดยภาพที่ใช้ในการทดสอบระบบนั้นได้มาด้วยการใช้กล้องถ่ายภาพดิจิตอลเก็บภาพเพื่อใช้ในการทดสอบ โดยที่ทำการตั้งค่าของกล้องถ่ายภาพดิจิตอลไว้ที่ความละเอียดระดับ VGA (640x480 pixel) ซึ่งนำภาพเข้าประมวลผลด้วยโปรแกรม Text Detection System ที่พัฒนาขึ้นด้วยโปรแกรม Microsoft Visual C++ 6.0 ร่วมกับ Microsoft Vision SDK 1.2

4.2 การทดสอบระบบ

การทดสอบระบบแบ่งออกเป็น 3 กรณี ดังนี้

- ทดสอบการตรวจจับบริเวณที่เป็นตัวอักษรโดยการใช้ Mask ขนาด 3x3
- ทดสอบการตรวจจับบริเวณที่เป็นตัวอักษรโดยการใช้ Mask ขนาด 5x5
- ทดสอบการตรวจจับบริเวณที่เป็นตัวอักษรโดยการใช้ Mask ขนาด 7x7

โดยในการทดสอบแต่ละกรณีนั้น จะนำเอาภาพที่ใช้เป็นฐานข้อมูลอ้างอิงในการทดสอบระบบทั้งหมด 36 ภาพ มาทำการทดสอบตรวจหาบริเวณที่เป็นตัวอักษร ถ้าหากว่าสามารถตรวจจับบริเวณที่เป็นตัวอักษรได้ครบถ้วน ถึงจะถือว่าการตรวจจับครั้งนั้นถือว่าถูกต้อง แต่ถ้าหากผลลัพธ์ของการตรวจจับตัวอักษรครั้งใดนั้นออกมาในลักษณะที่ว่าระบบทำการตรวจจับได้อย่างไม่ถูกต้อง เช่น ระบบสร้างกรอบล้อมรอบบริเวณที่เป็นตัวอักษรได้ไม่ครบถ้วน รวมถึงกรณีที่ระบบไม่สามารถตรวจจับบริเวณที่เป็นตัวอักษรได้ ซึ่งภาพที่ใช้เป็นฐานข้อมูลอ้างอิงในการทดสอบระบบทั้ง 36 ภาพ จะแสดงดังในภาพที่ 4.1

ภาพที่ 4.1 แสดงภาพที่ใช้เป็นฐานข้อมูลอ้างอิงในการทดสอบระบบ

หลังจากทำการทดสอบด้วยภาพฐานข้อมูลอ้างอิงทั้ง 36 ภาพ จะได้ผลการทดสอบ ดังแสดงในตารางที่ 4.1

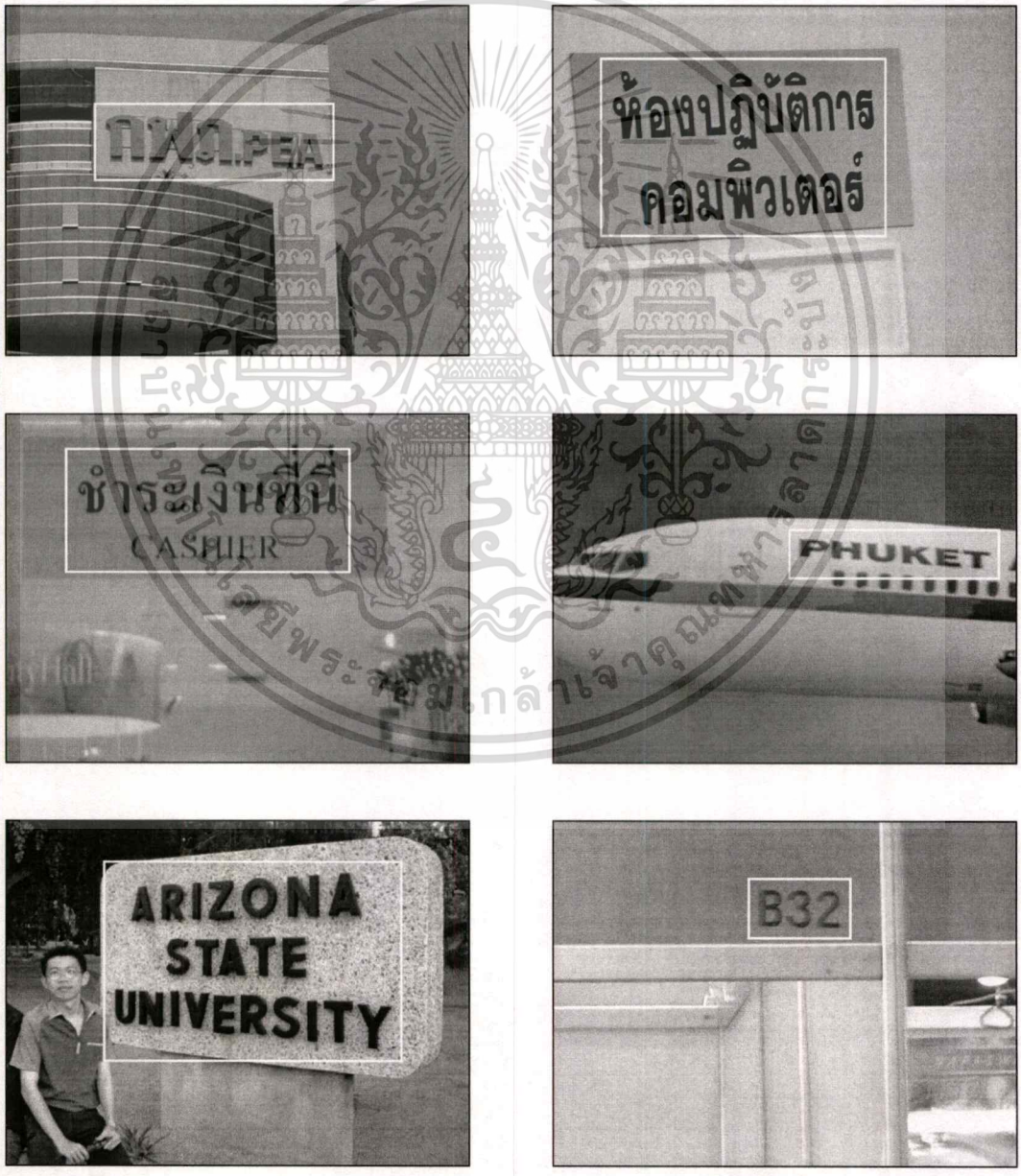
ตารางที่ 4.1 แสดงผลการทดสอบระบบ

ขนาด Mask ที่ใช้	จำนวนภาพ ที่ทำการทดสอบ	จำนวนภาพ ที่ได้ผลลัพธ์ถูกต้อง	คิดเป็นเปอร์เซ็นต์
3x3	36	34	94.44 %
5x5	36	31	86.11 %
7x7	36	30	83.33 %

จากการทดสอบระบบ พบว่าการเลือกใช้ขนาด Mask ที่ใช้ในขั้นตอนของการลดสัญญาณรบกวนในภาพ (กระบวนการ Filtering) มีผลต่อระบบในการตรวจสอบบริเวณที่เป็นตัวอักษร คือ ภาพแต่ละภาพนั้น หากใช้ขนาดของ Mask ที่แตกต่างกันออกไป ก็อาจจะส่งผลทำให้ผลลัพธ์ที่ได้จากระบบนั้นแตกต่างออกไปเช่นกัน ซึ่งเป็นการยากมากที่จะสามารถบอกได้ว่าภาพใดเหมาะกับการเลือกใช้ Mask ขนาดใด เพราะในความเป็นจริงแล้วนั้น ภาพที่ซึ่งเป็นภาพถ่ายตามธรรมชาตินั้นมีปัจจัยมากมายที่มีผลต่อการตรวจจับตัวอักษรของระบบ ไม่ว่าจะเป็นแสงเงาที่ตกกระทบลงไปบนตัวอักษร ความสว่างความมืดของภาพ และอื่นๆ อีกมากมาย ทำให้ยากที่จะสามารถระบุได้ว่าต้องใช้ Mask ขนาดเท่าใดจึงจะเหมาะสมกับภาพนั้นๆ

ปัญหาที่พบเวลาทำการทดลองระบบนั้นมีหลายประการ อาทิเช่น หากบริเวณที่เป็นตัวอักษรนั้นมีลักษณะของสีที่คล้ายคลึงเป็นโทนเดียวกับพื้นหลัง ระบบจะไม่สามารถตรวจจับบริเวณที่เป็นตัวอักษรในลักษณะดังกล่าวได้ เนื่องจากว่าเมื่อผ่านกระบวนการ Thresholding นั้นจะทำให้ทั้งพื้นหลังและส่วนที่เป็นตัวอักษรนั้นถูกกลืนหายไปหมด นอกจากนี้ปัญหาอื่นๆ ที่พบก็เช่น ถ้าหากบริเวณที่เป็นตัวอักษรนั้นมีวัตถุหรือองค์ประกอบอื่นๆ ที่มีขนาดใกล้เคียงไม่แตกต่างจากขนาดของตัวอักษรมากนัก ระบบจะไม่สามารถแยกแยะได้ว่า องค์ประกอบนั้นไม่ใช่ตัวอักษร โดยระบบจะประมวลผลและคิดว่าองค์ประกอบดังกล่าวคือส่วนของตัวอักษร นอกจากนี้ปัญหาอีกประการหนึ่งที่พบก็คือปัญหาที่เกิดจากพื้นฐานของโครงสร้างตัวอักขระภาษาไทย เนื่องจากภาษาไทยนั้นแบ่งการเขียนออกเป็นหลายระดับ ซึ่งในส่วนของปัญหาและข้อจำกัดของระบบนั้น จะกล่าวลงในรายละเอียดอีกครั้งในหัวข้อ 4.3

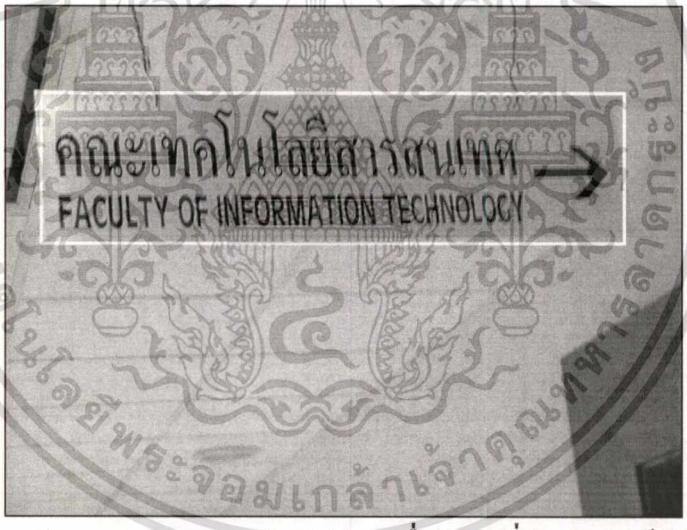
จากตารางที่ 4.1 ซึ่งเป็นผลการทดสอบระบบ จะเห็นว่าเปอร์เซ็นต์ความถูกต้องของระบบลดน้อยลงเมื่อใช้ Mask ขนาดใหญ่ขึ้น ทั้งนี้เมื่อลองมาวิเคราะห์ดูเพื่อหาสาเหตุว่าทำไมถึงเป็นเช่นนั้น เหตุที่เป็นเช่นนั้นก็อาจจะเป็นไปได้ว่า เมื่อเราเลือกใช้ Mask ขนาดใหญ่ขึ้น ทำให้ความชัดเจนของภาพลดน้อยลง (ภาพเบลอมากขึ้น) ทำให้บางครั้งตัวอักษรในภาพอาจจะเลือนจนทับกัน ทำให้ระบบมองเห็นว่าเป็นองค์ประกอบเดียวกัน ซึ่งส่งผลต่อระบบนั่นเอง โดยภาพที่ 4.2 เป็นภาพที่แสดงให้เห็นถึงตัวอย่างผลลัพธ์ที่ถูกต้องที่ได้จากระบบตรวจจับตัวอักษรจากภาพถ่ายตามธรรมชาติ



ภาพที่ 4.2 แสดงตัวอย่างผลลัพธ์ที่ถูกต้องที่ได้จากระบบตรวจจับตัวอักษร

4.3 ปัญหาและข้อจำกัดในการทำงานของระบบ

จากการที่ได้ทดสอบระบบตรวจจับตัวอักษรจากภาพถ่ายตามธรรมชาติแล้วพบว่ามีปัญหาและอุปสรรคหลายประการในการทำงานของระบบ เช่นถ้าหากว่าในภาพที่นำมาทดสอบนั้นมีวัตถุที่มีขนาดไม่แตกต่างจากขนาดของตัวอักษรมากนัก และอยู่ในบริเวณติดกับบริเวณที่เป็นบริเวณของตัวอักษร ระบบจะไม่สามารถที่จะแยกแยะได้ว่าเป็นส่วนที่ไม่ใช่ตัวอักษร เพราะถึงแม้ว่าระบบได้มีการนำเอาวิธีการทางการกรองขนาด (Size Filtering) และการวิเคราะห์ความสัมพันธ์โครงร่าง (Layout Relation Analysis) มาร่วมกันช่วยใช้ในการที่จะวิเคราะห์ถึงความน่าจะเป็นที่จะเป็นตัวอักษร เช่นการใช้อัตราส่วนระหว่างด้าน (Aspect Ratio) รวมทั้งระยะห่างระหว่างองค์ประกอบมาใช้วิเคราะห์แล้วก็ตาม แต่ระบบก็ยังไม่สามารถที่จะแยกแยะได้ โดยระบบจะมองว่าเป็นบริเวณที่น่าจะเป็นตัวอักษรไปด้วย ทำให้เกิดข้อผิดพลาดในการตรวจจับบริเวณที่เป็นตัวอักษรได้ ดังแสดงในภาพที่ 4.3



ภาพที่ 4.3 แสดงปัญหาของการตรวจจับตัวอักษรเมื่อมีวัตถุที่มีขนาดใกล้เคียงกับตัวอักษร

จากภาพที่ 4.3 จะเห็นว่าระบบทำงานได้ไม่ถูกต้องเนื่องจากว่า ผลลัพธ์ที่ควรจะเป็นคือสร้างกรอบล้อมรอบเพียงแค่ส่วนของข้อความ “คณะเทคโนโลยีสารสนเทศ FACULTY OF INFORMATION TECHNOLOGY” เท่านั้น โดยไม่รวมถึงเครื่องหมายลูกศรที่ชี้ไปทางด้านขวา แต่ตัวระบบไม่สามารถที่จะแยกแยะได้ ทำให้ระบบสร้างกรอบล้อมรอบบริเวณทั้งหมดนั่นเอง

ข้อจำกัดของระบบอีกประการหนึ่งที่พบก็คือ ผู้ใช้งานระบบจำเป็นจะต้องเลือกใช้ขนาดของ Mask ที่ใช้ในการประมวลผลในขั้นตอนที่ใช้ในการกรองสัญญาณรบกวน (Noise) ที่เหมาะสมด้วยตัวเอง โดยที่ระบบไม่มีความสามารถในการเลือกขนาดของ Mask ที่เหมาะสมกับแต่ละภาพได้

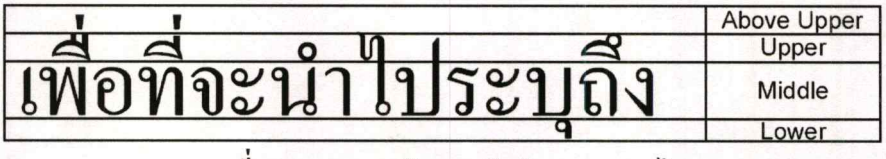
อย่างอัตโนมัติ ซึ่งหากเลือกใช้ขนาดของ Mask ที่แตกต่างกันออกไป อาจจะทำให้ผลลัพธ์ที่ได้จากการประมวลผลออกมาแตกต่างกันออกไป เนื่องจากในภาพถ่ายตามธรรมชาตินั้นมีปัจจัยมากมาย ทำให้ยากที่จะออกแบบให้ระบบสามารถเลือกค่าที่เหมาะสมได้เอง ทำให้การตรวจจับตัวอักษรได้ผลออกมาอาจจะไม่ถูกต้อง ดังแสดงในภาพที่ 4.4

ภาพที่ 4.4 แสดงข้อจำกัดของระบบในการเลือกใช้ขนาด Mask ที่เหมาะสมกับภาพ

จากภาพที่ 4.4 จะเห็นว่าภาพทางซ้ายมือนั้น ผลลัพธ์ที่ได้มาจากระบบนั้นผิดพลาด คือไม่สามารถสร้างกรอบล้อมรอบบริเวณที่เป็นตัวอักษรได้ทั้งหมด แต่ภาพทางขวามือนั้นแสดงผลลัพธ์ออกมาถูกต้อง ทั้งนี้เป็นเพราะการเลือกใช้ขนาดของ Mask ที่แตกต่างกันออกไป ทำให้หลายครั้งจะได้ผลลัพธ์ที่แตกต่างกันออกไป โดยที่จากภาพทางซ้ายมือนั้นเป็นการเลือกใช้ Mask ขนาด 5x5 ส่วนภาพทางขวามือนั้นเป็นการเลือกใช้ Mask ขนาด 7x7

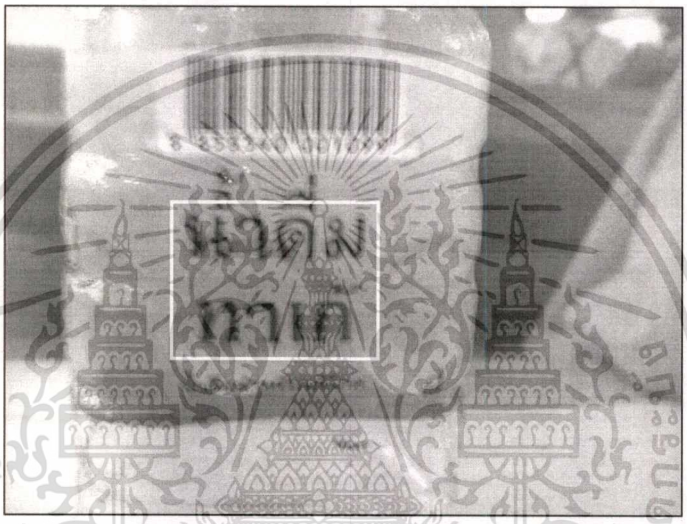
ส่วนปัญหาอีกประการหนึ่งของระบบที่พบนั้นก็คือข้อจำกัดทางด้านภาษา เนื่องด้วยพื้นฐานของโครงสร้างอักขระภาษาไทยเองนั้น ตัวหนังสือภาษาไทยนั้นมีความสลับซับซ้อนมากเมื่อเทียบกับภาษาอังกฤษ เพราะจะสังเกตได้ว่าตัวอักขระภาษาไทยนั้นมีหลายระดับ โดยเราสามารถจำแนกระดับในการเขียนตัวอักขระภาษาไทยออกเป็นทั้งหมด 4 ระดับดังนี้ (แสดงในภาพที่ 4.5)

- ระดับเหนือบน (Above Upper)
- ระดับบน (Upper)
- ระดับกลาง (Middle)
- ระดับล่าง (Lower)



ภาพที่ 4.5 แสดงระดับของตัวอักขระภาษาไทย

จากการที่ได้ทำการทดสอบระบบพบว่าหลายครั้ง ระบบจะมีปัญหาเมื่อข้อความในภาพที่นำมาทดสอบนั้นเป็นข้อความภาษาไทยที่ซึ่งมีตัวอักขระอยู่ในระดับเหนือบน (Above Upper) และระดับล่าง (Lower) หรือบางครั้งอาจจะมีปัญหากับตัวอักขระในระดับบน (Upper) อีกด้วย เนื่องจากว่าอักขระในระดับดังกล่าวนั้นจะมีขนาดค่อนข้างจะเล็กเมื่อเทียบกับอักขระที่ระดับกลาง (Middle) ทำให้หลายครั้งระบบจะทำการกรององค์ประกอบเหล่านั้นออกไป ทำให้ไม่สามารถที่จะตรวจจับอักขระดังกล่าวได้ในบางกรณี ดังแสดงตัวอย่างในภาพที่ 4.6



ภาพที่ 4.6 แสดงปัญหาของการตรวจจับตัวอักษรภาษาไทยในระดับเหนือบน (Above Upper)

จากภาพที่ 4.6 จะพบว่าระบบไม่สามารถที่จะตรวจจับบริเวณที่เป็นตัวอักษรได้ครอบคลุมทั้งหมด เนื่องจากว่า วรรณยุกต์ไม้เอกและวรรณยุกต์ไม้โท ซึ่งเป็นอักขระที่ในภาพนี้ถือว่าอยู่ในระดับเหนือบน (Above Upper) เมื่อระบบทำการประมวลผลภาพดังกล่าว จะทำให้ระบบประมวลผลเป็นว่าองค์ประกอบที่เป็นวรรณยุกต์ไม้เอกและวรรณยุกต์ไม้โทนั้นมีขนาดเล็กเกินไป จึงจัดองค์ประกอบเหล่านี้ให้อยู่ในกลุ่มของสัญญาณรบกวน ทำให้องค์ประกอบดังกล่าวถูกกรองออกไปจนทำให้ระบบไม่สามารถที่จะตรวจจับองค์ประกอบดังกล่าวได้

บทที่ 5
บทสรุปและข้อเสนอแนะ

5.1 บทสรุปโครงการโดยรวม

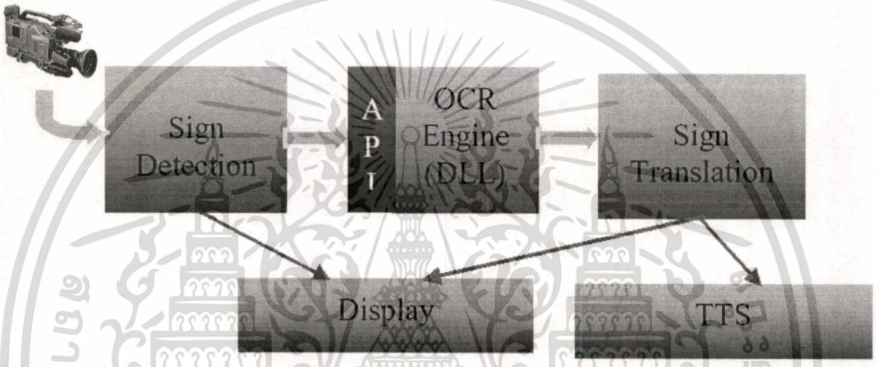
ระบบตรวจจับตัวอักษรจากภาพถ่ายตามธรรมชาติ (Text Detection System from Natural Scenes) เป็นระบบที่ใช้ในการตรวจหาบริเวณที่เป็นข้อความในภาพถ่าย ซึ่งผลลัพธ์ที่ได้จากระบบนั้นเป็นการสร้างกรอบล้อมรอบบริเวณที่น่าจะเป็นข้อความ โดยตัวระบบนั้นประกอบไปด้วยขั้นตอนและกระบวนการทำงานหลายอย่าง ไม่ว่าจะเป็นขั้นตอนการลดสัญญาณรบกวนในภาพ (Image Filtering), การแปลงภาพให้เป็นภาพ 2 ระดับ (Binary Image), การตรวจหาเส้นขอบหรือความไม่ต่อเนื่อง (Edge Detection), การใช้เทคนิคการวิเคราะห์องค์ประกอบที่เชื่อมต่อถึงกัน (Connected Component Analysis) เป็นต้น ซึ่งตัวระบบตรวจจับตัวอักษรจากภาพถ่ายตามธรรมชาติก็ได้มีการนำเอาเทคนิควิธีดังกล่าวมาประยุกต์ใช้สร้างขึ้นมาเป็นระบบ

จากที่ได้ผ่านการทดสอบระบบด้วยภาพที่ใช้ในการทดสอบ ซึ่งได้มาจากฐานข้อมูลอ้างอิงทั้งหมด 36 ภาพ พบว่าระบบมีความแม่นยำในการตรวจจับบริเวณที่เป็นตัวอักษร คิดเป็น 94.44%, 86.11% และ 83.33% ด้วยการใช้ Mask ขนาด 3x3, 5x5 และ 7x7 ตามลำดับ นอกจากนี้ยังพบว่าระบบไม่สามารถที่จะตรวจจับบริเวณที่เป็นตัวอักษรได้อย่างถูกต้อง หากว่าภาพที่นำมาทดลองนั้นมีลักษณะที่ซับซ้อนมากเกินไป โดยผลลัพธ์ที่ได้นั้นอาจจะไม่ถูกต้องโดยระบบจะทำการสร้างกรอบที่ไม่ตรงกับตำแหน่งที่เป็นบริเวณตัวอักษร เนื่องจากว่าระบบจะเกิดความสับสนในขั้นตอนของการวิเคราะห์ความสัมพันธ์โครงร่าง (Layout Relation Analysis) ซึ่งเป็นกระบวนการในการที่จะวิเคราะห์แต่ละองค์ประกอบว่าเป็นองค์ประกอบที่เป็นตัวอักษรหรือไม่นั่นเอง

5.2 ข้อเสนอแนะสำหรับแนวทางในการพัฒนาระบบต่อไป

ระบบตรวจจับตัวอักษรจากภาพถ่ายตามธรรมชาตินั้นถือได้ว่าเป็นระบบที่มีประโยชน์ไม่น้อยเลย เนื่องจากว่าตัวอักษรนั้นเป็นสิ่งที่มีความสำคัญอย่างมากสำหรับใช้สื่อสารกัน ดังนั้นหากเรามีระบบที่มีความสามารถในการที่จะดึงเอาข้อความต่างๆ ที่อยู่ในภาพออกมาก่อให้เกิดคุณประโยชน์ได้ต่างๆ โดยที่เราสามารถนำสิ่งนี้ไปประยุกต์ใช้ นำไปพัฒนาต่อเพื่อสร้างระบบต่างๆ ขึ้นมาใช้ประโยชน์ได้อย่างมากมายหลากหลายประการ

ตัวอย่างโปรแกรมประยุกต์หนึ่งที่เป็นระบบที่มีประโยชน์อย่างมากคือ ระบบรู้จำตัวอักษรบนป้าย (Sign Recognition System) ซึ่งเป็นระบบที่ใช้ตรวจจับตัวอักษรบนป้ายต่างๆ เช่น ป้ายบอกทางจราจร, ป้ายเตือนตามสถานที่ต่างๆ เป็นต้น โดยที่ระบบจะต้องมีการตรวจจับบริเวณที่เป็นตัวอักษรบนป้าย แล้วนำเอาบริเวณดังกล่าวที่ตรวจจับได้นั้น ไปผ่านกระบวนการรู้จำตัวอักษร (Optical Character Recognition) เพื่อที่จะทำการแยกองค์ประกอบ (Segmentation) แล้วทำการวิเคราะห์ว่าตัวอักษรที่อยู่ในภาพนั้นเป็นตัวอักษรใด โดยสามารถแสดงลักษณะการทำงานของระบบรู้จำข้อความบนป้ายในภาพที่ 5.1



ภาพที่ 5.1 แสดงลักษณะการทำงานของระบบรู้จำข้อความบนป้าย (Sign Recognition System)

จากภาพที่ 5.1 จะเห็นว่าการทำงานของระบบนั้น เริ่มต้นจะต้องใช้การตรวจจับบริเวณที่เป็นตัวอักษรจากภาพ แล้วจึงนำเข้าสู่กระบวนการรู้จำตัวอักษรหรือ OCR เพื่อวิเคราะห์ว่าเป็นตัวอักษรใด หลังจากนั้นก็จะทำการแปลความหมายของข้อความดังกล่าว เสร็จแล้วก็สามารถที่จะแสดงผลลัพธ์ออกมาได้ 2 ลักษณะคือ แสดงข้อความที่แปลความหมายเรียบร้อยแล้วออกมาแสดงให้ผู้ใช้ หรือไม่ก็ทำการแปลข้อความแล้วแสดงออกมาเป็นลักษณะของเสียง (Text-to-speech) ซึ่งเป็นคำแปลของข้อความดังกล่าว

เราคงจะปฏิเสธไม่ได้ว่าภาษาเป็นสิ่งที่มีความอย่างมากในการติดต่อสื่อสารกัน หากเราลองคิดดูว่าถ้าเราเป็นนักท่องเที่ยวชาวต่างชาติ เดินทางมาท่องเที่ยวในประเทศไทย แล้วพบป้ายเตือนอะไรสักอย่าง ซึ่งเป็นข้อความภาษาไทย แล้วเราจะเข้าใจข้อความดังกล่าวนั้นได้อย่างไร ถ้าหากเรามีโทรศัพท์เคลื่อนที่แบบพกพา (Cell Phone) ที่มีความสามารถในการตรวจจับตัวอักษรแล้วทำการรู้จำ (Recognition) เสร็จแล้วสามารถที่จะแปลข้อความที่เป็นภาษาท้องถิ่นนั้นๆ ให้เป็นภาษากลางที่ผู้คนใช้กัน เช่นภาษาอังกฤษ ก็ย่อมจะก่อให้เกิดประโยชน์อย่างมาก เพราะในปัจจุบันนี้ เทคโนโลยีก้าวหน้าไปด้วยความรวดเร็วมาก โทรศัพท์เคลื่อนที่แบบพกพาที่มีใช้กันอยู่ทุกวันนี้ก็เริ่มที่จะมี

กล้องถ่ายภาพดิจิตอลติดมาเป็นอุปกรณ์มาตรฐานสำหรับโทรศัพท์เคลื่อนที่แบบพกพา ถึงแม้ว่า ณ ปัจจุบันนี้ความสามารถของกล้องถ่ายภาพดิจิตอลที่ติดมากับโทรศัพท์เคลื่อนที่แบบพกพาจะมีความละเอียดของภาพที่ได้ไม่สูงมากนัก แต่ด้วยความเจริญก้าวหน้าทางเทคโนโลยีนั้น คงจะไม่เป็นสิ่งที่ยากเกินไปที่ในอนาคตอันใกล้นี้เราจะมีโทรศัพท์เคลื่อนที่แบบพกพาที่มีกล้องถ่ายภาพดิจิตอลคุณภาพสูงติดมาเป็นอุปกรณ์มาตรฐานอย่างแน่นอน

ภาคผนวก ก.
รูปแบบและโครงสร้างของบิตแมพ

1. รูปแบบของไฟล์บิตแมพ

ไฟล์บิตแมพใน Windows ถูกเก็บในรูปแบบที่เป็นอิสระจากเครื่องมือ (Device-independent bitmap) หรือ DIB Format ซึ่งยอมให้ Windows แสดงผลไฟล์บิตแมพจากเครื่องมือแสดงผลชนิดใดก็ได้

คำว่า "เป็นอิสระจากเครื่องมือ" (Device independent) หมายความว่า บิตแมพจะเฉพาะเจาะจง สีของพิกเซล ในรูปแบบที่ไม่ขึ้นกับวิธีการโดยใช้การนำเสนอโดยใช้สีมาแสดงผล

สิ่งที่ใช้เป็นนามสกุลของไฟล์ Windows DIB คือ .bmp โดยที่รูปแบบและโครงสร้างของบิตแมพ ที่จะอธิบายดังต่อไปนี้จะใช้ได้กับ ระบบปฏิบัติการ Microsoft Windows เท่านั้น

2. โครงสร้างของไฟล์บิตแมพ

ไฟล์บิตแมพ มี 4 ส่วน โดยประกอบด้วย

- ส่วนหัวของไฟล์ (Bitmap-file Header)
- ส่วนข้อมูลข่าวสารของส่วนหัว (Bitmap-information Header)
- ตารางสี (Color Table)
- อาร์เรย์ซึ่งกำหนดให้เป็นข้อมูลบิตแมพ

โดยที่ไฟล์บิตแมพทั้ง 4 ส่วนนั้น สามารถแสดงได้ดังนี้

```
BITMAPFILEHEADER        bmfh;
BITMAPINFOHEADER        bmih;
RGBQUAD                 aColors[];
BYTE                    aBitmapBits[];
```

2.1 ส่วนหัวของไฟล์ (Bitmap-file Header)

ส่วนหัวของไฟล์จะกำหนดให้เป็นโครงสร้างชื่อ BITMAPFILEHEADER ซึ่งประกอบด้วย

- Type หมายถึง ชนิดของข้อมูล
- Size หมายถึง ขนาดของไฟล์บิตแมพ
- Layout ของไฟล์ที่บรรจุ DIB
- ส่วนที่สงวนไว้ (Reserved) จำนวน 2 word
- Offset ของโครงสร้างของบิตแมพ มีหน่วยเป็นไบต์

ตำแหน่ง Offset ในไฟล์บิตแมพ มีความหมายดังนี้

00000000-00000001	จะแจ้งชนิดของภาพว่าเป็นชนิดใด ถ้าเป็นไฟล์บิตแมพ จะมีค่าเท่ากับ 0x42 และ 0x4D ตามลำดับ
00000002-00000005	จะระบุขนาดของไฟล์ หน่วยเป็นไบต์
00000012-00000015	จะระบุความกว้างของภาพเอาไว้ หน่วยเป็นพิกเซล
00000016-00000019	จะระบุความยาวของภาพเอาไว้ หน่วยเป็นพิกเซล
0000001C	ระบุโหมด ว่าเป็นภาพชนิดใด ถ้าเป็น 256 ระดับ จะมีค่าเท่ากับ 0x08 แต่ถ้าเป็น 16.7 ล้านระดับ จะมีค่าเท่ากับ 0x18
00000432-00000434	เป็นตำแหน่งของตารางสี (ยกเว้นภาพ 16.7 ล้านระดับ) โดยจะเรียง น้ำเงิน เขียว แดง ตามลำดับ ถ้าเป็นภาพระดับสีเทา ทั้งสามไบต์จะเท่ากัน
00000437	เป็นตำแหน่งเริ่มรูปของภาพ 256 ระดับ
00000036	เป็นตำแหน่งเริ่มรูปของภาพ 16.7 ล้านระดับ

โครงสร้าง BITMAPFILEHEADER สามารถแสดงได้ดังนี้

```
typedef struct    tagBITMAPFILEHEADER {    // bmfh
        WORD        bfType;
        DWORD       bfSize;
        WORD        bfReserved1;
```

```
    WORD        bfReserved2;
    DWORD       bfOffBits;
} BITMAPFILEHEADER;
```

2.2 ส่วนข้อมูลข่าวสารของส่วนหัว (Bitmap-information Header)

ส่วนข้อมูลข่าวสารของส่วนหัวจะกำหนดให้เป็นโครงสร้างชื่อ BITMAPINFOHEADER ซึ่งประกอบด้วย

- Size หมายถึง ระบุจำนวนไบต์ที่โครงสร้างต้องการ
- Width หมายถึง ความกว้างของบิตแมพเป็นพิกเซล
- Height หมายถึง ความสูงของบิตแมพเป็นพิกเซล ถ้ามีค่าเป็นบวกจะเป็นแบบ bottom-up จุดเริ่มต้นของข้อมูลจะเริ่มจากมุมซ้ายล่างของภาพ แต่ถ้ามีค่าเป็นลบจะเป็นแบบ top-down จุดเริ่มต้นของข้อมูลจะเริ่มจากมุมซ้ายบนของภาพ
- Plain หมายถึง มิติของรูปภาพ
- BitCount จะระบุจำนวนบิตต่อพิกเซล ค่านี้จะเป็นได้เฉพาะ 1, 4, 8, 16, 24, 32

ถ้ามีค่าเป็น 1	ภาพจะเป็นภาพขาว-ดำ (Monochrome) และในตารางสีจะมีเพียง 2 สี และแต่ละบิตในข้อมูลบิตแมพ จะหมายถึง 1 พิกเซล ถ้าบิตเป็น 0 หรือ bit is clear พิกเซลนั้นจะแสดงด้วยสีแรกของตารางสี ถ้าบิตนั้นเป็น 1 หรือ bit is set ค่าพิกเซลนั้นจะมีสีเป็นสีที่ 2 ของตารางสี
ถ้ามีค่าเป็น 4	บิตแมพจะมีจำนวนสีมากที่สุด 16 สี และจะใช้ 4 บิต เป็นตัวชี้ไปยังตารางสี ตัวอย่างเช่น ถ้าไบต์แรกของบิตแมพ เป็น 0x1F ไบต์นี้จะแทนได้ 2 พิกเซล พิกเซลแรกจะเป็นสีที่ 2 ในตารางสี และพิกเซลที่ 2 จะเป็นสีที่ 16 ในตารางสี
ถ้ามีค่าเป็น 8	บิตแมพจะมีจำนวนสีมากที่สุด 256 สี แต่ละพิกเซลจะแทนด้วยตัวชี้ขนาด 1 ไบต์ ตัวอย่างเช่น ถ้าไบต์แรกของบิตแมพเป็น 0x1F พิกเซลแรกจะมีสีเป็นสีที่ 32 ในตารางสี

ถ้ามีค่าเป็น 24 บิตแมพจะมีจำนวนสีมากที่สุด 2^{24} สี จะไม่ใช้ตารางสี แต่จะใช้จำนวนบิตที่มีอยู่เก็บค่าของสีไว้ ถ้าสมาชิกของ bmiColors (หรือ bmciColors) เป็น NULL และทุกๆ 3 ไบต์ที่เรียงติดในอาร์เรย์ของบิตแมพ จะแสดงผลสีด้วยความสัมพันธ์ของความเข้มระหว่างสีแดง เขียว น้ำเงิน ตามลำดับให้เป็น 1 พิกเซล หรือเป็นการผสมสีแดง เขียว น้ำเงิน นั่นเอง

- Compression จะระบุชนิดของการบีบอัดข้อมูลสำหรับการบีบอัดข้อมูลบิตแมพแบบ bottom-up (การบีบอัดข้อมูลบิตแมพแบบ top-down ไม่สามารถบีบอัดได้) ค่าที่ใช้ในแต่ละชนิดมีดังตารางที่ ก.1

ตารางที่ ก.1 แสดงค่าที่ใช้และชนิดในการบีบอัดข้อมูล

ค่า	คำอธิบาย
BI_RGB	ไม่มีการบีบอัดข้อมูล
BI_RLE8	ใช้การบีบอัดข้อมูลแบบ Run Length Encoding (RLE) ด้วย 8 บิตต่อพิกเซล รูปแบบของการบีบอัดแบบนี้จะมี 2 ไบต์ ซึ่งประกอบด้วย ไบต์ที่แสดงจำนวนตามด้วยไบต์ซึ่งมีดัชนีของสี (Color Index)
BI_RLE4	ใช้การบีบอัดข้อมูลแบบ Run Length Encoding (RLE) ด้วย 8 บิตต่อพิกเซล รูปแบบของการบีบอัดแบบนี้จะมี 2 ไบต์ ซึ่งประกอบด้วย ไบต์ที่แสดงจำนวนตามด้วยดัชนีของสี จำนวน 2 word
BI_BITFIELDS	จะระบุว่าไม่ใช้การบีบอัดข้อมูลแบบใด แต่จะใช้ตารางสีแทนซึ่งประกอบด้วย Red, Green, Blue ซึ่งแต่ละตัวเป็น Doubleword วิธีนี้มักใช้กับ 16 หรือ 32 บิตต่อพิกเซล

- SizeImage ระบุขนาดของรูปภาพเป็นไบต์ และจะมีค่าเป็น 0 เมื่อมีการบีบอัดแบบ BI_RGB
- XPelsPerMeter ระบุความละเอียด (Resolution) ในแนวนอน (Horizontal) ในหน่วยพิกเซลต่อเมตร
- YPelsPerMeter ระบุความละเอียด (Resolution) ในแนวตั้ง (Vertical) ในหน่วยพิกเซลต่อเมตร
- ClrUsed ระบุจำนวนดัชนีสีในตารางสีที่ถูกใช้จริงๆ
- ClrImportant ระบุจำนวนดัชนีสีที่ถูกพิจารณาว่าสำคัญสำหรับการแสดงผลของบิตแมพ ถ้ามีค่าเท่ากับ 0 หมายถึงทุกสีมีความสำคัญหมด

โครงสร้าง BITMAPINFOHEADER สามารถแสดงได้ดังนี้

```
typedef struct    tagBITMAPINFOHEADER {     // bmih
        DWORD       biSize;
        LONG        biWidth;
        LONG        biHeight;
        WORD        biPlanes;
        WORD        biBitCount;
        DWORD       biCompression;
        DWORD       biSizeImage;
        LONG        biXPelsPerMeter;
        LONG        biYPelsPerMeter;
        DWORD       biClrUsed;
        DWORD       biClrImportant;
} BITMAPINFOHEADER;
```

2.3 ตารางสี (Color Table)

จะกำหนดให้เป็นโครงสร้างชื่อ RGBQUAD ซึ่งประกอบด้วย

- Blue เป็นตัวกำหนดความเข้มของโทนสีฟ้า, น้ำเงิน
- Green เป็นตัวกำหนดความเข้มของโทนสีเขียว
- Red เป็นตัวกำหนดความเข้มของโทนสีแดง

โครงสร้างของ RGBQUAD สามารถแสดงได้ดังนี้

```
typedef struct    tagRGBQUAD {  // rgb
        BYTE            rgbBlue;
        BYTE            rgbGreen;
        BYTE            rgbRed;
        BYTE            rgbReserved;
} RGBQUAD;
```

โครงสร้าง BITMAPINFOHEADER และ RGBQUAD สามารถรวมเป็นโครงสร้างใหม่ได้ คือ BITMAPINFO ซึ่งมีโครงสร้างดังนี้

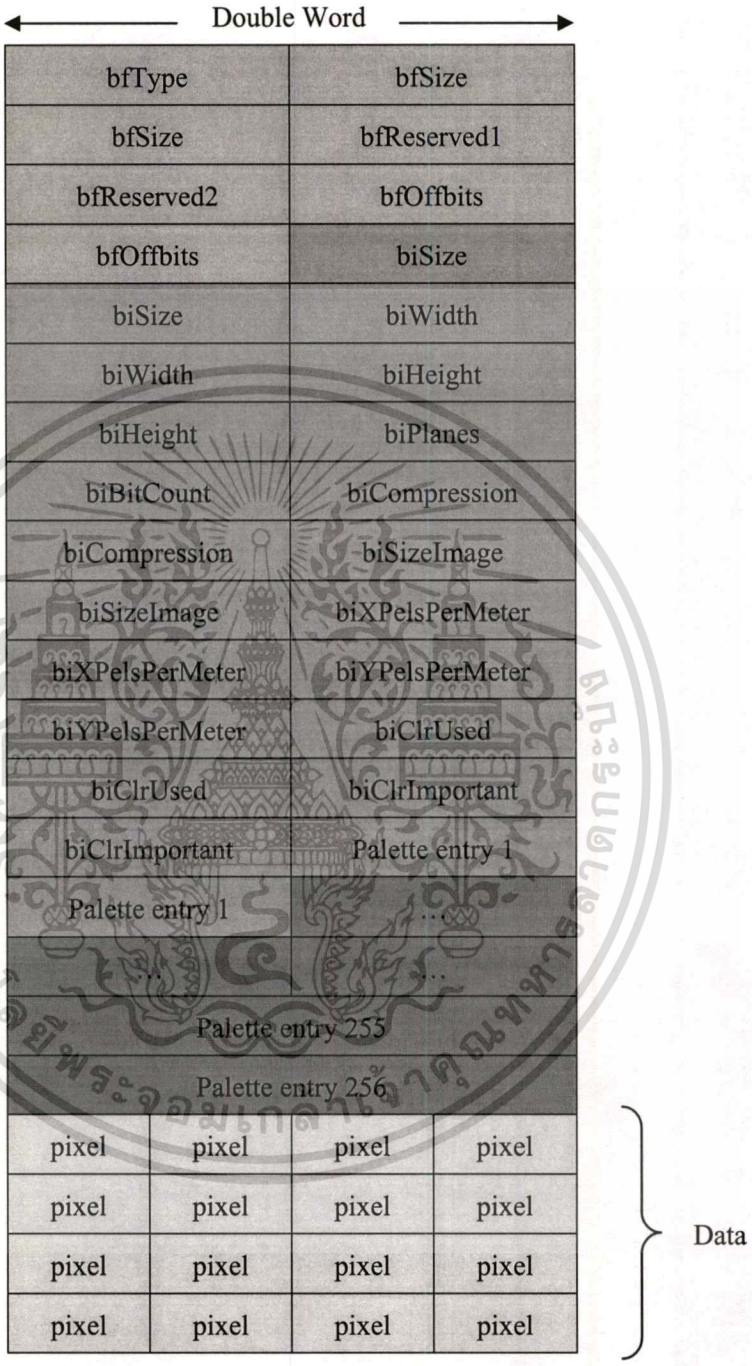
```
typedef struct    tagBITMAPINFO {  // bmi
        BITMAPINFOHEADER        bmiHeader;
        RGBQUAD                 bmicolor[1];
} BITMAPINFO;
```

โดยที่ bmiHeader จะเป็นโครงสร้างของ BITMAPINFOHEADER ซึ่งประกอบข้อมูลมิติและความกว้าง ความยาวของบิตแมพ ดังที่กล่าวมาแล้ว ส่วน bmicolor เป็นอาร์เรย์ของตารางสีที่มีโครงสร้างเป็น RGBQUAD

2.4 อาร์เรย์ซึ่งกำหนดให้เป็นข้อมูลของบิตแมพ

- ไฟล์บิตแมพ 8 บิต ข้อมูลบิตแมพจะเป็นค่าดัชนี อ้างอิงไปที่ตารางสี
- ไฟล์บิตแมพ 24 บิต ข้อมูลบิตแมพ 3 ไบต์ที่ติดกันแทนได้เป็น 1 พิกเซล โดย 3 ไบต์ที่ติดกันนั้น จะเป็นค่า Red, Green, Blue โดยตัวของมันเอง

3. ตัวอย่างโครงสร้างของไฟล์บิตแมพ 256 สี

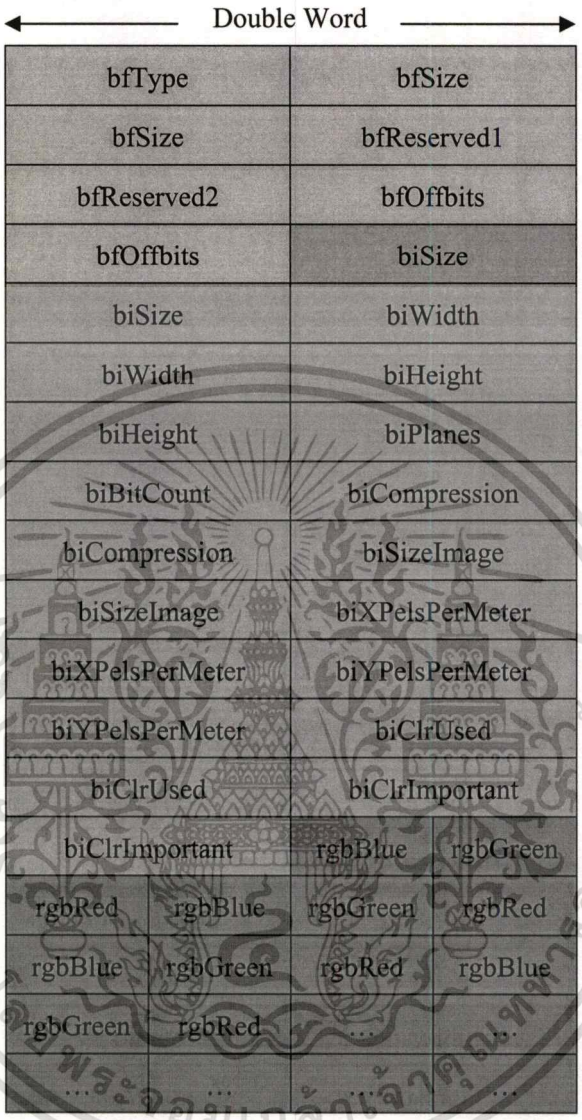


ภาพที่ ก.1 แสดงโครงสร้างของไฟล์บิตแมพ 8 บิต (256 สี)

หมายเหตุ

- เป็นส่วนของ BITMAPFILEHEADER
- เป็นส่วนของ BITMAPINFOHEADER
- เป็นส่วนของตารางสี (Palette) ในที่นี้มี 256 สี โดย 1 Palette entry ประกอบด้วย Red, Green, Blue
- เป็นส่วนข้อมูลบิตแมพ เป็นพิกเซลโดยแต่ละพิกเซลมีขนาด 1 ไบต์

4. ตัวอย่างโครงสร้างของไฟล์บิตแมพ 24 บิต (16.7 ล้านสี)



ภาพที่ ก.2 แสดงโครงสร้างของไฟล์บิตแมพ 8 บิต (256 สี)

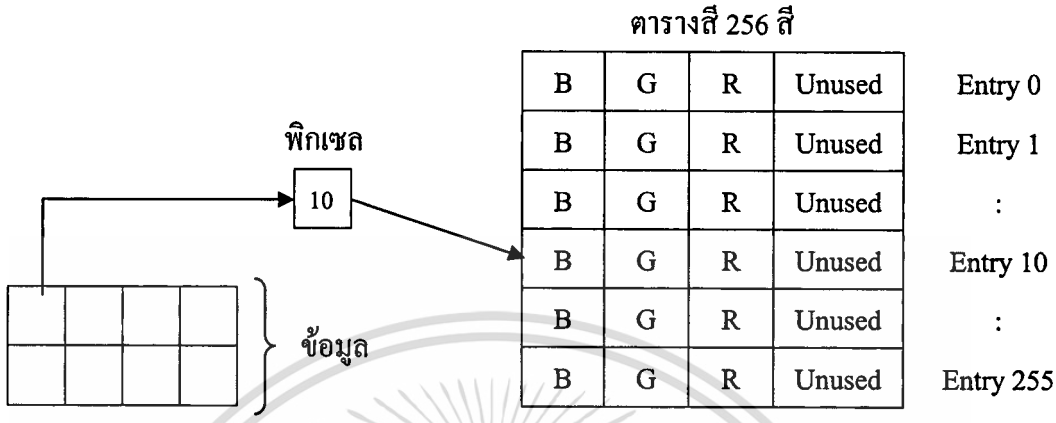
<u>หมายเหตุ</u>

เป็นส่วนของ BITMAPFILEHEADER

เป็นส่วนของ BITMAPINFOHEADER

เป็นส่วนของข้อมูลโดยข้อมูลจะเป็นตัวแสดงสีได้เลย ไม่ต้องมีตารางสี และ 3 ไบต์ที่ติดกัน (Red, Green, Blue จะมองเป็น 1 พิกเซล

5. ตัวอย่างการอ้างอิงตารางสี (Palette) ของไฟล์บิตแมพ 8 บิต (256 สี)



ภาพที่ ก.3 แสดงตัวอย่างการอ้างอิงตารางสีของไฟล์บิตแมพ 8 บิต (256 สี)

หมายเหตุ: BYTE เท่ากับ 1 ไบต์

WORD เท่ากับ 2 ไบต์

DWORD เท่ากับ 4 ไบต์

LONG เท่ากับ 4 ไบต์

บรรณานุกรม

Claypoole, R. et al. 1997. **Image Processing : Edge Detection.** [Online]. Available: http://www.owlnet.rice.edu/~elec539/Projects97/morphjrks/moredge.html

Gao, J. and Yang, J. 2001. "An Adaptive Algorithm for Text Detection from Natural Scenes." **Proceedings of Computer Vision and Pattern Recognition (CVPR).** pp. 84-89.

Jung, K., Kim, KI., Kurata, T., Kourogi, M. and Han, J. 2002. "Text Scanner with Text Detection Technology on Image Sequences." **Proceedings of IEEE International Conference of Pattern Recognition (ICPR).** Vol. 3. pp. 473-476.

Microsoft Corporation. 2003. **The Microsoft Vision SDK, version 1.2.** [Online] Available: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnvissdk/html/vissdk.asp

Microsoft Corporation. 2003. **Vision Technology.** [Online]. Available: http://research.microsoft.com/projects/VisSDK/

Shapiro, L.G. and Stockman, G.C. 2001. **Computer Vision.** New Jersey: Prentice Hall.

Sonka, M., Hlavac, V. and Boyle, R. 1999. **Image Processing, Analysis, and Machine Vision.** Second Edition. Monterey, Calif: Brooks/Cole Publishing.

Wu, V. and Manmatha, R. 1998. "Document Image Clean-Up and Binarization." **Proc. SPIE'98 Document Recognition V.** pp. 263-273.

Wu, V., Manmatha, R. and Riseman, E.M. 1999. "Textfinder: An Automatic System to Detect and Recognize Text in Images." **IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI).** Vol. 21. No. 11. pp. 1224-1229.

ประวัติผู้เขียน

ชื่อ	นายพูลศักดิ์ จีรบุณย์
วัน เดือน ปีเกิด	23 มกราคม พ.ศ. 2523
สถานที่เกิด	กรุงเทพมหานคร
วุฒิการศึกษาระดับปริญญาตรี	วิศวกรรมศาสตรบัณฑิต สาขาวิชาวิศวกรรมไฟฟ้า (ไฟฟ้ากำลัง) มหาวิทยาลัยเกษตรศาสตร์
ปีที่สำเร็จการศึกษา	2544