จอสัมผัสโดยใช้เวบแคม

TOUCH SCREEN USING WEBCAM

นายรัฐพล ผลประพฤติ
นายวาทิศ อัชชะกุลวิสุทธิ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2547

จอสัมผัสโดยใช้เวบแคม

TOUCH SCREEN USING WEBCAM

นายรัฐพล ผลประพฤติ
นายวาทิศ อัชชะกุลวิสุทธิ์

อาจารย์ที่ปรึกษา
อ.สมเกียรติ วังศิริพิทักษ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2547

ปริญญานิพนธ์ ปีการศึกษา 2547

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง จอสัมผัสโดยใช้เวบแคม

TOUCH SCREEN USING WEBCAM

คณะผู้จัดทำ นายรัฐพล ผลประพฤติ รหัส 44010398

นายวาทิศ อัชชะกุลวิสุทธิ์ รหัส 44010440

..... อาจารย์ที่ปรึกษา

(อ.สมเกียรติ วังศิริพิทักษ์)

จอสัมผัสโดยใช้เวบแคม

นายรัฐพล ผลประพฤติ 44010398
นายวาทิศ อัชชะกุลวิสุทธิ์ 44010440
อ.สมเกียรติ วังศิริพิทักษ์ อาจารย์ที่ปรึกษา
ปีการศึกษา 2547

บทคัดย่อ

โครงงานจอสัมผัสโดยใช้เวบแคม เป็นโครงงานที่พัฒนาขึ้นมาเพื่อใช้ทดแทน จอสัมผัสแบบเก่าที่ใช้แผ่นทัชสกรีนติดบนหน้าจอ ซึ่งการใช้แผ่นทัชสกรีนนั้นอาจไม่สะดวกสำหรับเครื่องคอมพิวเตอร์ทั่วไป เนื่องจากราคาแพง และความจำกัดของขนาดของแผ่นจอทัชสกรีน คือแผ่นนึงสามารถจะใช้ได้กับจอภาพขนาดเดียวเท่านั้น ต่างจากเวบแคมที่มีราคาถูกกว่า อุปกรณ์ที่ต้องใช้ในโครงงานนี้ มีกล้องเวบแคม 1 ตัวกับเครื่องคอมพิวเตอร์ส่วนตัวที่รันโปรแกรมที่พัฒนาขึ้น ซึ่งตัวโปรแกรมนั้นใช้หลักการการประมวลผลภาพดิจิตอล โดยจะนำภาพเฟรมปัจจุบันมาเปรียบเทียบกับเฟรมก่อนหน้าโดยใช้หลักการ Frame Difference เพื่อหาความเปลี่ยนแปลงของปลายนิ้วแล้วนำมาตัดสินว่าปลายนิ้วของผู้ใช้อยู่ที่ตำแหน่งไหน ส่วนวิธีการขยับนิ้วจะเป็นตัวตัดสินว่าผู้ใช้ต้องการจะทำอะไร นอกจากนี้โปรแกรมยังใช้สีของปลายนิ้วที่ได้จากตอนคอนฟิกโปรแกรมมาช่วยในการค้นหาตำแหน่งด้วย เมื่อตัดสินได้แล้วโปรแกรมก็จะทำการสร้างสัญญาณเมาส์เพื่อทำงานตามที่ผู้ใช้งานต้องการ

TOUCH SCREEN USING WEBCAM

Mr. Rathapon Polprapreut
Mr. Watis Achakulvisut
Mr. Somkiat Wangsiripituk Advisor

Academic Year 2004

ABSTRACT

The purpose of this project is to create a touch screen by using a webcam and a general monitor. The advantage of this method is the expense of the monitor is less than a real touch screen monitor. Furthermore, this method is also suitable to use with the various size monitor. With the digital image processing, the algorithm of this project uses frame difference technique to find the differences of images between an old frame and a new frame to find out the changing of user's finger and decide the position of user's finger. The way of moving finger is the decision for the user's requirement. Additionally, this application also use the color of finger from the program's configuration to finding out the right position. After that, the application will generate the mouse's signal and send it to do the command.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงได้อย่างดี ด้วยคำแนะนำ คำปรึกษาและคอยดูแลจากหลายๆฝ่ายด้วยกัน โดยเฉพาะอาจารย์ที่ปรึกษา อ.สมเกียรติ วังศิริพิทักษ์ ที่ให้โอกาสให้ข้าพเจ้าได้ทำปริญญานิพนธ์ฉบับนี้ คอยให้ความเอาใจใส่ แนะนำและความช่วยเหลือเสมอมาซึ่งต้องขอขอบพระคุณเป็นอย่างสูง

ขอขอบคุณภาควิชาวิศวกรรมคอมพิวเตอร์ที่ได้จัดเตรียมสิ่งอำนวยความสะดวก เพื่อให้การวิจัยและพัฒนาโปรแกรมเป็นไปได้ด้วยความสะดวก

ขอขอบคุณเพื่อนๆ ในห้องมัลติมีเดียที่คอยสร้างความคึกครื้นยามอยู่ในห้อง เป็นกำลังใจ และคอยช่วยเหลือซึ่งกันและกันเสมอมา

และสุดท้ายต้องขอขอบคุณบุคคลที่สำคัญที่สุดในชีวิตที่ทำให้ข้าพเจ้ามีวันนี้ นั่นคือ บิดา มารดา และบุคคลในครอบครัว อันเป็นที่เคารพรัก ซึ่งได้เลี้ยงดู คอยสั่งสอนข้าพเจ้ามาเป็นอย่างดี พร้อมให้โอกาสในการศึกษาอย่างเต็มที่ และยังให้กำลังใจ ความรักเสมอมา ข้าพเจ้าขอกราบพระคุณมา ณ ที่นี้ด้วย

รัฐพล ผลประพฤติ

วาทิศ อัชชะกุลวิสุทธิ์

สารบัญ

สารบัญตาราง

สารบัญรูป

บทที่ 1
บทนำ

1.1 หลักการและเหตุผล

ในปัจจุบันมีการใช้จอสัมผัสหรือแผ่นจอสัมผัส เพื่อช่วยอำนวยความสะดวก แต่จอภาพประเภทนี้จะมีราคาแพง และความจำกัดของขนาดของแผ่นจอทัชสกรีน แผ่นหนึ่งสามารถจะใช้ได้กับจอภาพขนาดเดียวเท่านั้น ซึ่งเป็นผลให้ไม่สามารถนำแผ่นจอสัมผัสไปใช้งานให้เกิดความคุ้มค่าเท่าที่ควร และในขณะนี้เวบแคมก็กำลังเป็นที่นิยมในการใช้สนทนากันผ่านทางอินเทอร์เนต เนื่องจากมีราคาไม่แพงมากเมื่อเทียบกับจอสัมผัส ถ้าสามารถนำเวบแคมที่มีอยู่มาช่วยให้จอคอมพิวเตอร์ธรรมดาเป็นเสมือนจอสัมผัสก็จะเป็นการใช้งานเวบแคมอย่างคุ้มค่า

โครงงานนี้จึงมีแนวคิดที่จะทำการจำลองหน้าจอคอมพิวเตอร์ทั่วไปให้เสมือนเป็นหน้าจอคอมพิวเตอร์แบบจอสัมผัสโดยใช้เวบแคม 1 ตัว ในการจับภาพหน้าจอคอมพิวเตอร์ และใช้ปลายนิ้วในการชี้ตำแหน่งต่างๆ บนหน้าจอคอมพิวเตอร์ เพื่อนำไปเป็นคำสั่งให้คอมพิวเตอร์ทำงานต่อไป

1.2 ขอบเขตของโครงงาน

1. สามารถจำลองการเลื่อนตำแหน่งเมาส์ได้
2. สามารถจำลองการคลิ้กเมาส์ได้
3. สามารถจำลองการดับเบิลคลิ้กเมาส์ได้
4. สามารถจำลองการทำ Drag Drop ได้
5. สามารถใช้งานบนหน้าจอที่ไม่มีการเคลื่อนไหวได้

1.3 วัตถุประสงค์ของโครงงาน

1. ศึกษาทฤษฎีการประมวลผลภาพด้วยคอมพิวเตอร์
2. ศึกษาการเขียนโปรแกรมโดยใช้การประมวลผลภาพด้วยคอมพิวเตอร์
3. ศึกษาการเขียนโปรแกรม ด้วยภาษา C++ โดยใช้ MFC
4. ศึกษาการเขียนโปรแกรมติดต่อกับเวบแคม
5. พัฒนาหน้าจอธรรมดาให้เป็นเสมือนหน้าจอสัมผัส

1.4 เป้าหมายของโครงงาน

โครงการนี้มีเป้าหมายในการทำหน้าจอธรรมดา ให้เป็นเสมือนหน้าจอสัมผัส โดยผู้ใช้สามารถจะใช้ปลายนิ้วมือสัมผัสไปยังตำแหน่งบนหน้าจอที่ต้องการจะเลื่อนตำแหน่งเมาส์ หรือคลิ้กเมาส์ซึ่งจะใช้เวบแคม 1 ตัว เพื่อจับการเคลื่อนไหว และตำแหน่งของปลายนิ้วที่มาสัมผัสลงบนหน้าจอ เพื่อใช้เป็น

การแทนการทำงานของเมาส์ ได้ดังนี้คือ การเลื่อนตำแหน่งเมาส์ การกดคลิ้ก ดับเบิลคลิ้ก และการ Drag Drop โดยจะจำลองสัญญาณเมาส์ส่งไปให้กับระบบปฏิบัติการแทนสัญญาณที่เกิดจากอุปกรณ์จริงๆ โดยโปรแกรมที่พัฒนาขึ้นมานั้นใช้ภาษา C++ และ MFC ในการพัฒนาโปรแกรมและจำลองสัญญาณเมาส์ และติดต่อกับกล้องเวบแคมด้วย ActiveX control ชื่อว่า VideoOCX

บทที่ 2
ทฤษฎี

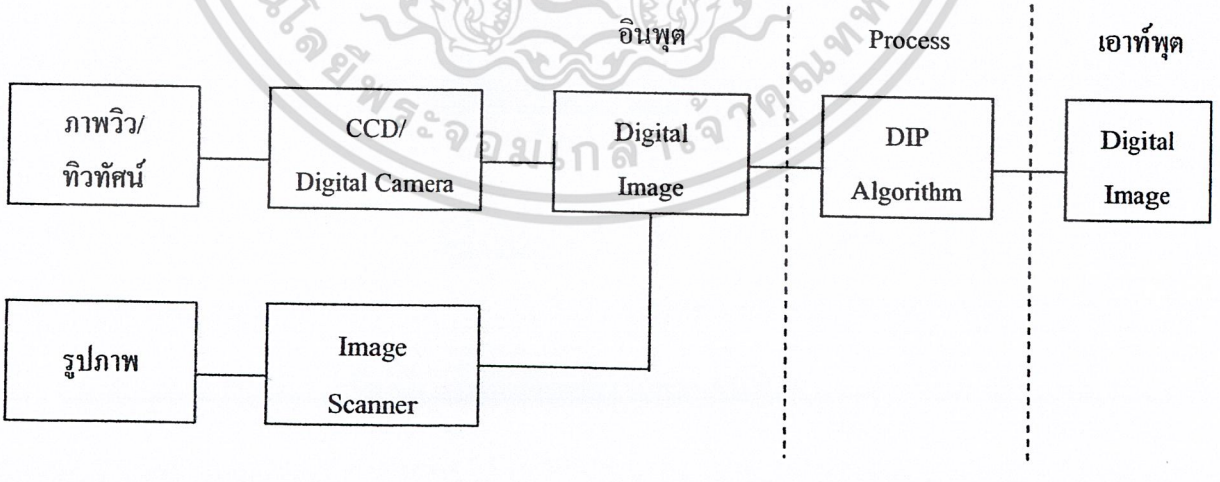
2.1 ความรู้ข้อมูลภาพดิจิตอล

โดยปกติแล้วสายตาของบุคคลทั่วไปจะมองเห็นภาพทิวทัศน์วิวต่างๆ เป็นลักษณะอนาล็อก ซึ่งสามารถอธิบายได้ด้วยคณิตศาสตร์ที่มีตัวแปรแบบนับได้อย่างต่อเนื่อง แต่เครื่องคอมพิวเตอร์จะใช้เลขฐานสองเป็นหลักในการคำนวณ เมื่อนำภาพนั้นมาแปลงเข้าสู่เครื่องคอมพิวเตอร์ ก็จะกลายเป็นภาพดิจิตอล (Digital Image)

2.1.1 กระบวนการทำภาพดิจิตอล

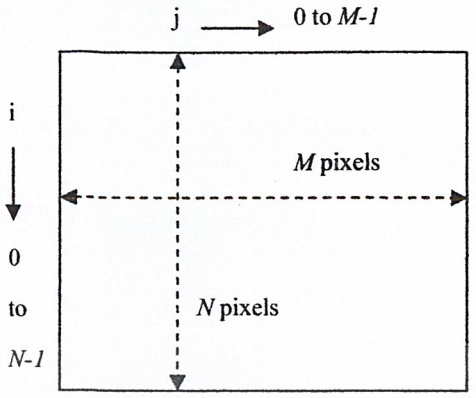
มีสองเหตุผลใหญ่ๆ ที่ต้องการทำกระบวนการทำภาพดิจิตอล คือ เพื่อปรับปรุงภาพดิจิตอลให้มองเห็นได้ง่ายขึ้น และเพื่อปรับปรุงภาพให้หุ่นยนต์ตีความหมาย หรือเข้าใจจดจำรูปร่างลักษณะได้อย่างแม่นยำ ตัวอย่างเช่น การจดจำตัวอักษร หรือ Optical Character Recognition (OCR) ที่สามารถจดจำตัวอักขระได้ถึง 99.9% การปรับปรุงภาพให้ใช้พื้นที่เก็บน้อยลง การตรวจสอบลายพิมพ์มือของแต่ละบุคคล เป็นต้น

กระบวนการทำภาพดิจิตอล คือ การนำภาพดิจิตอลเข้ามาทำกระบวนการโดยการใส่ฟังก์ชันอัลกอริทึมต่างๆ เข้าไป ก็จะได้เอาท์พุตเป็นภาพดิจิตอล ที่ตรงตามแนวความคิดของการทำกระบวนการภาพดิจิตอล เราสามารถแสดงภาพตั้งแต่การนำภาพวิว ทิวทัศน์ จนถึงเอาท์พุต ดังรูปที่ 2-1



รูปที่ 2-1 กระบวนการทำภาพดิจิตอล

2.1.2 พิกเซล (Pixel)



รูปที่ 2-2 ลักษณะของตำแหน่งพิกเซล

ในภาพหนึ่งๆ เราสามารถอธิบายได้ด้วยเมทริกซ์ของจุดพิกเซลขนาด N X M โดยใช้คู่ลำดับ p(i,j) แทนค่าของจุดแต่ละจุด โดย i มีค่าตั้งแต่ 0 ถึง N-1 และ j มีค่าตั้งแต่ 0 ถึง M-1 และ p(i,j) นี้จะบ่งชี้ความเข้มของแสงที่จุดนั้นๆ ของภาพ

ค่าที่กำกับแต่ละพิกเซลจะแสดงถึง ค่าเฉลี่ยของความเข้มแสงในภาพที่จุดพิกเซลนั้นแทนอยู่ โดยค่าของพิกเซลนี้จะเขียนแทนด้วย P_{ij} ซึ่งมีค่าตั้งแต่ 0 ถึง 1

2.1.3 เกรย์สเกล

เกรย์สเกล หมายถึง ความแตกต่างของระดับความเข้มแสง โดยเกรย์สเกลหนึ่งๆ อาจแบ่งเป็น 13,20 และ 9 ระดับ โดยระดับที่ว่านี้ก็คือ ระดับสีเทา ในภาพหนึ่งๆ ถ้าต้องการแบ่งระดับความเข้มแสงหรือระดับสีเทาให้มีหลายๆ ค่านั้น จำเป็นอย่างยิ่งจะต้องเพิ่มจำนวนบิตที่แสดงค่าพิกเซล ตัวอย่าง เช่น ภาพที่มีระดับสีเทา 4 ระดับต้องแทนด้วยเลขฐานสอง จำนวน 2 บิต ถ้าต้องการภาพที่มีระดับสีเทา 16 ระดับ ต้องแทนด้วยเลขฐานสองจำนวน 4 บิต และถ้าต้องการภาพที่มีระดับสีเทา 256 ระดับต้องแทนด้วยเลขฐานสองจำนวน 8 บิต เป็นต้น

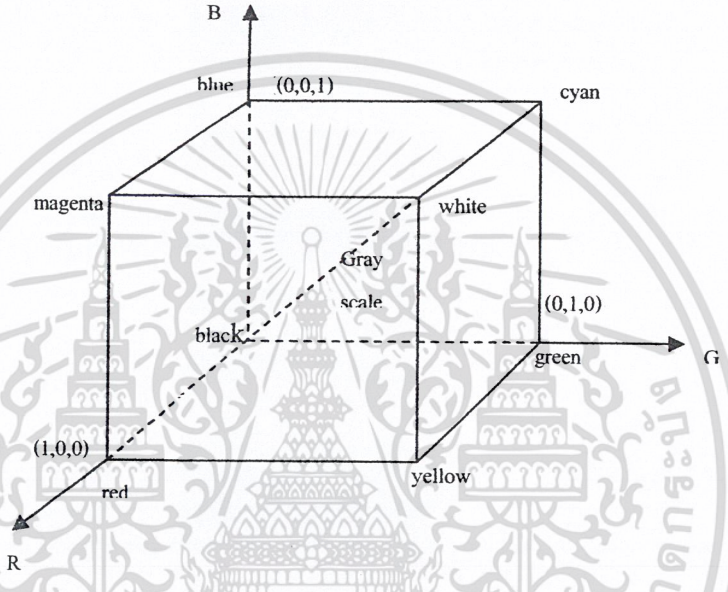
จำนวนระดับสีเทาที่ต้องการนี้ก็คือ ค่าเลขยกกำลัง 2 นั่นเอง ซึ่งค่าต่ำสุดหรือ 0 จะแทนสีดำหรือไม่มีความสว่างเลย และค่าที่มากก็คือค่าที่น้อยกว่าจำนวนระดับสีเทาอยู่ 1 เช่นค่า 15 ในระบบที่มีระดับสีเทา 16 ระดับ จะมีสีขาวหรือสว่างมากที่สุด เป็นต้น

2.1.4 พื้นฐานและระบบสีโมเดล RGB

ในโมเดลนี้ สีแต่ละสีจะอยู่ในรูปของสีปฐมภูมิ (แดง, เขียว, น้ำเงิน) โมเดลนี้มีโครงสร้างอยู่ในระบบของพิกัดคาร์ทีเซียน (Cartesian coordinate) มีลักษณะเป็นทรงลูกบาศก์ ดังรูปที่ 2-3 ค่าสี แดง, เขียว, น้ำเงิน จะอยู่ที่มุมทั้งสามบนเส้นทแยงมุมตรงข้ามกัน และค่าสีคราม, มาเจนตา, เหลือง จะอยู่ที่มุมทั้งสามในลักษณะเดียวกัน ส่วนสีดำอยู่ที่จุดกำเนิด สีขาวอยู่ที่มุมที่มีระยะทางไกลที่สุดจากแหล่งกำเนิด ค่าของระดับสีเทา (Gray Scale) จะอยู่บนเส้นที่เชื่อมระหว่างสีดำกับขาว ค่าสี คือ จุดที่อยู่

บนผิวหรือในลูกบาศก์ถูกกำหนดค่าโดยเวกเตอร์ที่ชี้ออกจากจุดกำเนิด เพื่อความสะดวกเราจะสมมุติให้ค่าสีถูกนอมอลไลซ์ (Normalize) ทั้งสามสี โดยให้มีค่าอยู่ในช่วง 0 ถึง 1 ลูกบาศก์ที่แสดงในรูปที่ 2 จึงเป็นลูกบาศก์หนึ่งหน่วย

ภาพในโมเดล RGB ประกอบด้วยภาพสามระนาบที่เป็นอิสระจากกัน สำหรับแต่ละสีปฐมภูมิเมื่อป้อนเข้าไปในมอนิเตอร์ที่เป็นแบบ RGB ภาพทั้งสามสีจะรวมตัวกันที่จอภาพกลายเป็นภาพสีผสม ดังนั้นการใช้โมเดล RGB ในการประมวลผลภาพนั้นจะสมเหตุสมผลเมื่อภาพถูกแยกออกโดยธรรมชาติให้อยู่ในเทอมของทั้งสามสี กล้องภาพสีส่วนใหญ่ที่ให้ภาพสีดิจิตอลจะอยู่ในรูปแบบของโมเดล RGB ดังนั้นโมเดลนี้จึงเป็นโมเดลที่สำคัญมากในการประมวลผลภาพ

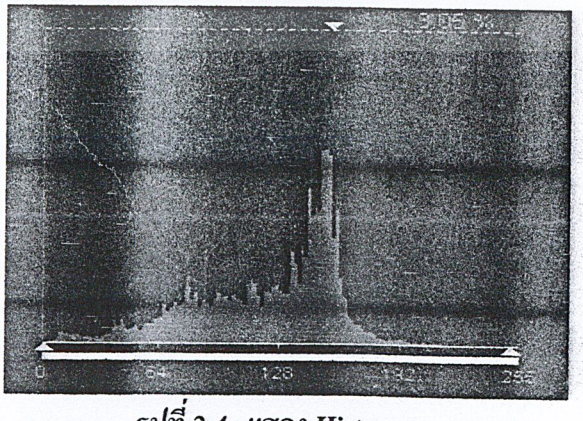


รูปที่ 2-3 โมเดล RGB

2.2 ทฤษฎีพื้นฐานของการประมวลผลภาพ

2.2.1 Histogram

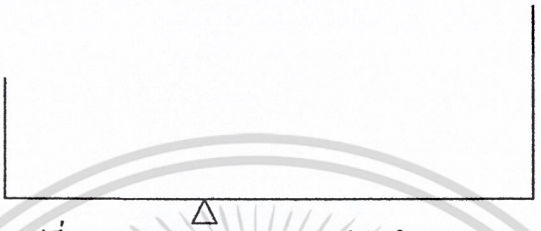
Histogram คือ กราฟที่แสดงความสัมพันธ์ระหว่างจำนวน Pixel ของแต่ละระดับ Gray-level ของภาพ เช่น ภาพ 256 สี จะมีค่า Gray-level = 0 ถึง Gray-level = 255



รูปที่ 2-4 แสดง Histogram

2.2.2 Threshold

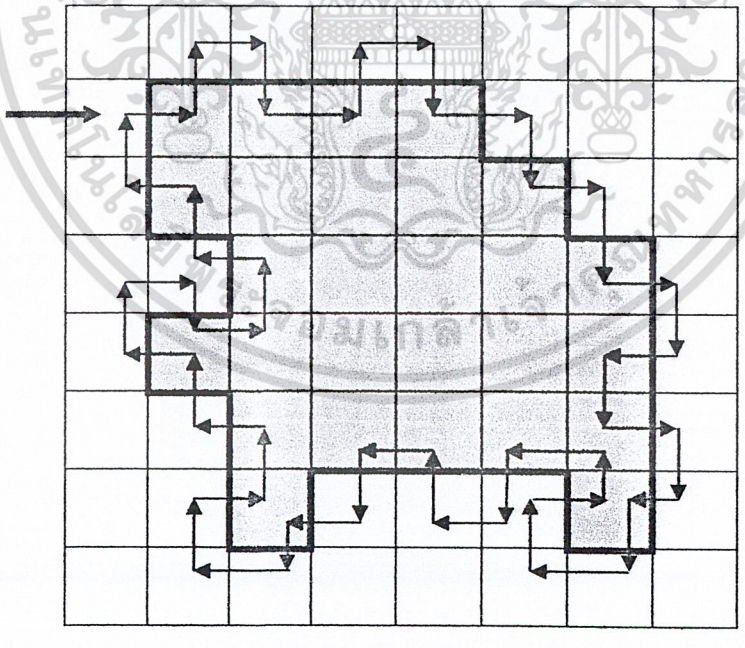
การปรับค่า Threshold นี้คือ ลักษณะการแยก (Split) ระดับ Gray-level ออกเป็น 2 ค่า คือ ค่าสูงสุด (Gray-level = 255) และค่าต่ำสุด (Gray-level = 0) ตามที่กำหนด โดยการเปรียบเทียบค่า Threshold ที่ตั้งไว้ว่ามีค่ามากกว่าหรือมีค่าน้อยกว่า คือ ถ้าจุดภาพนั้นมีระดับ Gray-level มากกว่าค่า Threshold ก็เปลี่ยนให้จุดภาพนั้นมีระดับ Gray-level = 255 แล้วถ้าจุดภาพนั้นมีระดับ Gray-level น้อยกว่าค่า Threshold ก็เปลี่ยนให้จุดภาพนั้นมีระดับ Gray-level = 0



รูปที่ 2-5 Histogram หลังจากปรับค่า Threshold

2.2.3 เทคนิคการตามรอยขอบภาพ (Contour Following)

เทคนิคการตามรอยขอบภาพจะช่วยให้เราทำการหาวัตถุได้ง่ายขึ้น โดยไม่ต้องคำนึงถึงรูปร่างของวัตถุ เมื่อพิจารณาขอบของวัตถุจะมีค่าสีที่มีความแตกต่างกันกับสีพื้นหลัง เช่น ภาพตัวอักขระสีดำที่วางอยู่บนพื้นสีขาว



รูปที่ 2-6 การหาขอบด้วยวิธี Contour Following

การทำงานของเทคนิคการตามรอยขอบภาพ ตามรูปที่ 2-6 เป็นการเดินไต่ไปตามขอบระหว่างส่วนที่เป็นรูปภาพ กับส่วนที่เป็นพื้นหลัง โดยจะตรวจกวาดไปทุกๆ จุดภาพ โดยจะเริ่มจากจุดมุมซ้ายบนของข้อมูลภาพ ตรวจกวาดไปในทิศทางจากซ้ายไปขวา และเลื่อนจากบนลงล่าง โดยมีเงื่อนไขดังต่อไปนี้

1. ถ้าจุดที่อยู่ปัจจุบันมีข้อมูลเป็นสีดำ ให้เลี้ยวซ้ายจากทิศทางปัจจุบัน
2. ถ้าจุดที่อยู่ปัจจุบันมีข้อมูลเป็นสีขาว ให้เลี้ยวขวาจากทิศทางปัจจุบัน
3. การเคลื่อนที่จะสิ้นสุดลง เมื่อจุดที่อยู่ปัจจุบันเป็นจุดเดียวกันกับจุดเริ่มต้น

รูปที่ 2-7 Flow Chart การหาขอบด้วยวิธี Contour Following

อธิบาย Flow chart วิธีการ Contour Following จากรูปที่ 2-7

1. หาจุดดำ เป็นการหาจุดดำจุดแรกที่พบ โดยการสแกนตามแนวนอน
2. จำตำแหน่งของจุด เป็นการจำตำแหน่งของจุดดำที่พบ
3. วิเคราะห์จุด มี 2 กรณี
   - ขาว เลี้ยวขวาจากทิศทางปัจจุบัน
   - ดำ เลี้ยวซ้ายจากทิศทางปัจจุบัน
4. จำทิศทางปัจจุบัน จดจำทิศทางและตำแหน่งปัจจุบัน
5. จุดปัจจุบัน = จุดแรก เป็นการเปรียบเทียบตำแหน่งปัจจุบันกับจุดเริ่มต้น
   - ใช่ จบการทำ Contour Following
   - ไม่ใช่

ข้อดีของการทำวิธี Contour Following ก็คือ ไม่ต้องสนใจว่ารูปร่างของวัตถุจะมีลักษณะเป็นอย่างไร เมื่อเราวิ่งไปตามขอบของภาพ เราก็จะได้ขอบของวัตถุออกมา

ข้อเสียของการทำวิธีการนี้ก็คือ ถ้าวัตถุมีส่วนที่ติดกันอยู่จะไม่สามารถใช้วิธีนี้ในการแยกวัตถุออกจากกันได้

2.2.4 เทคนิคการแยกโดยใช้วิธีการ Contour with Matrix

เทคนิคนี้คล้ายกับการทำ Contour Following แต่จะต่างกันตรงที่เทคนิคนี้จะมีการดึงเอาข้อมูลภาพที่อยู่ในขอบเขตของการทำ Contour ไปใช้ด้วย ทำให้เราได้วัตถุทั้งอันไปใช้งาน และการใช้งานไม่เพียงแต่ดูว่าข้อมูลตรงจุดนั้นเป็นสีดำ หรือสีขาว แต่จะใช้เมทริกซ์ช่วยตรวจสอบแทน เมทริกซ์ที่ใช้มีขนาด 3*3 ดังรูปที่ 2-8

a1	a2	a3
a4	a5	a6
a7	a8	a9

*รูปที่ 2-8 เมทริกซ์ ขนาด 3*3*

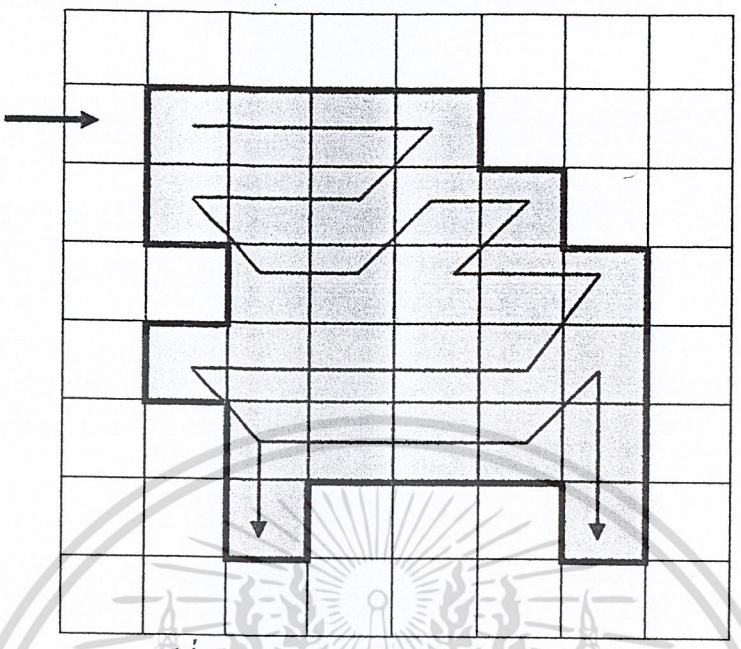
แนวคิดของการทำ Contour with Matrix ดังรูปที่ 2-8 มีขั้นตอนการทำงานดังต่อไปนี้

1. ให้กำหนดจุดดำที่พบเป็นจุดกลางของเมทริกซ์ (a5) คัดลอกจุดนี้ลงไปในบัฟเฟอร์ แล้วลบจุดกลางนั้นออกจากรูปภาพ

2. หาจุดกลางถัดไป โดยตรวจสอบจากข้อมูลรอบๆ จุดกลางปัจจุบัน(a5) ซึ่งเป็นจุดสีดำ คือ การตรวจสอบจะเริ่มจากด้านบนซ้ายก่อน คือ ตำแหน่ง a1,a2,a3,a4,a5,a6,a7,a8 และ a9 ตามลำดับ โดยกำหนดจุดดำที่พบจุดแรกให้เป็นจุดกึ่งกลางเมทริกซ์

3. เมื่อรอบๆ จุดกลางนั้นไม่มีจุดดำแล้ว จะทำการย้อนกลับไปยังจุดกลางตัวก่อน

4. การทำงานจะเป็นเช่นนี้ไปเรื่อยๆ จนกระทั่งไม่มีจุดดำรอบจุดกลางทุกตัว



รูปที่ 2-9 การหาขอบโดยวิธี Contour with matrix

ตัวอย่างการทำงานด้วยวิธีนี้แสดงดังรูปที่ 2-9 เส้นทางการวิ่งจะเป็นไปตามลูกศรที่แสดงในรูป และสามารถใช้ Flow chart จากรูปที่ 2-10 อธิบายได้ดังนี้

1. หาจุดดำ เป็นการหาจุดดำจุดแรกที่พบโดยการสแกนตามแนวนอน
2. จำตำแหน่งของจุด จำตำแหน่งของจุดดำที่พบ
3. Copy Pixels จะทำการคัดลอกจุดที่พบไว้ในบัฟเฟอร์ และทำการลบจุดที่พบออกจากรูปภาพ
4. จุดดำรอบจุดปัจจุบัน ตรวจว่ารอบจุดปัจจุบันมีจุดสีดำล้อมรอบหรือไม่
   - มี จำตำแหน่งของจุด
   - ไม่มี จุดที่ไม่ได้วิเคราะห์
5. จุดที่ไม่ได้วิเคราะห์ ตรวจว่ามีจุดที่ยังไม่ได้ทำการวิเคราะห์หรือไม่
   - มี Copy Pixels
   - ไม่มี จบการทำ Contour with matrix

ข้อดีของวิธี Contour with Matrix ก็คือ สามารถดึงเอาส่วนที่เป็นเนื้อวัตถุออกมาได้ทั้งหมด เป็นประโยชน์เมื่อต้องการลบวัตถุนั้นๆ ออกจากภาพ

ข้อเสียของวิธีนี้ก็คือ ต้องเอาข้อมูลทุกพิกเซลของวัตถุออกมา ทำให้ต้องใช้เวลานานกับการคัดลอก และลบข้อมูล วิธีการนี้ทำงานแบบ Recursive ถ้ามีการ backtracking มากเกินไปอาจทำให้ stack overflow ได้

รูปที่ 2-10 Flow Chart การหาขอบโดยวิธี Contour with matrix

2.2.5 Noise Removing

เป็นวิธีที่ใช้ในการกำจัดข้อมูลภาพส่วนที่ไม่ต้องการ (Noise) ออกจากข้อมูลภาพที่นำมาประมวล โดยใช้หลักการดังนี้ คือ

- ใช้วินโดว์ขนาด 3*3 เป็นขอบเขตที่ใช้ทำการรวมค่าที่อยู่ในขอบเขตของวินโดส์

a1	a2	a3
a4	a5	a6
a7	a8	a9

รูปที่ 2-11 ขอบเขตวินโดส์

- ทำการรวมค่าที่อยู่ในขอบเขตของวินโดส์ โดยสมการที่ใช้ในการรวมค่าเป็นดังนี้

$$S(x,y) = \sum_{b=y-1}^{y+1} \sum_{a=x-1}^{x+1} S(a,b)$$

โดยที่ $S(a,b) = 1$ เมื่อ $S(a,b) > ts$

$S(a,b) = 0$ เมื่อ $S(a,b) <= ts$

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

รูปที่ 2-12 การหาผลรวมของข้อมูลภาพตำแหน่งที่ (x = 1, y = 1)

เช่น จากข้อมูลภาพ รูปที่ 2-12 ถ้าต้องการหาผลรวมของข้อมูลตำแหน่งที่ (x = 1, y = 1) จะได้ดังนี้

$$\begin{aligned} S(1,1) &= S(0,0) + S(0,1) + S(0,2) + \\ &\quad S(1,0) + S(1,1) + S(1,2) + \\ &\quad S(2,0) + S(2,1) + S(2,2) \\ &= 0+1+0+0+1+1+0+1+1 \end{aligned}$$

= 5

ดังนั้น S(1,1) = 5 ดังปรากฏในรูปที่ 2-13

x	x	x
x	5	x
x	x	x

รูปที่ 2-13 ผลรวมของข้อมูลภาพตำแหน่งที่ (x = 1, y = 1)

- ทำการรวมค่าของข้อมูลภาพดังที่ได้กล่าวมาแล้วข้างต้นจนครบทุกจุดภาพ โดยจากข้อมูลภาพที่ยกตัวอย่างมาข้างต้น เมื่อทำการรวมค่าของข้อมูลภาพจนครบทุกจุดภาพ จะมีข้อมูลปรากฏดังรูปที่ 2-14

x	x	x	x	x	x	x	x
x	5	7	8	8	7	4	x
x	6	8	9	9	9	6	x
x	5	7	9	8	8	5	x
x	3	5	8	7	5	4	x
x	3	4	7	6	4	4	x
x	3	4	6	6	4	4	x
x	x	x	x	x	x	x	x

รูปที่ 2-14 ผลลัพธ์การรวมค่าของข้อมูลภาพครบทุกจุดภาพ

- กำหนดค่า threshold เพื่อใช้เป็นตัวกำหนดการกำจัดข้อมูลภาพส่วนที่ไม่ต้องการ โดยจากข้อมูลภาพที่ยกตัวอย่างมาข้างต้น เมื่อทำการกำหนดค่า threshold = 6 จะมีข้อมูลปรากฏดังรูปที่ 2-15

x	x	x	x	x	x	x	x
x	0	1	1	1	1	0	x
x	1	1	1	1	1	1	x
x	0	1	1	1	1	0	x
x	0	0	1	1	0	0	x
x	0	0	1	1	0	0	x
x	0	0	1	1	0	0	x
x	x	x	x	x	x	x	x

รูปที่ 2-15 ผลลัพธ์ของการทำ Noise Removing เมื่อกำหนดค่า threshold = 6

หมายเหตุ x หมายถึง ข้อมูลภาพที่ไม่สามารถนำมาประมวลผลได้ (ไม่นำมาใช้ในการพิจารณา)

2.2.6 Frame Difference

เป็นวิธีการหาความแตกต่างของภาพ 2 ภาพ ซึ่งสามารถนำมาใช้หาวัตถุที่กำลังเคลื่อนที่อยู่ได้ โดยเมื่อทำ Frame Difference แล้ว ส่วนของภาพที่เหมือนกันจะเป็นสีดำ ในขณะที่ส่วนของภาพที่แตกต่างกันจะเป็นสีขาว โดยใช้หลักการดังนี้ คือ

100	110	150	150	150	110	100
200	120	125	127	129	130	100
10	11	10	5	10	11	10
10	11	20	25	30	40	50
10	11	10	5	10	11	10
250	255	220	210	220	255	250
70	80	85	20	90	65	30

รูปที่ 2-16 เฟรมภาพก่อนหน้า(G)

100	110	150	150	150	110	100
200	120	125	127	129	130	100
10	11	110	150	180	11	10
10	11	40	75	100	40	50
10	11	100	105	100	11	10
250	255	220	210	220	255	250
70	80	85	20	90	65	30

รูปที่ 2-17 เฟรมภาพปัจจุบัน (F)

นำภาพปัจจุบันมาลบกับภาพก่อนหน้านี้ โดยทำการลบพิกเซลที่อยู่ที่ตำแหน่งเดียวกัน จนครบทุกพิกเซล เช่น จากรูปที่ 2-16 และรูปที่ 2-17 ถ้าต้องการหาความแตกต่างของตำแหน่งที่ (x = 3, y = 3) จะได้

$$H(3,3) = |F(3,3) - G(3,3)|$$

$$H(3,3) = |110 - 10|$$

$$H(3,3) = 100$$

รูปที่ 2-18 แสดงผลลัพธ์ของการทำ Frame Difference ทุกพิกเซล จะเห็นได้ว่าพิกเซลที่ไม่มีการเปลี่ยนแปลงจะมีค่าเป็น 0 ซึ่งก็คือ สีดำ ในขณะที่พิกเซลที่มีการเปลี่ยนแปลงจะมีค่ามากกว่า 0 ทำให้เราสามารถหาเปลี่ยนแปลงนั้นได้

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	90	145	170	0	0
0	0	20	50	70	0	0
0	0	90	100	90	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

รูปที่ 2-18 ภาพที่ได้ผ่านการทำ Frame Difference (H)

2.2.7 Geometric Transformations

Geometric Transformations จะประกอบด้วย 2 ขั้นตอนพื้นฐาน คือ (1) Spatial Transformation ซึ่งจะอธิบายถึงการจัดพิกเซลใหม่บนระนาบของรูปภาพ และ (2) Gray-level Interpolation ซึ่งเกี่ยวข้องกับการกำหนดระดับของ Gray-level ไปเป็นพิกเซลในการแปลงรูปภาพ

1. Spatial Transformations

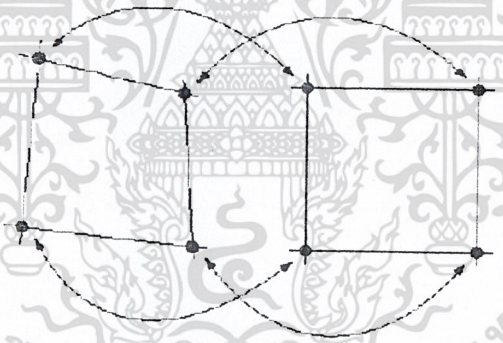
รูปภาพ f ประกอบด้วยพิกเซลพิกัด (x, y) เมื่อได้รับผลจาก Geometric distortion จะได้รูปภาพ g ที่ประกอบด้วยพิกัด (x', y') การทรานฟอร์มที่เกิดขึ้นนี้แสดงได้ดังสมการ

$$x' = r(x, y) \tag{1}$$

และ

$$y' = s(x, y) \tag{2}$$

โดยที่ $r(x, y)$ และ $s(x, y)$ เป็น Spatial Transformations ที่ทำให้เกิดรูปภาพ $g(x', y')$ ซึ่งเป็นรูปภาพแบบ Geometrically distorted image อาทิเช่น ถ้า $r(x, y) = x/2$ และ $s(x, y) = y/2$ ความผิดเพี้ยนที่เกิดขึ้นจะเป็นการลดลงของขนาดของ $f(x, y)$ ซึ่งจะเป็นครึ่งหนึ่งทั้ง 2 ทิศทาง



รูปที่ 2-19 การทำ Spatial Transformations

รูปที่ 2-19 แสดงพื้นที่สี่เหลี่ยมในภาพที่เกิดการผิดเพี้ยนและในภาพที่ถูกต้อง มุมทั้ง 4 ของรูป 4 ด้าน (quadrilaterals) นี้ จะเป็น tiepoints ที่สอดคล้องกัน กระบวนการ Geometric distortion ในพื้นที่สี่เหลี่ยมจะถูกจำลองด้วย Bilinear equation ดังนี้

$$r(x, y) = c_1 x + c_2 y + c_3 xy + c_4 \tag{3}$$

และ

$$s(x, y) = c_5 x + c_6 y + c_7 xy + c_8 \tag{4}$$

ดังนั้น จากสมการที่ 1 และ 2

$$x' = c_1 x + c_2 y + c_3 xy + c_4 \tag{5}$$

และ

$$y' = c_5 x + c_6 y + c_7 xy + c_8 \qquad (6)$$

ถ้ามีจำนวน tiepoints ที่ทราบทั้งหมด 8 จุด การแก้สมการจะได้สัมประสิทธิ์ทั้งหมด 8 ตัว คือ c_i , $i = 1,2,...,8$ สัมประสิทธิ์นี้จะประกอบกันขึ้นเป็นแบบจำลองของ Geometric distortion ซึ่งใช้ในการแปลงทุกพิกเซลในพื้นที่สี่เหลี่ยมซึ่งถูกกำหนดไว้แน่นอนด้วย tiepoints โดยปกติแล้วจะต้องมีจำนวน tiepoints มากเพียงพอที่จะสร้างเซตของพื้นที่สี่เหลี่ยมจำนวนหนึ่งที่ ครอบคลุมทั้งรูปภาพ และแต่ละพื้นที่สี่เหลี่ยมต้องมีเซตของสัมประสิทธิ์ซึ่งเป็นของแต่ละพื้นที่สี่เหลี่ยมนั่นด้วย

สัมประสิทธิ์ที่ได้จะนำมาใช้ในขั้นตอนการสร้างรูปภาพให้กลับคืนมา เมื่อเราต้องการหาค่าของภาพที่ถูกต้องที่จุด (x_0, y_0) ใดๆ จำเป็นต้องทราบว่าตำแหน่งนั้นถูกแทนด้วยตำแหน่งใดบนใน ภาพที่เกิดการผิดเพี้ยน $f(x_0, y_0)$ ซึ่งจะหาได้โดยการแทนค่า (x_0, y_0) ลงในสมการที่ 5 และ 6 เพื่อหาพิกัด (x'_0, y'_0) ค่าของจุดในภาพที่ถูกต้อง ที่ถูกแมปไป (x'_0, y'_0) คือ $g(x'_0, y'_0)$ ดังนั้นจะได้ค่าของ restored image โดยให้ $\hat{f}(x_0, y_0) = g(x'_0, y'_0)$ ตัวอย่างเช่น $\hat{f}(0,0)$ เราจะแทนค่า $(x, y) = (0,0)$ ในสมการที่ 5 และ 6 เพื่อหาค่าพิกัดจุด (x', y') แล้วให้ $\hat{f}(0,0) = g(x', y')$ จากนั้น แทนค่า $(x, y) = (0,1)$ ในสมการที่ 5 และ 6 จะได้พิกัดจุด (x', y') อีกจุดหนึ่ง จากนั้นให้ $\hat{f}(0,1) = g(x', y')$ และทำขั้นตอนนี้ต่อไปเรื่อยๆ จากพิกเซลหนึ่งไปยังอีกพิกเซลหนึ่ง และจากแถวหนึ่งไปยังอีกแถวหนึ่ง โดยขนาดของอาร์เรย์จะต้องไม่เกินขนาดของรูปภาพ g ส่วนการสแกนทางคอลัมน์จะให้ผลเหมือนกันทุกประการ

2. Gray-Level Interpolation

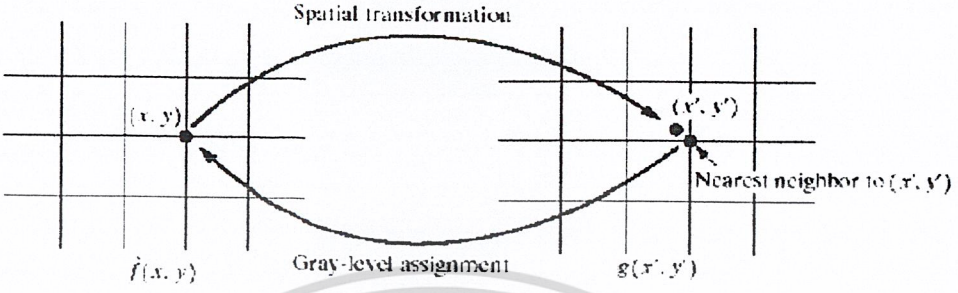
เนื่องจากรูปภาพเป็นดิจิตอล ค่าพิกเซลจะถูกระบุที่พิกัดที่เป็นจำนวนเต็มเท่านั้น แต่ค่า x' และ y' ที่ได้จากการทำ Spatial Transformations นั้นไม่เป็นจำนวนเต็ม ดังนั้นจึงต้องมีวิธีที่จะกำหนดค่า gray-level ให้กับตำแหน่งใดๆ ซึ่งวิธีนี้เรียกว่า Gray-level interpolation

วิธีที่ง่ายที่สุดสำหรับ Gray-level interpolation นั้นอยู่บนพื้นฐานของการเข้าถึงตำแหน่งใกล้เคียงที่ใกล้ที่สุด (nearest neighbor approach) วิธีนี้เรียกว่า Zero-order interpolation ซึ่งแสดงไว้ดังรูปที่ 2-20 โดยจะแบ่งเป็น (1) การแมปปิ้งของพิกัด (x, y) ซึ่งเป็นจำนวนเต็มไปเป็นพิกัด (x', y') โดยสมการที่ 5 และ 6 ,(2) การเลือกจุดที่เป็นจำนวนเต็มที่ใกล้เคียงกับพิกัด (x', y') ที่สุด และ (3) การกำหนดระดับของ gray-level ของจุดที่ใกล้เคียงที่สุดไปเป็นพิกเซลที่ตำแหน่ง (x, y)

แต่สำหรับการใช้งานทั่วไป จะใช้วิธี Bilinear interpolation ซึ่งจะต้องใช้ Gray-level 4 ระดับจึงจะเพียงพอ แต่ละระดับที่เป็นค่าใกล้เคียงที่สุดของพิกัดที่ไม่เป็นจำนวนเต็ม (x', y') ค่าของ Gray-level ที่พิกัดจุดนี้จะแทนด้วย $v(x', y')$ ซึ่งสามารถหาได้จาก

$$v(x', y') = ax' + by' + cx'y' + d \qquad (7)$$

โดยที่สัมประสิทธิ์ทั้ง 4 ตัว นี้จะสามารถหาค่าได้จากสมการ 4 สมการ และเมื่อทราบค่าสัมประสิทธิ์แล้วก็จะหาค่า $v(x',y')$ ได้ โดยที่ค่านี้จะถูกกำหนดไปยังตำแหน่งใน $f(x,y)$ ซึ่งผลที่ได้จาก Spatial mapping ของตำแหน่งนี้จะไปเป็นตำแหน่ง (x',y') ดังรูปที่ 2-20

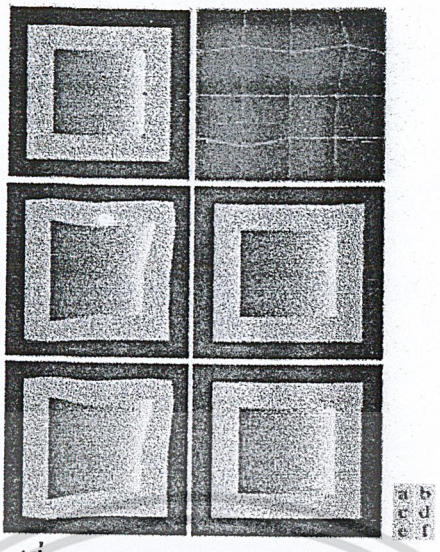


รูปที่ 2-20 การทำ Gray-level assignment

รูปที่ 2-21(a) แสดงรูปภาพประกอบด้วย tiepoints 25 จุด รูปที่ 2-21(b) แสดงการจัด tiepoints ใหม่ เพื่อสร้าง geometric distortion และจากกระบวนการที่เกี่ยวข้องกับสมการที่ 5 และ 6 นั้น สัมประสิทธิ์ของสมการดังกล่าวเป็นผลของการแมปปิ้งจาก undistorted coordinates ไปยัง distorted coordinates เมื่อทราบค่าของสัมประสิทธิ์แล้วจะสามารถสร้างความผิดเพี้ยนให้กับรูปภาพได้ และยังสามารถกู้รูปภาพกลับคืนมาได้ด้วย

หากเราต้องการทำการสร้างความผิดเพี้ยนให้กับรูปภาพในรูปที่ 2-21(a) จะต้องทำการแทนค่าของพิกเซล (x_0,y_0) ใดๆ ลงในสมการที่ 5 และ 6 แล้วสร้างพิกัดจุด (x_0',y_0') ที่สอดคล้องกันขึ้น ค่าของ distorted image ที่จุดดังกล่าวจะเป็น $g(x_0',y_0') = f(x_0,y_0)$ หรือจะทำโดยใช้ Gray-level interpolation บนค่าของ f ด้วยค่าใกล้เคียงของ (x_0,y_0)

ผลของความผิดเพี้ยนที่เกิดขึ้นของรูปที่ 2-21(a) ซึ่งใช้แบบแผนของ The nearest neighbor gray-level assignment แสดงไว้ในรูปที่ 2-21(c) ถ้าเราใช้การแมปปิ้งตามสมการที่ 5 และ 6 ร่วมกับเทคนิคการกำหนดระดับ gray–level แบบ The nearest neighbor gray-level assignment ผลที่ได้จะเป็นดังรูปที่ 2-21(d) ซึ่งวิธีนี้ให้ผลที่ดีพอสมควร แต่จะมีความผิดพลาดเกิดขึ้นในการกำหนดระดับ gray –level โดยเฉพาะอย่างยิ่งที่บริเวณขอบระหว่างพื้นที่สีเทาและพื้นที่สีดำ รูปที่ 2-21(e) และ (f) แสดงผลที่ได้จากการใช้วิธี Bilinear gray-level assignment ซึ่งจะปรับปรุงปัญหาที่ขอบระหว่างพื้นที่สีเทาและพื้นที่สีดำให้ดีขึ้น



รูปที่ 2-21 การทำ Gray-level interpolation

2.3 การปรับแสง

เนื่องจากกล้องเวบแคมจะมีการปรับแสงให้โดยอัตโนมัติ ถ้าแสงโดยรวมของภาพมืดก็จะทำการปรับแสงให้มีความสว่างมากขึ้น หรือถ้าแสงโดยรวมสว่างก็จะปรับให้ความสว่างลดลง ดังนั้นจึงจำเป็นต้องปรับแสงของภาพให้กลับมาเท่ากับเมื่อเริ่มต้นโปรแกรม เพราะเมื่อเริ่มต้นโปรแกรมจะทำการจดจำสีนิ้วมือของผู้ใช้ด้วย ถ้าไม่ทำการปรับแสง สีนิ้วมือของผู้ใช้ก็จะเปลี่ยน ทำให้ไม่สามารถหานิ้วมือได้

ขั้นตอนการปรับแสง เมื่อเริ่มต้นโปรแกรมมีดังต่อไปนี้

1. หาจุดอ้างอิง โดยจะใช้บริเวณขอบหน้าจอด้านบน เป็นจุดอ้างอิง
2. จำค่า RGB ของจุดอ้างอิงนั้นไว้

ขั้นตอนการปรับแสง เมื่อรันโปรแกรมมีดังต่อไปนี้

1. หาตำแหน่งของจุดอ้างอิง ซึ่งเป็นตำแหน่งเดียวกับเมื่อเริ่มต้นโปรแกรม
2. นำค่า RGB ของจุดที่ได้ มาลบกับค่า RGB ของจุดอ้างอิง
3. นำผลลัพธ์ที่ได้ของแต่ละระนาบมาหาค่าเฉลี่ย แล้วเก็บไว้เป็นค่าอ้างอิง
4. นำค่าอ้างอิงที่ได้ไปลบกับทุกพิกเซลในภาพ

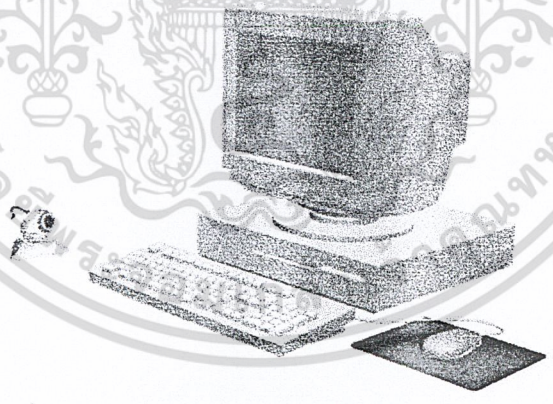
บทที่ 3
การสร้างจอสัมผัสโดยใช้เวบแคม

การสร้างจอสัมผัสโดยใช้เวบแคมจะใช้เวบแคม 1 ตัว ในการจับการเคลื่อนที่ของนิ้วมือ โดยจะวางไว้เยื้องกับหน้าจอ ซึ่งเวบแคมที่นำมาใช้ถ้ามีคุณภาพดี คือ มีความละเอียดของภาพมาก และมี เฟรมเรตสูง ก็จะทำให้หาตำแหน่งของปลายนิ้วได้แม่นยำขึ้น และการตอบสนองของโปรแกรมจะเร็วขึ้นตามไปด้วย

การพัฒนาโปรแกรมจะใช้ Visual C++ และใช้ ActiveX Control ที่ชื่อว่า VideoOCX ในการติดต่อกับเวบแคม อีกทั้ง VideoOCX ยังมีฟังก์ชันเกี่ยวกับการประมวลผลภาพให้ใช้งานอีกด้วย

3.1 การจัดวางตำแหน่งของเวบแคม

การจัดวางตำแหน่งของกล้องเวบแคมควรจะอยู่เยื้องกับหน้าจอด้านซ้าย หรือด้านขวา ขึ้นอยู่กับมือที่ผู้ใช้จะใช้ในการสั่งงาน ถ้าผู้ใช้ใช้มือซ้ายสั่งงาน กล้องควรวางเยื้องกับหน้าจอด้านขวา เพื่อไม่ให้แขนของผู้ใช้บังส่วนต่างๆ ของหน้าจอ ดังรูปที่ 3-1 ในทางกลับกันถ้าผู้ใช้ใช้แขนขวาสั่งงาน กล้องควรวางเยื้องกับหน้าจอด้านซ้าย โดยตำแหน่งของกล้องต้องสามารถเห็นหน้าจอได้ทั้งหมด และมุมระหว่างผู้ใช้กับหน้าจอ และกล้องกับหน้าจอต้องน้อยที่สุดเท่าที่จะเป็นไปได้ เพื่อที่จะให้ตำแหน่งเมาส์คลาดเคลื่อนจากมุมมองของผู้ใช้น้อยที่สุด



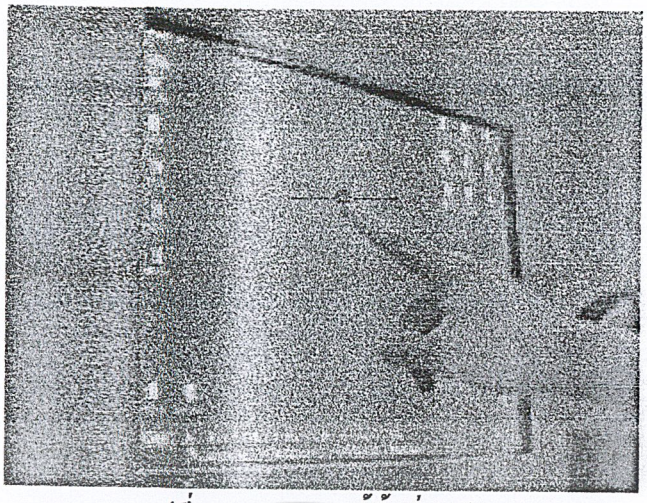
รูปที่ 3-1 ตำแหน่งการวางกล้องเมื่อผู้ใช้ใช้มือขวาสั่งงาน

3.2 รูปแบบการใช้งาน

ในการที่จะสั่งงานให้เมาส์เลื่อนตำแหน่ง เมาส์คลิ้ก ดับเบิลคลิ้ก หรือ Drag Drop จะต้องสั่งงานตามรูปแบบดังต่อไปนี้

3.2.1 เลื่อนตำแหน่งเมาส์

ในการเลื่อนตำแหน่งเมาส์จะต้องชี้นิ้วให้ปลายนิ้วอยู่ที่ตำแหน่งสูงที่สุด ดังรูปที่ 3-2



รูปที่ 3-2 แสดงการชี้นิ้วที่ถูกต้อง

3.2.2 เมาส์คลิ้ก

ในการทำเมาส์คลิ้กนั้นจะต้องชี้นิ้วให้อยู่ที่ตำแหน่งเดิมนานกว่า 1 วินาที หลังจากนั้นต้องเลื่อนนิ้วออกจากตำแหน่งเดิม เมื่อเลื่อนออกแล้วจะต้องกลับมาอยู่จุดเดิมภายใน 1 วินาที แล้วเลื่อนออกอีกครั้งภายใน 1 วินาที

3.2.3 ดับเบิลคลิ้ก

ในการทำดับเบิลคลิ้กจะต้องชี้นิ้วให้อยู่ที่ตำแหน่งเดิมนานกว่า 1 วินาที แล้วต้องเลื่อนนิ้วออกจากตำแหน่งเดิม เมื่อเลื่อนออกแล้วจะต้องกลับมาอยู่จุดเดิมภายใน 1 วินาที หลังจากนั้นก็จะต้องออกจากตำแหน่งเดิมอีกครั้งภายใน 1 วินาที แล้วกลับมาอยู่จุดเดิมภายใน 1 วินาที ก็จะถือว่าเป็นการดับเบิลคลิ้ก

3.2.4 Drag Drop

ในการ Drag นั้นจะต้องชี้นิ้วให้อยู่ที่ตำแหน่งเดิมนานกว่า 1 วินาที แล้วต้องเลื่อนนิ้วออกจากตำแหน่งเดิม เมื่อเลื่อนออกแล้วจะต้องกลับมาอยู่จุดเดิมภายใน 1 วินาที แล้วค้างไว้ 1 วินาที ก็จะถือว่าเป็นการเริ่มต้นการ Drag ผู้ใช้สามารถเลื่อนนิ้วเพื่อลากวัตถุไปยังตำแหน่งที่ต้องการ

ส่วนการ Drop จะต้องชี้นิ้วให้อยู่ที่ตำแหน่งเดิมนานกว่า 1 วินาที แล้วต้องเลื่อนนิ้วออกจากตำแหน่งเดิม เมื่อเลื่อนออกแล้วจะต้องกลับมาอยู่จุดเดิมภายใน 1 วินาที ซึ่งก็จะเหมือนกับการทำเมาส์คลิ้ก กระบวนการทำเช่นนี้ก็จะถือว่าเป็นการ Drop

3.3 การออกแบบระบบ

ระบบจะแบ่งออกเป็น 3 ส่วนใหญ่ๆ ดังรูปที่ 3-3 คือ

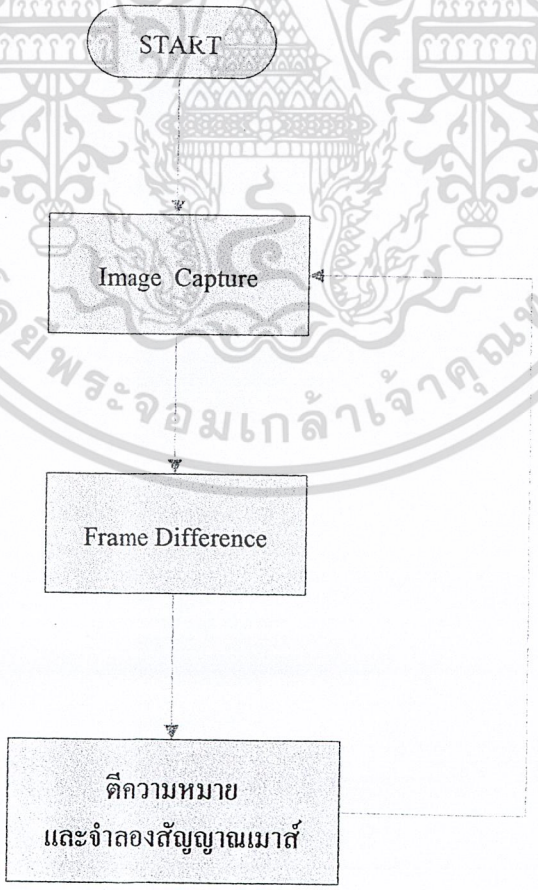
1. **Image Capture** เป็นส่วนที่ทำการดึงภาพมาจากเวบแคม

2. **Frame Difference** จะทำ Frame Difference หลังจากนั้นจะนำภาพจาก Frame Difference

3. **ตีความหมายและจำลองสัญญาณเมาส์** จะทำการหาตำแหน่งปลายนิ้ว และตีความหมายว่าผู้ใช้จะทำการเลื่อนเมาส์ คลิ๊กเมาส์ ดับเบิลคลิ๊ก หรือว่า Drag Drop แล้วทำการจำลองสัญญาณนั้นไปให้กับระบบปฏิบัติการทำงานต่อไป โดยการหาตำแหน่งปลายนิ้ว จะใช้ 2 วิธี คือ

- **Frame Difference** เป็นวิธีที่นำเอาเฟรมภาพปัจจุบันมาลบกับเฟรมภาพก่อนหน้านี้ โดยถ้าตำแหน่งนั้นๆ ไม่มีการเปลี่ยนแปลงก็จะมีค่าเป็นศูนย์ หรือสีดำ แต่ถ้าตำแหน่งนั้นๆ มีการเปลี่ยนแปลงก็จะมีค่ามากกว่าศูนย์ ซึ่งเราสามารถนำวิธีนี้มาหาตำแหน่งของปลายนิ้วขณะกำลังเคลื่อนที่ได้ แต่ถ้านิ้วไม่ได้เคลื่อนที่วิธีนี้จะไม่สามารถหาตำแหน่งของปลายนิ้วได้ เพราะทุกตำแหน่งจะมีค่าเป็นศูนย์
- หาสีปลายนิ้ว วิธีนี้จะทำการเก็บระดับสีของนิ้วมือเมื่อเริ่มต้นโปรแกรมไว้ก่อน แล้วนำค่าระดับสีมาหาตำแหน่งปลายนิ้วมือ

การหาตำแหน่งปลายนิ้วมือจะใช้ 2 วิธีนี้ร่วมกัน คือ จะนำตำแหน่งปลายนิ้วที่ได้ของแต่ละวิธีมาเปรียบเทียบตำแหน่งกันว่ามีค่าใกล้เคียงกันหรือไม่ ถ้าใกล้เคียงกันก็จะถือว่าตำแหน่งนั้นเป็นปลายนิ้ว



รูปที่ 3-3 Flow Chart ภาพรวมของระบบ

3.3.1 Image Capture

เป็นส่วนที่ทำการดึงภาพจากเวบแคมโดยจะใช้ ActiveX Controls ที่ชื่อว่า VideoOCX ไปติดต่อกับเวบแคม แล้วนำภาพที่ได้ไปประมวลผลในส่วนต่อไป

```
void CPreProgramDlg::OnInit()  // set interface of webcam
{
        if (m_Image != 0)
                return;

        m_video.SetErrorMessages(0);

        if (!m_video.Init())
        {
                // Error while trying to init control
                MessageBox(m_video.GetLastErrorString(), "VideoOCX Error", MB_OK);
                return;          // exit
        }

        m_Image = m_video.GetColorImageHandle();
        m_ResultImage = m_video.GetGrayImageHandle();

        Sleep(100);              // make sure VideoOCX has time to be properly initialized

        if (m_video.Start()){
                AfxBeginThread(CaptureThread, this);   // start capture thread
        }
}
```

ตารางที่ 3-1 ฟังก์ชัน OnInit()

ฟังก์ชัน OnInit() ไว้กำหนดค่าเริ่มต้นในการติดต่อกับเวบแคม โดย m_video เป็น handle ของ VideoOCX ที่จะไปเรียกฟังก์ชัน Init() เพื่อกำหนดค่าเริ่มต้นของ handle หลังจากนั้นจะไปเรียกฟังก์ชัน Start() เพื่อที่จะทำการเริ่มดึงภาพจากเวบแคม โดยกระบวนการในการดึงภาพจะทำงานแบบเธรด และมี m_Image เป็น handle ของภาพสี ส่วน m_ResultImage เป็น handle ของภาพขาวดำ

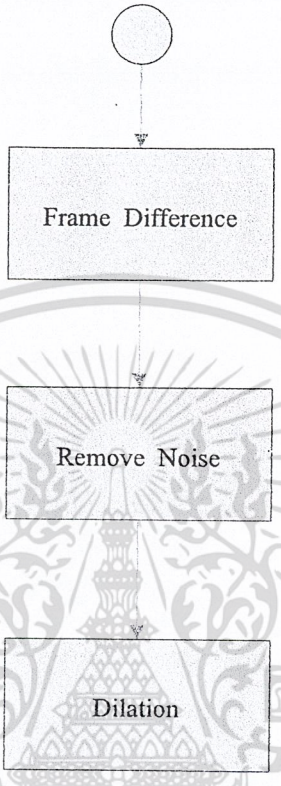
```
        m_video.Capture(m_Image)
        BYTE *data = (BYTE*)m_video.GetDataPointer(m_Image);
        int size = parent->m_video.GetImageDataSize(parent->m_Image);
        for (int i=0; i<size; i++) {
                data[i] = 255;
        }
```

ตารางที่ 3-2 การเข้าถึงแต่ละตำแหน่งในรูปภาพ

ในส่วนของการดึงภาพจากเวบแคมจะใช้ฟังก์ชัน Capture() ซึ่งภาพที่ได้นั้นจะเป็นภาพสีหรือภาพขาวดำขึ้นอยู่กับตัว handle ที่ส่งเข้าไป ส่วนฟังก์ชัน GetDatapointer() จะนำข้อมูลจาก handle มาแปลงให้เป็นไบต์ ถ้าเป็นภาพขาวดำก็จะแปลงเป็นค่า Gray level ส่วนถ้าเป็นภาพสีจะมีขนาดใหญ่ขึ้นเป็น 3 เท่า โดยจะเก็บค่าของ RGB แทนค่า Gray level ซึ่งในการเข้าถึงข้อมูลแต่ละตำแหน่งในรูปภาพจะใช้ฟังก์ชัน GetImageDataSize() ในการหาขนาดของรูปภาพ แล้วใช้ลูป for ไล่แต่ละตำแหน่ง โดย

ข้อมูลภาพที่เก็บจะเป็นแบบอาร์เรย์ ซึ่งเราสามารถเปลี่ยนแปลงค่าได้ จากรูปข้างบนจะเป็นการเปลี่ยนค่าให้ทุกตำแหน่งมีค่าเป็น 255 หรือสีขาว

3.3.2 Frame Difference



รูปที่ 3-4 Flow chart ส่วนของ Frame Difference

ในส่วนของ Frame Difference อธิบายโดยใช้ Flow Chart ในรูปที่ 3-4 ได้ดังนี้คือ

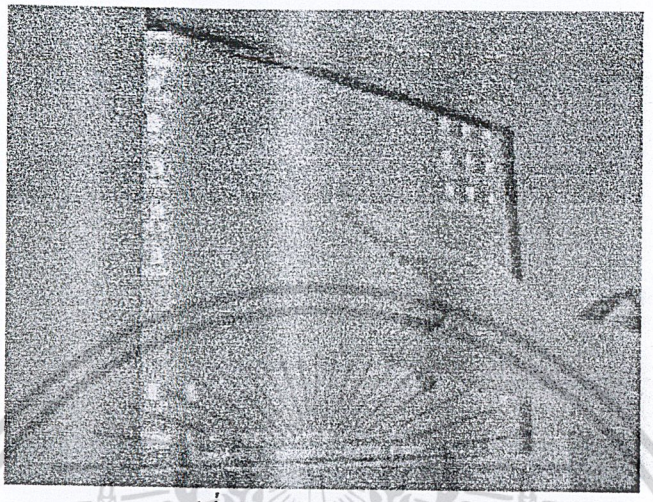
3.3.2.1 Frame Difference

การทำ Frame difference จะใช้ภาพสี 2 ภาพ คือ ภาพปัจจุบัน และภาพก่อนหน้า มาลบกัน

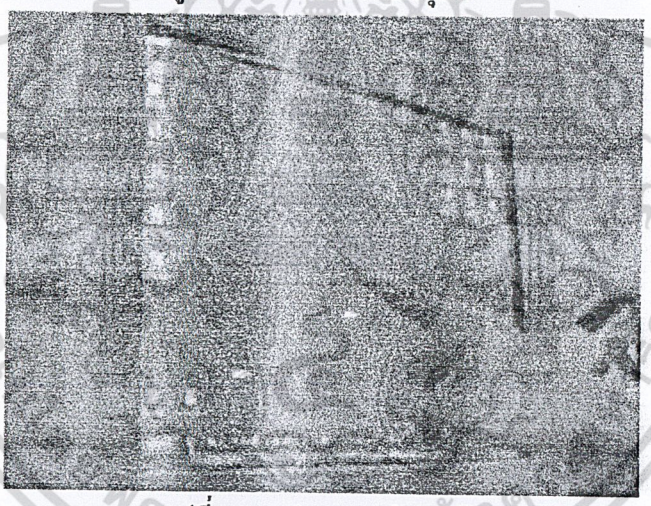
```
for (int i=0; i<Csize; i++) {
        int temp = (int)d2[i] - (int)d1[i];
        if (temp>255) {
                temp = 255;
        }else if (temp<0) {
                temp = -temp;
        }
        Cdata[i] = temp;

}
```

ตารางที่ 3-3 การทำ Frame Difference

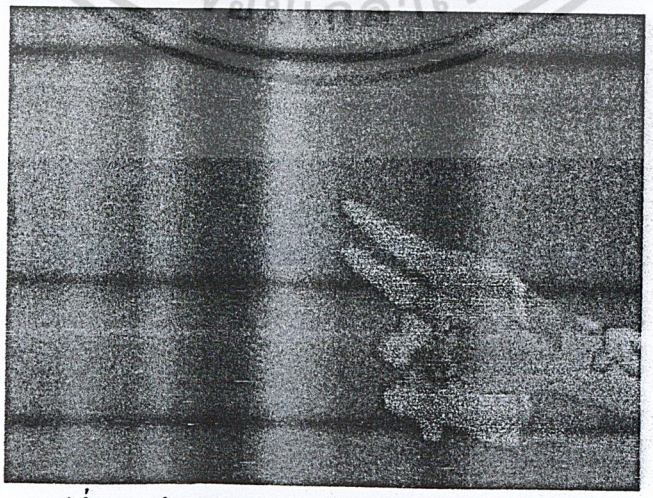
Source code ข้างบนเป็นการทำ Frame Difference เริ่มจากนำข้อมูลแต่ละตำแหน่งของทั้ง 2 ภาพมาลบกัน แล้วทำการตรวจสอบค่าที่ได้ถ้าเกิน 255 เปลี่ยนให้เป็น 255 แต่ถ้าค่าน้อยกว่า 0 ก็จะเปลี่ยนให้เป็นค่าบวก ผลจากการทำ Frame Difference แสดงได้ดังรูปข้างล่าง



รูปที่ 3-5 เฟรมภาพปัจจุบัน



รูปที่ 3-6 เฟรมภาพก่อนหน้า



รูปที่ 3-7 เฟรมภาพหลังจากการทำ Frame Difference

3.3.2.2 Noise Removing

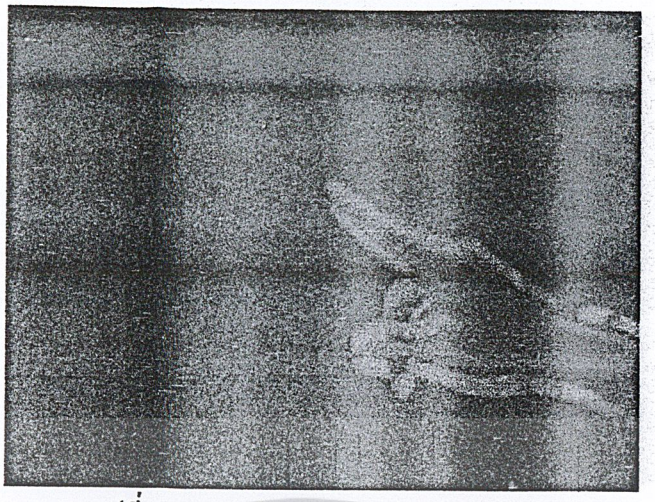
หลังจากที่ทำ Frame Difference แล้วภาพที่ได้จะมีข้อมูลภาพในส่วนที่ไม่ต้องการเกิดขึ้นเป็นจำนวนมาก ซึ่งส่วนใหญ่จะอยู่เดี่ยวๆ หรืออยู่เป็นกลุ่มเล็กๆ ถ้าไม่ทำการลบข้อมูลในส่วนนี้ทิ้งไปก่อนอาจจะทำให้หาตำแหน่งปลายนิ้วผิดพลาดได้ ดังนั้นในส่วนนี้จะทำการลบข้อมูลภาพในส่วนที่ไม่ต้องการทิ้งไป โดยจะใช้การพิจารณากับตำแหน่งรอบข้าง ถ้ารอบข้างของจุดนั้นๆ ไม่มีข้อมูลที่ต้องการอยู่ก็จะลบตำแหน่งนั้นทิ้งไป ในทางกลับกันถ้ารอบข้างของจุดนั้นๆ มีข้อมูลอยู่ก็จะไม่ทำการลบจุดนั้นทิ้ง

```
for (int i = 0; i < size; i++){
        if (data[i] > 40)
                data[i] = 1;
        else
                data[i] = 0;
        a[i] = data[i];
}
for (i = 0; i < y; i++){
        for (int j = 0; j < x; j++){
            data[(i*x)+j] = a[(i*x)+j] + a[((i-1)*x)+j] + a[((i-1)*x)+j-1]
                    + a[(i*x)+j-1] + a[((i+1)*x)+j-1] + a[((i+1)*x)+j]
                    + a[((i+1)*x)+j+1] + a[((i-1)*x)+j+1] + a[(i*x)+j+1];

                if (data[(i*x)+j] > 4)
                        data[(i*x)+j] = 255;
                else
                        data[(i*x)+j] = 0;
        }
}
```

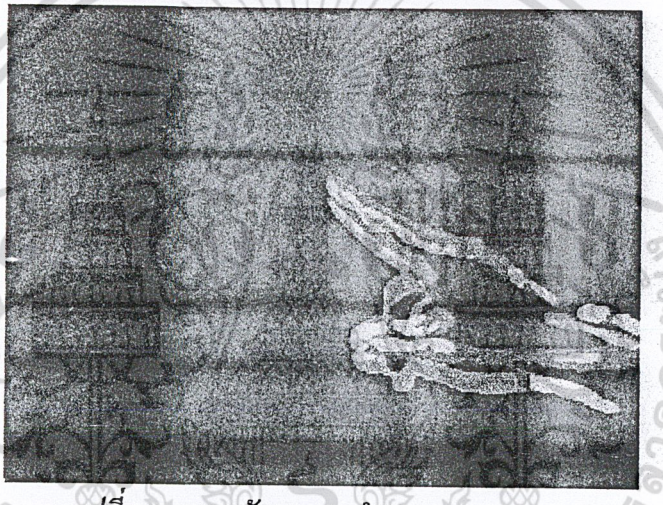
ตารางที่ 3-4 การทำ Noise Removing

จาก source code ข้างบน ลูป for อันบนจะเป็นการกำหนดค่าประจำตำแหน่งให้แต่ละตำแหน่ง โดยจะมีการกำหนดค่า threshold ขึ้นมา ซึ่งจะกำหนดเป็นเท่าไหร่แล้วแต่ความเหมาะสม ในที่นี้จะมีค่า 40 ซึ่งถ้าตำแหน่งนั้นมีค่ามากกว่า 40 จะทำการกำหนดค่าประจำตำแหน่งเป็น 1 แต่ถ้ามีค่าน้อยกว่า 40 จะกำหนดค่าประจำตำแหน่งเป็น 0

ส่วนลูป for อันล่างจะทำการพิจารณาจุดรอบข้าง 8 จุด แล้วนำค่าประจำตำแหน่งมาบวกกันทั้ง 9 จุด รวมตำแหน่งปัจจุบันด้วย และกำหนดค่า threshold = 4 โดยถ้าผลบวกออกมามีค่ามากกว่า 4 จะถือว่าตำแหน่งนั้นเป็นข้อมูลภาพที่ต้องการ แต่ถ้ามีค่าน้อยกว่า 4 จะถือว่าเป็นข้อมูลภาพที่ไม่ต้องการ ซึ่งการกำหนดค่า threshold อาจจะกำหนดมากกว่า หรือน้อยกว่า 4 ก็ได้ แต่ถ้าเราทำการกำหนดให้มีค่ามากจนเกินไป อาจจะไปลบในส่วนที่เป็นข้อมูลที่เราต้องการได้ และถ้ากำหนดให้น้อยจนเกินไป ข้อมูลที่เราไม่ต้องการอาจจะไม่ถูกลบอีกได้เช่นกัน ผลจากการทำ Noise Removing แสดงได้ดังรูปข้างล่าง



รูปที่ 3-8 ภาพก่อนการทำ Noise Removing



รูปที่ 3-9 ภาพหลังจากการทำ Noise Removing

3.3.2.3 Dilation

คือ การขยายพิกเซล โดยสามารถกำหนดได้ว่าให้ขยายออกในทิศทางใด ใช้เพื่อเพิ่มขนาดของจุดให้มีขนาดใหญ่ขึ้น เพราะว่าหลังจากการทำ Noise Removing นั้นอาจทำให้ขอบของรูปหายไป การเพิ่มขนาดก็จะทำให้การหาปลายนิ้วโดยใช้ Frame Difference หาได้ง่ายขึ้น ซึ่งจะทำให้เห็นความแตกต่างชัดเจนมากขึ้น ถึงแม้ว่าจะมีการเปลี่ยนแปลงของเฟรมที่เอามาเปรียบเทียบกันน้อยก็ตาม โดยใช้วินโดว์ขนาด 3*3 เป็นขอบเขตที่ใช้ในการขยายพิกเซล

0	0	0
0	255	0
0	0	0

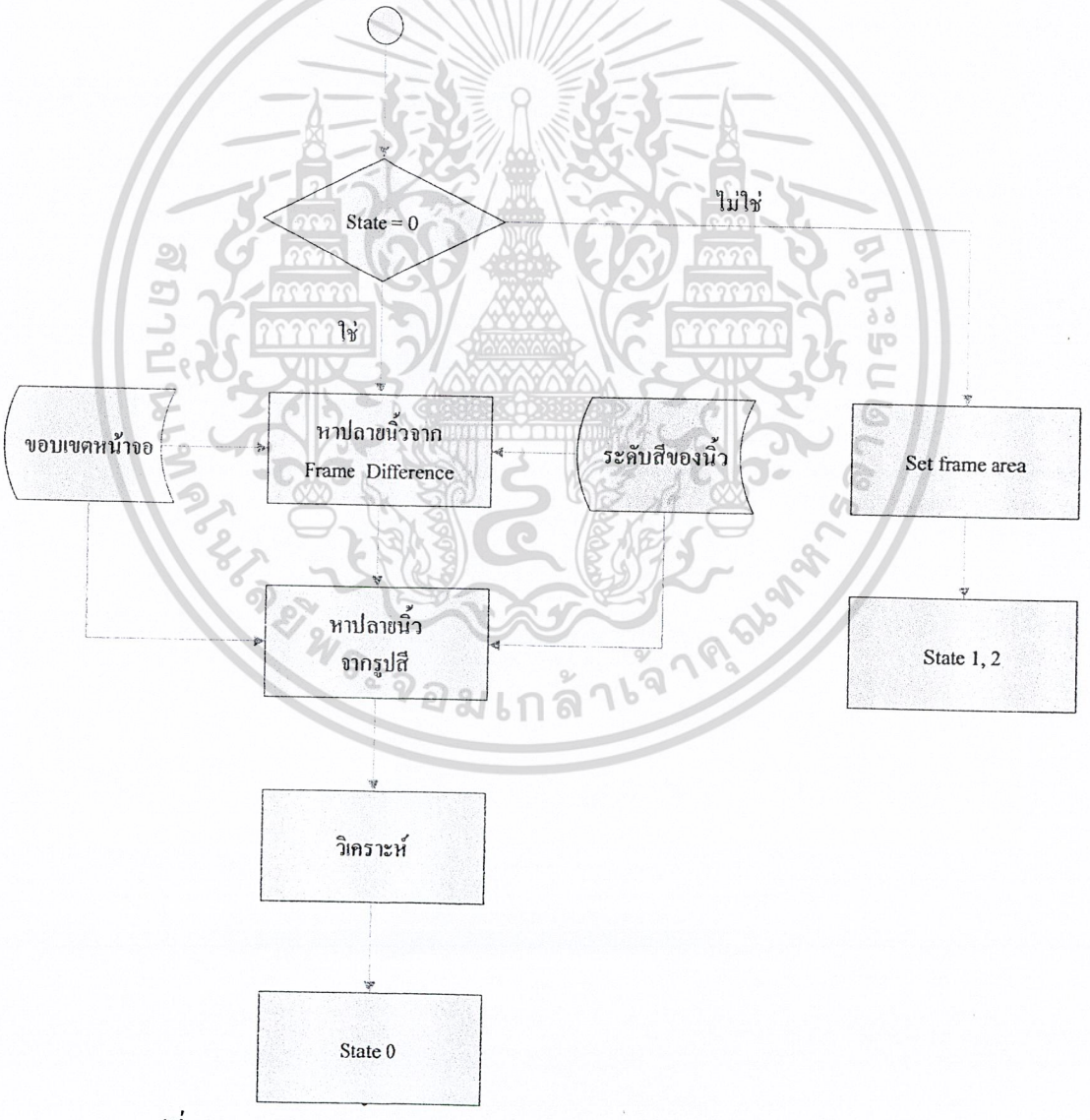
รูปที่ 3-10 ก่อนการทำ Dilation

255	255	255
255	255	255
255	255	255

รูปที่ 3-11 หลังการทำ Dilation

3.3.3 ตีความหมายและจำลองสัญญาณเมาส์

ในส่วนนี้จะทำการหาตำแหน่งปลายนิ้ว เพื่อนำมาทำการตีความหมายว่าผู้ใช้จะทำการเลื่อนเมาส์ คลิ๊กเมาส์ ดับเบิลคลิ๊ก หรือ Drag Drop แล้วจะทำการจำลองสัญญาณเมาส์ให้กับระบบปฏิบัติการทำงานต่อไป



รูปที่ 3-12 Flow chart ในส่วนของการตีความหมายและจำลองสัญญาณเมาส์

3.3.3.1 ขอบเขตหน้าจอ

เป็นข้อมูลที่เก็บไว้เมื่อเริ่มต้นโปรแกรม โดยจะเก็บขอบเขตของหน้าจอไว้เพื่อที่จะหาปลายนิ้วเฉพาะบริเวณนี้เท่านั้น ทำให้ไม่เสียเวลาในการหาปลายนิ้วมากจนเกินไป

```
bool CPreProgramDlg::InScreen(long point, int width, int height) // check point in screen
{
        int xxx = point % width;
        int yyy = point / width;
        int count = 0;

        if ( ((height/2)-Inf.LeftUpY) >= (Inf.m1*((width/2)-Inf.LeftUpX)) ) {
                if ((yyy-Inf.LeftUpY) >= (Inf.m1*(xxx-Inf.LeftUpX))) {
                        count++;
                }
        }else {
                if ((yyy-Inf.LeftUpY) <= (Inf.m1*(xxx-Inf.LeftUpX))) {
                        count++;
                }
        }

        if (((height/2)-Inf.LeftDownY) >= (Inf.m2*((width/2)-Inf.LeftDownX))) {
                if ((yyy-Inf.LeftDownY) >= (Inf.m2*(xxx-Inf.LeftDownX))) {
                        count++;
                }
        }else {
                if ((yyy-Inf.LeftDownY) >= (Inf.m2*(xxx-Inf.LeftDownX))) {
                        count++;
                }
        }

        if (((height/2)-Inf.LeftUpY) <= (Inf.m3*((width/2)-Inf.LeftUpX))) {
                if ((yyy-Inf.LeftUpY) <= (Inf.m3*(xxx-Inf.LeftUpX))) {
                        count++;
                }
        }else {
                if ((yyy-Inf.LeftUpY) >= (Inf.m3*(xxx-Inf.LeftUpX+20))) {
                        count++;
                }
        }

        if (((height/2)-Inf.RightUpY) >= (Inf.m4*((width/2)-Inf.RightUpX))) {
                if ((yyy-Inf.RightUpY) >= (Inf.m4*(xxx-Inf.RightUpX))) {
                        count++;
                }
        }else {
                if ((yyy-Inf.RightUpY) <= (Inf.m4*(xxx-Inf.RightUpX))) {
                        count++;
                }
        }

        if (count == 4) {
                return true;
        }
```

สมการเส้นตรง

1

2

3

4

```
        return false;
}
```

ตารางที่ 3-5 การหาขอบเขตของหน้าจอ

โดยที่ 1 คือ การตรวจสอบกับขอบจอด้านบน

2 คือ การตรวจสอบกับขอบจอด้านล่าง

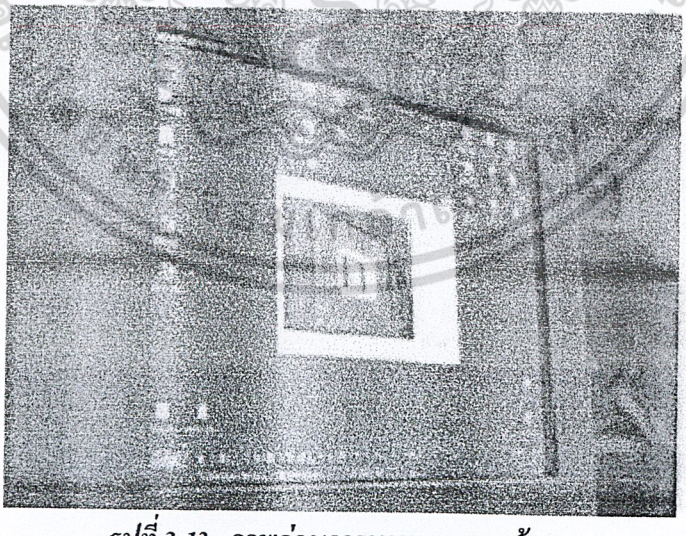
3 คือ การตรวจสอบกับขอบจอด้านซ้าย

4 คือ การตรวจสอบกับขอบจอด้านขวา

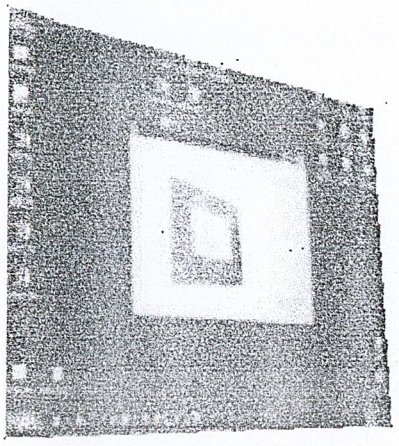
ตารางข้างบนเป็นฟังก์ชันที่ใช้ในการตรวจสอบว่าตำแหน่งปลายนิ้วอยู่ในจอหรือไม่โดยใช้วิธีการก็คือ มีการคำนวณสมการเส้นตรง 4 สมการที่ใช้ระบุขอบเขตของจอ และใช้จุดกลางจอเป็นจุดในการเปรียบเทียบว่า ควรจะใช้ค่ามากกว่าหรือน้อยกว่าในการเปรียบเทียบ

$$y - y_1 = m(x - x_1)$$

จากสมการจะเห็นว่า ถ้าใช้เครื่องหมายน้อยกว่า จะบอกไม่ได้ว่าจุดที่รับเข้ามานั้น อยู่ทางซ้ายหรือขวาของเส้นที่เป็นสมการจนกว่าจะรู้ค่า m ซึ่งก็คือความชัน ทำให้ไม่สามารถเขียนโปรมแกรมระบุได้ว่าให้ใช้เครื่องหมายมากกว่าหรือน้อยว่าในการเปรียบเทียบ จึงใช้จุดกลางจอเป็นจุดในการอ้างอิงเพื่อเลือกเครื่องหมายเพราะจุดกลางจอจะอยู่ในจอเสมอ ผลจากการหาขอบเขตหน้าจอสามารถแสดงได้ดังรูปข้างล่าง



รูปที่ 3-13 ภาพก่อนการหาขอบเขตหน้าจอ



รูปที่ 3-14 ภาพหลังจากการหาขอบเขตหน้าจอ

3.3.3.2 หาปลายนิ้วจาก Frame Difference

นำภาพที่ได้จากการทำ Frame Difference มาหาปลายนิ้ว ซึ่งจะทำการหาเฉพาะบนหน้าจอเท่านั้น โดยจะทำการหาตำแหน่งที่มีค่ามากกว่าศูนย์ที่อยู่สูงที่สุด ซึ่งจะถือว่าเป็นปลายนิ้วนั่นเอง ดังนั้นในการชี้ตำแหน่งบนหน้าจอ จะต้องทำการชี้ให้ปลายนิ้วคงอยู่ตำแหน่งสูงสุดด้วย ไม่เช่นนั้นตำแหน่งที่ได้จะไม่ใช่ตำแหน่งของปลายนิ้วมือ

```
if (PointFinger != -1) {
        long begin = PointFinger - (width*20) - 10;
        long end = PointFinger + (width*10) + 20;
        for (int i=begin; i<=end; i++) {
                if ( InScreen(i,width,height) ) {
                        if (Cdata[i*3] > ts) {
                                if (i > tempPoint) {
                                        tempPoint = i;
                                }
                        }else if (Cdata[i*3+1] > ts) {
                                if (i > tempPoint) {
                                        tempPoint = i;
                                }
                        }else if (Cdata[i*3+2] > ts) {
                                if (i > tempPoint) {
                                        tempPoint = i;
                                }
                        }
                }

                s--;
                if (s == 0) {
                        s = size;
                        i = i+(width-size);
                }
        }
}else {
        for (int i=0; i<Csize; i++) {
```

1

```
                if ( InScreen(i,width,height)  ) {
                        if (Cdata[i*3] > ts) {
                                if (i > tempPoint) {
                                        tempPoint = i;
                                }
                        }else if (Cdata[i*3+1] > ts) {
                                if (i > tempPoint) {
                                        tempPoint = i;
                                }
                        }else if (Cdata[i*3+2] > ts) {
                                if (i > tempPoint) {
                                        tempPoint = i;
                                }
                        }
                }
        }
}
```

2

ตารางที่ 3-6 การหาปลายนิ้วจากภาพที่ทำ frame difference

จาก source code ด้านบนจะแบ่งออกเป็น 2 ส่วน ดังนี้

1. ในส่วนที่ 1 จะทำงานเมื่อมีตำแหน่งปลายนิ้วค่าเก่าอยู่ หมายความว่า ถ้าเฟรมภาพก่อนหน้านี้สามารถหาตำแหน่งปลายนิ้วได้ ก็จะเข้ามาทำในส่วนนี้ เริ่มแรกจะใช้ตำแหน่งปลายนิ้วค่าเก่าเป็นจุดศูนย์กลางแล้วตีกรอบ ขนาด 30*30 พิกเซล เพื่อที่จะให้พื้นที่ในการหาลดลง หลังจากนั้นจะใช้ฟังก์ชัน InScreen() ในการตรวจสอบว่า ตำแหน่งของรูปภาพที่กำลังพิจารณาอยู่นั้นอยู่บนหน้าจอหรือไม่ ถ้าอยู่บนหน้าจอก็จะทำการตรวจสอบตำแหน่งนั้น ซึ่งในตำแหน่งหนึ่งๆ จะมีค่าอยู่ 3 ค่า คือ R, G, B เราจะทำการตรวจสอบทั้ง 3 ค่า ถ้าค่าใดค่าหนึ่งมีค่ามากกว่าค่า threshold ก็จะถือว่าตำแหน่งนั้นเกิดการเปลี่ยนแปลงขึ้น แล้วเก็บค่าตำแหน่งนั้นไว้ ถ้าตำแหน่งใดมีค่าพิกัดแกน Y สูงสุด จะถือว่าตำแหน่งนั้นเป็นปลายนิ้ว

2. ในส่วนที่ 2 จะทำงานเมื่อไม่มีตำแหน่งปลายนิ้วค่าเก่าอยู่ หมายความว่า ระบบไม่สามารถหาตำแหน่งปลายนิ้วในเฟรมที่แล้วได้ อาจเกิดจากไม่มีนิ้วปรากฏบนหน้าจอ หรือมีนิ้วปรากฏบนหน้าจอแต่เกิดข้อผิดพลาดในการตรวจสอบ ดังนั้นระบบจะต้องตรวจสอบหาตำแหน่งปลายนิ้วโดยทำการสแกนทั้งหน้าจอ และใช้วิธีการตรวจสอบเช่นเดียวกับกรณีข้างต้น

3.3.3.3 หาปลายนิ้วจากรูปสี

นำภาพที่ได้จากส่วน Image Capture มาหาตำแหน่งปลายนิ้ว โดยจะใช้ระดับสีของนิ้วมาเป็นข้อมูลในการหา โดยก่อนที่จะทำการหาจะต้องทำการปรับแสงภาพให้เท่ากับเมื่อเริ่มต้นโปรแกรมเสียก่อน มิฉะนั้นอาจตรวจสอบปลายนิ้วไม่พบ

```
long CPreProgramDlg::FindColorPoint(BYTE *Cdata, int size, int width, int height)
{
        int s = size; // 30*3
        long begin = (FDPointFinger - (width*20) - 10)*3;
        long end = (FDPointFinger + (width*10) + 20)*3;
        long tempPointC = -1;
```

```
for (int i=begin; i<=end; i++) {
        if ( (i%3) == 0 ) {
            // White Balance //
1           int ref1 = Inf.BaseB1 - tempdata[Inf.Base1];
            int ref2 = Inf.BaseG1 - tempdata[Inf.Base1+1];
            int ref3 = Inf.BaseR1 - tempdata[Inf.Base1+2];

2           Cdata[i] = Cdata[i] + ref1;
            Cdata[i+1]  = Cdata[i+1] + ref2;
            Cdata[i+2]  = Cdata[i+2] + ref3;

                                                    การปรับแสง

            if (InScreen(i/3,width,height)) {
                if (isFingerColor()){
                    if (i > tempPointC)
                        tempPointC = i;
                }
            }
    }
    return tempPointC;
}
```

หาปลายนิ้ว

ตารางที่ 3-7 หาปลายนิ้วจากภาพสี

ในการหาปลายนิ้วจากภาพสีนั้นเริ่มแรกจะนำค่าตำแหน่งปลายนิ้วที่ได้จากการทำ Frame Difference เป็นจุดศูนย์กลาง แล้วตีกรอบเพื่อหาปลายนิ้วขนาด 30*30 หลังจากนั้นจะต้องทำการปรับแสงภาพก่อน โดนการปรับแสงจะแบ่งออกเป็น 3 ส่วนดังนี้

1. เนื่องจากการปรับแสงนั้นจะต้องมีจุดไว้อ้างอิงก่อน ซึ่งจุดนี้จะหาได้เมื่อเริ่มต้นโปรแกรม โดยจะใช้ตำแหน่งขอบจอเป็นจุดอ้างอิง โดยในส่วนแรกเป็นการหาผลต่างของจุดอ้างอิงกับตำแหน่งอ้างอิงบนภาพปัจจุบัน

2. ส่วนที่ 2 จะนำค่าได้ มาบวกเข้ากับทุกตำแหน่งในกรอบขนาด 30*30

หลังจากนั้นจะใช้ฟังก์ชัน InScreen() เพื่อตรวจสอบการอยู่บนหน้าจอของตำแหน่งที่รับเข้ามา แล้วถ้าตำแหน่งนั้นมีสีใกล้เคียงกับระดับสีนิ้วมือ ก็จะเก็บตำแหน่งนั้นไว้ และถ้าตำแหน่งนั้นมีค่าพิกัดแกน Y มากสุดแล้วก็จะถือว่าเป็นปลายนิ้ว

3.3.3.4 วิเคราะห์

เป็นส่วนที่นำเอาตำแหน่งปลายนิ้วที่ได้จากการหาโดยใช้ Frame Difference และการหาโดยใช้สีมาเปรียบเทียบกัน ถ้าอยู่ห่างกันไม่เกิน 5 พิกเซล ก็จะถือว่าตำแหน่งนั้นคือปลายนิ้ว แต่ถ้าไม่ใกล้เคียงกันก็จะกลับไปในส่วน Image Capture

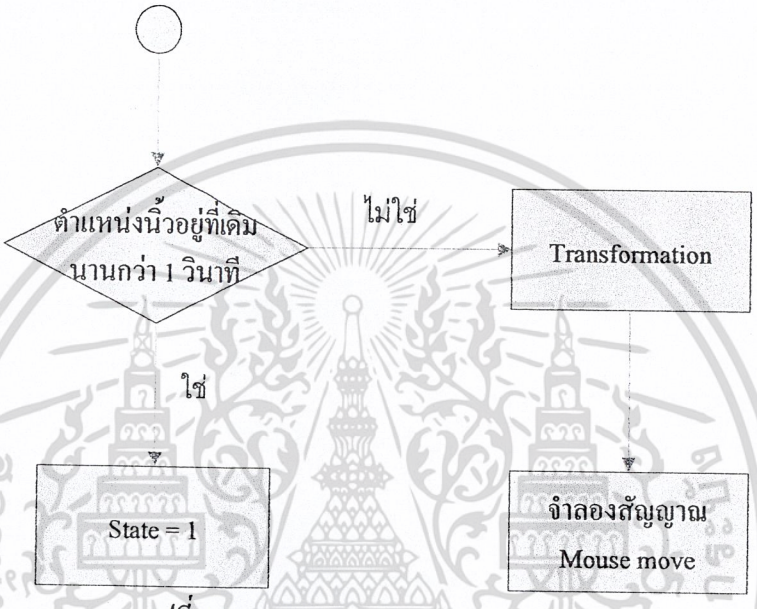
```
if ( ((Cx-Gx) >= -size) && ((Cx-Gx) <= size) &&
     ((Cy-Gy) >= -size) && ((Cy-Gy) <= size) ) {
        return true;
}else{
        return false;
```

```
}
```

ตารางที่ 3-8 ตรวจสอบความใกล้เคียงกันของจุด 2 จุด

3.3.3.5 ตรวจสอบการเลื่อนตำแหน่งเมาส์ (State 0)

State นี้จะตรวจสอบว่าถ้าตำแหน่งปลายนิ้วอยู่ที่เดิมนานกว่า 1 วินาที จะเปลี่ยน state เป็น state 1 แต่ถ้าอยู่ที่เดิมน้อยกว่า 1 วินาที จะถือว่าเป็นการเลื่อนตำแหน่งของเมาส์ ก็จะไปทำการจำลองสัญญาณ mouse move ดังรูปที่ 3-15



รูปที่ 3-15 Flow chart การทำงานของ state 0

```
if (PointFinger == FDPointFinger) {
        countBClick++;
        if (countBClick >= Inf.FrameRate) {
                m_state = 1;
                TempClickPoint = FDPointFinger;
        }
}else{
        PointFinger = FDPointFinger;
        SetCursorPos(PosX,PosY);
}
```

ตารางที่ 3-9 การทำงานของ state 0

เริ่มแรกจะทำการตรวจสอบว่าถ้าตำแหน่งปลายนิ้วปัจจุบัน เท่ากับตำแหน่งปลายนิ้วของเฟรมภาพก่อนหน้า ซึ่งก็คือปลายนิ้วอยู่ที่เดิม แล้วก็จะทำการตรวจสอบว่าถ้าปลายนิ้วอยู่ที่เดิมนานกว่า Inf.FrameRate ซึ่งก็คือ ค่าเฟรมเรตของเวบแคม ก็จะเปลี่ยน state เป็น state 1 แล้วเก็บตำแหน่งนั้นไว้ แต่ถ้าตำแหน่งปลายนิ้วไม่อยู่ที่เดิม จะเรียกฟังก์ชัน SetCursorPos(x,y) ซึ่งจะทำการเลื่อนตำแหน่งเมาส์ไปอยู่ในตำแหน่งที่ใส่ค่าเข้าไป

3.3.3.6 Geometric Transformation

เป็นส่วนที่ทำการแปลงตำแหน่งปลายนิ้วที่ได้ไปเป็นตำแหน่งบนหน้าจอจริงๆ โดยการแปลงจะใช้วิธี Geometric Transformations ซึ่งจะใช้สมการดังต่อไปนี้

$$r(x,y) = c_1x^3 + c_2y^3 + c_3x^2y + c_4xy^2 + c_5xy + c_6x + c_7y + c_8$$

$$s(x,y) = c_9x^3 + c_{10}y^3 + c_{11}x^2y + c_{12}xy^2 + c_{13}xy + c_{14}x + c_{15}y + c_{16}$$

$$\begin{pmatrix} x_1^3 & y_1^3 & x_1^2y_1 & x_1y_1^2 & x_1y_1 & x_1 & y_1 & 1 \\ x_2^3 & y_2^3 & x_2^2y_2 & x_2y_2^2 & x_2y_2 & x_2 & y_2 & 1 \\ x_3^3 & y_3^3 & x_3^2y_3 & x_3y_3^2 & x_3y_3 & x_3 & y_3 & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ x_8^3 & y_8^3 & x_8^2y_8 & x_8y_8^2 & x_8y_8 & x_8 & y_8 & 1 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ \cdot \\ \cdot \\ \cdot \\ c_8 \end{pmatrix} = \begin{pmatrix} x_1' \\ x_2' \\ x_3' \\ \cdot \\ \cdot \\ \cdot \\ x_8' \end{pmatrix}$$

a1[8][8] result[8] b1[8]

โดยตำแหน่งของ $x_1, x_2, \ldots, x_8$ และ $x_1', x_2', \ldots, x_8'$ แสดงได้ดังรูปที่ 3-16 ซึ่งรูปด้านซ้ายเป็นรูปหน้าจอที่เวบแคมมองเห็น ส่วนรูปด้านขวาเป็นรูปหน้าจอจริงๆ

รูปที่ 3-16 การแปลงหน้าจอที่เวบแคมมองเห็น มาเป็นหน้าจอจริงๆ

จากสมการข้างบนสามารถหาค่าคงที่ $c_1, c_2, \ldots, c_8$ ได้โดยใช้ source code ในตารางที่ 3-10 ซึ่งตัวแปรต่างๆ จะเป็นไปตามรูปแบบการคูณกันของเมทริกซ์ด้านล่าง

```
double a1[8][8] = { ... };
double b1[8] = { ... };
double result[8];
double m = 0;
```

```
int n = 8;

for (int i=0; i<=(n-2); i++) {
        for (int j=i+1; j<=(n-1); j++) {
                m = a1[j][i] / a1[i][i];
                a1[j][i] = 0;
                b1[j] = b1[j] - m*b1[i];
                for (int k=i+1; k<=(n-1); k++) {
                        a1[j][k] = a1[j][k] - m*a1[i][k];
                }
        }
}

Inf->result[n-1] = b1[n-1] / a1[n-1][n-1];

for (i=(n-2); i>=0; i--) {
        for (int j=i+1; j<=(n-1); j++) {
                b1[i] = b1[i] - a1[i][j]*Inf->result[j];
                Inf->result[i] = b1[i] / a1[i][i];
        }
}
```

ตารางที่ 3-10 แสดงการหาค่าคงที่ $c_1, c_2 \ldots, c_8$

3.3.3.7 ตรวจสอบการคลิ๊กเมาส์ (State 1)

หลังจากที่ตรวจสอบแล้วว่าตำแหน่งปลายนิ้วอยู่ที่เดิมนานกว่า 1 วินาที ก็จะเข้ามาทำงานใน state 1 ซึ่งจะทำการตรวจสอบการออกจากตำแหน่งเดิม และการกลับมาที่ตำแหน่งเดิมของปลายนิ้ว โดยสามารถอธิบายได้ด้วยรูปที่ 3-17 ซึ่งมีขั้นตอนดังต่อไปนี้

- ตรวจสอบว่าตำแหน่งปลายนิ้วเลื่อนออกจากกรอบสี่เหลี่ยมขนาด 13*13 พิกเซล หรือไม่ดังสมการด้านล่าง

$$x-6 <= x' <= x+6$$
$$y-6 <= y' <= y+6$$

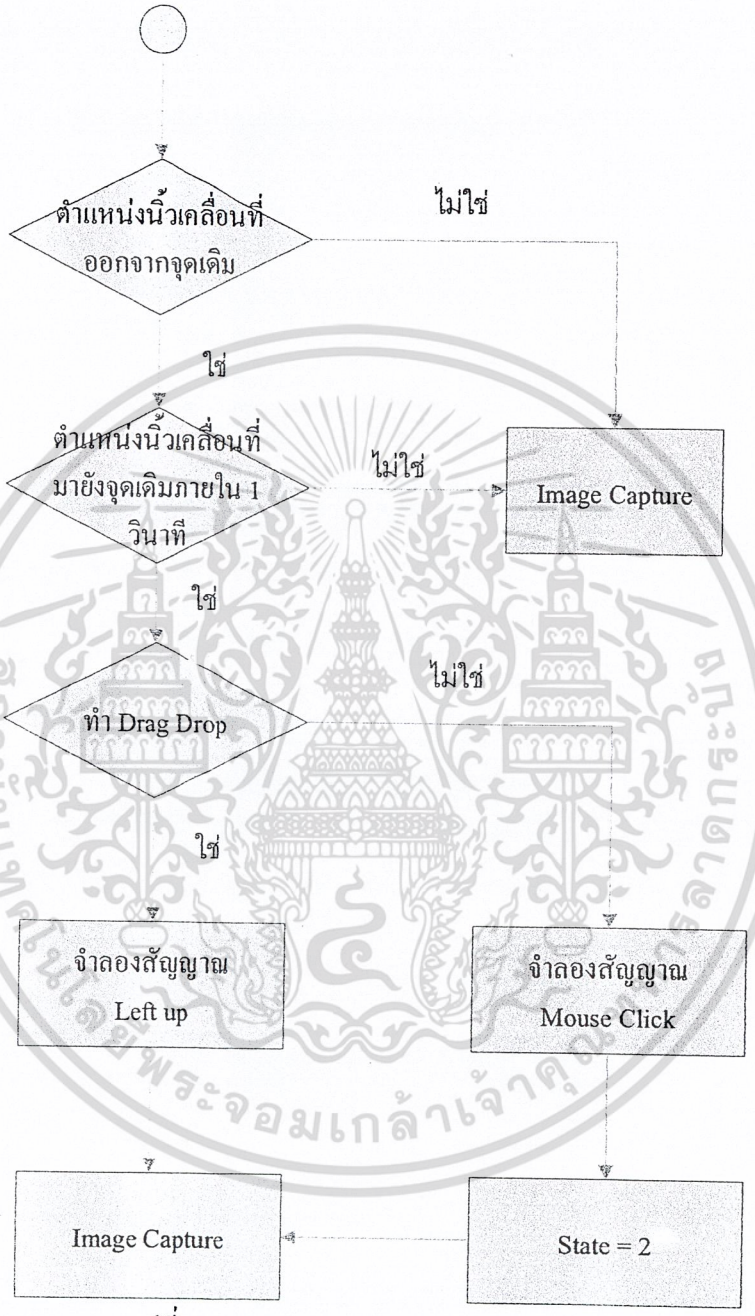
โดยที่ (x,y) คือ ตำแหน่งเดิม

(x′,y′) คือ ตำแหน่งปัจจุบัน

ถ้าใช่ก็จะไปตรวจสอบในขั้นตอนต่อไป แต่ถ้าไม่ใช่ก็จะกลับไปในส่วน Image Capture ซึ่งจะไปดึงรูปจากเวบแคมมาใหม่

- ตรวจสอบว่าตำแหน่งปลายนิ้วกลับมายังจุดที่ค้างไว้เดิมภายใน 1 วินาทีหรือไม่ ถ้าใช่ก็จะไปตรวจสอบต่อในขั้นตอนต่อไป แต่ถ้าไม่ใช่ก็จะกลับไปในส่วน Image Capture

- ตรวจสอบว่าก่อนหน้านั้นมีการทำ Drag ค้างไว้หรือไม่ ถ้าใช่จะถือว่าเป็นการ Drop ก็จะทำการจำลองสัญญาณ Left up แต่ถ้าไม่ใช่ก็จะทำการจำลองสัญญาณเมาส์คลิ้กและเปลี่ยนไป state 2



รูปที่ 3-17 Flow chart การทำงานของ state 1

```
if (outClick) {
        CountScreen = 0;
        countClick++;
        if (countClick <= Inf.FrameRate) {
                if (CheckPointClose(TempClickPoint,PointFinger)) {
                        if (DragDrop) {
                                m_state = 0;
                                MouseLeftUP();
                        }else{
                                m_state = 2;
                                outClick = false;
                                MouseClick();
                        }
                }
        }else {
                if (DragDrop) {
                        m_state = 0;
                        DragDrop = true;
                }else {
                        m_state = 0;
                }
        }
}else {
        if (!CheckPointClose(TempClickPoint,PointFinger))
                outClick = true;
}
```

ตารางที่ 3-11 การทำงานใน state 1

ในการจำลองสัญญาณ Left up สามารถทำได้โดยใช้ source code ในตารางที่ 3-12 โดยจะใช้ฟังก์ชัน SendInput () ซึ่งเป็นฟังก์ชั่นในการจำลองสัญญาณคีย์บอร์ด และเมาส์ซึ่งจะไม่ใช่การส่งเมสเซสไปหาระบบปฏิบัติการ แต่เป็นการจำลองสัญญาณทางฮาร์ดแวร์ คือเราสามารถสร้างสัญญาณการคลิกส์เมาส์หรือกดคีย์บอร์ดโดยที่ไม่ต้องกดที่เมาส์หรือกดปุ่มคีย์บอร์ดเลย ซึ่งในการใช้ฟังก์ชัน SendInput() จะต้องกำหนด type ให้เป็น INPUT_MOUSE และกำหนด Flags เป็น MOUSEEVENTF_LEFTUP

```
in.type = INPUT_MOUSE;
in.mi.dx = point->x;
in.mi.dy = point->y;
in.mi.dwFlags = MOUSEEVENTF_LEFTUP;
in.mi.mouseData = 0;
in.mi.time = 0;
SendInput(1,&in,sizeof(in));
```

ตารางที่ 3-12 การจำลองสัญญาณ Left up

ในการจำลองสัญญาณเมาส์คลิ๊ก จะต้องใช้ฟังก์ชัน SendInput() 2 ครั้ง โดยครั้งแรกจะส่งสัญญาณ Left down แล้วตามด้วยสัญญาณ Left up ซึ่งก็จะถือว่าเป็นการคลิกส์ 1 ครั้ง

```
in.type = INPUT_MOUSE;
in.mi.dx = point->x;
in.mi.dy = point->y;
in.mi.dwFlags = MOUSEEVENTF_LEFTDOWN;
in.mi.mouseData = 0;
in.mi.time = 0;
SendInput(1,&in,sizeof(in));
```

Left down

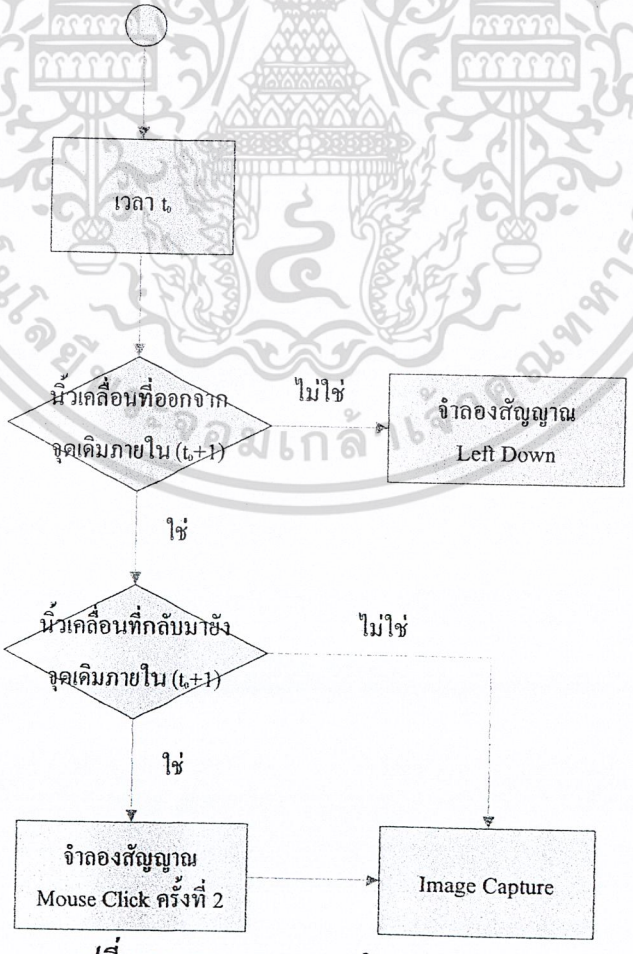
```
in.type = INPUT_MOUSE;
in.mi.dx = point->x;
in.mi.dy = point->y;
in.mi.dwFlags = MOUSEEVENTF_LEFTUP;
in.mi.mouseData = 0;
in.mi.time = 0;
SendInput(1,&in,sizeof(in));
```

Left up

ตารางที่ 3-13 การจำลองสัญญาณเมาส์คลิ๊ก

3.3.3.8 ตรวจสอบการดับเบิลคลิ๊กและการ Drag (State 2)

หลังจากที่ชี้นิ้วที่อยู่ตำแหน่งเดิมนานกว่า 1 วินาที แล้วออกจากจุดเดิม และกลับเข้ามาภายใน 1 วินาที ก็จะเข้ามาทำงานใน state 2 ซึ่ง state นี้จะทำตรวจสอบการดับเบิลคลิ๊กเมาส์ และการ Drag โดยมีขั้นตอนการทำงานดังนี้



รูปที่ 3-18 Flow chart การทำงานของ state 2

- ตรวจสอบว่าตำแหน่งนิ้วเคลื่อนที่ออกจากจุดเดิม และกลับมาภายใน 1 วินาทีหรือไม่ ถ้าใช่จะเป็นการทำดับเบิลคลิ้ก
- ถ้านิ้วอยู่ตำแหน่งเดิมเกิน 1 วินาทีแล้วก็จะถือว่าเป็นการ Drag

```
if (countDBClick <= Inf.FrameRate) {
        if (outClick) {
                outClick = false;
                MouseClick();
                m_state = 0;
        }
}else {
        PointFinger = -1;
        if (!outClick) {
                DragDrop = true;
                m_state = 0;
                outClick = false;
                MouseLeftDowd();
        }else {
                PointFinger = -1;
                m_state = 0;
        }
}
```

ตารางที่ 3-14 การทำงานใน state 2

ในการจำลองสัญญาณดับเบิลคลิ้ก จะต้องส่งสัญญาณ Left down และ Left up ตามลำดับอีกหนึ่งครั้ง เนื่องจากใน State 1 มีการส่งสัญญาณเมาส์คลิ้กไปแล้วหนึ่งครั้ง และใน State นี้ส่งไปอีกหนึ่งครั้งติดต่อกัน ก็จะถือว่าเป็นดับเบิลคลิ้ก

```
in.type = INPUT_MOUSE;
in.mi.dx = point->x;
in.mi.dy = point->y;
in.mi.dwFlags = MOUSEEVENTF_LEFTDOWN;
in.mi.mouseData = 0;
in.mi.time = 0;
SendInput(1,&in,sizeof(in));
```

ตารางที่ 3-15 การจำลองสัญญาณ Left down

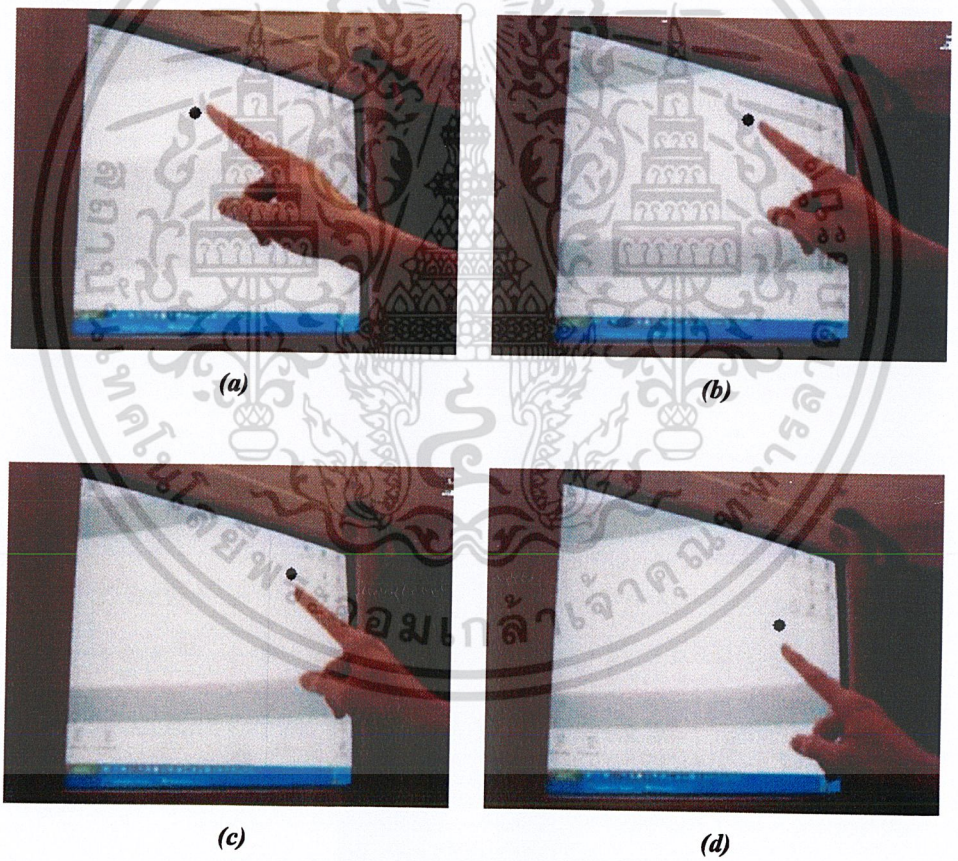
บทที่ 4

ผลการทดลอง

4.1 การทดลองการใช้งานโปรแกรมโดยสังเกตการเคลื่อนที่ของเมาส์

ในการทดลองในหัวข้อนี้จะทำการรันโปรแกรมแล้วเลื่อนตำแหน่งปลายนิ้วไปยังจุดต่างๆ บนหน้าจอ แล้วสังเกตการเคลื่อนที่ของเมาส์ว่าเป็นไปตามตำแหน่งที่ผู้ใช้ชี้หรือไม่ โดยในการทดลองจะทำการเปลี่ยนพื้นหลังของหน้าจอให้มีความหลากหลาย คือ เปลี่ยนพื้นหลังให้มีลวดลายต่างๆ และพื้นหลังที่เป็นสีเดียวทั้งหน้าจอ โดยการทดลองจะแบ่งพื้นหลังออกเป็น 10 แบบ ดังต่อไปนี้

4.1.1 พื้นหลังสีขาว



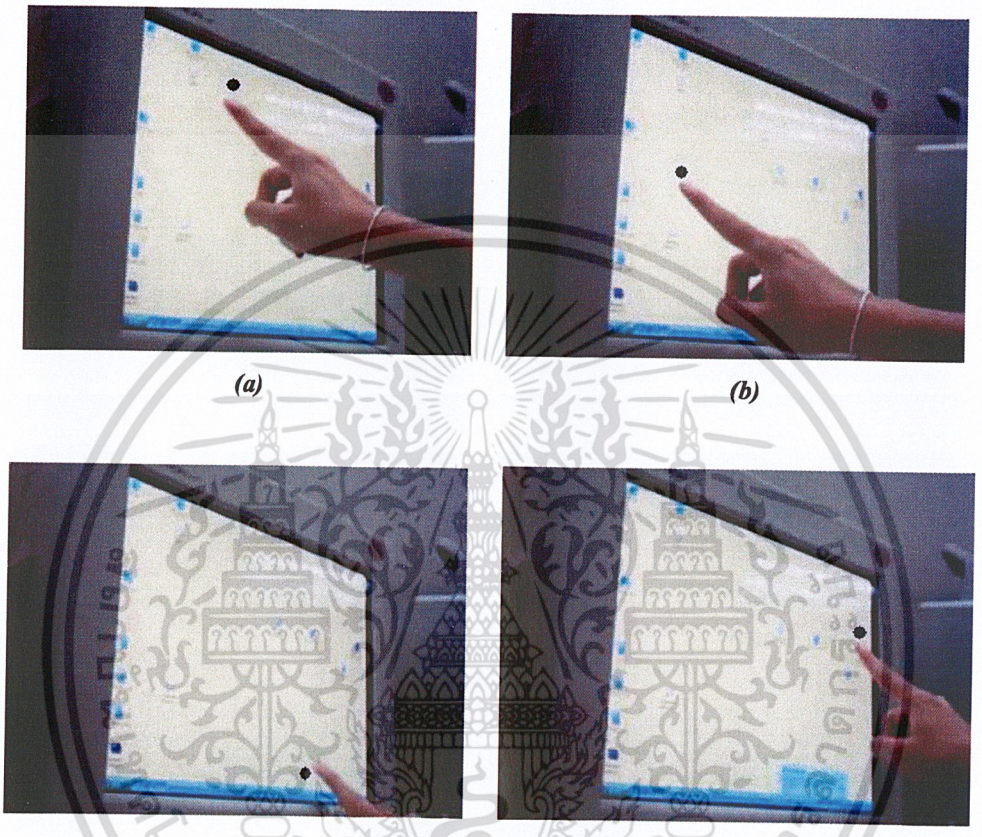
(a) *(b)* *(c)* *(d)*

รูปที่ 4-1 การเลื่อนตำแหน่งเมาส์บนพื้นหลังสีขาว

จากรูปที่ 4-1 จะใช้พื้นหลังเป็นสีขาว แล้วทำการเลื่อนนิ้วไปยังจุดต่างๆ บนหน้าจอ ซึ่งรูปที่ได้ในแต่ละเฟรมนั้นจะมีระดับสีต่างจากเมื่อเริ่มโปรแกรมเพียงเล็กน้อยเท่านั้น เนื่องจากสีพื้นเมื่อเริ่มโปรแกมเป็นสีเทาอ่อน ทำให้การหาตำแหน่งของปลายนิ้วจากการทำ Frame difference และการหาระดับสีของนิ้วหาได้อย่างสม่ำเสมอและได้ตำแหน่งที่ใกล้กัน จึงทำให้การเคลื่อนที่ของเมาส์นั้นเป็นไปอย่างต่อเนื่อง แต่

จะเห็นได้ว่าตำแหน่งที่เมาส์อยู่กับที่ปลายนิ้วชี้จะห่างกัน สาเหตุส่วนหนึ่งเป็นเพราะรูปที่ได้อยู่ในมุมมองของเวบแคม

4.1.2 พื้นหลังสีเหลือง



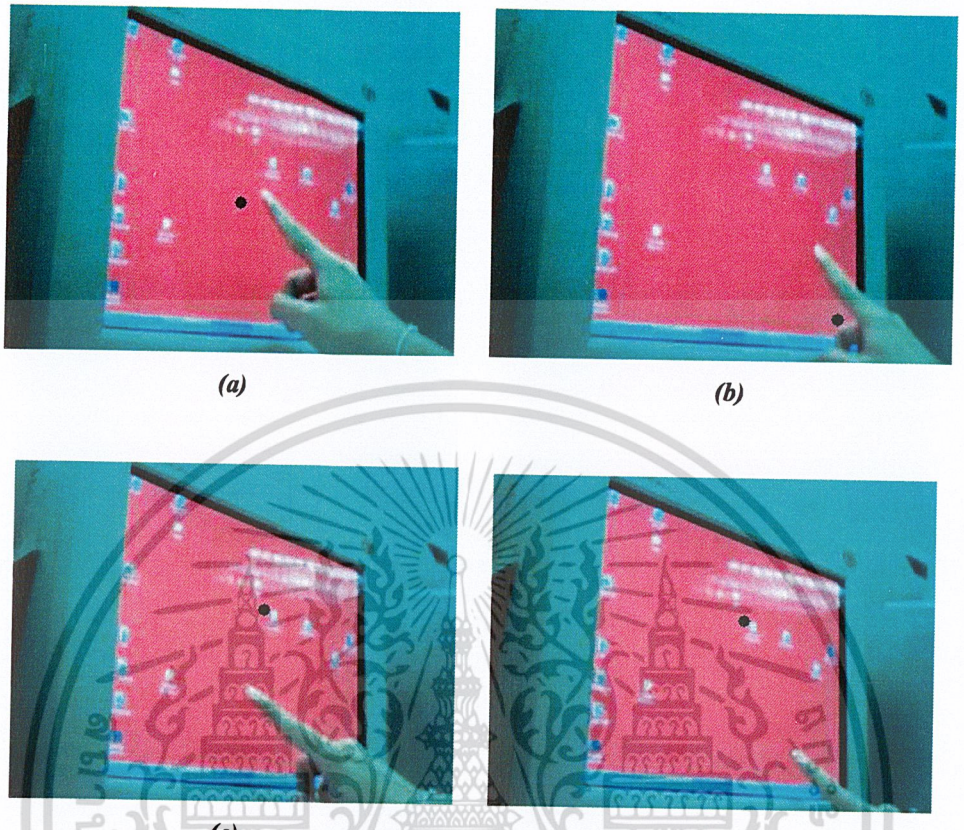
(a) *(b)*

(c) *(d)*

รูปที่ 4-2 การเลื่อนตำแหน่งเมาส์บนพื้นหลังสีเหลือง

จากรูปที่ 4-2 จะใช้พื้นหลังเป็นสีเหลือง ซึ่งรูปที่ได้ในแต่ละเฟรมนั้นจะอ่อนลงมาก เนื่องจากสีเหลืองที่ใช้เป็นสีที่สว่างมากเวบแคมจึงได้ทำการปรับแสงเพื่อลดความสว่างของภาพลงมา ระดับของสีนิ้วจึงมีความแตกต่างกับเมื่อเริ่มโปรแกรม แต่ในส่วนของโปรแกรมที่สร้างขึ้นมาได้ทำการปรับแสงให้กลับไปคล้ายกับเมื่อเริ่มโปรแกรม และเนื่องจากสีพื้นหลังต่างจากสีของนิ้วมือมาก ทำให้การหาตำแหน่งของปลายนิ้วหาได้อย่างสม่ำเสมอ และใกล้เคียงกับที่ผู้ใช้ชี้อยู่ จึงทำให้การเคลื่อนที่ของเมาส์นั้นเป็นไปอย่างต่อเนื่อง

4.1.3 พื้นหลังสีแดง



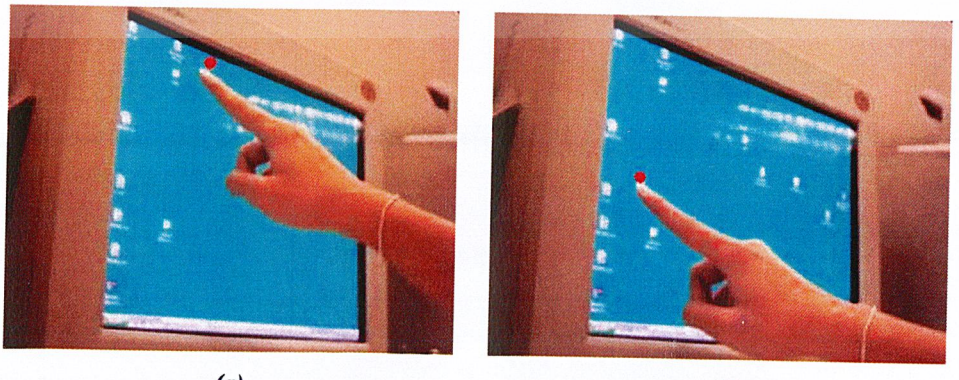
(a) *(b)*

(c) *(d)*

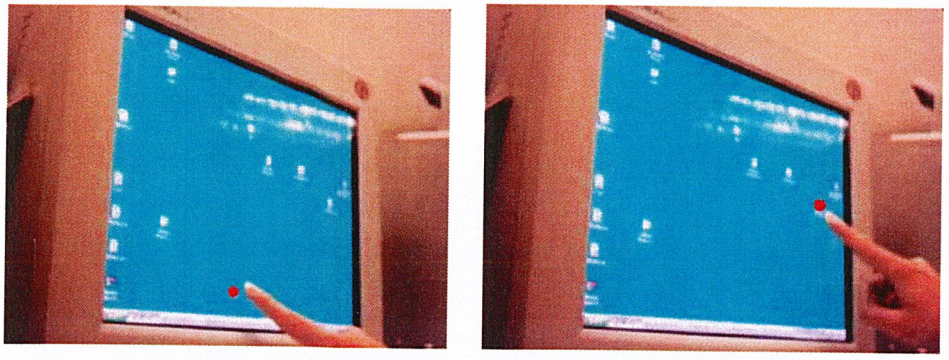
รูปที่ 4-3 การเลื่อนตำแหน่งเมาส์บนพื้นหลังสีแดง

จากรูปที่ 4-3 จะใช้พื้นหลังเป็นสีแดง ซึ่งเมื่อทำการปรับแสงแล้วระดับสีของนิ้วมือที่ได้ไม่ใกล้เคียงกับเมื่อเริ่มต้นโปรแกรม และหา Frame Difference ได้น้อยมาก ทำให้ไม่สามารถหาตำแหน่งของปลายนิ้วได้ จึงมีผลให้การเคลื่อนที่ของเมาส์นั้นไม่สามารถทำได้

4.1.4 พื้นหลังสีเขียว



(a) *(b)*

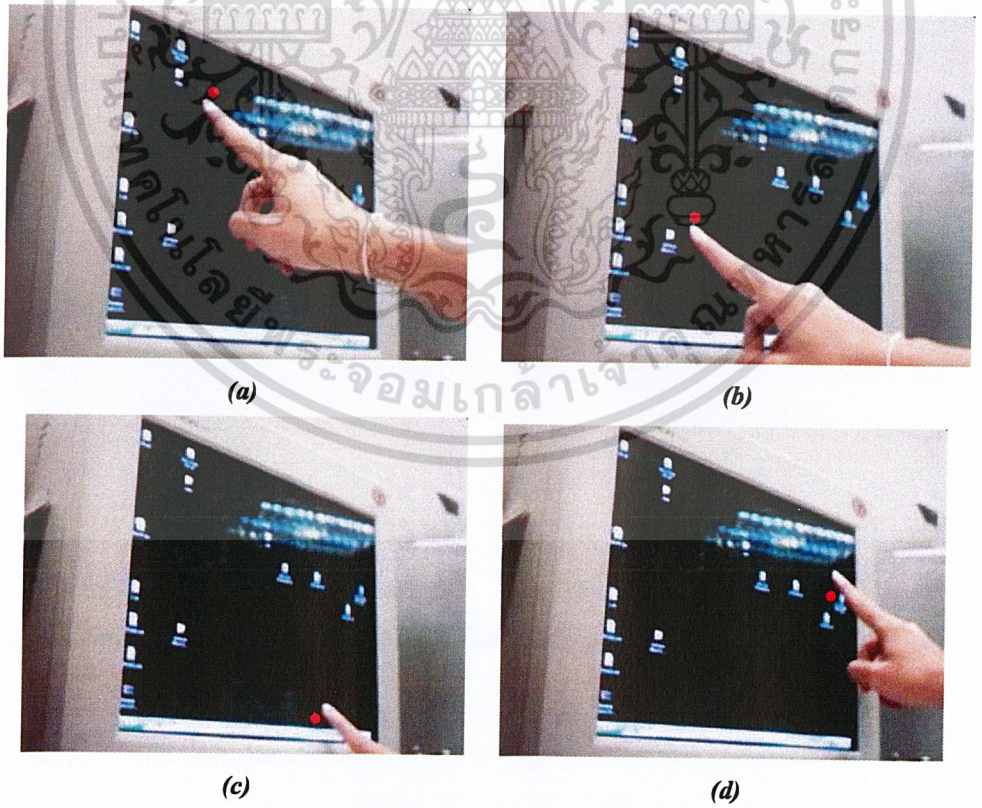


(c) *(d)*

รูปที่ 4-4 การเลื่อนตำแหน่งเมาส์บนพื้นหลังสีเขียว

จากรูปที่ 4-4 จะใช้พื้นหลังเป็นสีเขียว ซึ่งเมื่อทำการปรับแสงแล้วสีนิ้วมือจะกลับมาใกล้เคียงกับเมื่อเริ่มต้นโปรแกรมได้ ทำให้การหาตำแหน่งของปลายนิ้วจึงมีผลคล้ายกับแบบที่ 2 ทำให้การเคลื่อนที่ของเมาส์นั้นสามารถทำได้อย่างสม่ำเสมอ และใกล้เคียงกับที่ผู้ใช้ชี้อยู่ จึงทำให้การเคลื่อนที่ของเมาส์นั้นเป็นไปอย่างต่อเนื่อง

4.1.5 พื้นหลังสีดำ



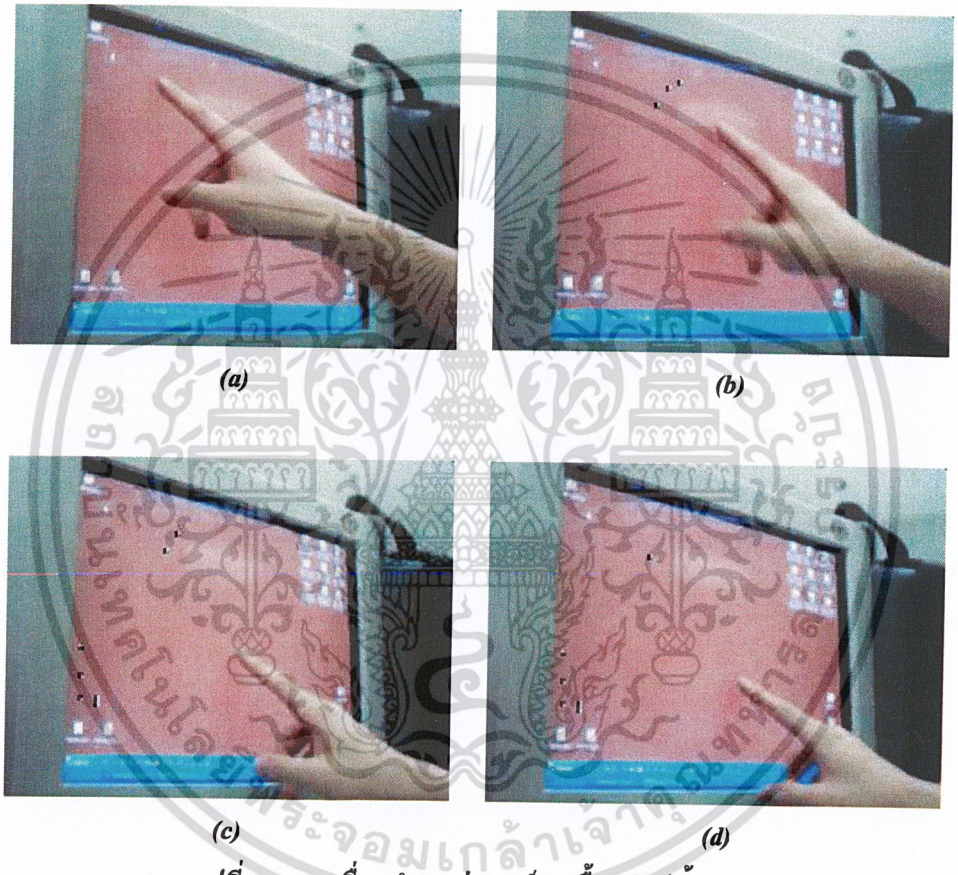
(a) *(b)*

(c) *(d)*

รูปที่ 4-5 การเลื่อนตำแหน่งเมาส์บนพื้นหลังสีดำ

จากรูปที่ 4-5 จะใช้พื้นหลังเป็นสีดำ ซึ่งรูปที่ได้ในแต่ละเฟรมนั้นจะสว่างมากขึ้น เนื่องจากสีดำที่ใช้เป็นสีที่มืดมากเวบแคมจึงได้ทำการปรับแสงเพื่อเพิ่มความสว่างของภาพขึ้น ระดับของสีนิ้วจึงมีความแตกต่างกับเมื่อเริ่มโปรแกรม ซึ่งเมื่อทำการปรับแสงแล้วสีนิ้วมือจะกลับมาใกล้เคียงกับเมื่อเริ่มต้นโปรแกรมได้บ้างในบางเฟรม จึงมีผลให้การเคลื่อนที่ของเมาส์นั้นสามารถทำได้แต่อาจจะไม่ต่อเนื่อง ซึ่งจะทำให้ตำแหน่งของเมาส์เกิดการกระโดดได้

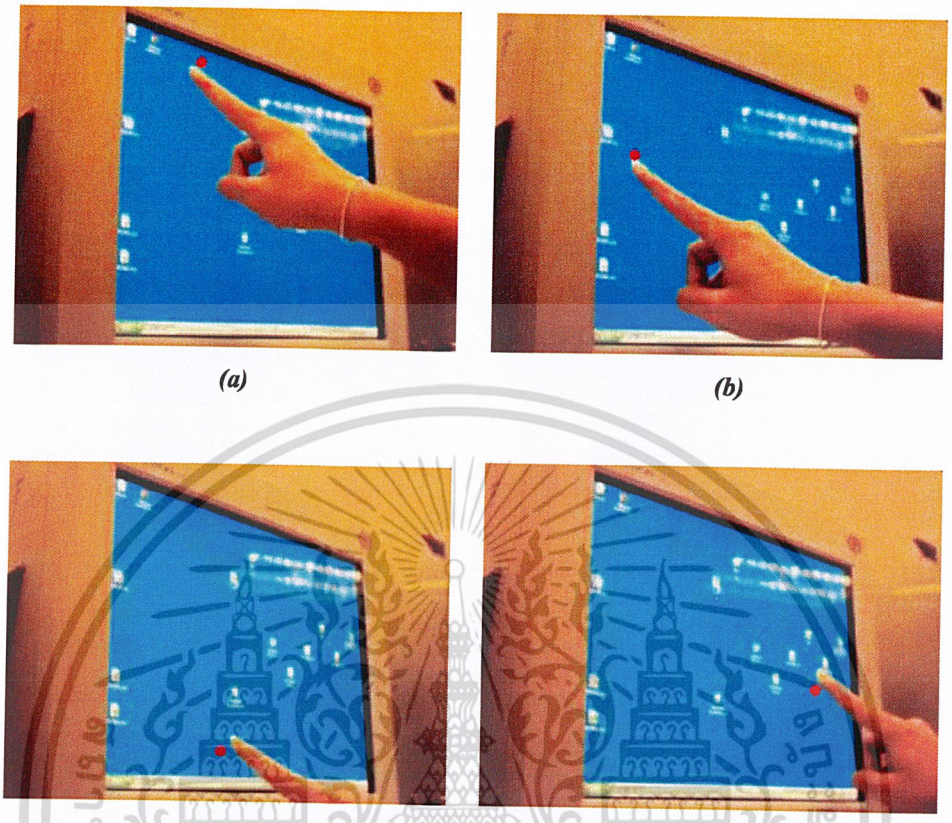
4.1.6 พื้นหลังสีน้ำตาล



รูปที่ 4-6 การเลื่อนตำแหน่งเมาส์บนพื้นหลังสีน้ำตาล

จากรูปที่ 4-6 จะใช้พื้นหลังเป็นสีน้ำตาล ซึ่งสีของพื้นหลังนั้นจะมีระดับสีใกล้เคียงกับสีของนิ้ว การหาปลายนิ้วโดยใช้การทำ Frame difference นั้นไม่สามารถทำได้ และยังทำให้การหาระดับสีของนิ้วนั้นบางครั้งไปได้ตำแหน่งที่เป็นพื้นหลังของหน้าจอ เนื่องจากมีระดับสีใกล้เคียงกับนิ้ว จึงมีผลให้เกือบทุกเฟรมไม่สามารถหาตำแหน่งของปลายนิ้วได้อย่างถูกต้อง ดังนั้นการเคลื่อนที่ของเมาส์จึงไม่สามารถทำได้

4.1.7 พื้นหลังสีน้ำเงิน



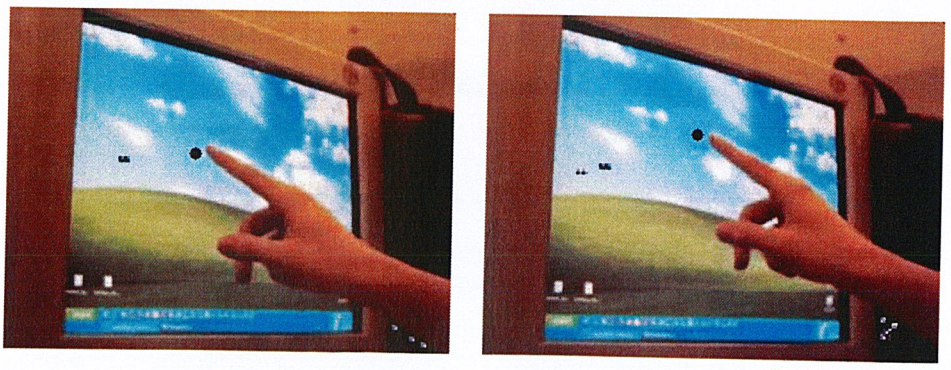
(a) *(b)*

(c) *(d)*

รูปที่ 4-7 การเลื่อนตำแหน่งเมาส์บนพื้นหลังสีน้ำเงิน

จากรูปที่ 4-7 จะใช้พื้นหลังเป็นสีน้ำเงิน ซึ่งเมื่อทำการปรับแสงแล้วสีนิ้วมือจะกลับมาใกล้เคียงกับเมื่อเริ่มต้นโปรแกรมได้ ทำให้สามารถหาตำแหน่งของปลายนิ้วได้ จึงมีผลให้การเคลื่อนที่ของเมาส์นั้นสามารถทำได้อย่างสม่ำเสมอ และใกล้เคียงกับที่ผู้ใช้ชี้อยู่ จึงทำให้การเคลื่อนที่ของเมาส์นั้นเป็นไปอย่างต่อเนื่อง

4.1.8 พื้นหลังมีลวดลาย และไม่มีส่วนที่มีสีใกล้เคียงกับสีของนิ้วแบบที่ 1



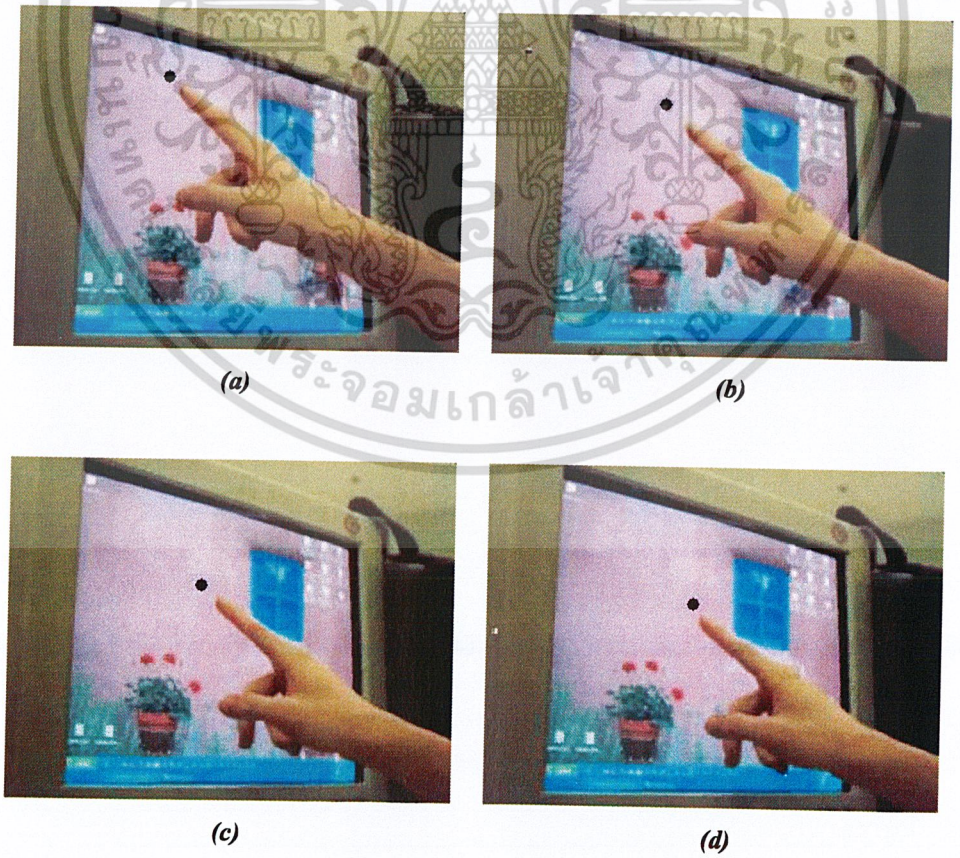
(a) *(b)*

(c) *(d)*

รูปที่ 4-8 การเลื่อนตำแหน่งเมาส์บนพื้นหลังมีลวดลาย และไม่มีส่วนที่มีสีใกล้เคียงกับสีของนิ้วแบบที่ 1

จากรูปที่ 4-8 จะใช้พื้นหลังที่มีลวดลาย ที่ไม่มีระดับสีใกล้เคียงกับสีของนิ้ว แต่เวบแคมยังมีการปรับแสงอย่างอัตโนมัติ ซึ่งโปรแกรมสามารถทำการปรับแสงกลับให้ใกล้เคียงกับระดับแสงเมื่อเริ่มโปรแกรมได้ ทำให้การหาตำแหน่งของปลายนิ้วได้ จึงมีผลให้การเคลื่อนที่ของเมาส์นั้นเป็นไปอย่างต่อเนื่อง

4.1.9 พื้นหลังมีลวดลาย และไม่มีส่วนที่มีสีใกล้เคียงกับสีของนิ้วแบบที่ 2



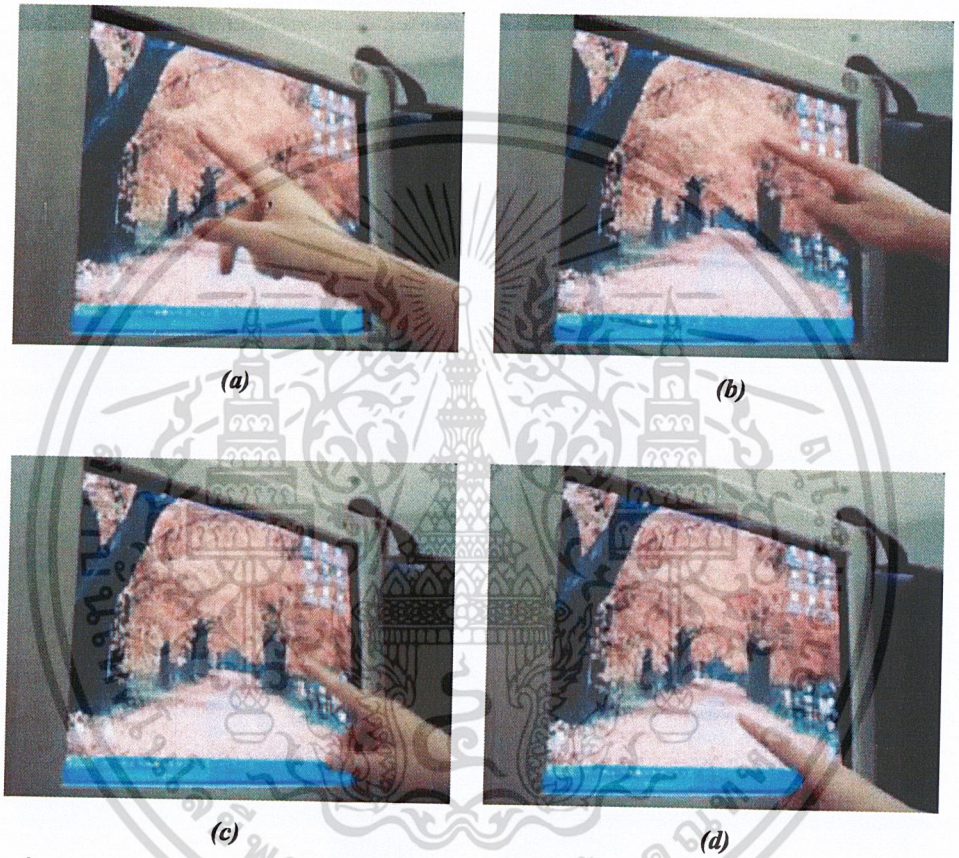
(a) *(b)*

(c) *(d)*

รูปที่ 4-9 การเลื่อนตำแหน่งเมาส์บนพื้นหลังมีลวดลาย และไม่มีส่วนที่มีสีใกล้เคียงกับสีของนิ้วแบบที่ 2

จากรูปที่ 4-9 จะใช้พื้นหลังที่มีลวดลาย ที่ไม่มีระดับสีใกล้เคียงกับสีของนิ้ว แต่เวบแคมยังมีการปรับแสงอย่างอัตโนมัติ ซึ่งโปรแกรมสามารถทำการปรับแสงกลับให้ใกล้เคียงกับระดับแสงเมื่อเริ่มโปรแกรมได้ ทำให้การหาตำแหน่งของปลายนิ้วจะได้ผลคล้ายกับหัวข้อที่ผ่านมา จึงมีผลให้การเคลื่อนที่ของเมาส์นั้นเป็นไปอย่างต่อเนื่อง

10. พื้นหลังมีลวดลาย และส่วนใหญ่มีสีใกล้เคียงกับสีของนิ้ว



รูปที่ 4-10 การเลื่อนตำแหน่งเมาส์บนพื้นหลังมีลวดลาย และส่วนใหญ่มีสีใกล้เคียงกับสีของนิ้ว

จากรูปที่ 4-10 จะใช้พื้นหลังที่มีลวดลาย โดยในรูปพื้นหลังส่วนใหญ่จะมีสีใกล้เคียงกับสีของนิ้ว ทำให้ผลที่ได้จากการทำ Frame difference ไม่สามารถทำได้ และยังทำให้การหาระดับสีของนิ้วนั้นบางครั้งไปได้ตำแหน่งที่เป็นพื้นหลังของหน้าจอ เนื่องจากมีระดับสีใกล้เคียงกับนิ้ว จึงเป็นผลให้รูปภาพส่วนใหญ่ไม่สามารถหาตำแหน่งของปลายนิ้วได้อย่างถูกต้อง ดังนั้นการเคลื่อนที่ของเมาส์จึงไม่สามารถทำได้

4.2 การทดลองการใช้งานโปรแกรมบนพื้นหลังต่างๆ กัน

ในการทดลองจะใช้พื้นหลังที่ใช้ในการทดลองที่ 1 มาทำการทดลอง ซึ่งจะทำการคลิ้กเมาส์ดับเบิลคลิ้ก และ Drag Drop อย่างละ 10 ครั้ง ในแต่ละพื้นหลัง

	พื้นหลังแบบที่ 1	พื้นหลังแบบที่ 2	พื้นหลังแบบที่ 3	พื้นหลังแบบที่ 4	พื้นหลังแบบที่ 5
คลิ้ก	10	9	0	9	6
ดับเบิลคลิ้ก	9	9	0	9	5
Drag Drop	9	7	0	8	5

	พื้นหลังแบบที่ 6	พื้นหลังแบบที่ 7	พื้นหลังแบบที่ 8	พื้นหลังแบบที่ 9	พื้นหลังแบบที่ 10
คลิ้ก	0	8	7	8	0
ดับเบิลคลิ้ก	0	8	7	8	0
Drag Drop	0	7	8	7	0

ตารางที่ 4-1 ผลการใช้งานโปรแกรมบนพื้นหลังต่างๆ

การทดลองนี้เป็นการทดลองที่สืบเนื่องมาจากการทดลองที่ 1 คือ ถ้าการหาตำแหน่งของปลายนิ้วผิดพลาดจะส่งผลให้การคลิ้ก ดับเบิลคลิ้ก และ Drag and Drop ไม่สามารถทำได้

ในพื้นหลังแบบที่ 1 ซึ่งก็คือสีขาว จะสามารถหาตำแหน่งของปลายนิ้วได้ดีที่สุด ทำให้สามารถทำการคลิ้ก ดับเบิลคลิ้ก หรือ Drag Drop ได้อย่างมีประสิทธิภาพ

ในพื้นหลังแบบที่ 2, 4, 5 และ 7 เมื่อทำการปรับแสงกลับมา ระดับสีของนิ้วจะกลับมาได้ใกล้เคียงกับเมื่อเริ่มต้นโปรแกรม แต่ยังไม่เท่าเดิม ทำให้ไม่สามารถหาตำแหน่งของปลายนิ้วได้ดีเท่าแบบที่ 1 ทำให้ทำการคลิ้ก ดับเบิลคลิ้ก หรือ Drag Drop ได้จำนวนครั้งน้อยกว่าแบบที่ 1 เล็กน้อย

ในพื้นหลังแบบที่ 3 และ 6 ซึ่งก็คือ สีแดงและสีน้ำตาล ไม่สามารถหาตำแหน่งของปลายนิ้วได้ ทำให้ไม่สามารถทำการคลิ้ก ดับเบิลคลิ้ก หรือ Drag Drop ได้

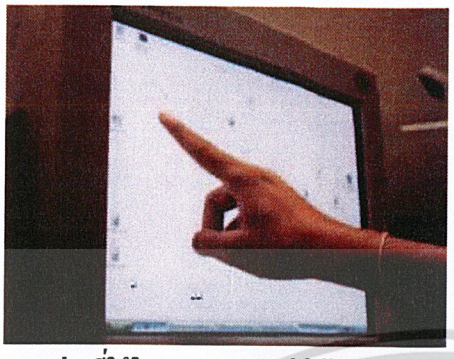
ในพื้นหลังแบบที่ 8 และ 9 เป็นพื้นหลังที่มีลวดลาย แต่ไม่มีสีที่เหมือนกับนิ้วมือ ทำให้สามารถหาตำแหน่งของปลายนิ้วได้ เป็นผลให้สามารถทำการคลิ้ก ดับเบิลคลิ้ก หรือ Drag Drop ได้จำนวนครั้งพอๆ กัน

ในพื้นหลังแบบที่ 10 ซึ่งเป็นพื้นหลังที่มีลวดลาย และมีสีเหมือนกับนิ้วมือเป็นส่วนใหญ่ จึงไม่สามารถหาตำแหน่งของปลายนิ้วได้ ทำให้ไม่สามารถทำการคลิ้ก ดับเบิลคลิ้ก หรือ Drag Drop ได้

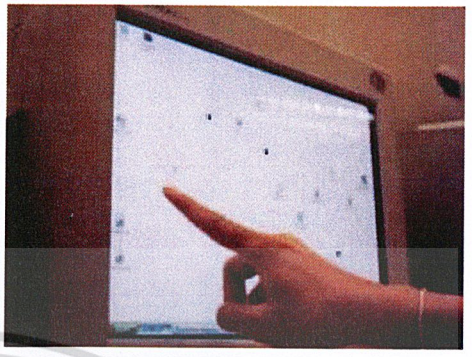
4.3 การทดลองการใช้งานโปรแกรมโดยมีความสว่างของสภาพแวดล้อมต่างกัน

การทดลองนี้จะเป็นการทดลองการใช้งานโปรแกรมโดยมีความสว่างของสภาพแวดล้อมต่างๆ กัน โดยในขั้นตอนของการกำหนดค่าให้กับโปรแกรมจะต้องทำในสภาวะที่มีแสงปานกลางที่จะสามารถ

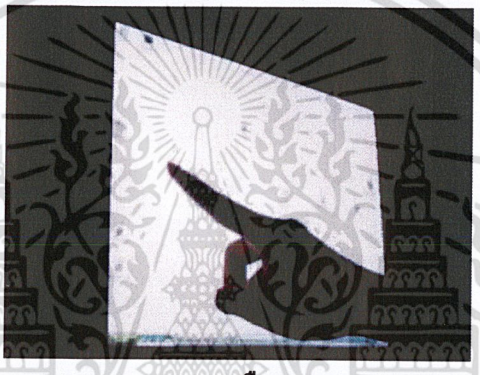
เก็บข้อมูลของระดับสีของนิ้วได้ (ความสว่างในรูปที่ 4-11(a) ใช้ในการกำหนดค่าให้กับโปรแกรม) โดยความสว่างในการทดลองนี้จะแบ่งออกเป็น 3 ระดับ ดังรูปที่ 4-11



(a) ความสว่างที่ใช้ในการกำหนดค่าให้กับโปรแกรม



(b) สว่าง



(c) มืด

รูปที่ 4-11 การใช้งานโปรแกรมในความสว่างที่ต่างกัน

	รูปที่ 4-11(a)	รูปที่ 4-11(b)	รูปที่ 4-11(c)
คลิ้ก	9	9	7
ดับเบิลคลิ้ก	8	9	8
Drag Drop	7	8	6

ตารางที่ 4-2 ผลการใช้งานโปรแกรมในสภาวะที่แสงไม่เท่ากัน

จากผลการทดลอง รูปที่ 4-11(a) เป็นสภาวะเดียวกับเมื่อเริ่มต้นโปรแกรม ดังนั้นจึงสามารถทำการคลิ้ก ดับเบิลคลิ้ก และ Drag Drop ได้พอๆ กับรูปที่ 4-11(b) ที่มีแสงสว่างกว่า แต่ในรูปที่4-11(c) ที่ไม่มีความสว่างทำให้ระดับสีของนิ้วนั้นน้อยลง ทำให้บางครั้งไม่สามารถหาตำแหน่งของปลายนิ้วได้ ทำให้ความสามารถในการคลิ้ก ดับเบิลคลิ้ก และ Drag Drop ลดลง

4.4 การทดลองการเลื่อนตำแหน่งเมาส์

การทดลองนี้จะเลื่อนตำแหน่งเมาส์โดยทำการเคลื่อนที่ปลายนิ้วด้วยความเร็วที่ต่างกัน เนื่องจากการหาตำแหน่งปลายนิ้วนั้นจะทำการตีกรอบขนาด 30*30 พิกเซล ดังนั้นถ้าปลายนิ้วมีการเคลื่อนที่เร็ว

มากเกินไปก็จะหลุดจากกรอบการหาตำแหน่งปลายนิ้ว ทำให้ไม่สามารถหาตำแหน่งปลายนิ้วภายในกรอบได้ ซึ่งการทดลองนี้ได้ทำการเคลื่อนที่ปลายนิ้วจากขอบหน้าจอด้านซ้าย ไปขอบหน้าจอด้านขวาของหน้าจอ 17 นิ้ว เป็นระยะทาง 0.33 เมตร แล้วจับความเร็วการเคลื่อนที่ของปลายนิ้ว ดังตารางที่ 4-3

เวลา (วินาที)	การตอบสนองของเมาส์
1.51	เมาส์เลื่อนตำแหน่งตามปลายนิ้ว
1.30	เมาส์เลื่อนตำแหน่งตามปลายนิ้ว
1.23	เมาส์ไม่เคลื่อนที่
0.99	เมาส์ไม่เคลื่อนที่

ตารางที่ 4-3 ผลการทดลองการเลื่อนตำแหน่งเมาส์

จากผลการทดลองจะเห็นได้ว่าถ้าเคลื่อนปลายนิ้วระยะทาง 0.33 เมตร ด้วยเวลาที่มากกว่าหรือเท่ากับ 1.30 วินาที (ความเร็ว 0.254 m/s) ตำแหน่งปลายนิ้วจะไม่หลุดจากกรอบ จึงสามารถหาตำแหน่งปลายนิ้วได้ เป็นผลให้เมาส์สามารถเลื่อนตำแหน่งตามปลายนิ้วได้ แต่ถ้าเคลื่อนปลายนิ้วด้วยเวลาที่น้อยกว่าหรือเท่ากับ 1.23 วินาที (ความเร็ว 0.268 m/s) ตำแหน่งปลายนิ้วจะหลุดจากกรอบ จึงไม่สามารถหาตำแหน่งปลายนิ้วเจอ เป็นผลให้เมาส์ไม่เคลื่อนที่

4.4 การทดลองการคลิ้กบนไอคอนขนาดต่างๆ

ขนาดของไอคอน	ผลการทดลอง
30*30	สามารถคลิ้กได้
25*25	สามารถคลิ้กได้
20*20	สามารถคลิ้กได้
15*15	ไม่สามารถคลิ้กได้
10*10	ไม่สามารถคลิ้กได้

ตารางที่ 4-4 ผลการทดลองการคลิ้กบนไปคอนขนาดต่างๆ

เนื่องจากขนาดของรูปภาพที่ได้จากเวบแคมมีขนาด 320*240 พิกเซล แต่ขนาดของหน้าจอที่ใช้มีขนาด 1024*768 พิกเซล ดังนั้นการแปลงตำแหน่งบนรูปภาพไปเป็นตำแหน่งบนหน้าจอ จะไม่สามารถอ้างตำแหน่งบนหน้าจอได้ทุกพิกเซล เป็นผลให้ไอคอนที่มีขนาดเล็กจะไม่สามารถทำการคลิ้กได้ดังตารางที่ 4-4 ซึ่งไอคอนที่มีขนาดใหญ่กว่า 20*20 จะทำการคลิ้กได้ แต่ไอคอนที่มีขนาดเล็กกว่า 15*15 จะไม่สามารถคลิ้กได้

บทที่ 5
บทวิจารณ์และสรุป

5.1 วิจารณ์และสรุปอุปกรณ์ฮาร์ดแวร์

อุปกรณ์ฮาร์ดแวร์ที่ใช้ในโครงงานนี้ คือ กล้องเวบแคม 1 ตัว ซึ่งการใช้เวบแคมในการช่วยทำให้หน้าจอธรรมดาเป็นเสมือนหน้าจอสัมผัสนั้นมีข้อดีกว่าการใช้แผ่นจอสัมผัส ก็คือ ในขณะนี้เวบแคมเป็นอุปกรณ์ที่กำลังได้รับความนิยมสำหรับการสนทนากันผ่านอินเทอร์เนต ทำให้หาง่าย มีราคาที่ถูกกว่าแผ่นจอสัมผัส และยังสามารถใช้ประโยชน์อย่างอื่นได้อีกด้วย

แต่ว่าการใช้เวบแคมก็ยังมีปัญหาในเรื่องข้อจำกัดของตัวอุปกรณ์เองนั้นก็คือ

1. เวบแคมบางรุ่นมีเฟรมเรตที่ต่ำจนเกินไป ทำให้ไม่สามารถจับภาพที่มีการเลื่อนปลายนิ้วเร็วๆ ได้ทัน ในขณะเดียวกันถ้าทำการคลิ้ก ดับเบิลคลิ้ก หรือ Drag Drop เร็วๆ บางครั้งก็จะไม่สามารถทำงานได้ตามที่ต้องการ

2. เฟรมเรตของเวบแคมต้องมากกว่าเฟรมเรตของจอคอมพิวเตอร์ เพื่อที่จะให้ภาพจากเวบแคมที่ถ่ายหน้าจอไม่เกิดการกระพริบ ถ้าเกิดการกระพริบแล้วตำแหน่งปลายนิ้วที่หาได้อาจคลาดเคลื่อนจากตำแหน่งที่ผู้ใช้ชี้อยู่จริง

3. ขนาดภาพของเวบแคมบางรุ่นนั้นจะมีขนาดไม่ใหญ่มาก ซึ่งเมื่อนำมาแปลงให้มีขนาดเท่ากับจอภาพซึ่งมีขนาดใหญ่กว่ามาก จะทำให้ตำแหน่งปลายนิ้วที่ผู้ใช้มองเห็นกับเวบแคมมองเห็นจะคลาดเคลื่อนไปได้

การใช้เวบแคมเพียง 1 ตัว นั้นจะมองเห็นภาพแค่ 2 มิติ ทำให้ไม่สามารถแยกแยะความแตกต่างระหว่างการเคลื่อนที่ของนิ้ว เพื่อเป็นการเคลื่อนย้ายตำแหน่งเคอร์เซอร์ของเมาส์ กับการขยับนิ้วขึ้นลงเพื่อเป็นการกดเมาส์ เพราะฉะนั้นต้องกำหนดวิธีการที่แตกต่างกันของการเคลื่อนที่ของเมาส์กับการกดเมาส์ โดยใน โครงงานนี้กำหนดว่าการเคลื่อนนิ้วเป็นการเคลื่อนย้ายตำแหน่งเคอร์เซอร์ของเมาส์ และถ้าต้องการกดเมาส์ให้ทำการหยุดอยู่กับที่ 1 วินาที ก่อนที่จะขยับนิ้วขึ้นลง ซึ่งจะสามารถแยกความแตกต่างของการเคลื่อนเมาส์ และการกดเมาส์ได้ และการวางตำแหน่งของเวบแคมจะต้องให้มุมระหว่างผู้ใช้กับหน้าจอ และเวบแคมกับหน้าจอน้อยที่สุดเท่าที่จะเป็นไปได้ มิฉะนั้นแล้วจะทำให้ตำแหน่งปลายนิ้วที่ผู้ใช้มองเห็นกับเวบแคมมองเห็นจะคลาดเคลื่อนไปได้มาก

5.1 วิจารณ์และสรุปในส่วนของซอร์ฟแวร์

1. เนื่องจากโครงงานนี้จะต้องทำงานแบบ real time แต่ในส่วนของโปรแกรมมีการคำนวณค่าต่างๆ มาก ทำให้เกิดการดีเลย์ เช่น ถ้าผู้ใช้ต้องการจะคลิ้ก ผู้ใช้จะไม่สามารถเลื่อนนิ้วได้อย่างรวดเร็วได้ เพราะโปรแกรมจะไม่สามารถไปดึงรูปภาพมาได้ทัน ดังนั้นอาจจะต้องทำการการ optimize ในส่วนของซอร์ฟแวร์ เพื่อลดเวลาในการประมวลผล

2. อัลกอริทึมที่ใช้ในการหาปลายนิ้วจะใช้การทำ Frame difference และใช้การหาสีของนิ้ว ดังนั้นถ้าพื้นหลังมีระดับสีใกล้เคียงกับนิ้วมือของผู้ใช้ จะทำให้การทำ Frame difference ไม่สามารถหาความแตกต่างของส่วนที่เปลี่ยนแปลงได้ เนื่องจากจะมีความแตกต่างกันน้อยมาก จนถือว่าเป็นข้อมูลภาพในส่วนที่ไม่ต้องการ และการหาสีของนิ้วจะได้ตำแหน่งที่ผิดพลาด ดังนั้นอาจจะต้องหาอัลกอริทึมอื่นมาช่วยเพื่อให้หาตำแหน่งปลายนิ้วได้แน่นอนขึ้น

3. เนื่องจากการวางตำแหน่งของเวบแคมอยู่เยื้องจากหน้าจอ ทำให้ต้องมีการแปลงตำแหน่งจากหน้าจอในรูปไปเป็นหน้าจอจริงๆ โดยในโปรแกรมนี้ใช้สมการดีกรี 3 ซึ่งยังไม่เพียงพอที่จะทำให้ตำแหน่งของเมาส์ตรงกับที่ผู้ใช้ชี้อยู่ ดังนั้นควรจะใช้สมการดีกรีที่สูงกว่านี้ เพื่อที่จะให้ตำแหน่งของเมาส์กับตำแหน่งที่ผู้ใช้ชี้ใกล้เคียงกันมากขึ้น

บรรณานุกรม

[1] Rafael C.Gonzalez, Richard E. Woods, *Digital Image Processing*, Prentice Hall, 2002

[2] Zhengyou Zhang, *Vision-based Interaction with Fingers and Papers*, In proc. International Symposium on the CREST, 2003

[3] Daniel Heckkenberg, *A Video Based Hand Gesture Interface*, Department of Computer Science and Electical Engineering, University of Queensland, 1999

[4] Zhengyou Zhang, Ying Wu, Ying Shan, Steven Shafer, *Visual Panel*

[5] Claudio S. Pinhanez, Frederik C. Kjeldsen, Anthony Levas, Gopal S. Pingali, Mark E. Podlaseck, Paul B. Chou, *Ubiquitous Interactive Graphics*, IBM Research Division, 2002