การตรวจจับการเคลื่อนไหวของมนุษย์โดยใช้ภาษา C

HUMAN MOTION CAPTURE WITH C

โดย

นาย นฤดล สุรจารุสาร

นาย พีรพัฒน์ ด่านถาวรเจริญ

นาย มนัชกร คำริพัตนโชติ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2553

การตรวจจับการเคลื่อนไหวของมนุษย์โดยใช้ภาษา C

HUMAN MOTION CAPTURE WITH C

โดย

นายนฤดล สุรจารุสาร รหัสนักศึกษา 50010773

นายพีรพัฒน์ ด่านถาวรเจริญ รหัสนักศึกษา 50011117

นายมนัชกร ดำริพัฒนโชติ รหัสนักศึกษา 50011212

อาจารย์ที่ปรึกษา

ผศ.ดร. กิติพล ชิตสกุล

ดร. สุรเดช ตรีไตรลักษณะ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2553

รายงานวิชาการประยุกต์วงจรอิเล็กทรอนิกส์2 ปีการศึกษา 2553

ภาควิชา อิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การตรวจจับการเคลื่อนไหวของมนุษย์โดยใช้ C

HUMAN MOTION CAPTURE WITH C

ผู้จัดทำ 1. นาย นฤดล สุรจารุสาร

2. นาย พีรพัฒน์ ด่านถาวรเจริญ

3. นาย มนัชกร ดำริพัตนโชติ

..... อาจารย์ที่ปรึกษา

(ดร. สุรเดช ตรีไตรลักษณะ)

..... อาจารย์ที่ปรึกษา

(ผศ.ดร. กิติพล ชิตสกุล)

การตรวจจับภาพการเคลื่อนไหวของมนุษย์

นาย นฤดล สุรจารุสาร 50010773

นาย พีรพัฒน์ ด่านถาวรเจริญ 50011117

นาย มนัชกร ดำริพัตนโชติ 50011212

ดร. สุรเดช ตรีไตรลักษณะ อาจารย์ที่ปรึกษา

ผศ.ดร. กิติพล ชิตสกุล อาจารย์ที่ปรึกษา

ปีการศึกษา 2553

บทคัดย่อ

รายงานฉบับนี้ได้ทำศึกษาเกี่ยวกับการประมวลผลภาพการเคลื่อนไหวของมนุษย์จากกล้อง Basler-A640 ซึ่งมีเป้าหมายในการหาพิกัดในระนาบ 2 มิติ และความเร็วในการรับภาพของกล้อง 100 fps. โดยใช้ภาษา C++ ในส่วนของการรับภาพเพื่อนำภาพที่ได้ไปประมวลผลหาตำแหน่งมาร์กเกอร์เป็นพิกัด (x,y) โดยคำนวณจากกลุ่มจุดของแอคทีฟมาร์คเกอร์ที่ติดบริเวณเอว หัวเข่า และข้อเท้า จากกล้อง 3 กล้อง และจุดพิกัดที่ได้จะนำไปใช้ในการสร้างภาพเสมือนการเคลื่อนไหวใน 2 มิติ รวมถึงสามารถนำไปใช้ในการพัฒนาการจำลองภาพการเคลื่อนไหวในระบบแกน 3 มิติต่อไปได้

HUMAN MOTION CAPTURE

Mr. Naruedol Surajarusarn 50010773
Mr. Peerapat Dantavornjaroen 50011117
Mr. Manatchagorn Dumripattanachod 50011212
Prof. Dr. Surade Tretriraksana Advisor
Asst. Prof. Dr. Kittipol Chitsakul Advisor
Education Year 2010

Abstract

This report has done the studies on the processing of movement of people from the camera Basler-A640, which aim to find the coordinates in the plane 2D and the speed of the video camera 100fps by using C++ language in the section of the screen. The image is processed to locate the marker coordinates as (x,y) calculated from the point of the active maker on the waist, knees and ankles from the three cameras. The coordinated points can be used in a visualization of the movement in 2D including the development of the simulation of the core movement system in 3D hereafter.

กิตติกรรมประกาศ

การจัดทำรายงานฉบับนี้ที่ว่าด้วยการศึกษาการตรวจจับการเคลื่อนไหวของมนุษย์ สำเร็จได้ด้วยคำแนะนำและความเอื้อเฟื้อจาก ดร.สุรเดช ตรีไตรลักษณะ และ ผศ.ดร. กิตติพล ชิตสกุล ซึ่งเป็นอาจารย์ที่ปรึกษา รวมถึงข้อมูลจากเอกสารอ้างอิงต่างๆ ที่ได้ให้ความรู้เพิ่มเติม จนการทำโครงการวิจัยดังกล่าวได้ลุล่วงสำเร็จด้วยดี

สุดท้ายนี้ขอกราบขอบพระคุณ คุณพ่อ คุณแม่ ผู้ซึ่งให้กำลังใจกับทางคณะผู้จัดทำเสมอมา ทำให้คณะผู้จัดทำมีความมานะ พากเพียรพยายามทำรายงานฉบับนี้ให้สำเร็จลุล่วงไปได้ด้วยดี

คณะผู้จัดทำขอขอบพระคุณเป็นอย่างสูงมา ณ ที่นี้

คณะผู้จัดทำ

นายนฤดล สุรจารุสาร

นายพีรพัฒน์ ด่านถาวรเจริญ

นายมนัชกร ดำริพัฒนโชติ

สารบัญ

สารบัญ(ต่อ)

สารบัญรูปภาพ

สารบัญรูปภาพ(ต่อ)

หน้า

สารบัญรูปภาพ(ต่อ)

หน้า

สารบัญตาราง

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ปัจจุบันได้มีการนำการตรวจจับการเคลื่อนไหวของมนุษย์ไปประยุกต์ใช้เพื่อการสร้างภาพ อย่าง เช่น การทำการ์ตูนเอนนิเมชั่น การสร้างภาพทางการแพทย์ เป็นต้น ในการประยุกต์ใช้การตรวจจับการเคลื่อนไหวเพื่อการสร้างภาพนั้น สิ่งที่จำเป็นและสำคัญยิ่งอย่างหนึ่งคือ ตัวอุปกรณ์ที่ใช้ในการตรวจจับการเคลื่อนไหว ซึ่งมีอยู่ด้วยกันหลายชนิดโดยแต่ละชนิดนั้นจะมีความเหมาะสมต่างๆกันไปตามการใช้งาน ซึ่งชนิดของตัวตรวจจับการเคลื่อนไหวที่นำมาใช้งานคือ GigE Camera, CAT6 Lan, Computer

ในการสร้างภาพทั่วๆไปจากการตรวจจับการเคลื่อนไหวของมนุษย์ที่สามารถพบในชีวิตประจำวัน เช่น ภาพทางการ์ตูนเอนนิเมชั่น หรือการสร้างภาพเพื่อวิเคราะห์ร่างกายนักกีฬา นั้น โดยการรับภาพจากกล้องไปประมวลผลด้วยคอมพิวเตอร์เพื่อสร้างเป็นภาพ 2 มิติ และ 3 มิติ

จากความสำคัญดังกล่าว งานวิจัยนี้จึงมีแนวความคิดที่จะศึกษาการตรวจจับการเคลื่อนไหวเพื่อนำมาประยุกต์ใช้ทางด้านชีวะการแพทย์ เพื่อนำข้อมูลที่ได้ไปใช้ในการบำบัดรักษา

1.2 ความมุ่งหมายและวัตถุประสงค์ของการศึกษา

รายงานฉบับนี้มุ่งหวังการศึกษา และวิจัย การตรวจจับการเคลื่อนไหวของมนุษย์ และนำภาพข้อมูลที่ได้มาจำลองในระบบ 3 มิติ โดยที่มีความเร็วในการรับภาพ 100 fps. และกล้องที่นำมาใช้ในการรับภาพแต่ละครั้งตำแหน่งไม่แน่นอน

ข้อมูลที่ได้จากการทดลองสามารถนำไปใช้ในด้านการแพทย์ ในวิเคราะห์ความผิดปกติเรื่องการเคลื่อนไหวของผู้ป่วย และยังใช้ในวงการบันเทิงได้อีกด้วย

1.3 สมมติฐานของการศึกษา

โดยทั่วไป ในการตรวจจับการเคลื่อนไหวจำเป็นต้องใช้กล้องที่มีเฟรมเรทสูง เพื่อสามารถจับการเคลื่อนไหวมนุษย์ในลักษณะต่างๆได้ครอบคลุม และยังจำเป็นต้องใช้คอมพิวเตอร์ที่มีประสิทธิภาพสูง รวมถึงการเขียนโปรแกรมที่ใช้ในการรับและประมวลผลภาพ ก็มีความจำเป็นที่จะต้องมีความรวดเร็วในการประมวลผล เพราะข้อมูลที่ใช้ในการประมวลผลมีขนาดใหญ่ และมีขอบเขตเวลาที่จำกัด จึงเลือกใช้ภาษา C/C++ ในการออกแบบอัลกอริทึม

1.4 ขอบเขตงานวิจัย

ทำการรับภาพจากกล้อง GigE ที่ความเร็ว 100 fps. จำนวน 3 กล้อง โดยที่การวางกล้องแต่ละครั้งไม่แน่นอน และนำข้อมูลที่ได้มาคำนวณเพื่อหาพิกัดการเคลื่อนที่ในระนาบสองมิติ

เนื้อหาในรายงานจะแบ่งเนื้อหาออกเป็นบทต่าง ตามรายละเอียดดังนี้

บทที่ 1 บทนำ เป็นการกล่าวถึงประเด็นของที่มาของปัญหา ความมุ่งหมายและวัตถุประสงค์ แนวคิดที่ใช้ ขอบเขตงานวิจัย และขั้นตอนการศึกษารายงานฉบับนี้เพื่อทำความเข้าใจภาพรวมของรายงาน

บทที่ 2 กล่าวถึงทฤษฎีพื้นฐานของการตรวจจับการเคลื่อนไหวของมนุษย์

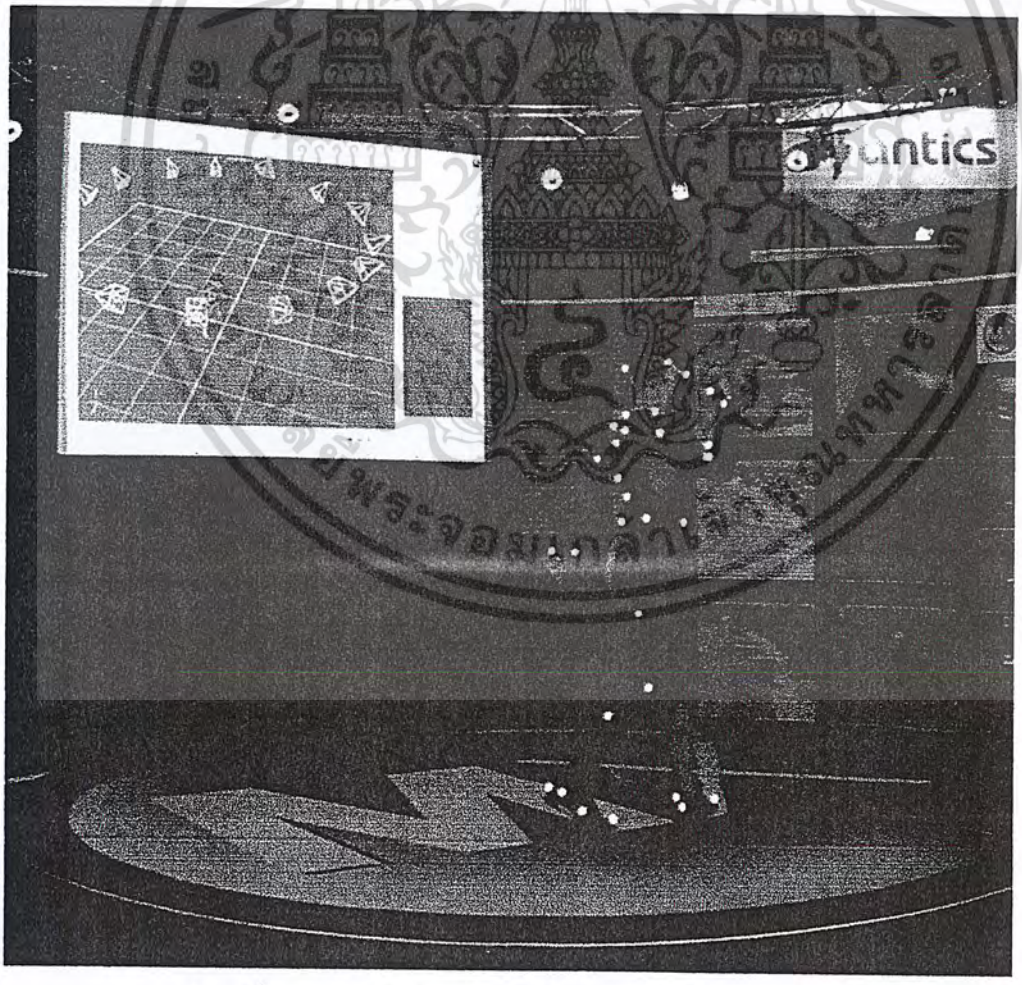
บทที่ 3 การทดลอง กล่าวถึงเทคโนโลยีที่ใช้ในการตรวจจับการเคลื่อนไหวของมนุษย์ เทคโนโลยีที่ใช้ในการประมวลผลภาพ และผลการทดลองที่ได้

บทที่ 4 สรุปผลการวิจัย เป็นการสรุปผลการทดลองที่ได้จากบทที่ 3

บทที่ 2
ทฤษฎีที่เกี่ยวข้อง

2.1 การบันทึกความเคลื่อนไหว

การบันทึกความเคลื่อนไหว หมายถึง ข้อมูลบันทึกการเคลื่อนไหวและทำการแปลงการเคลื่อนไหวนั้นให้อยู่ในรูปแบบของข้อมูลดิจิตอล การบันทึกความเคลื่อนไหว มีการนำมาใช้งานครั้งแรกที่ประเทศสกอตแลนด์ ในเรื่องของการทหาร ความบันเทิง กีฬา และประยุกต์ใช้กับการแพทย์ ซึ่งในปัจจุบันนี้ได้รับความสนใจเพิ่มมากขึ้นจากอดีต ในแง่ทั้งด้านความสะดวกในการตรวจวินิจฉัยโรคของแพทย์ ด้านความสะดวกสบายของแพทย์และคนไข้ ซึ่งการเก็บข้อมูลการเคลื่อนไหวในรูปแบบของข้อมูลดิจิตัลนี้ สามารถนำมาวิเคราะห์ได้ในภายหลัง



รูปภาพที่ 2.1 ภาพตัวอย่างการบันทึกการเคลื่อนไหวของมนุษย์

2.1.1 ประเภทของการบันทึกความเคลื่อนไหว

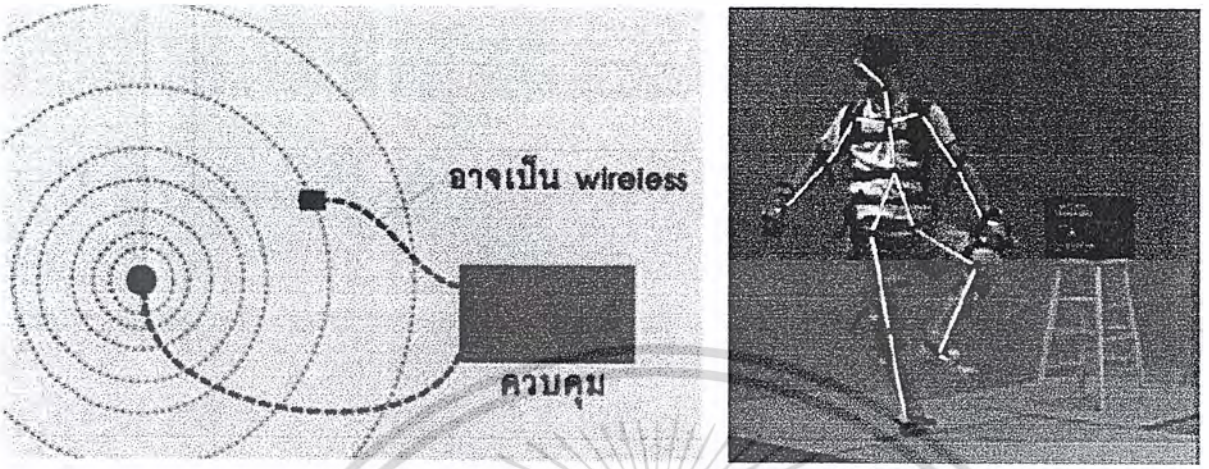
การบันทึกความเคลื่อนไหวแบ่งได้เป็น 3 ประเภทหลักๆ ตามเทคโนโลยีที่ใช้ ดังนี้

2.1.1.1 การบันทึกความเคลื่อนไหวโดยชุดเชิงกล (Electro-mechanical Motion Capture System) การบันทึกประเภทนี้มักใช้กับการจับการเคลื่อนไหวของมนุษย์ โดยเฉพาะ โดยใช้ชุดที่ทำขึ้นพิเศษสำหรับให้มนุษย์สวมใส่สำหรับการบันทึกความเคลื่อนไหว โดยชุดจะมีลักษณะเป็นโครงสร้างที่เชื่อมต่อกันโดยมีตัวต้านทานปรับค่าได้ในการวัดการหมุนของจุดต่างๆ ตามข้อต่อสำคัญของร่างกายการรู้มุมการหมุนของจุดต่างๆ ทำให้เราสามารถรู้ ท่าทางการเคลื่อนไหวของผู้สวมชุดได้

รูปภาพที่ 2.2 ภาพตัวอย่างของชุดเชิงกล

2.1.1.2 การบันทึกความเคลื่อนไหวโดยใช้แม่เหล็ก (Magnetic Motion Capture System)

ใช้เซ็นเซอร์ติดไว้ตามส่วนต่างๆ เพื่อวัดคลื่นแม่เหล็กไฟฟ้าความถี่ต่ำที่ ผลิตออกมาจากแหล่งกำเนิดคลื่นที่ติดตั้งไว้ในบริเวณที่ทำการบันทึก เซ็นเซอร์แต่ละตัวจะถูกต่อ สายมาเข้ายังวงจรวัดค่าเพื่อหาตำแหน่งของเซ็นเซอร์แต่ละตัวในสนามแม่เหล็ก วงจรวัดค่าจะส่งข้อมูลไปเข้าเครื่องคอมพิวเตอร์เพื่อแสดงตำแหน่งและการหมุนในรูปแบบสามมิติ



รูปภาพที่ 2.3 ภาพตัวอย่างของการบันทึกความเคลื่อนไหวโดยใช้แม่เหล็ก

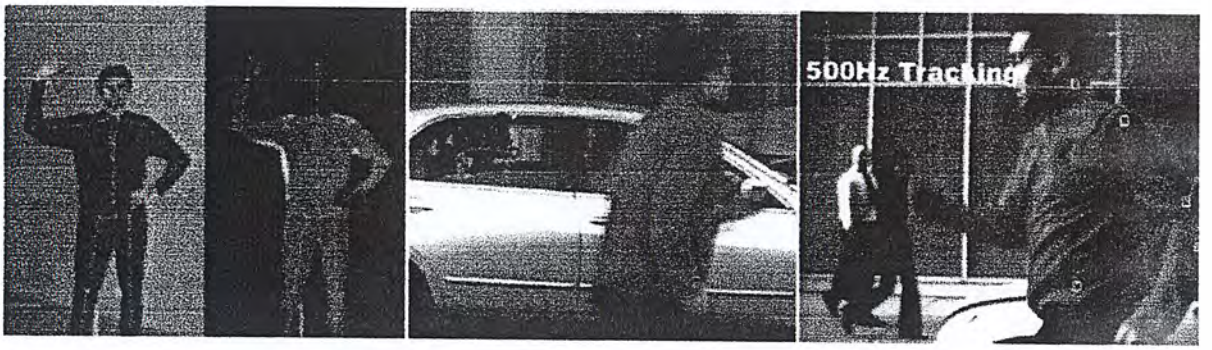
2.1.1.3 การบันทึกความเคลื่อนไหวโดยใช้กล้องวีดีโอ (Optical Motion Capture System)

รูปภาพที่ 2.4 ภาพตัวอย่างการบันทึกความเคลื่อนไหวด้วยกล้องวีดีโอ

วิธีการบันทึกแบบใช้วีดีโอยังสามารถแบ่งได้อีก 3 ชนิดคือ

2.1.1.3.1 แบบไม่มีมาร์กเกอร์ (Markerless)

เป็นการผสมผสานเทคโนโลยีและการวิจัยในเรื่องมุมมองของ คอมพิวเตอร์ ซึ่งนำไปสู่การพัฒนาของ การบันทึก ความเคลื่อนไหว แบบไม่มีมาร์กเกอร์ ที่เป็นกระบวนการซึ่งถูกพัฒนาขึ้นที่ Stanford, MIT, Mac Planck โดยไม่ต้องสวมใส่อุปกรณ์พิเศษสำหรับการ ติดตามการเคลื่อน ไหวโดยอาศัยหลักของอัลกอริทึมที่ออกแบบมาเพื่อให้ระบบ สามารถวิเคราะห์ได้ว่าข้อมูลที่ส่งเข้ามานั้นเป็นรูปร่างของมนุษย์ และสามารถแยกแยะได้ว่าส่วนที่ ทำการติดตามมีการเคลื่อนไหวอย่างไร โดยจุดที่ถูกติดตามนั้นจะยังคงอยู่ในจุดที่กำหนด



รูปภาพที่ 2.5 ภาพตัวอย่างของการบันทึกความเคลื่อนไหวแบบไม่มีมาร์กเกอร์

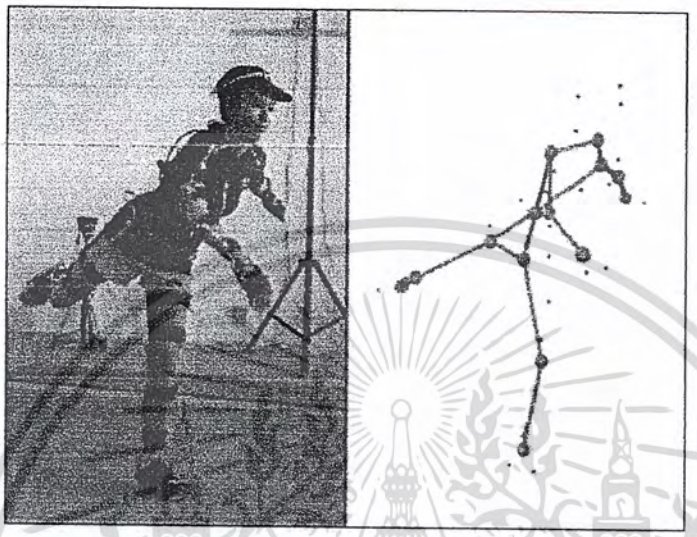
2.1.1.3.2 มาร์กเกอร์แบบสะท้อนแสงอินฟราเรด (Reflective Marker or Passive Marker)

เป็นการใช้ถ่ายภาพวิดีโอชนิดพิเศษในการติดตามตำแหน่ง ของมาร์กเกอร์ที่ติดอยู่ตามส่วนต่างๆ โดยกล้องวิดีโอนี้จะใช้เลนส์แบบรับแสงอินฟราเรด แล้วใช้ แสงอินฟราเรดจากแหล่งกำเนิดแสงที่ติดไว้รอบๆ กล้องถ่ายภาพ แสงจะสะท้อนที่มาร์กเกอร์ทำให้เห็นเป็นจุดที่มีความเข้มแสงมากกว่าบริเวณอื่นๆ

รูปภาพที่ 2.6 ภาพตัวอย่างของการบันทึกความเคลื่อนไหวแบบมาร์กเกอร์แบบสะท้อนแสงอินฟราเรด

2.1.1.3.3 มาร์กเกอร์แบบหลอดแอลอีดี (Active Marker or Pulsed-LED)

เป็นการบันทึกภาพท่าทางของผู้แสดงโดยใช้วิธีวัดจากมาร์เกอร์ ที่เป็นหลอดแอลอีดีโดยการวัดความเข้มของแสงสำหรับการบันทึกข้อมูล



รูปภาพที่ 2.7 ภาพตัวอย่างของการบันทึกความเคลื่อนไหวแบบมาร์กเกอร์แบบหลอดแอลอีดี

ตาราง 2.1 ข้อดีและข้อเสียของการบันทึกความเคลื่อนไหวโดยชุดเชิงกล

ข้อดี	- พื้นที่บันทึกมีขนาดกว้างและสามารถเคลื่อนย้ายได้สะดวก - สามารถบันทึกแบบทันที (Real-time) ได้ - มีราคาถูกกว่าอุปกรณ์การบันทึกแบบอื่นๆ - ไม่มีการบดบังของตัวเซ็นเซอร์
ข้อเสีย	- มีอัตราการบันทึกที่ต่ากว่าระบบอื่นๆ - มีข้อจากัดในการเคลื่อนไหว เนื่องมาจากชุดที่สวมใส่ขณะทำการบันทึก - ระบบมักจำกัดอยู่กับการบันทึกความเคลื่อนไหวของมนุษย์ - จำนวนและตำแหน่งของตัวเซ็นเซอร์ไม่สามารถเปลี่ยนแปลงได้ - ไม่สามารถคำนวณหาตำแหน่งจริงได้

ตาราง 2.2 ข้อดีและข้อเสียของการบันทึกความเคลื่อนไหวโดยใช้กล้องวีดีโอ

ข้อดี	- สามารถใช้มาร์กเกอร์ได้หลายจุด และสามารถเปลี่ยนอุปกรณ์ได้ง่าย ในกรณีที่ใช้มาร์กเกอร์ - สามารถกำหนดพื้นที่บันทึกได้ขนาดใหญ่ - ไม่มีอุปกรณ์หรือสายไฟที่เป็นอุปสรรคต่อการเคลื่อนที่ของผู้ทดสอบ - มีความถี่ที่สูง ทำให้ได้ข้อมูลที่มีความละเอียดสูงกว่าแบบอื่น
ข้อเสีย	- ต้องใช้การประมวลผลที่ซับซ้อนกว่าแบบอื่น - อุปกรณ์ที่ใช้ในงานจริง เป็นอุปกรณ์เฉพาะที่มีราคาแพง - เมื่อมาร์กเกอร์หรือจุดที่ติดตามถูกบดบังหรือซ้อนทับกัน จะไม่สามารถบันทึกได้ - การบันทึกต้องกระทำในสภาพแวดล้อมที่ควบคุมได้

ตาราง 2.3 ข้อดีและข้อเสียของการบันทึกความเคลื่อนไหวโดยใช้แม่เหล็ก

ข้อดี	- ให้ข้อมูลแบบ Real-time ทำให้สามารถให้ข้อมูลแบบผลป้อนกลับได้ (Feedback) - ให้ข้อมูลตำแหน่งและการหมุนได้โดยไม่ต้องไปประมวลผลต่อ - มักมีราคาถูกกว่าอุปกรณ์แบบใช้กล้อง - ไม่มีการบดบังเซ็นเซอร์
ข้อเสีย	- เซ็นเซอร์มีการตอบสนองต่อโลหะสูง ทำให้ข้อมูลมีความผิดปกติได้มาก - ผู้ทดสอบไม่สามารถเคลื่อนไหวได้สะดวก เนื่องจากชุดที่สวมใส่ - เซ็นเซอร์แม่เหล็กมีอัตราการบันทึกที่ต่ากว่าแบบใช้กล้อง - พื้นที่การบันทึกมักมีขนาดเล็ก และยากต่อการเปลี่ยนจำนวนและตำแหน่งของเซ็นเซอร์

ตาราง 2.4 เปรียบเทียบความแตกต่างของการเคลื่อนไหวแบบต่างๆ

	Optical Motion Capture	Electro-mechanical Motion Capture System	Magnetic Motion Capture
ความเร็วในการประมวลผล	ใช้เวลาในการคำนวณการเคลื่อนไหวและการหมุนยาวนานสุด	สามารถส่งตำแหน่งและการหมุนได้โดยไม่ต้องประมวลผลต่อ	สามารถส่งตำแหน่งและการหมุนได้โดยไม่ต้องประมวลผลต่อ
จำนวนและตำแหน่งของมาร์กเกอร์	ง่ายต่อการเปลี่ยนแปลงจำนวนและตำแหน่ง	ไม่สามารถเปลี่ยนแปลงตำแหน่งได้	ยากต่อการเปลี่ยนแปลงจำนวนและตำแหน่ง
การบดบังมาร์กเกอร์	มีการบดบังตำแหน่ง	ไม่มีการบดบังตำแหน่ง	ไม่มีการบดบังตาแหน่ง
พื้นที่ในการบันทึก	สามารถทำให้ใหญ่ได้	ขนาดใหญ่	ขนาดเล็ก

ตาราง 2.5 เปรียบเทียบความแตกต่างของอุปกรณ์บันทึกความเคลื่อนไหว

กล้อง	Webcam	GigE
Frame Rate	30 fps	100 fps
ความแม่นยำในการบันทึก	ปานกลาง	สูง
การบดบังมาร์กเกอร์	มี	มี
มาร์กเกอร์	ใช้การติดตามการเคลื่อนไหวด้วยซอฟท์แวร์แทนการใช้ฮาร์ดแวร์	LED
สภาพแวดล้อมและพื้นหลัง	ในห้องที่ไม่มีแสงรบกวนมีพื้นหลังสีโทนเดียวกัน	มีการจำกัดสภาพแวดล้อมขึ้นอยู่กับความเหมาะสม
ค่าใช้จ่าย	น้อย	ค่าใช้จ่ายตามคุณภาพของอุปกรณ์

2.1.1.4 GigE Vision

Gigabit Ethernet (IEEE802.3z) เป็นมาตรฐานใหม่ของเทคโนโลยีเครือข่าย LAN พัฒนามาจาก Ethernet ที่มีความเร็ว 10 Mbps ให้สามารถรับส่งข้อมูลได้ที่ระดับความเร็ว 1 Gbps คือ เทคโนโลยีกล้องรูปแบบใหม่ ที่นำมาใช้ควบคู่กับอุตสาหกรรม ได้มีการนำไปใช้ในระบบ machine vision และ intelligent traffic system และ medical imaging

2.2 การทำงานของกล้อง

จากทฤษฎีการเกิดภาพคู่ด้วยวิธีการถ่ายภาพ เราสามารถนำภาพที่ได้มาคำนวณหาระยะห่างระหว่างวัตถุกับกล้อง โดยเราต้องเข้าใจหลักการทำงานของกล้องเสียก่อน

2.2.1 หลักการทำงานของกล้องบันทึกวิดีโอ

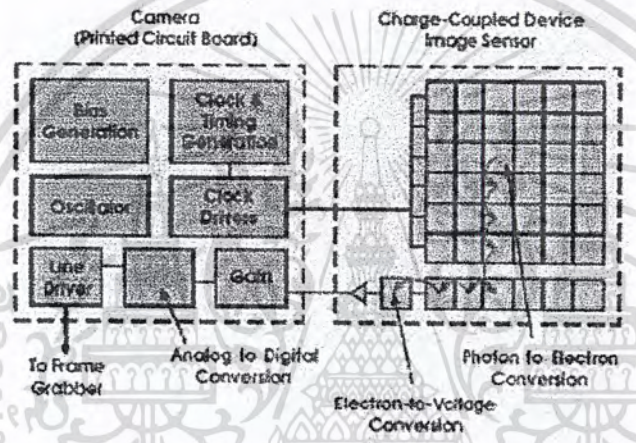
การถ่ายภาพที่จริงแล้วก็คือการวาดภาพด้วยแสง โดยอาศัยกล้องเป็นตัวกลางถ่ายทอดแสงลงบนแผ่นฟิล์มหรือตัวรับภาพในกล้องดิจิตอล หากไม่มีแสงก็ไม่มีภาพ

2.2.2 กลไกการทางานของกล้องดิจิตอล

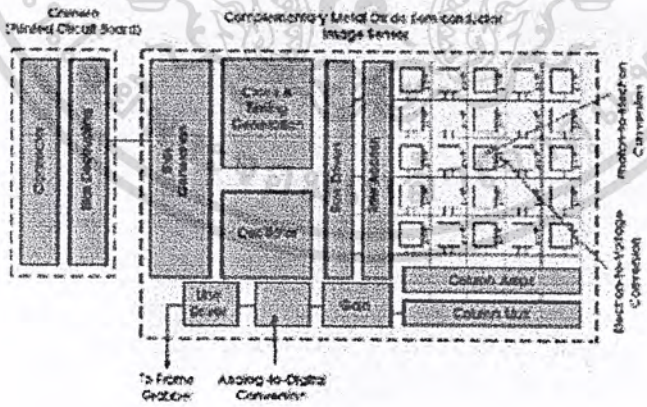
หัวใจของกล้องถ่ายภาพก็คือ การรับภาพและการแปลงภาพ สมัยก่อนกล้องใช้ฟิล์มก็อาศัยหลักการว่า เมื่อภาพที่ผ่านจากเลนส์มาตกกระทบที่แผ่นฟิล์ม จะทำให้แสงไปตกกระทบที่ฟิล์มในระดับต่างๆ กัน หลังจากฟิล์มได้รับแสงแล้วจะทำให้เกิดการเปลี่ยนแปลงและเกิดเป็นภาพขึ้น
ในส่วนของกล้องดิจิตอลนั้นก็มีหลักการที่คล้ายกับกล้องใช้ฟิล์ม แต่ต่างกันที่กล้องดิจิตอลใช้ตัวเซ็นเซอร์ในการแปลงภาพแทนฟิล์ม ซึ่งเซ็นเซอร์นั้นแบ่งออกเป็น 2 ประเภทคือ CCD และ CMOS

CCD (Charge Coupled Device) และ CMOS (Complementary Metal Oxide Semiconductor) ใช้หลักการทำงาน คือ ใช้ Photosite เปลี่ยนแสงที่ตกกระทบให้กลายเป็นอิเล็กตรอน เพื่อบ่งบอกค่าของแสงสีนั้นๆ ภายในตัว Sensor ทั้งสองชนิดนี้จึงประกอบ ไปด้วย Photosite ขนาดเล็กๆ นับล้านๆ ชิ้น เพื่อทำหน้าที่ในการรับแสง ข้อแตกต่างของทั้งสองเทคโนโลยีเกิดขึ้นเมื่อเข้าสู่ขั้นตอนของการคำนวณค่าของแสงนั้นๆ จากแต่ละ Photosite ในอุปกรณ์ ที่ใช้ CCD นั้น จะทำการประจุค่านั้นๆ โดยตรง ในแต่ละ Photosite จากนั้นจะแปลงค่าอนาล็อกของแสงที่ตกกระทบให้กลายไปเป็นค่าดิจิตอล กระบวนการทั้งหมดนี้สามารเกิดขึ้น ได้อย่างรวดเร็ว แตกต่างจาก CMOS ที่ถึงแม้ว่า แต่ละ Photosite จะสามารถประจุค่าได้โดยตรงเช่นกัน แต่กระบวนการส่งผ่านข้อมูลเหล่านั้น ยังต้องอาศัยสายขนาดเล็กมากๆเพื่อทำการส่งผ่านข้อมูลเหล่านั้น เพื่อประมวลผล อีกต่อหนึ่ง

ในกระบวนการผลิตนั้น CCD จะใช้กรรมวิธีพิเศษในการสร้างความสามารถในการส่งผ่านประจุโดยตรงไปยังตัว Chip โดยไม่มีปัญหาในการตัดทอนสัญญาณ ซึ่งด้วยวิธีการนี้จึง ต้องอาศัยขบวนการผลิตที่มีคุณภาพสูงมากเป็นพิเศษ เพื่อให้ได้ตัว Sensor ที่มีคุณภาพและมีความไวต่อแสงอย่างยิ่งยวด ในขณะที่ CMOS นั้น ยังอาศัยเทคโนโลยีการผลิตในรูปแบบเดิมๆ ซึ่งอาศัยเทคนิค เช่นเดียวกับ การผลิต Microprocessor ดังนั้น ด้วยความแตกต่าง ในเทคโนโลยีและกระบวนการผลิตนี้เอง ทำให้ทั้ง CCD และ CMOS นี้ มีต้นทุนในการผลิตและมีคุณภาพของการประมวลผล ที่แตกต่างกันไปด้วย



รูปภาพที่ 2.8 โครงสร้างของ CCD



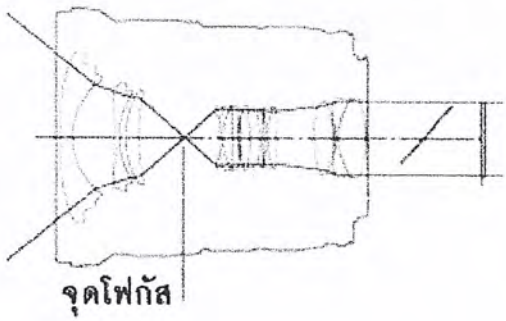
รูปภาพที่ 2.9 โครงสร้างของ CMOS

ตาราง 2.6 การเปรียบเทียบคุณสมบัติระหว่าง CCD และ CMOS

	CCD	CMOS
Signal Out of Pixel	Electron Packet	Voltage
Signal Out of Chip	Voltage (Analog)	Bits (Digital)
Signal Out of Camera	Bits (Digital)	Bits (Digital)
Fill Factor	High	Moderate
Amplifier Mismatch	N/A	Moderate
System Noise	Low	Moderate to High
System Complexity	High	Low
Sensor Complexity	Low	High
Camera Components	PCB + Multiple Chips + Lens	Chip + Lens
Responsibility	Moderate	Slightly Better
Dynamic Range	High	Moderate
Uniformity	High	Low to Moderate
Uniform Shuttering	Fast, Common	Poor
Speed	Moderate to High	Higher
Windowing	Limited	Extensive
Antiblooming	High to None	High
Biasing and Clocking	Multiple, Higher Voltage	Single, Low-Voltage

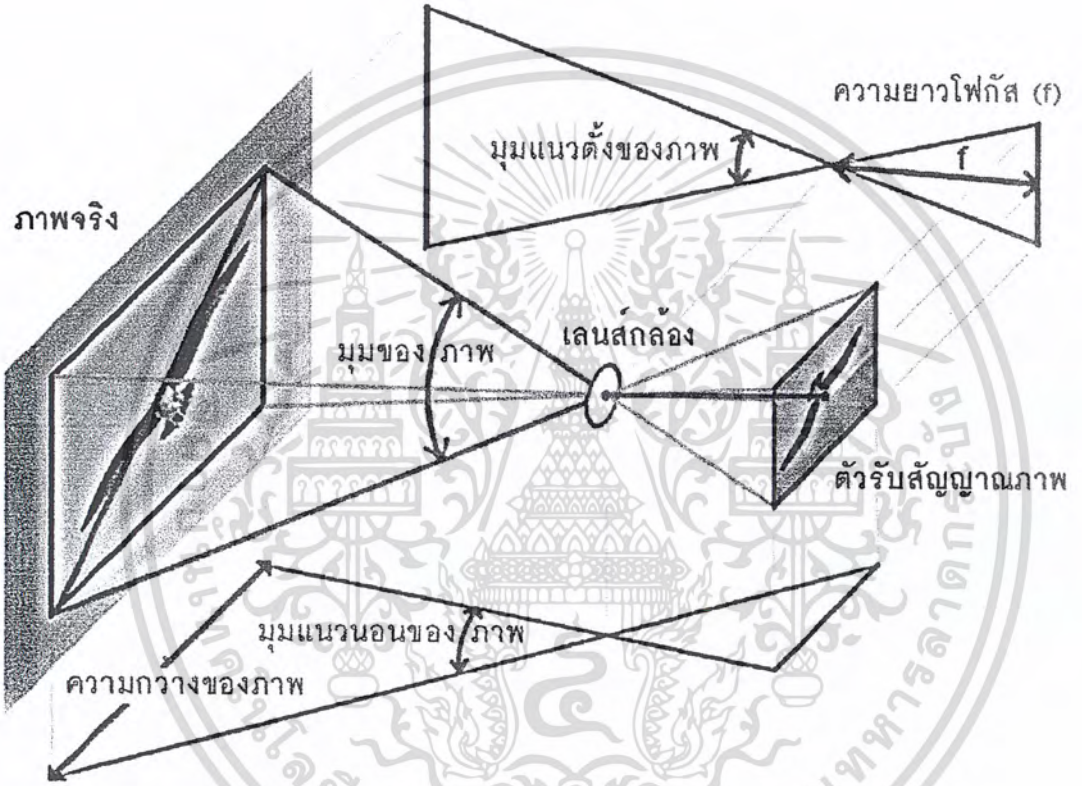
2.2.3 ทางยาวโฟกัสของเลนส์ (Lens Focal Length)

เลนส์เป็นส่วนประกอบที่สำคัญของกล้องถ่ายภาพ ในกระบอกเลนส์จะประกอบไปด้วยชิ้นเลนส์หลายชิ้น จัดเรียงกันเป็นกลุ่มเพื่อทำหน้าที่ในการโฟกัสแสงให้ไปตกยังตัวรับภาพ (ฟิล์มหรือตัวรับภาพในกล้องดิจิตอล) เป็นตัวกลางในการถ่ายทอดแสงที่สะท้อนจากวัตถุไปยังตำแหน่งของตัวรับภาพ ดังนั้นคุณภาพของเลนส์จึงนับว่ามีส่วนสำคัญที่สุดส่วนหนึ่งในการบันทึกภาพ การถ่ายทอดความคมชัดทั้งในแง่ของรายละเอียดและสีสันเลนส์ที่ใช้ในการบันทึกภาพจะมีทางยาวโฟกัสที่หลากหลาย เพื่อถ่ายทอดมุมมองหรือองศาการรับภาพที่แตกต่างกันตามวัตถุประสงค์ของผู้ใช้งาน เลนส์ในปัจจุบันที่นิยมใช้กันส่วนใหญ่จะเป็นเลนส์ซูม (มีทางยาวโฟกัสปรับเปลี่ยนได้) ทำให้สามารถบันทึกภาพได้ในหลายมุมมองโดยไม่ต้องเปลี่ยนตำแหน่งในการบันทึกภาพ



รูปภาพที่ 2.10 จุดโฟกัสในกล้องดิจิตอล

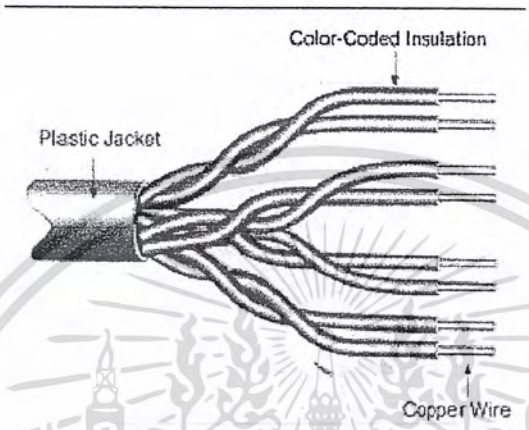
ซึ่งเลนส์ด้านหน้าจะเป็นเลนส์นูนเมื่อแสงผ่านเข้ามา แสงก็จะเอียงเข้าหากันมารวมกันเป็นรูปกรวยจากนั้นก็จะกระจายออกมา ซึ่งจุดรวมแสงนี้เรียกว่าจุดโฟกัส (Focal Point) ซึ่งการกระจายแสงหลังจากผ่านจุดโฟกัสจะกระจายแสงออกจนได้แสงคุณภาพครอบคลุมพื้นที่ตัวรับสัญญาณภาพ ภาพที่ได้จะเป็นภาพจริงหัวกลับ ซึ่งระยะห่างจากจุดโฟกัสไปยังตัวรับสัญญาณภาพจะเรียกว่าความยาวโฟกัส (Focal Length)



รูปภาพที่ 2.11 จุดโฟกัสในกล้องดิจิตอลตามแกน x, y

2.2.4 สาย UTP (Unshielded Twisted-Pair)

ภายในประกอบด้วยจำนวนลวดตัวนำ (Copper Wire) ที่หุ้มด้วยฉนวน (Insulation) ตั้งแต่ 1 คู่ขึ้นไป โดยแต่ละคู่จะบิดกันเป็นเกลียว และถูกห่อหุ้มด้วยเปลือกพลาสติก (Plastic Jacket)



รูปที่ 2.12 ภาพสาย UTP (Unshielded Twisted-Pair)

ข้อสังเกตของสาย UTP คือ จะไม่มีชีลด์ห่อหุ้มสายสัญญาณ ส่งผลให้สายชนิดนี้ถูกสัญญาณรบกวนได้ง่าย แต่ก็เป็นสายที่ถูกนามาใช้งานมากที่สุด เนื่องจากมีราคาถูก

มาตรฐาน ANSI/TIA/EIA 568 เป็นมาตรฐานที่ช่วยสร้างความมั่นใจด้านประสิทธิภาพแก่ผลิตภัณฑ์สายสัญญาณรุ่นต่างๆ โดยปัจจุบันได้ผนวกประเภทของสาย UTP ไว้ 7 ประเภท ใช้คาย่อว่า CAT (Category) แล้วตามด้วยหมายเลขที่แสดง

ระดับคุณภาพของสาย ซึ่งประกอบด้วย

1. CAT 1 (Category 1) แก้เลขหน้า

เป็นสาย UTP มาตรฐานที่นามาใช้กับระบบโทรศัพท์ ได้รับการออกแบบเพื่อนาส่งข้อมูลเสียงหรือข้อมูลดิจิตอลด้วยความเร็วต่า สามารถใช้งานได้ดีกับการส่งข้อมูลเสียงหรือข้อมูลดิจิตอลที่มอดูเลตเป็นสัญญาณแอนะล็อก ซึ่งสามารถส่งผ่านบนระยะทางหลายไมล์จากบ้านไปยังชุมสายโทรศัพท์ แต่สาย CAT 1 จะไวต่อสัญญาณรบกวนและการอ่อนตัวของสัญญาณ

2. CAT 2 (Category 2)

เป็นสาย UTP ที่นามาใช้กับวงจรโทรศัพท์ ภายในมีสายสัญญาณจานวน 4 คู่ แต่คุณภาพดีกว่าสาย CAT 1 ด้วยการลดสัญญาณรบกวนลง และเรื่องการเบาบางของสัญญาณ สายประเภทนี้พบได้ในสายชนิด T-1 และ ISDN T-1 ออกแบบมาสาหรับโทรศัพท์ดิจิตอลที่สามารถส่งผ่านเสียงหรือข้อมูลได้ที่ความเร็ว 1.544 Mbps ISDN เป็นเทคโนโลยีใหม่ที่สามารถส่งผ่านเสียง ข้อมูล หรือทั้งสองด้วยความเร็ว 64 Kbps ถึง 1.511 Mbps โดยสาย CAT 2 จะใช้ลวดทองแดงและฉนวนที่หุ้มลวดทองแดงที่มีคุณภาพสูงกว่าแบบ CAT 1

3. CAT 3 (Category 3)

เป็นสาย UTP ที่ภายในมีสายสัญญาณจานวน 4 คู่ โดยมีข้อกาหนดระบุไว้ว่า สายแต่ละคู่จะต้องนามาบิดเกลียวอย่างน้อย 3 รอบต่อ 1 ฟุตรองรับความเร็วในการส่งข้อมูลที่ 10 Mbps สายชนิดนี้นามาใช้กับเครือข่ายอีเทอร์เน็ต (10 Mbps) โทเคนริง (4 Mbps) ที่ใช้งานบนเครือข่ายยุคเก่า ปัจจุบันสายประเภทนี้ถูกทดแทนด้วยสาย CAT 5 เนื่องจากมีประสิทธิภาพเหนือกว่า

4. CAT 4 (Category 4)

เป็นสาย UTP ที่ภายในมีสายสัญญาณจานวน 4 คู่ รองรับความเร็วสูงสุดที่ 20 Mbps ถูกออกแบบให้ป้องกันสัญญาณรบกวนได้ดีกว่า CAT 1, CAT 2 และ CAT 3 รวมถึงส่งข้อมูลได้เร็วกว่า

5. CAT 5 (Category 5)

เป็นสาย UTP ที่ภายในมีสายสัญญาณจานวน 4 คู่ รองรับความเร็วสูงถึง 100 Mbps นิยมนามาใช้กับเครือข่ายท้องถิ่น โดยสายสัญญาณทั้ง 4 คู่ (8 เส้น) จะถูกหุ้มด้วยฉนวนพลาสติกที่มีสีต่างๆ กากับอยู่ จัดเป็นสายสัญญาณที่ป้องกันสัญญาณรบกวนได้ดี เนื่องจากการบิดเกลียวของสายที่มี Twist Ratio จานวนมากถึง 12 รอบต่อความยาว 1 ฟุต ปัจจุบันสาย CAT 5 จัดเป็นมาตรฐานขั้นต่าที่นามาใช้บนเครือข่ายท้องถิ่น

6. CAT 5e (Enhanced Category 5)

เป็นสาย UTP ที่มีคุณภาพสูง โดยใช้ลวดตัวนาสัญญาณคุณภาพสูง และมีการบิดเกลียวของ Twist Ratio ที่เพิ่มขึ้น จึงสามารถป้องกันสัญญาณรบกวนแบบครอสทอล์กได้เป็นอย่างดี สาย CAT 5e สามารถนามาใช้งานบนเครือข่ายท้องถิ่น รองรับความเร็วสูงสุดที่ 100 Mbps

7. CAT 6 (Category 6)

เป็นสาย UTP ที่รองรับความเร็วสูงถึง 1 Gbps เป็นสายคู่บิดเกลียวที่เพิ่มส่วนของฉนวนที่เรียกว่า ฟอยล์ (Foil) ซึ่งเป็นแผ่นโลหะบางๆ ใช้ป้องกันสัญญาณรบกวนได้ดียิ่งขึ้น รองรับอัตราความเร็วที่ 250 MHz

8. CAT 6e (Enhanced Category 6)

เป็นสายที่พัฒนามาจาก CAT 6 เพื่อให้มีประสิทธิภาพมากยิ่งขึ้น โดยสามารถรองรับอัตราความเร็วได้สูงถึง 550 MHz และส่งผ่านข้อมูลได้หลายกิกะบิตต่อวินาที

9. CAT 7 (Category 7)

เป็นสายคุณภาพสูง โดยสายแต่ละคู่นอกจากจะมีฉนวนฟอยล์ป้องกันสัญญาณรบกวนแล้ว ยังมีชีลด์ที่เป็นเส้นใยโลหะถักห่อหุ้มเพิ่มเข้าไปอีก อย่างไรก็ตาม CAT 7 ยังไม่ได้รับรองเป็นมาตรฐานที่ชัดเจน แต่ได้มีการผลิตและจำหน่ายบ้างแล้ว

ข้อดีของสาย CAT 7 คือ รองรับความเร็วสูงสุดที่ 1 GHz แต่จะใช้คอนเน็กเตอร์ที่แตกต่างจากสาย UTP (ปกติคือ RJ-45) เนื่องจากภายในมีฉนวนฟอยล์พร้อมชีลด์ห่อหุ้มจึงทำให้ขนาดของสายใหญ่ขึ้นกว่าเดิม

รูปภาพที่ 2.13 ภาพสายแลน CAT 6 (สายที่ใช้ในการทดลอง)

2.2.5 ขั้นตอนการทำงานจากการใช้ข้อมูลบันทึกความเคลื่อนไหว

ในส่วนของการบันทึกความเคลื่อนไหวนั้น การเคลื่อนไหวที่จุดใดจุดหนึ่งขึ้นไป จะถูกนามาวิเคราะห์เป็นจำนวนหลายๆ ครั้งในเวลาหนึ่งวินาที อย่างไรก็ตามเทคนิคส่วนใหญ่ที่ใช้สำหรับการบันทึกการบันทึกความเคลื่อนไหวนั้นคือการบันทึกการเคลื่อนไหวของผู้ป่วย ในลักษณะอริยาบทต่างๆ เช่น การเดิน การวิ่ง การนั่ง หรือแม้แต่การกระโดด ซึ่งข้อมูลเหล่านี้จะถูกเก็บไว้ในรูปแบบข้อมูลดิจิตอล ข้อมูลดิบเหล่านี้ยังไม่สามารถไปใช้ประโยชน์ได้อย่างเต็มประสิทธิภาพ จำเป็นที่จะต้องทำการแปลงข้อมูลที่ได้ให้

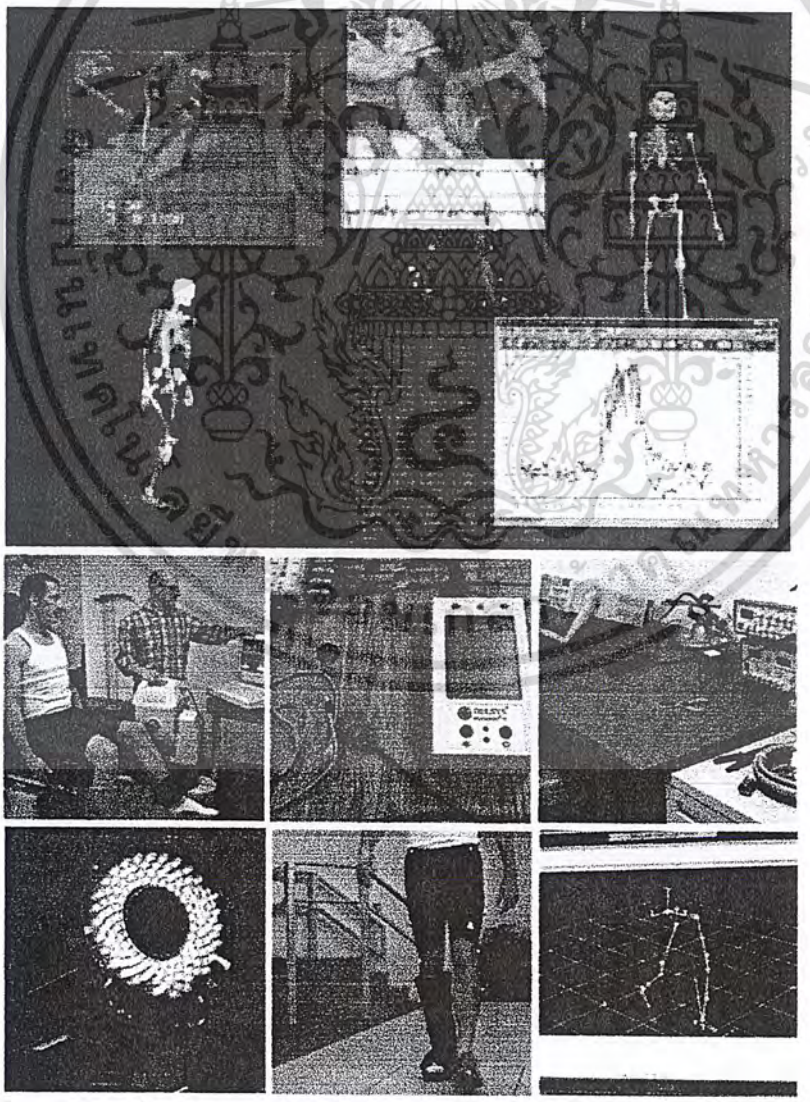
ออกมาในรูปแบบของโมเดลสามมิติเพื่อความสะดวกของแพทย์ผู้ใช้งาน ซึ่งแพทย์ผู้เชี่ยวชาญในสาขาต่างๆ จะนำข้อมูลเหล่านี้ไปทำการตรวจวิเคราะห์คนไข้ของตน

2.3 การประยุกต์ใช้งานการบันทึกความเคลื่อนไหว

การบันทึกความเคลื่อนไหวสามารถนำไปประยุกต์ใช้งานด้านต่างๆ ได้ เช่น

2.3.1 การประยุกต์ใช้งานในด้านทางการแพทย์

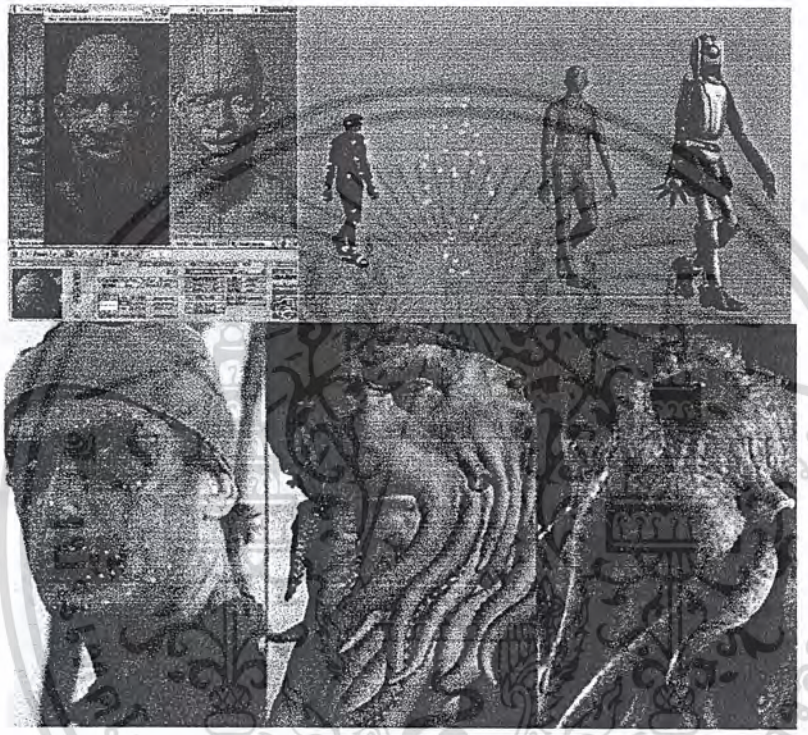
การบันทึกความเคลื่อนไหวนั้นเป็นการประยุคต์การใช้งานเพื่อใช้แพทย์และผู้ป่วยนั้นนั้นมีความสะดวก และเพิ่มความแม่นยำในการวินิจฉัยมากยิ่งขึ้น โดยจะทำการเก็บข้อมูลจำเพาะที่ของผู้ป่วย เช่น ลักษณะการเดินของผู้ป่วย หรืออริยาบทต่างๆของคนไข้ เป็นต้น เพื่อที่จะนำข้อมูลเหล่านั้นมาทำการวิเคราะห์หาสาเหตุ หนทางการรักษาผู้ป่วยต่อไป ซึ่งในวงการแพทย์เริ่มมีความสนใจในเทคโนโลยีนี้เพิ่มมากขึ้นตามลำดับ



รูปภาพที่ 2.14 ภาพตัวอย่างการประยุกต์ใช้งานในทางการแพทย์

2.3.2 การประยุกต์ใช้งานในด้านอุตสาหกรรมบันเทิง

การบันทึกความเคลื่อนไหวช่วยลดเวลาในการสร้างภาพเคลื่อนไหว ทั้งยังทำให้ได้การเคลื่อนไหวที่สมจริง มากกว่าการสร้างการเคลื่อนไหวด้วยมือ ทำให้การบันทึกความเคลื่อนไหวเป็นที่นิยมใช้กันมากในอุตสาหกรรมแอนิเมชันหรือบันเทิง เช่น การนำไปใช้ สร้างการเคลื่อนไหวให้กับตัวละครในเกมส์ คอมพิวเตอร์ ภาพยนตร์แอนิเมชัน หรือแม้แต่การตกแต่งด้วยคอมพิวเตอร์ในการสร้างภาพยนตร์



รูปภาพที่ 2.15 ภาพตัวอย่างการประยุกต์ใช้งานในด้านอุตสาหกรรมบันเทิง

2.4 การติดตามการเคลื่อนไหว (Tracking Motion)

2.4.1 พื้นฐานของการติดตามการเคลื่อนไหว

เมื่อต้องทางานกับไฟล์วิดีโอซึ่งมีความแตกต่างจากไฟล์ภาพนิ่ง เรามักจะเลือกจับภาพเฉพาะวัตถุใดวัตถุหนึ่งหรือจับภาพตามวัตถุเคลื่อนที่ที่เราต้องการให้อยู่ภายในกรอบของภาพ ในการติดตามนั้นต้องใช้ความเข้าใจในเรื่องการเคลื่อนที่ของวัตถุซึ่งแบ่งการทางานออกเป็นสองส่วนประกอบหลักๆ นั่นคือการจำแนกลักษณะและการทำโมเดล การจำแนกลักษณะมีความสำคัญในการค้นหาวัตถุที่เราสนใจ จากภาพหนึ่งเฟรมในแต่ละเฟรมที่ต่อเนื่องกัน (Subsequent Frame) ของไฟล์วิดีโอ เทคนิคในเรื่องของโมเมนต์ (Moment) หรือฮิสโตรแกรม (Histograms) ของสีจะช่วยให้สามารถจำแนกวัตถุที่ต้องการได้ สิ่งที่จะทำการติดตามที่ยังจำแนกลักษณะไม่ได้จะเป็นปัญหาเกี่ยวข้องสัมพันธ์กัน การติดตามวัตถุโดยปราศจากการจำแนก

ลักษณะของวัตถุจะใช้ก็ต่อเมื่อต้องการจะพิจารณาการเคลื่อนไหวของวัตถุต่างๆ ว่าวัตถุใดที่มีการเคลื่อนไหวซึ่งน่าสนใจต่อการติดตาม เทคนิคสำหรับการติดตามวัตถุ โดยปราศจากการจำแนกลักษณะมักจะเกี่ยวข้องกับการติดตามวัตถุที่เมื่อสังเกตด้วยตาดูแล้วน่าจะเป็นวัตถุที่มีนัยสำคัญกว่าวัตถุอื่นๆ

2.5 สามมิติด้วยสเตอริออพซิส

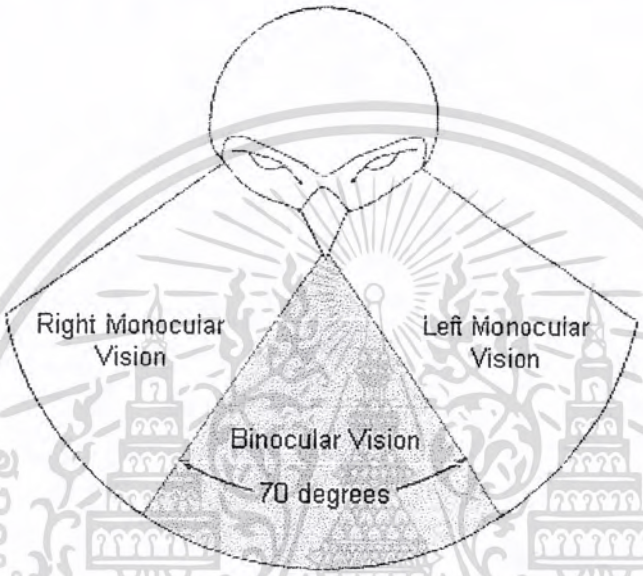
สเตอริออปซิส มาจากคาว่า สเตอริโอ (Stereo) และ ออปซิส มาจากคาว่าวิชัน (Vision) หมายถึง ความสามารถในการมองเห็นวัตถุ โดยเกิดจากกระบวนการหาค่าความลึกหรือตำแหน่งจริงของวัตถุในสามมิติจากภาพถ่ายตั้งแต่ 2 ภาพขึ้นไปในหลายๆมุม ซึ่งภาพถ่ายจะต้องมีมุมมองไปยังวัตถุที่สนใจนั้นๆ แตกต่างกัน ทำให้ภาพที่ได้มีตำแหน่งของวัตถุในภาพไม่เหมือนกัน การคำนวณหาตำแหน่งจริงของวัตถุเกิดจากการหาตำแหน่งของวัตถุในภาพกับตำแหน่งและมุมมองของกล้องที่ใช้ในการมองวัตถุนั้นมาคำนวณหาค่าความลึกของวัตถุ การมองเห็นวัตถุด้วยวิธีสเตอริออปซิสถูกนาไปใช้ประโยชน์ในด้านต่างๆ เช่น การมองเห็นของมนุษย์ การมองเห็นของหุ่นยนต์ การสารวจจากระยะไกล การวัดและจำแนกวัตถุสามมิติ

รูปภาพที่ 2.16 ภาพสเตอริโอสองภาพที่นามาซ้อนกัน

2.5.1 ธรรมชาติของการมองเห็นภาพสามมิติ

2.5.1.1 การเห็นวัตถุของมนุษย์ในสภาวะสามมิติ (Stereoscopic Vision)

การเห็นวัตถุของมนุษย์ในสภาวะสามมิติ ซึ่งมนุษย์มองเห็นระยะลึกของวัตถุที่เห็นได้ด้วยการมองในสองลักษณะ ลักษณะแรกคือการเห็นภาพจากสองตา (Binocular Vision) คือการที่เรามีตาสองข้างเหมือนกัน แล้วมองวัตถุเดียวกัน ตาแต่ละข้างก็จะได้ภาพของวัตถุที่ต่างมุมมองกัน ทำให้เห็นทรวดทรงของวัตถุได้



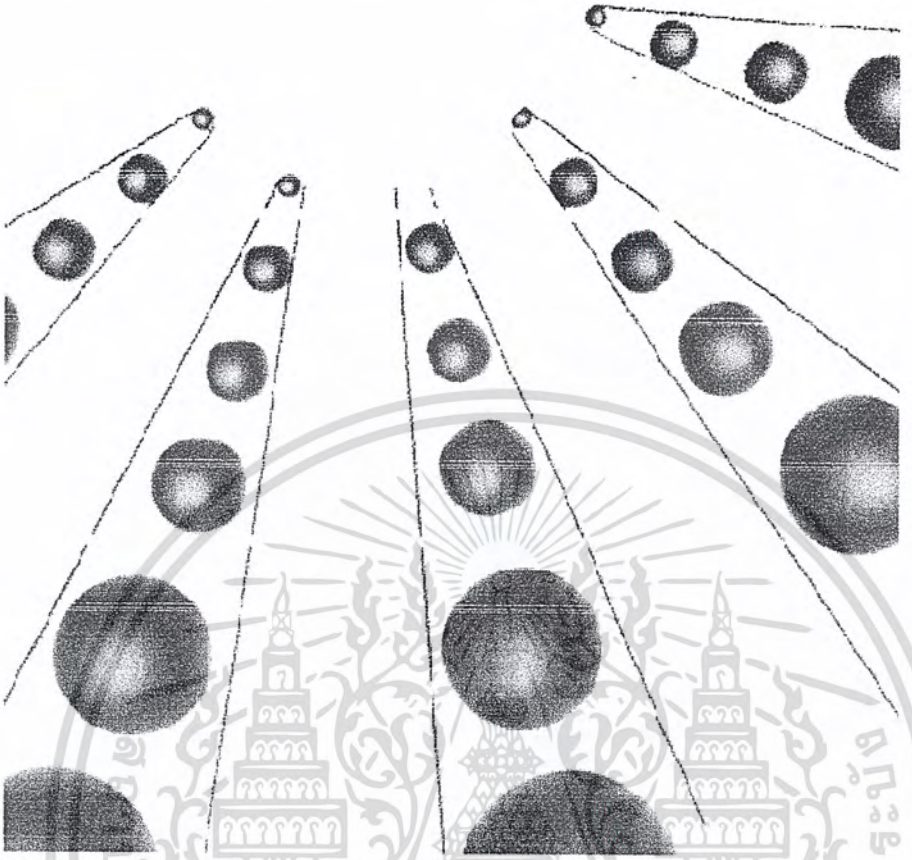
รูปภาพที่ 2.17 ภาพแสดงการมองเห็นของมนุษย์แบบ Binocular Vision

ลักษณะที่สองคือการเห็นภาพจากการมองแบบใช้ตาข้างเดียว (Monocular Vision) คือการที่เราสามารถเห็นวัตถุและเข้าใจในลักษณะสามมิติได้ด้วยการมองเห็นด้วยตา ข้างเดียวได้โดยอาศัยหลักการต่างๆ เหล่านี้อย่างใดอย่างหนึ่งประกอบกัน คือ

1) ความคุ้นเคย คือการที่มนุษย์สามารถใช้ความคุ้นเคยจากการได้เห็นมาก่อนแล้วช่วยในการตัดสินใจว่าวัตถุมีลักษณะสามมิติหรือไม่ ดังนั้นคนที่มีตาเพียงข้างเดียวมาตั้งแต่กำเนิดก็จะ ไม่มีความเข้าใจการเห็นลักษณะสามมิติที่แท้จริงได้

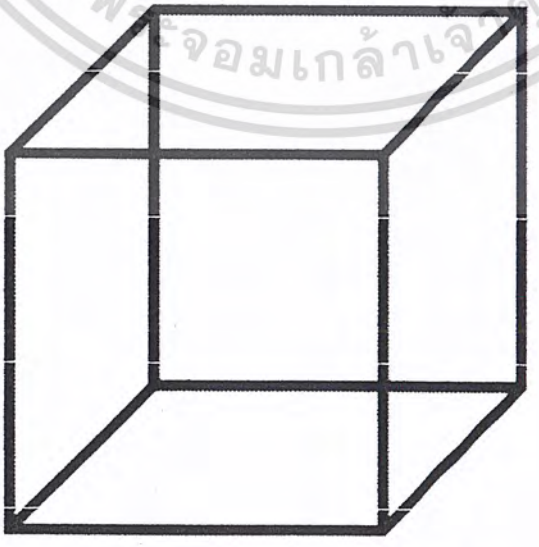
2) การรับรู้และเข้าใจได้โดยการใช้ปัจจัยเหล่านี้ประกอบ คือ

2.1) การเปรียบเทียบขนาดสัมพัทธ์ของวัตถุ (Relative size) คือ การใช้การเปรียบเทียบขนาดของวัตถุอย่างเดียวกัน หรือต่างชนิดกันที่รู้จัก ทำให้สามารถมองออกว่าวัตถุวางตำ แหน่งเหลื่อมหรือใกล้ ไกลจากผู้สังเกตุอย่างไร สภาพการเหล่านี้คือ การเห็นและเข้าใจในลักษณะของสามมิตินั่นเอง



รูปภาพที่ 2.18 ตัวอย่างภาพของ Relative Size

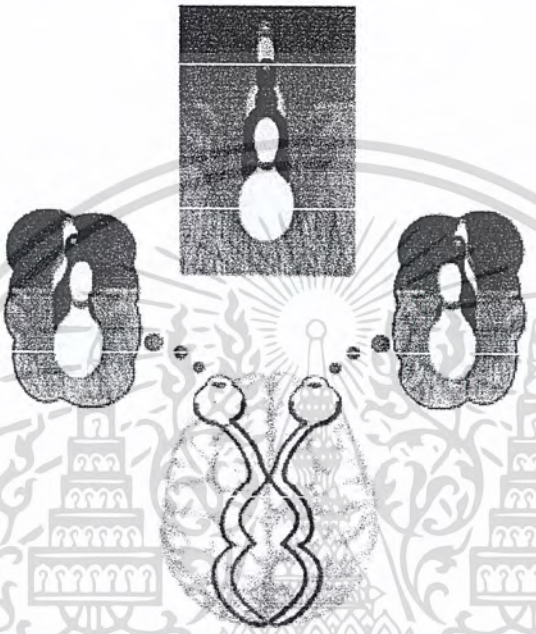
2.2) การดูรูปร่างของวัตถุ (Shape of objects) คือ เมื่อรูปร่างของวัตถุที่เห็นมีทรวดทรงที่ทำให้การมองด้วยตาเดียว แล้วสามารถวินิจฉัย ได้ว่ามีลักษณะขนาดใดในสามมิติ



รูปภาพที่ 2.19 ตัวอย่างภาพของ Shape of objects

2.5.1.2 หลักการเกิดภาพสามมิติโดยภาพสเตอริโอ

หลักการอย่างง่ายของการมองภาพให้เกิดภาพเป็นสามมิตินั้น คือ การที่ มนุษย์มี 2 ตา การที่สามารถมองเห็นภาพต่างๆ เป็นสามมิติได้นั้นเกิดจากมุมมองของสายตาที่เห็น ภาพของวัตถุ เมื่อมองวัตถุด้วยตาข้างใดข้างหนึ่งเพียงข้างเดียว จะไม่สามารถมองให้เป็น สามมิติ ได้เพราะจะขาดส่วนลึกของภาพอีกด้านหนึ่งไป



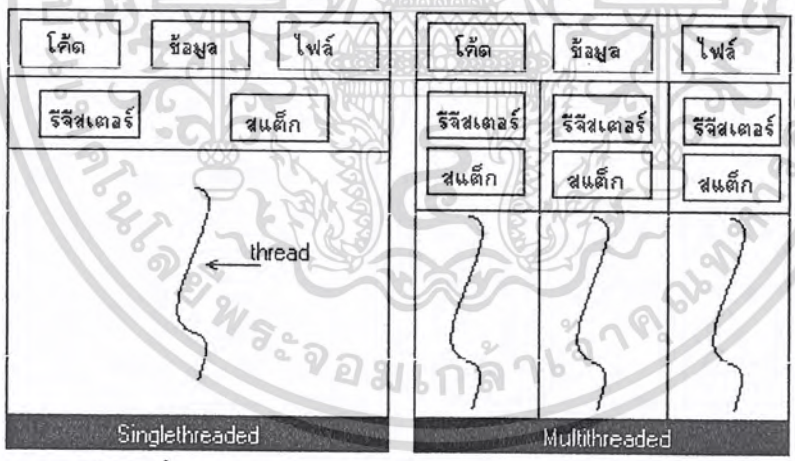
รูปภาพที่ 2.20 ภาพการมองเห็นในรูปแบบ 3D ของตามนุษย์

จะเห็นว่าตาแต่ละข้างจะมีมุมมองต่างกัน ถ้าหากปิดตาทีละข้าง คือ เมื่อปิดตาข้างขวาและมองวัตถุด้วยตาซ้ายจะมองเห็นด้านข้าง (ส่วนลึก) ด้านซ้ายของวัตถุ และหากปิดตาซ้ายใช้ตาข้างขวามองวัตถุจะมองเห็นด้านข้าง (ส่วนลึก) ด้านขวา จากนั้นเมื่อลืมตา 2 ข้างพร้อมกัน ตาแต่ละข้างจะมองเห็นวัตถุทั้ง 2 ด้านคือด้านซ้ายและขวาพร้อมกัน ภาพที่มองเห็นจากตาทั้งสองข้างนี้จะผ่านไปสู่กระบวนการของสมองที่จะรวมและประมวลผลภาพทั้งสองให้เป็นรูปเดียวกัน ทาให้มองเห็นเป็นภาพมิติของวัตถุนั้นๆ โดยอาศัยหลักการมองเห็นของตามนุษย์นี้เองจึงสามารถจาลองการมองภาพทั่วไปให้เป็นภาพสามมิติได้

2.6 Multithreaded Programming

ในอดีตการประมวลผลเป็นแบบ single thread ที่ CPU ถูกครอบครองโดย process ครั้งละ 1 process เท่านั้น แต่ปัจจุบันระบบปฏิบัติการยอมให้เป็นระบบ Multithreaded ซึ่งในแต่ละระบบปฏิบัติการมีรูปแบบแตกต่างกันไปเช่น Java, Windows, Linux หรือ Unix ในบางครั้งเราเรียก thread ว่า LWP (Light Weight Process) ซึ่งเป็นความสามารถพื้นฐานของ CPU ในการจัดสรร Thread ID, Program counter, Register set และ Stack ให้กับทุก ๆ Thread ส่วนในอดีตเราจะเรียกว่า Heavy weight process เพราะเป็นแบบ Single thread of control

ซอฟแวร์หลาย ๆ ตัวที่รันอยู่บนคอมพิวเตอร์ PC ที่ทันสมัยคือระบบ Multithreaded การประยุกต์ใช้แบบฉบับของมันคือการควบคุมจะแยกโปรเซสออกไปหลาย ๆ เส้นงาน อย่าง Web Browser ก็ต้องใช้หนึ่งเส้นงานเพื่อแสดงรูปภาพหรือตัวอักษรในขณะที่เส้นอื่น ๆ ดึงเอาข้อมูลจากเครือข่ายตัวอย่างเช่น word processing เส้นงานหนึ่งจะแสดงในส่วนของการแสดงผลของตัวอักษร เส้นงานที่สองจะใช้ในเรื่องของการตอบสนองจากคีย์บอร์ด ส่วนเส้นที่สามจะช่วยในเรื่องของการตรวจสอบคำผิด เป็นต้น



รูปภาพที่ 2.21 รูปภาพเปรียบเทียบของการ Multithreaded

ในสถานการณ์บางอย่างการทำงานแบบ single อาจจะต้องการทำงานพร้อมกันหลาย ๆ งาน เช่น web server ยอมรับสิ่งที่เครื่องลูกข่ายต้องการพวก web page image sound ถ้าเครื่องให้บริการมีระบบการทำงานแบบ single เครื่องลูกข่ายนับพัน ๆ เครื่องที่ติดต่อเข้ามาก็จะได้รับการให้บริการเพียงเครื่องเดียวเท่านั้น

วิธีหนึ่งคือให้ server run โปรเซสขึ้นมาหนึ่งโปรเซสและรอรับ request เมื่อได้รับแล้ว จะสร้างโปรเซสแยกออกมาเพื่อให้บริการในทุก request ที่ขอมา ซึ่งจริง ๆ แล้วการสร้างโปรเซสในลักษณะนี้เป็นวิธีแบบธรรมดาที่เคยใช้กันก่อนที่จะมีระบบ เส้นงานขึ้นมาอีก การสร้างโปรเซสต้องใช้เวลาในการสร้างมากและต้องละเอียดถี่ถ้วน ดังที่แสดงให้ดูในบทก่อนหน้า ถ้ามีโปรเซสใหม่ถูกสร้างขึ้นก็จะทำงานเหมือนกับโปรเซสเดิมที่มีอยู่แล้ว และทำไมการทำงานแบบนี้จึงไม่ overhead

2.6.1 ข้อได้เปรียบหรือประโยชน์ของ multithread programming มีอยู่ 4 ข้อหลัก ๆคือ

1. การตอบสนอง (Responsiveness) การทำงานแบบ Multithreading สื่อสารกันระหว่าง application จะยอมให้โปรแกรมรันต่อไปได้ถ้าส่วนของโปรแกรมถูก block หรือมีการทำงานหรือรอคำสั่งเป็นเวลานาน ๆ ด้วยวิธีนี้จะเพิ่มการตอบสนองไปยัง user ตัวอย่างเช่น web browser จะยังคงยอมให้ user ติดต่อกันในหนึ่ง thread ในขณะที่รูปภาพถูกโหลดจาก thread อื่น

2. การแชร์ทรัพยากร(resource sharing) โดยปกติ thread จะแชร์มันจะแชร์หน่วยความจำและทรัพยากรกันเอง ข้อดีตรงส่วนนี้ในการแชร์ code และ data คือจะยอมให้ application มีหลาย ๆ thread ที่แตกต่างกันทำงานอยู่บน space เดียวกัน

3. ความประหยัด (Economy) การแบ่งหน่วยความจำและทรัพยากรต่าง ๆ เพื่อที่จะสร้างโปรเซสซึ่งเป็นการฟุ่มเฟือยและใช้เวลามากเพราะว่ามันแชร์และดึงข้อมูลกันอยู่ดังนั้นการทำงานแบบ multithread จึงมีความประหยัดในการสร้างและ context-switch thread

4. การเอื้อเฟื้อประโยชน์ของสถาปัตยกรรม Multiprocessor ประโยชน์ของ Multithreading นั้นมีมากมายในสถาปัตยกรรม Multiprocessor ซึ่งมันสามารถทำงานคู่ขนานกันไปบนตัวสั่งการที่ต่างกันได้ แบบ single thread process นั้นจำถูกจำกัดให้ทำงานบน CPU ตัวเดียวเท่านั้น ส่วนในแบบ Multithread นั้นจะทำงานบน CPU หลายตัวในเวลาเดียวกันได้

2.6.2 รูปแบบหลายเส้นงาน

เราจะอธิบายการทำงานของเส้นงาน ซึ่งมีการพัฒนามาตลอด แต่อย่างไรก็ตามการสนับสนุนการทำงานของเส้นงานจะขึ้นอยู่กับระดับของผู้ใช้ (user level) จากเส้นงานของผู้ใช้หรือจาก kernel แต่เส้นงานของผู้ใช้จะสนับสนุนมากกว่า kernel และสามารถควบคุมโดยไม่ต้องใช้ kernel support ส่วนเส้นงานของ

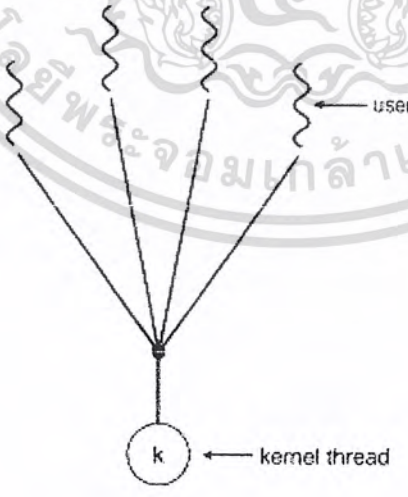
kernel นั้นจะสนับสนุนและควบคุมโดยตรงจากระบบปฏิบัติการ และเกือบทุกรุ่นของระบบปฏิบัติการไม่ว่าจะเป็น Windows XP, Linux, Mac OS X, Solaris และ Tru64 UNIX (สมัยก่อนคือ Digital UNIX) ที่สนับสนุนการทำงานของ kernel

ในที่สุดแล้วเส้นงานของผู้ใช้และเส้นงานของ kernel ก็ยังเชื่อมโยงกันอยู่ดี ในส่วนนี้เราจะพิสูจน์ความสัมพันธ์ซึ่งสามารถแบ่งออกเป็น 3 รูปแบบที่เป็นที่รู้จักกันโดยทั่วไป

2.6.2.1 รูปแบบหลายต่อหนึ่ง (M:1)

รูปแบบหลายต่อหนึ่ง (รูป 4.2) หลายเส้นงานของผู้ใช้จะทำงานกับหนึ่งเส้นงานของ kernel ตัวจัดการเส้นงานนั้นจะแก้ปัญหาจาก thread library ในพื้นที่ของผู้ใช้ ซึ่งมันจะมีประสิทธิภาพมาก แต่กระบวนการทั้งหมดจะกีดขวางกัน ถ้าเส้นงานไปทำการกีดขวาง system call อีกทั้งเป็นเพราะว่ามีเพียงหนึ่งเส้นงานที่สามารถเข้าไปใน kernel ในหนึ่งช่วงเวลา เส้นงานหลายเส้นงานนั้นไปสามารถทำงานขนานในรูปแบบหลายกระบวนการได้

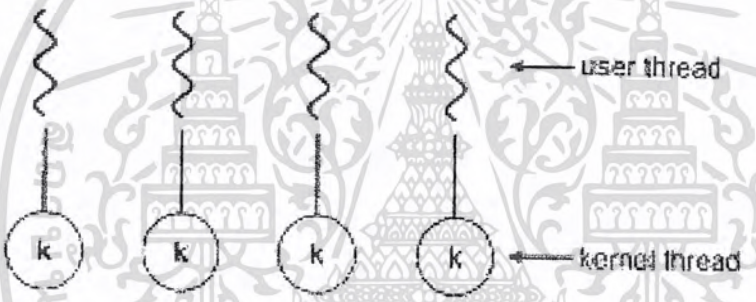
Green threads เป็น thread library ที่สะดวกสำหรับเครื่อง Solaris และระบบจำพวก GNU Portable thread



รูปภาพที่ 2.22 Many-to-one model

2.6.2.2 รูปแบบหนึ่งต่อหนึ่ง (1:1)

รูปแบบหนึ่งต่อหนึ่ง (รูป 4.3) จากรูปในแต่ละเส้นงานของผู้ใช้จะทำงานกับหนึ่งเส้นงานของ kernel ซึ่งมันจะจัดให้มีการทำงานเกิดขึ้นพร้อมกันมากกว่ารูปแบบหลายต่อหนึ่ง (M:1) โดยจะปล่อยให้เส้นงานอื่นทำงานในขณะที่มีเส้นงานทำงานกับ system call อีกทั้งยังปล่อยให้หลายเส้นงานนั้นทำงานในรูปแบบขนานของหลายกระบวนการอีกด้วย มีข้อเสียเพียงข้อเดียวสำหรับรูปแบบนี้คือ การสร้างเส้นงานของผู้ใช้จะต้องสร้างเส้นงานของ kernel ให้สอดคล้องกันเพราะว่า การสร้างเส้นงานของ kernel นั้นรับผิดชอบโดยชั้น application ส่วนใหญ่แล้วการจัดหารูปแบบ การลดค่าของ ค่าสนับสนุนเส้นงานจะถูกกระทำโดยระบบ ไม่ว่าจะเป็นระบบปฏิบัติการ Linux หรือ Windows ก็ตามซึ่งรวมไปถึง Windows 95, 98, NT, 2000 และ XP จะใช้รูปแบบหนึ่งต่อหนึ่ง (1:1) เป็นเครื่องมือ



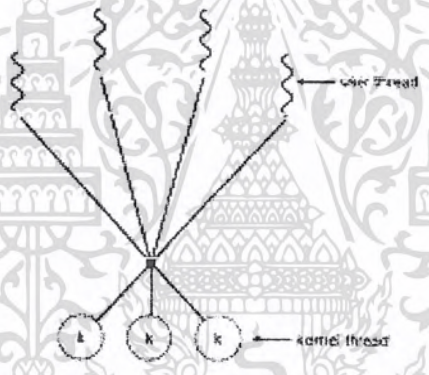
รูปภาพที่ 2.23 One-to-one model

2.6.2.3 รูปแบบหลายต่อหลาย (M:M)

รูปแบบหลายต่อหลาย (รูป 4.4) หลายเส้นงานของผู้ใช้จะทำงานกับเส้นงานกับเส้นงานของ kernel ที่น้อยกว่าหรือเท่ากันได้ จำนวนเส้นงานของ kernel บางครั้งจะถูกระบุเจาะจงจากชั้น application หรือ โดยเครื่อง (การแบ่งเส้นงาน kernel บน application ในกระบวนการหลายจะแบ่งได้มากกว่าแบบกระบวนการเดี่ยว) ที่รูปแบบหนึ่งต่อหนึ่ง (1:1) เป็นที่ยอมรับโดยผู้พัฒนานั้นคือการสร้างเส้นงานที่ผู้ใช้ต้องการ ถูกที่ว่ากระบวนการทำงานพร้อมกันแต่ไม่ได้เข้าไปทำงานใน system call พร้อมกัน เพราะว่า kernel สามารถที่จะทำงานกับหนึ่งเส้นงานในหนึ่งช่วงเวลา รูปแบบหนึ่งต่อหนึ่ง (1:1) ยอมให้เกิดการทำงานพร้อมกันได้มากแต่ในการขยายนั้นจะต้องระวังไม่ให้สร้างเส้นงานมากเกินไปภายใน application

(และในบางกรณีอาจมีข้อจำกัดในกระบวนการสร้างเส้นงาน) รูปแบบหลายต่อหลาย (M:M) ไม่สามารถจัดการกับข้อบกพร่องสองประการคือ ผู้พัฒนาไม่จำเป็นต้องสร้างเส้นงานของผู้ใช้หลายเส้น และเส้นงานของ kernel ที่คล้ายกันสามารถทำงานในแบบขนานบนระบบหลายกระบวนกรได้ อีกทั้งเมื่อมีเส้นงานทำงานกับ system call, kernel สามารถที่จะดำเนินการกับเส้นงานอื่นได้ อีกประการหนึ่งคือ แม้ว่าเส้นงานของผู้ใช้จะมีจำนวนมากกว่าหรือเท่ากันกับจำนวนเส้นงานของ kernel แต่ยังยอมรับว่าเส้นงานของผู้ใช้นั้นมีขอบเขตที่จำกัดต่อหนึ่งเส้นงานของ kernel ความแตกต่างบางอย่างเกี่ยวกับรูปแบบ two level

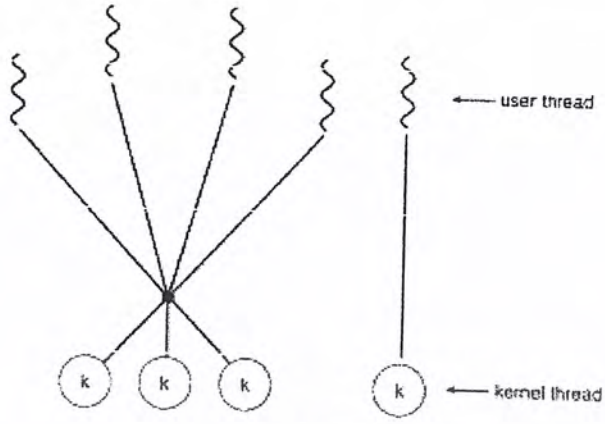
(รูป 4.5) เป็นการรองรับโดยระบบปฏิบัติการ เช่น IRIX, HP-UX และ Tru 64 UNIX ระบบปฏิบัติการ Solaris รองรับรูปแบบ two level ในรุ่นเก่ากว่า Solaris 9 แต่อย่างไรก็ตามระบบ Solaris 9 เริ่มต้นใช้งานด้วยรูปแบบหนึ่งต่อหนึ่ง (1:1)



รูปภาพที่ 2.24 Many-to-many model

2.6.3 Thread Library

Thread Library เป็นข้อกำหนดที่สร้างโดยโปรแกรมเมอร์ สำหรับสร้างและจัดการเกี่ยวกับเส้นงาน โดยมีสองขั้นตอนในการทำงานคือ



รูปภาพที่ 2.25 Two-level model

1. การเข้าถึงแหล่งของข้อมูลทั้งหมดในเนื้อที่ของผู้ใช้ โดยไม่ผ่าน kernel ทุกค่าและทุกโครงสร้างข้อมูลของแหล่งข้อมูลในพื้นที่ของผู้ใช้ หรือเป็นการเรียกจากฟังก์ชันเฉพาะในพื้นที่ของผู้ใช้และไม่ใช้ system call

2. การเข้าถึงเครื่องมือที่รับรองแหล่งของ kernel โดยตรงด้วยระบบปฏิบัติการ ในส่วนนี้โค้ดและโครงสร้างข้อมูลสำหรับแหล่งข้อมูลในพื้นที่ของ kernel การเรียกฟังก์ชันใน API แหล่งข้อมูลแบบธรรมดาจะส่งผลใน system call ถึง kernel

Thread Library 3 อย่างหลัก ๆ ที่ใช้ในปัจจุบันคือ

1. POSIX Pthreads

2. Win32

3. Java

Pthreads เป็นเส้นงานมาตราฐานของ POSIX จะจัดการเกี่ยวกับระดับของผู้ใช้ หรือ ระดับของ kernel Threads Library ของ Win32 เป็นแหล่งข้อมูลของ kernel level ที่สะดวกของระบบวินโดวส์ เส้นงานของจาวาเป็น API ที่ยอมรับในการสร้างและจัดการเส้นงานโดยตรงในโปรแกรมจาวา อย่างไรก็ตาม เพราะว่ากรณีส่วนใหญ่แล้ว JVM จะเป็นตัวทำงานทางด้านบนสุดของระบบปฏิบัติการ เส้นงานของจาวา API นั้นเป็นรูปแบบเครื่องมือที่ใช้ threads library ได้สะดวกบนระบบโฮส ซึ่งคล้ายกับระบบปฏิบัติการ

วินโดวส์ เส้นงานของจาวาเป็นรูปแบบเครื่องมือที่ใช้ในระบบ Win32 API, UNIX และระบบปฏิบัติการ Linux เช่นเดียวกับการใช้ Pthreads

ในส่วนที่เหลือของเรื่องส่วนนี้เราจะอธิบายถึงการสร้างเส้นงานพื้นฐาน ซึ่งใช้ 3 thread library และมีภาพประกอบตัวอย่าง เราออกแบบโปรแกรมของรูปแบบหลายเส้นงานเมื่อทำการหาผลรวมของค่าที่เป็นจำนวนเต็มบวกในเส้นงานที่ใช้ซึ่งรู้จักกันในรูปฟังก์ชันการหาผลรวม

$$sum = \sum_{i=0}^{N} \mathrm{i}$$

จากตัวอย่าง ถ้า N มีค่าเท่ากับ 5 ในฟังก์ชันนี้ค่าผลรวม จาก 0 ถึง 5 คือ 15 โปรแกรมนี้จะทำงานด้วยค่าขอบบนของการหาผลรวมขนเส้นเส้นมาตราฐาน ถ้าผู้ใช้ใส่ค่า 8 ผลรวมของจำนวนค่าจาก 0 ถึง 8 จะออกมาเป็นอย่างไร

2.6.3.1 Pthreads

Pthreads เป็นตัวพื้นฐานของ POSIX (IEEE 103.1C) เรียกได้ว่าเป็น API สำหรับการสร้างเส้นงานและสิ่งที่เกิดขึ้นในเวลาเดียวกัน เป็นตัวบ่งบอกถึงพฤติกรรมของเส้นงานโดยไม่ใช้เครื่องมือ การออกแบบระบบปฏิบัติการมักจะใช้เครื่องมือในงานที่ต้องการ ระบบปฏิบัติการส่วนใหญ่ใช้ Pthreads เป็นเครื่องมือ ไม่ว่าจะเป็น Solaris, Linux, Mac OS X และ Tru64 UNIX Shareware เป็นเครื่องมือที่สะดวกในการใช้งานทั่วไปสำหรับงานที่หลากหลายซึ่งดีกับ ระบบปฏิบัติการวินโดวส์

โปรแกรมภาษา C ที่แสดงในรูป 4.6 เป็นตัวสาธิตแบบพื้นฐานของ Pthreads API สำหรับสร้างโปรแกรมรูปแบบหลายเส้นงาน ที่เราทำการคำนวณหาผลรวมของจำนวนเต็มบวกในการแบ่งเส้นงาน ในโปรแกรม Pthreads แบ่งเส้นงานและเริ่มประมวลผล โดยแบ่งออกเป็นฟังก์ชัน ในรูป 4.6 ส่วนของฟังก์ชัน runner() เมื่อโปรแกรมเริ่มทำงาน เส้นงานเส้นหนึ่งจะควบคุมการทำงานของ main() หลังจากนั้น main() จะสร้างเส้นงานที่สองและเริ่มควบคุมการทำงานของฟังก์ชัน runner()

เส้นงานสองเส้นสามารถใช้ข้อมูลร่วมกันได้ทั้งหมด เมื่อเรามองลึกลงไปในโปรแกรมจะเห็นว่าทุกโปรแกรม Pthreads นั้นจะถูกรวบรวมไว้ในส่วนของเฮดเดอร์ไฟล์คือ Pthread.h ส่วนของ pthreads จะ

รองรับเกี่ยวกับการจำแนกของการสร้างเส้นงาน แต่ละเส้นงานสามารถที่จะกำหนดขนาดของ attribute และตารางการใช้ข้อมูล

Pthread_attr_t จะเป็นตัวแทน attribute ของเส้นงาน เราจะกำหนดค่า attribute ในส่วนของฟังก์ชันเรียก pthread_atr_int(&attr) เพราะเราไม่สามารถกำหนดค่าของ attribute ได้แน่นอน เราสามารถใช้ค่า attribute พื้นฐานได้ (ในบทที่ 5 เราจะอธิบายการทำงานของ attribute จาก Pthreads API)

การแบ่งเส้นงานถือเป็นการสร้างเส้นงานด้วย pthreads_create() ของฟังก์ชันเรียก ในการเพิ่มและผ่านของเส้นงาน สามารถจำแนกเส้นงานได้ อีกทั้งชื่อของฟังก์ชันเมื่อมีเส้นงานใหม่เริ่มประมวลผล ในส่วนนี้ฟังก์ชัน runner() สุดท้ายเราจะส่งค่าจำนวนเต็มคงที่ไปยังเส้นงานที่ควบคุมมัน

```
#include <pthread.h>
#include <stdio.h>

int sum; /* this data is shared by the thread(s) */
void *runner(void *param); /* the thread */

int main(int argc, char *argv[])
{
   pthread_t tid; /* the thread identifier */
   pthread_attr_t attr; /* set of thread attributes */

   if (argc != 2) {
      fprintf(stderr,"usage: a.out <integer value>\n");
      return -1;
   }
   if (atoi(argv[1]) < 0) {
      fprintf(stderr,"%d must be >= 0\n",atoi(argv[1]))
      return -1;
   }

   /* get the default attributes */
   pthread_attr_init(&attr);
   /* create the thread */
   pthread_create(&tid,&attr,runner,argv[1]);
   /* wait for the thread to exit */
   pthread_join(tid,NULL);

   printf("sum = %d\n",sum);
}

/* The thread will begin control in this function */
void *runner(void *param)
{
   int i, upper = atoi(param);
   sum = 0;

   for (i = 1; i <= upper; i++)
      sum += i;

   pthread_exit(0);
}
```

รูปภาพที่ 2.26 Multithreaded C program using the Pthreads API

จุดนี้โปรแกรมมี 2 เส้นงานคือ เส้นงานพ่ออยู่ใน main() และ เส้นงานลูก ทำการรวมการทำงานในฟังก์ชัน runner() หลังจากการสร้างเส้นงานลูก เส้นงานพ่อจะรอคำสั่งสิ้นสุดการทำงานจากฟังก์ชัน pthread_join() เส้นงานลูกจะสิ้นสุดการทำงานเมื่อมันเรียกฟังก์ชัน pthread_exit() เส้นงานลูกสามารถที่จะกลับมาทำงานได้อีกครั้งเมื่อเส้นงานพ่อส่งข้อมูลไปยังส่วนของข้อมูลที่ใช้ร่วมกัน (shared data)

2.6.3.2 เส้นงานของระบบ Win32

วิธีการสำหรับสร้างเส้นงานที่ใช้ในระบบ Win32 วีธีการจะคล้ายๆ Pthreads ในระบบอื่น ๆ เราจะรวมไปถึงเส้นงานระบบ Win32 ในโปแกรมภาษา C ที่แสดงดังรูป 4.7 จะเห็นได้ว่า เราจะใช้ Windows.h เป็นหัวของไฟล์ เมื่อใช้รูปแบบ win32 เป็นรูปแบบในการนำเสนอ

Pthreads ที่แสดงในรูป 4.6 ข้อมูลถูกแบ่งโดยเส้นงานที่ทำการแบ่ง ในส่วนนี้ ผลรวมทั้งหมดจะรองรับ (ข้อมูล DWORD) รูปแบบเป็นจำนวนเต็มบวกขนาด 32 บิต เราจะอธิบายฟังก์ชัน summation() นี้เป็นการทำงานในเส้นงานแบ่ง ฟังก์ชันนี้เป็นการส่ง pointer ไปยังค่าว่าง ระบบ Win32 ถูกกำหนดโดย LPVOID การทำงานของเส้นงานนี้ฟังก์ชันที่กำหนดข้อมูลทั้งหมด จากผลรวมไปถึงค่าของผลรวม จาก 0 ถึงค่าดงที่ที่ผ่านไปถึงฟังก์ชัน summation()

เส้นงานที่สร้างขึ้นในระบบ Win32 จะใช้ฟังก์ชัน CreateThread() และใน Pthreads จะกำหนดขนาด attribute สำหรับเส้นงานที่แบ่งตัวส่งค่าไปยังฟังก์ชัน attribute ที่รวบรวมข้อมูลที่ปลอดภัย ขนาดของข้อมูลนั้นจะเพิ่มหรือลดสามารถที่จะกำหนดได้ ถ้าเส้นงานอยู่ในสถานะรอการทำงาน ในโปรแกรมนี้เราจะใช้ค่าพื้นฐาน สำหรับ attribute นี้ (เราจะกำหนดค่าของเส้นงาน ในสถานะรอแทนที่เราจะทำมันในตอนที่ CPU ทำงาน) ช่วงที่เส้นงานลูกสร้าง เส้นงานพ่อจะรอมันทำงานจนสิ้นสุดการทำงาน ก่อนที่จะแสดงค่าผลลัพธ์ออกมา ซึ่งค่านั้นจะถูกกำหนดโดยเส้นงานลูก จะถูกยกเลิกโดยโปรแกรม Pthreads (รูป 4.6) เส้นงานพ่อจะรอเส้นงานลูกใช้ pthread_join() ซึ่งการทำงานคล้ายกับระบบ Win32 ที่ใช้ฟังก์ชัน WaitForSingleObject() เพราะการสร้างเส้นงานจะไปปิดส่วนของการสิ้นสุดการทำงานของเส้นงานลูก (เราจะป้องกัน การพ้องกันของวัตถุ ซึ่งเราจะอธิบายเพิ่มในบทที่ 6)

2.6.3.3 เส้นงานของระบบ Java

เส้นงานเป็นรูปแบบแรกของการประมวลผลโปรแกรมในโปรแกรมจาวา และ ภาษาจาวา และมันเป็นข้อกำหนดลักษณะของกลุ่มสมาชิกขนาดใหญ่ในการสร้างและจัดการเกี่ยวกับเส้นงาน ทุกโปรแกรมจาวาประกอบไปด้วยเส้นงานเดี่ยวขนาดเล็กทำการควบคุมอยู่ ตัวอย่างคือ โปรแกรมจาวาที่มีเมธอด main() เพียงอย่างเดียวที่ทำงานแบบเส้นงานเดี่ยวของ JVM

เรามี 2 วิธีในการสร้างเส้นงานในโปรแกรมจาวา วิธีที่หนึ่ง คือ การสร้างคลาสใหม่ที่มาจากเส้นงานคลาสเส้นงานและผ่านเมธอด run() อีกทางเลือกหนึ่ง สามารถควบคุมได้หลายผู้ใช้ซึ่งเป็นวิธีการกำหนดขอบเขตของคลาสที่เป็นตัวทำงานอยู่ ซึ่งตัวที่ทำงานนั้นเราจะใช้เป็นตัวที่เราคิดตาม เมื่อคลาสทำงานมันจะเป็นตัวกำหนดเมธอด run() โค้ดที่ใช้ในเมธอด run() เป็นอะไรที่ทำการแบ่งเส้นงาน

```
#include <pthread.h>
#include <stdio.h>

int sum; /* this data is shared by the thread(s) */
void *runner(void *param); /* the thread */

int main(int argc, char *argv[])
{
  pthread_t tid; /* the thread identifier */
  pthread_attr_t attr; /* set of thread attributes */

  if (argc != 2) {
    fprintf(stderr,"usage: a.out <integer value>\n");
    return -1;
  }
  if (atoi(argv[1]) < 0) {
    fprintf(stderr,"%d must be >= 0\n",atoi(argv[1]));
#include <windows.h>
#include <stdio.h>
DWORD Sum; /* data is shared by the thread(s) */
/* the thread runs in this separate function */

DWORD WINAPI Summation(LPVOID Param)
{
  DWORD Upper = *(DWORD*)Param;
  for (DWORD i = 0; i <= Upper; i++)
    Sum += i;
  return 0;
}

int main(int argc, char *argv[])
{
  DWORD ThreadId;
  HANDLE ThreadHandle;
  int Param;
  /* perform some basic error checking */
  if (argc != 2) {
    fprintf(stderr,"An integer parameter is required\n");
    return -1;
  }
  Param = atoi(argv[1]);
  if (Param < 0) {
    fprintf(stderr,"An integer >= 0 is required\n");
    return -1;
  }
  // create the thread
  ThreadHandle = CreateThread(
    NULL, // default security attributes
    0, // default stack size
    Summation, // thread function
    &Param, // parameter to thread function
    0, // default creation flags
    &ThreadId); // returns the thread identifier

  if (ThreadHandle != NULL) {
    // now wait for the thread to finish
    WaitForSingleObject(ThreadHandle,INFINITE);

    // close the thread handle
    CloseHandle(ThreadHandle);

    printf("sum = %d\n",Sum);
  }
}
```

รูปภาพที่ 2.27 Multithreaded C program using the Win32 API

คลาสผลรวมที่ใช้แสดงส่วนที่ทำงานได้ การสร้างเส้นงานเป็นการทำงานโดยส่วนของการสร้างวัตถุตัวอย่าง ของคลาสเส้นงานและผ่านการสร้างวัตถุที่สามารถทำงานได้

การสร้างเส้นงานที่ไม่เจาะจงการสร้างเส้นงานใหม่ ในทางตรงกันข้ามมันจะเป็นเมธอด start() ที่ทำการสร้างเส้นงานใหม่อย่างแท้จริง การเรียกเมธอด start() เพื่อการทำวัตถุใหม่จะต้องใช้ 2 สิ่งคือ

1). การจัดการหน่วยความจำและการเริ่มใช้เส้นงาน JVM

2). เรียกเมธอด run() ให้เส้นงานทำงานที่เหมาะสมด้วย JVM (หมายเหตุ เราไม่เคยเรียกใช้เมธอด run() โดยตรงหรือในทางตรงกันข้ามเราเรียกเมธอด start() และมันจะเป็นตัวเรียก เมธอด run() ต่อไป)

เมื่อโปรแกรมทำงานเสร็จ 2 เส้นงานที่ถูกสร้างโดย JVM อย่างที่หนึ่ง เส้นงานพ่อเริ่มการทำงานในเมธอด main() อย่างที่สอง เส้นงานจะถูกสร้างเมื่อ เมธอด start() ในวัตถุเส้นงานเป็นตัวเรียก เส้นงานลูกนี้จะเริ่มทำการประมวลผลในเมธอด run() ของคลาส Summation หลังจากส่งค่าผลลัพธ์ เส้นงานนี้จะถูกกำจัดเมื่อมันออกจากเมธอด run()

การแบ่งปันข้อมูลระหว่างเส้นงานจะเกิดขึ้นง่ายในระบบ WIn32 และ Pthreads ข้อมูลที่ถูกแบ่งปันนั้นจะถูกเปิดเผยทั่วไปอย่างแท้จริงซึ่งเป็นภาษาแนวคิดเชิงวัตถุ จาวาไม่มีแนวคิดของข้อมูลแบบนี้ ถ้าเส้นงานอย่างน้อยสองเส้นงานทำการแบ่งปันข้อมูลในโปรแกรมจาวา การแบ่งปันข้อมูลเกิดขึ้นจากการส่งข้อมูลจากแหล่งที่แบ่งปันข้อมูลถึงเส้นงานที่เหมาะสม ในโปรแกรมจาวาที่แสดงในรูป 4.8 เส้นงานหลักและเส้นงานรองทำการแบ่งปันข้อมูลกันในคลาส sum วัตถุที่ถูกแบ่งถูกอ้างอิงมาจากเมธอด getSum() และ setSum() (คุณจะสงสัยว่าทำไมเราไม่ใช้วัตถุที่เป็นจำนวนเต็ม แทนที่เราจะใช้การออกแบบคลาสใหม่ มีสาเหตุมาจากคลาสของจำนวนเต็มเปลี่ยนแปลงไม่ได้ นั่นคือ ค่าของข้อมูลในเซตมันไม่สามารถเปลี่ยนแปลงได้)

การเรียกเส้นงานพ่อใน Pthreads และ Win32 ใช้ pthread_join() และ WaitForSingle

Object() (ตามลำดับ) รอเส้นงานลูกเสร็จสิ้นการทำงาน เมธอด join() ในจาวาทำงานคลายกัน (ข้อสังเกต join() ในจาวาไม่ใช้ InterruptedException ซึ่งเราเลือกที่จะไม่ใช้มัน)

```
class Sum
{
   private int sum;

   public int getSum() {
     return sum;
   }

   public void setSum(int sum) {
     this.sum = sum;
   }
}

class Summation implements Runnable
{
   private int upper;
   private Sum sumValue;

   public Summation(int upper, Sum sumValue) {
     this.upper = upper;
     this.sumValue = sumValue;
   }

   public void run() {
     int sum = 0;
     for (int i = 0; i <= upper; i++)
        sum += i;
     sumValue.setSum(sum);
   }
}
public class Driver
{
   public static void main(String[] args) {
     if (args.length > 0) {
       if (Integer.parseInt(args[0]) < 0)
         System.err.println(args[0] + " must be >= 0.");
       else {
         // create the object to be shared
         Sum sumObject = new Sum();
         int upper = Integer.parseInt(args[0]);
         Thread thrd = new Thread(new Summation(upper, sumObject));
         thrd.start();
         try {
            thrd.join();
            System.out.println
                    ("The sum of "+upper+" is "+sumObject.getSum());
       } catch (InterruptedException ie) { }
       }
     }
     else
       System.err.println("Usage: Summation <integer value>"); }
}
```

รูปภาพที่ 2.28 Java program for the summation of a non-negative integer

2.6.4 Threading Issues

ในส่วนนี้เราจะมาพูดถึง บางส่วน บางเนื้อหา ในเรื่องเกี่ยวกับ multithreaded program

```
class Sum

{

        private int sum;



        public int getSum()

        {

                return sum;

        }

        public void setSum (int sum)

        {

                this.sum = sum;

        }

}

class Summation implements Runnable

{

        private int upper;

        private Sum sumValue;
```

```
    public Summation (int upper , Sum sumValue)
    {
        this.upper = upper;
        this.sumValue = sumValue;
    }
    public void run ()
    {
        int sum = 0;

        for (int i = 0;i <= upper ; i++)
            sum += i;

        sumValue.setSum(sum);
    }
}

public class Driver
```

```
{
    public static void main (String[] args)
    {
        if (args.length > 0)
        {
            if (Integer.parseInt(args[0]) > 0)
                System.err.println(args[0] + " must be >= 0. ");
            else
            {
                //create the object to be shared
                Sum sumObject = new Sum();
                int upper = Integer.paseInt(args[0]);
                Thread thrd = new Thread(new Summation(upper,
                                        sumObject));
                thrd.start();
                try
                {
                    thrd.join();
```

```
                    System.out.println("The sum of " + upper + " is
                                        " + sumObject.getSum());
            }catch(InterruptedException ie){}
                }
            }

            else
                System.err.println("Usage : Summation < Integer value >");
        }
}
```

ตัวอย่างโปรแกรมภาษา Java ในการหาค่าผลรวมของ จำนวน non-negative

2.6.4.1 The fork() and exec() System call

ใน บทที่ 3 เราได้อธิบาย ถึงหลักการทำงานของการเรียกใช้ fork ไปแล้ว ในการช่วย แยก และสำเนา process โดดถ้า มี Thread หนึ่งใน program เรียก fork ก็จะทำกับprocess ใหม่ๆ ซ้ำๆ ทุก Thread หรือทำแค่ครั้งเดียว

ระบบปฏิบัติการ UNIX บางตัวมีให้เลือก fork ได้ ทั้ง 2 รูปแบบ แบบที่หนึ่งคือการทำซ้ำ ๆ ทุก Thread อีกแบบคือการเลือกทำเฉพาะ Thread ที่ มีการเรียกใช้ fork จากระบบ exec system call ทำงานในทางแบบเดียวกับที่อธิบายไว้ใน บทที่ 3 และ ถ้ามี Thread เรียกการ exec จากระบบ โปรแกรมจะมีการกำหนด parameter ไว้ สำหรับ exec และจะจัดเก็บไว้ใน process ทุกครั้งที่มีการเรียกใช้

ทางเลือก ของ 2 รูปแบบในการ fork ขึ้นอยู่กับ application ถ้าที่ exec เรียกนั้นมีขนาดใหญ่ หลังจากที่ทำการ fork แล้วจะหลีกเลี่ยงในการทำซ้ำทุก Thread และ program จะมีการกำหนด parameter ไว้ exec เพื่อจัดเก็บ process ยกตัวอย่างการทำซ้ำอย่างเดียวในการเรียก Thread สมควรเป็น โดย ถ้า process คน ละ process กัน ไม่ได้เรียก exec หลังจากการ fork แล้ว process แต่ละ process น่าจะมีการทำสำเนาทุก ๆ Thread ไว้

2.6.4.2 Cancellation

Thread cancellation เป็นขั้นตอนสุดท้ายของ Thread ก่อนจะทำงานเสร็จสิ้น ยกตัวอย่าง ในการทำงานหลาย ๆ Thread พร้อม ๆ กัน การค้นหาผ่าน Database และ Thread หนึ่งมีการส่งค่าผลที่ได้กลับมา Thread อื่น ๆ ที่เหลือก็จะยกเลิกการทำงานต่อทันที ในรูปแบบอื่นที่เกิดขึ้นก็เมื่อผู้ใช้กดปุ่ม Stop บน Web page เมื่อมีการ loading ที่ต่างไปจากที่ต้องการ ในหลาย ๆ ครั้งจะเห็นได้ว่าเมื่อมีการใช้งานใน web browser จะมีการแยกกัน โหลดรูปภาพ ในหน้า นั้น ๆ แต่เมื่อใดถ้าผู้ใช้กดปุ่ม Stop บน web browser ทุก Thread ที่ทำงานอยู่รูปในการโหลดรูปในหน้านั้นก็จะยกเลิกการทำงานทันที

บ่อยครั้งที่การยกเลิก Thread เกิดกับการที่เราเลือกที่จะยกเลิกเฉพาะ Thread เป้าหมายการยกเลิก Thread เป้าหมายจะเกิดขึ้นใน 2 กรณีดังนี้

1. Asynchronous cancellation เป็นการเลือกจบ Thread ที่มีขนาดใหญ่โตมาก
2. Deferred cancellation เช็คดูว่า Thread นั้นหมดระยะเวลาเรียกหรือยัง หรือ ใกล้ที่จะหมดเวลาก็ให้ดูว่าควรจะหยุดการทำงาน ถ้างานที่ทำอยู่เป็นงานเก่า

ความยากของการ cancellation อยู่ที่ตำแหน่งที่เราจะทำงานยกเลิก Thread รวมไปถึงการที่เราจะยกเลิก Thread ไป แล้วถ้าเราต้องการที่จะกลับไปทำงานตรงส่วนนั้นอีกครั้งจะทำได้อย่างไร บ่อยครั้งที่ระบบต้องการที่จะกลับไปทำงานส่วนที่ยกเลิกไปแต่ไม่สามารถกลับไปใช้งานได้ดังเดิม ดังนั้นการยกเลิกการทำงานของ Thread ที่ไม่สอดคล้องกันจึงไม่เหมาะกับการทำงานในระบบที่มีขนาดใหญ่ เมื่อเรามองตามระบบ cancellation ในข้อแตกต่างระหว่าง thread ที่ถูกยกเลิกชี้ให้เห็นว่าการที่เราเลือกที่จะยกเลิก Thread นั้น ๆ จำเป็นต้องมีตัวชี้เอาไว้ เพื่อที่เราจะสามารถกลับไปทำงานเดิมได้อีกแม้มีการยกเลิกไปแล้ว และเพื่อความปลอดภัยกันผิดพลาด เราเรียกว่า cancellation point

2.6.4.3 Signal Handling

signal ใช้ในระบบปฏิบัติการ UNIX เพื่อบอกถึงการทำงานของ process ที่กำลังทำงานอยู่ โดยเฉพาะ signal จะมีความสอดคล้องหรือไม่นั้นขึ้นอยู่กับ เหตุการณ์ที่กำลังทำงานอยู่ ทุก signal ไม่ว่าจะเป็นแบบสอดคล้องหรือไม่สอดคล้อง มีรูปแบบดังต่อไปนี้

1. signal ที่แบ่งออกตามความจำเพาะ ของเหตุการณ์ นั้นๆ

2. signal ที่แบ่งออกตาม process ที่ทำงานอยู่

3. แบ่งตามประเภท ผู้ใช้งาน

ทุก ๆ signal จะต้องมีตัวควบคุม และตัวควบคุมแบ่งออกได้ดังนี้

- ตัวควบคุมอัตโนมัติ
- ตัวควบคุมที่ผู้ใช้กำหนด

ทุก signal จะมีตัวควบคุมอัตโนมัติ เมื่อมีการทำงานบน kernel เมื่อมีการเรียกใช้งาน และการกระทำพื้นฐานของ signal จะถูกเปลี่ยน เมื่อมี user defined signal handler เข้ามาควบคุมแทน signal จะถูกเปลี่ยนเส้นทาง บางครั้ง signal ก็จะถูกปล่อยไว้เฉย ๆ จนมีการยกเลิกโปรแกรม สำหรับการใช้งานแค่ Thread เดียวก็ไม่มีปัญหาในโปรแกรม แต่ถ้าเรานำมาใช้ใน multithread program แล้วเราควรจะใช้งาน signal อย่างไร เมื่อไหร่ ตอนไหนที่ ควรจะส่ง signal

ต่อไปนี้เป็นการแบ่งแบบทั่วๆ ไปในกรณีที่ส่ง signal

- ส่งให้เฉพาะ Thread ที่มีการเรียกขอ
- ส่งทุกๆ ทุก ๆ Thread ใน process
- ส่งเฉพาะ Thread หนึ่งใน process
- มอบให้เฉพาะ Thread ที่ได้รับ signal สำหรับ process

Method ที่ใช้สำหรับส่ง signal นั้นขึ้นอยู่กับประเภท ของ signal ด้วย ยกตัวอย่าง signal ที่ไม่สอดคล้อง ต้องการที่จะส่งไปบอกสาเหตุที่ Thread อื่น ๆ ไม่ทำงานใน process ไม่ว่าอย่างไรก็ตาม signal ที่ส่งไปนี้ก็ไม่สามารถบอกได้ชัดเจน แต่บาง signal สามารถส่งไปทุก ๆ Thread ได้ส่วนมาก multithread ใน UNIX จะขึ้นอยู่กับการตอบรับของ signal อาจจะมีการปิดกั้นในบางกรณี signal ที่ไม่มีความสอดคล้อง สามารถส่งไปถึงเฉพาะ Thread ที่ไม่ถูกปิดกั้นเท่านั้น อย่างไรก็ตามเพราะ signal จำเป็นต่อการควบคุม พื้นฐาน UNIX จึงมีคำสั่งบางตัวที่สามารถทำให้ควบคุม signal ได้ ด้วยการ kill(aid_t aid , int signal) ใน Window ไม่มีระบบรองรับ signal เท่าไหร่ จึงมีการ ใช้ asynchronous procedure calls(ADC)ในการควบคุม Thread ต่างๆดูเหตุการณ์ต่าง ๆ ที่เกิดขึ้น คล้าย ๆ กับการควบคุมการทำงานของ signal ใน UNIX อย่างไรก็ตาม ADC ใน Window ค่อนข้างมีข้อจำกัดในการใช้งาน

2.6.4.4 Thread Pool

ในหัวข้อ 4.1 ที่เรายกตัวอย่างเกี่ยวกับ web sever ที่ได้รับการติดต่อขอร้องมา ก็จะมีการสร้าง Thread ต่างๆ เพื่อที่จะไปแก้ไขปัญหาที่มีเข้ามา ในทางตรงข้าม การสร้าง Thread ต่าง ๆ นั้นยากกว่าการ แตก process อย่างแน่นอน และปัญหาที่เกิดขึ้นกับการทำ multithread นั้นก็ไม่ได้มีน้อยเช่นกัน

ข้อ แรก คือปัญหาการที่เราต้องเลือกตัดสินใจ เกี่ยวกับช่วงเวลา กับความสำคัญของแต่ละ Thread ในการที่เราจะนำไปจัดคิว ไว้ทำงานให้เสร็จเป็นงาน ๆ ไป อย่างที่สอง การกังวลกับปัญหาบางอย่าง ถ้าเรายอมรับข้อเรียกร้องทุก ๆ ข้อที่เรียกร้องมาก็ จะทำให้เราไม่มีข้อบ่งชี้ถึงลำดับ การกระทำที่จะเข้ามาทำงานร่วมกันในระบบ ทรัพยากรเครื่อง ไม่ว่าจะเป็น CPU เวลา หรือ แม้กระทั่งหน่วยความจำ จะถูกใช้หมดไปกับ Thread งานที่มีจำนวนมหาศาล ดั้งนั้นจึงมีกระบวนการหนึ่งถูกนำขึ้นมาใช้เรียกว่า Thread Pool

แนวคิด thread pool คือการนำตัวเลขมาไว้ที่ thread ในแต่ละ process ที่จะทำงานและ จัดหาพื้นที่ที่มาสำรองไว้ คือ pool และให้แต่ละ thread รอการเรียกใช้งานตามตัวเลขที่ได้รับ เมื่อsever มีการตอบรับจากการร้องขอ มันก็จะออกมาจาก pool มาทำหน้าที่ให้เสร็จแล้วตอบกลับเมื่อทำงานเสร็จแล้วจึงกลับไปที่ pool เหมือนเดิมรอการเรียกครั้งต่อไป แต่ถ้า pool เต็ม ไม่สามารถรับ thread ได้แล้ว ระบบก็ต้องรอจนกว่าจะมีที่ว่าง จึงสามารถทำงานอย่างอื่นได้

ประโยชน์ ของแนวคิดนี้

1.ตอบสนองกับสิ่งที่เรียกขอเร็วกว่าเดิม

2.จำกัดการใช้ทรัพยากรในเครื่องได้

ตัวเลขของ thread ใน pool สามารถช่วยแก้ปัญหาการจัดลำดับในการเรียกใช้ได้เป็นอย่างดี เพราะได้คำนึงถึง ปัจจัยหลาย ๆ ด้าน ทั้งด้าน CPU พื้นที่ใน memory และ กลุ่มที่มีความต้องการในด้านเดียวกัน

ใน Win32 API มีการจัดหา function ที่เกี่ยวข้องกับ thread pool การใช้ thread pool ใน API การสร้าง thread นั้น มี function ในการสร้างคือ Thread Create() มีการพูดถึงแล้วใน 4.3.2 ตัวรูปแบบ function ประกอบด้วยดังนี้

```
DWORD  WINAPI   PoolFunction ( AVOID Parameter)
{
        /**
        * this function runs as a separate thread.
        **/
}
```

ตัวชี้ใน PoolFunction() จะส่งค่าหนึ่งไปยัง thread pool API แล้วจึงดำเนินการต่อไป โดยมีตัว QueueUserWorkItem() เป็น function ที่อยู่ใน pool API ไว้ส่งค่า parameter กลับมี 3 ตัว ดังนี้

- LPTHREAD_START_ROURINE เป็นตัวชี้ไปยัง function
- PVOID เป็นค่า parameter ที่ต้องการส่งไปดำเนินการ
- ULONG เป็นตัวบ่งชี้ว่าจะดำเนินการแบบไหน

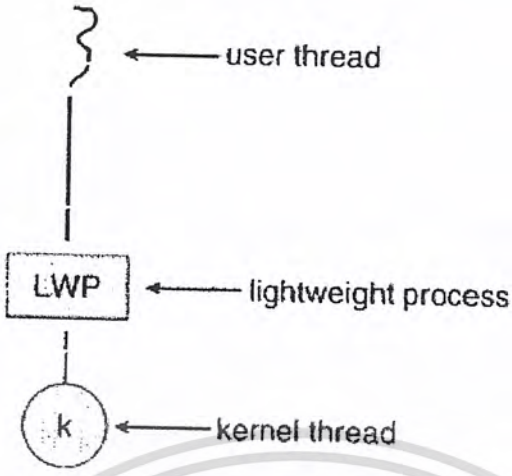
ตัวอย่างการส่งค่า

```
QueueUserWorkItem(&PoolFunction, NULL, 0);
```

การสร้าง thread จาก programmer จากการเรียกใช้ function PoolFunction() ดังตัวอย่างนี้ ไม่มีการส่งค่า parameter ใด ๆ ไป และ Flag เป็น 0 คือ ไม่ได้สั่งให้ทำคำสั่งพิเศษ ใด ๆ เราได้สร้าง thread ที่ไม่ได้ใส่คำสั่งพิเศษใด ๆ ลงไปขึ้นมาสิ่งอื่น ๆ ที่ได้มีการจัดเก็บไว้ใน Win32 thread pool API ก็คือส่วนที่ช่วยเหลือต่าง ๆ ที่มีประโยชน์ ในต่างช่วงเวลาของการเข้าถึง Thread ของ I/O ใน java.ultil.concurrent package ใน Java 1.5 ก็ได้จัดหาสิ่งที่มีประโยชน์และช่วยเหลือ ที่เกี่ยวกับ thread pool ไว้ให้ด้วย

2.6.4.5 Thread -Specific Data

thread เป็นส่วนหนึ่งของ process ที่ใช้ข้อมูลร่วมกับ process จริง ๆ แล้วการที่นำข้อมูลมาใช้ร่วมกันทำให้เกิดประโยชน์ ต่อการ multithread programming ด้วย อย่างไรก็ตาม ในบางสถานการณ์ thread ก็สมควรที่จะมี ข้อมูลเป็นของตัวเองโดดชัดเจน เราจะเรียกข้อมูลส่วนนี้ว่า thread-specific data ตัวอย่าง ในการดำเนินการระหว่าง process ในระบบ เราควรที่จะมีการรองรับการดำเนินการในแต่ละ thread นอกจากนี้แต่ละการดำเนินการน่าจะมีสิ่งที่บ่งบอกเป็นลักษณะ เฉพาะการเชื่อโยงแต่ละ thread ที่มีความแตกต่างกันนี้ เราจะใช้ thread-specific data



รูปภาพที่ 2.29 Lightweight process (LWP)

2.6.4.6 Scheduler Activations

สุดท้ายนี้เราจะพิจารณาถึงการเชื่อมต่อระหว่าง kernel และ thread library ใน multithread programs อย่างไหนจะเป็นที่ต้องการ ระหว่าง many-to-many และ two level model จึงมีการปรับปรุงและเพิ่มเติมให้เป็นที่ยอมรับและกลายเป็นรูปแบบที่ดีที่สุด

หลายระบบที่มีการจัดเตรียมรองรับไว้ เฉพาะ แค่แบบใดแบบหนึ่งเท่านั้น ระหว่าง many-to-many และ two level model ถ้าใช้โครงสร้างที่เป็นกลางระหว่าง ผู้ใช้งาน และ kernel thread จะต้องมี lightweight process หรือ LWP ดังรูปที่แสดงด้านบน (figure 4.9)เป็นตัวทำหน้าที่ประสานงานส่วนกลาง โดยจะสามารถสร้างตาราง Schedule เพื่อเป็นตารางในการควบคุมการทำงานของ thread ของ user ได้ แต่ละ LWP จะเชื่อมต่อกับ kernel thread และ kernel thread จะเป็นตัวประมวลผลงานแต่ละงานที่ผ่านเข้ามาถ้า มีการปิดกั้นเกิดขึ้นในชั้นนี้ ระหว่างผู้ใช้กับ LWP ก็จะถูกปิดไปด้วย

แต่ละ application ต้องการลำดับของ LWP เพื่อที่จะช่วยให้สามารถทำงานได้อย่างมีประสิทธิภาพ ปกติแล้ว application ต่างๆสามารถทำงานได้ถ้ามี LWP เพียงพอ ส่วน application ที่เกี่ยวกับ I/O ที่มีความละเอียดสูง ต้องใช้ LWP หลากหลายตัวในการที่จะให้สามารถทำงานได้

รูปแบบหนึ่งในการเชื่อมต่อระหว่าง user thread library และ kernel เรียกว่า scheduler activation มีการทำงานในรูปแบบที่ kernel จะจัด application กับผู้ดำเนินการ (LWPs)และ application สามารถเลือกตาราง user thread ที่เหมาะแก่การประมวลผล นอกจากนี้ kernel จะต้องแจ้งให้ application ทราบถึงงานที่

ได้รับ ขั้นตอนนี้เรียกว่า up-call จะมีการจัดการขั้นตอนนี้โดยมี thread library และ up-call handler จะทำงานในระบบประมวลผลเสมือนและจะมีตัวที่ทำหน้าที่ส่งสัญญานบอกเมื่อ application ถูก block อีกรูปแบบหนึ่ง kernel จะทำการ up-call ไปยัง application ให้ทราบว่า thread ถูก block และสามารถระบุ thread ได้ชัดเจน kernel จะจัดหาตัวประมวลผลใหม่ ให้ application โดยก่อนที่จะมาทำงานที่หน่วยประมวลผลใหม่ application จะต้องจำสถานะ ในที่เดิมไว้ แล้วทำการยกเลิกการทำงานที่เดิม แล้วมาทำงานในหน่วยประมวลผลใหม่แทนการ up call handle จะมีการเลือก application ที่มีความเหมาะสมที่จะขึ้นมาทำงานแทน application เดิมที่ถูก block เมื่อ อยู่ในสถานะ ถูก block จะต้องรอจนกว่า kernelจะเรียกใช้งานจึงสามารถกลับไปทำงานได้

2.6.5 Operating-System Example

ในส่วนนี้ เราจะดูว่า thread ใน windows XP และ Linux นั้นจะมีการทำงานอย่างไร

2.6.5.1 Windows XP Thread

Windows XP ใช้ตัวimplement เป็น Win32 Application Programming Interface เป็น Library เป็น API หลักของตระกูลระบบปฏิบัติการ Microsoft จริง ๆ และจะกล่าวถึงอะไรในการนำไปประยุกต์ใช้ในระบบปฏิบัติการ

โปรแกรม Windows XP จะทำงานโดยแยกโปรเซสและแต่ละ โปรเซสจะประกอบด้วย thread หนึ่ง thread หรือมากกว่านั้น Win32 api ที่ใช้ในการสร้าง thread จะอยู่ในหัวข้อที่ 4.3.2

Windows XP จะใช้ การอธิบายแบบ one to one หมายถึง Kernel thread 1 หน่วย กับ User thread 1 หน่วย ซึ่งระบบปฏิบัติการจะยอมให้ thread อื่นประมวลผลได้เป็นระบบขนาน ที่ทำงานแบบมัลติโปรเซสเซอร์ มีการใช้หลักการนี้อยู่ในระบบปฏิบัติการ Windows ในปัจจุบัน โดยโมเดลนี้ต้องไม่ยอมให้สร้าง user thread มากเกินไป จะอยู่ในหัวข้อที่4.2.2

องค์ประกอบทั่วไปของ Thread มีดังนี้

- Thread ID เป็นตัวบ่งบอกถึง Thread แต่ละตัว
- Register เป็นการเซทตัวแทนค่าของตัวคำสั่ง
- User stack จะถูกใช้เมื่อ thread ทำงานอยู่ใน user mode และ kernel stack จะถูกใช้
- เมื่อ thread ทำงานอยู่ใน kernel mode
- Private storage area
- register set, stacks และ Private storage area รวมเรียกว่า **context** of the threads

โครงสร้างข้อมูลหลักของ thread ประกอบด้วย:

- ETHREAD (executive thread block)
- KTHREAD (kernel thread block)
- TEB (thread environment block)

องค์ประกอบขอ ETHREAD ประกอบไปด้วย pointer ที่เป็นตัวชี้โปรเซสของthread และ หน้าที่ของ Address ซึ่ง ควบคุมโดย thread start control ETHREAD โดยส่วนใหญ่จะมี pointer ที่สอดคล้องกันคือ KTHREAD

KTHREAD ประกอบด้วย ตารางและข้อมูลที่สอดคล้องกันของ thread และยังมี kernel stack ที่จะทำงานใน mode kernel ตัวชี้ไปยัง TEBETHREAD และ KTHREAD ที่อยู่ใน kernel space นั้นหมายความว่ามันจะอยู่เฉพาะในkernel เท่านั้น

TEB คือโครงสร้างข้อมูลที่อยู่ใน user-space ที่ผ่านเข้ามาเมื่อ thread ทำงานอยู่ใน user mode ในขณะที่มันอยู่ในพื้นที่อื่นที่ไม่ใช่ user space มันจะประกอบด้วยตัวชี้ thread user mode stack และ array สำหรับ thread

2.6.5.2 Linux Threads

Linux มักจะเรียกว่า task มากกว่าเรียกว่า thread การสร้าง thread ใช้ clone() system call clone() ยอมให้ child task ใช้ address space ร่วมกันกับ parent task (process)

ตัวอย่างเช่น ถ้า cone() ผ่านแล้ว flagของCLONE_FSCLONE_VMCLONE_SIGHAND และ

CLONE_FILES task ของ parent and child จะแชร์ข้อมูลระบบไฟล์เดียวกัน ใช้หน่วยความจำ

เดียวกัน ใช้ตัวส่งสัญญาณเดียวกัน และติดตั้งการเปิดไฟล์เหมือนกัน การใช้Clone() ที่เป็นที่นิยมเท่ากับเป็นการสร้างThread (จะอธิบายในบทนี้)เมื่อ parent task แชร์ทรัพยากรส่วนใหญ่ให้กับ child task อย่างไรก็ตามถ้าไม่เซทรูปแบบของ Flag เมื่อทำการ clone()มันจะทำให้ไม่สามารถแชร์ได้ผลลัพธ์ในฟังก์ชั่นนี้คล้ายกับการเรียกใช้ fork()

ระดับของการแชร์มีด้วยกันหลายระดับซึ่งเป็นไปได้เพราะว่างานที่ทำแทนอยู่ใน Linux kernel โครงสร้างข้อมูล kernel พิเศษนี้มีอยู่ในแต่ละ task ในระบบ โครงสร้างข้อมูลนี้เป็นเหมือนการเก็บข้อมูลของ task เอาไว้ ซึ่งมี pointer เป็นตัวชี้ไปยังโครงสร้างข้อมูลอื่นที่ข้อมูลเหล่านี้ถูกเก็บไว้ตัวอย่างเช่น โครงสร้างข้อมูลของการแทนแบบ list ผลคือ จะมี task ใหม่ถูกสร้างขึ้นมา และมีการสำเนาข้อมูลทั้งหมดของโครงสร้างข้อมูลของ parent processที่มีความเกี่ยวเนื่องกันไปควบคู่กัน task ใหม่ที่สร้างขึ้นมาเมื่อทำการ clone() จะถูกสร้างขึ้นมาอย่างไรก็ตามมันแทนที่จะสำเนาข้อมูลทั้งหมดของโครงสร้างข้อมูล ให้กำหนดจุดของ task ใหม่ไปยังโครงสร้างข้อมูลของ parent task ขึ้นอยู่กับเซทค่าของ flag ให้ผ่าน ในการ clone()

Thread เป็นการควบคุมการทำงานภายในของโปรเซส Multithreaded โปรเซสจะประกอบไปด้วย thread ที่ทำหน้าที่ต่างกันแต่ทำงานอยู่บน address เดียวกันข้อดีของ Mulithreding ประกอบไปด้วย การเพิ่มการตอบสนองต่อผู้ใช้ ทั้งแชร์ ทรัพยากรในโปรเซส ความประหยัดและความสามารถของ task ในสถาปัตยกรรมแบบ Multiprocessor

ในระดับของ user thread คือสิ่งที่โปรแกรมเมอร์มองเห็นและ ระบบปฏิบัติการ kernel จะสนับสนุนจัดการในระดับ kernel โดยทั่วไปแล้ว user-level จะเร็วกว่าในการสร้างและจัดการมากกว่า kernel thread และไม่ก้าวก่ายสิ่งที่ kernel ต้องการ

3 รูปแบบที่แตกต่างกันของ ความสัมพันธ์ของ model ระหว่าง user thread กับ kernel thread ดังนี้

1. Many-to-one model หมายถึง Kernel thread 1 หน่วย กับ User thread หลายหน่วย เป็นการออกแบบที่จะยอมให้เพียง thread เดียวที่เข้าถึง kernel ในกรณีที่ thread ไป block system call จะทำให้ process ทั้งหมดถูก block ไปด้วย โดยโมเดลนี้ยอมให้สร้าง user thread ได้ตามต้องการ แต่ไม่สามารถประมวลผลได้พร้อมกัน เพราะยอมให้เข้าใช้ kernel thread ได้ครั้งละ thread เท่านั้น

2. One-to-one model หมายถึง Kernel thread 1 หน่วย กับ User thread 1 หน่วย ซึ่งระบบปฏิบัติการจะยอมให้ thread อื่นประมวลผลได้เป็นระบบขนาน ที่ทำงานแบบมัลติโปรเซสเซอร์ มีการใช้หลักการนี้อยู่ในระบบปฏิบัติการ Windows ในปัจจุบัน โดยโมเดลนี้ต้องไม่ยอมให้สร้าง user thread มากเกินไป

3. Many-to-many model หมายถึง โมเดลที่ลดข้อจำกัดของ 2 แบบแรก ผู้ใช้สามารถสร้าง user thread เท่าที่จำเป็น และสัมพันธ์กับ kernel thread ที่รับการทำงานแบบขนานในแบบมัลติโปรเซสเซอร์ เมื่อมี thread ที่ block system call ทาง kernel จะจัดเวลาให้ thread อื่นเข้ามาประมวลผลก่อนได้

ระบบปฏิบัติการที่ทันสมัยส่วนใหญ่จะให้ kernel สนับสนุน thread ในระบบ OS แบบ Windows 98 NT 2000 และ XP และยังมี Solaris ของ Linux อีกด้วย

thread library จะให้โปรแกรมเมอร์ที่จัดการเกี่ยวกับ application เพื่อสร้างและจัดการเกี่ยวกับ thread Thread Library มีอยู่3ส่วนหลัก ๆ คือ

- POSIX Pthreads
- Win32 threads
- Java threads

2.7 DLT camera calibration (Direct Linear Transform)

การเทียบกล้องส่วนใหญ่จะใช้วิธี DLT โดยวิธีการ DLT จะใช้เซ็ตของจุดที่เราทราบค่าที่แน่นอนบนระนาบหรือบริเวณที่สนใจมาคำนวณ และเรียกระนาบที่มีจุดที่ใช้เทียบว่า Calibrate Frame เพื่อที่จะสามารถแก้ปัญหาในเรื่องของการเปลี่ยนระนาบ 2 มิติ เพื่อให้เป็นระนาบในระบบ 3 มิติ ด้วยการใช้เมทริกซ์ 3x4 เพื่อเป็นการ โปรเจ็คชั่นระนาบสองมิติบนพื้นที่ของระนาบสามมิติ

2D DLT Algorithm

ในการคำนวณโดยใช้วิธี DLT แบบสองมิติจะใช้ในการคำนวณเพื่อหาค่าความสัมพันธ์เชิงเส้นของข้อมูลสองชุด โดยในกรณีของสองมิติจะต่างกับกรณีของสามมิติในเรื่องของขนาดเมทริกซ์ของความสัมพันธ์ ซึ่งถ้าเป็นสองมิติเมทริกซ์ความสัมพันธ์จะเป็น 3x3 แต่ในกรณีของสามมิติจะเป็น 3x4 เพราะต่างกันในเรื่องจำนวนของพารามิเตอร์ และผลของความสัมพันธ์ที่จะจะเป็นพารามิเตอร์ของโคออดเนตสองมิติ ซึ่งเป็นการ Calibrate กล้องแบบสองมิตินั่นเอง

รูปแบบอัลกอริทึมอย่างง่ายของ DLT แบบสองมิติ โดยจะหาความสัมพันธ์ของ $x_i \leftrightarrow x'_i$. ดังนั้นจะได้สมการความสัมพันธ์เป็น $x'_i = Hx_i$. และสามารถเขียนอยู่ในรูป cross product ของระบบเวกเตอร์ ได้เป็น $x'_i \times Hx_i = 0$. จากสมการดังกล่าวทำให้สามารถเขียนตัวแปรให้อยู่ในรูปเวกเตอร์ได้ดังนี้

$$Hx_i = \begin{pmatrix} h^{1T} x_i \\ h^{2T} x_i \\ h^{3T} x_i \end{pmatrix}$$

โดยจะแทนแถวที่ j ของเมทริกซ์ H ด้วย H^{jT} และให้ X'_i แทนด้วย $(x'_i, y'_i, w'_i)^T$ และเมื่อนำไปแทนค่าจะได้

$$x'_i \times Hx_i = \begin{pmatrix} y'_i h^{3T} x_i - w'_i h^{2T} x_i \\ w'_i h^{1T} x_i - x'_i h^{3T} x_i \\ x'_i h^{2T} x_i - y'_i h^{1T} x_i \end{pmatrix}$$

เนื่องจาก $h^{jT}x_i = X_i^T h_j$ ที่ $j = 1, 2, 3$ และเมื่อนำไปแทนค่าจะได้

$$\begin{bmatrix} 0^T & -w'_i x_i^T & y'_i x_i^T \\ w'_i x_i^T & 0^T & x'_i x_i^T \\ -y'_i x_i^T & x'_i x_i^T & 0^T \end{bmatrix} \begin{pmatrix} h^1 \\ h^2 \\ h^3 \end{pmatrix} = 0$$

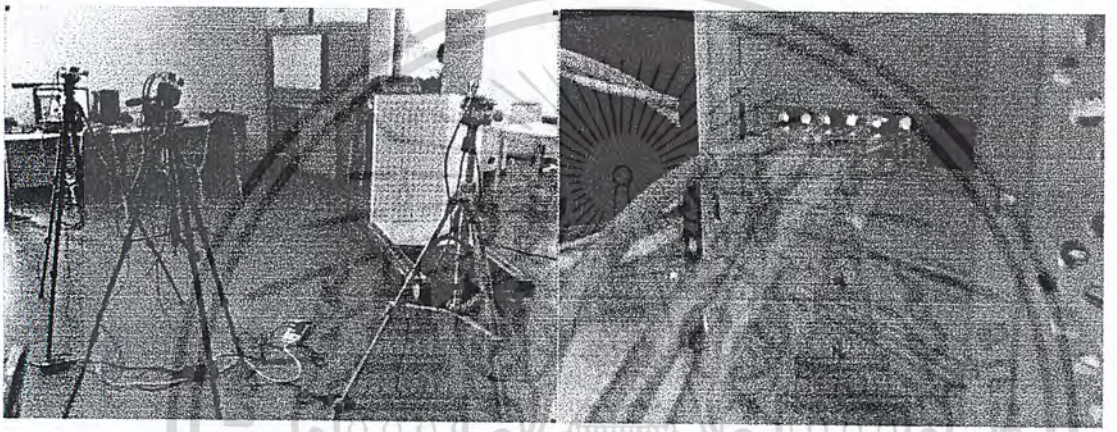
3D DLT Algorithm

จากการแปลงสมการสองมิติของ DLT สามารถแปลงเป็นสมการสามมิติได้ดังนี้

$$\begin{bmatrix} 0^T & -w_i X_i^T & y_i X_i^T \\ w_i X_i^T & 0^T & -x_i X_i^T \\ -y_i X_i^T & x_i X_i^T & 0^T \end{bmatrix} \begin{pmatrix} P^1 \\ P^2 \\ P^3 \end{pmatrix} = 0$$

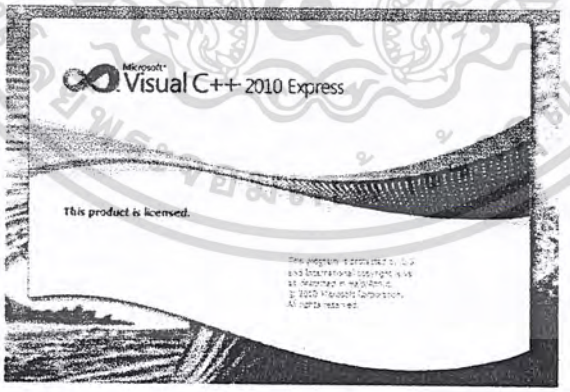
บทที่ 3
การทดลอง

ก่อนจะเริ่มทำการทดลองเราต้องทำการขั้นเตรียมการก่อนการทดลอง โดยทำการเชื่อมต่อกล้อง GigE Cameras จำนวน 3 กล้อง เข้ากับ PC ผ่านทาง Ethernet ด้วยสาย LAN cat.6 ตามรูปด้านล่าง



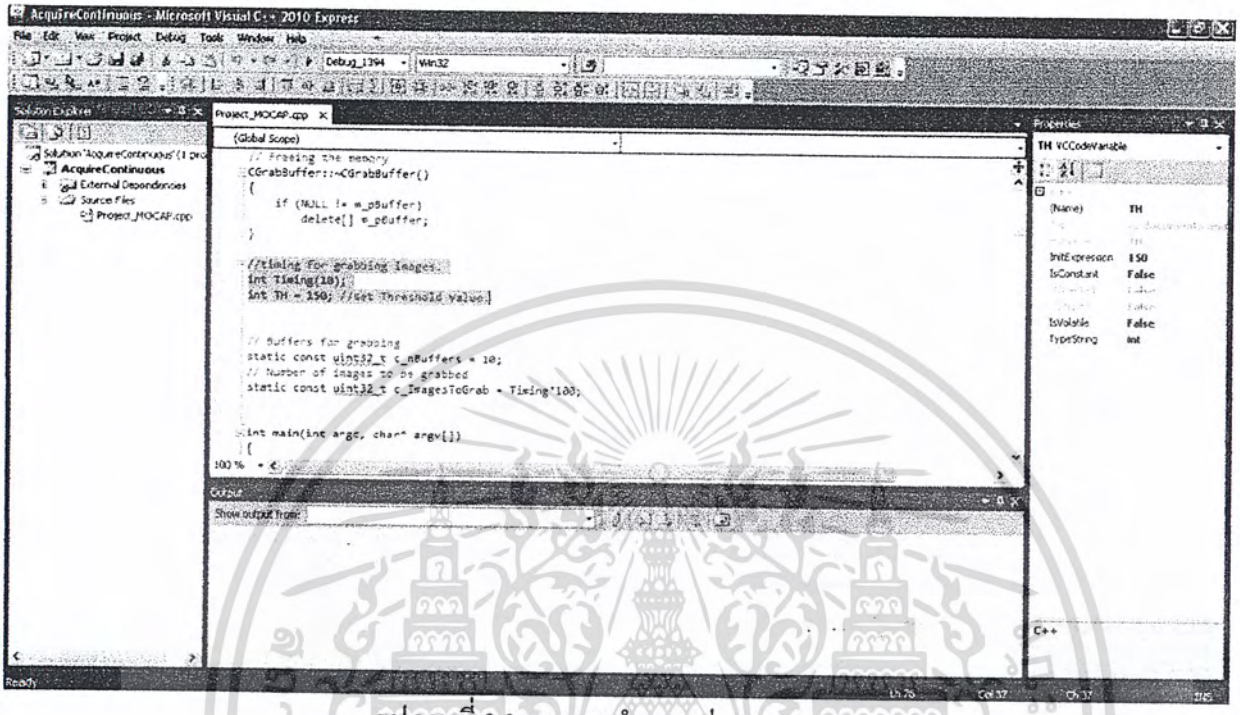
รูปภาพที่ 3.1 ภาพการเชื่อมต่อของกล้อง

ทำการเรียกใช้โปรแกรม Visual C++ 2010 ในการคอมไพล์และดีบักโค้ดที่ใช้



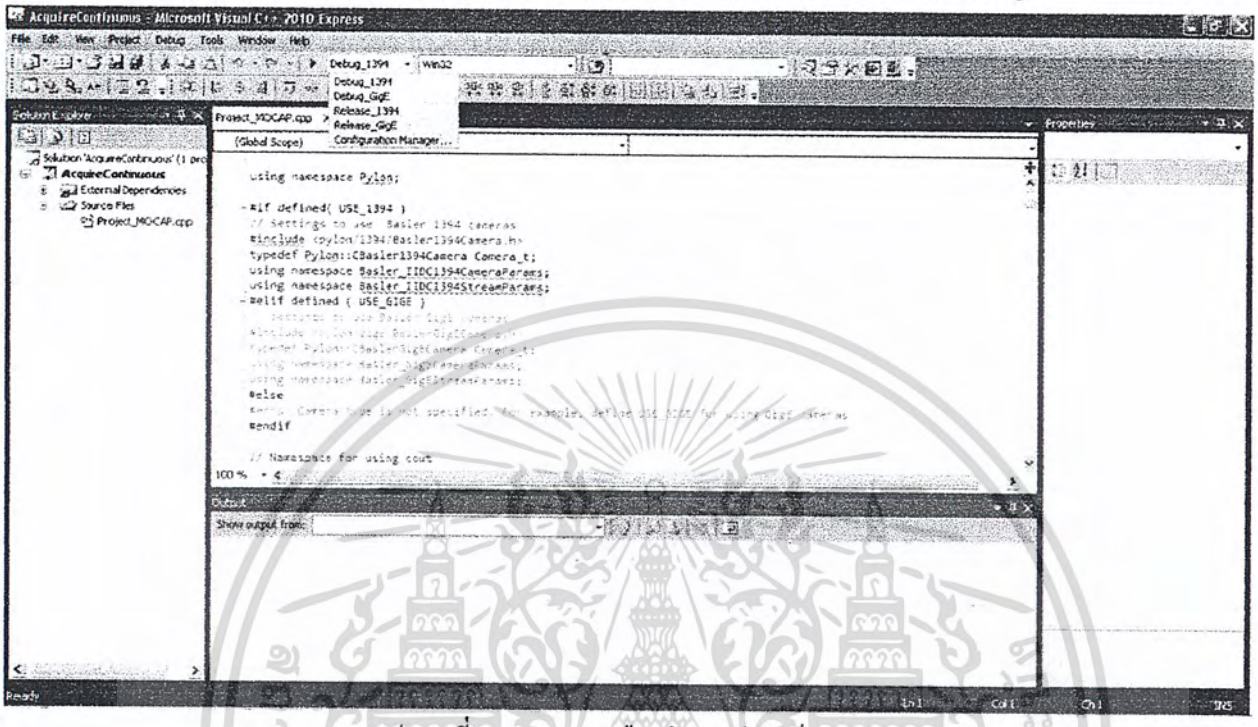
รูปภาพที่ 3.2 ภาพโปรแกรม Visual C++ 2010

ทำการกำหนดเวลาในการรับภาพ และกำหนดค่า Threshold ในการทำ Image Processing ภาพ



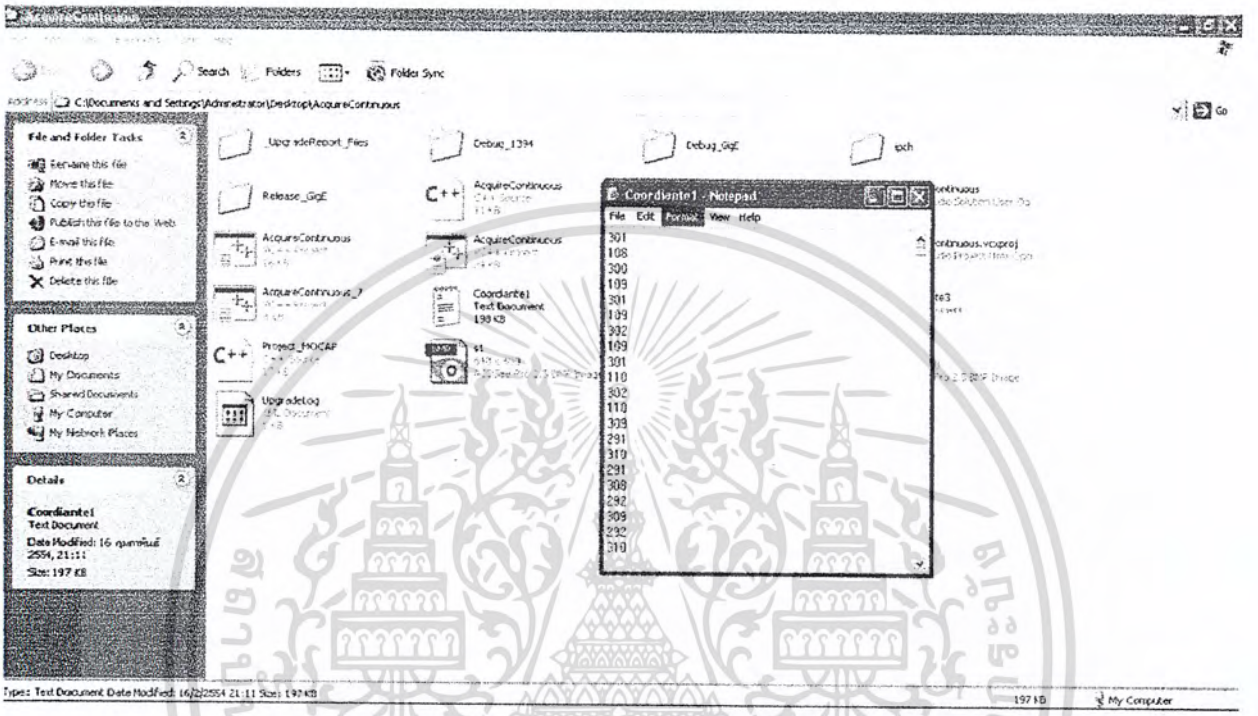
รูปภาพที่ 3.3 ภาพการกำหนดค่า Threshold

กำหนดรูปแบบการดีบักโปรแกรมให้เป็น Gig_E เนื่องจากกล้องที่ใช้เป็นประเภท GigE

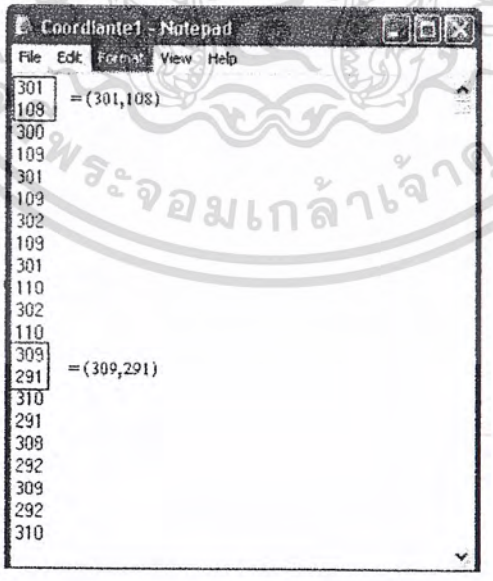


รูปภาพที่ 3.4 ภาพการเลือกโหมดดีบักเป็น gige

ทำการรันโค้ด และผลจากการรันโค้ดจะได้ไฟล์ที่มีข้อมูลของพิกัด 2 มิติออกมา 3 ไฟล์ โดยจะมีชื่อไฟล์ว่า Coordinate 1, Coordinate 2, Coordinate 3 ซึ่งภายในไฟล์จะมีลักษณะของข้อมูล ดังรูป



รูปภาพที่ 3.5 ภาพ Coordinate ที่ได้



รูปภาพที่ 3.6 ภาพวิธีตีความหมายไฟล์ผลลัพธ์

ส่วนของโค้ดโปรแกรมที่ใช้เก็บภาพและประมวลผลภาพ

```
// AcquireContinuous.cpp
```

ส่วนการประกาศรับ Header Files ในการใช้ฟังก์ชัน

```
// Include files to use the PYLON API
#include <pylon/PylonIncludes.h>
#include <stdlib.h>
#include <cmath>
#include <fstream>

using namespace Pylon;

#if defined( USE_1394 )
// Settings to use  Basler 1394 cameras
#include <pylon/1394/Basler1394Camera.h>
typedef Pylon::CBasler1394Camera Camera_t;
using namespace Basler_IIDC1394CameraParams;
using namespace Basler_IIDC1394StreamParams;
#elif defined ( USE_GIGE )
// settings to use Basler GigE cameras
#include <pylon/gige/BaslerGigECamera.h>
typedef Pylon::CBaslerGigECamera Camera_t;
using namespace Basler_GigECameraParams;
using namespace Basler_GigEStreamParams;
#else
#error Camera type is not specified. For example, define USE_GIGE for using GigE cameras
#endif

// Namespace for using cout
using namespace std;
```

ประกาศส่วนของรายละเอียดของฟังก์ชันที่เรียกใช้ภายในโปรแกรม

```
// This function can be used to wait for user input at the end of the sample program.
void pressEnterToExit()
{
    //comment the following two lines to disable wait on exit here
    cerr << endl << "Press enter to exit." << endl;
    while( cin.get() != '\n');
}

class CGrabBuffer
{
    public:
        CGrabBuffer(const size_t ImageSize);
        ~CGrabBuffer();
        uint8_t* GetBufferPointer(void) { return m_pBuffer; }
        StreamBufferHandle GetBufferHandle(void) { return m_hBuffer; }
        void SetBufferHandle(StreamBufferHandle hBuffer) { m_hBuffer = hBuffer; };
```

```
    protected:
        uint8_t *m_pBuffer;
        StreamBufferHandle m_hBuffer;
};

// Constructor allocates the image buffer
CGrabBuffer::CGrabBuffer(const size_t ImageSize):
        m_pBuffer(NULL)
{
    m_pBuffer = new uint8_t[ ImageSize ];
    if (NULL == m_pBuffer)
    {
        GenICam::GenericException e("Not enough memory to allocate image buffer",
__FILE__, __LINE__);
        throw e;
    }
}

// Freeing the memory
CGrabBuffer::~CGrabBuffer()
{
    if (NULL != m_pBuffer)
        delete[] m_pBuffer;
}
```

ส่วนการกำหนดเวลาในการรับโค้ดรับภาพ ค่าThreshold จำนวนบัพเฟอร์ที่ต้องการใช้และจำนวนภาพที่ต้องการ

```
//timing for grabbing images.
int Timing(10);
int TH = 150; //set Threshold Value.


// Buffers for grabbing
static const uint32_t c_nBuffers = 10;
// Number of images to be grabbed
static const uint32_t c_ImagesToGrab = Timing*100;
```

ส่วนเริ่มต้นฟังก์ชันหลักของโค้ดและเรียกใช้ Pylon Driver ในการเชื่อมต่อกับกล้อง GigE

```
int main(int argc, char* argv[])
{
    // Automagically call PylonInitialize and PylonTerminate to ensure the pylon runtime
system
    // is initialized during the lifetime of this object
    Pylon::PylonAutoInitTerm autoInitTerm;

    try
    {
        // Get the transport layer factory
        CTlFactory& TlFactory = CTlFactory::GetInstance();
```

```
// Create the transport layer object needed to enumerate or
// create a camera object of type Camera_t::DeviceClass()
ITransportLayer *pTl = TlFactory.CreateTl(Camera_t::DeviceClass());

// Exit application if the specific transport layer is not available
if (! pTl)
{
    cerr << "Failed to create transport layer!" << endl;
    pressEnterToExit();
    return 1;
}
```

ส่วนของการระบุกล้องเพื่อจำแนกกล้องแต่ละตัวในการออกคำสั่ง และทำการเปิดกล้อง

```
// Get all attached cameras and exit application if no camera is found
DeviceInfoList_t devices;
if (0 == pTl->EnumerateDevices(devices))
{
    cerr << "No camera present!" << endl;
    pressEnterToExit();
    return 1;
}

// Create the camera object of the first available camera.
// The camera object is used to set and get all available
// camera features.
Camera_t Camera1(pTl->CreateDevice(devices[ 0 ]));
      Camera_t Camera2(pTl->CreateDevice(devices[ 1 ]));
      Camera_t Camera3(pTl->CreateDevice(devices[ 2 ]));


// Open the camera
Camera1.Open();
      Camera2.Open();
      Camera3.Open();
```

ประกาศ Stream Grabber เพื่อกำหนดเป็นเส้นทางการรับส่งบัพเฟอร์ในการเก็บภาพ

```
// Get the first stream grabber object of the selected camera
Camera_t::StreamGrabber_t StreamGrabber1(Camera1.GetStreamGrabber(0));
      Camera_t::StreamGrabber_t StreamGrabber2(Camera2.GetStreamGrabber(0));
      Camera_t::StreamGrabber_t StreamGrabber3(Camera3.GetStreamGrabber(0));


// Open the stream grabber
StreamGrabber1.Open();
       StreamGrabber2.Open();
        StreamGrabber3.Open();
```

กำหนดพารามิเตอร์ต่างๆของกล้องเพื่อกำหนดรูปแบบการรับภาพ

```
    // Set the image format and AOI
        Camera1.PixelFormat.SetValue(PixelFormat_Mono8);
        Camera1.OffsetX.SetValue(0);
        Camera1.OffsetY.SetValue(0);
        Camera1.Width.SetValue(640);
        Camera1.Height.SetValue(480);

            Camera2.PixelFormat.SetValue(PixelFormat_Mono8);
        Camera2.OffsetX.SetValue(0);
        Camera2.OffsetY.SetValue(0);
        Camera2.Width.SetValue(640);
        Camera2.Height.SetValue(480);

            Camera3.PixelFormat.SetValue(PixelFormat_Mono8);
        Camera3.OffsetX.SetValue(0);
        Camera3.OffsetY.SetValue(0);
        Camera3.Width.SetValue(640);
        Camera3.Height.SetValue(480);

        //Disable acquisition start trigger if available
        {
            GenApi::IEnumEntry* acquisitionStart1 = Camera1.TriggerSelector.GetEntry(
TriggerSelector_AcquisitionStart);
            if ( acquisitionStart1 && GenApi::IsAvailable( acquisitionStart1))
            {
                Camera1.TriggerSelector.SetValue( TriggerSelector_AcquisitionStart);
                Camera1.TriggerMode.SetValue( TriggerMode_Off);
            }

                    GenApi::IEnumEntry* acquisitionStart2 =
Camera2.TriggerSelector.GetEntry( TriggerSelector_AcquisitionStart);
            if ( acquisitionStart2 && GenApi::IsAvailable( acquisitionStart2))
            {
                Camera2.TriggerSelector.SetValue( TriggerSelector_AcquisitionStart);
                Camera2.TriggerMode.SetValue( TriggerMode_Off);
            }

                    GenApi::IEnumEntry* acquisitionStart3 =
Camera3.TriggerSelector.GetEntry( TriggerSelector_AcquisitionStart);
            if ( acquisitionStart3 && GenApi::IsAvailable( acquisitionStart3))
            {
                Camera3.TriggerSelector.SetValue( TriggerSelector_AcquisitionStart);
                Camera3.TriggerMode.SetValue( TriggerMode_Off);
            }
        }

        //Disable frame start trigger if available
        {
            GenApi::IEnumEntry* frameStart1 = Camera1.TriggerSelector.GetEntry(
TriggerSelector_FrameStart);
            if ( frameStart1 && GenApi::IsAvailable( frameStart1))
```

```
            {
                Camera1.TriggerSelector.SetValue( TriggerSelector_FrameStart);
                Camera1.TriggerMode.SetValue( TriggerMode_Off);
            }

        GenApi::IEnumEntry* frameStart2 = Camera2.TriggerSelector.GetEntry(
TriggerSelector_FrameStart);
            if ( frameStart2 && GenApi::IsAvailable( frameStart2))
            {
                Camera2.TriggerSelector.SetValue( TriggerSelector_FrameStart);
                Camera2.TriggerMode.SetValue( TriggerMode_Off);
            }

        GenApi::IEnumEntry* frameStart3 = Camera3.TriggerSelector.GetEntry(
TriggerSelector_FrameStart);
            if ( frameStart3 && GenApi::IsAvailable( frameStart3))
            {
                Camera3.TriggerSelector.SetValue( TriggerSelector_FrameStart);
                Camera3.TriggerMode.SetValue( TriggerMode_Off);
            }
        }

        //Set acquisition mode
        Camera1.AcquisitionMode.SetValue(AcquisitionMode_Continuous);
              Camera2.AcquisitionMode.SetValue(AcquisitionMode_Continuous);
               Camera3.AcquisitionMode.SetValue(AcquisitionMode_Continuous);

        //Set exposure settings
        Camera1.ExposureMode.SetValue(ExposureMode_Timed);
        Camera1.ExposureTimeRaw.SetValue(9800);

              Camera2.ExposureMode.SetValue(ExposureMode_Timed);
        Camera2.ExposureTimeRaw.SetValue(9800);

              Camera3.ExposureMode.SetValue(ExposureMode_Timed);
        Camera3.ExposureTimeRaw.SetValue(9800);
```

กำหนดขนาดของแต่ละบัฟเฟอร์และกำหนดขนาดของ Stream Buffer รวมถึงจัดลำดับของบัฟเฟอร์ด้วย

```
        // Get the image buffer size
        const size_t ImageSize = (size_t)(Camera1.PayloadSize.GetValue());


        // We won't use image buffers greater than ImageSize
        StreamGrabber1.MaxBufferSize.SetValue(ImageSize);
              StreamGrabber2.MaxBufferSize.SetValue(ImageSize);
              StreamGrabber3.MaxBufferSize.SetValue(ImageSize);

        // We won't queue more than c_nBuffers image buffers at a time
        StreamGrabber1.MaxNumBuffer.SetValue(c_nBuffers);
              StreamGrabber2.MaxNumBuffer.SetValue(c_nBuffers);
              StreamGrabber3.MaxNumBuffer.SetValue(c_nBuffers);
```

```
        // Allocate all resources for grabbing. Critical parameters like image
        // size now must not be changed until FinishGrab() is called.
        StreamGrabber1.PrepareGrab();
              StreamGrabber2.PrepareGrab();
              StreamGrabber3.PrepareGrab();

        // Buffers used for grabbing must be registered at the stream grabber.
        // The registration returns a handle to be used for queuing the buffer.
        std::vector<CGrabBuffer*> BufferList1;
              std::vector<CGrabBuffer*> BufferList2;
              std::vector<CGrabBuffer*> BufferList3;

        for (uint32_t i = 0; i < c_nBuffers; ++i)
        {
            CGrabBuffer *pGrabBuffer1 = new CGrabBuffer(ImageSize);
                    pGrabBuffer1->SetBufferHandle(StreamGrabber1.RegisterBuffer(
                pGrabBuffer1->GetBufferPointer(), ImageSize));
                    CGrabBuffer *pGrabBuffer2 = new CGrabBuffer(ImageSize);
                    pGrabBuffer2->SetBufferHandle(StreamGrabber2.RegisterBuffer(
                pGrabBuffer2->GetBufferPointer(), ImageSize));
                    CGrabBuffer *pGrabBuffer3 = new CGrabBuffer(ImageSize);
                    pGrabBuffer3->SetBufferHandle(StreamGrabber3.RegisterBuffer(
                pGrabBuffer3->GetBufferPointer(), ImageSize));

            // Put the grab buffer object into the buffer list
            BufferList1.push_back(pGrabBuffer1);
                    BufferList2.push_back(pGrabBuffer2);
                    BufferList3.push_back(pGrabBuffer3);
                }


        for (std::vector<CGrabBuffer*>::const_iterator x = BufferList1.begin(); x !=
BufferList1.end(); ++x)
        {
            // Put buffer into the grab queue for grabbing
            StreamGrabber1.QueueBuffer((*x)->GetBufferHandle(), NULL);
        }
             for (std::vector<CGrabBuffer*>::const_iterator x = BufferList2.begin(); x
!= BufferList2.end(); ++x)
        {
            // Put buffer into the grab queue for grabbing
            StreamGrabber2.QueueBuffer((*x)->GetBufferHandle(), NULL);
        }
             for (std::vector<CGrabBuffer*>::const_iterator x = BufferList3.begin(); x
!= BufferList3.end(); ++x)
        {
            // Put buffer into the grab queue for grabbing
            StreamGrabber3.QueueBuffer((*x)->GetBufferHandle(), NULL);
        }


        // Let the camera acquire images continuously ( Acquisiton mode equals
        // Continuous! )
        Camera1.AcquisitionStart.Execute();
              Camera2.AcquisitionStart.Execute();
              Camera3.AcquisitionStart.Execute();
```

ส่วนการเก็บภาพและประมวลผลหาจุดที่เกินค่า Threshold เก็บไว้ในไฟล์ .txt

```
                    ofstream myfile1;
                    ofstream myfile2;
                    ofstream myfile3;

                    myfile1.open ("Coordiante1.txt");
                    myfile2.open ("Coordiante2.txt");
                    myfile3.open ("Coordiante3.txt");


        // Grab c_ImagesToGrab times
        for (int n = 0; n < c_ImagesToGrab; n++)
        {


            // Wait for the grabbed image with timeout of 3 seconds
            if
(StreamGrabber1.GetWaitObject().Wait(20)&&StreamGrabber2.GetWaitObject().Wait(20)&&Stream
Grabber3.GetWaitObject().Wait(20))
            {
                // Get the grab result from the grabber's result queue
                GrabResult Result1;
                            GrabResult Result2;
                            GrabResult Result3;

                StreamGrabber1.RetrieveResult(Result1);
                            StreamGrabber2.RetrieveResult(Result2);
                            StreamGrabber3.RetrieveResult(Result3);


              if (Grabbed == Result1.Status()&&Grabbed == Result2.Status()&&Grabbed ==
Result3.Status())
                {
             /*     // Grabbing was successful, process image
                cout << "Image #" << n << " acquired!" << endl;
                cout << "Size: " << Result1.GetSizeX() << " x "
                << Result1.GetSizeY() << endl;
                                    cout << "Image #" << n << " acquired!" << endl;
                cout << "Size: " << Result2.GetSizeX() << " x "
                << Result2.GetSizeY() << endl;
                   */
                // Get the pointer to the image buffer
                const uint8_t *pImageBuffer1 = (uint8_t *) Result1.Buffer();
                            const uint8_t *pImageBuffer2 = (uint8_t *)
Result2.Buffer();

                            const uint8_t *pImageBuffer3 = (uint8_t *)
Result3.Buffer();


              // Get the pointer to the image buffer
```

```
                        //Begin loop for Array of binary image.
                                for (int p=0;p<307200;p++)
                                {
                                        if (pImageBuffer1[p]>TH)
                                        {
  cout<<"first Camera Point At: ("<<p%640<<','<<(p/640)+1<<')'<<endl;
                                myfile1 <<p%640<<endl<<(p/640)+1<<endl;
                                        }
                                         if (pImageBuffer2[p]>TH)
                                        {
  cout<<"Second Camera Point At: ("<<p%640<<','<<(p/640)+1<<')'<<endl;
                                myfile2 <<p%640<<endl<<(p/640)+1<<endl;
                                        }
                                          if (pImageBuffer3[p]>TH)
                                        {
  cout<<"Third Camera Point At: ("<<p%640<<','<<(p/640)+1<<')'<<endl;
                                myfile3 <<p%640<<endl<<(p/640)+1<<endl;
                                        }


                                }


/*      cout << "Gray value of first pixel: " << (uint32_t) pImageBuffer[0]
                << endl << endl;                        */

                // Reuse the buffer for grabbing the next image
                if (n < c_ImagesToGrab - c_nBuffers){
                StreamGrabber1.QueueBuffer(Result1.Handle(), NULL);
                 StreamGrabber2.QueueBuffer(Result2.Handle(), NULL);
                 StreamGrabber3.QueueBuffer(Result3.Handle(), NULL);
                        }
            }

    else if (Failed == Result1.Status()||Failed == Result2.Status()||Failed ==
Result3.Status())
            {
                // Error handling
                cerr << "No image acquired!" << endl;
            /*    cerr << "Error code : 0x" << hex
                << Result1.GetErrorCode() << endl;
                cerr << "Error description : "
                << Result1.GetErrorDescription() << endl;
                        */
                // Reuse the buffer for grabbing the next image
                if (n < c_ImagesToGrab - c_nBuffers){
                    StreamGrabber1.QueueBuffer(Result1.Handle(), NULL);
                                StreamGrabber2.QueueBuffer(Result2.Handle(), NULL);
                                StreamGrabber3.QueueBuffer(Result3.Handle(), NULL);
                                }
            }
        }
        else
        {
            // Timeout
```

```
            cerr << "Timeout occurred!" << endl;

            // Get the pending buffer back (You are not allowed to deregister
            // buffers when they are still queued)
            StreamGrabber1.CancelGrab();
                        StreamGrabber2.CancelGrab();
                        StreamGrabber3.CancelGrab();

            // Get all buffers back
            for (GrabResult r; StreamGrabber1.RetrieveResult(r););
                        for (GrabResult r; StreamGrabber2.RetrieveResult(r););
                        for (GrabResult r; StreamGrabber3.RetrieveResult(r););

            // Cancel loop
            break;
        }


    }

        myfile1.close();
        myfile2.close();
        myfile3.close();
```

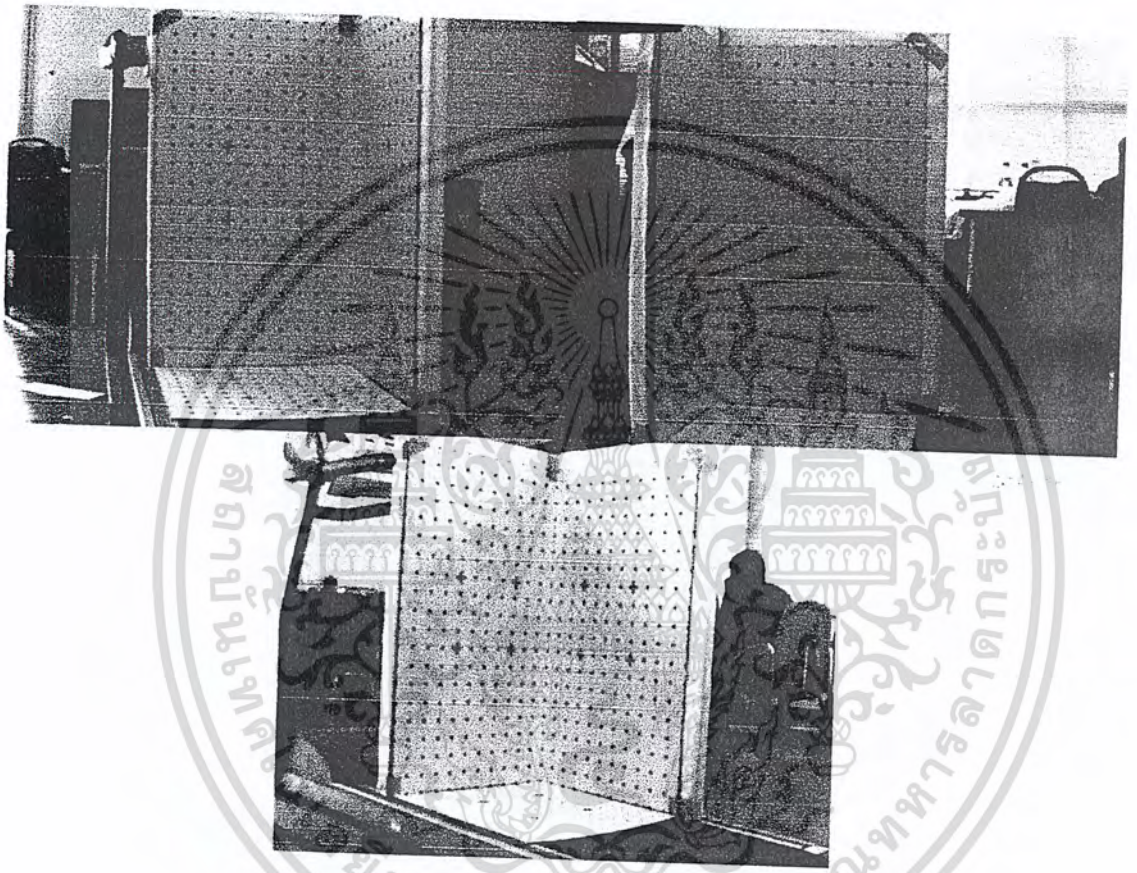
หยุดการรับภาพและคืนเมมโมรี่ที่จองไว้สำหรับบัฟเฟอร์ และสตรีมบัฟเฟอร์สู่ระบบ

```
    // Stop acquisition
    Camera1.AcquisitionStop.Execute();
          Camera2.AcquisitionStop.Execute();
          Camera3.AcquisitionStop.Execute();

    // Clean up

    // You must deregister the buffers before freeing the memory
    for (std::vector<CGrabBuffer*>::iterator it = BufferList1.begin(); it !=
BufferList1.end(); it++)
    {
        StreamGrabber1.DeregisterBuffer((*it)->GetBufferHandle());
        delete *it;
        *it = NULL;
    }
          for (std::vector<CGrabBuffer*>::iterator it = BufferList2.begin(); it !=
BufferList2.end(); it++)
    {
        StreamGrabber2.DeregisterBuffer((*it)->GetBufferHandle());
        delete *it;
        *it = NULL;
    }
          for (std::vector<CGrabBuffer*>::iterator it = BufferList3.begin(); it !=
BufferList3.end(); it++)
    {
        StreamGrabber3.DeregisterBuffer((*it)->GetBufferHandle());
        delete *it;
        *it = NULL;
```

```
        }

        // Free all resources used for grabbing
        StreamGrabber1.FinishGrab();
              StreamGrabber2.FinishGrab();
              StreamGrabber3.FinishGrab();

        // Close stream grabber
        StreamGrabber1.Close();
              StreamGrabber2.Close();
              StreamGrabber3.Close();

        // Close camera
        Camera1.Close();
              Camera2.Close();
              Camera3.Close();

    }
    catch (GenICam::GenericException &e)
    {
        // Error handling
        cerr << "An exception occurred!" << endl

        << e.GetDescription() << endl;
        pressEnterToExit();
        return 1;
    }
    // Quit the application
    pressEnterToExit();
    return 0;
}
```

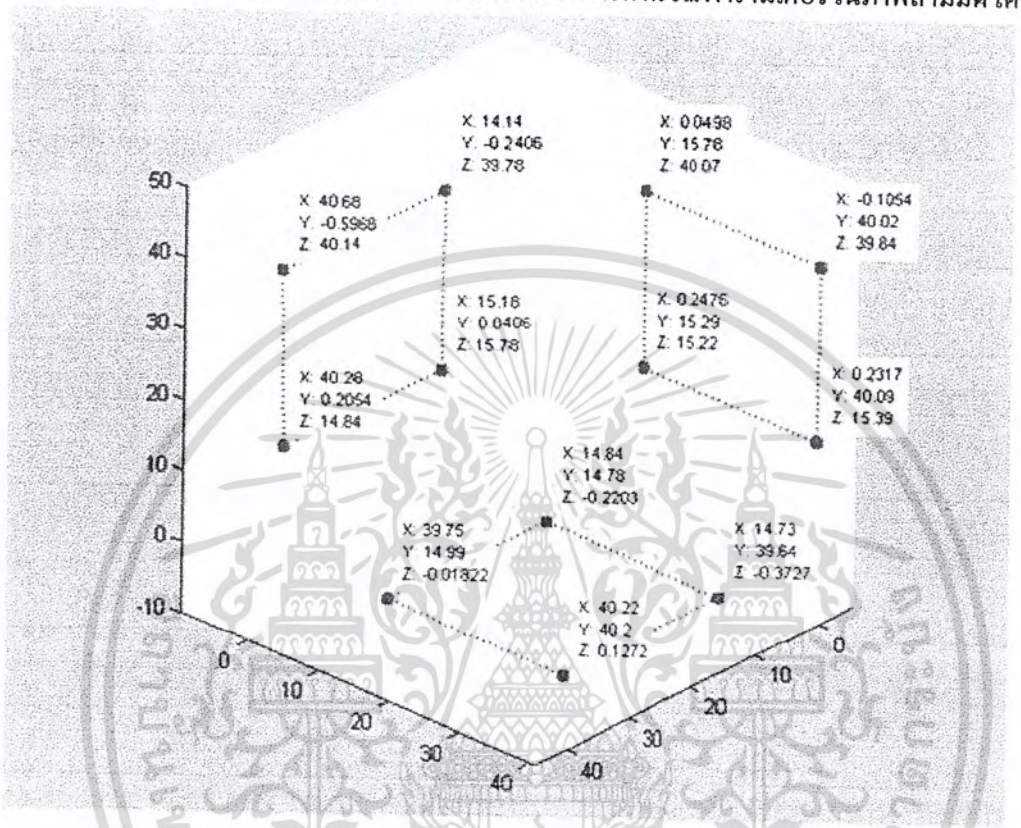
การทดลองเพื่อหา Error ของกระบวนการสร้างภาพสามมิติที่รับภาพโดยอัลกอริทึมของภาษาซี

1. ภาพถ่าย 3 ภาพจากกล้อง 3 ตัวที่วางต่างมุมกันด้วยโปรแกรม Pylon Viewer เพื่อนำภาพที่ได้มาทำการ Calibrate หาค่าตัวแปรพารามิเตอร์



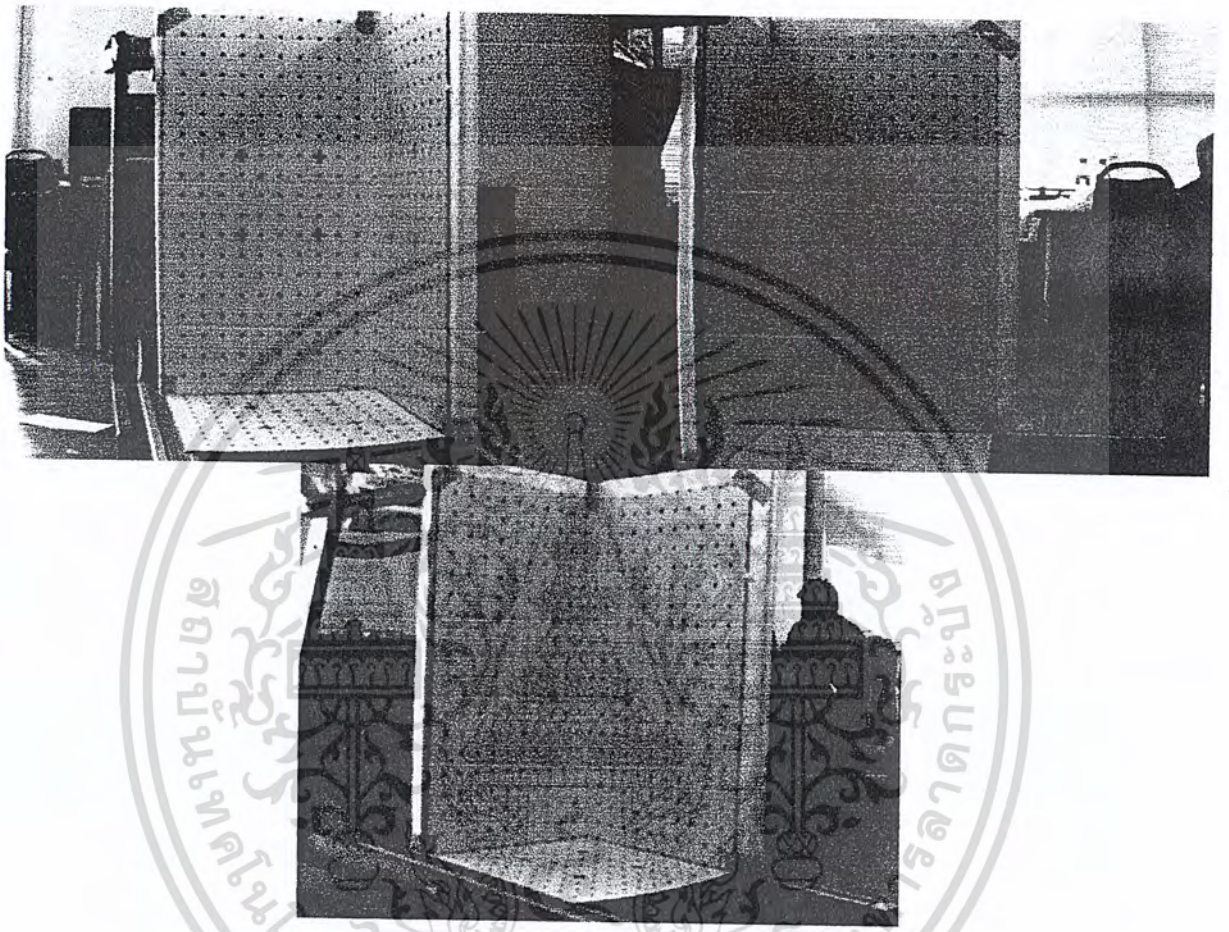
รูปภาพที่ 3.7 ภาพจากกล้อง 3 กล้องที่ใช้ในการ Calibrate

2. หลังจากได้ถ่ายภาพนิ่งของแต่ละกล้องได้ ก็นำไปใช้กับอัลกอริทึมของ Matlab เพื่อทำการ Calibrate ให้ได้ค่าพารามิเตอร์ในการสร้างภาพสามมิติ โดยจะแสดงการคำนวณพารามิเตอร์ในภาพสามมิติได้ดังภาพ



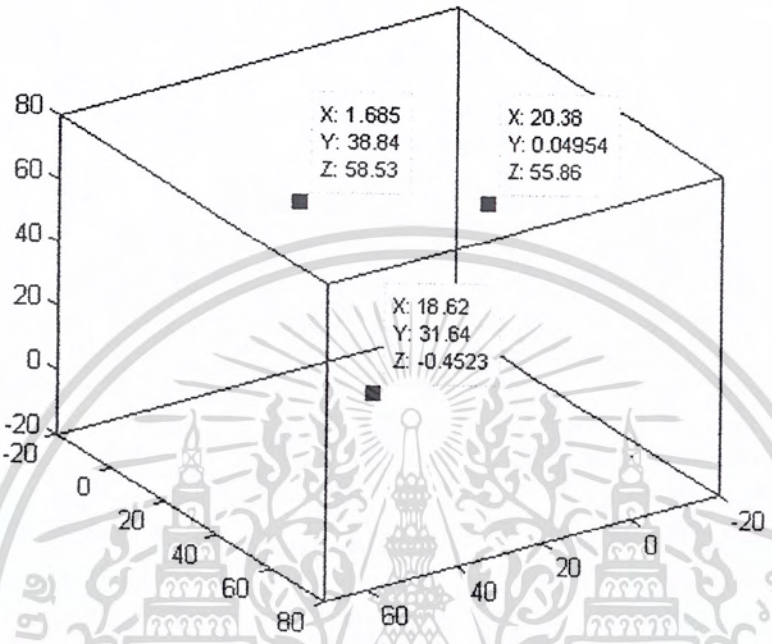
รูปภาพที่ 3.8 ภาพจากหลังการ Calibrate

3. สุ่มกำหนดจุดที่ต้องการหาตำแหน่งเพื่อใช้ในการเปรียบเทียบภาพสามมิติที่สร้างขึ้นกับระยะของจริง โดยระยะห่างระหว่างจุดจะมีระยะห่าง 5 เซนติเมตร (วัดจากจุดศูนย์กลาง)



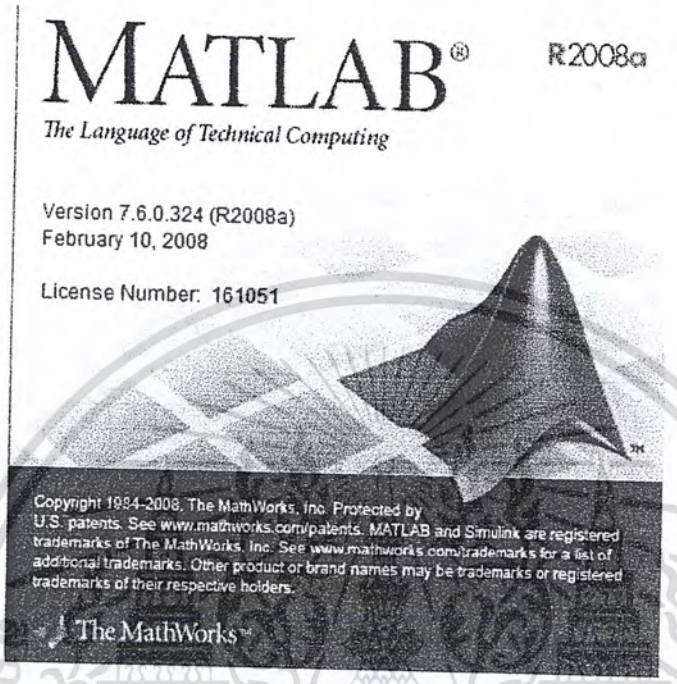
รูปภาพที่ 3.9 ภาพจากกล้อง 3 กล้องที่ทำการมาร์กจุดที่สนใจ

4. นำภาพที่มีจุดที่สนใจผ่านอัลกอริทึมของ Matlab เพื่อพล็อตจุดที่สนใจเป็นโคออดิเนตสามมิติ และนำผลที่ได้มาเปรียบเทียบกับพิกัดในความจริง



รูปภาพที่ 3.10 ภาพของตำแหน่งมาร์กในระนาบสามมิติ

หลังจากนั้นเราจะนำข้อมูลที่ได้มาประมวลผลภาพด้วยโปรแกรม MATLAB



รูปภาพที่ 3.11 ภาพโปรแกรม MATLAB

ส่วนของโค้ดที่ใช้ในการประมวลผลภาพ 3 มิติ

```
t1= xlsread('C:\Users\notebook\Documents\Project4\cam1.xls')

t2= xlsread(' C:\Users\notebook\Documents\Project4\cam2.xls')

t3= xlsread(' C:\Users\notebook\Documents\Project4\cam3.xls')

n1= 0

for i=1:1:2850;

n1= n1+1

a1= t1(n1,1);
```

```
b1= t1(n1,2);

c1= t1(n1,3);

d1= t1(n1,4);

e1= t1(n1,5);

f1= t1(n1,6);

plot(a1,b1,'.r',c1,d1,'.r',e1,f1,'.r',[a1 c1],[b1 d1],'-r',[c1 e1],[d1 f1],'-r')

xlim('manual')

ylim('manual')

xlim([0 658])

ylim([0 492])

set(gca,'YDir','reverse')

drawnow

pause(.01);

end

n2= 0

for i=1:1:2850;

n2= n2+1

a2= t2(n2,1);
```

```
b2= t2(n2,2);

c2= t2(n2,3);

d2= t2(n2,4);

e2= t2(n2,5);

f2= t2(n2,6);

plot(a2,b2,'.r',c2,d2,'.r',e2,f2,'.r',[a2 c2],[b2 d2],'-r',[c2 e2],[d2 f2],'-r')

xlim('manual')

ylim('manual')

xlim([0 658])

ylim([0 492])

set(gca,'YDir','reverse')

drawnow

pause(.01);

end



n3= 0

for i=1:1:2850;

n3= n3+1
```

```
a3= t3(n3,1);

b3= t3(n3,2);

c3= t3(n3,3);

d3= t3(n3,4);

e3= t3(n3,5);

f3= t3(n3,6);

plot(a3,b3,'.r',c3,d3,'.r',e3,f3,'.r',[a3 c3],[b3 d3],'-r',[c3 e3],[d3 f3],'-r')

xlim('manual')

ylim('manual')

xlim([0 658])

ylim([0 492])

set(gca,'YDir','reverse')

drawnow

pause(.01);

end
```

บทที่ 4
ผลการทดลอง และสรุปการทดลอง

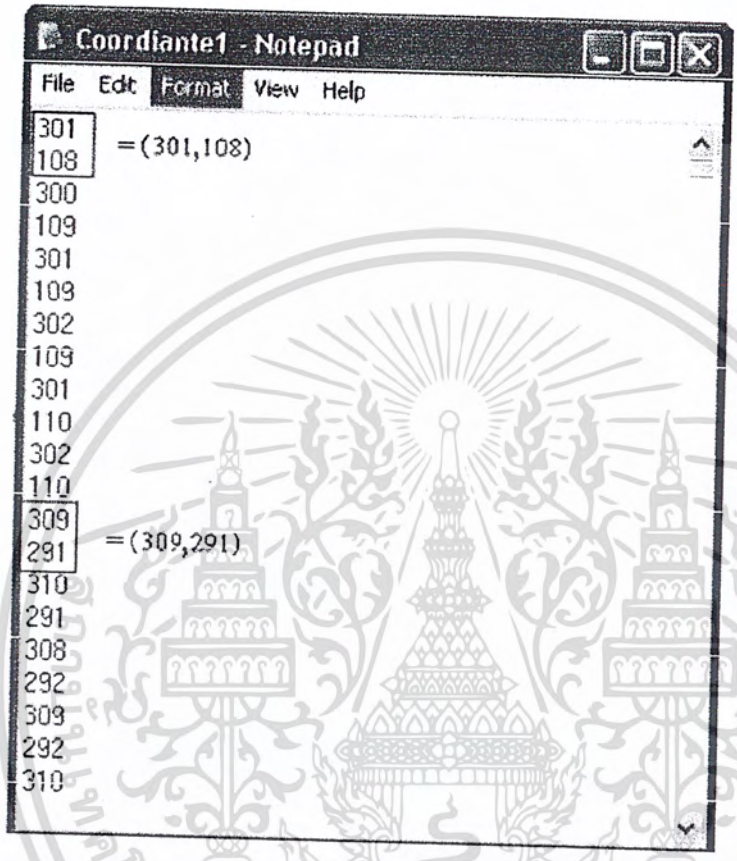
ผลการทดลอง

จากการทดลองการตรวจจับภาพการเคลื่อนไหวของมนุษย์ หลังจากเราทำการประมวลผลภาพโดยใช้โปรแกรม Visual Studio 2010 จะได้ไฟล์ข้อมูลนามสกุล .txt ที่บันทึกกลุ่มจุดสิ่งที่เราต้องการทำการตรวจจับ จำนวน 3 ไฟล์ เนื่องจากเราได้ทำการบันทึกโดยใช้กล้อง 3 กล้อง ดังภาพที่ 4.1

รูปภาพที่ 4.1 ภาพข้อมูลจุดที่บันทึกได้ในระนาบ 2 มิติ

โดยไฟล์ที่ชื่อ Coordinate1.txt , Coordinate2.txt และ Coordinate3.txt คือข้อมูลที่ได้จากกล้องที่ 1 2 และ 3 ตามลำดับ

การตีความหมายของไฟล์เพื่อนำไปใช้ในการหาโคออดิเนตจะเป็นตามรูปภาพด้านล่าง โดยโคออดิเนตที่ได้จะใช้โค้ดในส่วนของการประมวลผลภาพในการเรียกไฟล์ไปใช้



รูปภาพที่ 4.2 ภาพการแปลความหมายของไฟล์ที่ได้จากโค้ดส่วนรับภาพ

การตรวจสอบข้อมูลที่ได้โดยการเปรียบเทียบตำแหน่งของข้อมูล

	A	B	C	D
1	339			
2	109			
3	340			
4	109			
5	341			
6	109			
7	342			
8	109			
9	339			
10	110			
11	340			
12	110			
13	341			
14	110			
15	342			
16	110			
17	319			
18	293			
19	320			
20	293			
21	318			
22	294			
23	319			
24	294			

รูปภาพที่ 4.3 ภาพการเปรียบเทียบระหว่างข้อมูลจากผลการทดลอง กับภาพจากกล้อง

ตำแหน่งที่ใช้ตรวจจับ	เปอร์เซ็นความคลาดเคลื่อนของตำแหน่ง	
	แนวแกน X	แนวแกน Y
เอว	4.29	8.41
หัวเข่า	1.91	0.86
ข้อเท้า	0.32	0.34

ตารางที่ 4.1 ผลของการเปรียบเทียบหาเปอร์เซ็นความคลาดเคลื่อนของตำแหน่ง

ผลที่ได้จากการทดลองเพื่อหา Error ของกระบวนการสร้างภาพสามมิติที่รับภาพโดยอัลกอริทึมของภาษาซี

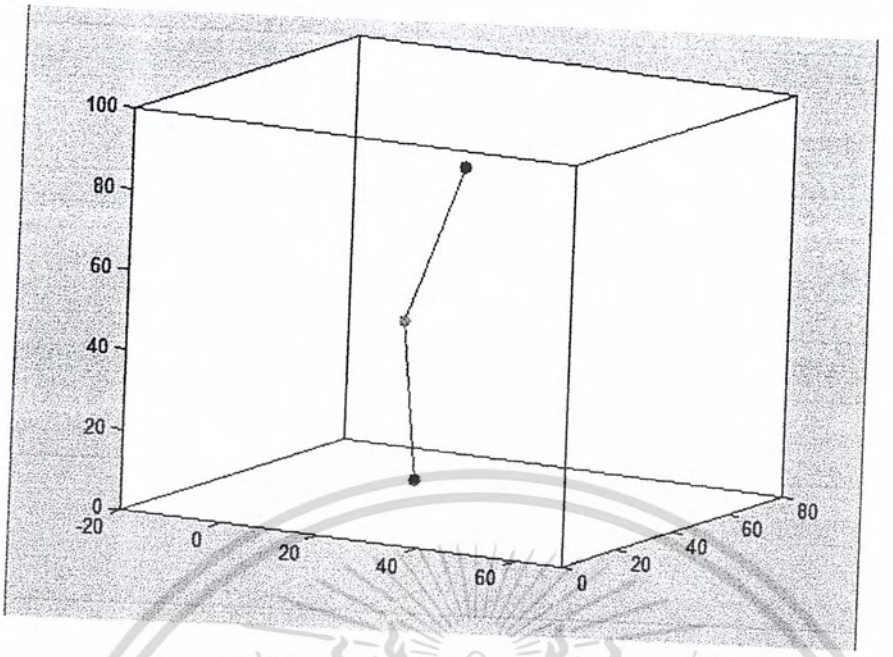
พิกัดสามมิติจริง	พิกัดสามมิติที่ได้จากการคำนวณ	% ความคลาดเคลื่อน
(0,35,55)	(1.685,38.84,58.53)	x =? y = 11.72% z = 6.418%
(20,0,55)	(20.38,0.04954,55.86)	x = 1.9% y =? z = 1.56%
(20,30,0)	(18.62,31.64,-0.4523)	x = 6.9% y= 2.13% z= ?

ตารางที่ 4.2 ผลของการเปรียบเทียบพิกัดในระนาบสามมิติ

หลังจากการนำไฟล์ข้อมูลนามสกุล .txt มาแปลงเป็น .xls ใน Microsoft Office Excel และทำการประมวลผลภาพด้วย MATLAB จะได้การจำลองการเดินในระบบสามมิติ

	A	B	C	D	E	F
1	Cam 3 X1	Y1	X2	Y2	X3	Y3
2	187	74	194	245	203	414
3	187	75	193	246	203	415
4	186	74	193	246	203	416
5	185	74	192	247	204	417
6	185	74	192	248	205	418
7	184	74	192	249	205	418
8	184	73	192	249	206	420
9	183	73	192	249	206	420
10	184	72	192	250	207	421
11	183	72	193	249	207	422
12	183	71	193	249	207	422
13	183	71	194	249	207	422
14	183	70	195	249	207	423
15	183	69	196	248	208	423
16	184	68	197	248	208	424
17	184	67	197	248	209	424
18	185	67	198	248	209	425
19	185	66	199	248	209	426
20	185	66	199	248	210	426

รูปภาพที่ 4.4 ภาพข้อมูล .xls ที่เป็นอินพุทใน MATLAB



รูปภาพที่ 4.5 ภาพการจำลองการเคลื่อนไหวในสามมิติ

สรุป

จากผลการทดลองที่ได้ในการเปรียบเทียบพิกัดสามมิติจากโปรแกรมกับพิกัดสามมิติในโลกจริง ทำให้เห็นว่าเปอร์เซ็นต์ค่าความคลาดเคลื่อนที่ได้ สูงสุดถึง 11.72% และต่ำสุดถึง 1.56% ซึ่งความคลาดเคลื่อนที่เกิดขึ้นนี้มาจาก 2 ส่วน ส่วนแรกเป็นความคลาดเคลื่อนในการรับภาพสองมิติเป็นความคลาดเคลื่อนที่เกิดจากการคำนวณเพื่อหาจุดศูนย์กลางของมาร์กเกอร์ และสภาพแวดล้อมในขณะที่ทำการรับภาพ ในส่วนที่สองเกิดจากการประกอบภาพสามมิติโดยใช้อัลกอริทึมของ MATLAB

เอกสารอ้างอิง

1. Real-time 3D Character Animation with Visual C++ by Nik Lever

2. http://en.wikipedia.org/wiki/Gigabit_Ethernet

3. http://www.baslerweb.com/beitraege/beitrag_en_99200.html

4. http://www.baslerweb.com/beitraege/maildownload_formular_en_59126.html

5. http://www.naturalpoint.com/optitrack/support/sample-applications.html

6. http://en.wikipedia.org/wiki/Motion_capture

7. http://en.wikipedia.org/wiki/Match_moving

8. http://en.wikipedia.org/wiki/3D_projection