การเข้ารหัสสัญญาณภาพแบบอัตโนมัติสำหรับกล้องไอพี โดยใช้ตัวควบคุมฟัซซี่ลอจิก

นายภูมินท์ ดวงมณี

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมโทรคมนาคม
มหาวิทยาลัยเทคโนโลยีสุรนารี
ปีการศึกษา 2553

AUTOMATIC VIDEO ENCODING FOR IP CAMERA USING FUZZY LOGIC CONTROLLER

Pumin Duangmanee

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Engineering in Telecommunication Engineering

Suranaree University of Technology

Academic Year 2010

การเข้ารหัสสัญญาณภาพแบบอัตโนมัติสำหรับกล้องไอพี
โดยใช้ตัวควบคุมฟัซซี่ลอจิก

มหาวิทยาลัยเทคโนโลยีสุรนารี อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

คณะกรรมการสอบวิทยานิพนธ์

_____

(ผศ. ดร.ชาญชัย ทองโสภา)
ประธานกรรมการ

_____

(ผศ. ดร.พีระพงษ์ อุฑารสกุล)
กรรมการ (อาจารย์ที่ปรึกษาวิทยานิพนธ์)

_____

(ผศ. ดร.ปิยาภรณ์ กระฉอดนอก)
กรรมการ

_____

(ศ. ดร.ชูกิจ ลิมปิจำนงค์)
รองอธิการบดีฝ่ายวิชาการ

_____

(รศ. น.อ. ดร.วรพจน์ ขำพิศ)
คณบดีสำนักวิชาวิศวกรรมศาสตร์

ภูมินท์ ดวงมณี : การเข้ารหัสสัญญาณภาพแบบอัตโนมัติสำหรับกล้องไอพีโดยใช้ตัวควบคุมฟัซซี่ลอจิก (AUTOMATIC VIDEO ENCODING FOR IP CAMERA USING FUZZY LOGIC CONTROLLER) อาจารย์ที่ปรึกษา : ผู้ช่วยศาสตราจารย์ ดร.พีระพงษ์ อุฑารสกุล, 154 หน้า

ในปัจจุบันเทคโนโลยีทางด้านระบบสมองกลฝังตัวมีความก้าวหน้าไปอย่างรวดเร็ว ทั้งในด้านสมรรถนะ ราคา และขนาดที่เล็กลง อีกทั้งความเร็วในการสื่อสารก็มีความเร็วสูงขึ้นตามไปด้วย อุปกรณ์กล้องไอพี (IP Camera) ที่ทำให้ผู้ใช้ดูภาพและเสียงผ่านเครือข่ายไอพี (IP Network) ก็ได้มีการพัฒนาไปในรูปแบบต่าง ๆ มากมาย เช่น ใช้เป็นตัวเฝ้าระวัง (Monitoring) เป็นต้น ในการดูข้อมูลภาพและเสียงจากกล้องไอพีผ่านเครือข่ายต่าง ๆ นั้น ในบางครั้งจะพบว่าข้อมูลภาพและเสียงที่ถูกส่งมานั้นไม่เหมาะสมกับแบนด์วิดท์ (Bandwidth) ของช่องสัญญาณที่ใช้งานอยู่ หรือในบางครั้งแบนด์วิดท์ของช่องสัญญาณนั้นสูงแต่มีความคับคั่งในการใช้งานมาก ซึ่งจะสังเกตเห็นได้จากการที่ภาพกระตุก ภาพค้าง ภาพมีการหน่วงทางเวลา เป็นต้น วิทยานิพนธ์ฉบับนี้จึงเสนอวิธีการปรับโครงสร้างของการเข้ารหัสในการส่งสัญญาณภาพและเสียงของกล้องไอพีโดยอัตโนมัติด้วยตัวควบคุมฟัซซี่ลอจิก โดยโปรแกรมขนาดเล็กที่ฝังตัวอยู่กับโปรแกรมดูภาพและเสียงจะทำหน้าที่ส่งข้อมูลสภาวะของช่องสัญญาณกลับไปยังกล้องไอพี จากนั้นตัวควบคุมฟัซซี่ลอจิกก็จะทำการควบคุมการเข้ารหัสภาพและเสียงภายในตัวกล้องไอพีให้เหมาะสมกับสภาวะช่องสัญญาณขณะนั้น ๆ ข้อดีของตัวควบคุมฟัซซี่ลอจิกคือสามารถใช้ควบคุมกระบวนการที่รู้แบบจำลองเพียงบางส่วนได้ดี และยังสามารถนำประสบการณ์และความเชี่ยวชาญของมนุษย์มาเป็นส่วนหนึ่งในการควบคุมได้ด้วย

สาขาวิชาวิศวกรรมโทรคมนาคม ลายมือชื่อนักศึกษา_____

ปีการศึกษา 2553 ลายมือชื่ออาจารย์ที่ปรึกษา_____

PUMIN DUANGMANEE : AUTOMATIC VIDEO ENCODING FOR IP CAMERA USING FUZZY LOGIC CONTROLLER. THESIS ADVISOR : ASST. PROF. PEERAPONG UTHANSAKUL, Ph.D., 154 PP.

VIDEO ENCODING/ IP CAMERA/ FUZZY LOGIC CONTROLLER/ EMBEDDED SYSTEM

Trends in an embedded technology have rapidly grown in terms of performance, cost and smaller size. Moreover, the concern on communication speed of embedded system is also increasing. IP camera allowing user to monitor via IP network has been popularly developed on embedded technology. To monitor a multimedia via networks, sometimes the quality of picture is not suitable for an available bandwidth because the data rate transmission is too high to send through network unpredictable congestions. This can be observed by either quality or delay of picture transmission. In this light, this thesis presents the method to automatically adjust a video encoding of IP camera by using fuzzy logic controller. Additional small program that runs on a player program sends congestion information back to IP camera. Then the fuzzy logic controller on IP camera controls a video encoding according to bandwidth condition at that time. The good point in fuzzy logic controller is to control the process that is known only some parts of process and use an experience of human expert to set a fuzzy control rule.

School of Telecommunication Engineering Student's Signature _____

Academic Year 2010 Advisor's Signature _____

กิตติกรรมประกาศ

ผู้วิจัยขอกราบขอบพระคุณ ผู้ช่วยศาสตราจารย์ ดร.พีรพงษ์ อุฑารสกุล อาจารย์ที่ปรึกษาวิทยานิพนธ์ ที่ได้ให้ความรู้ ข้อคิด และวิธีการต่าง ๆ ในการวิจัย รวมถึงได้กรุณาช่วยตรวจทานและแก้ไขรายงานวิทยานิพนธ์ตลอดระยะเวลาการวิจัย

ขอกราบขอบพระคุณ ผู้ช่วยศาสตราจารย์ ดร.ชาญชัย ทองโสภา ผู้ช่วยศาสตราจารย์ ดร.ปิยาภรณ์ กระฉอดนอก กรรมการสอบวิทยานิพนธ์ ที่ได้กรุณาให้คำปรึกษาแนะนำและข้อคิดเห็นต่าง ๆ ซึ่งเป็นประโยชน์ต่องานของผู้วิจัย

ขอกราบขอบพระคุณ รองศาสตราจารย์ ดร.รังสรรค์ วงศ์สรรค์ ที่ได้ให้คำแนะนำและโอกาสทางการศึกษาตลอดมา

ขอขอบคุณ มหาวิทยาลัยเทคโนโลยีสุรนารี สำนักงานกองทุนสนับสนุนการวิจัย (สกว.) ที่ได้ให้เงินทุนสนับสนุนอย่างดียิ่งจนงานวิจัยนี้สำเร็จลุล่วงด้วยดี

ขอขอบคุณ เพื่อน และน้องบัณฑิตศึกษาทุกท่านที่ให้การช่วยเหลือและให้คำแนะนำต่าง ๆ ในด้านการศึกษามาโดยตลอด

สุดท้ายนี้ ผู้วิจัยขอกราบขอบพระคุณบิดา มารดา ครอบครัว ญาติพี่น้องทุกท่านที่ให้การสนับสนุนด้านการศึกษาอย่างดียิ่ง และขอกราบขอบพระคุณอาจารย์ผู้สอนทุกท่านนับแต่อดีตจนถึงปัจจุบันที่ได้ประสิทธิ์ประสาทวิชาความรู้ในด้านต่าง ๆ แก่ผู้วิจัย

ภูมินท์ ดวงมณี

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

ตารางที่ **หน้า**

สารบัญรูป

รูปที่ **หน้า**

สารบัญรูป (ต่อ)

รูปที่ **หน้า**

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ในปัจจุบันเทคโนโลยีทางด้านระบบสมองกลฝังตัวมีความก้าวหน้าไปอย่างรวดเร็ว ทั้งในด้านสมรรถนะ ราคา ขนาดที่เล็กลง อีกทั้งความเร็วในการสื่อสารก็มีความเร็วสูงขึ้นตามไปด้วย อุปกรณ์พวกกล้องไอพี (IP Camera) ที่ทำให้ผู้ใช้สามารถดูภาพและเสียงผ่านเครือข่ายไอพี (IP Network) ก็ได้มีการพัฒนาไปในรูปแบบต่าง ๆ มากมาย เช่นใช้เป็นตัวเฝ้าระวัง (Monitoring) เป็นต้น

ในการดูข้อมูลภาพและเสียงจากกล้องไอพีผ่านเครือข่ายต่าง ๆ นั้น ในบางครั้งจะพบว่าข้อมูลที่ถูกส่งมาจากต้นทางนั้นไม่เหมาะสมกับแบนด์วิดท์ (Bandwidth) ของช่องสัญญาณที่ใช้งานอยู่ หรือในบางครั้งแบนด์วิดท์ของช่องสัญญาณนั้นสูงแต่มีความคับคั่งในการใช้งานมาก ซึ่งสังเกตเห็นได้จากการที่ภาพมีอาการกระตุก ภาพค้าง ภาพมีการหน่วงทางเวลา เป็นต้น ปัญหาเหล่านี้ได้รับการปรับปรุงแก้ไขให้ดีขึ้นด้วยการให้ผู้ใช้เลือกขนาดแบนด์วิดท์ของช่องทางการสื่อสารก่อนที่จะเลือกรับชมข้อมูลภาพและเสียง เช่น 128 kbps 192 kbps 384 kbps เป็นต้น เมื่อผู้ใช้เลือกขนาดแบนด์วิดท์แล้วกล้องไอพีก็จะทำการส่งข้อมูลภาพและเสียงที่ตรงตามแบนด์วิดท์นั้น ๆ

แต่ปัญหาก็ยังไม่หมดไปหากช่องทางการสื่อสารที่ผู้ใช้ใช้งานอยู่นั้นมีความคับคั่งในการใช้งานมาก เช่นผู้ใช้ที่ใช้ช่องทางการสื่อการแบบ 192 kbps ทำการเลือกขนาดของแบนด์วิดท์ที่ 192 kbps ซึ่งในขณะนั้นมีผู้ใช้คนอื่น ๆ อีกที่ใช้ช่องทางนี้ร่วมกัน ทำให้เกิดความคับคั่งในการใช้งานช่องสัญญาณขึ้นซึ่งทำให้แบนด์วิดท์จริงสำหรับผู้ใช้ขณะนั้นจะมีค่าต่ำกว่า 192 kbps

การบีบอัดข้อมูลก่อนที่จะส่งออกไปจึงเป็นแนวทางหนึ่งที่จะสามารถแก้ปัญหานี้ได้ ดังเช่นงานวิจัยของ Dong, He, and Zheng (2001) ได้นำเสนอรูปแบบการส่งข้อมูลภาพและเสียงแบบใหม่ที่ทำหน้าที่คล้ายกับเรียลไทม์โปรโตคอล (Real Time Protocol : RTP) แตกต่างกันตรงที่ตัดส่วนที่เป็นไทม์สแตมป์ (Time Stamp) ที่อยู่ในส่วนเฮดเดอร์ (RTP header) ทิ้งไปเพื่อลดขนาดข้อมูล มีการแบ่งสัญญาณเสียงออกเป็นกลุ่ม มีการใส่สัญญาณเสียงหลาย ๆ ช่องสัญญาณไว้ในข้อมูล 1 แพ็คเกจ

งานวิจัยของ Zhuang and Wang (2006) ได้นำเสนอระบบส่งสัญญาณภาพและเสียงโดยที่อุปกรณ์จับสัญญาณภาพและเสียงที่เป็นบอร์ดที่ใช้ซีพียูตระกูล ARM9 ที่มีระบบปฏิบัติการเอ็มเบ็ดเด็ดลีนุกซ์ (Embedded Linux) ทำงานอยู่บนบอร์ด ข้อมูลภาพและเสียงจะถูกบีบอัดก่อนที่จะส่งออกไปเก็บไว้ที่ดาต้าเบสเซอร์เวอร์ (Database Server) ผู้ใช้สามารถดูข้อมูลภาพและเสียงได้ที่เซอร์เวอร์นี้

ยังมีแนวทางที่จะรักษาคุณภาพการให้บริการ (Quality of Service : QoS) ในการส่งสัญญาณภาพและเสียง ก็คือการปรับคุณสมบัติของตัวส่งภาพและเสียง เช่น ความละเอียดของภาพ อัตราการส่งภาพ (Frame per Second : fps) โดยอัตโนมัติเมื่อแบนด์วิดท์เปลี่ยนแปลงไป

งานวิจัยของ Chaddha and Gupta (1996) ได้นำเสนอวิธีการส่งสัญญาณภาพและเสียงตั้งแต่ความเร็วระดับ ISDN ไปจนถึง Ethernet โดยได้นำเสนอวิธีการบีบอัดข้อมูลที่สามารถปรับระดับได้ (Scalable)

งานวิจัยของ Liu and He (2005) ได้สำเสนอเน้นไปทาง GPRS/CDMA เป็นหลัก โดยอาศัยแบบจำลองทางคณิตศาสตร์มาเป็นตัวช่วยในการปรับแต่งอัตราการส่งข้อมูลของตัวส่ง ซึ่งได้นำวิธีการ Rate-distortion และ Frame Skip มาใช้

การจองทรัพยากรของระบบก็เป็นอีกแนวทางคงคุณภาพการให้บริการการส่งข้อมูลได้ ดังเช่นงานวิจัยของ Shin and Hahm (1999) นั้นได้มีการนำวิธีที่เรียกว่า Resource Reservation Protocol ที่มีลักษณะเป็นการจองทรัพยากรขอบระบบไว้ชดเชยในช่วงเวลาที่แบนด์วิดท์เกิดความคับคั่งขึ้น

งานวิจัยของ Wei and Xu (2006) ได้มีการใช้ตัวควบคุมฟัซซี่ลอจิก (Fuzzy Logic Controller) ที่สามารถปรับแต่งตัวเอง (Self-Tuning) โดยเน้นไปที่การควบคุม เวลาตอบสนอง (Response Time) ของเว็บเพ็จ ระหว่างผู้ใช้งานชั้นธรรมดากับผู้ใช้งานชั้นพิเศษภายใต้การใช้งานที่คับคั่ง โดยที่ตัวควบคุมสามารถรักษาค่าเวลาตอบสนองให้อยู่ในค่าที่กำหนดได้ภายใต้การใช้งานที่คับคั่งและจะเห็นได้ว่าตัวควบคุมฟัซซี่ลอจิกนั้นสามารถใช้ควบคุมกระบวนการที่รู้แบบจำลองของเน็ตเวิร์คเพียงบางส่วนได้เป็นอย่างดี

วิทยานิพนธ์นี้จึงได้นำเสนอแนวทางการปรับแต่งโครงสร้างการเข้ารหัสข้อมูลภาพและเสียงที่ส่งออกมาจากกล้องไอพี โดยมีการปรับขนาดของภาพ การปรับอัตราการส่งภาพ การปรับทรัพยากรของระบบตามสภาพแบนด์วิดท์โดยอัตโนมัติด้วยตัวควบคุมฟัซซี่ลอจิก

ข้อดีของตัวควบคุมแบบฟัซซี่ลอจิกคือสามารถใช้ควบคุมกระบวนการที่รู้แบบจำลองเพียงบางส่วนได้ดี อีกทั้งสามารถแทรกประสบการณ์และความเชี่ยวชาญของมนุษย์ในเรื่องนั้น ๆ เข้าไปในกฎการควบคุมได้ โดยจะทำงานอยู่บนบอร์ดที่ใช้ซีพียูสถาปัตยกรรม ARM7 ที่มีราคาไม่แพง

1.2 วัตถุประสงค์ของการวิจัย

1. เพื่อออกแบบและสร้างกล้องไอพีที่สามารถปรับแต่งการเข้ารหัสภาพและเสียงตามสภาพแบนด์วิดท์ได้โดยอัตโนมัติ

2. ใช้ตัวควบคุมฟัซซี่ลอจิกมาเป็นตัวควบคุมการเข้ารหัส ตัวควบคุมนี้มีข้อดีคือสามารถใช้ควบคุมกระบวนการที่รู้แบบจำลองเพียงบางส่วนได้ดี

3. ตัวควบคุมฟัซซี่ลอจิกนั้นสามารถนำประสบการณ์และความเชี่ยวชาญของมนุษย์มาสร้างเป็นกฎในการควบคุมได้ง่าย

1.3 สมมุติฐานของการวิจัย

1. เมื่อมีโปรแกรมที่ทำหน้าที่ตรวจสอบสภาพแบนด์วิดท์ที่ปลายทาง และทำการส่งค่าสภาพแบนด์วิดท์กลับมายังกล้องไอพี จะช่วยให้กล้องไอพีส่งภาพและเสียงที่เหมาะสมกับสภาพแบนด์วิดท์ขณะนั้น ๆ ได้

2. ตัวควบคุมฟัซซี่ลอจิกซึ่งเป็นตัวควบคุมที่เหมาะสมกับกระบวนการที่รู้แบบจำลองเพียงบางส่วน สามารถใช้ควบคุมการเข้ารหัสส่งข้อมูลภาพและเสียงตามสภาพแบนด์วิดท์ที่มีความไม่แน่นอนได้

1.4 ขอบเขตของการวิจัย

1. จำลองการส่งสัญญาณภาพและเสียงระหว่างคอมพิวเตอร์โดยใช้โปรแกรมพัฒนาบนระบบปฏิบัติการวินโดส์ (Windows)

2. สร้าง/ปรับแต่งโปรแกรมส่งภาพและเสียงที่เป็นโอเพนซอร์ส (Open Source) โดยแทรกตัวควบคุมและตัวตรวจสอบสถานะแบนด์วิดท์ลงไป

3. โปรแกรมทำงานบนบอร์ดโปรเซสเซอร์ตระกูล ARM7

4. ทดสอบและวิเคราะห์ผลระหว่างการควบคุมแบบต่าง ๆ คือ แบบปรับอัตราการส่งภาพ แบบปรับคุณภาพของภาพและแบบผสมกัน

1.5 ประโยชน์ที่ได้รับจากการวิจัย

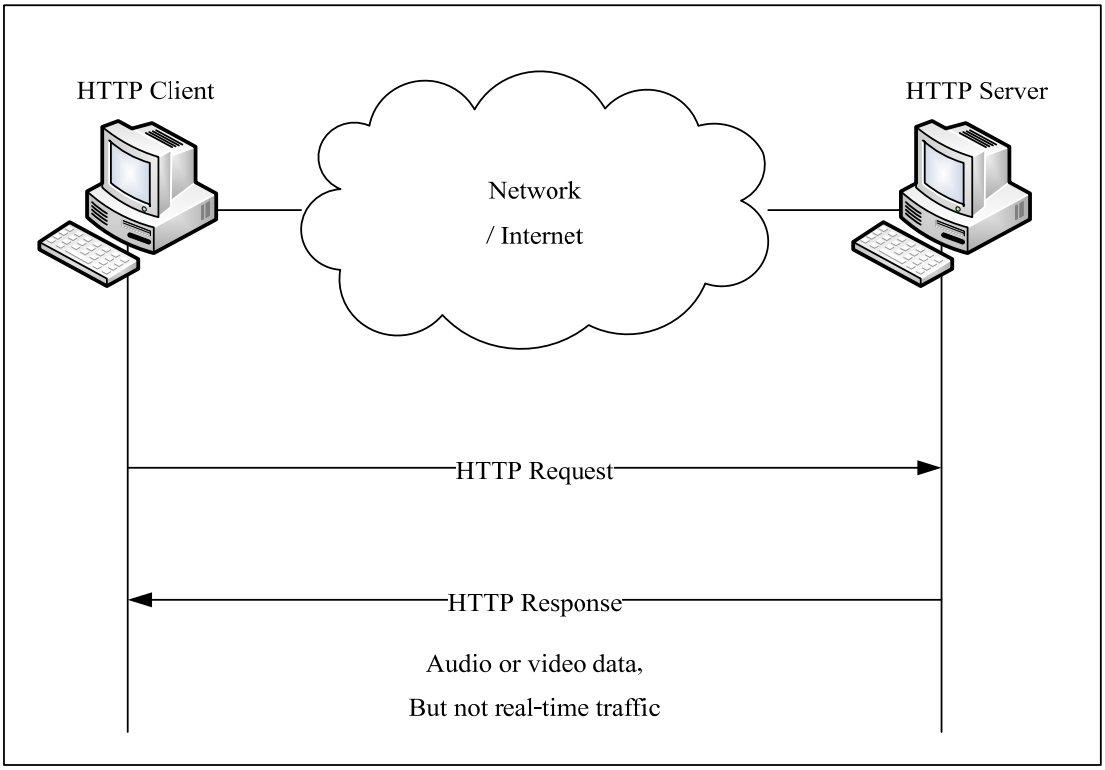
1. กล้องไอพีที่สามารถส่งภาพและเสียงที่มีคุณภาพเหมาะสมกับแบนด์วิดท์ขณะนั้น ๆ โดยอัตโนมัติ

2. สามารถนำประสบการณ์และความเชี่ยวชาญของมนุษย์มาเป็นส่วนหนึ่งของตัวควบคุมได้ง่าย

บทที่ 2
การส่งข้อมูลมัลติมีเดียผ่านเครือข่าย

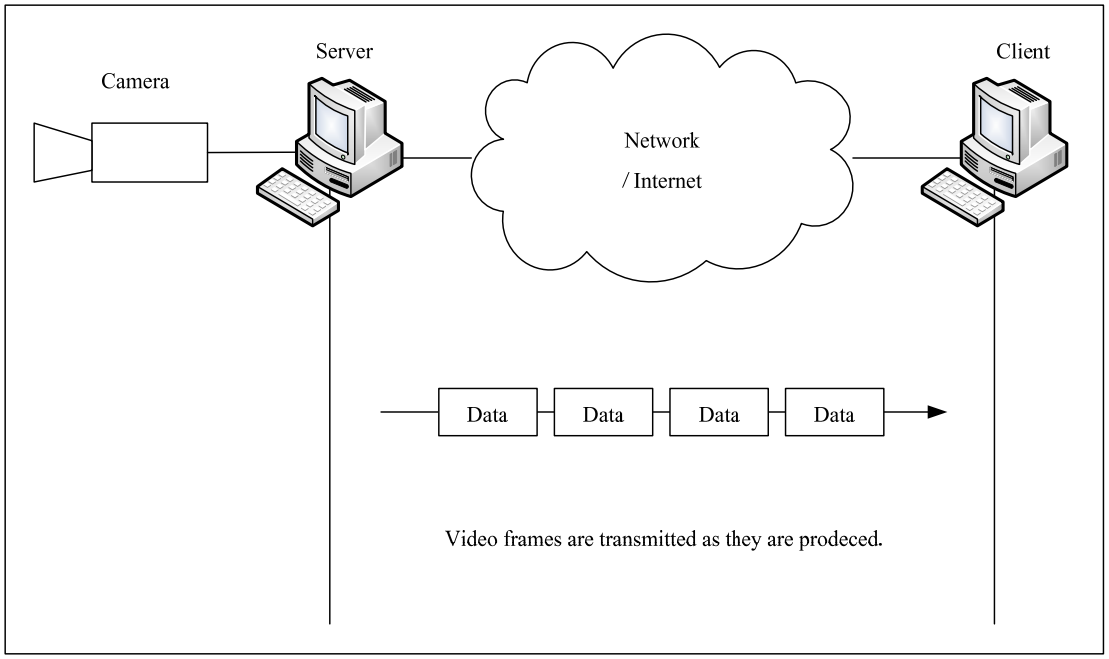
2.1 บทนำ

ในการส่งข้อมูลมัลติมีเดียผ่านเครือข่ายนั้นสามารถทำได้หลายรูปแบบ เช่นส่งแบบ HTTP TCP UDP RTP Unicast และ Multicast เป็นต้น ซึ่งสามารถแบ่งออกเป็นการส่งแบบ Real-Time และ Non Real-Time ประเภทของการส่งแบบ Non Real-Time คือการส่งในลักษณะที่การผลิตเนื้อหา การส่งข้อมูล การรับชม เกิดขึ้นในเวลาที่ต่างกัน เช่นการดาวน์โหลดภาพยนตร์หรือเพลงผ่านเว็บไซต์ เนื้อหาของเพลงหรือภาพยนตร์จะถูกสร้างขึ้นมาก่อนแล้วเก็บไว้เป็นข้อมูลในลักษณะไฟล์แล้วผู้ใช้จึงดาวน์โหลดข้อมูลเหล่านั้นมาลงที่คอมพิวเตอร์ของผู้ใช้หลังจากนั้นจึงรับชมเวลาใด ๆ ก็ได้ซึ่งสามารถแสดงได้ดังรูปที่ 2.1



รูปที่ 2.1 แสดงการส่งข้อมูลภาพและเสียงผ่านเครือข่ายแบบ Non Real-Time

ส่วนการส่งข้อมูลแบบเรียลไทม์นั้นการสร้างข้อมูลภาพและเสียง การส่งข้อมูล และการรับชมข้อมูลจะเกิดขึ้นในเวลาเดียวกันซึ่งในขณะที่รับชมข้อมูลอยู่นั้นด้านฝั่งผลิตข้อมูลก็กำลังผลิตไปด้วย ดังแสดงในรูปที่ 2.2

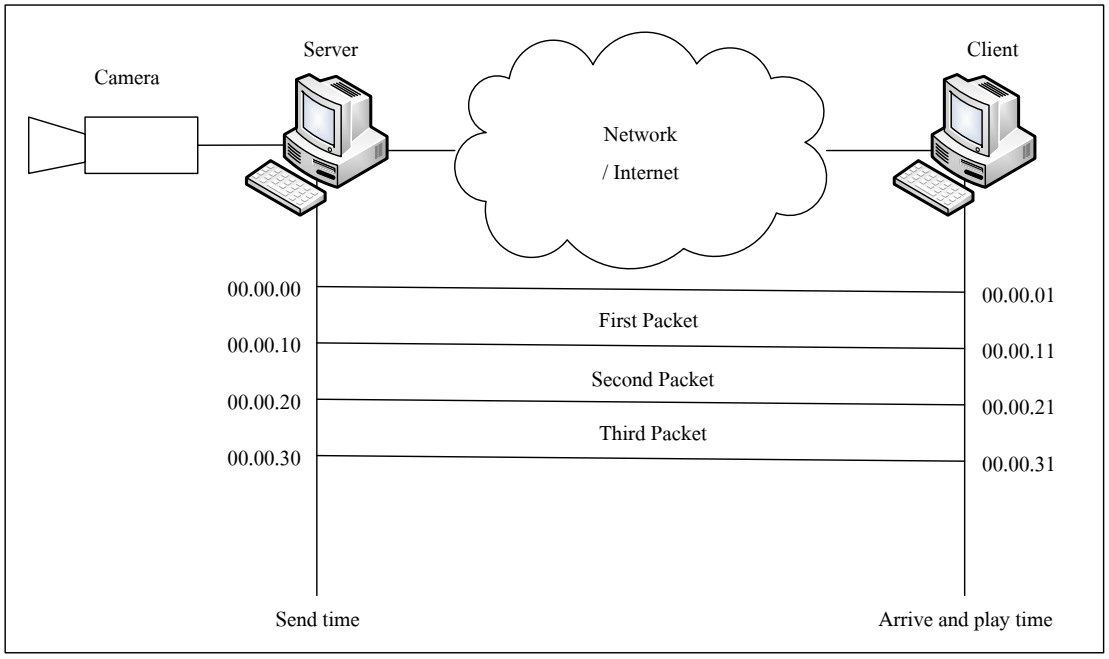


รูปที่ 2.2 แสดงรูปแบบการส่งข้อมูลภาพและเสียงแบบเรียลไทม์

โดยทั่วไปแล้วการรับส่งข้อมูลแบบเรียลไทม์นั้นจะมีลักษณะดังต่อไปนี้คือ

2.2 ความสัมพันธ์ทางด้านเวลา (Time Relationship)

ข้อมูลที่เป็นเรียลไทม์ในเครือข่ายจำเป็นที่จะต้องรู้ค่าความสำพันธ์ทางด้านเวลาในระหว่างแพ็คเกจในช่วงการส่งข้อมูลใด ๆ ตัวอย่างเช่น วิดีโอเซอร์เวอร์แบบเรียลไทม์ทำการส่งภาพวิดีโอแบบถ่ายทอดสดและทำการส่งภาพออกไปในเครือข่าย ซึ่งข้อมูลวิดีโอนี้มีทั้งหมด 3 ส่วนด้วยกัน โดยแต่ละส่วนจะมีความยาว 10 วินาที ข้อมูลวิดีโอส่วนแรกนั้นเป็นข้อมูลในช่วงเวลา 00.00.00 ข้อมูลวิดีโอในส่วนที่สองนั้นข้อมูลเริ่มต้นที่เวลา 00.00.10 และในส่วนที่สามข้อมูลเริ่มต้นที่ 00.00.20 โดยแต่ละแพ็คเกจใช้เวลา 1 วินาทีที่จะเดินทางไปถึงปลายทาง ดังนั้นที่ปลายทางจะสามารถแสดงผลข้อมูลภาพนี้ที่เวลา 00.00.01 สำหรับแพ็คเกจแรก ที่เวลา 00.00.11 สำหรับแพ็คเกจที่สอง และ 00.00.21 ในแพ็คเกจที่สาม ทั้งหมดนี้แสดงในรูปที่ 2.3

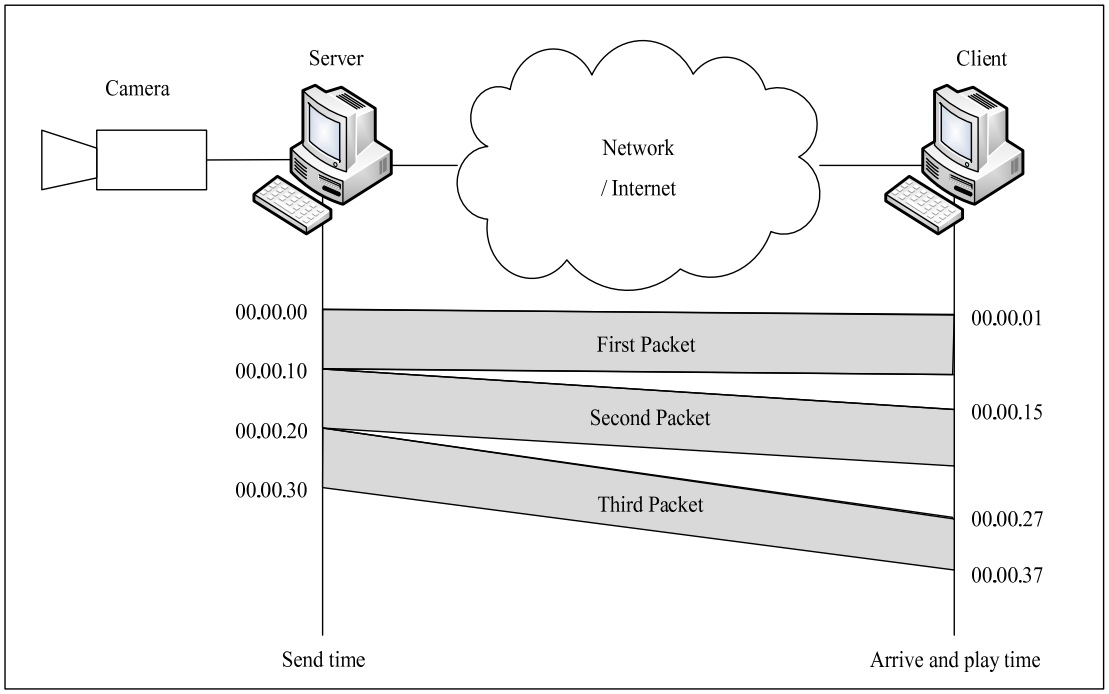


รูปที่ 2.3 ความสัมพันธ์ทางด้านเวลาของตัวส่งและตัวรับข้อมูล

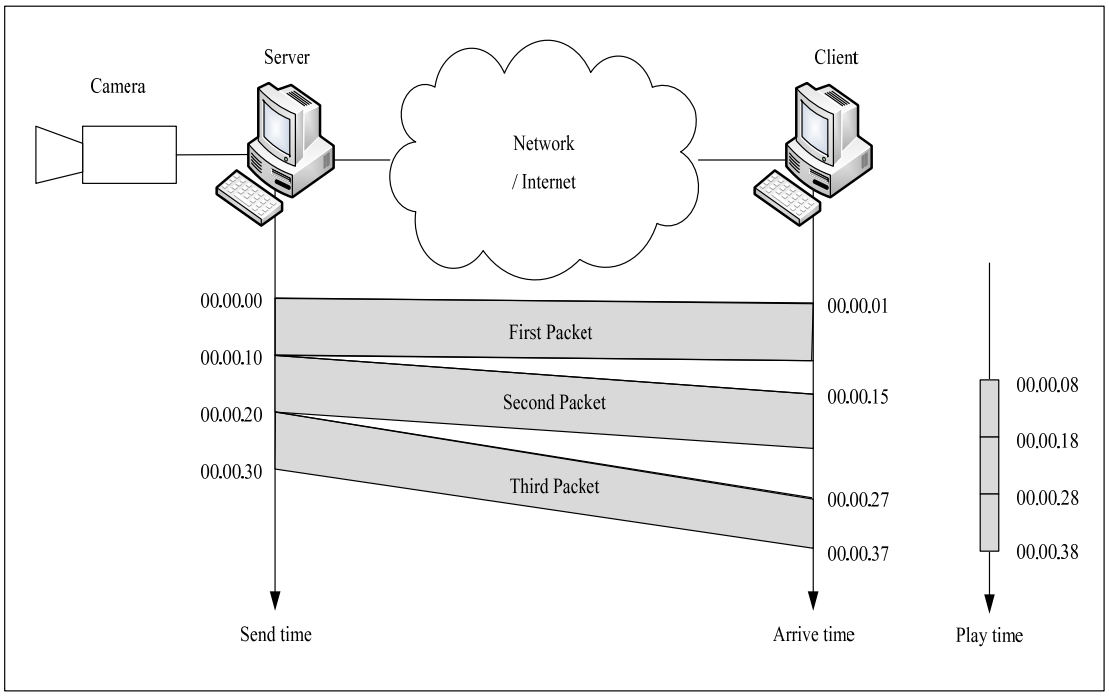
อะไรจะเกิดขึ้นถ้าแพ็คเกจข้อมูลมาถึงปลายทางด้วยค่าเวลาหน่วงที่แตกต่างกัน ตัวอย่างเช่น แพ็คเกจแรกมาถึงปลายทางในเวลา 00.00.01 (หน่วง 1 วินาที) แพ็คเกจที่สองมาถึงในเวลา 00.00.15 (หน่วง 5 วินาที) และแพ็คเกจที่ 3 มาถึงตอนเวลา 00.00.27 (หน่วง 7 วินาที) ถ้าโปรแกรมแสดงภาพเริ่มแสดงภาพที่เวลา 00.00.01 ซึ่งจะหมดข้อมูลที่เวลา 00.00.11 ถึงตอนนี้โปรแกรมแสดงภาพไม่สามารถแสดงภาพต่อไปได้เพราะว่าแพ็คเกจที่ 2 ยังเดินทางมาไม่ถึง ยังคงต้องใช้เวลาอีก 4 วินาทีกว่าจะมาถึง สิ่งนี้เป็นช่องว่างระหว่างแพ็คเกจข้อมูลทั้ง 3 แพ็คเกจ เหตุการณ์นี้เราเรียกว่า จิทเตอร์ (Jitter) ดังแสดงในรูปที่ 2.4

2.3 ไทม์แสตมป์ (Timestamp)

แนวทางที่จะสามารถแก้ปัญหาเรื่องจิทเตอร์ได้ก็คือไทม์แสตมป์ ถ้าข้อมูลแต่ละแพ็คเกจมีไทม์แสตมป์ที่ตัวมันเองได้ถูกผลิตขึ้นมาที่สามารถใช้เปรียบเทียบกับแพ็คเกจก่อนหน้าหรือหลังได้ ไคลเอนท์ก็จะสามารถให้ข้อมูลไทม์แสตมป์นี้ในการแสดงข้อมูลภาพได้อย่างถูกต้อง ตัวอย่างเช่น ให้แพ็คเกจแรกมีค่าไทม์แสตมป์เป็น 00.00.00 แพ็คเกจที่สอง 00.00.10 และแพ็คเก็จที่สาม 00.00.20 ถ้าจิทเตอร์เกิดขึ้นในแบบตัวอย่างที่ผ่านมา หากโปรแกรมทางด้านไคลเอนท์เริ่มแสดงข้อมูลภาพที่เวลา 00.00.08 ซึ่งแพ็คเกจที่ 2 จะถูกแสดงที่เวลา 00.00.18 และแพ็คเกจที่ 3 จะถูกแสดงที่เวลา 00.00.28 ซึ่งในกรณีนี้ช่องว่างระหว่างแพ็คเกจในการแสดงผลก็จะหายไป ดังแสดงให้เห็นในรูปที่ 2.5



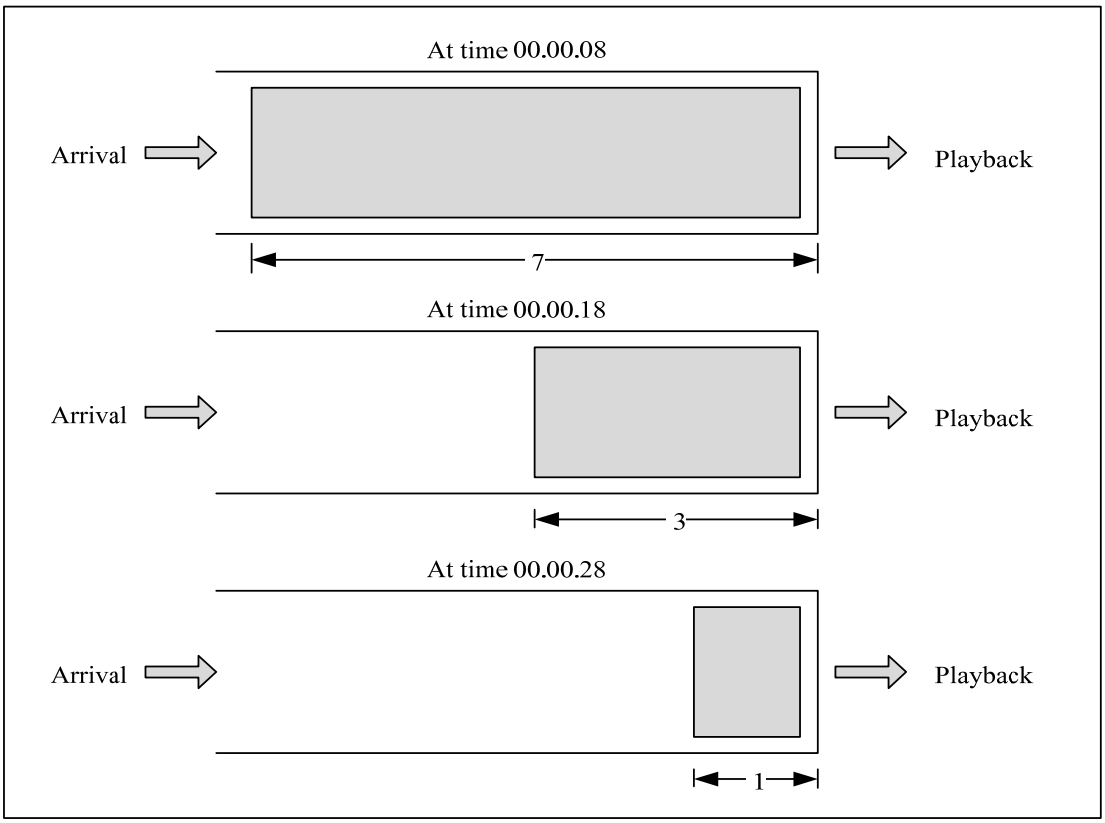
รูปที่ 2.4 การเกิดจิทเตอร์ในการส่งข้อมูลภาพและเสียงแบบเรียลไทม์



รูปที่ 2.5 ค่าไทม์แสตมป์ในส่วนหัวของข้อมูลภาพและเสียง

2.4 บัฟเฟอร์สำหรับแสดงข้อมูลภาพและเสียง (Playback Buffer)

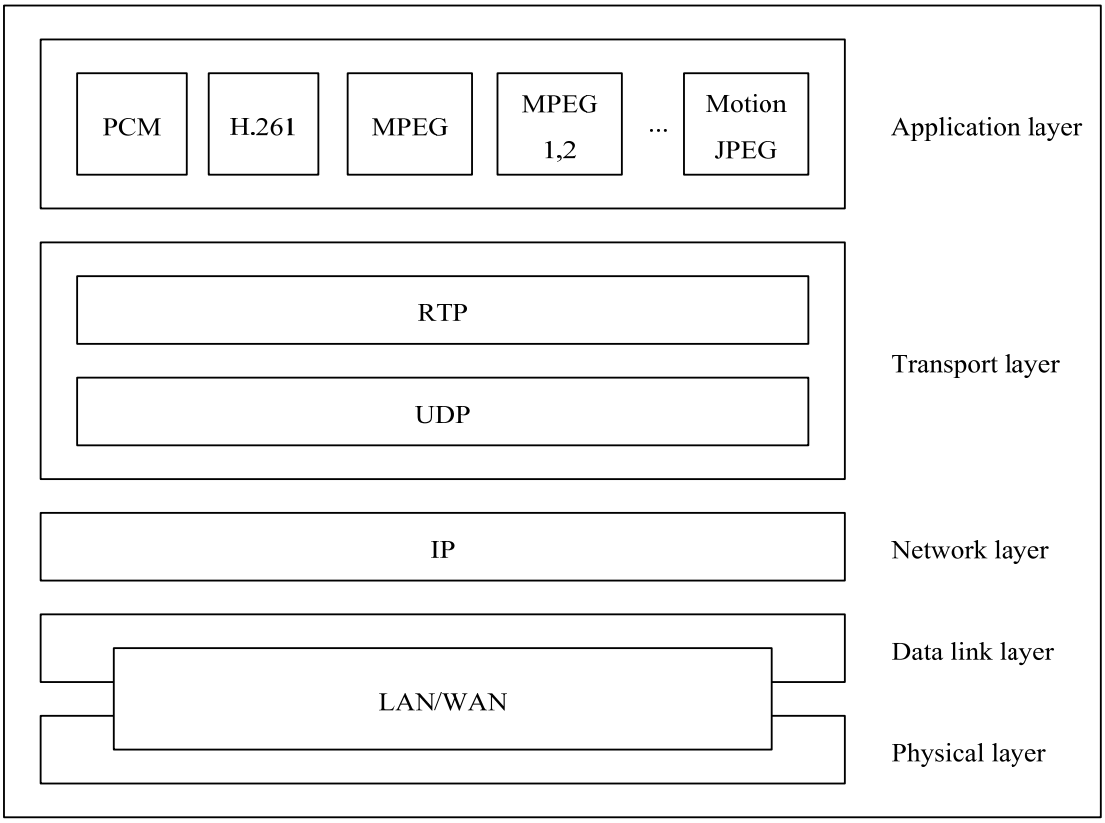
ในการที่จะสามารถแยกเวลาที่แพ็คเกจข้อมูลเดินทางมาถึงและเวลาที่จะแสดงข้อมูลให้ผู้ใช้ได้รับชม เราจำเป็นที่จะต้องมีบัฟเฟอร์สำหรับเก็บข้อมูลก่อนที่จะทำการแสดงข้อมูลออกไป เราเรียกสิ่งนี้ว่าเพลย์แบ็คบัฟเฟอร์ เมื่อการส่งข้อมูลเริ่มต้นขึ้นโปรแกรมที่แสดงภาพและเสียงทำการหน่วงเวลาออกไปจนกระทั่งปริมาณทางเวลาที่ได้เก็บไว้ถึงระดับที่กำหนดไว้จึงเริ่มแสดงผลภาพและเสียง ซึ่งสามารถแสดงได้ดังรูปที่ 2.6



รูปที่ 2.6 เพลย์แบ็คบัฟเฟอร์

2.5 RTP

เรียลไทม์ทรานสปอตโปรโตคอล (Realtime Transport Protocol : RTP) ถูกออกแบบมาเพื่อใช้จัดการกับการรับส่งข้อมูลแบบเรียลไทม์ในอินเตอร์เน็ต RTP นั้นไม่มีกลไกในการส่งมอบข้อมูลไปยังปลายทาง (มัลติคาส (Multicast) หมายเลขพอร์ต (Port number) และอื่น ๆ) มันจะต้องในงานร่วมกับ UDP โดย RTP นั้นจะวางตัวอยู่ระหว่าง UDP และแอปพลิเคชั่นที่ติดต่อกับผู้ใช้งาน หน้าที่หลักของ RTP ก็คือการใส่ค่าไทม์แสตมป์ การใส่ลำดับข้อมูล ซึ่งแสดงได้ในรูปที่ 2.7



รูปที่ 2.7 ระดับชั้นในการส่งข้อมูลภาพและเสียง

Ver	P	X	Contr. Count	M	Payload Type	Sequence Number
Timestamp						
Synchronization Source Idendifier						
Contributor Idendifier						
. . .						
Contributor Idendifier						

รูปที่ 2.8 ส่วนเฮดเดอร์ของ RTP แพ็คเกจ

ในรูปที่ 2.8 แสดงรูปแบบของข้อมูลในส่วนหัวของ RTP รูปแบบนี้สามารถใช้งานได้อย่างกว้างขวางในทุกการใช้งานในแบบเรียลไทม์

1) Ver : เป็นข้อมูล 2 บิตที่ใช้แสดงค่าเวอร์ชัน ซึ่งเวอร์ชันที่ใช้ในปัจจุบันคือ 2

2) P : เป็นข้อมูล 1 บิต ถ้ามีค่าเป็น 1 จะแสดงถึงการที่มีข้อมูลแพ็ดดิ้งที่ส่วนท้ายของแพ็คเกจข้อมูล ซึ่งในกรณีนี้ข้อมูลที่อยู่ด้านท้ายในส่วนแพ็ดดิงจะเป็นการกำหนดค่าความยาวของแพ็ดดิง ถ้าไม่มีแพ็ดดิงค่าในส่วนนี้จะเป็น 0

3) X : เป็นข้อมูล 1 บิต ถ้ามีค่าเป็น 1 จะแสดงถึงการขยายส่วนหัวของ RTP แบบพิเศษที่อยู่ระหว่างส่วนหัวปกติและข้อมูล ถ้ามีค่าเป็น 0 แสดงว่าไม่มีส่วนขยายนี้

4) Contributor Count : เป็นข้อมูล 4 บิต ที่แสดงถึงจำนวน Contributor ซึ่งมีได้สูงสุด 15 Contributor

5) M : เป็นข้อมูล 1 บิต เป็นตัวกำหนดค่าที่ใช้โดยระดับแอปพลิเคชัน เช่นใช้เป็นตัวบอกว่าเป็นข้อมูลชุดสุดท้าย

6) Payload Type : เป็นข้อมูล 8 บิตที่แสดงประเภทของข้อมูลที่นำมาซึ่งสามารถแสดงตัวอย่างบางส่วนได้ดังตารางที่ 2.1

7) Sequence Number : เป็นข้อมูล 16 บิต ใช้สำหรับใส่หมายเลขลำดับในการส่งแต่ละแพ็คเกจ ตัวเลขนี้ถูกเลือกมาในตอนเริ่มต้นในแบบสุ่ม และมันจะเพิ่มค่าขึ้น 1 ค่าในแต่ละลำดับของแพ็คเกจ ซึ่งจะถูกใช้ที่ผู้รับข้อมูลในการที่จะตรวจจับ Packet Loss หรือการที่ข้อมูลมาผิดลำดับ

8) Timestamp : เป็นข้อมูล 32 บิตที่แสดงถึงความสำพันธ์ทางด้านเวลาระหว่างแพ็คเกจค่าไทม์แสตมป์ที่ใช้ในแพ็คเกจแรกถูกกำหนดค่าแบบสุ่ม

9) Synchronization Source Identifier: ถ้ามีต้นกำเนิดของข้อมูลเพียงชุดเดียว ข้อมูล 32 บิตนี้จะเป็นการกำหนดค่าต้นกำเนินนั้น

10) Contributor Identifier : เป็นข้อมูล 32 บิต โดยแต่ละส่วนเป็นการนิยามค่าต้นกำเนิดข้อมูล

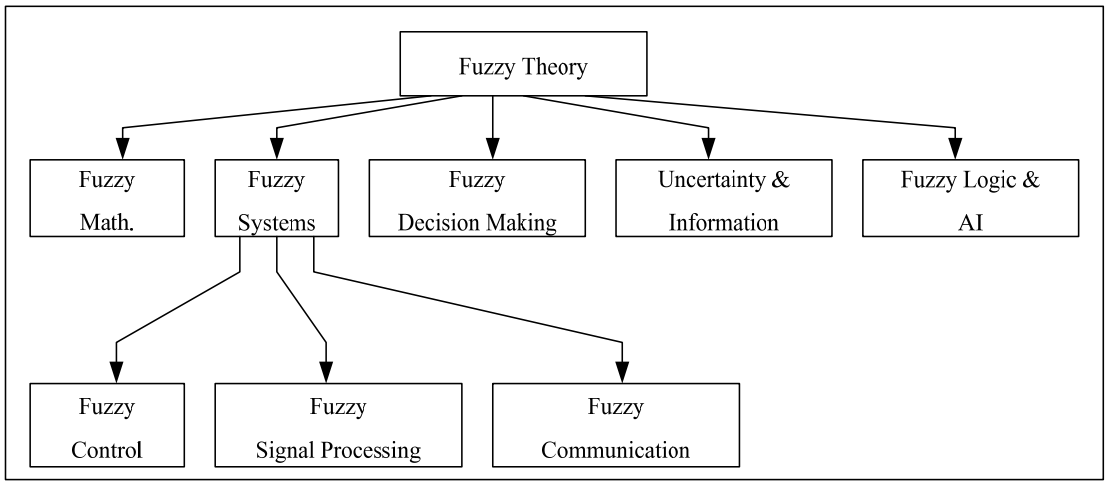
ตารางที่ 2.1 Playload Type ในส่วนหัวของ RTP

Type	Application	Type	Application	Type	Application
0	PCMu Audio	7	LPC audio	15	G728 audio
1	1016	8	PCMA audio	26	Motion JPEG
2	G721 audio	9	G722 audio	31	H.261
3	GSM audio	10-11	L16 audio	32	MPEG1 video
5-6	DV14 audio	14	MPEG audio	33	MPEG2 video

บทที่ 3
ตัวควบคุมฟัซซี่ลอจิก

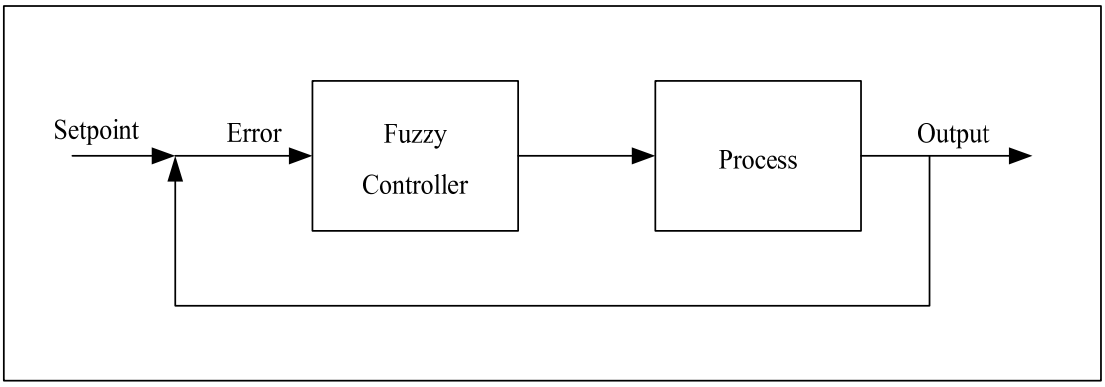
3.1 บทนำ

ทฤษฎีฟัซซี่ลอจิกนั้นมีจุดเริ่มต้นจากงานวิจัยของ Zadeh (1965) เขาได้พัฒนาแนวคิดของเสตท (State) ที่ใช้อยู่ในระบบควบคุม ในทศวรรษ 60 เขามีความคิดว่าการทฤษฎีควบคุมแบบเก่านั้นไม่สามารถแก้ปัญหาบางอย่างที่มีความแม่นยำสูงได้ และยังไม่สามารถจัดการกับระบบที่มีความซับซ้อนสูง ซึ่งต่อมาสิ่งที่เข้ามาแก้ปัญหาเหล่านี้ได้ถูกเรียกว่า ฟัซซี่เซต การนำฟัซซี่ไปใช้งานนั้นแตกแขนงใช้งานในสาขาต่าง ๆ มากมายซึ่งแสดงได้ดังรูปที่ 3.1



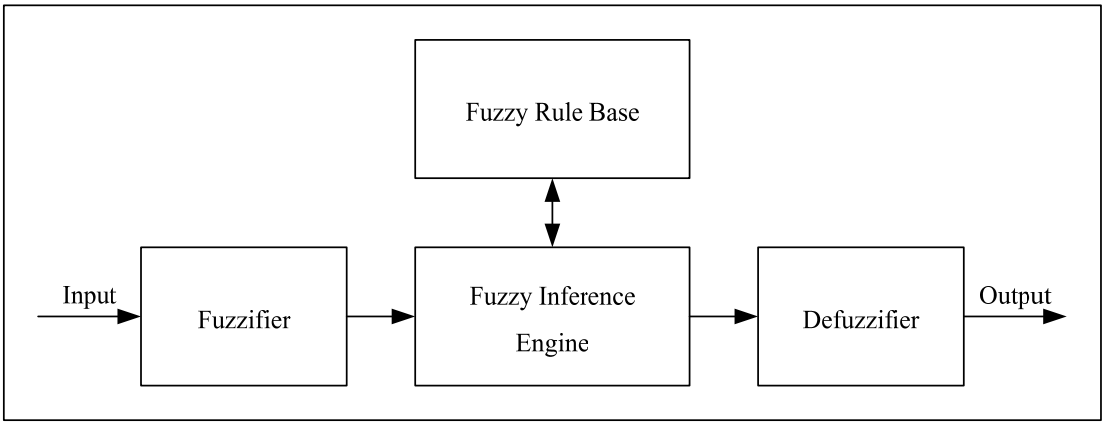
รูปที่ 3.1 แสดงการจำแนกการใช้งานของทฤษฎีฟัซซี่

สำหรับวิทยานิพนธ์นี้ได้นำทฤษฎีของฟัซซี่ในด้านระบบควบคุมมาประยุกต์ใช้ควบคุมการส่งข้อมูลภาพผ่านเครือข่าย ลักษณะการนำตัวควบคุมฟัซซี่ลอจิกไปใช้งานนั้นสามารถแสดงได้ดังรูปที่ 3.2



รูปที่ 3.2 ตัวควบคุมฟัซซี่ลอจิกในงานควบคุม

Process คือกระบวนการใด ๆ ที่ต้องการควบคุม โดยส่วนใหญ่ระบบเหล่านี้จะเป็นระบบที่มีความซับซ้อนสูงหรือเป็นระบบที่สามารถใช้ประสบการณ์ของผู้เชี่ยวชาญมาเป็นส่วนหนึ่งของการควบคุมได้ โครงสร้างภายในของตัวควบคุมฟัซซี่ลอจิกเป็นดังรูป 3.3



รูปที่ 3.3 โครงสร้างตัวควบคุมฟัซซี่ลอจิก

โครงสร้างตัวควบคุมฟัซซี่ลอจิกประกอบด้วย 4 ส่วนคือ

1) ฟัซซิไฟเออร์ (Fuzzifier) ทำหน้าที่แปลงค่าอินพุทจริงไปเป็นฟัซซี่เซต

2) ตัวอนุมานฟัซซี่ (Fuzzy Inference Engine) ทำหน้าที่นำค่าอินพุทไปประมวลผลร่วมกับฐานข้อมูลกฎฟัซซี่ (Fuzzy Rule Base) แล้วผลลัพธ์จากการประมวลผลส่งต่อไปยังดีฟัซซิไฟเออร์ (Defuzzifier)

3) ฐานข้อมูลกฎฟัซซี่ (Fuzzy Rule Base) เป็นฐานข้อมูลที่เก็บกฎการควบคุม ซึ่งฐานข้อมูลนี้อาจจะมาจากผู้เชี่ยวชาญ

4) ดีฟัซซิไฟเออร์ (Defuzzifier) ทำหน้าที่แปลงฟัซซี่เซตไปเป็นค่าจริงที่จะไปควบคุมระบบต่อไป

3.2 ฟัซซิไฟเออร์

ฟัซซิไฟเออร์นั้นคือการค่าจำนวนจริงใด ๆ ให้ไปอยู่ในรูปฟัซซี่เซต โดยทั่วไปฟัซซิไฟเออร์มีด้วยกัน 3 ชนิดคือ

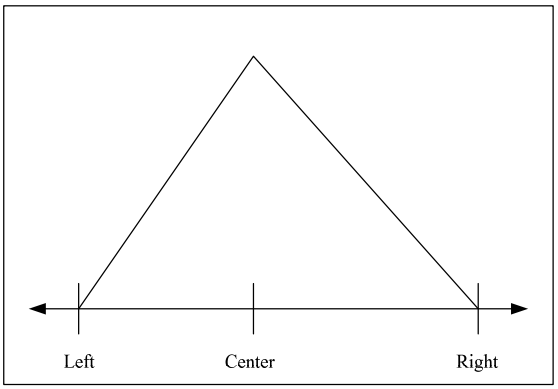
3.2.1 ฟัซซิไฟเออร์แบบเดี่ยว (Singleton Fuzzifier)

$$\mu_{A'}(x) = \begin{cases} 1 & \text{if } x=x^* \\ 0 & \text{Otherwise} \end{cases} \tag{3.1}$$

3.2.2 ฟัซซิไฟเออร์แบบเกาส์เซี่ยน (Gaussion Fuzzifier)

$$\mu_{A'}(x) = e^{-(\frac{x_1-x_1^*}{a_1})^2} * \ldots * e^{-(\frac{x_n-x_n^*}{a_n})^2} \tag{3.2}$$

3.2.3 ฟัซซิไฟเออร์แบบสามเหลี่ยม (Triangular Fuzzifier)



รูปที่ 3.4 ฟัซซิไฟเออร์แบบสามเหลี่ยม

ในกรณีนี้การคำนวณค่าเมมเบอร์ชิปฟังก์ชันจะเป็นการคำนวณแบบเชิงเส้น โดยมีเงื่อนไขดังนี้

$$\mu_{A'}(x) = \begin{cases} 0 & \text{if } x < \text{left} \\ 1 - \dfrac{|\text{center-x}|}{\dfrac{(\text{right-left})}{2}} & \text{if left}<x<\text{right} \\ 0 & \text{if } x > \text{right} \end{cases} \qquad (3.3)$$

3.3 ฐานข้อมูลกฎฟัซซี่

ฐานข้อมูลกฎฟัซซี่ประกอบได้ด้วยกฎการควบคุมที่อยู่ในรูปของ If-Then ที่มีรูปแบบดังนี้

Rule:

If (Fuzzy Condition 1) and … and (Fuzzy Condition N)

Then (Fuzzy Operation)

ตัวอย่างเช่น

If (Frame Loss) is (Low)

Then (Change on Frame Rate) is (Negative Low)

โดยทั่วไปแล้วกฎการควบคุมนี้จะได้มาจากผู้เชี่ยวชาญ ซึ่งมีความรู้เชิงลึกในเรื่องนั้น ๆ โดยซึ่งไม่จำเป็นต้องอาศัยรูปแบบทางคณิตศาสตร์ที่ซับซ้อน

3.4 ตัวอนุมานฟัซซี่

ทำหน้าที่ประมวลผลสัญญาณอินพุทที่เข้ามาที่อยู่ในรูปของเมมเบอร์ชิปฟังก์ชัน นำไปประมวลผลกับกฎในการควบคุมที่อยู่ในฐานข้อมูลกฎฟัซซี่ ซึ่งมีอยู่หลายรูปแบบด้วยกันแต่ที่เป็นที่นิยมมี 2 รูปแบบด้วยกันคือ แบบคูณและแบบน้อยที่สุดดังนี้

3.4.1 ตัวอนุมานฟัซซี่แบบคูณ (Product Inference Engine)

ความเป็นเมมเบอร์ชิปฟังก์ชันของแต่ละจะนำมาคูณกันดังสมการ

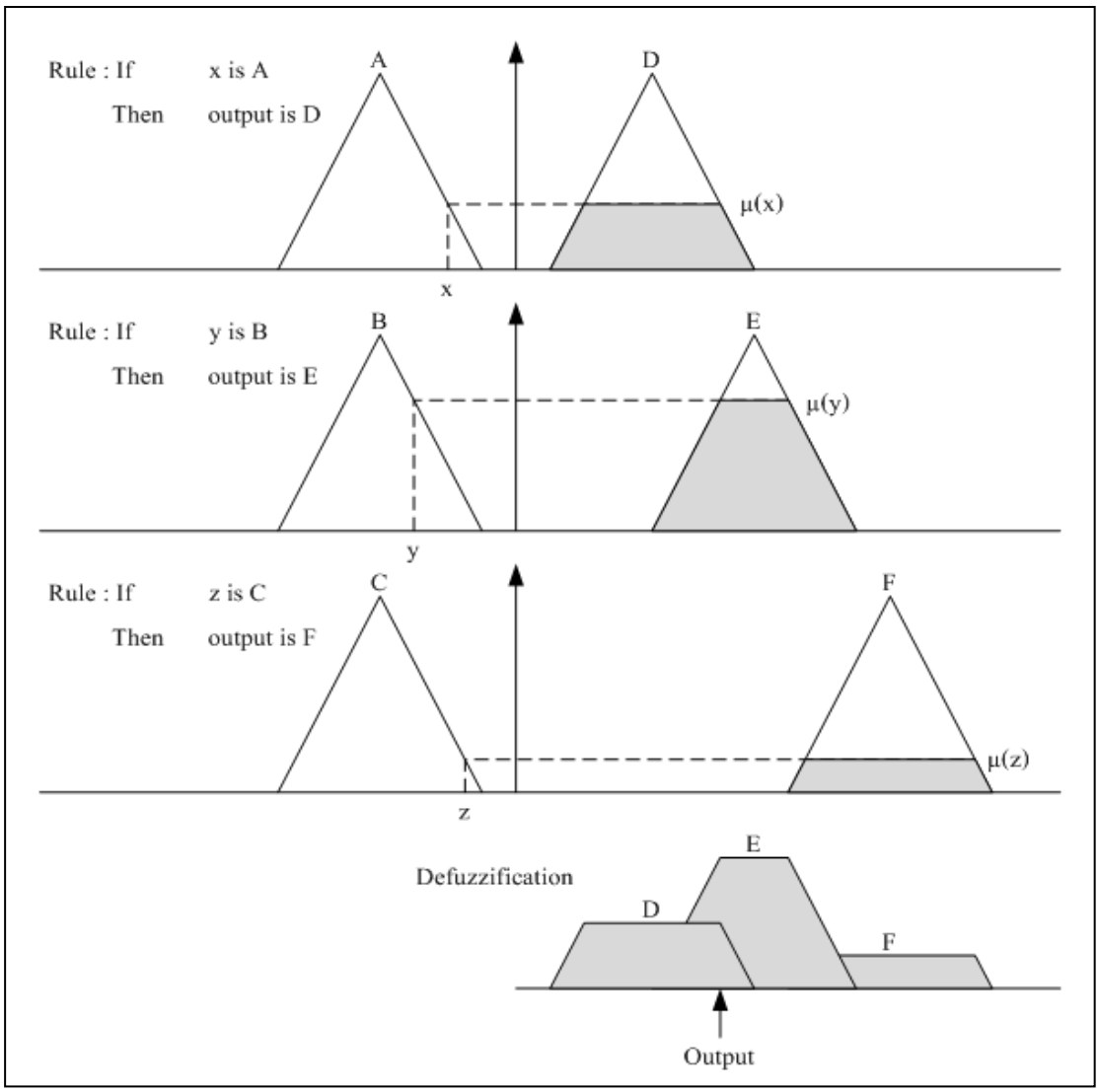
$$\mu_{B'}(y) = \max_{L=1}^{M} [\sup(\mu_{A'}(x) \prod_{i=1}^{n} \mu_{A_i^L}(x_i) \mu_{B^L}(y))] \quad (3.4)$$

3.4.2 ตัวอนุมานฟัซซี่แบบ Max-Min (Maximum-Minimum Inference Engine)

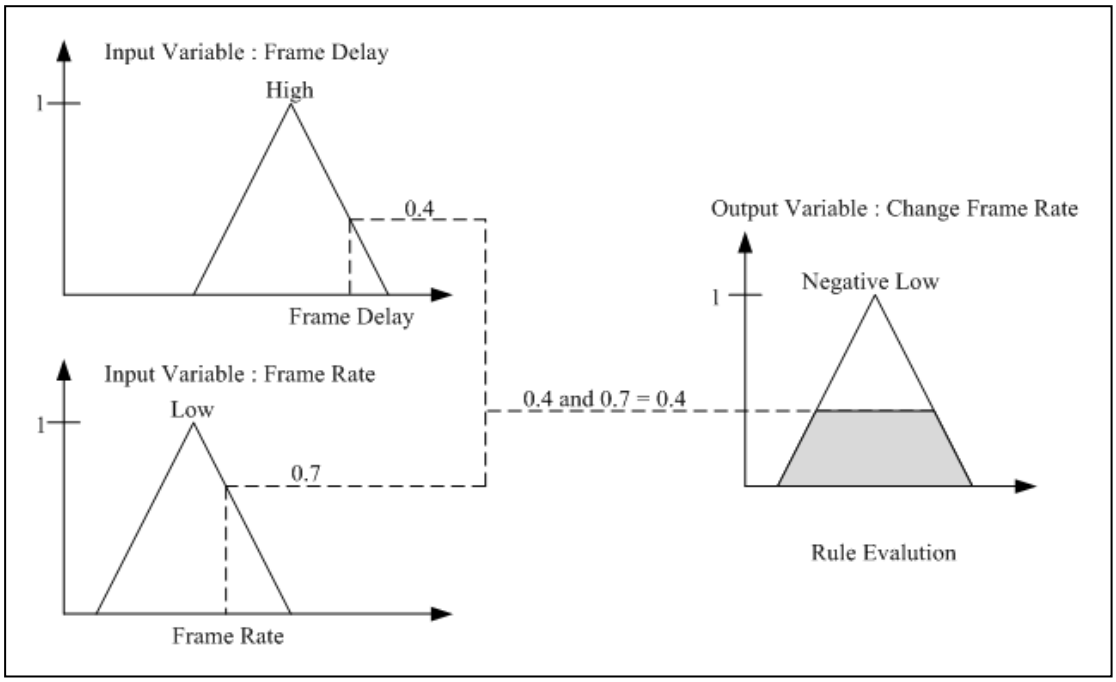
ในแบบนี้จะใช้ค่าน้อยที่สุดในการเป็นเมมเบอร์ชิปฟังก์ชันสามารถแสดงดังสมการ

$$\mu_{B'}(y) = \max_{L=1}^{M} [\sup_{X \in U} \min(\mu_{A'}(x), \mu_{A_1^L}(x_1), ..., \mu_{A_n^L}(x_n), \mu_{B^L}(y))] \quad (3.5)$$

สำหรับวิทยานิพนธ์นี้ได้เลือกใช้การอนุมานฟัซซี่แบบ Min-Max อันเนื่องมาจากมีความซับซ้อนน้อยกว่าและใช้พลังงานในการประมวลผลน้อยกว่าเมื่อเทียบกับการอนุมานแบบคูณ ซึ่งการทำงานในส่วนของการอนุมานสามารถแสดงได้ดังรูปที่ 3.5 และรูปที่ 3.6 ตามลำดับ



รูปที่ 3.5 แสดงขั้นตอนการประมวลผลในส่วนของการอนุมานแบบ Min-Max โดยในแต่ละกฎจะมีเงื่อนไขเพียงชุดเดียว



รูปที่ 3.6 แสดงขั้นตอนการประมวลผลในส่วนของการอนุมานแบบ Min-Max
โดยในแต่ละกฎจะมีเงื่อนไข 2 ชุด

3.5 ดีฟัซซิไฟเออร์

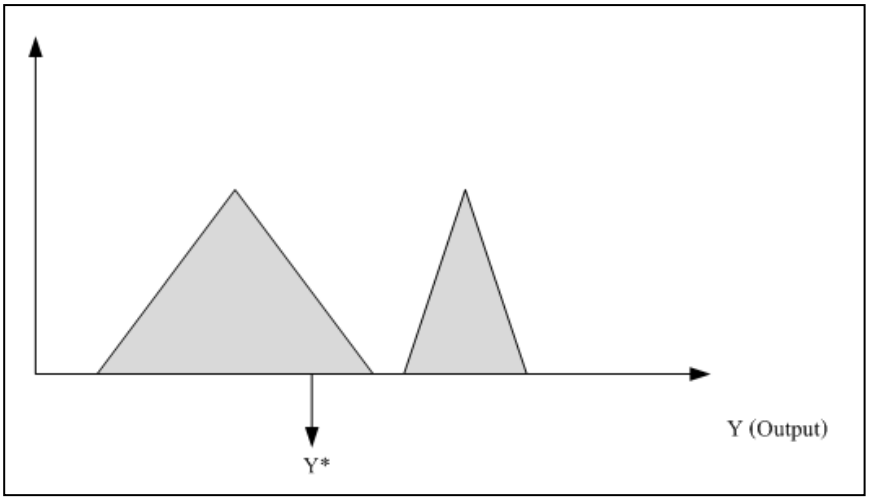
ดีฟัซซิไฟเออร์ ทำหน้าที่แปลงค่าฟัซซี่เซตให้เป็นค่าจริงซึ่งมีอยู่หลายรูปแบบ แต่ที่เป็นที่นิยมมีอยู่ด้วยกัน 2 รูปแบบคือ

3.5.1 ดีฟัซซิไฟเออร์แบบ Center of Gravity

เป็นการเฉลี่ยค่าน้ำหนักของพื้นที่ที่ได้จากการอนุมานในกฎฟัซซี่ต่าง ๆ ซึ่งสามารถแสดงได้ดังสมการและรูปที่ 3.7 ตามลำดับ

$$y^{*} = \frac{\int_{V} y \cdot \mu_{B'}(y)dy}{\int_{V} \mu_{B'}(y)dy} \qquad (3.6)$$

โดยที่ $\int_{V}$ คือการทำอินทีเกรตทั่ว ๆ ไป



รูปที่ 3.7 แสดงหลักการคำนวณดีฟัซซิไฟเออร์แบบ Center of Gravity

3.5.2 ดีฟัซซิไฟเออร์แบบ Center Average

เป็นการเฉลี่ยค่าน้ำหนักของค่าที่ได้จากการอนุมานในกฎฟัซซี่ต่าง ๆ ซึ่งสามารถแสดงได้ดังสมการ

$$y^* = \frac{\sum_{k=1}^{M} y^{-k} w_k}{\sum_{k=1}^{M} w_k} \qquad (3.7)$$

ซึ่งรูปแบบนี้เหมาะสมกับงานระบบฝังตัวเป็นอย่างยิ่ง เนื่องจากเป็นคณิตศาสตร์ที่มีความซับซ้อนน้อยกว่าเมื่อเทียบกับ Center of Gravity

บทที่ 4
ฮาร์ดแวร์และซอฟต์แวร์

4.1 บทนำ

โครงสร้างของฮาร์ดแวร์ที่ใช้ในวิทยานิพนธ์นี้ประกอบไปด้วย (1) บอร์ดประมวลผลตระกูล ARM7 (2) โมดูลกล้อง และ (3) โปรแกรม TCP/IP Stack ที่เป็นฟรีแวร์ (Freeware) โดยมีรายละเอียดดังนี้

4.2 บอร์ดประมวลผลตระกูล ARM7

บอร์ด ARM โพรเซสเซอร์ซึ่งเป็นตัวประมวลผลแบบ 32 บิต เป็นโพรเซสเซอร์ที่มีสมรรถนะสูง ราคาไม่แพง โดยลักษณะของบอร์ดแสดงดังรูปที่ 4.1

รูปที่ 4.1 บอร์ดประมวลผลตระกูล ARM7

ลักษณะเด่นของบอร์ดพัฒนามีดังนี้

1) เป็นโปรเซสเซอร์ตระกูล ARM7 TDMI-S เบอร์ LPC2368 มีหน่วยความจำโปรแกรมขนาด 512 KB RAM 58 KB 10Bit A/D 10Bit D/A RTP PWM CAN

2) ใช้ Crystal 12 MHz ที่สามารถทำ Phase-Locked Loop ทำงานได้สูงสุดที่ความถี่ 72 MHz

3) สามารถโปรแกรมแบบ In-System Programmer (ISP) ผ่านทาง RS-232

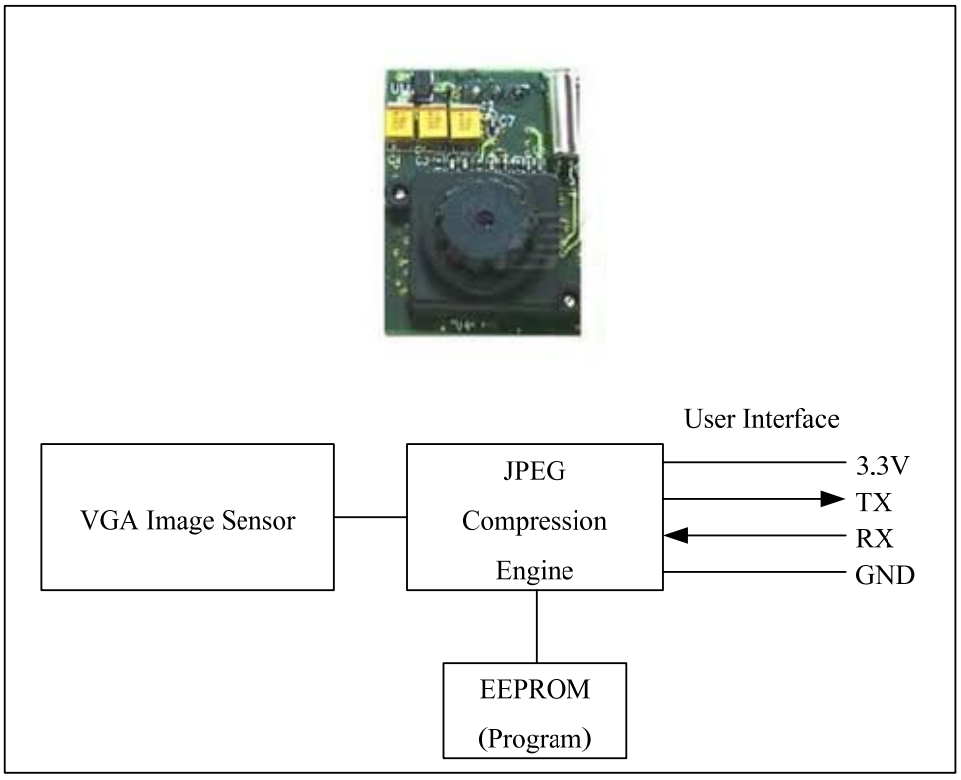
4) มี USB 2.0 แบบ Full Speed

5) มี Ethernet LAN 10/100 Mbps

6) มีวงจรเชื่อมต่อการ์ดหน่วยความจำแบบ SD หรือ MMC พร้อมขั้วต่อ

4.3 โมดูลกล้อง

เป็นอุปกรณ์ที่ทำหน้าที่รับภาพที่เป็นข้อมูลดิบแปลงเป็นข้อมูลดิจิตอล ในวิทยานิพนธ์นี้ได้ใช้ CMOS Image Camera Module C328R ของบริษัท Comedia ซึ่งโครงสร้างของกล้องรุ่นนี้เป็นดังนี้



รูปที่ 4.2 โครงสร้างของโมดูลกล้อง C328R

ซึ่งลักษณะเด่นของอุปกรณ์ตัวนี้เป็นดังนี้

1) ความละเอียดภาพแบบ VGA สามารถทำ Down Sample ไปเป็น QVGA หรือ CIF
2) ทำงานที่แรงดันไฟเลี้ยง 3.3 โวลท์
3) กินไฟที่ระดับ 60mA
4) มีชุดคำสั่งที่ง่ายต่อการควบคุมตัวโมดูล
5) ความเร็วในการสื่อสารแบบ UART สูงสุดอยู่ที่ 115200 bps
6) ตรวจจับค่า Baud Rate ได้โดยอัตโนมัติ
7) โหมดประหยัดพลังงาน
8) มีเลนส์ให้เลือกหลายรูปแบบ

4.4 TCP/IP Stack

เป็นซอฟท์แวร์ที่ทำหน้าที่ด้าน TCP/IP Stack โดยในวิทยานิพนธ์นี้ได้ใช้โปรแกรม uIP ซึ่งเป็นโปรแกรมที่มีขนาดเล็ก ใช้ทรัพยากรของระบบน้อยและเป็นโปรแกรมที่แจกซอร์ซโค้ดโดยไม่มีค่าใช้จ่าย โดยที่โปรแกรมนี้ได้ถูกพัฒนาให้สนับสนุนการทำงานบนตัวประมวลผลหลายตระกูลด้วยกันเช่น ARM AVR 8051 PIC เป็นต้น

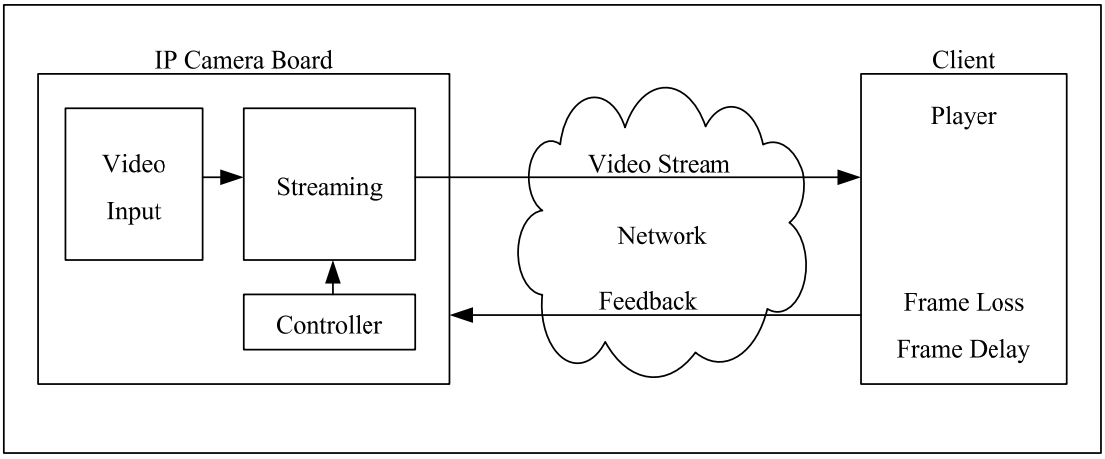
uIP พัฒนามาจาก Adam Dunkel ซึ่งเขาเป็นกลุ่ม Networked Embedded Systems แห่ง Swedish Institute of Computer Science ซึ่งลักษณะเด่นของโปรแกรมตัวนี้คือ

1) โปรแกรมมีขนาดเล็ก
2) ใช้หน่วยความจำน้อย
3) สนับสนุนโปรโตคอล ARP SLIP IP UDP ICMP (ping) และ TCP โปรโตคอล

บทที่ 5
การออกแบบระบบทดสอบการทำงานของกล้องไอพี

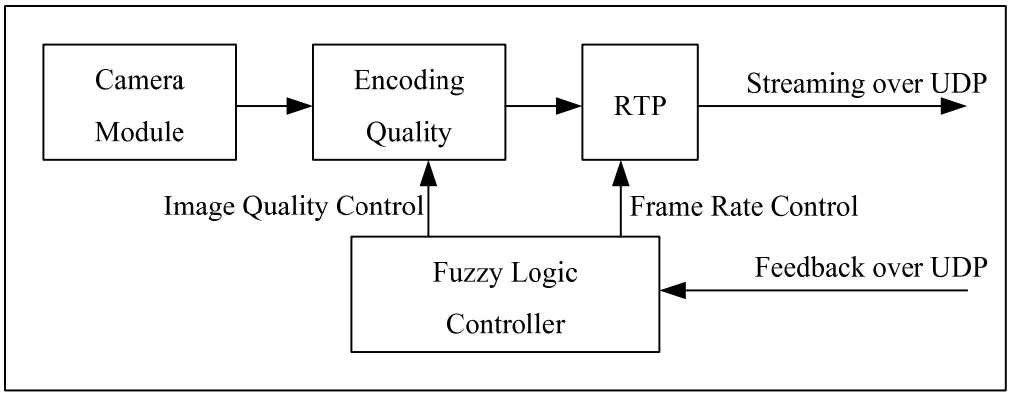
5.1 โครงสร้างการทำงานของระบบทดสอบ

กล้องไอพีที่ทำการออกแบบใช้บอร์ดที่มีตัวประมวลผลตระกูล ARM7 เชื่อมต่อกับกล้องแบบ UART ที่ความเร็ว 115200 baud ในส่วนของโปรแกรมที่ทำงานบนกล้องไอพีได้ใช้ uIP ซึ่งเป็น TCP/IP Stack ซึ่งแจกโปรแกรมต้นฉบับฟรีโดยไม่มีค่าใช้จ่าย สำหรับตัวคอมไพเลอร์ที่ทำหน้าที่แปลงโปรแกรมต้นฉบับให้เป็นไบนารีไฟล์สำหรับดาวโหลดโปรแกรมลงไปที่กล้องไอพีนั้นได้ใช้คอมไพเลอร์สำหรับตัวประมวลผล ARM ของบริษัท Keil แผนภาพการทำงานเป็นดังรูปที่ 5.1



รูปที่ 5.1 แสดงโครงสร้างของตัวควบคุมฟัซซี่ลอจิกที่ใช้ควบคุมการส่งข้อมูลภาพ

ภายในตัวกล้องไอพีนั้นจะแบ่งออกเป็นส่วนหลัก ๆ ได้ 4 ส่วนคือ (1) การอ่านข้อมูลจากกล้อง (2) การเข้ารหัสภาพ (3) ตัวควบคุมฟัซซี่ และ (4) การส่งข้อมูลออกไปให้ผู้ใช้ ตัวควบคุมฟัซซี่ลอจิกจะรับข้อมูลป้อนกลับจากผู้ใช้ทาง UDP ซึ่งค่าเหล่านี้คือ Frame Loss และค่า Frame Delay จากนั้นจะไปประมวลผลเทียบกับกฎฟัซซี่ซึ่งเอาท์พุทที่ได้จะไปควบคุม 2 ส่วนคือ ควบคุมการเข้ารหัสภาพและควบคุมอัตราการส่งภาพ ในการส่งภาพออกไปนั้นใช้ RTP เป็นโปรโตคอลในการส่งข้อมูล ซึ่งสามารถแสดงได้ดังรูปที่ 5.2



รูปที่ 5.2 แสดงโครงสร้างภายในตัวกล้องไอพี

5.2 การเชื่อมต่อกับโมดูลกล้อง

การเชื่อมต่อกับโมดูลกล้องเป็นการเชื่อมต่อแบบ UART โดยที่ความเร็วการการเชื่อมต่อคือ 115200 bps โมดูลกล้องรุ่นนี้ให้ภาพออกมาในรูปแบบ JPEG อยู่แล้วจึงลดขั้นตอนในการแปลงข้อมูลดิบไปเป็น JPEG หลักการในการติดต่อกับกล้องเบื้องต้นคือ Initialize และอ่านรูปภาพจากกล้องโดยที่การอ่านรูปภาพจากกล้องในรอบต่อ ๆ ไปไม่จำเป็นจะต้องทำการ Initialize อีก ชุดคำสั่งในการใช้งานทั้งหมดได้แสดงไว้ในตารางที่ 5.1

ตารางที่ 5.1 แสดงชุดคำสั่งที่ใช้กับโมดูลกล้อง

Command	ID Number	Parameter 1	Parameter 2	Parameter 3	Parameter 4
Initial	AA01h	00h	Color Type	RAW Resolution (Still image only)	JPEG Resolution
Get Picture	AA04h	Picture Type	00h	00h	00h
Snapshot	AA05h	Snapshot Type	Skip Frame Low Byte	Skip Frame High Byte	00h
Set Package Size	AA06h	08h	Package Size Low Byte	Package Size High Byte	00h
Set Baudrate	AA07h	1st Divider	2nd Divider	00h	00h
Reset	AA08h	Reset Type	00h	00h	xxh*
Power Off	AA09h	00h	00h	00h	00h
Data	AA0Ah	Data Type	Length Byte 0	Length Byte 1	Length Byte 2
SYNC	AA0Dh	00h	00h	00h	00h

ตารางที่ 5.1 แสดงชุดคำสั่งที่ใช้กับโมดูลกล้อง (ต่อ)

Command	ID Number	Parameter 1	Parameter 2	Parameter 3	Parameter 4
ACK	AA0Eh	Command ID	ACK Counter	00h / Package ID Byte 0	00h / Package ID Byte 1
NAK	AA0Fh	00h	NAK Counter	Error Number	00h
Light Frequency	AA13h	Frequency Type	00h	00h	00h

ในการอ่านข้อมูลภาพจากโมดูลกล้องนั้นจะกระทำตลอดเวลาด้วยความเร็วสูงสุด เพื่อให้ได้ความต่อเนื่องของภาพดีที่สุด ข้อมูลภาพนี้จะเก็บอยู่ในหน่วยความจำของบอร์ดประมวลผล ARM7 ซึ่งข้อมูลภาพนี้จะถูกใช้งานโดยฟังก์ชันการเข้ารหัสภาพอีกครั้งหนึ่งเพื่อลดขนาดภาพให้เหมาะสมกับแบนด์วิดท์ต่อไป ในการอ่านข้อมูลภาพแต่ละภาพมีขั้นตอนดังแสดงในรูปที่ 5.3

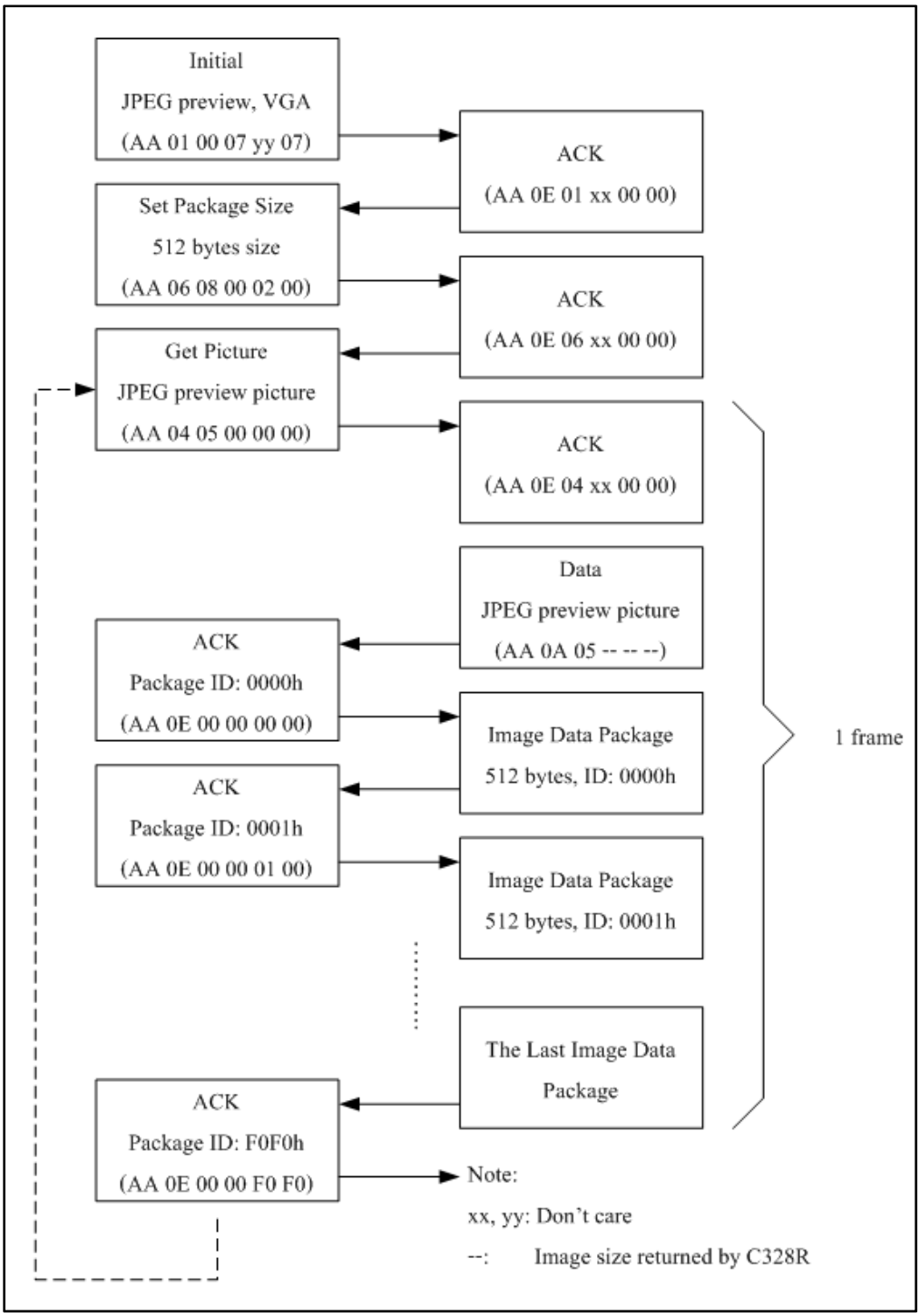
5.3 การเข้ารหัสสัญญาณภาพ (Video Encode)

ข้อมูลภาพที่ได้จากโมดูลกล้องนั้นเป็นข้อมูลในรูปแบบ JPEG อยู่แล้ว แต่ที่ต้องมีตัวเข้ารหัสภาพอีกครั้งหนึ่งก็เพื่อที่จะลดคุณภาพของภาพเพื่อให้ภาพมีขนาดเล็กลงเพื่อใช้ส่งในขณะที่แบนด์วิดท์คับคั่ง โดยที่การเข้ารหัสนี้เป็นฟังก์ชันที่อยู่ในรูปของภาษาซีซึ่งจะทำการคอมไพล์ร่วมกับ uIP ที่ทำหน้าที่ TCP/IP Stack ซึ่งรายละเอียดของฟังก์ชันนี้ได้แสดงไว้ในภาคผนวก ค.

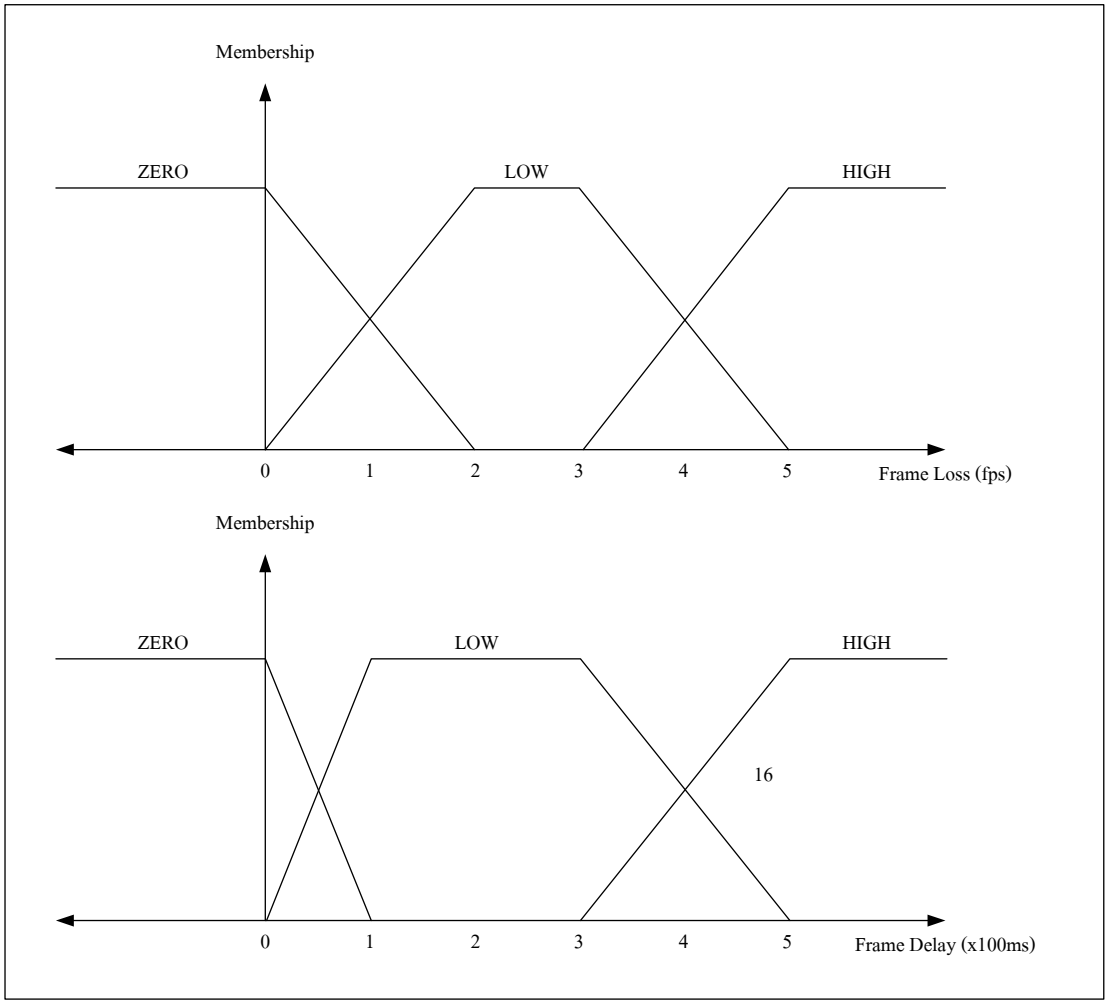
5.4 ตัวควบคุมฟัซซี่ลอจิก

5.4.1 ฟัซซิไฟเออร์ (Fuzzifier)

ในวิทยานิพนธ์นี้ฟัซซิไฟเออร์จะทำหน้าที่แปลงค่าสัญญาณจริง ในที่นี้คือ Frame Loss และ Frame Delay ให้ไปอยู่ในรูปแบบของฟัซซี่เซต ค่า Frame Loss และค่า Frame Delay ที่เข้ามาจะถูกแปลงให้อยู่ในรูปแบบของค่าเมมเบอร์ชิปซึ่งมีค่าสูงสุดไม่เกิน 1 ซึ่งค่าเมมเบอร์ชิปนี้จะถูกส่งต่อไปยังตัวอนุมานฟัซซี่ต่อไป ค่าเมมเบอร์ชิปของ Frame Loss และ Frame Delay แสดงดังรูปที่ 5.4

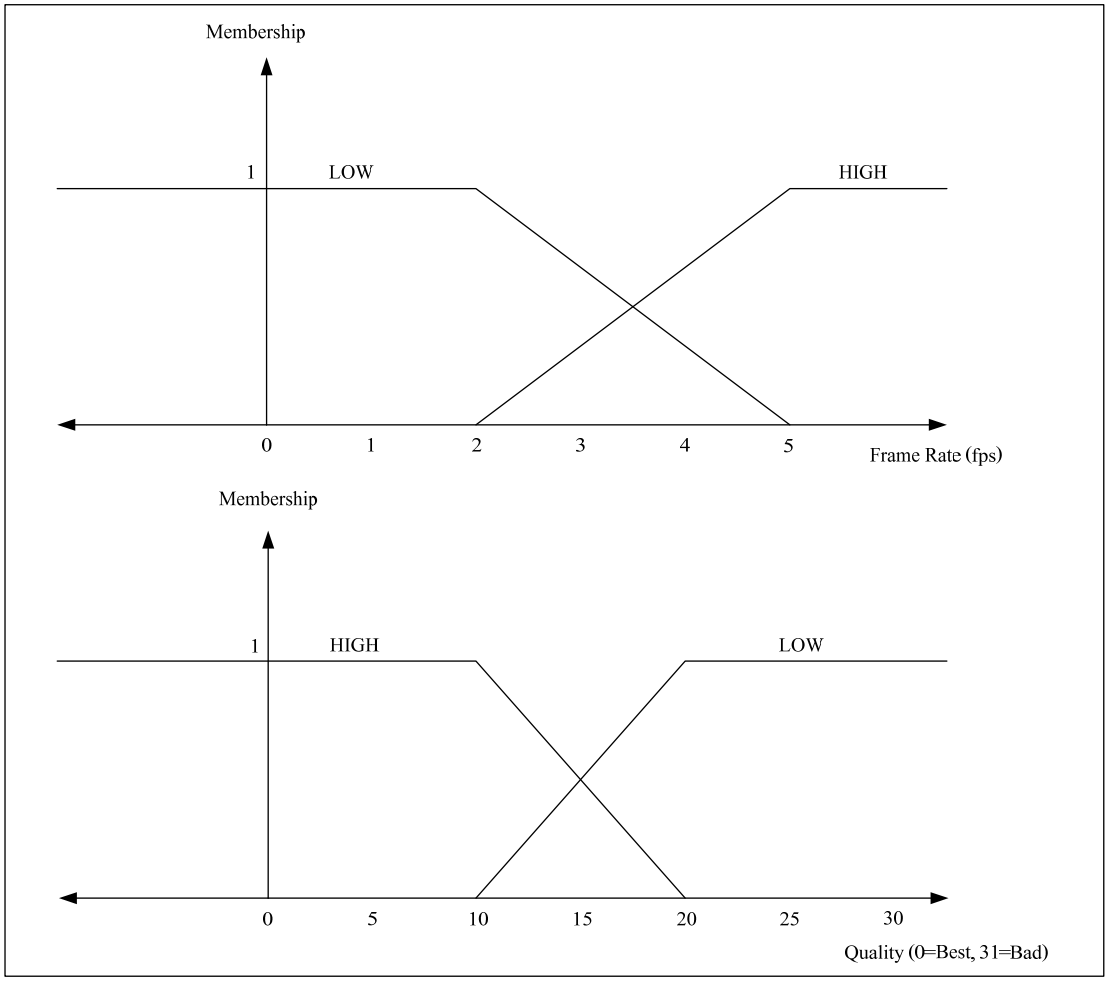


รูปที่ 5.3 แสดงขั้นตอนการอ่านข้อมูลภาพ 1 ภาพ



รูปที่ 5.4 แสดงเมมเบอร์ชิปฟังก์ชันของ Frame Loss และ Frame Delay

ค่าอัตราการส่งภาพและคุณภาพของภาพก็สามารถนำมาเป็นส่วนหนึ่งของกฎในการควบคุมได้เช่นกันเพื่อให้รูปแบบในการควบคุมมีความหลากหลายมากยิ่งขึ้น เมมเบอร์ชิปฟังก์ชันของอัตราการส่งภาพและคุณภาพของภาพเป็นดังรูปที่ 5.5



รูปที่ 5.5 แสดงเมมเบอร์ชิปฟังก์ชันของอัตราการส่งภาพและคุณภาพของภาพ

5.4.2 กฎฟัซซี่ (Fuzzy Rule Base)

กฎฟัซซี่นั้นจะมาจากประสบการณ์ของผู้เชี่ยวชาญที่สามารถแสดงออกมาในรูปของถ้า-แล้ว (If-Then) ในวิทยานิพนธ์นี้ได้สร้างกฎในการควบคุม 3 รูปแบบคือ (1) การควบคุมอัตราการส่งภาพ (2) ควบคุมคุณภาพของภาพ และ (3) ผสมทั้ง 2 แบบเข้าด้วยกัน

5.4.2.1 กฎในการควบคุมอัตราการส่งภาพ

Rule:

If (Frame Delay) is (High)

Then (Change Frame Rate) is (Negative High)

Rule:

If (Frame Delay) is (Low)

Then (Change Frame Rate) is (Negative Low)

Rule:

If (Frame Loss) is (High)

Then (Change Frame Rate) is (Negative High)

Rule:

If (Frame Loss) is (Low)

Then (Change Frame Rate) is (Negative Low)

Rule:

If (Frame Delay) is (Zero) and (Frame Loss) is (Zero)

Then (Change Frame Rate) is (Positive Low)

5.4.2.2 กฎในการควบคุมคุณภาพของภาพ

Rule:

If (Frame Delay) is (High)

Then (Change Quality) is (Positive High)

Rule:

If (Frame Delay) is (Low)

Then (Change Quality) is (Positive Low)

Rule:

If (Frame Loss) is (High)

Then (Change Quality) is (Positive High)

Rule:

If (Frame Loss) is (Low)

Then (Change Quality) is (Positive Low)

Rule:

If (Frame Delay) is (Zero) and (Frame Loss) is (Zero)

Then (Change Quality) is (Negative Low)

5.4.2.3 กฎในการควบคุมแบบผสม

Rule:

If (Frame Delay) is (High) and (Frame Rate) is (High)

Then (Change Frame Rate) is (Negative High)

Rule:

If (Frame Delay) is (High) and (Frame Rate) is (Low)

Then (Change Quality) is (Positive High)

Rule:

If (Frame Delay) is (Low) and (Frame Rate) is (High)

Then (Change Frame Rate) is (Negative Low)

Rule:

If (Frame Delay) is (Low) and (Frame Rate) is (Low)

Then (Change Quality) is (Positive Low)

Rule:

If (Frame Loss) is (High) and (Frame Rate) is (High)

Then (Change Frame Rate) is (Negative High)

Rule:

If (Frame Loss) is (High) and (Frame Rate) is (Low)

Then (Change Quality) is (Positive High)

Rule:

If (Frame Loss) is (Low) and (Frame Rate) is (High)

Then (Change Frame Rate) is (Negative Low)

Rule:

If (Frame Loss) is (Low) and (Frame Rate) is (Low)

Then (Change Quality) is (Positive Low)

Rule:

If (Frame Delay) and (Frame Loss) is (Zero) and
(Frame Rate) is (Low) and (Quality) is (Low)

Then (Change Quality) is (Negative Low)

Rule:

If (Frame Delay) and (Frame Loss) is (Zero) and
(Frame Rate) is (Low) and (Quality) is (High)

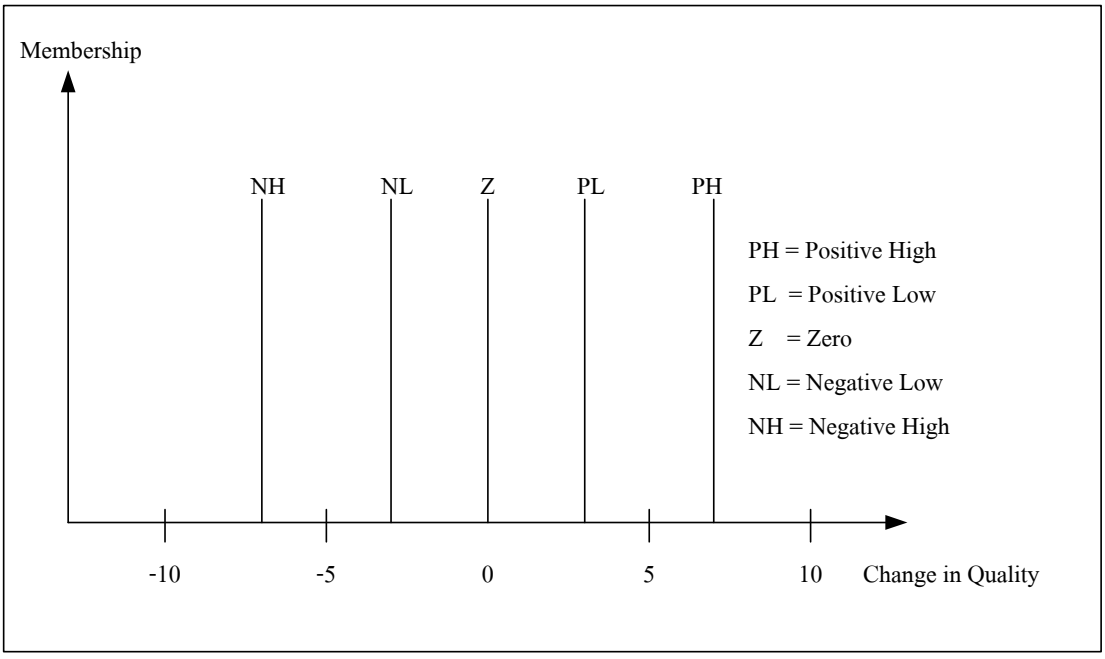
Then (Change Frame Rate) is (Positive Low)

5.4.3 ตัวอนุมานฟัซซี่ (Fuzzy Inference Engine)

ได้เลือกใช้การอนุมานแบบ Min-Max (ดังแสดงไว้แล้วในสมการที่ 3.5) เนื่องจากการอนุมานแบบนี้ใช้พลังงานในการประมวลผลน้อยและมีขนาดโปรแกรมที่ได้จากการคอมไพล์มีขนาดเล็กเหมาะกับระบบฝังตัวที่มีเนื้อที่ให้เขียนโปรแกรมน้อย

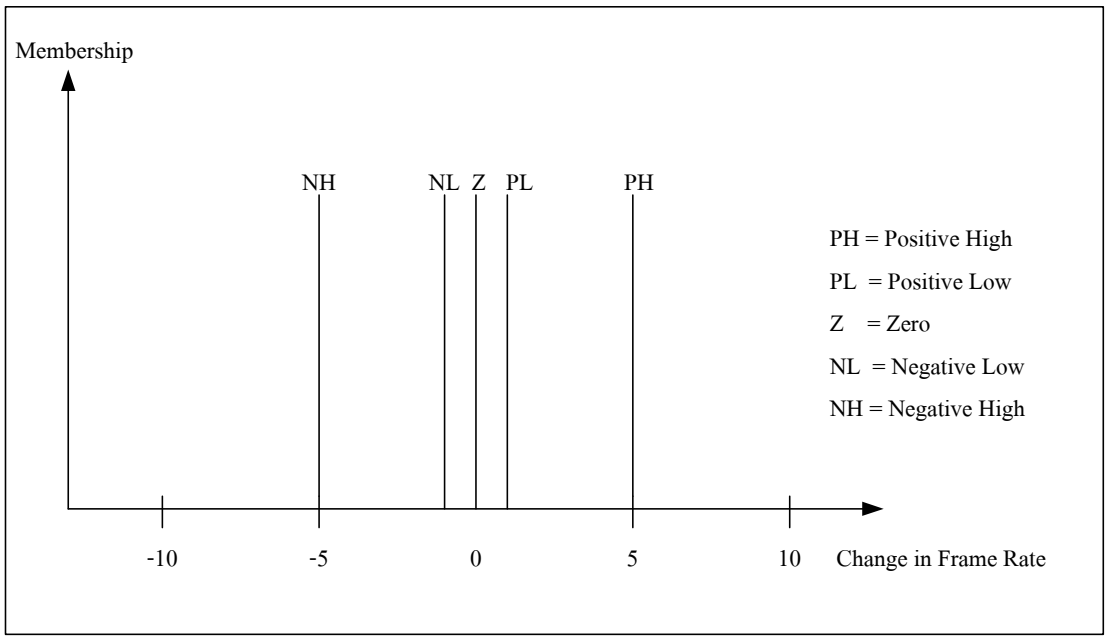
5.4.4 ดีฟัซซิไฟเออร์ (Defuzzifier)

ในส่วนของดีฟัซซิไฟเออร์จะมีการนิยามในส่วนของการเปลี่ยนแปลงคุณภาพและอัตราการส่งข้อมูลภาพ ซึ่งในส่วนของการเปลี่ยนแปลงคุณภาพได้มีการนิยามเมมเบอร์ชิปฟังก์ชันไว้ 5 ระดับคือ การเปลี่ยนแปลงแบบ Positive High (PH) Positive Low (PL) Zero (Z) Negative Low (NL) Negative High (NH) ดังแสดงในรูปที่ 5.6



รูปที่ 5.6 เมมเบอร์ชิปฟังก์ชันของการเปลี่ยนแปลงคุณภาพของภาพ

ในส่วนของการเปลี่ยนแปลงอัตราการส่งภาพได้ทำการนิยามเมมเบอร์ชิปฟังก์ชันไว้ 5 ระดับเช่นกันได้แก่ Positive High (PH) Positive Low (PL) Zero (Z) Negative Low (NL) Negative High (NH) แสดงดังรูปที่ 5.7



รูปที่ 5.7 เมมเบอร์ชิปฟังก์ชันของการเปลี่ยนแปลงอัตราการส่งภาพ

5.5 โปรแกรมแสดงภาพที่ผู้ใช้งาน

โปรแกรมรับภาพและแสดงภาพไปยังผู้ใช้ทำหน้าที่รับภาพที่ส่งมาจากกล้องไอพีที่ส่งข้อมูลมาทาง RTP ซึ่งโปรแกรมนี้จะดูค่าไทม์แสตมป์และหมายเลขลำดับของเฟรม RTP เพื่อประเมินค่า Frame Loss และ Frame Delay แล้วทำการส่งค่าเหล่านี้กลับไปยังกล้องไอพีผ่านทาง UDP โปรแกรมรับภาพนี้พัฒนาขึ้นจาก Borland Delphi 7 ซึ่งมีคอมโพเนนท์ในด้านเน็ตเวิร์คให้ใช้มากมาย รูปของโปรแกรมแสดงดังรูปที่ 5.8 โดยโครงสร้างของโปรแกรมประกอบไปด้วย

5.5.1 การปรับเวลาให้ตรงกัน (Time Synchronization)

ในช่วงเริ่มต้นของการรับข้อมูลภาพโปรแกรมแสดงภาพจะทำการสร้างฐานเวลาขึ้นมาเพื่อที่จะเป็นตัวเปรียบเทียบกับค่าไทม์แสตมป์ของข้อมูลภาพที่วิ่งเข้ามาซึ่งจะทำให้สามารถบอกถึงค่า Frame Loss และค่า Frame Delay ได้ เนื่องจากโปรแกรมที่ทำงานบนระบบปฏิบัติการต่าง ๆ ในปัจจุบันนี้สามารถเข้าถึงค่าฐานเวลาของระบบได้อยู่แล้ว แต่ค่าฐานเวลาของผู้ส่งและผู้รับอาจจะมี

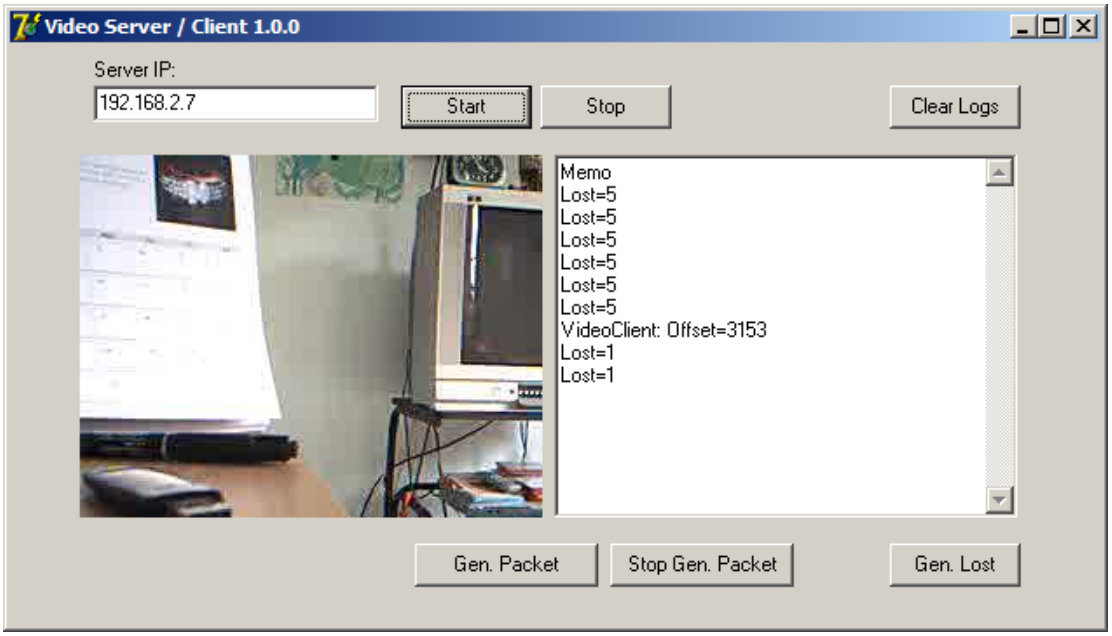
ค่าต่างกัน เช่น ตำแหน่งของผู้ส่งและผู้รับอยู่คนละช่วงเวลากัน (Time Zone) เป็นต้น จึงมีความจำเป็นที่จะต้องรู้ค่า Time Offset ระหว่างผู้ส่งกับผู้รับซึ่งจะเป็นไปตามสมการ

$$\text{server time base} = \text{client time base} + \text{time offset} \tag{5.1}$$

แพ็กเกจข้อมูลภาพที่ถูกส่งมาจากกล้องไอพีโดยใช้ RTP นั้นจะมีการประทับค่าไทม์แสตมป์ที่ส่วนหัวของ RTP แพ็กเกจ เมื่อข้อมูลนี้มาถึงโปรแกรมแสดงภาพย่อมสามารถคำนวณค่า Time Offset นี้ได้ แต่เพื่อให้ข้อมูลค่า Time Offset นี้มีความน่าเชื่อถือมากยิ่งขึ้นจึงมีความจำเป็นต้องทำการหาค่าเฉลี่ยจากหลายแพ็คเกจแทนที่จะใช้ข้อมูลจากแพ็คเกจเดียว โดยการหาค่าเฉลี่ยจากสมการ

$$\text{time offset} = \frac{\text{time offset}_1 + \ldots + \text{time offset}_n}{n} \tag{5.2}$$

โดยที่ n คือ จำนวนแพ็คเกจที่เข้ามา



รูปที่ 5.8 โปรแกรมแสดงภาพฝั่งผู้ใช้งานที่พัฒนาโดยใช้โปรแกรม Borland Delphi 7

5.5.2 การพิจารณาค่า Frame Loss และค่า Frame Delay

ในการหาค่า Frame Loss นั้นสามารถหาได้จาก Sequence Number ที่อยู่ในเฮดเดอร์ของ RTP ในแต่ละแพ็คเกจของ RTP อาจจะประกอบด้วยข้อมูลภาพหลาย ๆ ภาพ แต่ละแพ็คเกจของ RTP จึงจำเป็นต้องมีค่า Sequence Number นี้เพื่อบอกถึงลำดับของภาพที่จะไปแสดงผลบนหน้าจอได้อย่างถูกต้อง หากมีค่าลำดับนี้หายไปหรือมาถึงโปรแกรมแสดงภาพช้ามากเกินไปก็จะถือว่า Frame Loss ได้เกิดขึ้น

ในส่วนของ Frame Delay เมื่อได้ข้อมูล Time Offset แล้วก็จะนำข้อมูลนี้ไปใช้ในการหาค่า Frame Delay ของแต่ละแพ็คเกจที่วิ่งเข้ามาที่โปรแกรมแสดงภาพดังสมการ

packet frame delay = RTP timestamp - time offset (5.2)

ซึ่งหากค่า Packet Frame Delay นี้มีค่ามากเกินกว่าค่าที่กำหนด ก็จะถือว่าเกิด Frame Delay ขึ้น แต่ก็เป็นไปได้ที่ค่า Packet Frame Delay นี้จะมีค่าน้อยกว่า 0 ซึ่งมีความหมายว่า RTP แพ็คเกจเข้ามาถึงก่อนกำหนดซึ่งสามารถเป็นไปได้เนื่องจากค่า Time Offset ที่คำนวณในช่วงแรก ๆ นั้นอาจจะเป็นช่วงที่เครือข่ายมีความคับคั่งในการใช้งานมากกว่าในขณะปัจจุบัน ซึ่งในกรณีนี้ก็จะถือว่าไม่มี Frame Delay เกิดขึ้น

5.5.3 ข้อมูลป้อนกลับ (Feedback Information)

ค่า Frame Loss และค่า Frame Delay ที่คำนวณได้จะถูกส่งกลับไปยังกล้องไอพีผ่านทาง UDP โดยส่งเป็นคาบเวลาตามที่ตั้งไว้เช่น ทุก ๆ 5 วินาที เป็นต้น เมื่อข้อมูลได้ถูกส่งออกไปเรียบร้อยแล้วก็จะทำการรีเซตค่า Frame Loss และ Frame Delay เพื่อเริ่มนับใหม่

บทที่ 6

ผลการทดลอง

ในการทดลองการส่งข้อมูลผ่านเครือข่ายนั้นได้มีการทดสอบในสภาพแวดล้อม 2 รูปแบบ คือ ส่งข้อมูลภาพผ่าน LAN (Ethernet) และส่งข้อมูลผ่านเครือข่าย GPRS

6.1 ส่งข้อมูลภาพผ่านเครือข่าย LAN (Ethernet)

ในการทดลองนี้ได้ให้กล้องไอพีมีที่ตั้งอยู่ในวง LAN เดียวกับผู้รับชม ซึ่งโดยทั่วไปความเร็วใน LAN มีค่าสูงกว่าแบนด์วิดท์ที่ต้องใช้ในการส่งข้อมูล การทดลองนี้เป็นการทดสอบกฎควบคุมฟัซซี่ลอจิกในช่วงที่ไม่มี Frame Loss และ Frame Delay ตัวอย่างของกฎฟัซซี่คือ

Rule:

If (Frame Delay) is (Zero) and (Frame Loss) is (Zero)

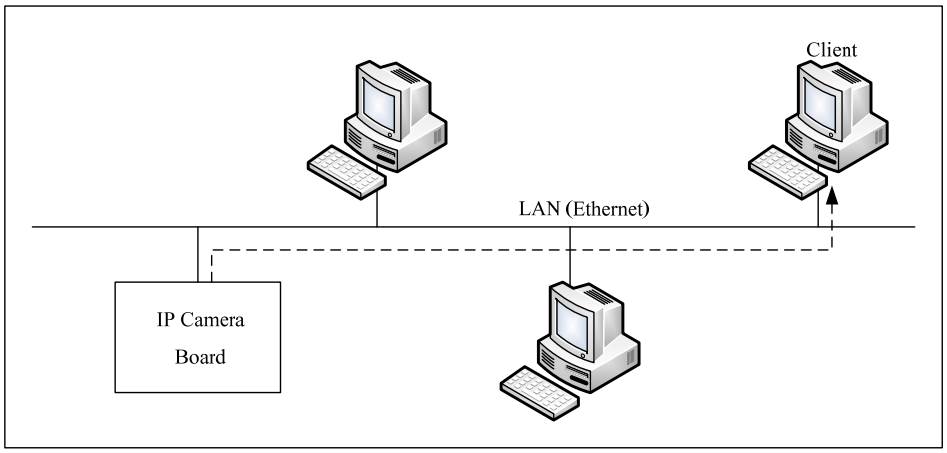
Then (Change Quality) is (Negative Low)

Rule:

If (Frame Delay) is (Zero) and (Frame Loss) is (Zero)

Then (Change Frame Rate) is (Positive Low)

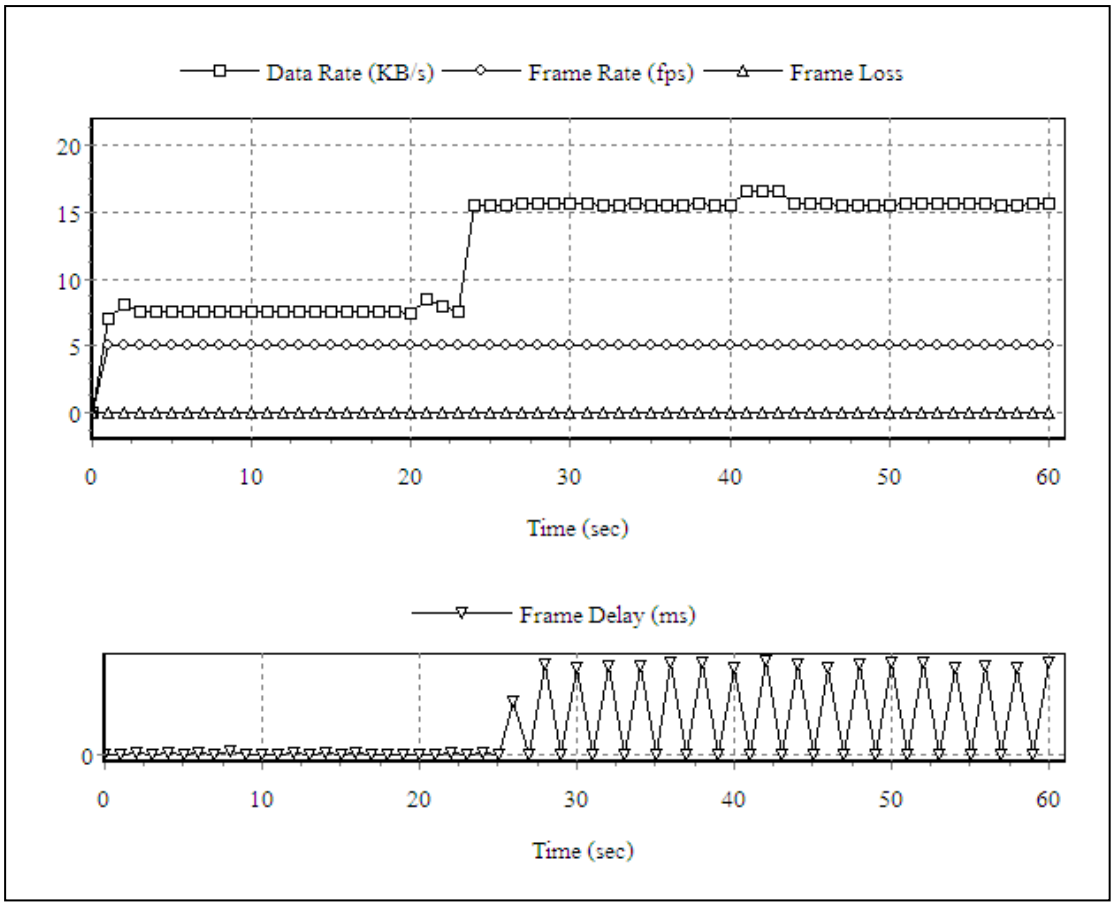
ซึ่งจะแบ่งการทดลองออกเป็น 2 ส่วนคือ การควบคุมคุณภาพของภาพแต่คงค่าอัตราการส่งภาพ และควบคุมอัตราการส่งภาพแต่คงค่าคุณภาพของภาพ



รูปที่ 6.1 แสดงแผนภาพการทดลองส่งข้อมูลผ่านเครือข่าย LAN

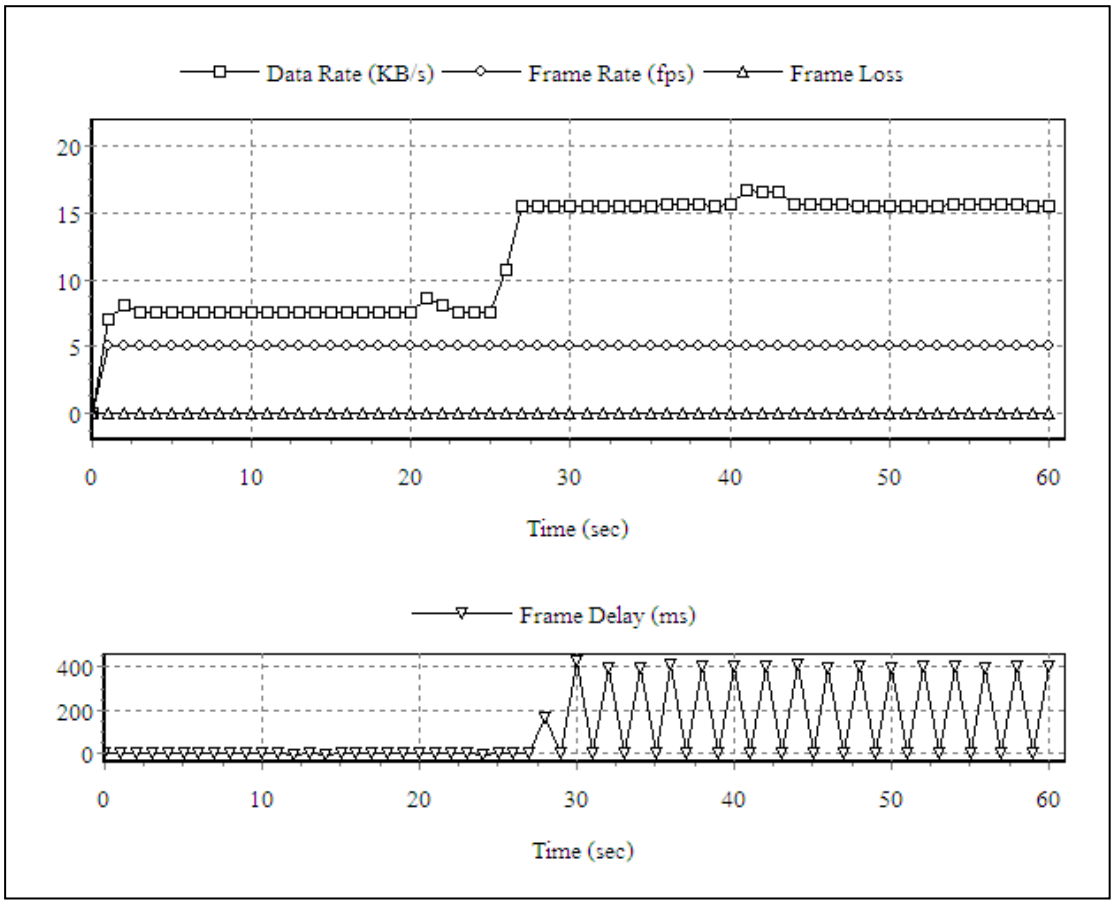
6.1.1 การควบคุมคุณภาพของภาพ

ในการทดลองนี้จะทำการคงค่าอัตราการส่งภาพ ตัวควบคุมฟัซซี่ลอจิกจะปรับเปลี่ยนเฉพาะคุณภาพของภาพเพียงเท่านั้น ผลลัพธ์การทำงานแสดงในรูปที่ 6.2



รูปที่ 6.2 การส่งข้อมูลภาพผ่าน LAN โดยให้อัตราการส่งภาพมีค่าคงที่

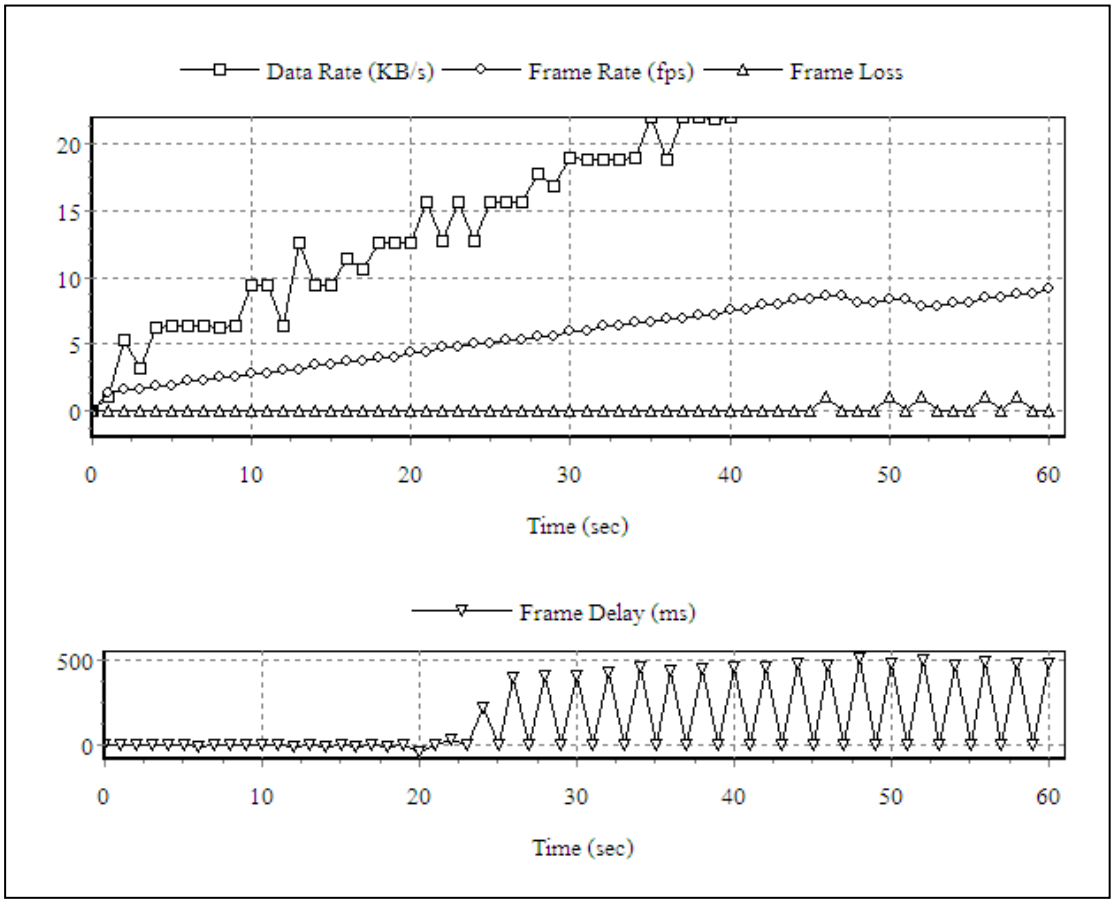
จากนั้นทำการทดลองเพิ่มเติมโดยการเพิ่มการใช้งานในเครือข่าย ณ เวลา 20 วินาที ได้ผลดังรูปที่ 6.3



รูปที่ 6.3 การส่งข้อมูลภาพผ่าน LAN โดยให้อัตราการส่งภาพมีค่าคงที่
และมีการเพิ่มการใช้งานในเครือข่ายที่เวลา 20 วินาที

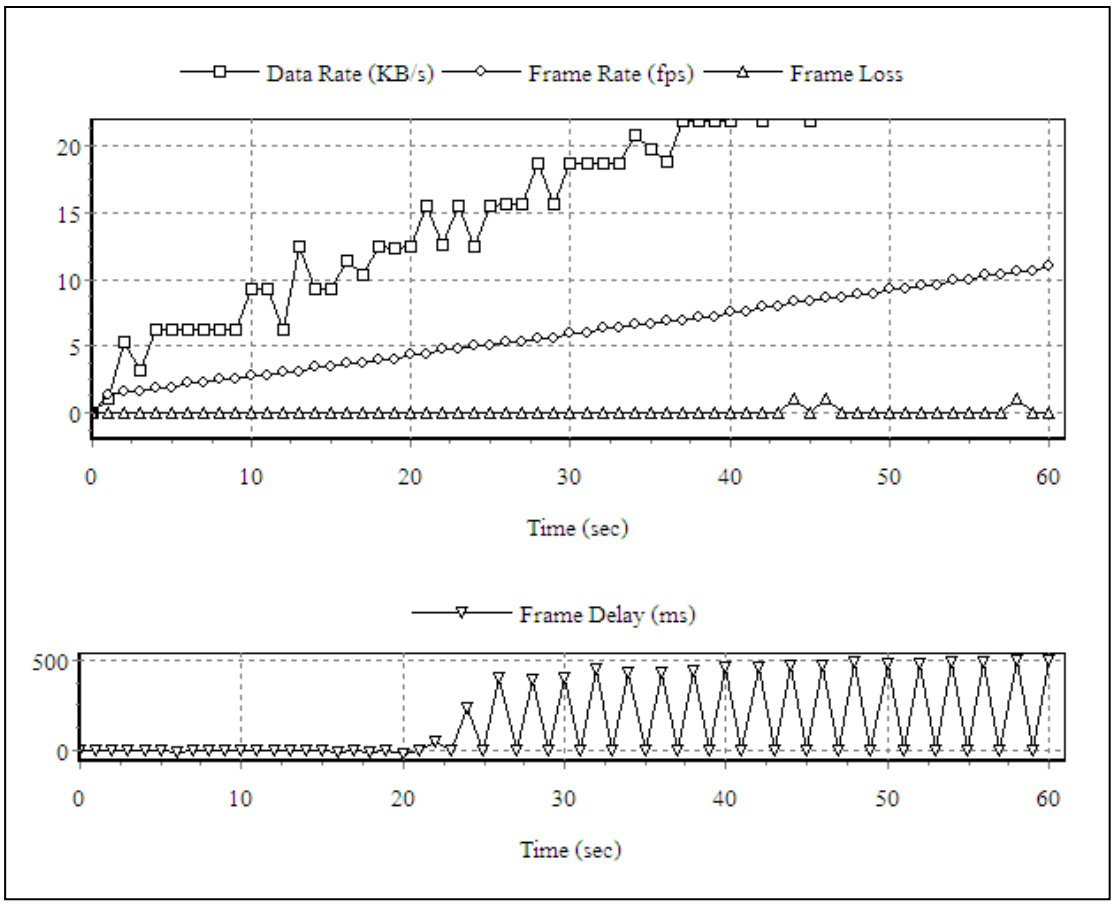
6.1.2 การควบคุมอัตราการส่งภาพ

ในการทดลองนี้จะให้คุณภาพของภาพมีค่าคงที่ ตัวควบคุมฟัซซี่ลอจิกจะปรับเปลี่ยนเฉพาะอัตราการส่งภาพเพียงเท่านั้น ผลลัพธ์การทำงานแสดงในรูปที่ 6.4



รูปที่ 6.4 การส่งข้อมูลภาพผ่าน LAN โดยให้คุณภาพของภาพมีค่าคงที่

จากนั้นทำการทดลองเพิ่มเติมโดยการเพิ่มการใช้งานในเครือข่าย ณ เวลา 20 วินาที ได้ผลดังรูปที่ 6.5



รูปที่ 6.5 การส่งข้อมูลภาพผ่าน LAN โดยให้คุณภาพของภาพมีค่าคงที่
และมีการเพิ่มการใช้งานในเครือข่ายที่เวลา 20 วินาที

6.2 ส่งข้อมูลภาพผ่าน GPRS

ในการทดลองนี้ได้ให้กล้องไอพีมีที่ตั้งอยู่คนละเครือข่ายกับผู้รับชม สำหรับเครือข่าย GPRS นั้นมีแนวโน้มที่จะเกิดความคับคั่งในเครือข่ายได้ง่ายเนื่องมาจากความเร็วของ GPRS นั้นไม่สูงมากนักเมื่อเทียบกับแบนด์วิดท์ที่ใช้ในการส่งข้อมูลภาพ การทดลองนี้เป็นการทดสอบกฎควบคุมฟัซซี่ลอจิกในเรื่องของการมี Frame Loss และ Frame Delay เช่นกฎดังนี้

Rule:

If (Frame Delay) is (High)

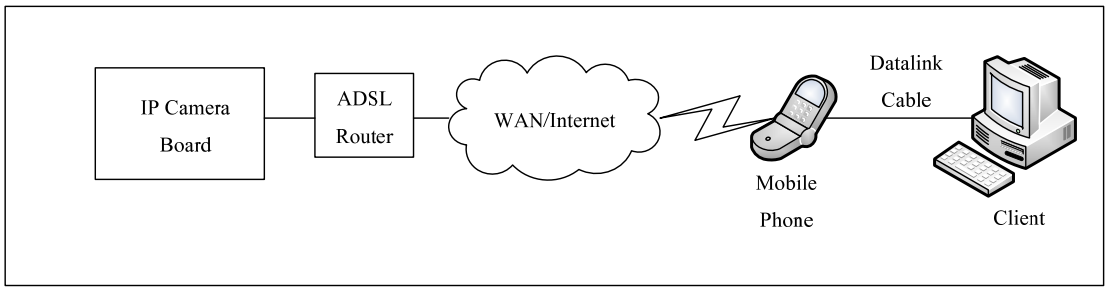
Then (Change Quality) is (Positive High)

Rule:

If (Frame Delay) is (Low)

Then (Change Quality) is (Positive Low)

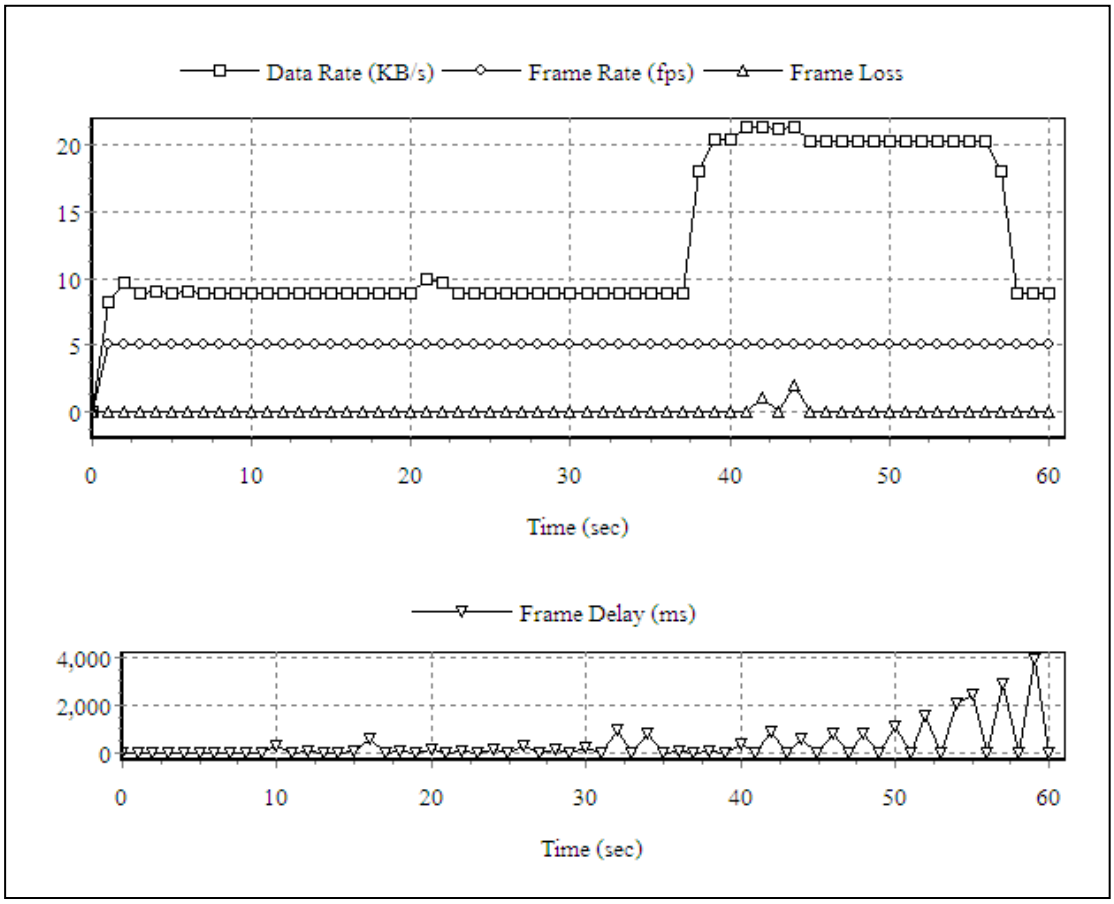
ซึ่งจะแบ่งการทดลองออกเป็น 2 ส่วนคือการควบคุมคุณภาพของภาพโดยให้อัตราการส่งภาพคงที่และควบคุมอัตราการส่งภาพโดยให้คุณภาพของภาพคงที่ รูปที่ 6.6 แสดงแผนภาพการเชื่อมต่อในแบบ GPRS



รูปที่ 6.6 แสดงแผนภาพการส่งข้อมูลภาพผ่านเครือข่าย GPRS

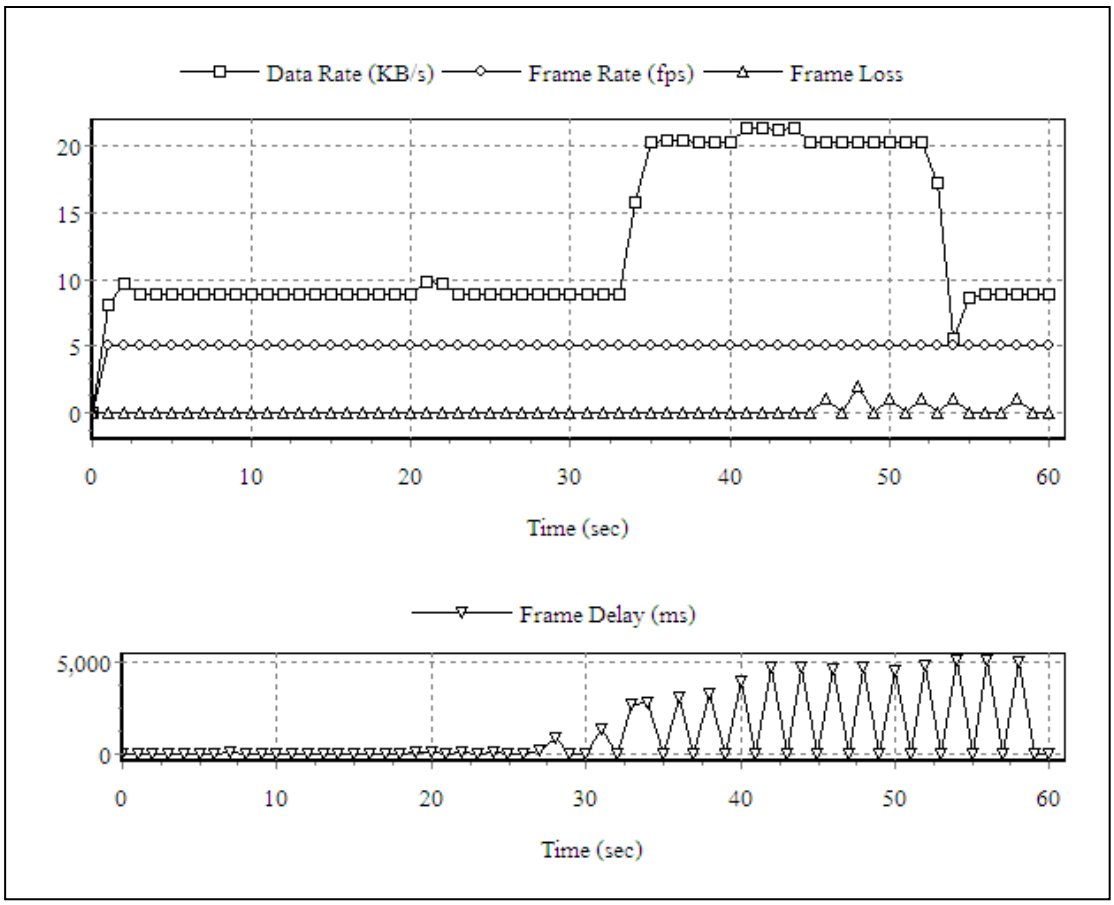
6.2.1 การควบคุมคุณภาพของภาพ

ในการทดลองนี้จะทำการคงค่าอัตราการส่งภาพ ตัวควบคุมฟัซซี่ลอจิกจะปรับเปลี่ยนเฉพาะคุณภาพของภาพเพียงเท่านั้น ผลลัพธ์การทำงานแสดงในรูปที่ 6.7



รูปที่ 6.7 การส่งข้อมูลภาพผ่าน GPRS โดยให้อัตราการส่งภาพมีค่าคงที่

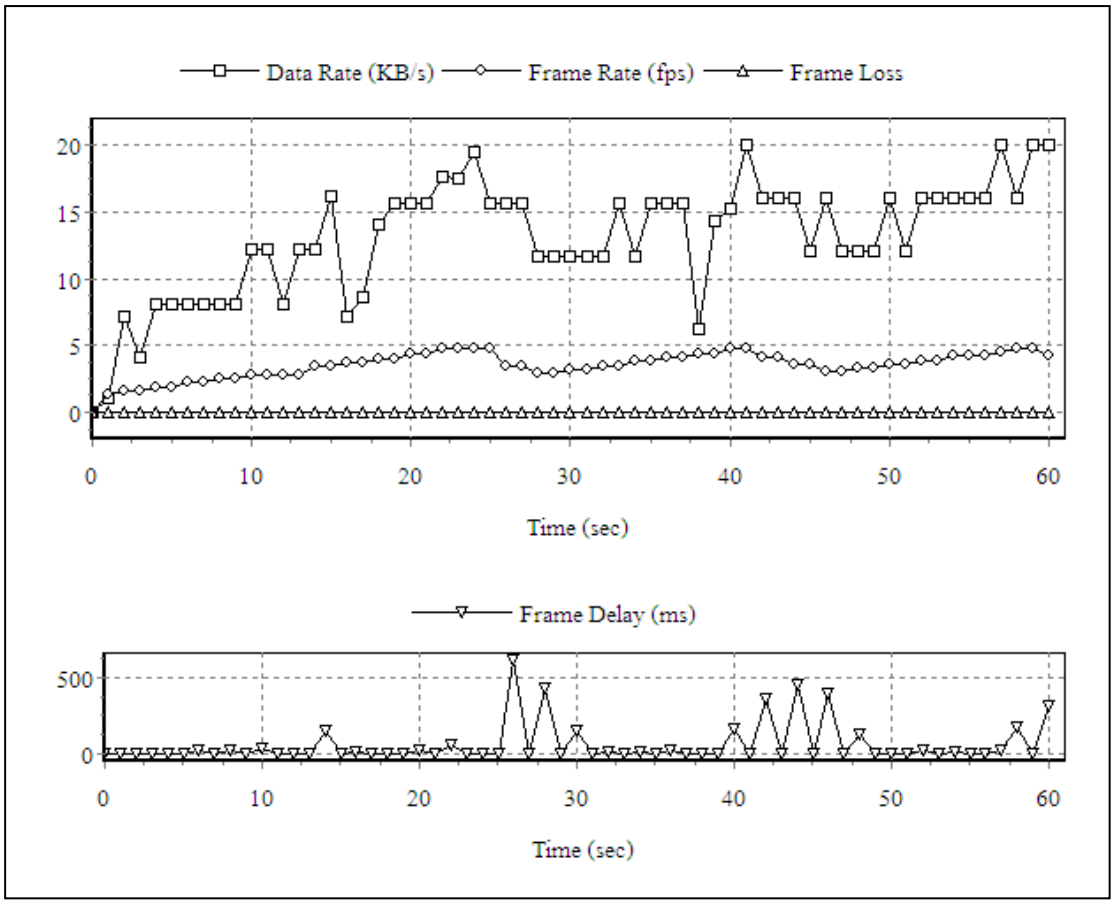
จากนั้นทำการทดลองเพิ่มเติมโดยการเพิ่มการใช้งานในเครือข่าย ณ เวลา 20 วินาที ได้ผลดังรูปที่ 6.8



รูปที่ 6.8 การส่งข้อมูลภาพผ่าน GPRS โดยให้อัตราการส่งภาพมีค่าคงที่
และมีการเพิ่มการใช้งานในเครือข่ายที่เวลา 20 วินาที

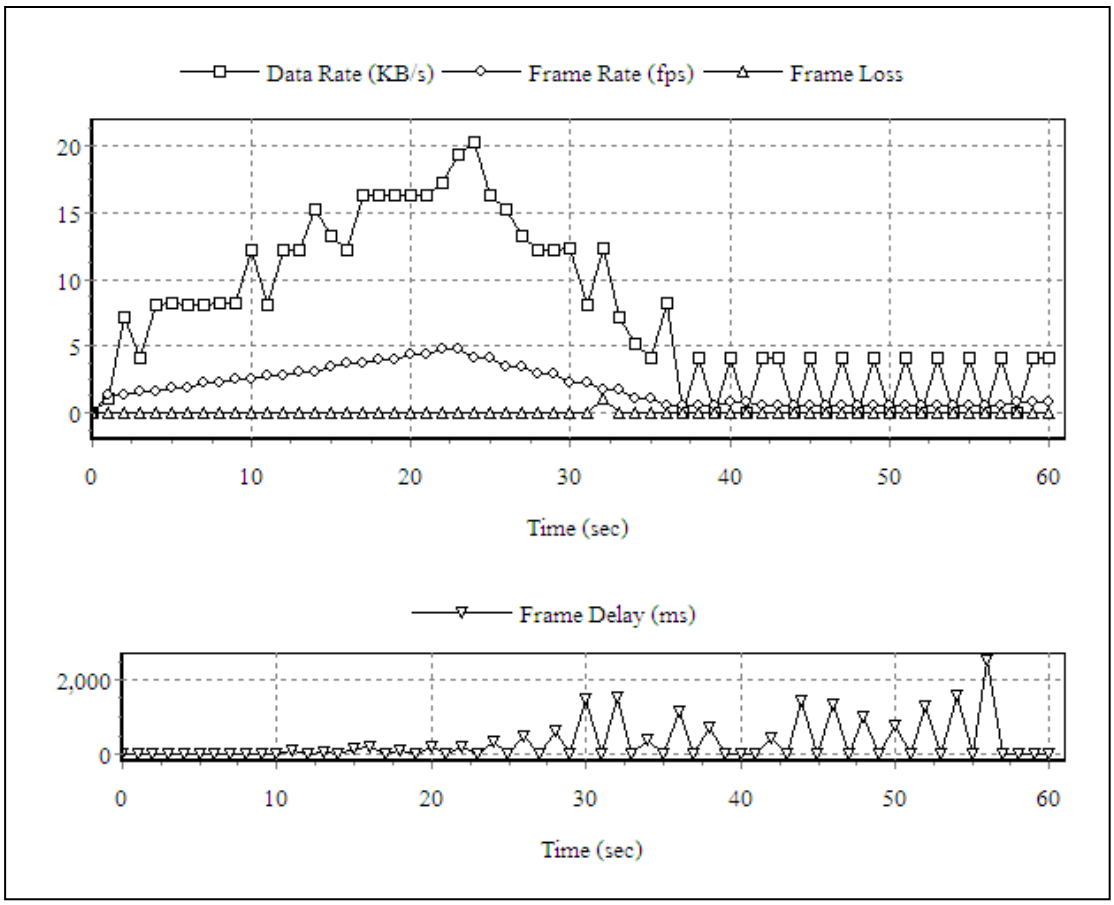
6.2.2 การควบคุมอัตราการส่งภาพ

ในการทดลองนี้จะให้คุณภาพของภาพมีค่าคงที่ ตัวควบคุมฟัซซี่ลอจิกจะปรับเปลี่ยนเฉพาะอัตราการส่งภาพเพียงเท่านั้น ผลลัพธ์การทำงานแสดงในรูปที่ 6.9



รูปที่ 6.9 การส่งข้อมูลภาพผ่าน GPRS โดยให้คุณภาพของภาพมีค่าคงที่

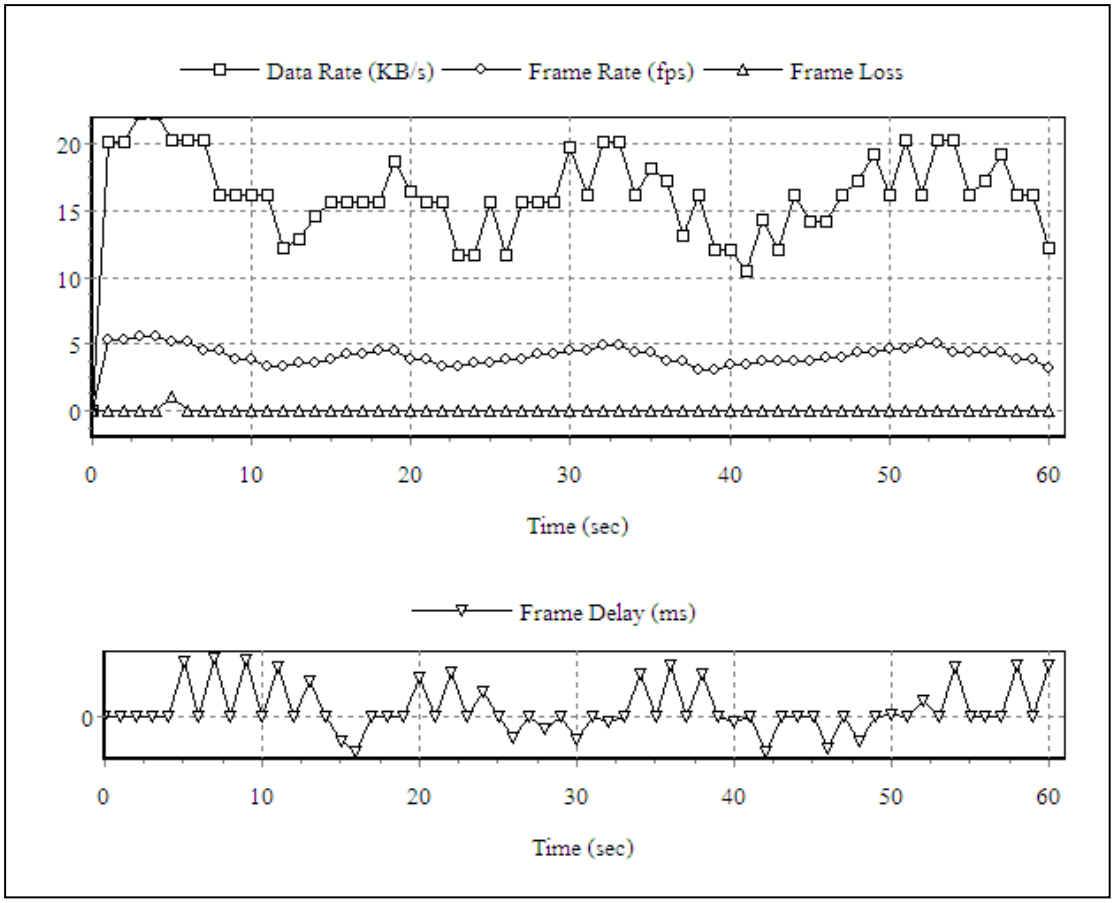
จากนั้นทำการทดลองเพิ่มเติมโดยการเพิ่มการใช้งานในเครือข่าย ณ เวลา 20 วินาที ได้ผลดังรูปที่ 6.10



รูปที่ 6.10 การส่งข้อมูลภาพผ่าน GPRS โดยให้คุณภาพของภาพมีค่าคงที่
และมีการเพิ่มการใช้งานในเครือข่ายที่เวลา 20 วินาที

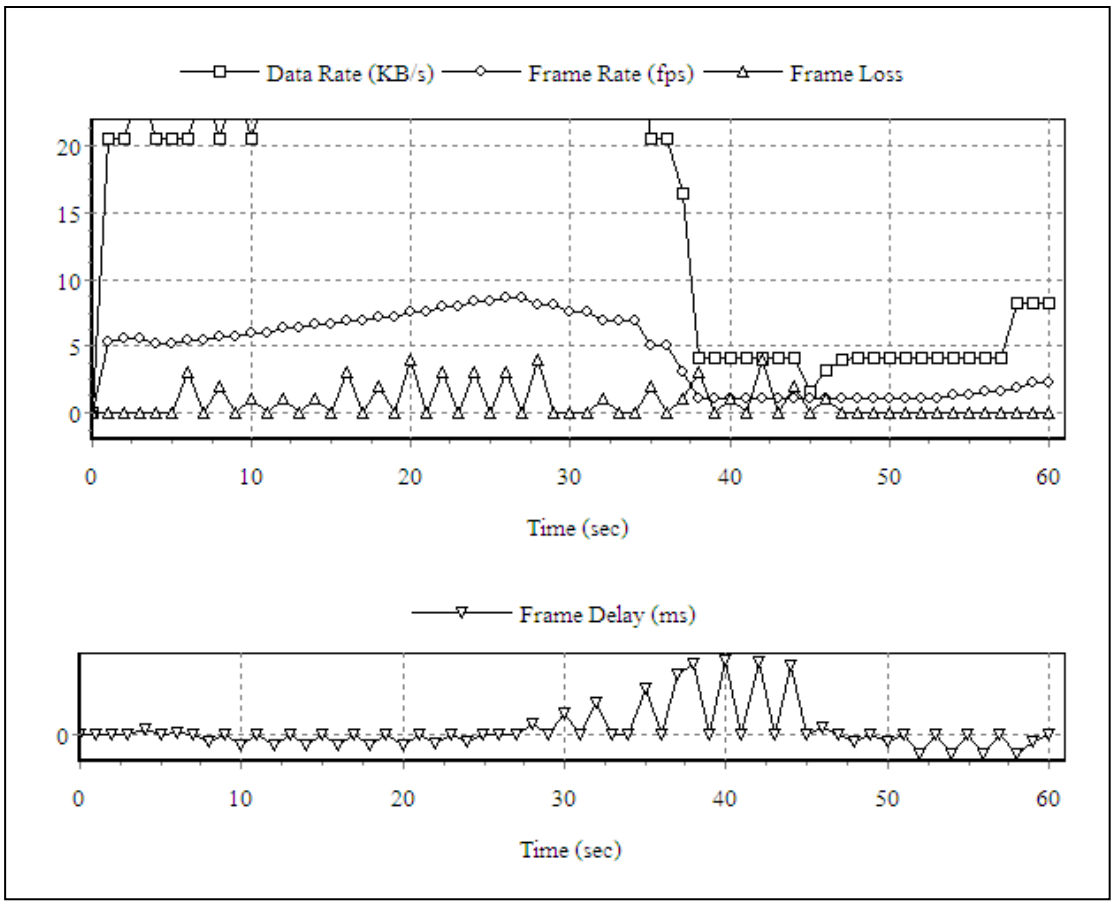
6.2.3 การควบคุมคุณภาพและอัตราการส่งภาพผสมกัน

ในรูปแบบนี้กฎฟัซซี่จะมีอยู่ด้วยกัน 2 ส่วนคือถ้าเครือข่ายไม่มีความคับคั่งก็จะเพิ่มอัตราการส่งภาพและคุณภาพของภาพไปเรื่อย ๆ แต่ถ้ามีความคับคั่งเกิดขึ้นในเครือข่ายก็จะลดอัตราการส่งภาพลงจนกระทั่งถึงค่าต่ำสุดที่ยอมรับได้จากนั้นจึงลดคุณภาพของภาพลง ผลลัพธ์แสดงดังรูปที่ 6.11



รูปที่ 6.11 การส่งข้อมูลภาพผ่าน GPRS โดยมีการปรับคุณภาพของภาพ
และอัตราการส่งภาพร่วมกัน

จากนั้นทำการทดลองเพิ่มเติมโดยการเพิ่มการใช้งานในเครือข่าย ณ เวลา 20 วินาที ได้ผลดังรูปที่ 6.12



รูปที่ 6.12 การส่งข้อมูลภาพผ่าน GPRS โดยมีการปรับคุณภาพของภาพและอัตราการส่งภาพร่วมกันและมีการใช้งานในเครือข่ายที่เวลา 20 วินาที

บทที่ 7
สรุปผลการวิจัยและข้อเสนอแนะ

จากผลการทดลองในวิทยานิพนธ์นี้ได้แสดงให้เห็นว่ากล้องไอพีที่ได้พัฒนาขึ้นมาสามารถส่งข้อมูลภาพด้วยรูปแบบที่เหมาะสมกับแบนด์วิดท์ของเน็ตเวิร์คได้ โดยมีการปรับอัตราการส่งข้อมูลภาพและคุณภาพของภาพ ซึ่งวิธีที่จะปรับนั้นก็มาจะกฎควบคุมฟัซซี่ซึ่งแต่ละกฎนั้นมาจากประสบการณ์ของผู้ใช้งานหรือผู้เชี่ยวชาญแทนที่จะเป็นคณิตศาสตร์ที่ซับซ้อน สำหรับโมดูลกล้องที่มาเชื่อมต่อกับบอร์ดประมวลผลนั้นเป็นกล้องที่ให้ข้อมูลในรูปแบบ JPG อยู่แล้วจึงเป็นการลดขนาดของหน่วยความจำของบอร์ดประมวลผลลง เนื่องจากว่าหากได้เป็นข้อมูลดิบในลักษณะบิตแมป (Bitmap) ก็จะใช้ขนาดของหน่วยความจำเพิ่มมากขึ้น แต่ก็มีข้อด้อยคือรูปแบบที่กล้องไอพีในงานวิจัยนี้จะสนับสนุนการทำงานก็จะมีเพียง Motion JPEG เท่านั้น ซึ่งรูปแบบนี้เป็นรูปแบบส่วนใหญ่ที่ใช้ในกล้องไอพีที่มีขายในตลาดทั่ว ๆ ไป สำหรับงานวิจัยที่จะทำต่อเพื่อเป็นการปรับปรุงยิ่งขึ้นไปก็คือการเชื่อมต่อกับโมดูลกล้องที่เป็นแบบบิตแมปซึ่งจะทำให้กล้องไอพีสามารถรองการรูปแบบของข้อมูลภาพและเสียงหลาย ๆ รูปแบบเช่น H.263 และ MPEG4 เป็นต้น ซึ่งจำเป็นจะต้องใช้บอร์ดประมวลผลที่มีสมรรถนะสูงยิ่งขึ้นไป

รายการอ้างอิง

Bramberger, M., Brunner, J., and Rinner, B. (1999). Real-Time Video Analysis on an Embedded Smart Camera for Traffic Surveillance. **Proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium**. 174-181.

Chaddha, N., and Gupta, A. (1996). A Frame-work for Live Multicast of Video Streams over the Internet. **IEEE International Conference on Image Processing**. 1: 1-4.

Dong, J., He, C., and Zheng, Y.F. (2001). AVP : **A** Highly Efficient Real-Time Protocol for Multimedia Communications on Internet. **IEEE International Conference on Information Technology : Coding and Computing**. 280-284.

Forouzan, B.A. (2003). **TCP/IP Protocol Suit**. McGraw-Hill.

Jamshidi, M., Vadiee, N., and Ross, T.J. (1993). **Fuzzy Logic and Control**. Prentice Hall PTR.

Kami, H., Kurashige, T., Higashi, M., Imaide, T., and Kinugasa, T. (1999). A Network MPEG Camera. **IEEE Transactions on Consumer Electronics**. 45(4): 1206-1212.

Liu, Z., and He, G. (2005). An Embedded Adaptive Live Video Transmission System over GPRS/CDMA Network. **IEEE Proceedings of the Second International Conference on Embedded Software and Systems**.

Shin, M.K., and Hahm, J.H. (1999). Applying QoS Guaranteed Multicast Audio and Video to the Web. **IEEE International Conference on Multimedia Computing and Systems**. 2: 26-30.

Wang, L.X. (1996). **A Course in Fuzzy Systems and Control**. Prentice Hall PTR.

Watkinson, J. (2004). **The MPEG Handbook**. Focal Press.

Wei, J., and Xu, C.Z. (2006). eQoS: Provisioning of Client-Perceived End-to-End QoS Guarantees in Web Servers. **IEEE Transactions on Computers**. 55(12): 1543-1556.

Wootton, C. (2005). **A Practical Guide to Video and Audio Compression: From Sprockets and Rasters to Macro Blocks**. Focal Press.

Zhuang, H., and Wang, Z. (2006). IP-based real time video monitoring system with controllable platform. **IEEE Proceeding of the 2nd Mechatronic and Embedded Systems and Applications**. 1-4.

ภาคผนวก ก

บทความวิชาการที่ได้รับการตีพิมพ์เผยแพร่

รายชื่อบทความวิชาการที่ได้รับการตีพิมพ์เผยแพร่

Duangmanee, P., and Uthansakul, P. (2010). **Implementation of Real-Time Video Streaming with Fuzzy Logic Controller.** International Conference of Wireless Communications and Signal Processing (WCSP 2010). Suzhou, Jiangsu, China; October, 21-23, 2010.

Implementation of Real-Time Video Streaming with Fuzzy Logic Controller

Pumin Duangmanee
School of Telecommunication Engineering
Suranaree University of Technology
NakhonRatchasima, Thailand
e-mail : pumin.duangmanee@gmail.com

Peerapong Uthansakul
School of Telecommunication Engineering
Suranaree University of Technology
NakhonRatchasima, Thailand
e-mail : uthansakul@sut.ac.th

Abstract— **The increasing demand of video streaming recently causes a trouble on network congestion due to a huge data rate transmission. One concept to reduce such a problem is to dynamically adjust the rate of video streaming according to an available bandwidth. This paper introduces a fuzzy logic controller to control a real-time video streaming by adjusting either a frame rate or a video quality. In this paper, two experimental environments, LAN or GPRS connections, are investigated. The results show that a fuzzy logic controller is able to control a frame rate and a quality of video streaming according to an available bandwidth automatically.**

Keywords: Video Streaming, Fuzzy Logic Controller

I. INTRODUCTION

To solve a problem when network traffic congestion occurs for video streaming, one solution is to let users make a decision on what is a current bandwidth available to received video streaming [1][2]. In literature, users can notice that the video is not played smoothly and then choose another step of lower bit rate until the quality of video streaming is acceptable. They use a frame skip technique on MPEG-1 video. The disadvantage of this technique is on a big step value of bit rate transmissions, for example as 64kbps, 128kbps. Another way to handle a congest problem is to use the mathematical model of communication channel in order to estimate an available bandwidth [3] before sending video files. The work in [3] focuses on GPRS/CDMA network. A packet loss and round trip time were used to estimate an available bandwidth and then adjust the encoder output rate. In [4], the technique of reserving a resource along a communication channel for real-time video streaming is proposed to solve a congestion problem. By reserving a router resource from source to destination, the efficient traffic usage is able to be realized. However, this technique can only comply with a local network.

In this paper, the scope of interest is the problem on video streaming over the internet that users do not exactly know a dynamic behavior on the congestion of the network traffic. Therefore, the flexible rules to handle such a problem are required. Fuzzy logic controller is a suitable engine to tackle this problem because it has a good performance under non linear constraints [5]. This paper introduces a fuzzy logic controller to control a real-time video streaming by adjusting an encoding rate transmission. The experiments under two scenarios, on LAN and GPRS networks, are undertaken. The remainders of this paper are given as follows. Section II describes on the architecture of systems. Then the experimental results are given in Section III. Finally, the conclusion is presented in Section IV.

II. SYSTEM ARCHITECTURE

A. System Overview

In Fig. 1, a video streaming server receives a raw video image input from video input device, example, CMOS image sensor. Each image is sent to a client via Real Time Protocol (RTP) over UDP and it is displayed to user via any player programs. The fuzzy logic controller receives feedback information from client and makes a decision to adjust rate of video streaming. The feedback information in this paper is frame loss which is determined by timestamp and sequence number of RTP header. With knowledge of frame loss, fuzzy logic controller can now have two options on adjusting a rate transmission. The first option is a quality of video streaming and the second is a frame rate. Both options influence to a channel bandwidth. For example, to increase fps (frame per second) will increase the bandwidth or to increase a video quality will also increase the bandwidth. The choice of choosing the options depends on a user profile, either (a) to keep a video quality or (b) to keep a smoothness of continuous video.

As mentioned above, a fuzzy logic controller can control a video streaming with two options, (a) image quality and (b) frame rate. The image quality can be changed by encoding a different quality block presented in Fig. 2. The output of encoded image is passed to RTP block sending a video stream to client. To control frame rate, the controller adjusts a sending time of RTP packet instead of encoding image in a quality option. It concerns a RTP timestamp and a sequence number of RTP header. The detail of RTP header is shown in Fig. 3. As seen in this figure, the useful information used for adjustment is a timestamp and a sequence number. The timestamp represents the sampling instance of the first octet in RTP packet data. The sequence number is a unit increment for each delivered RTP data packet. The client uses this value to detect a packet lost.

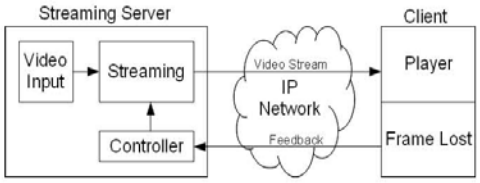


Fig. 1 System with fuzzy logic controller for a video streaming.

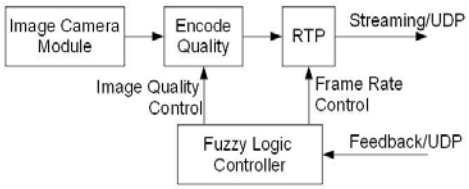


Fig. 2 Flow diagram of adaptive rate transmission.

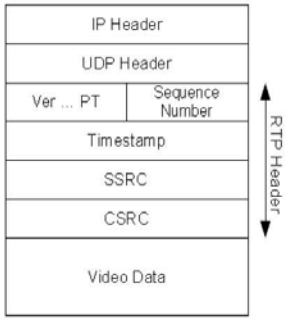


Fig. 3 RTP header.

B. Fuzzy logic controller

The aim of fuzzy logic controller is to substitute or replace a skilled human operator with a fuzzy rule-based system [6][7]. Fuzzy logic controller has an advantage on robustness to unknown system parameters. These parameters can be used in a controller to control a video streaming even the system cannot be modeled by any exactly known models. The components of the controller are shown in Fig. 4.

As seen in Fig. 4, there are four components of fuzzy logic controller. Fuzzifier is the first component to convert a real input into a fuzzy set. In this case, there are three kinds of fuzzy set: "low", "medium" or" high". The fuzzy set is sent to the next component named as a fuzzy inference engine. Fuzzy inference engine processes an input by looking up the rule in a fuzzy rule base. The output of each rule is sent back to fuzzy inference engine and then passed to defuzzifier.

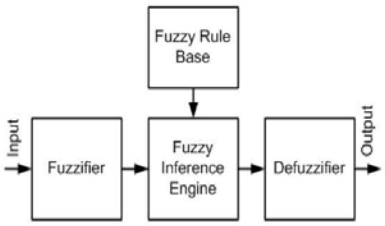


Fig. 4 Components of fuzzy logic controller.

Defuzzifier is defined as a mapping between a fuzzy set to an actual output. In this paper, we choose a weight average defuzzifier to produce an actual value because of its simplicity which can be directly implemented by a low cost embedded system. The formula of defuzzifier is expressed by

$$f(y) = (\sum \mu(w) * w) / \sum \mu(w) \qquad (1)$$

where

$f(y)$ is an actual output value.

$\mu(w)$ is a membership function of input w.

The process of fuzzy logic controller can be depicted as shown in Fig. 5.

The most important part of the fuzzy logic controller is its rule base. This part requires an experience or a skill of human operator to create a fuzzy rule base, input membership function as well as output membership function. For this paper, we determine a frame loss meaning by "low" at 1-3 frames/sec, "medium" if frame loss is around 3-6 frames/sec and "high" if frame lost is over 6 frame/sec.

The output of fuzzy logic controller is changed by its both frame rate and quality. If video frame lost at client side, the client will send this information back to a video streaming server. The fuzzy logic controller at server then tries to reduce a frame loss to zero. To reduce a frame loss, there are three choices as follows;

- Reduce a video frame rate, fix a video quality.
- Reduce a video quality, fix a video frame rate.
- Reduce both video quality and frame rate.

In this paper, the simple rule is implemented to the fuzzy rule base. The below are the rules of fixing a video quality.

- If frame loss is zero,
  then change frame rate is positive low.
- If frame loss is low,
  then change frame rate is negative low.
- If frame loss is high,
  then change frame rate is negative high.

For fixing a video frame rate, the rules are as follows.

- If frame loss is zero,
  then change quality is negative low.
- If frame loss is low,
  then change quality is positive low.
- If frame loss is high,
  then change quality is positive high.

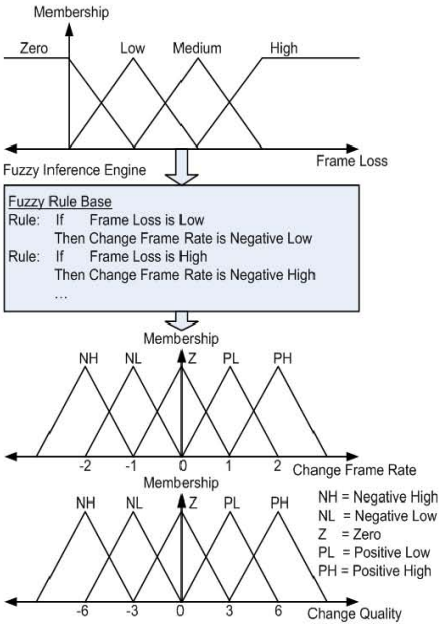


Fig. 5 Process of fuzzy logic controller.

There are many choices to reduce both video quality and frame rate at the same time. This depends on a choice of operator. For users, they may want to keep either a continuity of video stream or a video quality depending on a personal demand. If user is interested to continuously see a video, then the rule to keep a frame rate is a major priority. In this paper, the complex fuzzy rule as described above can be given by,

- If frame loss is high and frame rate is high,
  then change frame rate is negative high.
- If frame loss is low and frame rate is high,
  then change frame rate is negative low.
- If frame loss is high and frame rate is low,
  then change quality is positive high.
- If frame loss is low and frame rate is low,
  then change quality is positive low.

C. Feedback Information

A client sends feedback information to a streaming server. This feedback information can be analyzed by a received RTP header at the client. Client use a time stamp in RTP header at the beginning of stream to build a clock used as a reference for the rest of incoming RTP packet. Frame loss can be determined by timestamp compared with a client clock and a sequence number in an incoming RTP header. If time delay of incoming video frame is over predefined value, in the case, that value is not acceptable for real time video streaming. This will be counted as a video frame loss. The sequence number in RTP header can be used for determining how many frames are actually lost. Finally, the client sends this information to the fuzzy logic controller via UDP.

III. Experimental Results

There are many video formats widely used in video streaming, for examples as MPEG-1, MPEG-2, MPEG-4, H.264, MJEPG and so on. MPEG family has a relationship between frames consisting of I-frame or main image frame, P-frame or different frame and B-frame or bidirectional frame. To reduce such a complexity, this paper selects a Motion JPEG (MJPEG) format because each image frame of MJPEG is independent and can change a frame rate at any time easily.

A. Experimental environments

There are two environments to test the proposed system. The first scenario is that a server sends video streams via LAN (10/100Mbps) connection and the second scenario is via GPRS connection. Both scenarios are undertaken under an unknown network bandwidth. The software running in a streaming server is implemented by Borland Delphi and Borland C++ Builder. The camera interfaces to the streaming server by USB port. The program of server consists of fuzzy logic controller, image encoder and RTP unit. For the client software, it is also implemented by Borland Delphi consisting of RTP and video display for user. The screenshots of programs on both server and client are shown in Fig. 6 and Fig. 7, respectively.

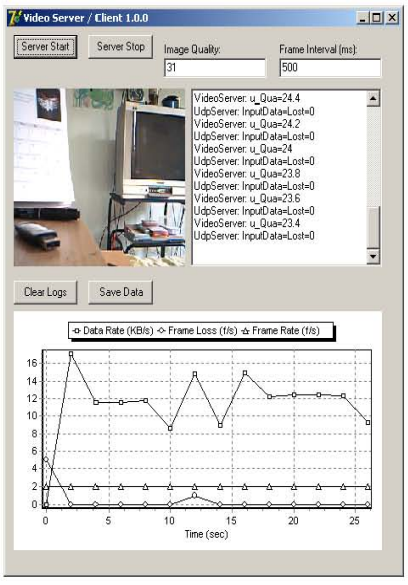


Fig. 6 Screenshot of server program.

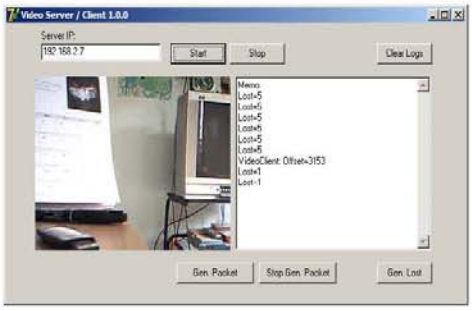


Fig. 7 Screenshot of client program.

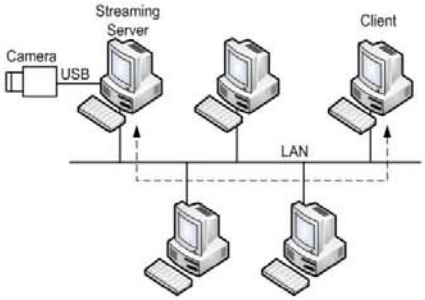


Fig. 8 Experimental environment via LAN connection.

B. Fuzzy Logic Controller with LAN (10/100Mbps)connection

The experimental environment is illustrated in Fig. 8. In this scenario, both client and server are located on the same LAN with the different IP addresses. LAN speed is very high in comparison with a bandwidth used by our video streaming. There are other computers on this LAN which they can cause the network congestion. However, our experimental results do not indicate any frame loss due to these users working in this network. Therefore, because of network congestion is not presented, only two types of experiments can be performed; to increase a frame rate and fix a quality or to increase a quality and fix a frame rate.

The rule for increasing a frame rate is given by

- If frame loss is zero, then increase frame rate to low.

For the other type of experiment, to increase a quality and fix a frame rate, the rule is given by

- If frame loss is zero, then increase quality to low.

The results in Fig. 9 and Fig. 10 are undertaken by two rules described above. If frame loss is not presented at client side, the streaming server can increase either frame rate or quality. Client in case of increasing a frame rate will see video stream smoother than client in case of increasing a video quality. In turn, the better video quality is achieved in case of increasing a video quality.

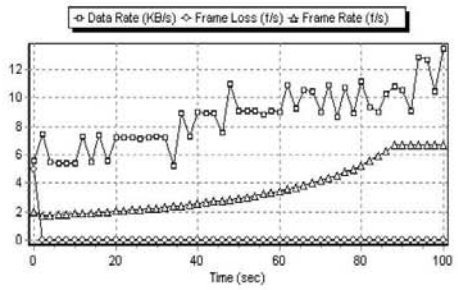


Fig. 9 The results of fuzzy logic controller via LAN for rule of increasing a frame rate (fix video quality).

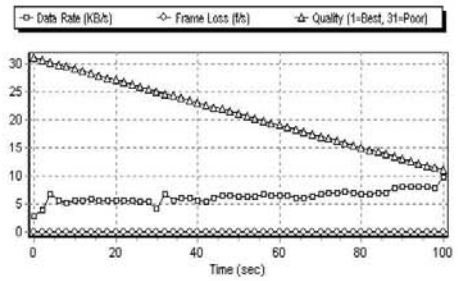


Fig. 10 The results of fuzzy logic controller via LAN for the rule of increasing a video quality (fix frame rate).

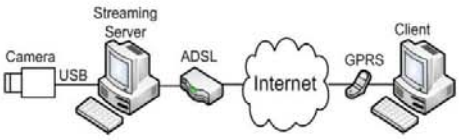


Fig. 11 Experimental environment via GPRS connection.

C. Fuzzy logic controller with GPRS connection

In this scenario, the client and streaming server are now located on the different IP networks. The client gets a video streaming via GPRS communication with an unknown amount of network congestion. The experimental environment is depicted in Fig.11.

The experiment begins with that the client gets a video streaming from server without a congestion feed at the client side and the streaming server fixes a quality of image. The results are presented in Fig. 12. As noticed in this figure, the

data rate increases up to 20 Kbytes/sec and then falls to 10 Kbyte/sec at time 62 sec. This is due to the unpredictable network congestion caused by real users on GPRS connection. The mean value of frame rate on a streaming server is about 2 frames/sec.

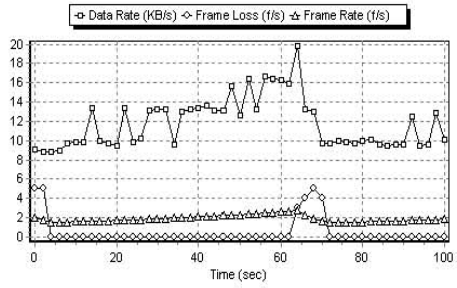


Fig. 12 The results of fuzzy logic controller via GPRS for the rule of increasing a frame rate (fix video quality).

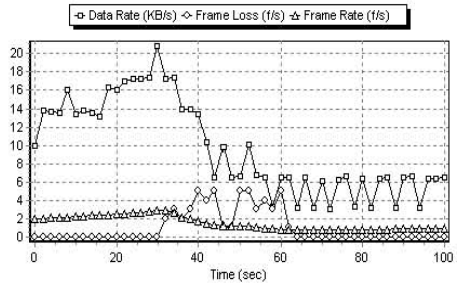


Fig. 13 The results of fuzzy logic controller via GPRS for the rule of increasing a frame rate (fix video quality), with network congestion at 25 sec.

At client side, we add more network congestion by a small program sending UDP broadcast packets at time 25 sec. The results are shown in Fig. 13. The fuzzy logic controller can reduce a video streaming frame rate in order to reduce the frame loss to zero. Before congestion feed, a video frame rate is about 2 frames/sec, but after that the fuzzy logic controller can reduce video frame rate to about 1 frame/sec automatically.

Next experiment is the rule of fixing a frame rate but changing a video quality, we fix a frame rate at streaming server to 2 frames/sec. The results are shown in Fig. 14. The quality of video stream begins from a poor quality. When running test, the fuzzy logic controller increases a video quality until a frame loss is found. Then, it decreases a video quality to make a frame loss being zero. The quality of video is varied in range of 20 to 25. This variation is caused by the fuzzy rules described before.

The last experiment is done by adding more network congestion again at time 25 sec. The results are shown in Fig. 15. The quality of video stream also begins from poor quality. As observed in this figure, the feeding congestion cause such a frame loss that the fuzzy controller has to reduce a video quality after 25 sec.

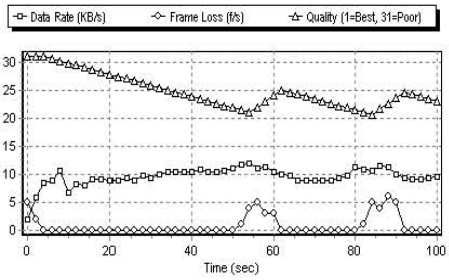


Fig. 14 The results of fuzzy logic controller via GPRS for the rule of increasing a video quality (fix frame rate).

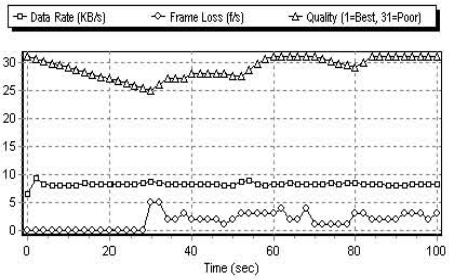


Fig. 15 The results of fuzzy logic controller via GPRS for the rule of increasing a video quality (fix frame rate), with network congestion at 25 sec.

IV. CONCLUSION

This paper presents a fuzzy logic controller to control a frame rate and quality of video streaming according to available bandwidth automatically. The performance of streaming depends on the fuzzy rules which they are come from the experience of operators, not from the mathematical model. In future work, the authors will focus on implementing a low cost embedded system by applying the proposed controller on more video streaming formats.

V. ACKNOWLEDGEMENT

The research project is financially supported by the Thailand Research Fund (TRF), Grant No: WI515E161.

REFERENCES

[1] Hiroaki Kami, Tomoyuki Kurashoge, Masayuki Higashi, Takuya Imaide, Toshiro Kinugasa, "A Network MPEG Camera", IEEE Transactions on Consumer Electronics, Vol. 45, No. 4, NOVEMBER 1999, pp. 1206-1212.

[2] Navin Chaddha and Anoop Gupta, "A Frame-work for Live Multicast of Video Streams over the Internet", IEEE International Conference on Image Processing, 1996.
[3] Zhixiong Liu and Guiming He, "An Embedded Adaptive Live Video Transmission System over GPRS/CDMA Network", Proceeding of the Second International Conference on Embedded Software and Systems, IEEE 2005.
[4] Myung-Ki Shin, Jin-Ho Hahm, "Applying QoS Guaranteed Multicast Audio and Video to the Web", IEEE International Conference on Multimedia Computing and Systems, 1999.

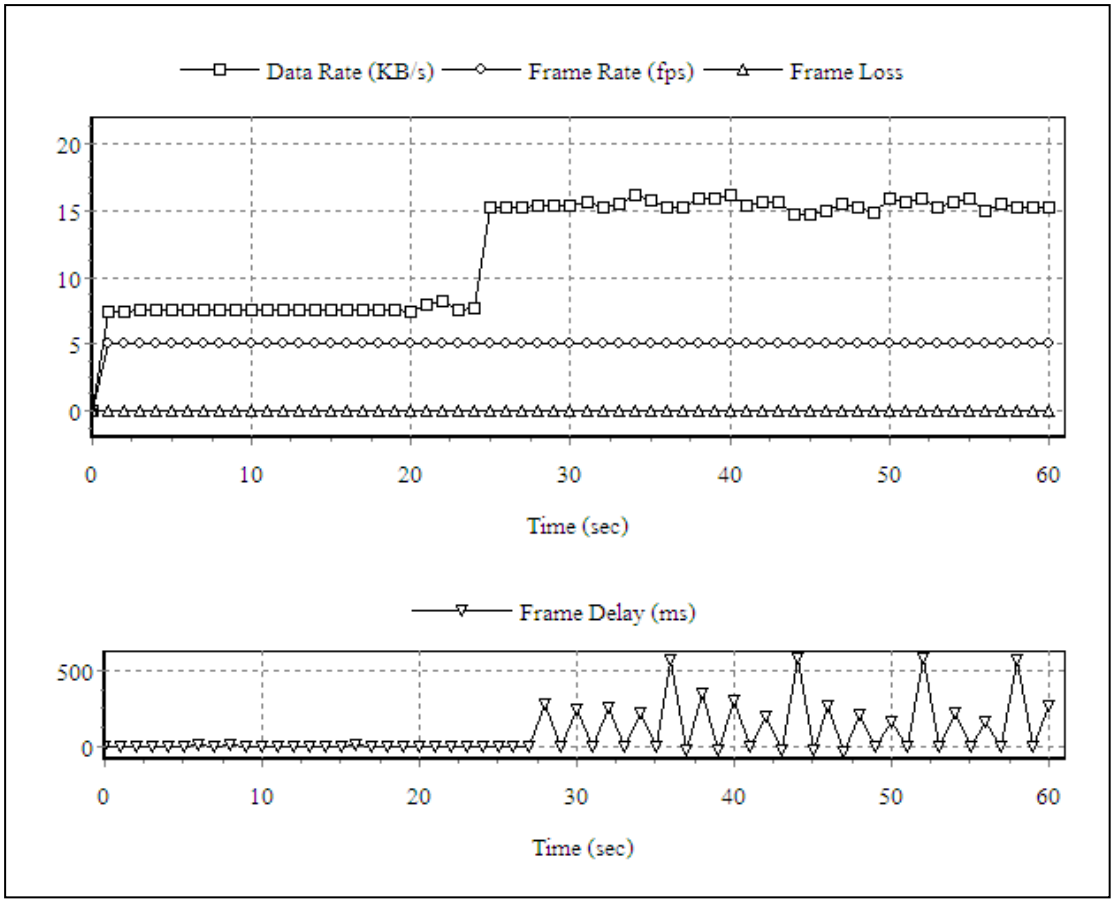
[5] Jianbin Wei, Cheng-Zhong Xu, "eQoS: Provisioning of Client-Perceived End-to-End QoS Guarantees in Web Servers", IEEE Transactions on Computers, Vol. 55, No. 12, December 2006.
[6] Mohammad Jamshidi, Nader Vadiee, Timothy J. Ross, "Fuzzy Logic and Control : Software and Hardware Application", Prentice-Hall International, Inc, 1993.
[7] Li-Xin Wang, "Adaptive Fuzzy Systems and Control : Design and Stability analysis", PTR Prentice Hall, 1994.

ภาคผนวก ข

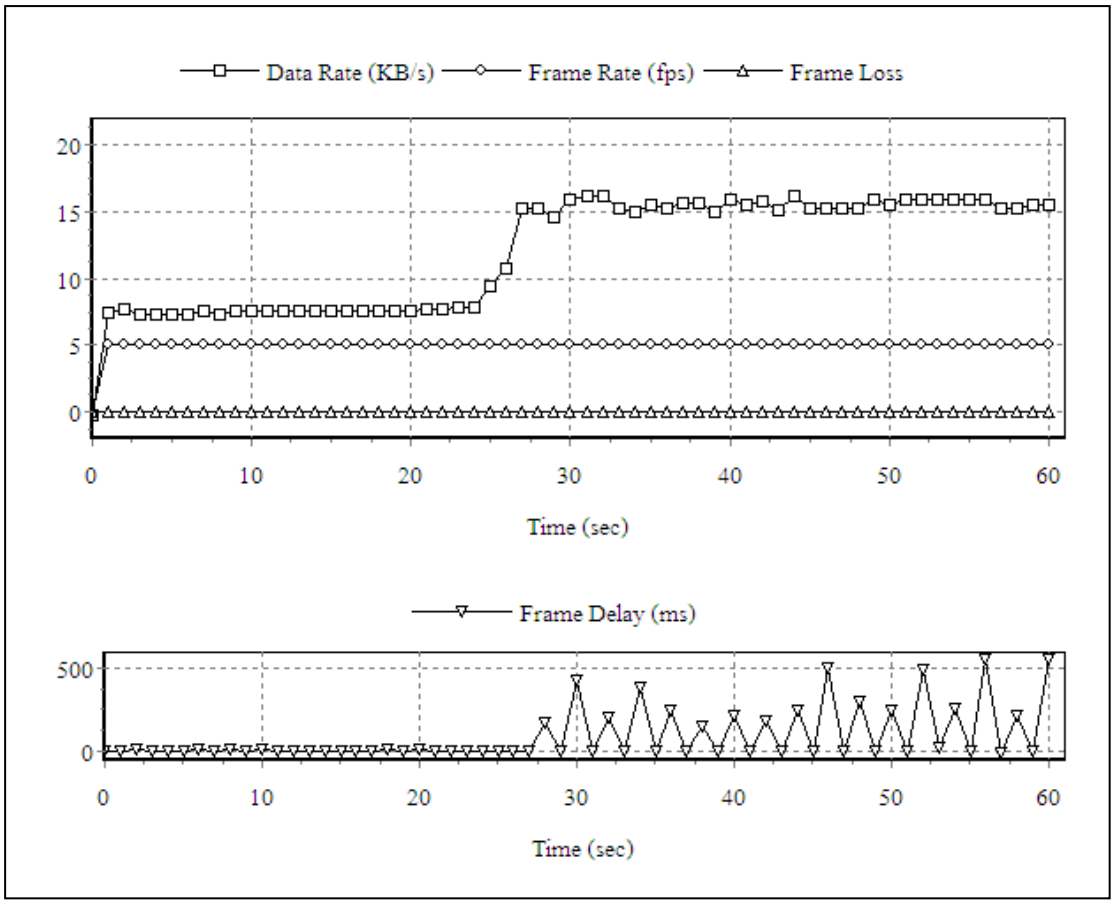
ผลการทดลองเพิ่มเติม

ผลการทดลองเพิ่มเติม

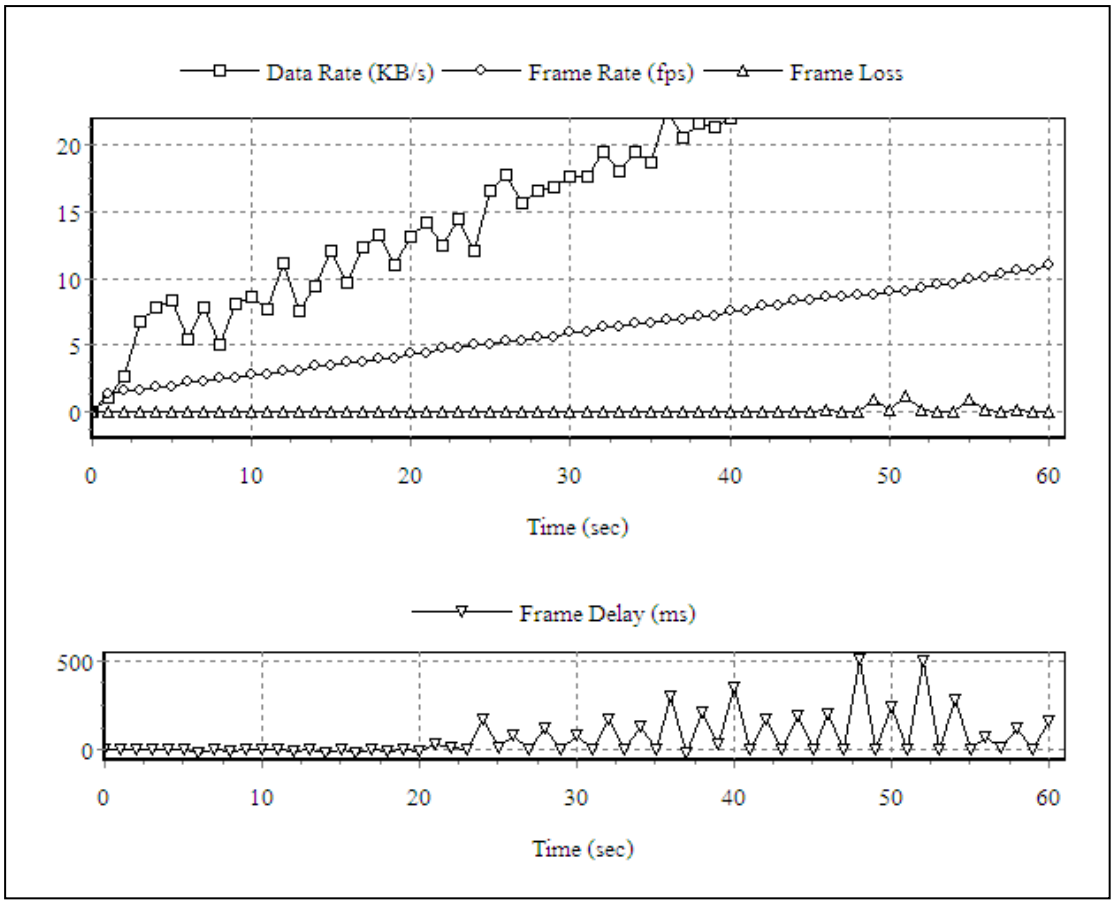
การทดลองครั้งที่ 2



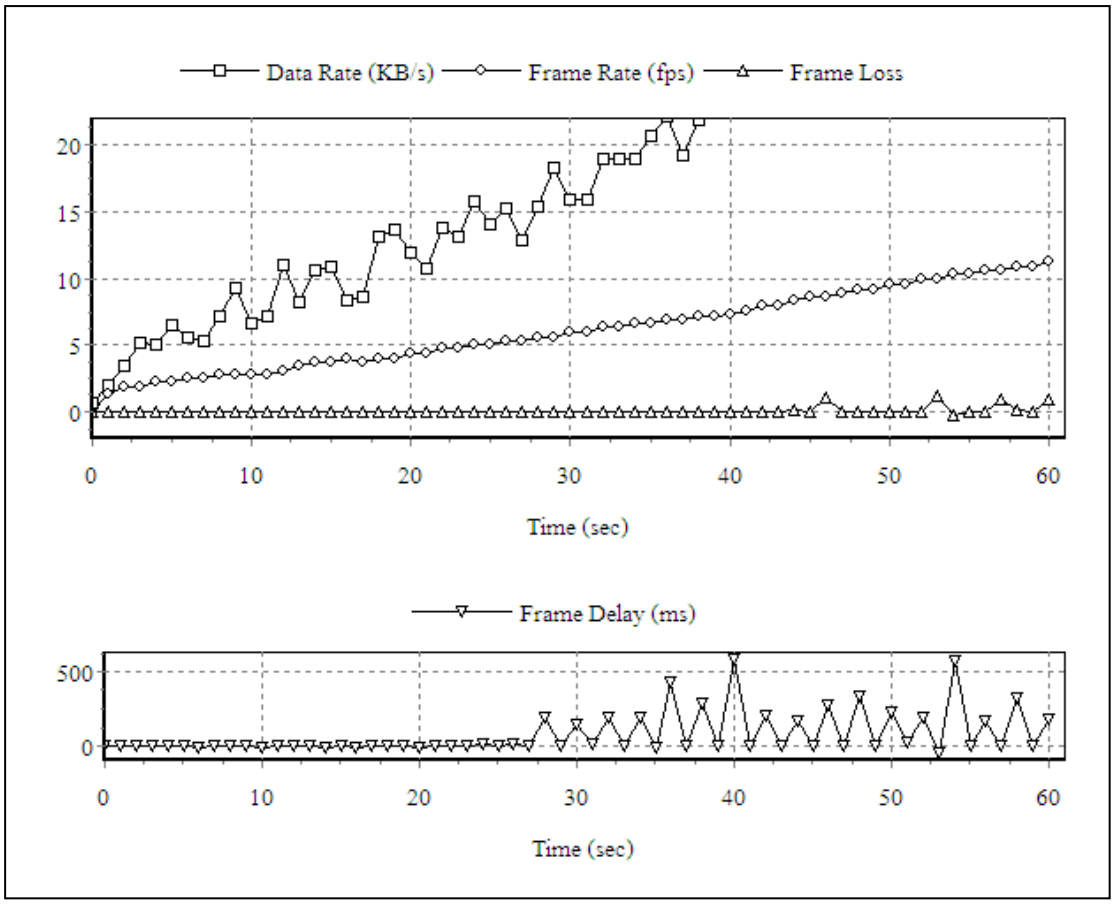
รูปที่ ข1 การส่งข้อมูลภาพผ่าน LAN โดยให้อัตราการส่งภาพมีค่าคงที่



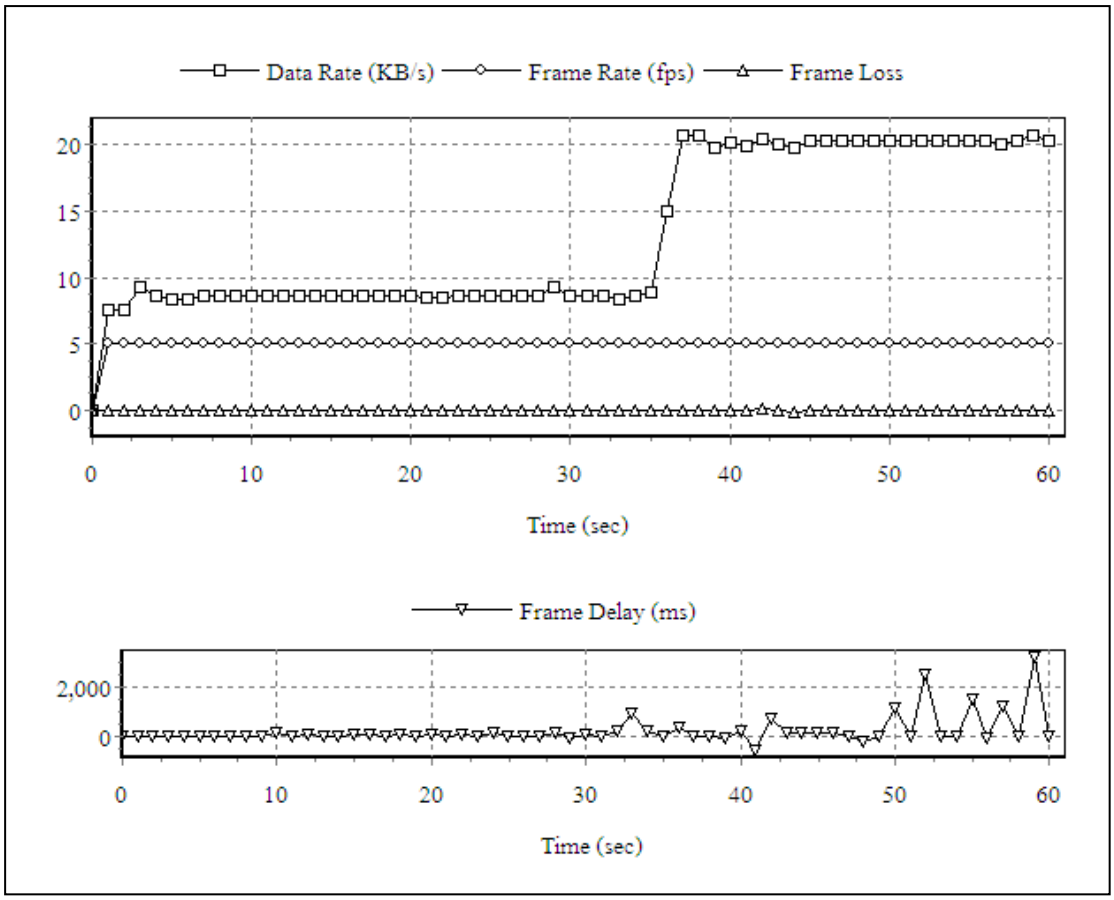
รูปที่ ข2 การส่งข้อมูลภาพผ่าน LAN โดยให้อัตราการส่งภาพมีค่าคงที่
และมีการเพิ่มการใช้งานในเครือข่ายที่เวลา 20 วินาที



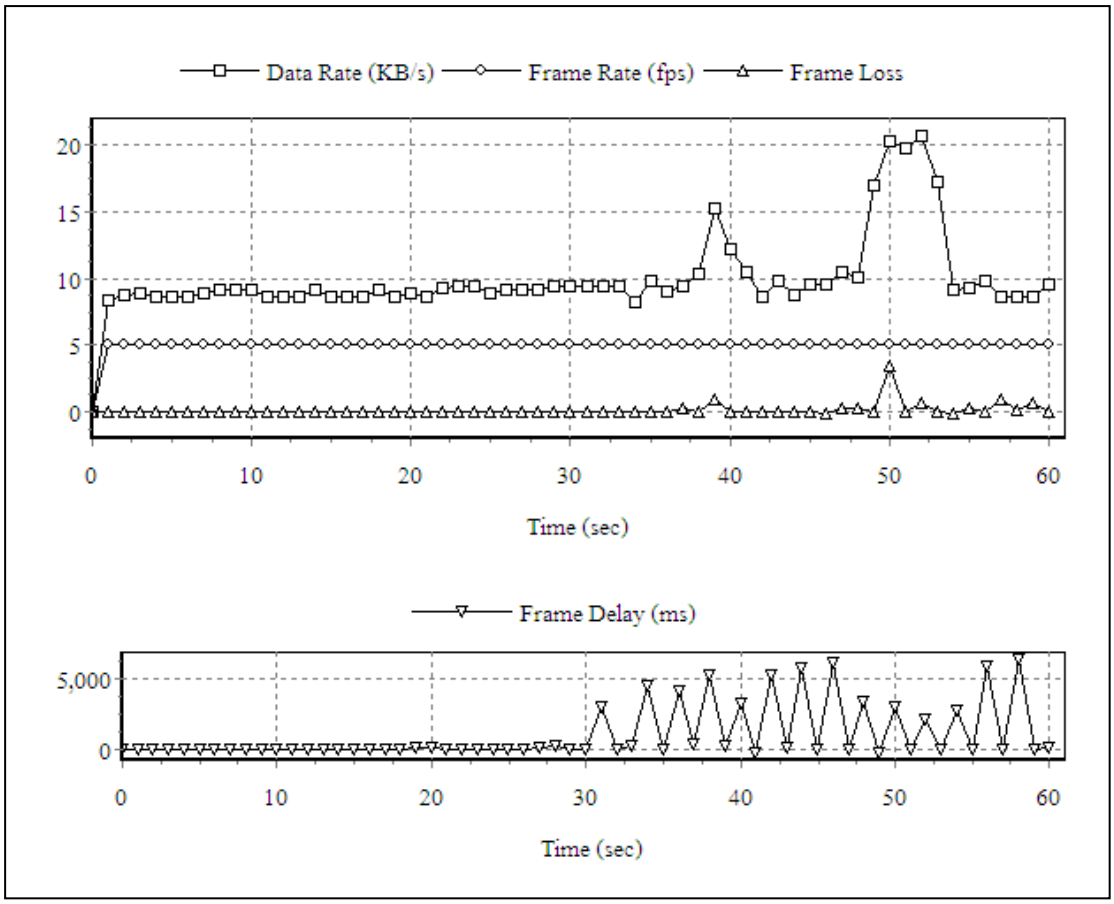
รูปที่ ข3 การส่งข้อมูลภาพผ่าน LAN โดยให้คุณภาพของภาพมีค่าคงที่



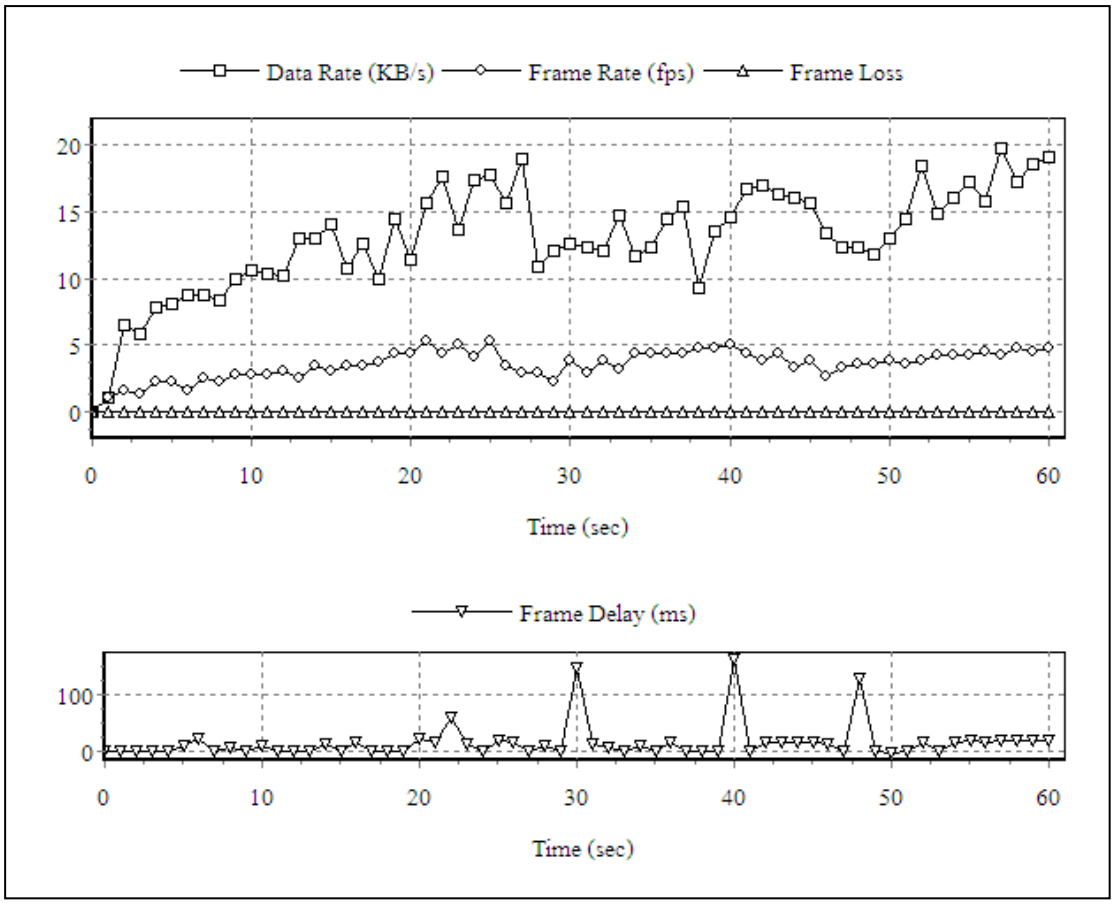
รูปที่ ข4 การส่งข้อมูลภาพผ่าน LAN โดยให้คุณภาพของภาพมีค่าคงที่
และมีการเพิ่มการใช้งานในเครือข่ายที่เวลา 20 วินาที



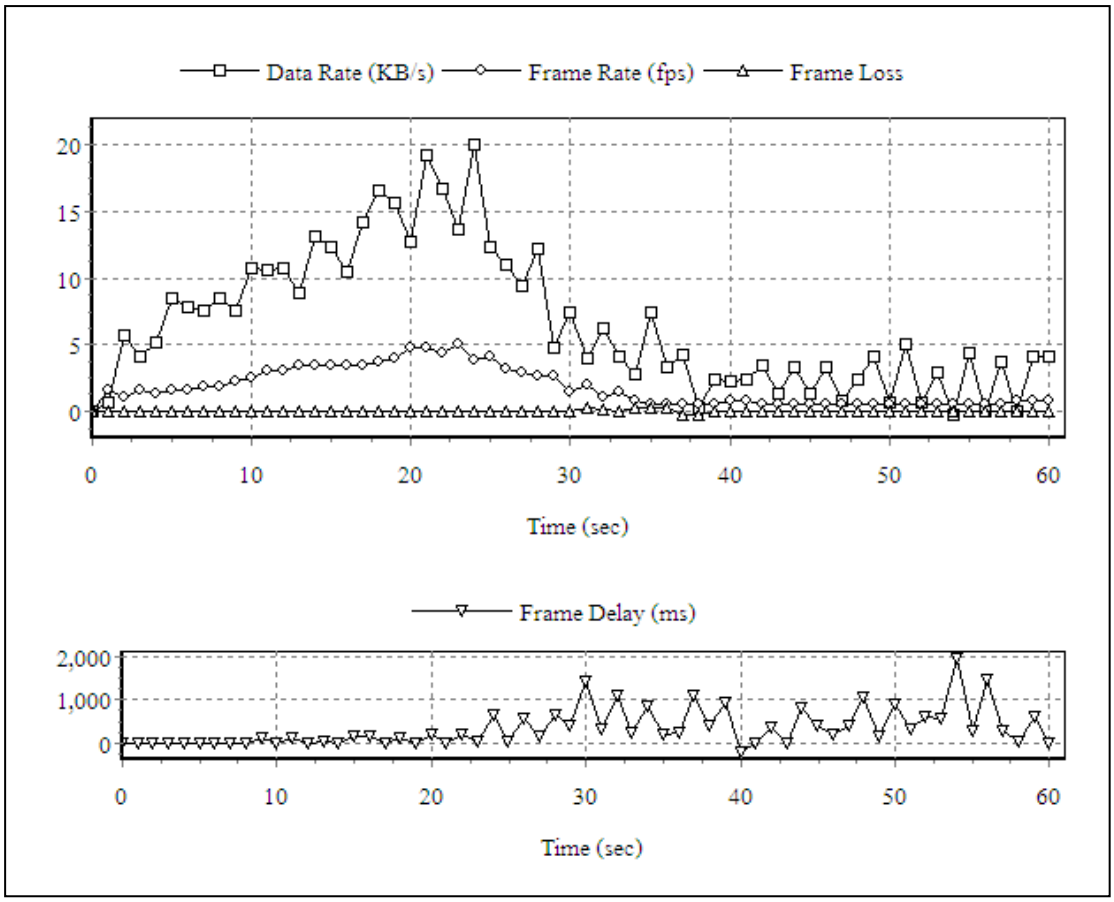
รูปที่ ข5 การส่งข้อมูลภาพผ่าน GPRS โดยให้อัตราการส่งภาพมีค่าคงที่



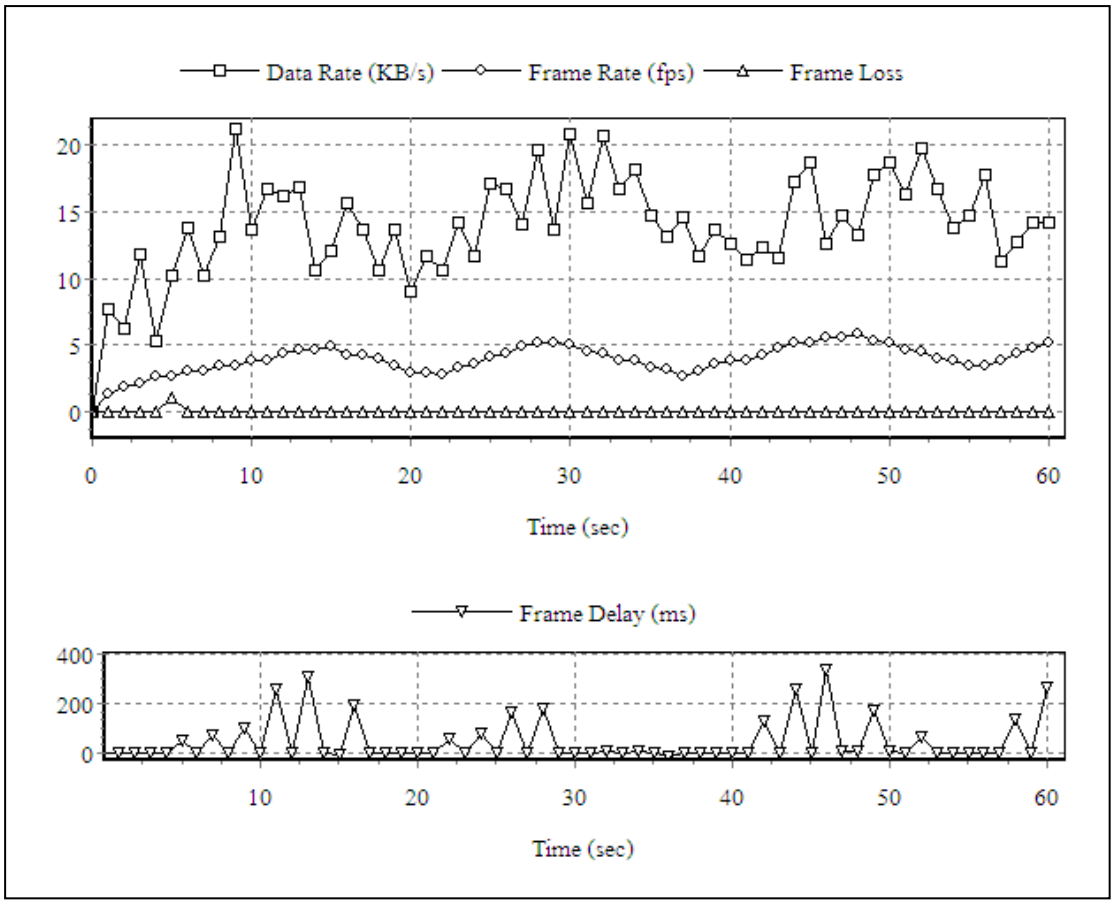
รูปที่ ข6 การส่งข้อมูลภาพผ่าน GPRS โดยให้อัตราการส่งภาพมีค่าคงที่
และมีการเพิ่มการใช้งานในเครือข่ายที่เวลา 20 วินาที



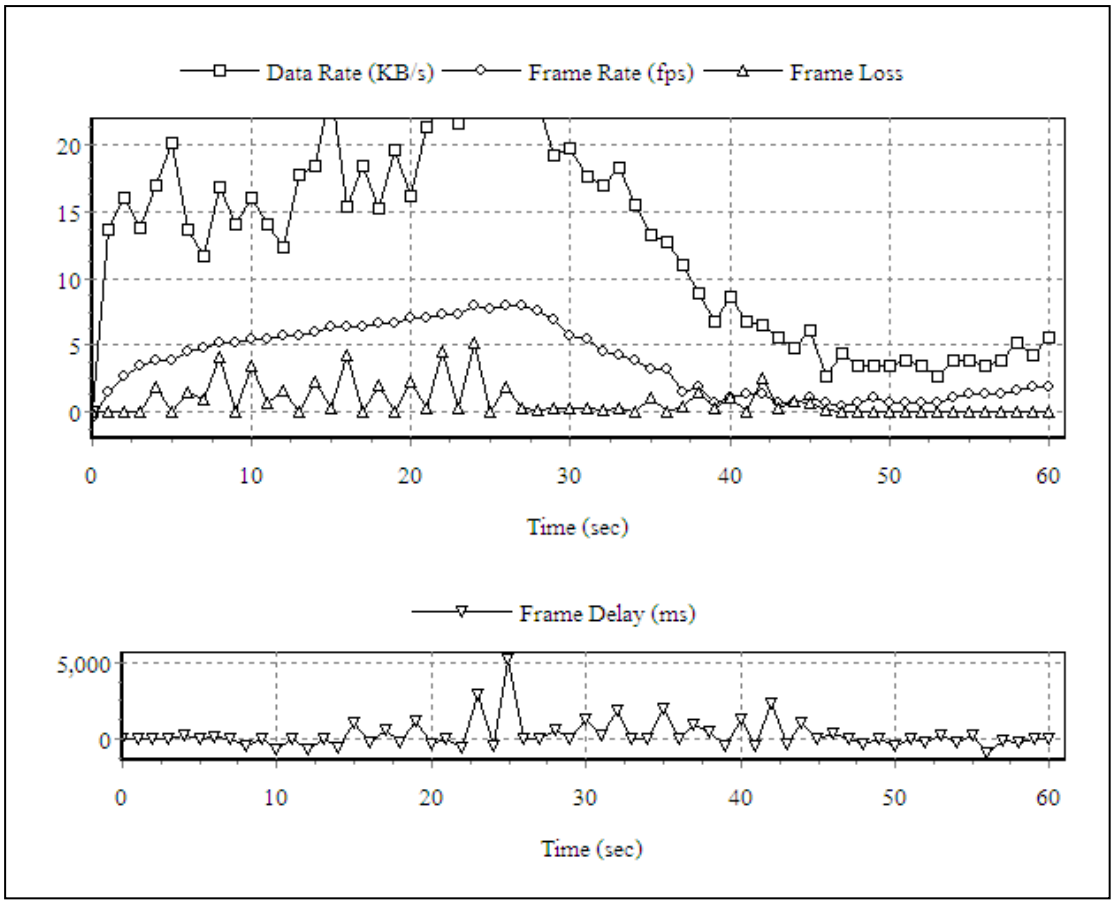
รูปที่ ข7 การส่งข้อมูลภาพผ่าน GPRS โดยให้คุณภาพของภาพมีค่าคงที่



รูปที่ ข8 การส่งข้อมูลภาพผ่าน GPRS โดยให้คุณภาพของภาพมีค่าคงที่
และมีการเพิ่มการใช้งานในเครือข่ายที่เวลา 20 วินาที

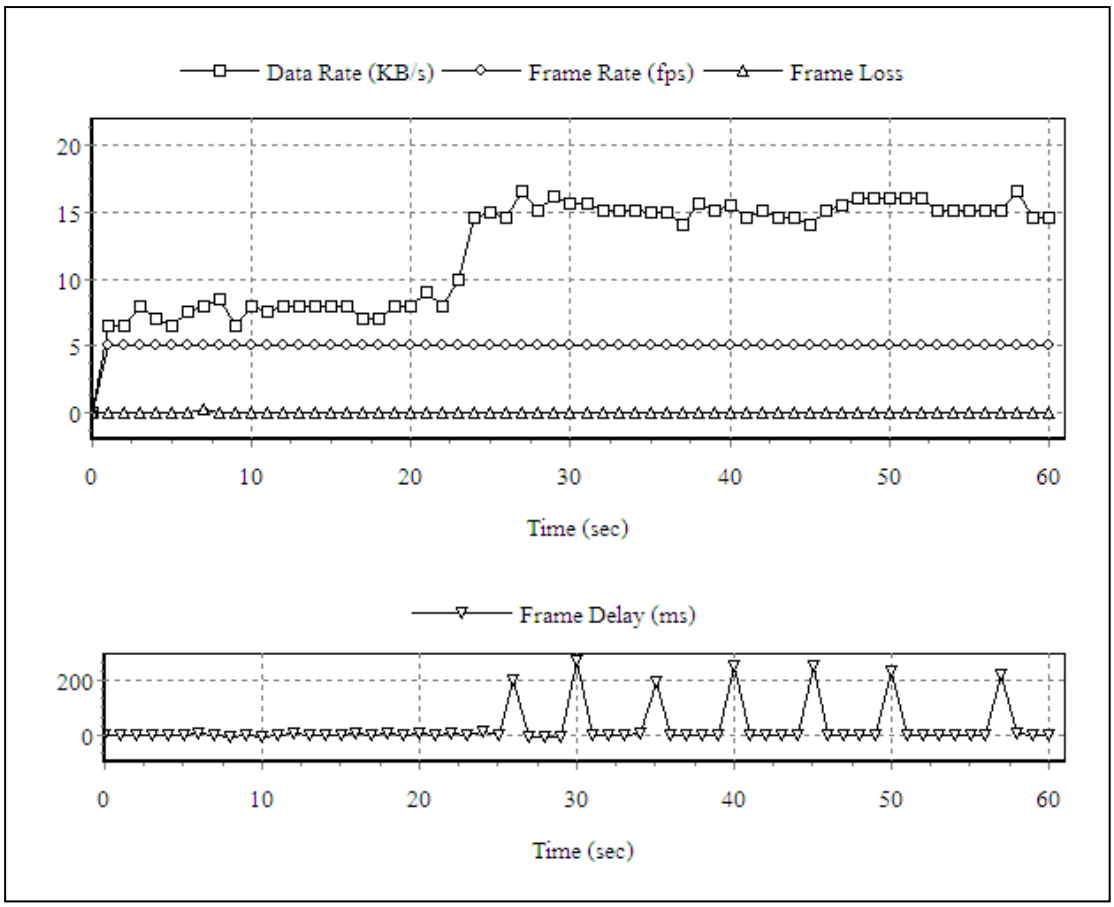


รูปที่ ข9 การส่งข้อมูลภาพผ่าน GPRS โดยมีการปรับคุณภาพของภาพ
และอัตราการส่งภาพร่วมกัน

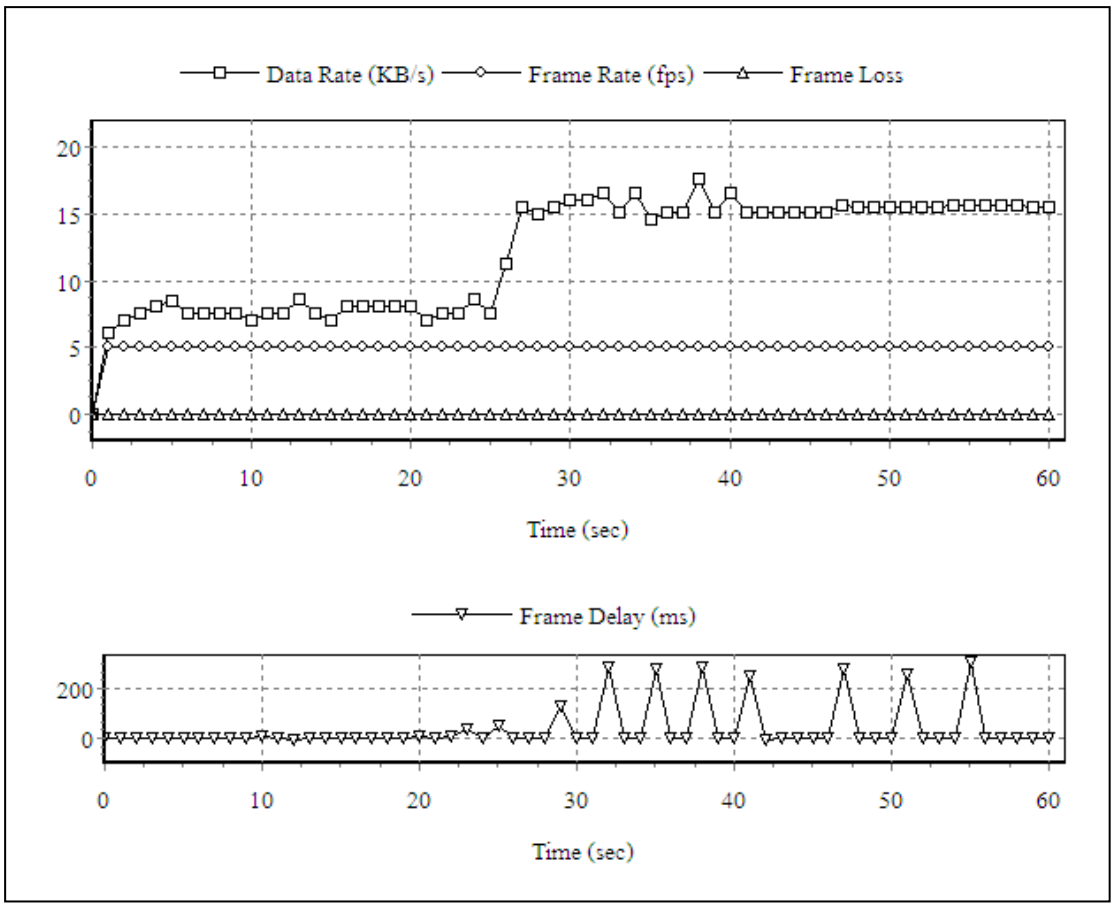


รูปที่ ข10 การส่งข้อมูลภาพผ่าน GPRS โดยมีการปรับคุณภาพของภาพและอัตราการส่งภาพร่วมกันและมีการใช้งานในเครือข่ายที่เวลา 20 วินาที

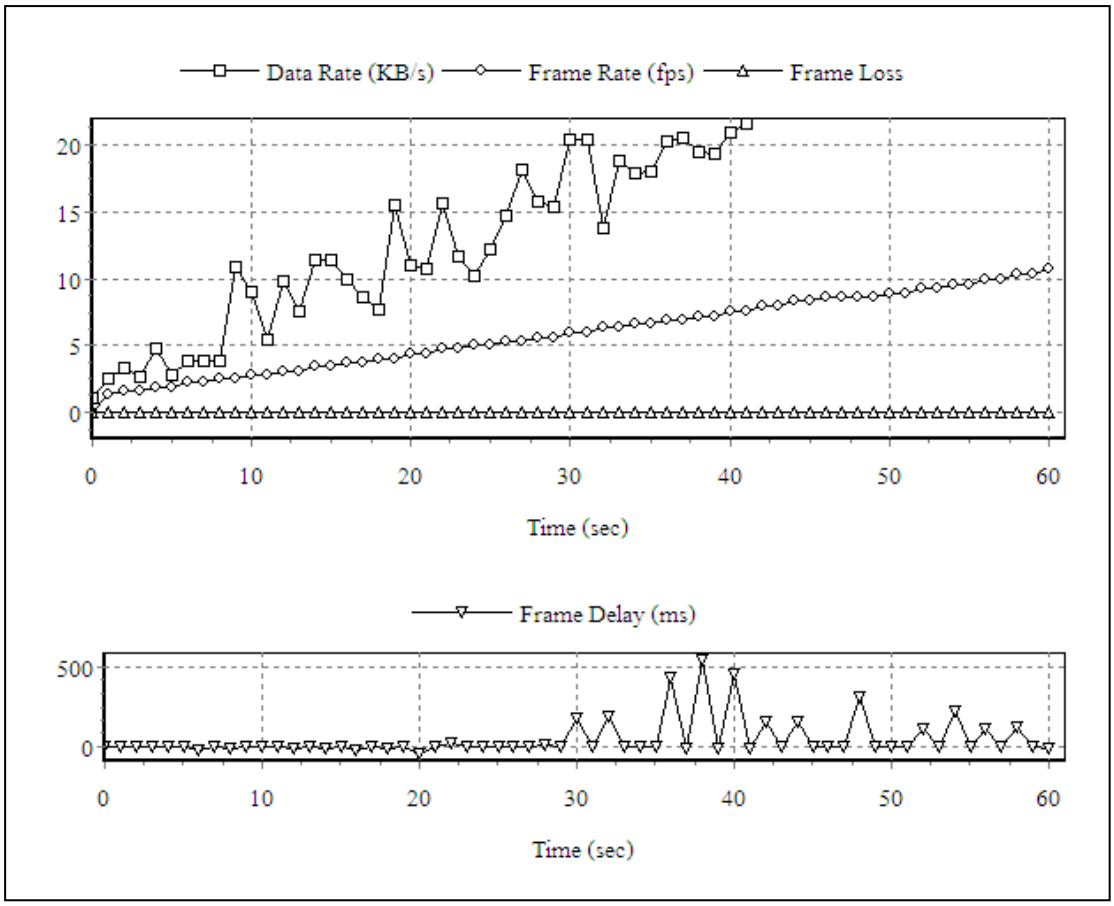
การทดลองครั้งที่ 3



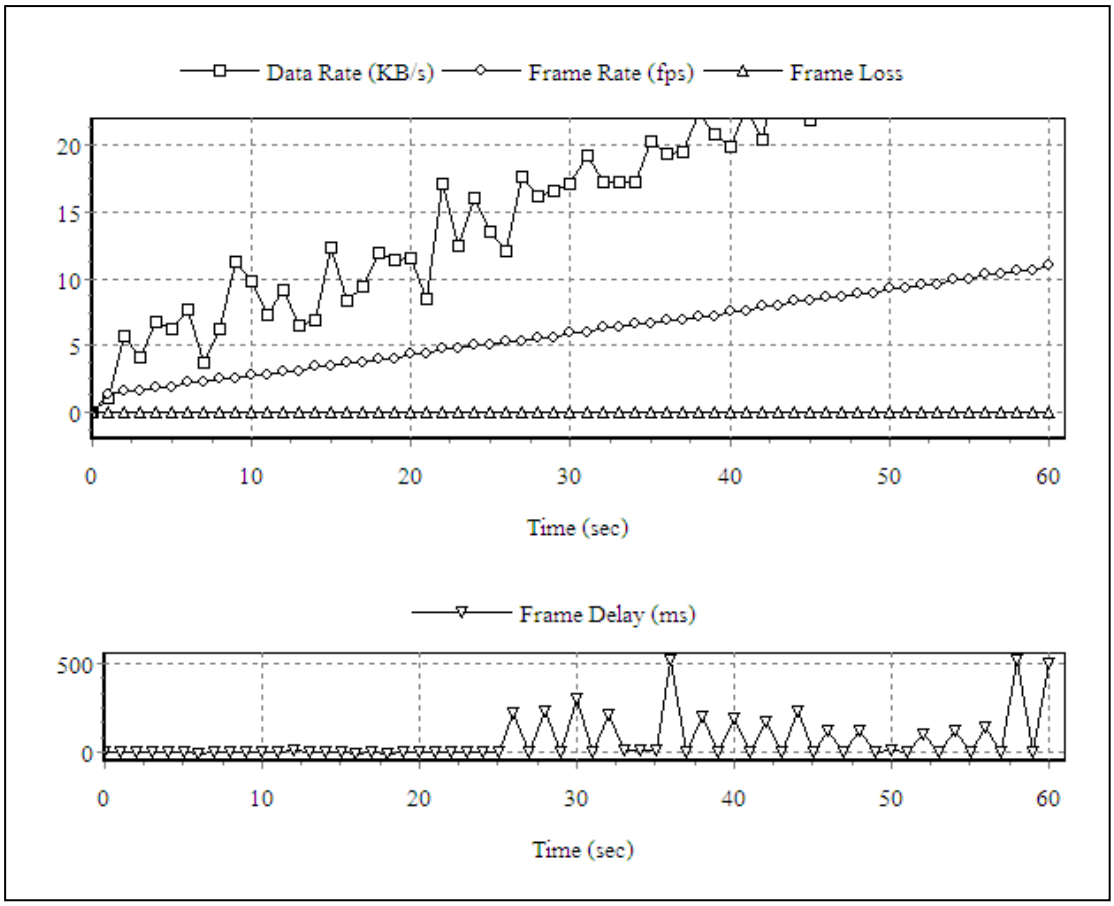
รูปที่ ข11 การส่งข้อมูลภาพผ่าน LAN โดยให้อัตราการส่งภาพมีค่าคงที่



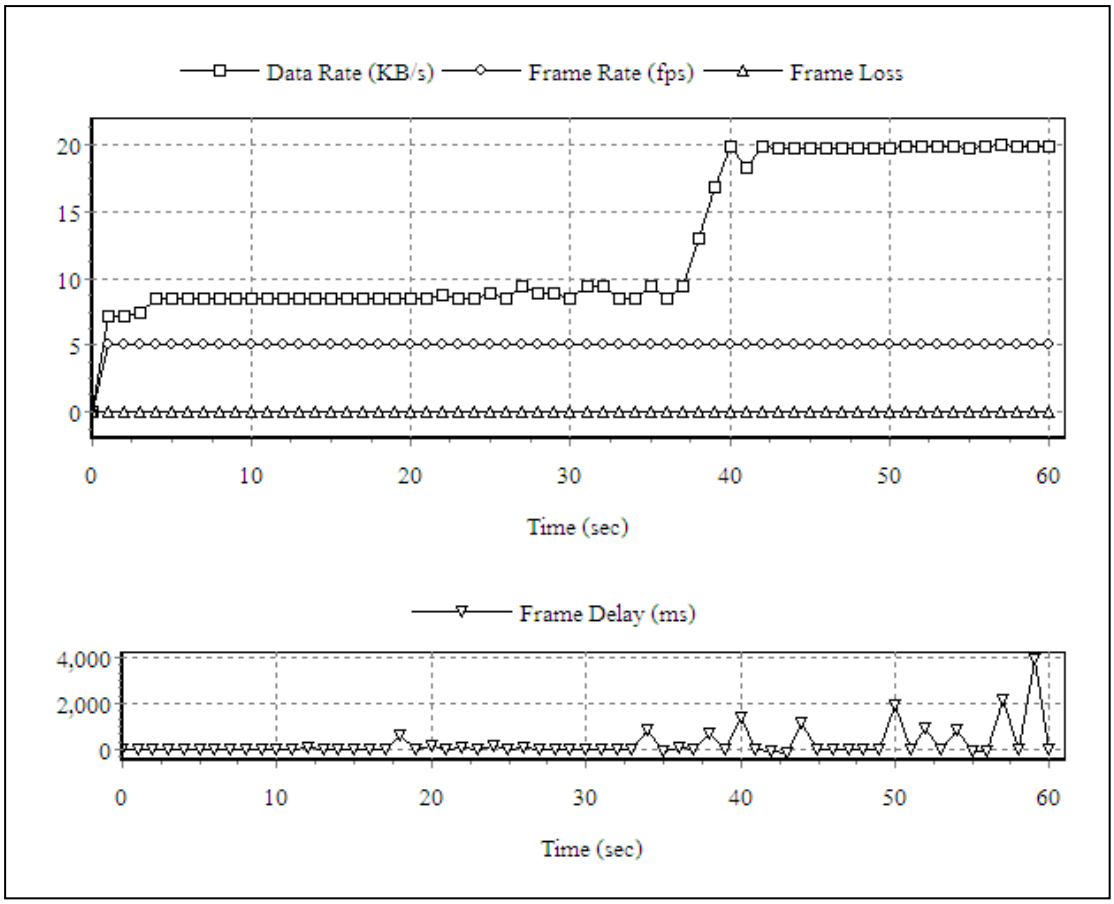
รูปที่ ข12 การส่งข้อมูลภาพผ่าน LAN โดยให้อัตราการส่งภาพมีค่าคงที่
และมีการเพิ่มการใช้งานในเครือข่ายที่เวลา 20 วินาที



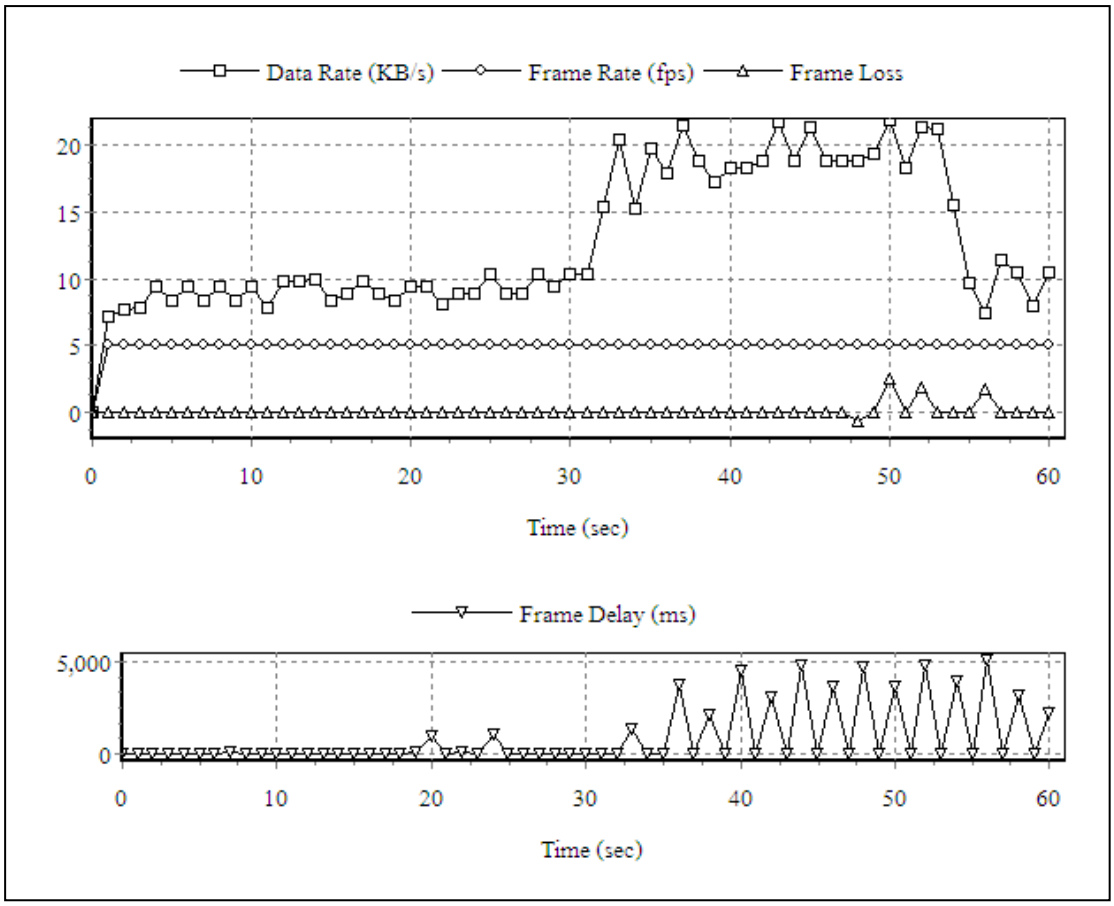
รูปที่ ข13 การส่งข้อมูลภาพผ่าน LAN โดยให้คุณภาพของภาพมีค่าคงที่



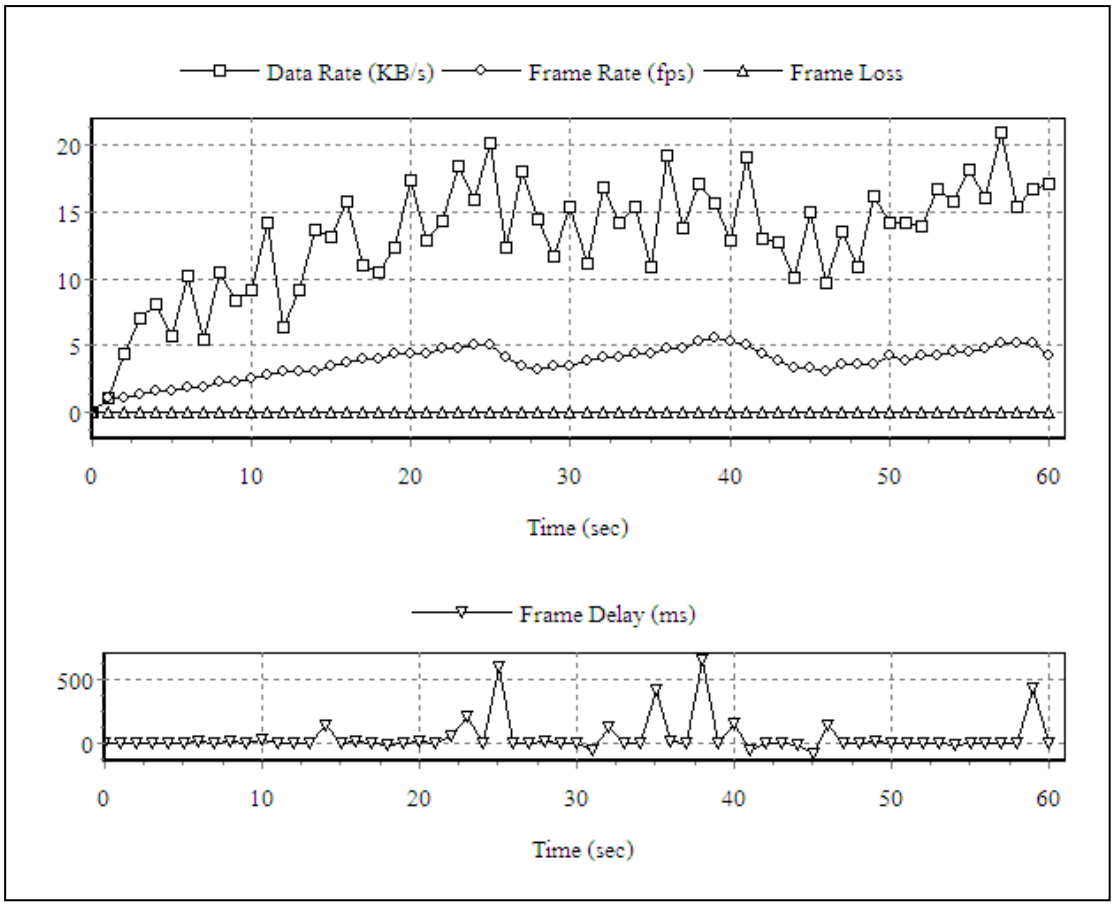
รูปที่ ข14 การส่งข้อมูลภาพผ่าน LAN โดยให้คุณภาพของภาพมีค่าคงที่
และมีการเพิ่มการใช้งานในเครือข่ายที่เวลา 20 วินาที



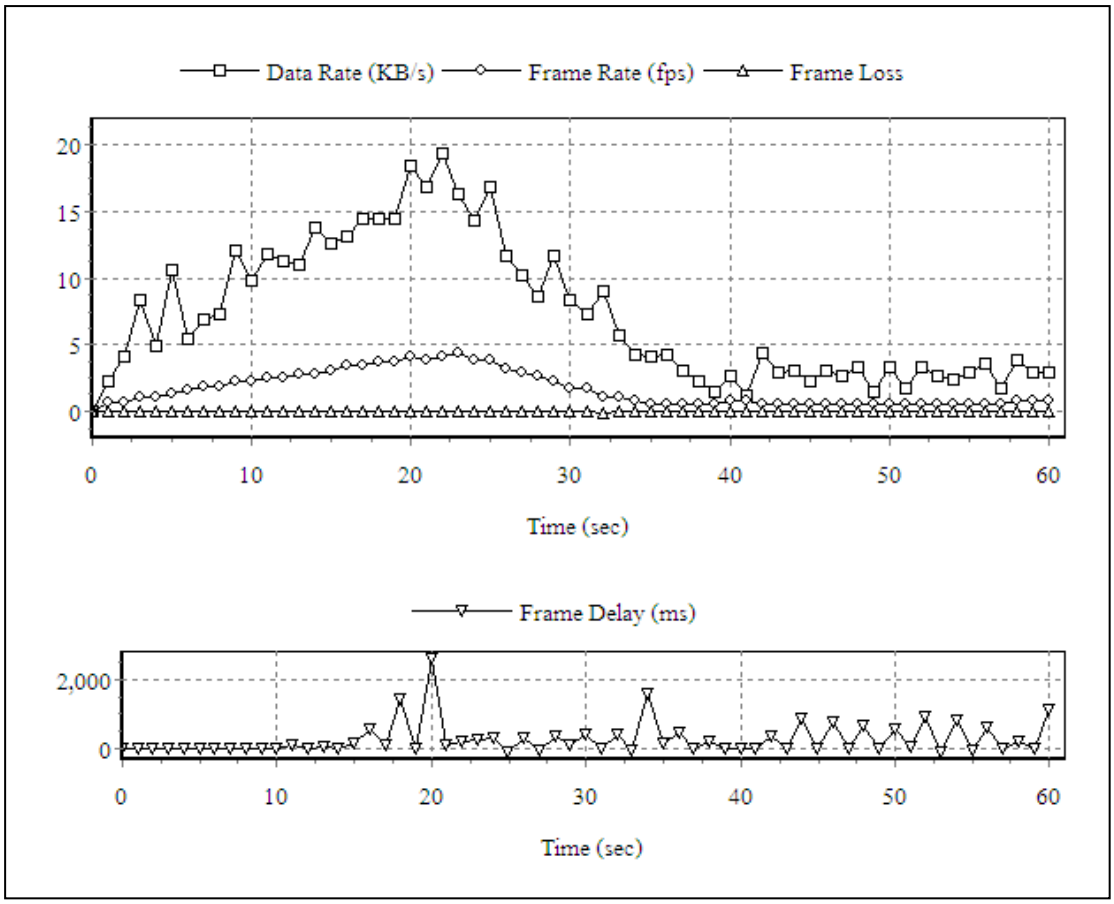
รูปที่ ข15 การส่งข้อมูลภาพผ่าน GPRS โดยให้อัตราการส่งภาพมีค่าคงที่



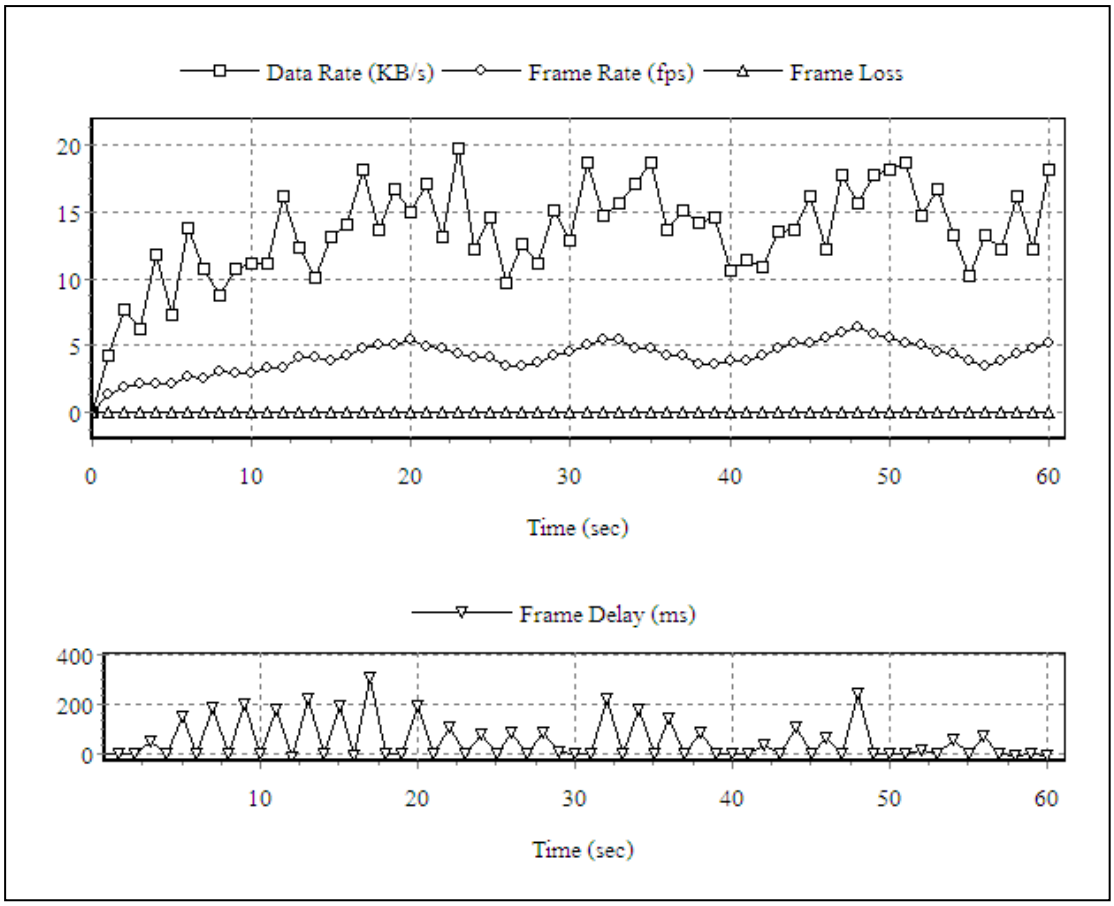
รูปที่ ข16 การส่งข้อมูลภาพผ่าน GPRS โดยให้อัตราการส่งภาพมีค่าคงที่
และมีการเพิ่มการใช้งานในเครือข่ายที่เวลา 20 วินาที



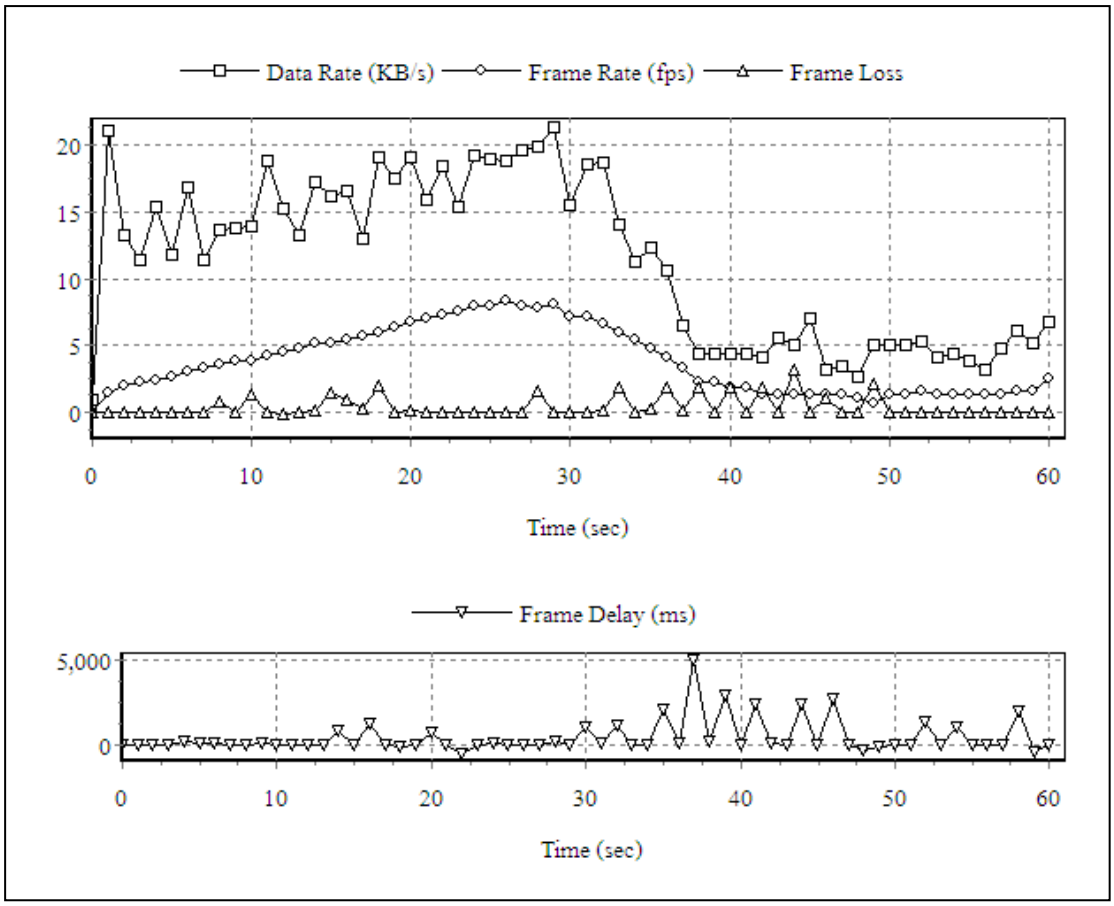
รูปที่ ข17 การส่งข้อมูลภาพผ่าน GPRS โดยให้คุณภาพของภาพมีค่าคงที่



รูปที่ ข18 การส่งข้อมูลภาพผ่าน GPRS โดยให้คุณภาพของภาพมีค่าคงที่
และมีการเพิ่มการใช้งานในเครือข่ายที่เวลา 20 วินาที



รูปที่ ข19 การส่งข้อมูลภาพผ่าน GPRS โดยมีการปรับคุณภาพของภาพ และอัตราการส่งภาพร่วมกัน



รูปที่ ข20 การส่งข้อมูลภาพผ่าน GPRS โดยมีการปรับคุณภาพของภาพและอัตราการส่งภาพร่วมกันและมีการใช้งานในเครือข่ายที่เวลา 20 วินาที

ภาคผนวก ค

ซอร์ซโค้ดการเข้ารหัส JPEG

```
/*****************************************************************************
   Filename : config.h
*****************************************************************************/
#ifndef CONFIG_H
#define CONFIG_H
#define    FOUR_ZERO_ZERO        0
#define    FOUR_TWO_ZERO         1
#define    FOUR_TWO_TWO          2
#define    FOUR_FOUR_FOUR        3
#define    RGB                   4
#define    BLOCK_SIZE            64
extern UINT8       Lqt [BLOCK_SIZE];
extern UINT8       Cqt [BLOCK_SIZE];
extern UINT16      ILqt [BLOCK_SIZE];
extern UINT16      ICqt [BLOCK_SIZE];
extern INT16       Y1 [BLOCK_SIZE];
extern INT16       Y2 [BLOCK_SIZE];
extern INT16       Y3 [BLOCK_SIZE];
extern INT16       Y4 [BLOCK_SIZE];
extern INT16       CB [BLOCK_SIZE];
extern INT16       CR [BLOCK_SIZE];
extern INT16       Temp [BLOCK_SIZE];
extern UINT32   lcode;
extern UINT16   bitindex;
#endif
/*****************************************************************************
    Filename : huffdata.h
*****************************************************************************/
#ifndef HUFFDATA_H
#define HUFFDATA_H
UINT16 luminance_dc_code_table [] = {
   0x0000, 0x0002, 0x0003, 0x0004, 0x0005, 0x0006,
   0x000E, 0x001E, 0x003E, 0x007E, 0x00FE, 0x01FE
};
```

```
UINT16 luminance_dc_size_table [] = {
   0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003,
   0x0004, 0x0005, 0x0006, 0x0007, 0x0008, 0x0009
};
UINT16 chrominance_dc_code_table [] = {
   0x0000, 0x0001, 0x0002, 0x0006, 0x000E, 0x001E,
   0x003E, 0x007E, 0x00FE, 0x01FE, 0x03FE, 0x07FE
};
UINT16 chrominance_dc_size_table [] = {
   0x0002, 0x0002, 0x0002, 0x0003, 0x0004, 0x0005,
   0x0006, 0x0007, 0x0008, 0x0009, 0x000A, 0x000B
};
UINT16 luminance_ac_code_table [] = {
   0x000A,  0x0000, 0x0001, 0x0004, 0x000B, 0x001A, 0x0078, 0x00F8, 0x03F6, 0xFF82, 0xFF83,
   0x000C, 0x001B, 0x0079, 0x01F6, 0x07F6, 0xFF84, 0xFF85, 0xFF86, 0xFF87, 0xFF88,
   0x001C, 0x00F9, 0x03F7, 0x0FF4, 0xFF89, 0xFF8A, 0xFF8b, 0xFF8C, 0xFF8D, 0xFF8E,
   0x003A, 0x01F7, 0x0FF5, 0xFF8F, 0xFF90, 0xFF91, 0xFF92, 0xFF93, 0xFF94, 0xFF95,
   0x003B, 0x03F8, 0xFF96, 0xFF97, 0xFF98, 0xFF99, 0xFF9A, 0xFF9B, 0xFF9C, 0xFF9D,
   0x007A, 0x07F7, 0xFF9E, 0xFF9F, 0xFFA0, 0xFFA1, 0xFFA2, 0xFFA3, 0xFFA4, 0xFFA5,
   0x007B, 0x0FF6, 0xFFA6, 0xFFA7, 0xFFA8, 0xFFA9, 0xFFAA, 0xFFAB, 0xFFAC, 0xFFAD,
   0x00FA, 0x0FF7, 0xFFAE, 0xFFAF, 0xFFB0, 0xFFB1, 0xFFB2, 0xFFB3, 0xFFB4, 0xFFB5,
   0x01F8, 0x7FC0, 0xFFB6, 0xFFB7, 0xFFB8, 0xFFB9, 0xFFBA, 0xFFBB, 0xFFBC, 0xFFBD,
   0x01F9, 0xFFBE, 0xFFBF, 0xFFC0, 0xFFC1, 0xFFC2, 0xFFC3, 0xFFC4, 0xFFC5, 0xFFC6,
   0x01FA, 0xFFC7, 0xFFC8, 0xFFC9, 0xFFCA, 0xFFCB, 0xFFCC, 0xFFCD, 0xFFCE, 0xFFCF,
   0x03F9, 0xFFD0, 0xFFD1, 0xFFD2, 0xFFD3, 0xFFD4, 0xFFD5, 0xFFD6, 0xFFD7, 0xFFD8,
   0x03FA, 0xFFD9, 0xFFDA, 0xFFDB, 0xFFDC, 0xFFDD, 0xFFDE, 0xFFDF, 0xFFE0, 0xFFE1,
   0x07F8, 0xFFE2, 0xFFE3, 0xFFE4, 0xFFE5, 0xFFE6, 0xFFE7, 0xFFE8, 0xFFE9, 0xFFEA,
   0xFFEB, 0xFFEC, 0xFFED, 0xFFEE, 0xFFEF, 0xFFF0, 0xFFF1, 0xFFF2, 0xFFF3, 0xFFF4,
   0xFFF5, 0xFFF6, 0xFFF7, 0xFFF8, 0xFFF9, 0xFFFA, 0xFFFB, 0xFFFC, 0xFFFD, 0xFFFE,
   0x07F9
};

UINT16 luminance_ac_size_table [] = {
   0x0004, 0x0002, 0x0002, 0x0003, 0x0004, 0x0005, 0x0007, 0x0008, 0x000A, 0x0010,
```

```
    0x0010, 0x0004, 0x0005, 0x0007, 0x0009, 0x000B, 0x0010, 0x0010, 0x0010, 0x0010,
    0x0010, 0x0005, 0x0008, 0x000A, 0x000C, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010,
    0x0010, 0x0006, 0x0009, 0x000C, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010,
    0x0010, 0x0006, 0x000A, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010,
    0x0010, 0x0007, 0x000B, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010,
    0x0010, 0x0007, 0x000C, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010,
    0x0010, 0x0008, 0x000C, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010,
    0x0010, 0x0009, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010,
    0x0010, 0x0009, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010,
    0x0010, 0x0009, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010,
    0x0010, 0x000A, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010,
    0x0010, 0x000A, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010,
    0x0010, 0x000B, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010,
    0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010,
    0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010,
    0x0010, 0x000B
};
UINT16 chrominance_ac_code_table [] = {
    0x0000,
    0x0001, 0x0004, 0x000A, 0x0018, 0x0019, 0x0038, 0x0078, 0x01F4, 0x03F6, 0x0FF4,
    0x000B, 0x0039, 0x00F6, 0x01F5, 0x07F6, 0x0FF5, 0xFF88, 0xFF89, 0xFF8A, 0xFF8B,
    0x001A, 0x00F7, 0x03F7, 0x0FF6, 0x7FC2, 0xFF8C, 0xFF8D, 0xFF8E, 0xFF8F, 0xFF90,
    0x001B, 0x00F8, 0x03F8, 0x0FF7, 0xFF91, 0xFF92, 0xFF93, 0xFF94, 0xFF95, 0xFF96,
    0x003A, 0x01F6, 0xFF97, 0xFF98, 0xFF99, 0xFF9A, 0xFF9B, 0xFF9C, 0xFF9D, 0xFF9E,
    0x003B, 0x03F9, 0xFF9F, 0xFFA0, 0xFFA1, 0xFFA2, 0xFFA3, 0xFFA4, 0xFFA5, 0xFFA6,
    0x0079, 0x07F7, 0xFFA7, 0xFFA8, 0xFFA9, 0xFFAA, 0xFFAB, 0xFFAC, 0xFFAD, 0xFFAE,
    0x007A, 0x07F8, 0xFFAF, 0xFFB0, 0xFFB1, 0xFFB2, 0xFFB3, 0xFFB4, 0xFFB5, 0xFFB6,
    0x00F9, 0xFFB7, 0xFFB8, 0xFFB9, 0xFFBA, 0xFFBB, 0xFFBC, 0xFFBD, 0xFFBE, 0xFFBF,
    0x01F7, 0xFFC0, 0xFFC1, 0xFFC2, 0xFFC3, 0xFFC4, 0xFFC5, 0xFFC6, 0xFFC7, 0xFFC8,
    0x01F8, 0xFFC9, 0xFFCA, 0xFFCB, 0xFFCC, 0xFFCD, 0xFFCE, 0xFFCF, 0xFFD0, 0xFFD1,
    0x01F9, 0xFFD2, 0xFFD3, 0xFFD4, 0xFFD5, 0xFFD6, 0xFFD7, 0xFFD8, 0xFFD9, 0xFFDA,
    0x01FA, 0xFFDB, 0xFFDC, 0xFFDD, 0xFFDE, 0xFFDF, 0xFFE0, 0xFFE1, 0xFFE2, 0xFFE3,
    0x07F9, 0xFFE4, 0xFFE5, 0xFFE6, 0xFFE7, 0xFFE8, 0xFFE9, 0xFFEA, 0xFFEb, 0xFFEC,
    0x3FE0, 0xFFED, 0xFFEE, 0xFFEF, 0xFFF0, 0xFFF1, 0xFFF2, 0xFFF3, 0xFFF4, 0xFFF5,
```

```
    0x7FC3, 0xFFF6, 0xFFF7, 0xFFF8, 0xFFF9, 0xFFFA, 0xFFFB, 0xFFFC, 0xFFFD, 0xFFFE,
    0x03FA
};
UINT16 chrominance_ac_size_table [] = {
    0x0002,
    0x0002, 0x0003, 0x0004, 0x0005, 0x0005, 0x0006, 0x0007, 0x0009, 0x000A, 0x000C,
    0x0004, 0x0006, 0x0008, 0x0009, 0x000B, 0x000C, 0x0010, 0x0010, 0x0010, 0x0010,
    0x0005, 0x0008, 0x000A, 0x000C, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010,
    0x0005, 0x0008, 0x000A, 0x000C, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010,
    0x0006, 0x0009, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010,
    0x0006, 0x000A, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010,
    0x0007, 0x000B, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010,
    0x0007, 0x000B, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010,
    0x0008, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010,
    0x0009, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010,
    0x0009, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010,
    0x0009, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010,
    0x0009, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010,
    0x000B, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010,
    0x000E, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010,
    0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010,
    0x000A
};
UINT8 bitsize [] = {
    0, 1, 2, 2, 3, 3, 3, 3,
    4, 4, 4, 4, 4, 4, 4, 4,
    5, 5, 5, 5, 5, 5, 5, 5,
    5, 5, 5, 5, 5, 5, 5, 5,
    6, 6, 6, 6, 6, 6, 6, 6,
    6, 6, 6, 6, 6, 6, 6, 6,
    6, 6, 6, 6, 6, 6, 6, 6,
    6, 6, 6, 6, 6, 6, 6, 6,
    7, 7, 7, 7, 7, 7, 7, 7,
    7, 7, 7, 7, 7, 7, 7, 7,
```

```
    7, 7, 7, 7, 7, 7, 7, 7,
    7, 7, 7, 7, 7, 7, 7, 7,
    7, 7, 7, 7, 7, 7, 7, 7,
    7, 7, 7, 7, 7, 7, 7, 7,
    7, 7, 7, 7, 7, 7, 7, 7,
    7, 7, 7, 7, 7, 7, 7, 7,
    8, 8, 8, 8, 8, 8, 8, 8,
    8, 8, 8, 8, 8, 8, 8, 8,
    8, 8, 8, 8, 8, 8, 8, 8,
    8, 8, 8, 8, 8, 8, 8, 8,
    8, 8, 8, 8, 8, 8, 8, 8,
    8, 8, 8, 8, 8, 8, 8, 8,
    8, 8, 8, 8, 8, 8, 8, 8,
    8, 8, 8, 8, 8, 8, 8, 8,
    8, 8, 8, 8, 8, 8, 8, 8,
    8, 8, 8, 8, 8, 8, 8, 8,
    8, 8, 8, 8, 8, 8, 8, 8,
    8, 8, 8, 8, 8, 8, 8, 8,
    8, 8, 8, 8, 8, 8, 8, 8,
    8, 8, 8, 8, 8, 8, 8, 8,
    8, 8, 8, 8, 8, 8, 8, 8
};
#endif
/*****************************************************************************
   Filename : jdatatype.h
*****************************************************************************/
#ifndef JDATATYPE_H
#define JDATATYPE_H
typedef struct JPEG_ENCODER_STRUCTURE
{
   UINT16  mcu_width;
   UINT16  mcu_height;
   UINT16  horizontal_mcus;
```

```
    UINT16  vertical_mcus;
    UINT16  cols_in_right_mcus;
    UINT16  rows_in_bottom_mcus;
    UINT16  rows;
    UINT16  cols;
    UINT16  length_minus_mcu_width;
    UINT16  length_minus_width;
    UINT16  incr;
    UINT16  mcu_width_size;
    UINT16  offset;
    INT16 ldc1;
    INT16 ldc2;
    INT16 ldc3;
} JPEG_ENCODER_STRUCTURE;
#endif
/*****************************************************************************
    Filename : markdata.h
*****************************************************************************/
#ifndef MARKDATA_H
#define MARKDATA_H
UINT16 markerdata [] = {
    // dht
    0xFFC4, 0x1A2, 0x00,
    // luminance dc (2 - 16) + 1
    0x0105, 0x0101, 0x00101, 0x0101, 0x0000, 0x00000, 00000, 00000,
    // luminance dc (2 - 12) + 1
    0x0102, 0x0304, 0x0506, 0x0708, 0x090A, 0x0B01,
    // chrominance dc (1 - 16)
    0x0003, 0x0101, 0x0101, 0x0101, 0x0101, 0x0100, 0x0000, 0x0000,
    // chrominance dc (1 - 12)
    0x0001, 0x00203, 0x0405, 0x0607, 0x0809, 0x00A0B,
    // luminance ac 1 + (1 - 15)
    0x1000, 0x0201, 0x0303, 0x0204, 0x0305, 0x0504, 0x0400, 0x0001,
    // luminance ac 1 + (1 - 162) + 1
```

```
    0x7D01, 0x0203, 0x0004, 0x1105, 0x1221, 0x3141, 0x0613, 0x5161,
    0x0722, 0x7114, 0x3281, 0x91A1, 0x0823, 0x42B1, 0xC115, 0x52D1,
    0xF024, 0x3362, 0x7282, 0x090A, 0x1617, 0x1819, 0x1A25, 0x2627,
    0x2829, 0x2A34, 0x3536, 0x3738, 0x393A, 0x4344, 0x4546, 0x4748,
    0x494A, 0x5354, 0x5556, 0x5758, 0x595A, 0x6364, 0x6566, 0x6768,
    0x696A, 0x7374, 0x7576, 0x7778, 0x797A, 0x8384, 0x8586, 0x8788,
    0x898A, 0x9293, 0x9495, 0x9697, 0x9899, 0x9AA2, 0xA3A4, 0xA5A6,
    0xA7A8, 0xA9AA, 0xB2B3, 0xB4B5, 0xB6B7, 0xB8B9, 0xBAC2, 0xC3C4,
    0xC5C6, 0xC7C8, 0xC9CA, 0xD2D3, 0xD4D5, 0xD6D7, 0xD8D9, 0xDAE1,
    0xE2E3, 0xE4E5, 0xE6E7, 0xE8E9, 0xEAF1, 0xF2F3, 0xF4F5, 0xF6F7,
    0xF8F9, 0xFA11,
    // chrominance ac (1 - 16)
    0x0002, 0x0102, 0x0404, 0x0304, 0x0705, 0x0404, 0x0001, 0x0277,
    // chrominance ac (1 - 162)
    0x0001, 0x0203, 0x1104, 0x0521, 0x3106, 0x1241, 0x5107, 0x6171,
    0x1322, 0x3281, 0x0814, 0x4291, 0xA1B1, 0xC109, 0x2333, 0x52F0,
    0x1562, 0x72D1, 0x0A16, 0x2434, 0xE125, 0xF117, 0x1819, 0x1A26,
    0x2728, 0x292A, 0x3536, 0x3738, 0x393A, 0x4344, 0x4546, 0x4748,
    0x494A, 0x5354, 0x5556, 0x5758, 0x595A, 0x6364, 0x6566, 0x6768,
    0x696A, 0x7374, 0x7576, 0x7778, 0x797A, 0x8283, 0x8485, 0x8687,
    0x8889, 0x8A92, 0x9394, 0x9596, 0x9798, 0x999A, 0xA2A3, 0xA4A5,
    0xA6A7, 0xA8A9, 0xAAB2, 0xB3B4, 0xB5B6, 0xB7B8, 0xB9BA, 0xC2C3,
    0xC4C5, 0xC6C7, 0xC8C9, 0xCAD2, 0xD3D4, 0xD5D6, 0xD7D8, 0xD9DA,
    0xE2E3, 0xE4E5, 0xE6E7, 0xE8E9, 0xEAF2, 0xF3F4, 0xF5F6, 0xF7F8,
    0xF9FA
};
#endif
/*******************************************************************************
   Filename : prototype.h
*******************************************************************************/
#ifndef PROTOTYPE_H
#define PROTOTYPE_H
void initialization (JPEG_ENCODER_STRUCTURE *, UINT32, UINT32, UINT32);
UINT16 DSP_Division (UINT32, UINT32);
```

```
void initialize_quantization_tables (UINT32);
UINT8* write_markers (UINT8 *, UINT32, UINT32, UINT32);
UINT8* encode_image (UINT8 *,UINT8 *, UINT32, UINT32, UINT32, UINT32);
void read_400_format (JPEG_ENCODER_STRUCTURE *, UINT8 *);
void read_420_format (JPEG_ENCODER_STRUCTURE *, UINT8 *);
void read_422_format (JPEG_ENCODER_STRUCTURE *, UINT8 *);
void read_444_format (JPEG_ENCODER_STRUCTURE *, UINT8 *);
void RGB_2_444 (UINT8 *, UINT8 *, UINT32, UINT32);
UINT8* encodeMCU (JPEG_ENCODER_STRUCTURE *, UINT32, UINT8 *);
void levelshift (INT16 *);
void DCT (INT16 *);
void quantization (INT16 *, UINT16 *);
UINT8* huffman (JPEG_ENCODER_STRUCTURE *, UINT16, UINT8 *);
UINT8* close_bitstream (UINT8 *);
void* jemalloc(size_t);
//void* jefree(void);
#endif
/*****************************************************************************
   Filename : quantdata.h
*****************************************************************************/
#ifndef QUANTDATA_H
#define QUANTDATA_H
UINT8 zigzag_table [] = {
   0,  1,   5,  6, 14, 15, 27, 28,
   2,  4,   7, 13, 16, 26, 29, 42,
   3,  8,  12, 17, 25, 30, 41, 43,
   9,  11, 18, 24, 31, 40, 44, 53,
   10, 19, 23, 32, 39, 45, 52, 54,
   20, 22, 33, 38, 46, 51, 55, 60,
   21, 34, 37, 47, 50, 56, 59, 61,
   35, 36, 48, 49, 57, 58, 62, 63
};
#endif
/*****************************************************************************
```

```
   Filename : dtc.c
*******************************************************************************/
#include "datatype.h"
/* Level shifting to get 8 bit SIGNED values for the data  */
void levelshift (INT16* const data)
{
   INT16 i;
   for (i=63; i>=0; i--)
      data [i] -= 128;
}
/* DCT for One block(8x8) */
void DCT (INT16 *data)
{
   UINT16 i;
   INT32 x0, x1, x2, x3, x4, x5, x6, x7, x8;
   /*  All values are shifted left by 10  and rounded off to nearest integer */
   static const UINT16 c1=1420;    /* cos PI/16 * root(2)  */
   static const UINT16 c2=1338;    /* cos PI/8 * root(2)   */
   static const UINT16 c3=1204;    /* cos 3PI/16 * root(2) */
   static const UINT16 c5=805;     /* cos 5PI/16 * root(2) */
   static const UINT16 c6=554;     /* cos 3PI/8 * root(2)  */
   static const UINT16 c7=283;     /* cos 7PI/16 * root(2) */

   static const UINT16 s1=3;
   static const UINT16 s2=10;
   static const UINT16 s3=13;

   for (i=8; i>0; i--)
   {
      x8 = data [0] + data [7];
      x0 = data [0] - data [7];

      x7 = data [1] + data [6];
      x1 = data [1] - data [6];
```

```
    x6 = data [2] + data [5];
    x2 = data [2] - data [5];
    x5 = data [3] + data [4];
    x3 = data [3] - data [4];
    x4 = x8 + x5;
    x8 -= x5;
    x5 = x7 + x6;
    x7 -= x6;

    data [0] = (INT16) (x4 + x5);
    data [4] = (INT16) (x4 - x5);
    data [2] = (INT16) ((x8*c2 + x7*c6) >> s2);
    data [6] = (INT16) ((x8*c6 - x7*c2) >> s2);
    data [7] = (INT16) ((x0*c7 - x1*c5 + x2*c3 - x3*c1) >> s2);
    data [5] = (INT16) ((x0*c5 - x1*c1 + x2*c7 + x3*c3) >> s2);
    data [3] = (INT16) ((x0*c3 - x1*c7 - x2*c1 - x3*c5) >> s2);
    data [1] = (INT16) ((x0*c1 + x1*c3 + x2*c5 + x3*c7) >> s2);
    data += 8;
}
data -= 64;
for (i=8; i>0; i--)
{
    x8 = data [0] + data [56];
    x0 = data [0] - data [56];

    x7 = data [8] + data [48];
    x1 = data [8] - data [48];

    x6 = data [16] + data [40];
    x2 = data [16] - data [40];

    x5 = data [24] + data [32];
    x3 = data [24] - data [32];
```

```
        x4 = x8 + x5;
        x8 -= x5;

        x5 = x7 + x6;
        x7 -= x6;

        data [0] = (INT16) ((x4 + x5) >> s1);
        data [32] = (INT16) ((x4 - x5) >> s1);

        data [16] = (INT16) ((x8*c2 + x7*c6) >> s3);
        data [48] = (INT16) ((x8*c6 - x7*c2) >> s3);

        data [56] = (INT16) ((x0*c7 - x1*c5 + x2*c3 - x3*c1) >> s3);
        data [40] = (INT16) ((x0*c5 - x1*c1 + x2*c7 + x3*c3) >> s3);
        data [24] = (INT16) ((x0*c3 - x1*c7 - x2*c1 - x3*c5) >> s3);
        data [8] = (INT16) ((x0*c1 + x1*c3 + x2*c5 + x3*c7) >> s3);

        data++;
    }
}
/*****************************************************************************
   Filename : encoder.c
*****************************************************************************/
#include "datatype.h"
#include "jdatatype.h"
#include "config.h"
#include "prototype.h"
UINT8   Lqt [BLOCK_SIZE];
UINT8   Cqt [BLOCK_SIZE];
UINT16  ILqt [BLOCK_SIZE];
UINT16  ICqt [BLOCK_SIZE];
INT16   Y1 [BLOCK_SIZE];
INT16   Y2 [BLOCK_SIZE];
```

```
INT16   Y3 [BLOCK_SIZE];
INT16   Y4 [BLOCK_SIZE];
INT16   CB [BLOCK_SIZE];
INT16   CR [BLOCK_SIZE];
INT16   Temp [BLOCK_SIZE];
UINT32 lcode = 0;
UINT16 bitindex = 0;
void (*read_format) (JPEG_ENCODER_STRUCTURE *jpeg_encoder_structure, UINT8 *input_ptr);
void initialization (JPEG_ENCODER_STRUCTURE *jpeg, UINT32 image_format, UINT32 image_width,
UINT32 image_height)
{
   UINT16 mcu_width, mcu_height, bytes_per_pixel;
   if (image_format == FOUR_ZERO_ZERO || image_format == FOUR_FOUR_FOUR)
   {
      jpeg->mcu_width = mcu_width = 8;
      jpeg->mcu_height = mcu_height = 8;
      jpeg->horizontal_mcus = (UINT16) ((image_width + mcu_width - 1) >> 3);
      jpeg->vertical_mcus = (UINT16) ((image_height + mcu_height - 1) >> 3);
      if (image_format == FOUR_ZERO_ZERO)
      {
         bytes_per_pixel = 1;
         read_format = read_400_format;
      }
      else
      {
         bytes_per_pixel = 3;
         read_format = read_444_format;
      }
   }
   else
   {
      jpeg->mcu_width = mcu_width = 16;
      jpeg->horizontal_mcus = (UINT16) ((image_width + mcu_width - 1) >> 4);
      if (image_format == FOUR_TWO_ZERO)
```

```
        {
            jpeg->mcu_height = mcu_height = 16;
            jpeg->vertical_mcus = (UINT16) ((image_height + mcu_height - 1) >> 4);
            bytes_per_pixel = 3;
            read_format = read_420_format;
        }
        else
        {
            jpeg->mcu_height = mcu_height = 8;
            jpeg->vertical_mcus = (UINT16) ((image_height + mcu_height - 1) >> 3);
            bytes_per_pixel = 2;
            read_format = read_422_format;
        }
    }
    jpeg->rows_in_bottom_mcus = (UINT16) (image_height - (jpeg->vertical_mcus - 1) * mcu_height);
    jpeg->cols_in_right_mcus = (UINT16) (image_width - (jpeg->horizontal_mcus - 1) * mcu_width);
    jpeg->length_minus_mcu_width = (UINT16) ((image_width - mcu_width) * bytes_per_pixel);
    jpeg->length_minus_width = (UINT16) ((image_width - jpeg->cols_in_right_mcus) * bytes_per_pixel);
    jpeg->mcu_width_size = (UINT16) (mcu_width * bytes_per_pixel);
    if (image_format != FOUR_TWO_ZERO)
        jpeg->offset = (UINT16) ((image_width * (mcu_height - 1)
                        - (mcu_width - jpeg->cols_in_right_mcus)) * bytes_per_pixel);
    else
        jpeg->offset = (UINT16) ((image_width * ((mcu_height >> 1) - 1)
                        - (mcu_width - jpeg->cols_in_right_mcus)) * bytes_per_pixel);
    jpeg->ldc1 = 0;
    jpeg->ldc2 = 0;
    jpeg->ldc3 = 0;
}
UINT8* encode_image (UINT8 *input_ptr,UINT8 *output_ptr, UINT32 quality_factor,
                UINT32 image_format, UINT32 image_width, UINT32 image_height)
{
    UINT16 i, j;
    JPEG_ENCODER_STRUCTURE JpegStruct;
```

```
JPEG_ENCODER_STRUCTURE *jpeg_encoder_structure = &JpegStruct;
if (image_format == RGB)
{
   image_format = FOUR_FOUR_FOUR;
   RGB_2_444 (input_ptr, output_ptr, image_width, image_height);
}
/* Initialization of JPEG control structure */
initialization (jpeg_encoder_structure,image_format,image_width,image_height);
/* Quantization Table Initialization */
initialize_quantization_tables (quality_factor);
/* Writing Marker Data */
output_ptr = write_markers (output_ptr, image_format, image_width, image_height);
for (i=1; i<=jpeg_encoder_structure->vertical_mcus; i++)
{
   if (i < jpeg_encoder_structure->vertical_mcus)
      jpeg_encoder_structure->rows = jpeg_encoder_structure->mcu_height;
   else
      jpeg_encoder_structure->rows = jpeg_encoder_structure->rows_in_bottom_mcus;
   for (j=1; j<=jpeg_encoder_structure->horizontal_mcus; j++)
   {
      if (j < jpeg_encoder_structure->horizontal_mcus)
      {
         jpeg_encoder_structure->cols = jpeg_encoder_structure->mcu_width;
         jpeg_encoder_structure->incr = jpeg_encoder_structure->length_minus_mcu_width;
      }
      else
      {
         jpeg_encoder_structure->cols = jpeg_encoder_structure->cols_in_right_mcus;
         jpeg_encoder_structure->incr = jpeg_encoder_structure->length_minus_width;
      }

      read_format (jpeg_encoder_structure, input_ptr);
      /* Encode the data in MCU */
      output_ptr = encodeMCU (jpeg_encoder_structure, image_format, output_ptr);
```

```
            input_ptr += jpeg_encoder_structure->mcu_width_size;
        }
        input_ptr += jpeg_encoder_structure->offset;
    }
    /* Close Routine */
    close_bitstream (output_ptr);
    return output_ptr;
}
UINT8* encodeMCU (JPEG_ENCODER_STRUCTURE *jpeg_encoder_structure,
                UINT32 image_format, UINT8 *output_ptr)
{
    levelshift (Y1);
    DCT (Y1);
    quantization (Y1, ILqt);
    output_ptr = huffman (jpeg_encoder_structure, 1, output_ptr);
    if (image_format == FOUR_TWO_ZERO || image_format == FOUR_TWO_TWO)
    {
        levelshift (Y2);
        DCT (Y2);
        quantization (Y2, ILqt);
        output_ptr = huffman (jpeg_encoder_structure, 1, output_ptr);
        if (image_format == FOUR_TWO_ZERO)
        {
            levelshift (Y3);
            DCT (Y3);
            quantization (Y3, ILqt);
            output_ptr = huffman (jpeg_encoder_structure, 1, output_ptr);

            levelshift (Y4);
            DCT (Y4);
            quantization (Y4, ILqt);
            output_ptr = huffman (jpeg_encoder_structure, 1, output_ptr);
        }
    }
```

```
    if (image_format != FOUR_ZERO_ZERO)
    {
        levelshift (CB);
        DCT (CB);
        quantization (CB, ICqt);
        output_ptr = huffman (jpeg_encoder_structure, 2, output_ptr);

        levelshift (CR);
        DCT (CR);
        quantization (CR, ICqt);
        output_ptr = huffman (jpeg_encoder_structure, 3, output_ptr);
    }
    return output_ptr;
}
/*****************************************************************************
    Filename : Huffman.c
*****************************************************************************/
#include "datatype.h"
#include "jdatatype.h"
#include "config.h"
#include "prototype.h"
#include "huffdata.h"
#define PUTBITS \
{   \
    bits_in_next_word = (INT16) (bitindex + numbits - 32);  \
    if (bits_in_next_word < 0)  \
    {   \
        lcode = (lcode << numbits) | data;  \
        bitindex += numbits;    \
    }   \
    else    \
    {   \
        lcode = (lcode << (32 - bitindex)) | (data >> bits_in_next_word);   \
        if ((*output_ptr++ = (UINT8)(lcode >> 24)) == 0xff) \
```

```
            *output_ptr++ = 0;  \
        if ((*output_ptr++ = (UINT8)(lcode >> 16)) == 0xff) \
            *output_ptr++ = 0;  \
        if ((*output_ptr++ = (UINT8)(lcode >> 8)) == 0xff)  \
            *output_ptr++ = 0;  \
        if ((*output_ptr++ = (UINT8) lcode) == 0xff)    \
            *output_ptr++ = 0;  \
        lcode = data;   \
        bitindex = bits_in_next_word;   \
    }   \
}
UINT8* huffman (JPEG_ENCODER_STRUCTURE *jpeg_encoder_structure, UINT16 component, UINT8
*output_ptr)
{
    UINT16 i;
    UINT16 *DcCodeTable, *DcSizeTable, *AcCodeTable, *AcSizeTable;

    INT16 *Temp_Ptr, Coeff, LastDc;
    UINT16 AbsCoeff, HuffCode, HuffSize, RunLength=0, DataSize=0, index;

    INT16 bits_in_next_word;
    UINT16 numbits;
    UINT32 data;
    Temp_Ptr = Temp;
    Coeff = *Temp_Ptr++;
    if (component == 1)
    {
        DcCodeTable = luminance_dc_code_table;
        DcSizeTable = luminance_dc_size_table;
        AcCodeTable = luminance_ac_code_table;
        AcSizeTable = luminance_ac_size_table;

        LastDc = jpeg_encoder_structure->ldc1;
        jpeg_encoder_structure->ldc1 = Coeff;
```

```
}
else
{
   DcCodeTable = chrominance_dc_code_table;
   DcSizeTable = chrominance_dc_size_table;
   AcCodeTable = chrominance_ac_code_table;
   AcSizeTable = chrominance_ac_size_table;
   if (component == 2)
   {
      LastDc = jpeg_encoder_structure->ldc2;
      jpeg_encoder_structure->ldc2 = Coeff;
   }
   else
   {
      LastDc = jpeg_encoder_structure->ldc3;
      jpeg_encoder_structure->ldc3 = Coeff;
   }
}
Coeff -= LastDc;
AbsCoeff = (Coeff < 0) ? -Coeff-- : Coeff;
while (AbsCoeff != 0)
{
   AbsCoeff >>= 1;
   DataSize++;
}
HuffCode = DcCodeTable [DataSize];
HuffSize = DcSizeTable [DataSize];
Coeff &= (1 << DataSize) - 1;
data = (HuffCode << DataSize) | Coeff;
numbits = HuffSize + DataSize;
PUTBITS
for (i=63; i>0; i--)
{
   if ((Coeff = *Temp_Ptr++) != 0)
```

```
        {
            while (RunLength > 15)
            {
                RunLength -= 16;
                data = AcCodeTable [161];
                numbits = AcSizeTable [161];
                PUTBITS
            }
            AbsCoeff = (Coeff < 0) ? -Coeff-- : Coeff;
            if (AbsCoeff >> 8 == 0)
                DataSize = bitsize [AbsCoeff];
            else
                DataSize = bitsize [AbsCoeff >> 8] + 8;

            index = RunLength * 10 + DataSize;
            HuffCode = AcCodeTable [index];
            HuffSize = AcSizeTable [index];
            Coeff &= (1 << DataSize) - 1;
            data = (HuffCode << DataSize) | Coeff;
            numbits = HuffSize + DataSize;
            PUTBITS
            RunLength = 0;
        }
        else
            RunLength++;
    }
    if (RunLength != 0)
    {
        data = AcCodeTable [0];
        numbits = AcSizeTable [0];
        PUTBITS
    }
    return output_ptr;
}
```

```
/* For bit Stuffing and EOI marker */
UINT8* close_bitstream (UINT8 *output_ptr)
{
   UINT16 i, count;
   UINT8 *ptr;
   if (bitindex > 0)
   {
      lcode <<= (32 - bitindex);
      count = (bitindex + 7) >> 3;
      ptr = (UINT8 *) &lcode + 3;

      for (i=count; i>0; i--)
      {
         if ((*output_ptr++ = *ptr--) == 0xff)
            *output_ptr++ = 0;
      }
   }
   // End of image marker
   *output_ptr++ = 0xFF;
   *output_ptr++ = 0xD9;
   return output_ptr;
}
/*****************************************************************************
   filename : main.c
*****************************************************************************/
#include <stdio.h>
#include <malloc.h>
#include "datatype.h"
#include "jdatatype.h"
#include "config.h"
#include "prototype.h"
void main ()
{
   INT8 input_file_name [150], output_file_name [150];
```

```
INT8 *input_file_name_ptr, *output_file_name_ptr;
UINT32 image_size;
UINT8 *input , *output ;
FILE *fp, *fpt;
UINT8 *output_ptr;
UINT32 quality_factor, image_format, image_width, image_height;
fp = fopen ("param.txt", "r");
while (fscanf (fp, "%s", input_file_name) != EOF)
{
   input_file_name_ptr = input_file_name;
   output_file_name_ptr = output_file_name;
   while ((*output_file_name_ptr++ = *input_file_name_ptr++) != '.');
   *output_file_name_ptr++ = 'j';
   *output_file_name_ptr++ = 'p';
   *output_file_name_ptr++ = 'g';
   *output_file_name_ptr++ = '\0';
   fscanf (fp, "%d", &quality_factor);
   fscanf (fp, "%d", &image_format);
   fscanf (fp, "%d", &image_width);
   fscanf (fp, "%d", &image_height);
   if (image_format == FOUR_ZERO_ZERO)
      image_size = image_width * image_height;
   else if (image_format == FOUR_TWO_ZERO)
      image_size = (image_width * image_height * 3) >> 1;
   else if (image_format == FOUR_TWO_TWO)
      image_size = image_width * image_height * 2;
   else
      image_size = image_width * image_height * 3;
   input=(UINT8 *)jemalloc(2500000*sizeof(UINT8*));
   fpt = fopen (input_file_name, "rb");
   fread (input, 1, image_size, fpt);
   fclose (fpt);
   output=(UINT8 *)jemalloc(250000*sizeof(UINT8*));
   output_ptr = output;
```

```
        output_ptr = encode_image (input, output_ptr, quality_factor,
                                   image_format, image_width, image_height);
        fpt = fopen (output_file_name, "wb");
        fwrite (output, 1, output_ptr - output, fpt);
        fclose (fpt);
        free (input);
        free (output);
    }
    fclose (fp);
}
/*****************************************************************************
    Filename : maker.c
*****************************************************************************/
#include "datatype.h"
#include "config.h"
#include "markdata.h"
// Header for JPEG Encoder
void write_markers (UINT8 *output_ptr, UINT32 image_format,
                    UINT32 image_width, UINT32 image_height)
{
    UINT16 i, header_length;
    UINT8 number_of_components;
    // Start of image marker
    *output_ptr++ = 0xFF;
    *output_ptr++ = 0xD8;
    // Quantization table marker
    *output_ptr++ = 0xFF;
    *output_ptr++ = 0xDB;
    // Quantization table length
    *output_ptr++ = 0x00;
    *output_ptr++ = 0x84;
    // Pq, Tq
    *output_ptr++ = 0x00;
    // Lqt table
```

```
for (i=0; i<64; i++)
   *output_ptr++ = Lqt [i];
// Pq, Tq
*output_ptr++ = 0x01;

// Cqt table
for (i=0; i<64; i++)
   *output_ptr++ = Cqt [i];
// huffman table(DHT)
for (i=0; i<210; i++)
{
   *output_ptr++ = (UINT8) (markerdata [i] >> 8);
   *output_ptr++ = (UINT8) markerdata [i];
}
if (image_format == FOUR_ZERO_ZERO)
   number_of_components = 1;
else
   number_of_components = 3;
// Frame header(SOF)
// Start of frame marker
*output_ptr++ = 0xFF;
*output_ptr++ = 0xC0;
header_length = (UINT16) (8 + 3 * number_of_components);
// Frame header length
*output_ptr++ = (UINT8) (header_length >> 8);
*output_ptr++ = (UINT8) header_length;
// Precision (P)
*output_ptr++ = 0x08;
// image height
*output_ptr++ = (UINT8) (image_height >> 8);
*output_ptr++ = (UINT8) image_height;
// image width
*output_ptr++ = (UINT8) (image_width >> 8);
*output_ptr++ = (UINT8) image_width;
```

```
// Nf
*output_ptr++ = number_of_components;
if (image_format == FOUR_ZERO_ZERO)
{
   *output_ptr++ = 0x01;
   *output_ptr++ = 0x11;
   *output_ptr++ = 0x00;
}
else
{
   *output_ptr++ = 0x01;
   if (image_format == FOUR_TWO_ZERO)
      *output_ptr++ = 0x22;
   else if (image_format == FOUR_TWO_TWO)
      *output_ptr++ = 0x21;
   else
      *output_ptr++ = 0x11;
   *output_ptr++ = 0x00;
   *output_ptr++ = 0x02;
   *output_ptr++ = 0x11;
   *output_ptr++ = 0x01;
   *output_ptr++ = 0x03;
   *output_ptr++ = 0x11;
   *output_ptr++ = 0x01;
}
// Scan header(SOF)
// Start of scan marker
*output_ptr++ = 0xFF;
*output_ptr++ = 0xDA;
header_length = (UINT16) (6 + (number_of_components << 1));
// Scan header length
*output_ptr++ = (UINT8) (header_length >> 8);
*output_ptr++ = (UINT8) header_length;
// Ns
```

```
    *output_ptr++ = number_of_components;
    if (image_format == FOUR_ZERO_ZERO)
    {
        *output_ptr++ = 0x01;
        *output_ptr++ = 0x00;
    }
    else
    {
        *output_ptr++ = 0x01;
        *output_ptr++ = 0x00;
        *output_ptr++ = 0x02;
        *output_ptr++ = 0x11;
        *output_ptr++ = 0x03;
        *output_ptr++ = 0x11;
    }
    *output_ptr++ = 0x00;
    *output_ptr++ = 0x3F;
    *output_ptr++ = 0x00;
}
/*****************************************************************************/
Filename : quant.c
*****************************************************************************/
#include "datatype.h"
#include "config.h"
#include "quantdata.h"
/*  This function implements 16 Step division for Q.15 format data */
UINT16 DSP_Division (UINT32 numer, UINT32 denom)
{
    UINT16 i;
    denom <<= 15;
    for (i=16; i>0; i--)
    {
        if (numer > denom)
        {
```

```
            numer -= denom;
            numer <<= 1;
            numer++;
        }
        else
            numer <<= 1;
    }
    return(UINT16) numer;
}
/* Multiply Quantization table with quality factor to get LQT and CQT */
void initialize_quantization_tables (UINT32 quality_factor)
{
    UINT16 i, index;
    UINT32 value;
    UINT8 luminance_quant_table [] = {
        16, 11, 10, 16,  24,  40,  51,  61,
        12, 12, 14, 19,  26,  58,  60,  55,
        14, 13, 16, 24,  40,  57,  69,  56,
        14, 17, 22, 29,  51,  87,  80,  62,
        18, 22, 37, 56,  68, 109, 103,  77,
        24, 35, 55, 64,  81, 104, 113,  92,
        49, 64, 78, 87, 103, 121, 120, 101,
        72, 92, 95, 98, 112, 100, 103,  99
    };
    UINT8 chrominance_quant_table [] = {
        17, 18, 24, 47, 99, 99, 99, 99,
        18, 21, 26, 66, 99, 99, 99, 99,
        24, 26, 56, 99, 99, 99, 99, 99,
        47, 66, 99, 99, 99, 99, 99, 99,
        99, 99, 99, 99, 99, 99, 99, 99,
        99, 99, 99, 99, 99, 99, 99, 99,
        99, 99, 99, 99, 99, 99, 99, 99,
        99, 99, 99, 99, 99, 99, 99, 99
    };
```

```
    for (i=0; i<64; i++)
    {
        index = zigzag_table [i];
        /* luminance quantization table * quality factor */
        value = luminance_quant_table [i] * quality_factor;
        value = (value + 0x200) >> 10;
        if (value == 0)
            value = 1;
        else if (value > 255)
            value = 255;
        Lqt [index] = (UINT8) value;
        ILqt [i] = DSP_Division (0x8000, value);

        /* chrominance quantization table * quality factor */
        value = chrominance_quant_table [i] * quality_factor;
        value = (value + 0x200) >> 10;
        if (value == 0)
            value = 1;
        else if (value > 255)
            value = 255;
        Cqt [index] = (UINT8) value;
        ICqt [i] = DSP_Division (0x8000, value);
    }
}
/* multiply DCT Coefficients with Quantization table and store in ZigZag location */
void quantization (INT16* const data, UINT16* const quant_table_ptr)
{
    INT16 i;
    INT32 value;
    for (i=63; i>=0; i--)
    {
        value = data [i] * quant_table_ptr [i];
        value = (value + 0x4000) >> 15;
        Temp [zigzag_table [i]] = (INT16) value;
```

```
    }
}
/*****************************************************************************
    Filename : readYUV.c
*****************************************************************************/
#include "datatype.h"
#include "jdatatype.h"
#include "config.h"
#include "prototype.h"
void read_400_format (JPEG_ENCODER_STRUCTURE *jpeg_encoder_structure, UINT8 *input_ptr)
{
    INT32 i, j;
    INT16 *Y1_Ptr = Y1;
    UINT16 rows = jpeg_encoder_structure->rows;
    UINT16 cols = jpeg_encoder_structure->cols;
    UINT16 incr = jpeg_encoder_structure->incr;
    for (i=rows; i>0; i--)
    {
        for (j=cols; j>0; j--)
            *Y1_Ptr++ = *input_ptr++;

        for (j=8-cols; j>0; j--)
            *Y1_Ptr++ = *(Y1_Ptr-1);

        input_ptr += incr;
    }
    for (i=8-rows; i>0; i--)
    {
        for (j=8; j>0; j--)
            *Y1_Ptr++ = *(Y1_Ptr - 8);
    }
}
void read_420_format (JPEG_ENCODER_STRUCTURE *jpeg_encoder_structure, UINT8 *input_ptr)
{
```

```
INT32 i, j;
UINT16 Y1_rows, Y3_rows, Y1_cols, Y2_cols;
INT16 *Y1_Ptr = Y1;
INT16 *Y2_Ptr = Y2;
INT16 *Y3_Ptr = Y3;
INT16 *Y4_Ptr = Y4;
INT16 *CB_Ptr = CB;
INT16 *CR_Ptr = CR;
INT16 *Y1Ptr = Y1 + 8;
INT16 *Y2Ptr = Y2 + 8;
INT16 *Y3Ptr = Y3 + 8;
INT16 *Y4Ptr = Y4 + 8;
UINT16 rows = jpeg_encoder_structure->rows;
UINT16 cols = jpeg_encoder_structure->cols;
UINT16 incr = jpeg_encoder_structure->incr;
if (rows <= 8)
{
   Y1_rows = rows;
   Y3_rows = 0;
}
else
{
   Y1_rows = 8;
   Y3_rows = (UINT16) (rows - 8);
}
if (cols <= 8)
{
   Y1_cols = cols;
   Y2_cols = 0;
}
else
{
   Y1_cols = 8;
   Y2_cols = (UINT16) (cols - 8);
```

```
}
for (i=Y1_rows>>1; i>0; i--)
{
   for (j=Y1_cols>>1; j>0; j--)
   {
      *Y1_Ptr++ = *input_ptr++;
      *Y1_Ptr++ = *input_ptr++;
      *Y1Ptr++ = *input_ptr++;
      *Y1Ptr++ = *input_ptr++;
      *CB_Ptr++ = *input_ptr++;
      *CR_Ptr++ = *input_ptr++;
   }

   for (j=Y2_cols>>1; j>0; j--)
   {
      *Y2_Ptr++ = *input_ptr++;
      *Y2_Ptr++ = *input_ptr++;
      *Y2Ptr++ = *input_ptr++;
      *Y2Ptr++ = *input_ptr++;
      *CB_Ptr++ = *input_ptr++;
      *CR_Ptr++ = *input_ptr++;
   }
   if (cols <= 8)
   {
      for (j=8-Y1_cols; j>0; j--)
      {
         *Y1_Ptr++ = *(Y1_Ptr - 1);
         *Y1Ptr++ = *(Y1Ptr - 1);
      }
      for (j=8; j>0; j--)
      {
         *Y2_Ptr++ = *(Y1_Ptr - 1);
         *Y2Ptr++ = *(Y1Ptr - 1);
      }
```

```
    }
    else
    {
        for (j=8-Y2_cols; j>0; j--)
        {
            *Y2_Ptr++ = *(Y2_Ptr - 1);
            *Y2Ptr++ = *(Y2Ptr - 1);
        }
    }
    for (j=(16-cols)>>1; j>0; j--)
    {
        *CB_Ptr++ = *(CB_Ptr-1);
        *CR_Ptr++ = *(CR_Ptr-1);
    }

    Y1_Ptr += 8;
    Y2_Ptr += 8;
    Y1Ptr += 8;
    Y2Ptr += 8;
    input_ptr += incr;
}
for (i=Y3_rows>>1; i>0; i--)
{
    for (j=Y1_cols>>1; j>0; j--)
    {
        *Y3_Ptr++ = *input_ptr++;
        *Y3_Ptr++ = *input_ptr++;
        *Y3Ptr++ = *input_ptr++;
        *Y3Ptr++ = *input_ptr++;
        *CB_Ptr++ = *input_ptr++;
        *CR_Ptr++ = *input_ptr++;
    }
    for (j=Y2_cols>>1; j>0; j--)
    {
```

```
    *Y4_Ptr++ = *input_ptr++;
    *Y4_Ptr++ = *input_ptr++;
    *Y4Ptr++ = *input_ptr++;
    *Y4Ptr++ = *input_ptr++;
    *CB_Ptr++ = *input_ptr++;
    *CR_Ptr++ = *input_ptr++;
}
if (cols <= 8)
{
    for (j=8-Y1_cols; j>0; j--)
    {
        *Y3_Ptr++ = *(Y3_Ptr - 1);
        *Y3Ptr++ = *(Y3Ptr - 1);
    }
    for (j=8; j>0; j--)
    {
        *Y4_Ptr++ = *(Y3_Ptr - 1);
        *Y4Ptr++ = *(Y3Ptr - 1);
    }
}
else
{
    for (j=8-Y2_cols; j>0; j--)
    {
        *Y4_Ptr++ = *(Y4_Ptr - 1);
        *Y4Ptr++ = *(Y4Ptr - 1);
    }
}
for (j=(16-cols)>>1; j>0; j--)
{
    *CB_Ptr++ = *(CB_Ptr-1);
    *CR_Ptr++ = *(CR_Ptr-1);
}
Y3_Ptr += 8;
```

```
    Y4_Ptr += 8;
    Y3Ptr += 8;
    Y4Ptr += 8;
    input_ptr += incr;
}
if (rows <= 8)
{
    for (i=8-rows; i>0; i--)
    {
        for (j=8; j>0; j--)
        {
            *Y1_Ptr++ = *(Y1_Ptr - 8);
            *Y2_Ptr++ = *(Y2_Ptr - 8);
        }
    }
    for (i=8; i>0; i--)
    {
        Y1_Ptr -= 8;
        Y2_Ptr -= 8;

        for (j=8; j>0; j--)
        {
            *Y3_Ptr++ = *Y1_Ptr++;
            *Y4_Ptr++ = *Y2_Ptr++;
        }
    }
}
else
{
    for (i=(16-rows); i>0; i--)
    {
        for (j=8; j>0; j--)
        {
            *Y3_Ptr++ = *(Y3_Ptr - 8);
```

```
            *Y4_Ptr++ = *(Y4_Ptr - 8);
          }
        }
    }
    for (i=((16-rows)>>1); i>0; i--)
    {
        for (j=8; j>0; j--)
        {
            *CB_Ptr++ = *(CB_Ptr-8);
            *CR_Ptr++ = *(CR_Ptr-8);
        }
    }
}
void read_422_format (JPEG_ENCODER_STRUCTURE *jpeg_encoder_structure, UINT8 *input_ptr)
{
    INT32 i, j;
    UINT16 Y1_cols, Y2_cols;
    INT16 *Y1_Ptr = Y1;
    INT16 *Y2_Ptr = Y2;
    INT16 *CB_Ptr = CB;
    INT16 *CR_Ptr = CR;
    UINT16 rows = jpeg_encoder_structure->rows;
    UINT16 cols = jpeg_encoder_structure->cols;
    UINT16 incr = jpeg_encoder_structure->incr;
    if (cols <= 8)
    {
        Y1_cols = cols;
        Y2_cols = 0;
    }
    else
    {
        Y1_cols = 8;
        Y2_cols = (UINT16) (cols - 8);
    }
```

```
for (i=rows; i>0; i--)
{
   for (j=Y1_cols>>1; j>0; j--)
   {
      *Y1_Ptr++ = *input_ptr++;
      *CB_Ptr++ = *input_ptr++;
      *Y1_Ptr++ = *input_ptr++;
      *CR_Ptr++ = *input_ptr++;
   }
   for (j=Y2_cols>>1; j>0; j--)
   {
      *Y2_Ptr++ = *input_ptr++;
      *CB_Ptr++ = *input_ptr++;
      *Y2_Ptr++ = *input_ptr++;
      *CR_Ptr++ = *input_ptr++;
   }
   if (cols <= 8)
   {
      for (j=8-Y1_cols; j>0; j--)
         *Y1_Ptr++ = *(Y1_Ptr - 1);

      for (j=8-Y2_cols; j>0; j--)
         *Y2_Ptr++ = *(Y1_Ptr - 1);
   }
   else
   {
      for (j=8-Y2_cols; j>0; j--)
         *Y2_Ptr++ = *(Y2_Ptr - 1);
   }
   for (j=(16-cols)>>1; j>0; j--)
   {
      *CB_Ptr++ = *(CB_Ptr-1);
      *CR_Ptr++ = *(CR_Ptr-1);
   }
```

```
        input_ptr += incr;
    }
    for (i=8-rows; i>0; i--)
    {
        for (j=8; j>0; j--)
        {
            *Y1_Ptr++ = *(Y1_Ptr - 8);
            *Y2_Ptr++ = *(Y2_Ptr - 8);
            *CB_Ptr++ = *(CB_Ptr - 8);
            *CR_Ptr++ = *(CR_Ptr - 8);
        }
    }
}
void read_444_format (JPEG_ENCODER_STRUCTURE *jpeg_encoder_structure, UINT8 *input_ptr)
{
    INT32 i, j;
    INT16 *Y1_Ptr = Y1;
    INT16 *CB_Ptr = CB;
    INT16 *CR_Ptr = CR;
    UINT16 rows = jpeg_encoder_structure->rows;
    UINT16 cols = jpeg_encoder_structure->cols;
    UINT16 incr = jpeg_encoder_structure->incr;
    for (i=rows; i>0; i--)
    {
        for (j=cols; j>0; j--)
        {
            *Y1_Ptr++ = *input_ptr++;
            *CB_Ptr++ = *input_ptr++;
            *CR_Ptr++ = *input_ptr++;
        }

        for (j=8-cols; j>0; j--)
        {
            *Y1_Ptr++ = *(Y1_Ptr-1);
```

```
            *CB_Ptr++ = *(CB_Ptr-1);
            *CR_Ptr++ = *(CR_Ptr-1);
        }
        input_ptr += incr;
    }
    for (i=8-rows; i>0; i--)
    {
        for (j=8; j>0; j--)
        {
            *Y1_Ptr++ = *(Y1_Ptr - 8);
            *CB_Ptr++ = *(CB_Ptr - 8);
            *CR_Ptr++ = *(CR_Ptr - 8);
        }
    }
}
void RGB_2_444 (UINT8 *input_ptr, UINT8 *output_ptr, UINT32 image_width, UINT32 image_height)
{
    UINT32 i, size;
    UINT8 R, G, B;
    INT32 Y, Cb, Cr;
    size = image_width * image_height;
    for (i=size; i>0; i--)
    {
        B = *input_ptr++;
        G = *input_ptr++;
        R = *input_ptr++;

        Y = ((77 * R + 150 * G + 29 * B) >> 8);
        Cb = ((-43 * R - 85 * G + 128 * B) >> 8) + 128;
        Cr = ((128 * R - 107 * G - 21 * B) >> 8) + 128;

        if (Y < 0)
            Y = 0;
        else if (Y > 255)
```

```
            Y = 255;

        if (Cb < 0)
            Cb = 0;
        else if (Cb > 255)
            Cb = 255;

        if (Cr < 0)
            Cr = 0;
        else if (Cr > 255)
            Cr = 255;

        *output_ptr++ = (UINT8) Y;
        *output_ptr++ = (UINT8) Cb;
        *output_ptr++ = (UINT8) Cr;
    }
}
```

ภาคผนวก ง

โปรแกรมแสดงภาพฝั่งผู้ใช้งาน (Borland Delphi 7)

```
unit Unit1;
interface
uses
        Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
        Dialogs, ExtCtrls, tscap32_rt, StdCtrls, tsLogBox, ShellApi, jpeg,
        IdTCPConnection, IdTCPClient, IdBaseComponent, IdComponent, IdTCPServer,
        IdUDPClient, IdUDPBase, IdUDPServer, IdSocketHandle, TeeProcs, TeEngine,
        Chart, Series, ComCtrls;
type
        TForm1 = class(TForm)
        Memo: TMemo;
        Button6: TButton;
        Button7: TButton;
        ImageShow: TImage;
        ButtonStart: TButton;
        udpClient: TIdUDPClient;
        Button12: TButton;
        VideoClientTimer: TTimer;
        EditIpAddr: TEdit;
        Label3: TLabel;
        Button5: TButton;
        NoiseTimer: TTimer;
        NoiseUdpClient: TIdUDPClient;
        Button8: TButton;
        procedure cap32Bitmap(Sender: TObject; Bitmap: TBitmap;
        msSinceFirstFrame: Cardinal);
        procedure cap32Dib(Sender: TObject; pBmi: PBitmapInfo;
        pBits: PAnsiChar; msSinceFirstFrame: Cardinal);
        procedure cap32Frame(Sender: TObject; pVidHdr: PVIDEOHDR);
```

```
        procedure cap32VideoStream(Sender: TObject; pVidHdr: PVIDEOHDR);
        procedure Button6Click(Sender: TObject);
        procedure tcpServerConnect(AThread: TIdPeerThread);
        procedure udpServerUDPRead(Sender: TObject; AData: TStream;
                ABinding: TIdSocketHandle);
        procedure ButtonStartClick(Sender: TObject);
        procedure FormCreate(Sender: TObject);
        procedure Button12Click(Sender: TObject);
        procedure VideoClientTimerTimer(Sender: TObject);
        procedure Button7Click(Sender: TObject);
        procedure EditFpsEnter(Sender: TObject);
        procedure NoiseTimerTimer(Sender: TObject);
        procedure Button5Click(Sender: TObject);
        procedure Button8Click(Sender: TObject);
private
        { Private declarations }
public
        { Public declarations }
        frame_count : integer;
        clientIpAddr : string;
        clientPort : integer;
        VideoServerFrameInterval : integer;
        VideoServerFrameQuality : double;
        VideoServerFrameNo : LongWord;
        VideoServerBandwidth : Integer;
        VideoServerState : Integer;
        VideoServerRunTime : Integer;
        VideoServerFrameLost : Integer;
  //
  // Video client property.
```

```
  //
        VideoClientTimeOffset : Integer;
        VideoClientFrameLostCnt: LongWord;
        videoClientInitCnt : Integer;
        VideoClientFeedbackTimer : Integer;
        VideoClientState : Integer;
        VideoClientInputCnt : Integer;
        procedure BmpToJpg(quality : Integer);
        function InferenceFps(frameLost: double) : double;
        function InferenceQua(frameLost: double) : double;
        function FuzzifyFps(rule : integer; input : double) : double;
        function FuzzifyQuality(rule : integer; input : double) : double;
 end;
var
        Form1: TForm1;
implementation
{$R *.dfm}
//#define C3
const
QUA_DO_DEC   = 2.0;
QUA_DO_ZERO  = 0.0;
QUA_DO_INC   = -4.0;
DO_FL_ZERO =   0.1;
DO_FL_LOW  =   0.0;
DO_FL_HIGH =  (-1.0);
// Zero.
ZERO_MIN = -1000000.0;
ZERO_CEN = 0.0;
ZERO_MAX = 2.0;
// Low
```

```
LOW_MIN = 1.0;
LOW_CEN = 2.0;
LOW_MAX = 3.0;
// High
HIGH_MIN = 2.0;
HIGH_CEN = 3.0;
HIGH_MAX = 1000000.0;
function SystemTimeMs : LongWord;
var
        st : TSystemTime;
        timeStamp : LongWord;
begin
        GetSystemTime(st);
        TimeStamp := st.wHour*60*60*1000;
        TimeStamp := timeStamp + (st.wMinute*60*1000);
        TimeStamp := timeStamp + (st.wSecond*1000);
        TimeStamp := timeStamp + (st.wMilliseconds);
        Result := timeStamp;
end;
procedure TForm1.cap32Bitmap(Sender: TObject; Bitmap: TBitmap;
        msSinceFirstFrame: Cardinal);
begin
        //Memo.Lines.Add('OnBitmap: ' + IntToStr(frame_count));
        //Bitmap.SaveToFile('out.bmp');
        //Inc(frame_count);
end;
procedure TForm1.cap32Dib(Sender: TObject; pBmi: PBitmapInfo;
        pBits: PAnsiChar; msSinceFirstFrame: Cardinal);
begin
        //Memo.Lines.Add('OnDip')
```

```
end;
//
//  Frame event
//
procedure TForm1.cap32Frame(Sender: TObject; pVidHdr: PVIDEOHDR);
var
        s : String;
        bmpHdr : Array[0..128] of byte;
        bmpData : Array[0..500000] of byte;
        fileHandle : Integer;
begin
        bmpHdr[$00] := $42;
        bmpHdr[$01] := $4d;
        bmpHdr[$02] := $36;
        bmpHdr[$03] := $84;
        bmpHdr[$04] := $03;
        bmpHdr[$05] := $00;
        bmpHdr[$06] := $00;
        bmpHdr[$07] := $00;
        bmpHdr[$08] := $00;
        bmpHdr[$09] := $00;
        bmpHdr[$0a] := $36;
        bmpHdr[$0b] := $00;
        bmpHdr[$0c] := $00;
        bmpHdr[$0d] := $00;
        bmpHdr[$0e] := $28;
        bmpHdr[$0f] := $00;
        bmpHdr[$10] := $00;
        bmpHdr[$11] := $00;
        bmpHdr[$12] := $40;
```

```
bmpHdr[$13] := $01;
bmpHdr[$14] := $00;
bmpHdr[$15] := $00;
bmpHdr[$16] := $f0;
bmpHdr[$17] := $00;
bmpHdr[$18] := $00;
bmpHdr[$19] := $00;
bmpHdr[$1a] := $01;
bmpHdr[$1b] := $00;
bmpHdr[$1c] := $18;
bmpHdr[$1d] := $00;
bmpHdr[$1e] := $00;
bmpHdr[$1f] := $00;

bmpHdr[$20] := $00;
bmpHdr[$21] := $00;
bmpHdr[$22] := $00;
bmpHdr[$23] := $84;
bmpHdr[$24] := $03;
bmpHdr[$25] := $00;
bmpHdr[$26] := $00;
bmpHdr[$27] := $00;
bmpHdr[$28] := $00;
bmpHdr[$29] := $00;
bmpHdr[$2a] := $00;
bmpHdr[$2b] := $00;
bmpHdr[$2c] := $00;
bmpHdr[$2d] := $00;
bmpHdr[$2e] := $00;
bmpHdr[$2f] := $00;
```

```
        bmpHdr[$30] := $00;
        bmpHdr[$31] := $00;
        bmpHdr[$32] := $00;
        bmpHdr[$33] := $00;
        bmpHdr[$34] := $00;
        bmpHdr[$35] := $00;
        // Move to BMP data buffer.
        Move(pVidHdr.lpData^, bmpData[0], pVidHdr.dwBytesUsed);
        // Save this frame to file.
        fileHandle := FileOpen('raw.bmp', fmOpenWrite);
        if FileHandle > 0 then
                begin
                FileWrite(fileHandle, bmpHdr[0], $36);
                FileWrite(fileHandle, bmpData[0], pVidHdr.dwBytesUsed);
                FileClose(fileHandle);
                end
        else
                begin
                Memo.Lines.Add('OnFrame: Failed to open file.');
                end;
        //Memo.Lines.Add('OnFrame: Event.')
end;
procedure TForm1.cap32VideoStream(Sender: TObject; pVidHdr: PVIDEOHDR);
begin
        Memo.Lines.Add('OnVideoStream')
end;
procedure TForm1.Button6Click(Sender: TObject);
begin
        Memo.Clear;
```

```
end;
//
// Convert BMP to JPG with quality.
//
procedure TForm1.BmpToJpg ( quality : Integer);
var
        i : integer;
        s, paramStr: string;
        charPtr : Pchar;
begin
        paramStr := '-qscale ' + IntToStr(quality) + ' -i raw.bmp out.jpg';
        charPtr := Addr(paramStr[1]);
        i := ShellExecute(Handle,
                'open',
                'c:\work\video_server\ffmpeg.exe',
                charPtr,
                'c:\work\video_server',
                SW_HIDE) ;
        //Memo.Lines.Add('BmpToJpg: Reture '+ IntToStr(i) );
end;
procedure TForm1.tcpServerConnect(AThread: TIdPeerThread);
begin
        memo.Lines.Add('tcpServer: connect');
end;
procedure TForm1.udpServerUDPRead(Sender: TObject; AData: TStream;
        ABinding: TIdSocketHandle);
var
        i : integer;
        s, packetStr, lostStr : string;
begin
```

```
i := Adata.Size;
//s := 'UdpServer: InputDataSize=' + Inttostr(i);
//memo.Lines.Add(s);
//clientIpAddr := Abinding.PeerIP;
//clientPort := Abinding.PeerPort;

SetLength(packetStr, Adata.size);
Adata.Read(packetStr[1], Adata.Size);
//memo.Lines.Add('UdpServer: InputData='+packetStr);
// Check input data.
if packetStr = 'Start' then
begin
        memo.Lines.Add('VideoServer: Start.');
        VideoClientState := 1;
        VideoServerBandwidth := 0;
        VideoServerState := 1;
        VideoServerRunTime := 0;
end;
if packetStr = 'Stop' then
begin
        memo.Lines.Add('VideoServer: Stop.');
        VideoClientState := 0;
        clientPort := 0;
end;
lostStr := 'Lost=';
if StrLComp(Pchar(packetStr), Pchar(lostStr), 5) = 0 then
begin
        i := 0;
        if (Ord(packetStr[6]) >= Ord('0')) and (Ord(packetStr[6]) <= Ord('9')) then
        begin
```

```
                    i := Ord(packetStr[6]) - Ord('0');
                    if (Ord(packetStr[7]) >= Ord('0'))
                      and (Ord(packetStr[7]) <= Ord('9')) then
                    begin
                        i := i*10;
                        i := i + (Ord(packetStr[7]) - Ord('0'));
                        if (Ord(packetStr[8]) >= Ord('0')) and (Ord(packetStr[8]) <=
Ord('9')) then
                        begin
                            i := i*10;
                            i := i + (Ord(packetStr[8]) - Ord('0'));
                        end;
                    end;
                end;
                // If frame lost not 0.
                if i <> 0 then
                begin
                    VideoServerFrameLost := i;
                    Memo.Lines.Add('VideoServer: FrameLost=' + IntToStr(i));
                end
                else
                begin
                    VideoServerFrameLost := 0;
                end;
end;
end;
//
// Start play a stream.
//
procedure TForm1.ButtonStartClick(Sender: TObject);
```

```
begin
        UdpClient.Host := EditIpAddr.Text;
        UdpClient.Port := 12345;
        UdpClient.Active := True;
        udpClient.Send('Start');
        videoClientInitCnt := 0;
        VideoClientTimeOffset := 0;
        VideoClientState := 1;
end;
procedure TForm1.FormCreate(Sender: TObject);
begin
        clientPort := 0;
        VideoServerState := 0;
end;
//
//  Stop video server stream.
//
procedure TForm1.Button12Click(Sender: TObject);
begin
        udpClient.Send('Stop');
        UdpClient.Active := False;
        VideoClientState := 0;
end;
procedure TForm1.VideoClientTimerTimer(Sender: TObject);
var
        len : integer;
        s : string;
        buf : array[0..50000] of byte;
        fileHandle : integer;
        fileLen : integer;
```

```
        frameNo : LongWord;
        remoteTimeStamp : LongWord;
        bytePtr : Pbyte;
        st : TSystemTime;
        localTimeStamp : LongWord;
        timeDiff : Integer;
begin
        GetSystemTime(st);
        localTimeStamp := SystemTimeMs();
        if localTimeStamp - VideoClientFeedbackTimer > 2000 then
        begin
                VideoClientFeedbackTimer := localTimeStamp;
                if VideoClientState = 1 then
                begin
                        if VideoClientInputCnt = 0 then
                        begin
                                Inc(VideoClientFrameLostCnt,5);
                        end;
                        VideoClientInputCnt := 0 ;
                        UdpClient.Send('Lost='+IntToStr(VideoClientFrameLostCnt));
                        if (VideoClientFrameLostCnt <> 0) then
                        begin
                                Memo.Lines.Add('Lost=' +
IntToStr(VideoClientFrameLostCnt));
                        end;
                        VideoClientFrameLostCnt := 0; // Reset counter;
                end;
        end;
        len := udpClient.ReceiveBuffer(buf[0], 50000, 1);
        if len > 0 then
```

```
        begin
                //s := 'VideoClient: Input data size=' + IntToStr(len);
                //Memo.Lines.Add(s);
                fileHandle := FileOpen('recv.jpg', fmOpenWrite);
                if fileHandle > 0 then
                begin
                        GetSystemTime(st);
                        localTimeStamp := SystemTimeMs();
                        bytePtr := Addr(frameNo);
                        Move(buf[0], bytePtr^, 4);
                        bytePtr := Addr(remoteTimeStamp);
                        Move(buf[4], bytePtr^, 4);
                        //Memo.Lines.Add('VideoClient: RecvFrameNo=' +
IntToStr(frameNo));
                        if VideoClientInitCnt < 7 then
                        begin
                                if VideoClientInitCnt = 6 then
                                begin
                                        VideoClientTimeOffset : = VideoClientTimeOffset div
6;
                                        Memo.Lines.Add('VideoClient: Offset=' +
IntToStr(VideoClientTimeOffset));
                                end
                                else
                                begin
                                         VideoClientTimeOffset := VideoClientTimeOffset  +
(localTimeStamp - remoteTimeStamp);
                                         //Memo.Lines.Add('VideoClient: Offset=' +
IntToStr(VideoClientTimeOffset));
                                end;
```

```
                                Inc(VideoClientInitCnt);
                        end;
                        if VideoClientInitCnt >= 7 then
                        begin
                                timeDiff := (localTimeStamp - remoteTimeStamp) -
VideoClientTimeOffset;
                                //Memo.Lines.Add('VideoClient: tDiff=' + IntToStr(timeDiff)
                                        //+' lTs=' + IntToStr(localTimeStamp)
                                        //+' rTs=' + Inttostr(RemoteTimeStamp));
                                // If time different more than 1000ms.
                                if Abs(timeDiff) > 1000 then
                                begin
                                        Inc(VideoClientFrameLostCnt);
                                end;
                        end;
                        Inc(VideoClientInputCnt);
                        FileWrite(fileHandle, buf[8], len-8);
                        FileClose(fileHandle);
                        ImageShow.Picture.LoadFromFile('c:\work\video_server\recv.jpg');
                end;
        end;
end;
procedure TForm1.Button7Click(Sender: TObject);
var
  i : Integer;
begin
        Inc(VideoClientFrameLostCnt);
end;

procedure TForm1.EditFpsEnter(Sender: TObject);
```

```
begin
        //ShowMessage('OnEnter');
end;
procedure TForm1.NoiseTimerTimer(Sender: TObject);
var
        buf : Array[0..5000] of byte;
begin
        NoiseUdpClient.SendBuffer(buf[0],5000);
end;
procedure TForm1.Button5Click(Sender: TObject);
begin
        NoiseUdpClient.Active := True;
        NoiseTimer.Enabled := True;
end;
procedure TForm1.Button8Click(Sender: TObject);
begin
        NoiseUdpClient.Active := False;
        NoiseTimer.Enabled := False;
end;
end.
```

ภาคผนวก จ

โปรแกรมติดต่อโมดูลกล้อง C328R

```
#include "LPC23xx.H"
#include "type.h"
#include "uart.h"
#include "timer.h"
#include "irq.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <absacc.h>
#define DEBUG_CR328
#define CMD_SYNC                    1
#define CMD_SYNC_WAIT               2
#define CMD_INIT                    3
#define CMD_INIT_WAIT               4
#define CMD_SET_PACKET_SIZE         5
#define CMD_SET_PACKET_SIZE_WAIT    6
#define CMD_SNAPSHOT                7
#define CMD_SNAPSHOT_WAIT           8
#define CMD_GET_PICTURE             9
#define CMD_GET_PICTURE_WAIT        10
#define CMD_RECV_PICTURE            11
/**
 * end of ethernet test.
 ************************************************************************/
//const u32t param_init_all=0x00000000;
// sector number 22
//const unsigned long param_init = 0x00000000 _at_ 0x00078000;
#ifdef DEBUG
#endif
u8t img_data_buf[5000];
```

```
void debug_send(u8t* p, int len);
void debug_recv(u8t* p, int len);
void disp_buf_hex(u8t *p, int len);
void image_to_serial(u8t *p, int len);
int jpg_res=3;
int pic_type = 5;
void c328_init(void) {
	u8t buf[32];
	int i;
	//uart0_write_str("# CR328 Init.\r\n");
	buf[0] = 0xAA;
	buf[1] = 0x01;
	buf[2] = 0x00;
	buf[3] = 0x07;// color type
	buf[4] = 0x05;// preview resolution.
	buf[5] = (u8t)jpg_res; //0x03;// jpeg resolution.
	for (i=0; i<6; i++) {
		uart3_write_byte(buf[i]);
	}
	uart0_write_str("# Init.\r\n");
#ifdef DEBUG_CR328
	debug_send(buf, 6);
#endif
}
void c328_getpicture(void) {
	u8t buf[32];
	int i;
	buf[0] = 0xAA;
	buf[1] = 0x04;
	//1 = Snapshot picture
```

```
	//2 = Preview Picture.
	//5 = JPEG preview picture
	buf[2] = pic_type;// 0x05;
	buf[3] = 0x00;
	buf[4] = 0x00;
	buf[5] = 0x00;
	for (i=0; i<6; i++) {
		uart3_write_byte(buf[i]);
	}
	uart0_write_str("# GetPicture.\r\n");
#ifdef DEBUG_CR328
	debug_send(buf, 6);
#endif
}
void c328_snapshot(void) {
	u8t buf[32];
	int i;
	buf[0] = 0xAA;
	buf[1] = 0x05;
	// Snap shot type
	// 0=compressed.
	// 1=uncompressed.
	buf[2] = 0x00;
	// Skip frame low byte.
	buf[3] = 0x00;
	// Skip frame high byte.
	buf[4] = 0x00;
	buf[5] = 0x00;
	for (i=0; i<6; i++) {
		uart3_write_byte(buf[i]);
```

```
        }
        uart0_write_str("# SnapShot.\r\n");
#ifdef DEBUG_CR328
        debug_send(buf, 6);
#endif
}
void c328_set_package_size(void) {
        u8t buf[32];
        int i;
        //uart0_write_str("# CR328 SetPacketSize.\r\n");
        buf[0] = 0xAA;
        buf[1] = 0x06;
        // fixed=0x08
        buf[2] = 0x08;
        // Packet size low byte
        buf[3] = 0x00;
        // Packet size high byte.
        buf[4] = 0x02;
        // Fixed=0
        buf[5] = 0x00;
        for (i=0; i<6; i++) {
                uart3_write_byte(buf[i]);
        }
        uart0_write_str("# PacketSize.\r\n");
#ifdef DEBUG_CR328
        debug_send(buf, 6);
#endif
}
void c328_set_baudrate(void) {
        u8t buf[32];
```

```
	int i;
	buf[0] = 0xAA;
	buf[1] = 0x07;
	// first divider
	buf[2] = 0x1f;
	// second divider.
	buf[3] = 0x01;
	// fixed=0
	buf[4] = 0x00;
	// Fixed=0
	buf[5] = 0x00;
	for (i=0; i<6; i++) {
		uart3_write_byte(buf[i]);
	}
	uart0_write_str("# SetBaudRate.\r\n");
#ifdef DEBUG_CR328
	debug_send(buf, 6);
#endif
}
void c328_reset(void) {
	u8t buf[32];
	int i;
	buf[0] = 0xAA;
	buf[1] = 0x08;
	// reset type
	buf[2] = 0x00;
	// fixed=0
	buf[3] = 0x00;
	// fixed=0
	buf[4] = 0x00;
```

```
	// Fixed=0xff
	buf[5] = 0xff;
	for (i=0; i<6; i++) {
		uart3_write_byte(buf[i]);
	}
	uart0_write_str("# Reset.\r\n");
#ifdef DEBUG_CR328
	debug_send(buf, 6);
#endif
}
void c328_power_off(void) {
	u8t buf[32];
	int i;
	//uart0_write_str("# CR328 PowerOff.\r\n");
	buf[0] = 0xAA;
	buf[1] = 0x09;
	// fix=0, 4 bytes.
	buf[2] = 0x00;
	buf[3] = 0x00;
	buf[4] = 0x00;
	buf[5] = 0x00;
	for (i=0; i<6; i++) {
		uart3_write_byte(buf[i]);
	}
	uart0_write_str("# PowerOff.\r\n");
#ifdef DEBUG_CR328
	debug_send(buf, 6);
#endif
}
void c328_data(void) {
```

```
	u8t buf[32];
	int i;
	buf[0] = 0xAA;
	buf[1] = 0x0A;
	buf[2] = 1;// data type.
	//1=snapshot type
	//2=preview picture.
	//5=jpeg preview picture.
	buf[3] = 0;// lenght byte 0.
	buf[4] = 0;// lenght byte 1.
	buf[5] = 0;// lenght byte 2.
	for (i=0; i<6; i++) {
		uart3_write_byte(buf[i]);
	}
	uart0_write_str("# data: ");
	debug_send(buf, 6);
}
void c328_sync(void) {
	u8t buf[32];
	int i;
	//uart0_write_str("# CR328 Sync.\r\n");
	buf[0] = 0xAA;
	buf[1] = 0x0D;
	buf[2] = 0x00;//command id.
	buf[3] = 0x00;//ack counter.
	buf[4] = 0x00;//00h / packet id byte0
	buf[5] = 0x00;//00h / packet id byte1
	for (i=0; i<6; i++) {
		uart3_write_byte(buf[i]);
	}
```

```
	uart0_write_str("# Sync.\r\n");
#ifdef DEBUG_CR328
	debug_send(buf, 6);
#endif
}
void c328_ack(u8t packet_id) {
	u8t buf[32];
	char str[32];
	int i;
	buf[0] = 0xAA;
	buf[1] = 0x0E;
	buf[2] = 0x00;//command id.
	buf[3] = 0x00;//ack counter.
	buf[4] = packet_id;//00h / packet id byte0
	buf[5] = 0x00;//00h / packet id byte1
	for (i=0; i<6; i++) {
		uart3_write_byte(buf[i]);
	}
	uart0_write_str("# Ack.\r\n");
#ifdef DEBUG_CR328
	debug_send(buf, 6);
#endif
}
void c328_nak(void) {
	u8t buf[32];
	int i;
	uart0_write_str("# CR328 Nak.\r\n");
	buf[0] = 0xAA;
	buf[1] = 0x0F;
	buf[2] = 0x00;//fix=0
```

```
        buf[3] = 0x00;//nak counter.
        buf[4] = 0x00;//error number.
        buf[5] = 0x00;//fix=0;
        for (i=0; i<6; i++) {
                uart3_write_byte(buf[i]);
        }
}
void c328_light_frequency(void) {
        u8t buf[32];
        int i;
        uart0_write_str("# CR328 LightFrequency.\r\n");
        buf[0] = 0xAA;
        buf[1] = 0x13;
        buf[2] = 0x00;// frequency type.
        buf[3] = 0x00;//fix=0
        buf[4] = 0x00;//fix=0
        buf[5] = 0x00;//fix=0;
        for (i=0; i<6; i++) {
                uart3_write_byte(buf[i]);
        }
}
int main(void) {
        // local var.
        int     i;
        u8t     byte;
        u8t     buf[20];
        char    str[100];
        char    dbg[128];
        char    *p;
        int     c328r_state = 0;
```

```
	int	len = 0;
	int	img_size;


	u8t	c328r_rx_buf[1024];
	int	c328r_rx_len;
	u8t	my_packet_id;
	int	img_byte_cnt;
	char	cmd[128];
	int	cmdlen=0;
	//char	resp[]
	uart0_init();
	//Uart2Init();
	uart3_init();
	timer0_init(TIME_INTERVAL);
	EnableTimer(0);
	c328r_rx_len = 0;
	IENABLE;
	uart0_write_str("### C328 Driver ###.\r\n");
	while (1) {
#if 0
		if (uart0_read_byte(&byte)) {
			uart3_write_byte(byte);
		}
		if (uart3_read_byte(&byte)) {
			uart0_write_byte(byte);
		}
		continue;
#else
		if (uart0_read_byte(&byte)) {
			if (cmdlen < sizeof(cmd)) {
```

```
                    cmd[cmdlen] = byte;
                    cmdlen++;
                    }
                    // end of command.
                    if (byte=='\r') {
                    if (!strncmp(cmd, "start", 5)) {
                            c328r_state=1;
                    }
                    if (!strncmp(cmd, "reset", 5)) {
                            c328_reset();
                    }
                    else if (!strncmp(cmd, "jr=", 3)) {
                            if (cmd[3]=='?') {
                                    ;
                            }
                            else if (cmd[3]=='1' || cmd[3]=='3' || cmd[3]=='5' ||
cmd[3]=='7') {
                                    jpg_res = atoi(cmd+3);
                            }
                            sprintf(dbg, "# jpgres(1,3,5,7)=%u.\r\n", jpg_res);
                            uart0_write_str(dbg);
                    }
                    else if (!strncmp(cmd, "pt=", 3)) {
                            if (cmd[3]=='?') {
                                    ;
                            }
                            else if (cmd[3]=='1' || cmd[3]=='2' || cmd[3]=='5') {
                                    pic_type = atoi(cmd+3);
                            }
                            sprintf(dbg, "# pictype(1,2,5)=%u.\r\n", pic_type);
```

```
                    uart0_write_str(dbg);
                }
                cmdlen=0;
                memset(cmd, 0, sizeof(cmd));
                }
        }
        if (c328r_state == CMD_SYNC) {
                c328r_state = CMD_SYNC_WAIT;
                c328r_rx_len = 0;
                c328r_timer = 100;
                c328_sync();
        }
        else if (c328r_state == CMD_SYNC_WAIT) {
                if (c328r_timer == 0) {
                if (c328r_rx_len == 12) {
                        //debug_recv(c328r_rx_buf, c328r_rx_len);
                        c328_ack(0);
                        c328r_state = CMD_INIT;
                }
                else {
                        //uart0_write_str("# sync timeout.\r\n");
                        c328r_state = CMD_SYNC;
                }
                 }
                if (uart3_read_byte(&byte)) {
                if (c328r_rx_len < sizeof(c328r_rx_buf)) {
                        c328r_rx_buf[c328r_rx_len] = byte;
                                c328r_rx_len++;
                                 if (c328r_rx_len==12) {
                                c328r_timer=0;
```

```
                        }
                    }
                    }
 }
// Init
 else if (c328r_state == CMD_INIT) {
                    c328r_state = CMD_INIT_WAIT;
                    c328r_rx_len = 0;
                    c328r_timer = 1000;
                    c328_init();
}
 else if (c328r_state == CMD_INIT_WAIT) {
                    if (c328r_timer == 0) {
                    if (c328r_rx_len == 6) {
                        c328r_state = CMD_SET_PACKET_SIZE;
                    }
                    else {
                        //uart0_write_str("# timeout.\r\n");
                        c328r_state = CMD_SYNC;
                    }
                    }
                    else if (uart3_read_byte(&byte)) {
                    if (c328r_rx_len < sizeof(c328r_rx_buf)) {
                        c328r_rx_buf[c328r_rx_len] = byte;
                        c328r_rx_len++;
                        if (c328r_rx_len == 6) {
                            c328r_timer = 0;
                        }
                    }
                    }
```

```
}
 // Set packet size
else if (c328r_state == CMD_SET_PACKET_SIZE) {
                c328r_state = CMD_SET_PACKET_SIZE_WAIT;
                c328r_rx_len = 0;
                c328r_timer = 1000;
                c328_set_package_size();
}
else if (c328r_state == CMD_SET_PACKET_SIZE_WAIT) {
                if (c328r_timer == 0) {
                if (c328r_rx_len == 6) {
                        //disp_buf_hex(c328r_rx_buf, c328r_rx_len);
                        //debug_recv(c328r_rx_buf, c328r_rx_len);
                        //c328_ack(0);
                        c328r_state = CMD_SNAPSHOT;//9;//7
                }
                else {
                        //uart0_write_str("# timeout.\r\n");
                        c328r_state = CMD_SYNC;
                }
                }
                else if (uart3_read_byte(&byte)) {
                  if (c328r_rx_len < sizeof(c328r_rx_buf)) {
                        c328r_rx_buf[c328r_rx_len] = byte;
                        c328r_rx_len++;
                        if (c328r_rx_len == 6) {
                                c328r_timer = 0;
                        }
                }
                }
```

```
}
// take snap shot.
else if (c328r_state == CMD_SNAPSHOT) {
                //uart0_write_str("# do snap shot.\r\n");
                c328r_state = CMD_SNAPSHOT_WAIT;
                c328r_rx_len = 0;
                c328r_timer = 1000;
                c328_snapshot();
}
if (c328r_state == CMD_SNAPSHOT_WAIT) {
                if (c328r_timer == 0) {
                if (c328r_rx_len == 6) {
                        c328r_state = CMD_GET_PICTURE;
                }
                else {
                        //uart0_write_str("# timeout.\r\n");
                        c328r_state = CMD_SYNC;
                }
                }
                if (uart3_read_byte(&byte)) {
                if (c328r_rx_len < sizeof(c328r_rx_buf)) {
                        c328r_rx_buf[c328r_rx_len] = byte;
                        c328r_rx_len++;
                        if (c328r_rx_len == 6) {
                                c328r_timer = 0;
                        }
                }
                }
}
// Get picture.
```

```
                if (c328r_state == CMD_GET_PICTURE) {
                                //uart0_write_str("# do get picture.\r\n");
                                c328r_state = CMD_GET_PICTURE_WAIT;
                                c328r_rx_len = 0;
                                c328r_timer = 1000;
                                c328_getpicture();
                }
                if (c328r_state == CMD_GET_PICTURE_WAIT) {
                                if (c328r_timer == 0) {
                                 if (c328r_rx_len == 12) {
                                                //uart0_write_str("# getpicture ok.\r\n");
                                                img_size = (c328r_rx_buf[11]<<16) +
(c328r_rx_buf[10]<<8) + c328r_rx_buf[9];
                                                //sprintf(str, "# ImgDataSize=%u.\r\n", img_size);
                                                //uart0_write_str(str);
                                                //disp_buf_hex(c328r_rx_buf, c328r_rx_len);
                                                //debug_recv(c328r_rx_buf, c328r_rx_len);
                                                c328r_state = CMD_RECV_PICTURE;
                                                my_packet_id = 0;
                                                img_byte_cnt = 0;
                                                c328r_timer = 1000;
                                                c328r_rx_len = 0;
                                                c328_ack(my_packet_id);
                                }
                                else {
                                                //uart0_write_str("# timeout.\r\n");
                                                c328r_state = CMD_SYNC;
                                }
                                }
                                else if (uart3_read_byte(&byte)) {
```

```
                        //c328r_timer = 100;
                        if (c328r_rx_len < sizeof(c328r_rx_buf)) {
                                c328r_rx_buf[c328r_rx_len] = byte;
                                c328r_rx_len++;
                                if (c328r_rx_len == 12) {
                                        c328r_timer = 0;
                                }
                        }
                        }
                }
                if (c328r_state == CMD_RECV_PICTURE) {
                        if (c328r_timer == 0) {
                        //c328r_state = 0;
                        //uart0_write_str("# CmdRecvPictTimeout.\r\n");
                        i = c328r_rx_buf[2] + (c328r_rx_buf[3]*256);
                        if (c328r_rx_len == i+6) {
                                if (i+img_byte_cnt < sizeof(img_data_buf)) {

        memcpy(img_data_buf+img_byte_cnt, c328r_rx_buf+4, i);
                                }
                                img_byte_cnt += i;
                                //sprintf(str,"# PacketSize=%u, ImgSize=%u,
TotolSize=%u.\r\n",
                          //c328r_rx_len, i, img_byte_cnt);
                                //uart0_write_str(str);
                                //my_packet_id = c328r_rx_buf[0] +
(c328r_rx_buf[1]*256);
                                c328_ack(++my_packet_id);
                                if (img_byte_cnt >= img_size) {
```

```
                                                //sprintf(str, "# ImgComplete,
Total=%u, Recv=%u.\r\n",
                                        //img_size, img_byte_cnt);
                                                //uart0_write_str(str);
                                                //disp_buf_hex(img_data_buf,
img_size);
                                                image_to_serial(img_data_buf,
img_size);
                                                c328r_state =
CMD_GET_PICTURE;// 9;
                                        }
                                        c328r_rx_len = 0;
                                        c328r_timer = 1000;
                                }
                                else {
                                        c328r_state = CMD_GET_PICTURE;
                                 }
                                }
                                if (uart3_read_byte(&byte)) {
                                //sprintf(str,"[%02X]",byte);
                                //uart0_write_str(str);
                                c328r_timer = 10;
                                if (c328r_rx_len < sizeof(c328r_rx_buf)) {
                                        c328r_rx_buf[c328r_rx_len] = byte;
                                        c328r_rx_len++;
                                }
                                }
                        }
#endif
        }
```

```
}
void disp_buf_hex(u8t *p, int len) {
	char str[32];
	int i;
	uart0_write_str("# DataHex:\r\n");
	for (i=0; i< len; i++) {
		sprintf(str, "[%02X]", p[i]);
		uart0_write_str(str);
	}
	uart0_write_str(".\r\n");
}
/**
 * image serial protocol.
 *
 * @author Administrator
 *
 * @param p
 * @param len
 */
void image_to_serial(u8t *p, int len) {
	char str[32];
	int i;
	uart0_write_str("{");
	for (i=0; i< len; i++) {
		sprintf(str, "$%02X", p[i]);
		uart0_write_str(str);
	}
	uart0_write_str("}\r\n");
}
void debug_send(u8t *p, int len){
```

```
	char str[32];
	int i;
	uart0_write_str("TX=");
	for (i=0; i< len; i++) {
		sprintf(str, "[%02X]", p[i]);
		uart0_write_str(str);
	}
	uart0_write_str(".\r\n");
}
void debug_recv(u8t *p, int len){
	char str[32];
	int i;
	uart0_write_str("Rx=");
	for (i=0; i< len; i++) {
		sprintf(str, "[%02X]", p[i]);
		uart0_write_str(str);
	}
	uart0_write_str(".\r\n");
}
```

ประวัติผู้เขียน

นายภูมินท์ ดวงมณี เกิดเมื่อวันที่ 7 กุมภาพันธ์ พ.ศ. 2518 ที่จังหวัดสุพรรณบุรี เริ่มศึกษาชั้นประถมศึกษาปีที่ 1-4 โรงเรียนวัดสามชุก จังหวัดสุพรรณบุรี ชั้นประถมศึกษาปีที่ 4-5 มัธยมศึกษาปีที่ 1-6 โรงเรียน ภ.ป.ร. ราชวิทยาลัย จังหวัดนครปฐม และสำเร็จการศึกษาระดับปริญญาวิศวกรรมศาสตรบัณฑิต (วิศวกรรมโทรคมนาคม) จากมหาวิทยาลัยเทคโนโลยีสุรนารี จังหวัดนครราชสีมา เมื่อปี พ.ศ. 2539 และในปี พ.ศ. 2550 ได้เข้าศึกษาต่อในระดับวิศวกรรมศาสตรมหาบัณฑิต สาขาวิศวกรรมโทรคมนาคม มหาวิทยาลัยเทคโนโลยีสุรนารี โดยในขณะศึกษาได้รับทุนในการทำวิจัยจากสำนักงานกองทุนสนับสนุนการวิจัย (สกว.)

ในระหว่างที่ทำการศึกษาได้เผยแพร่บทความทางวิชาการเรื่อง **"Implementation of Real-Time Video Streaming with Fuzzy Logic Controller"** ในงานสัมมนาวิชาการนานาชาติ WCSP 2010 (The IEEE 2010 International Conference of Wireless Comunications and Signal Processing) ในระหว่างวันที่ 21-23 ตุลาคม พ.ศ. 2553 ณ ซูโจว ประเทศจีน