Delays in Axis IP Surveillance Cameras

Linus Svensson
Petter Söderlund

LUND
UNIVERSITY

MASTER THESIS in Automatic Control 2013

Supervisors: Petter Johansson, Axis Communications

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

ISSN 0280-5316
ISRN LUTFD2/TFRT--5923--SE

Printed in Sweden by Media-Tryck.
Lund 2013

Abstract

Controlling motors over IP networks naturally introduces delays. For an operator of a surveillance camera to point the camera (e.g. following an object) delays are of importance.

This master's thesis investigates the sources of steering delays in IP-cameras, testing on AXIS Q6035 and AXIS Q6032 with different video management software and video codec. Camera firmware and communication protocol are evaluated with aim to find improvements.

The results show that the client computer, displaying video and sending steering commands, contributes with the largest uncertainty in the system. Video management software and computer load are the key factors. Camera firmware proved to have inefficient communication schemes, which when changed, reduced delay by 70 ms.

Acknowledgements

We want to thank Axis Communications for giving us the opportunity to write this master's thesis. We have received help from many of Axis employees and through good discussions given us advices to help our work. A special thanks to Joakim Andersson for helping us with our equipment and the development environment at Axis.

Thanks to Karl-Erik Årzén that through continuous meetings provided us with advice and guidance to complete this thesis.

Finally, a large thanks to our supervisor Petter Johansson at Axis for all his help and encouragement throughout the whole work.

Contents

List of Figures

List of Tables

Abbreviations

AF	Assured Forwarding
API	Application Programming Interface
CGI	Common Gateway Interface
CS	Class Selector
DiffServ	Differentiated Services
DSCP	Differentiated Services Code Point
DTU	Data Transfer Unit
ECN	Explicit Congestion Notification
EF	Expedited Forwarding
FPS	Frames Per Second
GOP	Group of Pictures
HTTP	HyperText Transfer Protocol
IntServ	Integrated Services
IP	Internet Protocol
LED	Light-Emitting Diode
MJPG	Motion JPG
OSI	Open System Interconnection
PC	Personal Computer
PHB	Per-Hop Behavior
PTZ	Pan, Tilt, Zoom
QoS	Quality of Service
RSVP	Resource Reservation Protocol
RTT	Round Trip Time
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
ToS	Type of Service
UDP	User Datagram Protocol
USB	Universal Serial Bus
VMS	Video Management Software
VoIP	Voice over IP

1. Background

1.1 Introduction

Axis Communications is a company based in southern Sweden that since 1996 has been developing and producing digital surveillance cameras. A segment of these cameras are steerable with motors connected to the camera enabling the user to change camera view. This segment is called pan, tilt, zoom-cameras (PTZ) and can be controlled in the manners the name implies. The steering of a camera is controlled through a client computer with a certain software, either with a joystick or mouse interaction. The client computer is connected to the camera via an Internet Protocol (IP) network. A client computer can be connected to several cameras simultaneously showing several individual live streams, a camera can at the same time deliver video streams to multiple clients.

Of all surveillance cameras installed the analogue cameras are of a majority. Customers upgrading their systems still have a choice of installing analouge or digital surveillance systems [1]. Even if the analogue technology seems to soon be replaced fully by its digital ditto some characteristics still separate the technologies. A drawback of the digital cameras is a reputation of having a longer delay for the picture to reach the monitor and steering the camera when e.g. following an object [6]. A typical setup of a digital surveillance camera is illustrated in Figure 1.1. This thesis is intended to investigate the delays associated with the steering of the cameras, and find ways to make them shorter.

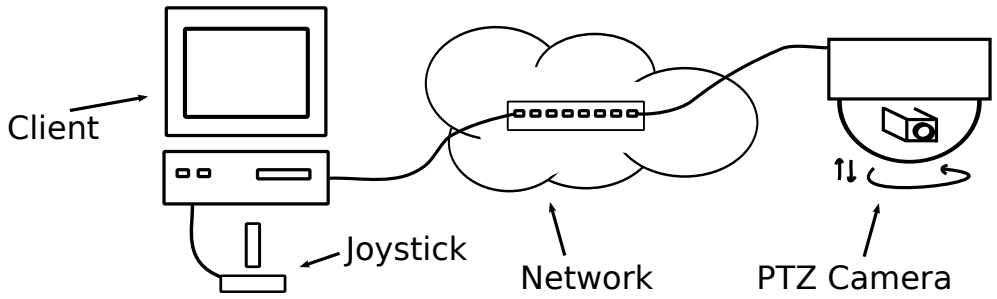


Figure 1.1 A basic overview of the environment investigated. The cloud holding a switch represents an unknown IP-network. A client PC with a joystick is attached to the left and a network camera to the right.

1.2 First principles

When controlling a camera with a joystick the time delay from pulling the joystick to a change on the screen showing the camera view is noticable by a human. To improve the user experience steering a camera and decreasing errors e.g. not succeeding to follow a target on screen is in direct correlation to the duration of the input delay from joystick movement to screen feedback, [8].

To identify what changes that could have a positive effect on the steering latencies detailed measurements of the system are needed. With a measuring mechanism that

shows the time intervals of the individual processes throughout the whole chain of communication would ease the decision of which changes that could make a difference to the delay. The main goal of this thesis is to lower the latencies associated to steering PTZ cameras over IP networks.

Surveillance cameras are used to monitor and record events of a designated area such as a grocery store, on public transportation, or in a museum. Larger sites such as airports can have a massive amount of camera units installed. The cameras are usually monitored and controlled by personnel equipped with Personal Computers (PC) in a command central. The PC connected to the cameras will also be denoted as the client.

The network structure connecting a client to an IP camera is very dynamic compared to analog surveillance networks. Analog surveillance networks require designated cables for each unit compared to the digital technology where it is easier for several digital units to communicate using the same network. The many options of constructing an IP network does, however, make it impossible to simulate all possible use cases.

There are a number of ways for a client to communicate with a camera. Software made for interacting with surveillance cameras are denoted Video Management Software (VMS). PTZ movement commands along with many other commands are when using a VMS usually sent through HTTP-requests.

1.3 Delimitations

The total delay for steering the camera is defined as:

> The time from the first motion of the joystick to a reaction on the monitor corresponding to the joystick movement.

The investigation and measurements covers the whole system but only the delay from client move command to motor movement will be optimized. PTZ movement commands will be limited to only pan and tilt commands, not including measurements of the zoom command. The network structure used throughout the tests in this study is a dedicated, isolated 100 Mbps IP-network unless stated otherwise. Furthermore, analysis is concerned with use cases where only one client is connected to the camera with a single video stream.

1.4 Goals and strategies

The main goal is to investigate the steering delay, and if possible find solutions that may improve the response time. A set of four different tracks to follow were identified in the work lowering the delays in the system:

1. Mapping of the current system
   It is known that there is a delay when steering a PTZ camera. The total time for that delay can be split up into separate components. It is, however, not known how much time is spent in what component. One of the major activities is to determine how much time is currently spent in what part of the system.
2. Skipping steps within the camera and changing protocol
   The camera steering is communicated through HTTP requests where a new connection is established and torn down for every movement command sent.

Within the firmware several processes are involved when a movement request is served. From earlier projects within Axis it has been shown that when the camera is put under the pressure of several streams the camera performance does not scale up well. A choice of placing a socket connection dedicated for PTZ movement commands deeper within the camera firmware could prove valuable in time since that would skip several context switch steps and inter-process communication steps. The new socket could then be based on another protocol than the HTTP based protocol used today.

3. Prioritize network traffic
   The amount of communication data concerning PTZ movement is small in relation to video streams which are processed over the same network. It could prove valuable to give the PTZ steering commands higher priority on the network.

4. Physical relocation of the joystick
   The joystick is currently connected to a client Windows PC. It may prove valuable to move the joystick in the network making it its own host with dedicated hardware only handling joystick operations. Internal management of signal from USB interface to package sent may be a problem with the current setup. Another problem difficult to control is prioritisation of processes in the client, which deadlines are most important - recording video, displaying video or controlling a joystick?

   Isolating the joystick to a minimal embedded system dedicated to run PTZ-commands connected separately to the network might give a higher control of how the control signals are handled in the client and assure stability of the system.

1.5 Thesis outline

Chapter 2 describes the environment, surrounding software applications, and the internal structure for a PTZ camera.

Chapter 3 describes basic theory needed to understand design choices and why certain strategies to improve the delays were attempted.

Chapter 4 presents methods for measuring performance in subsystems and their results. Construction of prototypes and improvements of the product are also investigated.

Chapter 5 summarises the test results presenting an overview of the system delays.

Chapter 6 presents a discussion and conclusions. Finally a short discussion on future work is performed.

1.6 Division of labour

Petter has had a main responsibility of measuring and testing of the current system whilst Linus has spent more time of design and implementation of new communication protocols for sending commands to the camera.

2. Environment

When measuring delays in any real-time system it is important to know the environment. This chapter has a description of all parts included in software and hardware used to steer a PTZ camera (Figure 2.1).

Figure 2.1 Photo of an AXIS Q6032 PTZ dome camera, which is one of the camera models used in this thesis.

2.1 Overview

There are many components involved when steering a camera and the possible sources of delays are many. The actual system setup has two isolated activities which is to send PTZ movement commands to the camera and retrieve a video stream from the camera, which is illustrated in Figure 2.2. When evaluating the PTZ functionality regarding delays, it is also important to include image streaming.

Move command

The components involved when sending the command can be divided into four categories:

- Input device capturing human commands.
- Client application sending move commands to the camera.
- Network transporting the steering commands.
- Axis camera receiving steering commands.

The typical use case for a command to be sent is that a human triggers a desired action and the client application sends a request to the camera through one of the Application Programming Interfaces (API) that the camera supports. The command is sent over an IP based network and reaches the camera where the firmware and hardware reacts to the command.

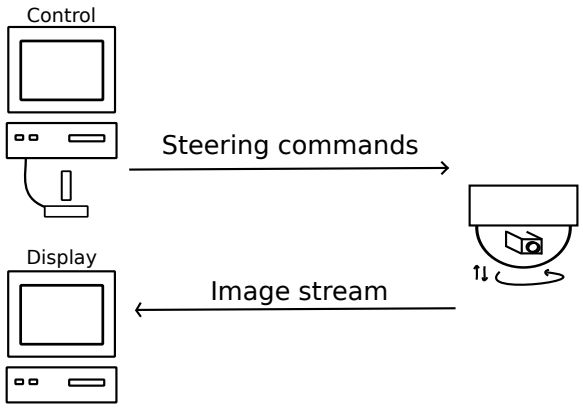


Figure 2.2 An overview of the environment for the system setup when controlling a PTZ camera. From the camera point of view there is no connection between streaming video and control the PTZ functionality.

The video stream can be accessed from a camera in many ways with different settings for attributes like resolution, frame rate or video codec. In this thesis the delay of the video stream is of importance, since it contributes to the total delay of the system. The delay can vary depending on the attributes of the stream, load on the network, and even what client software is used to display the stream.

Video stream

There are several phases involved when streaming video including capture from the camera lens, encode to a digital representation and decode before the picture is displayed on the monitor. This experiment does not intend to investigate properties and delays in parts of the stream in detail. The phases in the video stream is, however, structured in three phases:

- Axis camera sending video stream.
- Network transporting the video stream.
- Client application displaying the video stream.

2.2 Camera platform

Control of Axis products is done using an API, and Axis has developed an own API called VAPIX which is based on HTTP. Axis cameras also supports the global ONVIF[1] standard. ONVIF is an organization developing global standards for interfaces in IP-based security products.

In this thesis the VAPIX API has been used since it is fully developed by Axis and well suited for the products.

In this section a brief description of VAPIX is presented, followed by a description of the camera platform and the framework for PTZ.

VAPIX

The VAPIX documentation is available at [2] and there are separate documents for the different kind of Common Gateway Interfaces (CGI) that can be accessed using

[1] http://www.onvif.org

the VAPIX API. One reason for the existence of VAPIX, and a large advantage, is that the API is the same for all Axis cameras. If the camera for example supports move commands the external API is the same independently of the camera hardware.

To control the steering of the camera there are commands for absolute, relative and continuous movement. The absolute movement command moves the camera to a fixed coordinate and is suited when the exact position is known. Relative movement moves the camera a fixed distance in pan and tilt direction relative to the current position. The continuous movement is for the purpose of control as a human operator using e.g. a joystick. When such a command is sent to the camera it moves in that direction until it receives a command that stops that movement.

PTZ framework and the camera platform

An Axis camera is an embedded microcomputer that runs Linux as operating system. There is a lightweight web server installed that serves incoming HTTP requests, including requests via the VAPIX API.

There is a framework to handle PTZ requests which is illustrated in Figure 2.3. When a command reaches the camera the web server handles the request and if it is a CGI call to the PTZ framework the whole file descriptor of the socket is transferred to the PTZ framework. The PTZ framework is a daemon always running, a new process is not spawned on every request. The PTZ framework actually consists of individual processes itself and the one that retrieves messages from the web server is the VAPIX parser. The parser parses the input parameters and performs error checking.

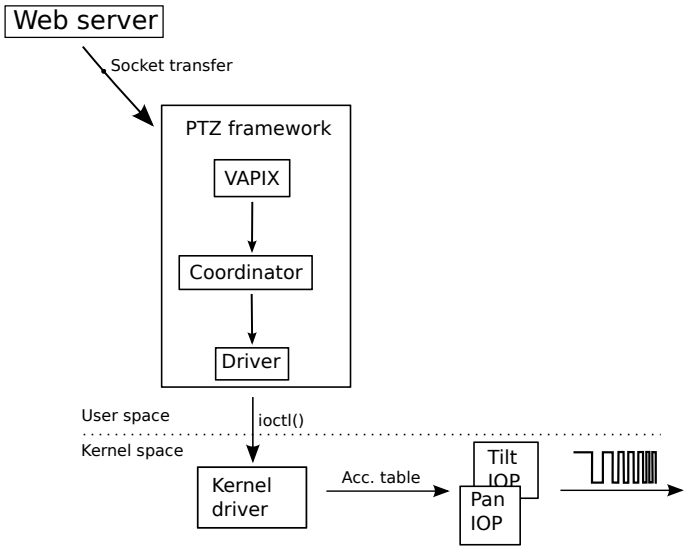


Figure 2.3 PTZ framework overview with processes and communication involved when serving movement commands. The web server transfers sockets to the PTZ framework, which is the core system for the PTZ products. The PTZ framework communicate with the motors through a kernel module via the `ioctl()` system call. Both the pan and tilt motors has one IO processor each that generates a pulse determining the direction and speed of the motor.

When the options has been parsed the request is transferred to the coordinating process, using the D-Bus message bus system. The coordinating process has responsibilities at the system startup and as the name implies it is responsible for coordination. If a camera is mounted on a wall or in the ceiling it supports image rotation of the camera view. The horizontal positioning of the camera is unknown to other devices, like VMS, which means that steering commands to the camera have to be adjusted to the image rotation. The coordinating process in the PTZ framework is responsible to

change the movement commands according to image rotation.

When all parameters in a request have been processed and translated by the coordinator they are packed as a command and sent to the driver process. There can actually be more than one driver process, where one can handle the pan and tilt motors and another one camera zoom. The setup varies between camera models but the coordinator always sends the command to the driver processes, which then decides if it needs to do anything with the command.

The driver process is designed in two blocks. Since there can be more than one driver process running at the same time the external interface for IPC looks the same for every driver. This can be referred to as the driver frontend. The driver has a corresponding backend whose implementation is directly correlated to the camera hardware. The backend is responsible for communicating with a module uploaded in the Linux kernel.

The camera direction is controlled through two stepper motors, one for pan and one for tilt. Apart from the direction both motors work with the same kind of mechanics. The motors are controlled with IO processors that generates logical pulses. The kernel module translates move commands to pulses with help of the IO processors.

2.3 Network

Axis cameras communicate over IP networks. The impact on the delay is dependent on the size of the network and the amount of other traffic on the network. Chapter 3 has an explanation of the basic concepts for IP networks and how they work.

2.4 Video management software

The client software that the user runs is called Video Management Software (VMS) and is run on a desktop computer. Axis develops two systems on their own namely AXIS Camera Companion, for small systems with up to 16 cameras, and AXIS Camera Station, for medium sized systems with up to 100 cameras. For systems with more than 100 cameras Axis do not perform the development themselves.

Every camera supplies a web page with a live view from the camera, which is accessed from a web browser by typing the IP address of the camera.

Pan, Tilt, Zoom control

The continuous move command is suited to steer the camera with a joystick. The only task the VMS need to do is to send the current joystick position to the camera. The control is based on events, where the client needs to send a new command every time the position of the joystick has changed. The implemented solution in the client software can impact, not only the delay of a single move command, but also how it feels to maneuver the camera.

The communication from the client software to the camera is event based, however, it is not certain that the interaction between the joystick and the client software is event based or not.

If the communication with the joystick is based on polling the worst case scenario happens when the position was read just before a movement. The worst case for the event driven technique is the case where a huge amount commands are triggered simultaneously and this generates a congestion somewhere in the system. It is

unknown which principle is used for the client software in this case since both will work.

The sources of delay is not restricted to but include:

- Hardware delay in the joystick
- Design principles of the client software

2.5 Video streaming

Video streams are obtained via the VAPIX API. Axis products can stream video with different properties, including:

- Video codec
- Resolution
- Frame rate
- Compression

A major part of the settings has a range of valid values where the frame rate can be any number between 1 and 30 for example. The properties are product dependent and the highest possible resolution may not always be combined with the highest possible frame rate.

Since the video is going to be passed over an IP network the video need to be encoded in the camera, and later on decoded in the client application receiving the video stream. Every step takes time to perform and may cause delay in the video stream. If the encoding algorithm is advanced and consumes lots of calculation power, the encoding delay is large and if the video codec is simple and no compression is used the network need to carry more data and the network delay is large.

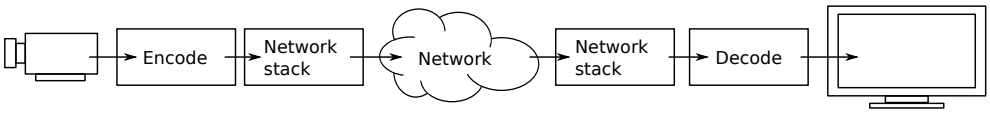


Figure 2.4 The stages involved when streaming video over an IP based network. A camera captures the video, encodes it and sends it on the network. The client retrieves it, decodes it and displays it at the monitor.

On the client side the video stream needs to be decoded and more advanced video codec demands more processor power as it does on the camera side. A simple view of involved blocks when streaming video over IP based networks can be seen in Figure 2.4. The client side often use a buffer to get a better flow in the video stream, making it robust and not sensitive to temporarily disturbance in the transmission. The size of such a buffer may introduce longer delay to display at the monitor.

3. Theory

This chapter is intended to give an introduction to the theory used in this thesis.

3.1 The OSI model

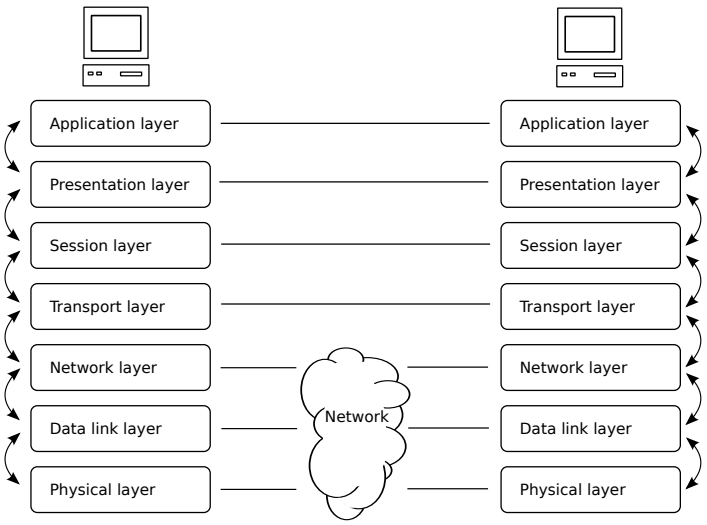


Figure 3.1 The concept of the OSI model is that the layers indirectly communicates with layers on the same level in the other host, thereof the horizontal lines between layers. Directly a layer only communicates with the layer below and the layer above, this makes the layers interchangeable and is good for maintainability – You can change one layer without rebuilding the whole stack. The network between the two hosts is represented by a cloud in the middle which intercepts the lines of the lower three levels. This represents that the information from these layers are of importance for the network and might be checked before the data reaches its intended receiver.

The Open System Interconnection model (OSI), from International Standards Organization, is a standard for network communication. The model consists of 7 different layers, each with their own responsibilities, and can be seen in Figure 3.1. The purpose of the model, and its righteousness, is not to define a protocol for network communication, but to define a robust and flexible way to let otherwise incompatible devices communicate with each other.

The 7 layers are: physical (layer 1), data link (layer 2), network (layer 3), transport (layer 4), session (layer 5), presentation (layer 6), and application (layer 7). When two devices communicate with each other the applications or protocols at the same layer in the OSI model communicate with each other directly. This is however not exactly correct since the communication in layer one, two and three is performed device-to-device along the path if two communicating devices is connected through several intermediate nodes. In one node a layer uses services from the layer beneath it and serves the layer just above it. One important advantage of this layout is the ability to change the implementation at one layer without making any difference at the other layers.

The layers can be divided into groups where layer five to seven is about support for different software on a host to communicate with other software on other machines. Layer one to three is responsible for the transportation of data, electronic

encodings, connections, physical and logical addresses, reliability of the transmission etc. Layer 4 can be seen as a middle layer that gives the higher layers the support they need to transport data. The first three layers are the only ones involved in so called hop-to-hop delivery and large networks can be built up of nodes that only support these three layers (or even fewer in some cases), [4].

In each layer the Data Transfer Unit (DTU) refers to the entity that is being sent.

Physical layer

The physical layer is responsible for the transmission of individual bits between nodes and the encoding of bits. It also defines type of transmission medium like copper wire or optical fibre, and the physical interfaces. The DTU are individual bits.

Data link layer

The data link layer is responsible for transportation of data packed in frames between two directly connected nodes in the network. The layer is also responsible for physical addressing and to deliver an error free stream of data to the network layer. If two nodes in an end-to-end communication is located at two different networks the physical destination address of the data link transmission is the node that connects the sending node to the receiving nodes network. The DTU is frames which are composed of a stream of bits.

Network layer

The network layer is responsible for the transmission from a source to the end destination of individual DTUs called packets or datagrams. The packets can travel through lots of connected networks and at the routers or switches the service operating at this layer must provide some routing mechanism to route the packet in the right direction.

Transport layer

While the three first layers address devices like routers, switches or computers the transport layer is addressing processes. The address is a number called port number. This layer is responsible for delivering whole messages between processes at different machines. This layer also provides functionality of error control and flow control. The communication at this layer can be, so called, connectionless or connection oriented. The transport layer is only involved in the source and final destination nodes. Error control is often implemented through retransmission of packets.

Session layer

In some cases a process may need more services than the lower layers can offer in terms of synchronization and control of the dialogue. The session layer offers communication in either full- or half-duplex. If a large file is to be transferred the session layer protocol may insert checkpoints to ensure that the whole file does not need to be resent if the connection breaks during the transfer.

Presentation layer

The presentation layer is as it names implies responsible for how data is presented. At this layer data is encrypted, compressed and translated if two communicating devices does not share the same representation of data. If a standard integer is to be sent over the network and the sender uses two complement representation of integers while the receiver does not the presentation layer application transforms the integer to a common format before transmitting and transforms it for the target platform at the receiving end.

Application layer

The topmost layer provides services for user such as email, distributed file systems and HTTP. Some services provide users the ability to influence the communication in some ways like, use HTTP with SSL/TLS[1].

3.2 Protocols

Internet protocol

The Internet Protocol (IP) [12] is a network layer protocol based on best effort delivery of data - the protocol itself does not provide any flow control treating each internet datagram independently unrelated to other internet datagrams . Error control of the header is obtained through a checksum protecting the header information, IP does not supply any error control for the payload of the datagram. Figure 3.2 shows the structure of an IP header.

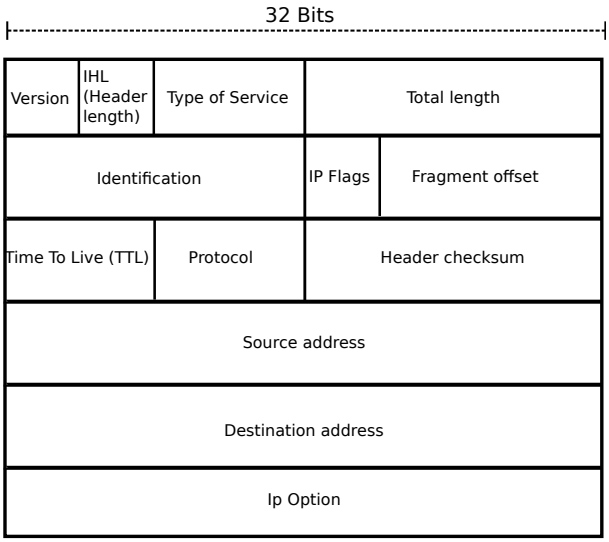


Figure 3.2 The structure of an IP header illustrated.

Transmission Control Protocol

Transmission Control Protocol (TCP) [13] is a transport layer protocol with a connection orientation which means that the protocol keeps track of both error control and flow control. Connection orientation makes the protocol reliable. Flow control makes sure that all packages will be received in the order they were sent. Error control ensures that broken and lost packages are resent. To transmit information a connection is established through a three-way handshake, see Figure 3.3. The life cycle of a TCP connection is:

1. Establish connection
2. Send data
3. Close connection

[1] SSL/TLS are protocols at the presentation layer in the OSI model.

Compared to UDP it is easier to encrypt information and establish safe connections e.g. with Secure Sockets Layer (SSL). TCP is used for applications where a reliable connection is needed and all data is important to the receiver such as file transfer and e-mail.

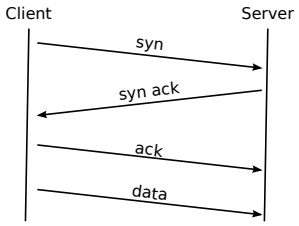


Figure 3.3 An initiation of a TCP session requires at least three steps. Syn packet from the client, server responding with a syn ack packet completed by the client answer of an ack packet.

User Datagram Protocol

The User Datagram Protocol (UDP) [11] is a connectionless protocol and is therefore considered unreliable because of its lack of flow control and error control limited to the header. The data is pushed to the network containing little overhead consisting of sender and receiver ports, length of the data and a checksum. In a best effort manner the sender usually sends data periodically and it does not matter if all are received. Real time applications where delays is of importance such as computer games and media streaming can often use the benefits of the UDP protocol.

3.3 Quality of service in networking

Quality of Service (QoS) is a widely used expression, and has different meanings depending on the context. In network communication different kind of applications have their own demands on the characteristics of the communication. There are four categories of attributes of a transmission:

Delay the time it takes from source to destination.

Throughput the rate of the transfer. The amount of data being delivered in a time unit (bits per second).

Reliability the entity for the probability of a packet loss in the transmission.

Jitter the variation in delay of individual packets.

There are lots of applications with different purposes and each with their own requirements from the categories. An e-mail transfer for example can tolerate delays and lack of throughput but is very dependent on reliability, to ensure that no data is lost. A telephone call over IP on the other hand does not suffer that much from reliability but has a huge requirement on the delay.

At the IP level of networking there are two approaches to ensure QoS, one based on flows and the other one on classes. Integrated Services (IntServ) is based on flows where each session for communication needs to reserve resources amongst the nodes it passes from the source to destination. With help of the Resource Reservation Protocol (RSVP) resources like bandwidth, space in router buffers etc, is reserved to create a predefined flow with desired attributes, in the otherwise connectionless IP protocol.

With IntServ each router must store all information of each flow it serves for the moment and all nodes at the path must support IntServ. One shortcoming with IntServ is that the number of flows can be very large because of the growth of internet, which introduce a limitation in the scalability. The other approach is Differentiated Services (DiffServ) which is based on classes and supports scalability and does not require so much from the intermediate nodes. DiffServ is described in the next section.

3.4 Differentiated services

Differentiated Services defines a way to assign priorities to individual packets in IP communication. The Type of Service (ToS) field in the IP specification was proposed to be replaced with the DiffServ field which is a codepoint for devices that implement Differentiated Services [9].

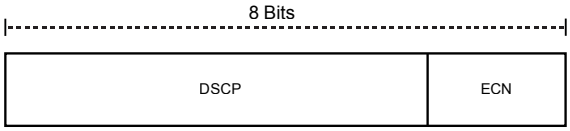


Figure 3.4 The structure for the DiffServ field. It replaces the eight bit ToS field in the original IP header. The first six bits are DSCP and the last two ECN.

The DiffServ field is illustrated in Figure 3.4. The first 6 bits are used for the Differentiated Services Code Point (DSCP), which is used for prioritisation of IP packets. The last two bits was first denoted Currently Unused (CU) but is now used for Explicit Congestion Notification (ECN).

DiffServ defines so called Per-Hop Behaviors (PHB) which is a rule for the forwarding of a single IP packet. A router implementing DiffServ needs only to know how to treat a packet of a certain PHB. In IntServ the router also needs to maintain information of all active flows and how to treat them. In DiffServ the responsibility for assigning the right priority of a transmission is maintained by the sending node only.

There exists standard recommendations for PHBs. Some of the standard PHBs are listed in Table 3.1, with their corresponding DSCP and ToS values.

Class Selector

In the original ToS field of the IP header the first three bits were for IP precedence and used for prioritization in case of troubles like congestion. Packets with lower precedence were discarded first. When the three rightmost bits in the DSCP is set to zero the first three bits is treated as precedence in the original implementation. The reason for this is backward compatibility.

Assured Forwarding

Assured forwarding (AF) PHB [5] is used to divide traffic in classes to assure a level of forwarding in each class. A class can be assigned a proportion of the available resources in terms of bandwidth and/or buffer space within a routing device. The first three bits in the DSCP are used to indicate the AF class. Four standard classes are defined and they are AF1, AF2, AF3 and AF4. Within each class there is three drop rates levels: low, medium and high. To address a class and a certain drop rate one use AF11 (low), AF12 (medium) and AF13 (high). When there is congestion within

Per-Hop Behavior	DSCP (bin)	DSCP (dec)	ToS value
Default	000 000	0	0
CS Priority	001 000	8	32
AF11	001 010	10	40
AF12	001 100	12	48
AF13	001 110	14	56
CS Immediate	010 000	16	64
AF21	010 010	18	72
AF22	010 100	20	80
AF23	010 110	22	88
CS Flash	011 000	24	96
AF31	011 010	26	104
AF32	011 100	28	112
AF33	011 110	30	120
CS Flash Override	100 000	32	128
AF41	100 010	34	136
AF42	100 100	36	144
AF43	100 110	38	152
CS CRITIC/ECP	101 000	40	160
Expedited Forwarding (EF)	101 110	46	184
CS Internetwork Control	110 000	48	192
CS Network Control	111 000	56	224

Table 3.1 Recommended PHBs and their DSCP values first given in binary form, followed by DSCP and ToS value in decimal form.

a class packets with high drop probability has a higher chance of being dropped. The drop chance is encoded in the fourth and fifth bit in the DSCP field.

If one class has reached its limit of resource usage, it is allowed to use even more as long as another class has resources left that it does not need for the moment.

Expedited Forwarding

Expedited forwarding (EF) PHB [3] is used to mark traffic that is sensible to delay, reliability and jitter. Voice over IP (VoIP) is a recognizable service that make use of this forwarding policy. Routing devices often have restrictions on the proportion of EF traffic allowed.

The benefit with EF PHB is a low delayed stable transmission at the cost of bandwidth.

3.5 Video streaming

Digital representation of a video stream is defined by its video codec. The video codec describes the video encoding, compression, etc. The mathematical complexity of the

video codec determines how computationally intensive the video will be to encode, decode, record and play.

Motion JPG

Motion JPG (MJPG) is a video stream of individual pictures encoded with JPEG compression put in a series. Compared to other established video codecs such as MPEG-2 there is no official standard for MJPG, there are, however, several vendor specific implementations. The frames in a MJPG stream are individually encoded not interlacing each other making the video codec insensitive to movement in the video. The way of treating frames individually leads to a lower grade of compression compared to video codecs that analyzes video frames in sequences. MJPG is well suited for frame by frame analysis and modification, [10].

H.264

H.264 is an advanced video codec. Frames in a H.264 video are of different types: I-frames, P-frames and B-frames. I-frame is short for intraframe and also called keyframe delivering a full picture. P-frames, or predictive frames, starts in an I-frame and only describes the differences from the last I-frame. The ratio of I-frames to P-frames are sometimes denoted Group of Pictures (GOP) ratio. A lower GOP ratio value (few P-frames, many I-frames) gives a higher image quality always delivering the full information of the picture and therefore also consists of more bits putting higher requirements on storage or network capacity. B-frames, bi-directional predictive frames, rely on frames preceding and following them containing only the data that has been changed. A typical GOP frame order could be IBBPBBP. Compared to the MJPG video codec where the decoder only has to display pictures in a certain order the H.264 video codec has higher demands on receiving hardware, [14].

4. Evaluation

The focus for this thesis has been to evaluate the delays when controlling a PTZ camera and find room for improvements, that reduces the delay.

The first task was to choose what test method to use to measure the PTZ steering delays. The decision was to use a method where the test setup was recorded with a camera capable of recording at 60 Frames Per Second (FPS). The choice of the method was based on previous tests and knowledge at Axis. During the study many tests with different kind of softwares and/or configurations were executed. Three delays were chosen as both of great value and possible to measure:

- Total delay as the time it takes from a steering device like a joystick is moved until one can see the change in the video stream at a monitor.
- Video delay as the time it takes from that the camera captured an image until it is transferred and presented at the client monitor.
- PTZ delay as the time it takes from steering device movement to actual movement of the camera.

Ideally the equation

$$\text{Total delay} = \text{Video delay} + \text{PTZ delay}$$

holds perfectly, but that is not the case because there is always an uncertain factor with the timing inside the camera. As mentioned in Chapter 2, control commands are separated from the video stream in the system and that leads to the possibility of having timing misses inside the camera. In Figure 4.1 it is illustrated as the time between

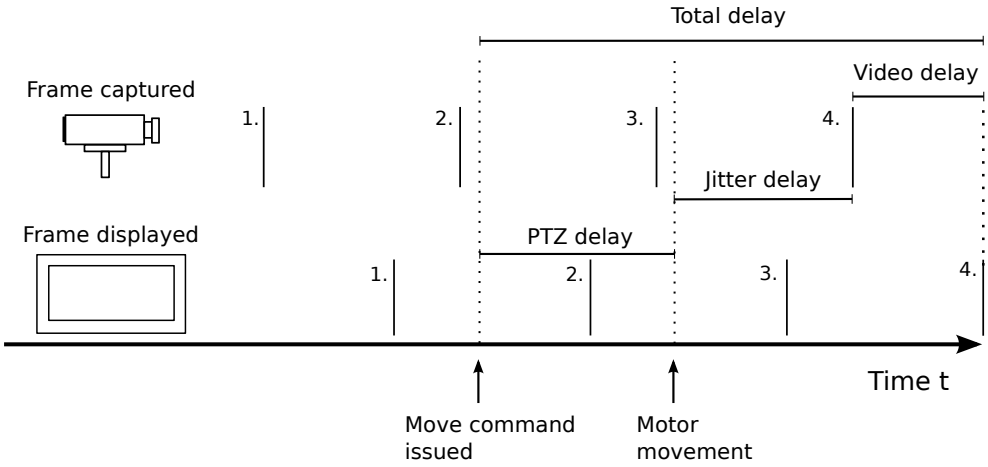


Figure 4.1 The first row (Frame captured) of numbered lines represents each time the camera has captured a frame. The second row (Frame displayed) corresponds to the time when the frame with equal number from the first row is displayed on a monitor. Move command issued corresponds to an external event requesting camera movement (joystick movement). Jitter delay is the time it takes for the camera to register a change in the camera view.

the start of motor movement to the time when the next picture is captured. This yields that the total delay may get a contribution to up to the time it takes between two individual frames. That extra delay is denoted as jitter delay. That jitter delay is limited to the time between two adjacent frames, which for a camera streaming at 30 FPS

means a jitter delay up to 34 ms. In average the jitter delay is half of the time between two adjacent frames. The real relation for the delays is described by the equation:

$$\text{Total delay} = \text{Video delay} + \text{PTZ delay} + \text{jitter delay}.$$

Methods to measure the total delay and video delay was defined as they were suitable candidates to measure. The PTZ delay was defined to be the calculated difference between the total delay and the video delay. The tests measuring the PTZ delays also includes the jitter delay.

In many of our tests the time difference between two events was measured with help of a video camera. The video stream from that camera was retrieved with the MJPG codec and split up in a sequence of images. The process of defining the methods is described in Section 4.1 and 4.2 with the resulting measurement procedures presented in Appendix A and B.

4.1 Video delay

The final method used for testing streaming delays was developed through testing alternatives seeking which method could deliver the most consistent results. All methods tried had the same basic idea of the camera to be tested to film its own video stream creating a loop of images. Placing a high precision stopwatch in the view of the camera giving timestamps to the separate frames in the image loop, see Figure B.3 on page 44. The timestamps gives a measurement of time from camera recorded frame to its display on a monitor.

Test construction

The extraction of the monitor's displayed frames has to be done with as little interference with the camera and client computer as possible. Before determining the final test method, experiments with taking screenshots of the monitor was tried out. The effect of taking screenshots was cumbersome for the client system and lowered the

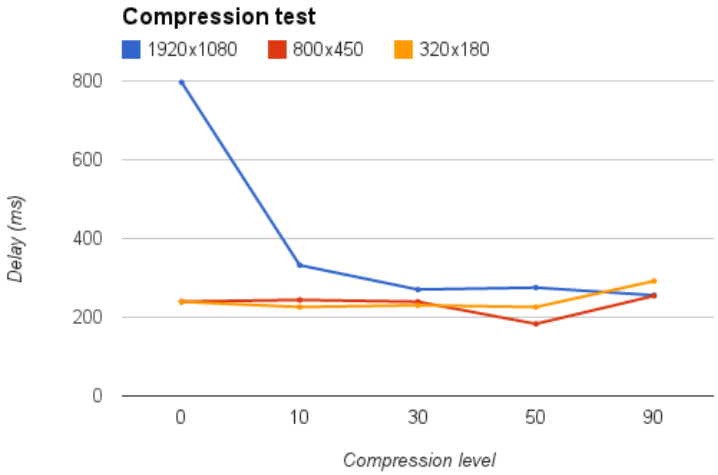


Figure 4.2 Video delay results from tests different levels of compression combined with three different resolutions. Testing was done with VLC according to the test method described in Appendix B using MJPG as video codec.

capability of putting through the video stream at a constant rate. Another method tried was to record a second image stream from the camera being tested, this did, however, introduce a constant delay compared to filming the monitor with a second camera to a second computer which ended up as the method used. For a full description of the video delay test method see Appendix B.

There are several parameters to set when streaming video; Video codec, compression level, frame rate and image resolution were the ones of focus in the tests with the goal to understand where and why delays are introduced. The camera tested is an AXIS Q6035, a large PTZ dome camera from Axis' high-end product selection. The camera can deliver video through MJPG and H.264 with resolutions from 320x180 pixels to 1920x1080 pixels. There is an option to compress the video in the camera on a scale from 0-100 where 100 gives maximal compression on an Axis specific scale.

To decide what settings to use when evaluating the camera a series of tests were done. The first test treated image compression using a Linux environment with VLC[1] as the streaming software client. The media player chosen has been recommended by other studies and is available to a large selection of platforms [7]. The tests treated different compression levels using MJPG video codec at a variety of image resolutions. Test results for compression are presented in Figure 4.2. Conclusion drawn from the compression test was to do all future tests at a compression level of 30 which gives a clear performance advantage to no compression at all, with shorter delay times still keeping a lot of information in the picture.

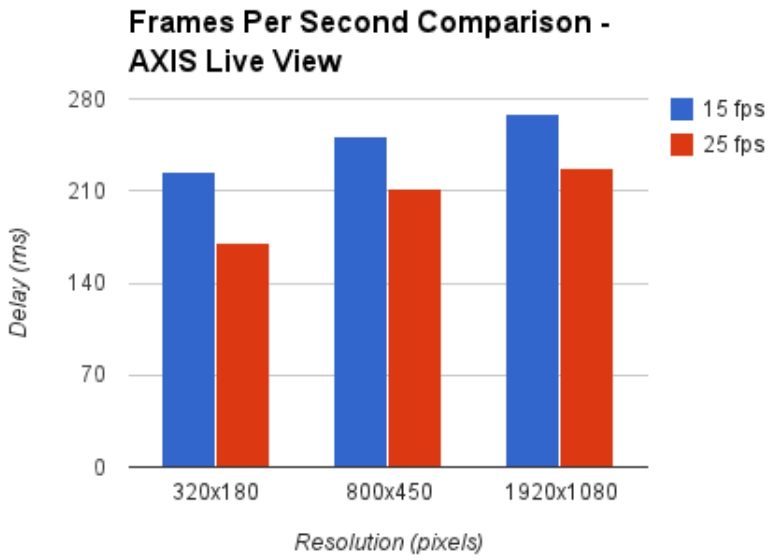


Figure 4.3 Video delay results from tests with 15 and 25 frames per second at three different resolutions. Tests were carried out according to the procedure in described in Appendix B using Axis Live View in Google Chrome web browser and MJPG as codec with compression level 30.

A test concerning video FPS was performed on a Linux computer using Axis Live View as video player, where there is a possibility to control the maximum amount of FPS streamed to the client computer. The computer hardware is noted as PC4 in Table 4.1. Tests were carried out comparing 15 FPS to 25 FPS at different resolutions with the video delay test described in Appendix B. Results from the FPS test are presented in Figure 4.3. The result that the lower FPS setting gives longer stream-

[1] http://www.videolan.org/vlc

PC #	OS	CPU	RAM	Graphics
1	Windows XP SP3	Intel i7 @ 2.93 GHz	3.5 GB	Geforce 210
2	Windows 7	Intel i7 @ 3.40 GHz	8 GB	Geforce GT430
3	Windows Vista	Intel C2D @ 2.4 GHz	4 GB	GMA4500MHD
4	Debian	Intel C2D @ 3.00GHz	4 GB	GeForce 8400

VMS	PC #
VMS1	1
VMS2	2
AXIS Camera Station	3
Axis Live View	3

Table 4.1 The upper table shows specifications of the computers used for testing video delays. Note that PC 3 is a laptop with an integrated graphics circuit. The lower table shows which VMS was tested with which hardware.

ing delay is assumed to being a test technical occurrence, it is not believed to have higher delays with less frames per second. The average result grows larger because of the fact of longer delay between the frames in the image loop. The minimum and maximum values does, however, stay the same at the different FPS settings. The conclusion from this test is that a high image frequency makes the tests easier to implement which is why the higher frame rate of 25 FPS was chosen for the remaining tests.

Tests were constructed to investigate how different parameters affect the video delay. Parameters altered in the tests were: different kinds of VMS, video at different resolutions, and two different video codecs. Because of license issues the different kinds of VMS were tested on separate, non-identical hardware which gives the test an additional dimension of hardware and OS differences. The resolutions tested were:

- 1920x1080 pixels (Full HD) is the largest resolution available from the AXIS Q6035.
- 800x450 pixels is a medium resolution which still gives a good view from the camera in a steering point of view.
- 320x180 pixels is the smallest resolution available from the AXIS Q6035 camera which enables us to tell what difference the amount of data makes to the video delay compared to the other two, larger resolutions.

The video codecs used were H.264 and MJPG being the two video codecs the AXIS Q6035 supports. The following four VMS clients were tested:

- AXIS Camera Station is a Windows application for systems with up to 100 cameras.
- Axis Live View is not an actual VMS in the sense of recording video and managing several cameras but a well developed web interface accessible on most platforms and web browsers. The joystick compatibility does, however, require a Windows PC and the browser Internet Explorer with an Axis specific plugin.
- Additionally two major third party VMS clients were tested. These will be referred to as VMS1 and VMS2.

Computer hardware and operating system used with the different VMS's are presented in Table 4.1.

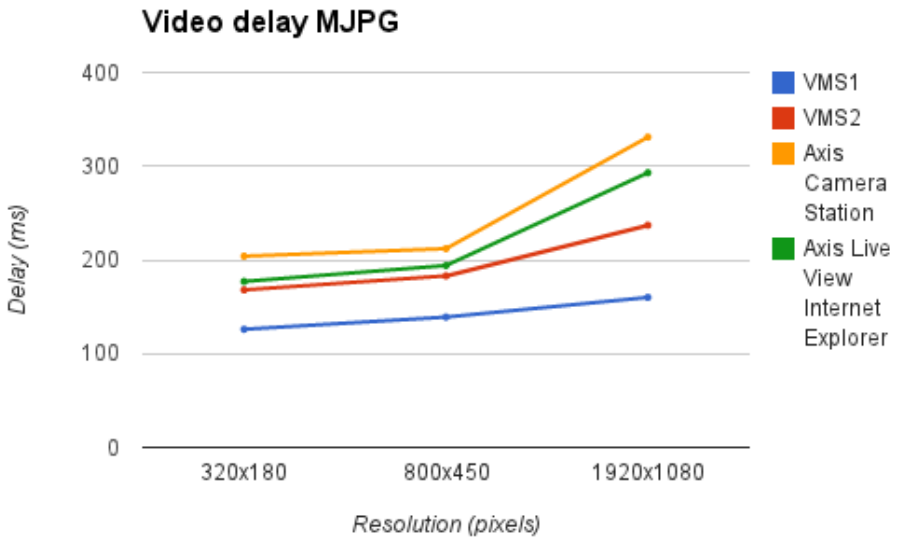


Figure 4.4 Video delay tests using video codec MJPG at a set of resolutions tested with a number of different VMS programs. Tests were carried out according to the procedure described in Appendix B.

Test results from the MJPG video delay tests are presented in Figure 4.4. The time difference from the lowest measurement to the highest measurement with the different VMS clients playing the MJPG video were:

- 78 ms (320x180 pixels)
- 73 ms (800x450 pixels)
- 171 ms (1920x1080 pixels)

The fact that the high resolution of 1920x1080 pixels gives a larger difference in delay is consistent throughout all video delay tests indicating that large video pictures (resolution wise) are more demanding to handle than smaller ones.

In the H.264 test results shown in Figure 4.5 the span between max and min measurements were:

- 133 ms (320x180 pixels)
- 133 ms (800x450 pixels)
- 455 ms (1920x1080 pixels)

The differences compared to the MJPG tests are larger and showing the same behaviour at the maximum resolution with a significant increase in delay compared to the lower resolutions. The overall higher delays indicates the H.264 video codec to be more demanding compared to MJPG in the sense of time delays when presenting video.

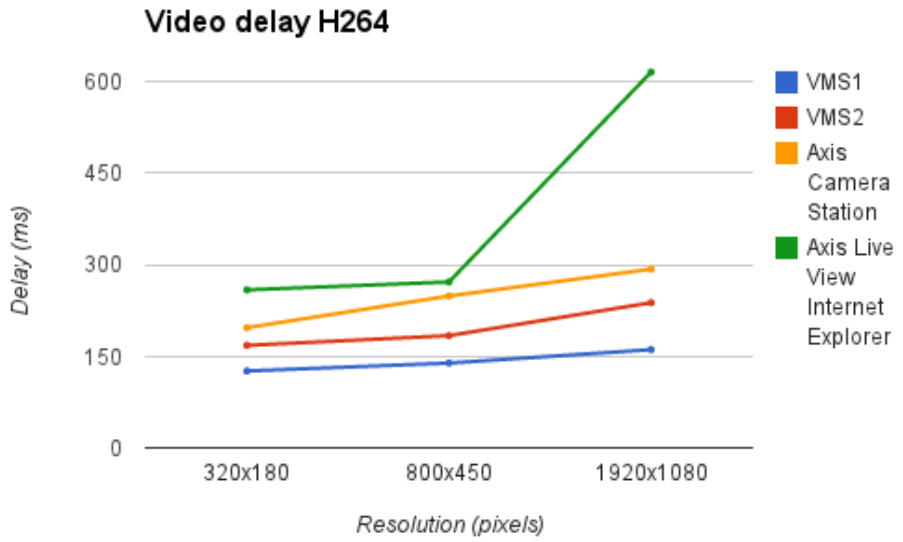


Figure 4.5 Video delay tests using video codec H.264 at a set of resolutions tested with a number of different VMS programs. Tests were carried out according to the procedure described in Appendix B.

4.2 Total delay

The intention of developing a total delay test was to give a full picture of how system changes makes a difference to the end user. The total delay test is a natural extension of the Video delay test in Section 4.1 giving the opportunity to use the test results together. To send movement commands to the camera there is a possibility to chose between using a mouse or a joystick. A joystick was used all through the tests because of it being easier to identify joystick movement in a recorded video in contrast to a mouse click. The final test procedure is described in Appendix A.

Factors investigated in the total delay test are:

- Resolutions: 1980x1080 pixels and 800x450 pixels both at 30 FPS and 30 compression.
- Video codecs: MJPG and H.264
- VMS: AXIS Camera Station, Axis Live View and two third part clients VMS1, VMS2 described in detail in Table 4.1. A custom build VMS using VLC and a simple VAPIX client for steering commands on a Linux platform was also used in the MJPG tests.
- Hardware and OS according to Table 4.1

Figure 4.6 shows test results of a total delay test at a high resolution using video codec H.264. The different VMS clients gives a large difference of total delay in this test. That fact of Axis Live View giving the longest delays did not come as a surprise looking at the video delay results in Figure 4.5. It seems like the hardware running Axis Live View is not able to handle both running the video codec and listening to the joystick in a real time manner. AXIS Camera Station was, however, tested on the same machine indicating the benefits of running a dedicated application compared to a web browser plug-in. Looking at the results from VMS2 there is no guarantee

that a natively running program and relatively good hardware is enough for a stable behaviour.

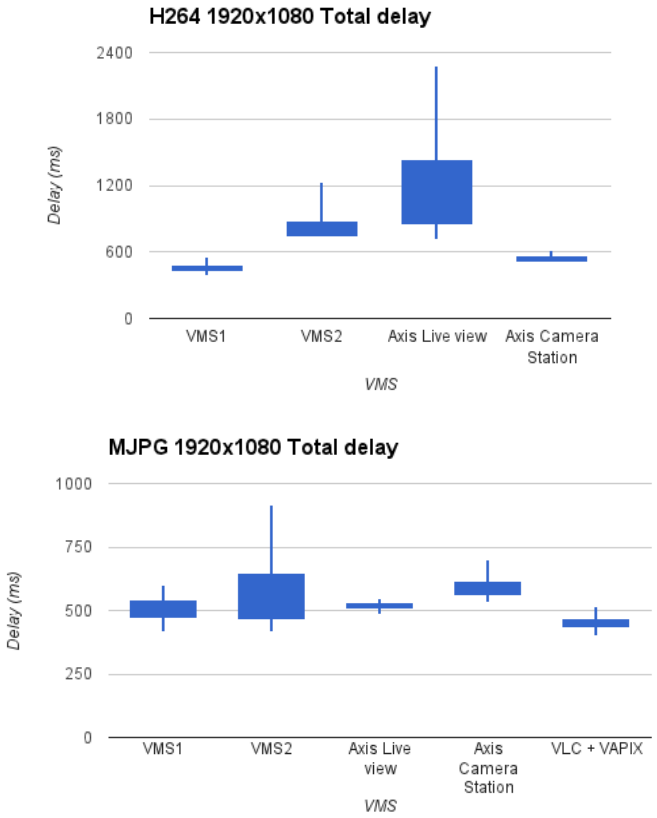


Figure 4.6 Total delay tests using video codec MJPG and H.264 at a resolution of 1920x1080 with different VMS clients, hardware and operating systems. Each thin vertical line represents the range of all test samples from the same category. The thick block is centered to the mean of the test samples with the height of an estimate of the standard deviation of the population indicating the spread of the measurements. The tests were conducted according to the description in Appendix A.

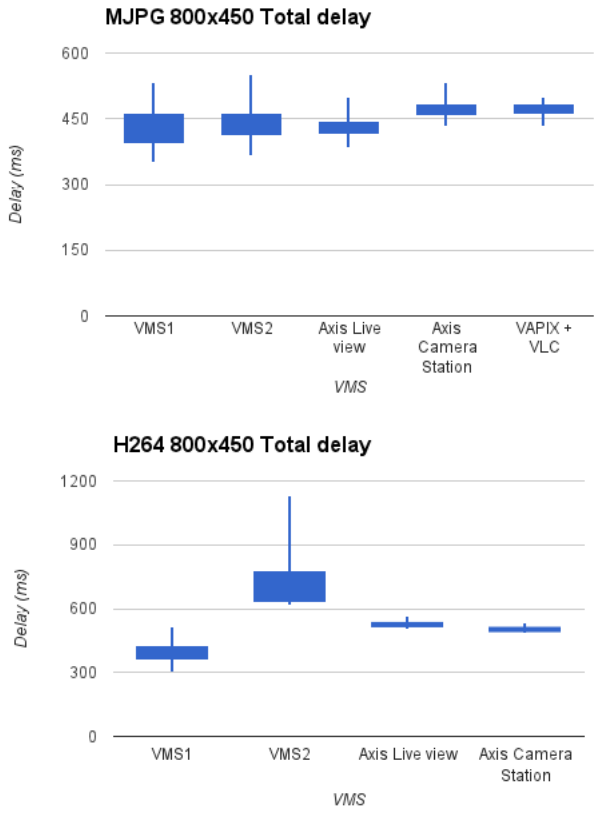


Figure 4.7 Total delay tests using video codec MJPG and H.264 at a resolution of 800x450 with different VMS clients, hardware and operating systems. Each thin vertical line represents the range of all test samples from the same category. The thick block is centered to the mean of the test samples with the height of an estimate of the standard deviation of the population indicating the spread of the measurements. The tests were conducted according to the description in Appendix A.

4.3 Network delay

The PTZ move control is carried out over the same IP network as the video stream. In this thesis the VAPIX API has been used in the tests and all requests are collected at the camera web server. Many packets are being sent when an HTTP request is performed since it is a protocol using TCP as transport layer protocol.

From the implementation of TCP the setup phase has to be established for every request sent. That is

- One SYN packet from client to camera.
- One SYN ACK packet from camera to client.
- One message packet from client to camera.

A command then has to travel through the network three times to transport a move request to the camera. Introduction of QoS on the network traffic is applied at the network layer and the most important attributes of the transmission is the delay and reliability of a transfer. From Section 3.3 and Section 3.4 the DiffServ EF PHB is the forwarding policy aimed to give the shortest delays. From Table 3.1 on page 14 the DSCP value 48 corresponds to the desired forwarding policy.

A program to test the network delay to the camera was developed. The program was designed to send a HTTP request to the camera requesting a non existing page on the camera, then the camera responded with an "HTTP 404 File not found"-error. The Round Trip Time (RTT) is being measured where the time is stamped before delivery and after the answer. The program has options to choose the number of times to run, a specified ToS value in the IP header and to use a delay between each request sent. When all requests are done the program computes average RTT, minimum RTT, maximum RTT and standard deviation RTT.

ToS	Samples	Mean (ms)	Min (ms)	Max (ms)	Stddev (ms)
0	10000	1.6	1.0	19.1	1.1
0	10000	1.4	0.9	9.2	0.5
0	10000	1.7	1.0	14.3	0.9
184	10000	1.7	1.2	9.5	0.9
184	10000	1.8	1.2	30.2	1.3
184	10000	1.6	1.2	229.0	2.5

Table 4.2 RTT for a HTTP request for a non existing page (/errorpage) on an AXIS Q6035 camera. The network was a local network within Axis.

The test was run with an AXIS Q6035 camera. One test sequence was configured to send 10000 requests with 1 ms delay between the requests. The test sequence was executed six times for two different network setups: three with Type of Service (ToS) = 0 and three with ToS = 184. The two networks were a local network within Axis and a city scale Network (Lund). The results from the tests can be seen in Table 4.2 and Table 4.3. Histograms for the RTT can be seen in Figure 4.8 and Figure 4.9. The histograms cover all three test sequences with ToS = 0 to the left and all three test sequences with ToS = 184 to the right.

On the local net the priority packets did not get short RTT. On the city scale network the priority requests did, however, get shorter RTT in the average case. The histograms cover the RTT range for the majority of the requests. For the priority

ToS	Samples	Mean (ms)	Min (ms)	Max (ms)	Stddev (ms)
0	10000	26.7	14.3	634.7	23.7
0	10000	27.8	14.2	8815.6	103.2
0	10000	27.3	14.1	770.4	28.8
184	10000	24.3	13.8	735.7	22.5
184	10000	24.3	13.8	737.0	22.1
184	10000	24.5	13.8	1254.6	26.5

Table 4.3 RTT for a HTTP request for a non existing page (/errorpage) on an AXIS Q6035 camera. The network was a city network (Lund). The camera was the same as in the test from Table 4.2.

packets it can be seen on the shapes of the histograms, where the priority shape is tilted towards the minimum time.

The packets sent over the network were analysed with the Wireshark Network Protocol Analyser[2] and when using ToS = 184 the answering SYN ACK packet from the web server (camera) still used ToS = 0. To prioritise packages sent from the camera, the web server has to be modified to use a priority flag in the ToS field. The modification would preferably be to copy the value in the ToS field in the incoming request's IP package.

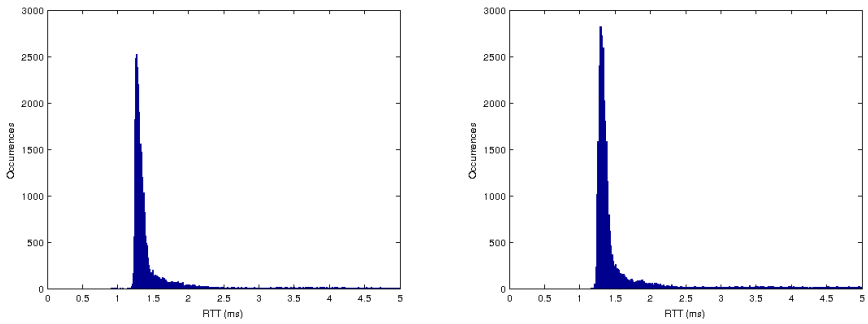


Figure 4.8 Histogram for RTT on a local network. ToS = 0 to the left and ToS = 184 to the right.

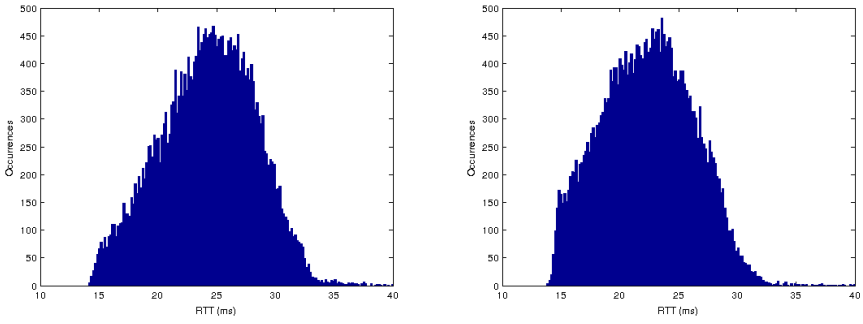


Figure 4.9 Histogram for RTT on a city network. ToS = 0 to the left and ToS = 184 to the right.

[2]http://www.wireshark.org

4.4 Separate joystick client with new protocol

In an attempt to lower the steering delay a new approach for the joystick communication was evaluated. The key idea with the new approach was to open up a socket connection in the camera dedicated for communication with a joystick.

The first step in the design of the new approach was to select the type for the communication. Two approaches came up as candidates for the implementation, namely event based communication with TCP and continuous communication with UDP. The major reason for choosing UDP over TCP is the risk for delay derived from the network communication. From the test with QoS on the network communication the network communication delay was discovered to be in the magnitude of 1 ms on a local network, which is not a measurable unit with the PTZ delay measurement technique available.

Choosing continuous communication implies extra work in the camera since it has to process each new incoming command. That is a drawback since the camera has limited hardware resources. The final design was based on events with TCP, since it implies a simpler implementation in the camera software and the used transport protocol would not result in a decrease of the total delay of the system.

The second step in the design was to place the socket connection in the camera software. One of the early strategies in the work was to place a PTZ dedicated socket in the camera software as close as possible to the drivers, but in a place where most camera firmware look alike. The PTZ driver process was considered a suitable place for the new socket connection. When a command has reached the PTZ driver process several steps has passed in the software and it is close to the Kernel driver for the camera motors. The placement is illustrated in the system overview in Figure 4.10.

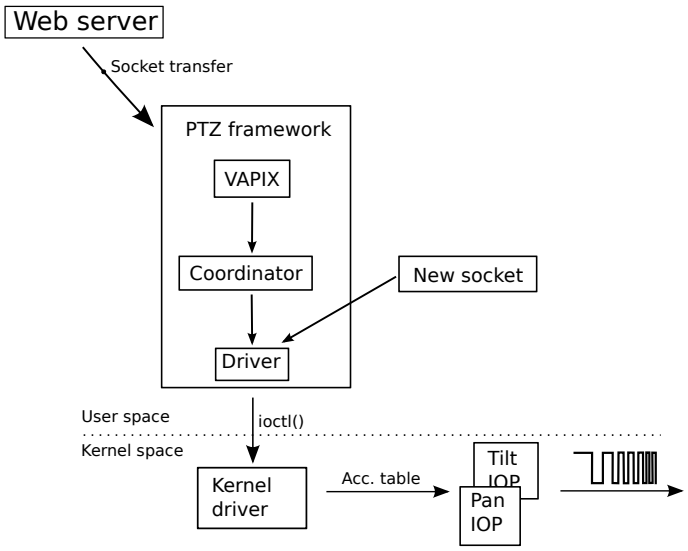


Figure 4.10 The place for the new socket connection pointed out in the system overview.

The new socket connection uses the existing support for joystick steering control in the PTZ Driver. The only information needed to control the movement is two integers: one for pan control and one for tilt control. From the two integers the camera software can determine the speed and direction for the motors. When a command arrives at the new socket connection the parameters are transformed into an internal representation and passed into the PTZ Driver as if it was a VAPIX command.

In addition to the new socket connection in the camera a client application supporting the AXIS T8311 JOYSTICK was developed. The application retrieves input

from the joystick and sends it to camera every time a new event is triggered. An event is triggered every time the joystick is moved. The application supports communication with either the VAPIX API or the new socket connection.

Test execution

The full system test described in Appendix A was used to evaluate the new protocol. Equipment used for the test::

- Camera: AXIS Q6032.
- Joystick: AXIS T8311 JOYSTICK.
- VMS: Axis Live view + Linux joystick client.
- Resolutions: 720x480 pixels and 352x240 pixels.
- OS+HW: Debian, Intel C2D @ 3.00 GHz, 4 GB RAM and GeForce 8400 graphics card.

Results from the protocol change tests are presented in Figure 4.11. The intention of implementing a new protocol closer to the drivers in the firmware was to lower the total steering delay, according to the test results there was no noticeable improvement with the test method used.

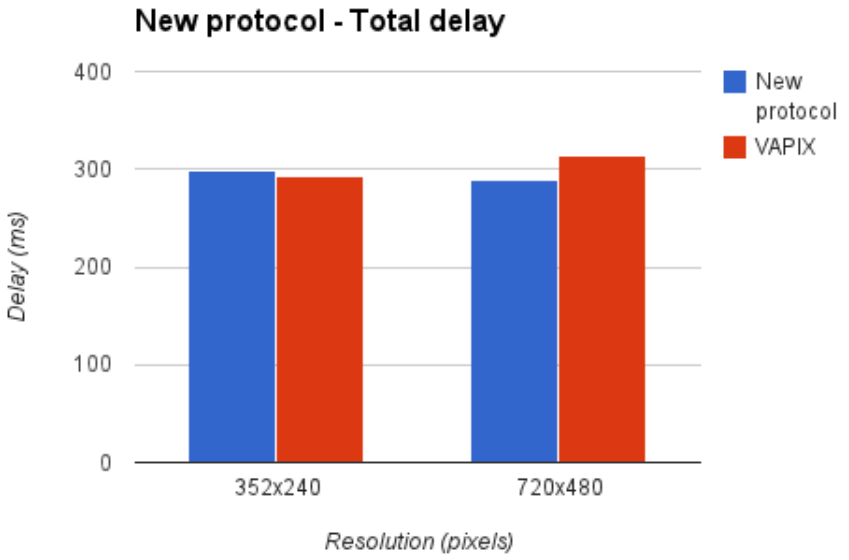


Figure 4.11 Total time of the original camera movement command protocol (VAPIX) compared to a custom made protocol. The test procedure is described in Appendix A. An AXIS Q6032 camera was used with Axis Live View in this test.

4.5 Motor delay

To get a separation between the software and the hardware delays inside the camera a video recording test was performed. A Light-Emitting Diode (LED) is placed on the circuit board and it is visible for external observation. The LED, called statusled, can be controlled using the VAPIX API. The idea behind the test is to light the statusled deep inside the camera firmware, to get some knowledge how long it takes after the PTZ framework has been executed until it can be captured on video that the camera is moving.

The `ioctl()` system call in Linux is used to manipulate filedescriptors that does not fit into the Unix IO model, and it is used to change the statusled in the tested camera. What was actually modified in the camera is a predefined call to `ioctl()` with instructions to light the LED. It is reasonable to suspect this extra work to have a small impact on the firmware performance.

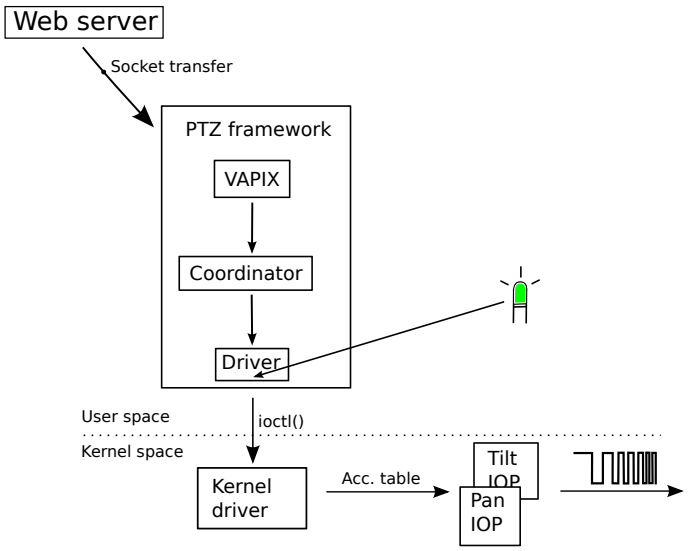


Figure 4.12 The placement for the modification in the motor delay test code illustrated. The `ioctl()` call to light the LED was inserted at the end of the Driver process in the PTZ framework.

Figure 4.12 is an illustration of the placement for the modification, where it can be seen that almost all software in user space has been executed and the only thing being done after the LED has been lit is a call to a device driver in the Linux kernel.

When the firmware got this modification a test setup similar to the video delay and total delay was designed. In the test two cameras were used, where the recording camera and tested camera are connected to two different computers. From the recording camera an MJPG stream was collected. The view from the recording camera is illustrated in Figure 4.13. In the experiment the GNU wget[3] download utility was used to retrieve the video stream.

An assumption was made that the video stream from the recording camera holds a constant frame rate. That gives that the time between two frames is

$$frame\ delay = \frac{1}{frame\ rate} = \frac{1}{60} = 16.7\ \texttt{ms}.$$

The delay from the test is calculated as the number of frames (`n`) multiplied with the frame delay. The camera used for recording was capable of filming with 60 FPS.

[3] http://www.gnu.org/software/wget

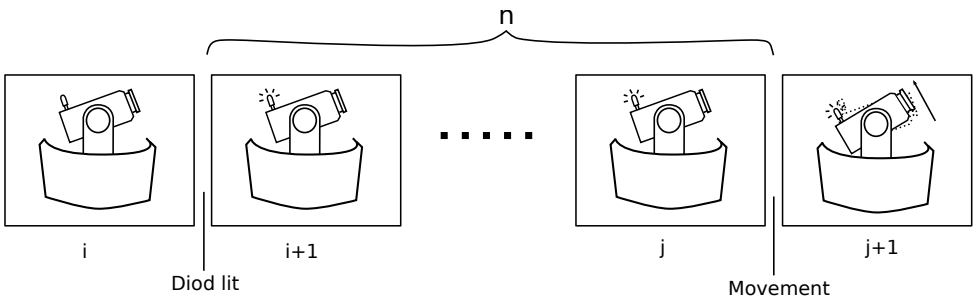


Figure 4.13 An illustration of received video stream, frame by frame, in the motor delay LED test. The two events as they occur are marked between frame (i) and (i + 1) and frame (j) and (j + 1).

The test was launched on an AXIS Q6035 repeated 20 times for both the pan and tilt motor. The result, which can be read in Table 4.4, shows that there is a delay from the last step in the PTZ framework to the actual movement of the camera.

Motor	Samples	Mean (ms)	Min (ms)	Max (ms)	Stddev (ms)
Pan	20	133	117	150	9
Tilt	20	148	117	167	10

Table 4.4 Delays from the motor delay test on an AXIS Q6035.

Second, a couple of a tests were conducted with various load on the camera since delivering different kinds of streams to a number of connected clients can put a heavy load on the camera. The intention of these tests was not to collect more samples but to test if the load have a significant impact on the delay in the firmware.

The results from the the load tests can be seen in Table 4.5.

Setup	Motor	Samples	Mean (ms)	Min (ms)	Max (ms)	Stddev (ms)
A	Pan	10	145	133	150	8
	Tilt	10	152	150	167	5
B	Pan	10	133	133	133	0
	Tilt	10	158	150	167	9
C	Pan	10	137	117	150	10
	Tilt	10	143	133	150	8

Table 4.5 Motor delays with three different setups for load on the camera. The three setups are: A. One 1920x1080 MJPG stream with 30 % compression, B. Six 1920x1080 MJPG streams with 30 % compression, and C. One 1920x1080, one 1280x720 and one 800x450 MJPG stream with 30 % compression each.

The average time between the LED light and motor movement does not grow due to load on the camera. The variation of the delay can be related to the uncertainty in the test setup, coming from the fact that a recording camera are being used to measure the time intervals.

4.6 Firmware delay

The motor delay test (Section 4.5) showed that it takes considerable time from the PTZ framework to actual movement of the motors. To get better performance analysis of the firmware a second test was constructed. The purpose of the new test was to investigate the delay in the different parts in the firmware. The target for measurement was:

- PTZ framework.
- PTZ framework to Kernel Driver Module.
- Kernel Driver Module.

Basic concepts

The Kernel Driver Module does not send signals directly to the motors, but through processors dedicated for motor control. The motor control processors generate pulses that control the motors. The generated pulses form patterns determining acceleration and direction for the motors.

The aim for the test is to measure the delay in the firmware to the start of the pulses. The generated pulses are logical signals that can be observed with an instrument for observation of electrical signals, like an oscilloscope or a logic analyser. Oscilloscopes and logic analysers are, on the other hand, not made for measuring execution time in software components.

Measuring execution time in software can be done directly in the software by collecting timestamps at predefined points and later on measure the time differences between them.

A measurement being a combination of timestamping and observation of the pulse pattern was chosen as the solution with the intention to measure both software and the pulse with the same test. The basic concept for the test is to transform software timestamps to logical signals measurable with a logic analyser.

Logic analyser

A logic analyser is an instrument that measures signals from a digital system, determining if they are high or low (1 or 0). Many signals can be observed (or recorded) at the same time. In a recording with several signals all shares the same clock and it is possible to compare signals with each other and observe a chain of events.

For this test the Saleae Logic16[4] analyser was used, which is a small portable analyser capable of observing 16 different signals at the same time. The device is connected to logical pins on a circuit and samples each pin, determining if the value is high or low (1 or 0). The device can sample its input channels with varying sample rates depending on number of active channels that is being used. The highest possible sample rate are only available when a small number of signals are used.

The device is controlled through a special software program installed on a PC. The device is connected to the PC with a USB cable and is then solely managed with the special software. The software's user interface can be seen in Figure 4.14, with a sample output from a measurement.

The measurements are carried out by recording the input channels for a time period, meaning that the analyser has to be started recording before the target logical signals appears.

[4] http://www.saleae.com/logic16

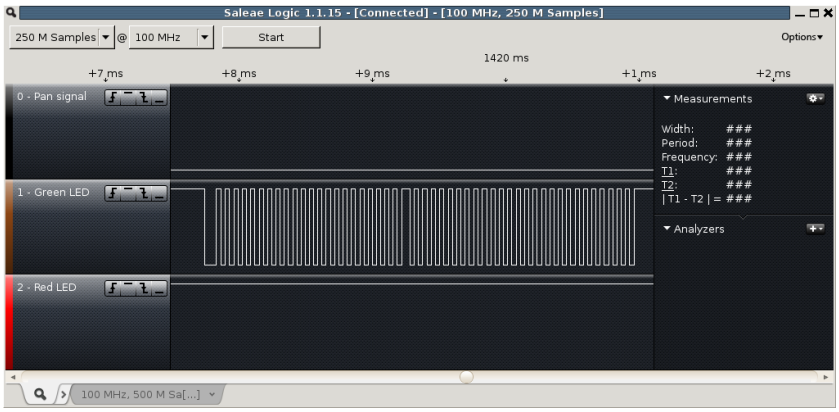


Figure 4.14 User interface for the Saleae Logic16 logic analyser. The analyzes is controlled through the software as well as its output is observed with the software.

Circuit board modification

To connect the logic analyser to the motor pulse signal one of the circuit boards used in the AXIS Q6032 had to be modified. The logic analyser is connected to hardware pins with debug cables or special clips. Neither the cables, nor the clips, could be connected to the signals that needed to be measured.

The motor pulse is generated on one of the pins from the two motor control processors. The power pulse pin on the pan IO processor was chosen to be provided with a debug pin, suitable for pairing with the logic analysers input channels.

To measure time differences from the software and the motor pulse more debug pins had to be equipped to the circuit board. In the motor delay test (Section 4.5) a LED was used, for which code to lit and unlit already was written. The LED has two channels: one red and one green. The two channels was provided with a debug pin each.

The circuit board was soldered with additional debug pins on:

- Pan motor power pin.
- Green and red LED channel.

Software modifications

To transform software timestamps into hardware signals visible on the newly added debug pins on the circuit board code had to be written. The signals that needed to be triggered in the firmware were power signals to the green and red LED channels.

Some parts of the firmware is executed in the user space segment and some part in the kernel space segment. The procedure to lit the LED from user space is a call to the `ioctl()` system call, while in a kernel module the LED pin is set to high or low. Template code was developed for both user space applications and kernel modules.

The purpose for the test is to measure the delay in the camera firmware but in the test setup the firmware need modifications. The new code may introduce different behavior and in worst case longer delays. A delay test for the template code was performed consisting of a program loop turning the LED on and off repeatedly. The output from the test can be seen in Figure 4.14. In the test the LED is turned on and directly turned off resulting in a light pulse (not noticeable to a human) measured with the logic analyser.

In the test from Figure 4.14 the first pulse is 90 μs and the rest vary from 31 μs to 33 μs. In the test the first pulse was always longer then the rest. The duration of the light pulse did not exceed 100 μs in the test.

The modification in the firmware does not impact the total firmware delay because the extra time to light the LED is very short compared to the total delay, which is approximately 50 ms to 100 ms.

Test execution

With the test setup described in this section the firmware needed to be modified. A test setup was constructed to measure:

- PTZ framework execution time.
- Time from PTZ framework to Kernel Driver.
- Kernel Driver to motor pulse.

The test procedure is described in Appendix C.

When the modified circuit board and the logic analyser were tested the firmware delay seemed to be different if commands were sent with the joystick application developed in this thesis or with a VAPIX request through a web browser (a HTTP request). It was decided to test the firmware delay with two different approaches: sending one command and sending two commands successively.

The test was performed with four different camera movement speeds: 1, 10, 50 and 100 (for the continuous movement command in the VAPIX API). When two commands were sent successive the first command was always of speed 1. For speeds 50 and 100 the test was also performed at the same time as video stream was received from the camera.

	PTZ framework	PTZ to kernel	kernel to motor pulse	Total
Speed	mean (stddev) ms (ms)	mean (stddev) ms (ms)	mean (stddev) ms (ms)	mean (stddev) ms (ms)
1	6.31 (0.57)	54.40 (2.98)	35.56 (0.86)	96.28 (3.40)
10	6.55 (0.59)	55.57 (2.22)	35.24 (0.41)	97.37 (2.86)
50	6.82 (0.85)	55.26 (2.01)	35.51 (0.30)	97.60 (2.42)
100	6.23 (0.27)	52.76 (2.93)	35.28 (0.44)	94.27 (2.88)

Table 4.6 Firmware delay test results when sending a single move command.

	PTZ framework	PTZ to kernel	kernel to motor pulse	Total
Speed	mean (stddev) ms (ms)	mean (stddev) ms (ms)	mean (stddev) ms (ms)	mean (stddev) ms (ms)
50	8.94 (3.68)	55.34 (3.60)	35.39 (0.44)	99.67 (5.84)
100	7.93 (2.51)	55.46 (2.21)	35.42 (0.62)	98.81 (4.02)

Table 4.7 Firmware delay test results when sending a single move command at the same time as a video stream was retrieved. The stream setting was MJPG with 720x480 resolution and 30 % compression.

Table 4.6 and Table 4.7 contains the results from the tests with one command. A one stream load on the camera has small impact on the total delay since it only raises the total time with 2 ms to 3 ms.

Speed	PTZ framework mean (stddev) ms (ms)	PTZ to kernel mean (stddev) ms (ms)	kernel to motor pulse mean (stddev) ms (ms)	Total mean (stddev) ms (ms)
1	6.90 (1.14)	38.97 (8.49)	32.40 (3.55)	78.27 (9.40)
10	6.21 (0.40)	34.24 (3.01)	31.54 (2.95)	72.00 (5.11)
50	6.54 (0.79)	33.97 (3.18)	33.48 (2.50)	74.00 (4.95)
100	6.38 (0.55)	36.70 (6.30)	32.58 (2.38)	75.66 (7.83)

Table 4.8 Firmware delay test results when sending two move commands.

Speed	PTZ framework mean (stddev) ms (ms)	PTZ to kernel mean (stddev) ms (ms)	kernel to motor pulse mean (stddev) ms (ms)	Total mean (stddev) ms (ms)
50	7.81 (3.01)	42.36 (2.18)	32.54 (4.83)	83.92 (6.26)
100	7.60 (3.18)	45.60 (6.85)	33.44 (4.32)	87.07 (7.40)

Table 4.9 Firmware delay test results when sending two move command at the same time as a video stream was retrieved. The stream setting was MJPG with 720x480 resolution and 30 % compression.

The results from the test with two commands successive can be seen in Table 4.8 and Table 4.9. With two commands the load of one stream has a larger impact on the total delay (8 ms to 10 ms).

The result showed that the delay is basically the same with and without a one stream load on the camera. When sending two commands the delay was lowered by approximately 20 ms. The generated pulse pattern looks different for the various speeds and it was observed that the generated pulse pattern corresponded to the latter command in the case when two commands were sent.

4.7 Test for improvements in firmware

From the firmware delay test (Section 4.6) it was suspected that the delay from the PTZ framework to the kernel module could be shortened. It was also suspected that the time in the kernel module could be shortened. With guidance and help from Axis employees the firmware was modified with intention to reduce the delay. Two improvements were made to the firmware.

The time from PTZ framework to the Kernel driver was related to polling and a state machine. When the motors are started from stationary state, the voltage has to be raised to power the motors. There is a sleep time inserted to wait for the voltage value to get stable. In this test the sleep function was removed and the voltage value were set to high at all times.

The extended motor delay test, motor delay test and the full system test were executed with the modified code. As can be seen from the test results the firmware delay was reduced to less than 15 ms.

Firmware delay

	PTZ framework	PTZ to kernel	kernel to motor pulse	Total
Speed	mean (stddev) ms (ms)	mean (stddev) ms (ms)	mean (stddev) ms (ms)	mean (stddev) ms (ms)
1	6.44 (0.57)	0.70 (0.04)	5.65 (0.0006)	12.78 (0.57)
10	6.32 (0.60)	0.70 (0.03)	5.65 (0.0005)	12.67 (0.60)
50	6.49 (0.46)	0.70 (0.04)	5.69 (0.0007)	12.87 (0.45)
100	6.51 (0.75)	0.70 (0.03)	5.78 (0.0007)	12.99 (0.75)

Table 4.10 Firmware delay with improved firmware.

	PTZ framework	PTZ to kernel	kernel to motor pulse	Total
Speed	mean (stddev) ms (ms)	mean (stddev) ms (ms)	mean (stddev) ms (ms)	mean (stddev) ms (ms)
50	7.02 (0.59)	0.69 (0.008)	5.69 (0.0006)	13.41 (0.59)
100	7.25 (2.26)	0.81 (0.36)	5.78 (0.001)	13.84 (2.61)

Table 4.11 Firmware delay with improved firmware at the same time as a video stream was retrieved.

The decrease of the delay in this test (see Table 4.10 and Table 4.11) compared to the test without improvements (see Table 4.6 to Table 4.9 on pages 32 to 33) is remarkable. The results can be compared to either the original test with one command or the original test with two commands. The reason for not running the test with two commands with the modified code is that the total time for the improved firmware is shorter than the time interval between two executions in the PTZ framework, the second command would start its execution after the pulse had been generated for the first command.

Comparing the results with the case with one command from the original test gives a delay decrease of approximately 86 % (83 ms) and for the case with two commands approximately 82 % (61 ms). No matter which test to compare with, the results showed a major decrease of the delay.

In a real case with joystick usage it is not clear what the situation is (one command or two commands). It is thus reasonable to guess that the case with two commands is likely. The joystick client developed in this thesis caused the case with two commands.

Motor delay

Motor	Samples	Mean (ms)	Min (ms)	Max (ms)	Stddev (ms)
Pan	10	57	50	67	8
Tilt	10	55	50	67	8

Table 4.12 Results from the motor delay test with improved firmware.

From the result of the extended motor delay test a decrease in the delay was expected for this test which also was the case (see Table 4.12 and Table 4.4 on page 29).

Both the pan and tilt motors work with the same mechanics and therefore the results from the motor delay test are expected to give the same results for both the pan and tilt motor. In the test with improved firmware that was the case with only 2 ms difference for the mean values. The motor delay test with original firmware was run on an AXIS Q6035 and the motor delay test with the improved firmware was run on an AXIS Q6032. The mechanics for the camera movement are the same for the two camera models but the possibility for different delays are of course a consideration to care about. The motor delay test with the original firmware had greater spread in the result.

The delay with the improved firmware was 76 ms respective 93 ms shorter which can be compared to the 83 ms decrease in the extended motor delay test.

Total delay

The total delay test, described in Appendix A, was used to test with the new firmware and the AXIS Q6032 camera. From the previous test with the modified firmware a shorter total delay was expected.

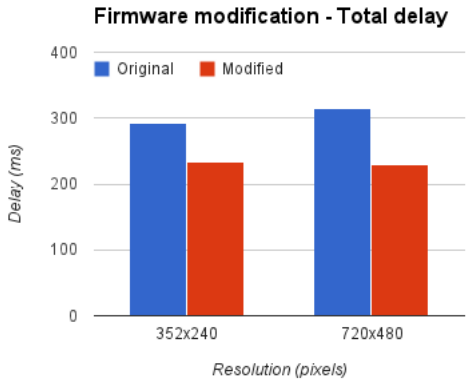


Figure 4.15 Tests results comparing firmware modifications. The test was executed on an AXIS Q6032 according to the procedure described in Appendix A.

In Figure 4.15 it can be seen that the total delay is about 70 ms shorter with the modified firmware.

5. Test results

This chapter summarises the test results from Chapter 4. By collecting all results and putting them in relation to each other a better overview of the system delays and improvements of the system is obtained.

5.1 Measurements of the current system

Figure 5.1 shows a breakdown of measured delay times from joystick movement to camera movement. The separate parts of the system have been measured with different techniques with varying reliability and precision.

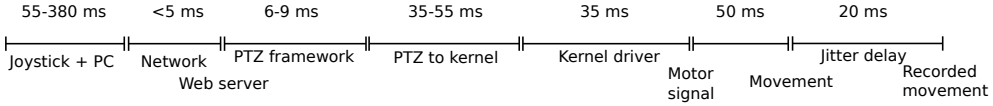


Figure 5.1 An overview picture of delay times from joystick movement to the left to camera recorded movement on the right. Test results from the measurements of the AXIS Q6035 and AXIS Q6032.

Joystick + PC

The "Joystick + PC" part includes the time for the client computer to retrieve and process the joystick movement and to send it to the camera. Factors influencing the client contribution of delay has been identified as VMS client, video settings, client hardware and OS.

This part of the delay was not measured directly but through looking at differences in other tests. By subtracting all other measured sub parts of the system from the measured total delay the time left is the "Joystick + PC"-time. Not measuring directly gives the "PC + Joystick"-time the uncertainties from all the other tests accumulated.

Network

The network delay on a local network is approximately 1 ms. The RTT test from Section 4.3 resulted in an average RTT of 1.4 ms for a local network.

The RTT was 27 ms in a test with a camera on Axis office network from a PC placed at another site in the same city as the office.

PTZ framework

The execution time in the PTZ framework is in average 6 - 7 ms if there is no load on the camera and it raises to 8 - 9 ms with one stream.

PTZ framework to Kernel driver

The time is dependent of the communication from the connected client. If two move commands are sent close to each other the time gap is 35 ms. With one command sent the time is 55 ms.

With a modified firmware this delay was reduced to 0.7 ms.

Kernel driver

The Kernel driver execution time is a steady 35 ms in the case of one single command sent to the camera. When two commands are sent in a row the duration is 2 - 3 ms shorter in average, but with larger variations.

With a modification in the Kernel driver the delay was reduced to 5.7 ms.

Motor acceleration

The motor acceleration was not measured directly, but through the difference between two other tests. The tests are described in Section 4.7 on page 34. The duration is estimated to 50 ms.

Jitter delay

The jitter delay depends on the camera frame rate. The jitter delay is in average half of the time between two frames. The jitter delay is described in Figure 4.1 on page 16.

5.2 Video from camera to monitor

Figure 5.2 shows the range of video delay with different setups of hardware and software used. Comparing video codecs the lower delay times are identical where the upper delay times varies. H.264 shows longer max delay times compared to MJPG confirming that it is a more computationally heavy video codec giving the system harder requirements on client hardware to function well.

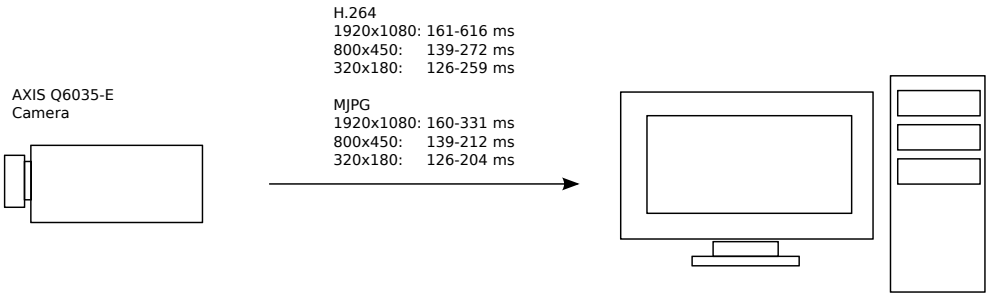


Figure 5.2 A summation of all video delay of the AXIS Q6035 with a variety of different VMS clients, OS's, and hardware. The two lists headed H.264 and MJPG shows the range of average video delay times for the listed resolutions using different setups.

6. Discussion

6.1 Mapping of the current system

Gaining knowledge of the individual time intervals of the system gives rigid grounds to make a correct analysis of where to improve the camera steering. Regular test measurements of the separate intervals concerning camera steering would make it easier to identify and pinpoint possible problems in the product. There are still complications to maintain the knowledge about the different time intervals with tests being time consuming and requiring manual labour. Some of the tests also require modification to the camera hardware.

With the results from the measurements of the firmware one delay from PTZ framework to kernel driver, a delay of up to 55 ms was completely removed. Another questionable delay is kernel driver which has a relation to handling of the motors. According to discussions with Axis employees a wait time is included to ensure correct voltage values to power the motors. The wait time seems to have a connection to power saving functionality that in an experiment was removed successfully with the consequence of feeding the motors with full power at all times. The outcome from that experiment was an improvement of delay time in the kernel driver up to 30 ms.

6.2 Changing protocol and skipping steps within the camera firmware

Theories concerning process changes taking time within the camera firmware proved out to not make any significant changes of the total steering delay. Since the new socket implementation was placed in the end of the PTZ framework the most time consuming part of the framework was still included in the new solution not saving any time in the delay.

Comparing the move command communication of using HTTP to a custom-made protocol connected directly to the PTZ driver in the camera seems to have made small or no difference at all. A risk with opening a new socket is security not passing the error checking in the previous processes in the PTZ framework.

6.3 Prioritization of network traffic

Measurements on small networks shows that a request to the camera web server takes less than 2 ms, this fact proves the network factor to be a minor part of the steering delays. QoS service settings did not change the network delay on small networks. From measurements on large networks (city scale) a request to the camera web server was measured to approximately 27 ms, a result later reduced by 3 ms through the use of QoS. Only the packets sent to the camera will change priority and not the responding packets from the web server. To achieve prioritized traffic in both directions changes to the web server would have to be done.

6.4 Physical relocation of the joystick

Variations from tests using different VMS clients to steer a joystick were measured from 75 ms to 400 ms, a larger variation compared to all other parts measured when steering a camera. In a custom made client using VAPIX this delay was measured to less than 55 ms indicating that running the joystick on a separate client from the PC running the VMS could enable a stable and quick joystick connection regardless of VMS client, resolution and video codec.

6.5 Future work

Testing over several camera models

Comparing the two camera models AXIS Q6035 and AXIS Q6032 there are differences in delays of video streaming. An interesting project could be to try out some basic tests with all Axis PTZ camera models and perhaps construct guidelines for a watermark level of quality guaranteed by all Axis cameras.

Improved test method

To carry out a total delay test described in Appendix A takes approximately 30 minutes if the test equipment is in place. Video delay test in Appendix B takes approximately 10 minutes to perform and analyze. To maintain a test procedure on a regular basis when developing new cameras an automated test method would be of large help. There have been discussions of developing a test setup where the analysis of the video to use a motion detection algorithm instead of scrolling through the pictures by hand deciding when the picture seems to have changed. Implementing such a solution would be helpful but not solving the problem of connecting the video movement in relation to the joystick movement, something that would have to be made in real time if not recording the events in some way.

Joystick

Moving the joystick functionality to a separate client would give a consequent behaviour of controlling the camera not depending on computer load of the client displaying the video.

Treehaven Technologies Inc. is an American based company producing Ethernet camera joystick controllers. An interesting test would be to evaluate their product with Axis cameras and a variation of VMS clients.

Acceleration tables

The IO processors working with pan and tilt control use a certain pattern when accelerating a motor i.e. acceleration tables. These tables may not currently be optimized to get the wanted motor speed as quick as possible. From tests it has been shown that the current tables introduce a delay of 30 ms until the desired speed is achieved.

7. Bibliography

[1] Axis Communications AB. “Annual report.” 2011.

[2] Axis Communications AB. “VAPIX.” `http://www.axis.com/techsup/cam_servers/dev/cam_http_ap_index.php`, April 2013.

[3] B. Davie, A. Charny, J. Bennett, K. Benson, J.-Y. Le Boudec, and B. Courtney. “An Expedited Forwarding PHB (Per-Hop Behavior). RFC 3246.” March 2002.

[4] B. Forouzan, S. Fegan, *et al.*. *Data communications and networking*, vol. 4. McGraw-Hill, 2001.

[5] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. “Assured Forwarding PHB Group. RFC 2597.” June 1999. Updated by RFC 3260.

[6] R. Hill, C. Madden, A. v. d. Hengel, H. Detmold, and A. Dick. “Measuring latency for video surveillance systems.” *Digital Image Computing: Techniques and Applications, 2009. DICTA ’09*, pp. 89–95, 2009.

[7] E. Huang, J. Sisk, T. Kirk, G. Coryell, and J. Stewart. “Searching for an ideal live video streaming technology.” `http://www.iupui.edu/~nmstream/live/introduction.php`, 2007. School of Informatics, Indiana University-Purdue University Indianapolis. Visited May 2013.

[8] I. S. MacKenzie and C. Ware. “Lag as a determinant of human performance in interactive systems.” *Proceedings of the ACM Conference on Human Factors in Computing Systems - INTERCHI ’93*, pp. 488–493, 1993.

[9] K. Nichols, S. Blake, F. Baker, and D. L. Black. “Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. RFC 2474.” December 1998. Updated by RFCs 3168, 3260.

[10] S. of Digital Formats. Planning for Library of Congress Collections. “MJPEG (Motion JPEG) Video Codec.” `http://www.digitalpreservation.gov/formats/fdd/fdd000063.shtml`, 2007. Visited June 2013.

[11] J. Postel. “User Datagram Protocol. STD 6. RFC 768.” August 1980.

[12] J. Postel. “Internet Protocol STD 5. RFC 791.” September 1981. Updated by RFCs 1349, 2474, 6864.

[13] J. Postel. “Transmission Control Protocol. STD 7. RFC 793.” September 1981. Updated by RFCs 1122, 3168, 6093, 6528.

[14] T. Wiegand, G. Sullivan, G. Bjøntegaard, and A. Luthra. “Overview of the h.264/avc video coding standard.” *IEEE Transactions on Circuits and Systems for Video Technology, Vol. 13, No. 7*, pp. 560–576, 2003.

A. Full system test

The full system test measures the time from moved joystick to a noticeable change on a monitor displaying the camera live stream. To make this measurement a second camera is introduced targeting the joystick and the monitor of the camera being tested, see Figure A.1 for an illustration of the setup. The video from the second camera is recorded on a second computer to not interfere with the streaming of the camera being tested.

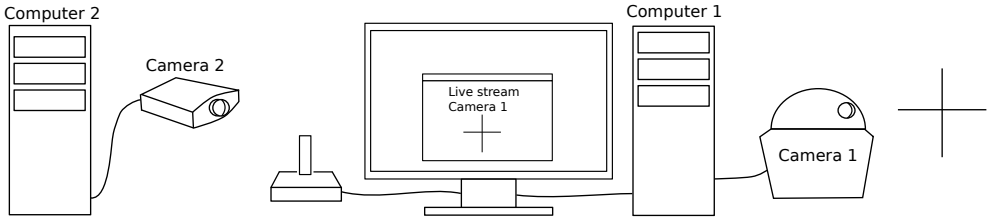


Figure A.1 Test setup. Camera 2 films the joystick and the monitor of Camera 1 which is the camera being tested.

Test setup

Two computers and two cameras are needed for this test. One camera/computer pair to be tested and one pair to record the test. For the tests in this thesis the recording camera was capable of recording at 60 FPS. The lighting of the testing environment should be bright to easily make out movements in the recording. The camera to be tested should target something with sharp edges, making the smallest movement clear in the recording.

Test Execution

A video stream is recorded from the camera filming the joystick and monitor through wget. The joystick is moved by hand making sure the tip of the joystick is showing in the recording to be able to make out when the joystick was pulled.

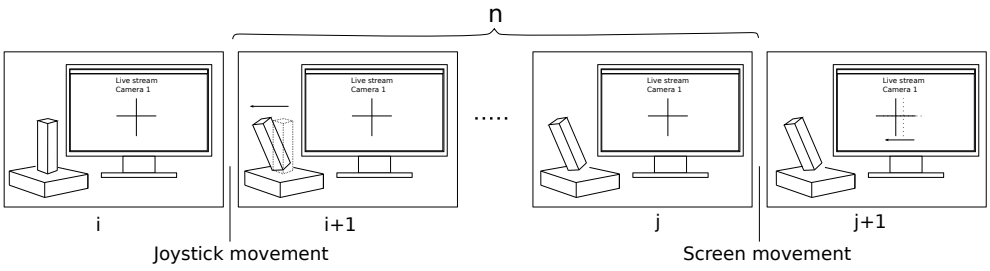


Figure A.2 The frames of importance when analyzing the video is the frame before (i) and after movement(i+1) of the joystick. The frame pair of interest is before movement on the monitor (j) and after first movement on the monitor (j+1) adding up to a total of n frames between events.

Analysis

The recorded video is analyzed frame by frame to make out when the joystick was pulled and when the monitor of the camera showed movement.

Since the camera filming has a limited speed of frames per second an estimate has to be done when in the time between frames movement has occurred. The probability for an event to occur soon after a frame is captured rather than close to the next frame is considered equal. The time between frames is therefore counted as $j - i$ according to the notation in Figure A.2 which illustrates the examination frame by frame of the recorded video. The number of frames in difference of the events is multiplied with the length of a video frames time slot. A camera shooting of 25 frames per second gives a frame time of $1/25s$.

B. Video delay test

The purpose of this test is to measure the video delay by comparing the time from the moment a video frame is captured to the display of a live stream on a connected client.

Test setup

To perform this test two computers and two cameras are needed. The camera to be tested is connected to one of the computers displaying a live stream from the camera. Additional to the live stream a digital stopwatch with millisecond precision is placed. The camera to be tested is aimed at its own monitor creating a loop of images showing old pictures in the same frame. The screen layout should look like Figure B.1. The second camera is setup to film the screen of the the first camera. A stream is recorded from the second camera. The full setup can be viewed in Figure B.2.

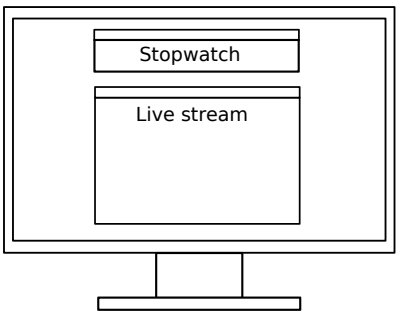


Figure B.1 Screen layout of the video delay test. Both the live stream and the stopwatch should be visible.

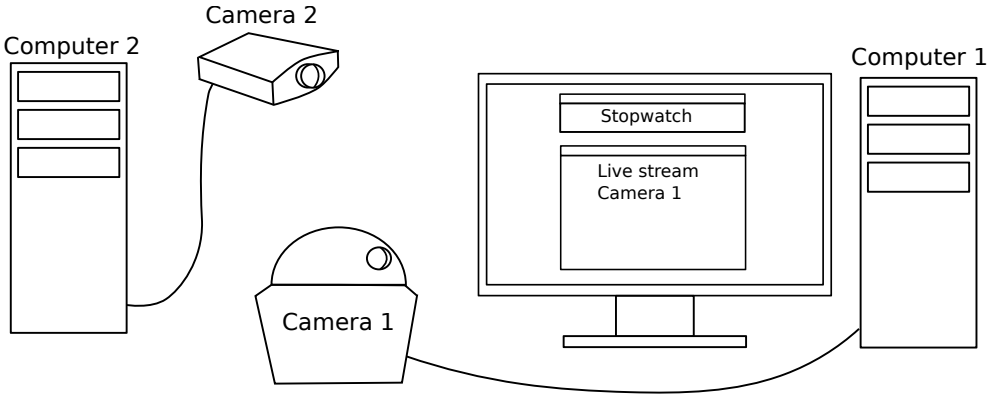


Figure B.2 An overview of the setup for the video delay test. Camera 1 is the camera tested in pair with Computer 1. Camera 2 records Camera 1's live stream through Computer 2. The two camera computer sets are separated to minimize interference in the test.

Test Execution

When the cameras are setup in the right positions and no additional streams are collected from the hosts to minimize interference of the test. The video stream from the camera not being tested is extracted through wget.

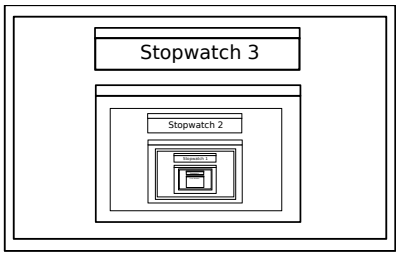


Figure B.3 Extracted frame from the test. The video delay is calculated by subtracting Stopwatch 2 from Stopwatch 3. Stopwatch 3 is referred to as the "outer" frame and Stopwatch 2 as the "inner" frame.

Analysis

The video is analyzed frame by frame to identify timestamps from the first camera in "outer" and "inner" frames where the time difference equals the video delay time, see Figure B.3 for an illustration. Time samples are to be extracted from every tenth frame. It is important to choose a frame where both timestamps are visible. This procedure is repeated ten times per test over approximately 130 frames. It is not always possible to choose frames from a certain interval since due to a low updating frequency of the monitor many of the timestamps shown in the captured frames becomes blurred.

C. Firmware delay test procedure

The firmware delay test is meant to measure the delay in the camera firmware from the start of the PTZ framework to occurance of the motor pulse from one of the motor control IO processors. When the test was executed the following equipment was used:

- Saleae Logic16 - Logic analyser.
- AXIS Q6032 - PTZ camera with modified circuit board.

C.1 Hardware modifications

The logic analyser used in this test can not be connected to an arbitrary circuit board and hence one circuit board was modified with debug pins.

- Power pin from the pan IO processor.
- Green LED channel.
- Red LED channel.

C.2 Software modifications

To measure points in the software execution with the logic analyser code to light and un-light the two LED:s need to be added to the firmware. The LED channels was modified at three places in the source code:

- Turn green LED channel off in start of VAPIX parser process.
- Turn green LED channel on in end of PTZ Driver backend.
- Turn red LED channel on in start of Kernel Driver.

The placements for the code to light and un-light the LED:s is illustrated in Figure C.1When a command is executed the LED:s need to be in a prefined state as:

LED	Initial state
Green	ON
Red	OFF

In this test the LED are lit when the channel are low (0) and unlit when the channel are high (1).

C.3 Connect the analyser

When hardware and firmware are modified as above the logic analyser is connected to the hardware signals (pan IO power pin, green LED channel, red LED channel). In this test the logic analysers input channels was connected in the same order. The logic analyser must also be connected to the analysed device's ground signal.

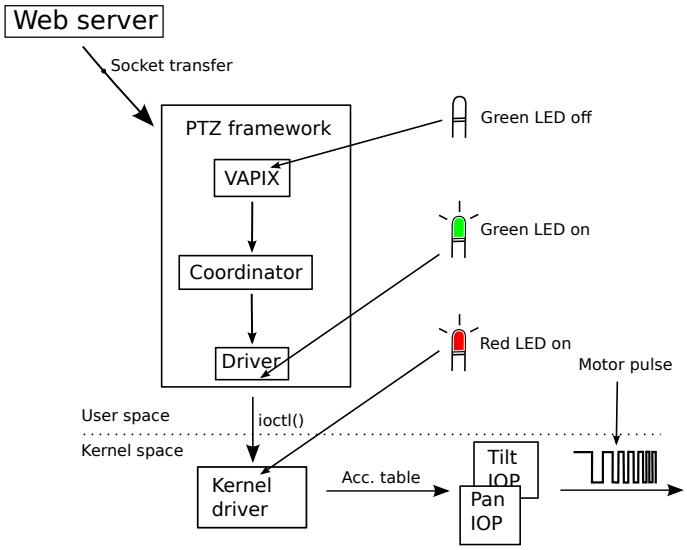


Figure C.1 Illustration of placements for the code to light and un-light the LED:s in the firmware delay test.

C.4 Logic analyser output

To obtain output from the analyser a special software[1] is used. The analyser samples its input signals and saves them to a recording.

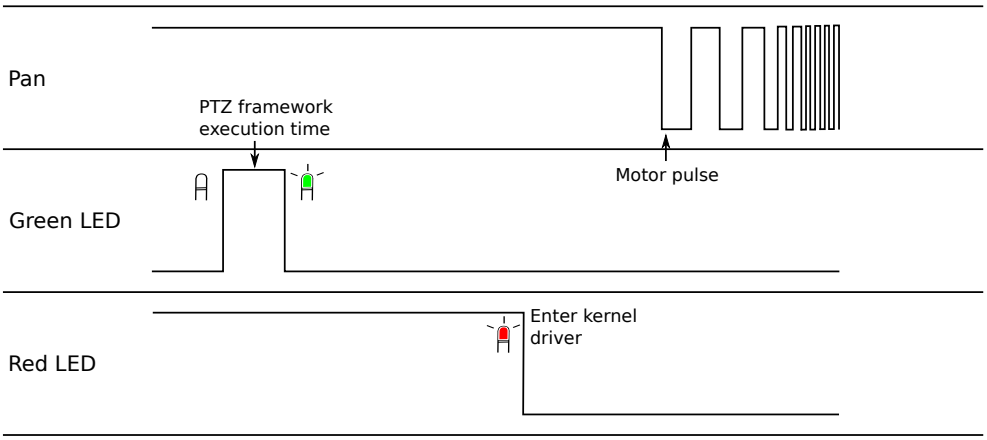


Figure C.2 Output from Logic analyser and how to interpret it. Three channels were used, from top to bottom: Pan power pin, Green LED channel (user space) and red LED channel (kernel space). The LED illustrations are the same as in Figure C.1 and is supposed to be a mapping between the two figures. Execution in the PTZ framework appears as a pulse in the middle row. The entrance to the kernel module is a shift in the bottommost signal.

Figure C.2 presents how to interpret the output from the logic analyser. The test aims to measure the following:

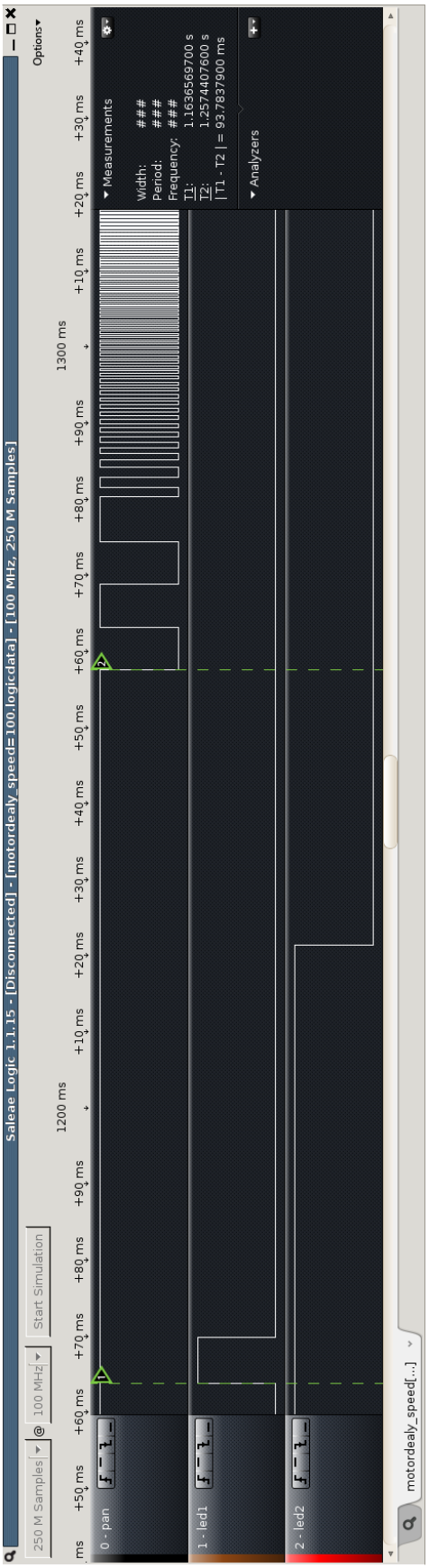
- PTZ framework execution.
- Time from PTZ framework to Kernel Driver.
- Time from Kernel Driver start to motor pulse.

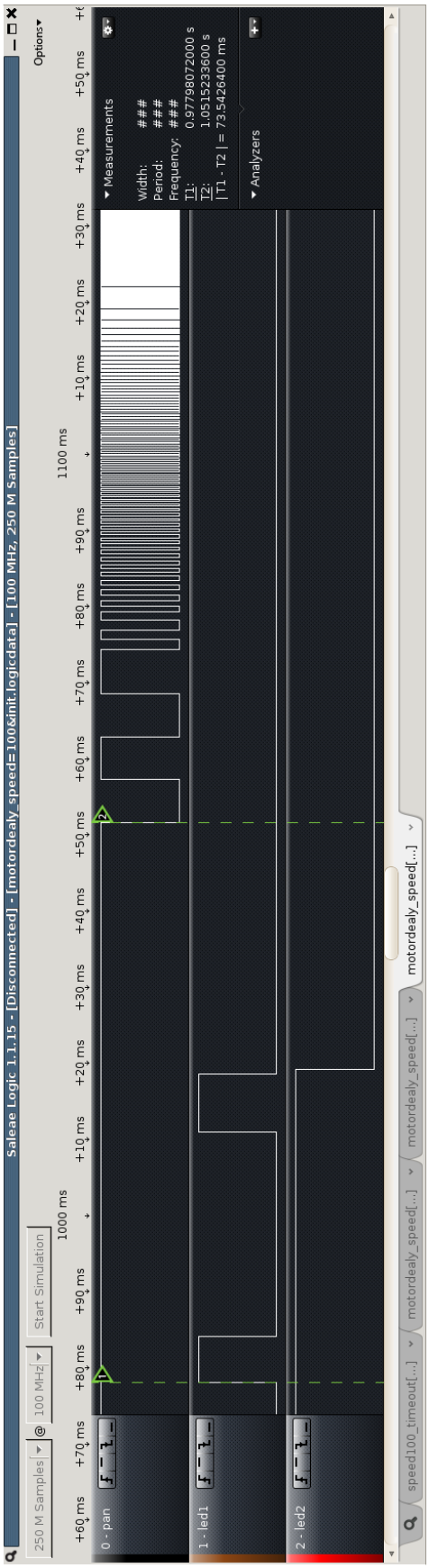
[1] The Logic16 software can be retreived from http://www.saleae.com/logic16

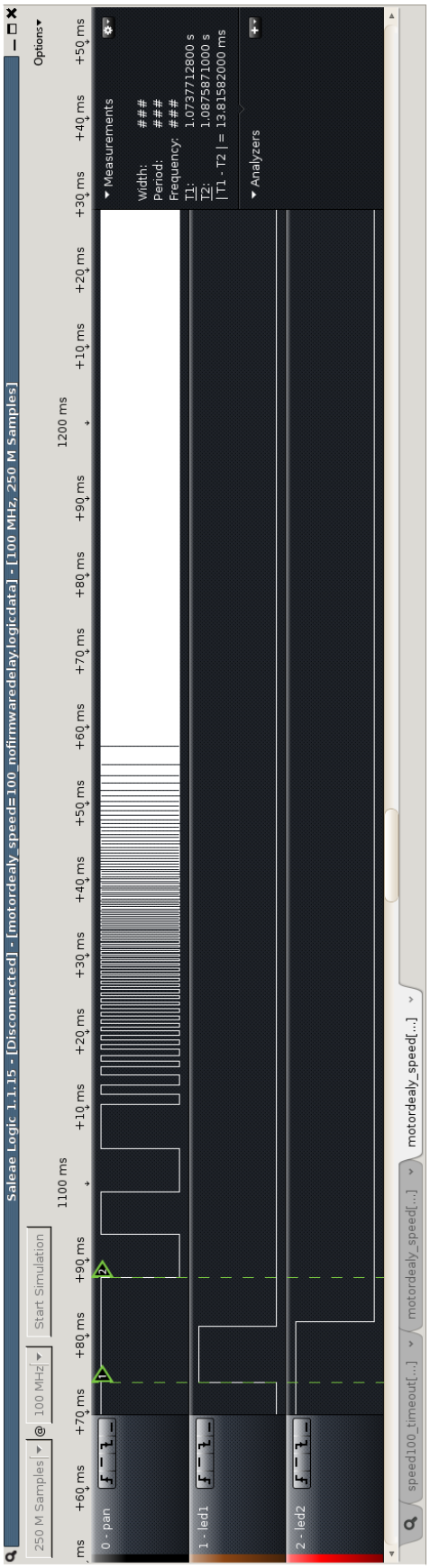
The analyser software has a tool making it easy to measure the distance between two signal flips. With the software two timing markers can be placed in the output. The interface displays the distance between the two markers. When a timing marker are being placed near a signal flip it automaticly gets attached to the flip, making it an easy to task to measure the time between the triggered signals in this test.

C.5 Example outputs

On the next three pages follows output from three tests. All presents the output when speed 100 are sent to the camera via the VAPIX API. The first two figures are from tests with original AXIS Q6032 firmware, where the first is when one move command is sent and the second when two move commands are sent. The third figure is the output from the test with modified firmware.


Saleae Logic 1.1.15 - [Disconnected] - [motordealy_speed=100.logicdata] - [100 MHz, 250 M Samples]
Options
250 M Samples
100 MHz
Start Simulation
1200 ms
1300 ms
+10 ms
+20 ms
+30 ms
+40 ms
+50 ms
+60 ms
+70 ms
+80 ms
+90 ms
Measurements
Width: ###
Period: ###
Frequency: ###
T1: 1.1636569700 s
T2: 1.2574407600 s
|T1 - T2| = 93.7837900 ms
Analyzers
0 - pan
1 - led1
2 - led2
motordealy_speed[...]


Saleae Logic 1.1.15 - [Disconnected] - [motordealy_speed=100&init.logicdata] - [100 MHz, 250 M Samples]
Options
250 M Samples
100 MHz
Start Simulation
1000 ms
1100 ms
+60 ms
+70 ms
+80 ms
+90 ms
+10 ms
+20 ms
+30 ms
+40 ms
+50 ms
+60 ms
+70 ms
+80 ms
+90 ms
+10 ms
+20 ms
+30 ms
+40 ms
+50 ms
0 - pan
1 - led1
2 - led2
Measurements
Width: ###
Period: ###
Frequency: ###
T1: 0.97798072000 s
T2: 1.05152336000 s
|T1 - T2| = 73.54264000 ms
Analyzers
speed100_timeout[...]
motordealy_speed[...]
motordealy_speed[...]
motordealy_speed[...]


Saleae Logic 1.1.15 - [Disconnected] - [motordealy_speed=100_nofirmwaredelay.logicdata] - [100 MHz, 250 M Samples]
Options
250 M Samples
100 MHz
Start Simulation
1100 ms
1200 ms
+10 ms
+20 ms
+30 ms
+40 ms
+50 ms
+60 ms
+70 ms
+80 ms
+90 ms
Measurements
Width: ###
Period: ###
Frequency: ###
T1: 1.073712800 s
T2: 1.087587100 s
|T1 - T2| = 13.81582000 ms
Analyzers
0 - pan
1 - led1
2 - led2
speed100_timeout[...]
motordealy_speed[...]
motordealy_speed[...]

Lund University
Department of Automatic Control
Box 118
SE-221 00 Lund Sweden

Document name
MASTER THESIS

Date of issue
August 2013

Document Number
ISRN LUTFD2/TFRT--5923--SE

Author(s)
Linus Svensson
Petter Söderlund

Supervisor
Petter Johansson, Axis Communications
Karl-Erik Årzén, Dept. of Automatic Control, Lund University, Sweden (examiner)

Sponsoring organization

Title and subtitle
Delays in Axis IP surveillance cameras

Abstract
Controlling motors over IP networks naturally introduces delays. For an operator of a surveillance camera to point the camera (e.g. following an object) delays are of importance.
This master's thesis investigates the sources of steering delays in IP-cameras, testing on AXIS Q6035 and AXIS Q6032 with different video management software and video codec. Camera firmware and communication protocol are evaluated with aim to find improvements.
The results show that the client computer, displaying video and sending steering commands, contributes with the largest uncertainty in the system. Video management software and computer load are the key factors. Camera firmware proved to have inefficient communication schemes, which when changed, reduced delay by 70 ms.

Keywords

Classification system and/ or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
1-60

Recipient's notes

Security classification