Självständigt arbete på grundnivå

Independent degree project - first cycle

Datateknik
Computer Engineering

Videokonferens med Raspberry Pi

Roger Sundh

Mittuniversitetet
MID SWEDEN UNIVERSITY

Campus Härnösand Universitetsbacken 1, SE-871 88. **Campus Sundsvall** Holmgatan 10, SE-851 70 Sundsvall.
Campus Östersund Kunskapens väg 8, SE-831 25 Östersund.
Phone: +46 (0)771 97 50 00, Fax: +46 (0)771 97 50 01.

MITTUNIVERSITETET
Avdelningen för informations- och kommunikationssystem

Examinator: Ulf Jennehag, ulf.jennehag@miun.se
Handledare: Ulf Jennehag, ulf.jennehag@miun.se
Christian Johansson, christian.johansson@consat.se
Författare: Roger Sundh, rosu0602@student.miun.se
Utbildningsprogram: Högskoleingenjör, IngOnline, 180 hp
Huvudområde: Datateknik
Termin, år: VT, 2015

Sammanfattning

Föreliggande examensarbetsrapport behandlar möjligheterna att skapa ett system för audio- och videokommunikation mellan två noder, med hjälp av enkel och relativt billig hårdvara. Kommunikationen ska ske över ett lokalt nätverk som använder Ethernet och IP. Arbetet fokuserade på den prisvärda mikrodatorn Raspberry Pi och dess möjligheter och begränsningar i sammanhanget. Jag har även undersökt de möjligheter som en så kallad mini-PC erbjöd och gjort jämförelser mellan dessa system. Avsikten var att få till stånd en tvåvägskommunikation med både rörlig bild och ljud av hög kvalitet, helst av HD-kvalitet. Målet var att användarna ska få en känsla av personlig närvaro och för detta ändamål ska en stor bildskärm användas vid varje nod. Bildskärm, mikrofon, kamera och högtalare ska byggas ihop till en enhet som ska bilda något som liknar ett fönster. Detta system kan då monteras på en vägg och eftersom uppkopplingen är avsedd att vara permanent, kan man få känslan av direkt närvaro, om man placerar sig vid ”fönstret” och tittar på den man kommunicerar med. Denna typ av kommunikation är användbar vid exempelvis distansmöten, föreläsningar och liknande situationer. Resultaten visade att Raspberry Pi utmärkt kunde hantera antingen sändning eller mottagning, men när båda sakerna skulle utföras samtidigt, som är fallet vid videokonferenser, räckte den inte riktigt till. En lösning var då att använda två stycken Raspberry Pi per nod; en för sändning och en för mottagning. En annan lösning var att byta ut Raspberry Pi mot mer kraftfull hårdvara, exempelvis mot en mini-PC.

Nyckelord: Videokonferens, Raspberry Pi, H264.

Abstract

This report deals with the possibilities of creating a system for a point-to-point audio- and video communication link, using cheap and simple hardware. The system will be using a local area network based on Ethernet and IP. This work was focused around the Raspberry Pi and its possibilities and limitations, regarding this type of application. The usage of a Raspberry Pi was also compared to using a conventional mini-PC for the same purpose. The overall goal was to achieve a system for point to point audio- and video communication of a very high quality, preferrably HD quality. The users of the system are supposed to get a feeling of live communication. To achieve this a very large monitor was used at each node. The monitor, microphone, camera and speaker are to be integrated into a frame that is hung on a wall in order to resemble a window. When you want to talk to somebody at the other node, you just walk up to this window and call upon the person you want to speak to, since the communications link is always active. This system could be used for distance meetings, distance lectures and in other similar situations. The results indicated that the Raspberry Pi was very capable of handling either transmission or reception of HD quality signals, but for full duplex communication a setup using two separate Raspberry Pis per node is recommended – one for transmission and one for reception. Another solution would be to choose a more powerful hardware platform, for instance a mini-PC, instead of the Raspberry Pi.

Keywords: Video conferencing, Raspberry Pi, H264

Förord

Jag vill här passa på att tacka Christian Johansson och Meysam Eriksson, mina handledare på Consat AB i Göteborg och Ulf Jennehag, min handledare på Mittuniversitetet, för deras hjälp vid genomförandet av detta examensarbete. Ett stort tack går också till min sambo Petra Torgilsson för hennes stöd och tålamod under de perioder jag varit uppslukad av arbetet vid datorn och således inte i tillräcklig utsträckning fullgjort min del av det nödvändiga gemensamma hushållsarbetet. Slutligen vill jag också tacka våra katter Freja och Tiger, för deras tålmodiga sällskap under många sena nätter.

Innehållsförteckning

Figurförteckning

Tabellförteckning

Terminologi

Här följer en liten förklaring av några begrepp som används i denna rapport.

Förkortningar

H264	En metod att komprimera digitala videosignaler. Vissa webbkameror kan sända direkt i detta komprimerade format, till exempel kameran C920 från Logitech.
IDE	Integrated Development Environment. Utvecklingsmiljö för programmering
API	Application Programming Interface. Paket med rutiner att anropa för ett visst användningsområde, t. ex. videokommunikation.
RTP	Realtime Transport Protocol. Protokoll i applikationslagret som sköter överföring av realtidsdata i ett IP-baserat nätverk.
RTCP	Realtime Transport Control Protocol. Protokoll som arbetar tillsammans med RTP för att sköta synkroniseringen mellan sändare och mottagare vid dataöverföring i realtid.

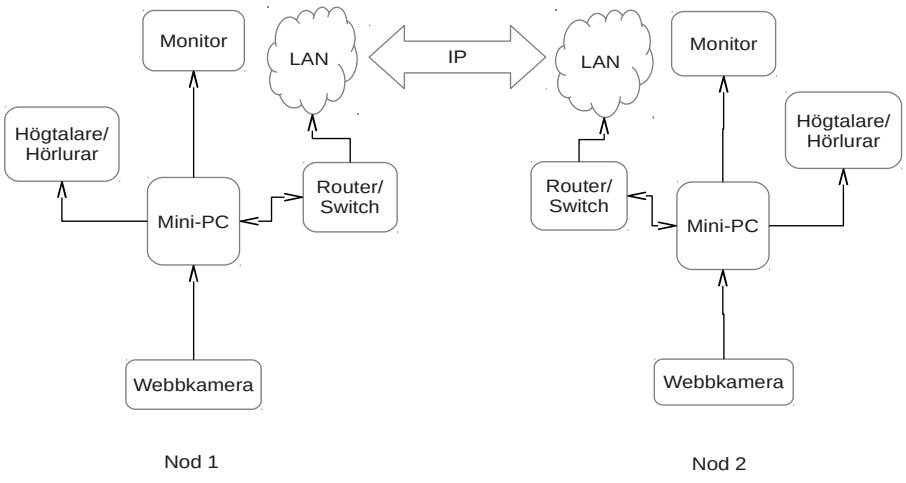
1 Inledning

I dagsläget finns det många produkter du kan använda dig av, när du vill kommunicera med en annan person. Persondatorn har där sin givna plats, exempelvis i kombination med programmet Skype. Ibland kan det dock finnas ett behov av en kommunikationslänk med ett lite annorlunda fokus än vad det i mina ögon ganska generella programmet Skype erbjuder. Om man skalar bort en del flexibilitet till förmån för lågt pris och enkelhet kanske det är möjligt att åstadkomma någonting liknande Skype, men som exempelvis kan användas på en hårdvaruplattform med ett lägre pris än en persondator. Avsikten med examensarbetet är att undersöka om det är möjligt att använda det inbäddade systemet Raspberry Pi som bas i ett system för audio- och videokommunikation mellan två stycken noder i ett lokalt nätverk. Noderna ska vara permanent uppkopplade mot varandra och kommunikationen ska hålla så hög kvalitet att användarna får känslan av att de befinner sig i samma rum när de pratar med varandra över nätverket.

Det tänkta kommunikationssystemet kan användas för spontana distansmöten, intervjuer eller föreläsningar.

Efter diskussion med handledaren Christian Johansson på Consat AB i Göteborg, kom vi överens om att det mesta av arbetet ska ske på distans, då min hemort är Osby i norra Skåne. För ändamålet har jag anpassat mitt kontor hemma genom att koppla upp nödvändig utrustning till ett redan befintligt hemmanätverk. Det mesta av utrustningen tillhandahålls av Consat AB.

I figur 1 syns en översikt över den slutligen valda lösningen.



Figur 1: Systemöversikt

1.1 Bakgrund och problemmotivering

Consat AB arbetar bland annat med industriell IT och telematik för exempelvis kollektivtrafik. Consat AB har verksamheter i olika länder, bland andra Kanada, Brasilien och Sverige. Huvudkontoret finns i Partille strax utanför Göteborg och det finns lokala kontor i centrala Göteborg och även i Stockholm. För att korta beslutsvägarna, öka samhörighetskänslan och förenkla kommunikationen påbörjades på huvudkontoret i Partille tester med ett kommunikationssystem baserat på Skype. Företaget önskade en punkt-till-punktkommunikation med både ljud och bild som erbjöd känslan av närvaro, trots att de som kommunicerade befann sig på olika lokalkontor. Resultaten av dessa inledande försök var positiva och beslut togs om att erbjuda ett examensarbete, vars syfte var att ta fram ett fristående system för denna punkt-till-punktförbindelse.

1.2 Övergripande syfte

Det övergripande syftet med detta examensarbete är att undersöka om det går att skapa ett system för högkvalitativ punkt-till punktöverföring av audio- och videosignaler över ett lokalt nätverk med hjälp av billig och enkel hårdvara, exempelvis med en Raspberry Pi eller en mindre persondator, så kallad mini-PC. Förhoppningen är att resultatet ska kunna ligga till grund för utvecklingen av en ny produkt för Consat AB.

1.3 Avgränsningar

Examensarbetet är avgränsat till att undersöka konceptet och dess möjligheter och begränsningar. I bästa fall kan en prototyp framställas, men den slutliga produktframtagningen blir ett senare arbete.

1.4 Detaljerad problemformulering

Arbetet ska som resultat ge minst ett förslag på hur det önskade kommunikationssystemet ska kunna realiseras. De framarbetade förslagen kan sedan jämföras, med avseende på pris, prestanda och kvalitet och resultaten kan bilda underlag för en framtida ny produkt.

1.5 Översikt

Den här examensrapporten är indelad i 6 st. kapitel. Det första kapitlet ger läsaren en orientering om examensarbetets syfte. Detta följs av ett kapitel om ämnets teoretiska bakgrund, vilket följs av en metodikbeskrivning i kapitel 3. Det fjärde kapitlet behandlar olika lösningar av det givna problemet och i det efterföljande kapitlet redovisas de erhållna resultaten. Slutligen diskuteras resultaten och vissa slutsatser dras i kapitel 6. Allra sist finns en referenslista och en förteckning över bilagor.

2 Teoretiskt bakgrundsmaterial

Kapitlet ger en teoretisk bakgrund till audio- och videokommunikation i nätverk, med fokus på de tekniker och verktyg som använts i detta examensarbete.

2.1 Definition av termer

Ordförklaringar

GStreamer	Ett programpaket (API) för audio- och videokommunikation. Finns för många olika plattformar, exv. Linux, Windows, OSX och Android. Öppen källkod.
Codeblocks	Utvecklingsmiljö (IDE) för bland annat C/C++-programmering. Öppen källkod.
Raspbian	Operativsystem för Raspberry Pi. Baserat på Linuxdistributionen Debian.
Raspivid	Program för att visa högupplöst video på en Raspberry Pi.
NetCat	Mångsidigt program för nätverkskommunikation. Kan bland annat användas för att föra över data mellan två noder i ett nätverk.
Bash	Kommandoskal för linuxmiljöer. Används ofta som bas för så kallade skalskript (eng. Shellscript), även kallat kommandofiler.
Sink	I kommunikationssammanhang betecknar sink en mottagare av data, exempelvis en bildskärm eller ett ljudkorts utgång.
Source	Betecknar en kommunikationskälla, exempelvis en videokamera, eller en mikrofoningång
Pipeline	Beteckning i GStreamer som anger hela signalvägen från source till sink.
SoC	System on a Chip. Ett helt system med minne, CPU och GPU i en enda krets.
CPU	Central Processing Unit. Enhet i dator som sköter all grundläggande datahantering och beräkningar m.m.
GPU	Graphics Processing Unit. Hjälpprocessor, speciellt konstruerad för snabba grafikberäkningar.

2.2 Audio- och videokommunikation

När jag startade examensarbetet hade jag inte så stor kännedom om alla kodekar för audio och video som existerar. Nu när jag genomfört det har jag lärt mig en hel del mera, men samtidigt har jag också förstått att det finns väldigt mycket ytterligare att lära sig inom detta område.

2.2.1 Allmän bakgrundsinformation

För att överföra audio- och videodata mellan 2 st. noder i ett nätverk behöver många kriterier vara uppfyllda. Den (eventuellt) analoga informationen måste i ett första steg digitaliseras innan den kan skickas över en digitalt baserad förbindelse. Idag finns det webbkameror (C920 från Logitech) som kan leverera video i digitalt format enligt specifikationen H.264 och digitaliserad audio i många olika format, allt via en USB-port.

Typ av nätverk måste bestämmas – idag är förmodligen ett IP-baserat nät det rätta valet. Vidare kan nätet vara trådlöst eller trådbundet. Noderna i nätet kan bestå av vanliga skrivbordsdatorer, mobiltelefoner eller inbäddade system. Beroende av den valda hårdvaran erbjuds sedan ytterligare en mängd valmöjligheter gällande mjukvaran, i första hand val av operativsystem. För att anpassa nodernas hårdvara och mjukvara till varandra finns det också möjlighet att skriva skalskript eller kommandofiler i det valda operativsystemet. För att ytterligare få kontroll över systemet kan en programmeringsmiljö användas.

I det allmänna fallet kommer vi alltså att ha 2 st. noder som kan ha helt olika hårdvara och mjukvara, men som ändå ska kunna kommunicera med varandra.

2.2.2 Digitala format

Det existerar en uppsjö av format för digital audio [10] och video [11]. Vissa format är bättre lämpade än andra, när det gäller användning för videokonferenser. Det är också nödvändigt att skilja på det s.k. lagringsformatet (container) och komprimeringsalgoritmer och kodekar. För ett konferenssystem är det inte i första hand aktuellt med någon lagring av informationen, utan här gäller det främst att hitta ett format lämpat för att strömma ljud och bild.

2.2.3 Protokoll

När data ska överföras mellan noder i ett nätverk krävs ett kommunikationsprotokoll för att administrera kommunikationen. Eftersom det existerar mängder av kommunikationsprotokoll, gäller det att även här försöka välja ett som är lämpat för överföring av audio- och videodata i realtid.

Den vanligaste modellen för att beskriva nätverkskommunikation är att använda en flerlagermodell, där varje lager har en specifik uppgift och kommunicerar med närliggande lager med hjälp av ett protokoll. Ett sätt att beskriva den kommunikationen är att använda OSI-modellen, med 7 st. olika lager och ett annat är att använda TCP/IP-modellen, med 4 olika lager.

Om vi antar att vi har en IP-baserad kommunikation, kommer vi alltså i videokonferensfallet att ha 2 st. noder som kommer att överföra data mellan var-

andra med hjälp av något protokoll från IP-familjen, vanligen UDP eller TCP. En av fördelarna med TCP är i allmänhet att det ger en garanterad dataöverföring, tack vare dess inbyggda förmåga till omsändning vid överföringsfel. I videokonferenssammanhang är detta ingen större fördel, eftersom det ökar bandbreddsbehovet och kan skapa synkroniseringsproblem i bild och ljud. Eftersom människans ögon och öron har sina begränsningar, är de i vissa fall förlåtande mot överföringsfel och reagerar mindre på förluster av enstaka datapaket, än på synkroniseringsproblem. Ur den synvinkeln är det vanligare att använda UDP för denna typ av kommunikation, eftersom det protokollet i transportlagret inte utför några automatiska omsändningar av förlorade paket.

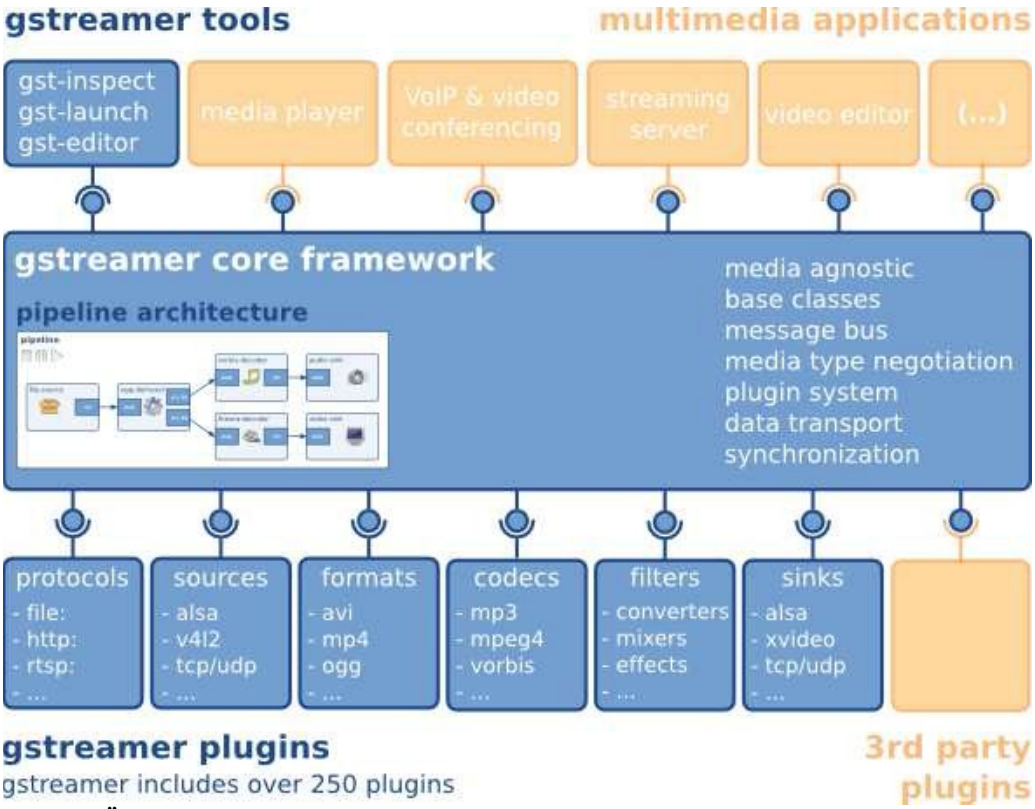
Vanligare är att använda RTP [13] i applikationslagret och UDP är då transportprotokollet. Eventuella omsändningar sköts av RTP (m.h.a. kompanjonprotokollet RTCP).

2.2.4 Kodekar

En kodek är mjukvara eller hårdvara som kan koda (komprimera) och avkoda audio- och videodata enligt en viss komprimeringsalgoritm, exempelvis H264 [12] för video och A-law för audio. Till exempel kan programpaketet GStreamer [3] kan ta hand om denna uppgift. Kodekar ingår även i Windows Media Player, Skype och andra multimediaprodukter. I många fall går det att ladda ner separata kodekar och installera i sina befintliga audio- och videospelare. Vissa av dem är gratis (öppen källkod), medan andra leverantörer vill ha betalt.

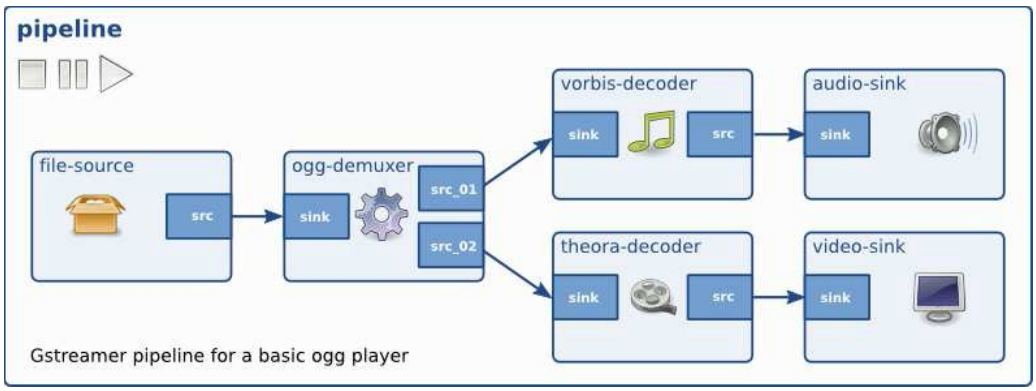
2.2.5 GStreamer

För den som inte känner till GStreamer kommer här en kort introduktion till dess funktion och användning. GStreamer är en miljö för utveckling av applikationer inom audio och video. GStreamer används både för att exempelvis spela upp ljud och bild över nätverk, såväl som för att skapa videoredigeringsprogram med mera. Det är uppbyggt som ett API, med möjliga kopplingar till en mängd programmeringsspråk, av vilka C/C++ och Python är mest intressant för detta examensarbete. Dessvärre finns det 2 st. oberoende versioner av GStreamer – en äldre med beteckningen 0.10 och ett nyare utvecklingsspår med beteckningen 1.x. I detta arbete har jag i huvudsak använt mig av GStreamer 1.2. När detta skrivs finns det dock en version med beteckningen 1.4.5, men den har jag inte testat. I figur 2 syns en översikt över GStreamer och dess möjligheter.



Figur 2: Översikt över GStreamer. Bilden är hämtad från manualen för GStreamer. [5]

Grundläggande begrepp inom strömmande data är orden *sink* och *source*. Source betecknar en källa för exempelvis audio eller video, t. ex. en mikrofon eller en webbkamera. Sink betecknar en mottagare i form av en högtalare eller en bildskärm. Dessa båda begrepp är tillsammans med begreppet pipeline, grundpelarna i GStreamer. En *pipeline* är en kedja av moduler (*plugins*), vilka ansluts till varandra i serie, enligt figur 3.



Figur 3: Ett exempel på en så kallad pipeline i GStreamer. Exemplet visar en enkel mediaspelare för ogg-filer. Bilden är hämtad från manualen för GStreamer. [5]

En plugin är en mjukvarumodul som utför en viss avgränsad uppgift. Det finns mängder av olika plugins för GStreamer och fler tillkommer. För den intresserade finns även möjlighet att skapa egna moduler för GStreamer.

GStreamer kan installeras på olika sätt. Det enklaste är kanske att använda kommandot:

```
sudo apt-get install gstreamer1.0-*
```

men eftersom GStreamer innehåller en massa moduler är det bäst att vända sig till en mera ingående beskrivning om man får problem med installationen. En möjlighet är att installera från källkoden och kompilera alla moduler själv [6]. Jag har provat båda dessa metoder och den senare metoden är nödvändig om den modul du behöver inte finns att installera i färdigkompilerad form, som exempelvis modulen uvch264src, vilken behövdes för att kunna sända video direkt kodad i formatet H264.

I GStreamer ingår ett par kommandoradsverktyg. Det ena heter *gst-inspect* och det andra heter *gst-launch*.

2.2.5.1 Gst-inspect

Verktyget gst-inspect används för att se vilka insticksmoduler som är installerade i systemet och vilka möjligheter dessa moduler har i form av parametervärden m.m. Vid kommandoprompten skriver du följande kommando om du använder dig av GStreamer 1.x.

```
gst-inspect-1.0
```

Detta ger dig en lista över de installerade modulerna. Vill du se en moduls inställningsmöjligheter kan du exempelvis skriva följande kommando:

```
gst-inspect-1.0 uvch264src
```

Som svar får du i de flesta fall en omfattande beskrivning av vad modulen klarar av och dess förvalda parametervärden m.m.

2.2.5.2 Gst-launch

Med hjälp av gst-launch kan du snabbt testa en pipeline. Det går att bygga upp de mest avancerade kommandokombinationerna i detta verktyg och jag har använt det under hela examensarbetet. Ett exempel kan se ut så här

```
gst-launch-1.0 filesrc location=Video/trailer_1080p.ogv !
oggdemux name=demux ! queue ! vorbisdec ! audioconvert !
autoaudiosink demux. ! queue ! theoradec ! videoconvert !
autovideosink
```

Detta exempel motsvarar den pipeline som visas i figur 3.

2.3 CodeBlocks IDE

CodeBlocks är en utvecklingsmiljö för bland annat programmering i C/C++. Den går att använda i både Windows, Linux och Mac OS X. Jag har använt den i Linux på både Raspberry Pi och Zotac mini-PC. Miljön är relativt lätt att lära sig och den är flexibel.

2.4 Raspberry Pi

Raspberry Pi [14] är ett inbäddat system. Dess huvudkomponent är ett SoC från Broadcom och systemet finns i ett antal olika modeller och dess utmärkande drag är att den är billig, liten och har blivit otroligt populär bland ingenjörer och programmerare. Den har en kraftfull GPU, kan köra Linux, har HDMI- och nätverkskontakter och arbetar med SD-kort för lagring av filer.

2.4.1 Raspbian

Detta är en variant av Linuxdistributionen Debian. Raspbian är speciellt anpassad för att köras på Raspberry Pi. Eftersom det rör sig om en fullfjädrad linuxdistribution går det att i systemet installera alla möjliga applikationer och utvecklingsverktyg, däribland CodeBlocks och GStreamer.

2.5 Likartade system

2.5.1 Skype

Skype [15] är ett verktyg för kommunikation över nätverk. Det använder sig av både ljud och bild. Företaget köptes för en tid sedan upp av Microsoft. Att använda programmets grundfunktioner är gratis för privatpersoner, men för de mera avancerade funktionerna krävs en licensavgift. I dagsläget är Skype ett fristående program som du laddar ner till din dator, men på webbsidan finns en länk till en testversion av en webbaserad produktvariant.

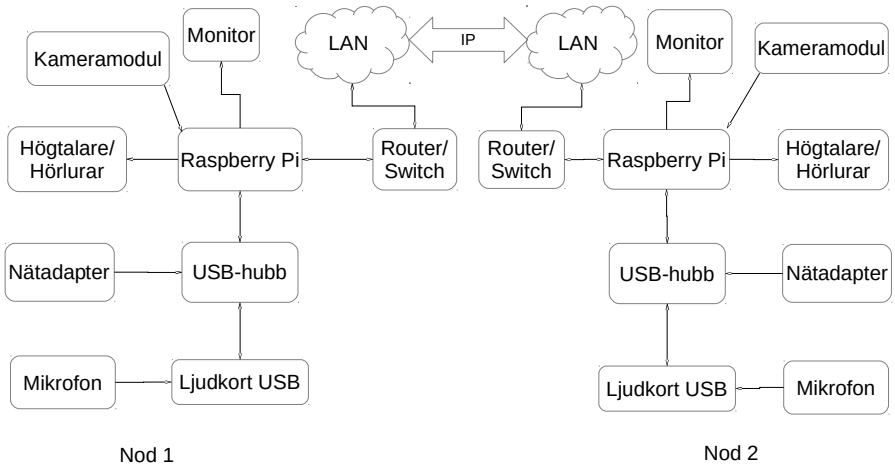
2.5.2 WebRTC

WebRTC [16] är ett API som kan användas av webbläsare och mobiler för att implementera audio- och videokommunikation i realtid. Det finns ett antal demonstrationsapplikationer att ladda ner från deras webbplats. Ett exempel är ett program [17] som kan köras direkt i webbläsaren (Chrome, Firefox eller Opera) som fungerar på liknande sätt som detta examensarbete – dock saknas exempelvis en förtitt.

3 Metod

Efter inledande diskussioner med Consat AB beslutades att arbetet kunde utföras på distans. Arbetsmetoden bestod alltså i att koppla upp utrustningen hemma och sedan genomföra de tester och utveckla den programvara som krävdes för att utvärdera om det önskade målet kunde uppnås. Jag har använt mig av mitt eget lokala nät, med en nod placerad i mitt arbetsrum och den andra noden ute i hallen. Längre fram i arbetet placerades den andra noden i källaren, för att helt undvika problem med akustiskt ljudläckage.

Vid en första träff med handledarna hos Consat AB fick jag med mig lite utrustning (monitor, bok om Raspberry Pi [7] m.m.). Övriga komponenter beställde handledarna senare och de skickades direkt hem till mig. I ett första skede jobbade jag med en Raspberry Pi per nod enligt figur 4.



Figur 4: En (1) Raspberry Pi per nod med RaspiCam kameramodul.

Av detta framgår att varje nod består av följande komponenter:

- 1 st. Raspberry Pi
- 1 st. Raspberry Pi kameramodul
- 1 st. PC-mikrofon, via ett separat USB-ljudkort
- 1 st. LCD-monitor
- 1 st. hörlurar eller högtalare.

Eftersom Raspberry Pi inte har en egen audioingång, behövdes ett externt ljudkort, med en ingång för mikrofon.

Detaljer om de olika konfigurationer jag testat följer i kapitel 4.

Av de inledande samtalen med Consat AB, framgick att de hade gjort preliminära tester genom att använda Skype på PC och deras önskemål var att detta arbete skulle uppnå motsvarande kvalitet i kommunikationen, men med en fast uppkoppling mellan endast 2 st. noder. Eftersom jag initialt inte alls visste vilka programvaror som var lämpliga för detta arbete, fick jag börja med att göra vissa efterforskningar. Med hjälp av Internet visade det sig att om jag hade kommit igång lite tidigare hade jag kunnat använda ett utvecklingskit från Skype [8], men de hade precis slutat att stödja detta, när jag skulle starta arbetet.

Efterforskningarna ledde till att jag valde att använda följande verktyg för det fortsatta arbetet:

- Raspbian [1]
- GStreamer 1.0 [3]
- Codeblocks [4]

En närmare förklaring av dessa verktyg finns i kapitel 2.

3.1 Val av API

Att från grunden försöka skapa ett ramverk för kommunikation av digitaliserad ljud- och bildinformation över ett lokalt nätverk är en gigantisk uppgift. Som tur är finns det redan många som arbetat med detta sedan tidigare och som skapat system som är avsedda att användas av andra i just detta syfte. För att uppnå tillräckligt god kvalitet krävs det att den hårdvara som används är snabb och att programvaran som används kan utnyttja alla hårdvarans möjligheter.

På Raspberry Pi finns ett huvudchip som är ett s.k. SoC. Det är tillverkat av BroadCom och heter BCM2835. Detta chip innehåller bland annat processorn ARM1176 (som bygger på ARMv6-arkitekturen) med klockfekvensen 700 MHz och en grafikprocessor baserad på multimediaarkitekturen Videocore, version 4.

Efter många timmars efterforskningar genom sökningar på nätet hittade jag GStreamer, vilket är väldokumenterat och relativt enkelt att förstå sig på, vilket är beskrivet i teorikapitlet. GStreamer har utmärkt stöd för olika hårdvarutyper.

3.2 Det löpande arbetet

När jag börjat förstå lite mer av hur systemet skulle byggas upp kopplades 2 st. noder upp och jag kunde börja utvärdera enligt de kvalitetskriterier jag fått (d.v.s. att det skulle ”fungera ungefär som Skype”). Lite mer detaljerat innebär detta att det ska vara god synkronisering mellan ljud och bild och bildfrekvensen ska vara så pass hög att det ser naturligt ut. Känslan ska vara att man står

och pratar med varandra i realtid, ”på riktigt”. Att översätta detta till mera påtagliga siffervärden är inte helt lätt, men en bildfrekvens på minst 20 bildrutor/s och en tidsfördröjning under 100 ms är rimliga utgångsvärden.

Initialt använde jag mig av möjligheten att skriva skalskript i bash. Bash är ett vanligt kommandoskal i Linuxmiljö. I dessa skript anropades bland annat GStreamers kommandoradsprogram gst-launch, tack vare dess flexibla möjligheter att experimentera med parametrarna för kommunikationens s.k. pipeline. På det sättet testade jag mig framåt och kunde lära mig mera om GStreamer och dess moduler. Exempel på dessa skalskript finns i bilaga A.

En svårighet med att arbeta empiriskt på detta sätt är att det snabbt växer fram många olika alternativ och att då sovra i det materialet kan bli väldigt tidsödande. Under arbetets gång har jag försökt att bokföra de mest lyckade parametervalen och inställningarna genom att spara dessa i en slags dagbok, parallellt med att jag också noterat vilka program jag installerat på Raspberry Pi (eller mini-PC). Dessa anteckningar återfinns i bilaga C.

3.3 Utvärdering och test

Det är inte möjligt att testa och utvärdera den här typen av system på egen hand. Det måste finnas någon som kommunicerar i båda noderna av systemet. Detta problem löstes genom att jag lyckades övertala min sambo att vara min testassistent. Vid vissa tillfällen har även andra vänner varit behjälpliga som assistenter, genom att prata och gestikulera i ”den andra änden”.

Att utvärdera de i kapitel 4 beskrivna uppställningarna är inte helt enkelt. Vilka kriterier ska användas? Ett sätt är att försöka hålla sig till rent mätbara saker såsom CPU-belastning, minnesanvändning och nätverksbelastning, men i det här fallet anser jag att det är enklare att gå efter den subjektiva kvalitetsuppfattning som de som kommunicerar uttrycker. Saker som är intressanta att titta på är upplevd bildkvalitet, upplevd ljudkvalitet, fördröjningar i ljud och bild och synkroniseringsproblem mellan ljud och bild. Eftersom förbindelsen är tänkt att ständigt vara igång, är även långsiktig stabilitet i uppkopplingen mycket betydelsefull.

Av de kvalitetsaspekter som tas upp av H. Gulliksson [9] har jag förutom de ovan nämnda valt att fokusera på bildfrekvens, komprimeringsgrad (bithastighet) och upplösning.

4 Lösningsalternativ

Jag har definierat några olika möjliga lösningar för hur kommunikationen kan utföras. I samtliga fall är det även möjligt att experimentera med den mjukvarumässiga lösningen, exempelvis om kommunikationen ska ske över UDP eller TCP och vilka kodekar som används.

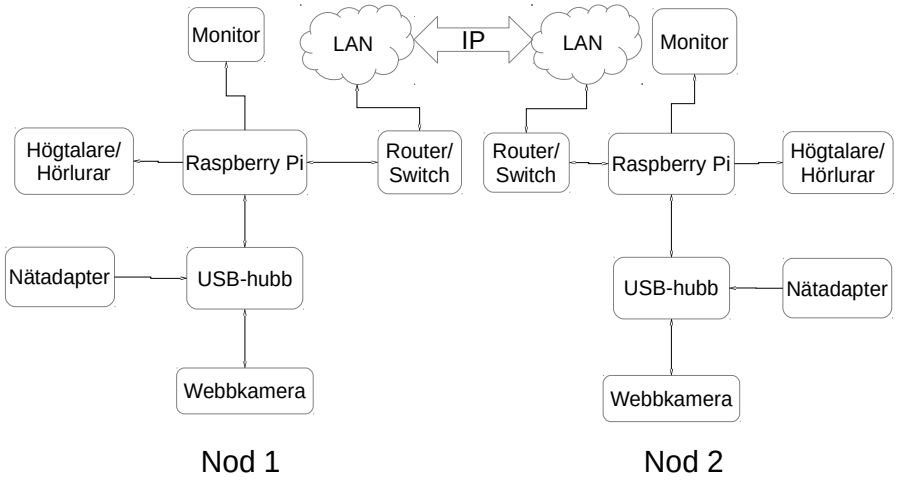
Från början experimenterade jag förutsättningslöst med olika alternativ för att lösa uppgiften. Jag testade flera olika operativsystem på Raspberry Pi (Raspbian, ArchLinux, RiscOS m.fl.). Slutligen kom jag fram till att det var bäst att använda Raspbian, mestadels tack vare detta operativsystems stora stöd från både användarna av Raspberry Pi och från systemets upphovspersoner. Det finns många nätforum där du kan hitta intressanta diskussioner om både Raspbian och Raspberry Pi. [2]

Under mina tester har jag använt ett antal olika metoder för att överföra data mellan noderna. Programmet NetCat (nc) kan överföra godtyckligt data mellan datorer i ett nätverk och GStreamer används för att överföra audio- och videodata. Audioöverföringen kan i sin tur använda olika format, exempelvis A-law-kodat data eller rent PCM-format.

Jag har i olika sammansättningar via kommandofiler använt mig av dessa verktyg. Jag redovisar här en del av de kombinationer av verktygen jag använt i de olika systemkonfigurationerna. Vid samtliga dessa tester har jag använt mig av kommandofiler (shellscript) för bash i Linux. En del av dessa finns redovisade i bilaga A.

4.1 A – 1 Rpi per nod, webbkamera

I detta fall användes en (1) Raspberry Pi per nod. En översikt över komponenterna visas i figur 5.



Figur 5: En Raspberry Pi per nod med webbkamera via USB-porten.

Pi1 sänder och tar emot och pi2 sänder och tar emot. Både bild och ljud använder netcat, vilket innebär att mottagarkommandot bör startas först.

I detta fall använder jag följande kommandofiler på systemen.

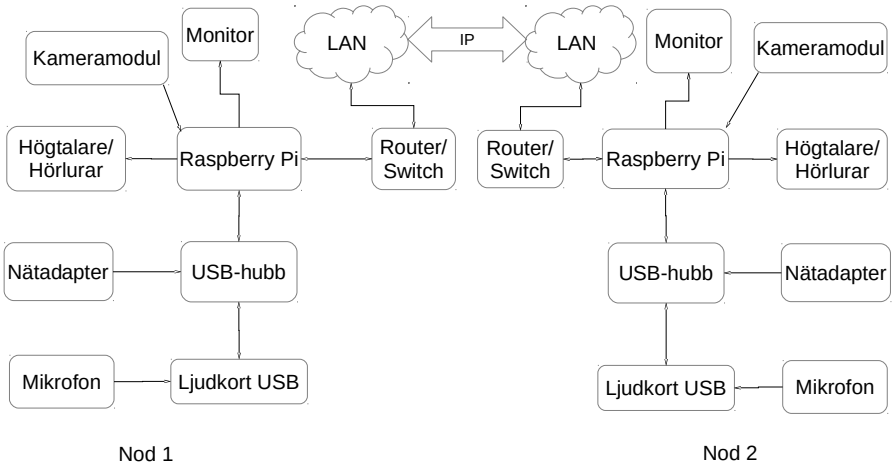
I mappen /home/pi/scripts/
Kör på pi1
./tx_video_c920.sh 1280 720 24 1000 raspbianpi2.local 5001
./tx_audio_c920.sh 2 usbaudio 32000 raspbianpi2.local 6001
./rx_video.sh 5002
./rx_audio.sh 6002

Kör på pi2
./tx_video_c920.sh 1280 720 24 1000 raspbianpi1.local 5002
./tx_audio_c920.sh 2 usbaudio 32000 raspbianpi1.local 6002
./rx_video.sh 5001
./rx_audio.sh 6001

Detta innebär att jag använder mig av video med storleken 1280x720, 24 bilder per sekund och bithastigheten 1000 kbps, samt audio med samplingsfrekvensen 32 kHz, stereo.

4.2 B – 1 Rpi per nod, RaspiCam och USB-mikrofon

I detta fall användes en (1) Raspberry Pi per nod. En översikt över komponenterna visas i figur 6.



Figur 6: En Raspberry Pi per nod med kameramodul och separat mikrofon.

Pi1 sänder och tar emot och pi2 sänder och tar emot. Ljudet är i mono (A-law) och bilden är i valfritt format. Både ljud och bild använder GStreamer. Bilden använder även programmet *raspivid*.

I detta fall använder jag följande kommandofiler på systemen.

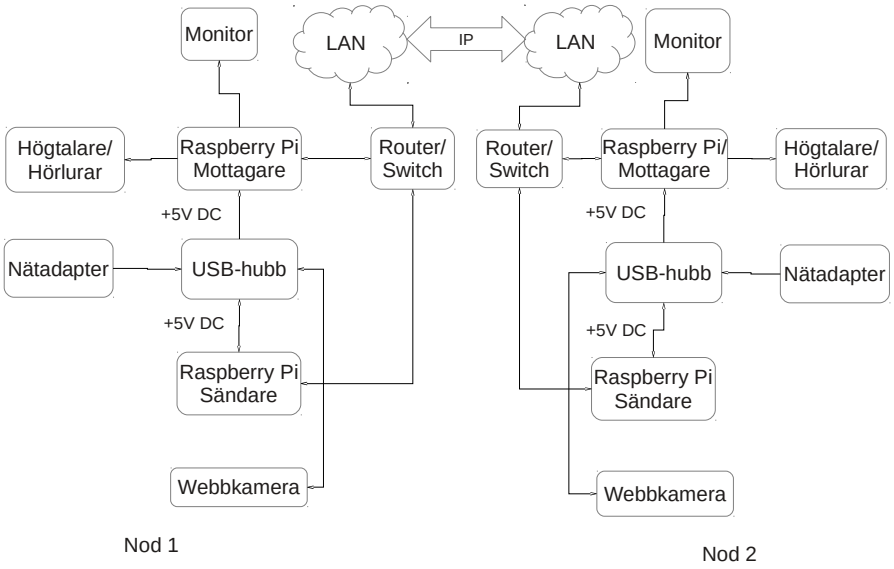
I mappen /home/pi/scripts/
Kör på pi1
./tx_video_gst_rpicam.sh 640 360 25 500000 raspbianpi2.local 5001
./tx_audio_gst_alaw.sh usbaudio raspbianpi2.local 6001
./rx_video_gst_rpicam.sh 5002
./rx_audio_gst_alaw.sh 6002

Kör på pi2
./tx_video_gst_rpicam.sh 640 360 25 500000 raspbianpi1.local 5002
./tx_audio_gst_alaw.sh usbaudio raspbianpi1.local 6002
./rx_video_gst_rpicam.sh 5001
./rx_audio_gst_alaw.sh 6001

Detta innebär att jag använder mig av video med storleken 640x360, 25 bilder per sekund och bithastigheten 500 kbps, samt audio med samplingsfrekvensen 8 kHz, mono, kodat med A-lawkompression.

4.3 AA – 2 Rpi per nod, webbkamera

I detta fall användes två Raspberry Pi per nod. En översikt över komponenterna visas i figur 7.



Figur 7: Två st. Raspberry Pi per nod med webbkamera via USB-porten.

Vid nod 1 finns Pi1 och Pi3. Vid nod 2 finns Pi2 och Pi4. Pi1 sänder, Pi3 tar emot, Pi2 sänder och Pi4 tar emot. Eventuellt bör webbkameran kopplas direkt till USB-porten på Pi1(Pi2 för nod 2) och den andra av USB-portarna på Raspberry Pi kopplas till USB-hubben.

Var observant på ljudkortens inställningar i programmet *alsamixer*; se till att inspelningsvolymen inte är på noll. Både ljud och bild använder GStreamer.

I detta fall använder jag följande kommandofiler på systemen.

I mappen /home/pi/scripts/
Kör på pi1
./tx_video_gst_c920.sh 5 500000 raspbianpi4.local 5001
./tx_audio_gst_alaw.sh usbaudio raspbianpi4.local 6001

Kör på pi2
./tx_video_gst_c920.sh 5 500000 raspbianpi3.local 5001
./tx_audio_gst_alaw.sh usbaudio raspbianpi3.local 6001

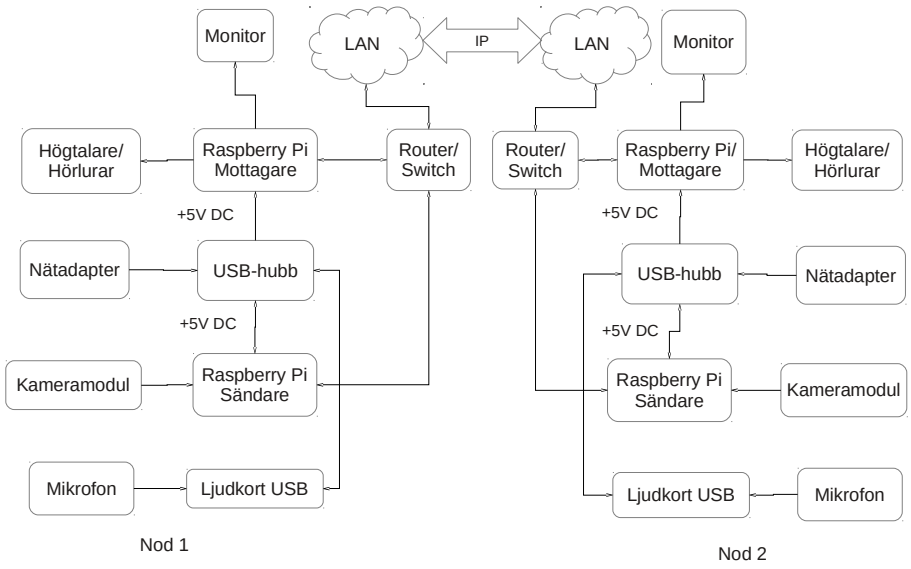
Kör på pi3
./rx_video_gst_c920_1280.sh 5001
./rx_audio_gst_alaw.sh 6001

Kör på pi4
./rx_video_gst_c920_1280.sh 5001
./rx_audio_gst_alaw.sh 6001

Ljudet är i mono, 8 kHz, komprimerat enligt A-law och bilden är i formatet 1280x720, 24 bilder per sekund, kodat med H264 och bithastigheten 0,5 Mbps.

4.4 BB – 2 Rpi per nod, RaspiCam och USB-mikrofon

I detta fall användes två Raspberry Pi per nod, kameramodulen och en separat USB-mikrofon. En översikt över komponenterna visas i figur 8.



Figur 8: Två st. Raspberry Pi per nod med kameramodul och extern mikrofon.

Vid nod 1 finns pi1 och pi3. Vid nod 2 finns pi2 och pi4. Pi1 sänder, pi3 tar emot, pi2 sänder och pi4 tar emot. Var observant på ljudkortens inställningar i programmet *alsamixer*; se till att inspelningsvolymen inte är på noll. Både ljud och bild använder GStreamer.

I detta fall använder jag följande kommandofiler på systemen.

I mappen /home/pi/scripts/
Kör på pi1
./tx_video_gst_rpicam.sh 1280 720 25 1000000 raspbianpi4.local 5001
./tx_audio_gst_alaw.sh usbaudio raspbianpi4.local 6001

Kör på pi2
./tx_video_gst_rpicam.sh 1280 720 25 1000000 raspbianpi3.local 5001
./tx_audio_gst_alaw.sh usbaudio raspbianpi3.local 6001

Kör på pi3
./rx_video_gst_rpicam.sh 5001
./rx_audio_gst_alaw.sh 6001

Kör på pi4
./rx_video_gst_rpicam.sh 5001
./rx_audio_gst_alaw.sh 6001.

Ljudet är i mono (A-law) och bilden är i storleken 1280x720, 25 bilder per sekund och en med bithastighet på 1Mbps. Både ljud och bild använder GStreamer. Bilden använder även programmet *raspivid*.

4.5 C – 1 Mini-PC per nod, webbkamera

I detta fall användes en (1) Zotac mini-PC per nod. En översikt över komponenterna visas i figur 9. Detta alternativ erbjuder en lösning med minimalt antal hårdvarukomponenter. En av anledningarna är att Zotac mini-PC till skillnad från Raspberry Pi har 4 st USB-portar, vilket gör att en extern USB-hubb är överflödig. I skrivande stund har dock en reviderad version av Raspberry Pi utkommit, vilken har just 4 st USB-portar, men den har jag inte använt mig av i det här examensarbetet.

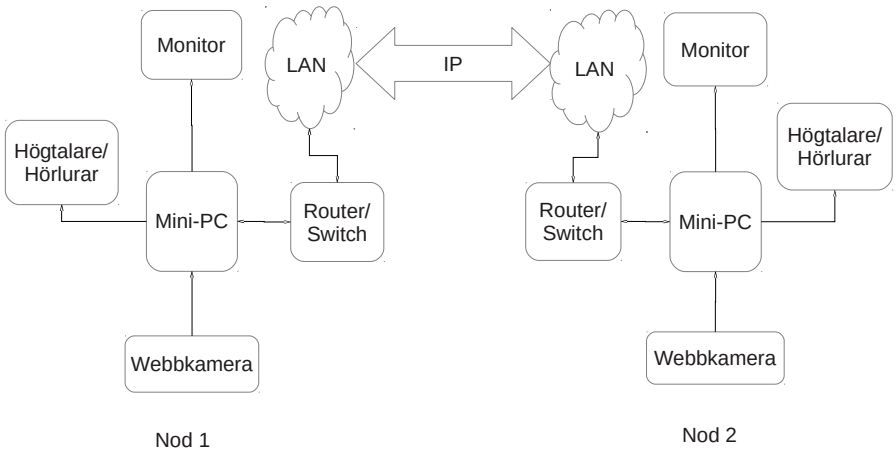
4.5.1 Hårdvara

Varje nod består av följande hårdvara:

- 1 st. Zotac mini-PC (Zotac Zbox Nano plus ID 61) med 2 st processorkärnor (Intel Celeron ULV-867, 1.3 GHz), 2GB RAM, 300 GB HD, Intel HM65 Express chipset och GPU är Intel HD Graphics.
- 1 st. webbkamera Logitech C920 med inbyggd stereomikrofon, max 32 kHz samplingsfrekvens och inbyggd H264-komprimering.
- 1 st. aktiv högtalare med 3,5 mm telekontakt.
- 1 st. 22” LED-monitor.
- 1 st. router/switch för IP-nätverk.

4.5.2 Mjukvara

Datorn har Linuxdistributionen Debian(Wheezy) installerad. Jag har sedan installerat GStreamer version 1.2 från källkoden, kompilerad med hjälp av GNU:s C-kompilator. Kamerastyrningsprogrammet är utvecklat i CodeBlocks och skrivet i C. Själva kommunikationsprogrammet använder sig av rutinerna i GStreamer:s API för C/C++ och GTK+ för grafikhanteringen. För att knyta ihop allt använder jag även skript skrivna för bash.



Figur 9: En Zotac mini-PC per nod med webbkamera via USB-porten.

4.5.3 Övergripande funktion

Enligt kravspecifikationen ska systemet vara lättanvänt och noderna ska koppla upp sig automatiskt mot varandra. Gränssnittet ska vara rent och det enda som i grunden ska behöva ställas in är eventuellt nivån hos ljudet. Detta görs lämpligast med volymratten på de aktiva högtalarna.

När datorn startar går en sekvens av skript igång för att till slut starta både sändare och mottagare i varje nod. Eftersom ett program skrivet för fönstermiljö med hjälp av GTK+ normalt körs i ett fönster har jag i koden för mottagarprogrammet sett till att det exekverar i fullskärmsläge, samtidigt som muspekaren har tagits bort. Eftersom det också är meningen att noderna ska vara ständigt uppkopplade, har jag i ett skript stängt av skärmsläckaren.

4.5.4 Ekoutsläckningen

I ett första skede var den här lösningen gjord på likartat sätt som alla de tidigare beskrivna lösningarna med Raspberry Pi, när det gäller ljudet – d.v.s. personer som kommunicerade använde sig av hörlurar. Om hörlurar inte användes blev det problem med rundgång och ljudfördröjningar. Den slutliga varianten av denna lösning med mini-PC innefattar att jag använder en möjlighet i PulseAudio-systemet att använda dess inbyggda funktion för ekoutsläckning. Den kan anropas från GStreamer. Ett problem med detta är bara att för att fungera optimalt bör audiokommunikationen använda sig av in- och utgångarna på samma fysiska ljudkort. Detta utgör i detta fall ett dilemma, eftersom webbkameran inte har någon högtalare. Då skulle en lösning kunna vara att använda en annan separat mikrofon, vilken kopplas till ingången på det ljudkort som finns i Zotac mini-PC. Den lösningen försökte jag med, men jag ansåg att den audiokvalitet jag då uppnådde var sämre än den kvalitet jag erhöll genom att göra på ett annat

sätt. En anledning kan ha varit den relativt billiga mikrofonen – det kvarstår att undersöka för framtiden.

Anledningen till att man bör använda samma fysiska ljudkort för både in- och utgång är att ekoutsläckningsalgoritmen är känslig för synkroniseringen i audiodatat. De empiriska testerna visade dock att i början av kommunikationen, när noderna varit sammankopplade en kort tid, fungerade ekoutsläckningen bra, även om jag använde den inbyggda mikrofonen i webbkameran och den ordinarie audioutgången på Zotac mini-PC. Efter några minuter försämrades dock synkroniseringen, men efter en återstart av uppkopplingen fungerade det återigen.

Detta sammantaget ledde till att jag valde att göra på följande sätt. Jag har separerat audio- och videodatat i två skilda GStreamer-uppkopplingar. Videouppkopplingen är ständigt aktiverad, men audiouppkopplingen startar jag om var 10:e minut med hjälp av programmet *crontab*. Den valda tiden är endast experimentellt fastställd, men tycks fungera bra. Omstarten av audiouppkopplingen sker mycket snabbt, så det ger inte några märkbara störningar i ljudet. På detta sätt kan det finnas en risk för sämre synkronisering mellan ljudet och bilden, men mina tester visar att det i ett normalsnabbt nätverk (100Mbps) inte medför några synkroniseringsproblem. Hela startsekvensen ser ut enligt nedan:

/home/pi/**restart_communication.sh**

/home/pi/**restart_audio.sh**

/home/pi/**restart_pulsesrc.sh**

```
gst-launch-1.0 -v pulsesrc device=noechosrc stream-properties="props,filter.want=echo-cancel,media.role=phone" ! audioconvert ! lamemp3enc bitrate=128 ! udpsink port=6001 host=pcx.local
```

/home/pi/**restart_pulsesink.sh**

```
gst-launch-1.0 -v udpsrc port=6002 ! mad ! pulsesink device=noechosink stream-properties="props,filter.want=echo-cancel,media.role=phone"
```

/home/pi/**send_and_receive_noaudio.sh**

```
/home/pi/src/consat/codeblocks/production/gst-sender-preview-noaudio/gst-sender-preview-noaudio

/home/pi/src/consat/codeblocks/production/gst-receiver-preview-noaudio/gst-receiver-preview-noaudio
```

/home/pi/**set_camera_defaults.sh**

5 Resultat

Att mäta upp fördröjningar i ljud- och bild är inte helt lätt, så jag har hållit mig till de subjektiva intrycken, men för att mäta upp CPU-belastning och minnesanvändning har jag använt mig av programmet *htop*. För att mäta nätverksbelastningen har jag använt programmet *ifstat*. Båda dessa program är tillgängliga via Raspbians pakethanterare apt-get.

Givna prisuppgifter är ungefärliga, inklusive moms och gäller endast för komponenterna, inte arbetstid för genomförande av experimenten. Komponenter såsom switchar och routrar, vilka kan hänföras till nätverket, är inte heller medtagna i totalpriset. En mer detaljerad prisspecifikation finns i bilaga B.

Här följer en sammanfattning av de resultat jag erhållit med de tidigare beskrivna systemkonfigurationerna och kvalitetskriterierna från metodkapitlet.

5.1 A - 1 Raspberry Pi per nod, webbkamera

Tabell 1: Resultat för alternativ A

CPU-belastning	46,7%,
Minnesanvändning	53/374 MB
Nätverksbelastning	130 kB/s, 500kB/s,
Fördröjningar	ca 500 ms
Synkronisering	God
Ljudet	En del störningar och knatter
Bilden	Låg bildfrekvens, ca 5-10 fps, ”pixlade” partier tydliga vid rörelser
Upplevd totalkvalitet	Inte bra, ger inte någon närvarokänsla
Stabilitet	Uppkopplingen är inte stabil. Efter någon minut ger den ena Raspberry Pi upp och måste startas om.
Pris	C:a 16 200 kr

5.2 B - 1 Raspberry Pi per nod, RaspiCAM och USB-mikrofon

Tabell 2: Resultat för alternativ B

CPU-belastning	100%
Minnesanvändning	57/374MB
Nätverksbelastning	12 kB/s, 70 kB/s
Fördröjningar	Stora, > 1 s
Synkronisering	Dålig
Ljudet	Brummar och knastrar
Bilden	Skaplig, men ganska grynig (låg upplösning, ej HD)
Upplevd totalkvalitet	Ej bra
Stabilitet	Låg, förbindelsen bryts efter ett tag.
Pris	C:a 14 700 kr

5.3 AA - 2 Raspberry Pi per nod, webbkamera

Tabell 3: Resultat för alternativ AA

CPU-belastning	78%
Minnesanvändning	53/374MB
Nätverksbelastning	130 kB/s, 250 kB/s
Fördröjningar	Låg till en början, men sedan varierande.
Synkronisering	God
Ljudet	Ok, men något diskantfattigt
Bilden	Lite grynig, men ibland förekommer ”pixlade” partier, främst vid rörelse.
Upplevd totalkvalitet	Skaplig, men inte närvarokänsla
Stabilitet	God
Pris	C:a 17 000 kr

5.4 BB - 2 Raspberry Pi per nod, RaspiCAM och USB-mikrofon

Tabell 4: Resultat för alternativ BB

CPU-belastning	70%
Minnesanvändning	45/374MB
Nätverksbelastning	11 kB/s, 120 kB/s
Fördröjningar	Initialt låg fördröjning för både ljud och bild, men efter ett tag ökar bildfördröjningen.
Synkronisering	Bra (initialt).
Ljudet	Ganska mycket brum och knäppar
Bilden	Något grynig, men följsam, d.v.s. hög uppdateringsfrekvens. Stabil
Upplevd totalkvalitet	Ok, men varken HD-kvalitet eller närvarokänsla
Stabilitet	God
Pris	C:a 16 200 kr

5.5 C - 1 Zotac Mini-PC per nod, webbkamera

Tabell 5: Resultat för alternativ C

CPU-belastning	C:a 40% per processorkärna. (2 st kärnor)
Minnesanvändning	265/1852 MB
Nätverksbelastning	140 kB/s
Fördröjningar	Total fördröjning på c:a 300-400 ms
Synkronisering	Utmärkt
Ljudet	Acceptabelt, men en del störningar när ekoutsläckningsalgoritmen arbetar.
Bilden	Bra, med endast små färgskiftningar på grund av komprimeringen.
Upplevd totalkvalitet	Mycket bra, på gränsen till närvarokänsla
Stabilitet	Utmärkt
Pris	C:a 20 200 kr

6 Slutsatser

En jämförelse av resultaten av de olika systemkonfigurationerna leder till slutsatsen att det finns endast en av de testade konfigurationerna som är möjlig att använda ut teknisk synvinkel. Det är konfiguration C, med mini-PC och webbkamera.

Om kvalitetskravet hade varit något lägre skulle alternativ BB kunnat vara användbart, men efter diskussioner med handledarna på företaget drogs slutsatsen att det totalt sett mest lämpade alternativet i nuläget var att använda alternativ C, d.v.s. en mini-PC per nod, med tillhörande webbkamera. Kostnaderna för de olika alternativen skiljer sig inte så mycket - alternativ C är dock det dyraste.

Eftersom teknikutvecklingen går i rasande fart är det inte helt lätt att fatta långsiktiga beslut. När examensarbetet i stort sett var genomfört släppte organisationen bakom Raspberry Pi en uppdaterad version (Raspberry Pi version 2), som bygger på mer avancerad hårdvara (ARM v7) än den Raspberry Pi som använts för att genomföra detta examensarbete. Det är fullt tänkbart att den nya versionen av Raspberry Pi skulle ge ett annat slutresultat, men det får bli en möjlighet för framtida examensarbetare att undersöka.

6.1 Reflektioner kring arbetet

Under detta arbete har jag undersökt många olika alternativ och läst igenom otaliga beskrivningar och instruktioner på en mängd webbplatser. Jag har installerat och ominstallerat olika operativsystem på de Raspberry Pi som jag använt. Jag har fört anteckningar över hur jag installerat operativsystemen och vilka komponenter jag installerat. Arbetet har också dragit ut på tiden på grund av att jag av ekonomiska skäl även nödgats att lönearbeta samtidigt som jag arbetat med examensarbetet.

Sammantaget har jag använt och testat massor av material och det är nära nog en omöjlighet att i pappersform redovisa allt material jag skapat och alla referenser jag använt. Den föreliggande rapporten tar upp de viktigaste delarna, men för att vara mer komplett skulle arbetet behöva redovisas i digital form istället, exempelvis i form av en webbplats, där alla filer och länkar kan läggas upp. Kanske blir det sättet att redovisa en realitet för kommande studentgenerationer.

6.2 Val av API

Det finns många API:er att välja mellan när det gäller att programmera grafikprocessorn (EGL, OpenMAX, OpenGL ES, OpenVG) och jag testade några av de exempel som medföljer Raspberry Pi (/opt/vc), men dokumentationen var enligt mig bristfällig och svår att förstå.

Även om det kanske skulle gå att få bättre prestanda med något annat alternativ, anser jag att fördelarna med GStreamer gjorde det till det bästa valet i detta fall.

6.3 Säkerhet – kryptering

I dagens läge när så mycket av samhällets tjänster är beroende av Internet är det av stor betydelse att all kommunikationen görs så säker som möjligt, för att undvika att information kommer i felaktiga händer. Under arbetet med detta system har inga resurser lagts på säkerheten. Anledningen är att för detta projekt fanns inte de medel i form av tid och pengar som skulle krävts för detta. Att införa någon slags kryptering av dataöverföringen är en möjlig framtida förbättring av systemet.

Referenser

[1] Raspbian, ”Raspbians hemsida”,
http://www.raspbian.org/FrontPage/
Hämtad 2015-01-03.

[2] Diskussionsforum för Raspberry Pi, ”Raspberry Pi, forums”,
https://www.raspberrypi.org/forums/
Hämtad 2015-07-31.

[3] GStreamer, ”Gstreamer, Freedesktop”,
http://gstreamer.freedesktop.org/
Hämtad 2015-01-03.

[4] CodeBlocks, ”CodeBlocks IDE”,
http://www.codeblocks.org/
Hämtad 2015-01-03.

[5] W. Taymans, S. Baker, A. Wingo, R.S. Bultje, S. Kost, 2006: *GStreamer Application Development Manual (1.1.2.1)*

[6] Installera GStreamer, ”Gstreamer, Installing on Linux”,
http://docs.gstreamer.com/display/GstSDK/Installing+on+Linux/
Hämtad 2015-01-04.

[7] M. Richardsson, S. Wallace, *Getting Started with Raspberry Pi*. 1:a uppl. Sebastopol: O'Reilly, 2012

[8] Skypekit, ”Skypekit discontinued”,
https://support.skype.com/en/faq/FA12322/is-skypekit-being-discontinued/
Hämtad 2015-07-31.

[9] H. Gulliksson, J. Lindström, *Ljud och bild över nätverk,* 3:e uppl., s.41-48, Lund: Studentlitteratur, 2004

[10] Olika audioformat, ”Sustainability of Digital Formats”,
http://www.digitalpreservation.gov/formats/fdd/browse_list.shtml
Hämtad 2015-08-01.

[11] Olika videoformat, ”Format Descriptions for Moving Images”,
http://www.digitalpreservation.gov/formats/fdd/video_fdd.shtml
Hämtad 2015-08-01.

[12] T. Wiegand, G.J. Sullivan, G. Bjontegaard, A.Luthra, ”Overview of the H.264/AVC video coding standard”, *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 13, nr. 7, 2003, pp. 560 - 576, IEEE Journals & Magazines

[13] RFC3550, RTP-protokollet, ”RTP: A Transport Protocol for Real-Time Applications”,
https://tools.ietf.org/html/rfc3550/
Hämtad 2015-08-01.

[14] Raspberry Pi, ”Raspberry Pi Foundation”,
https://www.raspberrypi.org/
Hämtad 2015-08-01.

[15] Skype, ”Skype”,
http://www.skype.com/sv/
Hämtad 2015-08-01.

[16] WebRTC, ”WebRTC”,
http://www.webrtc.org/
Hämtad 2015-08-01.

[17] Kommunikation via webbläsare, ”AppRTC”,
https://apprtc.appspot.com/
Hämtad 2015-08-01.

Bilaga A:
Dokumentation av programkod

Kommandofiler

tx_video_c920.sh

```
#!/bin/bash
#
# tx_video_c920.sh
#
# roger.sundh@musiktronik.se
# 2013-10-08
#
# Sänder ut video i H264-format från webbkameran C920 (Logitech)
# Använder capture, netcat (nc), v4l2-ctl och UDP.
# Programmet capture (/usr/local/bin/capture) är anpassat efter en bearbetning
av ett exempelprogram
# som medföljer paketet video4linux2 (v4l2).
# v4l2-ctl finns i paketet v4l-utils
#
# Om miljövariabeln RS_VERBOSE är definierad
# visas utförlig information från nc.
#
# Inparameter - $1, bildens bredd, exv 1280
#           $2, bildens höjd, exv 720
#           $3, bildfrekvensen, exv 30
#           $4, bithastigheten för H264 i kbps, exv 3000
#           $5, mottagarens IP-adress, exv 192.168.0.231
#           $6, mottagarens UDP-port, exv 5001
#

if [ $# -ne 6 ]; then
    echo "Usage: $0  <width> <height> <frame_rate> <bit_rate_kbps> <recv_ip>
<recv_udp_port>"
    echo "Example: $0 1280 720 30 3000 192.168.0.231 5001"
    exit 0
fi

echo "Sending video of size $1x$2, $3 fps from C920 to $5 at port $6 using
bitrate $4 kbps"

#H264-format
v4l2-ctl --set-fmt-video=width=$1,height=$2,pixelformat=1

#Framerate
v4l2-ctl --set-parm=$3

if [ $RS_VERBOSE ]; then
      capture -b $4 -c -1 -o | nc -v -u $5 $6

    else

      capture -b $4 -c -1 -o | nc -u $5 $6
fi
```

tx_audio_c920.sh

```
#!/bin/bash
#
# tx_audio_c920.sh
#
# roger.sundh@musiktronik.se
# 2014-03-01
#
# Sänder ut audio i 16 bitars PCM till angiven mottagare via UDP
# Använder ALSA-programmet arecord och netcat.
#
# Om miljövariabeln RS_VERBOSE är definierad
# visas utförlig information från arecord och nc.
#
# Inparameter - $1, antal kanaler att sända (1 eller 2)
#           $2, audioenhet (ingång), exv hw:1,0 eller usbaudio eller webcam
#           $3, samplingsfrekvens, exv 48000
#           $4, mottagarens IP-adress, exv 192.168.0.231
#           $5, mottagarens UDP-port, exv 6001
#
# arecord  -c 1 -f S16_LE -D hw:1,0 -r 48000 | nc -u 192.168.0.232 6001

if [ $# -ne 5 ]; then
    echo "Usage: ./tx_audio_c920.sh  <channels> <audio_card> <sampling_rate>
<recv_ip> <recv_udp_port>"
    echo "Example: ./tx_audio_c920.sh 2 webcam 32000 raspbianpi4.local 6001"
    exit 0
fi

if [ "$1" -eq 1 ]; then
    echo "Sending $1 audio channel from $2 using PCM $3 Hz to $4 at port $5"
  else
    echo "Sending $1 audio channels from $2 using PCM $3 Hz to $4 at port $5"
fi

if [ $RS_VERBOSE ]; then

      arecord -v -c $1 -f S16_LE -D $2 -r $2 | nc -v -u $4 $5

    else

      arecord -c $1 -f S16_LE -D $2 -r $3 | nc -u $4 $5

fi
```

rx_video.sh

```
#!/bin/bash
#
# rx_video.sh
#
# roger.sundh@musiktronik.se
# 2013-09-29
#
# Tar emot en videoström via UDP och spelar upp på
# systemets videoutgång (HDMI). Använder netcat (nc)
# Videodatat mellanlagras i FIFO-kön "buffer" och skickas
# sedan vidare till applikationen hello_video.bin, vilken
# spelar upp videon på ansluten utgång. Programmet hello_video.bin
# ingår som ett demonstrationsprogram för OpenMAX, distribuerat och
# skrivet av personal på Broadcom Inc.
#
# Om miljövariabeln RS_VERBOSE är definierad
# visas utförlig information från netcat.
#
# Inparameter - $1, port för inkommande UDP-data
#

if [ $# -ne 1 ]; then
    echo "Usage: $0 <recv_udp_port>"
    exit 0
fi

echo "Receiving video via UDP on port $1"

if [ $RS_VERBOSE ]; then

      nc -v -u -l -p $1 > /home/pi/buffer |
/opt/vc/src/hello_pi/hello_video/hello_video.bin /home/pi/buffer

    else

      nc -u -l -p $1 > /home/pi/buffer |
/opt/vc/src/hello_pi/hello_video/hello_video.bin /home/pi/buffer

fi
```

rx_audio.sh

```
#!/bin/bash
#
# rx_audio.sh
#
# roger.sundh@musiktronik.se
# 2013-09-29
#
# Tar emot en ljudström via UDP och spelar upp på
# systemets standardljudutgång. Använder netcat (nc)
# och aplay från ALSA-projektet.
#
# Inparameter - $1, port för inkommande UDP-data
#

if [ $# -ne 1 ]; then
    echo "Usage: $0 <recv_udp_port>"
    exit 0
fi

echo "Receiving audio on port $1"
```

```
if [ $RS_VERBOSE ]; then

      nc -v -u -l -p $1 | aplay -v

    else

      nc -u -l -p $1 | aplay

fi
```

tx_video_gst_rpicam.sh

```
#!/bin/bash
#
# tx_video_gst_rpicam.sh
#
# roger.sundh@musiktronik.se
# 2013-10-01
#
# Sänder ut video i H264-format från ansluten RPiCam
# Använder GStreamer-1.0 och RTP (UDP).
#
# Om miljövariabeln RS_VERBOSE är definierad
# visas utförlig information från GStreamer.
#
# Inparameter - $1, bildens bredd, exv 1280
#           $2, bildens höjd, exv 720
#           $3, bildfrekvensen, exv 25
#           $4, bithastigheten för H264, exv 2000000
#           $5, mottagarens IP-adress, exv 192.168.0.231
#           $6, mottagarens UDP-port, exv 5001
#

if [ $# -ne 6 ]; then
    echo "Usage: $0  <width> <height> <frame_rate> <bit_rate> <recv_ip>
<recv_udp_port>"
    echo "Example: $0 1280 720 25 2000000 192.168.0.231 5001"
    exit 0
fi

echo "Sending video of size $1x$2, $3 fps from RPiCam to $5 at port $6 using
bitrate $4 bps"

if [ $RS_VERBOSE ]; then

#RpiCam
raspivid -n -t 0 -w $1 -h $2 -fps $3 -b $4 -o - | gst-launch-1.0 -v fdsrc !
h264parse ! queue ! rtph264pay ! udpsink host=$5 port=$6

    else

#RpiCam
raspivid -n -t 0 -w $1 -h $2 -fps $3 -b $4 -o - | gst-launch-1.0 fdsrc ! h264-
parse ! queue ! rtph264pay ! udpsink host=$5 port=$6

fi
```

tx_audio_gst_alaw.sh

```
#!/bin/bash
#
# tx_audio_gst_alaw.sh
#
# roger.sundh@musiktronik.se
# 2013-10-01
#
# Sänder ut audio i 16 bitars A-LAW-kodat PCM till angiven mottagare via UDP
# Använder GStreamer-1.0. Måste använda ljudkortet plughw:x,y
#
# Om miljövariabeln RS_VERBOSE är definierad
# visas utförlig information från GStreamer.
#
# Inparameter - $1, audioenhet (ingång), exv plughw:1,0
#           $3, mottagarens IP-adress, exv 192.168.0.231
#           $4, mottagarens UDP-port, exv 6001
#

if [ $# -ne 3 ]; then
    echo "Usage: $0  <audio_card> <recv_ip> <recv_udp_port>"
    echo "Example: $0 plughw:1,0 192.168.0.231 6001"
    exit 0
fi

echo "Sending mono audio from $1 using A-LAW to $2 at port $3"

if [ $RS_VERBOSE ]; then

#RTP A-law
gst-launch-1.0 -v alsasrc device=$1 ! queue ! audioresample ! audioconvert !
alawenc ! rtppcmapay  ! udpsink host=$2 port=$3

    else

#RTP A-law
gst-launch-1.0 alsasrc device=$1 ! queue ! audioresample ! audioconvert !
alawenc ! rtppcmapay  ! udpsink host=$2 port=$3

fi
```

rx_video_gst_rpicam.sh

```
#!/bin/bash
#
# rx_video_gst_rpicam.sh
#
# roger.sundh@musiktronik.se
# 2013-09-29
#
# Tar emot en videoström via RTP (UDP) och spelar upp på
# systemets videoutgång (HDMI). Använder GStreamer-1.0
#
# Om miljövariabeln RS_VERBOSE är definierad
# visas utförlig information från GStreamer.
#
# Inparameter - $1, port för inkommande UDP-data
#

if [ $# -ne 1 ]; then
    echo "Usage: $0 <recv_udp_port>"
    exit 0
fi
```

```
echo "Receiving video via UDP on port $1"

if [ $RS_VERBOSE ]; then

#WebCam - RTP (384x216)
gst-launch-1.0 -vvv udpsrc port=$1 caps = 'application/x-rtp, media=(string)vi-
deo, clock-rate=(int)90000, encoding-name=(string)H264, sprop-parameter-
sets=(string)"J2QADawrQMDvywDxImo\=\,KO4CXLA\=", payload=(int)96' ! rtph264de-
pay ! queue ! decodebin ! eglglessink sync=false

    else

#WebCam - RTP (384x216)
gst-launch-1.0 udpsrc port=$1 caps = 'application/x-rtp, media=(string)video,
clock-rate=(int)90000, encoding-name=(string)H264, sprop-parameter-
sets=(string)"J2QADawrQMDvywDxImo\=\,KO4CXLA\=", payload=(int)96' ! rtph264de-
pay ! queue ! decodebin ! eglglessink sync=false

fi
```

rx_audio_gst_alaw.sh

```
#!/bin/bash
#
# rx_audio_gst_alaw.sh
#
# roger.sundh@musiktronik.se
# 2013-09-29
#
# Tar emot en ljudström via RTP (UDP) och spelar upp på
# systemets standardljudutgång. Använder GStreamer-1.0
# Audiodatat är kodat i alaw, 8kHz, 16 bitar, mono.
#
# Om miljövariabeln RS_VERBOSE är definierad
# visas utförlig information från GStreamer.
#
# Inparameter - $1, port för inkommande UDP-data
#

if [ $# -ne 1 ]; then
    echo "Usage: $0 <recv_udp_port>"
    exit 0
fi

echo "Receiving alaw-audio via RTP on port $1"

if [ $RS_VERBOSE ]; then

#RTP A-law
gst-launch-1.0 -v udpsrc port=$1 caps = 'application/x-rtp, media=(string)au-
dio, clock-rate=(int)8000, encoding-name=(string)PCMA, payload=(int)8' ! rtppc-
madepay ! queue ! decodebin ! audioconvert ! alsasink device=default sync=false

    else

#RTP A-law
gst-launch-1.0 udpsrc port=$1 caps = 'application/x-rtp, media=(string)audio,
clock-rate=(int)8000, encoding-name=(string)PCMA, payload=(int)8' ! rtppcmade-
pay ! queue ! decodebin ! audioconvert ! alsasink device=default sync=false

fi
```

tx_video_gst_c920.sh

```
#!/bin/bash
#
# tx_video_gst_c920.sh
#
# roger.sundh@musiktronik.se
# 2013-10-08
#
# Sänder ut video i H264-format från webbkameran C920 (Logitech)
# Använder GStreamer, v4l2-ctl och RTP(UDP). Använder fixerad storlek
# Bredd=1280, Höjd=720, Bildfrekvens=24
#
# Om miljövariabeln RS_VERBOSE är definierad
# visas utförlig information från GStreamer.
#
# Inparameter - $1, tid mellan i-frames i sekunder, exv 10
#           $2, bithastighet i bps, exv 1000000
#           $3, mottagarens IP-adress, exv 192.168.0.231
#           $4, mottagarens UDP-port, exv 5001
#

if [ $# -ne 4 ]; then
    echo "Usage: $0  <iframe_period> <bit_rate> <recv_ip> <recv_udp_port>"
    echo "Example: $0 10 1000000 192.168.0.231 5001"
    exit 0
fi


#H264-format
#v4l2-ctl --set-fmt-video=width=1280,height=720,pixelformat=1

#Framerate
#v4l2-ctl --set-parm=24

#/home/pi/scripts/setup_c920.sh 1280 720 30 > /dev/null

echo "Sending video of size 1280x720, 24 fps from C920 to $3 at port $4 using
bitrate $2 bps"

if [ $RS_VERBOSE ]; then
gst-launch-1.0 -v uvch264src device=/dev/video0 name=src auto-start=true  ifra-
me-period=$1 initial-bitrate=$2 src.vidsrc ! queue ! 'video/x-
h264,width=1280,height=720,framerate=24/1' ! h264parse ! rtph264pay ! udpsink
host=$3 port=$4

    else

gst-launch-1.0 uvch264src device=/dev/video0 name=src auto-start=true  iframe-
period=$1 initial-bitrate=$2 src.vidsrc ! queue ! 'video/x-
h264,width=1280,height=720,framerate=24/1' ! h264parse ! rtph264pay ! udpsink
host=$3 port=$4

fi
```

tx_audio_gst_c920_stereo.sh

```
#!/bin/bash
#
# tx_audio_gst_c920_stereo.sh
#
# roger.sundh@musiktronik.se
# 2014-03-01
#
# Sänder ut audio i 16 bitars 32 kHz PCM stereo till angiven mottagare via UDP
# Använder GStreamer-1.0
#
# Om miljövariabeln RS_VERBOSE är definierad
# visas utförlig information från GStreamer.
#
# Inparameter - $1, audioenhet (ingång), exv hw:1,0 eller usbaudio eller webcam
#           $2, mottagarens IP-adress, exv 192.168.0.231
#           $3, mottagarens UDP-port, exv 6001

#gst-launch-1.0 -e -v alsasrc device=hw:1 ! audioconvert ! audio/x-raw, chan-
nels=1, depth=16, #width=16, rate=44100 ! rtpL16pay  ! udpsink
host=192.168.0.234 port=6001

if [ $# -ne 3 ]; then
    echo "Usage: $0  <audio_card> <recv_ip> <recv_udp_port>"
    echo "Example: $0 hw:1,0 192.168.0.231 6001"
    exit 0
fi

echo "Sending audio using 2 channels from $1 using PCM 32000 Hz to $2 at port
$3"

if [ $RS_VERBOSE ]; then

gst-launch-1.0 -v alsasrc device=$1 ! audioconvert ! audio/x-raw, channels=2,
depth=16, width=16, rate=32000 ! rtpL16pay  ! udpsink host=$2 port=$3

    else

gst-launch-1.0 alsasrc device=$1 ! audioconvert ! audio/x-raw, channels=2,
depth=16, width=16, rate=32000 ! rtpL16pay  ! udpsink host=$2 port=$3

fi
```

rx_video_gst_c920_1280.sh

```
#!/bin/bash
#
# rx_video_gst_c920_1280.sh
#
# roger.sundh@musiktronik.se
# 2013-10-08
#
# Tar emot en videoström (1280x720, 24 fps) via RTP (UDP) och spelar upp på
# systemets videoutgång (HDMI). Använder GStreamer-1.0
#
# Om miljövariabeln RS_VERBOSE är definierad
# visas utförlig information från GStreamer.
#
# Inparameter - $1, port för inkommande UDP-data
#

if [ $# -ne 1 ]; then
    echo "Usage: $0 <recv_udp_port>"
    exit 0
fi

echo "Receiving video via UDP on port $1"

if [ $RS_VERBOSE ]; then

#C920 1280x720,24fps
gst-launch-1.0 -v udpsrc port=$1 caps = 'application/x-rtp, media=(string)vi-
deo, clock-rate=(int)90000, encoding-name=(string)H264, sprop-parameter-
sets=(string)"Z2QAKKx2gFAFv/AUSAAAAwAIAAADAYMCAAFuNgAW4973wv-
CIRqA\=\,aO44sA\=\=", payload=(int)96' ! rtph264depay ! decodebin ! autovideo-
sink sync=false

    else

#C920 1280x720,24fps
gst-launch-1.0 udpsrc port=$1 caps = 'application/x-rtp, media=(string)video,
clock-rate=(int)90000, encoding-name=(string)H264, sprop-parameter-
sets=(string)"Z2QAKKx2gFAFv/AUSAAAAwAIAAADAYMCAAFuNgAW4973wv-
CIRqA\=\,aO44sA\=\=", payload=(int)96' ! rtph264depay ! decodebin ! autovideo-
sink sync=false

fi
```

rx_audio_gst_c920_stereo.sh

```
#!/bin/bash
#
# rx_audio_gst_c920_stereo.sh
#
# roger.sundh@musiktronik.se
# 2014-03-01
#
# Tar emot en ljudström via RTP (UDP) och spelar upp på
# systemets standardljudutgång. Använder GStreamer-1.0
# Audiodatat är ren PCM, 32kHz, 16 bitar, stereo, från Logitech C920 webbkame-
ra.
#
# Om miljövariabeln RS_VERBOSE är definierad
# visas utförlig information från GStreamer.
#
# Inparameter - $1, port för inkommande UDP-data
#

if [ $# -ne 1 ]; then
    echo "Usage: $0 <recv_udp_port>"
    exit 0
fi

echo "Receiving audio via RTP on port $1"

if [ $RS_VERBOSE ]; then

gst-launch-1.0 -vvv udpsrc port=$1 caps = 'application/x-rtp,media=(string)au-
dio, clock-rate=(int)32000, width=16, height=16, encoding-name=(string)L16, en-
coding-params=(string)2, channels=(int)2, payload=(int)96' ! rtpL16depay !
queue ! audioconvert ! alsasink device=default sync=false

    else

gst-launch-1.0 udpsrc port=$1 caps = 'application/x-rtp,media=(string)audio,
clock-rate=(int)32000, width=16, height=16, encoding-name=(string)L16, enco-
ding-params=(string)2, channels=(int)2, payload=(int)96' ! rtpL16depay !
queue ! audioconvert ! alsasink device=default sync=false

fi
```

tx_video_gst_rpicam.sh

```
#!/bin/bash
#
# tx_video_gst_rpicam.sh
#
# roger.sundh@musiktronik.se
# 2013-10-01
#
# Sänder ut video i H264-format från ansluten RPiCam
# Använder GStreamer-1.0 och RTP (UDP).
#
# Om miljövariabeln RS_VERBOSE är definierad
# visas utförlig information från GStreamer.
#
# Inparameter - $1, bildens bredd, exv 1280
#           $2, bildens höjd, exv 720
#           $3, bildfrekvensen, exv 25
#           $4, bithastigheten för H264, exv 2000000
#           $5, mottagarens IP-adress, exv 192.168.0.231
#           $6, mottagarens UDP-port, exv 5001
#

if [ $# -ne 6 ]; then
    echo "Usage: $0  <width> <height> <frame_rate> <bit_rate> <recv_ip>
<recv_udp_port>"
    echo "Example: $0 1280 720 25 2000000 192.168.0.231 5001"
    exit 0
fi

echo "Sending video of size $1x$2, $3 fps from RPiCam to $5 at port $6 using
bitrate $4 bps"

if [ $RS_VERBOSE ]; then

#RpiCam 384x216
raspivid -n -t 0 -w $1 -h $2 -fps $3 -b $4 -o - | gst-launch-1.0 -v fdsrc !
h264parse ! queue ! rtph264pay ! udpsink host=$5 port=$6

    else

#RpiCam 384x216
raspivid -n -t 0 -w $1 -h $2 -fps $3 -b $4 -o - | gst-launch-1.0 fdsrc ! h264-
parse ! queue ! rtph264pay ! udpsink host=$5 port=$6

fi
```

tx_audio_gst_alaw.sh

```
#!/bin/bash
#
# tx_audio_gst_alaw.sh
#
# roger.sundh@musiktronik.se
# 2013-10-01
#
# Sänder ut audio i 16 bitars A-LAW kodat PCM till angiven mottagare via UDP
# Använder GStreamer-1.0. Måste använda ljudkortet plughw:x,y
#
# Om miljövariabeln RS_VERBOSE är definierad
# visas utförlig information från GStreamer.
#
# Inparameter - $1, audioenhet (ingång), exv plughw:1,0
#           $3, mottagarens IP-adress, exv 192.168.0.231
#           $4, mottagarens UDP-port, exv 6001
#

if [ $# -ne 3 ]; then
    echo "Usage: $0  <audio_card> <recv_ip> <recv_udp_port>"
    echo "Example: $0 plughw:1,0 192.168.0.231 6001"
    exit 0
fi

echo "Sending mono audio from $1 using A-LAW to $2 at port $3"

if [ $RS_VERBOSE ]; then

#RTP A-law
gst-launch-1.0 -v alsasrc device=$1 ! queue ! audioresample ! audioconvert !
alawenc ! rtppcmapay  ! udpsink host=$2 port=$3

    else

#RTP A-law
gst-launch-1.0 alsasrc device=$1 ! queue ! audioresample ! audioconvert !
alawenc ! rtppcmapay  ! udpsink host=$2 port=$3

fi
```

rx_video_gst_rpicam.sh

```
#!/bin/bash
#
# rx_video_gst_rpicam.sh
#
# roger.sundh@musiktronik.se
# 2013-09-29
#
# Tar emot en videoström via RTP (UDP) och spelar upp på
# systemets videoutgång (HDMI). Använder GStreamer-1.0
#
# Om miljövariabeln RS_VERBOSE är definierad
# visas utförlig information från GStreamer.
#
# Inparameter - $1, port för inkommande UDP-data
#

if [ $# -ne 1 ]; then
    echo "Usage: $0 <recv_udp_port>"
    exit 0
fi

echo "Receiving video via UDP on port $1"

if [ $RS_VERBOSE ]; then

#WebCam - RTP (384x216)
gst-launch-1.0 -vvv udpsrc port=$1 caps = 'application/x-rtp, media=(string)vi-
deo, clock-rate=(int)90000, encoding-name=(string)H264, sprop-parameter-
sets=(string)"J2QADawrQMDvywDxImo\=\,KO4CXLA\=", payload=(int)96' ! rtph264de-
pay ! queue ! decodebin ! eglglessink sync=false

    else

#WebCam - RTP (384x216)
gst-launch-1.0 udpsrc port=$1 caps = 'application/x-rtp, media=(string)video,
clock-rate=(int)90000, encoding-name=(string)H264, sprop-parameter-
sets=(string)"J2QADawrQMDvywDxImo\=\,KO4CXLA\=", payload=(int)96' ! rtph264de-
pay ! queue ! decodebin ! eglglessink sync=false

fi
```

rx_audio_gst_alaw.sh

```
#!/bin/bash
#
# rx_audio_gst_alaw.sh
#
# roger.sundh@musiktronik.se
# 2013-09-29
#
# Tar emot en ljudström via RTP (UDP) och spelar upp på
# systemets standardljudutgång. Använder GStreamer-1.0
# Audiodatat är kodat i alaw, 8kHz, 16 bitar, mono.
#
# Om miljövariabeln RS_VERBOSE är definierad
# visas utförlig information från GStreamer.
#
# Inparameter - $1, port för inkommande UDP-data
#

if [ $# -ne 1 ]; then
    echo "Usage: $0 <recv_udp_port>"
    exit 0
fi

echo "Receiving alaw-audio via RTP on port $1"

if [ $RS_VERBOSE ]; then

#RTP A-law
gst-launch-1.0 -v udpsrc port=$1 caps = 'application/x-rtp, media=(string)au-
dio, clock-rate=(int)8000, encoding-name=(string)PCMA, payload=(int)8' ! rtppc-
madepay ! queue ! decodebin ! audioconvert ! alsasink device=default sync=false

    else

#RTP A-law
gst-launch-1.0 udpsrc port=$1 caps = 'application/x-rtp, media=(string)audio,
clock-rate=(int)8000, encoding-name=(string)PCMA, payload=(int)8' ! rtppcmade-
pay ! queue ! decodebin ! audioconvert ! alsasink device=default sync=false

fi
```

Källkod

GStreamer sändare (gst-sender-preview-noaudio)

```
/* main.cpp
* Roger Sundh
* 2014-08-26
* Kompilera med kommandot
* g++ main.cpp -o gst-sender-preview-noaudio `pkg-config --cflags --libs gs-
treamer-1.0 gtk+-3.0 gstreamer-video-1.0`
*
*/

#include <gst/gst.h>
#include <gst/video/videooverlay.h>
#include <gtk/gtk.h>
#include <gdk/gdkx.h>  // for GDK_WINDOW_XID

static guintptr video_window_handle = 0;

static GstBusSyncReply
bus_sync_handler (GstBus * bus, GstMessage * message, gpointer user_data)
{
// ignore anything but 'prepare-window-handle' element messages
    if (!gst_is_video_overlay_prepare_window_handle_message (message))
        return GST_BUS_PASS;

    if (video_window_handle != 0)
    {
        GstVideoOverlay *overlay;

        // GST_MESSAGE_SRC (message) will be the video sink element
        overlay = GST_VIDEO_OVERLAY (GST_MESSAGE_SRC (message));
        gst_video_overlay_set_window_handle (overlay, video_window_handle);
    }
    else
    {
        g_warning ("Should have obtained video_window_handle by now!");
    }

    gst_message_unref (message);
    return GST_BUS_DROP;
}

static void
video_widget_realize_cb (GtkWidget *widget, gpointer data)
{
#if GTK_CHECK_VERSION(2,18,0)
    // Tell Gtk+/Gdk to create a native window for this widget instead of
    // drawing onto the parent widget.
    // This is here just for pedagogical purposes, GDK_WINDOW_XID will call
    // it as well in newer Gtk versions
    if (!gdk_window_ensure_native (gtk_widget_get_window(widget)))
        g_error ("Couldn't create native window needed for GstVideoOverlay!");
#endif

    gulong xid = GDK_WINDOW_XID (gtk_widget_get_window(widget));
    video_window_handle = xid;
}


void make_window_black(GtkWidget *window)
{
    GdkColor color;
    gdk_color_parse("black", &color);
```

```
    gtk_widget_modify_bg(window, GTK_STATE_NORMAL, &color);
}

void destroy_cb(GtkWidget * widget, gpointer data)
{
    GMainLoop *loop = (GMainLoop*) data;
    g_print("Window destroyed\n");
    g_main_loop_quit(loop);
}

gboolean bus_call(GstBus * bus, GstMessage *msg, gpointer data)
{
    switch(GST_MESSAGE_TYPE(msg))
    {
    case GST_MESSAGE_ELEMENT:
        break;

    case GST_MESSAGE_EOS:
    {
        GtkWidget *window = (GtkWidget*) data;
        gtk_widget_destroy(window);
        break;
    }

    default:
        break;
    }

    return TRUE;
}

int
main (int argc, char **argv)
{
    GstStateChangeReturn ret;
    GtkWidget *video_window;
    GtkWidget *app_window;
    GstElement *pipeline;
    GstBus *bus;
    GMainLoop *loop;

    /* Initialize GStreamer */
    gst_init (&argc, &argv);

    /* Build the pipeline */

    //OK - sender
/*    pipeline = gst_parse_launch ("uvch264src device=/dev/video0 auto-
start=true iframe-period=10 initial-bitrate=1000000 name=src src.vidsrc ! \
                                video/x-h264, width=1920, height=1080, frame-
rate=24/1 ! h264parse ! rtph264pay ! \
                                udpsink host=pc2.local port=5001 pulsesrc
stream-properties=\"props,filter.want=echo-cancel,media.role=phone\" ! \
                                audioconvert ! lamemp3enc bitrate=128 !
udpsink host=pc2.local port=6001 . src.vfsrc ! queue ! fakesink", NULL);
*/
    //OK - sender med queue och förtitt, utan audio
    pipeline = gst_parse_launch ("uvch264src device=/dev/video0 auto-start=true
iframe-period=10 initial-bitrate=1000000 name=src src.vidsrc ! \
                                video/x-h264, width=1920, height=1080, frame-
rate=24/1 ! h264parse ! rtph264pay ! \
                                udpsink host=pc2.local port=5001 src.vfsrc !
queue ! video/x-raw, width=160 ! \
                                udpsink host=127.0.0.1 port=5003", NULL);
```

```
    gtk_init(&argc, &argv);

    loop = g_main_loop_new(NULL, FALSE);

    app_window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

    video_window = gtk_drawing_area_new ();
    g_signal_connect (video_window, "realize",
                      G_CALLBACK (video_widget_realize_cb), NULL);
    gtk_widget_set_double_buffered (video_window, FALSE);

    // usually the video_window will not be directly embedded into the
    // application window like this, but there will be many other widgets
    // and the video window will be embedded in one of them instead
    gtk_container_add (GTK_CONTAINER (app_window), video_window);

    g_signal_connect(G_OBJECT(app_window), "destroy", G_CALLBACK(destroy_cb),
loop);

    // show the GUI
    gtk_widget_show_all (app_window);

    // realize window now so that the video window gets created and we can
    // obtain its XID/HWND before the pipeline is started up and the videosink
    // asks for the XID/HWND of the window to render onto
    gtk_widget_realize (video_window);

    // we should have the XID/HWND now
    g_assert (video_window_handle != 0);

    make_window_black(app_window);

    //gtk_window_fullscreen((GtkWindow*)app_window);

    //Hide mouse pointer - not needed for sender
/*    GdkCursor *cur;
    cur = gdk_cursor_new(GDK_BLANK_CURSOR);
    gdk_window_set_cursor(gtk_widget_get_window( GTK_WIDGET(app_window)), cur);
    g_object_unref(cur);
*/

    // set up sync handler for setting the xid once the pipeline is started
    bus = gst_pipeline_get_bus (GST_PIPELINE (pipeline));
    gst_bus_set_sync_handler (bus, (GstBusSyncHandler) bus_sync_handler, NULL,
NULL);
    gst_bus_add_watch(gst_pipeline_get_bus(GST_PIPELINE(pipeline)), (GstBus-
Func) bus_call, app_window);
    g_signal_connect (G_OBJECT (bus), "message::eos", G_CALLBACK(destroy_cb),
loop);


    ret = gst_element_set_state(pipeline, GST_STATE_PLAYING);

    if (ret == GST_STATE_CHANGE_FAILURE)
    {
        g_print("Failed to start up pipeline!\n");
        return 1;
    }

    gst_object_unref (bus);

    g_main_loop_run(loop);
```

```
    /* Free resources */
    //g_main_loop_unref(loop);
    gst_element_set_state (pipeline, GST_STATE_NULL);
    gst_object_unref (pipeline);
    return 0;
}
```

GStreamer mottagare (gst-receiver-preview-noaudio)

```
/* main.cpp
* Roger Sundh
* 2014-08-26
* Kompilera med kommandot
* g++ main.cpp -o gst-receiver-preview-noaudio `pkg-config --cflags --libs gs-
treamer-1.0 gtk+-3.0 gstreamer-video-1.0`
*
*/

#include <gst/gst.h>
#include <gst/video/videooverlay.h>
#include <gtk/gtk.h>
#include <gdk/gdkx.h>  // for GDK_WINDOW_XID

static guintptr video_window_handle = 0;

static GstBusSyncReply
bus_sync_handler (GstBus * bus, GstMessage * message, gpointer user_data)
{
// ignore anything but 'prepare-window-handle' element messages
    if (!gst_is_video_overlay_prepare_window_handle_message (message))
        return GST_BUS_PASS;

    if (video_window_handle != 0)
    {
        GstVideoOverlay *overlay;

        // GST_MESSAGE_SRC (message) will be the video sink element
        overlay = GST_VIDEO_OVERLAY (GST_MESSAGE_SRC (message));
        gst_video_overlay_set_window_handle (overlay, video_window_handle);
    }
    else
    {
        g_warning ("Should have obtained video_window_handle by now!");
    }

    gst_message_unref (message);
    return GST_BUS_DROP;
}

static void
video_widget_realize_cb (GtkWidget *widget, gpointer data)
{
#if GTK_CHECK_VERSION(2,18,0)
    // Tell Gtk+/Gdk to create a native window for this widget instead of
    // drawing onto the parent widget.
    // This is here just for pedagogical purposes, GDK_WINDOW_XID will call
    // it as well in newer Gtk versions
//  if (!gdk_window_ensure_native (widget->window)) //This is old gtk+-2.0
    if (!gdk_window_ensure_native (gtk_widget_get_window(widget)))
        g_error ("Couldn't create native window needed for GstVideoOverlay!");
#endif

    gulong xid = GDK_WINDOW_XID (gtk_widget_get_window(widget));
    video_window_handle = xid;
}

void make_window_black(GtkWidget *window)
{
    GdkColor color;
    gdk_color_parse("black", &color);
    gtk_widget_modify_bg(window, GTK_STATE_NORMAL, &color);
}
```

```
void destroy_cb(GtkWidget * widget, gpointer data)
{
    GMainLoop *loop = (GMainLoop*) data;
    g_print("Window destroyed\n");
    g_main_loop_quit(loop);
}

gboolean bus_call(GstBus * bus, GstMessage *msg, gpointer data)
{
    switch(GST_MESSAGE_TYPE(msg))
    {
    case GST_MESSAGE_ELEMENT:
        break;

    case GST_MESSAGE_EOS:
    {
        GtkWidget *window = (GtkWidget*) data;
        gtk_widget_destroy(window);
        break;
    }

    default:
        break;
    }

    return TRUE;
}

int
main (int argc, char **argv)
{
    GstStateChangeReturn ret;
    GtkWidget *video_window;
    GtkWidget *app_window;
    GstElement *pipeline;
    GstBus *bus;
    GMainLoop *loop;

    /* Initialize GStreamer */
    gst_init (&argc, &argv);

    /* Build the pipeline for receiver */
/*    pipeline = gst_parse_launch ("udpsrc port=5002 caps = \"application/x-
rtp, media=(string)video, clock-rate=(int)90000, encoding-name=(string)H264,
payload=(int)96\" ! \
 *                                decodebin ! xvimagesink sync=false udpsrc
port=6002 ! mad ! pulsesink stream-properties=\"props,filter.want=echo-
cancel,media.role=phone\"", NULL);
 */

    //Receiver - no audio
    pipeline = gst_parse_launch ("videomixer background=black name=mix
sink_0::alpha=1 sink_1::alpha=1 sink_0::zorder=0 sink_1::zorder=1 ! videocon-
vert ! xvimagesink sync=false \
                                udpsrc port=5002 caps = \"application/x-rtp,
media=(string)video, clock-rate=(int)90000, encoding-name=(string)H264, paylo-
ad=(int)96\" ! \
                                queue ! decodebin ! mix.sink_0 udpsrc
port=5003 caps = \"video/x-raw, format=(string)YUY2, width=(int)160,
height=(int)120, framerate=(fraction)15/1\" ! \
                                queue ! decodebin ! mix.sink_1", NULL);
```

```
    gtk_init(&argc, &argv);

    //gsettings set org.gnome.desktop.screensaver idle-activation-enabled fal-
se;

    loop = g_main_loop_new(NULL, FALSE);

    app_window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

    video_window = gtk_drawing_area_new ();
    g_signal_connect (video_window, "realize",
                      G_CALLBACK (video_widget_realize_cb), NULL);
    gtk_widget_set_double_buffered (video_window, FALSE);

    // usually the video_window will not be directly embedded into the
    // application window like this, but there will be many other widgets
    // and the video window will be embedded in one of them instead
    gtk_container_add (GTK_CONTAINER (app_window), video_window);

    g_signal_connect(G_OBJECT(app_window), "destroy", G_CALLBACK(destroy_cb),
loop);

    // show the GUI
    gtk_widget_show_all (app_window);

    // realize window now so that the video window gets created and we can
    // obtain its XID/HWND before the pipeline is started up and the videosink
    // asks for the XID/HWND of the window to render onto
    gtk_widget_realize (video_window);

    // we should have the XID/HWND now
    g_assert (video_window_handle != 0);

    make_window_black(app_window);
    make_window_black(video_window);
    gtk_window_fullscreen((GtkWindow*)app_window);

    //Hide mouse pointer
    GdkCursor *cur;
    cur = gdk_cursor_new(GDK_BLANK_CURSOR);
    gdk_window_set_cursor(gtk_widget_get_window( GTK_WIDGET(app_window)), cur);
    g_object_unref(cur);

    // set up sync handler for setting the xid once the pipeline is started
    bus = gst_pipeline_get_bus (GST_PIPELINE (pipeline));
    gst_bus_set_sync_handler (bus, (GstBusSyncHandler) bus_sync_handler, NULL,
                              NULL);
    gst_bus_add_watch(gst_pipeline_get_bus(GST_PIPELINE(pipeline)), (GstBus-
Func) bus_call, app_window);
    g_signal_connect (G_OBJECT (bus), "message::eos", G_CALLBACK(destroy_cb),
loop);


    ret = gst_element_set_state(pipeline, GST_STATE_PLAYING);

    if (ret == GST_STATE_CHANGE_FAILURE)
    {
        g_print("Failed to start up pipeline!\n");
        return 1;
    }
    gst_object_unref (bus);


    g_main_loop_run(loop);
```

```
    /* Free resources */
    //g_main_loop_unref(loop);
    gst_element_set_state (pipeline, GST_STATE_NULL);
    gst_object_unref (pipeline);
    return 0;
}
```

Kamerastyrningsprogrammet

```
/* gtk-c920-controller.c
 * Roger Sundh 2014-11-20
 * This program controls the function of the camera
 */

#include <gtk/gtk.h>
#include <stdlib.h>

static void init_camera(void)
{
  char cmd[100];

  //Update value
  sprintf(cmd, "uvcdynctrl -v -s \"Brightness\" %d", 128);
  system(cmd);

  sprintf(cmd, "uvcdynctrl -v -s \"Contrast\" %d", 128);
  system(cmd);

  sprintf(cmd, "uvcdynctrl -v -s \"Saturation\" %d", 128);
  system(cmd);

  sprintf(cmd, "uvcdynctrl -v -s \"Gain\" %d", 0);
  system(cmd);

  sprintf(cmd, "uvcdynctrl -v -s \"Sharpness\" %d", 128);
  system(cmd);

  sprintf(cmd, "uvcdynctrl -v -s \"White Balance Temperature, Auto\" %d", 0);
  system(cmd);

  sprintf(cmd, "uvcdynctrl -v -s \"White Balance Temperature\" %d", 2200);
  system(cmd);

  sprintf(cmd, "uvcdynctrl -v -s \"White Balance Temperature, Auto\" %d", 1);
  system(cmd);

  sprintf(cmd, "uvcdynctrl -v -s \"Power Line Frequency\" %d", 1); //50Hz
  system(cmd);

  sprintf(cmd, "uvcdynctrl -v -s \"Backlight Compensation\" %d", 0);
  system(cmd);

  sprintf(cmd, "uvcdynctrl -v -s \"Exposure, Auto\" %d", 1); //1 = Off
  system(cmd);

  sprintf(cmd, "uvcdynctrl -v -s \"Exposure, Auto Priority\" %d", 1);
  system(cmd);

  sprintf(cmd, "uvcdynctrl -v -s \"Exposure (Absolute)\" %d", 250);
  system(cmd);

  sprintf(cmd, "uvcdynctrl -v -s \"Exposure, Auto\" %d", 3); //3 = On
  system(cmd);

  sprintf(cmd, "uvcdynctrl -v -s \"Focus, Auto\" %d", 0); //Off
  system(cmd);

  sprintf(cmd, "uvcdynctrl -v -s \"Focus (absolute)\" %d", 0);
  system(cmd);

  sprintf(cmd, "uvcdynctrl -v -s \"Focus, Auto\" %d", 1); //On
  system(cmd);
```

```
  sprintf(cmd, "uvcdynctrl -v -s \"Pan (Absolute)\" -- %d", 0);
  system(cmd);

  sprintf(cmd, "uvcdynctrl -v -s \"Tilt (Absolute)\" -- %d", 0);
  system(cmd);

  sprintf(cmd, "uvcdynctrl -v -s \"Zoom, Absolute\" %d", 100);
  system(cmd);

}

static void print_hello (GtkWidget *widget, gpointer data)
{
  g_print ("Hello World\n");
}

static gboolean update_brightness(GtkWidget *widget, gpointer data)
{
  GtkRange *range = (GtkRange *) widget;
  //g_print("Brightness=%g\n",gtk_range_get_value(range));
  gdouble brightness = gtk_range_get_value(range);
  //g_print("Brightness=%g\n",brightness);

  //Make temp string
  char cmd[100];

  //Show value before update
  sprintf(cmd, "uvcdynctrl -v -g \"Brightness\"");
  system(cmd);

  //Update value
  sprintf(cmd, "uvcdynctrl -v -s \"Brightness\" %g", brightness);
  system(cmd);

  //Show value after update
  system("uvcdynctrl -v -g \"Brightness\"");

  return TRUE;
}

static gboolean update_contrast(GtkWidget *widget, gpointer data)
{
  GtkRange *range = (GtkRange *) widget;
  gdouble contrast = gtk_range_get_value(range);

  //Make temp string
  char cmd[100];

  //Show value before update
  sprintf(cmd, "uvcdynctrl -v -g \"Contrast\"");
  system(cmd);

  //Update value
  sprintf(cmd, "uvcdynctrl -v -s \"Contrast\" %g", contrast);
  system(cmd);

  //Show value after update
  system("uvcdynctrl -v -g \"Contrast\"");

  return TRUE;
}

static gboolean update_saturation(GtkWidget *widget, gpointer data)
{
```

```
  GtkRange *range = (GtkRange *) widget;
  gdouble saturation = gtk_range_get_value(range);

  //Make temp string
  char cmd[100];

  //Show value before update
  sprintf(cmd, "uvcdynctrl -v -g \"Saturation\"");
  system(cmd);

  //Update value
  sprintf(cmd, "uvcdynctrl -v -s \"Saturation\" %g", saturation);
  system(cmd);

  //Show value after update
  system("uvcdynctrl -v -g \"Saturation\"");

  return TRUE;
}

static gboolean update_gain(GtkWidget *widget, gpointer data)
{
  GtkRange *range = (GtkRange *) widget;
  gdouble gain = gtk_range_get_value(range);

  //Make temp string
  char cmd[100];

  //Show value before update
  sprintf(cmd, "uvcdynctrl -v -g \"Gain\"");
  system(cmd);

  //Update value
  sprintf(cmd, "uvcdynctrl -v -s \"Gain\" %g", gain);
  system(cmd);

  //Show value after update
  system("uvcdynctrl -v -g \"Gain\"");

  return TRUE;
}

static gboolean update_sharpness(GtkWidget *widget, gpointer data)
{
  GtkRange *range = (GtkRange *) widget;
  gdouble sharpness = gtk_range_get_value(range);

  //Make temp string
  char cmd[100];

  //Show value before update
  sprintf(cmd, "uvcdynctrl -v -g \"Sharpness\"");
  system(cmd);

  //Update value
  sprintf(cmd, "uvcdynctrl -v -s \"Sharpness\" %g", sharpness);
  system(cmd);

  //Show value after update
  system("uvcdynctrl -v -g \"Sharpness\"");

  return TRUE;
}

static gboolean update_white_balance_temperature (GtkWidget *widget, gpointer
```

```
data)
{
  GtkRange *range = (GtkRange *) widget;
  gdouble wbt = gtk_range_get_value(range);

  //Make temp string
  char cmd[100];

  //Show value before update
  sprintf(cmd, "uvcdynctrl -v -g \"White Balance Temperature\"");
  system(cmd);

  //Update value
  sprintf(cmd, "uvcdynctrl -v -s \"White Balance Temperature\" %g", wbt);
  system(cmd);

  //Show value after update
  system("uvcdynctrl -v -g \"White Balance Temperature\"");

  return TRUE;
}

static gboolean update_white_balance_auto(GtkWidget *widget, gpointer data)
{
  GtkSwitch *sw = (GtkSwitch *) widget;
  gboolean state;
  state = gtk_switch_get_active(sw);
  //Make temp string
  char cmd[100];

  //Show value before update
  sprintf(cmd, "uvcdynctrl -v -g \"White Balance Temperature, Auto\"");
  system(cmd);

  //Update value
  sprintf(cmd, "uvcdynctrl -v -s \"White Balance Temperature, Auto\" %d", sta-
te);
  system(cmd);

  //Show value after update
  system("uvcdynctrl -v -g \"White Balance Temperature, Auto\"");

  return TRUE;
}

static gboolean update_filter(GtkWidget *widget, gpointer data)
{
  GtkToggleButton *btn = (GtkToggleButton *) widget;
  gboolean state;
  state = gtk_toggle_button_get_active(btn);
  //Make temp string
  char cmd[100];

  //Show value before update
  sprintf(cmd, "uvcdynctrl -v -g \"Power Line Frequency\"");
  system(cmd);

  //Update value
  sprintf(cmd, "uvcdynctrl -v -s \"Power Line Frequency\" %d", state);
  system(cmd);

  //Show value after update
  system("uvcdynctrl -v -g \"Power Line Frequency\"");

  return TRUE;
```

```
}

static gboolean update_switch_backlight_compensation(GtkWidget *widget, gpoin-
ter data)
{
  GtkSwitch *sw = (GtkSwitch *) widget;
  gboolean state;
  state = gtk_switch_get_active(sw);
  //Make temp string
  char cmd[100];

  //Show value before update
  sprintf(cmd, "uvcdynctrl -v -g \"Backlight Compensation\"");
  system(cmd);

  //Update value
  sprintf(cmd, "uvcdynctrl -v -s \"Backlight Compensation\" %d", state);
  system(cmd);

  //Show value after update
  system("uvcdynctrl -v -g \"Backlight Compensation\"");

  return TRUE;
}

static gboolean update_exposure(GtkWidget *widget, gpointer data)
{
  GtkRange *range = (GtkRange *) widget;
  gdouble exposure = gtk_range_get_value(range);

  //Make temp string
  char cmd[100];

  //Show value before update
  sprintf(cmd, "uvcdynctrl -v -g \"Exposure (Absolute)\"");
  system(cmd);

  //Update value
  sprintf(cmd, "uvcdynctrl -v -s \"Exposure (Absolute)\" %g", exposure);
  system(cmd);

  //Show value after update
  system("uvcdynctrl -v -g \"Exposure (Absolute)\"");

  return TRUE;
}

static gboolean update_exposure_auto(GtkWidget *widget, gpointer data)
{
  GtkSwitch *sw = (GtkSwitch *) widget;
  gboolean state;
  gint val;
  state = gtk_switch_get_active(sw);
  if (state)
    val = 3;
  else
  {
    val = 1;
  }

  //Make temp string
  char cmd[100];

  //Show value before update
  sprintf(cmd, "uvcdynctrl -v -g \"Exposure, Auto\"");
```

```
  system(cmd);

  //Update value
  sprintf(cmd, "uvcdynctrl -v -s \"Exposure, Auto\" %d", val);
  system(cmd);

  //Show value after update
  system("uvcdynctrl -v -g \"Exposure, Auto\"");

  return TRUE;
}

static gboolean update_exposure_auto_priority(GtkWidget *widget, gpointer data)
{
  GtkSwitch *sw = (GtkSwitch *) widget;
  gboolean state;
  state = gtk_switch_get_active(sw);
  //Make temp string
  char cmd[100];

  //Show value before update
  sprintf(cmd, "uvcdynctrl -v -g \"Exposure, Auto Priority\"");
  system(cmd);

  //Update value
  sprintf(cmd, "uvcdynctrl -v -s \"Exposure, Auto Priority\" %d", state);
  system(cmd);

  //Show value after update
  system("uvcdynctrl -v -g \"Exposure, Auto Priority\"");

  return TRUE;
}

static gboolean update_focus(GtkWidget *widget, gpointer data)
{
  GtkRange *range = (GtkRange *) widget;
  gdouble focus = gtk_range_get_value(range);

  //Quantize in steps of 5
  guint val = 5 * ((guint)focus / 5);
  gtk_range_set_value(range, 1.0*val);

  //For debugging
  //g_print("%u\n", val);
  //g_print("Adj=%g\n",gtk_range_get_value(range));

  //Make temp string
  char cmd[100];

  //Show value before update
  sprintf(cmd, "uvcdynctrl -v -g \"Focus (absolute)\"");
  system(cmd);

  //Update value
  sprintf(cmd, "uvcdynctrl -v -s \"Focus (absolute)\" %g", 1.0*val);
  system(cmd);

  //Show value after update
  system("uvcdynctrl -v -g \"Focus (absolute)\"");

  return TRUE;
}

static gboolean update_focus_auto(GtkWidget *widget, gpointer data)
```

```
{
  GtkSwitch *sw = (GtkSwitch *) widget;
  gboolean state;
  state = gtk_switch_get_active(sw);
  //Make temp string
  char cmd[100];

  //Show value before update
  sprintf(cmd, "uvcdynctrl -v -g \"Focus, Auto\"");
  system(cmd);

  //Update value
  sprintf(cmd, "uvcdynctrl -v -s \"Focus, Auto\" %d", state);
  system(cmd);

  //Show value after update
  system("uvcdynctrl -v -g \"Focus, Auto\"");

  return TRUE;
}

static gboolean update_pan(GtkWidget *widget, gpointer data)
{
  GtkRange *range = (GtkRange *) widget;
  gdouble pan = gtk_range_get_value(range);

  //Quantize in steps of 5
  gint val = 3600 * ((gint)pan / 3600);
  gtk_range_set_value(range, 1.0*val);

  //For debugging
  //g_print("%d\n", val);
  //g_print("Adj=%g\n",gtk_range_get_value(range));

  //Make temp string
  char cmd[100];

  //Show value before update
  sprintf(cmd, "uvcdynctrl -v -g \"Pan (Absolute)\"");
  system(cmd);



  //Update value
  sprintf(cmd, "uvcdynctrl -v -s \"Pan (Absolute)\" -- %d", val);
  system(cmd);

  //Show value after update
  system("uvcdynctrl -v -g \"Pan (Absolute)\"");

  return TRUE;
}

static gboolean update_tilt(GtkWidget *widget, gpointer data)
{
  GtkRange *range = (GtkRange *) widget;
  gdouble tilt = gtk_range_get_value(range);

  //Quantize in steps of 3600
  gint val = 3600 * ((gint)tilt / 3600);
  gtk_range_set_value(range, 1.0*val);

  //For debugging
  //g_print("%d\n", val);
  //g_print("Adj=%g\n",gtk_range_get_value(range));
```

```
  //Make temp string
  char cmd[100];

  //Show value before update
  sprintf(cmd, "uvcdynctrl -v -g \"Tilt (Absolute)\"");
  system(cmd);

  //Update value
  sprintf(cmd, "uvcdynctrl -v -s \"Tilt (Absolute)\" -- %d", val);
  system(cmd);

  //Show value after update
  system("uvcdynctrl -v -g \"Tilt (Absolute)\"");

  return TRUE;
}

static gboolean update_zoom(GtkWidget *widget, gpointer data)
{
  GtkRange *range = (GtkRange *) widget;
  gdouble zoom = gtk_range_get_value(range);

  //Make temp string
  char cmd[100];

  //Show value before update
  sprintf(cmd, "uvcdynctrl -v -g \"Zoom, Absolute\"");
  system(cmd);

  //Update value
  sprintf(cmd, "uvcdynctrl -v -s \"Zoom, Absolute\" %g", zoom);
  system(cmd);

  //Show value after update
  system("uvcdynctrl -v -g \"Zoom, Absolute\"");

  return TRUE;
}



int main (int argc, char *argv[])
{
  GtkBuilder *builder;
  GObject *window;
  GObject *button;
  GObject *scale;
  GObject *sw;

  gtk_init (&argc, &argv);

  /* Construct a GtkBuilder instance and load our UI description */
  builder = gtk_builder_new ();
  gtk_builder_add_from_file (builder, "builder.ui", NULL);

  /* Connect signal handlers to the constructed widgets. */
  window = gtk_builder_get_object (builder, "windowMain");
  g_signal_connect (window, "destroy", G_CALLBACK (gtk_main_quit), NULL);

  button = gtk_builder_get_object (builder, "btnOk");
  g_signal_connect (button, "clicked", G_CALLBACK (print_hello), NULL);

  scale = gtk_builder_get_object (builder, "scaleBrightness");
  g_signal_connect (scale, "value-changed", G_CALLBACK (update_brightness),
```

```
NULL);

  scale = gtk_builder_get_object (builder, "scaleContrast");
  g_signal_connect (scale, "value-changed", G_CALLBACK (update_contrast),
NULL);

  scale = gtk_builder_get_object (builder, "scaleSaturation");
  g_signal_connect (scale, "value-changed", G_CALLBACK (update_saturation),
NULL);

  scale = gtk_builder_get_object (builder, "scaleGain");
  g_signal_connect (scale, "value-changed", G_CALLBACK (update_gain), NULL);

  scale = gtk_builder_get_object (builder, "scaleSharpness");
  g_signal_connect (scale, "value-changed", G_CALLBACK (update_sharpness),
NULL);

  scale = gtk_builder_get_object (builder, "scaleWhiteBalanceTemperature");
  g_signal_connect (scale, "value-changed", G_CALLBACK
(update_white_balance_temperature), NULL);

  sw = gtk_builder_get_object (builder, "switchWhiteBalanceAuto");
  g_signal_connect (sw, "notify::active", G_CALLBACK
(update_white_balance_auto), NULL);

  button = gtk_builder_get_object (builder, "radiobutton50Hz");
  g_signal_connect (button, "toggled", G_CALLBACK (update_filter), NULL);

  sw = gtk_builder_get_object (builder, "switchBackLight");
  g_signal_connect (sw, "notify::active", G_CALLBACK
(update_switch_backlight_compensation), NULL);

  scale = gtk_builder_get_object (builder, "scaleExposure");
  g_signal_connect (scale, "value-changed", G_CALLBACK (update_exposure),
NULL);

  sw = gtk_builder_get_object (builder, "switchExposureAuto");
  g_signal_connect (sw, "notify::active", G_CALLBACK (update_exposure_auto),
NULL);

  sw = gtk_builder_get_object (builder, "switchExposureAutoPriority");
  g_signal_connect (sw, "notify::active", G_CALLBACK (update_exposure_auto_pri-
ority), NULL);

  scale = gtk_builder_get_object (builder, "scaleFocus");
  g_signal_connect (scale, "value-changed", G_CALLBACK (update_focus), NULL);

  sw = gtk_builder_get_object (builder, "switchFocusAuto");
  g_signal_connect (sw, "notify::active", G_CALLBACK (update_focus_auto),
NULL);

  scale = gtk_builder_get_object (builder, "scalePan");
  g_signal_connect (scale, "value-changed", G_CALLBACK (update_pan), NULL);

  scale = gtk_builder_get_object (builder, "scaleTilt");
  g_signal_connect (scale, "value-changed", G_CALLBACK (update_tilt), NULL);

  scale = gtk_builder_get_object (builder, "scaleZoom");
  g_signal_connect (scale, "value-changed", G_CALLBACK (update_zoom), NULL);

  button = gtk_builder_get_object (builder, "btnQuit");
  g_signal_connect (button, "clicked", G_CALLBACK (gtk_main_quit), NULL);

  //Init camera
  init_camera();
```

```
  gtk_main();

  return 0;
}
```

Bilaga B: Specifikation av komponentpriser

Samtliga priser är inklusive moms.

Fall A

Komponent	Pris
Raspberry Pi	239 kr
SD-kort, 8GB	100 kr
Webbkamera C920	771 kr
Monitor 48"	6 600 kr
USB-hubb	200 kr
Högtalare	199 kr
Totalt per nod	8 109 kr
Antal noder	2
Totalpris	**16 218 kr**

Fall B

Komponent	Pris
Raspberry Pi	239 kr
SD-kort, 8GB	100 kr
PC-mikrofon	69 kr
RPI-kameramodul	279 kr
Monitor 48"	6 600 kr
USB-hubb	200 kr
Högtalare	199 kr
Ljudkort USB	75 kr
Totalt per nod	7 353 kr
Antal noder	2
Totalpris	**14 706 kr**

Fall C

Komponent	Pris
Webbkamera C920	771 kr
Monitor 48"	6 600 kr
Zotac Mini-PC	3 300 kr
Högtalare	199 kr
Totalt per nod	10 099 kr
Antal noder	2
Totalpris	**20 198 kr**

Fall AA

Komponent	Pris
Sändare	
Raspberry Pi	239 kr
SD-kort, 8GB	100 kr
Webbkamera C920	771 kr
Delsumma	1 110 kr
Mottagare	
Raspberry Pi	239 kr
Monitor 48"	6 600 kr
USB-hubb	200 kr
Högtalare	199 kr
Delsumma	7 238 kr
Totalt per nod	8 348 kr
Antal noder	2
Totalpris	**16 696 kr**

Fall BB

Komponent	Pris
Sändare	
Raspberry Pi	239 kr
SD-kort, 8GB	100 kr
PC-mikrofon	69 kr
RPI-kameramodul	279 kr
Ljudkort USB	75 kr
Delsumma	762 kr
Mottagare	
Raspberry Pi	239 kr
SD-kort, 8GB	100 kr
Monitor 48"	6 600 kr
USB-hubb	200 kr
Högtalare	199 kr
Delsumma	7 338 kr
Totalt per nod	8 100 kr
Antal noder	2
Totalpris	**16 200 kr**

Bilaga C: Dagboksanteckningar

#2014-08-13
Frågeställningar

Köra ett grafiskt program i fullskärmsläge i GNOME

Styra inspelnings- och uppspelningsvolym i realtid

Kunna välja ekoutsläckningsalgoritm

Skriva ett grafiskt program i GTK+ eller Qt som anropar GStreamer

Undvika att skärmen går i viloläge (ingen skärmsläckare)

Ta bort muspekaren

#2014-08-16
Codeblocks
Börjat testa tutorials för gstreamer-1.0 i Codeblocks.
Testat ett exempel för att växla till fullskärmsläge med hjälp av gtk+ och gstreamer-0.10
Ska försöka anpassa det till gstreamer-1.0, så att jag kan använda uvch264src för webbkameran

Qt
Testat ett exempel med videotestsrc och gstreamer-1.0. Fungerar i fullskärmsläge och utan muspekare
Utgått från detta exempel
http://programmingintherain.blogspot.se/2013/03/xoverlay.html

Ett annat ställe med bra exempel:
http://gstreamer.freedesktop.org/data/doc/gstreamer/head/gst-plugins-base-libs/html/gst-plugins-base-libs-gstvideooverlay.html

#2014-08-18
Skrivit gst-gtk-programmen
gst-overlay-launcher-receiver och gst-overlay-launcher-sender.
De har hårdkodade värden på alla parametrar. Ekoutsläckningen ska vara aktiverad.
Problem med att pc1.local tappar ekoutsläckningen. Kan bero på att den kör flera program än pc2, eftersom jag använder den som utvecklingsdator.

#Sändaren
uvch264src device=/dev/video0 auto-start=true iframe-period=10 initial-bitrate=1000000 name=src src.vidsrc ! video/x-h264, width=1920, height=1080, framerate=24/1 ! h264parse ! rtph264pay ! udpsink host=pc2.local port=5001 pulsesrc stream-properties="props,filter.want=echo-cancel,media.role=phone" ! audioconvert ! lamemp3enc bitrate=128 ! udpsink host=pc2.local port=6001 . src.vfsrc ! queue ! fakesink

#2014-08-20
Fixat till så att en förtitt mixas in längst upp till vänster i bilden. Den sänds på port 5003(pc1) resp. 5004(pc2) och den sänds på adress 127.0.0.1 (men det funkar inte med att skriva localhost).
Jobbat mycket med olika mixerexempel.
Skärmsläckaren är fortfarande problematisk.

#2014-08-21
Överfört förtittsalternativen till kod i programmen gst-receiver-preview, respektive gst-sender-preview

kommando för att styra skärmsläckaren
gsettings set org.gnome.desktop.screensaver idle-activation-enabled false
gsettings set org.gnome.desktop.screensaver idle-activation-enabled true

xset -display :0 -dpms s off s noblank
xset -display :0 +dpms s on s blank

Problem med att ekoutsläckningen förloras efter en kort tid. Kanske fungerar det bättre med pulseservern direkt för ljudet?

#2014-08-24
Ekoutsläckningen tappar inte synken om man använder en separat mikrofon.
Exempel:
Sändaren(pc1)
gst-launch-1.0 -v pulsesrc device=alsa_input.pci-0000_00_1b.0.analog-stereo.echo-cancel stream-properties="props,filter.want=echo-cancel,media.role=phone" ! audioconvert ! lamemp3enc bitrate=128 ! udpsink port=6001 host=pc2.local

Mottagaren(pc2)
gst-launch-1.0 -v udpsrc port=6001 ! mad ! pulsesink stream-properties="props,filter.want=echo-cancel,media.role=phone"

Kommando för att hitta ljudkortsingångar (pulseaudio)
pactl list | grep -A2 'Source #' | grep 'Name: ' | cut -d" " -f2

Kommando för att hitta ljudkortsutgångar
pactl list | grep -A2 'Source #' | grep 'Name: .*\.monitor$' | cut -d" " -f2

Kommando för att välja mikrofoningång
pactl set-source-port alsa_input.pci-0000_00_1b.0.analog-stereo analog-input-microphone-rear
pactl set-source-port alsa_input.usb-046d_HD_Pro_Webcam_C920_E5E477AF-02-C920.analog-stereo analog-input-microphone

Kommando för att ställa in default ljudkort
echo "set-default-source alsa_input.pci-0000_00_1b.0.analog-stereo" | pacmd
echo "set-default-source alsa_input.usb-046d_HD_Pro_Webcam_C920_E5E477AF-02-C920.analog-stereo" | pacmd

Från http://freedesktop.org/software/pulseaudio/doxygen/proplist_8h.html
#define PA_PROP_MEDIA_ROLE "media.role"
For streams: logic role of this media.

One of the strings "video", "music", "game", "event", "phone", "animation", "production", "a11y", "test"

#2014-08-25
Installerat paprefs - Pulseaudio konfiguration
sudo apt-get install paprefs

Manuellt val av ekoutsläckning
1. Starta om pulseaudio
pulseaudio -k && pulseaudio --start

2. Ladda in ekoutsläckningsmodulen
pactl load-module module-echo-cancel aec_method=webrtc
(pactl load-module module-echo-cancel aec_method=speex)
(pactl load-module module-echo-cancel aec_method=adrian)

3. Starta sändaren (pc1)
gst-launch-1.0 -v pulsesrc device=alsa_input.pci-0000_00_1b.0.analog-stereo.echo-cancel stream-properties="props,filter.want=echo-cancel,media.role=phone" ! audioconvert ! lamemp3enc bitrate=128 ! udpsink port=6001 host=pc2.local

#Alternativ
gst-launch-1.0 -v pulsesrc device=alsa_input.pci-0000_00_1b.0.analog-stereo.echo-cancel ! audioconvert ! lamemp3enc bitrate=128 ! udpsink port=6001 host=pc2.local

4. Starta mottagaren (pc1)
gst-launch-1.0 -v udpsrc port=6002 ! mad ! pulsesink stream-properties="props,filter.want=echo-cancel,media.role=phone"

5. Gör samma sak på pc2

---
1. (pc1.local)
pactl load-module module-echo-cancel aec_method=webrtc source_master=alsa_input.usb-046d_HD_Pro_Webcam_C920_E5E477AF-02-C920.analog-stereo sink_master=alsa_output.pci-0000_00_1b.0.analog-stereo

pactl load-module module-echo-cancel aec_method=webrtc source_master=alsa_input.usb-046d_HD_Pro_Webcam_C920_E5E477AF-02-C920.analog-stereo sink_master=alsa_output.pci-0000_00_1b.0.analog-stereo adjust_threshold=100 adjust_time=60

(pc2.local)
pactl load-module module-echo-cancel aec_method=webrtc source_master=alsa_input.usb-046d_HD_Pro_Webcam_C920_A69F67AF-02-C920.analog-stereo sink_master=alsa_output.pci-0000_00_1b.0.analog-stereo

pactl load-module module-echo-cancel aec_method=webrtc source_master=alsa_input.usb-046d_HD_Pro_Webcam_C920_A69F67AF-02-C920.analog-stereo sink_master=alsa_output.pci-0000_00_1b.0.analog-stereo adjust_threshold=100 adjust_time=60

pactl load-module module-echo-cancel aec_method=webrtc source_master=alsa_input.usb-046d_HD_Pro_Webcam_C920_A69F67AF-02-C920.analog-stereo sink_master=alsa_output.pci-0000_00_1b.0.analog-stereo adjust_threshold=100 adjust_time=60 sink_name=noechosink source_name=noechosrc

2. (pc1)
gst-launch-1.0 -v pulsesrc device=alsa_input.usb-046d_HD_Pro_Webcam_C920_E5E477AF-02-C920.analog-stereo.echo-cancel stream-properties="props,filter.want=echo-cancel,media.role=phone" ! audioconvert ! lamemp3enc bitrate=128 ! udpsink port=6001 host=pc2.local

(pc2)
gst-launch-1.0 -v pulsesrc device=alsa_input.usb-046d_HD_Pro_Webcam_C920_A69F67AF-02-C920.analog-stereo.echo-cancel stream-properties="props,filter.want=echo-cancel,media.role=phone" ! audioconvert ! lamemp3enc bitrate=128 ! udpsink port=6002 host=pc1.local

3. (pc1)
gst-launch-1.0 -v udpsrc port=6002 ! mad ! pulsesink stream-properties="props,filter.want=echo-cancel,media.role=phone"

(pc2)
gst-launch-1.0 -v udpsrc port=6001 ! mad ! pulsesink stream-properties="props,filter.want=echo-cancel,media.role=phone"

Skapat script för att starta om pulsesrc och pulsesink uppkopplingarna (restart_audio)
Installerat ntp
sudo apt-get install ntp

Experimenterat med crontab
Planerar att lägga in omstart av audiouppkopplingen var 10:e minut via cron
http://www.debian-tutorials.com/crontab-tutorial-cron-howto
Kör kommandot crontab -l för att lista crontab-jobben och crontab -e för att editera filen.
Audiokommunikationen startar om var 10:e minut och detta loggas i "audiolog.txt"

#2014-10-20
Fortsatt med GStreamer tutorials
Lärt mig om wavescope och goom

#wavescope
gst-launch-1.0 filesrc location="/usr/share/sounds/alsa/Front_Center.wav" ! decodebin ! tee name=t ! queue ! autoaudiosink t. ! queue ! audioconvert ! wavescope ! videoconvert ! autovideosink

#goom
gst-launch-1.0 filesrc location="/usr/share/sounds/alsa/Front_Center.wav" ! decodebin ! tee name=t ! queue ! autoaudiosink t. ! queue ! audioconvert ! goom ! videoconvert ! autovideosink

#2014-10-26
Lärt mig om gst-discoverer
gst-discoverer-1.0 http://docs.gstreamer.com/media/sintel_trailer-480p.webm -v

#Installerat GraphViz, för att läsa debugfiler (*.dot), pipelinegrafer från GStreamer
sudo apt-get install graphviz

#Installerat clutter (för grafisk integration av GStreamer)
sudo apt-get install libclutter-gst-dev
Funkar inte i gstreamer-1.0 pga att elementet cluttersink saknas
Detta element finns i gstreamer1.0-clutter, men för att installera det paketet krävs en uppdatering av libc6 till minst version 2.14. Det går att försöka uppdatera libc6 manuellt (http://stackoverflow.com/questions/10863613/how-to-upgrade-glibc-from-version-2-13-to-2-15-on-debian), men jag tror att jag avstår det, pga risk för komplikationer. Cluttersink behövs för att kunna köra basic-tutorial-15 i GStreamer 1.0. Dock finns elementet cluttersink i GStreamer 0.10, varför det bör vara möjligt att köra den versionen istället.

#2014-10-28
Kört igenom GStreamer tutorials. Problem med att byta ljudströmmar i playback-tutorial-1

#2014-11-13
Kört igenom playback tutorials 4-7
Inga kvar att gå igenom nu.
Installerat systemuppdateringar

#2014-11-23
Arbetat med kamerastyrningen via ett gtk-program. Det verkar som om zoomningens maximala värde är 200 och inte 500, åtminstone tycks inget hända för värden över 200. Observera att panorering och tilt endast har effekt när zoomen är aktiverad.
På pc2 körs nu ett uppstartsscript som automatiskt kör igång videosändning och mottagning med förtitt. Det startar också igång audiokommunikationen, vilken sedan startar om via crontab, var 10:e minut, för att bibehålla synkroniseringen för ekoutsläckningen. På pc2 ställs ekoutsläckningen och audioenheterna in via filen /etc/pulse/default.pa. På pc1 får man själv köra skriptet (./restart_com-

munication.sh) och ekoutsläckningen initieras via skriptet restart_audio.sh.
Detta ska åtgärdas så att båda datorerna kör de automatiska uppstartsskripten.

Kommunikationen mellan pc1 på övervåningen och pc2 i källaren fungerar nu bra. Förmodligen kommer det att fungera bättre om kommunikationen inte sker över ett trådlöst nät.

#2014-12-04
Arbetat med att ställa in pc1 så att den automatiskt startar kommunikationen. Så här fungerar uppstarten:

Skriptet restart_communication.sh körs. Det innehåller hänvisningar till skript som startar videokommunikationen och audiokommunikationen i två separata kommandon, samt ett skript som ställer in förvalda värden för webbkameran.

Skriptet send_and_receive_noaudio.sh exporterar den lokala X-displayen så att X-programmen inte visas på fjärranslutna maskiner. Här stängs också skärmsläckaren av. Därefter anropas de egna program som startar videosändning och mottagning.
Observera att när ekoutsläckningsmodulen i GStreamer används, sänds audio och video i två helt separata uppkopplingar. Detta för att audiouppkopplingen ska kunna starta om automatiskt var 10:e minut, detta för att försöka behålla synkroniseringen för ekoutsläckningen mellan ljudkortens in- och utgångar.

Skript och filer som berörs:
restart_communication.sh
restart_audio.sh
send_and_receive_noaudio.sh
set_camera_defaults.sh
restart_pulsesrc.sh
restart_pulsesink.sh

Bilaga D: Användarmanual

Användarmanual

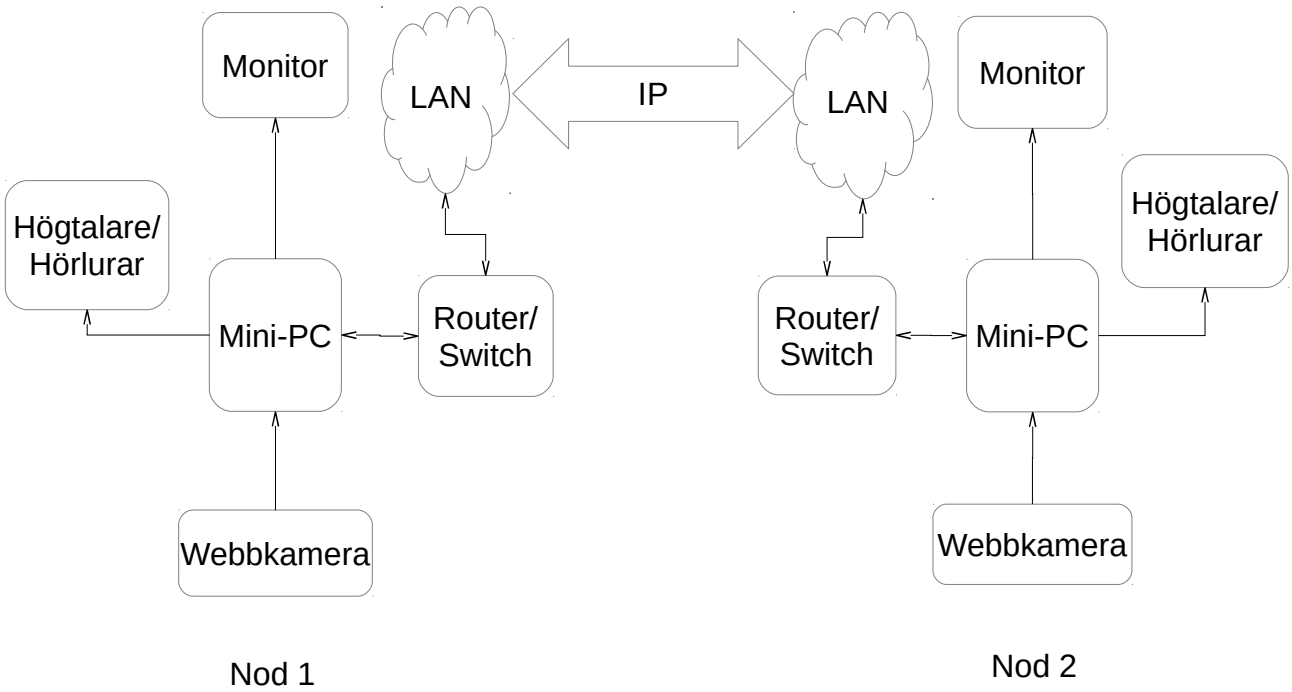
för P2P-kommunikationssystem

Consat AB

Roger Sundh, 2015-02-19
roger.sundh@musiktronik.se

Översikt

Systemet består av 2 st. likvärdiga noder, enligt figur nedan.



Figur 1. Systemöversikt

Systemet är avsett att användas för punkt-till-punktkommunikation med audio och video, över ett lokalt IP-baserat nätverk.

Beskrivning

Varje nod i systemet består av följande komponenter:

Hårdvara

Hårdvaran består av en Zotac Mini-PC med 2GB RAM. Kameran är en Logitech C-920 med inbyggd mikrofon och möjlighet att använda H264-formatet för videokodningen.

- 1 st. mini-PC (Zotac), med operativsystemet Linux Debian (wheezy).
- 1 st. webbkamera Logitech C-920
  Kameran sänder komprimerad video i formatet H264 och har en inbyggd stereomikrofon, med maximal samplingsfrekvens om 32 kHz.
- 1 st. stereohögtalare (alternativt hörlurar)
- 1 st. 45” LCD-monitor med HDMI-anslutning

Mjukvara

Operativsystemet är Linux Debian Wheezy, enligt resultatet från nedanstående kommando.

pi@pc1:~$ **uname -a**

Linux pc1 3.2.0-4-amd64 #1 SMP Debian 3.2.63-2+deb7u1 x86_64 GNU/Linux

För att sända audio och video över nätverket används version 1.2 av GStreamer. Nu finns det nyare versioner av detta ramverk (1.4), men det viktiga är att det finns stöd för H264-kodeken. Version 1.2 installerades från källkoden. Se filen **/home/pi/installationer.txt**

De program som styr kameran (gtk-c920-controller) och audio- och videokommunikationen (gst_sender_preview_noaudio, gst_receiver_preview_noaudio) är egenutvecklade. De är skrivna i C/C++ med hjälp av CodeBlocks IDE. Programmet *gtk-c920-controller* fjärrstartas via en terminalinloggning över ssh.

Kompilatorn är från GNU-projektet enligt detta kommando:

pi@pc1:~$ **gcc --version**

gcc (Debian 4.7.2-5) 4.7.2

Programmen använder GStreamers API.

De utvecklade programmen är beroende av X-windowsystemet, särskilt då den hårdvaruaccelererade grafiken som erbjuds via Gstreamerkomponenten *xvimagesink.*

Ekoutsläckningen använder sig av audiosystemet Pulseaudio, vilket numera är mer eller mindre standard för Linuxsystem (ersättare för ALSA). Detta följer med vid installationen av Debian.

För att styra webbkameran används programmet uvcdynctrl (Debian: `sudo apt-get install uvcdynctrl`).

Några länkar

https://www.debian.org/distrib/

http://gstreamer.freedesktop.org/

http://www.codeblocks.org/

Systemstart och skript

I nuvarande konfiguration, d.v.s. med aktiverad ekoutsläckningsalgoritm, sänds audio och video i två olika, samtidiga uppkopplingar. Synkroniseringen är beroende av hur väl nätverket kan hantera dessa olika strömmar. I de flesta genomförda försök har detta inte medfört några synkroniseringsproblem. I stora drag fungerar det så att audiokommunikationen styrs av GStreamer (gst-launch) anropat från ett kommandoskript, medan videokommunikationen styrs av de program som beskrivits ovan. När datorns startsekvens körs igång (via GRUB) startar en X-server igång så att det går att fjärransluta till datorn och köra fönsterbaserade program på den anslutande datorn. Detta utnyttjas t.ex. då kameran ska fjärrstyras via programmet *gtk-c920-controller*. Efter datorns normala start (till X-window Gnome Desktop) sker följande:

Skriptet *restart_communication.sh* körs. Det innehåller hänvisningar till skript som startar videokommunikationen och audiokommunikationen i två separata kommandon, samt ett skript som ställer in förvalda värden för webbkameran.

Skriptet *send_and_receive_noaudio.sh* exporterar den lokala X-displayen så att X-programmen inte visas på fjärranslutna maskiner. Här stängs också skärmsläckaren av. Därefter anropas de egna program som startar videosändning och mottagning.

Observera att när ekoutsläckningsmodulen i GStreamer används, sänds audio och video i två helt separata uppkopplingar. Detta för att audiouppkopplingen ska kunna starta om automatiskt var 10:e minut, detta för att försöka behålla synkroniseringen för ekoutsläckningen mellan ljudkortens in- och utgångar. Denna omstart sker via programmet *crontab*, enligt nedan:

pi@pc1:~$ **crontab -l**

0,10,20,30,40,50 * * * * /home/pi/restart_audio.sh && echo "Audio restarted `date`" >> audiolog.txt

Som framgår ovan loggas varje omstart i en textfil (audiolog.txt). Detta kan man möjligen vilja ändra på, så att hårddisken inte fylls (vilket i och för sig kan dröja rätt länge.....).

Konfiguration av ekoutsläckningen görs i filen /etc/pulse/default.pa. Ekoutsläckningen görs via en modul i pulseaudio.

I skriptet **set_camera_defaults.sh** ställs webbkamerans frekvensfilter in till 50 Hz, för att motverka flimmer i rum med lysrörsbelysning.

Hela startsekvensen ser ut enligt nedan:

/home/pi/**restart_communication.sh**

/home/pi/**restart_audio.sh**

/home/pi/**restart_pulsesrc.sh**

gst-launch-1.0 -v pulsesrc device=noechosrc stream-properties="props,filter.want=echo-cancel,media.role=phone" ! audioconvert ! lamemp3enc bitrate=128 ! udpsink port=6001 host=pcx.local

/home/pi/**restart_pulsesink.sh**

gst-launch-1.0 -v udpsrc port=6002 ! mad ! pulsesink device=noechosink stream-properties="props,filter.want=echo-cancel,media.role=phone"

/home/pi/**send_and_receive_noaudio.sh**

/home/pi/src/consat/codeblocks/production/gst-sender-preview-noaudio/gst-sender-preview-noaudio

/home/pi/src/consat/codeblocks/production/gst-receiver-preview-noaudio/gst-receiver-preview-noaudio

/home/pi/**set_camera_defaults.sh**

Program och skript

För att systemet ska fungera behövs följande skript och programfiler

Skript

/home/pi/restart_communication.sh

/home/pi/restart_audio.sh

/home/pi/restart_pulsesrc.sh

/home/pi/restart_pulsesink.sh

/home/pi/send_and_receive_noaudio.sh

/home/pi/set_camera_defaults.sh

Program

/home/pi/src/consat/codeblocks/production/gst-sender-preview-noaudio/gst-sender-preview-noaudio

/home/pi/src/consat/codeblocks/production/gst-receiver-preview-noaudio/gst-receiver-preview-noaudio

/home/pi/gtk-c920-controller

Till detta program hör gränssnittsfilen (gjord i GLADE) /home/pi/builder.ui

Källkoden till alla dessa program finns i mappen /home/pi/src/consat/code-blocks/production/

Konfigurationsfil för ekoutsläckningen

/etc/pulse/**default.pa (**utdrag)

#Manually load echo cancel module

load-module module-echo-cancel aec_method=webrtc source_master=alsa_input.usb-046d_HD_Pro_Webcam_C920_E5E477AF-02-C920.analog-stereo sink_master=alsa_output.pci-0000_00_1b.0.analog-stereo adjust_threshold=100 adjust_time=60 sink_name=noechosink source_name=noechosrc

Make some devices default

set-default-sink alsa_output.pci-0000_00_1b.0.analog-stereo

#set-default-sink alsa_output.pci-0000_00_1b.0.hdmi-stereo-extra2

set-default-source alsa_input.usb-046d_HD_Pro_Webcam_C920_E5E477AF-02-C920.analog-stereo

Övriga filer av intresse

/home/pi/dagbok.txt

/home/pi/c920-info.txt

/home/pi/bookmarks_14_12_21.html

/home/pi/consat-140307.txt

/home/pi/send_receive_info.txt

/home/pi/consattips_v2.txt

/home/pi/Fjärrkontroll.txt

/home/pi/hdmikortet.txt

/home/pi/installationer.txt

/home/pi/videomixerexempel.txt

I mappen /home/pi/scripts finns samtliga skript som använts vid utvecklingen av systemet. En del är avsedda för Raspberry Pi, men alla som krävs för att köra systemet på Zotac mini-PC, är kopierade till /home/pi.

Användning

Inkoppling

- Starta med nod 1. Den kommer att heta pc1.local i nätverket.
- Koppla in webbkameran i den övre av de två USB-3.0-uttagen på mini-PC:n. Dessa är placerade till höger om HDMI-kontakten.
- Anslut LCD-monitorn till HDMI-utgången på mini-PC:n.
- Anslut en nätverkskabel från mini-PC:n till ett ledigt uttag i en router som är ansluten till det lokala nätverket.
- Anslut en högtalare (aktiv) eller ett par hörlurar, till hörlursutgången på mini-PC:n.
- Anslut matningsspänningen till mini-PC:n.
- Starta monitor, högtalare och mini-PC.

Gör nu på samma sätt med nod 2. Den kommer att heta pc2.local i nätverket.

Start

När båda datorerna har startat kommer de att etablera kontakt och efter c:a 30 sekunder bör både audio- och videokommunikationen vara igång. Justera nu högtalarvolymen så att det blir lagom starkt för normal samtalsvolym. På skärmen bör du nu se dig själv i en liten förtitt längst upp till vänster i bild och på resterande yta bör den du ska kommunicera med synas. För att justera kamerainställningarna gör du på följande sätt:

Anslut en bärbar dator (alternativt en mobiltelefon, eller en stationär PC) till det lokala nätverket. Starta ett terminalemulatorprogram, såsom exv. Putty eller MobaXterm. Logga in dig på den nod du vill fjärrstyra kameran hos (i detta exempel pc1), med följande kommando:

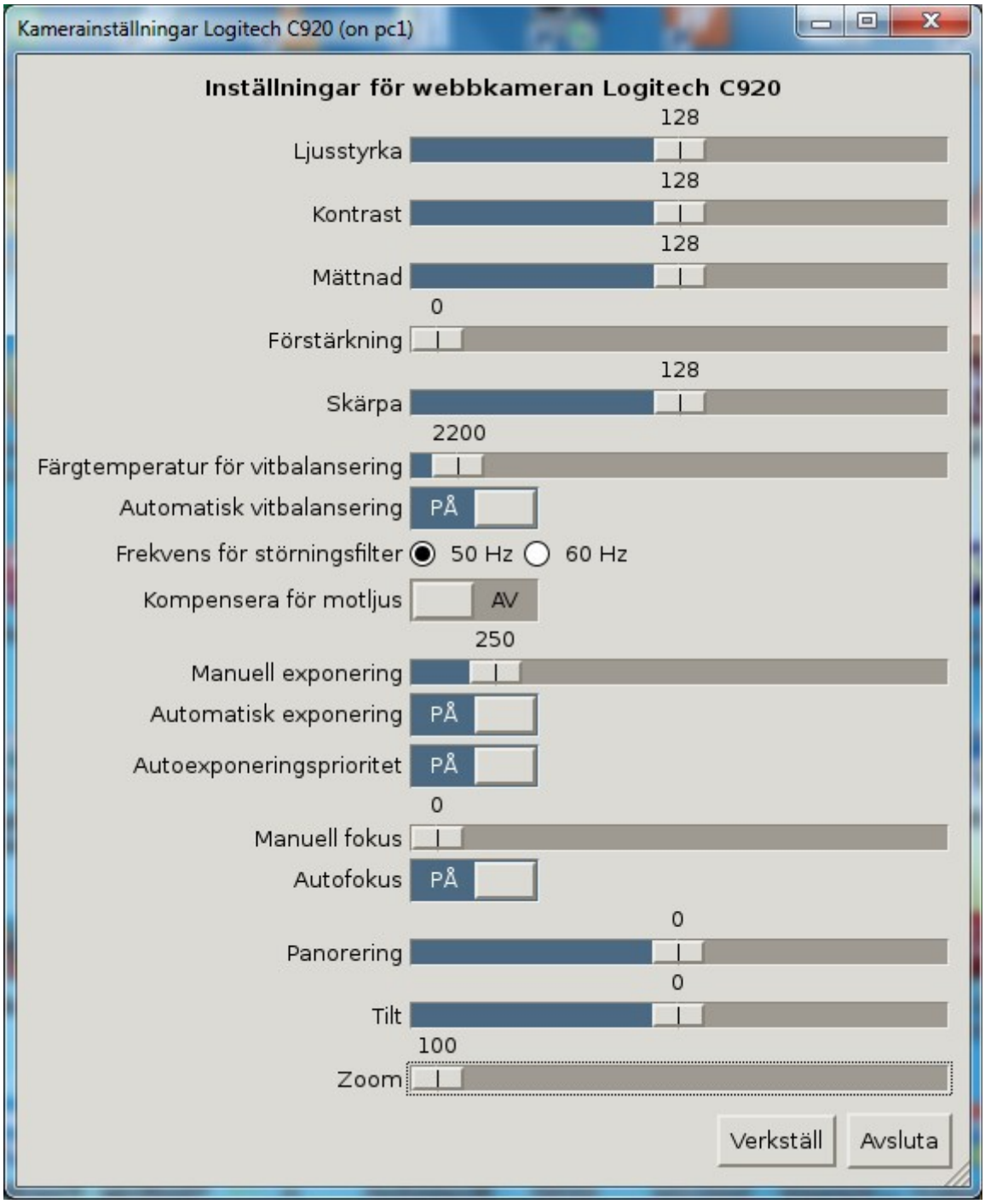
```
ssh -X pc1.local -l pi
```

Ange sedan lösenordet `raspberry`

Detta innebär att du loggar in dig som användaren pi på pc1.local, via ssh, med möjlighet att köra grafiska program från din fjärranslutna dator. Vid terminalprompten kan du nu starta programmet som kontrollerar kameran genom att skriva:

```
./gtk-c920-controller &
```

Efter någon sekund bör nedanstående gränssnitt dyka upp.



Här kan du styra de parametrar du önskar och resultatet syns i realtid på de anslutna bildskärmarna. När du är klar klickar du på Avsluta-knappen.

Skriv sedan kommandot `exit` för att avsluta fjärranslutningen via ssh.

Stänga av en nod

För att stänga av på ett kontrollerat sätt bör du logga in dig på samma sätt som ovan beskrivits.

```
ssh -X pc1.local -l pi
```

Ange sedan lösenordet `raspberry`

Vid kommandoprompten kan du nu ge kommandot:

```
shutdown -h now
```

vilket innebär en omedelbar avstängning av den fjärrstyrda datorn.

Begränsningar

På grund av ekoutsläckningsalgoritmen kan det ibland förekomma små störningar i ljudet, speciellt när man pratar i munnen på varandra.

Bilaga - Tekniska specifikationer

Video

Kodning: H264

Bildfrekvens: 24 fps

Bithastighet: 1 Mbps

Bredd: 1080 pixlar

Höjd: 1920 pixlar

Sändarport pc1.local: UDP:5001

Sändarport pc1.local, förtitt: UDP:5003

Sändarport pc2.local: UDP:5002

Sändarport pc2.local, förtitt: UDP:5004

Audio

Kodning: MP3

Bithastighet: 128 kbps

Antal kanaler: 1 (mono)

Ekoutsläckning via funktion i ljudmodulen PulseAudio

Sändarport pc1.local: UDP:6001

Sändarport pc2.local: UDP:6002

Totalt bandbreddsbehov med nuvarande parameterkonfiguration = 1,5 MB/s