Produktbeskrivning - Programvara till Realtidskamera

Inlämningsuppgift EDA040

Oskar Holmberg dat12oho
Erik Munkby dat12emu
Kasper Bratz dat12kbr
Niklas Karlsson dat12nka

Inledning

Produkten består av tre delar. Client, delen av programmet som tar emot bilder från nätverket och hanterar dem på nödvändigt sätt. Server, delen av programmet som körs “på” kameran/kamerorna. Packeterar och skickar bilder över nätverket. GUI, delen av programmet som visar bilderna och har funktioner för att hantera kamerorna. Produkten klarar ett flertal kameror genom ändring på konstanten kallad NBR_OF_CAMERAS, dock behöver GUI koden ändras för att kunna visa bilder från fler än två. Systemet befinner sig alltid i ett av tre lägen (modes). Idle som uppdaterar bilden en gång var femte sekund, movie som visar bilder i ett kontinuerligt flöde och auto som uppdaterar bilder lika ofta som idle tills rörelse (motion) upptäcks och går då över till att visa bilder på samma sätt som movie mode. Systemet kan även vara i synkroniserat/osynkroniserat läge (synchronous/asynchronous mode) där synkroniserat läge gör att bilder sparas i en *buffer* innan de visas för att han en konstant tid mellan visning.

Våra erfarenheter

I stora drag har utvecklingen av projektet flutit på bra men det har funnit en del saker som kunde gjorts bättre. De första problemen vi stötte på var redan första veckan. Ingen i gruppen hade någon större erfarenhet av git vilket ledde till att en del .class filer och andra saker som inte skulle vara där följde med upp i repot och liknande. Nu har vi dock bättre erfarenhet av git vilket skulle sparat oss mycket tid om projektet skulle göras om idag. Ett annat problem som framkom under första veckan var att ingen av oss hade mer än väldigt lite kunskap om hur sockets fungerade. Detta ledde till att planeringen av vilka trådar och klasser som behövdes försvårades lite. Nästa problem vi stötte på var p.g.a. att vi till en början utvecklade systemet med bara en kamera i åtanke och att få denna att fungera som den skulle. När vi sedan skulle skala om detta till flera kameror krävdes en hel del refaktorisering och ändringar av klasser. Möjligtvis hade det varit enklare att utveckla systemet för 2 eller flera kameror redan från början. Den sista saken som kunde gjorts annorlunda är att vi kunde varit snabbare på att ta datorer i salarna med kamerorna vi testade systemet med. Det faktum att vi satt i en sal långt från kamerorna ledde till en hel del springande fram och tillbaka mellan salarna. I stora drag var kommunikationen i gruppen väldigt bra och det var ett väldigt roligt projekt.

Nätverk

Det finns två klasser som sköter det mesta av nätverkshantering, på serversidan kallas klassen för **`ServerSocketHandler`** och på klientsidan heter klassen **`ClientSocket.`**

`ServerSocketHandler` hämtar ett **`ServerSocket`** objekt från **`ServerMonitor`** och börjar sen lyssna efter anslutningar. När en klient ansluter så skapar **`ServerSocketHandler`** ett **`TCPIPBuilder`** objekt och startar dennes tråd. **`TCPIPBuilder`** har inget stort ansvar utan det enda som det gör är att hämtar nya bilder från **`ServerMonitor`** och sen skickar dem till klienten. Är det så att det inte går att skicka någon data så kommer **`TCPIPBuilder`** stänga ner nätverksanslutningen. Paketen som

TCPIPBuilder skickar byggs i klassen **CameraReader.** Paketen har varierande längd men ser ut som följade:

[bildinformation](4)[rörelse](1)[tidsstämpel](8)[bilddata](varierat)

De fyra första byten säger hur stor resten av paketet är. Efter det så är det en byte som säger om det är rörelse eller inte i bilden. Är det rörelse så är det en etta, annars nolla. Sen kommer det en tidsstämpel som är åtta bytes stor, och denna tidsstämpel är den tidpunkt då kameran tog bilden och vi hämtar den från kameran. Sist så kommer bilden, och bildstorleken varierar beroende på vad som finns i bilden.

När ett **TCPIPBuilder** objekt har skapats så väntar **ServerSocketHandler** efter inkommande data. Paketen från klienten är bara en byte stor och byten säger till servern om vilket läge som systemet är i, till exempel om det är i force idle eller auto.

När anslutningen mellan klienten och servern bryts så börjar servern lyssna efter inkommande anslutningar.

På klientsidan så är det **ClientSocket** som sköter det mesta med nätverksanslutningarna. När ett **ClientSocket** körs så skapar den en socket och försöker ansluta till en ip-address och en port. Den kommer försöka att skapa anslutningen ända tills den lyckas göra det. När en anslutning har skett så skapar **ClientSocket** ett **ServerWriter** objekt och dess uppgift är att skicka ett paket till servern varje gång klientes läge ändras.

Det **ClientSocket** gör är att den först läser in 4 bytes för att se hur stort resten av paketet är och sen läser den tills den har läst tillräckligt många bytes. När detta är gjort så skickar den paketet till **ClientMonitor.** Är det så att nätverksanslutningen bryts så kommer **ClientSocket** försöka återansluta till servern såvida det inte blev nerstängt av användargränssnittet.

Client

Klient delen av programmet kretsar runt en monitor kallad **ClientMonitor**. Den innehåller större delen av logiken runt bildhantering och bildimportering. När klienten får in ett nytt paket plockas det isär i till ett flertal delar (se nätverk). Därefter läggs de i en buffert med en väntetid som räknas ut enligt formeln:

$$WaitTime = Delay\ modifier - network\ travel\ time$$

Där *delay modifier* är en fixt konstant på 300 ms (bilder kommer alltid visas 300 ms efter att de tas), och network travel time är tiden det tar för bilden att resa över nätverket.

ClientSocket är tråd klass som kopplas på nätverket för att hämta bilder från servern.

ServerWriter är en tråd klass som kopplas på nätverket för att meddela servrarna vilket läge (mode) som ska aktiveras.

ImageClass är en klass som bilder sparas i tillsammans med tiden de ska visas och tiden det tog för respektive bild att färdas över nätverket, och sedan läggs i bufferten.

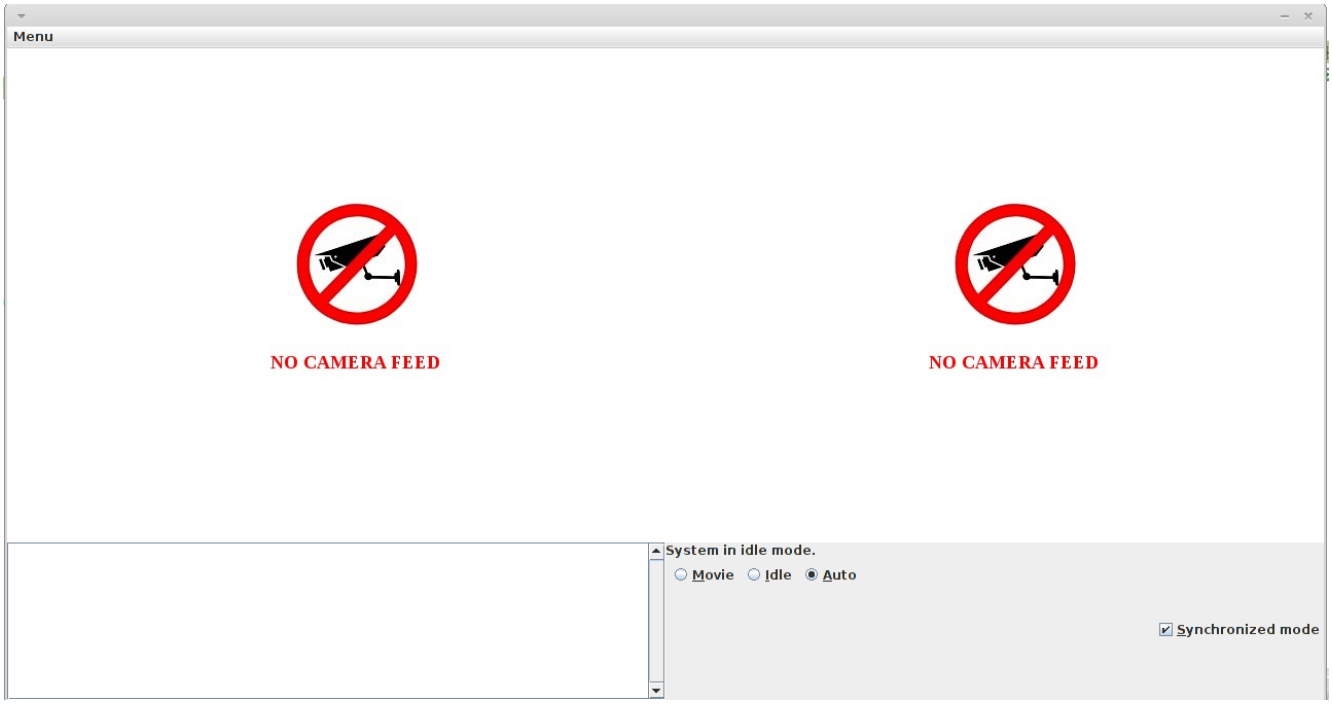
JPEGHTTPClient lägger upp bilderna på en http sida.

`ClientMain` startar hela klient delen av programmet.
`Constants` är en statisk klass som innehåller alla konstanter som används i produkten.
Vad metoder i respektive klass gör är beskrivet i java doc.

Server

Server sidan av programmet har tre ansvarsområden.
Det första är att via camera-API'n hämta hämta och behandla bilder som tas från kameran. Detta görs i klassen **CameraReader**. Denna tråd försöker hämta bilder från kameran så fort kameran tillåter. Ur bilden tas sedan en timestamp och motion kontrolleras. All denna information packas sedan ihop till en byteArray med formatet [*packeLength(4)motion(1)timestamp(8)image(len)*] och skickas till **ServerMonitor**.
Det andra ansvaret servern har är att via klassen **JPGHTTPServer** ta emot HTTP requests och att returnera den senaste bilden som sparats i monitorn. Detta ger möjligheten att se snapshots av systemet via internet.
Det sista ansvarsomådet servern har är kommunikation med clienten. Detta sker i klasserna **ServerSocketHandler** & **TCPIPBuilder**. **ServerSocketHandler** väntar på att monitorn får en ny bild och skickar sedan dess paket vi en socket (Se nätverk). Den lyssnar även efter paket från client-sidan vilka beskriver beteendet servern ska ha för skickandet av bilder.
ServerMonitor har ansvar för att lagra gemensama variabler för servern. Beroende på vilket läge kamera är i (Force movie och idle eller automatiskt) lagras bilder från kameran. Är systemet i movie lagras varje ny bild och är systemet i idle sparas endast var 125e bild(då kameran kör i 25hz).

GUI



Grafisk beskrivning

Det grafiska gränssnittet består av en meny med val att lägga till en kamera (skapas i metoden `GUI.addMenu()`), två intilligande kamerarutor som kan visa bilder (`JPanel camDisplay`), ett textfält där aktuell information om inre processer presenteras(klassen `MessageArea`), samt ett fält för diverse olika inställningar för kameraläge(klassen `CameraModePane`, metoden `addSyncCheckbox()` samt `JLabel systemMode`).
För att lägga till en kamera klickar användaren på “Menu” högst upp till vänster i fönstret, väljer sedan “Add camera” ur menyn som visas. Det dyker då upp ett fönster där ip-adressen till kameran som ska läggas till ska fyllas i. Tryck “OK”. Ett nytt fönster dyker upp där motsvarande port till kameran ska anges. Tryck “OK”. Metoden `GUI.addCamera()` körs och är all inmatning korrekt och kameran är online så dyker en bild av kameran upp i nästa lediga kameraruta.

Klassbeskrivning

Gränssnittet består av två huvudklasser, `GUI` och `GuiThread`, där `GUI` är klassen som bygger upp gränssnittet och har metoder för att uppdatera det. `GuiThread` (skapas en per kamera som läggs till) har sedan ansvar att köra uppdateringsmetoderna i `GUI`. Det finns också ytterligare klasser som används för att skapa diverse fönster och knappar i gränssnittet för att slippa ha den koden direkt i `GUI`.

Metodbeskrivning

Klassen GUI innehåller ett antal metoder med syfte att bygga upp själva gränssnittet. De mest intressanta metoderna är de som behandlar uppdatering och funktionalitet, följdaktligen:

`refreshImage`

Kallas av klassen GuiThread(), tar emot argumenten; `byte[] newPicture, boolean movieMode, int cameraNbr, long traveltime` och uppdaterar kameran på plats `cameraNbr`, med bilden representerad i `newPicture` i gränssnittet. `traveltime` och `movieMode` används för att för varje bilduppdatering visa aktuell fördröjning för kameran samt huruvida systemet är i “movie mode” eller ej.

`addCamera`

Tar emot argumenten `String host` och i`nt port` och lägger till en kamera med `host` som ip-adress och `port` som port.

`removeCamera`

Tar emot argumentet `int camNbr` och tar bort kameran på plats nummer `camNbr` från gränssnittet samt stänger ner klientens kontakt till denna kamera.

Användarmanual

Installation:
1. Starta terminalen.
2. Start av proxykamera enl. EDA040 hemsidas anvisningar.
3. Navigera till mappen jars som innehåller de körbara .jar filerna.
4. Starta de båda jarfilerna i valfri ordning.
(lägg märke till att det behövs en server.jar körandes för varje kamera)

Server.jar

Navigera till mappen som innehåller server.jar via terminalen. Starta server.jar genom att skriva "`java -jar server.jar args[0] args[1] args[2]`"
i terminalen. Där args[0] är klientporten, args[1] är kameraaddressen,
args[2] är serverporten.

Client.jar

Starta client.jar genom "java -jar client.jar".
5. För att lägga till en kamera välj från dropdown menyn "Menu" alternativet "Add camera".
6. Skriv in de efterfrågade uppgifterna i pop-up rutan.
7. För att ta bort en kamera välj istället “Remove camera” och mata in siffran som motsvarar kameran som ska tas bort (1 för första kameran).