CHALMERS

Sensorviewer 3D Core

Visualisering av mätdata kring ett tredimensionellt objekt

Examensarbete inom högskoleingenjörsprogrammet Dataingenjör

JOHAN SANDSTRÖM
DANIEL NICKLASSON

Institutionen för data- och informationsteknik
Avdelningen för Datorteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige, 2012

Abstract

This projects purpose was to simplify tests on products by measuring different kinds of values and then visualise the results in a computer software. The tests that the software should be able to perform are for example measuring the difference in temperature on the surface of an item or product to determine how the end product would be. This project was focusing on the core of the software which will visualise results from the hardware and also the communication between software and hardware interfaces. Moreover, the software was written in Java and it needs to modular to be able to expand the software in the future and to use different kinds of sensors and robots. We needed to determine what kind of software is needed to do these visualisations in a correct and user-friendly way. This project was a collaboration between a couple of groups which was working on different parts on the project. In this project we have been working closely with the group that was working on the graphical interface for our software. We have also been working with another group who was working on the hardware end of the project, which is a small robot that holds the sensors to be able to perform tests. The resulting software has some basic functionality that is needed to visualise the measured data together with the object. We discovered a problem that had to be solved which was the calibration between the hardware and the software. This project was held during spring of 2012 on Chalmers for the “Data och IT”-department.

Sammanfattning

Detta projekts syfte var att underlätta utförande av test av en produkt, och visualisering av dessa test. Tester som utförs kan vara temperaturskillnader runt ett objekt för att se hur objektet beter sig i praktiken. Projektet innefattar utvecklingen av kärnan till en programvara som visualiserar mätresultat och kommunicerar med en robot som utför testen. Mjukvaran skrevs i programmeringsspråket Java och behöver vara modulärt för att stödja utbyggningar och många olika sorters sensorer och robotar. Vi tar även reda på vad som krävs i programvaran för att kunna visualisera datan på ett korrekt och användarvänligt sätt. Detta projekt utfördes i samarbete med flera projektgrupper. Närmast samarbete har vi haft med projektgruppen som utvecklat det grafiska gränssnittet till programmet. Vi har även haft ett nära samarbete med en projektgrupp som utvecklat en robotarm för att utföra tester med. Den konstruerade mjukvaran har grundfunktionaliteterna som behövs för att visa upp mätdatan tillsammans med objektet. Vi upptäckte att ett stort problem som behövdes lösas var en kalibreringen mellan mjukvara och hårdvara. Projektet utfördes våren 2012 på Chalmers Tekniska Högskola för institutionen Data - och Informationsteknik.

Keywords: measurement, Java, visualisation, sensor, calibration

Innehållsförteckning

Terminologi

Här listas vanliga tekniska ord och uttryck inom bla. programmering som kan vara bra att ha lite kunskaper om för att kunna förstå innehållet av rapporten bättre.

API - Application Programming Interface

ASCII - En standard över teckenkodning

C++ - Ett objekt-orienterat programmeringsspråk

COM-port - Ett kommunikationsinterface som används för att skicka och ta emot data

CPU - Förkortning för Central Processing Unit. Ofta kallad hjärnan i en dator.

Eclipse - En utvecklingsmiljö som används för att skriva och testa kod

EDT - Event Dispatching Thread, är tråden som ritar upp grafiska komponenter i Java Swing

Event - Händelse som generar ett programanrop t.ex knapptryckning på tagentbordet

Git - Distribuerat versionshanteringssystem

GitHub - Hemsida för projecthosting. Projekten använder Git som versionhanteringssystem

Java - Ett objekt-orienterat programmeringsspråk

Java3D - Högnivå 3D-grafik bibliotek för Java

JavaFX - Ett bibliotek med modernare standard för grafiska komponenter inom Java

JOGL - Lågnivå 3D-grafik bibliotek för Java

Mikroprocessor - En CPU som ligger i en komponent.

Objekt - Ett verkligt fysiskt föremål eller en 3D-figur på skärmen som visas i programmet

Obj - Ett filformat för 3D-objekt

Polygon - Ett samlingsnamn för geometriska figurer som består av ett ändligt antal sträckor

STL - Ett filformat för 3D-objekt

Swing - En del av Java som hanterar grafiska gränssnitt

Tråd - En process som körs parallellt med andra processer i operativsystemet

Vertex - Ett hörn i en polygon. (En punkt i 3D-rymden)

1. Inledning

1.1. Bakgrund

Med hjälp av mjukvara borde det vara möjligt att kunna göra exakta mätningar och kunna visualisera det i en 3D-vy där man enkelt kan sammanställa en mängd information, manipulera objekt, göra grafer över indata osv. Denna typ av programvara kan vara ett bra verktyg för företag och kanske även privatpersoner som skall göra olika typer av mätningar på ett föremål. Till exempel skulle programmet kunna visualisera magnetfältet från ett magnetiskt objekt eller temperaturskillnader i ett rum eller runt en motor. Mycket av arbetet i att göra mätningar och manuellt skriva in resultaten i ett 3D-hanteringsprogram skall alltså automatiseras på ett bra sätt med hjälp av ett mjukvaruverktyg.

1.2. Problem

Det finns för närvarande inget program som kan ta in ett inläst 3-dimensionellt objekt och mätdata för att visualisera båda delarna tillsammans. Att koppla samman ett verkligt objekt och göra mätningar för att sedan skicka resultaten från mätningarna in i en 3D-vy över ett likadant virtuellt objekt är ett outforskat område som den här rapporten kommer gå nära in på. Eftersom det inte finns något liknande program så vet vi inte i uppstartsfasten vad programmet behöver för att bli användarvänligt. Därför skall programmet byggas så att det är enkelt att bygga ut.

1.3. Syfte

Uppdraget var att skriva ett program som tillhandahåller funktioner för att importera ett inläst 3-D objekt samt mätdata. Med programmet skall man alltså kunna göra det smidigare för en användare att få en omfattande överblick över t.ex hur magnetfältet ser ut kring ett kretskort. Av detta lärde vi oss hur vi kan strukturera framtida mjukvaruprojekt när det gäller outforskade områden. Vi behövde bland annat ta egna beslut om hur designen skall se ut på programmet och göra bedömningar över vad som är lämpligt att använda och var. Dessutom är denna typ av programvara väldigt ny och därför var det spännande att se vad liknande programvara kan göra i framtiden. Projektet har även tvingat oss till att lära oss nya programvarutekniker.

1.4. Avgränsning

För att begränsa omfattningen så handlade detta examensarbete endast om utvecklingen av kärnan till programvaran. Kärnan tar hand om objekt från en 3D-scanner och mätdatan från

sensorer när de först kommer in i programmet. Det är också kärnans uppgift att kommunicera med mikrokontrollern som placerar ut sensorerna. Vi kommer heller inte att gå in på hur kontrollen av robotarmen fungerar och vinkelberäkningarna som krävs för att styra den.

1.5. Samarbeten

I detta projekt har vi samarbetat med två andra grupper. Den ena gruppen, SensorViewer 3D View, utvecklade det grafiska gränssnittet för mjukvaran. Den andra gruppen behandlar utvecklingen av en robotarm som skall utföra mätningar samt ta emot och skicka data. Dessa grupper har skrivit sina egna rapporter och de finns refererade i källanvisningen.

2. Metod

I detta kapitel beskriver vi hur vi jobbade, vad för tidsplaner och deadlines vi hade och vilket typ av arbetsmetod som vi arbetade efter. Vi tar även upp hur rapporteringen till handledaren går till och hur vi arrangerade möten för att diskutera projektet.

2.1. Arbetsmetod

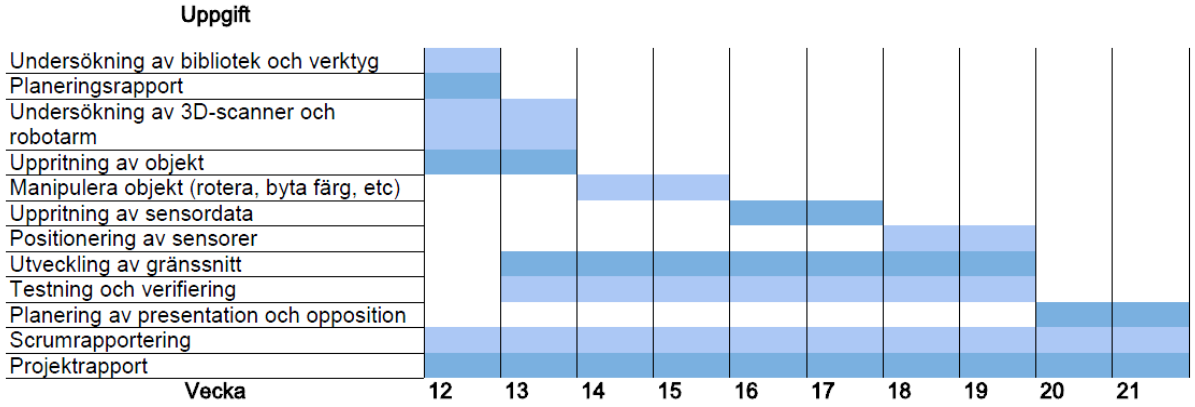
I det här projektet har vi använt oss av en anpassad variant av en systemutvecklingsmetodik som kallas för Scrum. Detta innebär att vi varje dag skriver en logg vad vi jobbar med, hur det gått och vad vi skall göra härnäst men även också vad vi eventuellt kan ha för problem. Utöver loggen så har vi haft möten med vår handledare minst en gång i veckan för att diskutera hur arbetet fortgår. På dessa möten diskuterar vi var vi står i projektet, om vi har några förhinder och vad som behövs för att kunna hålla de utsatta deadlines vi har satt. Om vi inte har möjlighet att bli klara efter utsatta deadlines får vi komma fram till en gemensam plan där vi bestämmer vad vi kan komma få ut av tiden vi har kvar.

Ett verktyg som vi har haft stor användning av under detta projekt var versionshanteringssystemet Git som vi har använt när vi skrivit våran programkod i utvecklingsverktyget Eclipse. Detta versionshanteringssystemet skapades av bla. Linux-grundaren Linus Torvalds i brist på bra alternativ för att kunna hantera stora mängder källkod. När det är flera personer som arbetar på samma program behövs ett sätt att synkronisera kodändringar, kodtillägg och refaktoreringar. Det är då Git och hemsidan GitHub kommer in i bilden och hjälper oss med detta. Varje användare som skall bidra med kod skapar ett konto på GitHub för att identifiera sig med för att sedan ha möjligheter att hämta ner den senaste revisionen av projektet och även kunna skapa egna delprojekt(Forks) utifrån grundprojektet. Om det till exempel är två personer som arbetar i samma fil men på olika ställen i koden så har Git en smart metod som kallas för Git-merge som smälter samman kodavsnitt från olika revisioner och skapar sedan en sammansmält version av dessa. Med Git-merge undviker vi en hel del problem med synkronisering som kallas för merge-conflicts. I Eclipse finns det inbyggt stöd för Git och detta har varit enkelt att använda. [1]

Vårt projekt är beroende av flera andra projektgrupper. Dessa projektgrupper är: gruppen som gjorde grafiska gränssnittet till vårt program[2] , gruppen som jobbade med styrningen av robotarmen[3] och andra projektgrupper som jobbade med olika slags 3D-skannrar. Detta samarbete har krävt många möten mellan de olika grupperna. Vi har jobbat särskilt nära med gruppen som har gjort det grafiska gränssnittet till programmet.

2.2. Planering

Det första vi gjorde var att forma en planering över projektet. I vårt projekt valde vi att skapa ett Gantt-schema som är en typ av flödeschema indelat i olika faser. Några av dessa faser valde vi att ha som kontinuerlig process, detta var bla. testning av kod, rapportskrivning och scrumrapportering som vi återkommer till nedan. Utöver de kontinuerliga processerna så valde vi att dela in olika delar av projektet i två-veckors delmål.



Figur 2.2.a. Gantt-schema över vår ursprungliga planering

2.3. Kunskapskrav

För att kunna slutföra projektet behövde vi först och främst lära oss hur Java 3D fungerar. För att kunna kommunicera med microprocessorn som styr robotarmen behöver man kunskap om hur man utför seriell kommunikation i Java.

Information vi behöver för att kunna lösa problemet inhämtades främst från internet. På internet har vi läst guider och API:er för de olika bibliotek som programmet är beroende av. Vi har fått mycket hjälp med att strukturera projektet i uppstarten av Johan Onsjö som var med i ett liknande examensarbete 2011 med titeln “Three-dimensional sensor scanner”.[4] Han hade många råd att ge oss för att undvika de problem de hade i sitt examensarbete.

3. Teknisk Bakgrund

I denna delen av rapporten behandlas verktyg, programmeringsspråk och principer som ger grund till de senare delarna av rapporten. Kapitlet beskriver också val av bibliotek som har gjorts.

3.1. Java 3D

Java 3D är ett API (Application Programming Interface) för utveckling av 3D-applikationer i Java. Java 3D är inte plattformsberoende vilket gör det passande att använda till en applikation som ska kunna köras på många olika system. Beroende på vilken plattform man använder så använder Java 3D OpenGL eller Direct3D. Java 3D är inte bara ett bibliotek som ligger runt OpenGL eller Direct3D utan är skrivet med ett objektorienterat synsätt. [5]

Ett alternativ till Java 3D är JOGL (Java OpenGL). Java 3D är ett språk på högre nivå än Java 3D vilket gör det enklare att förstå än JOGL. Eftersom vi är nybörjare på att programmera 3D-applikationer var detta en anledning till att vi valde Java 3D. En annan anledning var att vi i projektet skulle behöva rita upp 3D-objekt från filer. Java 3D har inbyggt stöd för att importera 3D-filer. JOGL har inget sådant stöd så därför hade vi behövt att programmera en egen 3D-fil läsare.

Här visas ett exempel som är något förenklat hur man gör för att rita upp en 3D-sfär på skärmen med hjälp av Java3D-biblioteket och Java Swing. Alla 3D-objekt som vi läser in sparas i liknande strukturer och i exemplet visas även att Java3D är objektorienterat.

Skapa 3D-rymden:

```
SimpleUniverse universe = new SimpleUniverse();
```

Skapa en huvudstruktur att lägga objekt i:

```
BranchGroup group = new BranchGroup();
```

Skapa ett sfär-objekt:

```
Sphere sphere = new Sphere();
```

Skapa ett utseendeobjekt

```
Appearance ap = new Appearance();
```

Skapa en färg(100% röd, 0% grön, 0% blå):

```
Color3f col = new Color3f(1.0f, 0.0f, 0.0f);
```

Skapa färgattribut:

```
ColoringAttributes ca = new ColoringAttributes(col, ColoringAttributes.NICEST);
```

Sätt färgattributen på utseendeobjektet:

```
ap.setColoringAttributes(ca);
```

Sätt utseendet på objektet:

```
sphere.setAppearance(ap);
```

Lägg till sfären i huvudstrukturen:

```
group.addChild(sphere);
```

Lägg till huvudstrukturen till universumet:

```
universe.addBranchGraph(group);
```

Då kommer vi att rita upp en sfär i en 3D-rymden.

Figur 3.1.a. Den uppritade röda sfären.

3.2. 3D-filer

3.2.1. Standard Tesselation Language

Filformatet STL är ett format som hanterar punkter för uppritning. Filen är uppbyggd genom att spara tre hörn med olika positioner i en 3D-rymd med X,Y och Z-koordinater för att forma en polygon. Denna polygon får även en normal kopplad till sig så STL-filen vet även från vilket håll som denna polygon skall synas. Men utöver detta så innehåller inte STL-filformatet mer data, vi vet till exempel inte vilken färg eller vilken textur som objektet har som läses in. Dessa filer kan antingen sparas i ASCII-format eller i binärt format, där det binära formatet är att föredra då filstorleken blir mindre pga. mer kompakt lagring. [6]

Användningsområdena för STL-filer är ganska bred och används främst i 3D-printing sammanhang. Detta innebär att man läser in stl-filen och sedan tillverkar en fysisk 3D-modell lager för lager för att kunna göra olika mätningar på modellen som t.ex är användbart vid produktdesign.

Exempel på STL-fil:

Här är ett exempel som är taget från en 3D-skanning(se kap 3.6) som vi utfört själva. En facet är en polygon som byggs upp av tre hörn, vertexes men som även har en normal. I filen finns

då tusentals facets som tillsammans bildar objektet. Detta är bara en liten del av innehållet i STL-filen.

```
facet normal -8.767923e-001 -2.880923e-001 -3.850171e-001
   outer loop
    vertex  -1.095568e+002 1.563624e+001 2.644070e+002
    vertex  -1.095201e+002 1.563624e+001 2.643234e+002
    vertex  -1.095201e+002 1.552453e+001 2.644070e+002
   endloop
endfacet
```

3.2.2. Wavefront obj file

Obj-filformatet är utvecklat av Wavefront Technologies för att kunna visualisera och animera 3D-objekt. Filformatet är globalt accepterat och används i många olika sammanhang när det gäller grafiska 3D-applikationer. Obj-formatet skiljer sig en del mot STL-formatet då Obj-filer innehåller mer information om hur själva objektet ser ut när det skall ritas ut. Det finns bland annat information om vilken färg och textur objektet har. Men obj-formatet har även likheter med STL som t.ex sättet att spara information på och forma en polygon eller mer generellt en face av dessa, samt en normal som visar ifrån vilket håll som polygonen är synlig.

Exempel på Obj-fil:

Detta är bara en del av innehållet, vanligtvis innehåller Obj-filer en stor mängd av liknande information som nedan.

De följande tre raderna är vertexes, som tillsammans bildar en polygon, de har varsin X,Y och Z koordinat.

```
v 0.79456 1.30449 0.546219
v 0.79456 1.30449 0.416066
v 0.783884 1.301879 0.350989
```

Dessa tre rader bestämmer var texturer skall vara på objektet.

```
vt 0.923774 0.352759 0
vt 0.910421 0.351418 0
vt 0.945893 0.355291 0
```

För att skapa en yta, en face, så tar man ett antal vertexes och bildar då en face av dessa.

```
f 5249/5434 5250/5435 5120/5305
f 5250/5435 5251/5436 5123/5308
f 5251/5436 5252/5437 5123/5308
```

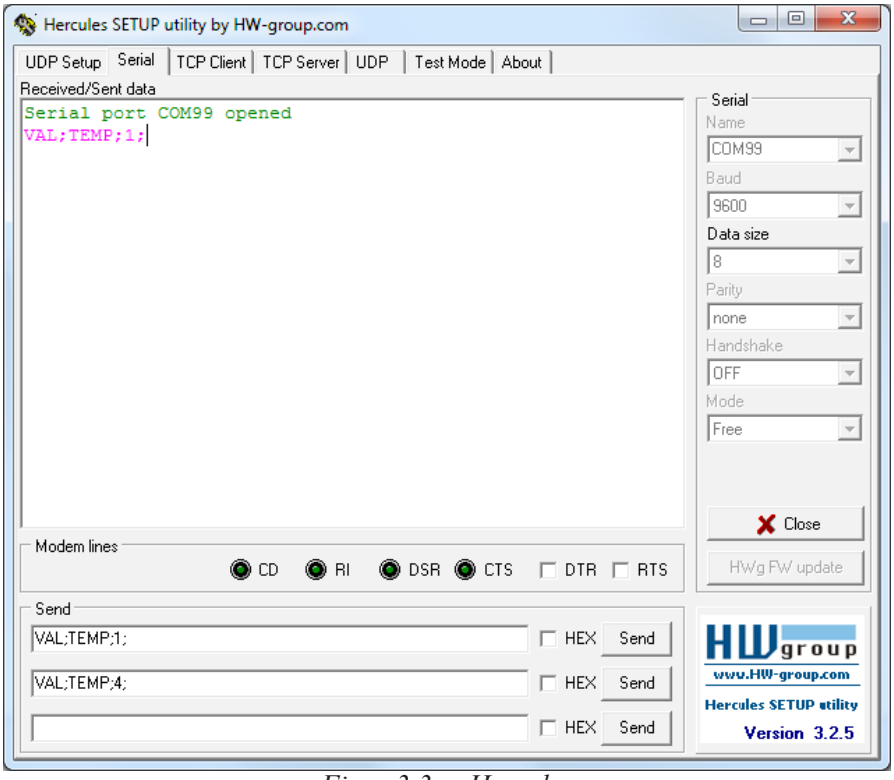
3.3. Seriell Kommunikation

För att kommunicera med robotarmen behöver vi skicka och ta emot data. För att skicka data används lediga COM-portar på datorn som kör huvudprogrammet. Då det inte är stora mängder av data som skall skickas räcker det med standarden RS-232 som är en ANSI-standard för en seriell databuss. För att kunna sätta upp kommunikationen behövs seriella interface och dessa kan enkelt sättas upp med hjälp av externa Java-bibliotek som t.ex RXTX eller Java Communications API. [7]

När man jobbar med seriell data så läser man/eller skriver in bytes i en buffer. Denna buffer bör sedan tolkas och för att veta om datan är bra eller dålig behövs åtgärder och funktioner som har hand om dessa olika fallen. Ett exempel kan vara att vi tar emot en sträng med text som innehåller data från temperatursensor, men datan är felaktig. I detta fall måste vi kontrollera datan och kanske skicka ett svar till källan och meddela att det var fel som inträffade och att vi behöver få datan skickad på nytt.

3.3.1. Emulering av seriell kommunikation

För att testa att seriell kommunikation behövde vi ett sätt att emulera detta på. Först och främst använde vi oss av en mjukvara som heter com0com som skapar två stycken virtuella COM-portar på datorn. [8] Dessa virtuella COM-portarna kan vi nu både läsa och skriva till med hjälp av ett konsolfönster.



Figur 3.3.a. Hercules

COM-portarna är kopplade till varandra så vi kan t.ex skriva på en COM-port 1 och läsa resultatet på COM-port 99.

För att skriva hela strängar kan man använda ett program som heter Hercules. Med detta program kan vi alltså simulera att robotarmen har mottagit en position som den skall gå till, för att sedan gå till denna position och läsa av ett mätvärde och sedan skicka tillbaka till vår programvara.

3.3.2. RXTX

RXTX är ett Java-bibliotek som används för seriell och parallell kommunikation för JDK (Java Development Toolkit) och är baserad på Java Oracle's Communications API. RXTX använder sig av en JNI-implementation (Java Native Interface) för att skapa en seriell eller parallell förbindelse mellan olika interface. [9]

3.4. Mikroprocessor

Mikroprocessorn som kommer ta emot meddelanden från våran programvara är av typen Arduino. En Arduino-enhet är en platform som är baserat på ett enda kretskort som bland annat har stöd för seriell kommunikation. Koden som används för att skriva program för Arduino är ett språk som liknar C++.[10] Denna enhet kommer användas av gruppen som vi samarbetar med som arbeterar med robotarmen och de kommer att skriva programkod för att kontrollera enheten.

3.5. Java Swing

Det grafiska gränssnittet av programmet har skrivits med Java Swing som ett API för att skapa grafiska gränssnitt med Java. Ett alternativ till Swing är JavaFX som är en mjukvaruplattform för att skapa Rich Internet Applications som släpptes i slutet av 2008.[11] Detta examensarbetet handlar inte om utvecklandet av det grafiska gränssnittet till programmet men valet av API för att göra gränssnittet påverkar utvecklandet av kärnan i programmet. Vi upptäckte att Java 3D som är vårt valda 3D-bibliotek inte är kompatibelt med JavaFX. Anledningen till detta är att Sun har slutat utveckla Java 3D för att istället arbeta på ett 3D-API för JavaFX.[12] Efter testande märkte vi att Java 3D universum kan bara ritas upp i Swingkomponenter. Eftersom vi ville använda Java 3D som 3D-bibliotek så valde vi att programmets gränssnitt skulle skrivas med Swing.

3.6. 3D-skanning

För att vi skall kunna rita upp något objekt på skärmen behöver vi först göra en avskanning av ett verkligt fysiskt objekt med en 3D-skanner. Denna skanner har Chalmers tillhändahållit oss att låna för att använda i projektet. Tekniken bakom 3D-skanning är komplicerad och tar tid att utföra. Med hjälp av två kameror, en projektor samt en mycket avancerat programvara används för att utföra kalibreringen och avskanningen. Programvaran jämför ljusstyrkan mellan de olika kamerorna och ritar upp ett 3D-objekt i programvaran. Dessa måste man sedan se till att de överensstämmer då man tar många bilder för att få ett korrekt objekt, man måste ibland manuellt positionerna bilder för att de skall överenstämma med helthetsbilden. 3D-objekten kan sedan sparas i olika format som t.ex Stl och Obj-format. Filerna vi får ut ifrån programvaran är oerhört detaljerade med många punkter som bildar det virtuella 3D-objektet. Då dessa filer är stora och komplicerade behöver vi ytterliggare en programvara för att behandla filerna för att sedan använda dem i vår egen programvara. Vi använder oss av MeshLab för att utföra denna process med att förenkla våra 3D-objekt vi fått från 3D-skannern.

3.6.1. MeshLab

MeshLab är en programvara för att göra ändringar på 3D-objekt. Denna programvara är helt gratis att använda. Man kan använda MeshLab för att sammanfoga punkter på ett 3D-objekt så att komplexiteten av objektet minskar och även informationen som behövs för att spara objektet. Om det finns något stort hål som egentligen skulle vara ifylld med en yta så kan man även använda MeshLab för att fylla i alla dessa ytor för oss och få en bättre version av objektet då 3D-skannern är bra, men skapar inte perfekta avbilder av objekten. I MeshLab finns en funktion som heter Ball-Pivoting som är en algoritm som skapar polygoner utifrån ett moln med punkter vilket kan göra uppritningen av objekt mer lika verkligheten.[13]

4. Genomförande

Här beskrivs hur vi gick tillväga för att lösa vårt problem och vad vi behöver för teknik för att lösa problemet.

4.1. Importering av objekt från fil

För att läsa in en fil i programvaran används först en filväljare som är implementerad i Java Swing-fönstret i huvudmenyn. När man sedan valt en fil kommer denna fil att bearbetas beroende på om det är en stl-fil eller en obj-fil. Om det är en stl-fil kontrollerar vi även om det är en binär eller ASCII-baserad fil då vi måste tolka dessa på olika sätt. Obj-hanteringen sker med hjälp av ett inbyggt Java-bibliotek. I själva inläsningen så bygger programvaran upp polygoner med hjälp av vertexes som ligger lagrade i objektfilerna, då filerna är uppbyggda på olika sätt och har olika egenskaper måste vi ta hänsyn till vad för typ av filerna är som tidigare nämnt.

Varje gång vi läser in ett objekt så behöver vi sätta ett antal egenskaper för 3D-rymden också. För att vi skall se objekt så måste vi sätta ut ljuskällor ifrån olika vinklar som lyser upp objektet för att man skall kunna se det tydligt. Vi sätter även materialegenskaper på objektet för att få en en tydlig yta på objektet.

Inläsningen av objekt kan ta tid då det är många punkter som skall hanteras och därför valde vi att skapa ett fönster som visar att inläsning sker, samtidigt som inläsningen sker i en separat tråd, en SwingWorker-tråd som gör att det grafiska gränssnittet inte låser sig när man läser in objektet. När inläsningen är klar meddelar SwingWorker-tråden programmet att objektet är färdigt att ritas upp på skärmen. Vi provade ett antal sätt att skapa trådar på, men SwingWorker-trådar var det som fungerade bäst eftersom det inte låser gränssnittet när tråden körs.

SwingWorker-trådar används vid tunga beräkningar där man inte vill att Event dispatching-tråden(EDT) låser sig. EDT:n har hand om renderingen av grafiska komponenter och om tråden är upptagen med en tung beräkning kommer inga uppdateringar synas i det grafiska gränsnittet tills beräkningen är klar men det kan framför allt bli problem med event-baserade kommandon som inte registreras som till exempel en knapptryckning som skall ha registrerats från menyn.

4.2. Kalibrering mellan robotarm och mjukvara

Innan vi påbörjade detta projekt tänkte vi inte på att en kalibrering måste göras mellan roboten och mjukvaran. En korrekt kalibrering är enormt viktig för att sensordatan skall visas på

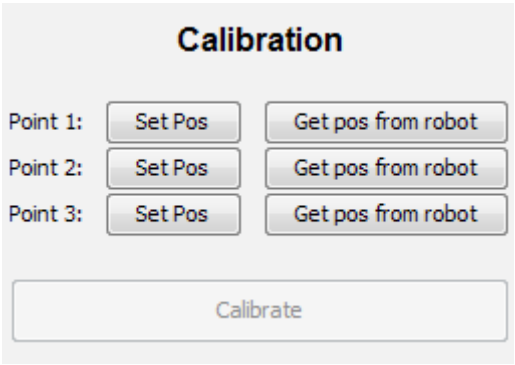
rätt ställe. Problemet är att programmet har sitt eget koordinatsystem och robotarmen har sitt eget koordinatsystem. Dessa koordinatsystem måste synkroniseras så att koordinaterna i programvaran stämmer överens med robotarmens koordinater. Om inte detta är gjort så kommer sensordatan ritas upp på fel ställe och robotarmen kommer läsa av sensordata på fel ställen.

För att göra denna kalibrering så måste det virtuella objektet flyttas i vår mjukvara. Det är vårt programs uppgift att flytta runt, skala upp eller ner, och rotera det virtuella objektet så att det stämmer överens med det fysiska objektet. För att genomföra detta måste användaren först klicka med datormusen på tre punkter på objektet i programmet. Sedan måste användaren manuellt styra robotarmen till samma punkter. De tre punkterna från programmet kommer sedan passas in med de tre punkterna som kommer från robotarmen med de här stegen:

1. Lägg första punkten i mjukvaran på samma koordinat som hårdvarans första punkt.
2. Skala upp eller skala ner det virtuella objektet så det blir lika stort som det fysiska objektet.
3. Rotera objektet så att mjukvarans andra punkt ligger på samma koordinat som hårdvarans andra punkt.
4. Rotera objektet igen så att mjukvarans tredje punkt ligger på samma koordinat som hårdvarans tredje punkt.

Eftersom det finns begränsningar i precisionen hos robotarmen och personen som styr robotarmen kommer inte mjukvarans punkt och hårdvarans punkt vara på exakt samma ställe i relation till objektet. På grund av dessa små fel kommer inte objektet bli fullkomligt synkroniserat med det fysiska objektet. Men vår kalibreringsalgoritm försöker göra det så bra som möjligt och felet borde bli litet.

Kalibreringen är viktig för att programmet skall vara användbart och därför valde vi att göra processen smidig och enkel. För att man skall kunna kalibrera programvaran för att motsvara robotarmens koordinatsystem så behövs tre punkter från hårdvaran som är känd sedan tidigare. Robotarmen måste styras till en punkt i verkligheten och användaren måste även i markera samma kända punkt i mjukvaran för att kunna kalibrera med hjälp av knapparna i programmet som kan ses i figuren nedan. När man har valt ut dessa punkter så kan man välja att kalibrera och då kommer objektet skalas om, flyttas och roteras för att lägga objektet på rätt plats i koordinatsystemet.



Figur 4.2.a. Kalibreringspanelen i programvaran. Alla punkter måste vara satta innan kalibrering kan utföras.

Ovan visas den färdiga panelen i programmet som hanterar kalibreringen. De sex punkterna kan tilldelas i valfri ordning och alla punkterna måste tilldelas innan kalibrering kan utföras.

4.3. Seriell kommunikation mellan robotarm och mjukvara

Den seriella kommunikationen sker via seriella portar och för att datan som skickades skall tolkas på ett korrekt sätt så skapades ett enkelt API. Detta API innehåller regler om hur datan skall organiseras när den skickas över portarna. API:t måste bland annat innehålla information om att datan som skickas antingen är ett mätvärde eller ett kalibreringsvärde. Med tanke på detta har vi skapat en miljö där vi kan i princip ha vilken typ av mätvärde vi vill. Eftersom vi från början ville att programmet skulle vara modulärt så har vi gjort det möjligt att utöka programmet för att t.ex mäta alltifrån temperaturer till att mäta styrkan och riktningen på ett magnetfält.

För att se att programmet uppförde sig korrekt använde vi oss av en två program för att simulera att kommunikationen fungerar mellan hårdvara och mjukvara. Vi använde oss av “Hercules” för att skicka data och simulera att denna var robotarmen. Vi använde “com0com” för att skapa virtuella seriella portar som programmet kunde ansluta sig till. Dessa verktyg beskriver vi i kapitel 3.3.1.

Vid seriell kommunikation där man skall skicka data av olika typ så behövs ett API som säger hur meddelanden skall se ut. Efter att ha diskuterat vad som skall skickas så bestämde vi oss att vi behöver skilja på mätdata och kalibreringsdata. Mätdatan skall bara innehålla ett mätdata som till exempel kan vara en temperatur och kalibreringsdatan skall innehålla en punkt i en 3D-rymd så den behöver alltså X,Y och Z. Vi bestämde oss för att inleda med en identifierare som säger vad för typ av data det är och separera med semikolon som avskiljare.

Ex. på kalibreringsdata: CAL;-4.3;0.2;9.5
Ex. på mätdata: VAL;TEMP;89

Vi tar emot dessa meddelanden som event, vilket innebär att så fort vi får ett meddelande så tas det om hand av en speciell Java-klass som tolkar meddelandet och utför ett arbete beroende på vad för meddelande det är. Ett mätdata till exempel, skall först typbestämmas och sedan sparas undan i en trädstruktur över mätdatan som finns i högra delen av programmet. I trädstrukturen skall man även kunna få möjligheten av påverka representationen av mätdatan visuellt med egenskaper som färg, synlighet och storlek.

Vid kalibreringen skall vi invänta ett värde från robotarmen och för att inte programmet skall låsa sig och vänta förgäves på mätdata så utvecklade vi en timeout-funktion som gör att programmet slutar vänta på ett kalibreringsvärde och meddelar användaren detta.

För att frånskilja felaktig data har vi bestämt att strängarna som vi får som indata måste antingen börja med “CAL” eller “VAL” för att den skall vara giltig, annars ignorerar vi den.

4.4. Sensorinläsning

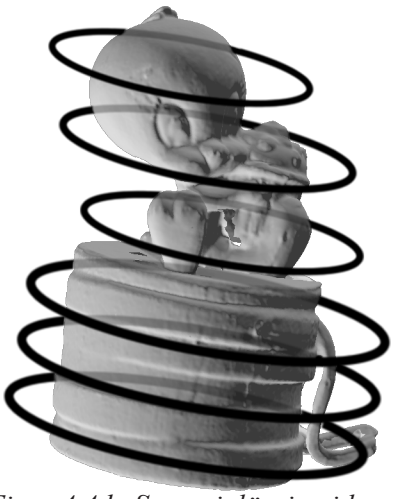
Vi valde att göra det enkelt först och låta programmet kunna göra en enkel mätning i taget. Efter att ha kalibrerat kan vi nu göra mätningar. Vi skickar då en position till robotarmen som skall förflytta sig till denna position och sedan göra en mätning. Mätresultatet skickas via den seriella kommunikationen och vi lägger in den i vår datastruktur och i vår trädstruktur.

Utöver den enkla typen av mätning så hade vi även idéer om automatiserad mätning runt ett objekt. En teknik vi funderade på var bla. att gå runt objektet i en spiralformad bana i ett enda svep där vi ökar höjden lite hela tiden.

Figur 4.4.a. Sensorinläsning i en spiral runt objektet

Förutom spiralmetoden så hade vi tankar om att använda en lagermätning där vi mäter på ett antal punkter på en viss höjdnivå där man t.ex börjar mäta från botten av ett objekt, sedan

ökar vi höjden på sensorn och fortsätter mäta på ett nytt lager och så fortsätter vi att mäta lager för lager. Vi mäter tills vi kommer till en höjd där man inte träffar någon punkt på objektet längre och då vet vi att vi är klara med mätningen.



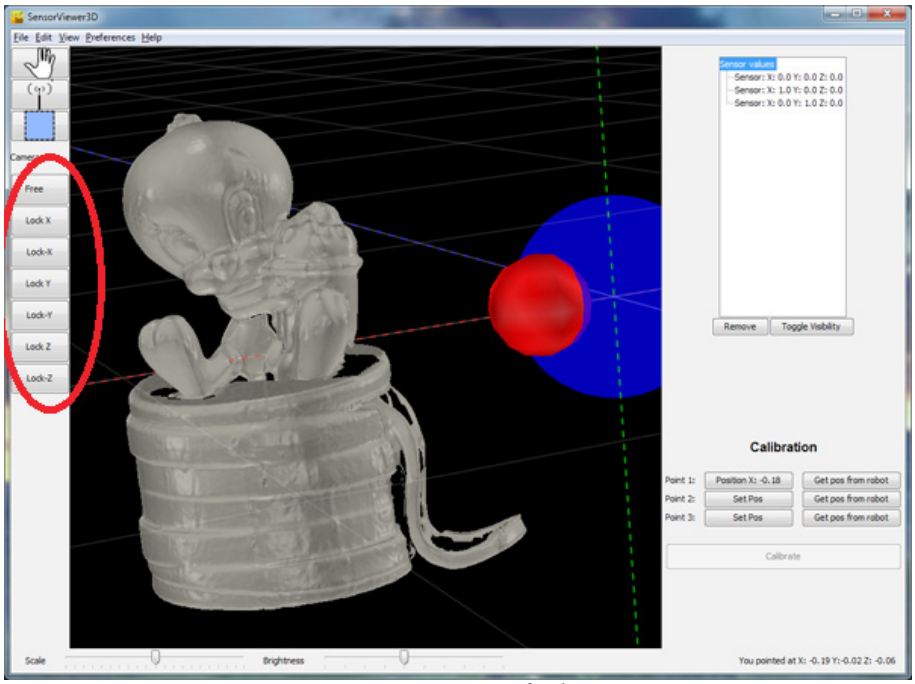
Figur 4.4.b. Sensorinläsning i lager

Eftersom objekt som man vill mäta på sällan har en enkel form så blev dessa automatiserade mätningar snabbt ett komplext problem. Tanken från början var att en mätning av ett objekt skulle helautomatiseras med att t.ex genom att gå kring i en spiralrörelse runt i ett objekt och skicka data. Denna funktion var något som vi fick tidsbrist i att implementera. Istället kan man göra mätningar på en punkt i taget, men vi hade även idéer om hur vi skulle kunna göra för att implementera den helautomatiserade metoden. På grund av tidsbrist kunde vi tyvärr bara fokusera bara på den enkla lösningen där man mäter på ett ställe i taget.

4.5. Grafiskt gränssnitt

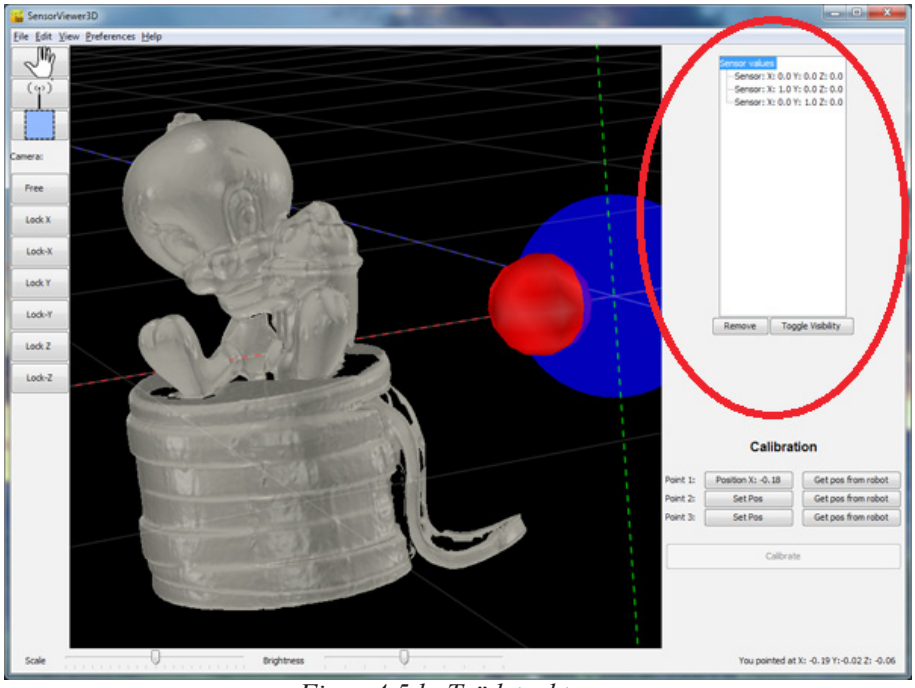
Denna del av programmet, det grafiska gränssnittet, beskrivs främst i en annan rapport som skrevs av den grupp som jobbade med det grafiska gränssnittet.

Med hjälp av Java3D kan vi rita upp objekt som vi har laddat in via parsingfunktioner som arbetar med polygoner. I programmet finns det möjlighet att zooma in och ut och rotera runt objektet på ett enkelt sätt med hjälp av kamerafunktioner.



Figur 4.5.a. Kamerafunktioner

Mätvärderna visualiseras med en sfär som har olika färg beroende på mätvärde och sensortyp. Man kan även göra mätvärdena osynliga genom att gömma dem via trädstrukturen i höger-panelen eller om man vill ta bort ett mätvärde som kanske är felaktigt.

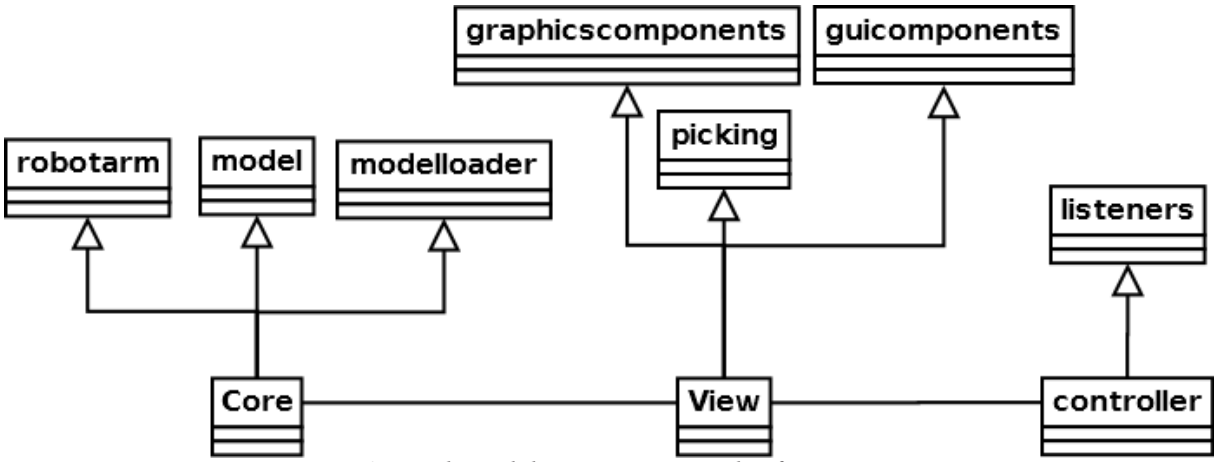


Figur 4.5.b. Trädstrukturer

Eftersom det finns olika typer av mätdata valde vi att skapa en Java-klass som håller reda på hur mätdatan visualiseras. Detta gör att programmet är lättare att bygga ut om du till exempel skall lägga in en ny typ av sensordata som skall visualiseras med en speciell färg.

4.6. Design

Kodningsprincipen vi valde kallas för MVC-modellen som innebär att man separerar modellen, grafiken och beräkningarna i olika undermoduler. Denna princip är vi väl bekanta med och den har fungerat bra tidigare så därför valde vi att även använda den i detta projekt. I källkoden skapar vi då en paketstruktur där paketen och undermodulerna i paketen har en tydlig avgränsad funktion.



Figur 4.6.a. Paketindelningen som gjordes för programmet

I bilagan till denna rapport finns en mer detaljerad bild av vad varje paket innehåller.

5. Resultat

Detta kapitel behandlar resultatet som vi fått fram efter att vårt försök att lösa problemet. Vi jämför hur resultatet ser ut sett utifrån problemställningen.

Vi har skapat ett program som kan ta ett inskannat objekt som det går att läsa om i kap 4.1 samt kap 3.6 och 3.6.1. För att sedan läsa in objektet i programmet och visa upp det på skärmen i programmet så beskrivs detta i kap 4.5. Hädanefter för att synkronisera verkligheten med 3D-rymden behövdes en kalibrering mellan hårdvara och mjukvara och det går att läsa om i kap 4.2.

För att sedan kunna utföra mätningar så har vi utvecklat ett API för den seriella kommunikation för att skicka och ta emot data som behandlas i kap 4.3 och för att läsa mer om vilka verktyg som användes för testning av programmet kan man läsa om det i kap 3.3.1 och 3.3.2. Mätningarna kan bara utföras på en koordinat i taget. Användaren måste själv markera en punkt där en mätning ska göras. Planerna att göra en automatiserad inläsning av flera sensorvärden var vi tvungna att ge upp på grund av tidsbrist. Våra planer för sensorinläsningen beskrivs i kapitel 4.4.

I kap 4.5 beskrivs det grafiska gränssnittet och för att göra programmet enkelt och användarvänligt valde vi att lägga in knappar med tydliga funktioner för olika behov. Denna del av programmet går att läsa mer om i en annan rapport som är skriven av Christian Fransson och Simon Ivarsson som vi haft tätt samarbete med under utvecklingen av programvaran. [2]

Designen som beskrivs i kap 4.6 hade ett viktigt syfte för att kunna göra det möjligt att bygga ut programmet. I och med att vi har strukturerat och radkommenterat samt skapat klass-dokumentation så går det enkelt att bygga ut och underhålla programmet.

6. Slutsats

I slutsatsen kommer vi in på vad vi kunde gjort bättre om vi fick göra om projektet med våra nya kunskaper. Vi tar även upp hur programvaran vi utvecklat kan användas i kommersiellt syfte och hur man kan vidareutveckla idén. Det beskrivs även om hur viktigt det är med struktur och beslutsfattande.

6.1. Resumé

Detta projektet handlar om datavisualisering av sensorer runt tredimensionella objekt. I projektet ingår utvecklandet av en programvara som ska visualisera testresultat grafiskt i en 3D-miljö. Det fysiska objektet är inskannat med en 3D-skanner på förhand för att visas tillsammans med mätdata från sensorer för att enklare kunna se resultatet från mätdatan. Programvaran skall också kommunicera med en robotarm som utför dessa tester. Syftet med denna programvara är att göra det enkelt att se hur en produkt beter sig i verkligheten. Ett exempel på test är att mäta magnetfältet runt ett kretskort.

Detta examensarbete handlar om konstruktionen av kärnan till programvaran. Vi har haft nära samarbete med två andra projektgrupper som har gjort andra delprojekt av ett större projekt. Dessa grupper är: gruppen som utvecklat användargränssnittet till programvaran och gruppen som har jobbat med en robotarm som skall utföra tester.[2][3] Den färdiga lösningen har den funktionalitet som krävs för att rita upp mätdata. Programvaran kan rita upp ett inskannat objekt och sedan kan användaren klicka i programmet för att begära en sensormätning på den koordinaten. För att robotarmen ska gå till rätt koordinat måste programvarans koordinatsystem synkroniseras med robotarmens koordinatsystem. Detta beskrivs i kapitel 4.2. På grund av tidsbrist har inte programmet testats med riktig hårdvara. Vi har istället testat genom att emulera datatrafik för att se att programmet fungerar korrekt.

6.2. Kritisk diskussion

Ett mål vi och vår handledare hade med den färdiga produkten var att programvaran skulle vara så modulär det bara går. Den färdiga programvaran har bra modularitet i vissa aspekter. Vår programvara tar ingen hänsyn till vad det är för typ av robot som den kommunicerar med. Därför är det enkelt att byta ut en robot mot en annan robot. Det enda som behöver göras är att implementera vårt kommunikationsprotokoll i roboten. Det finns för närvarande inget enkelt sätt att lägga till nya typer av sensorer. Stöd för detta måste läggas till om man vill att programmet ska bli enklare att använda för användaren.

Samarbete mellan grupperna har överlag fungerat bra. Vi har haft utmärkt samarbete med

gruppen som utvecklat det grafiska gränssnittet. Vi har ofta programmerat tillsammans i samma rum vilket gjorde det mycket enkelt att diskutera hur vi skulle ta oss an problem. Hemsidan GitHub.com har varit ett bra verktyg för oss när vi jobbade från hemmet. På GitHub ligger all källkod till programmet vilket gör det enkelt för oss att se vilka ändringar andra personer har gjort samt ge kritik på kod. På GitHub kan vi också organisera alla buggar vi känner till. Vi kan lätt organisera vem som skall jobba på en bugg och samtidigt diskutera hur buggen skall fixas. Dessa hjälpmedel har hjälpt samarbetet de dagar vi jobbade från hemmet.

Samarbetet mellan de båda mjukvarugrupperna och gruppen som jobbar med robotarmen har varit blandat. I andra halvan av projekttiden har samarbetet varit bra. Vi har diskuterat över mail, telefon, och i möten. I början av projektet kunde kommunikationen varit bättre. En anledning till att det blev så var att vi tänkte att vi inte behövde fundera på hårdvaran innan slutet på projektet. Om vi hade gjort det här projektet med de erfarenheter vi har fått nu skulle vi gjort detta annorlunda. Vi hade haft möten i uppstartsfasen av projektet för att se till att vi alla hade samma vision om den färdiga produkten. Detta hade sparat oss mycket tid senare som vi kunde lagt på annat.

Som det står i kapitel 3.5 så hade vi tänkt använda JavaFX för att bygga vårt grafiska gränssnitt men på grund av dåligt stöd för Java 3D så gick inte detta. På grund av bra efterforskning i uppstartfasen så hittade vi detta problem ganska tidigt. Om vi inte hade hittat detta problem så tidigt kunde vi ha förlorat enorma mängder tid i början av projektet. Det är lockande att direkt börja skriva på programmet när man startar ett nytt projekt. Vi motstod den frestelsen och tog istället en vecka i början av projektet bara för att göra små enskilda projekt i Java 3D. Eftersom vi aldrig jobbat med Java 3D förut så lärde vi oss jättemycket användbart. Vi tror denna nya kunskap hjälpte oss när vi sedan skulle börja jobba på projektet. Vi lärde oss vad Java 3D hade för begränsningar och fördelar vilket hjälpte oss att designa vår programvara.

6.3. Generaliseringar

Det finns stor potential för denna typ av teknik. Att få möjligheten att kunna göra dessa exakta mätningar kan vara nyttigt t.ex vid produktutveckling där man gör en modell över hur det kan se ut för att sedan kunna använda denna data då man tillverkar den färdiga produkten.

Ett annat exempel kan vara att man skall mäta magnetfältet på en elektrisk komponent för att ta reda på störningar och liknande så att man kan förbättra sin produkt.

6.4. Fortsatt forskning

Då vi inte hann med att göra en helautomatiserad läsning av ett antal punkter i en serie så är

detta en bra utgångspunkt för fortsatt arbete med projektet. Denna typ av helautomatisering krävs att robotarmen måste röra sig på ett visst sätt för att nå en position i det tredimensionella planet. Det krävs alltså kunskaper och beräkningar av inverterad kinematik för att få till detta på ett bra sätt. En annan fråga är vilka punkter som man skall använda sig av, då ett 3D-objekt som vi läser in i programmet kan ha hundratusentals punkter så vill vi ju inte mäta på alla dessa. Det behövs alltså ett sätt att ta fram en viss mängd punkter för att sidan skicka dessa till robotarmen som besöker dessa och skickar tillbaka mätvärdena i en följd.

6.5. Beslutstagande och strukturering

I projektet har vi gjort en del bedömningar ang. vilka grässnitt och tekniker vi skall använda. För att göra dessa bedömningar behövde vi antingen utifrån egen erfarenhet eller så behövde vi prova om det fungerade som vi ville. Vi behövde ta en del beslut för att komma vidare och vi tyckte det var viktigt att vi var eniga om beslutet så därför undersökte vi ett antal alternativ varje gång ett större beslut skulle tas. Det skall även understrykas att alla får komma till tals och att om det är några konstigheter behövs dessa tas upp på en gång. Ett exempel ett beslut vi gjorde var när vi skulle välja vilket grafiskt Java-bibliotek vi skulle ha till vårt program. Först så kollade vi vilka olika alternativ som fanns att välja mellan och sen fick väga upp dem mot varandra. Frågor som vi fick ställa oss då var t.ex om hur väl använt biblioteket var, vilka fördelar och nackdelar det hade och om det användes fortfarande. Ibland är det bra att använda ett helt nytt bibliotek då det kan innehålla nya egenskaper som gör att programmet t.ex blir smidigare och bättre. Men i vårt i fall så valde vi det lite äldre Java3D-biblioteket efter att vi undersökt att det fanns mycket mer information om detta biblioteket och att det var väl beprövat i jämförelse mot JavaFX som var en av de nyare alternativen men som inte var helt färdigutvecklat.

Att skriva små program brukar oftast inte vara några problem och blir sällan svåra att få översikt över. Men när man skall skriva större program som innehåller ett större antal kodrader så behövs det strukturering. Som nämnt tidigare i kap 5.6 så valde vi att strukturera vårt program med hjälp av MVC-modellen där man delar in programmet i olika paket. Detta är ett måste om programmet skall kunna få en bra översikt eller någon annan vill ändra eller utöka programmet.

Dessutom behövs det även radkommentarer samt klass-dokumentation för att beskriva hur metoder och kod fungerar. Om inte denna struktur finns, så kan det bli enormt svårt för även en erfaren programmerare att göra ändringar i ett program.

Referenser

1) http://git-scm.com/about

2) C. Fransson, S. Ivarsson, 2012. 3D Sensor View. Under publicering

3) H. Schörling, B. Johansson, 2012. Multirörelseplattform. Under publicering

4) J. Onsjö, A. Hansen, N. Johansson, 2011. Tredimensionell Sensor scanner. http://publications.lib.chalmers.se/records/fulltext/155208.pdf

5) http://en.wikipedia.org/wiki/Java_3D

6) http://en.wikipedia.org/wiki/STL_(file_format)

7) http://en.wikipedia.org/wiki/Serial_communication

8) http://com0com.sourceforge.net/

9) http://rxtx.qbang.org/wiki/index.php/FAQ

10) http://arduino.cc/en/Main/FAQ

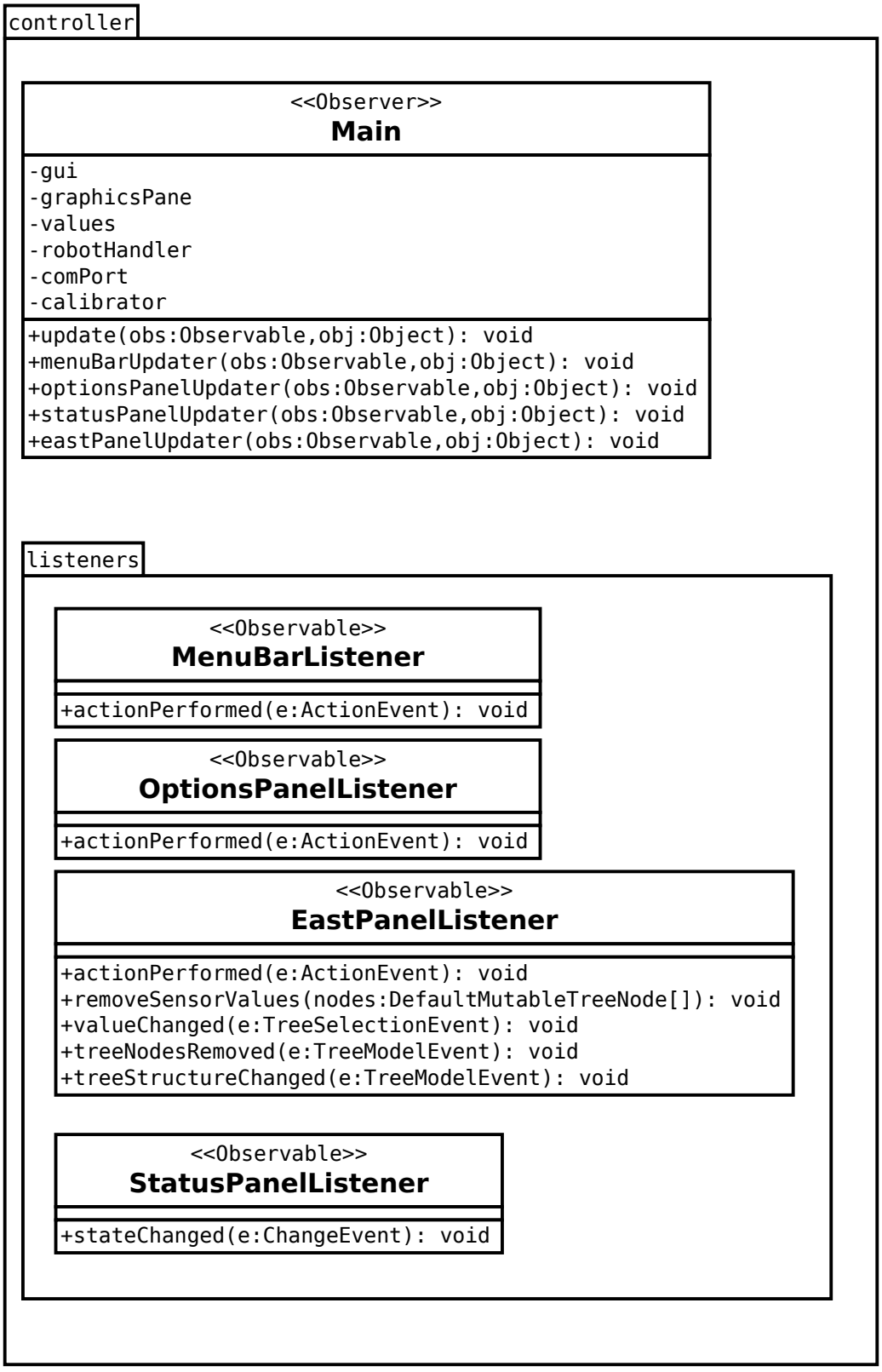
11) http://en.wikipedia.org/wiki/JavaFX

12) http://www.java.net/node/674071

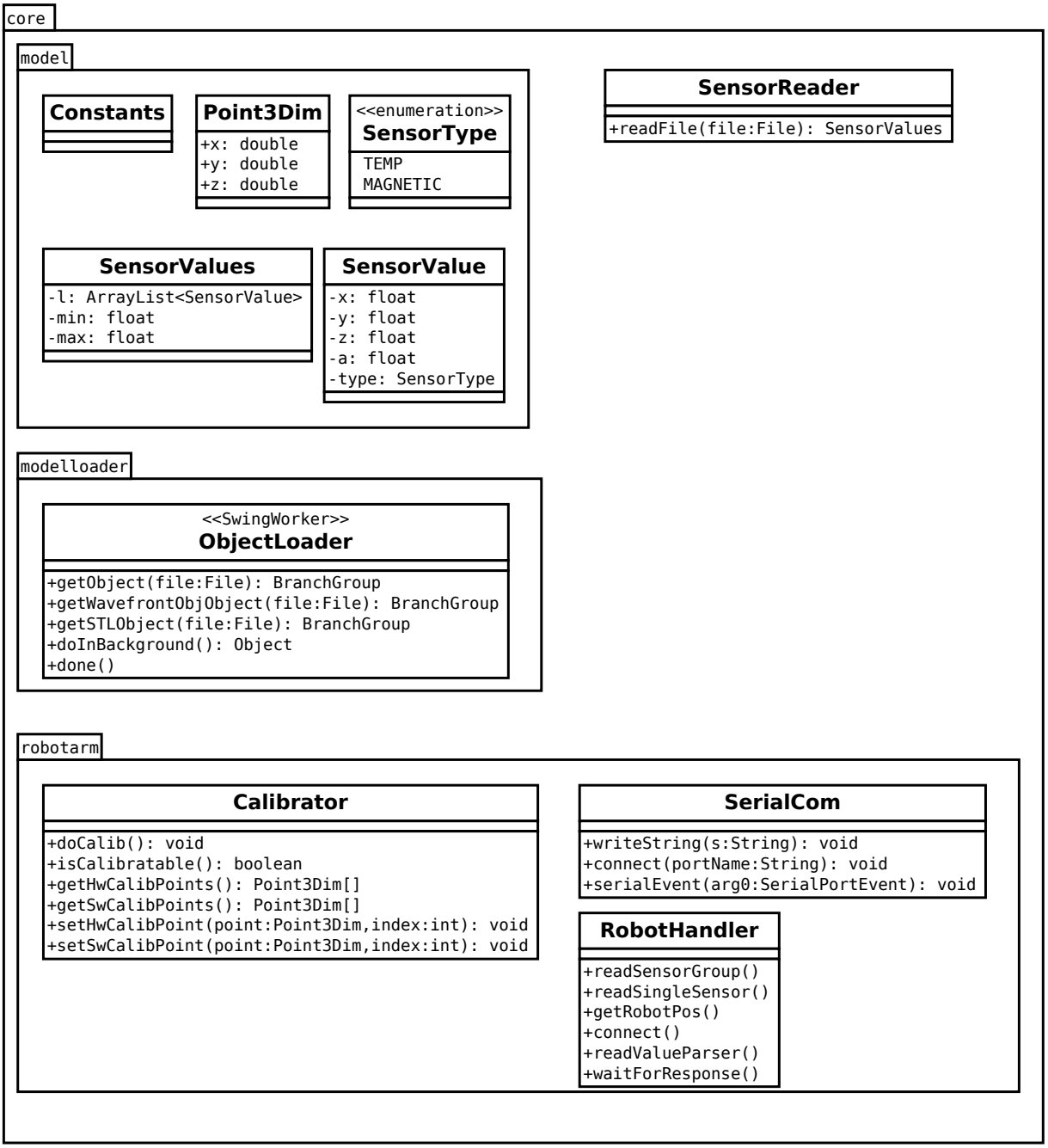
13) http://en.wikipedia.org/wiki/MeshLab

Bilagor

Bilaga 1: Controller-paketet



Bilaga 2: Core-paketet



Bilaga 3: View-paketet