CHALMERS

Sensorviewer 3D View

Visualisering av mätdata kring ett tredimensionellt objekt

Examensarbete inom högskoleingenjörsprogrammet Dataingenjör

CHRISTIAN FRANSSON

SIMON IVARSSON

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

Abstract

This thesis covers the development of a computer program that aims to visualize a scanned 3d object together with measurement data collected from around the object. The thesis covers only the visual aspects of the program. The problem of visualizing the object and measurement data in the same view is solved by importing and adding the object to a Java 3d transformgroup which is manipulated to align the object to the measurement data. This is done by using three points of reference picked out on the virtual object and the same three points identified on the physical object. After calibration, collected measurement data is displayed as spheres in varying color tone depending on the measured value, and with correct rotation and scale around the object. The development process resulted in a program capable of importing, calibrating and visualizing an object together with measurement data in the same window, with a graphical menu structure for user input and navigation.

Keywords: Visualization, measurement data, object, java 3d, calibration

Sammanfattning

Denna rapport och examensarbete beskriver datorprogrammet som utvecklats för att visualisera ett inscannat tredimesionellt objekt tillsammans med mätdata insamlad kring objektet. Rapporten är begränsad till de visuella aspekterna av programmet. Visualiseringen av objektet och mätdatan tillsammans löses genom att importera och lägga till objektet i en Java 3d transformgrupp. Denna manipuleras sedan för att anpassa sig till mätdatan med hjälp av tre referenspunkter som anges på det virtuella objektet och samma antal punkter som anges på motsvarande plats på det fysiska objektet. Mätdata kan sedan läsas in och visualiseras som tredimensionella sfärer kring objektet med varierande färgton beroende på det uppmätta värdet och visas med rätt rotation och storlek runt objektet. Arbetet resulterade i ett program som klarar av att läsa in, kalibrera och visualisera ett objekt tillsammans med mätdata i samma fönster, med ett grafiskt gränssnitt för användarinput och navigering.

Nyckelord: Visualisering, mätdata, objekt, java 3d, kalibrering

Innehållsförteckning

1. Inledning

Detta kapitel ger en översikt över problemet som behandlas i denna rapport och dess bakgrund. Vidare beskrivs de ämnen som denna rapport inte kommer att behandla samt en ordlista med ord och förkortningar som används i rapporten.

1.1 Bakgrund

Dagens teknik erbjuder många möjligheter för att visualisera data. En 3D-kamera har möjlighet att scanna in en tredimensionell bild av ett objekt som sedan kan hanteras i datorn. Denna teknologi används ofta av konstnärer och 3D-designers för att t.ex. infoga en figur till ett datorspel, eller för att kunna användas av en 3D-printer för att kunna skriva ut kopior av ett objekt, men hålla originalet intakt. En annan, mycket äldre, teknologi är att samla in data från olika sensorer som sedan sammanställs i till exempel tabellform eller används direkt för olika beräkningar.

1.2 Problem

I dagsläget finns ingen metod för att sammanföra sensordata med en inscannad tredimensionell representation av objektet man söker information om. Vårat program låter användaren göra just det.

1.3 Syfte

Syftet är att bestämma krav och utveckla programvara för hanteringen av sensordata kring ett inscannat tredimensionellt objekt. Detta inkluderar hur objektet ritas upp och hur sensordata representeras grafiskt på skärmen samt att detta görs på ett sådant sätt att det överensstämmer med verkligheten

1.4 Avgränsning

Den här rapporten beskriver endast den grafiska delen av programmet, dvs. visualiseringen av mätdata och objekt samt menyer. Programmet innehåller även en komponent för övrig logik, vilket inte kommer att presenteras i den här rapporten.[1] Inhämtningen av sensordata sker med hjälp av en robotarm med en sensor samt dess tillhörande styrkrets. Utvecklingen av robotarmen beskrivs även den i en separat rapport.[2]

1.5 Definitioner

API
: Application Programming Interface, innehåller regler för hur man använder de metoder som går att kalla på.

Bibliotek
: Innehåller metoder som kan anropas enligt ett API för att utföra operationer.

COM-port
: En kanal vilken en dator kan använda för att kommunicera seriellt med annan hårdvara.

Datastruktur
: En enhet som används för att spara information inom programmering.

Java
: Ett programmeringsspråk som utvecklas av Oracle.

Java 3d
: Ett bibliotek med metoder för att rendera 3d grafik.

Java Swing
: Ett bibliotek med metoder för att skapa menyer.

.obj
: Ett filformat för att spara 3d objekt.

.stl
: Ett filformat för att spara 3d objekt.

Versionshanteringssystem
: Låter användaren spara projekt på en server som även sparar gamla versioner av projektet.

2. Metod

Detta kapitel beskriver de tillvägagångssätt och arbetsmetoder vi använt i projektet. Det beskriver både generella metoder, som organisation, samt även specifika saker som vilken utvecklingsmiljö som använts.

2.1 Källor

Vår huvudsakliga källa till information om programmering har givetvis varit javas egna API'er på Internet, samt de för Java Swing[4] och Java 3d[13].
Utöver detta har även flera olika dokumentationer för de olika bibliotek som använts studerats, samt handledningsdokument rörande dem[5,6]

2.2 Verktyg

All programmering i projektet har utförts med programmeringsverktyget Eclipse[14] och vi har använt ett Git-repository[15] som versionshanteringssystem. Dessa val var naturliga, då samtliga medlemmar i projektet hade gedigen erfarenhet av båda dessa system.

2.3 Rapportering

Rapporteringen till handledaren för projektet skedde genom att dagligen skriva ner vad som har gjorts under dagen, att en gång per vecka skicka in en veckorapport till handledaren, samt att ha minst ett möte med handledaren per vecka.

2.4 Samarbeten

Grupp core

Vi arbetade mycket nära utvecklarna av core[1], dvs. logik-delen av programmet under hela utvecklingsarbetet. Vi arbetade oftast i samma sal, utbytte många idéer och hade långa diskussioner om strukturen av kod och lösningar, såsom hur core och view skall kommunicera med varandra.

Grupp hårdvara

Vi arbetade även mycket med hårdvarugruppen[2] för att utarbeta hur robotarmen skulle styras från vårat program och hur data skulle skickas mellan grupperna, vilket resulterade i ett API som båda grupperna använder vid all kommunikation.[16]

Grupp 3d-Scanner

I projektets början hade vi även ett möte med en grupp som hade arbetat med 3d-scannern tidigare. Under detta möte lärde vi oss hur vi använder 3d scannern praktiskt, varvid vi scannade in ett objekt för att använda i vårat program.

2.5 Förkunskapskrav

För att genomföra projektet krävs gedigen kunskap om objektorienterad programmering i språket Java. Kunskaper om biblioteket Java 3d krävs för all rendering i 3d-grafikfönstret och kunskap om biblioteket Java Swing krävs för att utveckla alla menyer.

2.6 Versionshanteringssystemet Git, Github

Git verisionhanteringssystem[18]

Versionshanteringssystemet git är användbart när flera personer arbetar på samma projekt, då det tillåter flera personer att göra ändringar i samma filer. Sedan kan användarna välja ut de bästa delarna av varandras arbete för att användas i slutversionen av filerna. Slutversionen sparas på en server som alla kan komma åt och därmed bygga vidare på för att sedan göra om processen med att sätta samman olika personers arbete till en ny slutfil.

Grenar, "Branches" [17]

Ett användbart verktyg som erbjuds av git är möjligheten att skapa grenar, eller "branches". Detta innebär att man först kopierar den senaste versionen av projektet till ett nytt sidoprojekt, en "branch". Ändringar i sidoprojektet påverkar inte huvudprojektet, och när man är nöjd med funktionaliteten i sidoprojektet kan man sätta ihop det med huvudprojektet igen. Detta är användbart när en del av gruppen som arbetar på projektet vill implementera ny funktionalitet som kan försämra eller ändra funktionaliteten av programmet. Då kan man skapa en ny gren, och resten av gruppmedlemmarna kan fortsätta arbetet som vanligt utan störningar från de nya funktionerna. Den här funktionaliteten använde vi oss av vid utvecklingen av funktionen för kalibrering, se kap. 4.5. Metoden som utvecklades för kalibrering skapade problem för resten av programmet medan den utvecklades, men behövdes ändå sparas på servern för att flera personer skulle kunna komma åt den under och mellan de tillfällen vi utvecklade kalibrerings-funktionen.

Github

Under arbetets gång så har Githubs funktionalitet för hantering av problem och milstenar använts för att hålla reda på problem i programmet. Github låter användaren hålla reda på problem som dyker upp i programmet för att senare kunna åtgärda dom. Samma funktionalitet användes för hanteringen av nya funktioner som skall implementeras i det färdiga programmet, vilka kallas milstenar. Verktyget gav oss en bra överblick över arbetet, både över vad som hade gjorts och vad som fanns kvar att göra

3. Teknisk Bakgrund

I detta kapitel beskrivs de tekniker, bibliotek och efterforskningar vi gjorde i början av projektet. Det beskriver varför vi valde vissa tekniker framför andra samt ger en överblick av de funktioner som dessa tekniker har att erhålla.

3.1 Java-bibliotek för 3D grafik

Då projektet tog sin början behövdes ett beslut fattas om vilket 3d-bibliotek som skulle användas. Det fanns ett antal nya bibliotek, däribland JMonkeyEngine[20], samt även de något äldre biblioteken JOGL[19] och Java3D. Det beslutades att något av de äldre biblioteken skulle användas, då dessa var mer väldokumenterade. Just JMonkeyEngine var dessutom primärt ämnad för spelprogrammering, och därför troligtvis inte lämpad för detta projekt. Java3D valdes till slut, dels på grund av ett råd vi fick från en av våra handledare[3], och dels på grund av att några projektmedlemmar hade tidigare erfarenhet av JOGL och tyckte att detta bibliotek var onödigt krångligt, och förhoppningen var att Java3D skulle vara enklare.

3.2 Java-bibliotek för grafiskt gränssnitt

Valet av uppritningen av det grafiska gränssnittet stod framför allt mellan Java Swing och JavaFX[12]. JavaFX verkade till en början vara väldigt bra, samtliga gruppmedlemmar testade att programmera små applikationer med det under en helg. Då framkom det att det var mycket enklare än Swing, som gruppen hade lite erfarenhet av sedan tidigare kurser. Dessvärre hade version 2.0, som släppts några månader innan projektets start, tagit bort sitt stöd för att infoga Java Swing fönster i JavaFX. Detta gjorde att JavaFX blev olämplig för vårat ändamål, då Java3D använder sig av ett Swing-fönster. Det diskuterades dock om vi skulle använda JavaFX metoder för att infoga sig självt i ett Swing fönster, men detta upplevdes som onödigt komplext. Så till slut valdes Java Swing som det bibliotek som skulle användas för det grafiska gränssnittet.

3.3 Java3D

Java3D är ett så kallat Scene Graph baserat bibliotek.[6] Detta betyder i korthet att varje objekt som ritas ut ligger i en logisk trädstruktur. Trädets rot är själva universumet. Universumet har en eller flera Locales, vilket är högupplösta koordinater varifrån alla andra noder har sin bas. En nod kan vara vad som helst som ingår i universumet, exempelvis en sfär eller en branchgroup. I sitt grundutförande finns det bara en Locale, som ligger i universumets origo. Nästa steg i hierarkin är BranchGroups. BranchGroups kan innehålla både andra BranchGroups och andra noder. Om en BranchGroup läggs till en Locale ritas alla dess noder ut på skärmen. Om BranchGroupen har andra BranchGroups som barn ritas även dessa och deras barn ut. Då en BranchGroup har lagts till en Locale betraktas den som “live”. Detta betyder bland annat att man nu inte kan ändra i den utan att först ta bort den ur sin Locale, eller explicit ställt in den för att kunna hantera dessa ändringar live.

TransformGroups är en nod som kan användas för att enkelt gruppera andra noder och flytta dem i ett stycke. En TransformGroup kan innehålla BranchGroups men måste också ha en BranchGroup högre upp i hierarkin för att kunna ritas upp. En TransformGroup har ett

Transform3D objekt som definierar vart i universumet TransformGroupen skall befinna sig, vilken rotation TransformGroupen har, samt vilken skala den har. All denna information beskrivs i en matris, och genom att göra olika matrisberäkningar kan man flytta och ändra på TransformGroupen. Java3D erhåller givetvis flertalet funktioner vilket gör att väldigt få beräkningar behövs göras för hand.

Java3D har en klass vid namn SimpleUniverse som förenklar skapandet av universumet något genom att automatiskt tillverka en Locale, samt även skapa objekten som hanterar “ögat” i universumet, dvs. kameran som genererar perspektivet som användaren ser universumet genom. Kameran kan man sedan enkelt få ut och manipulera för att exempelvis flytta runt den i universumet.

När ett objekt ligger i en BranchGroup eller i en TransformGroup är dess koordinater definierade från sin förälders origo och inte universums origo. Detta betyder att om ett objekt t.ex. ligger på koordinat (0,5,0) i en TransformGroup som ligger i universums origo, men flyttas till en annan TransformGroup som ligger i (2,3,0) så kommer objektet att ritas upp på position (2,8,0) i relation till universumets origo. Detta betyder också att om du har ett objekt som inte ligger i origo (som ofta är fallet med objekt inskannade med en 3D Scanner) och försöker rotera detta, så kommer objektet att rotera i omloppsbana kring TransformGroupens origo, och inte kring sig självt, som man annars intuitivt skulle gissa. Ett sätt att lösa detta på är att nästla TransformGroups. Om man vill att ett objekt skall rotera kring en specifik punkt kan man först lägga objektet i en TransformGroup som man sedan flyttar på ett sådant sätt att punkten som önskas hamnar i origo för dess föräldra-TransformGroup. Efter detta roterar man föräldern istället och på så sätt får man god kontroll över objektets position.

3.4 3D Scanner

3D-scannern, "HDI 3d Scanner" från 3D3 solutions[21], som vi använde, använder sig av synligt ljus för att ta bilder av objektet. Scannern har två kameror samt en projektor som belyser objektet, och scannerns programvara jämför sedan bilden från de två kamerorna för att sammansätta en tredimensionell bild. Innan den kan göra detta måste dock scannern kalibreras. Detta görs genom att ta flera bilder med scannern av en rutmönstrad platta. Scannern känner till storleken på rutorna och kan därför bestämma avståndet och skalan till plattan. För att göra kalibreringen nogrann krävs dock att ett flertal bilder tas, och att plattan flyttas något mellan varje bild. Programvaran visar i vilket område av bilden den är kalibrerad för och när man är nöjd kan man börja scanna in det faktiska objektet. Även denna gång krävs det att man tar flera bilder av objektet innan man får ett helt objekt. Varje bild ger en tredimensionell del av objektet, som man sedan får sätta ihop med de andra bilderna för att slutligen få objektet i sin helhet. Programvaran ger viss assistans vid ihopsättandet av bilden, men man måste ofta manuellt göra små justeringar för att det skall stämma exakt. Processen kan liknas vid att skala en apelsin och sedan sätta ihop skalet till en sfär igen.

När man är nöjd kan man exportera sitt objekt i valfritt format. Filerna blir dock väldigt stora och detaljerade, vilket kan vara olämpligt vid filöverföring och inladdning av objektet. Man kan få ner storleken genom att använda programvara som MeshLab, som kan minska storleken på filen genom att minska antalet polygoner samt ta bort onödig information i filen som blivit över efter 3D-scanningen.

3.5 Tidigare examensarbeten

Ett år innan detta examensarbetet sattes i gång hade en annan examensgrupp försökt göra ett liknande projekt. Dock visade det sig att det var lite för stort relativt den arbetsstyrka som sattes på projektet, vilket också var grunden till att fler grupper blev involverade denna gång. Dessutom försökte de att hantera både inscanningen och mätningen av objektet på en och samma gång. Detta visade sig vara svårt att göra, vilket är varför Chalmers denna gång införskaffade en 3D-scanner för ändamålet.

3.6 MVC design

Model-View-Controller design (MVC) användes för att strukturera koden, vilket bestämdes från dag ett i projektet. MVC design innebär att man logiskt separerar all kod som hanterar modell, grafiskt gränssnitt och logik från varandra. Designen valdes eftersom den uppdelning mellan de visuella delarna och de logiska delarna som designen erbjuder stämde väl överens med den uppdelning vi hade i projektet.

“Model” innehåller datastrukturer, såsom SensorValue.java, som instansieras en gång per inläst sensorvärde för att representera det. Model behandlas i separat rappport[1].

“View” innehåller alla grafiska komponenter I programmet, vilket inkluderar 3d renderingen och alla menyer. View behandlas i denna rapport.

”Controller” innehåller de logiska komponenterna i programmet. Controller initierar programmet, vilket inkluderar skapandet av grafikfönstret och menyerna, och alla lyssnare som tar hand om knapptryckningar i programmet. Controller behandlas i separat rappport[1].

Se bilagor för översikt över designen.

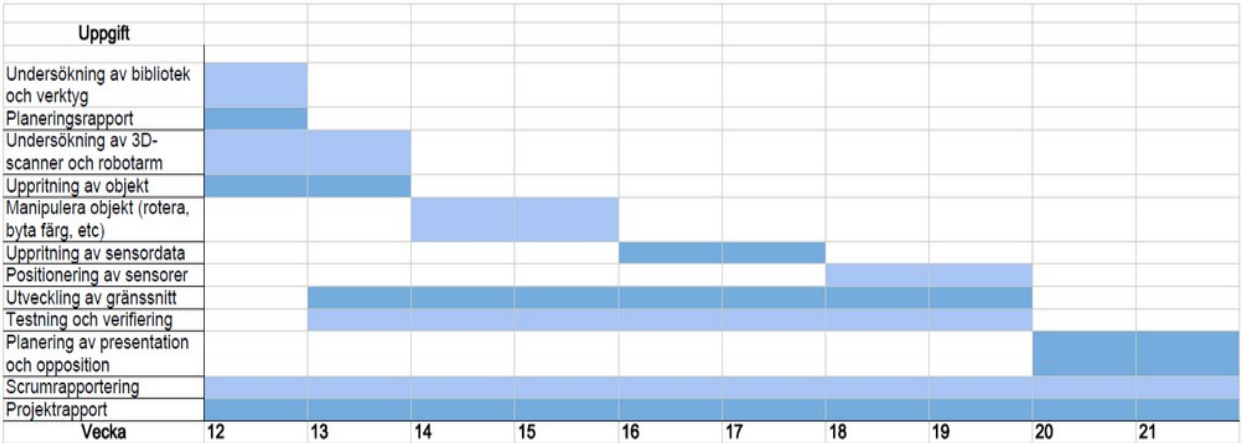
4. Genomförande

Detta kapitel är en genomgång av utvecklingen av programvaran samt dess funktionalitet. Programmets grafiska gränssnitt beskrivs i detalj. Menyers uppbyggnad och funktionalitet samt det tredimensionella fönstrets funktionalitet, kontroller och rendrering är alla innefattade i detta kapitel. En del i kapitlet har även tilldelats för att grafiskt beskriva hur kalibreringsprocessen går till.

4.1 Planering

I början av projektet fastställde vi en planering av vad vi förväntade oss hinna med varje given vecka. De moment som arbetas på varje vecka är markerade i gantt schemat nedan. Vi lyckades alltid ligga i fas eller före planeringen, vilket gav oss gott om spelrum i slutet av examensarbetet att skriva klart rapporten.

Då alla moment tidigt i projektet avklarades före eller på slutdatumet i planeringen så låg vi före planeringen vid slutet av vecka 15, varvid vi bestämde oss att lägga mer tid på att skriva rapporten de nästkommande två veckorna. Det resulterade i att vi vid början av vecka 18 låg i fas med planeringen igen, och hade skrivit mycket på rapporten.



figur 1.1. Planeringen av projektet veckovis. Aktuella moment är markerade för respektive vecka.

Vi skrev kontinuerligt på rapporten genom hela arbetet för att inte riskera att ha väldigt mycket arbete framför oss de sista veckorna samt för att säkerställa att vi inte glömde bort relevanta detaljer fram emot slutet.

Det beslutades att vi skulle initialt fokusera på den tredimensionella delen av programmet då ingen av gruppens medlemmar hade erfarenhet av Java 3D sedan tidigare. Om det i detta skede hade upptäckts att Java 3D varit olämpligt för vårt arbete hade vi på detta sätt säkerställt att vi fortfarande hade tid att byta till ett annat 3D-bibliotek.

Utvecklingen av gränssnittet lades över en lång tidsperiod då vi från början antog att gränssnittet gradvis skulle utvecklas allt eftersom vi lade till fler funktioner under projektets gång.

Ett alternativ till vår planering kunde ha varit att från början slå fast exakt vilken funktionalitet vi ville ha i programmet och därefter gjort gränssnittet baserat på detta och först efter detta gjort det tredimensionella fönstret. Denna planering hade passat oss dåligt, eftersom vi i planeringsstadiet inte ville låsa oss vid en lösning som kanske inte visade sig fungera på ett bra sätt med hårdvaran. Denna typ av planering hade dock med stor sannolikhet passat mycket bra i ett projekt där deltagarna hade erfarenhet av Java 3D och den robotarm som användes eller i ett projekt av mindre experimentell natur.

4.2 Krav

- Programmet skall kunna importera och visualisera ett inscannat tredimensionellt objekt.
- Programmet skall vara kapabelt till att importera och visa sensordata på ett begripligt sätt, både grafiskt och genom en tabellstruktur.
- Användaren skall kunna manipulera den tredimensionella bilden så att data som är relevant för användaren klargörs på ett enkelt sätt.
- Användaren skall kunna låsa kameran på en valfri axel runt objektet för att lättare kunna se mätvärdena från varje enskild sida av objektet.
- Användaren skall kunna rotera kameran runt objektet och kunna zooma in och ut relativt objektet.
- Användaren skall kunna justera zoom och ljusstyrka genom reglage.
- Användaren skall genom en menystruktur kunna:
  - Importera ett 3d objekt.
  - Avsluta programmet.
  - Synliggöra och dölja rutnätet och axlarna.
  - Välja com-port.
  - Utföra kalibrering.
  - Visa information om programmet och dess utvecklare.

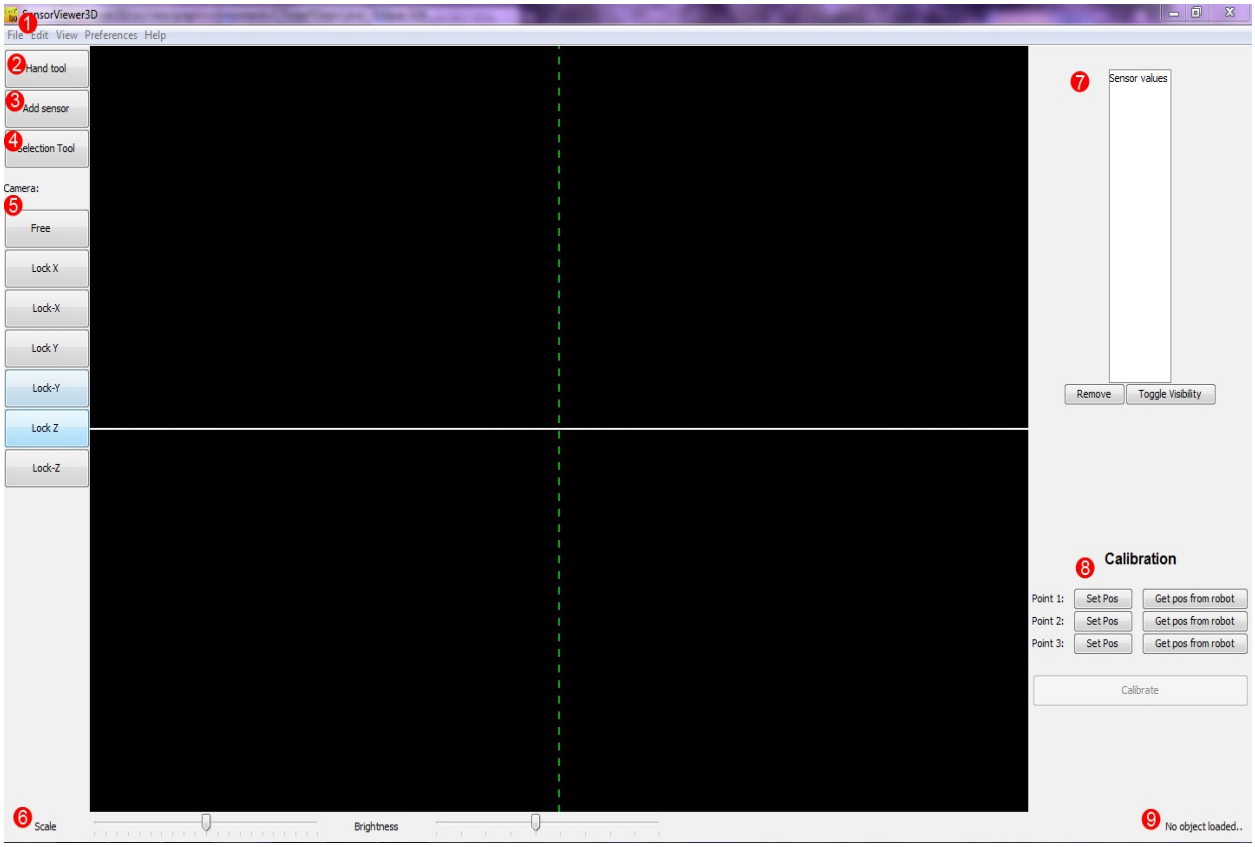
4.3 Java 3D, översikt

Rendering av objekt och mätdata sker med hjälp av biblioteket Java3d. Objektet ljussätts för att synas tydligt för användaren som även kan ställa in ljusstyrkan på objektet.

Det är även möjligt att visa tre linjer som representerar x, y och z-axlarna i universumet, samt att visa ett rutmönster som gör det enklare att se vilket avstånd olika objekt har från varandra.

Ett "OrbitBehavior" objekt tillhörande Java 3d biblioteket används för att låta användaren rotera sitt öga 360 grader runt objektet och tillhörande mätdata, och samtidig kunna zooma in och ut. Egna metoder har skrivits för att låta användaren låsa kameran vid valfri sida av varje axel kring objektet (x, y eller z) för att få en bättre vy över hur mätdatan är distribuerad i rummet. När användaren låser kameran på en axel så stängs även perspektivseendet av, med andra ord så renderas alla objekt av samma storlek som lika stora oavsett avstånd från kameran. Exempelvis ett objekt långt ifrån kameran som normalt hade renderats som relativt litet visas nu lika stort på skärmen som ett objekt av samma storlek som befinner sig närmare kameran.

4.4 Menyer och deras komponenter



Figur 1.2 illustrerar komponenters och menyers position i fönstret.

Övermeny

Övermenyn (1 i fig 1.2) innehåller en menystruktur för att utföra flera operationer i programmet. Undermenyerna detaljeras nedan:

File: Ger användaren två alternativ, att ladda in ett 3d objekt eller att avsluta programmet.

View: Låter användaren visa eller dölja rutnätet samt x, y och z -axlarna.

Preferences: Låter användaren ange robotarmens com-port och utföra kalibreringen.

Help: Visar information om programmet och dess utvecklare

Vänstermeny

Vänstermenyn innehåller knappar för att kontrollera kameran, samt knappar för att möjliggöra för användaren att välja mellan att låta musrörelser och musklick rotera kameran eller användas för att välja sensorvärden för att visa detaljerad information om varje sensorvärde.

Den översta knappen, "handtool", (2 i fig 1.2) låter användaren använda musen för att rotera sin kamera runt objektet och sätta ut referenspunkter på objektet.

Nästa knapp, "Add Sensor" (3 i fig 1.2) flyttar robotarmen till positionen på objektet där muspekaren senast trycktes ner, vilket markeras av den röda bollen, och läser in samt renderar det inlästa mätvärdet.

Tredje knappen "Selection tool" (4 i fig1.2), låter användaren välja en grupp sensorer för vilka gemensam data visas. Detta blev dock inte implementerat, och i dagsläget utfyller denna knapp ingen funktion. Denna funktionalitet kan istället nås från högermenyn, se nedan.

Under titeln "Camera" (5 i fig 1.2) finns knappar för att låsa och låsa upp kameran från en axel, se 4.1.

Undermeny

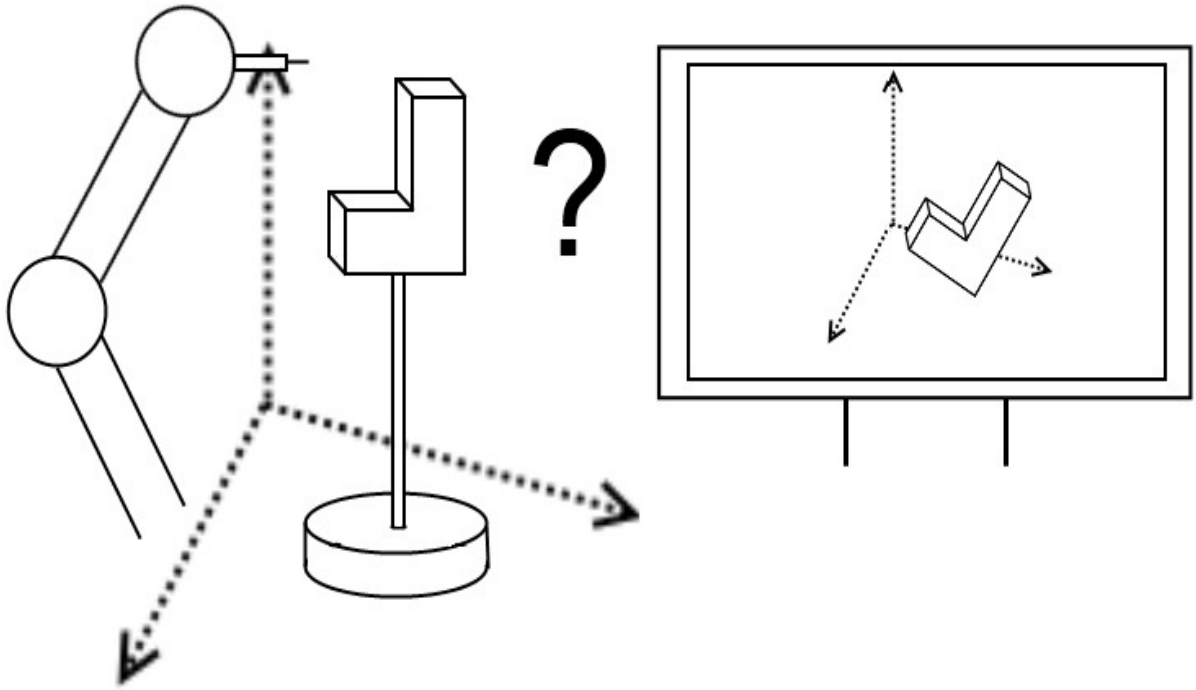
Reglaget med funktionalitet för att zooma in och ut, zoom-slider, samt reglaget för att justera ljusstyrkan (6 i fig 1.2) implementerades med swing-komponenten JSlider. Dessa reglage utgörs av så kallade "sliders" som dras till vänster och höger för att proportionerligt justera zoom respektive ljusstyrka i scenen. Undermenyns högra del visar information om det inlästa objektets namn, samt information om olika händelser i programmet, såsom på vilken punkt i scenen man senast klickade med musen. (9 i fig 1.2)

Högermeny

Högermenyn (7 i fig 1.2) innehåller ett träd med information om samtliga sensorpunkter. I detta träd är det möjligt att få en översikt över punkterna och deras koordinater och tillhörande värden. Det är dessutom möjligt att gruppera sensorer i olika grupper, att visa, dölja eller ta bort valfria mätvärden som visas i det grafiska fönstret.

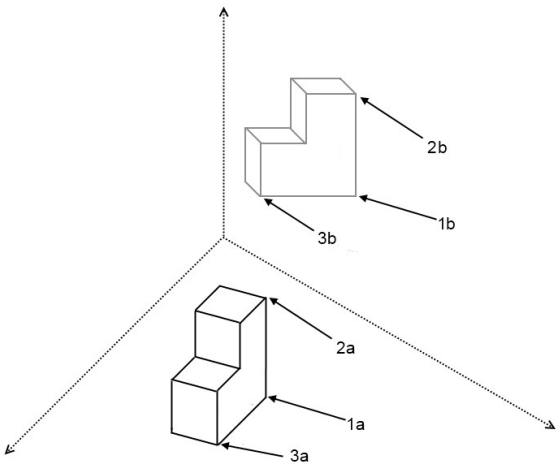
Under titeln "Calibration" (8 i fig 1.2) finns knapparna som används vid kalibrering mot robotarm. För varje av de tre referenspunkterna finns knappar för att spara referenspunkten som man har pekat ut med muspekaren på objektet, samt en knapp för att läsa in motsvarande referenspunkt från robotarmen. När alla tre referenspunkterna har placerats ut och lästs in från roborarmen ges möjligheten att trycka på knappen "Calibrate", vilket utför kalibreringen så att all mätdata som läses in härefter hamnar på rätt plats relativt objektet. (Se kap. 4.5)

4.5 Kalibrering



Figur 1.3 illustrerar robotarmen och objektet runt vilket data skall läsas in. (Figur av författarna)

För att kunna skicka och ta emot koordinater mellan robotarmen och programmet krävs det att programmet känner till vilken koordinat i programmet som motsvarar vilken i den rymd som robotarmen har att röra sig i. Därför behöver först robotarmen och programmet kalibreras gentemot varandra. Till att börja med bestäms tre punkter på objektet. Dessa punkter kan vara var som helst, men bör markeras på ett sådant sätt att de är både synliga på objektet i verkligheten och på den 3D modell som skannats in med 3D kameran. Efter att objektet har skannats in markeras de tre punkterna i programmet. (Se fig. 1.4) Därefter styrs robotarmen manuellt till de tre punkterna på det fysiska objektet och skickar dessa koordinater till programmet. Slutligen utför programmet kalibreringen enligt tillvägagångssättet nedan.



Figur 1.4 illustrerar de tre referenspunkterna på objektet. (Figur av författarna)

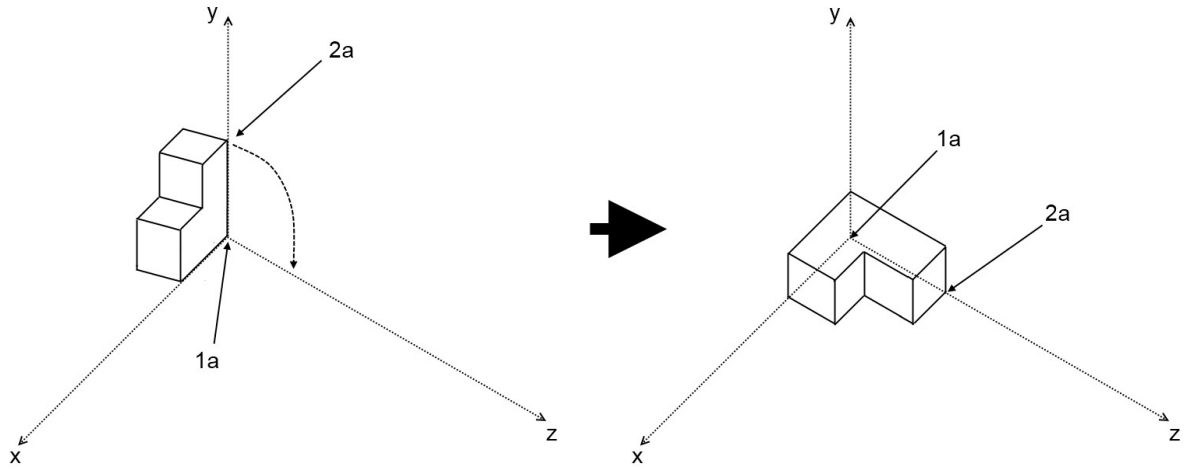
1. Grundposition

Innan någon kalibrering utförts ser situationen ut ungefär som i figur 1.4 i programmet. Det ljusare objektet representerar det objekt som robotarmen “ser”, det syns egentligen inte på skärmen, men visas här för extra tydlighet. Användaren har satt ut tre punkter (1a, 2a och 3a) och programmet har även fått tre punkter (1b, 2b och 3b) från robotarmen. Målet med kalibreringen är att få punkt 1a att överrensstämma med 1b, 2a med 2b samt 3a med 3b. Detta resulterar i att robotarmen och programmet efter kalibrering kommer att ha samma koordinatsystem. Då det är enklare att flytta det virtuella objektet än det fysiska är det således också det objekt som avses att flyttas så att det överrensstämmer med det fysiska objektet.

2. Omskalning

Då det fysiska och det virtuella objektet inte nödvändigtvis är lika stora krävs först att man förstorar eller förminskar det virtuella objektet för att de skall bli samma storlek. Detta utförs genom att jämföra avståndet mellan 1a och 2a med avståndet mellan 1b och 2b och därefter skala om det virtuella objektet med kvoten av dessa.

3. Centrering och rätning mot Z-axeln



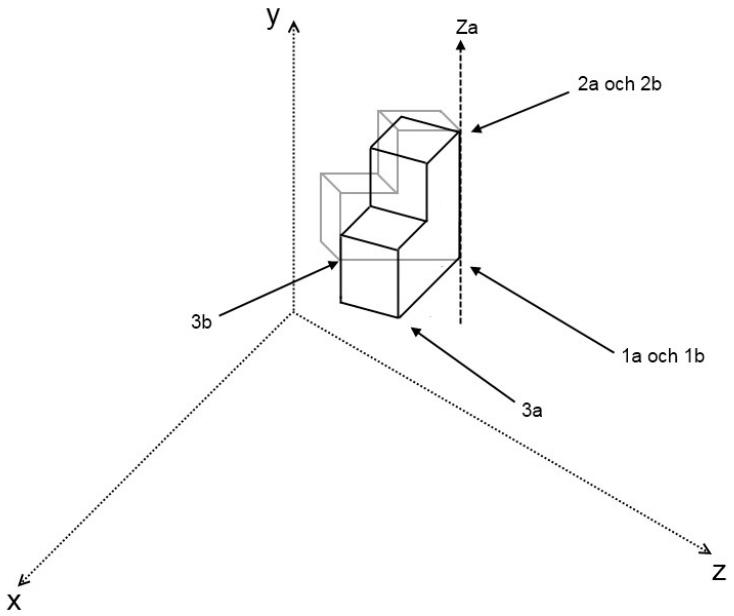
Figur 1.5 illustrerar rotationen av objektet. (Figur av författarna)

Efter omskalningen flyttas det virtuella objektet på ett sådant sätt att punkt 1a hamnar i origo. Därefter roteras objektet kring både X-axeln och Y-axeln på ett sådant sätt att punkt 2a hamnar på Z-axeln (Se fig 1.5).

4. Placering och rotation

I detta steg får vi nytta av de ompositioneringar som gjordes i steg 3. Vi har valt våra transformgrupper (se kap 3.3) på ett sådant sätt att vi nu har en transformgrupp innehållande det virtuella objektet som nu är rätat mot Z-axeln i transformgruppen (och för tillfället även universum). Vi kan nu använda en metod i Java3D som heter lookAt(). LookAt tar in tre stycken variabler; en position dit vi vill flytta objektet, en position som vi vill att objektet skall “titta på”, samt en vektor som beskriver vilket håll som är uppåt. När ett objekt “tittar” på en punkt avses egentligen att objektets Z-axel skär punkten som tittas på. Variabeln som beskriver var vi vill flytta objektet väljs till 1b. Detta betyder att 1a (som ligger i origo) hamnar på samma position som 1b. Variabeln som beskriver vart vi vill titta väljs till 2b. Då vi sett till att både det fysiska och

det virtuella objektet är av samma storlek kommer även punkt 2a och 2b överensstämma. Slutligen behöver vi bara se till att punkt 3a och 3b hamnar på samma ställe. Detta gör vi genom att rotera det virtuella objektet kring sin egen Z-axel (Za i fig 1.6) tills dess att avståndet mellan 3a och 3b är så litet som möjligt. I idealfallet kommer 3a och 3b nu befinna sig på exakt samma ställe, men på grund av den naturliga osäkerheten i att sätta ut kalibreringspunkter för hand så kommer rotationen anses som klar när avståndet mellan 3a och 3b nått sitt minimum.



Figur 1.6 illustrerar rotationen för att punkt 3a och 3b skall överänsstämma. (Figur av författarna)

4.6 Rendering

Objekt

Det inscannade objektet renderas från en obj eller stl fil till universumet genom ett anrop till en komponent i core som returnerar en BranchGroup innehållande objektet, varvid BranchGroupen omedelbart läggs till i universumet för uppritning.

Mätdata

Arbetsgång

Den inlästa mätdatan renderas direkt vid mottagning av datan på COM-porten. Detta sker genom ett observer-observable mönster, vilket består av att en händelse i observable-objektet orsakar en händelse i observer-objektet. Datastrukturen, innehållandes all mätdata valdes till observable och grafikuppritningen valdes till observer. När ny mätdata läggs till i datastrukturen, eller när datastrukturen muteras, så anropas en metod i grafik-delen av programmet som tar som argument datastrukturen, vilket metoden direkt använder för att rendera om all mätdata till fönstret.

Grafisk Representation

Mätdata representeras grafiskt runt objektet som sfärer. Varje sfär representerar ett mätvärde. Sfären ändrar färg beroende på amplituden av mätvärdet, där ett högt mätvärde representeras med en rödare ton och ett lägre mätvärde representeras med en blåare ton.

Den slutgiltiga sfärens utseende bestäms enligt metoden nedan:

Kodexempel från filen SensorValuesDrawer.java:

Ett flyttalsvärde deklareras och initieras med värdet 0.3 för att senare användas för att bestämma hur genomskinliga sfärerna skall vara.

```
public float transparency = 0.3f; // transparency of values
```

Ett Appearance objekt skapas, i vilket man senare sätter egenskaper som man till sist applicerar på sfären.

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

Egenskapen "TransparencyAttribute" appliceras på Appearance objektet, med argumentet "transparency" som deklarerades ovan, vilket resulterar i att appearance objektet får genomskinlighet motsvarande värdet 3.0.

```
ap.setTransparencyAttributes(new
TransparencyAttributes(TransparencyAttributes.NICEST,transparency));
```

Ännu ett flyttalsvärde deklareras och initieras denna gången med amplituden hos det inlästa sensorvärdet.

```
float a = sensVal.getValue();
```

På samma sätt som TransparencyAttribute-objektet skapades ovan skapas nu ett "ColoringAttribute" objekt, vilket innehåller färgen som skall appliceras på sfären.
Färgens röda, gröna och blåa komponenter räknas ut genom ekvationen:

```
Mängden röd färg = amplitud / högsta sensorvärde
Mängden grön färg = 0 , en grön färgton indikerar att ett objekt är speciellt markerat,
vilket förklaras senare.
Mängden blå färg = 1 - (amplitud / högsta sensorvärde)
```

Detta resulterar i att vardera färgkomponent definieras som ett värde mellan 0 och 1, där 0 motsvarar ingen färg av den aktuella komponenten, och 1 motsvarar maximal färg av komponenten.
Ekvationen resulterar i ett högt mätvärde resulterar i ett högre rött index och ett lågt blått index, samtidigt som ett lågt mätvärde orsakar motsatsen.

```
ColoringAttributes color = new ColoringAttributes(new Color3f(a/255,0.0f,(1-(a/255))),
0);
```

Objektet ovan appliceras nu på Appearance objektet.

```
ap.setColoringAttributes(color);
```

Ett sfär objekt skapas med radien 0.2, 100 enheter i uppritningskvalite, utseendet definierat i Appearance objektet, och en referens till sensorvärdet.

```
SensorRepresentation sphere = new SensorRepresentation(0.2f,100,ap, sensVal);
```

Sfären läggs till i en lista innehållandes alla sfärer

```
sensorRepList.add(sphere);
```

En TransformGroup och ett Transform3D-objekt skapas, se kap. 3.3

```
TransformGroup transformGrp = new TransformGroup();
Transform3D transform = new Transform3D();
```

Egenskapen som definierar vart i rummet TransformGroup-objektet skall befinna sig initieras till sensorvärdets x, y och z koordinater

```
transform.setTranslation(new Vector3f(sensVal.getX(), sensVal.getY(), sensVal.getZ()));
```

TransformGroup-objektet flyttas till ovan nämnda koordinat

```
transformGrp.setTransform(transform);
```

Sfären ritas ut på rätt plats i universumet genom att först läggas till i TransformGroup objektet, sedan läggs TransformGroup objektet till i BranchGroupen, vilket ritar ut objektet i fönstret.

```
transformGrp.addChild(sphere);
this.addChild(transformGrp);
```

5. Resultat

Detta kapitel diskuterar den problemställning som framlades i Kapitel 1 och hur problemen löstes.

Problemställning

Positionering och rotering av sensordata

För att sensordata skall visas på samma ställe virtuellt som i verkligheten krävs att sensordatan både förstoras, förflyttas och roteras efter inläsning. Detta åstadkoms genom grafiska manipuleringar i 3d-fönstret med nästlade transformgrupper som förstorades, förflyttades och roterades i olika steg för att kalibrera objektet mot sensordatan, se kap.4.5.

Visuell representation av sensordata

För att sensordata skall visas på ett så enkelt och överskådligt sätt som möjligt i en tredimensionell vy med ett befintligt inscannat tredimensionellt objekt representeras varje sensordata-värde av en sfär, som renderas på samma koordinat som sensorvärdet inhämtades. Genom att tillägna en färg baserat på mätvärdet till sfären kan man grafiskt se ungefär i vilket område mätvärdet befinner sig, se kap 4.6. För mer exakt information om läge och mätvärde kan man utöver detta leta upp värdet i listan till höger, se kap 4.4.

Användargränssnitt

Användaren ges ett enkelt sätt att manipulera sensordata genom att vid inläsning av mätvärden populera en lista över mätvärdena, utöver uppritningen på skärmen. Denna lista ger användaren god översikt över alla mätvärden. Användaren kan även markera en eller flera mätvärden i listan. Dessa markeras även i det grafiska fönstret för att användaren skall se exakt vilka mätvärden som valts. Slutligen kan användaren välja att dölja eller ta bort de markerade mätvärdena.

6. Slutsats

Detta kapitel ger en sammanfattning av projektets helhet. Det diskuterar vad som har varit positivt och negativt med projektet, vilka användningsområden som projektet kan tänkas tillämpas på samt hur man kan gå tillväga om man önskar utföra ett liknande projekt i framtiden.

6.1 Resumé

Vad

Detta projektet ämnar att skapa ett datorprogram som kan visualisera ett inscannat 3d objekt tillsammans med insamlad mätdata kring objektet. Se kap. 1 för mer information.

Varför

Programmet är användbart i många former av produktutveckling då man konstant gör förändringar på en produkt och vill se hur produktens egenskaper förändras med varje förändring. Se kap. 1.3, för mer utförlig förklaring av syftet.

Hur

Organisation

Projektet bedrevs i två examensarbeten, där den grafiska delen av programmet utvecklades som beskrivet i den här rapporten, och logikdelen utvecklades av en separat grupp[1]. Utöver detta hanterade en tredje grupp utvecklingen av en robotarm som kan användas vid insamlingen av mätdata.[2]

Teknik

Programmet använder sig av programmeringsspråket Java med det externa biblioteket Java 3D för all grafisk rendering, samt Java Swing för alla menyer. Se kap. 3.1 och 3.2 för utförligare förklaring av Java 3D och Swing.

Utförande

Kalibrering

För att mätdatan skall renderas i korrekt storlek och rotation så utförs skalning och rotation av objektet för att kalibrera sig mot en robotarm och därmed även mot mätdatan.

För att åstadkomma detta användes metoder i Java 3D, där objektet placeras i en transformgrupp, vilken manipulerades för att åstadkomma rätt skalning och rotation. Se kap.4.5 för mer information om kalibreringsprocessen.

Visualisering

För att representera mätdata renderas en sfär för varje mätvärde, vars färg representerar mätvärdet, där en rödare ton representerar ett högre mätvärde och en blåare ton representerar ett lägre mätvärde.

Objektet läses in från en .obj eller .stl fil och placeras automatiskt framför kameran vid inläsning. Se kap. 4.6 för mer information om renderingsarbetet.

Menyer

Programmets grafiska interface har framtagits med biblioteket Java Swing. Det tillåter bl.a. användaren att genomföra kamera-manipulationer, läsa in objekt från fil, utföra kalibreringen och få en översikt över inlästa mätvärden. Se kap 4.4 för mer information om menyer och reglage.

6.2 Kritisk diskussion

Samarbetet med hårdvarugruppen har lett till god förståelse för robotarmens styrkor och begränsningar. Detta innebär att lösningen grundat sig till stor del på vad som betraktas som möjligt att genomföra med de resurser som finns till handa. Trots detta utfördes aldrig någon testning mellan hårdvara och mjukvara på grund av att hårdvarugruppen ej lyckades ta fram en fungerande arm i tid. Dock testades programmet i helhet kontinuerligt under projektets gång genom att emulera robotarmens kommunikation i mjukvara.[1]

Modulariteten i programmet är hyfsat god, men kunde varit något bättre. Dock krävs det inte en stor ansträngning för att öka modulariteten, då programmet i övrigt är väl uppdelat med klara ansvarsområden för varje klass.

De bibliotek som användes för grafik (Java Swing och Java3D) är båda rätt gamla och håller på att fasas ut av Oracle[22]. JavaFX, som är tänkt att ersätta detta, har dock inget stöd för 3D grafik ännu. Om ett liknande projekt görs i framtiden då JavaFX har blivit mer utvecklat, så är det dock definitivt att föredra framför den väg vi valde i detta projekt. Våra efterforskningar kring detta i början av projektet hjälpte mycket och vi anser att Swing och Java3D var ett gott val i vår nuvarande situation.

En klarare uppdelning och definition av programmets Core och View del i ett mycket tidigare skede hade varit att föredra. En del debatt har förts om huruvida vissa klasser hör hemma i core eller view, och eftersom samtliga i projektets mjukvarudel har arbetat med både Core och View så har gränsen blivit mycket flytande. Detta ställde dock inte till med några större problem i programmeringsfasen, utan blev snarare något som skapade en del huvudbry under rapportskrivningen.

Större fokus borde ha lagts tidigare på möjligheten att spara ner ett inskannat och färdigkalibrerat objekt med mätdata. Det är troligt att en del kod hade behövt skrivas om för att implementera en sparningsfunktion nu i efterhand. Dock fick detta lägre prioritet då vi hellre fokuserade på att få grundfunktionaliteten igång först.

6.3 Generaliseringar

Exempel på tillämpningar av programmet finns inom att visualisera och analysera magnetfält från ett kretskort, värme från en motor, dator, eller annan godtycklig data från ett valfritt objekt. Programmet kan med andra ord hjälpa företag att hitta problem med eller aspekter av sin produkt som kan förbättras. Exempelvis var på produkten värme samlas, för att med den kunskapen göra ändringar på produkten.

Man skulle också kunna använda programmet i stickprovsmätningar i produktionsled för produkter som önskas ha vissa egenskaper. Till exempel kan man testa att värmeflöden, magnetfält eller andra mätbara storheter uppfyller de krav man tidigare satt för produkten.

I forskningssyften kan programmet användas då man önskar samla in data på ett objekt som har känd form, men i övrigt är okänt. Till exempel kan man studera magnetfältet kring en udda formad magnet eller objekt med ett mätbart magnetfält, eller radioaktiv strålning kring ett kärl innehållande radioaktivt material.

6.4 Fortsatt forskning

Möjlighet att applicera texturer på objektet kan implementeras.

Integrering av 3d fönstret i ett JavaFX fönster kan undersökas för att dra nytta av det enklare och modernare menyutvecklingsverktyget.

Möjligheten att spara inlästa mätdata och ta fram dem igen på ett enkelt sätt. Exempelvis genom att lägga till att alternativ i övermenyn, under file->import. Detta skulle även kunna utökas ytterliggare med funktionalitet som att jämföra två olika mätningar sida vid sida.

Möjlighet att markera en eller flera mätpunkter med ett klick eller svep av musen över grafikfönstret, så att mätpunkter kan gömmas, istället för att markera dem i trädstrukturen i högermenyn för att komma åt den funktionaliteten.

Referenser

Samtliga online-källor verifierades 2012-05-21.

1) D. Nicklasson, J. Sandström, 2012. Sensorviewer 3D Core. Under publicering

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

3) J. Onsjö, A. Hansen, N. Johansson, 2011. *Tredimensionell Sensorscanner.* Chalmers University of Technology. <http://publications.lib.chalmers.se/cpl/record/index.xsql?pubid=155208>

4) Oracle, 1993. *Overview (Java Platform SE 7)* [Online] <http://docs.oracle.com/javase/7/docs/api/>

5) Java3d.org. *Java 3D Tutorial* [Online] <http://www.java3d.org/tutorial.html>

6) Sun Microsystems, 1997. *Java 3D API Specification* [Online] <http://graphcomp.com/info/specs/java3d/j3dguide/j3dTOC.doc.html>

7) Stack Overflow, 2009. *opengl - How to rotate an object in Java3D?* [Online] <http://stackoverflow.com/questions/1330727/how-to-rotate-an-object-in-java-3d>

8) Developer.com, 2007. *Combining Translation and Rotation in Java 3d* [Online] <http://www.developer.com/java/other/article.php/3712226/Combining-Rotation-and-Translation-in-Java-3d.htm>

9) Oracle, 1995. *Trail: Creating a GUI With JFC/Swing (The Java Tutorials)* [Online] <http://docs.oracle.com/javase/tutorial/uiswing/>

10) Oracle. *JavaFX Developer Home* [Online] <http://javafx.com/>

11) Oracle, 2008. *Overview (JavaFX 2.1)* [Online] <http://docs.oracle.com/javafx/2.0/api/>

12) Oracle, 2008. *JavaFX 1.3.1 API | Overview | Java FX* [Online] <http://docs.oracle.com/cd/E17802_01/javafx/javafx/1.3/docs/api/>

13) Oracle. *Java SE Desktop Technologies - Java 3D API* [Online] <http://www.oracle.com/technetwork/java/javase/tech/index-jsp-138252.html>

14) The Eclipse Foundation, 2012. *Eclipse - The Eclipse Foundation open source community website* [Online] <http://www.eclipse.org/>

15) Github, 2012. *Github* [Online] <https://github.com/>

16) API för Seriell kommunikation skapad av D. Nicklasson och J. Sandström, 2012. *Serial Communication API* [Online] <https://github.com/sajohan/SensorViewer3D/wiki/Serial-Communication>

17) GitHub. *Git Reference* [Online] <http://gitref.org/branching/>

18) GitHub. *Git Reference* [Online] <http://gitref.org/>

19) JogAmp.org. *JOGL - Java Binding for the OpenGL API* [Online] <http://jogamp.org/jogl/www/>

20) JMonkeyEngine, 2012. *jMonkeyEngine 3.0 | Java OpenGL Game Engine* [Online] <http://jmonkeyengine.com/>

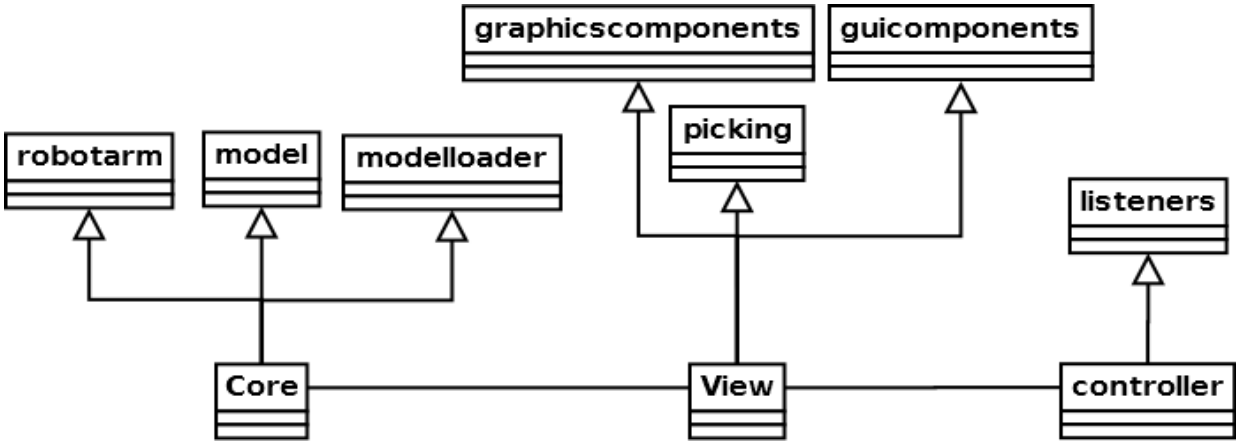
21) 3D3 Solutions, 2012. *Affordable White Light 3D Scanners - HDI 3D Scanner* [Online] <http://www.3d3solutions.com/products/3d-scanner/>

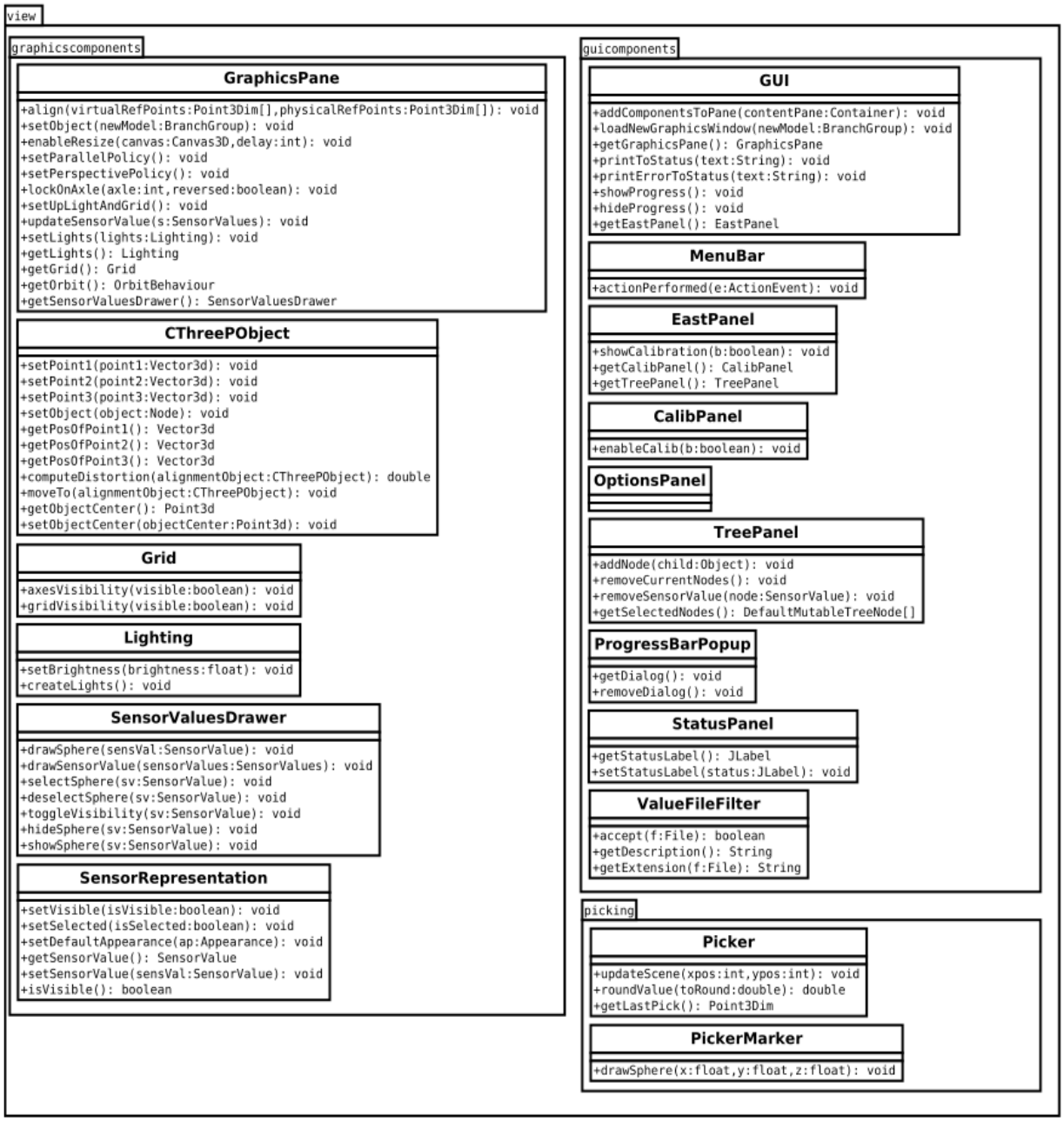
22) Oracle. *JavaFX FAQ* [Online] <http://www.oracle.com/technetwork/java/javafx/overview/faq-1446554.html>

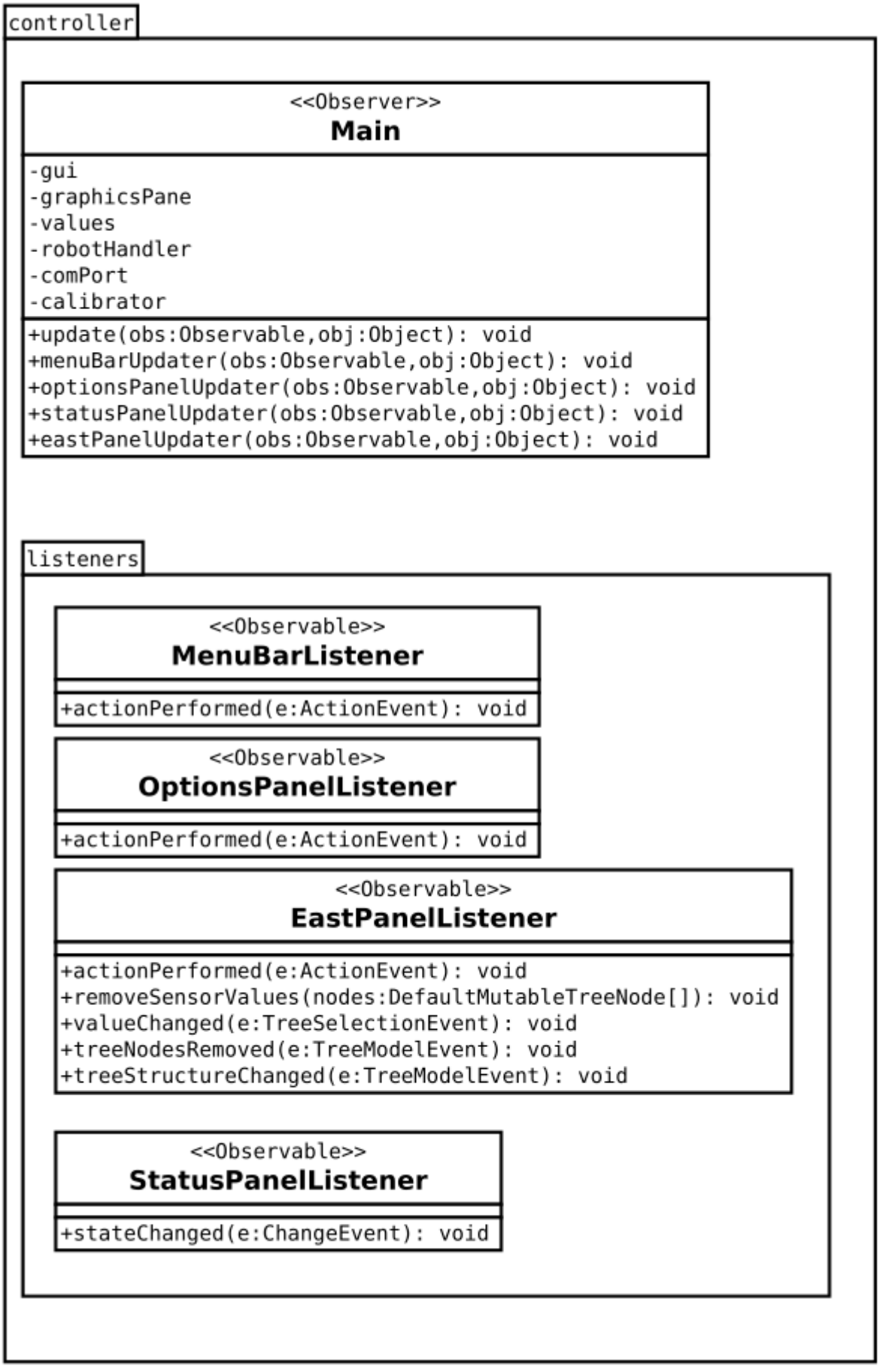
Bilagor

Kodstruktur

Nedan följer diagram över hur programmet strukturerades logiskt.




view
graphicscomponents
GraphicsPane
+align(virtualRefPoints:Point3Dim[],physicalRefPoints:Point3Dim[]): void
+setObject(newModel:BranchGroup): void
+enableResize(canvas:Canvas3D,delay:int): void
+setParallelPolicy(): void
+setPerspectivePolicy(): void
+lockOnAxle(axle:int,reversed:boolean): void
+setUpLightAndGrid(): void
+updateSensorValue(s:SensorValues): void
+setLights(lights:Lighting): void
+getLights(): Lighting
+getGrid(): Grid
+getOrbit(): OrbitBehaviour
+getSensorValuesDrawer(): SensorValuesDrawer
CThreePObject
+setPoint1(point1:Vector3d): void
+setPoint2(point2:Vector3d): void
+setPoint3(point3:Vector3d): void
+setObject(object:Node): void
+getPosOfPoint1(): Vector3d
+getPosOfPoint2(): Vector3d
+getPosOfPoint3(): Vector3d
+computeDistortion(alignmentObject:CThreePObject): double
+moveTo(alignmentObject:CThreePObject): void
+getObjectCenter(): Point3d
+setObjectCenter(objectCenter:Point3d): void
Grid
+axesVisibility(visible:boolean): void
+gridVisibility(visible:boolean): void
Lighting
+setBrightness(brightness:float): void
+createLights(): void
SensorValuesDrawer
+drawSphere(sensVal:SensorValue): void
+drawSensorValue(sensorValues:SensorValues): void
+selectSphere(sv:SensorValue): void
+deselectSphere(sv:SensorValue): void
+toggleVisibility(sv:SensorValue): void
+hideSphere(sv:SensorValue): void
+showSphere(sv:SensorValue): void
SensorRepresentation
+setVisible(isVisible:boolean): void
+setSelected(isSelected:boolean): void
+setDefaultAppearance(ap:Appearance): void
+getSensorValue(): SensorValue
+setSensorValue(sensVal:SensorValue): void
+isVisible(): boolean
guicomponents
GUI
+addComponentsToPane(contentPane:Container): void
+loadNewGraphicsWindow(newModel:BranchGroup): void
+getGraphicsPane(): GraphicsPane
+printToStatus(text:String): void
+printErrorToStatus(text:String): void
+showProgress(): void
+hideProgress(): void
+getEastPanel(): EastPanel
MenuBar
+actionPerformed(e:ActionEvent): void
EastPanel
+showCalibration(b:boolean): void
+getCalibPanel(): CalibPanel
+getTreePanel(): TreePanel
CalibPanel
+enableCalib(b:boolean): void
OptionsPanel
TreePanel
+addNode(child:Object): void
+removeCurrentNodes(): void
+removeSensorValue(node:SensorValue): void
+getSelectedNodes(): DefaultMutableTreeNode[]
ProgressBarPopup
+getDialog(): void
+removeDialog(): void
StatusPanel
+getStatusLabel(): JLabel
+setStatusLabel(status:JLabel): void
ValueFileFilter
+accept(f:File): boolean
+getDescription(): String
+getExtension(f:File): String
picking
Picker
+updateScene(xpos:int,ypos:int): void
+roundValue(toRound:double): double
+getLastPick(): Point3Dim
PickerMarker
+drawSphere(x:float,y:float,z:float): void


controller
<<Observer>>
Main
-gui
-graphicsPane
-values
-robotHandler
-comPort
-calibrator
+update(obs:Observable,obj:Object): void
+menuBarUpdater(obs:Observable,obj:Object): void
+optionsPanelUpdater(obs:Observable,obj:Object): void
+statusPanelUpdater(obs:Observable,obj:Object): void
+eastPanelUpdater(obs:Observable,obj:Object): void
listeners
<<Observable>>
MenuBarListener
+actionPerformed(e:ActionEvent): void
<<Observable>>
OptionsPanelListener
+actionPerformed(e:ActionEvent): void
<<Observable>>
EastPanelListener
+actionPerformed(e:ActionEvent): void
+removeSensorValues(nodes:DefaultMutableTreeNode[]): void
+valueChanged(e:TreeSelectionEvent): void
+treeNodesRemoved(e:TreeModelEvent): void
+treeStructureChanged(e:TreeModelEvent): void
<<Observable>>
StatusPanelListener
+stateChanged(e:ChangeEvent): void