CHALMERS

Three-dimensional sensor scanner

Tredimensionell sensor scanner

Bachelor of ScienceThesis in the Computer Engineer Programme

Andreas Hansen
Niklas Johansson
Johan Onsjº

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg Sweden 2011

Three-dimensional sensor scanner
Tredimensionell sensor scanner

Andreas Hansen Niklas Johansson Johan Onsjö

Department of Computer Engineering
Chalmers University of Technology
412 96 Göteborg

Göteborg juni 2011

Abstract

This report is a bachelor thesis presented at the department of Computer Science and Engineering at Chalmers University of Technology. The aim for the project is to find a simple method to measure and display physical quantities in a three dimensional room.
The report describes a method to construct a system for measuring different physical quantities via sensors. The system must be built in a modular and expandable way to be adaptable to different tasks.
The system explained in the report consists of two parts;
A physical scanner with moveable probes for carrying sensors and a microcontroller for controlling the probes and communicating with the next part.
An application written in Java for storing the measured values and presenting them in a lucid perspective.
It is possible to construct the scanner with different designs and circuitry, with multiplexers and flip-flops, to ensure that an unspecified number of probes can be added to the system and that they can be operated simultaneously.
The Java application is written in a modular way, using a design pattern that separates each individual part of the application. It contains exceptions and interfaces that ensure a safe and expandable application.

Sammanfattning

Denna rapport är ett examensarbete under institutionen för Data och Informationsteknik vid Chalmers tekniska högskola. Projektet ämnar att finna en enkel metod att mäta och visa fysikaliska storheter i ett tredimensionellt rum.
Rapporten beskriver en metod för att konstruera ett system som mäter olika fysikaliska storheter med sensorer. Systemet kräver att det är byggt på ett modulärt och expanderbart sätt så att det kan anpassas för olika ändamål.
Systemet rapporten förklarar består av två delar;

En fysisk scanner med rörliga sonder som bär sensorer och en microcontroller som styr sonderna och kommunicerar med den andra delen
En applikation skriven i Java som sparar mätdata och visar den i ett mänskligt överskådligt perspektiv.
Det är möjligt att konstruera scannern på olika sätt och kretsar, med multiplexrar och vippor, för att säkerställa att ett odefinierat antal sonder kan anslutas till systemet och att de kan styras samtidigt.
Java-applikationen är skriven på ett modulärt sätt, och använder sig av ett designmönster som separerar varje enskild del av applikationen. Det innehåller felmeddelanden och gränssnitt som säkerställer en säker och expanderbar applikation.

Förord

Utvecklandet av detta projekt har varit givande och utbildande, det har bjudit på många glada och sura miner under både tidiga och sena kvällar.
Vi vill ge ett stort tack till de lärare på Chalmers Lindholmen som gett oss stöd att genomföra detta projekt, framför allt Göran Hult och Robert Svensson. Men också ett speciellt stort tack till vår handledare Sakib Sistek som stått ut med oss under alla med- och motgångar vi haft under dygnens alla timmar.

Innehållsförteckning

1 Inledning

1.1 Bakgrund

I ett tidigare projekt utvecklades en elektronisk lykta och batteriladdning med trådlös energiöverföring. Frågan hurvida energifälten kan påverka kretsarna fanns men det saknades metoder för att mäta detta.

Inom arbetsområden så som produktutveckling eller materialanalys finns behovet av att läsa av sensorvärden för att analysera testobjektets egenskaper och möjliga defekter. Den traditionella metoden för att göra detta är att med enstaka avläsningar. I och med möjligheten att analysera sitt material kan interna skador så som konstruktions eller slitningar avslöjas.

Människans sinnen är trots sin stora kapacitet begränsade att ta in och analysera mätdata från sensorer då dessa oftast är en- eller tvådimensionella, här finns möjlighet att utveckla och mer anpassa ett system för mänsklig representation.
Det finns flera fördelar med att få en fullskalig bild över hur sensorvärden ligger över ett objekt.

1.2 Problem

Det saknas enkla metoder för att avläsa och visualisera mätbara storheter i ett tredimensionellt rum.

1.3 Syfte

Kartläggning av olika storheter kan bland annat

- Effektivisera placering av elektroniska komponenter så att elektromagnetiska fält som skapas av produkten inte stör dem.
- Användas för att undvika skador i en produkt då man vet hur temperatur sprider sig i omgivingen.
- Underlätta för att förstå hur elektromagnetiska fält sprider sig runt spolar.

1.4 Avgränsningar

Rapporten ägnar visa att man med redan befintlig teknologi kan skapa ett system som kan användas för att analysera och hantera mätdata från sensorer. Den kommer ta upp en stor del av den problematik och lösningar som existerar vid skapandet av tredimensionell sensorscanner.
Rapporten tar ej upp:

- Hur konstruktionen av en byggs upp. Rapporten fokuserar på tekninken och endast enklare modeller har konstruerats för testning.
- Analys av olika komponenter med samma funktion för att finna den mest lämpade för uppgiften.

2 Terminologi

Under detta kapitel beskrivs kortfattat de begrepp rapporten kommer använda sig av, om djupare förklaringar för dess funktion krävs återfinns dessa senare.

ALU	Arithmetic Logic Unit, central beräkningsdel i en processor.
API	Application programming interface
ASCII	En standard för tolkning av värden som tecken.
Assembler	Det mest grundläggande programeringsspråket i en dator.
Asynkron	En händelse interagerar med en annan händelse osynkroniserat.
AVR	En tillverkare av microcontrollers.
Bas	Mellanskiktet i en transistor
Ben	De anslutningar som finns på kretskort.
Bibliotek	En samling funktioner som kan användas inom programmering.
C	Ett maskinnära programmeringsspråk.
C++	Ett maskinnära och objektorienterat programmeringsspråk.
CAD	Computer Aided Design, används inom industrin.
Class-fil	En kompilerad javafil som kan läsas in av maskinen.
Community	En samlingsplats för likasinnade på internet.
Dataord	En samling av data.
DataSpaceManager (DSM)	Den klass i Java-applikationen som hanterar mätvärdena.
Datatyper	Sätt att tolka data i programmering med olika storlekar.
Debugga	Felsöka
Deklarera	Avsätta minne för att hålla ett visst värde.
Direct3D	Ett API för att rendera 3D-grafik i windowsmiljö.
Display	Ett fönster där 3d-scenen ritas upp.
EEPROM	Electronically Erasable Programable Read Only Memory, minnestyp som programkod ofta är sparad i på microcontrollers.
Elektromagnet	En spole runt en kärna som när energisatt skapar ett magnetfält.
Emitter	Yttre skickt i en transistor.
Ferrit	En variation av järn som har kubisk kristallstruktur med goda magnetiska egenskaper.
Flash-minne	En speciell typ av EEPROM.
Flatkabel	Anslutningskable med paralella ledare.
GND	Definitionen gör gjort i elektriska system.
Grindar	Komponent som kan blockera eller släppa igenom en elektrisk signal.
Header-fil	En programmeringsfil som definierar de funktioner som används.
Hårdvaruaccelererad	Användandet av en dedikerad processor för att utföra beräkningar.
Högnivåprogrammering	Ett programmeringssätt där man bortser från hur underliggande programstruktur beter sig utan låter kompilatorn lösa denna.
Identifier	En textsträng för att addresera meddelanden till och från Arduinon.
Inportar	Gränssnitt för inläsning av data till en processor.
Java	Objektorienterat multi-plattformsprogrammeringsspråk.
Kamera	Utgångpunkten för applikationens synvinkel.
Kanal	En väg för en ström att färdas.
Klockcykel	Minsta diskret indelbara operationstid för en processor.
Kollektor	Yttre skickt i en transistor.
Kompilerare	Program som översätter programkod till maskinkod.

Lågnivåprogrammering	Ett programmeringssätt där man jobbar närmare processorn och minnen.
Objektorienterat	En programmeringsmetod där man samlar sin kod i objekt som kan återanvändas för snabbare utveckling.
Open Source	Mjukvara med offentlig källkod.
Main-fil	Den fil som först startas vid exekvering.
Matris	Ett rektangulärt schema av tal eller storheter.
Mhz	Mega Hertz, svängningar per sekund, enhet för frekvens.
Mikrocontroller	Ett datasystem på en integrerad krets med mikroprocessor, minne, in-/utgångar och ofta analog-/digitalomvandlare.
Mikroprocessor	En processor i en integrerad krets.
MIPS	Miljoner Instruktioner Per Sekund, enhet för beräkningshastighet.
Netbeans	En utvecklingsmiljö för många olika programmeringsspråk.
Parameter	Ett värde som skickas till eller returneras från en funktion eller klass.
Pipeline	Används generellt för att beskriva en förändring från ett läge till ett annat.
Pixel	Minsta beståndspunkten i en grafiks bild.
Protokoll	En uppstrukturerad definition av hur kommunikationen fungerar.
Pseudokod	Används för att beskriva ett tankesätt snare än en korrekt syntax.
RAM	Random Access Memory, arbetsminne till en processor.
Ramverk	En samling av användbara klasser och bibliotek.
Relä	En elektronisk strömbrytare.
Rendera	Rita upp en grafisk bild.
Serieport	Ett hårdvarugränssnitt för bitvis informationsöverföring.
Sond	En rörlig konstruktion med sensorer.
Spolar	En spiraliserad ledande tråd.
Static Block	Ett block av kod som kan köras utan att ett obejkt skapas.
Sun Microsystems	Organitationen bakom programmeringsspråket Java.
Synkron	Händelser interagerar med varandra i synkronisation.
Tidskritiskt System	Ett system där funktioner måste ske inom en viss tidsram.
Transistor	Halvledarkomponent som kan förstärka och styra elektriska signaler.
Transistorförstärkare	Förstärkare av transistorer.
USB	Universal Serial Bus, en typ av seriell kommunikation.
Utportar	Gränssnitt för skrivning av data från en processor**.**
Vippor	Logisk krets som kommer ihåg ett digitalt värde.

3 Metod

3.1 Generellt om problemet

För att kunna mäta av storheter i ett rum krävs att sensordata kan insamlas från olika punkter. För detta behövs en metod för att manövrera sensorer samt ett sätt att veta exakt var sensorn befinner sin när mätdata inhämtas. Problemet tycks kunna lösas av ett system där en eller flera sonder utrustas med en eller flera sensorer. En konstruktion krävs som sonderna kan monteras på så att de är rörliga. Motordrift för att skapa rörlighet i sonderna och teknik för att mäta hur sonden är positionerad.

För att visa detta krävs en applikation som på ett användarvänligt sätt kan visa mätvärdena tredimensionellt och slutligen kommunikation mellan hårdvaran och applikationen för överföring av mätdata.

För att effektivt kunna mäta storheter krävs att systemet har stöd för en stor mängd sonder och sensorer och då mätning av olika storheter har olika krav på mätmetod krävs även att systemet är modulärt.

3.2 Upplägg av projektet

Projektet ämnar att ta fram tekniken för att bygga ett modulärt och expanderbart system för mätningar och visning av storheter i ett rum och identifiera problemen som kan uppstå.

För styrning av hårdvaran används en Arduino Mega 2560 *microcontroller* då den är enkel att använda och det då inte behöver läggas mycket tid på att lära sig ett nytt system. Sensorer för positionering av sonderna behövs. Hall-sensorer som mäter magnetfält används för att testa utrustningen. Applikationen programmeras i Java och OpenGL används för 3d-uppritningen. Projektet delas in i tre huvudområden:

- Hårdvara
- Kommunikation
- Applikation
- De tre projektmedlemmarna får ett varsitt område som huvudansvar.

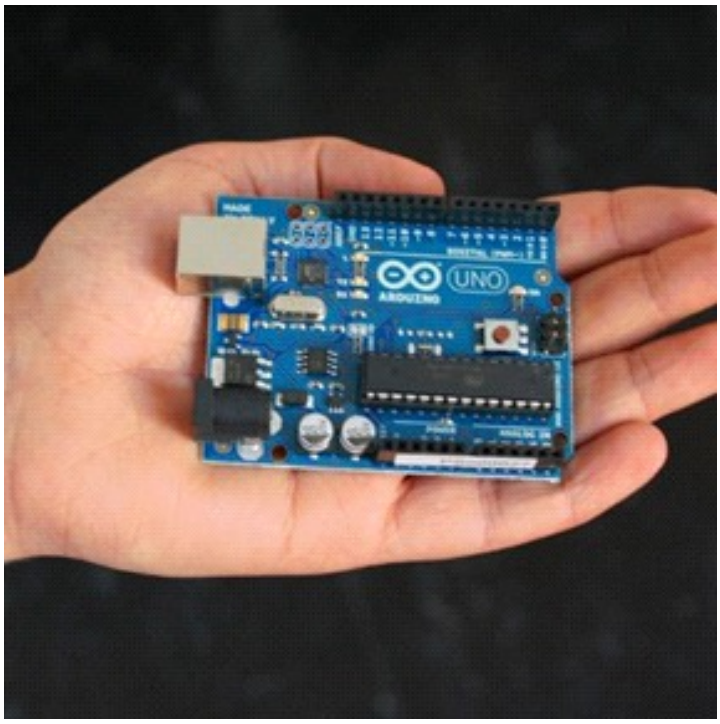
3.3 Kunskapskrav

För att kunna genomföra projektet krävs kunskap inom följande områden:

- Elektriska kretsar
- Sensorer
- *Microcontroller*
- Programering av *microcontroller* (C/C++)
- Programering i Java
- OpenGL

4 Teoretisk bakgrund

4.1 Arduino Kortet



Figur 4.1.a: En Arduino UNO, foto av Arduino teamet.

Arduino är ett italienskt *open source*-projekt som utvecklar kretskortslösningar för mikroprocessorer, de använder Atmega-processor och bygger runt detta en lätthanterlig miljö för att ansluta olika elektriska komponenter så som motorer och sensorer.
Kortet är designat för att vara så lättanvändligt som möjligt
Arduino Mega 2560 är ett kort med många anslutningsmöjligheter och lämpar sig därför vid utveckling av lite större projekt. Alla olika Arduino-kort använder sig av ett programspråk som är likt C++ och har en egen utvecklingsmiljö som är skriven i Java. Det har förenklat flera av de funktioner som vanligtvis finns under programmering av mikroprocessorer genom att skriva färdiga klasser för hantering av dessa, detta snabbar upp utvecklingen av elektriska system då man slipper justera de småinställningar som krävs för att få igång en *microcontrollers* funktioner.[1]

4.1.1 ATmega1260

Arduino Mega 2560 är baserad på AVR:s 8-bitars microcontroller ATmega1260. ATmega1260 använder sig av 'AVR enhanced RISC'-arkitektur som uppnår närmare 1 MIPS per MHz då de flesta instruktioner i arkitekturen tar endast en klockcykel att genomföra. Detta genom att alla 32 register är direkt kopplade till ALU:n (Arithmetic Logic Unit). ATmega2560 har:[2]

- 128kB flash-minne
- 4kB EEPROM
- 8kB RAM
- 86 fleranvändnings in-/utportar

4.1.2 Arduinos utvecklingsmiljö

Arduino har tillverkat sin egen utvecklat sitt eget programspråk som bygger på Wiring.
Wiring är en programmeringsmiljö som utvecklats för att ha närmare kontakt med elektroniska komponenter som sensorer och givare, språket utvecklades av Hernando Barragán på Los Andes Univerisitet för Arkitektur och Design.[3]

Programmeringskod skriven med Arduinos utvecklingsmiljö kallas för “sketches”, eller skisser på svenska. Dessa skisser är kompilerbar kod som liknar C++. Arduino har skrivit färdiga klasser för att hantera såväl mikroprocessorns olika inbyggda funktioner så som avläsning av portar, men även mer externa funktioner som drivningen av en stegmotor eller seriell kommunikation.
Arduinos utvecklingsmiljö innehåller hjälpmedel så som en texteditor, en konsoll och funktioner för att kompilera skisser och ladda ner dessa i kortets EEPROM-minne.[1]

Programmering i denna miljö sker väldigt enkelt genom att definiera en funktion som heter “setup” och en som heter “loop”. Först kommer funktionen setup att kallas och köras en gång, sedan kallas loop om och om igen tills antingen strömmen bryts eller resetknappen trycks ner.
Koden för detta ligger i en main-fil och har följande kodrader i sig:

```
1. setup();
2. while( true )
3.         loop();
```

Dessa kodrader ligger inbakade i Arduino-miljöns systemfiler och är vanligtvis inte nödvändiga att ändra i.

4.1.3 Bennummer för analoga signaler

När man arbetar med programmering av skisser i Arduino’s egna utvecklingsmiljö så finns de analoga portarna redan fördefinierade som A0 till AN, där N är antalet analoga portar ens modell på kortet har.

Om man vill arbeta med dessa portar utanför utvecklingsmiljön genom att till exempel externt skapade bibliotek eller styra mikroprocessorn via seriekommunikation så finns inte dessa deklarationer längre tillgängliga, detta kan man komma undan genom att kolla vilka nummer A0 till AN motsvara decimalt och definera dessa decimalt i sin egen kod.

Ett alternativt sätt till detta är att i startskedet av sitt programköra en funktion som kommunicerar med mikroprocessorn och dynamiskt frågar vad definitionen [A0, AN] har i Arduino’s miljö för att sedan sätta samma decimala definition i sin egen kod. Denna metod skulle eliminera problemet att man byter typ av kort som har en annorlunda definition för ben-nummerna.

4.2 Sensorer

Sensorer som kopplas till mikroprocessorer använder oftast signalstyrka på 0-5V, denna signal läses av med en analog inport på mikroprocessorn och kan på så sätt användas under exekvering av olika program. De drivs vanligtvis med 5V spänning som kan tas direkt från mikroprocessorn.

4.2.1 Avståndssensor - Sharp GP2D120

Det finns många olika sätt att mäta avstånd med sensorer, allt från Ultraljud till Laser. Sharps sensorer använder ljus i det infraröda spektrat som avges från en IR-lysdiod, detta ljus speglas på en reflektiv yta till en IR-sensor placerad jämte lysdioden. Avståndssensorn drivs med hjälp två kopplingar som går till 5V och gjord, en tredje koppling avger spänning från IR-sensorn vilken inte är linjär mot det avstånd den reflektiva ytan befinner sig på. Detta betyder att en omvandlingstabell måste användas för att räkna ut avståndet utifrån spänningen, denna tabell kan även skapas själv för att ge den optimala kurvan för ens egna förutsättningar.
Dess mätområde ligger mellan 4 och 30cm.

4.2.2 Hall-sensorer

När ett magnetfält passerar genom ett material som leder en ström uppstår en ström vinkelrätt som kan mätas. Hall-effektssensorer utnyttjar detta fenomen för att mäta magnetfält. En hall-sensor (A1301) har tre ben, ett ansluts till 5V och ett till jord. Spänningen i de tredje benet varierar då sensorn påverkas av magnetfält.[4]

4.3 Java Native Interface

Java Native Interface(JNI) är ett ramverk för att kommunicera mellan Java och andra programspråk så som C, C++ och assembler. Det är utvecklat för att utvidga funktionaliteten hos Java så det kommer åt mer maskinnära operationer, det används även då man har tidskritiska system.

Användningsmässigt under Java-programmering fungerar det som att kalla på en helt vanlig funktion, men att få allt att fungera felfritt är en komplicerad process. För att få applikationen att fungera så smidigt som möjligt så bör man försöka hålla sig i det ena programspråket så länge som möjligt. Vid kommunikation mellan de olika språken bör man begränsa denna i största möjliga mån, och vid kommunikationen bör man arbeta med så grundläggande datatyper som möjligt.[5]

4.3.1 Implementering av JNI i NetBeans

För att smidigt kunna arbeta med *JNI* under *NetBeans* så skapar man lämpligen två separata projekt, ett C/C++-projekt för Dynamiska Bibliotek och ett Java-projekt och sedan använda programmet javah för att skapa header-filer ur Java-koden. Javah finns inbyggt i Sun's *Java Development Kit.*

För att enkelt skapa en godtycklig *header-fil* till C-projektet skapar man först funktionsdeklarationerna i Java-projektet genom att lägga till ordet “native” före dessa, då vet programmet javah att funktionerna är länkade till ett externt bibliotek(vilket sedan skall skapas). När dessa är deklarerade så körs programmet javah med sökvägen till class-filen för Java-koden som inparameter. Javah skapar då en *header-fil* som i sin tur kan importeras till C-projektet.

C-projektet skall vara inställt för att skapa ett 32-bitars bibliotek(*Shared Object* i Unix och *Dynamic-link Library* i *Windows*) vilket ställs in med kompilerings-parametrarna -shared -m32.
Ut-filen för C-projektet bör ligga i Java-projektets katalog, lämpligtvis i en underkatalog som heter till exempel “lib” för “library”, denna ställs in med en absolut sökväg i runtime-parametern “Djava.library.path”.

För att få Java-projektet att acceptera det skapade biblioteket måste man lägga till kodraden “System.load(“sökväg/till/bibliotek.h”);” inom ett *static-block*. Detta gör att biblioteket laddas vid kompilering och funktionerna som finns i det nu laddade biblioteket kan kännas igen av Java-kompilatorn under resten av kompileringen och sedan under körning av programmet.

4.3.2 Nackdelar

Java Native Interface innehåller som de flesta andra program en del egenheter vilka bör finnas i åtanke under utveckling med detta ramverk.

Under exekvering av java-kod skriver JNI inte ut utskrifterna från C-biblioteket direkt utan med en fördröjning, detta medför en svårighet att debugga med hjälp av utskrifter och en omöjlighet att använda blandade utskrifter från de olika språken som till exempel redovisning för användaren.

4.4 OpenGL

OpenGL är ett plattformsoberoende *API* för grafisk uppritning i två eller tre dimensioner. *OpenGL* används i allt från datorspel till *Computer Aided Design*. *APIet* bygger på matrisoperationer vilket gör det snabbt och smidigt. *OpenGL* är skrivet i C-kod, men används även av andra programspråk via diverse lösningar, så som *JNI* i Java. *OpenGL* är ett lågnivåAPI därmed måste programmeraren rendera scenen steg för steg.

*OpenGL*s grundläggande funktion är att konvertera elementära figurer så som punkter och linjer till pixlar. *OpenGL State Machine* används vid konverteringen. *OpenGL State Machine* är en *graphics pipeline* som steg för steg omvandlar indatan till pixlar. De flesta kommandon i *OpenGL* skickar dessa fundamentala figurer till pipelinen eller påverkar hur pipelinen tolkar dessa figuer.

En eller flera figurer bildar en scen. Med scen avses allt som ska ritas ut i den grafiska bilden. Scenen kan påverkas med olika egenskaper exempelvis rotera objekt.[6]

4.4.1 OpenGL Graphics Pipeline

OpenGL Graphics Pipeline är en Rendering Pipeline för ovandling av figurer till pixlar. Detta sker i flera steg enligt löpandebandprincipen.[7]

- ***Vertex Specification*** *- Specifikation av utgångspunkter*
    - Applikationen skickar alla utgångspunkter för de figurer som ska ritas till pipelinen. Här finns även *Vertex Array Objects* som definierar vilken data som finns till varje punkt och *Vertex Buffer Objects* som innehåller själva datan.
- ***Vertex Processing*** *- Bearbetning av utgångspunkter*
    - Alla attribut i ursprungsdatan bearbetas och omvandlas till data baserat på ett användardefinierat program.
    - Varje enskild inkommande utgångspunkt motsvaras av exakt 1 utparameter.
- ***Primitive Assembly*** *- Grundläggande sammansättning*
    - Generar grundläggandefigurer med hjälp av utdatan från *Vertex Processing*.
    - Fungerar olika beroende på vilken typ av figur som ska ritas ut.
- ***Clipping*** **och** ***Culling*** *- Urklipp och utsortering*
    - De objekt som inte direkt syns från kamerans synvinkel renderas inte.
    - De objekt som ligger på gränsen delas upp i mindre objekt. Det som inte syns renderas inte.
- ***Rasterization*** *- Fragmentering*
    - Objekten delas upp i fragment.
    - Fragment innehåller data från tidigare steg och även fragmentets position.
- ***Fragment Processing*** *- Bearbetning av fragment*
    - Fragmenten får färg och ett djupvärde.
- ***Pre-Sample Operations*** *- Testning och uppritning*
    - ett flertal valfria test genomförs. Om ett test misslyckas så renderas inte fragmentet.
    - För varje färg genomförs en sammansmältning med färgen som redan finns på den positionen.
    - All fragmentdata skrivs till *framebuffern*. Scenen uppdateras på *displayen.*

4.5 Java 3D

Java 3D Api:et utvecklas på *communityt* Java.net, det är en samling småprojekt som tillsammans bygger upp den totala funktionalitet som *Java 3D* erbjuder. I grunden bygger *API:et* på funktioner som portats via *Java Native Interface* från en hårdvaruaccelererad 3d-motor så som *OpenGL* eller Microsofts *Direct3D.*

API:et tillåter skapandet av tredimesionell grafik för till exempel fristående applikationer eller applets för hemsidor. Det tillåter högnivå-objekt som kan manipuleras och enkelt ritas upp istället för att direkt manipulera uppritningen som man gör i en ren *OpenGL*- eller *Direct3D*-applikation. En *Java3D*-applikation använder olika sorters grupper för att skapa scenen. Dessa grupper bildar tillsammans en trädstruktur. Trädstrukturen ger möjlighet att påverka många figurer på samma gång.[8]

4.5.1 Exempel

Nedan följer ett psuedokodsexempel på hur *java 3D* kan användas:

```
1.  root = new RootObject()
2.  property1 = new Property(keyboardRotation)
3.  property2 = new Property (mouseRotation)
4.  cube1 = new Cube()
5.  cube2 = new Cube()
6.  points = new PointArray()
7.  property1.add(cube1)
8.  property1.add(points)
9.  property2.add(cube2)
10. root.add(property1)
11. root.add(property2)
12. root.render()
```

Först skapas ett rotobjekt som håller hela scenen. På rad 2 och 3 skapas två olika egenskaper, tangentbordsrotation och musrotation. På rad 4-6 skapas olika geometriska figurer. Två kuber och en *array* med flera punkter. På rad 7-8 får den ena kuben och *arrayen* med punkter egenskapen tangentbordsrotation. På rad 9 ges den andra kuben egenskapen musrotation.
På rad 10-11 så läggs alla objekt med med egenskapen tangentbordsrotation repektive musrotation till scenen. Rad 12 ritar upp scenen.

När scenen sedan har ritats upp och tangentbordsrotation används så kommer den första kuben och alla punkterna rotera.
Används musrotation så kommer den andra kuben att rotera.

4.6 Seriell Kommunikation

Seriell kommunikation skickar information på en ledning till skillnad från parallell kommunikation där informationen istället skickas genom flera parallella ledningar. Oftast används två ledningar med en dedikerad sändare och mottagare för att undvika kollision på ledningen, som inträffar när båda parter samtidigt försöker skicka eller ta emot data.

Kommunikationen sker genom att logiska ettor och nollor skickas efter varandra i ett bestämt tidsintervall. Ettorna och nollorna motsvarar binära siffror vilket i sin tur motsvarar tecken ur ASCII-tabellen. Seriell kommunikation finns såväl i datorns serieport som i *USB-portar*, det är ett effektivt

sätt att överföra data mellan olika fysiska enheter. Seriell kommunikation används i två huvudgrupper: synkron och asynkron överföring, vilka beskrivs i följande kapitel.

I och med överföringen skickas även en extra last som inte är del av det skickade meddelandet, anledningen till detta är att man behöver ha koll på när en överföring börjar och slutar. Denna last kallas *overhead* och består av extra tecken som strukturerar upp kommunikationen.[9]

4.6.1 Asynkron Överföring

När data överförs asynkront menas det att man skickar de olika tecknen i meddelandet ett och ett. Dessa tecken avgränsas oftast med en startbit och en stopbit, det kan även finnas möjlighet att lägga till en paritetsbit med vilken fungerar som en säkerhetskontroll för att se till att det inte mottagits ett korrumperat tecken. Detta ger en mindre felbenägen kommunikation vilket lämpar sig vid mindre hastigheter och system som kräver felsäker överföring, men ger också en större *overhead*.

4.6.2 Synkron Överföring

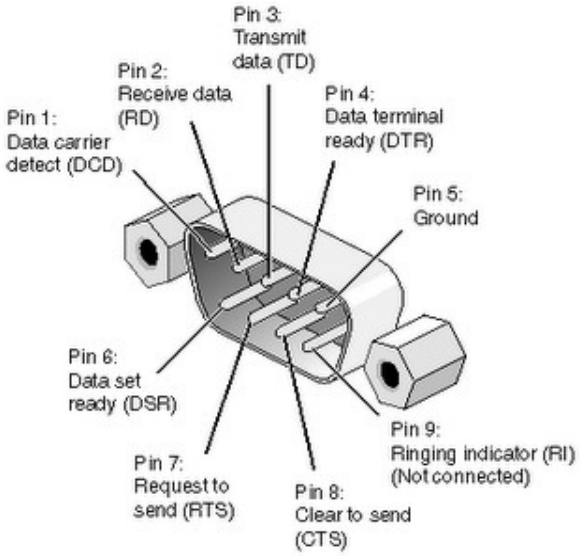
Med synkron överföring menas att datan som skickas över ledningen inte skickas tecken för tecken med start och stopbitar utan i klumpar, så kallade datablock. Detta innebär mindre overhead eftersom start- och stopbitar mellan tecknen är borttagen.

För att denna kommunikation ska vara möjlig krävs att både sändare och mottagare följer ett väldefinierat protokoll. De måste även ha en gemensam klocka så båda sidor av kommunikationen håller sig synkroniserade. Detta resulterar i större datahastigheter vilket bättre lämpar sig för överföring av större filer som inte har lika högt krav på felfrihet så som bilder och musik.

4.6.3 RS232

RS232 är protokollet för seriell kommunikation från datorns seriella port, denna port är tradionellt en föregångare till *USB*-porten som numera är mer använd.
Även om användandet av RS232-porten minskar så har de flesta datorer fortfarande stöd för denna, även om en port saknas på själva datorn kan man koppla en *USB* till RS232-konverterare till sin *USB*-port. Maxlängden på den seriella kabeln definieras i protokollet till 15,24 meter eller en kabel med kapacitans 2500pF.[11]

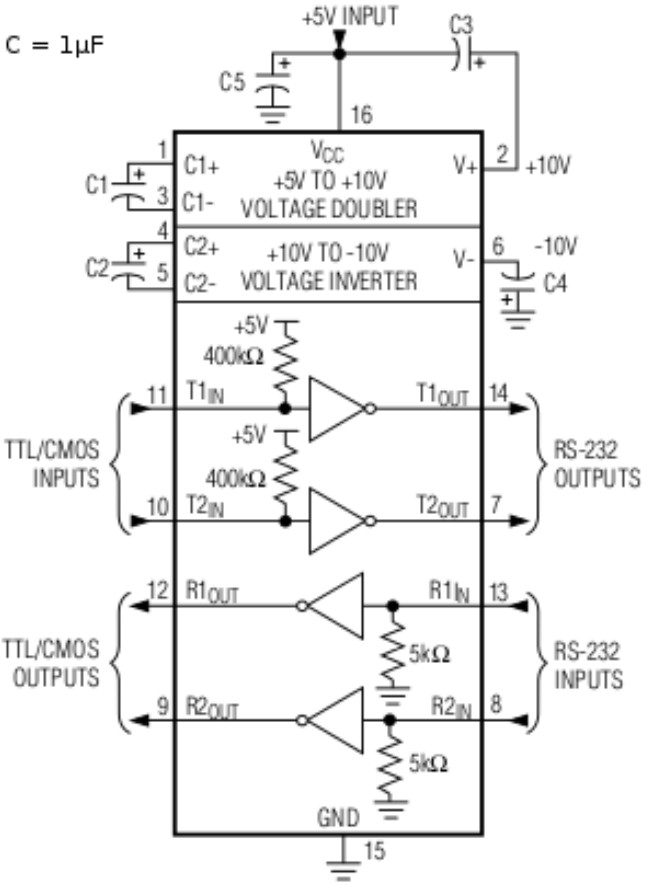
Protokollet definierar även de metoder som kan användas för felsökning, detta genom att lägga till en paritetsbit. Paritetsbiten är en logisk etta eller nolla beroende på vad som definierats. Denna metod är inte helt felsäker eftersom så länge antalet felaktiga bitar är ett jämnt antal så tror mottagaren att meddelandet är korrekt.[10]



Figur 4.6.3.a: En port(hane) för seriell anslutning och vilka signaler som går till vilket ben.[20]

4.6.4 MAX232CPE

Kretsen MAX232CPE används för att omvandla signalstyrkan från PCn:s seriella utport på 12V till Arduinokortets seriella inport på 5V, detta genom spänningsdelning. En fördel med denna krets är att den kan drivas med +5V som kan tas direkt från Arduinokortet vilket tar bort behovet av en extern strömkälla för att driva dc-konverteraren. Kretsen klarar datahastigheter upp till 200kbps. Och kopplas vanligtvis enligt kopplingsschemat som illustreras i figur4.6.4.a.[12]



Figur 4.6.4.a: Det vanligaste kopplingsschemat för användandet av den integrerade kretsen MAX232CPE.

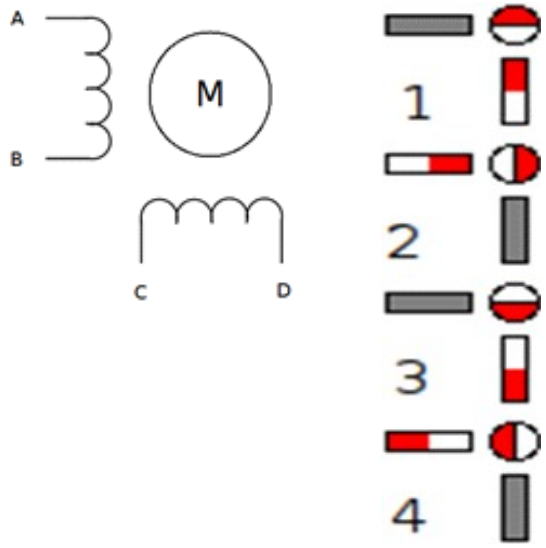
4.6.5 Baud Rate

Baud Rate är ett mått för hur många dataord per sekund som skickas via seriell kommunikation, eftersom storleken på dessa dataord kan varieras så är dess hastighet i ett mer påtagligt mått som *bps*(*bytes* per sekund) inte bestämbar förens man deklarerat alla parametrar för kommunikationen. Den vanligaste längden för dataord i nuläget är 10 bitar(en startbit, 8 bitars data och en stopbit) vilket resulterar i att *baud raten* i detta fall blir 10 gånger så hög som *bps* för överförningen.[13]

4.6.6 MiWi

Miwi är ett protokoll för trådlös kommunikation som utvecklats av *Microchip Technology*. Protokollet använder sig av IEEE 802.15.4 standarden, vilken är densamma som för trådlösa nätverk. Det är anpassat för att ha låg strömförbrukning och användas för låg datahastighet under korta avstånd, kostnaden för komponenterna är även låg vilket underlättar billig utveckling av produkter som använder sig av trådlös kommunikation.[14][15]

4.7 Stegmotor



A-B	C-D
+	0
0	+
-	0
0	-

A-B	C-D
+	0
+	+
0	+
-	+
-	0
-	-
0	-
+	-

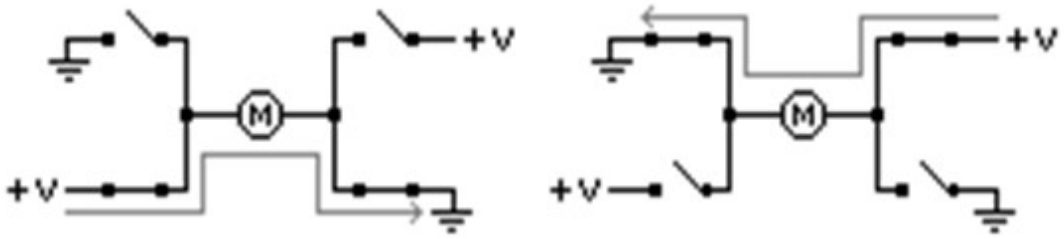
Figur 4.7.a: En stegmotor och sekvenser för energisättning av spolarna för drivning. Den ena sekvenser är enklare och kräver mindre energi medans den andra utnyttjar halvsteg för utökad noggrannhet på stegen.

En stegmotor drivs genom att elektromagneter energisätts stegvis så att motorn roterar ett steg i taget. Stegmotorns rotationshastighet och riktning styrs med vilken ordning elektromagneterna energisättssätts.

En stegmotor ges spänning med varierande polaritet för att kunna attrahera och repellera rotorn. Minimalt krävs fyra steg för att slutföra en cykel. Genom att använda andra sekvenser kan stegmotorn ta mindre så kallade halvsteg.

4.8 H-brygga

En H-brygga används när strömriktningen behöver kunna gå i båda riktningarna. Genom att öppna och stänga grindar kan en krets spänningssättas med varierande polaritet.



Figur 4.7.1.a: Strömmen går genom en h-brygga där de undre grindarna är stängda och passerar motorn i mitten i högerriktning. När se undre grindarna öppnas och de övre sluts så går strömmen genom motorn i andra riktningen istället.

H-bryggor är ett vanligt sätt att styra riktningen på motorer och kan byggas av transistorer eller reläer. Det kan vara av intresse att bygga in säkerhet så att fel grindar inte kan öppnas av misstag vilket kan resultera i att kretsen kortsluts.[16]

5 Genomförande

Detta kapitel avser att upplysa om hur en funktionsmodell för en tredimensionell sensorscanner kan utvecklas.

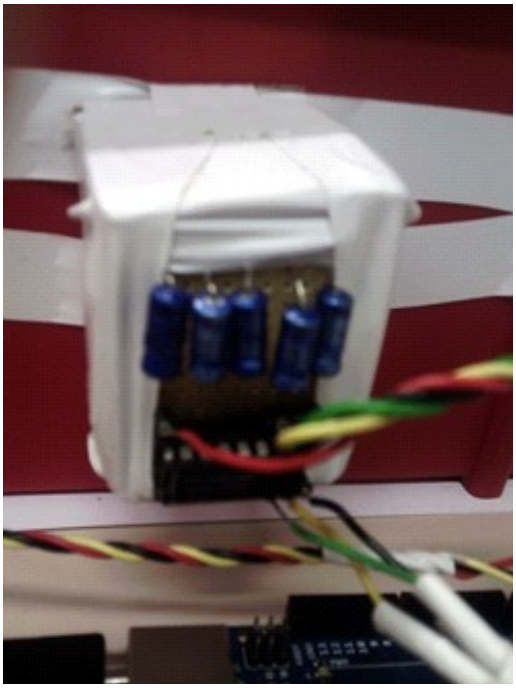
5.1 Kommunikationsprotokoll

För att få en effektiv och stabil kommunikation mellan Java-applikationen och Arduino-kortet definierades ett protokoll för hur denna sker.
Funktionerna inleds med en handskaksrutin som bekräftar att både Arduinons kod och Java-applikationens kod är *synkade* i samma händelseförlopp. Denna handskaksrutin fungerar genom att det ena programmet skickar ett meddelande till det andra som bekräftar detta med ett svarsmeddelande. Fram tills båda programmen mottagit ett meddelande från den andre hindras fortsatt exekvering av programmet.

Varje meddelande från Java-applikationen skickas med en identifierare vilken Arduinon använder för att svara till rätt funktion. Java-applikationens funktion för att ta emot ett meddelande kräver en identifierare som inparameter, vilket innebär att processen som väntar på ett meddelande inte lyssnar på meddelanden som skickats med en annan identifierare. Om ett meddelande inte kommer fram inom en inställningsbar tidsram så skapas det ett felmeddelande som talar om att ett förväntat meddelande inte kommit fram. Om det däremot skickats ett meddelande som inte kommit fram så sparas detta i en *buffer* och skrivs ut när applikationen stängs ner, detta för att möjliggöra *debugging* vid konstigt beteende.

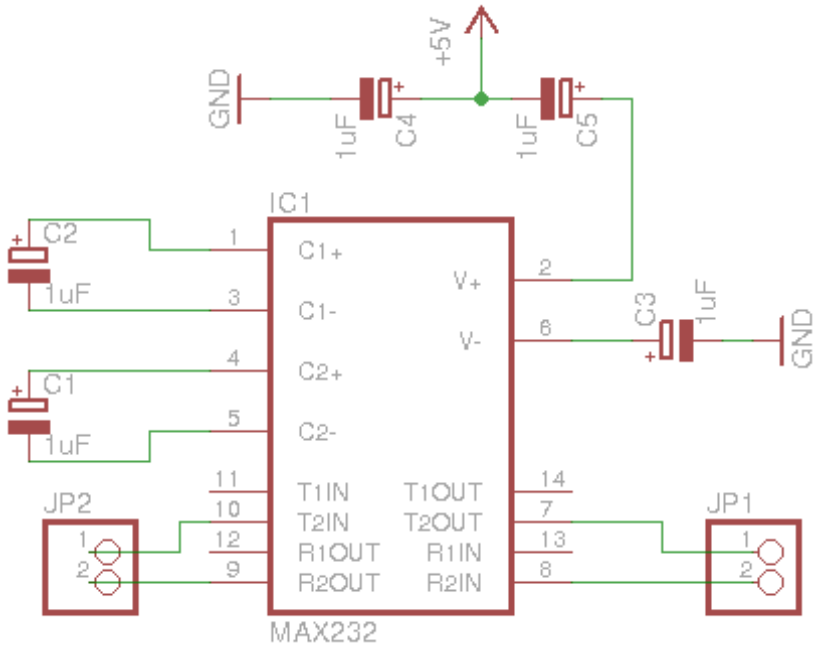
5.1.1 Fysikaliska problem vid seriekommunikation

Kommunikationen mellan hårdvaran och mjukvaran möjliggörs mellan Arduino-kortets serieport och serieporten på den dator som kör Java-applikationen.



Figur 5.1.1.a: Bild på den konstruerade kretsen för dc-omvandling.

Då Arduinokortet jobbar med spänning på 0V för en logisk nolla och 5V för en logisk etta motsvarar inte detta datorns serieport som arbetar med 0V för en nolla och 12V för en etta vilket definierats i RS232, lösningen på detta finns genom att konstruera krets för dc-omvanding(som kan ses i figur 5.1.1.a) med den integrerade kretsen MAX232CPE och 5 kondensatorer. Till denna kopplas jord, receive- och transmitsignal och både från datorn och från Arduinon, det kopplas även 5V från Arduinon vilket driver hela kretsen. Detta kan ses i kopplingsschemat i figur 5.1.1.b.



Figur 5.1.1.b: Kopplingsschema för dc-omvandlaren. Till JP1 kopplas sladdar från datorn, på ben 1 kopplas receive-signalen och på ben 2 kopplas transmit-signalen. Till JP2 kopplas sladdar från Arduinon, på ben 1 kopplas transmit-signalen och på ben 2 kopplas receive-signalen.

5.1.2 Överföringsalternativ

Som alternativ till trådbunden fysisk överföring skulle man kunna använda trådlösa överföringsmetoder så som *MiWi*, detta skulle möjliggöra en friare arbetsmiljö då korta sladdar inte längre behöver dras mellan dator och centralenheten.

Eftersom Java-applikationen är modulärt uppbyggd går det att byta ut klassen som hanterar seriekommunikationen och skapa en ny klass som kan hantera trådlös överföring.
Arduinon har även färdigkonstruerade komponenter som kopplas på kortet för att möjliggöra trådlös överföring.

5.1.3 Låsning vid seriell kommunikation

Under seriell kommunikation med Arduino-kortets första serieport så sker det en låsning som hindrar programmeraren i utvecklingsmiljön att få kontakt med kortet, detta sker enbart vid sändning och mottagning av data så om man låter kommunikationen lugna ner sig lite så går det att få kontakt med kortet från Arduinos utvecklingsmiljö och ladda ner ny kod.

En oönskad situation som kan uppstå är att man tidigt i sitt program skriver en funktion som hela tiden skickar data på serieporten, oavsett om detta är med avsikt eller av misstag så blockerar det möjligheten att ladda ner ny kod till kortet.

För att lösa detta kan man ägna flera försök åt att starta om kortet sen snabbt påbörja en ny nerladdning, har man tur så träffar man in den korta tidsram då kortet är igång och kan ta emot signaler fast ännu inte har hunnit köra igång programmet som ligger inprogrammerat.
En säkerhetsrutin som bör byggas in före körningen av en enda egenskriven rad med kod beskrivs följande:

```
1.  void setup(){
2.          pinMode(53, INPUT);
3.          if( digitalRead(53) == HIGH )
4.                  for(;;); /* ever */
5.          //resten av koden...
6.  }
```

Det denna kod gör att om man från en 5V utgång kopplar en sladd till den digitala ingången 53(sista ingången på Arduino Mega 2560) så kommer programmet stanna innan resten av koden hinner köras, då denna kod kan innehålla funktioner som gör att man förlorar kontakten med kortet från utvecklingsmiljön.

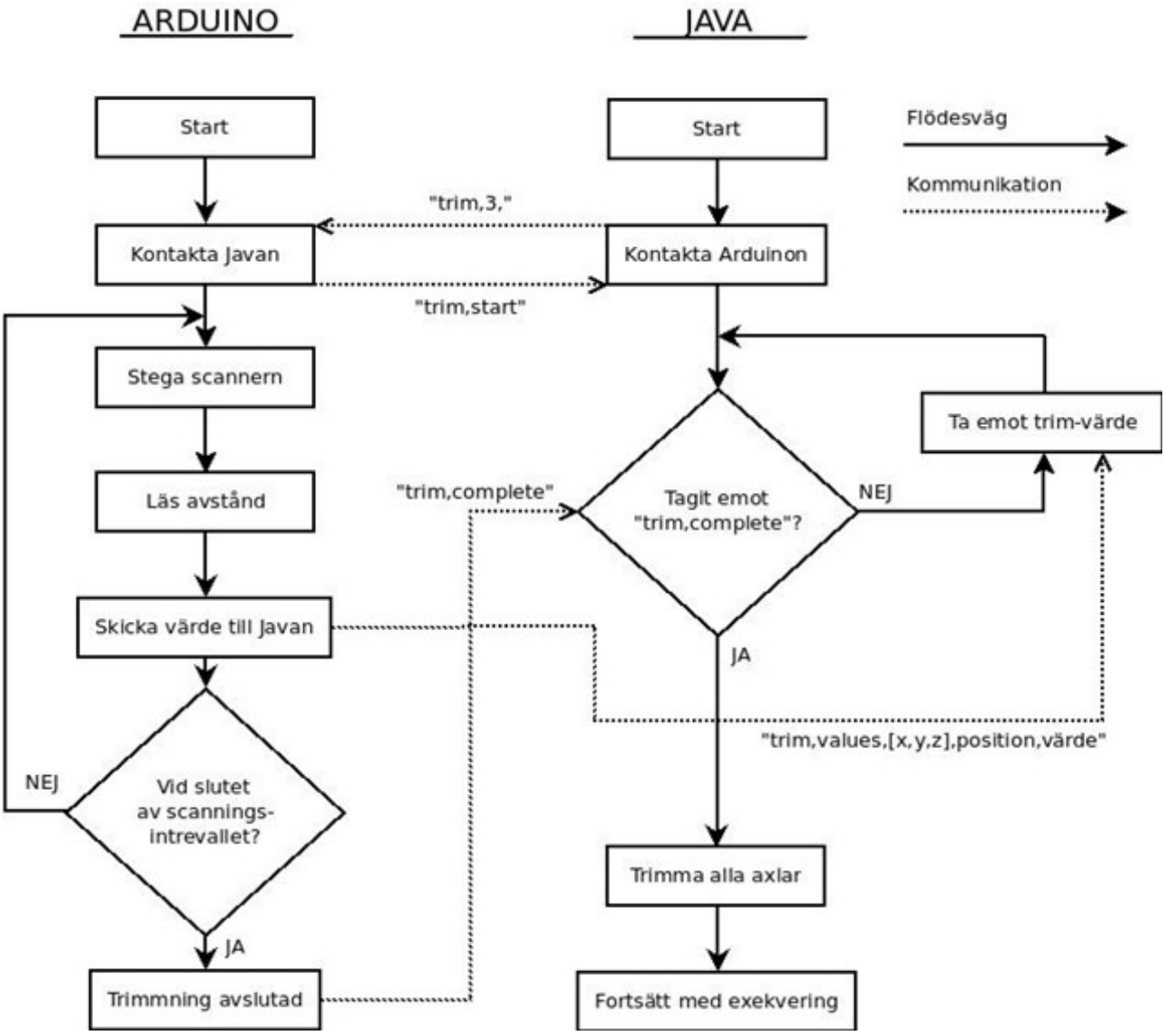
Denna funktion kan med fördel programmeras in i Arduino-miljöns *main*-fil innan setup-funktionen ens hinner kallas på, då blir den en del av alla skisser som programmeras med den utvecklingsmiljön. Det är en nödvändig livlina då alla Arduino-kort saknar en stop-knapp för exekvering av kod som ligger nedladdat på kortet som skulle möjliggöra ny nerladdning av ett säkrare program.

5.2 Trimningsfunktionen

För att få korrekta mätvärden används en trimningsfunktion, detta är en tomkörning av 3d-scannern som dels bestämmer max- och minvärden men den kan också använda de mätvärden den läser av för att skapa en kurva för vilket verkligt avstånd spänningen från avståndssensorn motsvarar.

Då olika sensorer inte alltid förmedlar samma spänning gentemot avstånd till mätobjektet finns behovet av att ha denna trimningsfunktion, detta möjliggör att byta sensorer utan att behöva ändra den befintliga koden.

Kommunikationen för trimningsfunktionen finns beskrivet i flödesdiagrammet(figur 5.2.a). De streckade linjerna representerar kommunikation mellan Arduinons program och Java-applikationen, de heldragna linjerna är flödesvägen.



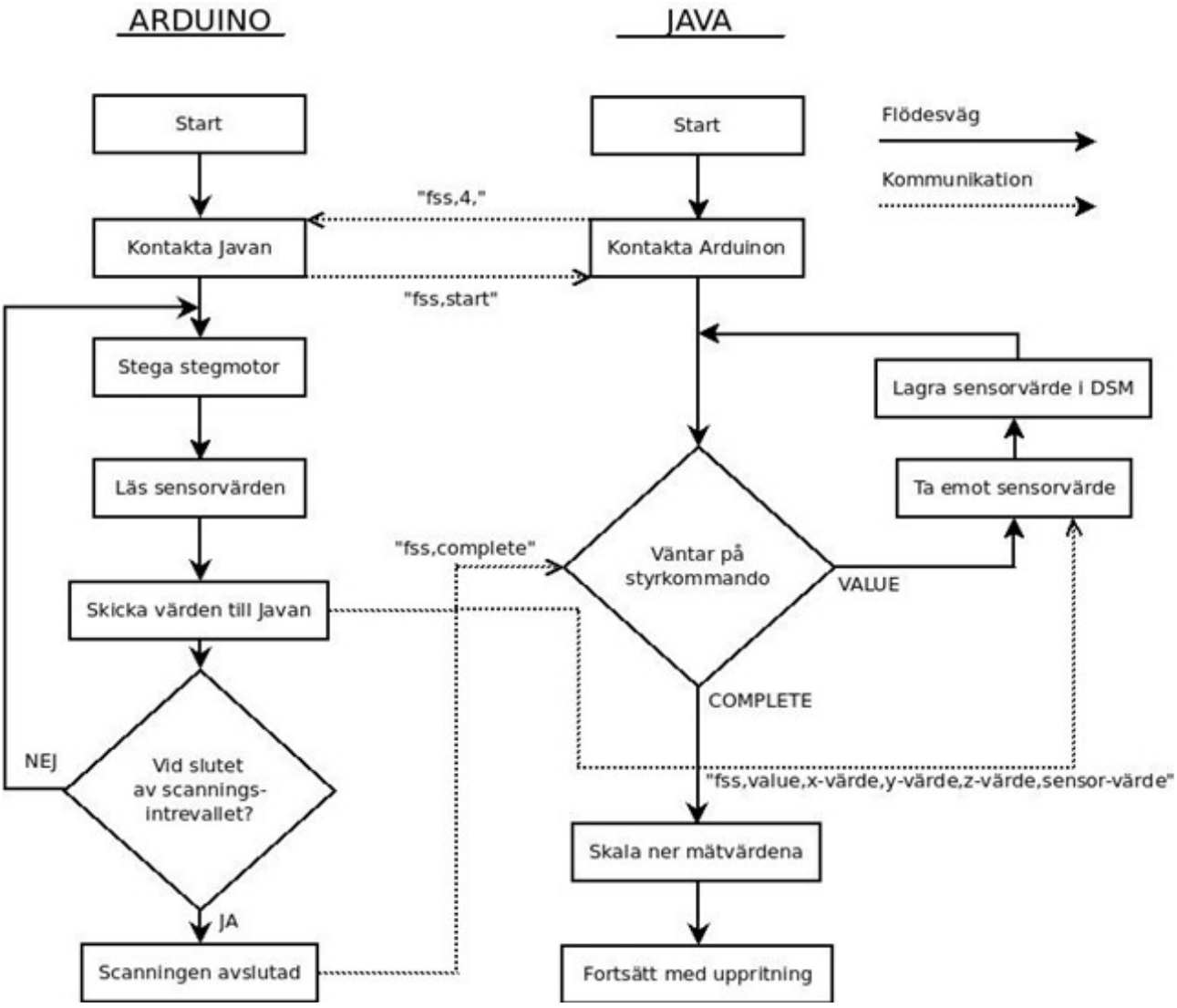
Figur 5.2.a: Flödesschema över trimningsfunktionen för Java-applikationen och Arduino-kortets exekvering och kommunikation.

Kontakten mellan de programmen initieras genom en handskakningsrutin där de båda verifierar varandras existens och bekräftar att den andre är redo att påbörja exekvering. Handskaksrutinen påbörjas med att Java-applikationen sänder “trim,3,” till Arduinon som svarar med “trim,start”, innan dessa meddelanden skickats och mottagits så pausas de bådas exekvering.

Därefter väntar Java-applikationen på att Arduinon skickar sondens position och vilket sensorvärde detta motsvara i ett decimalt tal med intervallet [0,1023]. Positionen anges av hur många steg stegmotorn tagit vilket ger en linjär kurva för avståndsmätning, anledningen till att sensorvärden används till avståndsmätning istället för stegräknaren är för att få mer tillförlitliga värden för sondens position.

När scannern nått slutet av sitt arbetsområde så skickar den “trim,complete” som meddelande till Java-applikationen och då avslutas väntan på trimningsvärden. Efter det startas funktionen för att skala ner och placera mätvärdena i dess korrekta intervall.

5.3 Scanningsfunktionen



Figur 5.3.a: Flödesschema över scanningsfunktionen för Java-applikationen och Arduino-kortets exekvering och kommunikation.

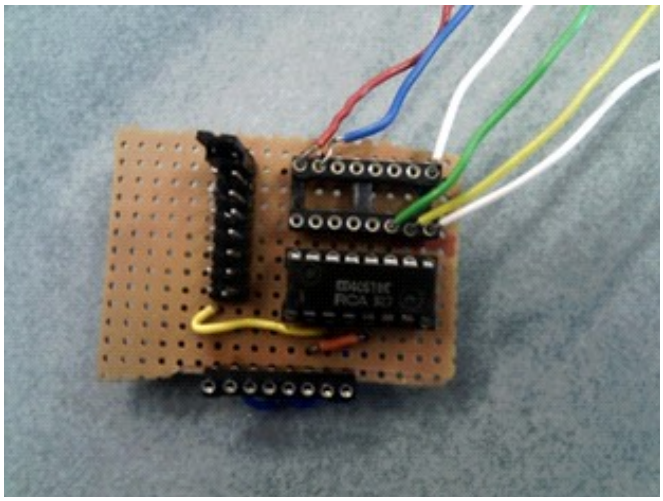
Scanningsfunktionen är mycket lik trimningsfunktionen, de båda börjar med en handskaksrutin för att etablera kontakt och bekräfta att båda programmen är synkroniserade. Arduinons program fungerar på samma sätt som i trimningsfunktionen, fast istället för att skicka stegantal och värde på avståndssensorn skickar den nu ett sensorvärde från sondens mätsensorer och x-, y-, z-koordinaterna för sondens position.

Java-applikationen väntar på ett meddelande från Arduinon, detta är antingen ett nytt mätvärde eller ett meddelande som bekräftar att scanningen är avslutad.
Efter att alla mätvärden matats in i DataSpaceManagern så skalar Java-applikationen ner alla värden för att sedan kunna ge en korrekt uppritning av 3d-miljön.

5.4 Multiplexenhet

För att kunna ansluta fler enheter, så som sensorer eller motorer, till Arduinon än vad det finns portar används multiplexrar. En multiplex har flera ingångar och en utgång, med styrsignaler kan en av insignalerna ledas om till utsignalen. En demultiplex gör motsatsen, den har en insignal som med styrsignalerna länkas till en av flera utsignaler.

Multiplexrar arbetar vanligtvis med digitala signaler men det finns analoga multiplexrar som kan leda om analoga signaler. En demultiplexer behövs för systemets utsignaler och en multiplexer för insignalerna. Efter som insignalerna är analoga krävs även att multiplexern är analog. Den analoga multiplexern som användes (74HCT4051) klarar av att både multiplexa och demultiplexa vilket betyder att multiplexenheten kan byggas med bara denna komponent.



Figur 5.4.a: En analog multiplexer med anslutningar.

Om 16 digitala ben från Arduinons reserveras för multiplexstyrsignaler kan en stor multiplexenhet användas med upp till 65536 kanaler. Då en enhet med den kapaciteten skulle behöva 9363 8-kanals multiplexrar valdes att bygga en mindre enhet för testning.[17]

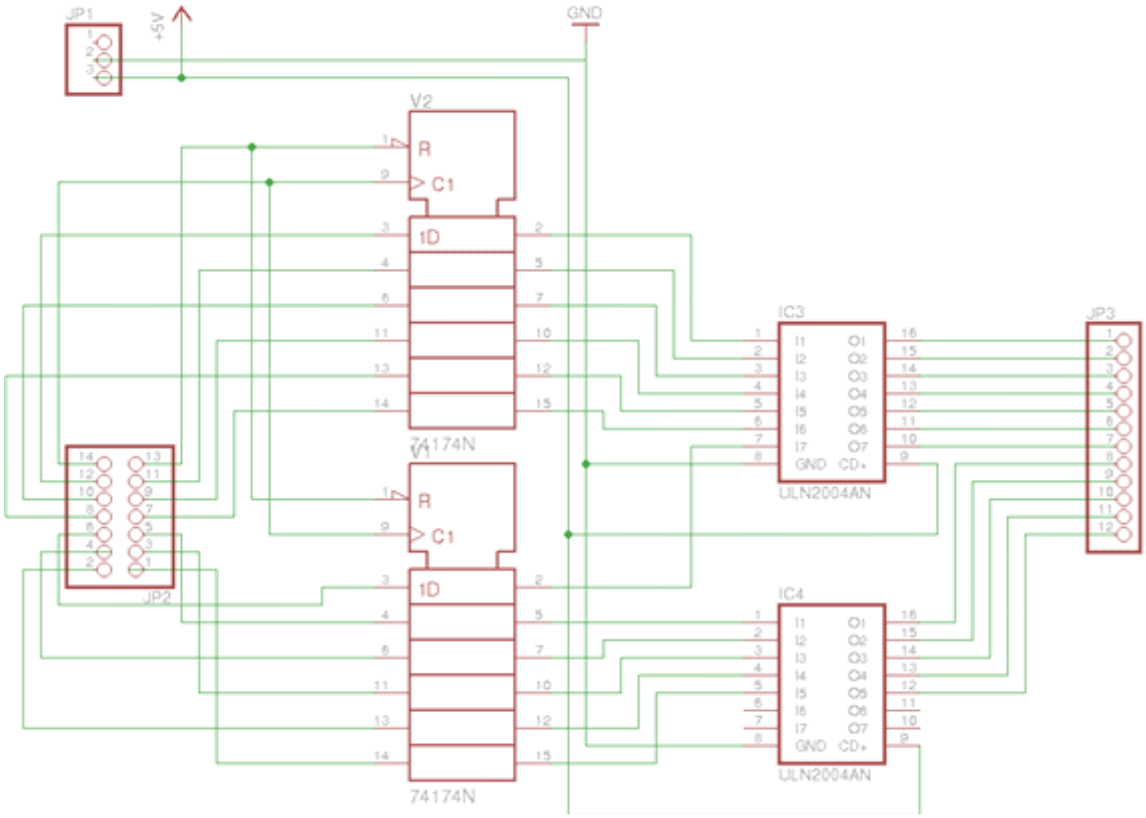
5.5 Styrkrets

En analog enhet som skall styras kräver en kontinuerlig styrsignal för att fungera. En stegmotor kräver att varje intern elektromagnet är energisatt i några millisekunder var för att den skall hinna ta ett steg. Utsignalen från multiplexenheten är bara kontinuerlig så länge rätt kanal är inställd. När enheten byter kanal kommer signalen brytas. För att mer än en enhet skall kunna styras i taget designades en styrkrets.

5.5.1 Styrminne

För att kretsen skall behålla sin styrsignal även när multiplexenheten är inställd på en annan kanal implementeras minne i kretsen i form av vippor. Kretsar med sex stycken d-vippor (74HCT174) används för detta. Kretsen har en insignal och en utsignal för varje vippa, en asynkron nollställning och en klocksignal. Utsignalerna ställer om sig till insignalernas värde då en positiv flank kommer på klocksignalen och etta ligger på nollställningen.

En styrkrets tar tolv styrsignaler vilket kräver två vippkretsar. Tolv styrsignaler skulle behöva tolv parallella kanaler i multiplexenheten. En 8-kanals enhet skulle då behöva tolv 8-kanals multiplexrar. Genom att låta klocksignalen till vipporna gå genom multiplexenheten istället behövs endast en enkel multiplexer för styrsignaler. Eftersom bara klocksignalen då når avsedd styrkrets är det bara den som uppdateras även då styrsignalerna är parallellkopplade mot alla styrkretsar.[18]

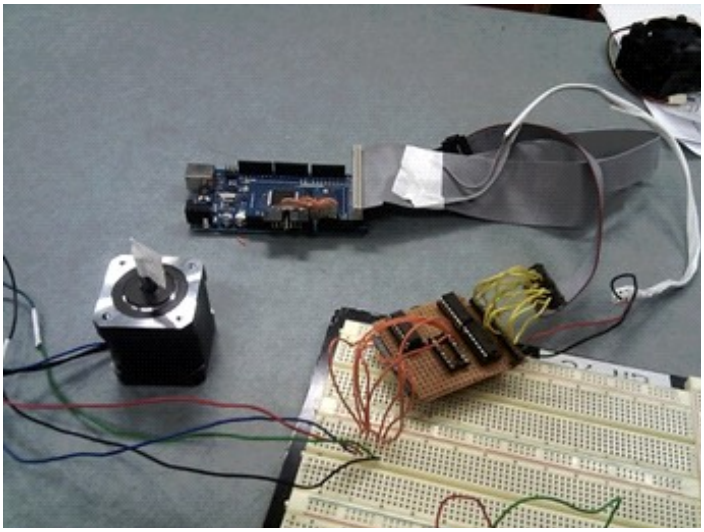


Figur 5.5.a: Kretsschema över styrkretsen med vippor och transistorförstärkare. JP1 är anslutning för jord och matningspänning (1: matningspänning till styrobjekt, 2: GND till Arduino, 3: 5V från Arduino). JP2 är anslutningar till Arduino (1-12: styrsignaler, 13: reset, 14: klocksignal). Klocksignalen behöver passera genom multiplexenheten innan styrkretsen. Styrobjektet ansluts mellan JP1:1 och JP3.

5.5.2 Förstärkning

HCT-kretsar arbetar med spänningar på 5V och låg ström. Detta är ofta inte tillräckligt för att driva en enhet så någon form av förstärkning krävs. Sjudubbla darlington transistorförstärkare (ULN2004A) användes för att uppnå detta. ULN2004A har en anslutning för varje bas och kollektor och en gemensam för samtliga emittrar.

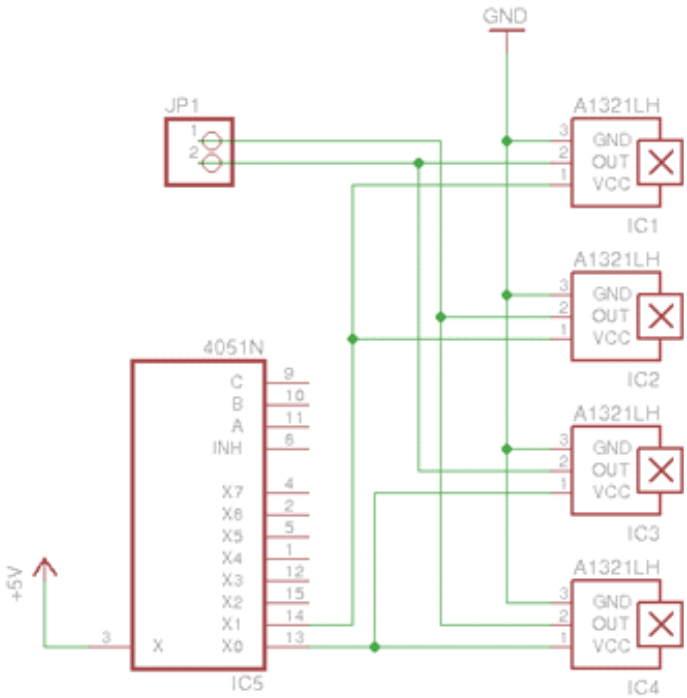
Styrsignalern ansluts efter vippan mot basen och emittern mot jord. Mot kollektorn ansluts styrobjektets jord och då kan en extern spänningskälla anslutas med positiv mot styrobjektet och negativ mot gemensam emitter.[19]



Figur 5.5.b: Styrkretsen med en stegmotor ansluten.

5.6 Implementering av sensorer

Sensorer används dels för att inhämta mätdata från mätobjektet och dels för att mäta var en sensor befinner sig efter förflyttning. De sensorer som använts under projektet har fungerat på liknande sätt vilket underlättar användandet av dem. De har en anslutning för 5V, en anslutning för jord och en anslutning för mätvärdet. När de är spänningssatta går mätvärdet att läsa av som ett spänningsvärde mellan 0V och 5V. För att kunna gruppera flera sensorer i en kanal från multiplexenheten kan 5V-anslutningen anslutas genom multiplexenheten och mätanslutningarna parallellkopplas.



Figur 5.6: Sensorer kan kopplas mot en multiplexer enligt figuren. När multiplexern är inställd på kanal X0 energisätts IC4 och IC2 och endast deras mätvärden avläses på JP1 även fast IC1 och IC3:s mätanslutningar också är kopplade mot JP1.

6 Resultat

Hur ett system för avläsning och visning av mätdata av storheter i ett tredimensionellt rum kan byggas beskrivs i detta kapitel. Systemet är uppbyggt modulärt och ändring av konstruktion, sensorer och visualiseringssätt kan ske med minsta möjliga modifieringar i övriga systemet.

6.1 Scannern

Scannern byggs upp av en central enhet med en *microcontroller* som styr kontroll, avläsning av data och kommunikation med applikationen. För att uppnå önskad expanderbarhet i systemet ansluts en multiplexenhet till centralenheten. Ett godtyckligt antal sonder kan då anslutas till centralenheten genom multiplexenheten. Då modularitet eftersträvas är systemet uppbyggt på ett sätt så att sondernas konstruktion inte är förbestämd, typer av sensorer lätt kan bytas ut och även olika sonder och sensortyper kan användas i samma system.

6.1.2 Centralenheten

Centralenheten består av en *microcontroller*, seriell anslutning mot Java-applikationen och anslutningar:

- MUX-kontroll: 16 pinnars anslutning för styrning av multiplexenheten, digitala utsignaler.
- Vippkontroll: 2 pinnars anslutning för kontroll av vippor. Nollställning och klocksignal, digitala utsignaler.
- Digitalstyrning: 12 pinnars anslutning för styrning av sonder, digitala utsignaler.
- Analogläsning: 12 pinnars anslutning för läsning av sensordata, analoga insignaler.
- Kraftanslutning: 3 pinnars anslutning för jord, 5V och 28V.

Microcontrollern tar emot styrkomandon från den seriella anslutningen och styr sonderna, inhämtar data från sondernas sensorer och skickar tillbaka det genom den seriella anslutningen.

6.1.3 Multiplexenhet

Multiplexenheten används för att inte antalet sonder skall vara begränsade till *microcontrollerns* utportar. Enheten använder multiplexrar och demultiplexrar som kan rikta en signal en vald väg. Demultiplexrar används för att styra utsignalar och analoga multiplexrar för insignaler från sensorer. Antalet kanaler en multiplex kan styra en signal är 2^n, där n är antalet styrsignaler. Centralenheten har en 16-pinnars anslutning för styrning av multiplexenheten vilket sätter en teoretisk gräns för antalet kanaler till 65536.

6.1.4 Sond

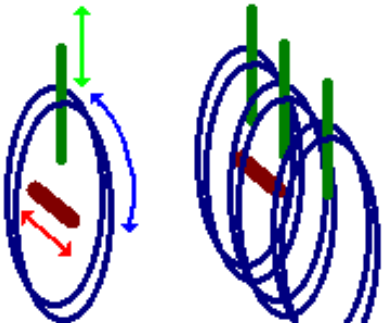
En sond är en gruppering av sensorer som flyttas tillsammans. En sond har upp till 12 sensorer och upp till 3 stegmotorer per multiplexkanal. En sond kan dock vara ansluten till flera kanaler från

multiplexenheten vilket möjliggör stor expanderbarhet. Systemets modulära uppbyggnad möjliggör flertalet konstruktioner av scannern.

En sond kan vara rörlig i ett led i riktning mot mätobjektet medans mätobjektet är rörligt i ett led parallellt mot sonderna samt kan roteras. Alternativt kan en sond vara rörlig mot mätobjektet samt i rotation kring det och mätobjektet rörligt parallellt mot sonderna.

Sonderna ansluts i serie mot centralenhetens digitalstyrning, analogläsning och kraftanslutning. Vippkontroll ansluts separat för varje multiplexkanal mot multiplexenheten.

Figur 6.1.4.a: Mätobjektet (rött) är rörligt i två led medans sonderna (grönt) är rörliga i ett led.



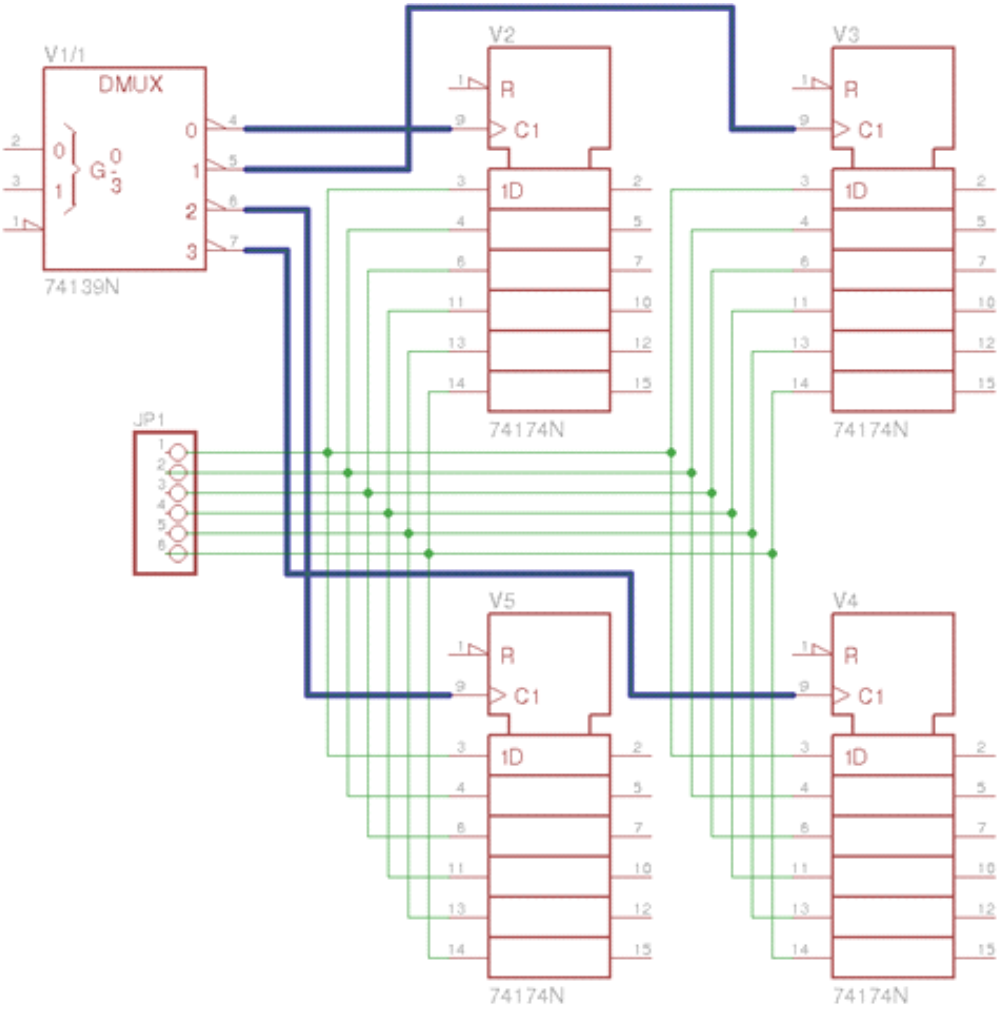
Figur 6.1.4.b: Mätobjektet (rött) är rörligt i ett led medans sonderna (grönt och blått) är rörliga i två led.

6.1.4.1 Styrkrets

För att kunna ansluta flera stegmotorer än vad centralenheten har anslutningar till används multiplexenheten. Multiplexenheten tillåter en signal att passera igenom vald kanal vidare till styrobjektet. När multiplexenheten omriktas mot annat styrobjekt bryts denna signal och styrningen av det första objektet går förlorat vilket innebär att endast ett objekt kan styras i taget. För att kunna styra flera objekt samtidigt används vippor som kommer ihåg en digital signal.

Styrförloppet blir att först ställs rätt kanal in i multiplexenheten och valda styrsignaler ges till digitala styrningsportarna. Sen skickas en exekveringssignal genom multiplexenheten till rätt styrkrets. Eftersom exekveringssignalen bara når den enheten som multiplexenheten är inställd mot så kommer bara vipporna i nämnd enhet uppdateras även fast de digitala styrsignalerna når samtliga enheter. Endast en signal behöver passera multiplexenheten med denna lösning vilket möjliggör en mindre multiplexenhet.

Då en stegmotor kräver mer effekt än vad den digitala kretsen arbetar med behövs förstärkning. En bipolär stegmotor kräver även att strömriktningen varieras. Därför har styrkretsen h-bryggor anslutna innan anslutningen mot stegmotorn.



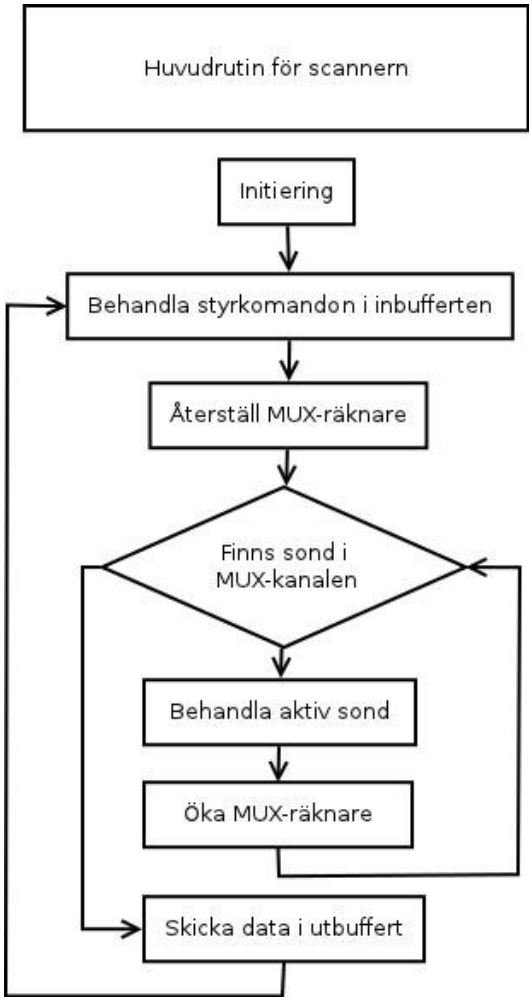
Figur 6.1.4.1.a: Klocksignalen ansluts till vipporna genom en demultiplexer.

6.1.5 Kod för styrning och avläsning av scanner

6.1.5.1 Probe

Probe-klassen representerar en sond eller en del av en sond på scannern. En sond på scannern är en samling av sensorer som tillsammans kan flyttas i ett eller fler led. *Microcontrollern* kommunicerar med sonderna genom multiplexenheten.

Varje instans av Probe-klassen representerar en kanal i multiplexenheten men en fysisk sond kan om fler sensorer är tillkopplade vara kopplad till flera kanaler i multiplexenheten. Probe-klassen har funktioner att komma åt dess sensor- och stegmotorklasser.



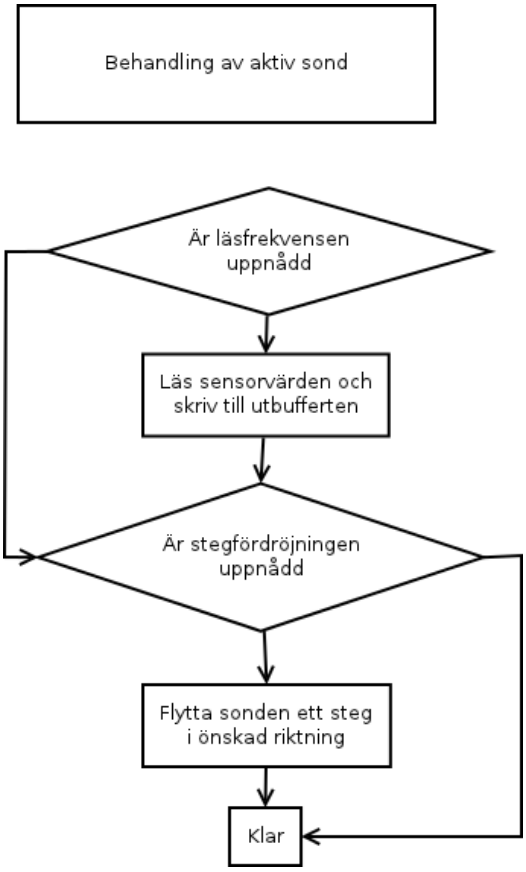
Figur 6.1.5.a: Flödesschema över centralenhetens huvudrutin.

6.1.5.2 Stepper

Stepper-klassen representerar en stegmotor och dess styrning. Den styr förflyttning av sonder. Stepper-klassen har funktioner för förflyttning ett steg framåt eller bakåt och förflyttning tills dess att ett sensorvärde uppnår önskat resultat, tänkt att användas tillsammans med en avståndssensor för att flytta sonden till ett önskat avstånd.

6.1.5.3 Sensor

Sensor-klassen representerar en individuell sensor på sonden. Klassen har funktioner för avläsning av aktuell sensor.



Figur 6.1.5.b: Flödesschema över behandling av aktiv sond från figur 6.1.5.a.

6.2 Java-applikationen

I följande kapitel beskrivs Java-applikationens uppbyggnad och dess funktionalitet. Huvudmålet i dess design är att den skall vara så modulär som möjligt, detta för att underlätta vidareutveckling samt utbytbarhet av sensorer och olika metoder som till exempel den för seriekommunikation.

ava-applikationen följer MVC-modellen där *model*-delen består av DataSpaceManagern och dess hjälpklasser, *viewen* består av klassen GLEngine med en klass för att skapa en applet och kontrollera Java3d's beteende samt en klass för mänsklig interaktion via mus och tangetbord, *controller*-delen består av kommunikationen med Arduinon och det som krävs för detta.

6.2.1 Hårdvarukommunikation

Hårdvarupaketet innehåller några användbara klasser för kommunikation med Arduinon. Klassen som används i resten av koden heter Communicator, den fungerar som en *buffer* mellan övrig kod i applikationen och den klass som verkligen sköter kommunikationen på detaljnivå. I UML-diagrammet(bilaga 1) är klassen kopplad till SerialCommunicator vilken kommunicerar över datorns serieport.

Klassen tar emot meddelanden från serieporten och placerar dessa i en *buffer* där ett mottaget meddelande kan hämtas med hjälp av en identifierare. Meddelanden kan även skickas över serieporten ifrån denna klass.

SerialCommunicator innehåller även klassen ShutdownHook vars kod körs när applikationen avslutas genom att lägga en lyssnare på eventet som kallas när ett Java-program avslutas. I detta fall stänger klassen serieporten, utan att stänga kopplingen till denna kan åtkomst för serieporten blockeras.

För att *debugga* utan en fysisk koppling kan man koppla in SerialCommunicatorHook som lägger sig som ett mellanskikt och skapar falska inmeddelanden som tas ifrån systemets inström.

Klassen NativeCommunicator sköter kommunikation med C-kod via JNI. Denna klass kan ersätta SerialCommunicator för att ändra kommunikationsvägen.
Man kan även skapa en instans av klassen ArduinoPort som möjliggör enskild styrning av ett enskilt ben på Arduinokortet, detta möjliggör att applikationen kan detaljstyra kortets beteende om behovet för detta skulle uppstå.

ServoEngine och SensorReader är specialfall som använder sig av ArduinoPort-klassen, dessa är utformade så att den ena skickar värden till ett ben och den andra läser av värdet på ett ben.
Ett objekt av klassen Modifier kan under körning ändra de objekt som redan placerats i DataSpaceManagern, vilka också ärvt gränssnittet IModifiable. Detta för att möjliggöra dynamisk uppritning av sensorer och positioner.

Klassen FullSensorScan utför en komplett scanning av ett mätobjekt.

6.2.2 Mätvärdeshantering

Alla inscannade mätvärden lagras i DataSpaceManagern som DataPositioner, dessa innehåller en tredimensionell position och sensorns avmätta värde. De inblandade klasserna finns även beskrivna i bilaga 2.

De tredimensionella positionerna och sensorernas värde kan skalas för att passa inom bestämda intervall med funktionerna trimScales och trimSensorValues, detta för att anpassa värdena till det intervall som den specifika uppritningen kräver.

6.2.2.1 Skalningsfunktionen

Skalningsfunktionen för axlarna fungerar enligt:

1. Placera kurvan centrerat på medianen för max- och minvärdena för intervallet.
2. Skala ner kurvan genom att dela den med en delningsfaktor

För att placera kurvan på medianen skapar man en *offset* genom att ta negernigen av det minsta värdet och dra ifrån hälften av skillnaden mellan max- och minvärdet. Sedan räknar man ut

delningsfaktorn genom att ta skillnaden mellan max- och minvärdet och dela denna med skillnaden mellan det önskade max- och minvärdet. Efter detta stegar man igenom alla DataPositioner och sätter ett nytt värde, detta värde är summan av det gamla värdet och *offseten* delat med delningsfaktorn.

6.2.3 Grafikuppritning

Klassen GLEngine är grafikuppritningens själva hjärta, i denna klass styrs de grafiska funktioner som används. Den kan rita upp tre specifika saker:

- **DataSpaceManager**:
  Alla DataPositions ritas upp på dess korrekta position och vars utseende kan anpassas beroende på vilket intervall värdena ligger i och vilken sorts sensorer som använts.
- **Axlar**:
  För att indikera vilken rotation uppritningen har kan man aktivera uppritning av x-, y- och z-axlarna. Dessa ritas upp i tre olika färger för att kunna särskiljas.
- **Rutnät**:
  Vid varje axel ritas det upp ett gråfärgat rutnät för att indikera storlek på zoomningen.

Canvas-klassen innehåller det objekt som allt ritas upp i, det är även en extension av klassen Applet som i Java skapar ett enkelt fönster. På en Applet kan man lägga lyssnare för tangetbordstryck och musrörelser, lyssnaren till dessa är ett Controls-objekt.

Controls-klassen styr vilka funktioner som kallas vid händelser från mus och tangentbord, exempel på dessa är till exempel rotation och zoomning.

6.2.4 Felmeddelanden

För att enklare kunna *debugga* när fel uppstår har 5 *exceptions* skapats, dessa beskrivs nedan och i bilaga 4:

- **NoMessageInBuffer**:
  Händer när en process väntar på ett meddelande med sin specifika *identifier*, om inte detta meddelande kommer inom den tillåtna tidsramen så kallas felmeddelandet.
- **NoValueReceivedException**:
  Klassen som får ett NoMessageInBuffer-felmeddelande skapar detta för att indikera att den inte kunnat ta emot sitt sensorvärde.
- **ReadingOnWriteOnly**:
  Dette felmeddelande skapas när ett objekt av typen ArduinoPort försöker läsa på ett Arduino-ben efter att definierats för skrivning.
- **ScanCouldNotFinish**:
  Händer när klassen FullSystemScan inte får slutföra sin funktion för scanning av objekt.

- **WritingOnReadOnly**:
  Detta felmeddelande skapas när ett objekt av typen ArduinoPort försöker skriva på ett Arduino-ben efter att definierats för läsning.

6.2.5 Gränssnitt

För att lättare hantera uppbyggnaden av programstrukturen skapades olika *interface*, detta gör man för att få en förberedd mall att jobba efter när man skapar nya klasser. De nuvarande *interface*n är de mest grundläggande byggstenar som behövs för att utveckla ny funktionalitet. *Interface*n som finns beskrivs nedan och i bilaga 5:

- **IArduinoPort**:
  Detta *interface* är skapat för att hantera direktstyrning av ett Arduino-ben. Man kan välja vilket ben det gäller och definiera om detta skall skrivas till eller läsas från.
- **ICommunicator**:
  Under kommunikation med Arduinokortet finns det två metoder, antingen att skicka ett meddelande eller att ta emot ett meddelande som skickat med en speciell identifierare.
- **IModifiable**:
  Om ett objekt är modifierbart så måste det innehålla ett värde som kan ändras, i detta fall i form av datatypen *float.*
- **IModifier**:
  Om ett objekt är en modifierare så måste man kunna hämta ett värde från denna som placeras i det modifierbara objektet. Man kan lägga till modifierare som vars värde skall uppdateras när funktionen updateTargets kallas på.

Dessa interface är egentligen bara tomma skal som inte själva innehåller någon logik, med hjälp av en abstrakt klass kan man skapa ett skal som även har inbyggd logik.

7 Slutsats

För att enkelt kunna avläsa och visualisera olika mätbara storheter i ett tredimensionellt rum krävs ett modulärt system med sensorer. Projektet har undersökt tekniken som krävs för att bygga ett sådant system och fokuserar på moduläritet och expanderbarhet.
Under projektet har inget färdigt system sammanställts som kan inhämta mätvärden från ett tredimensionellt rum utan en funktionsmodell där modulerna för ett sådant system separat byggts upp och testats.

7.1 Hårdvara

Arbetet med elektronik har tagit mer tid än väntat och det är i efterhand uppenbart att mer erfarenhet av arbete men elektroniska kretsar hade underlättat arbetet.
Enskilda enheter har konstruerats och testats men på grund av tidsbrist har inte alla enheter testats tillsammans. Även om inga tredimensionella mätningar har genomförts så är de tekniker som krävs undersökta och det största hindret för att fortsätta är den fysiska konstruktionen.

7.2 Mjukvara

En applikation har utvecklats för att hantera och visa upp de mätvärden som scannats av sensorerna. Det finns möjlighet att lägga till uppritningsalternativ för nya sensortyper utan att behöva omstrukturera befintlig kod.

Denna applikation är i ett fungerande stadiet även om vidareutveckling krävs, för att göra applikationen fullständigt fungerande behövs mer konfigurationsmöjligheter och ett användargränssnitt. Undersökande av olika tekniker och problem med kommunikationen gjorde att utvecklandet av applikationen blev fördröjt.

7.3 Kommunikation

Det finns ett färdigutvecklat system för kommunikation mellan Java-applikationen och Arduino-kortet. Detta skulle kunna förbättras med en mer stabil lösning i Arduino-kortet som har samma nivå av felhantering som Java-applikationen.

Möjligheter finns att byta ut kommunikationsmetoden mot stabilare trådbundna system som *USB* eller trådlösa som till exempel *MiWi*.

7.4 Kritisk diskussion

Under projektets gång har det stötts på både mot- och medgångar, många av de motgångar som försenat arbetet hade kunnat undvikas med bättre buffer i tidsplaneringen.

Då arbetet skett under en snäv budget så har alternativa lösningar till dyra sensorer undersökts. Några av dessa experiment har varit lärdomsfulla men inte resulterat i en hållbar lösning. Även en del av de elektriska komponenter som utvecklats under projektets gång finns att köpa till ett relativ billigt pris, dessa hade kunnat täckas av budgeten. Under efterforskningsstadiet vid projektstarten missades att fastställa behovet av vissa komponenter, så som dc-omvandlare för seriekommunikation eller multiplexenheten till Arduinon. Sedan vid påbörjandet av utvecklingsfasen kom behovet av dessa komponenter upp, istället för att då göra djupare efterforskningar om var dessa komponenter kunde införskaffas så beslutades det att helt enkelt konstruera dessa själva vilket ofta var mycket mer tidsödande än planerat. Alla dessa misstag och begränsningar har bidragit till att dyrbar projekttid gått förlorad, men också att värdefull kunskap anskaffats.

Flera olika tekniker för tredimisionell uppritning undersöktes. Mycket tid spenderades med att utvärdera de olika teknikernas för- och nackdelar. Efter att Java3D valdes så gick arbetet med visualiseringen mycket fortare.

Java-applikationen har utvecklats för att vara en stabil grund för implementering av fler sensortyper och drivmedel, mycket fokus lades på att skapa en miljö som var lättanpassad och flexibel för nya tankar kring hur den fysiska enheten sedan byggs upp. Den saknar ett användargränssnitt för kontroll av funktionaliteten i nuvarande läge så sköts styrning och bearbetning med hjälp av textinmatning i utvecklingsmiljön.

7.5 Rekommendationer för fortsatt arbete

Man bör se över designen av hela systemet innan vidare utveckling påbörjas, detta för att säkra att alla delar får en fungerande kommunikation mellan varandra.

7.5.1 Konstruktion

En grundlig undersökning av möjliga konstruktioner och material krävs innan en färdig prototyp kan byggas. En konstruktion måste vara stabil nog för att kunna flytta flertalet sonder till godtyckliga positioner samtidigt har olika mätbara fysikaliska storheter olika krav. Vid mätningar av elektromagnetiska fält bör metall undvikas då det kan påverka mätresultat och om temperatur skall mätas krävs ett material som klarar av de temperaturskillnader som kan uppstå.

Olika material att undersöka är metaller, plast och kolfiber, men även trä kan vara intressant. En

lösning kan vara att ha en yttre struktur med mekanik som sonder i varierande material för olika mätningar kan anslutas på.

7.5.2 Hårdvara

Systemets kretsar behöver tillverkas i större skala så att de kan testas ordentligt. Fler olika sensortyper behöver testas. Framför allt för insamling av mätdata men även för positionering. Valet av microcontroller kan ses över och kommunikationen med applikationen borde uppgraderas till USB-standard eller trådlöst som till exempel MiWi.

7.5.3 Mjukvara

För att ge en normal användare kontrollen att kunna hantera programmet bör ett lättanvändligt och väldesignat användargränssnitt implementeras.

Även fler konfigurationsmöjligheter bör implementeras, detta för att ge användaren fria händer att bestämma hur mätinstrumentet ska se ut och vilka sensorer detta ska innehålla. Med tanke på att olika sensorer har olika känslighet så borde möjligheten att ställa in mätavståndet implementeras.

Möjlighet att spara och ladda sina mätvärden bör utvecklas, dels för att kunna dokumentera gjorda mätningar men även för att kunna jämföra olika resultat när till exempel en produkt har ändrat sin utformning.

7.5.4 Fortsatta projekt

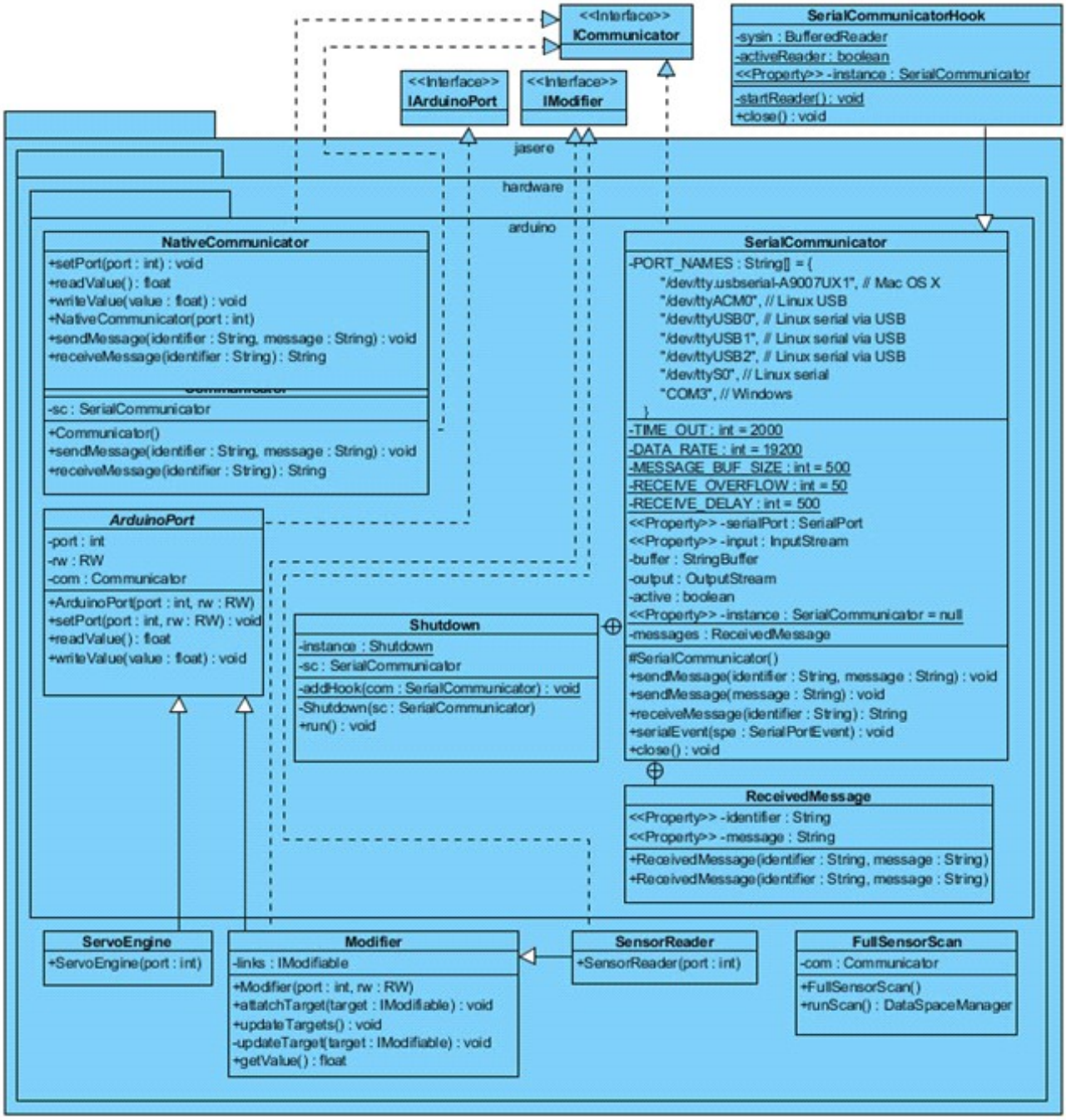
För att få en bra överblick över hur ett fält runt ett objekt ser ut så borde själva objektet ritas ut. Detta skulle kunna göras genom att scanna av formen på objektet samtidigt som fältvärdena mäts. Alternativt med en teknik som mäter av objektet med en laser och skannar av dess form i tredimensionella punkter vilka kan sammanfogas med mätvärdena från objektet.

Med uppnåd funktionalitet bör det gå att bygga en fungerande prototyp som med ett fungerande användargränssnitt skulle kunna användas till att fullständigt skanna av mätobjekt och representera detta i ett mänskligt överskådligt spektra. Med de rekomendationer som redan tagits upp bör detta bli en välfungerande tredimensionell sensor scanner.

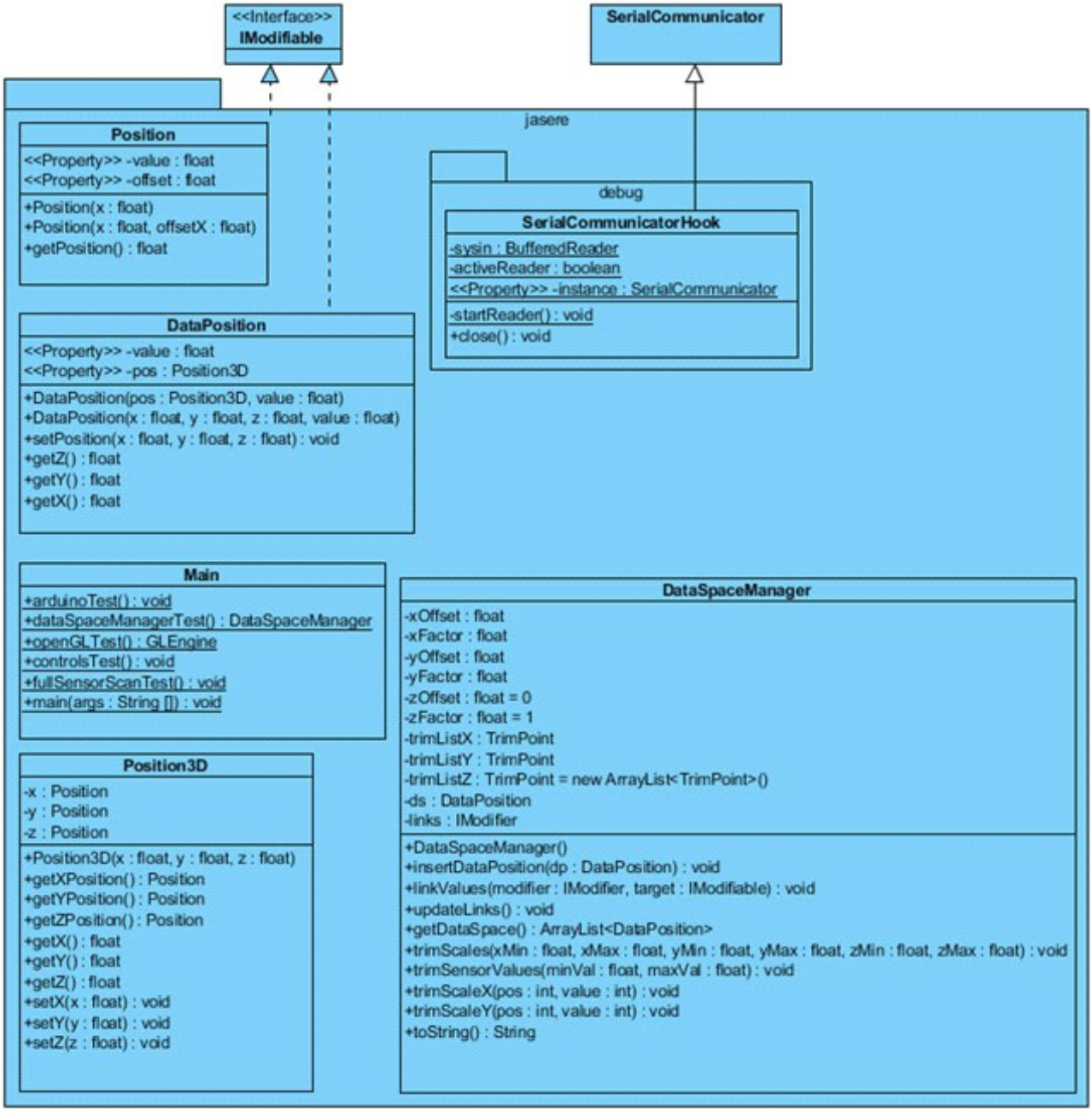
Referenser

1) http://arduino.cc
2) ATMEL (2011). “ATmega640/1280/1281/2560/2561 Preliminary Summary”.
3) http://wiring.org.co
4) http://arduino.cc
5) Ed Ramsden (2006). “Hall-effect sensors: theory and applications”.
6) http://www.opengl.org
7) http://www.opengl.org/wiki/Rendering_Pipeline_Overview
8) http://www.oracle.com/technetwork/java/javase/tech/index-jsp-138252.html
9) http://www.rejas.se/fritis/datorkommunikation/chap_serpar.html
10) http://www.camiresearch.com/Data_Com_Basics/RS232_standard.html
11) http://www.lammertbies.nl/comm/info/RS-232_specs.html
12) UTGIVARE (ÅR). “TITEL” datablad för MAX220-249
13) http://www.greenleafsoft.com/AboutDataRates.asp
14) http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2113&param=en520414
15) http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2664&param=en535767
16) Al Williams (2002). “Microcontroller projects using the Basic Stamp (2nd ed.). Focal Press. p. 344. ISBN 9781578201013”.
17) NXP (2011). “74HC4051; 74HCT4051 8-channel analog multiplexer/demultiplexer Product data sheet”.
18) Philips (1998). “Data sheet 74HC/74HCT174 Hex D-type flip-flop with reset; positive-edge trigger”.
19) Texas Instruments (2003). “ULN2001A, ULN2002A, ULN2003A, ULQ2003A, ULQ2004A High-voltage high-current darlington transistor array”.
20) http://jaromir-sysel.blogspot.com/2010/12/kzizeny-seriovy-kabel-crossover-cable.html

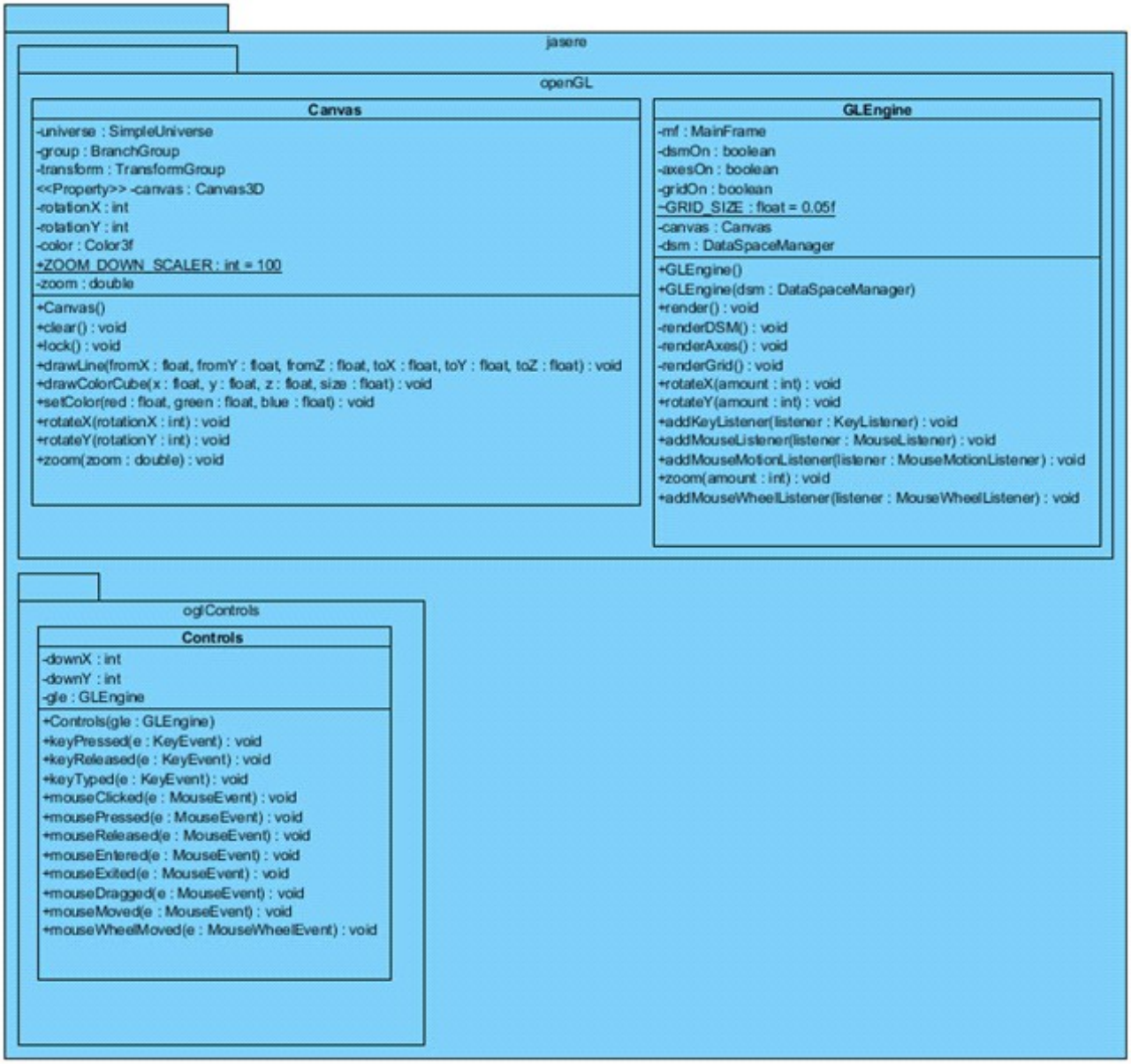
Bilaga 1: UML-diagram över hårdvaru-paketet



Bilaga 2: UML-diagram över mätvärdeshantering



Bilaga 3: UML-diagram över grafikuppritning



Bilaga 4: UML-diagram över gränssnitt