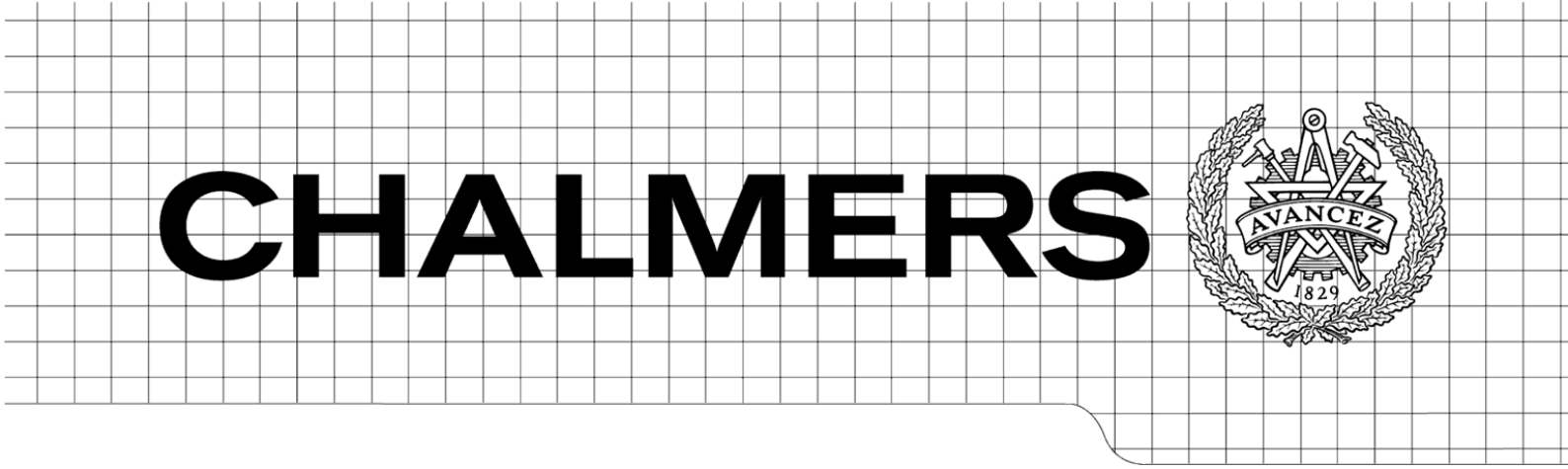


Multirörelseplattform

Med sensor

BJÖRN JOHANSSON

HJALMAR SCHÖRLING

Examensarbete

Högskoleingenjörsprogrammet för datateknik

CHALMERS TEKNISKA HÖGSKOLA
Institutionen för data- och informationsteknik
Göteborg 2012

Sammanfattning

Denna rapport beskriver metoden för konstruktion, styrning och utveckling av en prototyp av multirörelseplattform med sensor, konstruerad för att avläsa mätdata kring tre-dimensionella objekt. Rapporten beskriver såväl fungerande metoder, som mindre bra tillvägagångssätt samt dess för- och nackdelar. Metoden använder sig av en modifierad robotarm av modellen AL5A, microcontrollern Arduino Uno, proto shield R3 samt kod och kopplingar för styrning av processen. Kommunikationen sker via en seriell port i datorn där operativsystemet Windows 7 har använts för testning. Projektet har sträckt sig över en 10-veckors period och prototypen är den slutprodukt som tagits fram efter denna tid. Mycket av studien har fokuserat på praktiskt utförande, men är även till viss del en litteraturstudie. Den genomförda undersökningen har visat att det finns många olika tillvägagångssätt för att konstruera denna form av prototyp. Eftersträvar man en modell med mycket hög precision är det bäst att använda sig av så många rörliga axlar som möjligt, men för detta krävs även en stor matematisk förståelse samt större budget. Genom att använda sig av färre motorer får man en simplare och inte lika precis prototyp, även om denna fortfarande är kapabel till att uppfylla önskad funktionalitet. Slutprodukten som beskrivs i denna rapport har en felmarginal på ca 2% per motor, men även lösningar på detta problem beskrivs.

Abstract

This paper describes the method to construct, control and develop a multifunctional movement platform with sensor, made for scanning measured data around three-dimensional objects. The paper describes working methods as well as approaches that did not work out as we expected and their pros and cons. The process uses an AL5A robotic arm, the micro-controller Arduino Uno connected with a proto shield R3, along with wiring and code for steering the arm. The communication goes through the serial port of the computer and the operating system used for testing is Windows 7. The project lasted over a 10-week period and the prototype presented is the final product over this time. The results that we have got during this period have shown that there are many different ways to handle this problem, dependent of what result the project aim for. The most accurate way is to use as many motors as possible for the construction, though this method demands high knowledge in trigonometric as well as money to invest into the project. Similar result could be accomplished by using fewer motors, though the platform will not be as accurate as a more advanced construction, but for a cheaper price and easier development. The final prototype described in this paper has a margin of error of about 2% per motor, but solutions to solve this problem are also presented.

This report is written in Swedish.

Förord

Denna rapport är resultatet av vårat examensarbete på högskoleprogrammet för datateknik 180hp vid Chalmers Lindholmen. Arbetet utfördes på plats på Campus Lindholmen.

Vill rikta ett stort tack till Sakib Sistek för ett givande och roligt samarbete, där han har lagt stor tid på att hjälpa oss och svara på alla frågor vi haft, samtidigt som vi lärt oss mycket på ett väldigt brett område. Genom att testa väldigt många metoder och tillvägagångssätt under utvecklingen har vi både lärt oss mycket om hur olika saker fungerar, samt tagit erfarenhet till kommande projekt om att det finns många olika fallgropar, och att det är viktigt att se över hela bilden innan man sätter igång med arbetet.

Vi vill även tacka Raimond Emanuelsson på matematiska institutionen för sitt bidragande till våra uträkningar, samt personerna i de andra grupper som vi samarbetat med, som har gett oss många bra idéer och råd samt gjort tiden under projektet betydligt mer underhållande.

Göteborg 6 juni 2012

Hjalmar Schörling Björn Johansson

Innehållsförteckning

1. Inledning

1.1 Bakgrund

I dagsläget finns tekniken för att scanna in en 3D-bild av ett objekt och få ut dess virtuella x-,y- och z-koordinater. Problemet ligger dock i att överföra dessa koordinater till verkliga situationer så att en multirörelseplattform på så sätt kan träffa valfri punk kring objektet, där en godtycklig sensor på toppen av armen sedan ska kunna läsa av ett valfritt värde på objektet och kunna sända tillbaka dessa mätdata till användaren. Denna utveckling har stor potential inom flera områden. Har man exempelvis en stabil fungerande teknik för detta skulle man kunna tänkas använda det vid resor ut i rymden där man vill scanna av okända objekt och inhämta fakta om dem. Även här på jorden kan man tänkas utnyttja det för att tex mäta strålning i radioaktiva områden, gifthalter hos växter och djur eller för simplare funktioner som att mäta magnetfältet och värmen kring exempelvis ett kretskort. Detta examensarbete behandlar utvecklingen av denna typ av multirörelseplattform i syfte att skapa en fungerande prototyp av produkten.

1.2 Problemställning

Enligt uppdragsgivare[1] finns ingen färdig teknik för att mäta data från objekt endast kända från en 3D-bild. Problemställningen ligger således i att utveckla en prototyp av en robotarm som med givna koordinater för ett objekt skall kunna scanna och skicka mätdata kring hela objektets kropp.

1.3 Syfte

Syftet med denna rapport är att beskriva utvecklingen av en multirörelseplattform för avläsning av fält kring tredimensionella objekt. Rapporten beskriver hela projektets utveckling som tagit form i både en litterär studie och praktiska experiment av 3D-scannern och dess funktioner, hur den via programmet Meshlab får ett objekts virtuella koordinater samt slutligen konstruktionen av en prototyp av en multirörelseplattform som automatiskt kan besöka alla punkter på det tidigare scannade objektet. Rapporten tar även upp hur man går till väga för att skicka tre utvalda punkter på ett objekt så att ett mjukvaruprogram ska kunna kalibreras. Utvecklingen av styrningen kommer att ske genom ren c-kodning i programmet Arduino. Multirörelseplattformen i fråga kommer att vara anpassningsbar och således kunna använda sig av valfri sensor för olika mätvärden.

1.4 Avgränsning

Studien kommer att avgränsas till att enbart konstruera en prototyp och således inte en version som kommer tillämpas i verklig produktion.
Projektet lägger inte stor vikt vid val av operativsystem, utan fokus ligger vid val av hårdvara. Alla beskrivna koordinatsystem utgår från att x-ledet motsvarar djupet, y-ledet höjden och z-ledet motsvarar bredden i systemet.

[1] Sakib Sistek, handledare vid Chalmers tekniska högskola. Ansvarig för ett bestämt antal exjobb.

1.5 Disposition

I denna rapport presenterar vi i kapitel 3 den tekniska bakgrunden för de delar vi har använt oss av. Kapitlet börjar med att gå igenom alla hårdvarumässiga enheter, för att sedan beskriva mjukvaran till dem. Slutligen avslutar vi det med säkerhet som man bör tänka på under användandet av dessa.

Kapitel 4 är en kronologisk beskrivning av det arbete vi utfört under detta 10 veckor långa examensarbete, med såväl fungerande, som mindre bra metoder, och är således ingen optimering för utförande av arbetet. Del 4.2 beskriver beräkningar som krävdes för att komma fram till slutprodukten och den sista delen i kapitlet tar upp delar av kodningen som vi anser att läsaren kan ha nytta av att se för en bättre förståelse.

I kapitel 5 och 6 ses våra resultat och slutsatser som vi kommit fram till under projektets gång. Allra sist finns bifogat appendix med foton tagna under processen samt kod för styrning av plattformen.

2. Metod

Arbetet kommer att baseras dels på litteraturstudie, men främst genom praktiskt studerande av 3D scanner, multirörelseplattform och programmet Arduino, baserat på språket C. Studien kommer att ta form genom samarbete med 6 andra examensarbetsgrupper och innefattar mycket utbyte av kunskap och metoder.

Anledningen till att vi väljer att arbeta i programmet Arduino beror på att detta är en standardisering till mikroprocessorerna i fråga. Att språket sker i förenklat C istället för ren C beror på att Arduino har funktioner som fyllde våra behov och förenklar processen. Vi kommer att studera 3D-scannern på en basnivå för att få en insikt i hur objekten vi arbetar kring kommer att se ut i digitalt format och för att enklast komma fram till, i samarbete med dom andra grupperna, vilka filer som blir lättast och smidigast att läsa/skriva från/till.

För att uppnå målet krävs noggranna studier av C-språket, vilket hela studien är uppbyggd kring. Vi behöver även inhämta grundlig information om robotars rörelsemönster och positionering, och för detta krävs trigonometriska beräkningar av hög grad. För att koppla samman alla elektriska komponenter krävs viss forskning kring de olika komponenternas uppbyggnad.

3. Teknisk Bakgrund

I detta kapitel presenterar vi den tekniska bakgrunden som krävdes för att genomföra projektet. Vi går först igenom ren hårdvara och övergår sedan till mjukvaran.

3.1 Systemöversikt

Vi har under utvecklingen av vår prototyp använt oss av en mängd olika program och hårdvaror. Vår slutprodukt använder sig av Arduino som består av såväl hård- som mjukvara, vilket kommer beskrivas utförligare i kommande sidor. Robotarmen vi använt oss av är en modifierad version av AL5A robotic kit och alla kopplingar har gjorts på en proto shield R3, samt en extern breadboard. Vi har använt oss av Operativsystemet Windows 7, men projektet är ej låst till detta.

3.2 Hårdvara

3.2.1 Arduino

Arduino är en mikrokontroller byggd på ett enda kretskort och används för att styra och förenkla multipla funktioner i elektronikkretsar. Kortet består av ett antal in- och ut-portar för att ta emot signaler. Den ansluts till en dator genom USB anslutning och får även sin ström därifrån. En extra strömingång finns även i de fall då kretsen kräver extra mycket styrka. Det finns flera olika typer av kretskort, men den vi använder oss av i detta projekt är Arduino UNO, som skiljer sig från tidigare kort i serien genom att den inte använder FTDI USB-to-serial driver chip. Istället innehåller den Atmega16U2 som är programmerad som USB till serie konverterare. Version två av UNO har en resistor som kopplar SU2 HWB ledningen till jord, vilket förenklar att sätta den i DFU-mode. I den tredje och senaste versionen finns SDA och SCL nålar nära AREF-nålen samt två IOREF placerade vid reset-knappen, vilket stärker hela återställningskretsen. UNO är det nyaste USB-kretskortet från Arduino, men därmed inte det mest avancerade. För de kretsar vi skulle bygga räckte dock UNO och således föll valet på denna modell.[2]

Tabell 1: Specifikationer på Arduino UNO.

Arduino	Processor	USB Interface Typ	Dimension mm
UNO	ATmega328P	ATmega8U2, ATmega16U2	68.6 mm × 53.3 mm

[3]

3.2.2 3D Scanner

Denna scanner används av en annan grupp för att scanna in ett objekt till en tredimensionell bild i datorn. Genom bilden som skapas kan man sedan få ut

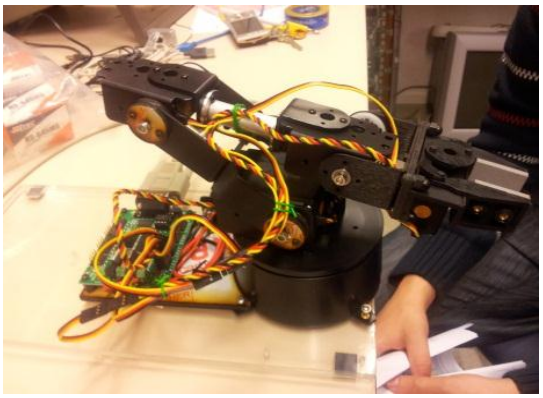
[2] Websida 2
[3] Websida 6

virtuella koordinaterna för objektet och spara detta till en mängd olika filtyper. I vårt fall har vi använt oss av .stl-filer. Att vi valt att använda oss av just stl-filer beror på att det är denna typ av filer som en 3D-printer använder sig av, och vi visste således att detta var en fungerande metod att arbeta efter. Senare så används då vissa punkter från den bilden av robotarmen för att kunna läsa av sensorvärden från den exakta platsen.

3.2.3 Robotarm

AL5A Robotic Arm ger noggranna och snabba rörelser som kan upprepas för att nå punkter. Den har fem stycken olika motorer så att axlarna kan justeras: Basplatta med rotation på 180grader (HS-475HB), axel (HS-805BB), armbåge (HS-755HB), handled (HS-645MG) och en nypa som kan ta tag i objekt (HS-422). Alla axlarna kan föras framåt och bakåt för att nå önskad punkt i ett (X,Y,Z) koordinatsystem.

Vi skulle dock komma att modifera denna arm så att axeln använde sig av HS-755HB, armbågen en HS-422 och handleden en HS-645MG. Övriga motorer kopplades således bort och basplattan flyttades och motorn i den ersattes av en stegmotor. I figur 3.1 ses AL5A i färdigmonterat stadie utan några modifieringar och med dess medföljande kretskort SSC-32 inkopplat.



Figur 3.1: Monterad robotarm.

3.2.4 SSC-32 Servo Controller

Detta är ett kretskort som används för att styra servomotorer och består av 32 kanaler, vilket innebär att man kan koppla in 32 stycken servomotorer till den om man skulle vilja det. Detta kretskort medföljde robotarmen och var mest till användning för att studera robotens rörelser och positionering. Det användes bland annat av programmen Lynx Term och Lynxmotion som beskrivs i kapitlen 3.3.5 – 3.3.6.

Kortet består av två uppsättningar med 16 rader av pinnar. Varje pinnled har tre utportar, vilka är fördefinierade för servomotorer. Mer om hur dessa kopplas kan läsas om i kapitel 3.2.8 Servomotorer. SSC-32 ansluts till datorn via en VGA-sladd, och då alla datorer ej har denna typ av ingång kan man behöva införskaffa en USB-to-Serial adapter. [4]

[4] Websida 5

3.2.5 PM 8041 X-Y Recorder
Den här maskinen används egentligen för att kunna rita ut linjer både i x och y led, men vi använde oss lite av den för just dess y led, då den kunde användas för att föra en sensor upp och ner samtidigt som objektet då skulle rotera 360 grader framför den och på så vis ha möjlighet att enkelt kunna läsa av cylinderformade objekt.

3.2.6 Proto Shield R3
Proto Shield R3 används för att förenkla byggandet av egna kretsar ovanpå Arduino. Den har extra inkopplingar för alla I/O-nålar på Arduinos olika modeller och är mycket användbar om man vill koppla samman hela sitt system på ett smidigt sätt. En protoshield består av nålar som kopplas in i de digitala och analoga portarna på Arduinon, varpå dess signaler överförs till de svarta leden ovanpå proto shielden. I mitten finns ett utrymme som används för att bygga sin krets på. Protoshielden behöver ingen extern ström, utan får allt den behöver från Arduinon.[5]

3.2.7 Breadboard
Breadboard är likt en proto shield en platta som används för att bygga elektriska kretsar ovanpå. Breadboarden är uppbygd kring rader av hål som är sammanlänkade mellan varandra, vilket förenklar ledningen av ström, samtidigt som man på så vis kan undvika kortslutningar. Anledningen att vi valt att använda både en proto shield och en breadboard beror på det begränsade utrymmet på protoshielden, då alla komponenter ej rymdes där.

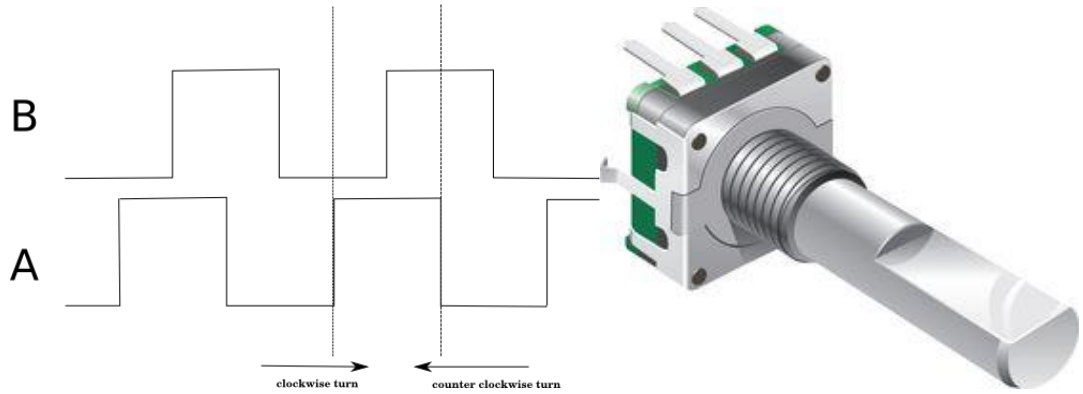
3.2.8 Potentiometer
För att manuellt kunna styra robotarmen valde vi att använda oss av tre stycken potentiometrar, en för varje motor på armen. En potentiometer är en typ av reglerbart motstånd med tre stycken anslutningar: Jord, 5V och en utsignal som skickar en signal om hur mycket potentiometern har vridits. Potentiometern är en spänningsdelare som man reglerar med hjälp av en vridbar kontakt. Genom att röra kontakten ändras således resistansen och olika pulser skickas till motorn som på så sätt kan styras. Från början var tanken att hela kretsen skulle byggas upp kring 4 potentiometerar, men eftersom en av våra servomotorer behövde röra sig 360 grader och inte 180 grader som de resterande behövde vi ett bra sätt att hålla koll på dess värden. Det var då vi beslutade oss för att styra den sista motorn med en rotary encoder, vilken beskrivs utförligare under punk 3.2.7.

3.2.7 Pec11 Rotary Encoder
En rotary encoder är mycket lik en potentiometer till utseende, men fördelen med att använda sig av denna är att den hela tiden håller koll på vad för position man har vridit kontakten till. Rotary encoders finns i två typer, absolute och incremental.

Vi valde att använda oss av en modell kallad Pec11 som är av typen incremental. Denna modell består av två stycken kanaler, A och B. Encodern är konstruerad så att då man vrider på kontakten så skickas en puls, antingen A eller B först

[5] Websida 3

beroende på vilket håll man vrider den. Kommer exempelvis puls A före puls B, vet encodern att man rör sig åt ena hållet och vice versa. Detta pulsmönster kan ses i figur 3.2 nedan. Dessa positioner kan sedan översättas till antingen digital eller analog kod. Pec11 har även en inbyggd tryckswitch, vilket dock inte är något som vi var i behov av i detta projekt, men kan vara bra att känna till. I slutprodukten valde vi att utesluta encodern och ersätta denna med en potentiometer, men bägge metoderna har sina för- och nackdelar. Anledningen till att vi valde att utesluta encodern berodde på att metodtiken kring denna är aningen mer komplicerad än att använda en potentiometer, och då vi redan hade studerat potentiometer grundligt beslutade vi oss för att detta var den smidigaste vägen att gå. Mer om denna positionering återkommer under del 3.2.11 Stegmotor.[6]



Figur 3.2: Pec11 samt dess pulsmönster.

3.2.8 Servomotorer

Under byggandet av robotarmen stötte vi på en mängd olika motorer. Samtliga servomotorer vi använt oss av har inbyggda potentiometrar för lättare styrning. AL5A robotic kit har som standard en HS-422 i basen, handleden samt gripklon, en HS-755HB i axeln samt en HS-475HB i armbågen, men som man kan läsa under 3.2.3 Robotarm valde vi att modifiera denna. Alla servomotorer har tre stycken sladdar för ground(svart), 5volt(röd), samt en insignal(gul).

3.2.8.1 HS-422

Detta är den svagaste av dom motorer vi använde. HS-422 är en mycket slittålig motor till ett billigt pris, med 180 graders rotation. Nackdelen med att använda denna modell är att den inte klarar av speciellt mycket vikt på armen, men då tanken inte är att lyfta objekt utan endast röra sig mot dem, fyllde denna motor det syfte vi ville uppnå. Det billiga priset visar sig dock till viss del i att precisionen inte alltid blev vad vi önskade.[7]

[6] Websida 8
[7] Websida 9

3.2.8.2 HS-475HB
Denna motor är en utgående modell som numer ersatts av HS-485HB. Motorn kan beställas med 90 eller 180 graders rotering, och den vi använt är den sistnämnda. Denna motor är aningen starkare än HS-422, och således är även priset lite högre.

3.2.8.3 HS-645MG
Denna motor skiljer sig från de tidigare nämnda såtillvida att den finns i utföranden om 90 grader, 180 grader samt oändligt rotation. Den motor vi hade tillgång var likt de tidigare även den 180 graders, men betydligt starkare och stabilare. Denna motor med oändligt rotation hade kunnat vara ett tänkbart val vid ersättning av bottenplattan, där vi ju önskade en 360graders rotering.

3.2.8.4 HS-755HB
Detta är den motor som sitter i botten av hela robotarmen och således styr hela axeln. HS-755HB har 180graders rotering och är den starkaste av de olika motorerna vi har använt oss av.

3.2.9 Stegmotor RS 53
Detta är den motor vi valde att använda för 360 graders roteringen av våra objekt. Stegmotorer finns i två utformingar; bipolär och unipolär, som man enkelt kan skilja åt genom att bipolära har 4 sladdar, medan unipolära består av 6 sladdar. Skillnaden mellan dessa är att bipolära kan styras genom både negativa som positiva spänningar, medan en unipolär motor endast arbetar med en polaritet. RS53 är en bipolär stegmotor som rör sig 400steg per varv, vilket således medför att vi kan få en mycket exakt positionering i vår rotation. Skillnaden mellan stegmotorer och servomotorer är kortfattat att stegmotorn baseras kring likström och stegen bestäms av det läge där det magnetiska fältet får motorn att placera sig i, medan en servomotor kan styras både med hjälp av växelström och likström. Servomotorer rör sig istället till de positioner man beordrar den att röra sig i. Anledningen till att vi valt att använda en stegmotor i detta fall, är för att vi inte behöver en noggrannhet större än de 0.9grader/steg som vi får med RS53, samtidigt som vi nu får möjligheten att kunna hantera hastigheten av rotationen, vilket var vitalt för detta projekt. Ytterligare information om hur en stegmotor ska kopplas in kan hittas i kaptiel 4.1 Konstruktion.

3.2.10 H-brygga L298
En h-brygga är en elektrisk komponent som tillåter användaren att ändra polaritet i sin krets. Vid kopplingar med likströmsmotorer används denna funktion för att kunna ändra rotationsriktningen i motorn. Anledningen till att vi valt L298-modellen är helt enkelt för detta var den billigaste, och vår krets inte krävde ytterligare funktionalitet än den som vi fick av L298. Bryggan består i princip av två stycken omkopplare i form av transistorer. Den består även av en tredje ingång som kan användas till att frikoppla motorn för att på så sätt stanna den. Denna kan även användas för att justera motorns varvtal, dvs hastigheten på rotationen, genom att skicka olika pulser. Desto lägre medelspänning som

skickas i pulserna, desto lägre hastighet rör sig motorn i. När man arbetar med h-bryggor i sin krets är det lätt att denna alstrar mycket värme och det kan därför vara klokt att även skruva fast en kylfläns.[8]

3.2.11 Magnetsensor

Vi använder oss av en modifierad sensor som mäter magnetfältet på objekt. På så vis kommer vi kunna fastställa om objekt är magnetiska och till vilken nivå. Sensorn består av två stycken spolar som skickar signaler till datorn. Den är inkopplad med tre stycken sladdar; 5V för att ge ström, jord samt en utsignal där matvärdet skickas till Arduino för inläsning genom C kod.

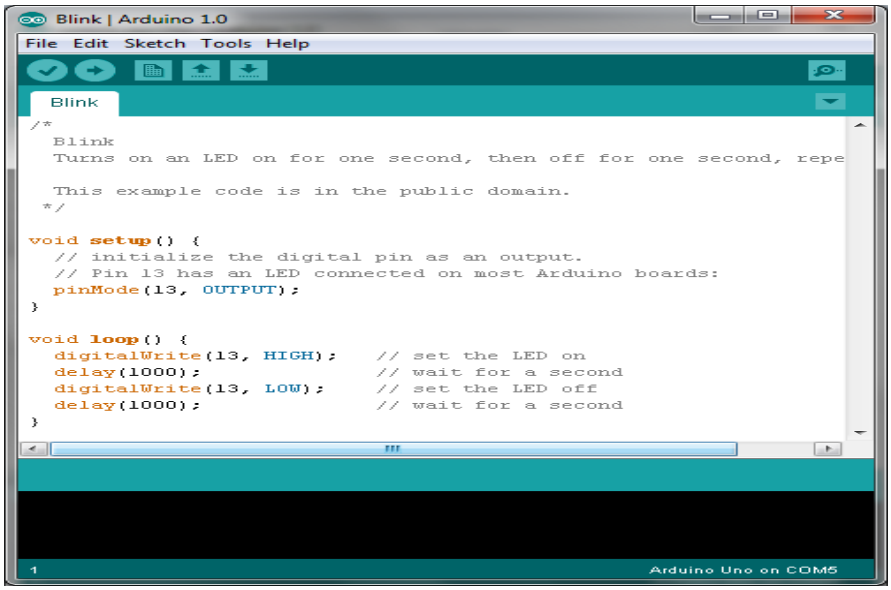
[8] Websida 10

3.3 Mjukvara

3.3.1 Arduino 1.0

Arduino 1.0 var vid projektets start den senaste versionen av mjukvaran för mikrokontrollen Arduino och används för att enkelt kunna skicka och testa sin kod på UNO. Under projektet släpptes Arduino 1.0.1, men vi valde att fortsätta arbeta i denna miljö, då detta var vad vi vant oss vid.

För att installera Arduino hämtade vi hem programmet från Arduinos hemsida. Vi började testa oss fram med en Arduino UNO och för att få detta att fungera krävdes att man stängde av brandväggar och virusskydd, gick vidare in i enhetshanteraren, högerklickade på den okända enheten (UNOn) och uppdaterade drivrutiner. Drivrutinerna fick hämtas via browse och Arduino 1.0-mapen. Vi började testa med lite färdiga exempel såsom blink och fade. Vi kopplade även in en LED-lampa och såg att det fungerade även med denna. Arduino är baserad på ren c-kodning, med vissa modifikationer. I figur 3.3 ses ett exempel på arduino med inlagd blink-funktion. För att köra koden används ikonen med högerpil nästa längst upp till vänster, och en monitor där man kan följa utskrifer får man upp genom förstoringsglaset uppe till höger.



Figur 3.3: Arduino mjukvara med exempelkod för blink-funktion.

3.3.2 Meshlab

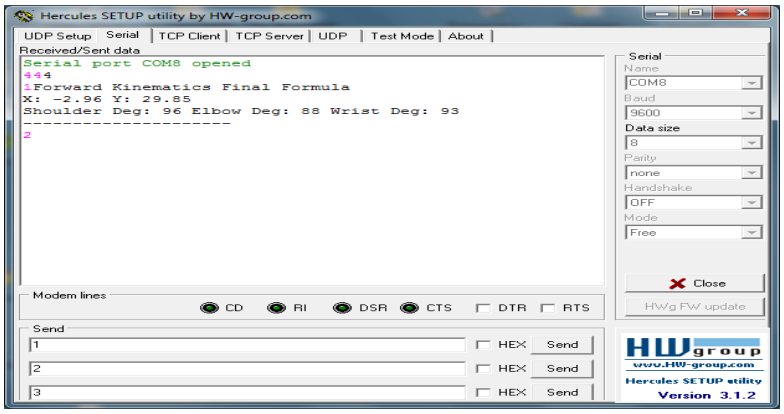
Meshlab är ett avancerat program inom 3D-utveckling som används för att redigera bilderna man tagit med sin kamera. Man kan använda det såväl för att klistra samman bilder tagna ur olika vyer till en gemensam bild, som att ta bort oönskade delar av bilden. Programmet i sig ger en klar och strukturerad bild över alla vinklar av en tagen bild. Detta är inget program som vi själva berörde mycket inom utvecklingen av armen, men är bra att bekanta sig med för att få förkunskap kring hela systemet och förstå processen från start till slutprodukt.

3.3.3 FlexScan 3D

Flexscan 3D är likt Meshlab ett program som används inom 3D-utveckling och direkt samarbetar med många kameror och projektorer utan modifieringar. Programmet är mycket användbart om man vill skapa sig en egen 3D-scanner oberoende av vad för utrustning man har att tillgå, och programmet scannar snabbt in bilderna man tar för att sedan bygga om dem till färdiga 3D objekt. Nackdelen är att funktionerna och resultatet är begränsade i jämförelse med Meshlab, men fördelen är att det är ett kostnadsfritt program, som går snabbt att sätta sig in i. (www.3d3solutions.com, 2012)

3.3.4 Hercules SETUP utility

Detta är en serieports-terminal som vi använde för att försäkra oss om att informationen som skulle skickas mellan de olika programmen kom fram som den skulle. Vi skickade data på den seriella com-porten som då kommunicerar med Arduino UNO. Från hårdvaran skrivs information ut och skickas till Hercules där man ser vad man har tagit emot. Programmet fungerar även bra för att testa TCP/UDP, och är gratis nedladdningsbart från hw-groups hemsida. Exempel på en utskrift av då vi testade att graderna på våra servomotorer stämde kan ses i figur 3.4. (www.hw-group.com, 2012)[9]

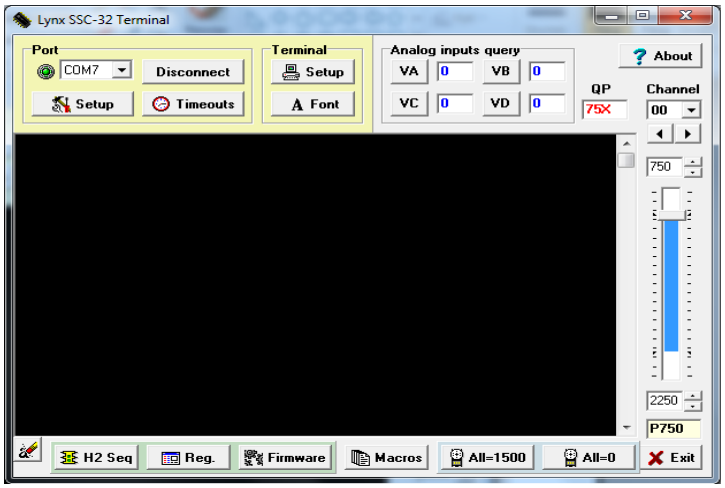


Figur 3.4: Hercules visar motorernas grader

3.3.5 Lynx Term

Lynxterm är ett gratis nedladdningsbart program som används för att testa alla funktioner hos SSC-32. I figur 3.5 ses huvudfönstret i Lynx Term, med alla dess knappar. Programmet består endast av detta fönster och går mycket snabbt att sätta sig in i. För att starta programmet väljer man först vilken port som kortet är ansluten till, i detta fall COM7. Genom att att välja ”Channel” på högersidan till den kanal som önskad motor är inkopplad till, så kan man byta mellan de olika axlarna, och genom att förflytta mätpinnen på scrollmätaren så kan man vrida axeln åt två olika håll. Nackdelen med detta program är att man endast kan testa en kanal åt gången och således inte sätta alla motorer i rörelse samtidigt.

[9] Websida 4



Figur 3.5: Lynx Term startview.

3.3.6 Lynxmotion RIOS SSC-32

Lynxmotion RIOS (Robotic arm Interactive Operating System) är ett program baserat på windows (95, 98SE, 2000, XP, Vista, 7), som används för att kontrollera AL5-serien bland robotarmar baserade på SSC-32. Lynxmotion kan användas för att spela in och lära roboten serier av rörelser, kalibrera koordinater för bättre resultat samt styra roboten med hjälp av datormusen eller en joystick. Lynxmotion använder externa inputs för att styra robotens rörelser inom stängda projekt, men även externa outputs kan användas. Lynxmotion RIOS följde med i samband med köpet av AL5A, och för att installera programmet krävs en serienyckel. Innan man kan börja testa roboten måste man koppla in den till rätt COM-port, och därefter i programmet även kontrollera att dessa portar överensstämmer. Lynxmotion är ett mycket användbart program med färdiga funktioner och uträkningar för rörelser och positionering, men fungerade inte speciellt bra i sammarbete med Arduino. Programmet är därför bra att använda för att lära sig hur robotarmen fungerar, men om man som i vårt fall vill specialkonstruera en arm är detta ingenting som är att rekommendera för slutprodukten.

I figur 3.6 ses första fönstret i lynxmotions interface då man startar det. I högerfältet vid **”COM”** finns inställningar för Com-port. På vänstersidan finner man sju stycken knappar vilka har följande funktioner:

”All=1.5mS”: placerar roboten i dess grundposition.

”SSC-32”: Används för att kalibrera robotens positionering.

”ARM”: Här kan man testa så önskade koordinater nås.

”Project”: Här kan man öppna både exempelprojekt som egensparade rörelsemönster.

”XYZ sys”: Detta är inställningarna för vilka axlar man önskar att X, Y, Z ska representera.

”Moves”: Används för att spela in sina egna rörelsemönster och spara som projekt.

”Play”: Spelar upp projekt och tillåter användaren att lägga in mer avancerade funktioner som loopar och villkor för rörelsemönstet.

Figur 3.6: Lynxmotion startview.

3.3.7 C Programmering

Språket C skapades mellan 1969 and 1973 av Dennis Ritchie och är ett av de mest använda språken i hela världen. I detta projekt kommer vi dock använda oss av programmet Arduino där C språket är lite modifierat genom att det finns vissa standardfunktioner som Setup() och Loop(). Detta begränsar möjligheterna med språket till en viss del, men inget som kommer påverka oss i detta arbetet.[10]

3.4 Prestanda & optimering

Om man skrivit i ren C och inte använde sig av Arduino så skulle man kunna få ett snabbare och stabilare system. Detta eftersom Arduino använder sig av en del förenklingar i koden, vilka tar upp onödigt med utrymme.

Att höja prestandan på själva robotarmen är till stor del en kostnadsfråga. Starkare motorer hade givit en precisare positionering, samtidigt som längre armar hade tillåtit avläsning på större objekt.

När det kommer till de elektriska komponenterna skulle hela kretsen kunna förminskas genom att koppla samman allt på ett effektivare sätt, använda kortare sladdar etc. Under utvecklingen ansåg vi dock att det var klokt att använda sig av en stor arbetsyta, då många delar skulle komma att flyttas fram och tillbaka, samt ersättas av nya komponenter. För många och för långa sladdar kan dock bli som antenner i en krets och på så vis störa signaler, vilket medför felmarginaler och vibrationer.

För att få robotarmen att arbeta på effektivaste vis är det viktigt att spänningssätta den rätt. Får roboten för lite spänning orkar motorerna inte arbeta, och ger man för mycket, riskerar komponenter att bli överhettade. Det är därför viktigt att se till att läsa de tekniska specifikationerna för varje separat del innan man spänningssätter sin krets.

[10] Websida 7

3.5 Utrustning & Säkerhet

Konstruktionen av robotarmen krävde ett antal olika verktyg såsom skruvmejslar och insexnycklar. Utöver det använde vi oss även av en voltmätare för att se till så att kablarna var kopplade rätt så att inte kortslutning skulle ske som kunde skada utrustningen. Vi använde oss även av voltmätaren för felsökning i hela det färdigbyggda kopplingssystemet. Vid tillverkning av bottenplattan var vi tvungna att använda en borrmaskin för att skapa hålen och för styrning av robotarm kan skyddsglasögon vara att rekommendera då roboten kan röra sig i hög hastighet och göra oförutsedda rörelser. Vid tillverkning av en hemmagjord arm så fick vi använda oss av en vinkelslip för att kapa vissa delar som vi behövde använda oss utav. Även här krävdes skyddsglasögon, men också hörselskydd.

Innan man startar robotarmen måste man kontrollera att strömadaptern som används är satt på 6volt, så att ingen annan person har råkat byta styrkan. Det är även viktigt att se till så att +/- sitter rätt. Detta resulterade för vår del i att kondensatorn i en motor smälte, vilket sedan ledde till lång felsökning, då motorn började bete sig underligt utan förklaring.

För hög ström ledde även till att vi brände två stycken rotary encoders, så för att vara helt säker på att man inte råkar ut för detta kan det vara bra att pröva sig fram med resistorer i kopplingen, samt även endast slå på spänningen under kortare perioder.

Spänningen får dock inte heller vara för låg. Kopplar man exempelvis Arduino UNO till datorn och enbart förlitar sig på denna spänningskälla så klarar den utan problem av att driva en servomotor, men kopplar man på ytterligare en eller två så räcker datorns spänning inte till, vilket visar sig genom att Arduino startar om och motorerna dör. Detta åtgärdas enkelt genom att koppla till ytterligare en spänningskälla, vilken bör ligga mellan 7-12 Volt. Arduino klarar dock av spänningar upp till 20V.

Under själva testandet av robotarmen kan det vara bra att ha ett mjukt underlag, då spänningsfall och hastiga rörelser kan leda till att armen slår i saker och således tar skada. På vår slutgiltiga prototyp ser man tydliga tendenser av detta, då armen inte är helt rak, vilket självfallet även påverkar dess positionering på ett negativt sätt och har bidragit till att slutprototypen har en felmarginal på 2-3 %.

Då vi själva modifierade rotorn för våra objekt och till att börja med ersatte den första motorn med en 360 graders servomotor uppstod det vissa problem även här. Den nya motorn var inte anpassad för den färdigbyggda rotorplattan och till att börja med skruvade vi åt hela stycket aningen för hårt, vilket resulterade i att plattan endast rörde sig små steg åt gången. Detta åtgärdades dock snabbt, men nästa problem låg i att ett kugghjul inne i själva motorn lossade. Vi fick således plocka isär hela motorn och laga den.

När vi senare hade bestämt oss för att använda en stegmotor för slutprodukten byggde vi en plattform av tekniklego. Lego fungerar mycket bra för att enkelt kunna konstruera ställningar till denna typ av prototyp, men vid en färdig slutprodukt bör Legot ersättas med ett stabilare material.

4. Genomförande

Detta kapitel går igenom hela genomförandefasen i uppdelad ordning av konstruktion, beräkningar och programmering för att läsare lättare ska förstå produkten. Hård- och mjukvarorna som tas upp i detta kapitel beskrivs endast ytligt och kan läsas om mera utförligt i kapitel 3. Vi har valt att bifoga bilder på hela genomförandefasen i appendix, varvid de olika momenten refereras till i texten. Alla resultat ur genomförandefasen presenteras i kapitel 5 Resultat.

Då projektet startade började vi med att studera vår uppgift väldigt övergripande, vad för fakta vi behövde inhämta, samt vilka områden vi skulle röra oss inom. De första dagarna gick åt till planering och studerande av mjukvara i väntan på att roboarmen skulle komma. Vi skapade även en förutspådd livscykel för vårat arbete som kan ses i Appendix A figur 4.20.

4.1 Konstruktion

4.1.1 Robotarm

Konstruktionen av robotarmen påbörjades direkt när produkten kom. För att kunna förstå hur den skulle monteras började vi först att läsa igenom en dokumentation som handlade om hur den skulle användas. Detta gav en viss insikt om vad man kunde göra med den när den var färdigmonterad, men inte vidare mycket kunskap om alla olika delarna. Nu gick vi på bruksanvisningsdokumentationen, där vi steg för steg kunde följa själva utvecklingen av produkten. Först så påbörjade vi monetring av själva bottenplattan för kretskortet (Figur 4.1).

Efter detta så påbörjades konstruktionen i en annan ände, nämligen första servo-motorn. Denna motor kan vridas i 180 grader. Detta var dock inte den funktion vi eftersökte, utan i framtiden skulle vi helt enkelt komma att byta ut denna motor mot en som kunde rotera 360 grader. På botten av den cirkelformade plattan finns skruvhål som paras ihop med kretskortets bottenplatta (kan ses på Figur 4.1 där ena kanten är cirkulär).

När sedan kretskortet var påskruvat på bottenplattan så monterades även den roterande plattan ihop med den (Figur 4.2). Utöver det så kapade vi ut en bit trä så att vi kunde få allt att sitta fast för en lättare fortsatt montering. Denna vita smala träskiva (Figur 4.3) kom senare att bytas ut mot plexiglas för en stabilare och mer elegant lösning.

Nästa steg blev att montera de övriga axlarna till robotarmen (figur 4.4). Utöver den roterande bottenplattan så kommer robotarmen även kunna vinklas på tre andra axlar, med hjälp av ytterligare tre servo-motorer. Varje motor har en utgående sladd som kopplas till kretskortet för att kunna styras av signaler.

Då alla motorerna blivit monterade, skruvarna åtdragna och alla sladdar kopplade så återstod det bara att skapa en solid bottenplatta så att hela systemet står stabilt. Vi valde att använda oss utav plexiglas. Hål borrades för att robotarmen skulle kunna sättas fast och svarta dämpande dynor klistrades på

under glasplattan (figur 4.5).

4.1.2 Testning

När hela konstruktionen var färdig var det dags att prova de olika axlarna. Det uppstod först ett problem vid installationen av mjukvaran LynxMotion, vilket visade sig vara lite strul med programnyckeln. Efter att installationen till slut lyckades provkörde vi den, dock så rörde sig inte axlarna. Det visade sig att SSC-32 krävde en strömkälla och motorerna en egen, så efter att båda hade fått ström kunde vi nu röra på axlarna. Vi använde oss först av LynxTerm, vilket är ett väldigt enkelt program där man kan styra en axel i taget. Vi övergick sedan till Lynxmotion som har betydligt fler funktioner för testning av axlarna. Detaljerad information om dessa program går att läsa i kapitel 3.3.5-3.3.6.

Med programmet Lynxmotion skapade vi även en csv fil som bestod av tre stycken punkter (X1,Y1,Z1, X2, Y2, Z2, X3, Y3, Z3). Detta var för att mjukvarugruppen i vårat sammarbete behövde veta hur koordinaterna sparas.

4.1.3 Experimenterande

Av olika anledningar om tidspress och kvalitet övergav vi dock vår robotarm ett tag och började skissa på en idé kring en fast robotarm som rörde sig på en hiss. Vi ville sedan ha en 360 graders roterande platta framför för avläsning. Målet var att sensorn skulle sättas fast i en arm som vi själva konstruerar. Armen består av en fjäder som ska tryckas ihop om sensorn går emot objektet som läses in. Eftersom alla koordinater av objekten vi ska scanna in redan är kända behövde vi inte veta hur armen låg i x och z led utan så länge vi höll koll på y-ledet och visste hur lång tid det tog att rotera objektet ett varv eller eventuellt höll koll på rotorns grader, samtidigt som vi visste att sensorn alltid låg mot objektet, kunde vi på så sätt få ut all data vi eftersökte. Enligt denna metod kan man även läsa in objekt som inte är helt cirkelformade. Vi började konstruera en hiss för armen som bestod av två stycken metallpinnar. Den ena med gängor och den andra helt plan (figur 4.6). Pinnen med gängor fästes sedan i en motor som kunde rotera den och genom dessa båda pinnar byggde vi ett fäste för armen. Ena hålet i fästet var gängat efter den ena pinnen, medan det andra hålet endast var till för att hålla fästet helt fast på axeln. På så vis skulle fästet röra sig upp och ner längst pinnarna då motorn roterade. Denna princip hämtade vi från en av laborationssalarna på skolan där liknande metod användes (figur 4.8). Tidsbristen gjorde dock att vi var tvungna att snarast få ut en prototyp och vi sköt upp denna metod tills vidare och arbetade även ett tag med motorerna i en skrivare där funktionerna för både hiss samt rotering redan fanns (figur 4.9). Denna variant blev dock svår att styra via seriell kommunikation och övergavs snabbt.

Nu påbörjade vi en lite enklare variant för att snabbare kunna prova programmet som gruppen ansvarig för 3D-uppritning hade skapat. För det så använder vi oss av PM 8041 X-Y Recorder som egentligen används för att kunna rita ut linjer vertikalt och horisontellt. Tanken var att vi skulle använda oss utav dess Y-axel, och på så vis kunna montera fast en sensor som kan flyttas upp och ner. Vi provade lite olika varianter samt byggde en prototyp av en fjäderarm, men märkte sedan att det var mycket svårt att hitta en riktig stabilitet i armen

(figur 4.7). Vi var även tvungna att använda oss av plast eftersom vi vill kunna läsa av magnetfält och då kan vi inte använda oss av något som är magnetiskt för det skulle ge störningar.

4.1.4 Återgång till robotarm

Hela denna idé fick dock läggas ner, dels pga materialbrist, dels för att systemet i sig blev för instabilt och inte fungerade helt i praktiken då armen inte följde objektet lika bra som vi hade tänkt oss. Allt arbete var dock inte förgäves då vi fått en mycket bättre insikt i hur exakt systemet måste vara och vad för metoder vi borde hålla oss borta ifrån. Vi återgick således till vår första robotarm, som hitills varit den mest precisa metoden, men påbörjade genast justeringar eftersom vi fortfarande ville bygga armen kring idén att objektet i fråga skall röra sig på en platta, medans armen håller sig på en fast position i XY-led. Vi monterande om robotarmen så den numera satt fast i plattan medan dess rotorerande motor nu var placerad framför, där objekt alltså skall placeras.

4.1.5 Kopplingar

För att kontrollera robotarmen med vår kod behövde vi nu börja använda oss av Arduino på allvar och vi bestämde oss således att koppla robotarmen direkt in till vår UNO. Detta gjorde vi med hjälp av en sammanlänkning av SSC-32 och Arduino genom att koppla två sladdar mellan RX och ground på SSC-32 till TX och ground på Arduino, samt en breadboard där vi kunde bygga vår elektriska krets (figur 4.10). Denna bräda blev dock lite för klumpig, då alla kretsar åkte ur så fort vi rörde på något, och vi flyttade snabbt över vår krets till en proto shield som är en platta som används för att förenkla byggandet av elektriska kretsar ovanpå Arduino. Till denna kopplade vi alla fyra servomotorer samt fyra stycken potentiometrar. Vi kunde nu även utesluta SSC-32, då allting kopplades direkt in i proto shielden. Varje motor kopplades till ground, 5V spänning samt varsin digital port. Potentiometrarna, som även dom har tre stycken nålar kopplades även dom till ground, 5V spänning samt varsin analog port (figur 4.11). På så sätt kunde vi nu börja testa vår koppling genom endast små ändringar i exempelkod som kan hittas i Arduinos mjukvara under File->Examples->Servo->Knob.

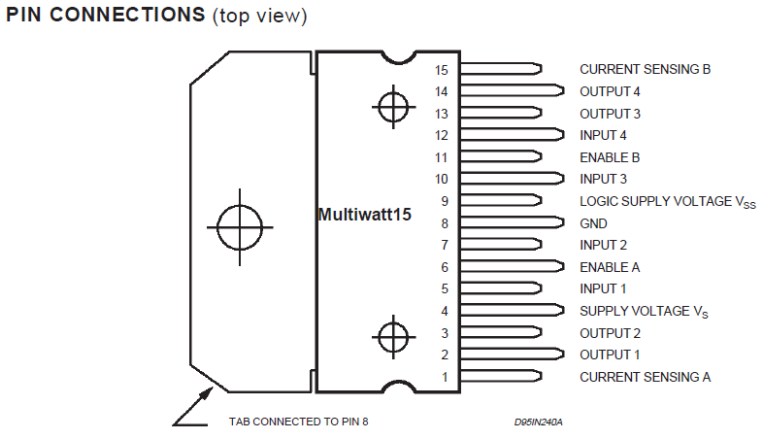
4.1.6 Styrning

För att styra den roterande plattan som ska kunna gå 360 grader så fann vi att en potentiometer troligen inte var det bästa tillvägagångssättet, då en potentiometer endast har 180 graders rotering och det finns ingen vidare bra metod för att kunna beräkna hur många grader plattan vridit sig. Vi beslutade oss då för att prova en Incremental Encoder (PEC11-4230F) som kan registrera sin startpunkt och på så vis känna av hur långt från startpunkten den sedan vridits. Själva servomotorn samarbetade dock inget vidare bra med den så vi kunde inte få ut något bra resultat. Felet låg i att servomotorn kan styras via kod, men på sättet som incremental encodern skickar pulserna så kan man inte fastställa hur många grader den har vridits. Man hade kunnat montera fast Encodern på den roterande plattan och på så vis skickat signal om att plattat ska roteras och när Encodern vrids en puls så slutar plattan rotera och då vet man hur mycket den har vridit sig. Detta hade dock kunnat ge en viss felmarginal

som skulle kunna ge större konsekvenser, då plattans vridning inte avslutas direkt när en puls har gått och den kan råka vridas mer. Kodmässigt skulle detta innebära att gradräkningen på plattan blev fel. För att verkställa den idèn hade väldigt mycket konstruktion krävts som i slutändan möjligtvis inte hade blivit praktiskt, vilket ledde oss till att utesluta servomotorn för den roterande plattan helt och hållet. Vi bestämde oss därför för att använda en stegmotor istället.

4.1.7 Stegmotor

Med hjälp av en stegmotor kan vi kodmässigt kunna styra den att gå ett visst antal grader utan vidare större problem, men för att kunna starta programmeringen runt den så började vi med att montera den på en broadboard. För att detta ska fungera skaffade vi oss en H-Brygga, som används för att kunna ändra rotationsriktningen på en likströmsmotor och är vital vid styrning av stegmotorn.



Figur 4.13: Kopplingsschema för H-brygga.

På bilden ovan ser man en numrering som går från 1 till 15 på H-Bryggan. Alla dessa måste kopplas för att stegmotorn ska fungera. Nummer 1, 8 och 15 är man tvungen att koppla till Ground. Utöver det så måste man koppla 4, 6, 9 och 11 till 5V för att ge ström. Nu har man även fyra sladdar som går ifrån stegmotorn (blå, grön, röd, svart), de fyra sladdarna är uppdelade i två par: Svart och Grön, Röd och Blå. Svart kopplas in i 14 och grön kopplas in i 13, medans röd kopplas in i 3 och blå i 2. Nu behöver man även fyra stycken input signaler som kopplas i 10, 11, 12 och 13 som går in i Arduino Uno på digitala ingångar.

4.1.8 Roterande platta

Samtidigt gick vi även tillbaka till att styra alla motorerna med potentiometerar. Istället för att rotera vår platta i fas med vridningen av potentiometern övergick vi nu till att arbeta på en metod som byggde på att då man vred potentiometern åt ena hållet, stegade motorn fram åt detta håll, enda tills man vred tillbaka den roterande kontakten till mittläget, vilket motsvarade ett stillastående läge på motorn. Vrider man kontakten åt andra hållet startar således en stegning även åt detta håll, ända tills ratten vrids tillbaka.

Stegmotorn vi använder oss av har en pinne som sticker upp ur motorn. Det är denna pinnen som roteras. För att kunna bygga en sorts plattform att ställa

objekt på så valde vi oss att använda tekniklego. Vi hittade bitar med hål i som kunde fästas på pinnen, och sedan kunde man väldig smidigt bygga på det för att få en stabil form. Utöver det så kapade vi till en bit plexiglas som fästes på legoformen för att få en fin och slät yta att ställa objekten på. För ett mer grafiskt utseende klippte vi även till en bit papper där vi ritade ut X och Z ledet på. Så att när plattan roteras kan man enkelt följa visuellt hur mycket den har vridit sig.

4.1.9 Kopplingsmetod

Arduino Uno har ett antal ingångar för analoga samt digitala insignaler. I vårat system så använder vi oss av dessa:

- **A0:** När robotarmen körs så styrs dess axlar av potentiometrar. Koden gör så att alla potentiometrar hela tiden läser från inportarna, vilket innebär att om man skulle vilja skicka armen till en viss koordinat (X,Y,Z) så skulle den först gå dit, men sedan åka tillbaka till det läge som potentiomterarna är inställda på. Därför skulle man då inte kunna besöka punkter smidigt. För att lösa detta problemet installerade vi en strömswitch. Denna fungerar så att den är kopplad mellan 5V som även går till potentiomterarna, och så är den kopplad in till A0. Om 5V förs igenom switchen läser programmet in det, om switchen är stängd säpper den inte igenom 5V in i analog0 porten vilket innebär att potentiomterarna avaktiveras från våran kod.
- **A1:** För att kunna läsa data från en punkt i xyz koordinatsystemet som våran arm kan åka runt i använder vi oss av en sensor. Sensorn består av tre sladdar: Jord, 5V, Utsignal. Utsignalen är kopplad till analog1, vilket gör så att koden kan läsa av värdet från sensorn och representera det.
- **A2:** Den andra axeln på robotarmen kallas för Elbow (armbåge). För att styra den manuellt använder vi oss av en potentiometer. Utsignalen på potentiometern går in i analog2 vilket gör så att elbow-axeln kan vridas.
- **A3:** Den sista axeln på robotarmen kallas för Wrist (handled). För att styra den manuellt använder vi oss av en potentiometer. Utsignalen på potentiometern går in i analog3 vilket gör så att wrist-axeln kan vridas.
- **A4:** Den första axeln på robotarmen kallas för Shoulder (axel). För att styra den manuellt använder vi oss av en potentiometer. Utsignalen på potentiometern går in i analog4 vilket gör så att shoulder-axeln kan vridas.
- **A5:** För att kunna snurra objekt använder vi oss av en stegmotor. Stegmotorn har 400steg för att gå 360grader. Då en potentiometer kan vridas cirka 180grader så passar den egentligen inte vidare bra för att styra en stegmotor. Men vi kopplade så att utsignalen från potentiometern går till analog5. Sedan om den är under 70 grader i vridning, stegas motorn åt ena hållet. Om den är över 130 grader så stegar motorn åt andra hållet. Vilket innebär då potentiometern är satt på en grad mellan 70 och 130 så händer ingenting och stegmotorn förblir stilla.
- **D2:** För att signaler om ändring av vinkeln på Elbow-axeln ska kunna tas

emot av motorerna kopplar vi den till digital2. Detta innebär att när man först vrider på potentiometern så skickas den vridningen till analog2. När koden sedan har läst in det skickas den vidare till digital2 för att flytta axeln.

- **D3:** För att signaler om ändring av vinkeln på Wrist-axeln ska kunna tas emot av motorerna kopplar vi den till digital3. Detta innebär att när man först vrider på potentiometern så skickas den vridningen till analog3. När koden sedan har läst in det skickas den vidare till digital3 för att flytta axeln.
- **D5:** För att signaler om ändring av vinkeln på Shoulder axeln ska kunna tas emot av motorerna kopplar vi den till digital5. Detta innebär att när man först vrider på potentiometern så skickas den vridningen till analog4. När koden sedan har läst in det skickas den vidare till digital5 för att flytta axeln.
- **D10, D11, D12, D13:** H-Bryggan behöver fyra stycken utsignaler för att fungera. Dessa är kopplade till digital10, digital11, digital12 och digital13.

4.1.10 Störningar i systemet

Vid styrning av robotarmen med våra potentiomterar märkte vi av en viss störning i systemet. Robotarmen hackar lite fram och tillbaka då den är i spänt läge. Enda sättet att få armen helt still är att ställa den rakt uppåt så alla axlarna är på 90 graders vinkel. Störningarna beror på att kablarna som används vid kopplingen fungerar lite som antenner och kan ta emot stötar som de egentligen inte ska få. Detta hade kunnat ordnas genom att koppla om hela systemet och möjligtvis använt oss av lödning för att få ett mer fast kretskort för komponenterna. Istället kopplade vi in en kondensator mellan 5V och jord i kopplingen som har förmågan att lagra en viss elektrisk laddning vilket gör så att störningarna i systemet minskar, men försvinner dock inte helt.

4.1.11 Positionering

När vi nu kunde få vår stegmotor att röra sig till exakta grader, hade vi det som krävdes för att kunna ta reda på vår z-position. Med hjälp av forward kinematics och inverse kinematics som beskrivs utförligare i punkt 4.2, kunde vi sedan räkna fram även våra x och y positioner och således få ut all information vi eftersökte kring positioneringen. Då dessa värden var beräknade hade vi all information som krävdes för att slutföra projektet. Efter samrådan med gruppen för 3D-uppritning beslutade vi oss för att styra positionering punktvis, dvs att roboten går till de punkter som använder trycker på det virtuella objektet i datorn. Vi hade tidigare funderat kring idéer om att arbeta kring en spiralrörelse kring objektet eller eventuellt jobba med lager i y-led, men då tiden började ta slut var detta den metod som gick snabbast att slutföra. Det var också nu som vi märkte att felmarginalen på våra motorer inte bara berodde på att robotarmen tagit skada, utan för att motorerna var felinställda. Vi hade under hela projektets gång förutsatt att Arduinos bibliotek för servo arbetade korrekt med våra motorer. När vi nu mätte pulserna för de olika motorerna märkte vi att istället för att vara inställda mellan 600-2400 pulser som vi hade trott, så låg värdet mellan 540-2400, och vi hade alltså hittat anledningen till vår felmarginal.

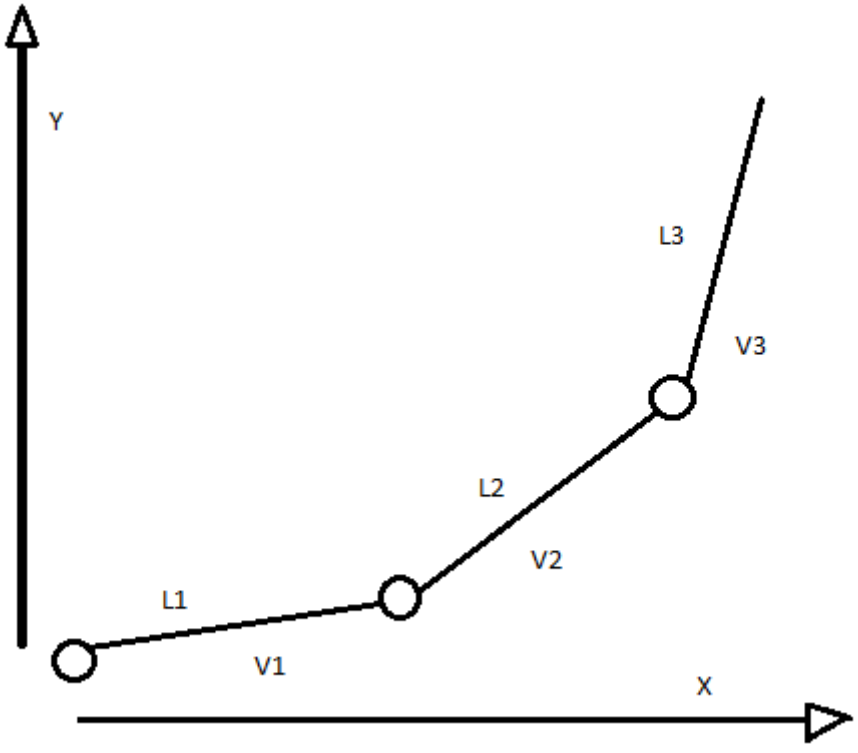
Lösningen på detta problem finns i ett bibliotek vid namn SoftwareServo.h. Den version av Arduino som vi använde oss av sammarbetade dock inget vidare med detta bibliotek, och armen började vibrera nått enormt när detta implemeterades. Eftersom tiden för projektet var slut valde vi därför att använda vår tidigare kod och avsluta här, trots upptäckt fel. När 10 veckor var slut hade vi således lyckats färdigställa vår prototyp. Hela processen kan följas i appendix A, där slutprodukten syns i figur 4.12. Hade ytterligare tid funnits hade vi önskat förfina våra kopplingar, byta till tydligare färger på sladdar samt importera funktionerna för SoftwareServo.h, men då projektet som sagt drog ut på tiden och krävde de 10 veckor vi hade, fick detta bortprioriteras.

4.2 Beräkningar

Olika sorters beräkningar krävs för att kunna få ut graderna på axlarna samt kunna få ut på vilken x,y och z punkt som armen är på. De metoder vi använt oss av för detta beskrivs nedan.

4.2.1 Forward kinematics

För att robotarmen ska kunna kalibrera sig med ett mjukvaruprogram som en annan grupp arbetar på behöver vi få ut tre stycken punkter på ett objekt. Dessa tre punkter består alla av ett x,y och z värde. För att kunna få ut våra koordinater till robotarmen var vi tvungna att räkna fram dem själva, då vi inte längre använde oss av Lynxmotion, som har färdiga funktioner för beräkning. Tanken är att mjukvaran som den andra gruppen har konstruerat ska pricka ut tre punkter, sedan ska robotarmen föras manuelt till dessa punkter. När den besöker en punkt ska dessa koordinater skickas till mjukvaran för kalibrering. Eftersom vi från början bara får ut graderna på våra axlar på robotarmen så behöver vi nu använda oss av Forward Kinematics för att kunna göra om axlarnas vinklar till en (X,Y,Z) punkt. Detta gjorde vi genom en metod som heter forward kinematics och bygger mycket kring Pythagoras sats och vinkelberäkning.[11]



Figur 4.14: Robotarm med tre axlar sedd i x,y-plan.

Denna bild beskriver en arm där varje cirkel motsvarar en motor och alltså en vridpunkt. L_X motsvarar här längden av våra armar medan V_X motsvarar

[11] Websida 11

vinkeln för motorerna. Dock så är denna bilden för en arm där den inte kan gå åt båda hållen. Utan exemplet ovan kan bara sträcka sig uppåt, alltså inte gå under X-axeln. Detta bidrog till lite fundering kring hur formeln skulle utvecklas.

Formel 1: För uträkning enligt figuren 4.14.

$$X = L_1\cos V1 + L_2\cos(V1 + V2) + L_3\cos(V1 + V2 + V3)$$

$$Y = L_1\sin V1 + L_2\sin(V1 + V2) + L_3\sin(V1 + V2 + V3)$$

Genom att formeln ovan är för simpelt gjord och inte fungerar i vårt fall där vi har en mer komplex robot, så var vi tvungna att kompensera vinklarna för de olika axlarna. Detta gjordes på grund av att alla axlar måste känna till varandras lägen för att kunna få ut vart den pekar.

Uträkningen som vi skapade för att få fram de olika positionerna baserades mycket på att vinkeln från den tidigare armen måste användas för att beräkna vinkeln på nästa arm (en sorts kompensering). Slutligen fick vi ut en position for Xpos och Ypos som kan ses i Appendix C formel 2 och 3.

Tabell 2: Förtydligande av formlerna.

Namn	**Förklaring**
Xpos	X positionen för robotarmens spets.
Ypos	Y positionen för robotarmens spets.
L1	Längden på armen från första motorn till andra. Från Shoulder till Elbow.
L2	Längden på armen från andra motorn till tredje. Från Elbow till Wrist.
L3	Längden på armen från tredje motorn till spetsen på armen. Från Wrist till toppen.
V1	Vilken grad som Shoulder är på.
V2	Vilken grad som Elbow är på.
V3	Vilken grad som Wrist är på.
90	Gradtalet för kompensering.

Detta gav oss korrekt positionering av robotarmen, dock så är det inte helt klart. För vi måste anpassa robotarmen gentemot den roterande plattan. Därför använder vi oss av variabeln LengthBetween som är avståndet i cm mellan centrum på robotarmen och centrum på den roterande plattan. Vi översatte sedan detta till c-kod (Appendix B: void ForwardKinematics), vilket gav följande formel.

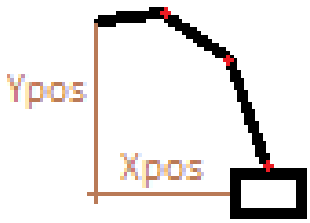
På ungefär samma sätt som vi gjorde för Xpos (Appendix C, formel 4) måste vi nu även anpassa Ypos (Appendix C, formel 5) för höjdskillnaden gentemot robotarmen. Därför använder vi oss av HeightBase vilket motsvarar höjden på stegmotorn.

Tabell 3: Beteckningar.

Namn	Samma som	Konstant Värde [cm]
LengthBetween		19.5
HeightBase		9.5
LengthShoulder	L1	9.5
LengthElbow	L2	10.5
LengthWrist	L3	8.5
ShoulderVal	V1	
ElbowVal	V2	
WristVal	V3	

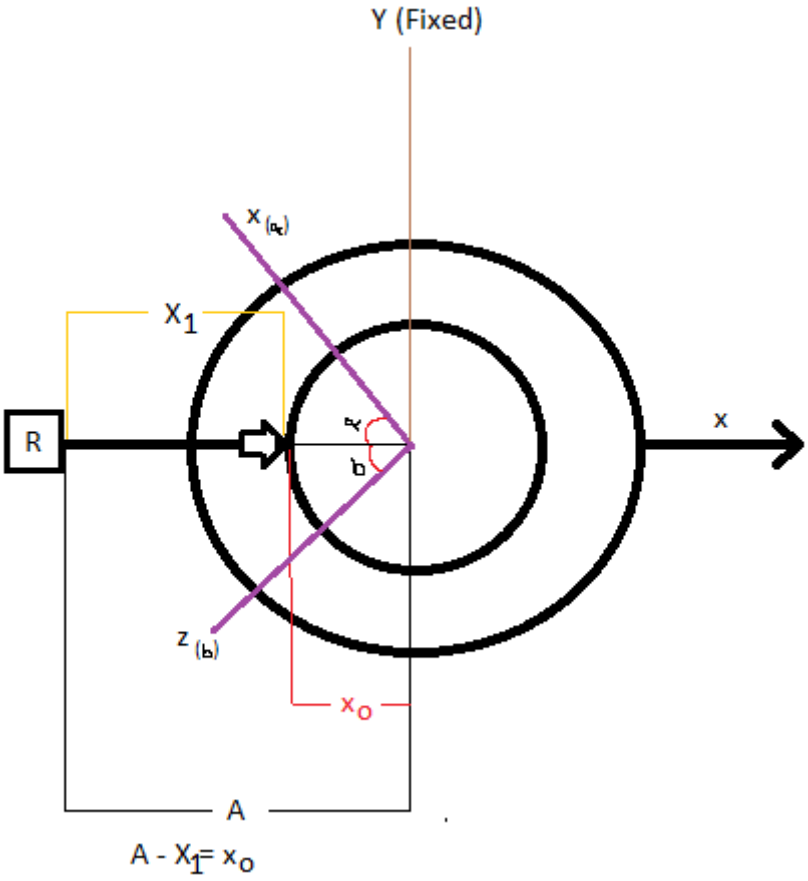
I själva C koden delar vi alla sin och cos uträkningar med (180/M_PI) då sin och cos tar in radianer. Dock valde vi att inte visa det i formlerna här för det skulle bara skapa mer förvirring.

Nu har vi fått ut Xpos och Ypos, detta motsvarar variablerna enligt en enkel sketch nedanför:



Figur 4.15: Visar hur Ypos och Xpos ser ut i verkligheten.

Nu stöter vi dock på ett nytt problem. Vi behöver nämligen även få ut en Z punkt, och vi vet att Xpos kommer ändras ytterligare då de berör varandra. Detta sker genom mer avancerad uträning enligt bilden nedan.



Figur 4.16: Visar gradbestämning kring roterande platta.

Bilden ovan är en översikt på hur vi beräknade X, Y och Z punkten för robotarmen då man bara känner till vinkeln på axlarna och armarnas längd, samt avståndet från robotarmens fäste i bottenplattan till centrum på den roterande plattan. R på bilden ovan står för ”robotarm”, och pilen på den är toppen på yttersta armen (där vi vill få ut X,Y,Z och där sensorn placeras). Från robotarmens placering (R), så går en X-axel rakt ut mot den roterande plattan som i standard läge står i 0 grader. Utöver det så har vi en fast Y axel rakt uppåt

från den roterande plattan, som inte påverkas av X och Z värdet. För att kunna få ut ett Z värde här ritar vi upp två linjer som är vinkelräta mot varandra. Dessa två linjer blir då x(α) och z(b). Alfa och beta är hur mycket den roterande plattan har vridit sig i grader från utgångspunkten 0 grader. Då den roterande plattan vrider sig, så ändras också koordinatsystemet. Vilket gör att själva X axeln inte längre går rakt ut från robotarmen (som det högra X'et på bilden ovan visar), utan X får en vridning alfa såsom Z får en vridning beta, vilket skapar en tredimensionell överblick av objektet.

Robotarmens origo är från början uträknat från punkten R, vilket innebär att den måste kompenseras så att origo istället är på den roterande plattan. Detta leder till formeln $A - X_1 = X_0$, som bilden ovan visar. A står för avståndet från centrum på R till centrum på den roterande plattan. X_1 är den beräknade X-punkten som vi fick tidigare i detta kapitlet (Xpos), dvs hur långt fram armen är i X led. X_0 representerar då med hjälp av alfa- och beta-vinkeln punkten X och Z för robotarmen. Genom att beräkna vinklarna får man att:

- X = $X_0(\alpha) \times \cos(\alpha)$
- Z = $X_0(\beta) \times \sin(\beta)$

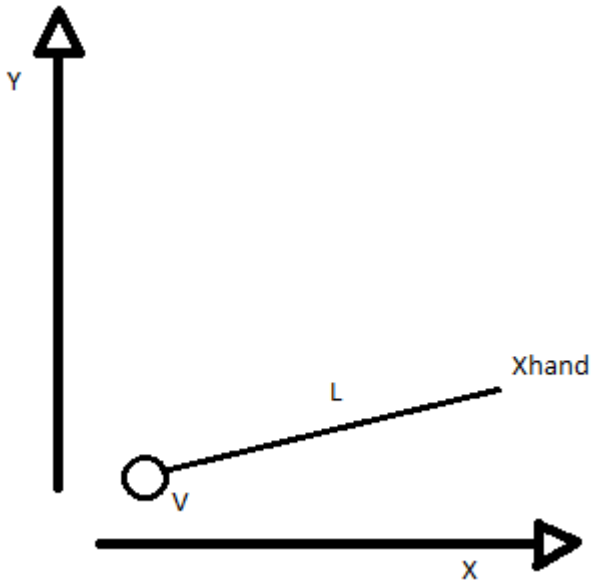
Y-punkten får man som sagt ut genom Forward Kinematics som visades tidigare. På så vis har man nu fått ut X-,Y- och Z-punkten för robotarmen. Detta kan ses i Appendix C på formlerna 6, 7 och 8.

Tabell 4: Förklaring av BaseDegrees.

Namn	Förklaring
BaseDegrees	Detta är hur mycket våran roterande platta som är positionerad framför robotarmen är vriden i grader.

4.2.2 Inverse Kinematics

Nu när vi har fått ut grundpositioneringen för robotarmen och mjukvaran, så ska även mjukvaran kunna skicka koordinater som robotarmen ska kunna besöka. För att detta ska uppnås behöver vi nu använda oss utav Inverse Kinematics. Detta innebär att vi nu får en (X,Y,Z) punkt och vill göra om detta till grader för alla axlarna på våran robotarm.



Figur 4.17: Exempel av robotarm med en axel.

I bilden ovan har vi en arm med längden L och en led med vinkeln V, samt positionen på robotens arm beskriven som Xhand. För att då svara på frågan i inverse kinematics börjar vi med att skriva upp:

Formel 9: Beräkning av vinkel V.

$$X_{hand} = L \times \cos(V)$$

$$\cos(V) = \frac{X_{hand}}{L}$$

$$V = \cos^{-1}(\frac{X_{hand}}{L})$$

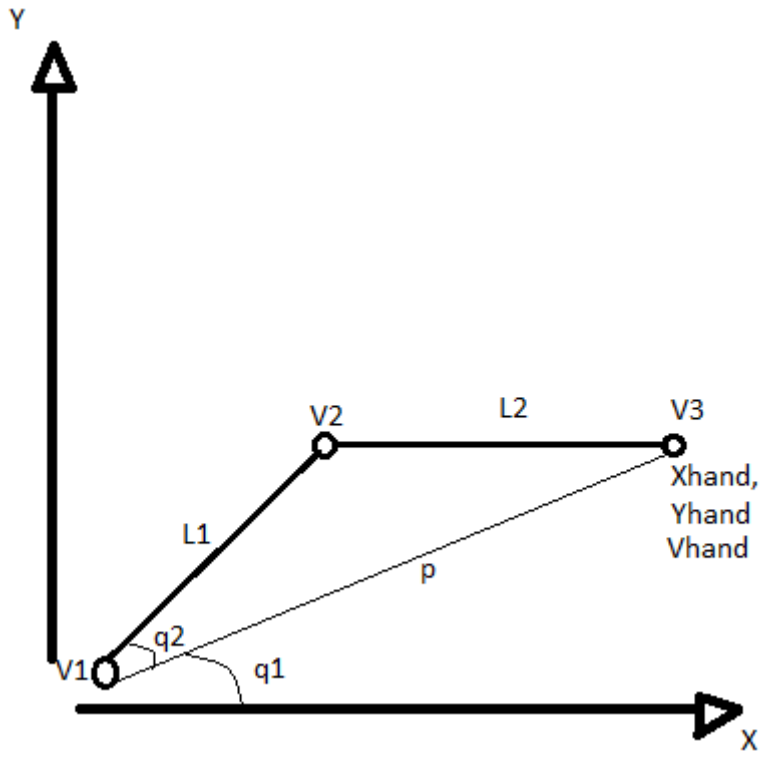
För att slutföra detta låt oss säga att armen har en längd på 1cm och vi vill att handen ska befinna sig i X = 0.72cm. Vi får då:

$$V = \cos^{-1}(0.72) = \pm 45°$$

Som vi ser här får vi även på ett mycket enkelt exempel två typer av lösningar, en med +45 och en med -45 grader. Som tur var så finns det redan fördefinerade funktioner för att lösa trigonometriska problem i de flesta programmeringsspråk och i vårt fall hade man kunnat använda sig av ATan2 som med hjälp av givna x- och y-värden hittar den rätta kvadranten.[12]

Problemet ovan är dock mycket simpelt formulerat och vi tittar således vidare på ett exempel med ytterligare en axel.

[12] Websida 12



Figur 4.18: Exempel på robotarm med 2 axlar.

Vi ser direkt att detta exempel blir betydligt mer komplext. I bilden ovan har en imaginär linje, p dragits. Givna varibaler är X_{hand}, Y_{hand}, samt V_{hand} och söker nu vinklarna V1, V2 och V3.

Till att börja med får vi ur pythagoras sats att $p^2 = X_{hand}^2 + Y_{hand}^2$

Genom vanliga trigonometriska beräkningar kan vi sedan få fram våra övriga parametrar.

Formel 10: Beräkning av vinklarna V1, V2 och V3

$$q1 = Atan2\left(\frac{Y_{hand}}{X_{hand}}\right)$$

$$q2 = \cos^{-1}\left(\frac{(L1^2 - L2^2 + p^2)}{2 \times L1 \times p}\right)$$

$$V1 = q1 + q2$$

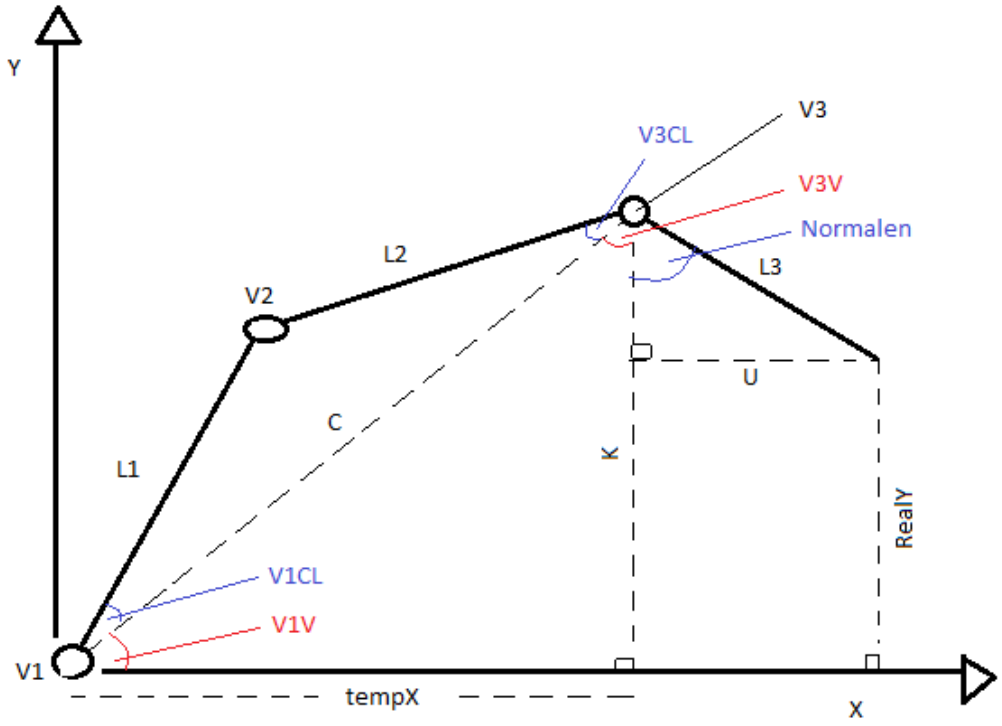
$$V2 = \cos^{-1}\left(\frac{L1^2 + L2^2 - p^2}{2 \times L1 \times L2}\right)$$

$$V3 = V_{hand} - V1 - V2$$

Som vi märker komplicerade ytterligare en arm formeln betydligt. Eftersom vårat mål byggde kring en arm med tre axlar plus en separat roterande motor gör detta att uträkningen som krävs blir extremt komplex. Möjligheten finns att armen hittar flera, ibland oändligt många lösningar. Det kan även inträffa att det inte finns några lösningar alls, då föremålet i fråga kan tänkas hamna utanför rörelseytan för armen.

När man ska beräkna rörelsen för tre eller fler axlar blir denna typ av

beräkningar således ofta för avancerade och man övergår vanligtvis därför till att utföra dessa algebraiskt.[13] I vårt fall fanns dock möjligheten att fortsätta på tidigare metod. Detta eftersom 3D-uppritningsprogrammet har möjligheten att beräkna normalen för den sista armen i förhållande till objektet, vilket ger oss en vinkel att arbeta utefter. Denna vinkel förenklar processen nått enormt och möjliggör uträkningen nedan.



Figur 4.19: Vinkelberäkning av robotarm med 3 axlar.

Precis som tidigare söker vi vinklarna V1, V2 och V3.

Normalen, X, och Y ges ur programmet för 3D-uppritning.

Eftersom fallet finns då normalen är 0, dvs att L3 står rakt nedåt har vi valt att dela upp formeln i två fall, ett för normalen = 0 och ett för övriga fall. Vinkelberäkningen för figur 4.19 ovan kan ses i Appendix C formel 11.

Tabell 5: Förklaring av beteckningar för figur 4.19.

Namn	**Samma som**	**Förklaring**
RealX		X-värdet från robotarmens origo
RealY	Y	Höjden
LengthShoulder	L1	Första armens längd
LengthElbow	L2	Andra armens längd
LengthWrist	L3	Tredje armens längd

[13] Bok 14

Nu har vi fått ut allt som krävs för att veta alla vinklar. Det sätt vi valt att montera motorerna gör dock att vi inte kommer åt alla vinklar på bilden, utan nolläget för V2 samt V3 befinner sig 90 grader över det egentliga nolläget. Detta måste således kompenceras för (Appendix C, formel 12).

Eftersom vi även hade en roterande motor, med ett z-plan behöver även denna bestämmas. Ur forward kinematics hade vi att Z = $X_0(\beta) \times \sin(\beta)$

Denna formel är som synes mycket simplare och kunde enkelt vändas då vi nu har Z och X vilket ger följande formel:

Formel 13: Beräkning av vinkeln för z-planet.

$$V4 = \sin^{-1}\left(\frac{Z}{LengthBetween - X}\right)$$

När man nu beräknat de fyra vinklarna kan man då förflytta armen till dessa punkter.

4.2.3 Denavit-Hartenberg

Detta var en metod som vi aldrig använde oss av i praktiken, utan använde oss om för att införskaffa bättre förståelse kring uträkningarna för robotens rörelse. Detta är en teori som skapades på 50-talet för att bevisa att alla transformationer mellan två leder i en robot kräver fyra stycken parametrar. Dessa är numera kända som D-H parametrar och används än idag för att beskriva en robots geometri. Här har vi en enkel förklaring för de olika parametrarna:

Tabell 6: Förklaringar av D-H-parametrar.

Symbol	Förklaring
a	Det vinkelräta avståndet mellan två leder som har en gemensam axel. Detta avstånd kallas x-ledet.
α	Den relativa vridningen mellan två leders axlar mätta kring deras gemensamma vinkelräta axlar.
d	Avståndet mellan två vinkelräta axlar mätta längst med ledaxeln.
Θ	Ledens vinkel längs z-axeln mellan två vinkelräta axlar.

Att lära sig detta fullt ut är dock en del av högre grads robotprogrammering och vi kommer således inte gå djupare in på detta. Så fort man löst ut alla dessa parametrar kan man sedan placera in dessa i formeln för varje led som ser ut på följande sätt[14]:

Formel 14: Formel för transformationen i en robotarm.

$$\text{Transformation} = \text{Screw}_X(a, \alpha)\text{Screw}_Z(d, \Theta) = \text{Trans}_X(a)\text{Rot}_X(\alpha)\text{Trans}_Z(d)\text{Rot}_Z(\Theta)$$

[14] Bok 13

4.3 Programmering

Detta stycke tar inte upp all kod som använts, utan fokuserar på de viktiga funktionerna så att läsaren kan få en bättre inblick av systemet. Vi använde oss av Arduino som består av C Programmering. För felsökning och testning använde vi oss av mjukvaruprogrammet Hercules som emulerar en seriell comport där vi kan skicka data på.

4.3.1 Styrning av motorer med potentiometrar

Vi arbetar med fyra stycken motorer (tre servomotorer och en stegmotor) för att kunna fullfölja vårat mål. För att kunna styra dessa med potentiometrar var vi först tvungna att definiera till vilken inport på Arduino Uno som utsignalerna från potentiomterarna är kopplade till. Efter det så styrs dem genom funktionen void MoveServos() genom att man läser in data (0 till 1023) från den analoga porten som potentiometern är kopplad till. Sedan mappar man om det värdet till ett värde mellan 0 och 179, vilket motsvarar 180grader. På så vis kan man nu använda sig av Servo.h biblioteket och skriva ut vinkeln till armen och så flyttas den dit. Detta kan göras för alla tre motorerna på robotarmen. Dock så är det lite annorlunda för stegmotorn, eftersom stegmotorn använder sig av ett eget bibliotek i Arduino som heter Stepper.h. Samtidigt som vi vrider stegmotorn måste vi hålla koll på hur mycket grader den har vridits totalt. Detta sker genom en omvandling då vi vet att våran stegmotor har 400 steg i sig, och varje steg motsvarar 0.9grader, så ett fullt varv blir 360grader. Utöver detta så har vi en switch för att kunna stänga av dessa potentiometrar så att de inte skriver över en ny koordinat som skickas till armen. Därför använder vi en if-sats som kollar om våran switch släpper igenom 5V eller inte, om den släpper igenom så är potentiometrarna aktiverade.

4.3.2 Skicka kalibreringskoordinater

En del av uppgiften med vårat arbete består av att skicka tre punkter som består av x,y och z från ett objekt till ett mjukvaruprogram som tar emot dem för att kalibrera sitt program emot vårat. Så att dom kan se en bild av objektet på datorn vid samma position som the reella objeket har. Detta sker genom funktionen void CAL_REQ(), där det skickas en sträng på den seriella comporten: REQ_CAL. Vårat program lyssnar då efter information på comporten. När vårat program tar emot data kollar den om strängen den tog emot är lika med samma ord. Om det är fallet så skriver vi ut data på comporten i detta formatet: CAL;X;Y;Z; med hjälp av Serial.print commandot. Detta upprepas tre gånger fast med olika punkter på objektet för att slutföra kalibreringen.

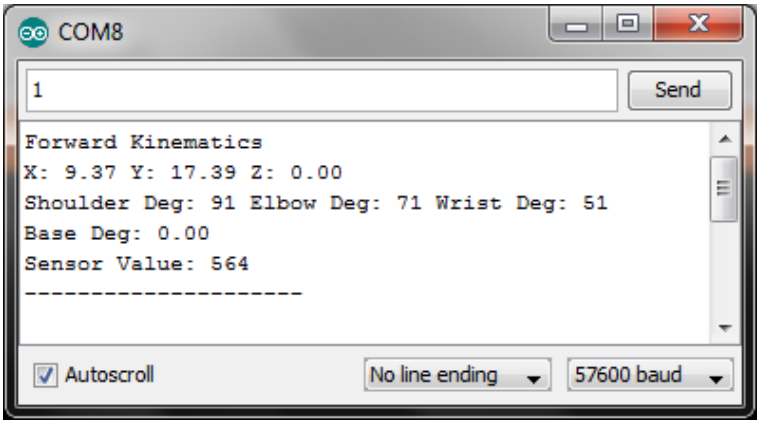
4.3.3 Besöka en mottagen koordinat

För att kunna besöka en mottagen punkt var vi först tvungna att koppla in en switch som avgör om potentiometrarna ska vara aktiverade eller inte. Om de är aktiverade så innebär det att vi inte kommer kunna vrida axlarna genom C-kod då själva potentiometrarna skriver över varje försök att ändra vinklarna. Därför måste man stänga av dem när man ska besöka en xyz-punkt. Detta görs genom en ganska enkel if-sats som kollar om man får in 5V i inporten analog5. För att besöka en xyz-punkt så måste man först göra om punkten till grader för de fyra motorerna. Efter det så flyttas de tre servo motorerna till vinkeln som behövs

genom kommandot Servo.write(degrees), där degrees är en grad mellan 0 och 179. Stegmotorn kräver en lite annorlunda variant (void MoveBase). Här skickar man in en grad som omvandlas till hur många steg som stegmotorn behöver ta för att uppnå den graden. Den kompenserar även för den graden som man för nuläget är på. Till exempel om man är på grad 45 på stegmotorn, och skickar in grad 90 till den. Så kommer den flyttas 45grader så att den totala graden blir 90.

4.3.4 Serial Monitor Interface

För att kunna provköra de olika motorerna och verkligen se så att uträkningarna har blivit rätt, skapade vi ett interface i Arduinos seriella com-port, vilken kan ses i figur 4.20 nedan. I tabell 7 kan de olika funktionerna för denna ses.



Figur 4.20: Arduinos Com-port.

Tabell 7: Funktioner för att kontrollera våra beräkningar.

Funtion	**Input Commando**	**Information**
void ForwardKinematics()	1	Skriva ut X,Y,Z punkten för robotarmen från kända vinklar på de fyra motorerna.
void MoveBase()	2 grad	Ställa in stegmotorn (roterande plattan) på vilken grad man vill. Där grad är ett nummer mellan 0-360. Nya graden på motorn sparas då i en variabel som heter BaseDegrees.
void InverseKinematics()	3	Skriva ut graderna på de fyra motorerna från en känd koordinat X,Y,Z.
void MoveTo(float X,	4	Flytta basplattan till en X,Y,Z

float Y, float Z)		punkt. Den påverkas dock bara av X och Z.
void MoveXYSer-vos(float V1, float V2, float V3)	5	Ändra vinkel V1,V2 och V3 för robotarmen.
void ReadSensor(int sensEnabled)	6	Skriver ut värdet på den magnetiska sensorn.
void MoveServos()		Denna samarbetar med ForwardKinematics då man först vrider motorerna till en position, och sedan skriver 1 i konsolen. Då får man ut X,Y,Z punkten för de graderna.

5. Resultat

5.1 Utveckling

Utvecklingen av den materiella produkten som i vårat fall är en plattform med en robotarm monterad på den visade sig ge ett bra resultat. Enligt kapitel 4.1 så lyckades vi montera ihop en köpt robotarm men även konstruera om den för att möta våra behov. Istället för att ha alla motorer monterade på varandra valde vi att ha basmotorn framför robotarmen för att rotera objekt. Kretskortet och våran Arduino Uno har även placerats på ett avstånd sådant att robotarmen aldrig kan stöta emot det och skapa problem.

5.2 Kalibrering

Vi lyckades även skapa ett kalibreringsprogram, förklarades i kapitel 4.2.1 och 4.3.1, sådant att mjukvarugruppens program kan ta emot tre punkter på ett objekt. Som tidigare förklarat visste vi bara om graderna på våra fyra motorer när vi skulle beräkna en xyz-punkt. Med hjälp av mycket forskning kring Forward Kinematics lyckades vi till slut härleda tre formler för uträkning av x,y och z från våra vinklar. Detta är avgörande för att mjukvaruprogrammets koordinatsystem skall kunna kalibreras med de reella systemet, så att både det virtuella och reella objektet hamnar på samma punkter. Då själva målet med projektet var att besöka punkter som skickas till oss så hade detta inte fungerat om inte båda bilderna av objektet var synkroniserade. För att kunna beöka punkter har vi forskat inom Inverse Kinematics som beskrivs i kapitel 4.2.2. Omvandligt från en xyz-punkt till grader för våra fyra motorer är något som kräver en väldigt hög matematisk kunkap, men vi har kunnat kringå de mest avancerade beräkningarna genom att utnyttja funktionen att mjukvaran kan skicka ett normalvärde för den sista axeln och således även kunnat härleda formler för att lösa ut dessa vinklar.

5.3 Sensorer

I hårdvarudelen, kapitel 3.2.11, beskrivs den sensorn vi har arbetat med för att kunna läsa av ytor på det reella objektet. Från början hade vi lite kunskap om sådana komponenter men det visade sig vara betydligt enklare än vad vi först hade trott. Den magnetiska sensorn är även monterad på ett sådant sätt att den smidigt kan bytas ut mot en annan sorts sensor om så skulle vara önskat, men för kod för andra sensorer finns inte implementerade i vårat program.

5.4 Verifering av problemställning

Arbetet för att lösa våran problemställning bestod inte bara utav konstruktion och matematiska lösningar, utan även av att genom mjukvara programmera så att alla delarna i skapelsen kunde sammarbeta för att nå önskat resultat. När vi väl hade fått ut våra formler för uträkning av punkter kunde dessa någorlunda enkelt överföras till C-kod. Utöver det så lyckades vi även styra alla fyra motorer och med en felmarginal på ungefär 2% flytta dom till de grader som vi önskade. Våran problemställning har genom detta verifierats på så vis att det går att skapa en funktionell prototyp för avläsning kring tredimensionella objekt med beskrivna metoder.

6. Slutsatser

6.1 Bakgrund till problemställning

Vår uppgift var att tillverka en robotarmprototyp som manuellt ska kunna styras med hjälp av spakar för att kunna ställa in robotarmen att peka på ett objekt. Anledningen till detta är att objektets position måste kalibreras med ett mjukvaruprogram så att den står på exakt samma position i ett xyz-koordinatsystem. När kalibreringen är färdig ska ett mjukvaruprogram, skapat av en annan examensarbetesgrupp, kunna skicka en punkt i koordinatsystemet som pekar på objektet och då ska robotarmen flyttas och peka på den punkten. Utöver det ska vi även tillverka skapa ett smidigt sätt att montera en sensor på robotarmens topp för att kunna läsa av värden från objektet. Anledningen till varför vi valt att göra detta kan läsas om i kapitel 1.3, men består till största del av att det är ett utmanande projekt och tekniken för detta är förhållandevis outvecklad, men potentiellt mycket viktig i framtiden. Själva tillverkningen av prototypen innefattar att först bygga en robotarm och koppla ihop den så att den kan styras av spakar. Därefter ska man genom C-programmering kunna skicka punkter till ett mjukvaruprogram på datorn för att kalibrera systemet (kapitel 4.3.2). Slutligen ska man kunna ta emot koordinater från programmet och robotarmen ska då automatiskt flytta sig till dessa punkter.

6.2 Utvecklingsmöjligheter

De slutsatser vi kan dra efter 10 veckors arbete är att konstruktionen inte blev så stabil som vi önskat. Dels på grund av kvalitén på motorerna men till väldigt stor del på grund av oss själva. Våra kunskaper innan projektet var väldigt begränsade inom elektriska kretsar, men genom arbetets gång har vi upptäckt och lärt oss saker som egentligen vill få oss att bygga om strukturen på kretskorten och sladdarna, något som dock tidsbristen förhindrar.
Önskar man färdigställa en fungerande slutprodukt av denna prototyp är det att rekomendera en större investering, då starka och stabila motorer är en förutsättning för att systemet ska vara tillförlitligt. Även lödning av alla trådar torde styrka systemet ytterligare. Vi kan nu i efterhand se att robotarmen tagit en hel del skada under testningen, pga hopstötningar med plattformen. Detta påverkar givetvis positioneringen negativt och bidrar till att vi har en felmarginal på motorerna.

6.3 Beräkningsanalys

De matematiska formlerna för att räkna ut graderna på våra fyra motorer av känd xyz-punkt visade sig vara extremt avancerade och för att lösa dessa var vi tvugna att kringå problemet genom att utnyttja funktionen för normalen i mjukvaruprogrammet. Den slutsats vi kan dra utav detta är att formeln endast är giltig då man känner normalen och således inte kan tillämpas i ett system med endast kända x-,y- och z-koordinater. Problemet med vårt tillvägagångssätt är att vi aldrig fick veta om den önskade punkten som ville nås fanns inom robotens arbetsyta. När fall dyker upp som roboten inte kan nå, förblir roboten

stillastående, vilket helt enkelt beror på att inskickade argument är felaktiva för cosinus lagar, som resulterar i att alla vinkelberäkningar skrivs ut som noll.

6.4 Arbetsgenomgång

Programmeringen visade sig, till skillnad från våra förväntningar, vara det enklaste momentet. Arduino består av många färdiga bibliotek och är en väldigt smidig utvecklingsmiljö för både erfarna programmerare som nybörjare.
Eftersom projektet byggde mycket kring sammarbete med andra grupper kan vi dra slutsatsen att projektet blev lidande pga dålig kommunikation. Hade vi som grupp kommit överens om en gemensam arbetsmetod redan från start hade mycket tid kunnat sparas på detta.
Sammanfattningsvis anser vi oss nöjda med det uppnådda resultatet, även om det finns mycket detaljer att arbeta vidare kring. Efter slutfört projekt har lärt oss enormt mycket inom elektronik och c-kodning, samt även ökat våra trigonometriska kunskaper och förståelse kring positionering av tre-dimensionella objekt.

6.5 Kritisk Diskussion

Trots att vi lade upp en planering i början av arbetet, hade vi stora problem att följa denna. Oväntade problem dök ständigt upp, vilket ledde till att vi gled ifrån vår ursprungliga plan och började leta efter nya sätt att tackla uppgiften på.
Trots att vi relativt tidigt in i projektet fick tillgång till AL5A robotic kit, lade vi en stor del av tiden till att försöka konstruera egna armar och lösningar. I efterhand var detta ett felbeslut och skulle vi nu få chansen att göra om projektet med den kunskap vi nu har, hade vi insett att det effektivaste hade varit att ta hjälp av de färdigbyggda armar och motorer vi hade tillgång till istället för att själva försöka lösa prolemen med onödigt komplicerade metoder.

Vi har nu insett att det är viktigt att testa alla kopplingar innan vi spänningsätter dom, för att på så sätt slippa onödiga kortslutningar och långa felsökningar, något som skedde frekvent under själva konstruktionen.

När man utvecklar en prototyp i samarbete med många andra människor och grupper är det viktigt att ha en bra kommunikation, samt att skriva ner utsatta mål och metodik innan arbetet påbörjas. Detta var till en början lite oklart då vi inom grupperna arbetade efter olika tillvägagångssätt och en hel del tid fick läggas på felsökning av småsaker enbart pga att kommunikationen oss emellan hade brustit.

Vi har under projektets gång stött på en mängd begränsningar som vi inte såg från början. Dessa har inte stoppat oss, utan snarare minskat precisionen och kvalitén på prototypen. Det första vi märkte när armen levererades var att den var mindre än förväntat. Måtten på armarna på AL5A är 9.5cm, 10,5cm samt 8,5cm vilket ger den en total längd på 28.5cm. Detta begränsar självfallet storleken på objekten som man kan mäta. Även vissa av servomotorerna som medföljde armen är av en billigare typ och har därmed inte samma precision som lite dyrare varianter. Detta problem uppstod även i potentiometrarna som senare kopplades in, då störningar ledde till att armen vibrerar. Detta motverkades till

viss del med hjälp av en kondensator, men alla störningar gick ej att få bort. Med mer tid hade vi kunnat bygga om konstruktionen av sladdar för att skapa en stabilare miljö där störningar inte existerade.

Under delen av utvecklingen då vi själva försökte konstruera en egen arm från grunden var materialbrist en stor begränsning. Mycket av arbetet baserades på improviserade metoder av det material vi kunde hitta. Detta var den mest omfattande begränsningen som slutligen ledde till att vi fick avbryta hela idén kring en egenbyggd arm, då materialet vi hade tillgång till helt enkelt inte var stabilt nog och inte fyllde våra behov. Alla dessa begränsningar anser vi dock som acceptabla eftersom projektet endast avser utvecklingen av en prototyp och inte en färdig produkt.

Då vi startade projektet insåg vi inte till fullo hur avancerad matematik det skulle krävas för att kunna positionera armen på önskvärt sätt. Trots mycket hjälp från lärare på matematiska institutionen, kunde dessa uträkningar till fullo aldrig lösas ut för positionering av 3 axlar. Tillvägagångssättet för att lösa denna typ av uträkningar tas dock upp, och eftersom problemställningen aldrig sätter som krav att lösningen måste göras för ett system med 3 axlar, anser vi även denna brist som acceptabel. Alternativa lösningar för problemet presenteras så tillvida att den ursprungliga problemställningen utifrån dessa kan lösas.

Alla dessa problem har dock lärt oss betydligt mycket och hade vi nu haft chansen att göra om projektet hade vi valt att direkt fokusera på en större modell av robotarm, med starkare motorer. Vi hade sett till att ha möten med grupper vi sammarbetade med varje vecka för att stämma av så att allt gick som planerat, samt sett till att skapa en genomtänkt planering, som vi lätt kunnat följa.

6.6 Fortsatt Forskning/Framtid

Fortsatt forskning av produkten hade inneburit att den inte längre ska vara en prototyp, utan kunna användas av företag och privatpersoner inom arbetsmarknaden.

Vid fortsatt utveckling av produkten hade vi kunnat förbättra hela konceptet, dels genom att lägga mer tid på kretsarna, men även för att öka funktionaliteten i robotarmen genom C programmering. Störningar i systemet hade varit tvunget att elimineras. För att uppnå det hade vi behövt koppla om alla kablarna och möjligtvis skapa egna kort där man satte fast dem i. Utöver det så hade det krävts att vara mer noggrann vid mätning av längder och avstånd. Då en liten felmätning kan ge ganska stor effekt på armens vinkelgrader. Detta gäller även programmeringsmässigt, då man inte får avrunda för tidigt och försöka vara mer exakt för att kunna få ut rätt positioner.

Inverse Kinematics är en del som hade krävt mycket arbete. För att kunna få ut graderna på fyra motorer från en xyz-punkt hade man behövt räkna och testa väldigt mycket för att uppnå ett önskat resultat. Problemet bakom uträkningen för det ligger i att man har fyra okända vinklar som man vill få ut av tre värden. Dock är det extremt komplicerat att bara försöka få ut en av dessa. Utöver det så finns det oändligt många svar på uträkningen då armen kan positionera sig i

flera olika vinklar för att ändå uppnå samma punkt. Därför hade man för ett företag varit tvungen att arbeta ut en punkt sådan att armen vet att den kan komma dit utan att stöta i något objekt på vägen. Dessa steg hade varit obligatoriska om vi ville att produkten skulle användas för ett företag och inte längre enbart vara en prototyp.

Användningsområdet för en robot som kan scanna av objekt och läsa av mätdata är ganska bred. Det kan användas vid tillverkning av föremål, då man till exempel har en prototyp vars värden man vill läsa in till en dator för vidare produktion. Det kan även användas för att läsa av värden på objekt som inte människan kan komma åt, till exempelvis för att läsa av radioaktivitet på föremål som annars hade varit skadligt för människan.

Systemet hade även kunnat utvecklas sådant att roboten först scannar objektet och på så vis vet hur långt avstånd det är till föremålet. Efter det röra sig framåt och hela tiden läsa av objektet och veta vart den kan och inte kan gå.

En potentiell vinst på fortsatt utveckling av en sådan här produkt är väldigt svårt för oss att förutspå, men användningsområdet för det är någorlunda stort och om bara produkten utvecklas och anpassas efter företags behov, så finns det förmodligen en marknad för det.

Källförteckning

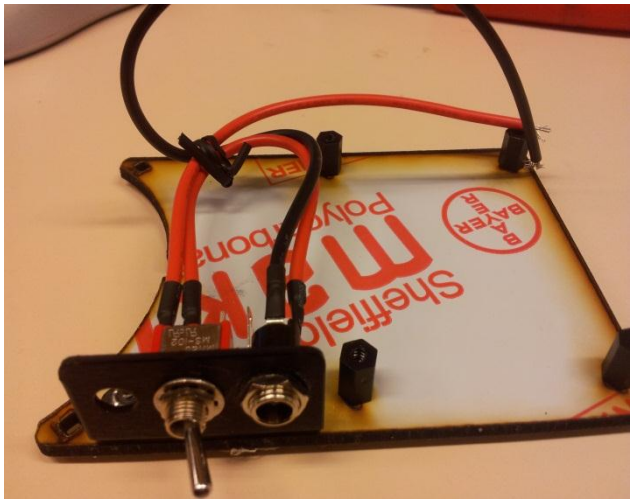
Websidor

1. *www.3d3solutions.com* (2012). Hämtat från http://www.3d3solutions.com/products/flexscan3d/ [2012]
2. *www.arduino.cc* (Juni, 2012). Hämtat från http://arduino.cc/en/Main/ArduinoBoardUno [2012]
3. *www.arduino.cc* (November 18, 2012). Hämtat från http://arduino.cc/en/Main/ArduinoProtoShield [2012]
4. *www.hw-group.com* (2012). Hämtat från http://www.hw-group.com/products/hercules/index_en.html [2012]
5. *www.lynxmotion.com* (2010). Hämtat från http://www.lynxmotion.com/p-395-ssc-32-servo-controller.aspx [2012]
6. *www.wikipedia.org* (Maj 22, 2012). Hämtat från http://en.wikipedia.org/wiki/Arduino [2012]
7. *www.wikipedia.org* (Maj 24, 2012). Hämtat från http://en.wikipedia.org/wiki/C_%28programming_language%29 [2012]
8. www.alldatasheet.com (2012). Hämtat från http://www.alldatasheet.com/datasheet-pdf/pdf/228212/BOURNS/PEC11.html [2012]
9. www.servocity.com (2012). Hämtat från http://www.servocity.com/html/hs-422_super_sport_.html [2012]
10. www.datasheetcatalog.com (2012). Hämtat från http://www.datasheetcatalog.org/datasheet/SGSThomsonMicroelectronics/mXrqqxz.pdf [2012]
11. www.learnaboutrobots.com (2012). Hämtat från http://www.learnaboutrobots.com/forwardKinematics.htm [2012]
12. www.learnaboutrobots.com (2012). Hämtat från http://www.learnaboutrobots.com/inverseKinematics.htm [2012]

Böcker

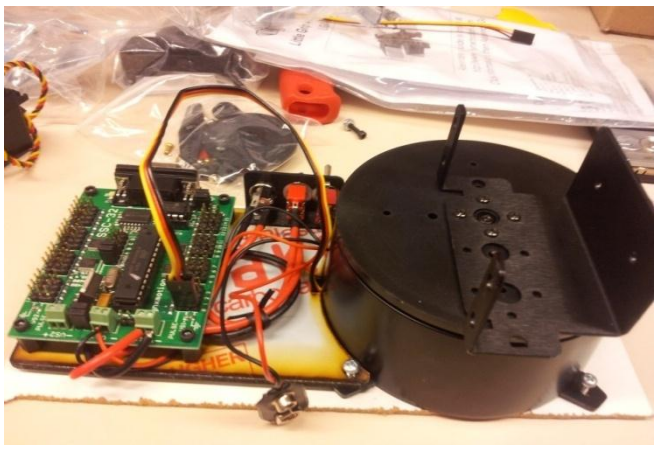
13. Paul, Richard (1981). *Robot manipulators: mathematics, programming, and control : the computer control of robot manipulators*. MIT Press, Cambridge, MA
14. J. M. McCarthy and G. S. Soh, 2010, *Geometric Design of Linkages,* Springer, New York.

Appendix A

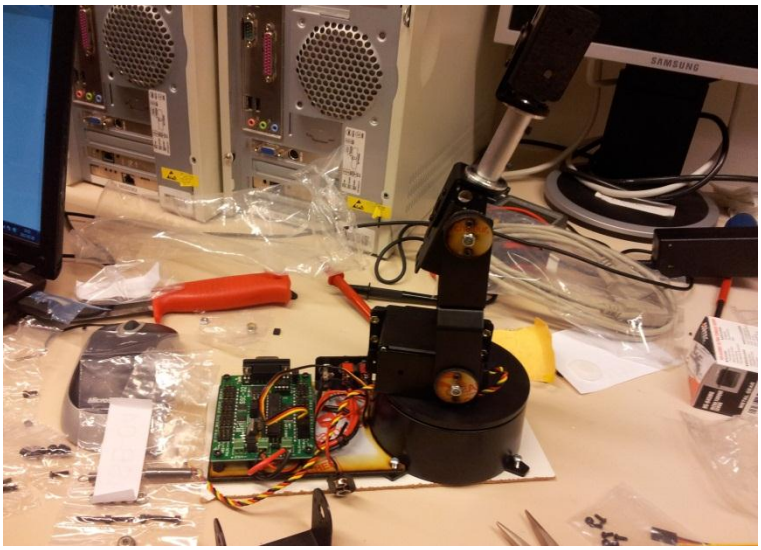


Figur 4.1: start på robotarmen.

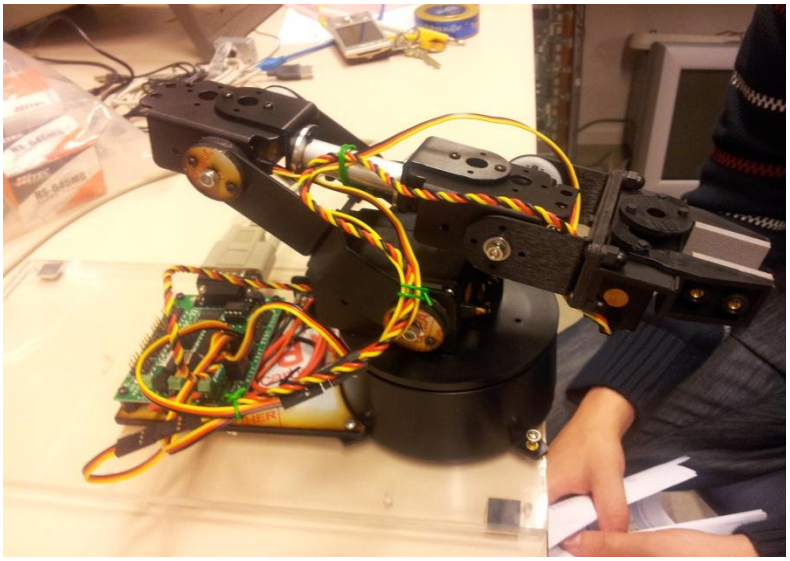
Figur 4.2: Roterande bottenplatta.



Figur 4.3: ssc-kretskortet påsatt.



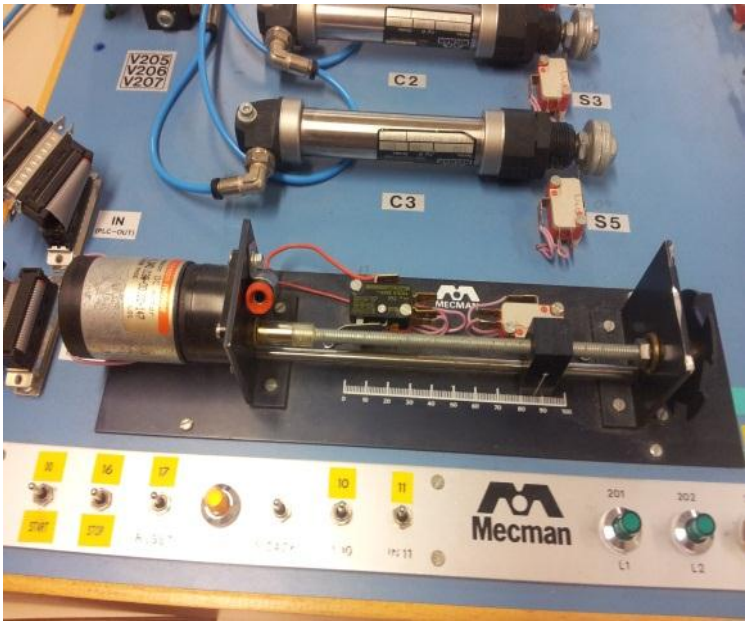
Figur 4.4: axlarna påsatta.



Figur 4.5: fullt monterad.

Figur 4.6: prototyp av hiss.

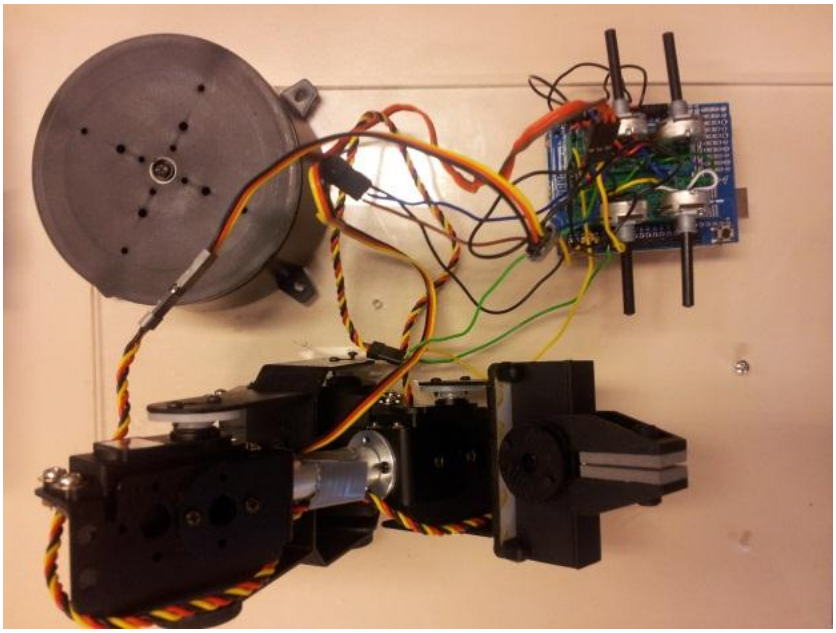
Figur 4.7: prototyp av fjäder-arm.



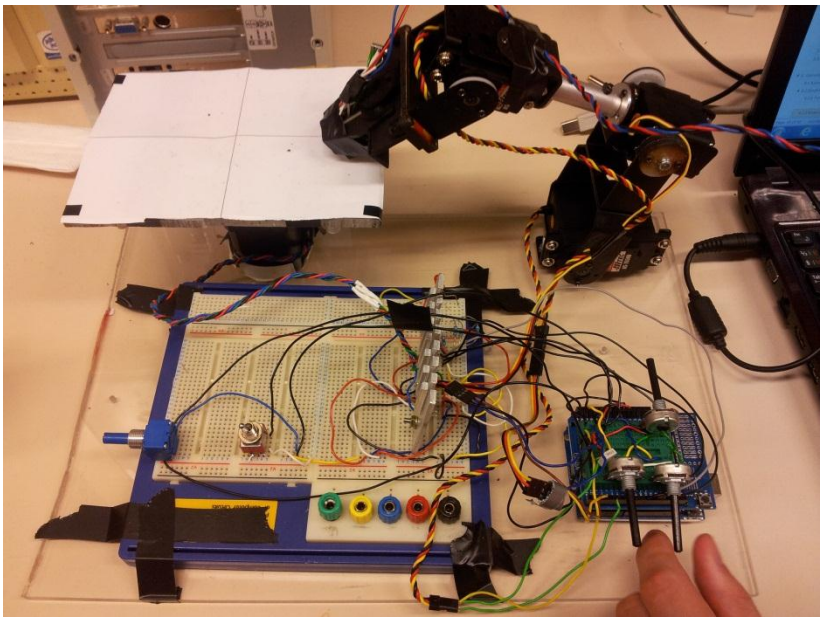
Figur 4.8: Tekniken som används i labbsal.

Figur 4.9: skrivare nedmonterad.

Figur 4.10: kopplingen med breadboard.



Figur 4.11: kopplingen med proto shield.



Figur 4.12: Den färdiga prototypen.

Moment	Vecka 1	Vecka 2	Vecka 3	Vecka 4	Vecka 5	Vecka 6	Vecka 7	Vecka 8	Vecka 9	Vecka 10
Analys										
Arduino										
Mechlab										
Lynxterm										
3D Scanner										
Flexscan 3D										
Robotarm										
Design										
Robotarm										
Sensor										
Konstruktion										
Robotarm										
Sensor										
Testning										
Robotarm										
Interface										

Figur 4.20: Produktens livscykel.

Appendix B

```
#include <Servo.h>
#include <Stepper.h>
//Antalet steg på våran stegmotor
#define STEPS 400
//Stegmotorn kopplas in i dessa portar
Stepper stepper(STEPS, 10, 11, 12, 13);
//Startar initialisering av servomotorerna
Servo ShoulderServo;
Servo ElbowServo;
Servo WristServo;
//De olika pinsen och gradvärdena på motorerna
int BasePin = 5;
int BaseVal;
int ElbowPin = 2;
int ElbowVal;
int WristPin = 3;
int WristVal;
int ShoulderPin = 4; //beofre 1
int ShoulderVal;
int sensorPin = A1;
//Används för att läsa på serieporten
String readString;
//Konstanta värden på alla längder som måste användas i uträkningarna
float LengthShoulder = 9.5;
float LengthElbow = 10.5;
float LengthWrist = 9.0;
float HeightBase = 7.0;
float LengthBetween = 20.0;
//Basplattan
float BaseDegrees;
//Våran magnetsensor
int switchPin = A0;
//Powerswitch
int isActive = 0;


void setup()
{
  //Startar seriporten på 57600 Baud, måste vara samma på javagruppens
program.
  Serial.begin(57600);
  //Här attachar vi servomotorerna till pins.
  ShoulderServo.attach(5);
  ElbowServo.attach(2);
```

```
  WristServo.attach(3);
  //Sätter stegmotorns roterande hastighet till 10.
  stepper.setSpeed(10);
  //När programmet startas får den roterande plattan graden 0.
  BaseDegrees=0;
}

void loop()
{
   //Här aktiverar vi eller inaktiverar potentiometrarna beroende på om man får
in 5V från switchen eller inte.
   double switchValue;
   switchValue = analogRead(switchPin);
   if(switchValue > 1000){ //5volt förs igenom därför blir den active. Active är den
när den är lika med 0.
     isActive = 0;
   }else{ //under 1000
     isActive = 1;
   }

   //Olika funktionera som loopas på.
   MoveServos();
   ReadInput(0);          //0 = Enabled
   CAL_REQ(1);            //0 = Enabled
   ReadSensor(1);         //0 = Enabled
}


void ReadSensor(int sensEnabled){
   if(sensEnabled == 0){
      double sensorValue;
      sensorValue = analogRead(sensorPin);//läs sensorvärdet
      Serial.print("Sensor Value: "); //skriv ut sensorvärdet
      Serial.println(sensorValue);
   }
}


void InverseKinematics(float X, float Y, float Z){
  //Här räknar vi ut hur vi får vinkarna på servomotorerna från en X och Y punkt.
     float R = (180/M_PI);
     float RealX = LengthBetween-X;
     float RealY = Y;
     float Normalen=0.0;
     float U = sin(Normalen/(180/M_PI)) * LengthWrist;
     float tempX = RealX-U;
     float K ;
```

```
    if(U == 0){
      K = LengthWrist + RealY;
    }else{
      K = sqrt(LengthWrist*LengthWrist-U*U)+RealY;
    }
    float C = sqrt(tempX*tempX+K*K);
    float V3V = asin(tempX/C)*R;
    float V1V = asin(K/C)*R;
    float V3CL = acos((LengthElbow*LengthElbow + C*C - LengthShoulder *
LengthShoulder)/(2*LengthElbow * C))*R;
    float V1CL = acos((LengthShoulder * LengthShoulder + C * C - LengthElbow
* LengthElbow)/(2*LengthShoulder * C))*R;
    float V2   =
acos((LengthShoulder*LengthShoulder+LengthElbow*LengthElbow-
C*C)/(2*LengthShoulder*LengthElbow))*R-90;
    float V3 = Normalen+V3CL+V3V-90;
    float V1 = V1V+V1CL;

    //Skriver ut de olika värdena på interfacet i arduino så man ser alla värdena
tydligt.
    Serial.print("U: ");
    Serial.print(U);
    Serial.print(" Tempx: ");
    Serial.print(tempX);
    Serial.print(" K: ");
    Serial.print(K);
    Serial.print(" C: ");
    Serial.println(C);
    Serial.print("V3V: ");
    Serial.print(V3V);
    Serial.print(" V1V: ");
    Serial.print(V1V);
    Serial.print(" V3CL: ");
    Serial.print(V3CL);
    Serial.print(" V1CL: ");
    Serial.println(V1CL);
    Serial.print("RealY=Y : ");
    Serial.print(RealY);
    Serial.print(" RealX: ");
    Serial.println(RealX);
    Serial.print("V1: ");
    Serial.print(V1);
    Serial.print(" V2: ");
    Serial.print(V2);
    Serial.print(" V3: ");
    Serial.println(V3);
```

```
    MoveXYServos(V1,V2,V3); //Flyttar XY motorerna (3servos).
    MoveTo(X,Y,Z); //Flyttar Z motorn.
}


void MoveXYServos(float V1, float V2, float V3){
  //Vrid servomotorernas vinklar till de nya uträknade vinklarna genom inverse kinematics
  //så delayar vi lite så de hinner flytta sig.
  ShoulderServo.write(V1);
  delay(55);

  ElbowServo.write(V2);
  delay(55);

  WristServo.write(V3);
  delay(55);
}



void MoveTo(float X, float Y, float Z){
    //Här flyttar vi den roterande basplattan ( i själva Z-ledet)
    //till den z punkten man får in gör man om det till en vinkel för motorn, som sedan ytterligare omvandlas
    //för att kunna användas till stegmotorn.

    float deg = 0;
    float yY = GetY();

    //Omvända formeln från forward kinematics. Har har man nu det kända G
    float zDeg = asin((Z/(GetInverseX(X))))*57.2957795;
    if(zDeg<0){
      zDeg=180-zDeg;
    }
    //Gör om från grader till steg.
    float StepNum = zDeg/0.9;
    //Stega motorn till nya graden.
    stepper.step(StepNum-(BaseDegrees/0.9));
    delay(2000);
    //Gör om man steg till grader för att spara den nya vinkeln som basmotorn ligger på.
    deg=StepNum*0.9;
    BaseDegrees = deg;
    Serial.print("Base Degrees: ");
    Serial.println(BaseDegrees);
```

```
}

void CAL_REQ(int recEnabled){//benämns enligt andra gruppens API
  if(recEnabled == 0){
     //Hämta koordinaterna.
     float RealX = GetX();
     float RealY = GetY();
     float RealZ = GetZ();
     readString = "";
     //kolla i serieporten, läs in från den.
     while (Serial.available()) {
       delay(10);
       char c = Serial.read();
       readString += c;
     }

    if (readString.length() >0) {
     //om serieporten hade REQ_CAL så betyder det att vi ska skicka
koordinaterna.
        if (readString == "REQ_CAL") {
          const int Max_Char = 50;
          //Skapa char array for de 3 punkterna.
          char thisX[Max_Char];
          char thisY[Max_Char];
          char thisZ[Max_Char];

          //lägg in koordinaterna i char arrayerna
          dtostrf(RealX,0,2,thisX);
          dtostrf(RealY,0,2,thisY);
          dtostrf(RealZ,0,2,thisZ);

          //skapar en ny chararray
          char Packet[Max_Char] = "";
          //I den lägger vi in datan i ett format som kan tas emot av den andra
gruppens javaprogram.
          strcat(Packet,"CAL");
          strcat(Packet,";");
          strcat(Packet,thisX);
          strcat(Packet,";");
          strcat(Packet,thisY);
          strcat(Packet,";");
          strcat(Packet,thisZ);
          strcat(Packet,";");
          //Nu skriver vi det till comporten och dom tar emot det.
          Serial.print(Packet);
        }
     }
```

```
  }
}

void MoveServos(){
//om isActive är lika med 1 är strömswitchen avständ på vårat kretskort och
servo motorerna kan inte styras med potentiometrar.
if (isActive == 0){
  double deg=0;
  double newNum;

    //här läser vi av potentiometrarna, mappar om värdet från vad de ger till
grader, som sedan kan skrivas till motorerna.
    ShoulderVal = analogRead(ShoulderPin);
    ShoulderVal = map(ShoulderVal, 0, 1023, 0, 179);
    ShoulderServo.write(ShoulderVal);
    delay(15);

    ElbowVal = analogRead(ElbowPin);
    ElbowVal = map(ElbowVal, 0, 1023, 0, 179);
    ElbowServo.write(ElbowVal);
    delay(15);

    WristVal = analogRead(WristPin);
    WristVal = map(WristVal, 0, 1023, 0, 179);
    WristServo.write(WristVal);
    delay(15);


    //Vi gör ungefär samma för bottenplattan
    //här kollar vi om potentiometerna är vriden under 70 grader eller över
130grader. I så fall roterar vi sakta stegmotorn.
    //om det är mellan 71 och 129 grader låter vi motorn stå still, i parkerat läge.
    BaseVal = analogRead(BasePin);
    BaseVal = map(BaseVal, 0, 1023, 0, 179);

    if(BaseVal > 130){
       newNum = 1;

       stepper.step(newNum);
       delay(100);

       deg=newNum*0.9;
       //åt ena hållet, men hjälp av plusstecknet.
       BaseDegrees = BaseDegrees+deg;
       Serial.println(BaseDegrees);
    }else if(BaseVal < 70){
       newNum = 1;
```

```
        stepper.step(-newNum);
        delay(100);

        deg=newNum*0.9;
        //åt andra hållet, men hjälp av minustecknet.
        BaseDegrees = BaseDegrees-deg;
        Serial.println(BaseDegrees);
    }else{
        //här är den i parkerat läge.
    }
    delay(15);
}
}

//**********************************//
//     Serial Monitor Interface     //
//**********************************//

void ReadInput(int Active){
  if(Active == 0){
    int temp = Serial.read();
    //här i lägger vi de olika metoderna som vi velat prova.
    if (temp == 49) { // 1
      ForwardKinematics(); //visa värden från forward kinematics
    }else if (temp == 50) { // 2
      MoveBase(); //rotera basplattan
    }else if (temp == 51) { // 3
      InverseKinematics(getX(),getY(),getZ()); //prova inversekinematics med de
nuvarande x,y,z värdena
    }else if (temp == 52){ // 4
      MoveTo(0,2,14); //flytta basplattan till ett x,y,z värde
    }else if (temp == 53){ // 5
      MoveXYServos(45,45,45); //flytta de tre servomotorernas axlar.
    }else if (temp == 54){ // 6
      ReadSensor(0);//skriv ut värdet på sensorn
    }
  }
}

//X värdet, från origo på den roterande bottenplattan
 float GetX(){
   float X = LengthShoul-
der*cos(ShoulderVal/(180/M_PI))+LengthElbow*cos((ElbowVal - (90-
ShoulderVal))/(180/M_PI))+LengthWrist*cos((WristVal - (90 - (ElbowVal - (90-
ShoulderVal))))/(180/M_PI));
   float xX = LengthBetween-X;
```

```
  xX = xX * cos(BaseDegrees/(180/M_PI));

  return xX;
 }

//X värdet, från origo på robotarmen
 float GetRealX(){
  float X = LengthShoul-
der*cos(ShoulderVal/(180/M_PI))+LengthElbow*cos((ElbowVal - (90-
ShoulderVal))/(180/M_PI))+LengthWrist*cos((WristVal - (90 - (ElbowVal - (90-
ShoulderVal))))/(180/M_PI));
  float xX = LengthBetween-X;
  return xX;
 }

 float GetInverseX(float X){
  float xX = LengthBetween-X;
  return xX;
 }

//Y värdet, är alltid samma. Påverkas inte av X och Z
 float GetY(){
  float yY = (LengthShoul-
der*sin(ShoulderVal/(180/M_PI))+LengthElbow*sin((ElbowVal - (90-
ShoulderVal))/(180/M_PI))+LengthWrist*sin((WristVal - (90 - (ElbowVal - (90-
ShoulderVal))))/(180/M_PI)))-HeightBase;
  return yY;
 }

//Z värdet.
 float GetZ(){
  float zZ = GetRealX();
  zZ = zZ * sin(BaseDegrees/(180/M_PI));
  return zZ;
 }


void ForwardKinematics(){
  //hämtar koordinaterna genom forwardkinematics. Och skriver ut dem i
intefacet.
  float xX = GetX();
  float zZ = GetZ();
  float yY = GetY();

  Serial.println("Forward Kinematics");
  Serial.print("X: ");
  Serial.print(xX);
```

```
  Serial.print(" Y: ");
  Serial.print(yY);
  Serial.print(" Z: ");
  Serial.println(zZ);
  Serial.print("Shoulder Deg: ");
  Serial.print(ShoulderVal);
  Serial.print(" Elbow Deg: ");
  Serial.print(ElbowVal);
  Serial.print(" Wrist Deg: ");
  Serial.println(WristVal);
  Serial.print("Base Deg: ");
  Serial.println(BaseDegrees);
  Serial.print("Sensor Value: ");
  Serial.println(analogRead(sensorPin));
  Serial.println("---------------------");
}

void MoveBase(){
  //Används för att rotera den nedre plattan som är framför armen genom
interfacet.
  double extraVal;
  double deg=0;

  readString = "";
     //loopar igenom allt man skrev i terminalen och tar ut hur många grader som
den ska vridas
     while (Serial.available()) {
       char c = Serial.read();
       readString += c;
       delay(10);
     }

     //Omvandla strängen till en char array så att den efter det akn göras om till
en double så att stegmotorn kan ta emot den.
     char test[readString.length()+1];
     readString.toCharArray(test,readString.length()+1);
     double newNum = atoi(test)/0.9;

     //stegar motorn, omvandlas till steg/grad.
     stepper.step(newNum-(BaseDegrees/0.9));
     delay(2000);

     //sparar det i BaseDegree så vi kommer ihåg vad motorn står på i grader.
     deg=newNum*0.9;
     BaseDegrees = deg;
     Serial.flush();
}
```

Appendix C

Formel 2: Beräkning av Xpos.

$$Xpos = L1 \times \cos(V1) + L2 \times \cos\big(V2 - (90 - V1)\big) + L3 \times \cos\Big(\big(V3 - \big(90 - \big(V2 - (90 - V1)\big)\big)\big)\Big)$$

Formell 3: Beräkning av Ypos.

$$Ypos = L1 \times \sin(V1) + L2 \times \sin\big(V2 - (90 - V1)\big) + L3 \times \sin\Big(\big(V3 - \big(90 - \big(V2 - (90 - V1)\big)\big)\big)\Big)$$

Formel 4: Beräkning av Xpos.

$$Xpos = \Big(LengthBetween - \Big(LengthShoulder \times \cos(ShoulderVal) + LengthElbow \times \cos\big(ElbowVal - (90 - ShoulderVal)\big) + LengthWrist \times \cos\Big(WristVal - \big(90 - \big(ElbowVal - (90 - ShoulderVal)\big)\big)\Big)\Big)\Big)$$

Formel 5: Beräkning av Ypos.

$$Ypos = \Big(\Big(LengthShoulder \times \sin(ShoulderVal) + LengthElbow \times \sin\big(ElbowVal - (90 - ShoulderVal)\big) + LengthWrist \times \sin\Big(WristVal - \big(90 - \big(ElbowVal - (90 - ShoulderVal)\big)\big)\Big)\Big) - \text{HeightBase}\Big)$$

Formel 6: Slutgiltiga formeln för beräkning av x-värde.

$$X = \Bigg(LengthBetween - \Bigg(LengthShoulder \times \cos(ShoulderVal) + LengthElbow \times \cos\big(ElbowVal - (90 - ShoulderVal)\big) + LengthWrist \times \cos\Big(WristVal - \big(90 - \big(ElbowVal - (90 - ShoulderVal)\big)\big)\Big)\Bigg)\Bigg) \times \cos(BaseDegrees)$$

Formel 12: Uträkning av vinkel V1, V2, V3 enligt fortsatt metod ur formel 11.

$$V1 = V1V + V1CL$$

$$V2 = \cos^{-1}\left(\frac{LengthShoulder \times LengthShoulder + LengthElbow \times LengthElbow - C \times C}{2 \times LengthShoulder \times LengthElbow}\right) - 90$$

$$V3 = Normalen + V3CL + V3V - 90$$

Formell 7: Slutgiltiga formeln för beräkning av y-värde.

$$Y = \Bigg(\Big(LengthShoulder \times \sin(ShoulderVal) + LengthElbow \times \sin\big(ElbowVal - (90 - ShoulderVal)\big) + LengthWrist \times \sin\Big(WristVal - \big(90 - \big(ElbowVal - (90 - ShoulderVal)\big)\big)\Big)\Big) - \text{HeightBase}\Bigg)$$

Formell 8: Slutgiltiga formeln för beräkning av z-värde.

$$Z = \Bigg(LengthBetween - \Bigg(LengthShoulder \times \cos(ShoulderVal) + LengthElbow \times \cos\big(ElbowVal - (90 - ShoulderVal)\big) + LengthWrist \times \cos\Big(WristVal - \big(90 - \big(ElbowVal - (90 - ShoulderVal)\big)\big)\Big)\Bigg)\Bigg) \times \sin(BaseDegrees)$$

Formel 11: Vinkelberäkning för figur 4.19.

$$RealX = LengthBetween - X$$

$$RealY = Y$$

$$U = \sin(Normalen) \times LengthWrist$$

$$tempX = RealX - U$$

***Fall 1, U = 0*:** $K = LengthWirst + RealY$

***Fall 2, U > 0*:** $K = \sqrt{(LengthWrist \times LengthWrist - U \times U)} + RealY$

$$C = \sqrt{tempX \times tempX + K \times K}$$

$$V3V = \sin^{-1}\left(\frac{tempX}{C}\right)$$

$$V1V = \sin^{-1}\left(\frac{K}{C}\right)$$

$$V3CL = \cos^{-1}\left(\frac{LengthElbow \times LengthElbow + C \times C - LengthShoulder \times LengthShoulder}{2 \times LengthElbow \times C}\right)$$

$$V1CL = \cos^{-1}\left(\frac{LengthShouder \times LengthShoulder + C \times C - LengthElbow \times LengthElbow}{2 \times LengthShoulder \times C}\right)$$