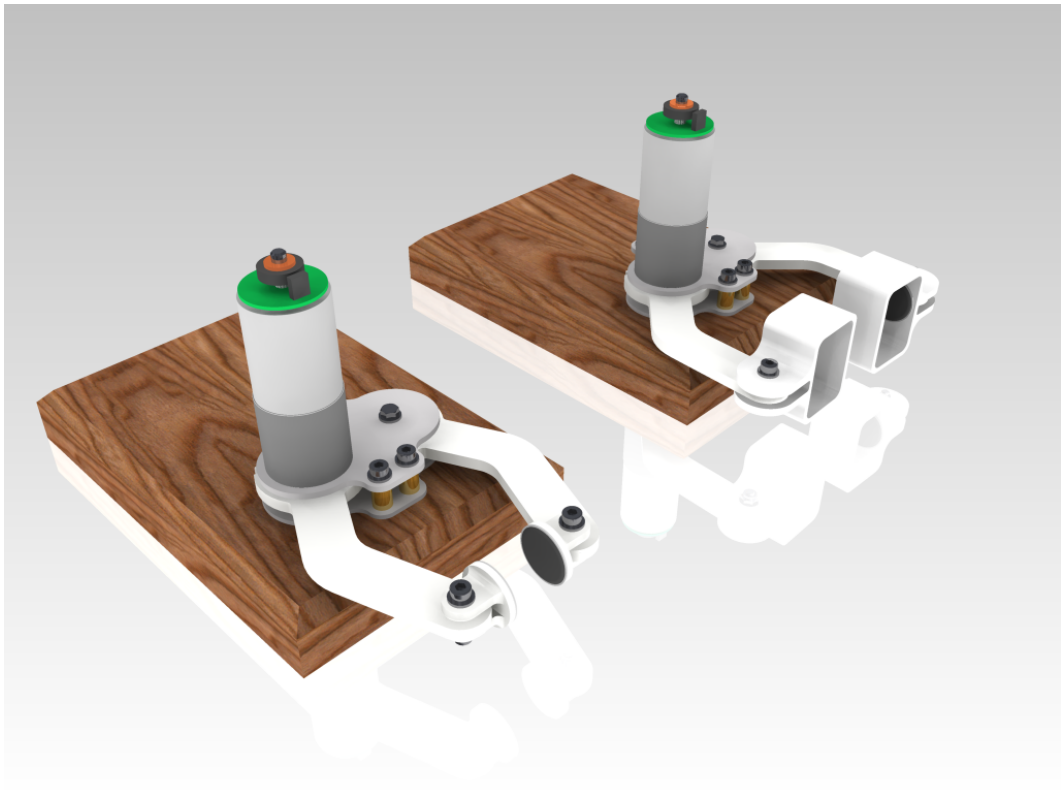
CHALMERS



Haptiskt gripdon

Ett kandidatarbete inom mekatronik

PAYAM CHEHRAZI
GUSTAV HANSSON
PATRIK HANSSON
MARKUS LINDELÖW

Avdelningen för Reglerteknik, Automation och Mekatronik
Institutionen för Signaler och System
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2012

Kandidatarbete/rapport nr. SSYX02-12-31

Abstract

Haptic technology and tactile feedback is an ever growing field with applications in many different industries. Precision surgery using robotics employ haptic feedback as a means to convey a sense of touch to the surgeon. Areas where people handle sensitive or potentially dangerous material e.g. in the nuclear industry make use of so called telemanipulators. Haptic feedback is also used in military applications and security. Bomb disposal units using teleoperated robots being one such application.

This thesis presents the results of a bachelors project involving the design and construction of a haptic feedback robotic gripper using engineering design principles. The gripper is to be operated through a computer network, where the user controls a display unit on one end and recieves haptic feedback from the gripper on the other end.

As information is sent through a network, the handling of system delays such as latency is one of the main challanges. Both grippers are to be controlled using Arduino mega microcontrollers. In the first phase, a theoretical foundation is set for the project where the overall system is modelled using general mechanics, bond graphs and mathematical transforms. The system is then implemented in simulink where simulation are made to verify the projects initial design criterion and choice of components. Primarily, simulations were done to evaluate the responsiveness of the dc motors used to run the gripper and display, as these have the greatest impact on the responsiveness felt by the user. The task of the computers is to bypass the information from the Arduino controller to the other part and reverse. Full-duplex asynchronous communication between the computers are managed by a delay optimized TCP/IP implementation. The robotic gripper and haptic display are custom designed in 3D with a CAD software and the most complex parts are extracted using rapid prototyping technology.

Samanfattning

Haptiskt teknologi och taktil återkoppling är ett växande område med tillämpningar i många olika industrier. Inom precisionskirurgi med robotteknik används haptisk återkoppling för att ge kirurgen en känsla av beröring. Områden där känsliga och potentiellt farliga material behandlas, t. ex. i kärnkraftsindustrin, används så kallade telemanipulatorer. Haptisk återkoppling används även i militära tillämpningar och säkerhet, exempelvis i bombdesarmeringsrobotar.

Den har rapporten presenterar ett kandidatarbete som innefattar design och konstruktion av ett haptiskt gripdon baserat på ingenjörsmässiga principer. Gripdonet styrs igenom ett datornätverk, där användaren kontrollerar en displayenhet i en ände och får haptiskt återkoppling från gripklon i andra änden.

Då information skickas genom ett nätverk är hanteringen av systemfördröjningar såsom latens en av de större utmaningarna. Båda gripenheterna ska kontrolleras med hjälp av Arduino mega microprocessorer. Arbetet inleds med utformningen av en teoretisk grund för systemet som helhet med hjälp av mekanik, bindningsgrafer och matematiska transformer. Systemet implementeras sedan i simulink och simuleras för att verifiera projektets grundkriterier och komponentval. Primärt användes simuleringarna för att utvärdera svaret från DC-motorerna som används för att styra båda enheterna, då de har störst verkan på återkopplingen till användaren. Datorernas uppgift är att vidarebefordra information mellan Arduino-kretsarna. Asynkron kommunikation mellan datorerna sker med full duplex och sköts av en fördröjningsoptimerad TCP/IP-implementation. Gripklon och displayen är designade i CAD-programvara och de mest komplexa delarna tillverkas med rapid prototyping-teknologi.

Innehåll

Förteckningslista

Arduino-enhet	Kretskort innehållande bland annat en AVR-mikroprocessor, seriell anslutning och flertalet in- och utgångar. Har programvara baserat på öppen källkod
Backlash	Glapp i mekaniska komponenter, till exempel mellan kuggarna i en växellåda.
C++	Objektorienterat programmeringsspråk.
CAD	Computer-aided design, datorstödd design och skapande av tekniska ritningar.
DC-motor	En likströmsmotor där vridmoment är proportionellt mot varvtalet, med negativ proportionalitetskonstant.
Display	Föremål som ger användaren information via något sinne, t.ex. ett haptiskt handtag.
Duty cycle	Berättar förhållandet mellan puls och ickepuls i en PWM-signal.
Fartreglage	Enhet som tar emot en datasignal och sedan styr en eller flera motorer, oftast m.h.a. PWM-spänning, med extern strömförsörjning.
FEA	Finit elementanalys, avser utnyttjandet av FEM för datorstödda hållfasthetsanalyser.
FEM	Finita elementmetoden, metod för att numeriskt finna lösningar till partiella differentialekvationer.
FSR-sensor	En resistor vars resistans beror på pålagd last. Används t.ex. för att mäta krafter.
Interrupt/Avbrott	En händelse i en mikroprocessor där den tar paus med det den håller på med för att utföra andra beräknigar tillfälligt.
Kontrollsumma	Värde som genereras utifrån en större datamängd. Används ofta till att kontrollera att två olika datasegment är samma utan att jämföra all data.
Latens	Fördröjningen i ett system mellan två enheter.

PPM	Puls Position Modulating, ett sätt att analogt överföra information där längden på signalen ger datavärdet. Används t.ex. till RC-servon.
PWM	Pulsbreddsmodulering, en regelbunden signal där olika långa pulser i hög frekvens ger ett medelvärde. Används t.ex. för att reglera spänning.
Rapid prototyping	Metod som innebär att en CAD-modell kan ”skrivas ut” i 3D direkt i en polymer för snabb prototyptillverkning.
Saturation	Begränsning på aktuatorns vridmoment, flödeskapacitet, vinkelhastighet etc. Ex. En aktuator har en begränsning på hur mycket spänning den kan ta innan motorn överhettas.
Simulink	Programvara för modellering av dynamiska system.
Transportlager	Beskriver hur datan skall skickas mellan två länkar i ett nätverk
Tråd	En tråd innehåller programinstruktioner till en processorkärna.

1 Inledning

1.1 Bakgrund

Rapportens titel *Haptiskt gripdon* grundar sig i begreppet haptisk teknologi vilket omfattar den teknik som använder känseln för att förmedla information till och från en användare. En användares indata ger svar i form av stimuli bestående av krafter, vibrationer och rörelser. Haptisk teknologi kan användas för att på avstånd få en vidare uppfattning om ett föremål än den information som kan förmedlas via visuella och auditiva hjälpmedel, det vill säga kamera och bildskärm samt mikrofon och högtalare. Ytterligare information om föremålet, så som elasticitet, struktur och ibland även storlek kan, till exempel när referensobjekt är utom synhåll, behöva uppfattas med känseln.

Många tekniska idéer finner spridning i skönlitteraturen, speciellt inom science fiction-genren. I novellen Waldo [1], skriven av Robert A. Heinlein 1942, är huvudpersonen försvagad av muskelsjukdomen myasteni. I sin strävan att fungera i samhället uppfinner han en maskin som han kallar ”Waldo F. Jones’ Synchronous Reduplicating Pantograph”. Maskinen är en kraftfull mekanisk hand som styrs via en handske och en sele som användaren tar på sig. Heinlein påstod att han fick idén till novellen efter att ha läst en artikel i Popular Mechanics om en person som led av just myasteni och som själv hade utvecklat ett system med hävarmar för att kunna använda den lilla styrka han hade [2].

Verklig utveckling inom området telemanipulation har pågått sedan 40-talet [3] då telemanipulatorer började användes inom industrin för fjärrstyrning i farliga eller oåtkomliga miljöer, bland annat inom kärnkraftsindustrin. Dessa fjärrmanipulatorer kallas också populärt för ”Waldoes”. På rymdstationen ISS används en robotarm [4] som kallas Dextre. Under senare år har haptisk teknologi även börjat användas för att återskapa mänsklig interaktion på stora avstånd och i virtuell verklighet [5].

Haptisk teknologi används också idag i stor omfattning i kirurgiska robotar [6] där uppfattningen av känsel är mycket viktig för att undvika för stora rörelser. Även i bombdesarmerigsrobotar [7] används haptik av samma anledning. Men den kanske vanligaste typen av haptisk teknologi är den som finns i hemmet i form av exempelvis vibrerande TV-spelshandkontroller eller till och med datortangentbord.

Projektet syftar till att genom konstruktion och experiment undersöka haptisk teknologi. Fokus ligger inom reglerteknik och mekatronik men även till viss del klassisk mekanik. Ett antal olika problemformuleringar och konceptförslag har diskuterats:

- Gaffeltruckreglage med feedback av belastningen på truckens gaffel. Problemdefinition: Truckföraren saknar känsla för tyngden på gafflarna. Vikten på lasten behöver kontrolleras manuellt, vilket tar tid. Slitage- och olycksrisk vid överlast. Kan dessa problem lösas med haptisk teknologi? Konceptförslag: Om trucken överbelastas kan detta informeras via känselinformation i reglaget. Av tidigare kandidatarbetsgrupper redan tillverkade prototyper kan användas för att simulera truckens gaffel.

- ”Bryta arm över nätverk”. Problemdefinition: Är det möjligt att använda haptisk teknologi i datorspel? Konceptförslag: Enkel konstruktion, två armar med elmotorer och växellåda kopplade till datorn. Poängsystem möjligt. Även handikappsystem möjligt. Man kan ”provbryta” för att få sitt handikapp, sedan justeras krafterna efter de tävlandes handikapp.

- Haptiskt gripdon. Problemdefinition: Kan man känna skillnad på och identifiera olika objekt utan att se? Får man en annan uppfattning av objekten om man kan se? Kan man greppa ett ägg utan att knäcka det? Konceptförslag: Gripdon med två ”fingrar” som ger feedback till användarens fingrar vid tryck eller moment i gripklons leder. Gripdonet ska fungera utom syn- och räckhåll för användaren.

Samtliga förslag är intressanta utifrån projektets syfte. Alla kräver en viss mekanisk konstruktion, reglering och elektronik. Tack vare möjligheten att köpa redan färdiga gripdon, utan reglering, ”off the shelf”, föll valet på det haptiska gripdonet. Därför kan fokus läggas på reglerdesign och datorkommunikation. Dock visade det sig att dessa gripdon var undermåliga, oftast för små eller för dyra. Det beslutades därför att gruppen även skulle konstruera hela gripdonet, även de mekaniska delarna.

1.2 Syfte och mål

Utifrån valet har tydligare mål har tagits fram: En person ska med en haptisk display på avstånd kunna styra en gripklo till att greppa ett objekt, varvid personen får återkoppling i form av reaktionskrafter från den haptiska displayen. Målet för slutprodukten är att personen ska kunna känna skillnad på elastiska egenskaper hos små objekt, till exempel en skumgummiboll och en stenkula då objektet är utom syn- och räckhåll. Även storlek på objekten ska kunna särskiljas. Gensvaret från displayen ska vara tillräckligt exakt för att en användare ska kunna ta tag i ett ägg utan att krossa det. Detta ger upphov till en rad problem. Ett koncept för att lösa dessa problem har tagits

fram och en kravspecifikation med mätbara krav presenteras längre fram i texten.

1.3 Koncept

För att nå målen som beskrivs i föregående kapitel togs ett lösningskoncept fram. För att göra systemet överskådligt delas det upp i tre delsystem. Se figur 1:

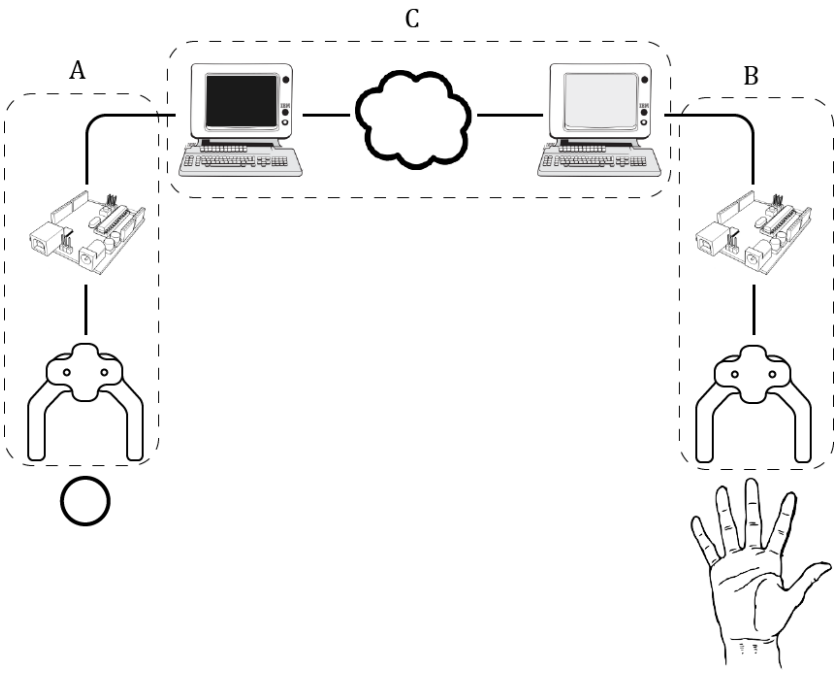
A En form av gripklo behövs för att ta in informationen av objektet som ska undersökas. För att inte göra systemet för storleksmässigt omfattande används en gripklo med två fingrar i ungefär samma storlek som en människas. Ett ägg ska kunna greppas.

B För att en person ska kunna känna av ett objektet krävs någon form av människa-maskin-gränssnitt. En så kallad haptisk display behövs. En haptisk display är ett form av reglage som ger återkoppling i form av rörelser och krafter av vad som händer i andra änden av gripdonet. För konstruktionsmässig enkelhets skull är displayen i samma storlek som gripklon.

C Datorkommunikation krävs för att de två delarna display och gripklo ska kunna utbyta information. Gripklon ska efterlikna displayens rörelser och återkoppling till displayen från gripklon måste ske.

1.4 Systemdesign

Gripklon (A) består av en mekanisk gripmekanism som drivs av någon form av aktuator. Längs ut på vardera av gripfingrarna sitter en sensor som registrerar kraften som uppstår mellan dessa när gripklon greppar ett objekt eller helt för samman gripfingrarna. Detta kan även lösas med trådtöjningsgivare monterade på gripfingrarna. Någonstans i fingrarnas led behövs en sensor som används för att hålla reda på gripfingrarnas position. Displayen (B) är kortfattat en bakvänd gripklo och fungerar som användargränssnitt. I stället för att kraftsensorerna sitter på gripfingrarnas toppar sitter de här istället i två fingerhållare. Både gripklo och display är kopplade till varsin Arduino-enhet, en microcontroller som sköter regleringen av motorerna. Dessa styrenheter är sammankopplade med det tredje delsystemet (C) som består av två PC-datorer. Dessa PC-datorer sköter nätverkskommunikationen.



Figur 1: Konceptuell översikt över produktsystemet. Molnet symboliserar nätverkskommunikationen.

1.5 Metod

I konstruktionen ingår olika typer systemmodellering som används som hjälpmedel vid komponentval och design av mekaniska delar samt reglersystem. För att undvika begränsningar på gripdonets räckvidd programmeras ett datorgränssnitt som tillåter kommunikation över datornätverk, till exempel internet. Verktygen som används är CAD, Simulink och C++.

Matematiska modeller som beskriver systemets dynamiska beteende tas fram och implementeras i matlab och simulink. En lämplig regulatordesign tas sedan fram baserat på resultat från simuleringar. Regulatorn diskretiseras sedan och implementeras.

Chalmers prototyplaboratorium tillhandahåller verktyg och material till bygget av gripdonet.

Projektet har en budget på 4000 sek och bedrivs under en termin.

1.6 Hållbar utveckling

Även ur hållbarhetssynpunkt är projektet intressant. Ett mer utbrett användande av haptisk teknologi i syftet att eliminera behovet av närvarande personal kan minska behovet av energikrävande transporter och transportvä-

gar och på så sätt hålla nere utsläppen av växthusgaser. Skulle möjligheterna till transporter i framtiden minska, på grund av dagens användande av de begränsade resurserna av fossila bränslen, kan haptisk teknologi vara ett alternativ för framtidens generationer att tillgodose dessa behov som annars skulle krävt transport. Vidare bidrar detta till en social hållbarhet genom att tekniken öppnar upp för t.ex. specialistläkare att lättare nå ut för att utföra undersökningar och operationer i områden som saknar infrastruktur. Humankapitalet tas tillvara på ett mer effektivt sätt.

2 Teori

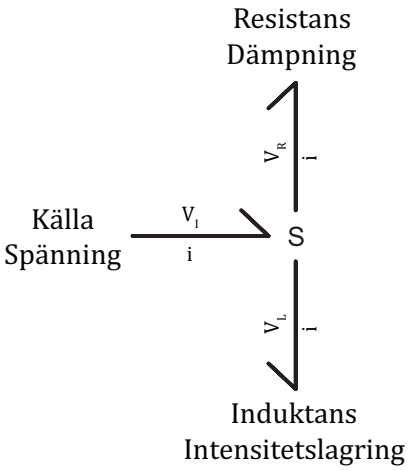
Detta avsnitt syftar till att beskriva de teoretiska delarna i projektet, alltså vilka matematiska verktyg, principer och datorprogram som används.

2.1 CAD

CAD-modeller används för att konstruera, analysera och simulera de mekaniska delarna av systemet. Detaljer konstrueras en och en och monteras i datormiljön precis som ska ske i verkligheten. På så sätt kan modellens mekaniska rörelser testas. Hållfasthetsberäkningar görs i form av FEA som finns inbyggt i CAD-programvaran. FEA utförs genom att i datormiljön förankra en detalj och tilldela den en belastning, i detta fallet en kraft. Programmet levererar efter simulering en rad ur hållfasthetssynpunkt intressanta data, så som eventuella spänningskoncentrationer och utböjningar.

2.2 Bindningsgrafer

Bindningsgrafer bygger på principen att många fysiska system kan representeras med flödes- (ström, hastighet, vinkelhastighet), f, och intesitetsvariabler (spänning, kraft, moment), e, och används för att ta fram tillståndsbeskrivningar av dynamiska system. Energiflödet genom systemet markeras med en halvpil, vars riktning indikerar energiflödets riktning. Knutpunkter beskriver hur flödet i ett system delas upp mellan de olika element som är kopplade till knutpunkten. En seriekopplad krets kan representeras med bindningsgrafen i figur 2. Här representerar insignalen en intensitetskälla i form av en matningsspänning, som delas upp över kretsens resistans (dämpning), induktor (flödeslagring). Summan av alla element i en seriell knutpunkt är noll, där bindningens riktning bestämmer elementets tecken. I en seriekopplad elektrisk krets kan detta jämföras med Kirchoffs lag.

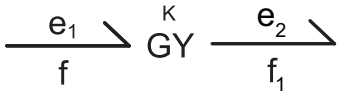


Figur 2: Exempel på bindningsgraf av en seriekopplad RL krets

En roterande massa med tröghetsmoment J och viskös friktion B följer samma princip, där J är intensitetslagringen och B dämpningen. Intensitetskällan är här ett vridmoment som ger upphov till en vinkelhastighet.

Slutligen finns även de effektbevarande genomflödeselementen gyratorer och transformatorer. En gyrator omvandlar en form av energi e_1 till en annan, e_2, t.ex elektrisk energi till rotationsenergi i en elektrisk motor. En transformator antingen förstärker eller dämpar olika flöden och intensiteter, men omvandlar inte energi. Ett exempel på en transformator är en utväxling, där vridmomentet ökas mellan transformatorns portar.

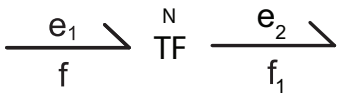
Sambanden mellan in och utsignal i en gyrator respektive transformator beskrivs med figurerna 3 och 4 samt ekvationerna (1), (2), (3) och (4):



Figur 3: Bindningsgraf av en gyrator. e = energi, f = flöde

$$e_1 = f_1 K \tag{1}$$

$$f_1 = \frac{e_1}{K} \tag{2}$$



Figur 4: Bindningsgraf av en transformator. e = energi, f = flöde

$$Ne_1 = e_2 \tag{3}$$

$$\frac{f}{N} = f_1 \tag{4}$$

I bindningsgrafer markeras även kausalitet, sambanden mellan orsak och verkan, vilket här indikerar vilken sida av en bindning som bestämmer de momentana flödena och intensiteterna. Detta markeras med en vertikal linje i början eller slutet av bindningen. Det finns ett antal regler eller restriktioner om hur och var man sätter kausalitetsmarkeringar. Kan dessa regler inte följas säger man att det finns en konflikt i systemet. För mer information se boken Modeling of Dynamic Systems [8].

2.3 Datornätverk

Ett datornäverk [9][10] består av en uppsättning sammankopplade datorer och nätverkslänkar, trådlöst som trådbundet, som kan skicka information till varandra. Dagens moderna nätverk identifierar varje dator med hjälp av ett unikt nummer, en IP-address, som används för att informationen skall komma fram till rätt dator. Sedan t.ex. hur informationen paketeras och hur fel behandlas bestäms av ett transportlager. Det finns flertalet olika transportlager.

2.3.1 Nagle's algoritm

Nagle's algoritm [11][9] är en teknik vars syfte är att slå ihop många små nätverkspaket till ett fåtal stora. Detta leder till att belastningen på nätverket minskas. Genom att vänta in en viss mängd data eller en bekräftelse på att föregående paket har nått fram innan det nya paketet paketeras och skickas blir paketen färre och större. Det leder också till längre fördröjningar innan paketen kommer iväg.

2.3.2 TCP

TCP, Transmission Control Protocol [9][12], är ett transportlager som upprättar en anslutning och en ström med paket skickas. Paketen kommer fram

i samma ordning som paketen skickas och en kontroll av att paketet innehåller rätt information sker. Skulle något av paketen vara fel skickas det om. TCP implementerar som standard Nagle's algoritm men den går att också att inaktivera.

2.3.3 UDP

Transportlagret UDP, User Datagram Protocol [9], skickar paket utan någon tidigare förbindelse och struntar som standard i vad som händer med dem. De kan komma fram i olika ordning, försvinna och informationen kan ha blivit korrupt. Det går att aktivera kontrollsummor på paketen för att verifiera dataintegriteten.

2.3.4 DCCP

DCCP, Datagram Congestion Control Protocol [13], fungerar likt transportlagret UDP men har en kontinuerlig förbindelse som förhandlas fram. Detta ger fördelen att flödet på datamängden kan regleras och att nätverket skulle översvämmas med paket kan förhindras. Det finns även med en kontrollsumma på paketen som används för dataverifiering. Den är utformad med strömmande media i fokus där låga latenser behövs, som t.ex. vid IP-telefoni. Det är en relativt ny teknik och har inte samma mängd av dokumentation och verktyg för utveckling som de äldre transportlagren.

2.3.5 RUDP

RUDP, Reliable User Datagram Protocol [14], är ett transportlager som är under utveckling. Syftet med RUDP är att åstadkomma en snabb och enkel överföring. Den är lättare än TCP i den bemärkelsen att den inte är lika komplex och det är mindre metainformation i paketen. Den är dock tyngre än UDP då den säkerställer att paketen är korrekta och kommer fram i rätt ordning. Tekniken överbuffring används för att uppnå korta responstider.

3 Material och metod

Material och metodavsnittet syftar till att mer ingående beskriva hur gruppen löst det praktiska tillvägagångssättet vid konstruktionen av det haptiska gripdonet samt hur de teoretiska verktygen beskrivna i föregående kapitel utnyttjas. Med resultaten av arbetet har en detaljerad kravspecifikation kunnat tas fram varpå komponentval grundar sig.

3.1 Konstruktion och CAD-modell

Display och gripklo är ritade i Autodesk Inventor 2012 och är i grunden samma modeller. Skillnaden mellan dem är endast utformningen av gripfingrarnas toppar. Nedan följer beskrivningar av de viktigaste ingående delarna.

3.1.1 Fingertoppar

Displayen har längst ut på gripfingrarna två ledade hållare där användaren för in tummen och pekfingret. I den ena av dessa hållare sitter två kraftsensorer för att registrera fingrarnas in- och utrörelser. Gripklon har två ledade plattor där kraftsensorer fästs. Dessa sensorer registrerar kraften när ett objekt greppas eller när fingrarna stängs fullt. Dessa ”fingertoppar” är, tillsammans med gripfingrarna, tillverkade med rapid prototyping.

3.1.2 Gripfingrar

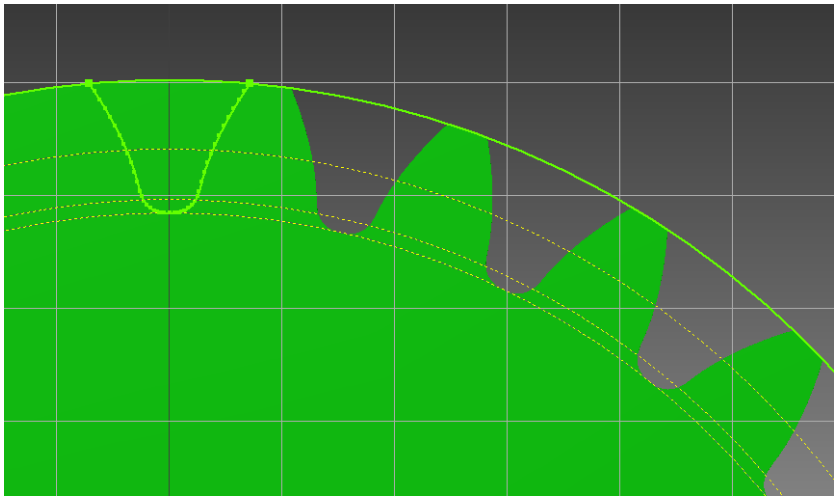
Gripfingrarna är vinklade 45° för att någorlunda efterlikna mänskliga fingrar och förbättra gripklons grepp samt göra det möjligt att placera ett skruvförband, som håller ihop hela paketet, mellan fingrarna och på så sätt förstärka konstruktionen. Fingrarna är ca. 8 cm långa och kan öppnas till omkring 10 cm mellan fingerspetsarna. Det ena gripfingret sitter monterat på ett mässingsfäste med låsskruv som i sin tur sitter fäst direkt på elmotorns växellådas axel. Det andra gripfingret är monterat med en skruv som axel. Gripfingrarna är sammankopplade för motriktad rörelse med en kuggväxel och fästa mellan två aluminiumplattor.

3.1.3 Kuggväxel

Att manuellt fräsa ut kuggar är tidskrävande och svårt men tack vare möjligheterna att snabbt bygga avancerade detaljer med rapid prototyping är kuggprofilen skräddarsydd. Den har därav ingen standardmodul. Figur 5 visar en närbild av kuggprofilen. I bilaga B visas kugghjulens dimensioner.

3.1.4 Fästplattor

Fästplattorna sågas ut i 1,5 mm aluminiumplåt för nederdelen respektive 2 mm för överdelen. Plattorna är monterade ihop med gripfingrarna och mässingsdistanser mellan sig i ett paket med fyra skruvförband. Den övre plattan har en hålbild som passar elmotorns växellåda.



Figur 5: Kuggprofil

3.1.5 Övriga detaljer

En bottenplatta för vardera gripklo och display konstrueras i trä. Utrymme för elektronik finns på plattan. En distans som motsvarar mässingsfästet till det ena gripfingret svarvas i teflon och monteras på det andra gripfingrets axel. Fyra mässingsdistanser var svarvas till gripklon och displayen. Dessa ser till att det finns utrymme för gripfingrar och motorfäste mellan fästplattorna. Övriga delar, så som skruvar och brickor är standardkomponenter i dimensionerna M3 och M4.

3.2 Motorer

För att kunna röra gripklon och också låta displayen friktionsfritt följa handens rörelser samt ge feedback används likströmsmotorer, här kallade DC-motorer. En inbyggd växellåda gör dessa optimala för varvtal- och momentreglering. Även andra typer av aktuatorer kan användas. Pneumatik och hydraulik är bra för stora krafter, men är svårare att reglera. Dessa typer av aktuatorer hade kanske varit bättre till större krafter som för att efterlikna rörelser i till exempel en människas armar och ben.

Så kallade RC-servon, som används i radiostyrda leksaker, är i stort sett DC-motorer med inbyggd växellåda och logik för reglering av axelvinkeln. För att kunna reglera momentet i ett RC-servo krävs vissa fysiska ingrepp. En enkel DC-motor med växellåda är därför det bästa valet.

Motorns axelvinkel måste kunna mätas för att display och gripklo inte ska

bli osynkroniserade och hamna i olika positioner. Detta görs enklast med en s.k. Hall-effektsensor. Dessa sitter ibland förmonterade på motoraxeln, vilket för tidsbesparing är att föredra.

En risk med att använda elmotorer med en växellåda med hög utväxling är att man vid för stora belastande moment riskerar att förstöra växellådan eller någon av gripfingrarna i display eller gripdon. Ett krav på hela systemet är därför att det tydligt ska framgå när det är överbelastat. Detta kan ske genom någon form av feedback när kraftsensorerna registrerar en kraft som överstiger ett satt gränsvärde. Till exempel en summersignal eller ännu hellre att systemet helt enkelt ”släpper greppet”.

Motorerna som valdes är två stycken Pololu Metal Gearmotor 6V med en utväxling på 172:1. Det ger dem ett vridmoment på ungefär 1.5 Nm och ett varvtal på 33 rpm. Då kravet på varvtalet är 35 rpm anses dessa motorer uppfylla kraven. Dessa motorer har en förmonterad Hall-effektsensor som ger motorns relativa position 48 gånger per varv. För att bestämma motorns absoluta position måste därför enheterna kalibreringsköras innan varje användning.

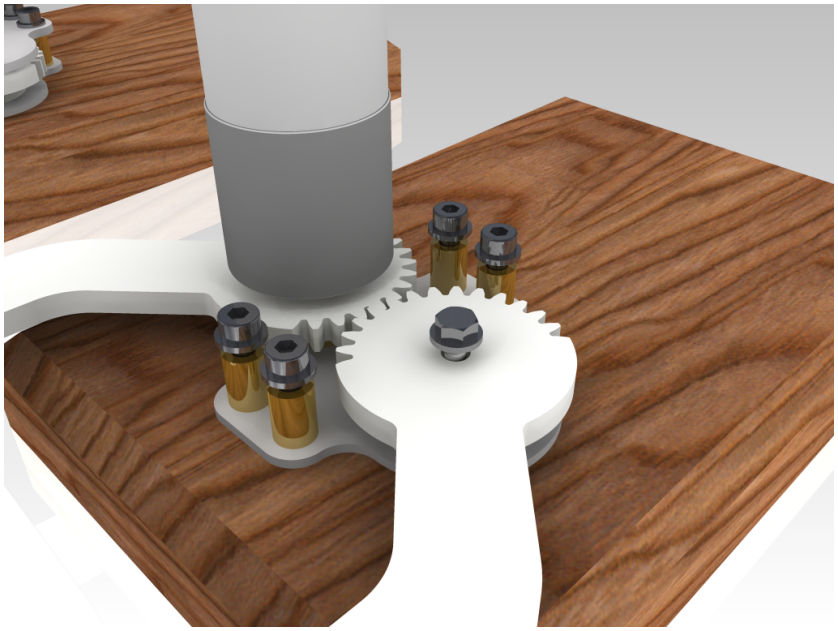
3.3 Fartreglage

Motorerna i display och gripklo behöver kunna drivas i olika hastigheter i två riktningar. Då Arduino-enheten endast klarar att leverera ström i storleksordningen 50 mA måste någon typ av motordrivare eller fartreglage användas. Även här finns många typer att välja mellan men då ett fartreglage av typen HB-25, som gott och väl leverupp till kraven, redan finns att tillgå valdes det att användas. Kraven är att det ska kunna driva elmotorerna vid tidigare beskrivna moment och hastigheter.

Ett till fartreglage av typen HB-25 köptes in för att undvika de problem som kan uppkomma då man använder olika reglage till motorerna.

3.4 Mekanik

Display och gripklo är av tillverknings- och reglermässiga orsaker lika till konstruktion och storlek. Samma krafter som registreras i displayen ska återskapas i gripklon och vice versa. Displayen blir enkelt uttryckt en omvänd gripklo. Konstruktionen där gripfingrarna är sammansatta med en kuggväxel gör att de rör sig åt motsatt håll. Se figur 6. Varvtalet ω på motor inklusive växellåda blir med en sådan kugghjulskonstruktion halva gripfingrarnas relativa vinkelhastighet. För en, för användarens upplevelse, följsam rörelse behöver displayens gripfingrar kunna röra sig från öppet läge till slutet på mellan $t_{min} = 0.15$ s till $t_{max} = 2$ s. Dessa tider är uppmätta med stoppur.



Figur 6: CAD-rendering av gripklon. Bilden visar gripfingrarnas kuggväxel, undre fästplatta (den övre är dold), motor samt bottenplatta i trä.

Öppet tillstånd räknas som $l_{max} = 8$ cm mellan fingerspetsarna och slutet $l_{min} = 0.5$ cm, vilket ger ett l på 7,5 cm. Att l_{min} inte är 0 beror på att konstruktionen inte tillåter att fingrarna förs samman helt. Med en fingerlängd på $R = 7$ cm kan en vinkelhastighet approximeras till enligt (5). Se figur 7 för förtydligande av beteckningar.

$$\omega = \frac{l/2(t_{max} - t_{min})}{R} \tag{5}$$

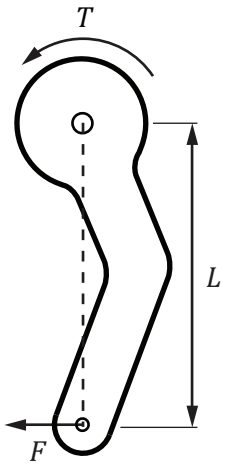
och därmed kan alltså ett önskat motorvarvtal n räknas fram enligt ekvation (6)

$$n = \omega \frac{60}{2\pi} = \frac{15l}{R\pi(t_{max} - t_{min})} \tag{6}$$

Med värdena ovan behöver motorns varvtal, inklusive växellåda, ligga ungefär mellan 3 och 35 rpm.

En vuxen mans maximala nypstyrka, ”tip pinch strength”, ligger runt 17 pounds eller ca. 75 N [15]. Med en hävarm, alltså displayens fingrar, på $L = 7$ cm, ger det ett moment på $T = 5.25$ Nm. Se figur 7. Detta är ett stort moment som skulle kräva en orimligt kraftig dimensionering av displaykonstruktionen. Då syftet med gripdonet inte är att kunna greppa hårt kan ett rimligt krav

på motorn istället uppskattas till ungefär en femtedel av detta moment, d.v.s. ca. 1 Nm.



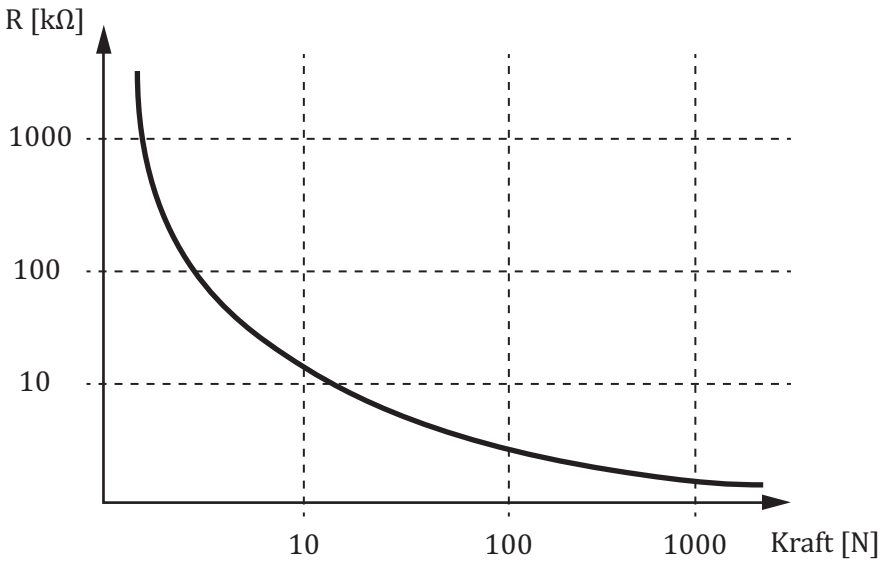
Figur 7: Frilagt gripfinger

3.5 Kraftsensorer

Sensorer behövs för att mäta kraften som läggs på display respektive gripklo. Även här finns flera olika typer att välja mellan. De är vanligtvis av typen *FSR*, *force sensitive resistors* [16] som enkelt uttryckt en resistor vars resistans minskar när en kraft läggs på dess sensoryta. De varierar i pris, upplösning och pålitlighet. Totalt tre eller fyra sensorer behövs för hela systemet, två i displayen som registrerar fingrarnas öppna- och stäng-rörelser samt en eller två, beroende på signalkvalité, i gripklon som registrerar reaktionskrafterna från det greppade objektet. Då projektet har en begränsad budget används den billiga varianten som består av två lager flexibel plastfilm med ett distanslager mellan. Kraft-resistanskurvan är olinjär. Figur 8 visar en typisk kraft-resistanskurva och kommer från en tillverkare av denna typen av kraftresistor. Krafterna som sensorn behöver kunna registrera räknas ut med:

$$F = \frac{T}{L} \tag{7}$$

Detta ger krafter mellan 0 och 15 N. För en naturlig greppkänsla så att motorn till exempel inte hackar sig fram när man klämmer på ett mjukt, elastiskt objekt krävs en tillräckligt god kraftupplösning. Denna uppskattas till åtminstone 1 N per steg. Större upplösning är önskvärt, men inte något krav.



Figur 8: Kraft-Resistanskurva för FSR-sensor

3.6 Kravspecifikation

Tabell 1 sammanställer och kompletterar de tidigare nämnda kraven. Ytterligare krav som livslängd, kostnad, vikt och förvaringsvolym är uppkomna av rent praktiska och ekonomiska orsaker. Tabell 2 visar i sin tur vad detta ställer för krav på komponenterna.

Tabell 1: Kravspecifikation, system

Specifikation	Krav	Önskemål
Högsta fingerhastighet	>7.5 cm/0.15 s	n/a
Lägsta fingerhastighet	<7.5 cm/2 s	n/a
Krafttolerans, upplösning	± 1 N	± 0.2 N
Friktion i obelastad display	0.5 N	0.1 N
Upplösning fingerposition	1 mm	0.5 mm
Kraft vid överbelastning	10 N	n/a
Varning vid överbelastning	Ljudsignal	Systemet ”släpper taget”
Vikt display	<1 kg	<0.5 kg
Vikt gripklo	<1 kg	<0.5 kg
Livslängd, system	500 h	1000 h
Förvaringsvolym, system	5x5x5 dm	n/a
Kostnad, system	<4000 kr	n/a

Med en nätverkslänk menas en enhet vars syfte enbart är att bygga upp nätverket, till exempel en router. Fler länkar leder till större latenser i nätverket. Ett hemnätverk har en länk, därför har det definierats som mätmiljö. Latenser i den typen av trådbundna nätverk ligger runt 1 ms.

Tabell 2: Kravspecifikation, komponenter

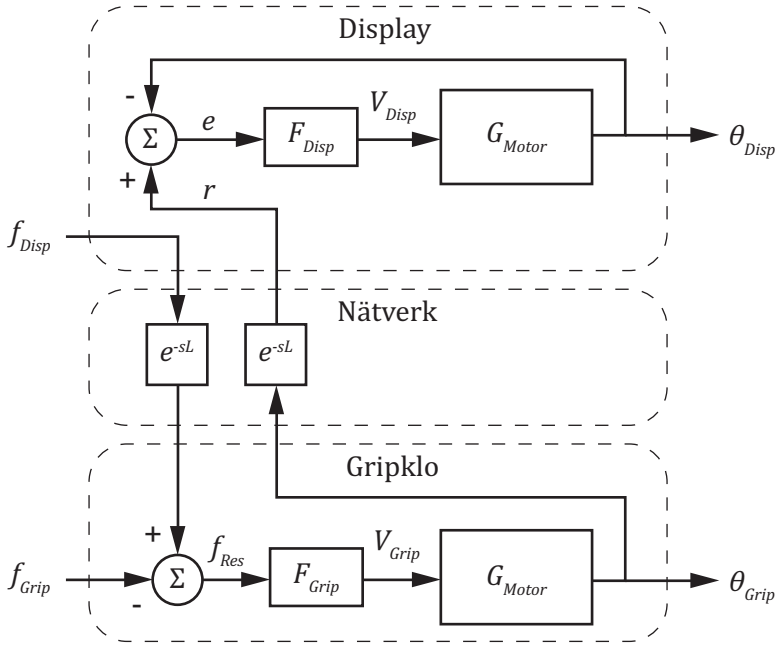
Specifikation	Krav	Önskemål
Motorhastighet	3 - 35 rpm	n/a
Motorvridmoment	1 Nm	n/a
Fartreglage ström, kont.	Driva motorn enl. ovan	n/a
Kraftsensor, intervall	0-15 N	0-20 N
Latens, en nätverkslänk	<40 ms	<10 ms

3.7 Reglerstrategi

Två typer av reglerstrategier testas. En synkron och en asynkron modell som löser problemen som uppstår vid latenser på olika sätt. Den synkrona försöker åstadkomma en minimal positionsavikelse mellan gripklo och display medan det asynkrona vill låta displayen efterlikna krafterna som uppkommer vid användning.

3.7.1 Synkron positionsreglering

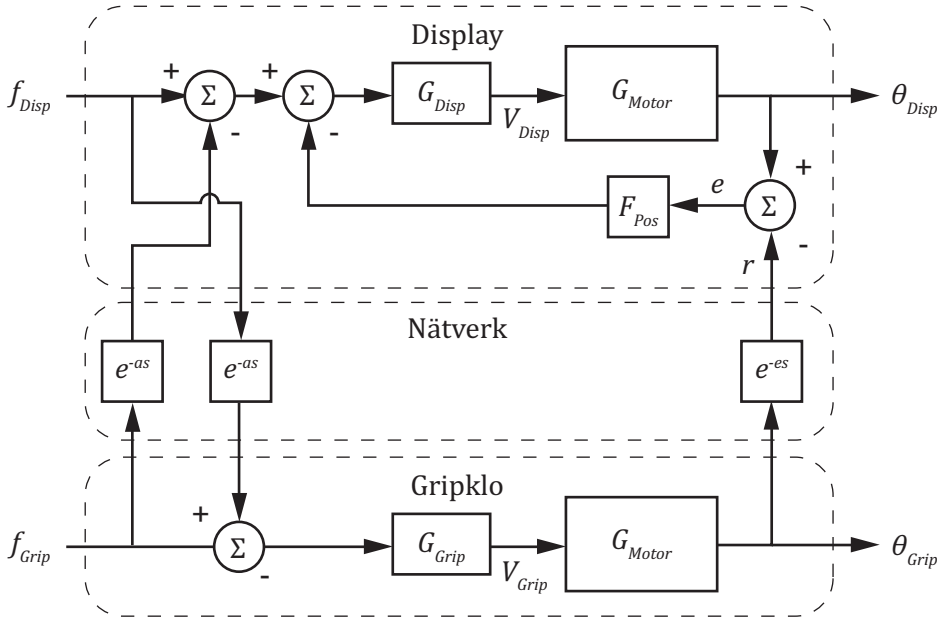
För att displayen och gripklon skall ha en så liten positionsavikelse från varandra som möjligt försöker displayen enbart att efterlika gripklons rörelser och positioner. Inget annat skall påverka displayens position - displayen rör sig när gripklon rör sig. För att gripklon skall komma i rörelse summeras grip- och displaykraften och översätts till en spänning till gripklons motor. En fördröjning kommer att uppstå mellan avläsningen av fingerkraften i displayen tills att gripklon får den informationen. Detta innebär att displayen inte reagerar direkt utifrån användarens rörelser utan måste först skicka kraften till gripklon som sedan svarar med en position. Om systemet inte reagerar tillräckligt snabbt finns det en risk för att användaren överkompenserar vilket kan leda till okontrollerade rörelser i gripklon.



Figur 9: Schema över det synkrona systemet

3.7.2 Asynkron positionsreglering

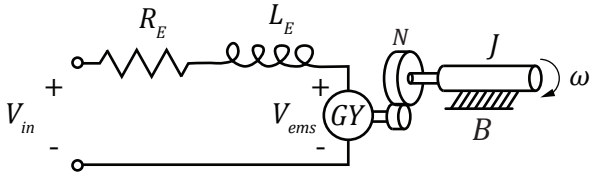
Den asynkrona modellen innebär att signalen från kraftsensorerna i displayen och gripklon summeras och används direkt för att reglera båda motorer. Det vill säga att displayens rörelser följer användarnas fingrar så att ett så litet motstånd som möjligt upplevs. När gripklons sensorer reagerar på en kraft, det vill säga då gripklon greppar ett objekt, skickas denna kraft tillbaka till displayen. Detta resulterar i att användaren upplever en reaktionskraft i displayen. Kraftsumman från displayen och gripklon kommer inte alltid att vara samma då det sker en viss fördröjning vid informationsutbytet. Detta innebär att displayen och gripklons position inte garanterat kommer att vara på samma ställe. För att reglera detta används en regulator vars uppgift är att justera displayens position efter gripklon. En risk med att reglera utifrån krafter är att enheterna även får en positionsavvikelse på grund av att de har olika tröghet. Om man försöker åtgärda det med en stark positionsreglering i displayen blir systemet likt det synkrona.



Figur 10: Schema över det asynkrona systemet

3.8 Modellering av DC-motor

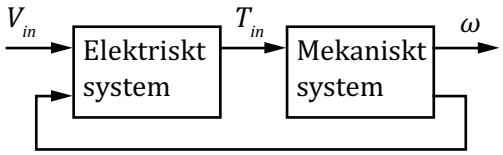
Nedan beskrivs G_{Motor} från figur 9 och 10. I figur 11 visas en DC-motors ekvivalenta kretsschema. DC-motorn delas upp i två delsystem, ett elektriskt



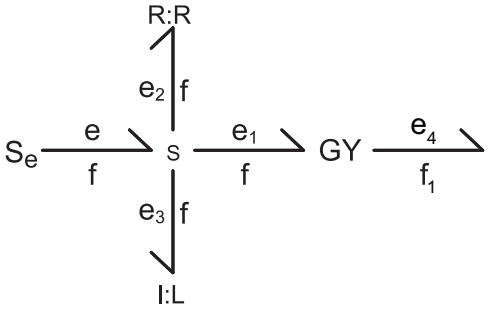
Figur 11: Ekvivalent kretsschema för en DC Motor

med en spänning som insignal, och ett mekaniskt delsystem med ett vridmoment som insignal. Systemets utsignal är rotorns position. Se tabell 3 för beteckningar.

Det elektriska delsystemet består av en resistans seriekopplad med en induktor och ett gyratorelement. Insignalen är matningspänningen. Bindningsgrafen med gyratorn medräknad visas i figur 13. Då det är en seriekopplad krets är flödet f, eller strömmen, samma över alla element kopplade



Figur 12: Systemet DC-motor



Figur 13: Bindningsgraf av motorns elektriska delsystem, se tabell 3

till knutpunkten, medan spänningarna e_1, e_2 och e_3 varierar över komponenterna (energiförluster, lagring). Samma princip gäller för den mekaniska delen. Gyratorn har här omvandlat spänning och ström till vridmoment och vinkelhastighet. Gyratorns bindningsgraf visas i figur 3. Sambanden mellan elektromotorisk spänning, ström, rotorns vridmoment och vinkelhastighet kan beskrivas med ekvationerna

$$T_{GY} = iK_a \tag{8}$$

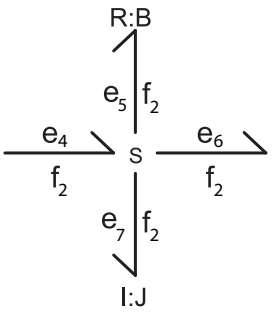
$$V_{ems} = \omega_{GY} K_a \tag{9}$$

där K_a är motorns ems-konstant. Energiförluster i motorns mekaniska del beror på rotorns viskösa friktion (B) samt dess tröghetsmoment (J). Friktionen beskrivs som en dämpning, då den leder till energiförluster medan tröghetsmomentet beskrivs som en intensitetslagring.

Motorns växellådas utväxling beskrivs med en transformator, där momentet ökas med en faktor N och hastigheten minskar med en faktor $1/N$. Transformatorns bindningsgraf visas i 4 och sambandet mellan in- och utvärde beskrivs ekvation (10).

$$iK_aN = T_{in} \tag{10}$$

$$\frac{\omega_{GY} K_a}{N} = \omega \tag{11}$$

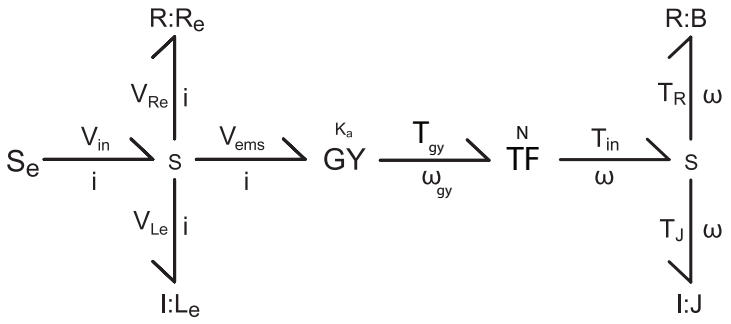


Figur 14: Bindningsgraf av motorns mekaniska delsystem

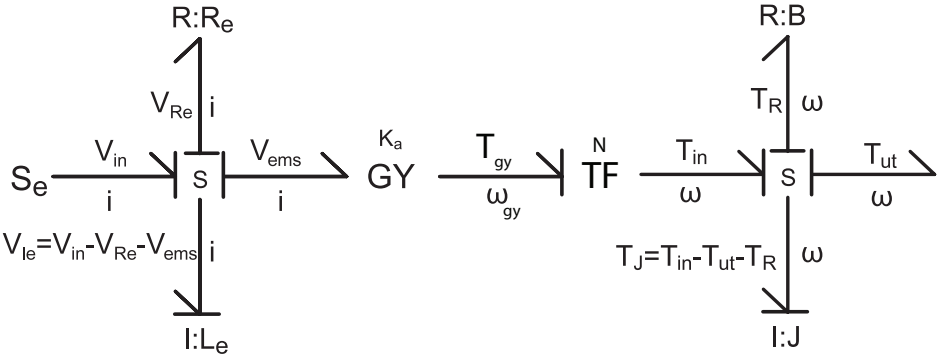
Tabell 3: Systemvariabler

Flödeselement	Variabel	Beskrivning
e	V_{in}	Matningsspänning
e_1	V_{ems}	Elektromotorisk spänning
e_2	V_R	Spänning över resistans
e_3	V_L	Spänning över induktor
f	i	Ström
e_4	T_{GY}	Rotorns vridmoment
f_1	ω_{GY}	Rotorns vinkelhastighet
f_2	ω	Rotorns vinkelhastighet efter utväxling
e_5	T_{in}	Rotorns moment efter utväxling
e_6	T_R	Motverkande moment från viskös friktion
e_7	T_J	Motverkande moment från rotorns massströghet
e_8	T_{ut}	Lastmoment från användarens fingrar

Insättning av systemets variabler i bindningsgrafen ger figur 15. I sista steget läggs ett lastmoment till i bindningsgrafen och kausalitet markeras. Lastmomentet kommer från operatörens fingrar, och är endast en belastande faktor då gripdonet bestämmer displayens position och räknas då som en störsignal. Kausalitetsmarkeringen visar att systemet är konfliktfritt, och kan representeras med ordinära differentialekvationer. Valet av tillståndsvariabler kommer enkelt och naturligt med bindningsgrafer. I ett dynamiskt system är utsignalen beroende på alla tidigare värden av systemets invärden. Systemets tillstånd vid tiden t_0 ger tillräckligt med information för att beräkna en utsignal för tider $t \geq t_0$ för en känd insignal. Tillståndsvariabler ger information för ett systems tillstånd vid tiden t. Valet av tillståndsvariabler blir här i och ω Sambandet mellan flöde och intensitet i en bindning kan



Figur 15: DC-motorns bindningsgraf utan last



Figur 16: DC-motorns bindningsgraf med last och kausalitetsmarkeringar

beskrivas med ekvation (12), där där α är elementets värde (J, L_e).

$$f(t) = \frac{1}{\alpha} \int^t e(\tau) d\tau \tag{12}$$

Sätt

$$x = f \Rightarrow \dot{x} = \frac{e}{\alpha} \tag{13}$$

Konstitutiva samband ger ekvationerna

$$V_{Re} = Ri(t) \tag{14}$$

$$T_B = B\omega \tag{15}$$

Enligt regel blir summan av alla intensiteter i en seriell knutpunkt noll.

$$0 = V_{Le} = V_{in} - V_{Re} - V_{ems} \tag{16}$$

$$0 = T_{in} - T_B - T_{ut} - T_J \tag{17}$$

Insättning av (16) och (17), i (13) ger

$$\frac{d}{dt}i(t) = \frac{V_{in} - V_{Re} - V_{ems}}{L_e} \tag{18}$$

$$\frac{d}{dt}\omega(t) = T_J = \frac{Tin - T_B - T_{ut}}{J} \tag{19}$$

Insättning av (14), (9), och (15),(8) i (18) respektive (19) ger slutligen de ordinära differentialekvationerna

$$L_e\frac{d}{dt}i(t) = V_{in} - R_e i - \omega K_a N \tag{20}$$

$$J\frac{d}{dt}\omega(t) = iK_a N - B\omega(t) - T_{ut} \tag{21}$$

Nästa steg är att ta fram delsystemens överföringsfunktioner och skapa ett blockschema. Överföringsfunktionerna kommer sedan att implementeras i Matlab och Simulink för simulering av systemet. Laplacetransformation av ekvationer 20 och 21 ger överföringsfunktioner av de olika delsystemen i frekvensplanet.

$$L_e sI(s) - i(0) = V_{in} - R_e I(s) - \Omega(s)K_a N \tag{22}$$

$$Js\Omega(s) - \omega(0) = I(s)K_a N - B\Omega(s) - T_u t(s) \tag{23}$$

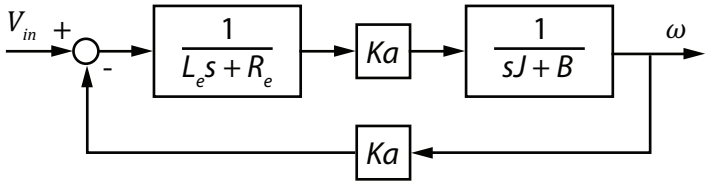
Begynnelsevärden för ström $i(0)$ och varvtalet $\omega(0)$ är noll. Ekvationerna utvecklas till:

$$L_e sI(s) - i(0) = V_{in} - R_e I(s) - \Omega(s)K_a N \Rightarrow I(s) = \frac{V_{in}(s) - \Omega(s)K_a N}{L_e s + R_e} \tag{24}$$

$$Js\Omega(s) - \omega(0) = I(s)K_a N - B\Omega(s) - T_{ut}(s) \Rightarrow \Omega(s) = \frac{I(s)K_a N - T_{ut}(s)}{L_e s + R_e} \tag{25}$$

Där $\frac{1}{L_e s + R_e}$ och $\frac{1}{Js+B}$ är de mekaniska respektive elektriska delsystemens överföringsfunktioner. Systemet kan nu enkelt beskrivas med följande blockschema. Överföringsfunktionen för hela systemet tas fram med räkneregel för blockschema av det återkopplade systemet, där $X(s)$ är börvärdet och $Y(s)$ ärvärdet.

$$\begin{aligned} Y(s) = X(s)\frac{G(s)}{1 + G(s)L(s)} \implies \\ \frac{Y(s)}{X(s)} = H(s) = \frac{G_1(s)G_2(s)G_3(s)}{1 + G_1(s)G_2(s)G_3(s)G_4(s)} \end{aligned} \tag{26}$$



Figur 17: Blockschema för DC-motor utan last och utväxling

Där $G_1(s)$, $G_2(s)$, $G_3(s)$ och $G_4(s)$ är,

$$G_1(s) = \frac{1}{L_e s + R_e} \tag{27}$$

$$G_2(s) = \frac{1}{Js + B} \tag{28}$$

$$G_3(s) = K_a \tag{29}$$

$$G_4(s) = K_a \tag{30}$$

Resultatet blir DC-motorns överföringsfunktion, med position som utsignal

$$G_{motor}(s) = \frac{K_a N}{s((L_e s + Re)(Js + B) + K_a^2 N^2)} \tag{31}$$

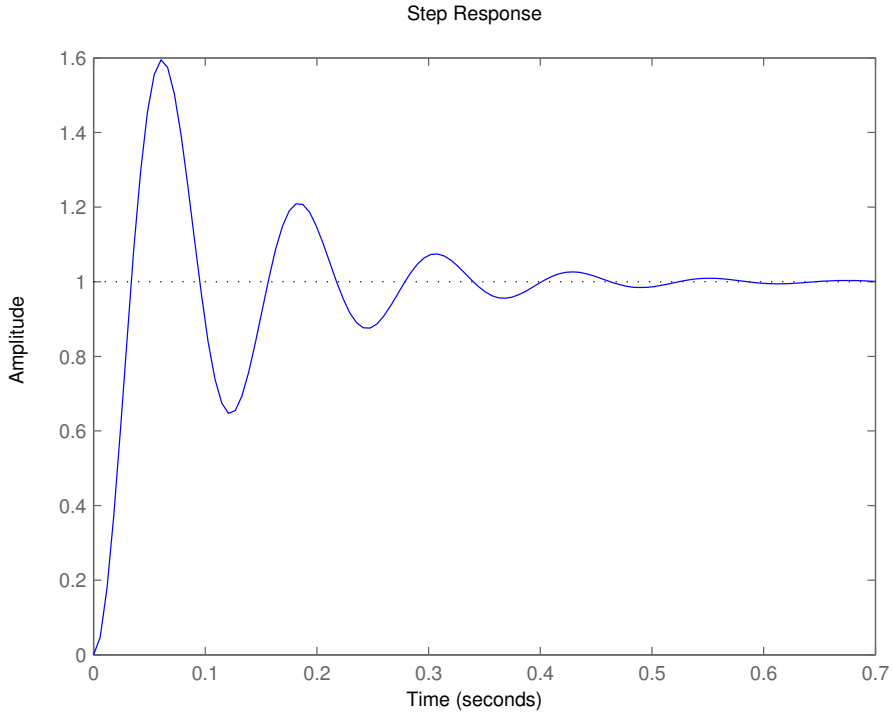
3.9 Regulatordesign

Till skillnad från motorn i gripklon styrs displayens motor efter gripklons position. En referenssignal skickas till displayens motor vars position ska ställas in efter denna. Parametrar från en Maxon RE25 DC motor [17] används då data för de valda motorerna inte finns att tillgå. Maxon-motorn har liknande egenskaper som den valda motorn.

Stegsvaret från utsignal till referens för det återkopplade systemet $G_{k.feedback}(s) = \frac{K_c G_{motor}(s)}{1+K_c G_{motor}(s)}$ med en förstärkning $K = 1$ visar en stigtid på 0.0228 och en insvängningstid på 0.443 s med 59 % översläng.

Latensen i nätverket innebär att en dödtid introduceras i systemet då referenssignalen $R(s)$ blir $R(s)e^{-sL}$ och överföringsfunktionen för återkoppling av systemet $G(s) = F_{disp}(s)G_{motor}(s)$ blir

$$\begin{aligned} Y(s) &= (R(s)e^{-sL} - Y(s))F_{disp}(s)G_{motor}(s) \\ \implies Y(s)(1 + F_{disp}(s)G_{motor}(s)) &= R(s)e^{-sL}F_{disp}(s)G_{motor}(s)\frac{Y(s)}{R(s)} \\ \implies G_{reg.feedback}(s) &= \frac{F_{disp}(s)G_{motor}(s)e^{-sL}}{1 + F_{disp}(s)G_{motor}(s)} \end{aligned} \tag{32}$$



Figur 18: Stegsvar för det återkopplade systemet $G_{k.feedback}(s)$

Då fördröjningen L ligger utanför den slutna reglerloopen påverkas inte regleringen av denna.

Det återkopplade systemets bodediagram visar en resonanstopp vid högre frekvenser vilket förklarar det återkopplade systemets svängiga beteende. För att åtgärda detta används en fasavancerande länk eller lead-filter. Ett lead-filter har överföringsfunktionen

$$K\frac{s+p}{s+z} \tag{33}$$

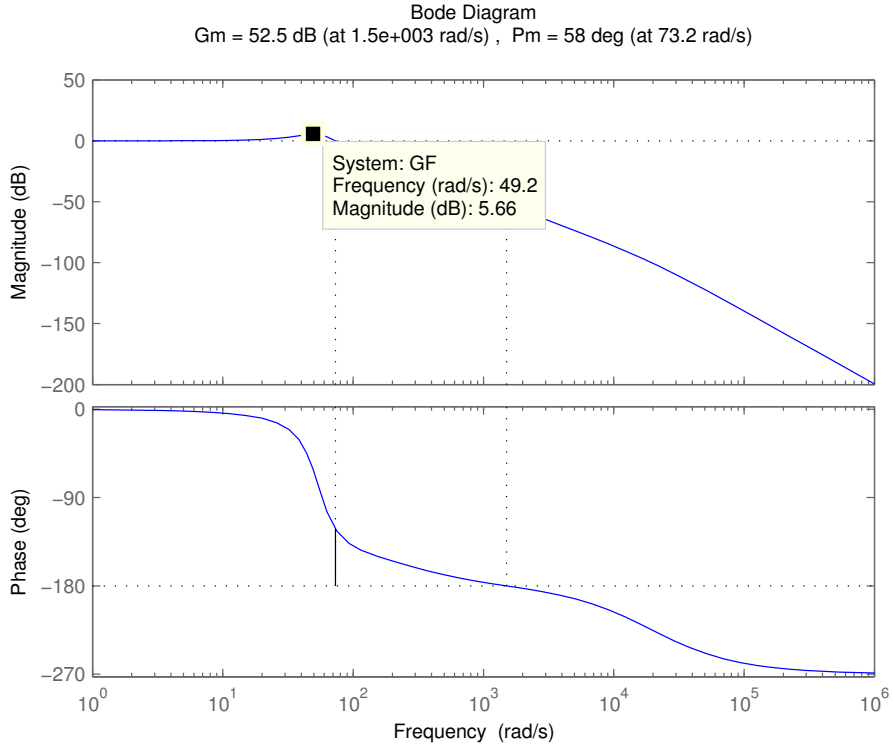
vilken även kan skrivas på formen

$$K\frac{1}{\alpha}\frac{\tau s+1}{\tau s+\frac{1}{\alpha}} \tag{34}$$

där

$$\tau = \frac{1}{z} = \frac{1}{\alpha p} \tag{35}$$

Designmetodiken är som följer:



Figur 19: Bodediagram för det återkopplade systemet $G_{k.feedback}(s)$

1. Sätt förstärkning innan reglering till 1 och observera det kvarstående felet. K_c väljs utifrån $K_c = e_{kvarstendefel}/e_{nskatfel}$, alternativt väljs K_c med sambandet:

$$K_c = \frac{1}{|G(j\omega_k)|\sqrt{\frac{1}{\alpha}}} \tag{36}$$

2. Bestäm faslyftet Φ_{max} som krävs av lead-filtret för att uppfylla designkravet på fasmarginal för $L(s) = F(s)G(s)$

3. Bestäm förstärkningen vid frekvensen ω_{max}

4. Beräkna polen z_c samt nollstället p_c

Designkriterier: Maximalt kvarstående fel efter reglering 2% Fasmarginal $F_{disp}(s)G(s)$ 55°. Insvängningstid ≤ 0.1 s.

Fasmarginalen för $G(s)$ avläses från bode-diagramet i figur 20 till Φ_{PMsys} = 18.5 grader. Faslyftet som krävs av lead-filtret blir därmed

$$\Phi_{max} = \Phi_m + 10^\circ - \Phi_{m.sys} \implies \Phi_{max} = 55^\circ + 10^\circ - 18.5^\circ = 46.5^\circ \quad (37)$$

Där 10° är en rekommenderad säkerhetsmarginal. α kan nu beräknas med sambandet

$$sin(\alpha) = \frac{1-\alpha}{1+\alpha} \implies \alpha = \frac{1 - sin(\Phi_{max})}{1 + sin(\Phi_{max})} = 0.1591 \quad (38)$$

Lead-filtret kommer att öka magnituden av $G(s)$ med en faktor $10log_{10}(\frac{1}{\alpha})$ vid frekvensen ω_{max} där det maximala faslyftet sker, vilket leder till att överkorsningsfrekvensen för $L(s) = F(s)G(s)$ ökar. För att överkorsningsfrekvensen för det kompenserade systemet ska hamna på ω_{max} väljs överkorsningsfrekvensen för leadfiltret ω_k där $|G(s)| = -10log_{10}(\frac{1}{\alpha})$.

$$|F(s)| = 10log_{10}(\frac{1}{\alpha}) = 7,98dB \quad (39)$$

Från bodediagramet i fig 20 avläses frekvensen ω_k för $|G(s)|$ = -7,9832 dB till 81.4 rad/s.

Polerna för lead-filtret kan nu tas fram med sambanden

$$z = \omega_k\sqrt{\alpha} \implies z = 81.4\sqrt{0.159} = 32.484 \quad (40)$$

$$p = \frac{z}{\alpha} \implies p = \frac{32,484}{0.159} = 204 \quad (41)$$

Slutligen väljs lämpligt värde för förstärkningen K_c.

Metod 1:

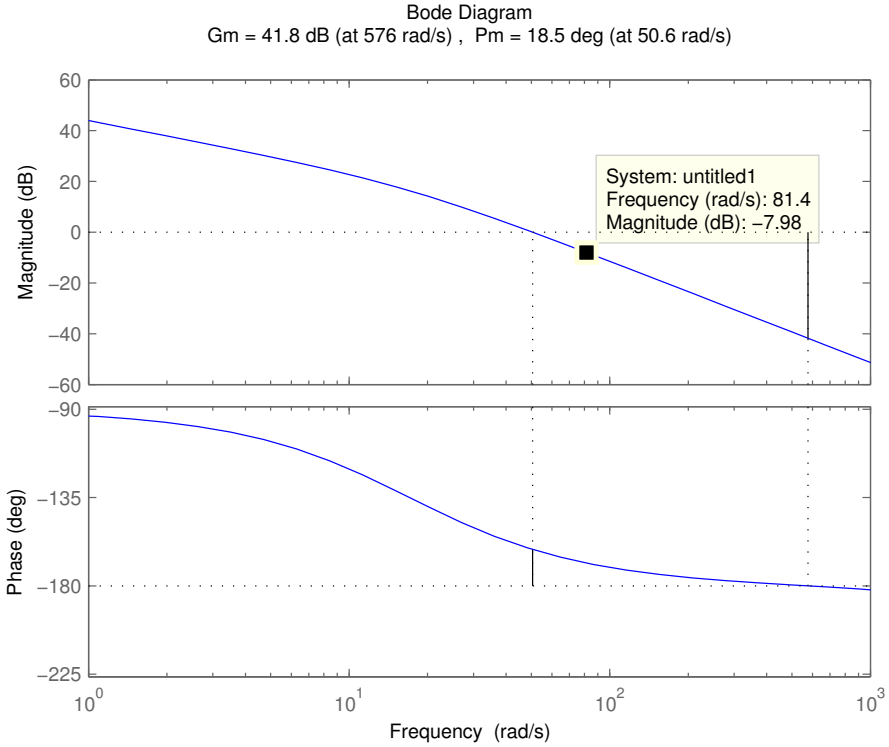
$$K_c = \frac{1}{|G(j\omega_k)|\sqrt{\frac{1}{\alpha}}} \implies \frac{1}{6.28058 * \sqrt{\frac{1}{0.1591}}} = 0.064 \quad (42)$$

Figur 21 visar att systemet med K_c framtaget med metod 1 är överdämpat.

Metod 2: Specificera krav på maximalt kvarstående fel efter reglering och beräkna K_c utifrån nuvarande kvarstående fel med sambandet

$$K_c = \frac{e_{motorfel}}{e_{nskatfel}} \quad (43)$$

där $e_{motorfel}$ är



Figur 20: Bodediagram för det öppna systemet $K_c G_{motor}(s)$

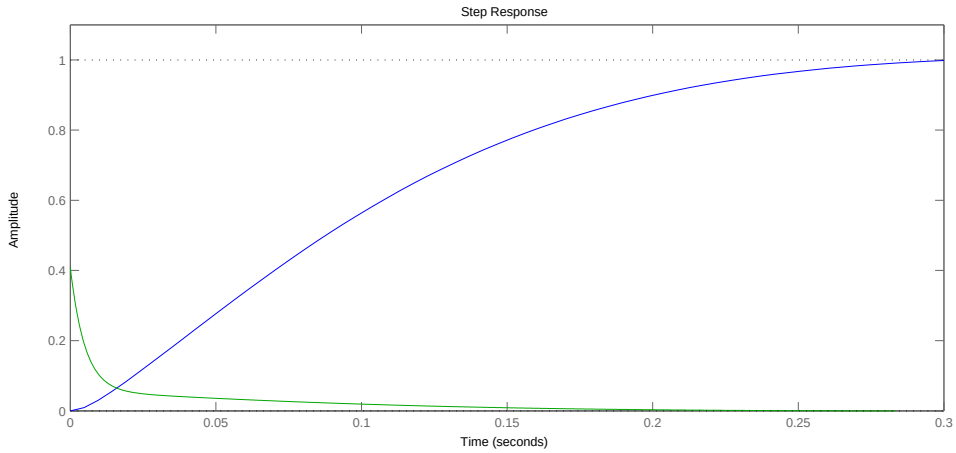
$$\lim_{s\to 0} sE(s) = \frac{1}{1 + G_{motor}(s)} R(s) = 0.0036 \tag{44}$$

Enligt kravet för kvarstående fel samt ekvation 43 fås

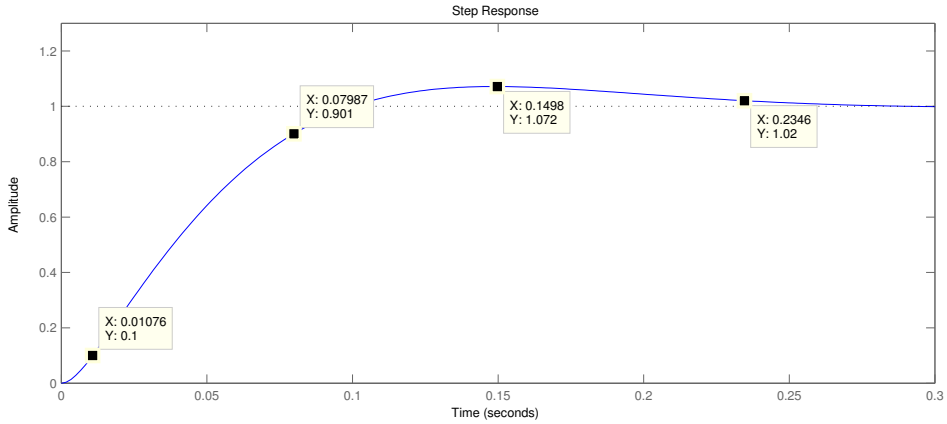
$$K_c = \frac{0.0036}{0.02} = 0.18 \tag{45}$$

Lead-filtret blir då

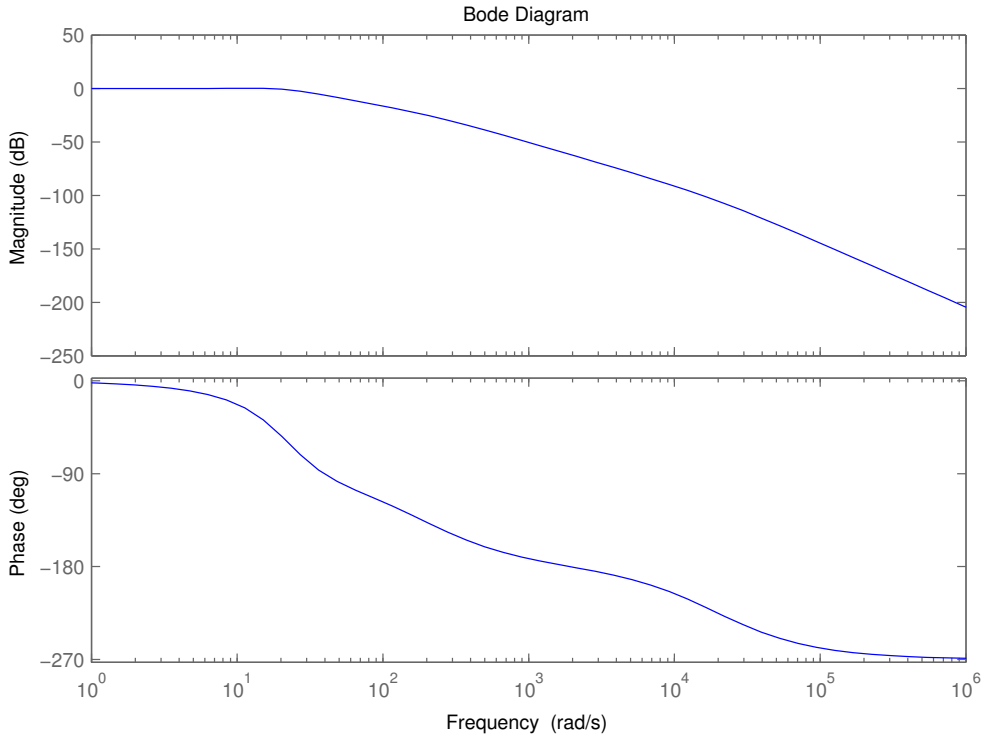
$$F(s) = 0.18 \frac{s + 32.484}{s + 204} \tag{46}$$



Figur 21: Stegsvar för $G_{reg.feedback}(s)$ för $K_c = 0.064$



Figur 22: Stegsvar för reglerat system



Figur 23: Bodediagram för reglerat system

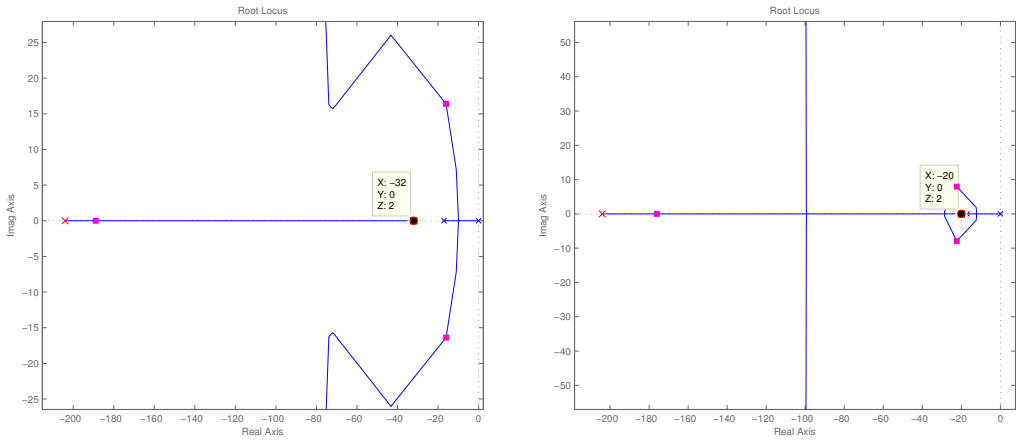
Från stegsvaret och bodediagramet för det återkopplade systemet med leadfilter ses en avsevärd förbättring av transienten och svängningar i systemet har eliminerats helt. Notera att resonanstoppen har jämnats ut i bodediagramet.

	$K_cG_{motor}(s)$	$F_{disp}(s)G_{motor}(s)$
$t_{stigtid}$	0.0235 s	0.006911 s
Översläng	59%	10%
$t_{insvängning}$	0.443 s	0.2346
Φ_{PM}	18.5°	66.4°
A_m	41 dB	61.5 dB

Tabell 4

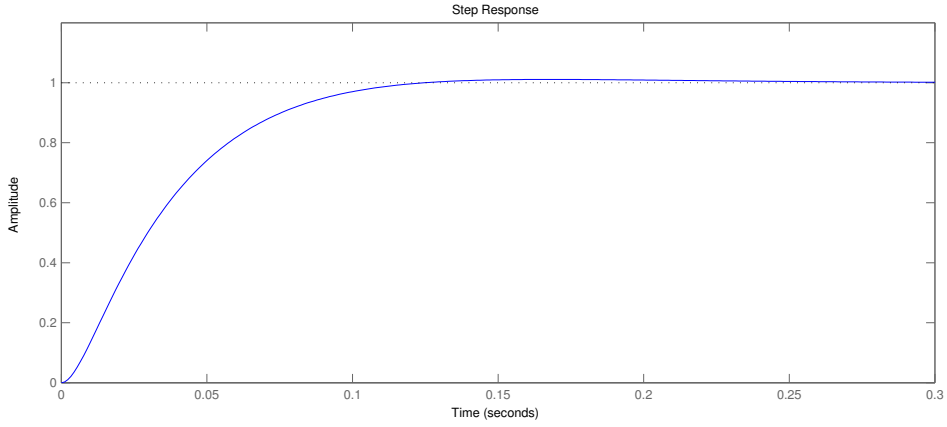
3.10 Trimning med Sisotool

Med Matlabs inbyggda program Sisotool kan man på ett enkelt sätt justera det återkopplade systemets poler via rotorten och se hur det påverkar systemets stegsvar i realtid. Regulatorn som togs fram i föregående kapitel implementeras här och förbättras något genom att flytta nollstället närmare imaginäraxeln från -32 till -20.



(a) (b)

Figur 24: Rotort för $F_{reg}(s)G_{motor}(s)$



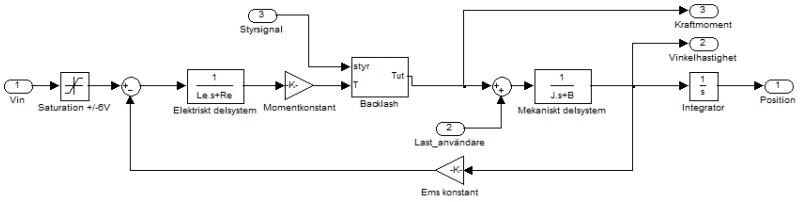
Figur 25: Stegsvar för reglerat, trimmat system

	$K_cG_{motor}(s)$	$F_{disp}(s)G_{motor}(s)$	$F_{lead.trim}(s)G_{motor}(s)$
$t_{stigtid}$	0.0235 s	0.006911 s	0.00666 s
Översläng	59%	10%	1.1%
$t_{insvängning}$	0.443 s	0.2346 s	0.1065 s
Φ_{PM}	18.5°	66.4°	78°
A_m	41 dB	61.5 dB	58 dB

Tabell 5

3.11 Trimning av PID Regulator i Simulink

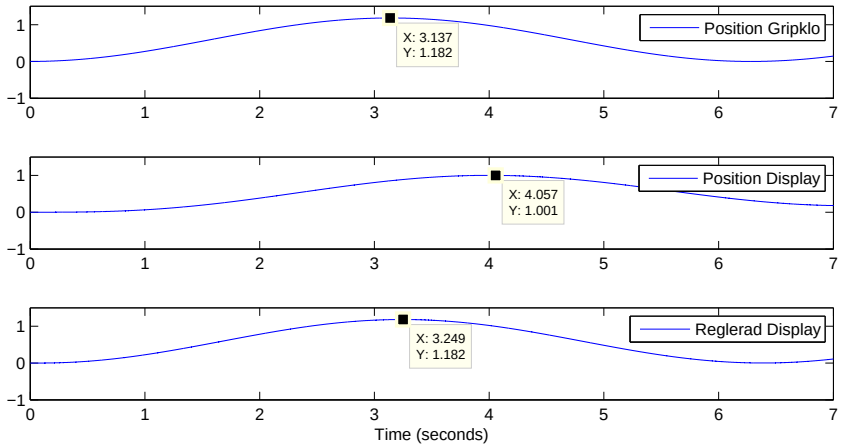
I föregående kapitel designades regulatorn lokalt. Det är intressant att titta på hela systemet från gripklo till display med fördröjningar inräknade i återkopplingen mellan de båda systemen, samt olinjäriteter som saturation och backlash(se bilaga C). En alternativ regulatordesign tas fram med Simulinks inbyggda trimningsverktyg för PID-regulatorer. Backlash modelleras här med en fördröjning mellan det elektriska och mekaniska delsystemet varje gång styrsignalen ändrar tecken.



Figur 26: Simulink-modell av $G_{motor}(s)$

Figur 26 visar G_{motor} implementerat i simulink med backlash och saturation.

När kraftsignalen ändrar tecken stoppar backlash blocket output i någon sekund. Detta simulerar att vridmomentet inte ger någon ändring i vinkelhastighet under en viss tid på grund av glappet i växellådan.



Figur 27: Stegsvar för Gripklo, display utan trimning samt display med trimmad reglering

Från stegsvaret kan man se effekten av P reglering på displayen. Trots fördröjningar och backlash följs gripklons position med nogrannhet. Det optimala värdet för P-verkan erhålls med PID blocket i simulink.

	$K_1 G_{motor}(s)$	$K_{19.55}(s) G_{motor}(s)$
$t_{stigtid}$	2.23 s	0.108 s
Översläng	0%	0%
$t_{insvängning}$	3.97 s	0.193 s
Φ_{PM}	90°	90°
A_m	85.9 dB	59.6 dB

Tabell 6

3.12 Diskretisering

För att implementera regulatorerna digitalt måste först tidsdiskreta modeller av dessa tas fram. En tidskontinuerlig överföringsfunktion kan diskretiseras till en motsvarande tidsdiskret överföringsfunktion med sambandet

$$s = \frac{z-1}{zT} \tag{47}$$

där T är samplingstiden. Det trimmade lead filtret som togs fram i kapitel Trimning med Sisotool blir då

$$F(s) = 0.18\frac{s+20}{s+204} \implies \frac{Y[z]}{X[z]} = 0.18\frac{[z-1]+20zT}{[z-1]+204zT} \tag{48}$$
$$\implies (1+204T)zY[z] - Y[z] = 0.18((1+20T)zU[z] - U[z])$$

Inverstransformen av 48 blir

$$y_{k+1} = \frac{0.18(1+20T)u_{k+1} - 0.18u_k + y_k}{1+204T} \tag{49}$$

där y_{k+1} är den nya styrsignalen beräknat från det samplade felet u_k och föregående styrsignal y_k. Variablerna u_k och y_k initieras till 0.

P regulatorn från kapitel Trimning med Simulink blir i tidsdiskret form

$$y_{k+1} = Ku_{k+1} \tag{50}$$

3.13 Datorkommunikation

För att de båda delsystemen, gripklo och display, skall kunna interagera med varandra krävs det att de kan kommunicera med varandra och berätta vilka tillstånd de befinner sig i. Kommunikationen utgår ifrån PC-datorer då de är vanligt förekommande och har bra nätverksstöd. Med bakgrund av detta används ett program för kommunikationssystemet mellan de båda enheterna. Kommunikationsprogrammets uppgift är att upprätta eller ta emot en förbindelse med den andra enheten och förmedla information mellan enheterna. För att underlätta för användaren används samma program oavsett om gripklo eller display kopplas in.

Enligt kravspecifikationen är en låg latens mellan gripklon och displayen önskvärd samtidigt som dataintegritet och minimal protokollbehandling i enheterna behövs. Transportlagret RUDP, som bl.a. använder tekniken överbuffring för låg latens, skulle vara ett bra val. Tyvärr är detta inte standard än och transportlagret är fortfarande under utveckling. DCCP som används idag av system som kräver låga latenser, t.ex. IP-telefoni, ger tillräckligt låg latens men är en relativt ny standard och goda utvecklingsmöjligheter till PC är bristfällig. Av de protokollen som finns att tillgå idag med bra stöd är UDP väldigt intressant vid en första överblick men om man stänger av Nagle's algoritm i TCP uppnår man likvärdig latens samtidigt som dataintegritet säkras och minimal ordningshantering av paketen behövs i programmen. Nackdelen är att datorn och nätverket belastas mer p.g.a. att många kontrollsummor skall beräknas och skickas. Detta leder dock inte till något

problem i systemet utan är en bra avvägning enligt kravspecifikationen och de tidigare beskrivna vinsterna med TCP utan Nagle's algoritm överväger. Utifrån dessa resonemang används TCP.

Det finns olika tillvägagångssätt för ett program att bekräfta att förbindelsen är önskvärd av båda sidor. Man kan t.ex. låta den lyssnande sidan manuellt acceptera förbindelsen. Detta kräver dock fysisk närvaro av en människa på båda sidor. En annan teknik är att välja ett förvalt lösenord som måste anges på den anropande sidan. Detta innebär att systemet kan användas smidigt av en person samtidigt som man inte kan störas av anropsbekräftelser från icke önskvärda anslutningar. Den senare tekniken är således mer lämplig för projektet och är även den som implementeras.

3.14 Testkörning

För att utvärdera huruvida gripdonet lever upp till förväntningar och för att finjustera regulatorer utförs ett par enkla experiment.

- En testkörning utan något objekt att greppa, för att undersöka hur följsamma display och gripklo är.
- Ett körning med en hård kropp att greppa. Detta experimentet testar att den grundläggande återkopplingen fungerar.
- Körningar med olika stora kroppar för att undersöka positionsåterkopplingen.
- Körningar med olika elastiska kroppar för att undersöka särskiljningsförmågan hos gripdonet.
- Slutligen testas om gripdonet klarar äggprovet, Är gensvaret från display tydligt och snabbt nog för att greppa ägget utan att krossa det?

4 Resultat

I detta avsnittet presenteras resultat från simuleringar och tester. Även mer detaljerade lösningar på under konstruktion uppstådda problem presenteras.

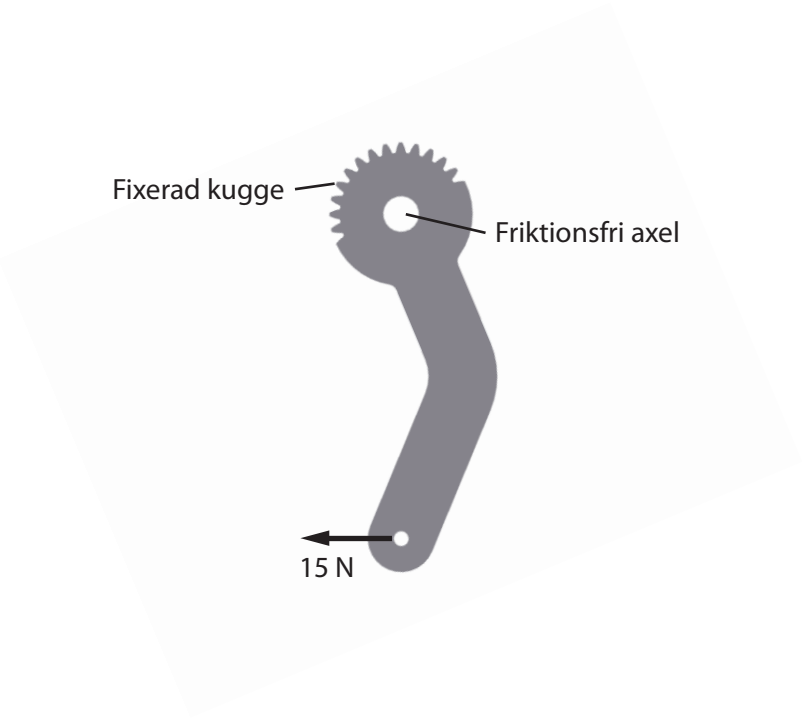
4.1 FEA-analys av gripfingrar

För att kontrollera gripfingrarnas hållbarhet utfördes hållfasthetssimuleringar i Autodesk Inventors inbyggda FEA-analys-mjukvara. Utgår man från motorns momentkrav på 1 Nm och räknar baklänges med gripfingrets hävarm

motsvarar det en kraft i fingertoppen på ca. 15 N:

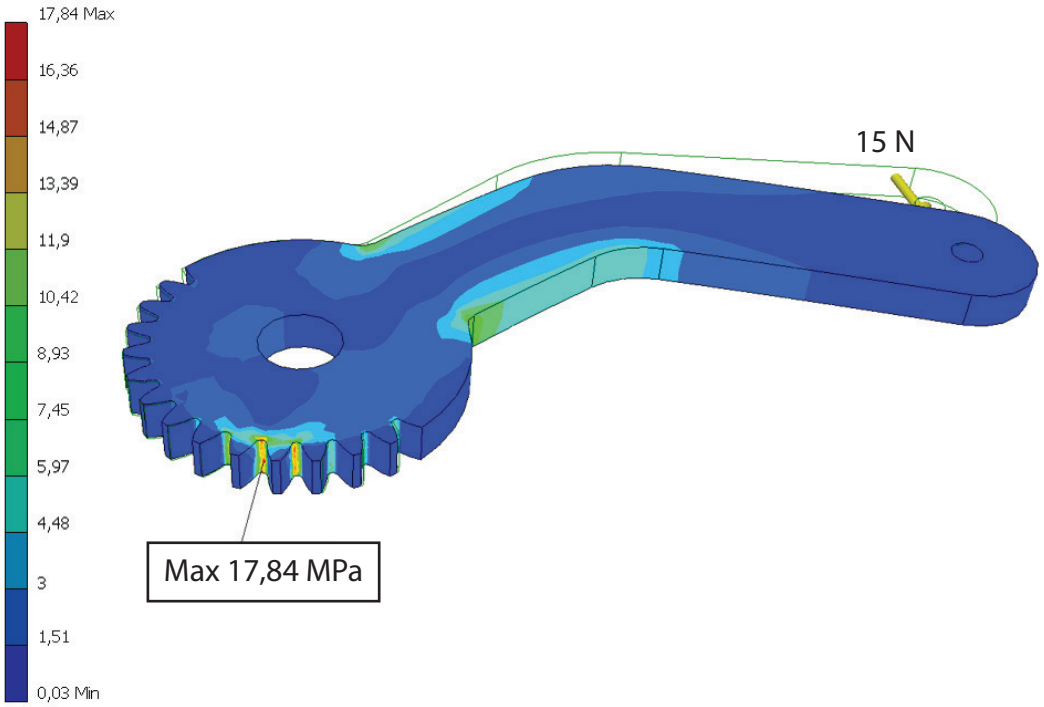
$$F = \frac{T}{L} = \frac{1}{0.07} \approx 14.3N \tag{51}$$

Simuleringen utfördes med ett gripfinger fäst på en friktionsfri axel där motorn eller den fria axeln sitter, en kugge fast inspänd samt kraften på 15 N pålagd längst ut i fästhålet på fingertoppen. Se figur 28. Materialdata för VeroWhite, som är det material som används i de delar som tillverkats med rapid prototyping finns tillgängligt på på tillverkarens hemsida [18]. VeroWhite har en brottspänning på 49.8 MPa. Resultatet av simuleringen, som



Figur 28: Inspänning av gripfinger

kan ses i figur 29, visar en maximal spänning på 17.84 MPa mitt mellan två kuggar. Även utböjningen vid rak belastning visas extremt överdrivet där trådmodellen är gripfingrets originalform.

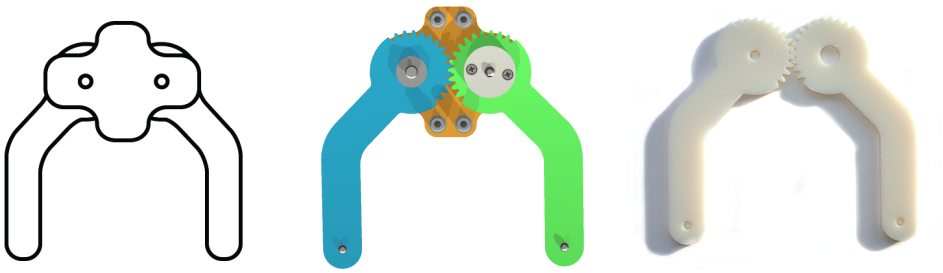


Figur 29: Simuleringsresultat

En ökning av kraften till det dubbla, 30 N, resulterar i en maxspänning i samma punkt på 36.16 MPa. Då detta fortfarande ligger under materialets brottspänning anses konstruktionen tillfredsställande dimensionerad. En ytterligare kraftökning till 40 N, skulle resultera i att spänningen i kuggen hamnar väldigt nära materialets brottspänning och ett materialbrott är sannolikt.

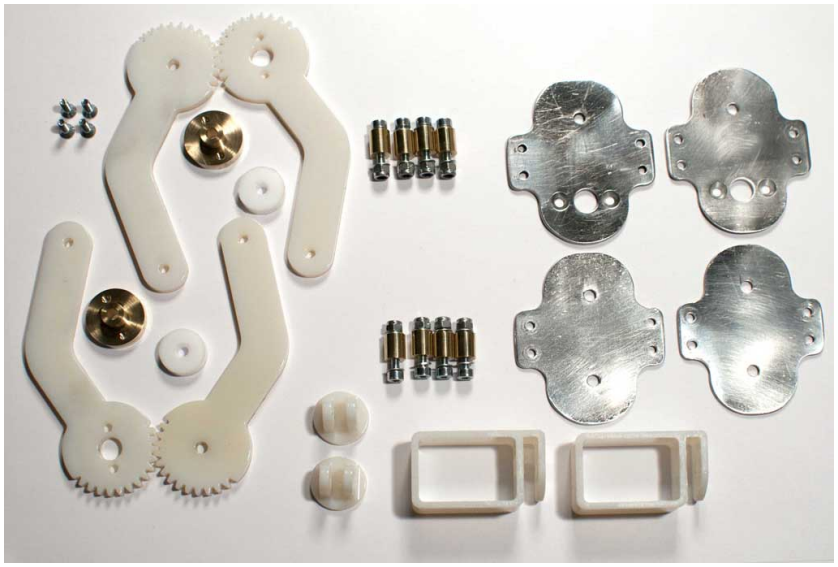
4.2 Prototyp

Figur 30 visar utvecklingen av displayens och gripklons gripfingrar från skiss (30a) till CAD-modell (30b) till fingrar färdiga att monteras (30c). Figur 31 visar delar till både gripklo och display färdiga för montering. Figur 32 visar en sammansatt bild av CAD-modellen av gripklon och den verkliga gripklon. Figur 33 visar gripdonet med all elektronik monterad. Figur 34 visar i detalj var elektronikdelarna sitter monterade.

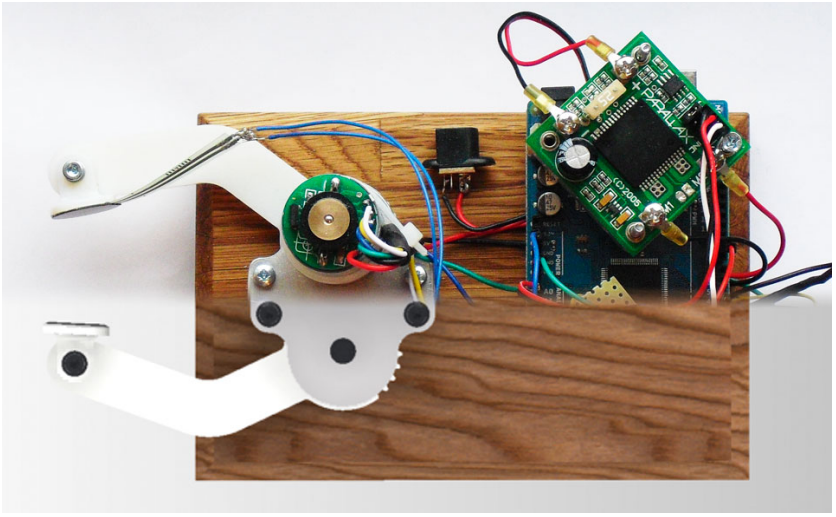


(a) Skiss (b) CAD-modell (c) Färdig prototyp

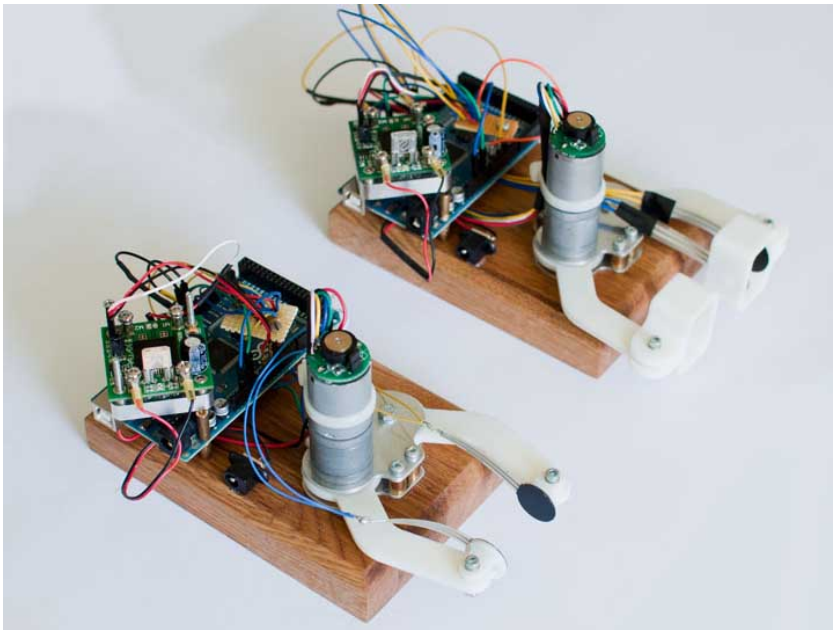
Figur 30: Utveckling från skiss till prototyp



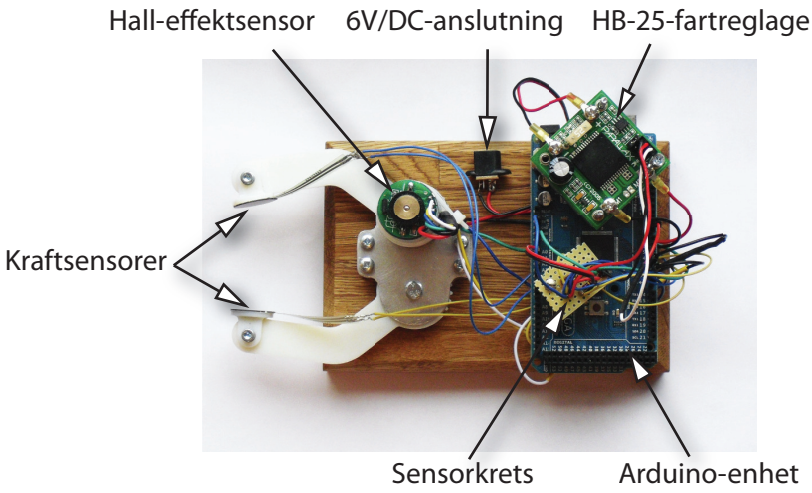
Figur 31: Delar för montering. Här med tidiga prototyper av fingerfästen.



Figur 32: 50-50, CAD-Verklighet



Figur 33: Gripdon med elektronik

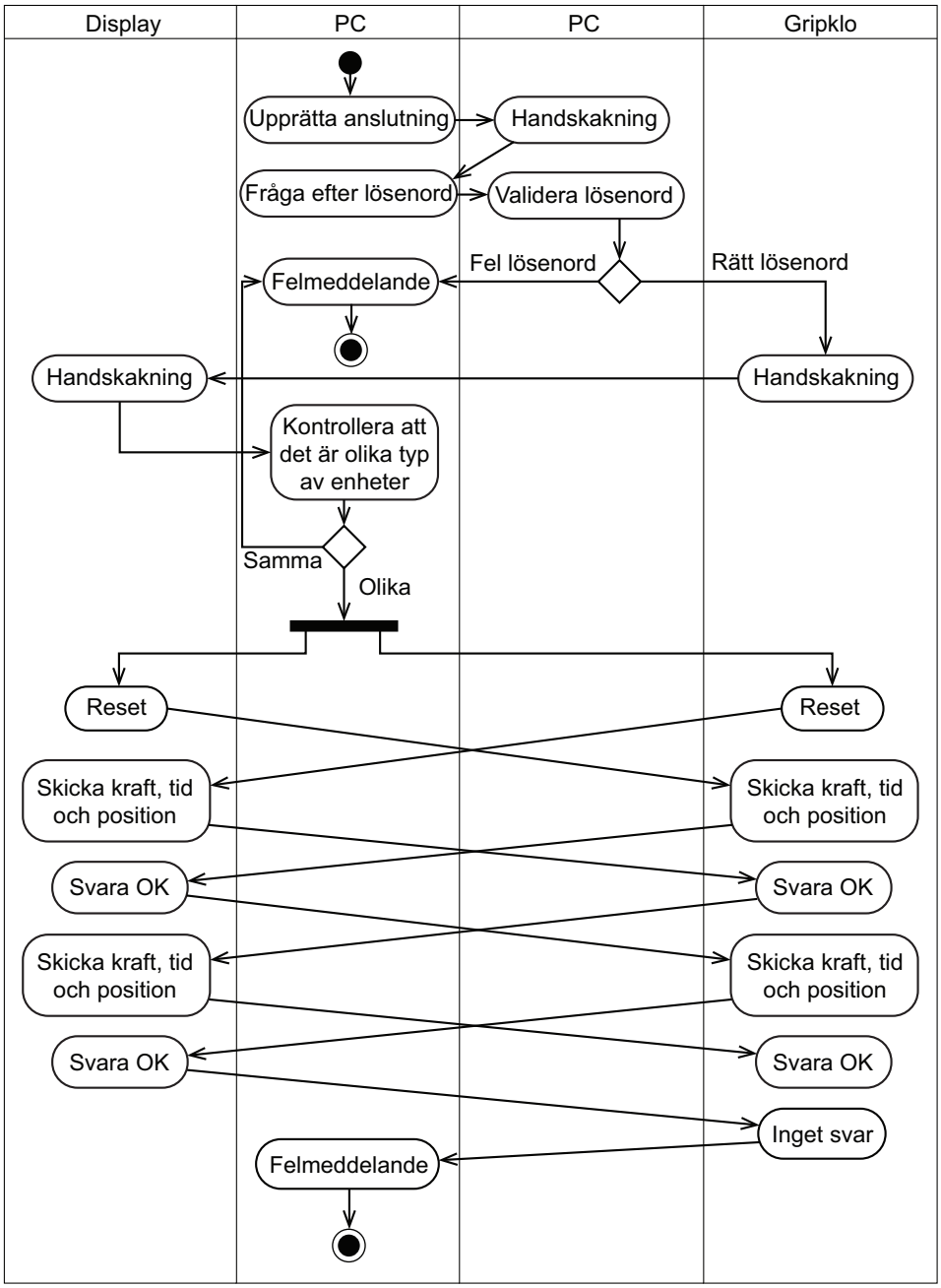


Figur 34: Gripklo med Arduino-enhet, fartreglage samt elektronik för sensorerna

4.3 Datorkomunikation

Varje haptisk enhet kopplas in till en dator. Dessa datorer måste ha ett program som fungerar som en kommunikationslänk mellan enheten och de andra enheterna. Det första som händer när man startar programmet är att man får välja lösenord för inkommande anslutningar och berätta var den haptiska enheten är inkopplad i datorn. Därefter sätts en initial kommunikation upp mellan programmet och Arduino-enheten. De kontrollerar att mjukvarorna de kör är kompatibla med varandra och vad det är för en sorts haptisk enhet, gripklo eller display. Den haptiska enheten återställs också till startposition. Nu är programmet klart för att ta emot eller upprätta förbindelser med andra enheter.

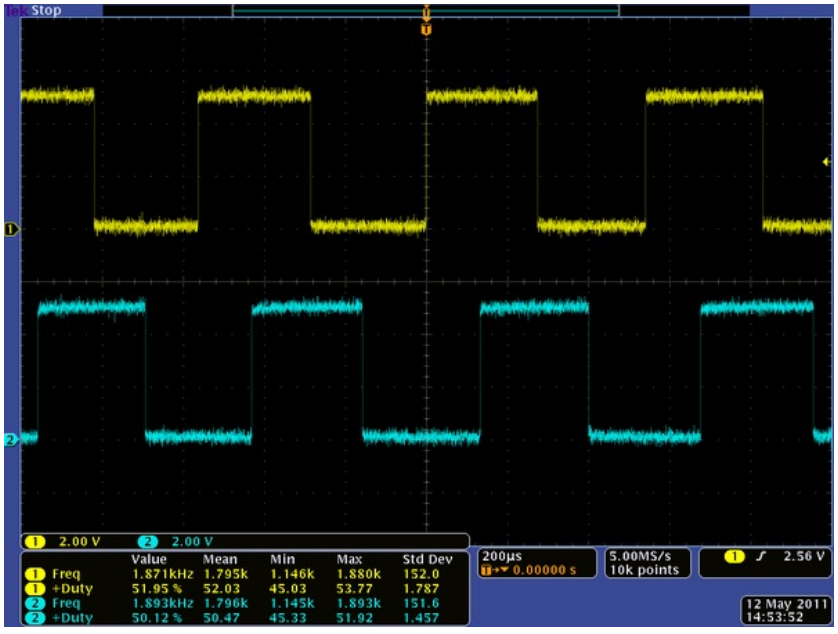
I det initiala skeendet i programmet, innan någon kontakt med andra enehter har uppstått, lyssnar programmet efter anrop samtidigt som den själv kan anropa andra. Ett utav programmen anropar den andra och upprättar en förbindelse som valideras med det förvalda lösenordet. Grundläggande felkontroller genomförs, t.ex. att programmen är av rätt version och att enheterna, gripklo eller display, i vardera ände är kompatibla med varandra. Efter att denna första handskakningen har ägt rum börjar system utbyta information med varandra. I figur 35 visas ett aktivitetsdiagram för PC-kommunikationen.



Figur 35: Aktivitetsdiagram för PC-kommunikationen

4.4 Styrsystem

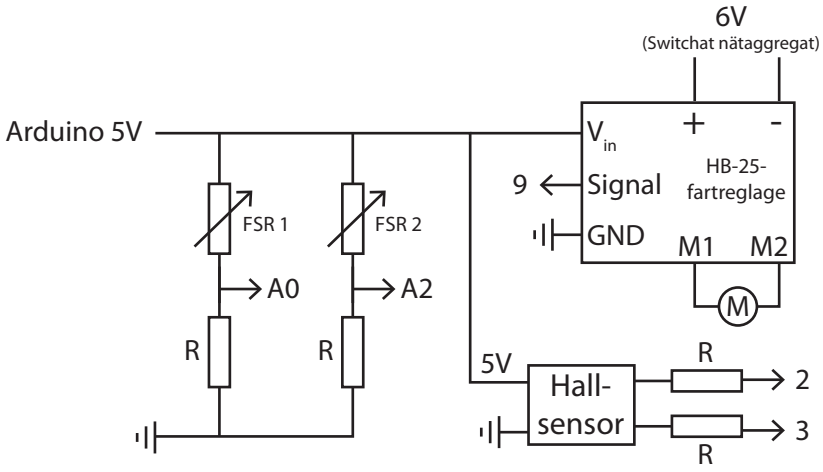
Arduino-enheternas syfte är att läsa av krafterna och reglera motorn. För att mäta kraften läses spänningsfallet över sensorn av analogt genom att koppla dem i serie med ett pull-down-motstånd . Spänningen läses av innan resistorn, se 37. Spänningsfallet är inte linjärt med kraften. Arduino-enheten omvandlar spänningsfallet med hjälp av en inversfunktion som är framtagen utifrån digrammen, 8, för kraftsensorns datablad. Motorn behöver kunna regleras på två sätt, dels via vridmoment och dels via position. Vridmomentet är proportionellt mot spänningen över motorn och kan enkelt regleras via pulståg till fartreglaget. Positionen däremot är mer beräkningstung. Först måste enheten veta var motorn är. Genom att först referensköra motorn och sedan ta reda på den relativa förflyttningen kan detta uppnås. Den relativa förflyttningen registreras av Hall-sensorn som ger två fyrkantsvågor som är fasförskjutna 90 grader, se bild 36. Dessa ger då en unik signalkombination om motorn går framåt eller bakåt. Interrupts i Aruino-enheten används för att registrera dessa förändringar och beräkna positionsförändringen. För att referensköra enheten körs den försiktigt till stängt läge, då inte någon positionsförändring registreras inom ett givet tidsintervall vet enheten att armarna är stängda och positionen nollställs.



Figur 36: Signaler från Hall-effektsensor

Figur 37 visar kopplingsschemat för styrsystemet, vilket är samma för

display och gripklo. M är motorn och siffrorna visar ingångarna på Arduinoenheten. Resistanserna R är 100 kΩ.



Figur 37: Sensorkrets

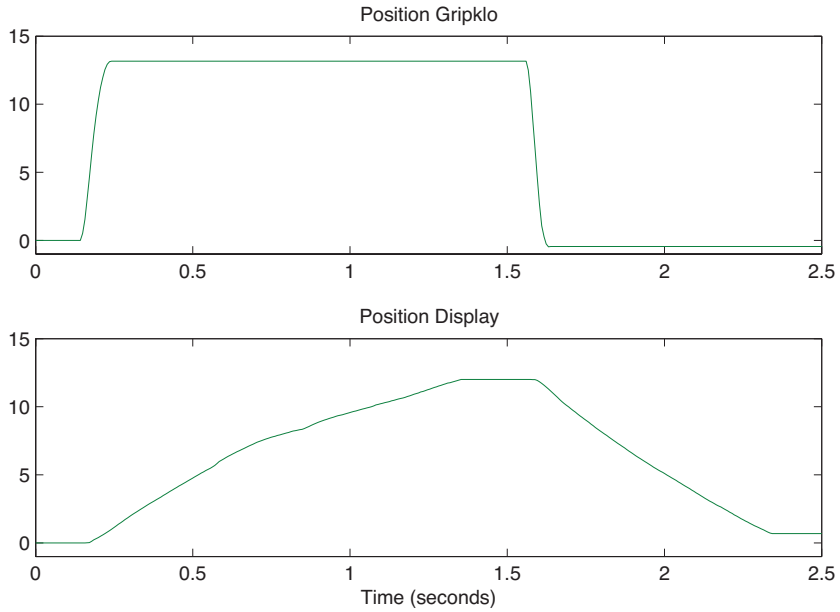
4.5 Testkörning

De framtagna regulatorerna implementerades med C-kod i Arduinoenheten och systemet testkördes. Lead-filtret gav till en början en något för låg styrsignal till displayen, vilket kan ses i figur 38. Displayen rör sig väldigt långsamt till referenspunkten.

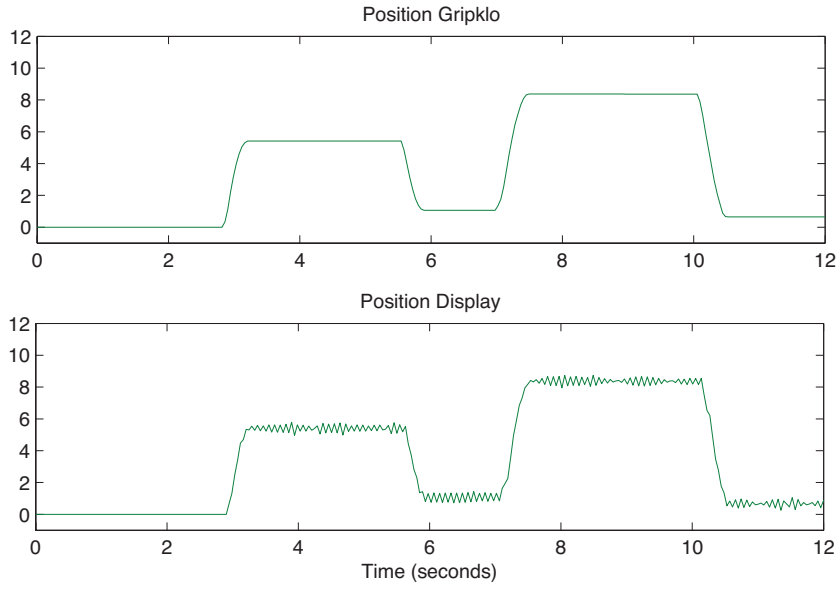
Förstärkningen K_c ökades och polen flyttades närmare imaginäraxeln. Resultatet kan ses i 39. Displayen går trots latensen i nätverket snabbt till referensposition, men oscillerar kring en referenspunkt.

De bästa resultaten erhölls dock när all D-verkan togs bort. Resultatet från tester utförda med P-reglering kan ses i figur 40. Värden från simuleringsresultat i simulink användes.

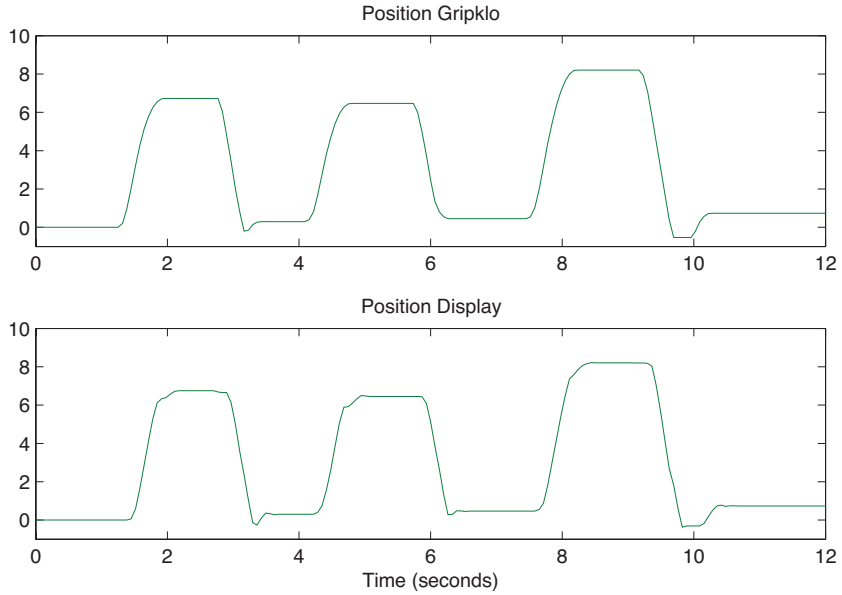
Gripdonets båda enheter följer handens rörelser bra. Då gripklon greppar en fast kropp känns detta mycket tydligt, och storleken på objektet kan lätt avgöras. Elasticitet är svårt att känna, men ett visst motstånd uppfattas. Slutligen klarar gripdonet av äggtestet, att greppa ett ägg utan att krossa det. Ett mycket tydligt stopp känns i displayen.



Figur 38: Enheternas positioner då lead-filtret används.



Figur 39: Enheternas positioner då det förändrade lead-filtret används.



Figur 40: Enheternas positioner då endast P-regulator används.

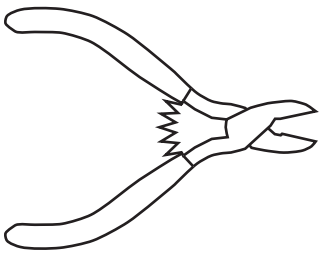
5 Diskussion och slutsatser

Syftet med kandidatarbetet var att genom konstruktion och experiment undersöka haptisk teknologi. Detta har genomförts och resultatet är ett någorlunda fungerande haptiskt gripdon. Ett antal mätbara krav på gripdonet sattes upp. I detta avsnittet diskuteras hur väl dessa mål uppnåtts och hur andra lösningar hade kunnat förbättra gripdonet.

5.1 Konstruktion och motorglapp

Överlag fungerar konstruktionen väl. Mekanismen fungerar relativt friktionsfritt, kuggarna har bra ingrepp. Bottenplattan i trä hade kunnat göras större för att få bättre plats med elektroniken.

Motorn klarar varvtal- och momentkraven, men det finns dock ett glapp i växellådan som resulterar i stora oönskade rörelser i gripfingrarna. Glapp eller backlash försvårar regleringen och gör gör att användarupplevelsen försämras. Greppas ett stort föremål kan det kännas som att man greppar ett litet föremål. I efterhand kan diskuteras om motorerna verkligen följer kravspecifikationen, tabell 1. Kravet på upplösning i fingerpositionen borde innefatta att motorns växellåda inte har ett för stort glapp. Man kan ställa om gripfingrarnas positioner i mjukvaran så att skillnaden i glappet justeras så att det vid ett ”normalt” grepp, alltså inte en öppnande rörelse, blir korrekt. Ett annat förslag är att flytta Hall-sensorn till växellådans axel istället för motorns. Upplösningen av axelpositionen skulle försämras, men tack vare att sensorn sitter direkt ansluten till gripfingrarna skulle kanske en bättre positionsangivelse uppnås. Men då glappet är så pass stort är den bästa lösningen att byta ut motorerna. Man kan dock låsa glappet i olika lägen för gripklon och displayen för att sedan simulera en automatisk tillbakagång av gripklon, likt en återfjädrande tång, se 41. Glappet kommer då alltid att ligga i ändläge på båda enheter.



Figur 41: Tång

5.2 FEA-analyser

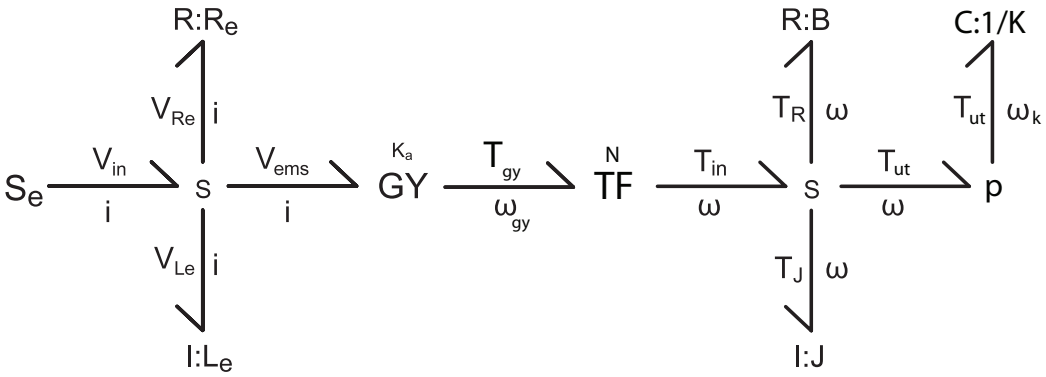
Som framgår av FEA-simuleringarna håller gripfingrarna för det specificerade momentet. Inga förstörande experiment på detaljer tillverkade med rapid prototyping har utförts. Skulle någon del gå av så går det snabbt att ta fram en ny. Det finns i dagsläget inte utrymme i konstruktionen att öka tjockleken på gripfingrarna för att på så sätt öka hållfastheten. Motoraxelns längd och mässingsfästet begränsar tjockleken. Det skulle eventuellt vara möjligt att konstruera om kuggarna så att de blir större och färre.

FEA-simuleringar har gjorts utan fingerhållarna eller sensorplattorna monterade. Tidsbrist begränsade FEA-simuleringarna till att endast innefatta ett separat gripfinger. De första prototyperna till fingerhållare gav vid belastning ett vridande moment i fingrets axel i gripfingrarna vilket efter verklig montering uppfattades som stort. Den senaste versionen av fingerhållare ger vid belastning endast en belastning i gripfingret motsvarande det som presenteras i FEA-analysen.

5.3 Reglerstrategi

Vid testkörningar har den synkrona modellen använts. Den asynkrona gav, som befarat, att enheterna befann sig i helt olika lägen trotts referenskörning. Detta beror antagligen på att krafterna inte kan uppmätas exakt, utan de uppmätta krafterna beror på var och hur sensorerna belastas. Lead filtret som togs fram i kapitel reglerdesign gav till början dåliga resultat i testkörningen. Anledningen till detta är troligtvis att modellen inte tog backlash, fördröjningar i hela det återkopplade systemet samt motorns saturation i åtanke. Lead-filtret kan skrivas om som en PD-regulator med ett lågpassfilter. Polens avstånd från imaginäraxeln leder till att filterkonstanten N blir runt 20, vilket är på gränsen för rekommenderade värden. Då högfrekventa störningar förstärks med en faktor KN kan detta vara orsaken till oscillationer i 39 efter höjning av förstärkningen K_c. Växellådans backlash var mycket större än väntat för den motortypen. I regulatordesignen hade man räknat med backlash som var ungefär fem gånger mindre än den verkliga. Simuleringar i simulink med backlash, saturation och fördröjningar medräknade stämde bäst överrens med resultat från testkörningar, vilket kan ses i figur 40 där regulatorn från resultat i simulink användes.

Rekommendationer för framtida arbeten är att använda en motor med mindre utväxling, alternativt använda gummisnoddar för att mildra glappet i växellådan på gripklon, och byta ut motorn i displayen. Gripklons motormodell måste skrivas om då gummisnoddarna fungerar som en fjädring(flödeslagring) som sänker rotorns vinkelhastighet.



Figur 42: Bindningsgraf med fjädring

I figur 42 har en fjädring lagts till i bindningsgrafen där T_{ut} kan uttryckas på formen

$$T_{ut} = K\theta \tag{52}$$

Parametrar för referensmotorn Maxon RE25 DC fungerade bra. Dock är massströghetsmomentet J mindre i den simulerade motorn än i det verkliga fallet då rotorn i den använda motorn är större än den i Maxon RE25 DC. Rekommenderat är att använda samma systemparametrar för motorn, men att skala ned tidskonstanten $T_i = \frac{B}{J}$, alternativ räkna fram ett nytt J om motorn finns till hands.

5.4 Summering och tack

Att utveckla ett fungerande haptiskt gripdon, med allt vad det innebär i reglering, programmering och konstruktion har visat sig tämligen krävande. Men tack vare synergieffekten av gruppens samlade kunskaper nåddes målet och en fungerande prototyp kan uppvisas. Den kommer inte i närheten av de haptiska robotar som idag används, och som beskrivs i inledningen av denna rapporten, vad gäller precision men med de begränsade resurser gruppen hade att tillgå anses ändå resultatet gott.

Vi i kandidatgruppen vill passa på att tacka våra handledare Jonas Nilsson och Jonas Fredriksson för stödet under arbetet. Tack till Patriks syster Malin Rohman för goda litteraturtips.

Tack för oss!

Figur 43: Kandidaterna

Referenser

[1] R. A. Heinlein, *Waldo*. Doubleday, 1942.

[2] B. Davenport, R. A. Heinlein, C. M. Kornbluth, A. Bester, and R. Bloch, *The Science Fiction Novel: Imagination and Social Criticism*. Advent: Publishers, 1959.

[3] "About crl." `http://centres.com/company.htm`, 03.

[4] "Dextre successfully completes its first official job." `http://www.nasa.gov/mission_pages/station/expeditions/expedition26/dextre_firstjob.html`, 04 2011.

[5] R. Stone, "Haptic feedback: A brief history from telepresence to virtual reality," *Haptic Human-Computer Interaction*, pp. 1–16, 2001.

[6] M. Tavakoli, R. Patel, and M. Moallem, *Haptics for teleoperated surgical robotic systems*, vol. 1. World Scientific Pub Co Inc, 2008.

[7] B. Baker, "Ieds: detection disarmament and disposal," *Defence and Security Systems International*, vol. 4, 2011.

[8] L. Ljung, *Modeling of Dynamc Systems*. Prentice Hall, 1994.

[9] L. L. Peterson and B. S. Davie, *Computer Networks: A Systems Approach, Second Edition (The Morgan Kaufmann Series in Networking)*. Morgan Kaufmann, 1999.

[10] S. Keshav, *An Engineering Approach to Computer Networking: ATM Networks, the Internet, and the Telephone Network*. Addison-Wesley Professional, 1997.

[11] J. Nagle, "Congestion control in IP/TCP internetworks." RFC 896, January 1984.

[12] J. Postel, "Transmission control protocol," RFC 793, Internet Engineering Task Force, September 1981.

[13] E. Kohler, M. Handley, and S. Floyd, "Datagram Congestion Control Protocol (DCCP)." RFC 4340 (Proposed Standard), March 2006. Updated by RFCs 5595, 5596.

[14] D. Velten, R. Hinden, and J. Sax, "Reliable Data Protocol." RFC 908 (Experimental), July 1984. Updated by RFC 1151.

[15] V. Mathiowetz, N. Kashman, G. Volland, K. Weber, M. Dowe, S. Rogers, *et al.*, "Grip and pinch strength: normative data for adults," *Arch Phys Med Rehabil*, vol. 66, no. 2, pp. 69–74, 1985.

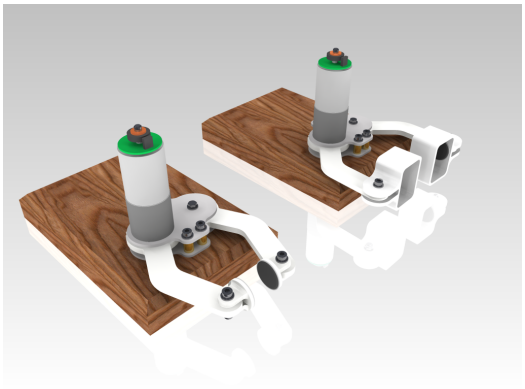
[16] "Fsr sensor tutorial, wiki." `http://www.ladyada.net/wiki/tutorials/learn/sensors/fsr.html`.

[17] "Maxon re25 precious metal brushes 10watt datasheet." `http://www.scribd.com/doc/30715497/Maxon-RE25-Precious-Metal-Brushes-10Watt`.

[18] "Objet rapid prototyping-material." `http://www.objet.com/3D-Printing-Materials`, 05.

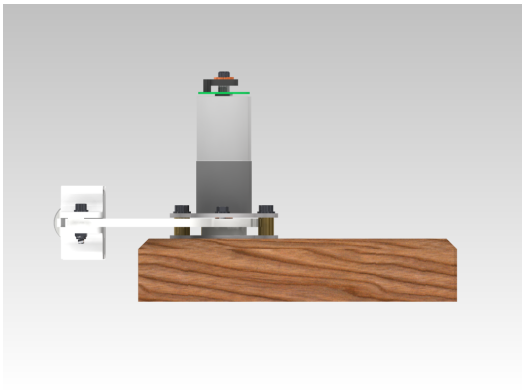
Bilaga A CAD-ritningar

A.1 Renderingar

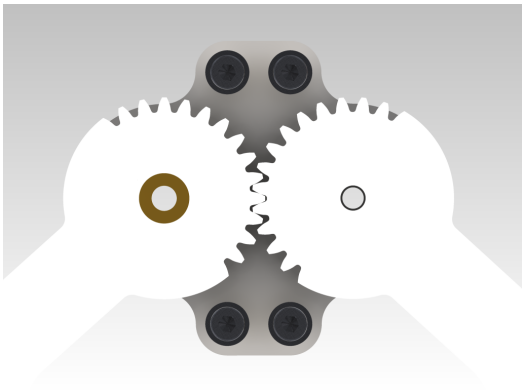


(a) Översiktsvy

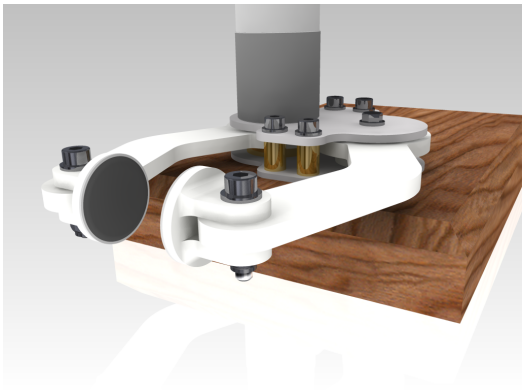
(b) Vy ovan



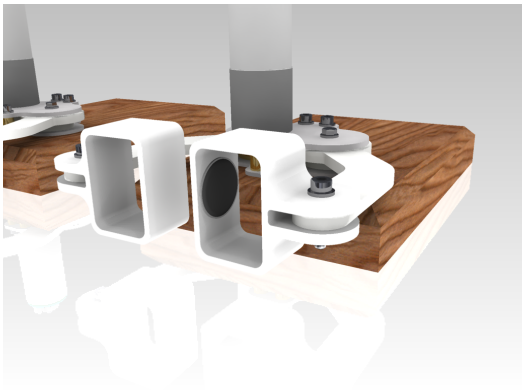
(c) Vy sida



(d) Detalj kuggar, ovan



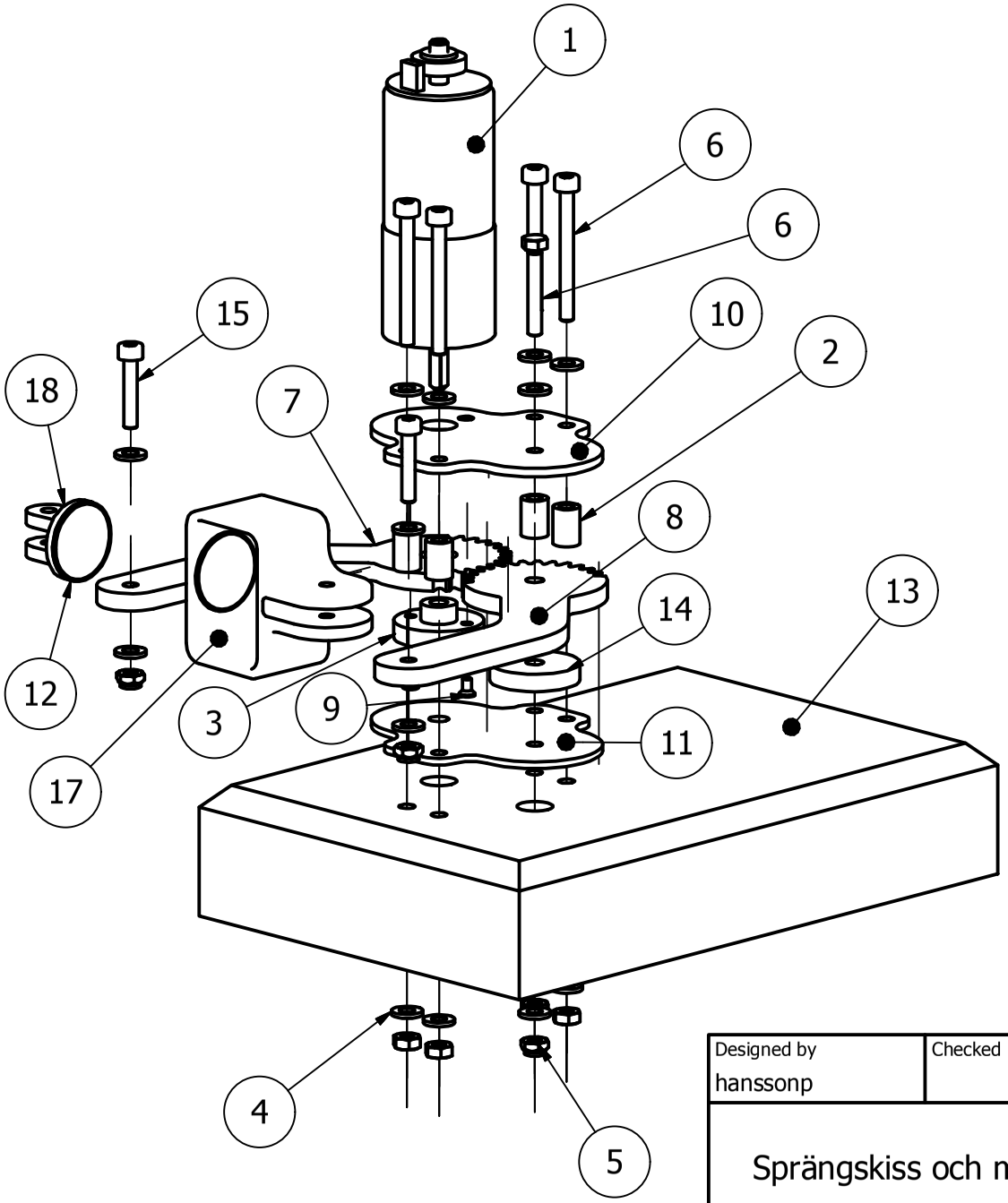
(e) Gripklo



(f) Display

Figur 44: Autodesk Inventor-CAD-renderingar

A.2 Ritningar



PARTS LIST			
ITEM	QTY	PART NUMBER	COMMENTS
1	1	DC-motor	
2	4	Mässingsdistans	
3	1	Mässingsmotorfäste	
4	14	M3-brickor	
5	7	M3-nylockmutter	
6	4	M3x0,5	1xSpeciallängd
7	1	Gripklo A	
8	1	Gripklo B	
9	2	M2x0,4 x 4	
10	1	Övre fästplatta	
11	1	Nedre fästplatta	
12	2	Kraftsensor	
13	1	Bottenplatta	
14	1	Teflondistans	
15	2	M3x0,5 x 16	
16	1	M3x0,5 x 20	
17	2	Fingerhållare	För display
18	2	Fingertopp	För gripdon

Designed by	Checked by	Approved by	Date		Date	
hanssonp					2012-05-09	

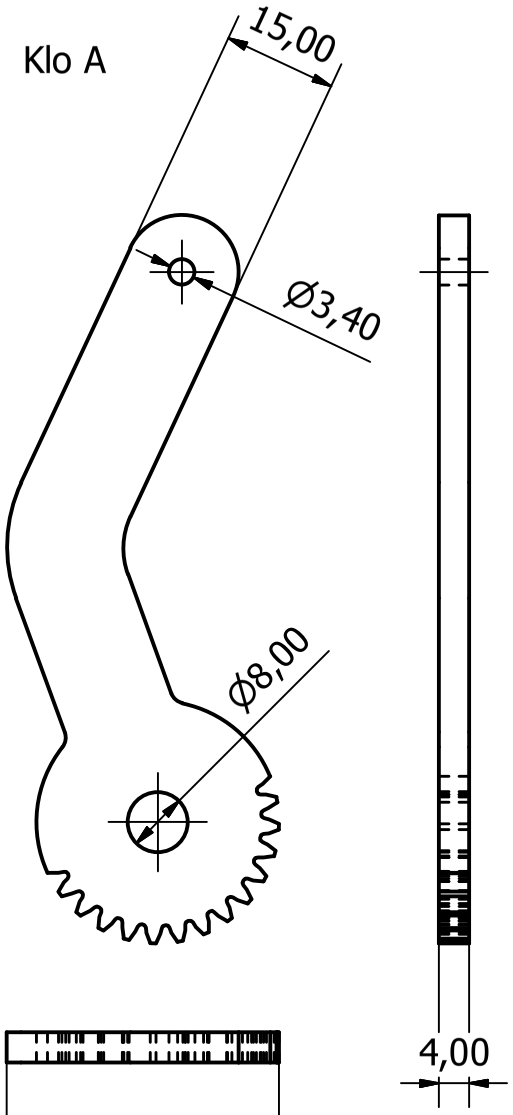
Sprängskiss och materiallista

AssemblySprangskiss

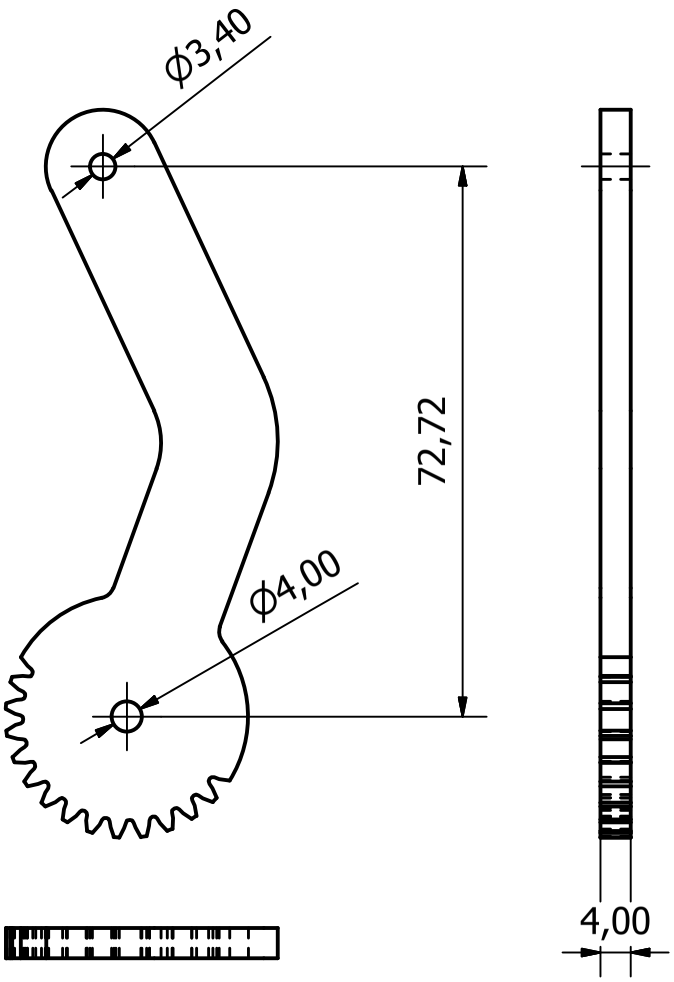
Edition

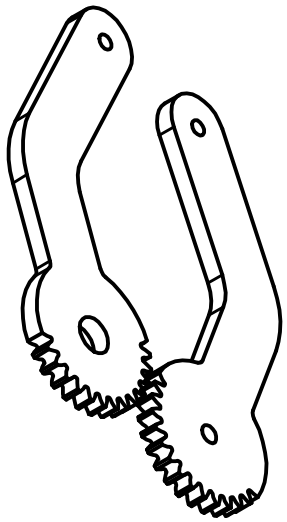
Sheet 1 / 1

Klo A



Klo B

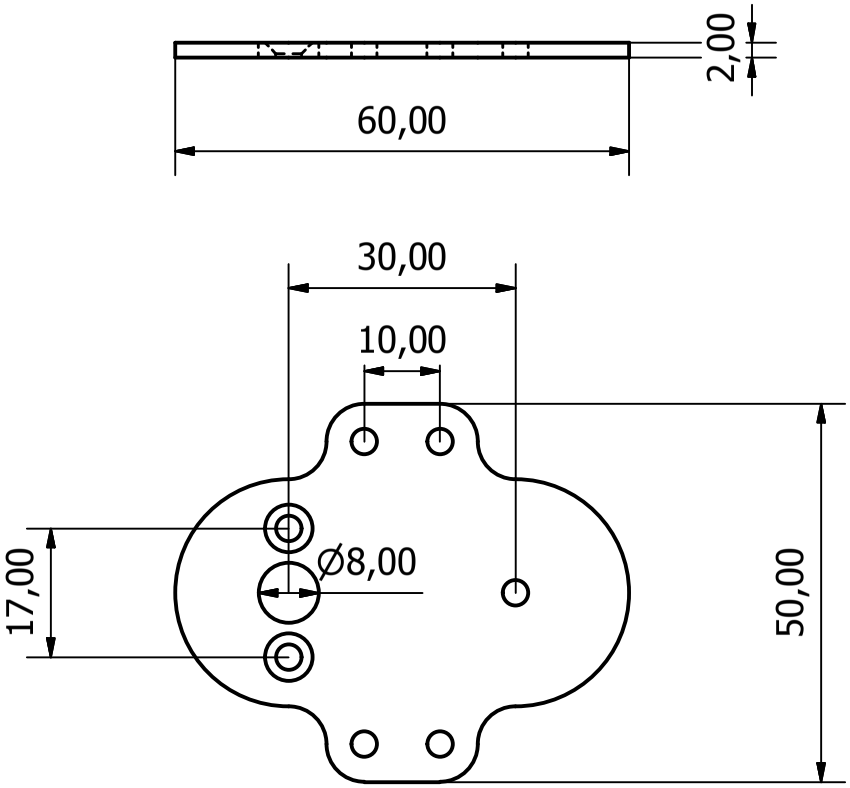




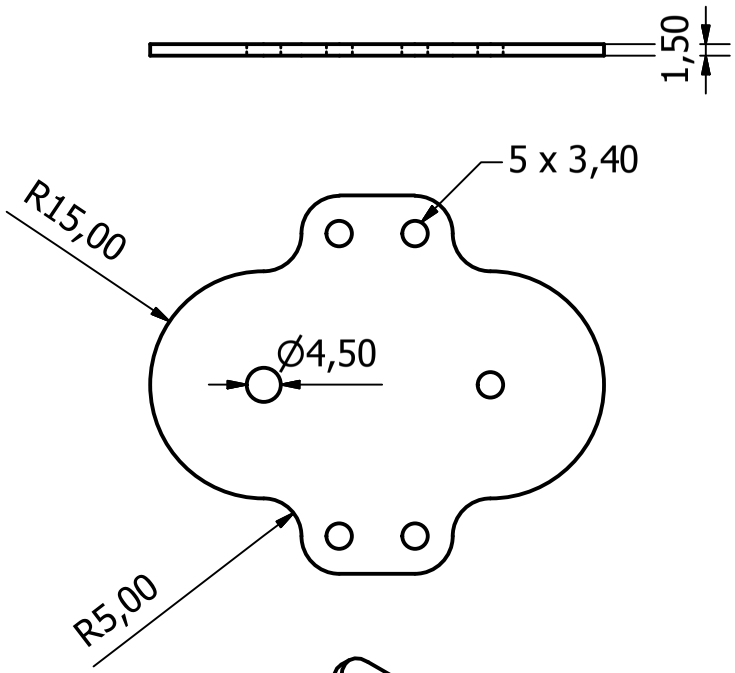
Designed by	Checked by	Approved by	Date		Date	
hanssonp					2012-03-03	1:1

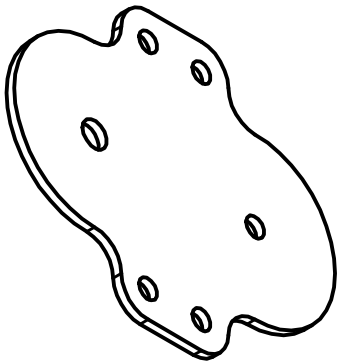
"Fingrar"			
New_klo	Edition	Sheet 1 / 1	

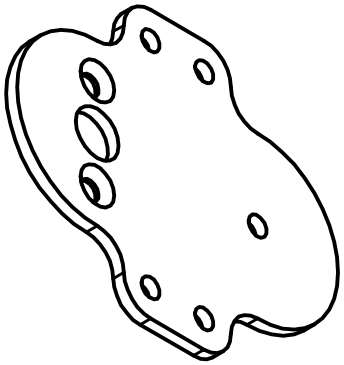
Disp_part1



Disp_part2



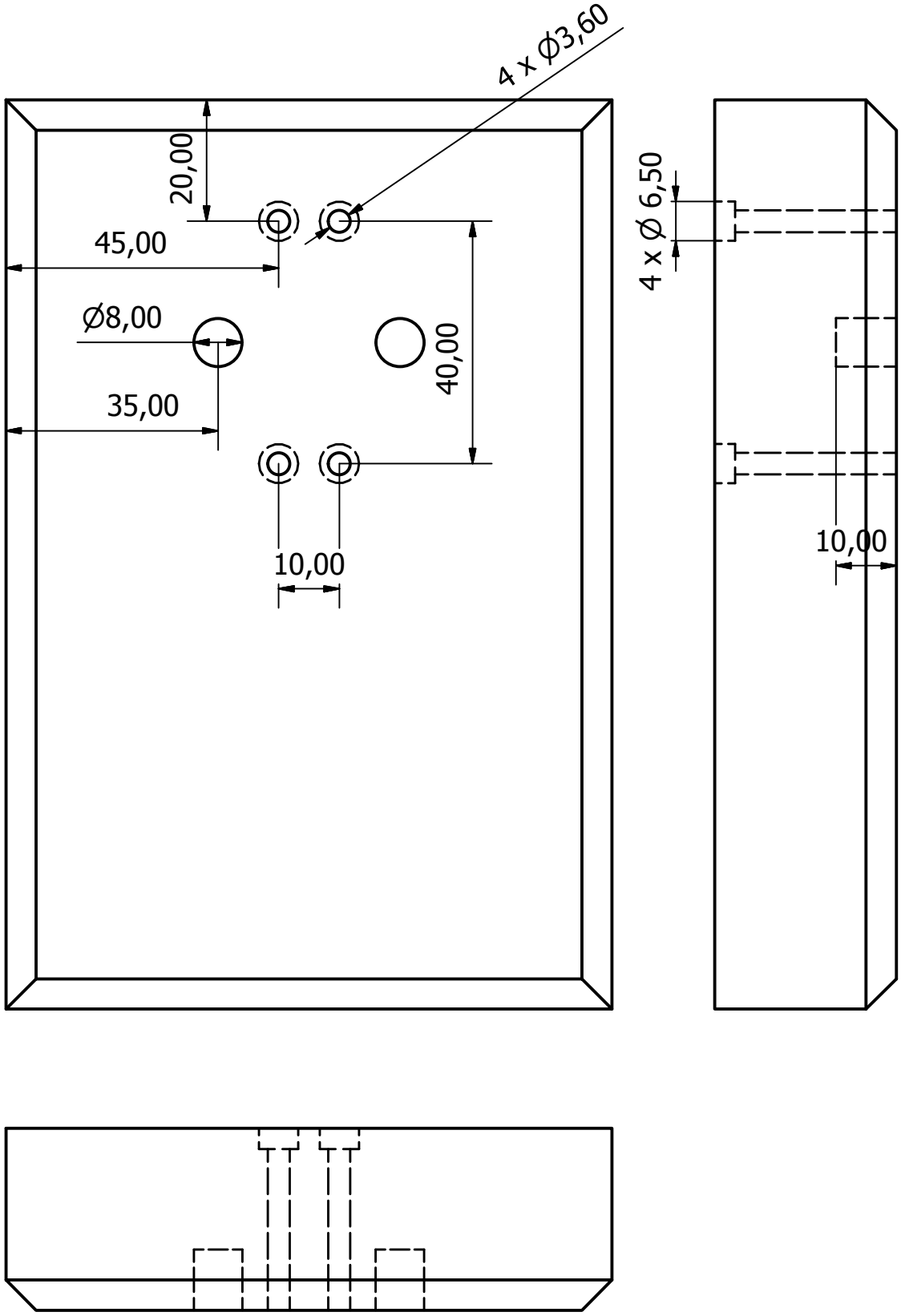




Designed by	Checked by	Approved by	Date		Date	
hanssonp					2012-03-03	

Fästplattor

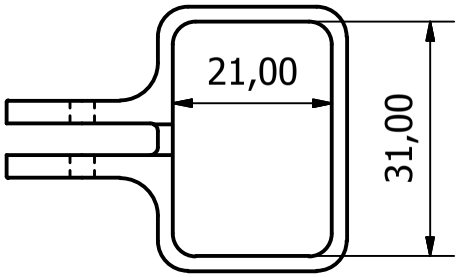
Disp_Part	Edition	Sheet
		1 / 1

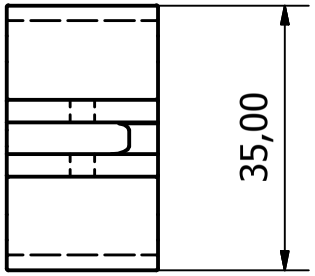


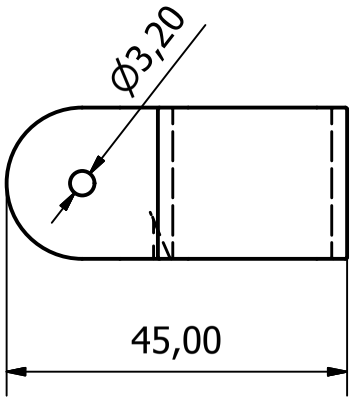
Designed by	Checked by	Approved by	Date		Date	1:1
hanssonp					2012-04-12	

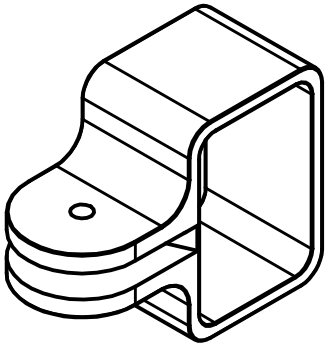
Bottenplatta	Bottenplatta	Edition	Sheet
			1 / 1

Fingergrepp

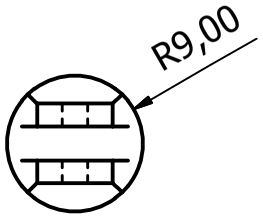


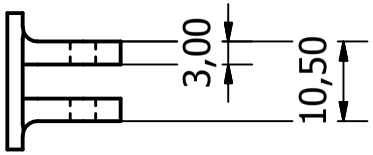


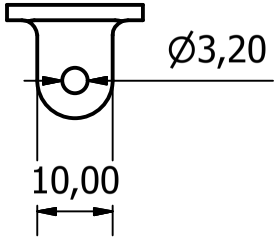


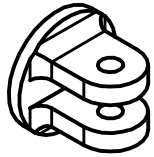


Sensorplatta





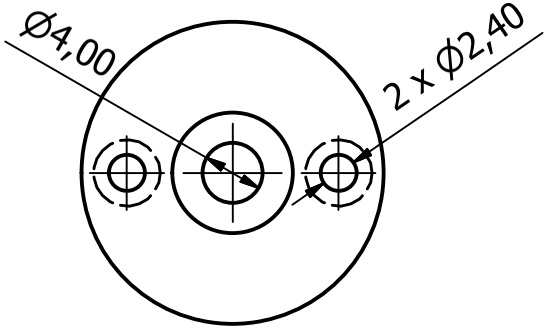


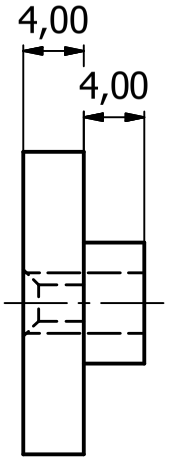


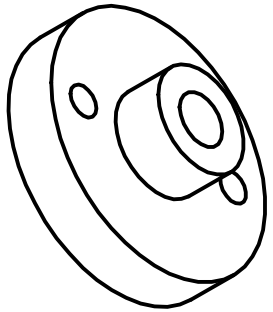
Designed by	Checked by	Approved by	Date		Date	1:1
hanssonp					2012-03-03	

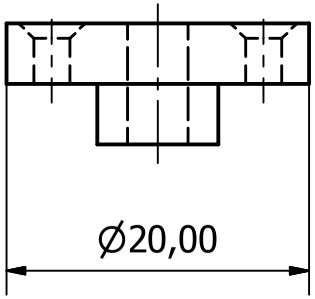
"Fingertoppar"

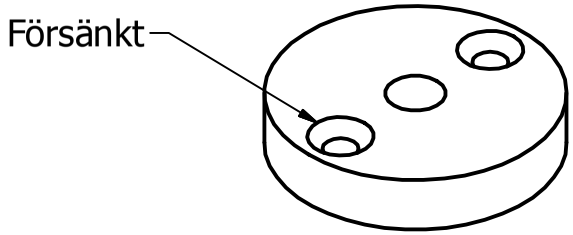
Fingertoppar	Edition	Sheet
		1 / 1





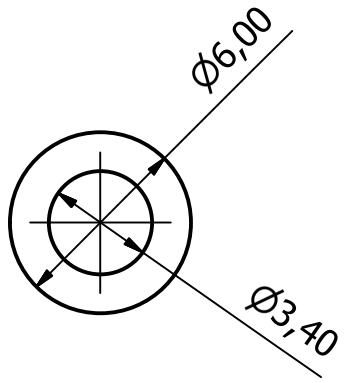


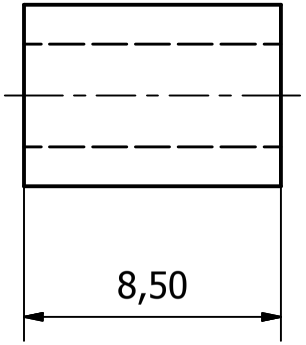


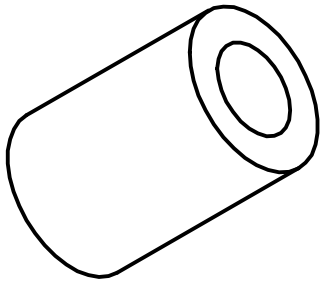


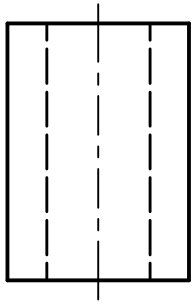
Designed by	Checked by	Approved by	Date		Date	
hanssonp					2012-03-13	2:1

Motoraxelfäste			
	Disp_motorfaste	Edition	Sheet 1 / 1









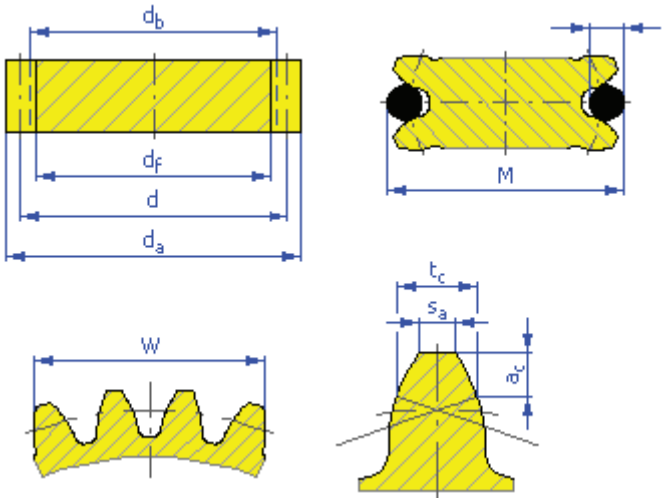
Designed by	Checked by	Approved by	Date		Date	4:1
hanssonp					2012-03-13	

Distans			
	Disp_Distans	Edition	Sheet 1 / 1

Bilaga B Kuggdata

Common Parameters

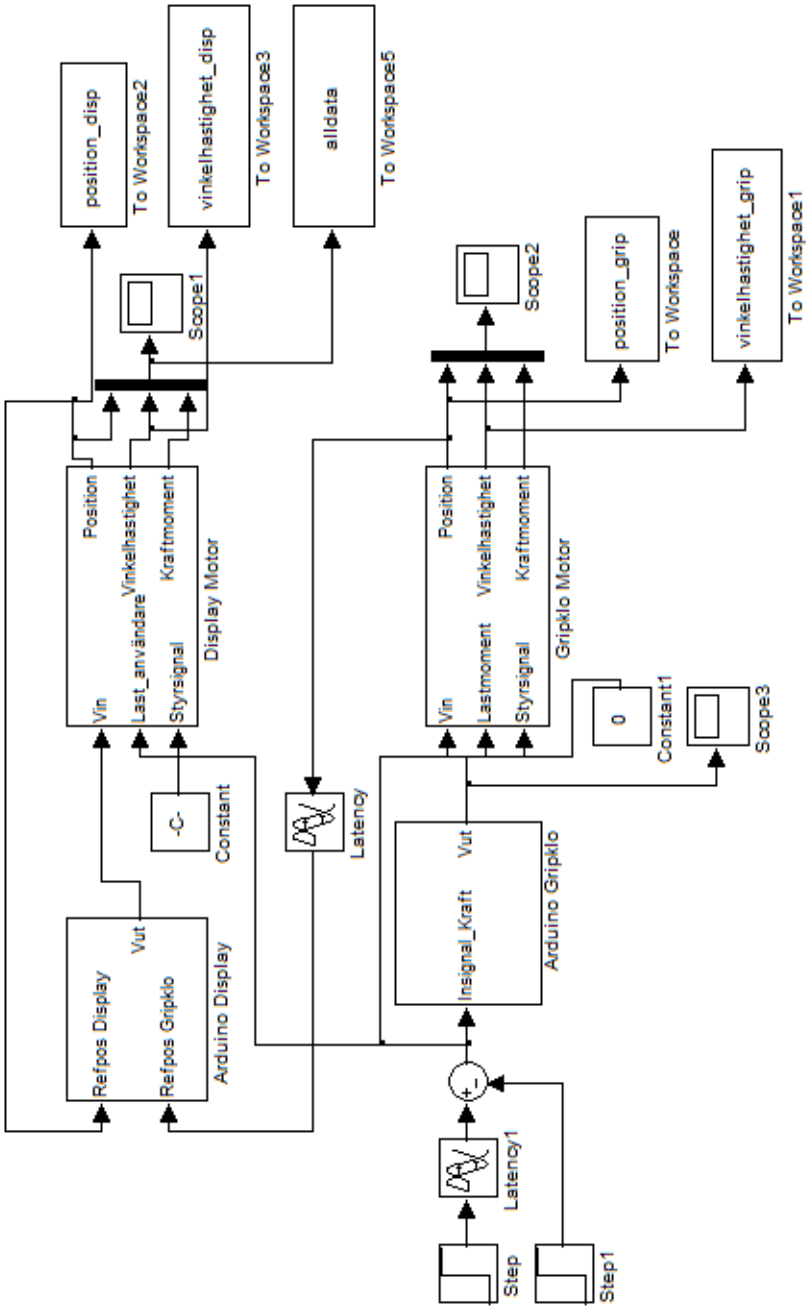
Gear Ratio	i	1,0000 ul
Desired Gear Ratio	i_{in}	1,0000 ul
Diametral Pitch	P	24,0000 ul/in
Helix Angle	β	0,0000 deg
Pressure Angle	α	20,0000 deg
Center Distance	a_w	30,000 mm
Product Center Distance	a	29,633 mm
Total Unit Correction	Σx	0,3620 ul
Circular Pitch	p	3,325 mm
Base Circular Pitch	p_{tb}	3,124 mm
Operating Pressure Angle	α_w	21,8429 deg
Contact Ratio	ε	1,5386 ul
Limit Deviation of Axis Parallelity	f_x	0,0090 mm
Limit Deviation of Axis Parallelity	f_y	0,0045 mm



Gear data

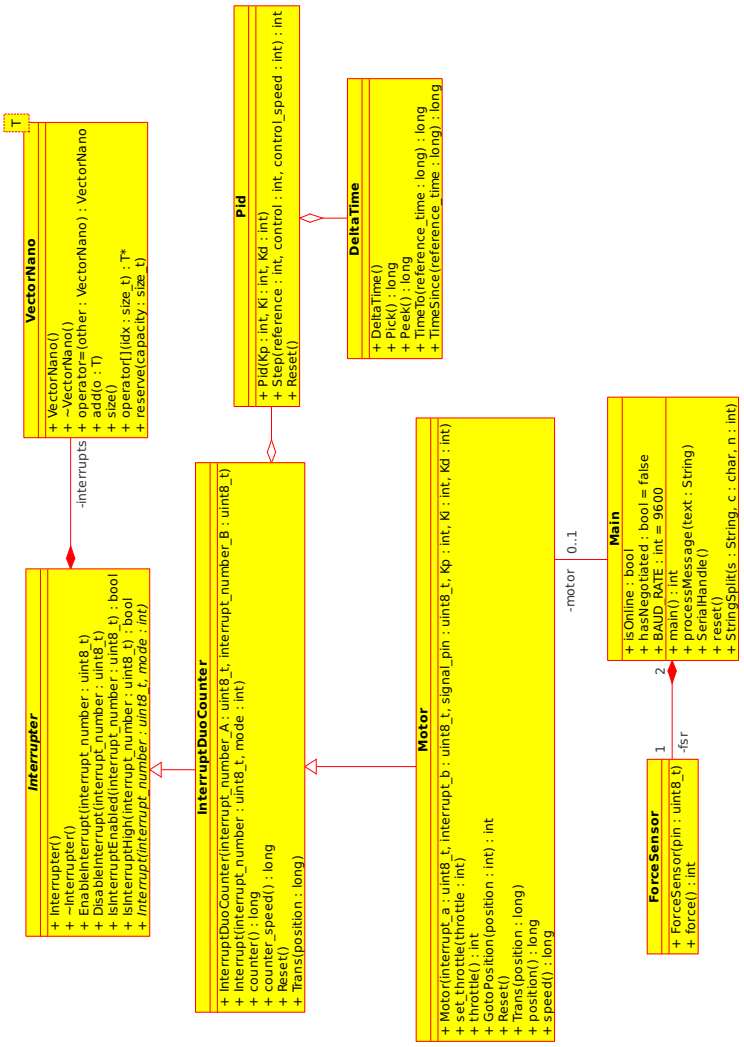
		Gear 1	Gear 2
Number of Teeth	z	28 ul	28 ul
Unit Correction	x	0,1810 ul	0,1810 ul
Pitch Diameter	d	29,633 mm	29,633 mm
Outside Diameter	d_a	32,100 mm	32,100 mm
Root Diameter	d_f	27,371 mm	27,371 mm
Base Circle Diameter	d_b	27,846 mm	27,846 mm
Work Pitch Diameter	d_w	30,000 mm	30,000 mm
Facewidth	b	4,000 mm	4,000 mm
Facewidth Ratio	b_r	0,1350 ul	0,1350 ul
Addendum	a*	1,0000 ul	1,0000 ul
Clearance	c*	0,2500 ul	0,2500 ul
Root Fillet	r_f*	0,3500 ul	0,3500 ul
Tooth Thickness	s	1,802 mm	1,802 mm
Tangential Tooth Thickness	s_t	1,802 mm	1,802 mm
Chordal Thickness	t_c	1,591 mm	1,591 mm
Chordal Addendum	a_c	0,944 mm	0,944 mm
Chordal Dimension	W	11,481 mm	11,481 mm
Chordal Dimension Teeth	z_w	4,000 ul	4,000 ul
Dimension Over (Between) Wires	M	33,069 mm	33,069 mm
Wire Diameter	d_M	2,000 mm	2,000 mm
Limit Deviation of Helix Angle	F_β	0,0090 mm	0,0090 mm
Limit Circumferential Run-out	F_r	0,0160 mm	0,0160 mm
Limit Deviation of Axial Pitch	f_{pt}	0,0070 mm	0,0070 mm
Limit Deviation of Basic Pitch	f_{pb}	0,0066 mm	0,0066 mm
Virtual Number of Teeth	z_v	28,000 ul	28,000 ul
Virtual Pitch Diameter	d_n	29,633 mm	29,633 mm
Virtual Outside Diameter	d_{an}	32,100 mm	32,100 mm
Virtual Base Circle Diameter	d_{bn}	27,846 mm	27,846 mm
Unit Correction without Tapering	x_z	0,2086 ul	0,2086 ul
Unit Correction without Undercut	x_p	-0,6180 ul	-0,6180 ul
Unit Correction Allowed Undercut	x_d	-0,7879 ul	-0,7879 ul
Addendum Truncation	k	0,0155 ul	0,0155 ul
Unit Outside Tooth Thickness	s_a	0,6952 ul	0,6952 ul
Tip Pressure Angle	α_a	29,8332 deg	29,8332 deg

Bilaga C Simulinkmodell



Figur 45: Simulinkmodell av det synkrona systemet

Bilaga D Klassdiagram



Figur 46: Klassdiagram för Arduino-enhet

Bilaga E Programkod

delta-time.hpp

```
1  #include <Arduino.h>
2
3  #ifndef DELTA_TIME_
4  #define DELTA_TIME_
5
6  class DeltaTime
7  {
8  public:
9          static const unsigned long NOT_MEASURED = 0l;
10
11         DeltaTime() : last_measure_(NOT_MEASURED) { }
12
13         signed long Pick()
14         {
15                 unsigned long current_time = micros();
16                 unsigned long delta_time = TimeTo(
                       current_time);
17                 last_measure_ = current_time;
18
19                 return delta_time;
20         }
21
22         signed long Peek() const
23         {
24                 return TimeTo(micros());
25         }
26
27         //WARNING DOES NOT HANDLE SIGN ERRORS
28         signed long TimeTo(unsigned long reference_time
               ) const
29         {
30                 //TODO: Implement integer overflow
                       control
31                 if (last_measure_ != NOT_MEASURED)
32                         return reference_time -
                               last_measure_;
33                 else
```

```
34                          return 0l;
35          }
36
37          //WARNING DOES NOT HANDLE SIGN ERRORS
38          signed long TimeSince(unsigned long
               reference_time) const
39          {
40                  //TODO: Implement integer overflow
                       control
41                  if (last_measure_ != NOT_MEASURED)
42                          return last_measure_ -
                               reference_time;
43                  else
44                          return 0l;
45          }
46
47  private:
48          volatile unsigned long last_measure_;
49  };
50
51  #endif
```

force-sensor.hpp

```
1  #include <Arduino.h>
2
3  #ifndef FORCE_SENSOR_
4  #define FORCE_SENSOR_
5
6  class ForceSensor
7  {
8  public:
9          ForceSensor(uint8_t pin) : pin_(pin) { }
10
11         //Force from 0 - 10000
12         int force()
13         {
14                 int raw = constrain(map(analogRead(pin_
                       ),0,1024,0,1000),0,1000);
15
16                 // en uppskattad invers
17                 // y^(-1)(x)=(x^2/1024)
18                 raw =(int)( (((float)raw)/1024.f) * (((
                       float)raw)));
19                 return raw;
20         }
21 private:
22         uint8_t pin_;
23 };
24
25 #endif
```

interrupt-duo-counter.hpp

```
1  #include <Arduino.h>
2  #include "interrupter.hpp"
3  #include "delta-time.hpp"
4
5  #ifndef INTERRUPT_DUO_COUNTER_
6  #define INTERRUPT_DUO_COUNTER_
7
8  class InterruptDuoCounter : public Interrupter
9  {
10 public:
11         InterruptDuoCounter(uint8_t interrupt_number_A,
               uint8_t interrupt_number_B)
12                 :       interrupt_A_(interrupt_number_A
                       ),
13                         interrupt_B_(interrupt_number_B
                           ),
14                         counter_(0l),
15                         counter_speed_(0l) {
16
17                 EnableInterrupt(interrupt_number_A);
18                 EnableInterrupt(interrupt_number_B);
19         }
20
21         void Interrupt(uint8_t interrupt_number, int
              mode)
22         {
23                 unsigned long counter_old = counter_;
24                 //Inte sa fager med den gor det den ska
                       med minnimal overhead
25                 //
26                 // forskjutning av interrupten vid
                      upprakning
27                 //-----       -----       -----
                      -----
28                 //     |      |      |      |      |      |
                          |
29                 //     |      |      |      |      |      |
                          |
30                 //     |-----|      |-----|      |-----|
```

```
                |----
31              // interrupt_A_
32              //
33              //   -----        -----        -----
                     -----
34              //  |     |      |     |      |     |
                    |     |
35              //  |     |      |     |      |     |
                    |     |
36              //--|     |-----|     |-----|     |
                  -----|     |-
37              // interrupt_B_
38
39              if (interrupt_number == interrupt_A_)
40              {
41                      if (IsInterruptHigh(
                           interrupt_A_))
42                              IsInterruptHigh(
                                   interrupt_B_) ? --
                                   counter_ : ++
                                   counter_;
43                      else
44                              IsInterruptHigh(
                                   interrupt_B_) ? ++
                                   counter_ : --
                                   counter_;
45              }
46              else if (interrupt_number ==
                   interrupt_B_)
47              {
48                      if (IsInterruptHigh(
                           interrupt_B_))
49                              IsInterruptHigh(
                                   interrupt_A_) ? ++
                                   counter_ : --
                                   counter_;
50                      else
51                              IsInterruptHigh(
                                   interrupt_A_) ? --
                                   counter_ : ++
                                   counter_;
```

```
52                  }
53                  int sign = counter_ - counter_old;
54
55                  counter_speed_ = (MICROS_PER_SECOND/
                        delta_time_.Pick()) * sign;
56          }
57
58          signed long counter() const { return counter_;
               }
59
60          //Counts per second
61          signed long counter_speed() const
62          {
63                  if (delta_time_.Peek() < DEAD_TIME)
64                          return counter_speed_;
65                  else
66                          return 0;
67          }
68
69          void Reset()
70          {
71                  counter_ = 0l;
72          }
73
74          void Trans(signed long pos)
75          {
76                  counter_ = pos;
77          }
78
79  private:
80          static const unsigned long MICROS_PER_SECOND =
               1000000l;
81          static const signed long DEAD_TIME = 3000l;
82          const uint8_t interrupt_A_, interrupt_B_;
83          volatile signed long counter_, counter_speed_;
84          DeltaTime delta_time_;
85  };
86
87  #endif
```

interrupter.hpp

```
1  //May use ATmega2560/ArduinoMega specific code
2
3  #include <Arduino.h>
4  #include <wiring_private.h>
5  #include "vector-nano.hpp"
6  #include <avr/interrupt.h>
7
8  #ifndef INTERRUPTER_
9  #define INTERRUPTER_
10
11
12 uint8_t Interrupt2Pin(uint8_t interrupt);
13 uint8_t Pin2Interrupt(uint8_t pin);
14 void EnableInterruptInCpu(uint8_t interrupt_number, int
       mode);
15
16 class Interrupter
17 {
18 public:
19         Interrupter();
20         ~Interrupter();
21         virtual void Interrupt(uint8_t interrupt_number
              , int mode)=0;
22         void EnableInterrupt(uint8_t interrupt_number);
23         void DisableInterrupt(uint8_t interrupt_number)
              ;
24         bool IsInterruptEnabled(uint8_t
              interrupt_number);
25         static bool IsInterruptHigh(uint8_t
              interrupt_number);
26         static void CallInterrupters(uint8_t
              interrupt_number);
27
28 private:
29         static bool is_high_[EXTERNAL_NUM_INTERRUPTS];
30         size_t number_;
31         bool is_activated_[EXTERNAL_NUM_INTERRUPTS];
32 };
33
```

```
34 static VectorNano<Interrupter> interrupters;
35
36 #endif
```

interrupter.cpp

```
1  #include "interrupter.hpp"
2
3  //Optimized for speed
4  uint8_t Interrupt2Pin(uint8_t interrupt)
5  {
6          switch (interrupt) {
7                  case 0:
8                          return 2;
9                          break;
10                 case 1:
11                         return 3;
12                         break;
13                 case 2:
14                         return 21;
15                         break;
16                 case 3:
17                         return 20;
18                         break;
19                 case 4:
20                         return 19;
21                         break;
22                 case 5:
23                         return 18;
24                         break;
25                 default:
26                         break;
27                 }
28         return 0; //EEEH FEL! ASSERT!!
29 }
30
31 //Optimized for speed
32 uint8_t Pin2Interrupt(uint8_t pin)
33 {
34         switch (pin)
35         {
36                 case 2:
37                         return 0;
38                         break;
39                 case 3:
```

```
40                          return 1;
41                          break;
42                  case 21:
43                          return 2;
44                          break;
45                  case 20:
46                          return 3;
47                          break;
48                  case 19:
49                          return 4;
50                          break;
51                  case 18:
52                          return 5;
53                          break;
54                  default:
55                          break;
56                  }
57          return 0; // EEEH FEL! ASSERT!!
58  }
59
60  void EnableInterruptInCpu(uint8_t interrupt_number, int
        mode)
61  {
62          switch (interrupt_number)
63          {
64                  case 2:
65                          EICRA = (EICRA & ~((1 << ISC00)
                                | (1 << ISC01))) | (mode <<
                                ISC00);
66                          EIMSK |= (1 << INT0);
67                          break;
68                  case 3:
69                          EICRA = (EICRA & ~((1 << ISC10)
                                | (1 << ISC11))) | (mode <<
                                ISC10);
70                          EIMSK |= (1 << INT1);
71                          break;
72                  case 4:
73                          EICRA = (EICRA & ~((1 << ISC20)
                                | (1 << ISC21))) | (mode <<
                                ISC20);
```

```
74                              EIMSK |= (1 << INT2);
75                              break;
76                      case 5:
77                              EICRA = (EICRA & ~((1 << ISC30)
                                    | (1 << ISC31))) | (mode <<
                                    ISC30);
78                              EIMSK |= (1 << INT3);
79                              break;
80                      case 0:
81                              EICRB = (EICRB & ~((1 << ISC40)
                                    | (1 << ISC41))) | (mode <<
                                    ISC40);
82                              EIMSK |= (1 << INT4);
83                              break;
84                      case 1:
85                              EICRB = (EICRB & ~((1 << ISC50)
                                    | (1 << ISC51))) | (mode <<
                                    ISC50);
86                              EIMSK |= (1 << INT5);
87                              break;
88                      default:
89                              break;
90              }
91      }
92
93      Interrupter::Interrupter()
94      {
95              number_ = interrupters.size();
96              interrupters.add(this);
97              for (size_t i=0; i<EXTERNAL_NUM_INTERRUPTS; ++i
                    )
98                      is_activated_[i] = false;
99      }
100
101     Interrupter::~Interrupter()
102     {
103             interrupters[number_] = NULL;
104     }
105
106     void Interrupter::EnableInterrupt(uint8_t
            interrupt_number)
```

```
107 {
108         if (interrupt_number < EXTERNAL_NUM_INTERRUPTS)
109         {
110                 is_activated_[interrupt_number] = true;
111                 is_high_[interrupt_number] =
                        digitalRead(Interrupt2Pin(
                        interrupt_number));
112                 EnableInterruptInCpu(interrupt_number,
                        CHANGE);
113         }
114 }
115
116 void Interrupter::DisableInterrupt(uint8_t
        interrupt_number)
117 {
118         if (interrupt_number < EXTERNAL_NUM_INTERRUPTS)
119                 is_activated_[interrupt_number] = false
                        ;
120 }
121
122 bool Interrupter::IsInterruptEnabled(uint8_t
        interrupt_number)
123 {
124         if (interrupt_number < EXTERNAL_NUM_INTERRUPTS)
125                 return is_activated_[interrupt_number];
126         else
127                 return false;
128 }
129
130 bool Interrupter::IsInterruptHigh(uint8_t
        interrupt_number)
131 {
132         if (interrupt_number < EXTERNAL_NUM_INTERRUPTS)
133                 return is_high_[interrupt_number];
134         else
135                 return false;
136 }
137
138 void Interrupter::CallInterrupters(uint8_t
        interrupt_number)
139 {
```

```
140         is_high_[interrupt_number] = !is_high_[
                interrupt_number];
141 //      int mode = digitalRead(Interrupt2Pin(
        interrupt_number) ? RISING : FALLING);
142         int mode = is_high_[interrupt_number] ? RISING
                : FALLING;
143
144         for (size_t i=0; i<interrupters.size(); ++i)
145             if (interrupters[i] != NULL &&
                    interrupters[i]->IsInterruptEnabled(
                    interrupt_number))
146                 interrupters[i]->Interrupt(
                        interrupt_number, mode);
147 }
148
149 bool Interrupter::is_high_[] = {0};
150
151 ISR(INT4_vect) { Interrupter::CallInterrupters(0); }
152 ISR(INT5_vect) { Interrupter::CallInterrupters(1); }
153 ISR(INT0_vect) { Interrupter::CallInterrupters(2); }
154 ISR(INT1_vect) { Interrupter::CallInterrupters(3); }
155 ISR(INT2_vect) { Interrupter::CallInterrupters(4); }
156 ISR(INT3_vect) { Interrupter::CallInterrupters(5); }
157 ISR(INT6_vect) { Interrupter::CallInterrupters(6); }
158 ISR(INT7_vect) { Interrupter::CallInterrupters(7); }
```

main.cpp

```
1  #include <Arduino.h>
2  #include <WString.h>
3  #include <HardwareSerial.h>
4  #include <avr/io.h>
5  #include "interrupter.hpp"
6  #include "vector-nano.hpp"
7  #include "interrupt-duo-counter.hpp"
8  #include "motor.hpp"
9  #include "force-sensor.hpp"
10 #include "delta-time.hpp"
11
12 void processMessage(String text);
13 void SerialHandle(void);
14 void start(void);
15 void stop(void);
16 void reset(void);
17 int main(void);
18 String StringSplit(String s, char c, int n);
19
20 String inputString = "";
21
22 volatile bool isOnline = false;
23 volatile bool hasNegotiated = false;
24
25 static const int BAUD_RATE = 9600;
26 static const int LED = 13;
27
28 static const int BUFFER_SIZE = 13;
29
30 static const char ENDLINE = '\n';
31 static const char EMPTY[] = "";
32
33 static const char NEGOTIATION[] = "GPK0001";
34 static const char RESET[] = "reset";
35 static const char DATA[] = "data";
36
37 static const unsigned long NOT_INIT = 0;
38
39 #if defined DISPLAY
```

```
40 static const char NEGOTIATION_RESPONSE[] = "DISPLAY0001
     ";
41 #elif defined GRIPKLO
42 static const char NEGOTIATION_RESPONSE[] = "GRIPKLO0001
     ";
43 #endif
44
45 int incomingByte = 0;
46
47 Motor* motor;
48
49 ForceSensor fsr1(A0);
50 ForceSensor fsr2(A2);
51
52 int force = 0;
53 int force_other = 0;
54 int position_other = 0;
55
56 int main(void)
57 {
58         init();
59
60 #if defined(USBCON)
61         USB.attach();
62 #endif
63
64         Serial.begin(BAUD_RATE);
65
66         motor = new Motor(0, 1, 9, 7, 0, 0);
67         motor->set_throttle(0);
68
69         pinMode(13, OUTPUT);
70
71         for(;;)
72         {
73 #if defined DISPLAY
74                 motor->GotoPosition(position_other);
75 #elif defined GRIPKLO
76                 motor->set_throttle(force_other*0.7);
77 #endif
78                 SerialHandle();
```

```
79          }
80          return 0;
81  }
82
83  DeltaTime dt;
84
85  void processMessage(String text)
86  {
87
88          if (text == NEGOTIATION)
89          {
90                  hasNegotiated = true;
91                  Serial.println(NEGOTIATION_RESPONSE);
92          }
93          else if (hasNegotiated)
94          {
95                  if (text.startsWith("data:"))
96                  {
97                          position_other = StringSplit(
                               text, ':', 1).toInt();
98                          force_other = StringSplit(text,
                                ':', 2).toInt();
99          //              String timen = StringSplit(text
                            , ':', 1);
100 //                      long int positionen =
        StringSplit(text, ':',2).toInt();
101         //              int forcen = StringSplit(text,
                            ':', 3).toInt();
102         //              Serial.println(timen);
103 //                      Serial.println(positionen);
104 //
105         //              Serial.println(forcen);
106                 }
107                 else if (text == DATA)
108                 {
109 //                      Serial.println(dt.Pick());
110                         Serial.println(motor->position
                            ());
111                         force = fsr1.force()-fsr2.force
                            ();
112                         Serial.println(force);
```

```
113                         Serial.println(millis());
114
115                 }
116                 else if (text == RESET)
117                 {
118                         reset();
119                         Serial.println("reseted");
120                 }
121                 else
122                 {
123                         Serial.print("unknown:");
124                         Serial.println(text);
125                 }
126         }
127 }
128
129 void SerialHandle()
130 {
131         while (Serial.available())
132         {
133                 char in_char = (char)Serial.read();
134                 if (in_char == ENDLINE)
135                 {
136                         //Serial.println(steg);
137                         processMessage(inputString);
138                         inputString = EMPTY;
139                 }
140                 else
141                 {
142                         inputString += in_char;
143                 }
144         }
145 }
146
147 String StringSplit(String s, char c, int n)
148 {
149         int i=0;
150         while (s.indexOf(c) != -1)
151         {
152                 if (i++ == n)
153                         return s.substring(0, s.indexOf
```

```
                                        (c));
154                     s = s.substring(s.indexOf(c)+1);
155             }
156             return s;
157     }
158
159     void reset()
160     {
161             motor->set_throttle(-500);
162
163             int pos;
164             do {
165                     pos = motor->position();
166                     delay(300);
167             } while (pos != motor->position());
168
169             motor->set_throttle(0);
170             delay(300);
171             motor->Reset();
172     //      motor->Trans(-500);
173     }
```

motor.hpp

```
1  #include <Arduino.h>
2  #include <Servo.h>
3  #include "interrupt-duo-counter.hpp"
4  #include "pid.hpp"
5
6  #ifndef MOTOR_
7  #define MOTOR_
8
9  class Motor
10 {
11 public:
12         Motor(uint8_t interrupt_a, uint8_t interrupt_b,
               uint8_t signal_pin, int Kp, int Ki, int Kd)
13             : idc_(interrupt_a, interrupt_b),
                 throttle_(0), pid_(Kp, Ki, Kd)
14         {
15             servo_.attach(signal_pin);
16         }
17
18         ~Motor()
19         {
20             servo_.detach();
21         }
22
23         // -1000          = full reverse
24         // 0              = stall
25         // 1000           = full forward
26         void set_throttle(int throttle)
27         {
28             if (throttle < NEW_DEAD_ZONE &&
                   throttle > -NEW_DEAD_ZONE)
29                 throttle = 0;
30
31             if (throttle > 0 && throttle <
                   DEAD_ZONE)
32                 throttle += DEAD_ZONE;
33             else if (throttle < 0 && throttle > -
                   DEAD_ZONE)
34                 throttle -= DEAD_ZONE;
```

```
35
36                  throttle_ = constrain(throttle/2, -500,
                        500);
37
38                  servo_.writeMicroseconds(1500+throttle_
                        );
39          }
40          inline int throttle() const { return throttle_;
                }
41
42          int GotoPosition(int position)
43          {
44                  int step = pid_.Step(position, this->
                        position(), this->speed());
45                  set_throttle(step);
46                  return step;
47          }
48
49          void Reset()
50          {
51                  idc_.Reset();
52          }
53
54          void Trans(signed long pos)
55          {
56                  idc_.Trans(pos);
57          }
58
59          signed long position() const { return idc_.
                counter(); }
60          signed long speed() const
61          {
62                  return idc_.counter_speed();
63          }
64  private:
65          //Special data for our dc-motor
66          static const int DEAD_ZONE = 110;
67          static const int NEW_DEAD_ZONE = 3;
68
69          InterruptDuoCounter idc_;
70          int throttle_;
```

```
71          Pid pid_;
72          Servo servo_;
73  };
74
75  #endif
```

pid.hpp

```
1  #include <Arduino.h>
2  #include "delta-time.hpp"
3
4  #ifndef PID_
5  #define PID_
6
7  class Pid
8  {
9  public:
10         Pid(int Kp, int Ki, int Kd) : Kp_(Kp), Ki_(Ki),
               Kd_(Kd), ek_(0), ik_(0), dk_(0) { }
11
12         int Step(int reference, int control, int
               control_speed)
13         {
14 /*              signed long sample_l = sampler_timer_.
       Pick();
15                 double sample = (double)sample_l;
16                 sample /= 1000000;
17                 double fel = (double)(reference -
                       control);
18                 double signal = (0.18*(1.0+20.0*sample)
                       *fel-0.18*last_error_+last_signal_)
                       /(1.0+204.0*sample);
19                 last_error_ = fel;
20                 last_signal_ = signal;
21                 return (int)signal;*/
22
23                 return (reference - control)*Kp_;
24
25 /*              ek_ = reference - control;
26                 double e = ((double)ek_)/1000.0;
27                 double k = (double)Kp_;
28                 double d = (double)Kd_;
29                 d /= 1000.0;
30                 double s = (double)control_speed;
31
32                 return (int)( (k * e) - (d * s) );*/
33         }
```

```
34
35          void Reset()
36          {
37                  ik_ = 0;
38          }
39
40  private:
41          const int Kp_, Ki_, Kd_;
42          int ek_, ik_, dk_;
43
44          //microseconds since the start of the program
               we sampled last time. Value 0 when no sample
                has occured
45          double last_error_;
46          double last_signal_;
47
48          DeltaTime sampler_timer_;
49  };
50
51  #endif
```

vector-nano.hpp

```
1  // A simple Vector wich is forced to be pointer based
2  #include <stddef.h> //size_t
3
4  #ifndef VECTOR_NANO_
5  #define VECTOR_NANO_
6
7  template<typename T>
8  class VectorNano {
9  public:
10         VectorNano() : size_(0), capacity_(0), data_(
               NULL) {}
11
12         VectorNano(VectorNano const &other) : size_(
               other.size_), capacity_(other.capacity_),
               data_(0)
13         {
14                 data_ = (T *)malloc(capacity_ * sizeof(
                       T));
15                 memcpy(data_, other.data_, size_ *
                       sizeof(T));
16         }
17
18         ~VectorNano() { free(data_); }
19
20         // Needed for memory management
21         VectorNano &operator=(VectorNano const &other)
22         {
23                 free(data_);
24                 size_ = other.size_;
25                 capacity_ = other.capacity_;
26                 data_ = (T**) malloc(capacity_ * sizeof
                       (T*));
27                 memcpy(data_, other.data_, size_ *
                       sizeof(T*));
28                 return *this;
29         }
30
31         // Adds new value. If needed, allocates more
               space
```

```
32          void add(T* const &o)
33          {
34                  if (capacity_ == size_)
35                          resize(size_ + 1);
36                  data_[size_ - 1] = o;
37          }
38
39          size_t size() const { return size_; }
40          T* const &operator[](size_t idx) const { return
                data_[idx]; }
41          T* &operator[](size_t idx) { return data_[idx];
                }
42
43          void reserve(size_t capacity)
44          {
45                  if (capacity_ > capacity)
46                          resize(capacity);
47          }
48  private:
49          void resize(size_t size) {
50                  capacity_ = size;
51                  T** newdata = (T** )malloc(capacity_ *
                        sizeof(T*));
52                  memcpy(newdata, data_, size_ * sizeof(T
                        ));
53                  free(data_);
54                  data_ = newdata;
55                  size_ = size;
56          }
57
58          size_t size_;
59          size_t capacity_;
60          T** data_;
61  };
62  #endif
```

main.cpp

```
1  #include <iostream>
2  #include "utilities.hpp"
3  #include "serial.hpp"
4  #include <boost/exception/all.hpp>
5
6  #include <boost/asio.hpp>
7  #include <boost/array.hpp>
8
9  #include <boost/algorithm/string/predicate.hpp>
10
11 #include "tcp-sync.hpp"
12
13 using namespace std;
14 using boost::asio::ip::tcp;
15
16 int main(int argc, char* argv[])
17 {
18         string device = "/dev/ttyACM0";
19         int speed = 9600;
20
21         if (argc >= 2)
22                 device = argv[1];
23
24
25         TcpSync conn;
26
27
28         try
29         {
30                 Serial serial(device, speed);
31                 cout << "GripKom ver 0.0001" << endl;
32                 cout << "Press `enter` to continue..."
                           << endl;
33                 Wait(INPUT);
34                 string data;
35                 serial.Write("GPK0001");
36                 data = serial.Read();
37
38                 if (data == "DISPLAY0001")
```

```
39                  {
40                          cout << "Display ikopplad" <<
                               endl;
41                          serial.Write("reset");
42                          data = serial.Read();
43                          if (data == "reseted")
44                          {
45                                  cout << "Display has
                                       been reseted" <<
                                       endl;
46                          }
47                          else
48                          {
49                                  cout << "Communication
                                       error" << endl;
50                                  return 1;
51                          }
52                          cout << "Password for
                               connections: " << endl;
53                          string password;
54                          cin >> password;
55                          //password = "123";
56                          cout << "Waiting for connection
                               ..." << endl;
57
58                          do {
59                                  conn.Close();
60                                  conn.Accept(1445);
61                          }
62                          while (password != conn.Read())
                               ;
63                          conn.Write("Hi!");
64                  }
65                  else if (data == "GRIPKLO0001")
66                  {
67                          cout << "Gripklo ikopplad" <<
                               endl;
68                          serial.Write("reset");
69
70                          data = serial.Read();
71
```

```
72                          if (data == "reseted")
73                          {
74                                  cout << "Display has
                                       been reseted" <<
                                       endl;
75                          }
76                          else
77                          {
78                                  cout << "Communication
                                       error" << endl;
79                                  return 1;
80                          }
81
82                          cout << "Ange ipadress att
                                 ansluta till:" << endl;
83                          string ip;
84                          cin >> ip;
85  //                      ip = "localhost";
86                          if (ip == "")
87                                  ip = "localhost";
88
89                          conn.Connect(ip, "1445");
90                          cout << "Password: " << endl;
91                          cin >> data;
92  //                      data = "123";
93                          conn.Write(data);
94                          conn.Read();
95
96                          serial.Write("data");
97  //                      string timen = serial.Read();
98                          string positionen = serial.Read
                                 ();
99                          string forcen = serial.Read();
100                         string timen = serial.Read();
101 //                      conn.Write("data:" + timen +
    ":" + positionen + ":" + forcen);
102                         conn.Write("data:" + positionen
                                + ":" + forcen);
103
104                 }
105                 else
```

```
106                     {
107                             cout << "Error: Version
                                   mismatch" << endl;
108                             return 1;
109                     }
110
111                     cout << "Connection accomplished" <<
                           endl;
112                     conn.noDelay(true);
113
114
115                     while (true)
116                     {
117                             data = conn.Read();
118 //                          cout << "received: " << data <<
        endl;
119                             if (boost::starts_with(data, "
                                   data:"))
120                             {
121                                     serial.Write(data);
122
123                                     serial.Write("data");
124 //                                  string timen = serial.
        Read();
125 //                                  string timen = "0";
126                                     string positionen =
                                           serial.Read();
127                                     string forcen = serial.
                                           Read();
128                                     string tiden = serial.
                                           Read();
129
130                                     string send = "data:" +
                                            positionen + ":" +
                                           forcen;
131                                     conn.Write(send);
132
133                                     cout << tiden << ", "
                                           << positionen <<
                                           endl;
134                             //      cout << "send: " <<
```

```
                                send << endl;
135 //                                  conn.Write("data:" +
    timen + ":" + positionen + ":" + forcen);
136 //                                  string temp = "data:" +
     timen + ":" + positionen + ":" + forcen;
137                             }
138                             else
139                             {
140                                     cout << "error: " <<
                                           data << endl;
141                                     return 1;
142                             }
143                     }
144
145
146             } catch (boost::system::system_error& e) {
147
148                     cout << "Error: " << e.what() << endl;
149                     return 1;
150             }
151
152             return 0;
153 }
```

utilities.hpp

```
1  #ifndef UTILITIES_HPP
2  #define UTILITIES_HPP
3
4  #include <string>
5  #include <iostream>
6  #include <unistd.h>
7
8  const int INPUT = -1;
9
10 void Logga(std::string str);
11
12
13 void Wait(int t = INPUT);
14
15 #endif  /* UTILITIES_HPP */
```

utilities.cpp

```
1  #include "utilities.hpp"
2
3  void Logga(std::string str)
4  {
5      std::cout << str << std::endl;
6  }
7
8  void Wait(int t)
9  {
10     if (t == INPUT)
11         std::cin.get();
12     else
13         sleep(t);
14
15 }
```

tcp-sync.hpp

```
1  #ifndef TCP_SYNC_HPP
2  #define TCP_SYNC_HPP
3
4  #include <boost/asio.hpp>
5  #include <boost/array.hpp>
6  #include <string>
7
8  using namespace std;
9  using boost::asio::ip::tcp;
10
11 class TcpSync
12 {
13         public:
14                 TcpSync() : socket_(0)
15                 {
16                         socket_ = new tcp::socket(
                               io_service_);
17                 }
18
19                 ~TcpSync()
20                 {
21                         delete socket_;
22                 }
23                 void Connect(std::string hostname,
                       string port)
24                 {
25
26                         boost::system::error_code error
                               ;
27                         tcp::resolver resolver(
                               io_service_);
28                         tcp::resolver::query query(
                               hostname, port);
29                         boost::asio::connect(*socket_,
                               resolver.resolve(query),
                               error);
30
31                         if (error)
32                                 throw boost::system::
```

```
                                system_error(error);
33
34              }
35
36              void Accept(int port)
37              {
38                      tcp::acceptor acceptor(
                           io_service_, tcp::endpoint(
                           tcp::v4(), 1445));
39                      acceptor.accept(*socket_);
40
41              }
42
43              void Write(string msg)
44              {
45                      socket_->write_some(boost::asio
                           ::buffer(msg));
46              }
47
48              string Read()
49              {
50
51                      boost::system::error_code error
                           ;
52                      size_t len = socket_->read_some
                           (boost::asio::buffer(buf_),
                           error);
53                      if (error == boost::asio::error
                           ::eof)
54                      {
55                              //connection down
56                      }
57                      else if (error)
58                              throw boost::system::
                                   system_error(error);
59
60                      string data;
61                      data.assign(buf_.data(), len);
62                      return data;
63              }
64
```

```
65                      void noDelay(bool no_delay)
66
67                      {
68                              socket_->set_option(tcp::
                                   no_delay(no_delay));
69                      }
70
71                      void Close()
72                      {
73                              socket_->close();
74                      }
75                      bool isAlive() const
76                      {
77                              return socket_->is_open();
78                      }
79
80              private:
81                      tcp::socket* socket_;
82                      boost::asio::io_service io_service_;
83                      boost::array<char, 128> buf_;
84  };
85
86  #endif
```

serial.hpp

```
1  #include <boost/asio.hpp>
2  #include <iostream>
3
4  using namespace std;
5  using namespace boost;
6
7  class Serial {
8  public:
9          Serial(std::string port, unsigned int baud_rate
               ) : io(), serial(io, port)
10         {
11                 serial.set_option(asio::
                       serial_port_base::baud_rate(
                       baud_rate));
12         }
13
14         void Write(std::string s) {
15                 s += '\n';
16                 asio::write(serial, asio::buffer(s.
                       c_str(), s.size()));
17         }
18
19         std::string Read() {
20                 char c;
21                 std::string result;
22                 for (;;) {
23                         asio::read(serial, asio::buffer
                               (&c, 1));
24                         switch (c) {
25                                 case '\r':
26                                 break;
27                         case '\n':
28                                 return result;
29                         default:
30                                 result+=c;
31                         }
32                 }
33         }
34 private:
```

```
35          asio::io_service io;
36          asio::serial_port serial;
37  };
```