YRKESHÖGSKOLAN NOVIA

Utveckling av prototyp för en industriell robotarm

Jakob Andersson

Examensarbete för ingenjörs (YH)-examen

Utbildningsprogrammet för el- och automationsteknik

Vasa 2020

EXAMENSARBETE

Författare: Jakob Andersson
Utbildning och ort: El- och automationsteknik, Vasa
Inriktningsalternativ: Automationsteknik
Handledare: Matts Nickull

Titel: Utveckling av prototyp för en industriell robotarm

Datum 10.05.2020 Sidantal 30

Abstrakt

Industriella robotlösningar har sedan länge varit en betydande del av världens industri. Industriella robotar används globalt i ett stort antal industrier och implementering av nya robotlösningar ökar årligen, med en ökning på 400 000 enheter under året 2018.
Examensarbetet behandlar uppbyggnaden av en artikulerande robotarm, som styrs via ett gränssnitt beläget på ett Linux-system genom seriell kommunikation.

Den teoretiska delen av arbetet beskriver tillvägagångsätt och riktlinjer som bör användas för att designa tredimensionella objekt, metoder för att lösa kinematik problem och beskriver den använda hårdvaran och mjukvaran. Den praktiska delen av arbetet beskriver designstegen av en fysisk robotarm, upprätthållning av seriell kommunikation mellan gränssnitt och hårdvara, konfigurering av hanteringsgränssnittet Rviz samt programmering i kodspråken C/C++, för att behandla ledpositionsdata och konvertera positionerna till användbara styrsignaler för stegmotorer.

Resultatet är en fungerande prototyp som är designad i CAD-verktyget Fusion 360 och styrs av seriella meddelanden från användargränssnittet Rviz, som är beläget på ett Linux-system.

Språk: svenska Nyckelord: industriell robotik, ROS, Fusion 360, kinematik

BACHELOR'S THESIS

Author: Jakob Andersson
Degree Programme: Electrical Engineering and Automation
Specialization: Automation
Supervisor: Matts Nickull

Title: Development of Prototype for an Industrial Robot Arm

Date: May 10, 2020 Number of pages: 30

Abstract

For a long time, industrial robotic solutions have been a significant part of the world's industry. Industrial robots are used globally in a large number of industries and the implementation of new robotic solutions is increasing annually, with an increase of 400,000 units in the year 2018.
The thesis deals with the construction of an articulating robot arm, which is controlled via an interface located on a Linux system through serial communication.

The theoretical part of the work describes the procedures and guidelines that should be used to design three-dimensional objects, methods to solve kinematics problems, as well as the hardware and software used. The practical part of the work describes the design steps of a physical robot arm, maintenance of serial communication between interface and hardware, configuration of the Rviz management interface and programming in the code languages C/C++ to process joint position data and convert the positions into useful control signals for stepper motors.

The result is a working prototype designed in the Fusion 360 CAD tool and controlled by serial messages from the Rviz user interface, which is located on a Linux system.

Language: Swedish Key words: industrial robotics, ROS, Fusion 360, kinematics

---

Förkortningar

ROS	Robot Operating System
PWM	Pulse Width Modulation
CAD	Computer Assisted Design
URDF	Universal Robot Description Format
SRDF	Semantic Robot Description Format
STL	Stereolithography
CNC	Computer Numerical Control
2D	Tvådimensionell
3D	Tredimensionell

Innehållsförteckning

1 Inledning ... 1
1.1 Syfte ... 1
1.2 Avgränsning ... 1
2 Teori ... 2
2.1 Design ... 2
2.1.1 Mekanisk belastning ... 2
2.1.2 Funktionalitet ... 3
2.1.3 DOF ... 3
2.1.4 Dimensionering av länkar ... 3
2.1.5 Dimensionering av motorer ... 4
2.2 Designverktyg ... 6
2.3 3D-printning ... 6
2.4 ROS ... 7
2.4.1 MoveIt ... 8
2.4.2 URDF ... 9
2.4.3 Rviz ... 9
2.5 Arduino ... 9
2.6 Seriell kommunikation ... 10
2.7 Kinematik ... 11
2.7.1 Framåt kinematik ... 12
2.7.2 Inverterad kinematik ... 12
3 Utförande ... 13
3.1 Designkrav ... 13
3.2 Design ... 14
3.3 3D-printning ... 19
3.4 URDF ... 19
3.5 Krets ... 21
3.6 ROS-server ... 22
3.7 Behandling av styrsignal ... 23
3.8 Mikrokontrollerkod ... 25
3.9 Test ... 26
4 Resultat ... 28
5 Diskussion ... 29
6 Referenser ... 31

1 Inledning

Detta examensarbete utförs på eget initiativ och jag har valt att skapa en industriell robotarm för att bygga kunskap inom ämnet och speciellt inom CAD-design. Artikulerande robotar i dessa kompakta dimensioner används generellt inte i industrier, men det finns trots allt flera arbetsområden var en kompakt robotarm kan utföra arbeten effektivt. Detta arbete kommer att fungera som en bas för att vidareutveckla prototypen till en användbar robotarm med flera verkliga användningsområden. Arbetets vikt ligger huvudsakligen på att skapa en lösning som kommunicerar seriellt mellan ett gränssnitt och en robot, där roboten utför givna arbeten/rörelser som är beräknade med inverterad kinematik.

Teoridelen ger en inblick i kunskapen och verktygen som krävs för att genomföra arbetet. Utförandet innehåller de viktigaste delarna av konfigureringar, kodsträngar och designegenskaper. Arbetet avslutas genom en sammanfattning av resultatet, och slutligen en diskussion kring arbetet.

1.1 Syfte

Syftet med detta arbete var att skapa en industriell robotarmsprototyp från grunden. Prototypen skall designas i programmet Autodesk Fusion 360 och designen printas ut med en 3D-printer. Roboten skall styras med en Arduino mikrokontroller, som kommunicerar via meddelanden från gränssnittet Rviz beläget på ett underliggande Linux-system.

1.2 Avgränsning

Ämnet är brett och processen att skapa en robotarmsprototyp är tidskrävande. Arbetet avgränsas till att huvudsakligen fokusera på kommunikationen mellan roboten och gränssnittet, samt omskrivning av seriella meddelanden för att anpassa sig för styrningen av stegmotorer.

2 Teori

Industriella robotarmar rör sig i en omgivning baserat på förhandsbestämda rörelseprocesser, vilket ofta är utlösta av givarsignaler. De förhandsbestämda rörelserna skapas i ett gränssnitt och samtliga rörelser beräknas med inverterad kinematik för att säkerställa en smidig samt stabil rörelse. Robotarmar är uppbyggda av leder och länkar som vanligtvis styrs av elmotorer.

Teorin ger en bild över viktiga designelement av en robotarm, designverktyget Autodesk Fusion 360, 3D-printning, Operativsystemet ROS, ROS-verktygen MoveIt och Rviz, mikrokontrollern Arduino samt seriell kommunikation mellan Linux-system och mikrokontrollerkort.

2.1 Design

Vid planering av en industriell robotarm bör följande aspekter tas i beaktande:

- Mekanisk belastning
- Funktionalitet
- Val samt dimensionering av motorer
- Viktfördelning
- Grader av frihet

2.1.1 Mekanisk belastning

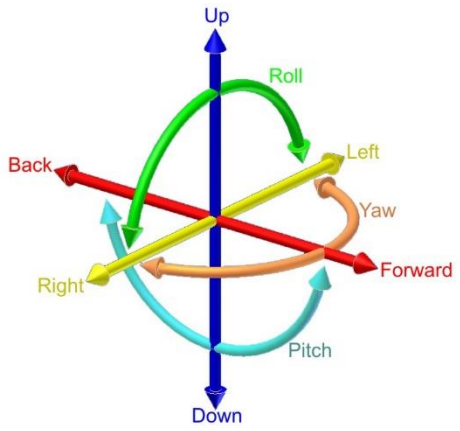
Den mekaniska belastningen på robotarmens komponenter bör planeras så att samtliga komponenter klarar av den önskade maximala belastning i ändmanipulatorn. Den mekaniska belastningen har en direkt påverkan på hur smidigt en robot rör sig i ett tredimensionellt utrymme. Om en design inte tagit delarnas belastbarhet i beaktan, kan en hög belastning orsaka centrerade stresspunkter.

2.1.2 Funktionalitet

En robotarm bör ha förmågan att röra sig i ett tredimensionellt utrymme samt lyfta, sänka, flytta objekt med hjälp av en ändmanipulator. En robotarms funktionalitet är direkt kopplat till robotens grader av frihet.

2.1.3 DOF

Grader av frihet (DOF), är i mekaniska sammanhang specifika definierade lägen där en mekanisk anordning kan röra sig. Antalet frihetsgrader är det samma som det totala antalet aspekter av rörelse.



Figur 1. Sex grader av frihet [1].

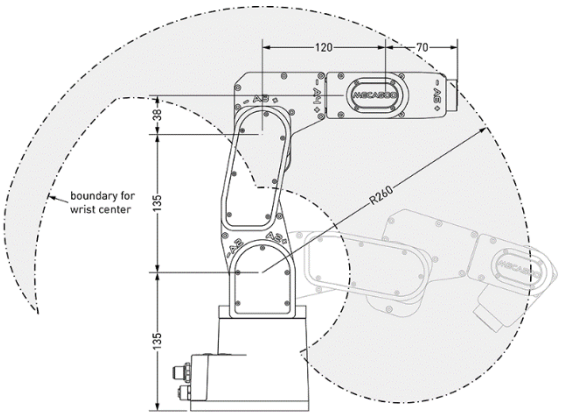
Det finns sex stycken rörelser i axlar som bör tas i beaktan. Rörelserna kategoriseras i två grupper, translationsaxelrörelser och rotationsaxelrörelser. Till translationsaxelrörelser hör förflyttning av X-, Y- och Z-axlar, d.v.s. upp/ner, framåt/bakåt och vänster/höger. Till rotationsaxelrörelser hör rotering av X-, Y- och Z-axlar. Rotering av X axeln benämns som Pitch, rotering av Y axeln benämns som Roll och rotering av Z axeln benämns som Yaw.

Vanligtvis har artikulerade robotarmar 5–7 grader av frihet. Ett högre antal grader av frihet kan även förenkla uträkningen av inverterad kinematikslösningar. [2].

2.1.4 Dimensionering av länkar

För att uppnå maximal arbetsyta bör samtliga leder ha största möjliga rotationslängd. Nackdelen med en hög rotationslängd är att belastningen centreras på en mindre yta och den maximala möjliga belastningen sjunker. Arbetsytan definieras som vertikal och horisontell

arbetsyta. Den tvådimensionella arbetsytan illustreras som en sidovy av roboten som visar dess vertikala räckvidd.

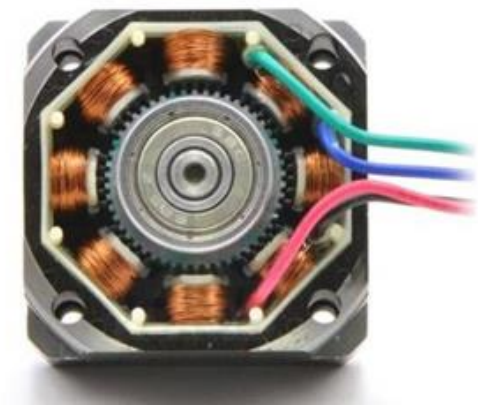


Figur 2. Räckvidden av en Meca500 robotarm. [3]

2.1.5 Dimensionering av motorer

Industriella robotoarmar i mindre skala styrs av servomotorer eller stegmotorer. Servomotorer styrs med pulsbreddsmodulering (PWM), vilket är ett kostnadseffektiv alternativ för att reglera motorer. PWM styrning skickar en pulserande signal till en servomotor. Pulserna skickas med ett konstant intervall och längden av pulserna definierar motorns hantering. När pulssignalen är 1,5 ms är servomotorns axel centrerad. Vid 2 ms är stegmotorns axel svängd 90° till höger och vid 1 ms är stegmotorns axel svängd 90° till vänster.

Stegmotorer styrs av kontinuerliga styrdonskretsar som styr motorns styrsignaler. Stegmotorer roterar genom att strömsätta elektromagneters lidningar och reglera lindningarnas strömriktning. Motorns rotoraxel roterar i förhållande till lindningarnas strömriktning för att reluktansen ska bli så låg som möjligt.



Figur 3. Stegmotorns uppbyggnad. [4]

För- och nackdelarna av de två typerna är följande:

Föredelar med servomotorer

- Kompakt design
- Hög precision
- Vinkeln är alltid känd

Nackdelar med servomotorer

- De vanligaste servomotortypernas rörliga vinkel är endast 180°
- Hög kostnad för större servomotorer

Föredelar med stegmotorer

- Starka
- Låg kostnad för hög effektivitet
- Hög precision
- Ingen maximal rotations vinkel

Nackdelar med stegmotorer

- Motorns position är okänd

- Hög vikt
- Stor design
- Kräver högre styrström samt en drivkrets.
- Låg rotationshastighet

2.2 Designverktyg

Designverktyget som används är Autodesk Fusion 360. Fusion 360 är ett CAD-verktyg som används för att skapa 3D- och 2D-objekt, skapa design animationer, rendera objekt, simulera belastning av objekt och förbereda objekt inför CNC-bearbetning. Det finns flera arbetsmiljöer och varje miljö förser användaren med särskilda funktioner.

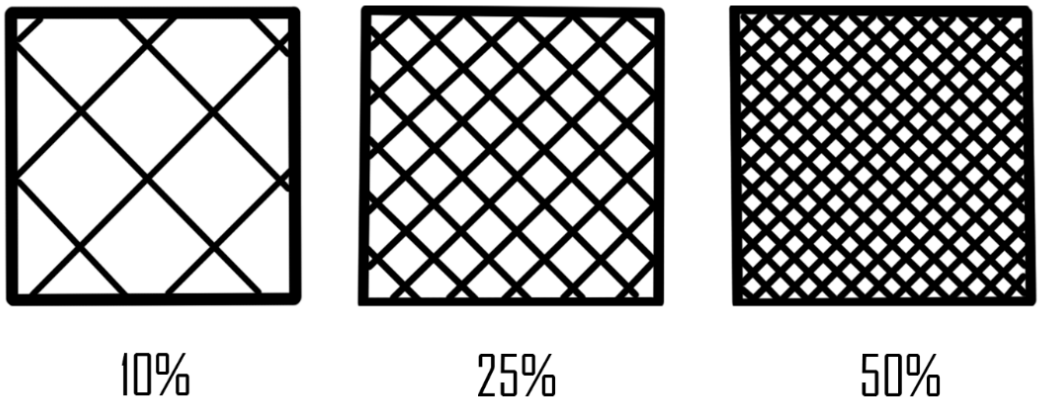
Ifall ett objekts massa inte är jämnt fördelad, är det möjligt att belastningen har en negativ inverkan på objektet. Därför bör det skapas en simulering för att kontrollera kritiska belastningspunkter. I simuleringsmiljön är det möjligt att granska belastningstyper och spänningar för att förstå produktens möjliga belstningsförmåga.

2.3 3D-printning

3D-printning hänvisar till processen att skapa tredimensionella föremål från en CAD-fil. Printningsprocessen fungerar genom att lägga lager på lager av material för att bygga upp ett tredimensionellt objekt. Termoplast filament är det materialet som huvudsakligen används, men printningsmöjligheter inkluderar även användningen av epoxihartser och metaller.

I majoriteten av fallen är inte printade delar totalt fasta objekt utan de har en infyllning, eftersom det är väldigt tidskrävande och kostsamt att fylla ett objekt till en fast massa. Genom att använda en infyllning är det möjligt att skapa ett robust objekt med mindre material. Hos ett objekt som endast planeras fungera som en modell och därför inte kräver strukturell rigiditet, rekommenderas en infyllning på 10 – 20 %. Om objektet belastas så rekommenderas en infyllnad på 40 – 100 %. Den strukturella rigiditeten i relation till infyllnadsmängd är inte linjär, utan rigiditeten ökar mindre per procentenhet infyllnad i samband med att infyllnaden ökar. Därför är den optimala infyllnaden mellan 40 – 50 % för att skapa ett robust objekt med liten mängd material och låg vikt. En annan metod för att

höja rigiditeten är att skapa tjockare ytor. Denna metod är mer tidskrävande men ger en starkare och jämnare yta.



Figur 4. Grov illustrering av infyllnad.

3D-printning kan göras med ett stort antal olika material. Varje material har sina styrkor och svagheter. Därför är det viktigt att veta vad komponenterna kommer utsättas för och vilka kvalitetskrav som måste uppnås. Det billigaste och populäraste materialet som används är PLA, vilket används för sin låga kostnad och lätta printbarhet. Svagheten med PLA är materialets låga värmebeständighet. Ett annat populärt material är ABS, vilket har en relativt låg kostnad och har högre värmebeständighet. Den stora nackdelen ABS är en hög risk för förvridning. [5].

2.4 ROS

ROS är ett operativsystem vars källkod är öppen. ROS är huvudsakligen framtaget för erbjuda ett strukturerat kommunikationslager över ett underliggande operativsystemen. ROS består av en samling bibliotek och verktyg vars ändamål är att förenkla skapandet av komplexa och robusta robotförfaranden över en mängd olika robotplattformer.

ROS är beroende av och kräver mycket funktionalitet utav det underliggande operativsystemet. Detta gör att ROS inte lämpar sig bra med Windows och Mac, vilket är mycket mer begränsade operativsystem när det gäller användarfunktionalitet. Dock är det optimalt för linux-system vars operativsystem har öppen källkod.

ROS är ett interoperabilitet operativsystem, vilket innebär att det är möjligt med funktionalitet mellan olika plattformar som är baserade på olika kodspråk. Detta betyder att

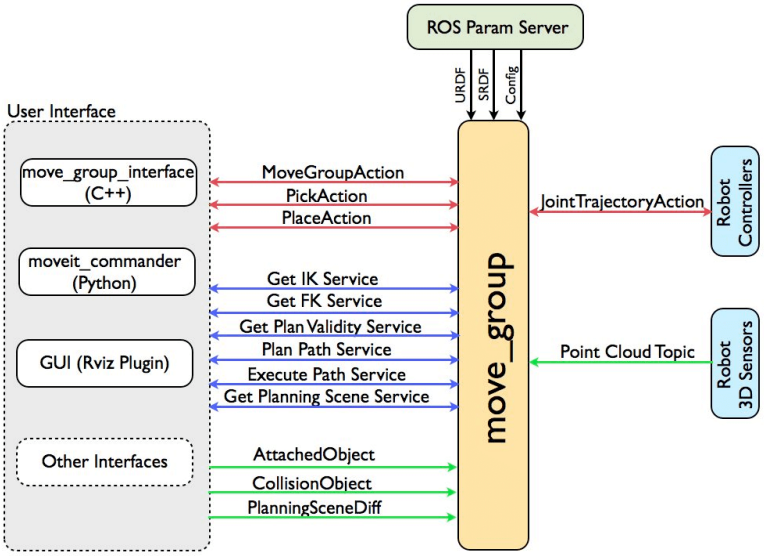
systemets olika noder kan struktureras med det språk som lämpar sig bäst för den specifika nodens funktion.

ROS är modulärt och samtliga processer skrivs på separata noder. Detta resulterar i att de resterande nodernas funktionalitet kan fortsätta även om en nods funktionalitet upphör. [6].

2.4.1 MoveIt

MoveIt erbjuder funktionalitet för kinematik, rörelseplanering, kollisionskontrollering, tredimensionell uppfattning, robotinteraktion och diverse andra funktioner. MoveIt är ett fundamentalt verktyg för en betydande del av funktionaliteten för robotmanipulation med ROS. Med verktyget skapar användaren en SRDF-fil, Yaml-konfigurationer och roslaunch-filer. SRDF-filen definierar robotens position i en miljö, hur den är uppbyggd samt robotens självkollisionsparametrar. SRDF-filen skapas genom att definiera robotens leder baserat på information från en URDF-fil. Lederna definieras i grupper och om flera ändmanipulator används så skall dessa definieras i olika ledgrupper. Yaml-konfigurationerna och Roslaunch-filerna används för att öppna roboten i olika verktyg och simuleringsmiljöer. [4] [6].

MoveIt fungerar med pluginbaserad arkitektur, vars centrala nod heter move_group. Den är avsedd för att hantera olika funktioner, samt integrera rörelseplanering och kinematik. [7].



Figur 5. MoveIt!'s strukturella uppbyggnad. [7]

2.4.2 URDF

Universal Robotic Description Format (URDF) är ett XML-filformat som används i ROS för att beskriva robotelement.

URDF-filer kan enbart specificera de kinematiska och dynamiska egenskaperna hos en robot och kan inte specificera robotens position i en miljö. Vid simulering bör istället en SRDF-fil användas som även specificerar robotens rörelsebegränsningar och ledgrupper.

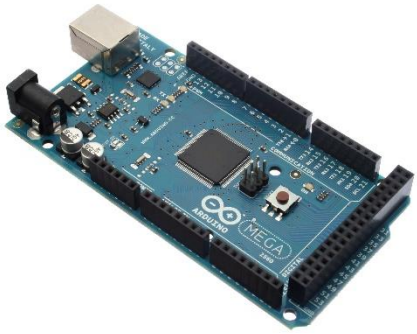
URDF-filer skapas i användarens designverktyg och är baserade på de designade komponenternas STL-filer och deras leder samt ledernas grader av rotation. [8].

2.4.3 Rviz

Rviz är ett tredimensionellt visualiseringsverktyg som används för att illustrera den simulerade robotmodellen, logga sensorinformation från robotens sensorer och spela upp den loggade sensorinformationen. I Rviz kan man utföra rörelseplanering, vilket innebär att skapa en sekvens av rörelser som flyttar roboten från sin nuvarande position till en bestämd destinationen. [9].

2.5 Arduino

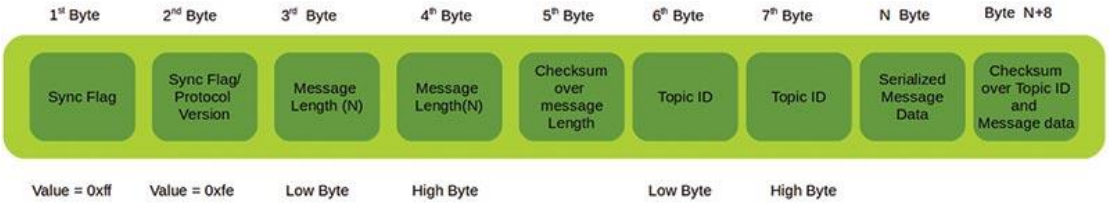
Arduino är ett mikrokontrollerkort som använder sig av Atmel AVR-mikroprocessorer och diverse styrenheter. Korten består av digitala och analoga in-/utgångar. Mikrokontrollerna kan programmeras i både programmeringsspråket C och C++. Arduino erbjuder användaren en integrerad utvecklingsmiljö vid namnet Arduino IDE, varifrån programkoden laddas till mikrokontrollerkortet genom kortets seriella kommunikationsport. Utvecklingsmiljön erbjuder även en mängd olika funktionsbibliotek. När koden kompileras och överförs, ändras C/C++ koden till en fil med hexadecimal kodning som laddas upp till mikrokontrollern. [10].



Figur 6. Arduino Mega 2560.

2.6 Seriell kommunikation

För att skapa en kommunikation mellan ROS och Arduino kortet används ett paket vid namnet Rosserial. Det är ett protokoll som används för att skicka data via ett seriellt gränssnitt. Rosserialprotokollet har en server och en klient. Servern är systemet var ROS befinner sig och klienten är en mikrokontroller. Servern är en publiceringsnod som sänder datapaket till klienten, vilken är en prenumerantnod.



Figur 7. ROS-datapakets struktur. [11]

ROS-meddelanden skickas som ett paket som har en header och tail som tillåter flera ämnen och tjänster från en enda enhet. Paketet innehåller också flaggor för att synkronisera kommunikationen mellan det underliggande operativsystemet och en enhet.

Den första byten kallas Sync Flag som används för att synkronisera kommunikationen mellan ROS och enheten. Den andra byten visar om ROS-versionen är lanserad före ROS Groovy eller efter. Den tredje och fjärde byten representerar längden på paketets meddelande. Den femte byten är en kontrollsumma för meddelandelängden. Den sjätte och den sjunde byten är ämnad för topic-ID för systemfunktioner. Resterande bytes används för seriedata och dess kontrollsumma.

Kontrollsummorna för paketets och datans längd beräknas med följande ekvation i kodspråket C++, där symbolen % är en operator som ger restsumman av en division av två värden:

255 - ((Topic ID Low Byte + Topic ID High Byte + data byte values) % 256)

Kommunikationen mellan serienheten och datorn startar från operativsystemet, vilket skickar ett frågepaket för att få antalet ämnen, namn och typer av ämnen från Arduino enheten. När Arduinon får frågepaketet, returneras en serie svarspaket till datorn. Serien av svar kommer i form av rosserial_msgs. [11].

2.7 Kinematik

Kinematik angående robotarmanipulatorer hänvisar till en analytisk undersökningen av rörelse hos en robotmanipulator. Det är viktigt att formulera lämpliga kinematikmodeller för en robotmekanism i avsikt att analysera beteendet hos robotmanipulatorer. I kinematikmodellering av manipulatorer finns det främst två olika utrymmen som används, Cartesian space och kvarterion space. Transformationen mellan två kartesiska koordinatsystem kan brytas upp till en rotation och en parallellförflyttning.

Figur 8. Kinematik.

Robotkinematik kan delas in i framåt kinematik och inverterad kinematik. Framåt kinematikproblem är simpla och det finns ingen komplexitet vid härledning av ekvationerna. Inverterad kinematik är ett mycket mer avancerat problem och på grund av singulariteter och olinjäriteter är problemen betydligt svårare att lösa. Lösningen av inverterad kinematikproblem är beräkningsmässigt stort och tar i allmänhet mycket lång tid i realtidskontrollen av manipulatorer.

2.7.1 Framåt kinematik

Framåt kinematik hänvisar till kalkylationer för att beräkna positionen av en ändmanipulator baserat på kända ledvinklar och hastigheter. Fördelen med denna metod är att den alltid returnerar en fungerande ekvation för varje önskad position.

2.7.2 Inverterad kinematik

Inverterade kinamtiksekvationer beräknar ledvinklarna som krävs för att flytta en ändmanipulator till en bestämd position. I ett komplext ledat system kan kinematiska ekvationer användas för att definiera icke-linjära begränsningar på konfigurationsparametrarna för systemet. Parametrarna i dessa ekvationer benämns som robotens grader av frihet.

Några av de vanligaste tillvägagångsätten för att lösa inverterad kinematik är CCD, Levenberg-Marquardt algoritm, Jacobian matris. [12].

Levenberg-Marquardt algoritm

LMA används för att lösa icke-linjära minsta kvadratproblem. Det är en kurvmonteringsmetod, vilken består av en kombination av de iterativa algoritmerna, gradientnedstigning och Gauss-Newton. Algoritmen använder en sekvens av beräkningar baserade på antagna värden för att finna en lösning. Med gradientnedstigningen uppdateras lösningnen genom att vid varje iteration välja värden som kommer att minska funktionsvärdet. Genom att röra sig mot den brantaste nedstigningen reduceras summan av kvadratfelen. Gauss-Newton algoritmen har en högre precision och är en effektivare algoritm än gradientnedstignings algoritmen då funktionsvärdet är nära minimifelet. I vissa fall om modellen har mer än tio parametrar, konvergerar LMA mycket långsamt i jämförelse med andra algoritmer. [13].

Cyclic Coordinate Descent

CCD-algoritmen fungerar genom att mäta skillnaden mellan en leds position och ändmanipulatorns position. Därefter beräknas antingen en rotation eller kvarternion för att reducera skillnaden ner till noll. Beräkningen görs för varje led, iterererande genom den kinematiska kedjan från ändmanipulatorn till den fixerade basen. [14].

Jacobian matris

Jacobian matrisalgoritmen är en inkrementell metod för att lösa inverterad kinematiksproblem. Noggranheten eller effektiviteten som krävs bestämmer om Jacobian beräknas ett flertal gånger medans ändmanipulatorn rör sig mot sitt mål eller ett flertal gånger per ram när ändmanipulatorn rör sig mot sin planerade position. [15].

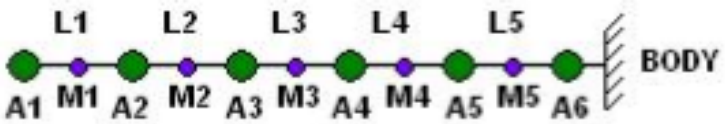
3 Utförande

Robotarmens design är en kompakt lösning som använder sig av utväxlingar för att reducera belastningen på motorerna samt säkerställa att ledernas belastbarhet uppnår de önskade kraven. Roboten styrs över ett gränssnitt som skickar seriella meddelanden som innhåller rörelsedata. Rörelsedatan konverteras för att möjliggöra styrning av stegmotorer och meddelandena skickas vidare till en mikrokontroller som styr den fysiska roboten och speglar rörelser som simuleras i verktyget Rviz.

Uförandet redogör för arbetets samtliga delar, och består av designprocessen i Fusion 360, 3D-printning, URDF-extrahering, vinkelkonvertering, programmering, konfigurering och testning.

3.1 Designkrav

Robotens önskade räckvidd är 40 cm + manipulatorlängden. Ändmanipulatorlängden varierar baserat på användningsändamål. Roboten kommer att styras med stegmotorer av modellen Nema 17. Komponenternas design måste placera motorerna så nära den första leden som möjligt för att minska vikten i ändan av armen, vilket reducerar belastningen på samtliga leder.



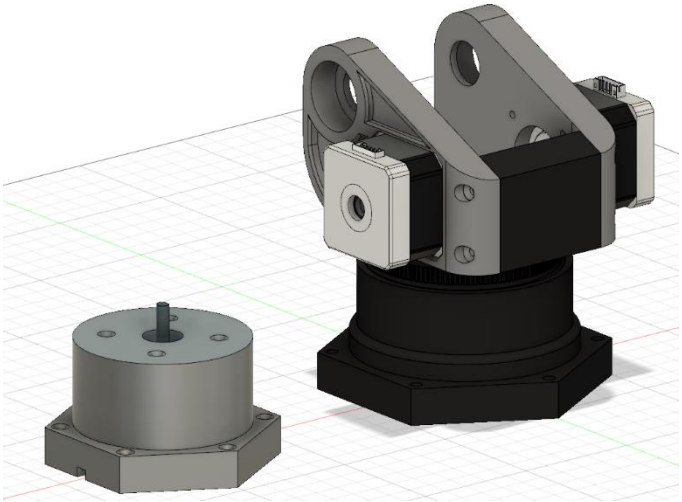
Figur 9. Visualisering av ledar och länkar [16].

Uträkningen som uförs är en approximal uträkning. För att beräkna en exakt belastning måste komponents viktfördelning och mittpunkt vara känd. Eftersom komponenterna inte är symmetriska är det komplicerat att beräkna viktfördelningen och mittpunkten. En antagen

massa adderas till belastning i beräkningarna för att säkerställa att belastningen inte överskrider den proportionerade belastningsförmågan.

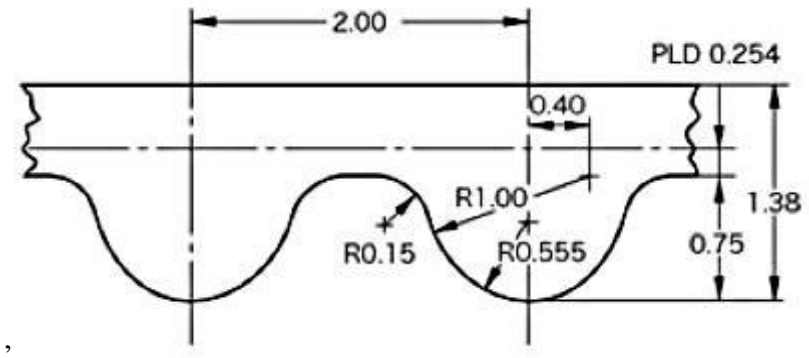
3.2 Design

För att minimera belastningen av motorn som roterar basen, monteras stegmotorn i en extern fixtur och en drivrem monteras mellan motorn och basen. Basen består av två delar, en baskomponent som är fixerad och en komponent som roterar inuti baskomponenten. Basens delar är ihopmonterade i varandra med ett robust kullager, vilket förser basen med en stadig montering, medans roterbarheten kvarstår.



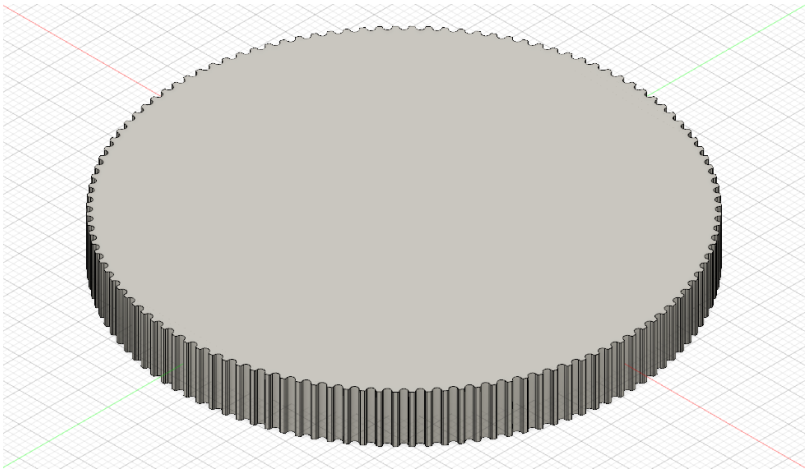
Figur 10. Robotarmens bas och basmotor.

Motorerna driver samtliga leder med remmar av typen GT2. För att möjliggöra användning av remtypen GT2 måste ett kugghjul med samma profil designas.



Figur 11. GT2-profil.

Måtten i figur 11 skrivs in som parametrar i Fusion 360. Parametrarna används för att skapa en tand med inversen av remprofilen och tanden kopieras och infogas runt en axel med ett önskat antal tänder. Basens kugghjul dimensioneras till 100 tänder och basmotorns drev har 20 tänder, vilket resulterar i utväxlingen 5:1.



Figur 12. GT2 kugghjulsdesign.

Den första länken styrs med två motorer för att öka designens belastbarhet. Belastbarheten beräknas med en kalkyl som beräknar den maximala belastbarheten av samtliga motorer, baserat på delarnas massor och längd samt den maximala vikten som robotarmen lyfter. Belastningen av en led beräknas genom att multiplicera ledens längd med ändbelastningen och addera denna produkt med tyngdpunkten. Formeln ser ut enligt följande:

$$Belastning = Ledlängd \cdot Ändbelastning + \frac{1}{2} Ledlängd \cdot Ledmassa$$

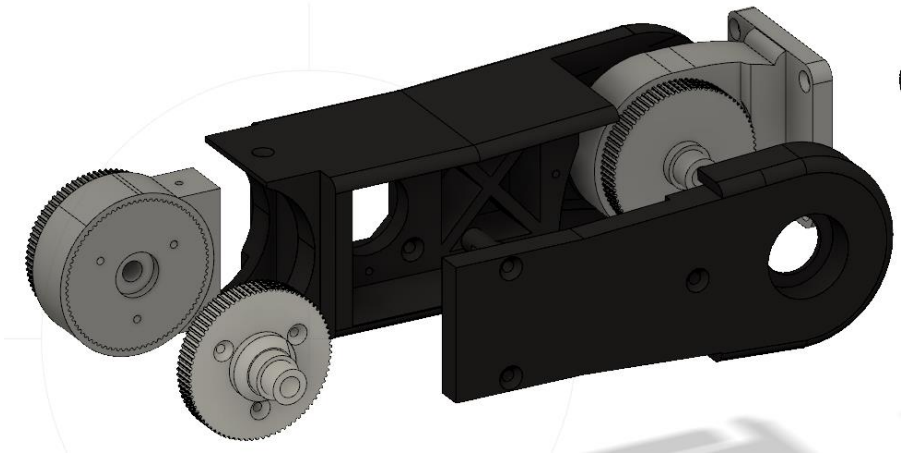
Belastningenberäkning vid basen på en arm som består av flera leder kan simplifieras genom att beräkna den gemensamma tyngpunkten, då alla leder är horisonetella. Genom att beräkna den gemensamma tyngdpunkten kan den ovanstående formel användas.

Enligt belastbarhetskalkylen [16] blir belastningen på länken mellan basen och armen approximalt 29 kg/cm. Leden styrs med två stycken stegmotorer med vridmomentet 4,5 kg/cm per motor och den valda utväxlingen är 4:1, vilket resulterar en belastbarhet på 36 kg/cm.

$$Belastbarhet_{tot} = Belastbarhet * motorer * Utväxling = \frac{4{,}5\ kg}{cm} \cdot 2 \cdot \frac{4}{1} = \frac{36\ kg}{cm}$$

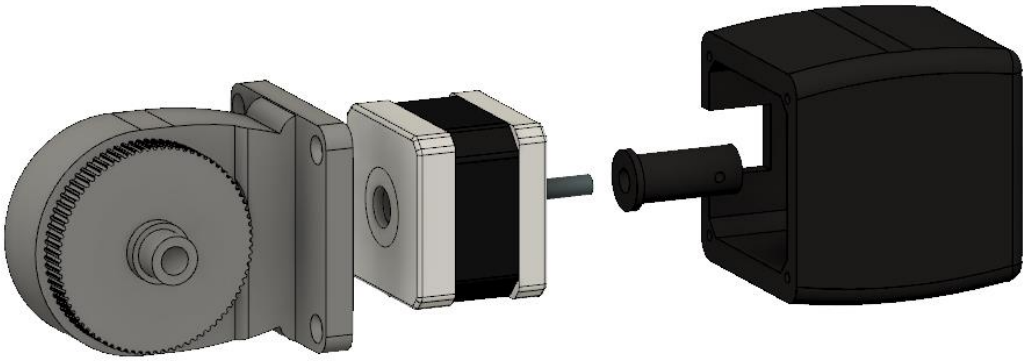
Länken görs i flera delar för att simplifiera 3D-printningsprocessen. I basen av armen monteras en motor som används för att driva leden mellan den första och andra länken. Motorn använder en 4:1 uväxling och bör ha en belastbarhet på 12,75 kg/cm. Baserat på uträkningarna klarar leden av en belastning på 18 kg/cm, då en motor med belastbaheten 4,5 kg/cm används.

$$Belastbarhet_{tot} = Belastbarhet * motorer * Utväxling = \frac{4{,}5\ kg}{cm} \cdot 1 \cdot \frac{4}{1} = \frac{18\ kg}{cm}$$



Figur 13. Sprängskiss av länk 1.

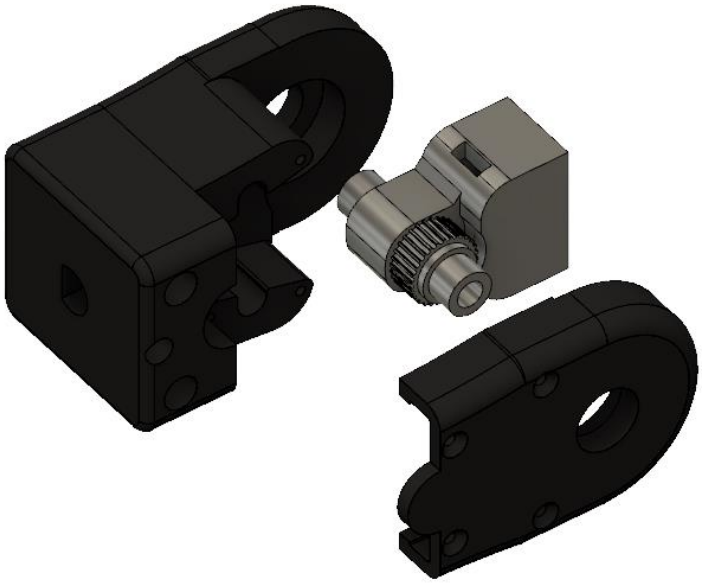
Den andra länken designas för att montera en motor med den utgående axeln i samma riktning som länken. Motorn monteras i samma riktning eftersom den tredje leden roterar i sidled i förhållande till de tidigare två lederna som roterar i höjdled. En motor med belastbarheten 4,5 kg/cm uppfyller kraven och ingen utväxling är nödvändig.



Figur 14. Sprängskiss av länk 2.

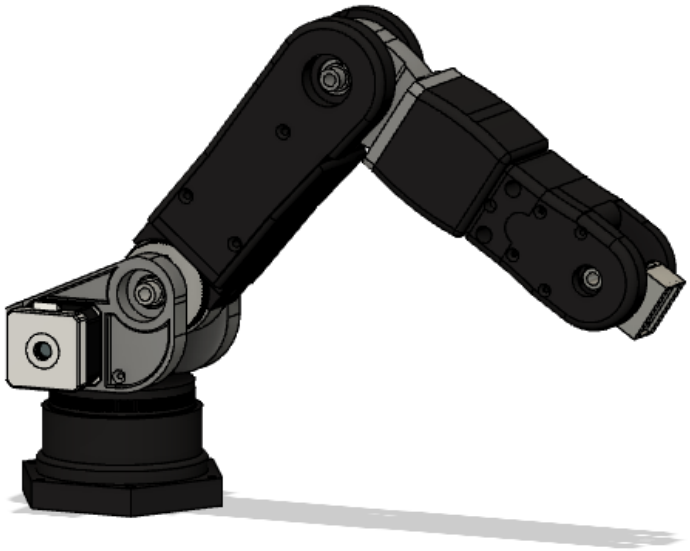
På den tredje länken monteras en motor som driver den slutliga leden, var ändmanipulatorn fästs. Ledens maximala belstning är 1,2 kg/cm. En stegmotor med 1,6 kg/cm belastbarhet och utväxlingen 3:2 används. Den totala belastbarheten för leden är 2,4 kg/cm, vilket är dubbelt mer än den maximala använda belastningen.

$$Holding\ torque_{tot} = \frac{1{,}6\ kg}{cm} \cdot 1 \cdot \frac{3}{2} = \frac{2{,}4\ kg}{cm}$$



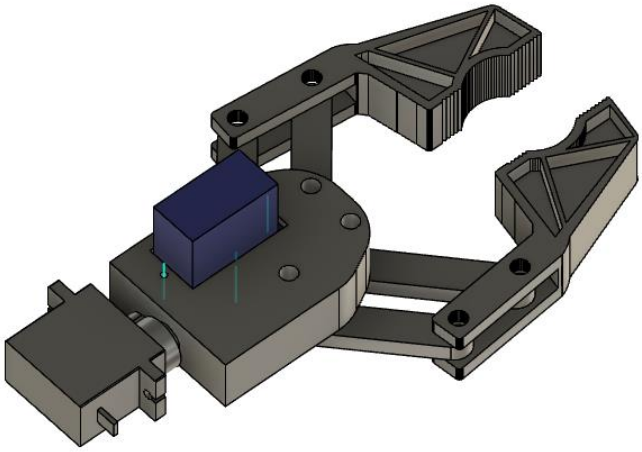
Figur 15. Sprängskiss av länk 3 och 4.

Komponentera förenas i Fusion 360 genom att definiera lederna. Lederna positioneras per automatik till sina korrekta positioner och genom att använda animationsfunktionen är det möjligt att säkerställa att ledernas konfiguration är korrekt.



Figur 16. Den slutgiltiga designen.

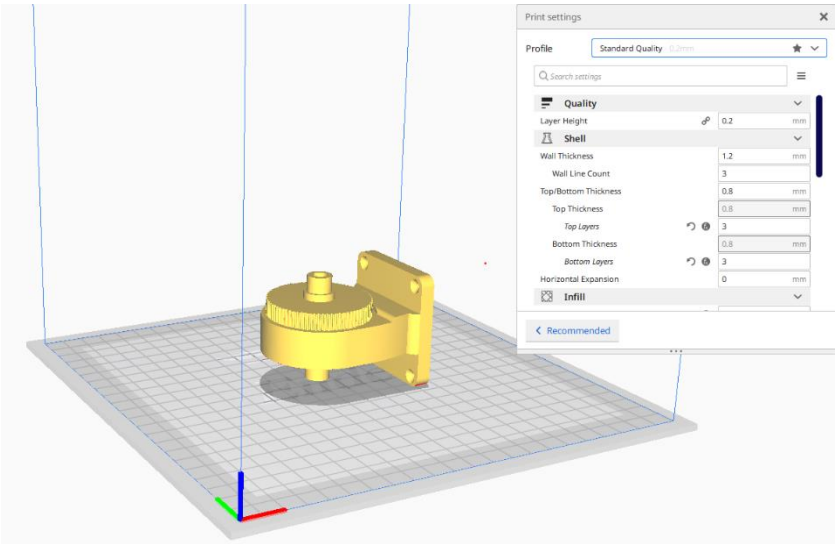
En ändmanipulator designas i form av en klo. Manipulatorn monteras i den fjärde länkens servomotor. För att öppna och stänga klon monteras en servomotor i en av stagens leder. Klon designas för ett universalt bruk, men har en design för att på optimalt sätt klara av att lyfta batterier med liknande design som batteritypen 18650.



Figur 17. Gripper klo.

3.3 3D-printning

Designens samtliga delar exporteras från Fusion 360 i form av STL-filer. För att printa delarna omvandlas STL-filerna till gkod i slicer verktyget Ultimaker Cura.

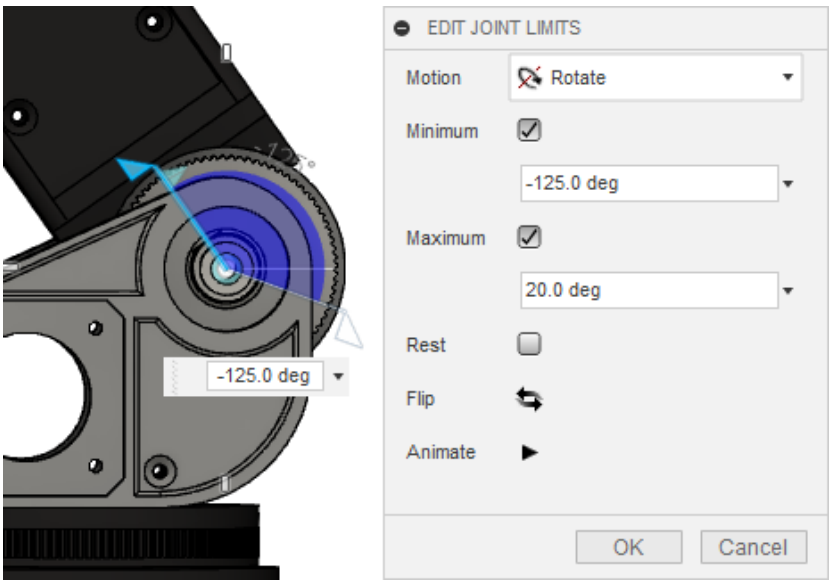


Figur 18. Verktyget Ultimaker Cure.

Samtliga delar printas på en printer av typen Ender Creality 3, vilket har byggvolymen 220 mm x 220 mm x 250 mm varav byggplattan är 220 mm x 220 mm. Detta gör det möjligt att printa samtliga komponenter utan att designa komponenterna i mindre sektioner. För att optimera tidsåtgången och även uppnå hög kvalitet har samtliga komponenter designats i sektioner och sedan monterats ihop. [17].

3.4 URDF

Fusion 360 erbjuder inget verktyg för att skapa en URDF-fil utav designen. Ett verktyg installeras i Windows Powershell och inkluderar ett script i Fusion 360 tilläggsfunktioner för att skapa en URDF-fil i fusion 360. För att verktyget skall fungera så krävs det att lederna och deras rotationsgränser definieras.



Figur 19. Ledens rotationsgräns.

När alla leder är definierade kan scriptet köras. Scriptet skapar en katalog som består av URDF-filen samt STL-filer för varje komponent för att visualisera roboten i diverse ROS-verktyg. URDF-filens innheåll ser ut som följande och innehåller viktig data för samtliga leder och komponenter.

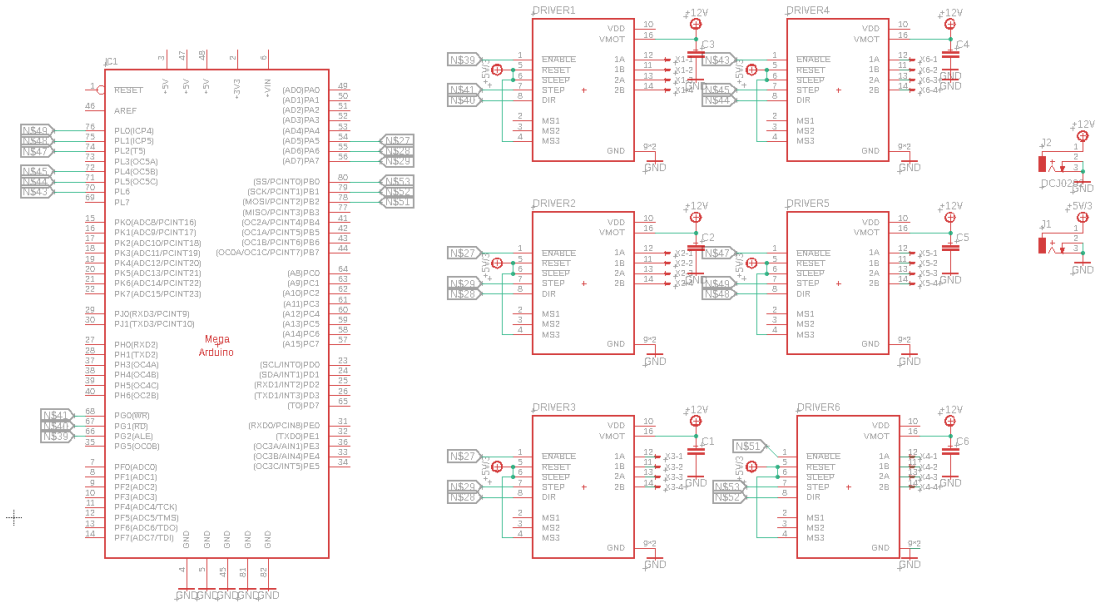
```
<robot name="Untitled">
<link name="base_link">
  <inertial>
    <origin rpy="0 0 0" xyz="3.3463279546094446e-17 5.354124727375112e-17 0.014498960388293965"/>
    <mass value="1.3334641556655853"/>
    <inertia ixx="0.001176" ixy="-0.0" ixz="0.0" iyy="0.001176" iyz="0.0" izz="0.002037"/>
  </inertial>
  <visual>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <geometry>
      <mesh filename="package://fusion2urdf/Untitled/bin_stl/base_link_m-binary.stl"/>
    </geometry>
    <material name="silver">
      <color rgba="1 0 0 1"/>
    </material>
  </visual>
  <collision>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <geometry>
      <mesh filename="package://fusion2urdf/Untitled/bin_stl/base_link_m-binary.stl"/>
    </geometry>
  </collision>
</link>
<link name="Component2_1">
  <inertial>
```

Kodexempel 1. URDF-filens struktur.

3.5 Kretsschema

Ett kretsschema bestående av samtliga komponenter ritas upp i Autodesk Eagle. Kretsen ritas upp genom att huvusakligen använda flaggor, i stället för att dra ledningar mellan komponterna. Detta görs eftersom mängden ledningar resulterar i att ritningen blir otydlig. Komponenterna på kretsen är följande:

- Arduino Mega 2560
- 1 x 12 VDC-input
- 1 x 3,3 VDC-input
- 6 x 100 µF kondensator
- 6 x Stepper Drivers
- 6 x kontakter till stegmotor



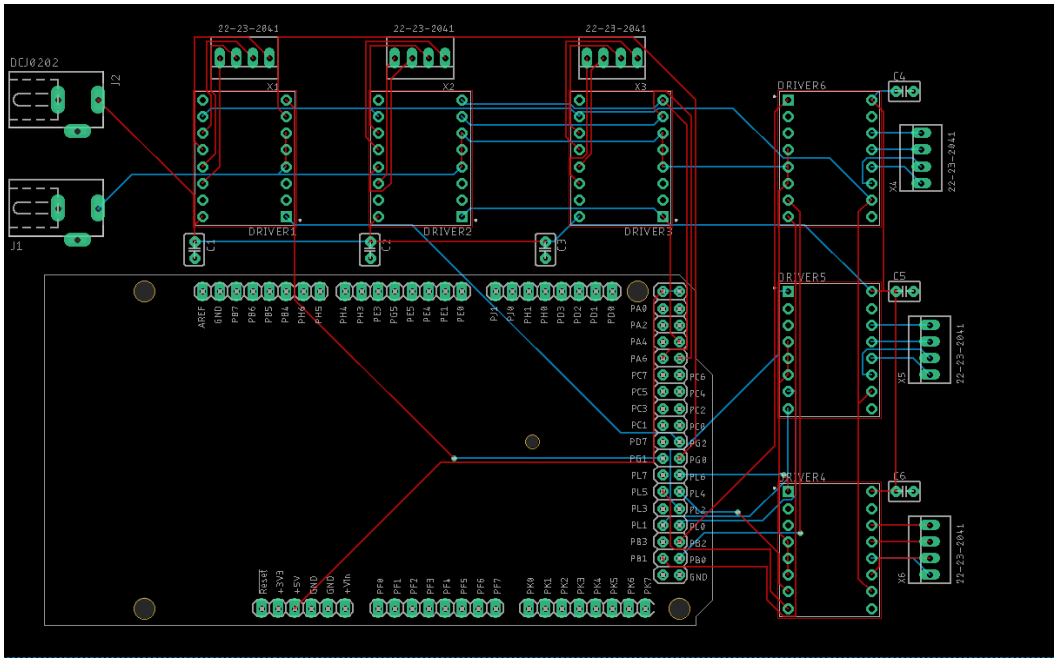
Figur 20. Kretsschema.

Stegmotorerna 3 och 4 hanterar samma led och ansluts till samma styrsignaler. Kondensatorerna ansluts parallellt med motorernas matningsspänning för att stabilisera belastningen. Drivkretsarna har en belastningsström på 1,5 A per komponent och kretsen kräver en total matningsström på minst 9 A. Matningsspänningen för bipolära stegmotorer

kan variera mellan 12 VDC och 48 VDC. I denna tillämpning används 12 VDC. Dock bör det noteras att en högre spänning gör att motorn når magnetisk mättnad snabbare.

Kretskortet ritas i Autodesk Eagle och kretsen ritas med dubbelsidig kretskortskonfiguration.

Testning av kretsen utförs på ett kopplingsdäck och då full funktion upprätthålls, löds komponenterna fast på kretskortet för en permant lösning.



Figur 21. Kretskort.

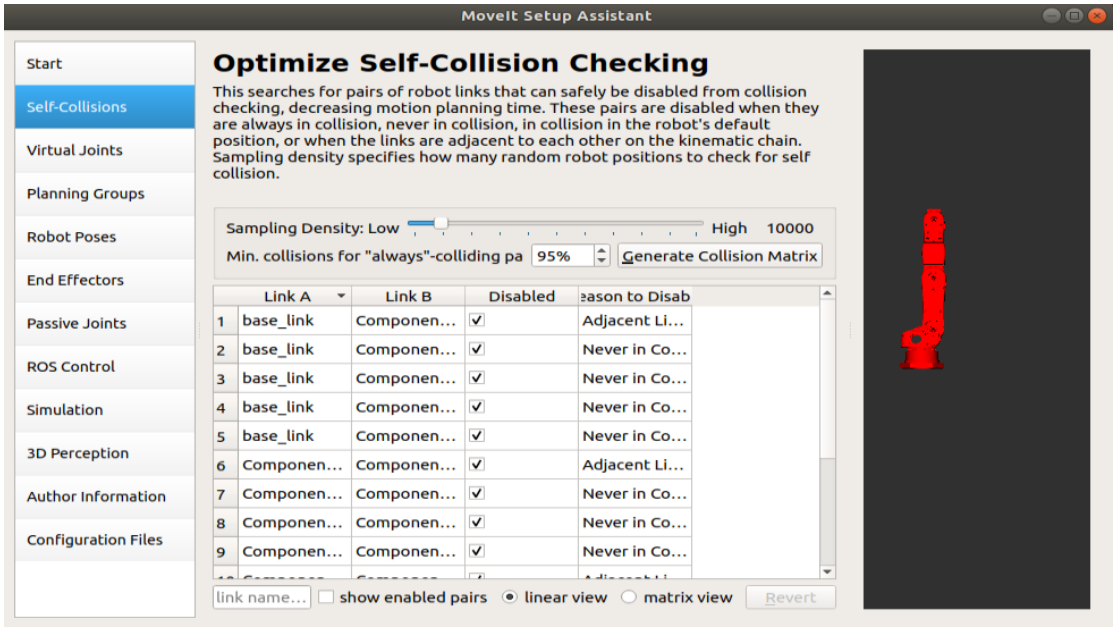
3.6 ROS-server

ROS-servern skapas på en virtuell Linux-maskin. I Linux-systemets terminal installeras ROS Melodic samt alla tillhörande verktyg och bibliotek, såsom rqt, rviz, 2D/3D-simulationsverktyg.

För att skapa en användbar simulering av den fysiska robotarmen används verktyget MoveIt. I samband med att MoveIt installeras, skapas ett Catkin workspace, var robotarmens konfigurering byggs upp i.

En SRDF-fil, roslaunch filer samt Yaml-konfigureringsfiler bör skapas. Detta görs med verktyget MoveIt Setup Assistant som använder sig av robotens URDF-fil. En av de viktigaste egenskaperna med ROS är möjligheten att implementera så kallade IK-solvers, vilket är färdigställda inverterade kinematikslösare. Detta gör att utförandet simplifieras. Med verktyget skapas planeringsgrupper som består av de leder som är önskade att användas

vid en rörelse. Till planeringsgrupperna finns möjligheten att välja ett par olika IK-solvers eller möjligheten att skapa en egen. Följande steg är att definiera en virtuell led, för att specifiera var i omgivningen som roboten befinner sig i. Leden definieras mellan basen och omgivningen och eftersom basen inte är rörlig så definieras leden som fixerad. Verktyget används även för att generera och definiera rörelserer som orsakar självkollision baserat på robotens design och ledernas maximalvinkel. Verktyget gör självkollisionsberäkningar baserat på ett manuellt valt antal samplingar. Ju fler samplingar man gör, desto bättre självkollisionsberäkningar gör verktyget. Samplingsantalet 10 000 väljs baserat på tillverkarnas rekommenderade samplingsantal.



Figur 22. Verktyget MoveIt Setup Assistant.

I verktyget bör samtliga ändmanipulatorer definieras. För att definiera dessa så måste de ingå i robotens URDF-fil. När samtliga konfigureringsfiler är skapade, är det möjligt att simulera roboten i verktyget Rviz.

3.7 Behandling av styrsignal

Den simulerade robotens rörelser i Rviz avläses för att spegla rörelsen med den fysiska roboten. Robotens rörelser baseras på ledvinklar och måste konverteras till steg för att styra stegmotorer. Konverteringskoden skrivs i kodspråket C++ och sparas i robotens konfigurationskatalog. Konverteringen görs med en simpel matematisk formel som används för samtliga leder. Formeln läser av vinkeln i radianer som en led har roterat och

multiplicerar värdet med det antal steg som ledens stegmotor tar för att roterara 360 grader. Denna produkt divideras med $2 * \pi$.

$$Steg = (vinkel\ - Tidigare\ vinkel) \cdot motorsteg\ per\ revolution/(2 \cdot \pi)$$

Uträkningen görs för samtliga leder och vinklarna adderas med det nuvarande vinkelpostionen för att fortsättningsvis veta robotens position.

```
void Callback(const sensor_msgs::JointState& cmd_arm)
{
    if (count == 0)
    {
      previous_angle[0] = cmd_arm.position[0];
      previous_angle[1] = cmd_arm.position[1];
      previous_angle[2] = cmd_arm.position[2];
      previous_angle[3] = cmd_arm.position[3];
      previous_angle[4] = cmd_arm.position[4];
      previous_angle[5] = cmd_arm.position[5];
    }

    arm_steps.position1 = (int)((cmd_arm.position[0]-previous_angle[0])*stepsREV[0]/(2*M_PI));
    arm_steps.position2 = (int)((cmd_arm.position[1]-previous_angle[1])*stepsREV[1]/(2*M_PI));
    arm_steps.position3 = (int)((cmd_arm.position[2]-previous_angle[2])*stepsREV[2]/(2*M_PI));
    arm_steps.position4 = (int)((cmd_arm.position[3]-previous_angle[3])*stepsREV[3]/(2*M_PI));
    arm_steps.position5 = (int)((cmd_arm.position[4]-previous_angle[4])*stepsREV[4]/(2*M_PI));
    arm_steps.position6 = (int)((cmd_arm.position[5]-previous_angle[5])*stepsREV[5]/(2*M_PI));

    total.position1 += arm_steps.position1;
    total.position2 += arm_steps.position2;
    total.position3 += arm_steps.position3;
    total.position4 += arm_steps.position4;
    total.position5 += arm_steps.position5;

    if (count != 0)
    {
      previous_angle[0] = cmd_arm.position[0];
      previous_angle[1] = cmd_arm.position[1];
      previous_angle[2] = cmd_arm.position[2];
      previous_angle[3] = cmd_arm.position[3];
      previous_angle[4] = cmd_arm.position[4];
      previous_angle[5] = cmd_arm.position[5];
    }

    joint_state = 1;
    count=1;
}
```

Kodexempel 2. Vinkel till stegs-konvertering.

De beräknade stegen läggs till i den verkliga robotens positionsvärde och publiseras till filen ArmJointState.

```
int main(int argc, char **argv)
{
  ros::init(argc, argv, "robot_moveit");
  ros::NodeHandle nh;
  ros::Subscriber sub = nh.subscribe("/move_group/fake_controller_joint_states",1000,Callback);
  ros::Publisher pub = nh.advertise<robot_moveit::ArmJointState>("joint_steps",50);

  ros::Rate loop_rate(20);

  while (ros::ok())
  {
    if(joint_state == 1)
    {
      joint_state = 0;
      pub.publish(total);
    }

    ros::spinOnce();
    loop_rate.sleep();
  }


  return 0;
}
```

Kodexempel 3. Det konverterade värdet publiserar.

3.8 Mikrokontrollerkod

Mikrokontrollerns kod skrivs i Arduino IDE-verktyget och i kodspråket C. I koden definieras stegmotorstyrkretsarnas In-/Out-pins till variabler och därefter definieras dessa variabler för Joint funktioner som fås från biblioteket Accelstepper. Accelstepper används för att förenkla hanteringen av stegmotorer.

```
MultiStepper steppers;
Servo gripper;
int jointSteps[6];
int jointStatus = 0;

AccelStepper joint1(1,J1_Step, J1_Dir);
AccelStepper joint2(1,J2_Step, J2_Dir);
AccelStepper joint3(1,J3_Step, J3_Dir);
AccelStepper joint4(1,J4_Step, J4_Dir);
AccelStepper joint5(1, J5_Step, J5_Dir);
```

Kodexempel 4. Definiering av variabler och biblioteksfunktioner.

Mikrokontrollern prenumererar på meddelanden från ArmJointState som innehåller motorernas önskade rörelser i form av steg.

Motorernas hastighet definieras och samtliga motorer läggs till i Multistepper bibliotekets funktioner för att hanteras. Stegmotorernas önskade position från prenumerationsmeddelanden lagras i en array och om servern skickar iväg ett meddelande, ankallas en funktion som ger jointStep arrayens innhåll till MultiStepper positionen och rörelsen utförs.

```
void arm_cb(const robot_moveit::ArmJointState& arm_steps)
{
  jointStatus = 1;

  jointSteps[0] = arm_steps.position1;
  jointSteps[1] = arm_steps.position2;
  jointSteps[2] = arm_steps.position3;
  jointSteps[3] = arm_steps.position4;
  jointSteps[4] = arm_steps.position5;
  jointSteps[5] = arm_steps.position6;
}

ros::Subscriber<robot_moveit::ArmJointState> arm_sub("jointSteps",arm_cb);
```

Kodexempel 5. Läser in stegen från ROS-meddelande.

När rörelsen är utförd, så stänger grippern genom givet meddelande från Rivz eller manuellt med terminalen. När den önskade rörelsen från meddelandet är utfört, stannar roboten och väntar på nästa meddelande.

```
void loop()
{
  if (jointStatus == 1)
  {
    long positions[5];
    positions[0] = jointSteps[0];
    positions[1] = -jointSteps[1];
    positions[2] = jointSteps[2];
    positions[3] = jointSteps[3];
    positions[4] = -jointSteps[4];

    steppers.moveTo(positions);
    nh.spinOnce();
    steppers.runSpeedToPosition();
    gripper.write(jointSteps[5]);
  }
jointStatus = 0;

nh.spinOnce();
delay(1);
}
```

Kodexempel 6. Stegen läses in och skickas till motorhanteringen.

3.9 Test

I Linux-terminaler startas verktyget Rviz och i en seperat terminal påbörjas kommunikationen mellan Arduinon och ROS, genom att definiera seriella porten som mikrokontrollern är ansluten till.

```
ub@ub-VirtualBox: ~/ws_moveit
File  Edit  View  Search  Terminal  Help
ub@ub-VirtualBox:~/ws_moveit$ source devel/setup.bash
ub@ub-VirtualBox:~/ws_moveit$ rosrun rosserial_python serial_node.py /dev/ttyUSB
0
[INFO] [1586876048.028448]: ROS Serial Python Node
[INFO] [1586876048.038181]: Connecting to /dev/ttyUSB0 at 57600 baud
[INFO] [1586876050.201651]: Requesting topics...
[INFO] [1586876050.261910]: Note: subscribe buffer size is 512 bytes
[INFO] [1586876050.262557]: Setup subscriber on joint_steps [robot_moveit/ArmJoi
ntState]
```

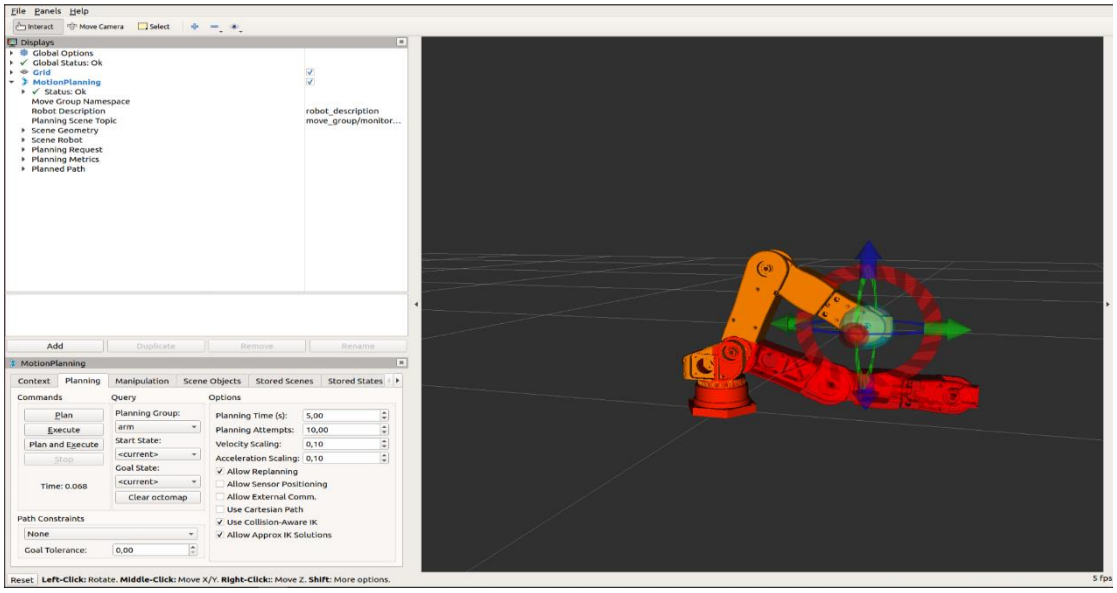
Figur 23. Seriell kommunikation upprättas.

När kommunikationen är upprättad, körs konverteringskoden i en separat terminal. I terminalen syns alla konverteringsmeddelanden som utförs och skickas vidare till prenumerantens meddelande Joint_state.

```
ub@ub-VirtualBox:~/ws_moveit$ rosrun robot_moveit moveit_convert
[ INFO] [1586876426.732063783]: In main function
[ INFO] [1586876426.992316978]: Received /move_group/fake_controller_joint_state
s
[ INFO] [1586876426.992440434]: arm_steps.position5 #2: 0
[ INFO] [1586876426.992522553]: total_steps[4]: 0.000000, total: 0
[ INFO] [1586876426.992608606]: arm_steps.position5 #3: 0
[ INFO] [1586876426.992686697]: Done conversion to /joint_steps
[ INFO] [1586876427.042139340]: Published to /joint_steps
```

Figur 24. Konverteringskoden utförs.

När all kommunikation är startad och nödvändiga verktyg/koder körs, skapas ett arbete i Rviz. Roboten rörs i programmet genom att flytta på ändmanipulatorn med hjälp av de interaktiva markörerna och roboten bibehåller vinkeln på ändmanipulatorn. När den önskade rörelsen är skapad, planeras och utförs rörelsen.



Figur 25. Förflyttning av roboten i Rviz.

När verktyget utför arbetet, är rörelsen åtkomstbar i Fake_controller_joint_states. Termialen som kör konverteringskoden läser aktivt data från Fake_controller_joint_states och konverterar vinkeldatan till steg som skickas i meddelandet joint_states till mikrokontrollern.

Meddelandet sänds ut och mikrokontrollern läser av meddelandet. Meddelandets innehåll behandlas i koden och samtliga motorers rörelser utförs i form av steg i realtid. Den fysiska robotens rörelse efterliknar den simulerade robotens rörelse.

```
[ INFO] [1586876717.392813176]: Received /move_group/fake_controller_joint_states
[ INFO] [1586876717.393034157]: arm_steps.position5 #2: 13
[ INFO] [1586876717.393162332]: total_steps[4]: 0.000000, total: 40
[ INFO] [1586876717.393284491]: arm_steps.position5 #3: 13
[ INFO] [1586876717.393407815]: Done conversion to /joint_steps
[ INFO] [1586876717.442208051]: Published to /joint_steps
[ INFO] [1586876717.496311645]: Received /move_group/fake_controller_joint_states
[ INFO] [1586876717.496541487]: arm_steps.position5 #2: 13
[ INFO] [1586876717.496692400]: total_steps[4]: 0.000000, total: 53
[ INFO] [1586876717.496809664]: arm_steps.position5 #3: 13
[ INFO] [1586876717.496928472]: Done conversion to /joint_steps
[ INFO] [1586876717.542741664]: Published to /joint_steps
[ INFO] [1586876717.601752048]: Received /move_group/fake_controller_joint_states
[ INFO] [1586876717.601975991]: arm_steps.position5 #2: 22
[ INFO] [1586876717.602097590]: total_steps[4]: 0.000000, total: 75
[ INFO] [1586876717.602212531]: arm_steps.position5 #3: 22
[ INFO] [1586876717.602328494]: Done conversion to /joint_steps
[ INFO] [1586876717.642828995]: Published to /joint_steps
[ INFO] [1586876717.692001938]: Received /move_group/fake_controller_joint_states
[ INFO] [1586876717.692225866]: arm_steps.position5 #2: 33
[ INFO] [1586876717.692354736]: total_steps[4]: 0.000000, total: 108
[ INFO] [1586876717.692503030]: arm_steps.position5 #3: 33
[ INFO] [1586876717.692625779]: Done conversion to /joint_steps
[ INFO] [1586876717.742462396]: Published to /joint_steps
```

Figur 26. Konvertering av positioner till steg.

4 Resultat

Industriell robotik är ett brett ämne och utvecklingen av en prototyp för en industriell robotarm kan utföras genom en mängd olika tillvägagångssätt. För att säkerställa att önskade egenskaper och mål uppfylls, studeras viktiga designaspekter gällande en robotarms funktionalitet. Kunskapen implementeras i designutförandet och en fungerande design skapas i CAD-verktyget Fusion 360.

Industriella robotar utför arbeten som är skapade i ett gränssnitt genom att definiera rörelser eller positioner. Gränssnitten är ofta skräddarsydda program, som är skapade av företaget som tillverkat roboten. Alternativet för detta arbetet är att använda ett gränssnitt vars källkod är öppen. Efter studering av möjliga gränssnitt, väljs operativsystemet ROS. Operativsystemet installeras på en Linuxmaskin och den prototypens design infogas i ROS genom användningen av en URDF-fil. Operativsystemets nödvändiga konfigureringar görs och en simulerad version av den designade prototypen kan illustreras i ROS-verktyget Rviz.

De vanligaste metoderna för att driva småskaliga robotarmar är med stegmotorer eller servomotorer. Båda alternativen har för- och nackdelar när det gäller hantering av leder. Baserat på de angivna kraven, är stegmotorer den drivmetod som lämpar sig bäst för denna prototyp. För att hantera prototypens stegmotorer via Rviz-gränssnittet måste den simulerade robotarmens positionsmeddelanden konverteras till steg. Ett skript skrivs i kodspråket C++ för att konvertera postionerna till steg. Stegen skickas via seriell kommunikation till en mikrokontroller som behandlar de mottagna meddelandena och skickar styrsignaler till

stegmotorernas drivkretsar. Rörelsen utförs och roboten förflyttar sig till den simulerade positionen. Roboten rör sig med en naturlig rörelse, vilket tyder på att den valda inverterad kinematikslösaren fungerar korrekt.

Upprätthållningen av kommunikation mellan gränssnittet och roboten är något instabil. Detta beror på att det underliggande Linux-systemet är beläget på en virtuell maskin

5 Diskussion

När tanken att skapa en fullt fungerande industriell robotarm kom till, var det ursprungligen tänkt att jag själv skulle skapa ett gränssnitt beläget på en Raspberry Pi dator och datorn skulle kommunicera med roboten genom gränssnittet. Vid djupare undersökning i hur en robotarm fungerar som en helhet blev det tydligt att utförandets tidsåtgång hade blivit väldigt stor. Därför började jag söka efter andra metoder för att hantera en robotarm. Jag valde att använda ROS eftersom det verkade vara ett robust verktyg, vilket skulle minska tidsåtgången, gällande problem som inte nödvändigtvis angår robotikområdet.

Vid inledning av arbetet var min kunskap om robotik väldigt begränsad, men genom arbetets gång har jag byggt upp en mycket omfattande kunskap om robotik, ROS och tredimensionell design. Arbetet har även väckt ett starkt intresse för industriella robotar och jag planerar att vidareutveckla prototypen fortsättningsvis.

Målet för arbetet var att designa en prototyp som styrs via seriell kommunikation från ett ROS-system. För att vidareutveckla prototypen till en robot som kan användas för verkliga arbeten så planerar jag att uppgradera drivkretsarna samt byta ut ett par motorer mot kraftfullare varianter. Detta gjordes inte i detta slutarbete eftersom komponenterna är kostsamma och användbarheten var inte ett prioriterat fokus, utan fokuset var på kommunikationen mellan gränssnitt och hårdvara. Vid vidareutveckling av komponenterna kommer även några 3D-printade delar designas om för att uppnå större vinklar. Gripklon i sig var inte ett objekt som krävdes i arbetet men den skapades för att kunna användas vid vidareutvecklingen av roboten.

Samtliga delar printades med materialet PLA, pga. dess lättillgänglighet och låga kostnad. Detta gör materialet optimalt för utveckling av en produkt. Nackdelen med PLA är materialets låga värmebeständighet. Därför kommer prototypens samtliga delar att printas i

ett annat material när utvecklingsfasen är utförd. De optimala materialen för denna tillämpning är ABS och PETG.

Testkörningen fungerade som planerat och meddelandena konverterades till steg som roboten utförde. Roboten testkördes först med 200 steg per rotation vilket gav en relativt hackig men tydlig rörelse. Microsteg har sedan dess implementerats och roboten utför arbeten med jämna rörelser.

6 Referenser

[1] D. Gregor, Artist, *The six degrees of freedom: forward/back, up/down, left/right, yaw, pitch, roll.* [Art]. Wikipedia, 2015.

[2] B. Lang, ”Roadtovr,” 12 Februari 2013. [Online]. Available: https://www.roadtovr.com/introduction-positional-tracking-degrees-freedom-dof/.

[3] ”Mecademic,” [Online]. Available: https://www.mecademic.com/resources/Workspace/Workspace.

[4] S. Cousins, I. Sucan och S. Chitta, ”MoveIt!,” *IEEE ROBOTICS & AUTOMATION MAGAZINE,* Mars 2012.

[5] B. Siber, ”all3dp,” 22 Maj 2018. [Online]. Available: https://all3dp.com/2/infill-3d-printing-what-it-means-and-how-to-use-it/.

[6] A. Koubaa, Robot Operating System (ROS): The Complete Reference (Volume 1), Springer International Publishing, 2016.

[7] ”Moveit.ros,” [Online]. Available: https://moveit.ros.org/documentation/concepts/.

[8] I. Sucan och J. Kay, ”wiki.ros,” 2019. [Online]. Available: http://wiki.ros.org/urdf.

[9] J. Faust, D. Gossow och D. Hershberger, ”wiki.ros,” [Online]. Available: http://wiki.ros.org/rviz.

[10] L. Louis, ”Working principle of arduino and using it as a tool for study and research,” *International Journal of Control, Automation, Communication and Systems (IJCACS),* vol. 1, nr 2, 2016.

[11] J. Lentin, ”Inroduction to ROS - Arduino interface,” *Servo Magazine,* pp. 28-31, 2016.

[12] V. Kumar, ”Introduction to Robot Geometry and Kinematics,” Penn Engineering, Pennsylvania.

[13] H. P. Gavin, ”The Levenberg-Marquardt algorithm for,” Duke University, Durham, 2019.

[14] L. Wang och C. Chen, ”A combined optimization method for solving the inverse kinematics problems of mechanical manipulators,” *IEEE Transactions on Robotics and Automation* , vol. 7, nr 4, pp. 489 - 499, 1991.

[15] D. Chen, Y. Zhang och S. Li, ”Tracking Control of Robot Manipulators with Unknown Models: A Jacobian-Matrix-Adaption Method,” *IEEE Transactions on Industrial Informatics, vol. 14, no. 7,* pp. 3044-3053, Juli 2018.

[16] C. Benson, ”Robotshop,” 2018. [Online]. Available: https://www.robotshop.com/community/tutorials/show/robot-arm-torque-calculator.

[17] C. Benson, ”Robotshop,” 2018. [Online]. Available: https://www.robotshop.com/community/tutorials/show/robot-arm-torque-tutorial.

[18] P. Mitrouchev, ”Kinematic Design and Description of Industrial Robotic Chains,” i *Industrial Robotics: Theory, Modelling and Control*, Ryssland, InTech, 2006.

[19] S. Kucuk och Z. Bingul, ”Robot Kinematics: Forward and Inverse Kinematics,” i *Industrial Robotics: Theory, Modelling and Control*, Ryssland, InTech, 2006, p. 34.

[20] ”Ros-planning,” [Online]. Available: https://ros-planning.github.io/moveit_tutorials/.

[21] S. J. Wright, ”optimization-online,” 2014. [Online]. Available: http://www.optimization-online.org/DB_FILE/2014/12/4679.pdf.