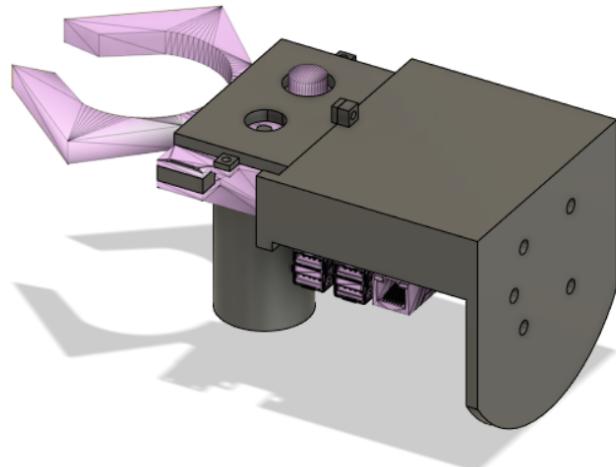


UR5 GRIPPER

Udvikling af en custom gripper
til en UR5 robot



Projektgruppe 6

Alex Ellegaard - ae11e20 August Büchert - aubry20
Maxim Milenkovic - mamil20 Simon Christensen - simch20
Thomas Therkelsen - ththe20 Victoria Jørgensen - vijoe20

Vejleder

Frederik Hagelskjær



Diplomingeniør Robotteknologi

TEK MMMI

Syddansk Universitet

24/05-2021

Abstract

This report presents the process of constructing and programming a gripper for the UR5-series industrial robotic arm, controlled through an XML-RPC-socket utilizing a URCap plugin on the robot. The gripper design is made for the purpose of hoisting a conventional 33 cl beverage can, which has the dimensions 122 mm in height and 65 mm in diameter. The final product is to be constructed with four main segments, firstly a URCap-plugin utilizing XML-RCP protocol to communicate with a Raspberry Pi. This single-board computer is connected to a motor, mounted to a 3D-printed gripper, which will open and close according to the XML-RPC-server method called by the URCap. Determinately, Industry 4.0 communication will run from the single-board computer to an external device, amassing and preserving information about the behaviour of the gripper.

First, the report introduces the different contents of the URCap-StarterPackage and the theoretical fundamentals of URCaps, following a walkthrough of the methods applied in building a URCap. This includes both an installation and program node fitting the criteria of the project. A more concrete view on how the individual Java-methods act together, to construct the interface and computer logics required in building a connection to an external device utilizing XML-RPC-protocol, is discussed. Furthermore, the structure of the C++-programmed XML-RPC-server will be covered, and additionally, how the different server method calls are utilized in another C++ program to instigate physical kineticism of the gripper. Thereafter, the designations and functionalities of the hardware components used will be listed and the process of modeling and printing the gripper, and the thoughts behind this design will be provided.

The proposed solution is presented in detail and discussed in terms of functionality in relation to the chosen scenario, and possible amendments, as well as thoughts on choices made throughout the process of effectuating a solution, will be shared. Lastly, the entire project will culminate in a summarized conclusion on the coalescence of URCaps, MotorControl, Industry 4.0 communication, and the 3D modeling and printing of the physical gripper - the four main segments embodying the objective of this project.

Forord

Heraf fremgår det i prioriteret rækkefølge hvad det enkelte gruppemedlem primært har ydet og bidraget med til projektet.

Simon og Victoria har primært beskæftiget sig med udvikling af URCap program- og installation node, og arbejdet på XML-RPC i forbindelse hermed. Alex og Thomas har beskæftiget sig med udvikling af C++-styring af motoren og programmering af XML-RPC-server på Raspberry Pi og imens har August og Maxim arbejdet med CAD-modellering og konstruering af gripperens hardware-komponenter. Sluteligt har Thomas og Maxim udarbejdet Industry 4.0-kommunikationen, hvorved Thomas har stået for programmering af en c++ socket på RaspberryPi, og Maxim for en python-programmeret server samt GUI. Alle gruppens medlemmer har efterfølgende arbejdet sammen om udviklingen af denne rapport.

Indhold i Zip-filen **Gruppe6.zip**:

URCap - Mappe som indeholder Java-koden der udgør URCap-pluginnet.

Raspberry Pi - Mappe med Gripper mappen som indeholder filerne til projektet som kører på Raspberry Pi'en, Dvs. både XML-RPC- og OPC/UA- server-klasser, MotorControl klassen, inklusivt en main hvor server-klasserne kaldes.

Gripper Demonstration.mp4 - Video-fil indeholdende en demonstation af hele gripper/URCap/-Industry4.0-setuppet

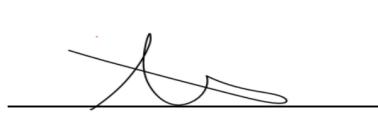
Alex Ellegaard



Simon Bork



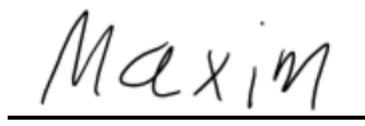
August Büchert



Thomas Therkelsen



Maxim Milenkovic



Victoria Jørgensen



Indholdsfortegnelse

1 Indledning	5
2 Teori	6
2.1 Hardware	6
2.1.1 Raspberry Pi 3 Model B+	6
2.1.2 HG37-300-AB-00	6
2.1.3 L298n - Dual Full-Bridge Driver IC	6
2.2 3D-print	7
2.2.1 Overvejelser før 3D-design	7
2.2.2 3D-modellering	8
2.2.3 3D-designprocessen	8
2.2.4 3D-print	9
2.3 URCap	9
2.3.1 URCap-starterPackage	9
2.3.2 Application Programming Interface (API) [6]	10
2.3.3 Opbygning af URCaps [3]	10
2.3.4 XML-RPC Gripper URCap	12
2.4 Styring	15
2.4.1 Elektronik	15
2.4.2 Software	16
2.4.3 XML-RPC server	18
2.5 Industry 4.0	18
3 Systemtest	19

3.1	Hardware	20
3.1.1	Motor	20
3.1.2	Switch	20
3.2	Software	20
3.2.1	MotorControl	20
3.2.2	XML-RPC	20
3.3	Test af komplet system	21
4	Diskussion	22
4.1	URCaps	22
4.1.1	Industry 4.0	23
4.2	Gripper	23
5	Konklusion	25
6	Appendix	26
Referenceliste		27

1 Indledning

Den fjerde industrielle revolution, som mange i vor tid vælger at betegne det, er under konstant udvikling, og det observeres hvordan flere af industriens, såvel, som hverdagens processer, automatiseres. En af de mange fordele ved automation er naturligvis, at de fejl, som mennesket foretager ved den manuelle proces, elimineres.

Med dette i mente, designes og konstrueres i dag robotter til alverdens forskellige formål og særligt er den 6-ledede robotarm, bl.a. grundet dens universalitet, blevet et uundværligt industrielt værktøj. Dette skyldes ikke blot bevægelsesfriheden, men - som det for de fleste robotarm producenter gør sig gældende - også friheden til at skifte mellem robotarmens tools. Allerede findes der et hav af tool-producenter, som opfinder alt mellem to- og tre-fingrede grippers, vacuum grippers og grippers som udnytter den såkaldte "soft robotics"-teknologi. Alle disse værende typer af grippers, som kan anvendes eksempelvis sammen med robotarmene fra Universal Robots, der med deres URCap-software i høj grad opnår ambitionen om at skabe en universel robotarm.

Netop muligheden for en tilpasset robot-konfiguration gennem URCap danner rammerne for dette projekt, hvor vi har fået til opgave selv at designe og udvikle en gripper til en UR5-robotarm. Denne skal kunne løfte én eller flere genstande efter eget valg. Vi har her valgt at designe en gripper, som har til formål at løfte en dåse, nærmere specifikt en standard 33 cl sodavands-/øldåse. Selve gripperen modelleres i CAD-programmet Fusion 360, og fremstilles herefter fysisk i 3D-printet plastik. Gripperens bevægelser styres af en DC-motor, som er tilsluttet en Raspberry Pi hvorpå der er i C++ er programmeret en motordriver til at kontrollere DC-motorens bevægelse.

Endvidere er der udviklet et URCap-plugin med henblik på at skabe et brugervenligt interface. Denne URCap, bestående af hhv. af en såkaldt program node og installation node, opretter en XML-RPC-forbindelse til en server på Raspberry Pi'en, derigenom kontrolleres aktiveringsten af gripperens forskellige funktioner.

2 Teori

2.1 Hardware

2.1.1 Raspberry Pi 3 Model B+

I dette projekt benyttes en Raspberry Pi 3 Model B +, som den computer der bruges til både hosting af XML-RPC socket, styring af motor, samt Industry 4.0 kommunikation via en TCP/IP-forbindelse. En Raspberry Pi er en enkeltkortscomputer på størrelse med et kreditkort, og den relativt lille størrelse på kortet er den primære grund til at den er perfekt til projekter som disse, hvor man er begrænset på tilgængelig plads.



Figur 1: Raspberry Pi 3 Model B+ [8]

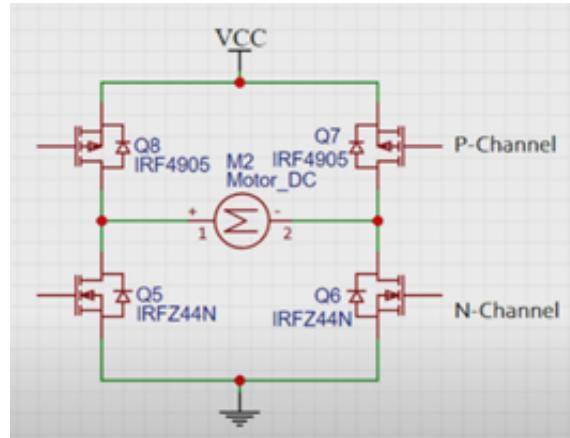
2.1.2 HG37-300-AB-00

Motoren brugt til gripperen i dette projekt er en Copal Electronics model HG37-300-AB-00. Den er en 24V jævnstrømsmotor udstyret med planetgearkasse med 300:1 gearing, for højere drejningsmoment på op til 588 mN·m, end på et konventionelt gearsystem. Motoren ses til venstre på **Figur 3**.

2.1.3 L298n - Dual Full-Bridge Driver IC

Chippen som bruges til at kontrollere motoren er en L298n dual full-bridge motordriver, som kan drive en motor med op til 46V. Et kredsløb som dette er også kendt som en H-bro, og for at styre vores motors retning og hastighed er det nødvendigt at have sådan en H-bro med PWM. Dette betyder for motoren at, den kan køre begge retninger ved at man blot sender strøm igennem til

forskellige transistorer i broen. Et eksempel på et H-bro kredsløb kan ses på **Figur 2**. I denne rapport vil H-bro-kredsløbet blive omtalt til som motordriver.



Figur 2: H-bro kredsløb [7]

L298n chippen kan egentlig drive to motorer på samme tid, men der er kun brug for at drive én i dette projekt. Chippen kan også findes integreret i et modul med indbygget radiator til køling samt nemmere adgang til pins, dette er dog ikke en prioritet i projektet. Chippen ses til højre på **Figur 3**.



Figur 3: HG37-300-AB-00 Motor [2] og L298n Bridge [1]

2.2 3D-print

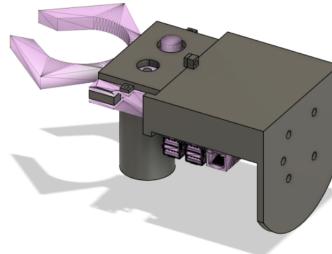
2.2.1 Overvejelser før 3D-design

Det blev hurtigt besluttet at alle delene, der skulle bruges til gripperen, skulle kunne være inde i selve gripperens kabinet. Dette ville være det mest praktiske, da man så kun skulle tilslutte ledninger til gripperen, før den var klar til brug.

Vi lavede en del overvejelser omkring hvilket objekt vi skulle gøre. En ide var blandt andet at gøre en hat, men det blev besluttet at en hat ikke nødvendigvis opfører sig forventeligt, da den kan have underlige former og den ville bøje sig når den bliver grebet. En anden ide var at gøre en flaske, men der stødte vi igen på de andre problemer. For eksempel er der mange forskellige

former for flasker, hvor nogle er smalle og andre er bredere, hvilket betyder der ville være forskel i hvor hårdt gripperen skal gibe for at få ordentligt fat i flasken.

Til sidst blev der besluttet at gibe en dåse, fordi de for det meste altid har samme bredde, selvom de har forskellige rumfang. Dåser har typisk samme diameter, hvilket ville sige at gripperen altid kan gibe med samme styrke, som giver mulighed for at gibe andre dåser med anden højde.



Figur 4: 3D-Model af Gripperen

2.2.2 3D-modellering

Vores gripper er designet i Fusion 360. Den består af to dele, hhv. en ”kasse” og en gripperdel, som kan samles med 3 skruer. Gripperdelen består derudover af to tænder der sidder sammen med tandhjul-lignende riller og derfor bevæger sig sammenhængende.

Vores første gripper havde motoren monteret lige under den ene af disse tænder med sit eget tandhjul. Tanken bag dette var at vi ville være i stand til at geare gripperen efter vilje. Vi gik væk fra denne idé, da det medførte for mange bevægelige dele i vores gripper.

Målet har været at hele indmaden (Raspberry Pi'en, breadboard, motor og sensor osv.) kunne være inde i gripperen og ikke ved siden af. Denne løsning viste sig senere hen ikke at være helt optimal, da vi nok har undervurderet den samlede vægt af vores grippers komponenter. Med dette i mente burde vi nok have printet vores grippers kabinet med en tykkere bund, eller med en højere filling end de 5% vores endelige print endte med at have.

2.2.3 3D-designprocessen

Til at starte med blev fingrene designet, og her var det vigtigt at gripperen havde et design der let kunne gibe en dåse. Derfor er der lavet to halvcirkler, én på hver finger, der til sammen passer rundt om en dåse. Derefter blev tandhjulene designet så motoren kunne monteres på det ene tandhjul, men stadig roterer begge fingre.

Da gripperfingrene og holderen var færdig skulle der designes endnu en kasse som kunne

monteres på UR-robotten. Som nævnt i afsnit 2.2.1 skulle den ydre kasse have plads til alle dele. Den ydre kasse er designet så der er plads til at skrue en Raspberry Pi fast, samt plads til yderligere elektronik. Den færdige 3D-model af gripperen kan ses øvest i afsnittet på **Figur 4**.

2.2.4 3D-print

Grunden til at der bruges 3D-print er, at dette er en hurtig måde at lave prototyper på, da man kun skal bruge en 3D-printer samt plast og en computer. 3D-print er godt egnet til enkelte prototyper og koncepter, grundet at det kan tage lang tid at printe flere kopier. I dette projekt skulle der kun printes fire gange, hvor fingrene blev printet på samme tid. Vores gripper tog 12 timer og 51 minutter at printe i det hele, her er der dog ikke udregnet tiden det tager for printeren at varme op. (Se appendix A: Liste over alle dele samt printetider)

Til gengæld er det forholdsvis billigt at 3D-printe. Til hele gripper blev der brugt 130g plastik hvor en rulle plastik koster ca. 150kr og indeholder 1 kg plastik. Det vil sige at det kun koster 19,5kr at producere hele gripperen.

2.3 URCap

2.3.1 URCap-starterPackage

Til at begynde arbejdet med URCaps, tilbyder UR en Starterpackage, som indeholder alle de nødvendige værktøjer. Starterpackagen i sig selv, er en virtuel maskine, som er baseret på et Ubuntu Linux miljø. Det betyder også der skal bruges en virtualiseringsmaskine til at køre Starterpackagen. Her valgte vi at bruge VirtualBox. Vha. denne Starterpackage er det muligt at arbejde med udvikling og simulering af UR-script, uden at have en fysisk robot foran sig.

Inde i systemet findes følgende komponenter:

- **Eclipse Java IDE**

Her udvikles kode i UR-script

- **URCaps SDK**

Dette er et software-udviklingskit til at lave URCaps.

- **URSim**

En simulator der meget nøje gengiver det rigtige Polyscope-interface på en rigtig robot.

Dette er et virkelig vigtigt værktøj til at teste og evaluere ens kode mens man udvikler.

2.3.2 Application Programming Interface (API) [6]

URCap API'en tilbyder et sæt funktioner til en URCap, installeret på en robot. Kernen i API'en er contributions, som er nye tjenester, der kan tilføjes til PolyScope for at give brugeren en mere brugbar funktionalitet. De to typer contributions, der er relevant for projektet er henholdsvis;

- **Installation node**

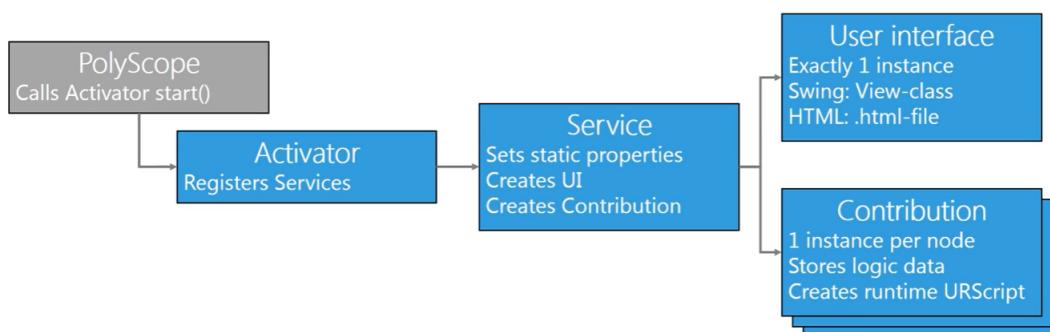
En installation node er en udvidelse af installation-domænet af robotten. Herfra kan URCap'en skabe et User Interface, som tilføjer funktionalitet til at konfigurere omgivelsesindstillerne, som fx portnummer og IP-adresse på en ekstern enhed, der skal oprettes forbindelse til. Koden som afvikles i installation nodes udføres kun en gang per Polyscope program, og det gælder på tværs af forskellige instanser af program nodes.

- **Program node**

En installation node er en udvidelse af program-domænet af robotten. Her kan URCap'en tilføje nye programmeringskoncepter til program-flowet af robotten. Program noden inkluderer et User Interface og tilføjer URScript kode til hele programmet.

API'en der er tilbuddt til disse typer nodes er typisk genkendt som Domæne-API'et, fordi det giver adgang til både domæneoplysninger og logik af robotten. Dog giver ikke alt funktionalitet i denne API mening til installation- og program noden, og derfor er Domæne-API'en delt op i subtyper. Vi beskæftiger os med subtypen Application-API, som giver adgang til API relevant til robotten, som fx inputs/outputs, programvariable eller koordinatsystemer. Herunder findes så installation- og program-API'en, som begge er en udvidelse af Application-API'en. Derfor har de begge adgang til alle features i Application-API'en, og med en tilføjelse af specielle API-features relevant for den individuelle node. Her kunne et eksempel være installation nodens benyttelse af Tool Center Points, eller program nodens bygning af forskellige program node templates.

2.3.3 Opbygning af URCaps [3]



Figur 5: XML-RPC protokollen

En URCap består typisk af fire klasser; Activator, Service, View og Contribution. På “**Figur 5**” ses funktionaliteten af disse fire klasser. Først startes PolyScope, som kalder activatorStart(). Activatoren registrerer her alle de forskellige services, som URCap’en leverer til Polyscope. Service-klassens funktion her er at skabe en instans af både View- og Contribution-klassen, hver gang der er brug for det. Her er det vigtigt at nævne at for hver URCap node oprettes kun én instans af View-klassen, hvorimod der kan være flere instanser af program node i et enkelt program. Når dette er tilfældet er der flere instanser af Contribution-klassen, som indeholder alt logikken og indstillinger af den individuelle program node. Derfor er Contribution-Provider-objektet stillet til rådighed, som et objekt i buildUi-metoden. Den tillader View-klassen at genkende den valgte program node i programtræet, ved at kalde provider.get.

Service

Målet med Service-klassen er at generere noget statisk konfigurationsdata om URCap program node. Siden implementeringen af Service-interfacet kræver en reference til både View- og Contributionklassen, er det anbefalet at skrive de to klasser først. Service-interfacet kræver en definition af en klasse ID, samt en titel på klassen. Eftersom Service-klassen refererer til både View- og Contribution-klassen, skal den returnere en instans af View-klassen når createView er kaldt, og en instans af Contribution-klassen når createNode er kaldt.

Activator

Når Service-klassen er færdigudviklet kan den blive registreret i Activator-klassen. Activator-klassens funktion er at registrere, alle tjenester som URCap’en skal tilbyde til Polyscope. Når Polyscope starter op, bliver activatorStart-metoden kaldt, hvorefter de relevante tjenester vil blive visuelle i interfacet.

View

Denne klasse repræsenterer UI’et af klassen i en URCap. I klassen findes en metode, buildUi(), som er den eneste overridede metode. Her bliver skabt et JPanel-objekt som et af argumenterne. JPanel kan ses som et blankt stykke papir, som URCap’en kan skrive på, og dette stykke papir er symbol for det blanke UI, hvor man igennem UR-script tilføjer elementer. Interfacet er baseret på Java Swing, som er et standard Java UI toolkit.

Contribution

I Contribution-klassen forbindes interface med logik, og her udvikles det egentlige program, som kører når der oprettes en eller flere instanser af den specifikke URCap program node, når programmet afvikles i Polyscope. Klassen består som minimum af fem forskellige autogenererede metoder, heriblandt getTitle() med returværdien String, som opdaterer teksten i programtræet i Polyscope for den enkelte program node. Metoden isDefined() med returværdien boolean, afgører hvorvidt betingelserne for den specifikke program node er opfyldt. De to metoder openView() og closeView(), tager hensyn til, at der kun oprettes én instans af view-klassen, på tværs af eventuelt flertallige program nodes. Når der skiftes mellem de individuelle program nodes i

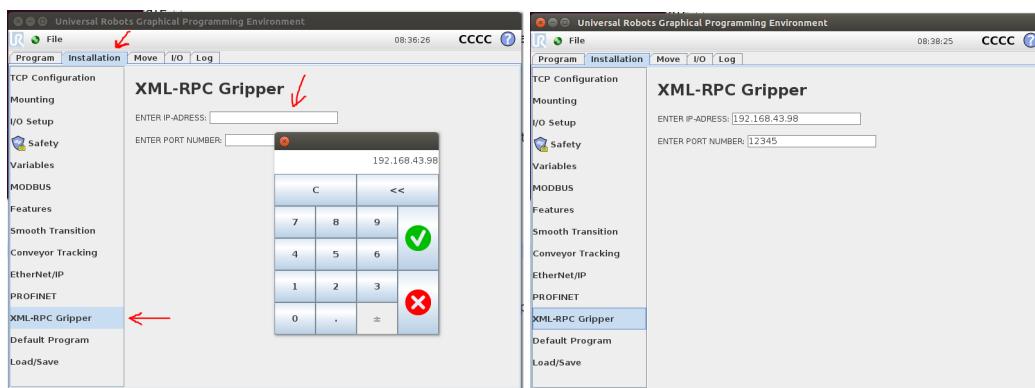
programtræet, sørger de to metoder for at interfacet opdateres i takt med dette. Klassens primære formål bygger på Script Generation, som foregår i generateScript-metoden. I generateScript() befinner den egentlige URScript-kode sig og det er den metode som kaldes når brugeren kører UR-programmet.

2.3.4 XML-RPC Gripper URCap

Som det implicit er beskrevet i navnet på URCap'en, så har denne til formål at oprette en forbindelse, vha. XML-RPC protokol, til en anden enhed, som i dette tilfælde er en grippercontroller i form af en Raspberry Pi. Mere konkret er der udviklet et plugin, som via et overskueligt interface, kan sende et signal for "Release" og "Grip" når det ønskes at den to-fingrede gripper hhv. åbnes og lukkes. URCap'en består både af installation- og program node, og følgende to afsnit beskriver anvendelse samt opbygning af dette UR-Plugin.

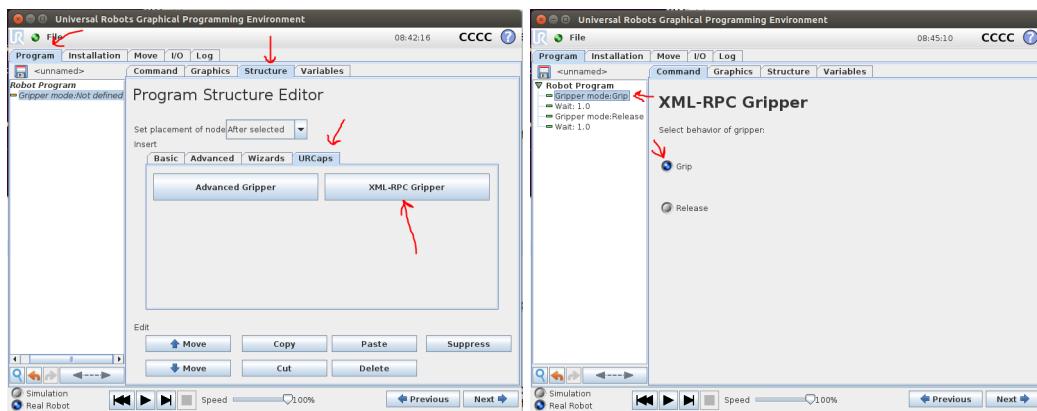
Anvendelse og brug af URCap

På billederne herunder fremgår opsætning af URCap'en i installation node, samt anvendelse af program nodens funktioner, i Polyscope.



Figur 6: XML-RPC Gripper installation node

Når XML-RPC Gripper pluginet først er uploadet til Polyscope er den, grundet det brugervenlige interface, meget simpel at anvende. Som det fremgår af **Figur 6** skiftes der til "installation"- vinduet i øverste menubar i Polyscope, for at indstille IP-adresse og portnummer til den ønskede XML-RPC server. Herinde vælges så XML-RPC Gripper i menuen til venstre, og interfacet bestående af to tekstfelter til hhv. IP-adresse og portnummer vises. Et numerisk tastatur popper op i et separat vindue når der trykkes på tekstfelterne og her indtastes de ønskede værdier, som gemmes i tekstfelterne og registreres i programmet ved tryk på grønt flueben. Således opsættes altså IP og port til XML-RPC forbindelsen.



Figur 7: XML-RPC Gripper program node

Herefter kan XML-RPC gripperens program node anvendes ved at skifte til program-vinduet i menubaren øverst i Polyscope, som det fremgår af **Figur 7**. Under ”Structures” og ”URCaps” vælges så igen XML-RPC Gripper og en instans af program noden vil derefter komme til syne i programtræet til venstre med beskrivelsen ”Gripper mode: Not defined”. Ved dobbeltklik på denne program node vises interfacet, bestående af 2 knapper med titlerne ”Grip” og ”Release”. Disse bestemmer naturligvis hvilken metode der kaldes i XML-RPC Serveren og ved klik på en af knapperne opdateres beskrivelsen i programtræet til venstre. Køres programmet, der er sammensat i programtræet på billedet til højre på **Figur 7**, kaldes først en metode i XML-RPC serveren, som giver anledning til at der gribes med gripperen, og herefter følger et wait på et sekund. Dernæst kaldes en metode i serveren, som giver anledning til at der slippes med gripperen, og dette efterfølges endnu engang af et wait. Herefter gentages programmet cyklisk.

Kodens opbygning [5] [4]

De to tekstfelter i installation noden udgøres af to JTextFields. Disse er oprettet som separate private members af view-klassen, og bruges i metoderne createIPInput() og createPortInput(), hvori de tilføjes MouseListeners, som registrerer hvorvidt der trykkes på det aktuelle JTextField. Når der trykkes udløses koden i en override-metode kaldt MousePressed(), som kalder en metode i contribution der opretter et KeyboardInputFactory. Det er dette objekt som opretter det virtuelle numeriske tastatur. Herefter kalder MousePressed metoden .show() på KeyboardFactory-objektet, som endvidere kalder en metode i contribution, der opdaterer datamodellen og gemmer det indtastede IP/portnummer i en lokal membervariabel.

```

95  @Override
96  public void generateScript(ScriptWriter writer) {
97      System.out.println("generateScript from installation called");
98      writer.assign(XMLRPC_VARIABLE, "rpc factory(\"xmlrpc\", \"http://" + getIPAdress() + ":"
99          + getPortNr() + "/RPC2\")");
100
101 }
102
103 public String getXMLRPCVariable(){
104     return XMLRPC_VARIABLE;
105 }
```

Figur 8: Udklip af installation node contribrution

Værdien af denne lokale membervariabel returneres så i separate get-metoder og benyttes til at oprette et såkaldt RPC-factory i generateScript() i contribution (Figur 8). Det er denne kodelinje der etablerer selve XML-RPC forbindelsen mellem URCap og serveren. Som det fremgår af koden i Figur 8, oprettes dette RPC-factory i en lokal variabel "XMLRPC_VARIABLE", som med en public get-metode bliver mulig at anvende i program nodens contribution-klasse.

I XML-RPC program node udgøres interfacet af to JRadioButtons. Disse to members oprettes ligeledes i separate Box-metoder ved navn createGripButton og createReleaseButton, som til sidst tilføjes til interfacet i BuildUI-metoden. På de fornævnte JRadioButtons er der tilføjet ActionListeners, som registrerer hvorvidt der trykkes på dem. Når dette sker benyttes createGripButton- og createReleaseButton-metodernes provider-parameter til at kalde en metode i contribution-klassen. I contribution-klassen ændres så en member String kaldt STATE, i datamodellen, til at være 1 eller 2 afhængig af om der gribes eller gives slip. I contribution er der endvidere oprettet en metode, getSTATE(), der returnerer state-værdien, og som danner grundlaget for resten af koden i klassen. I getTitle() indgår getSTATE() i et switch statement, som definerer hvad står skrevet i den specifikke program node instans, i programtræet i Polyscope. Derudover bruges getSTATE() i isDefined() til at vurdere hvorvidt alle nødvendige informationer er udfyldt. Her opfyldes det blot ved at der trykkes på en knap.

```

98  @Override
99  public void generateScript(ScriptWriter writer) {
100    System.out.println("generateScript from program called");
101
102    writer.ifCondition(getSTATE() + " == 1");
103    writer.appendLine(getInstallation().getXMLRPCVariable() + ".secure grip()");
104    writer.elseCondition();
105    writer.appendLine(getInstallation().getXMLRPCVariable() + ".release grip()");
106    writer.end();
107
108  }
109
110  private RPCXMLGripperInstallationNodeContribution getInstallation(){
111    return apiProvider.getProgramAPI().getInstallationNode(RPCXMLGripperInstallationNodeContribution.class);
112  }
113
114

```

Figur 9: Udklip af program node contribution

Til slut benyttes værdien af getSTATE() i generateScript() i de indbyggede if- og else conditions på writer-objektet. Afhængig af getSTATE-værdien kaldes .secureGrip() eller .releaseGrip() i XML-RPC serveren og metoderne kaldes på XMLRPC_VARIABLE (RPC factory objektet), som hentes fra installation node contribution-klassen med diverse get-metoder som vist i koden på Figur 9.

I afsnittet om URCaps forklares opbygningen af en URCap og hvordan de forskellige klasser kalder hinanden. Der uddybes yderligere de specielle API-features, der fører med til installation- og program node-API'en. Ydermere bliver der gået i dybden i forhold til de fire nødvendige klasser i en URCap, hvor der lægges mest vægt på View og Contribution, hvor hovedparten af funktionaliteten af URCap'en foregår. Til sidst beskrives vores løsning på URCap'en, hvor installationnoden opretter en XML-RPC forbindelse, som opretholdes gennem alle program nodes. Bagefter har vi i program noden udviklet et UI igennem View-klassen, bestående af to JRadioButton-

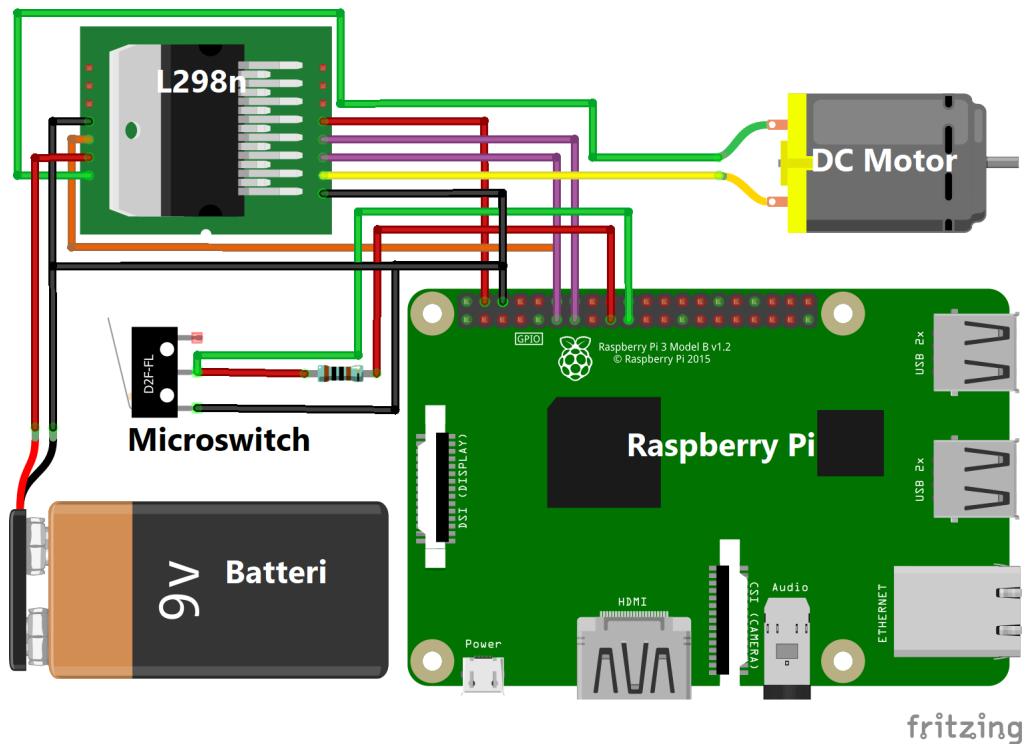
tons koblet til to box-metoder - Grip og Release. Når en af disse knapper trykkes kaldes en metode i contribution-klassen, som ændrer en membervariabel i datamodellen. Når denne membervariabel tilegnes en bestemt værdi, eksekveres et stykke kode i generateScript-metoden, som kalder en metode i XML-RPC serveren.

2.4 Styring

Styring af gripperen sker vha. en Raspberry Pi der med C++ kode kontrollerer en DC-motor med en motordriver gennem dens GPIO-pins.

2.4.1 Elektronik

Her ses et ledningsdiagram over motorstyringen. Pi'en er direkte forbundet til en L298n motordriver og en microswitch, hvor motordriveren er forbundet til en DC-motor. Til at drive motoren, bruges et 9V batteri.



Figur 10: Ledningsdiagram

Idéen er at Raspberry Pi'en får et signal gennem en XML-RPC forbindelse fra UR Simulatoren der kører et UR Caps plugin i en Ubuntu VM. Signalet bliver læst af et stykke C++ kode på Pi'en, som så sender de passende signaler gennem motordriveren, ud til motoren. Derudover sidder der en microswitch på gripperen, som sørger for at gripperens fingre ikke åbner sig så meget op, at den ødelægger sig selv.

2.4.2 Software

Det C++ program vi har kørende på vores Raspberry Pi, indeholder de to klasser XMLRPC og MotorControl. Den førstnævnte klasse er vores kommunikationsprotokol, og har til ansvar at kommunikere med robotarmens URCaps. Den sidstnævnte klasse har til ansvar at styre den DC-motor, som sørger for at gripperens fingre kan åbne og lukke.

Positioneringen af vores gripper bliver styret ud fra den microswitch vi har monteret på vores gripper. Microswitchen bliver aktiveret når gripperen står i sin yderste position, dvs. når den er helt åben. Fordi vi kender gripperens yderste position, og fordi vi ved at den er den samme hver gang, kan vi styre gripperens greb udelukkende ved hjælp af timings. Måden dette sker på er at vi kører motoren med en bestemt hastighed i et bestemt antal milisekunder, som svarer til den tid det tager for gripperen at lukke sig ind til det objekt den skal løfte. Under antagelsen af, at motoren kører med den samme hastighed hver gang, burde dette resultere i, at gripperen lukker sig med præcis den samme mængde for hvert greb. Det drejer sig derfor blot om, at kende det antal sekunder der svarer til at løfte et bestemt objekt. I vores tilfælde gælder det om, at kende den tid det tager for gripperen at bevæge sine fingre ind omkring en dåse. Når vi kender denne tid, burde vi derfor kunne køre vores grip-metode til dette tal, og dermed lave et perfekt greb om dåsen hver gang. På **Listing 1**, ses det sted i koden hvor gripperen finder sin åbne position. Metoden hvor det sker hedder calibrate(), og det ses hvordan polariteten på motoren først vendes og motoren dermed kører baglæns med 65% PWM, da en lavere motorhastighed gerne skulle resultere i en mere præcis kalibrering, end hvis motoren havde kørt på 100% PWM.

```

1 void MotorControl::calibrate()-
2     // Reverse polarity to run motor backwards
3     digitalWrite(motorPin1, HIGH);
4     digitalWrite(motorPin2, LOW);
5     // Run motor backwards until sensor is pressed
6     while(digitalRead(sensor != HIGH)) -
7         softPwmWrite(motorEnable, 65);
8
9     softPwmWrite(motorEnable, 0);
10    // Forwards polarity to prepare for gripping an object
11    digitalWrite(motorPin1, LOW);
12    digitalWrite(motorPin2, HIGH);
13

```

Listing 1: calibrate() metode i MotorControl klassen

Når calibrate-metoden har kørt færdig, og gripperen rent fysisk er på plads i sin åbne position, er vi klar til at gripe om et objekt. Dette sker i grip-metoden, som kan ses på **Listing 2**. Grip-metoden starter DC-motoren med 100% PWM, og venter det stykke tid, det bestemte grip tager, hvorefter motoren stoppes igen.

```

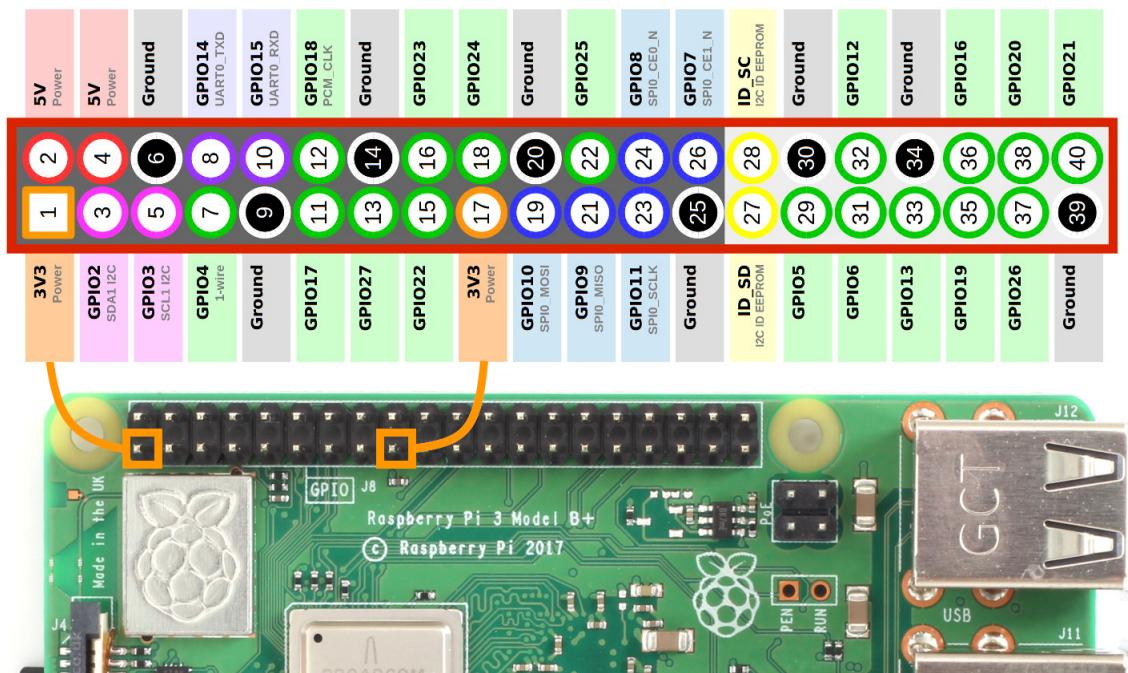
1 void MotorControl::grip(unsigned int time)-
2     // Calibrate gripper
3     calibrate();
4     // Run motor for x seconds, to grip given object
5     softPwmWrite(motorEnable, 100);
6     delay(time);
7     softPwmWrite(motorEnable, 0);
8

```

Listing 2: grip() metode i MotorControl klassen

Stedet hvor alt dette sker, er i vores MotorControl klasse under grip-metoden, som kan ses på **Listing 2**. Deri bliver vores motor styret af softPwmWrite-metoden, som fås via inklusionen af softPwm header-filen. SoftPwm gør det muligt at lave en software kontrolleret PWM gennem nogle specifikke GPIO-pins på Pi'en, så motorens hastighed kan varieres. I vores kode kører vi som udgangspunkt altid vores PWM på enten 100% når motoren skal køre, og 0% når motoren skal holde stille. Det er kun under calibrate-metoden at vi sænker hastigheden på vores PWM til 65%, for at opnå en højere præcision når vi stiller gripperens fingre ud i sine yderpunkter.

I dette projekt benytter vi os af WiringPi, som er et library der styrer boardets GPIO i overensstemmelse med det pin-out diagram, som kan ses på **Figur 11**. GPIO er en forkortelse for Generel Purpose Input/Output, hvor der her specifikt er tale om de 40 headers, der befinder sig på toppen af boardet. Det er igennem disse headers, at alt kommunikationen fra vores Raspberry Pi til vores motor og microswitch sker, og det er også herigennem vores motor får strøm fra det 9V batteri vi har valgt som dens strømkilde.

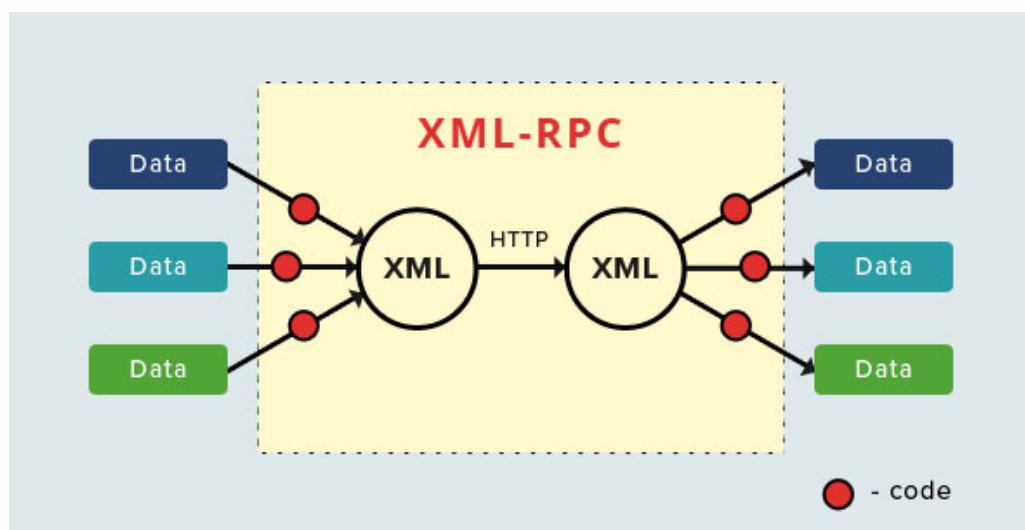


Figur 11: WiringPi pin-out diagram

2.4.3 XML-RPC server

XML-RPC er en RPC protokol, som benytter XML til at indkode data, og HTTP til at transportere de kodede instrukser mellem enheder. I vores projekt benytter vi en C/C++ implementering af XML-RPC konceptet, kaldet XMLRPC-C, som er et librabry vi har installeret på vores Raspberry Pi, der virker med den C++ kode vores motorControl klasse er skrevet i. Måden hvorpå XML-RPC protokollen indkoder og sender data kan ses på **Figur 12**.

Vores implementation af XML-RPC har resulteret i en XML-RPC klasse, som håndterer kommunikationen med URcap sideløbende med motorControl-klassen. Fordelen ved dette er at vi kan starte XML-RPC-serveren én gang fra vores main-klasse, og derefter kan vi lade den stå og vente på kommunikation fra URCap, som derefter kan medføre et funktionskald i vores motorControl-klasse.



Figur 12: XML-RPC protokollen

2.5 Industry 4.0

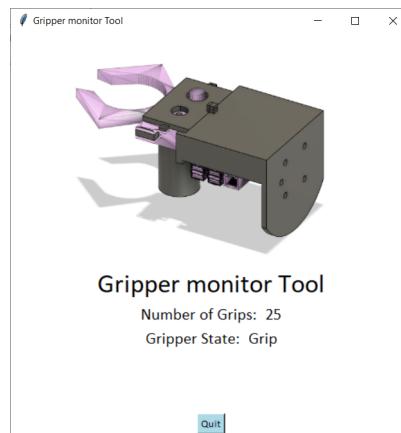
Industry 4.0 er integrationen mellem den digitale verden og den fysiske produktion. Det er visionen om at digitalisere de fysiske processer i en produktion, til formålet at forbedre produktionen eller som salg af serviceydelser.

I forhold til vores gripper, skulle vi tænke det som et produkt vi kan videresælge, og implementere i en produktion. Herved startede en masse idéer, som kunne hjælpe til at gøre vores gripper til et uafhængigt og ubemandet produkt. Det kunne for eksempel være med en TCP/IP forbindelse til en ekstern enhed, der gør det muligt at læse forskellige værdier fra vores kode. Her har vi valgt at sende "numberOfGrips", som tæller mængden af greb gripperen har udført. Denne information kunne være interessant ift. at følge med i gripperens produktivitet. Her kunne man også vælge sammenligne "numberOfGrips" fra forskellige dage og følge produktiviteten. Man

kunne også give et groft tidsestimat på den gennemsnitlige grip-release sekvens, ved at dividere et bestemt tidsinterval med antal greb. Dette vil dog ikke tage hensyn til nogen former for downtime. Hvis man fx tog udgangspunkt i en time med et antal på 150 greb, ville regnestykket se således ud:

$$3600\text{sek}/150\text{greb} = 24\text{sek}$$

I vores overvejelser omkring gripperens implementation i en reel virksomhed, besluttede vi også det ville være vigtigt at kunne læse "currentState" på gripperen. Med denne data kunne klientsiden følge med i, om gripperens state er i "Grip" eller "Release". Derved kunne man fx antage fejlmeldinger, hvis gripperens state sidder fast.



Figur 13: Industry 4.0 UI, set fra klientsiden

Som vist på **Figur 13**, ses det UI som bruges til overvågning af gripperen fra en ekstern enhed. Det er opbygget på den måde, at når der er oprettet forbindelse, sender klienten en vilkårlig string til serveren, for at vide om forbindelsen er oprettet. Herfra kan serveren sende "numberOfGrips" og "currentState" til klienten i en kommasepareret string, som opdaterer UI'et med de nye værdier. Klienten vil hele tiden prøve at skabe forbindelse, men det er kun når der sker en ændring i dataen, at serveren reagerer og sender en ny string tilbage.

3 Systemtest

Følgende afsnit beskriver isolerede testscenarier af de enkelte dele i projektet. Der vil blive gennemgået test af vores DC-motor og den microswitch vi bruger til at kontrollere den med. Der vil også blive beskrevet test af XML-RPC netværksprotokollen. Til slut rundes der af med en test af det komplette system.

3.1 Hardware

3.1.1 Motor

Test af DC-motoren kom som et ret naturligt skridt i projektforløbet, da det er en af de centrale dele i projektet, at kunne styre motoren. Der blev eksperimenteret med hvor meget strøm motoren skulle bruge, og det endte med at blive et 9V batteri - på trods af, at motoren kan tage op til 24V. Der er dog ikke brug for den høje hastighed, derfor bruges et 9V batteri som forsyning. Motoren blev tilsluttet L298n chippen, der blev valgt som værende projektets motordriver til at styre motoren på gripperen. Da motoren blev tilsluttet, var det forholdsvis enkelt at få den til at køre, både i den ene og den anden retning og kontrol af hastigheden foregik nemt gennem softPwm klassen.

3.1.2 Switch

Da der ikke benyttes en steppermotor til gripperen men en DC-motor, er det svært at holde styr på præcist hvor langt motoren og gripperen har bevæget sig - så der er brugt en microswitch med en forholdsvis lang arm, som et endstop. Under motortesting har denne fungereret som en omvendt dødmandsnap - ment på den måde, at motoren kører så længe knappen ikke trykkes på.

3.2 Software

3.2.1 MotorControl

Opsætning af MotorControl klassen blev gjort ved først at få motoren til at køre vha. L298n chippen samt WiringPi- og SoftPwm-klasserne. Derefter blev det gjort objektorienteret ved at skrive forskellige metoder ind i en klasse. Heriblandt metoder som kalibrerer, lukker og åbner gripperen, samt griber diverse objekter.

3.2.2 XML-RPC

Opsætning af XML-RPC blev udført ved først at testkøre noget eksempel kode fra XMLRPC-C hjemmesiden, ind i Raspberry Pi's C++ program. Det er nogle basale klasser, som blot udskriver tekst til konsollen når en metode bliver kaldt, og er dermed let at debugge. Da kommunikationen var sat op korrekt, blev der gjort så metoderne ud over at skrive i konsollen, også kaldte metoder fra MotorControl klassen, som styrer gripperens DC-Motor.

3.3 Test af komplet system

Formålet med vores projekt var at udvikle en gripper til en UR5-robotarm der havde til formål at løfte en tom 33 cl sodavands-/øldåse. Vi formæde dog aldrig at teste vores gripper monteret på en UR robotarm. I vores systemtest kører vi i stedet vores URCap på en simulator, og bevæger gripperen manuelt. Vi tester vores Industry 4.0 funktioner ved at have vores klient kørende på en separat computer, som er parat til at modtage de funktioner der sendes fra den TCP/IP server vi har kørende på vores Raspberry Pi inde i gripperen.

På **Figur 14** ses den færdige version af vores gripper, dens komponenter og den øldåse vi kørte vores komplette test af systemet på. I testen havde vi placeret øldåsen på bordet, og ville med gripperen løfte øldåsen og sætte den på bordet igen. Der kan i Gruppe6.zip filen findes en video kaldt "Gripper Demonstration", som viser vores test. I videoen vises det i Polyscope interfacet, at gripperen kører en grip-release sekvens, og der herefter bliver løftet en øldåse. Imens programmet kører i loop, har vi sat vores TCP/IP forbindelse op, hvor man kan se klienten modtage data fra serveren, dog med en smule forsinkelse.



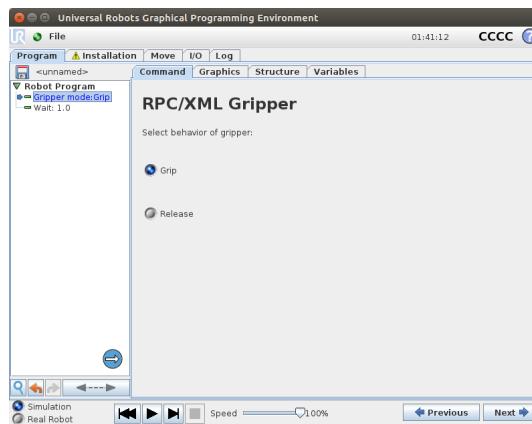
Figur 14: Billede af den færdige gripper sammen med en 33 cl øldåse

4 Diskussion

Inden man går i gang med et projekt som dette, er der mange ting at tage stilling til, og efterhånden som et projekt skrider frem får man ofte nye idéer til bedre løsninger. I dette afsnit vil vi kigge på nogle af disse beslutninger der er blevet taget i løbet processen og diskutere ting der kunne forbedres samt de forbedringer vi lavede undervejs.

4.1 URCaps

I begyndelsen af udviklingen af vores URCap, havde vi integreret J-Buttons med tekst i vores URCap UI. Her valgte vi dog i forbindelse med systemtesten at ændre dette, da det ikke var intuitivt nok hvilken mode der var valgt. Man kunne blive i tvivl om, hvorvidt bad gripper om at gibe eller release, fordi knapperne blev ikke markeret. Man kunne selvfølgelig klikke på dem, men når man gjorde, var der i interfacet ingen indikation på hvad man havde valgt. Her kunne man dog have kigget på robotten om den åbnede eller lukket sig, men efter validering var denne metode ikke brugervenlig eller sikker. Hvis man tænker det ud i virkeligheden med større robotter, som kan klemme med en større kraft om noget værdifuldt, så ville løsningen kunne skabe store konsekvenser, hvis håndteret forkert. Der ville nok i sådanne situationer være andre sikkerhedsforanstaltninger, men vi ville gerne udvikle noget URCap-software, som kunne implementeres uden de store rewrites.



Figur 15: Endegyldige URCap UI, JRadioButtons

Som vist på “**Figur 15**” er vores valg af JRadioButtons mere brugervenlig, da den tydeligt viser med en blå markering hvilken funktion der er valgt.

URCap’en til dette projekt består både af en program- og installation node. Denne beslutning er taget med to formål i mente, nemlig at hardcoding af IP-adresse og port-nummer undlades og erstattes af et brugervenligt interface og derudover undgås det at RPC factory oprettes flere gange end nødvendigt. Jævnfør afsnit 2.5 ekseveres koden i installation node kun en gang, og gælder

på tværs af program nodes, så netop af denne grund har vi valgt at implementere en installation node, for at optimere hele processen med oprettelsen af UR-scriptets RPC-factory. Alternativt kunne selve RPC-factory'et oprettes i generateScript() i program noden, hvilket ville medbære at selve forbindelsen etableres hver gang der grippes eller releases, og dermed for hver program node der er i programmet når det eksekveres i Polyscope, hvilket i grunden er unødvendigt. Det eneste der eventuelt kunne betragtes som en ulempe ved at opsætte IP- og port-nummer i en installation node, i stedet for direkte i program node contribution, er at installation noden naturligvis skal konfigureres hver gang der oprettes et program, hvori XML-RPC gripper program noden indgår.

4.1.1 Industry 4.0

I forhold til Industry 4.0 besluttede vi at modtage to slags data fra klientsiden hhv. "numberOfGrips" og "currentState", fordi det umiddelbart virkede som det mest indlysende data ift. ekstern overvågning af gripperen. Der kunne dog også arbejdes videre på flere forskellige slags data, for at skabe en mere grundig ekstern overvågning. Der kunne for eksempel være en "durationOfGrip"-metode, i form af en videreudvikling af "numberOfGrips". Her vil metoden kunne udregne hvor lang tid hvert grip ville tage, så man kunne lave et mere præcist tidsestimat på grip-release sekvensen. Dette ville fremme optimeringen af gripperen, fordi man kunne sammenligne hver individuel grip-tid, hvor man i vores eksempel "numberOfGrips" kun ville kunne udregne et groft estimat på gennemsnitlig grip-tid, fordi der ikke er taget hensyn til downtime. En anden smart metode at implementere ville være "numberOfFailedGrips", som er en metode der kan beregne antallet af greb, der er fejlet. Eftersom der kan være mange antagelser på hvornår et greb er fejlet, tager vi blot udgangspunkt i et eksempel, hvor man involverer "durationOfGrip", for at bestemme hvornår et greb er fejlet. Hvis man fx siger en normal grip-release sekvens tager 20 sekunder fra start til slut, kunne man sige, hvis "durationOfGrip" går over 25 sekunder, så antages grippet som fejlet. Her vil så eksekveres et stykke kode, som kører release-metoden, for at give slip på en eventuelt ødelagt eller mispositioneret dåse, og vil derefter gå til startpositionen, for at gøre klar til næste grip. Denne metode ville være smart både til optimering af gripperens funktionalitet, men også i form af overvågningen af gripperen, da der sandsynligvis er noget galt, hvis flere grips er fejlet efter hinanden.

4.2 Gripper

Da vi designede styringen og elektronikken til gripperen, tog vi valget om at drive den med et 9V batteri. Dette valg medfører selvfølgelig den åbenlyse ulempe i, at batteriet kræver udskiftning med jævne mellemrum. Udober dette, så medfører det dog også en ulempe i, at et batteri mister spænding over tid. Da vi har valgt at styre vores gripper ved at aktivere DC-motoren til en bestemt tid, kræver det en fast spænding for at vi kan være helt sikre på, at gripperen bevæger

sig den samme længde hver gang. Når vores batteri langsomt mister spænding kan vi derfor ikke være 100% sikre på den afstand gripperen bevæger sig. Vores gripper burde bevæge sig mindre i takt med at batteriet mister spænding. Valget om et 9V batteri er derfor ikke optimalt, og man kunne med fordel have valgt at drive gripperen med en fast strømforsyning hvor spændningen burde være mere konstant.

Vores 3D-modellering er heller ikke optimal, der er flere ting som kunne gøres smartere i senere generationer. Den ydre kasse kan forbedres en del, blandt andet kan den være helt lukket, dette ville give mere plads til komponenterne og det ville sikre at man ikke kommer til at røre elektronikken og dermed beskadige gripperen. Selve basen er ustabil og ikke lavet med nok filling, den kunne også laves i et helt andet materiale for at styrke det. Der blev i øvrigt ikke gjort nogle overvejelser omkring vægten af Pi'en, motoren, osv., som skulle monteres på gripperen, og det skabte nogle problemer ift. vægtfordelingen. Der er også problemer med det hul som "motor-stangen" sidder i, eftersom at den stang ikke er en fuld cirkel, men nærmere en cirkel minus et lille cirkelafsnit. Selvom vi har designet hullet til at matche med denne, sled den det af og derfor blev vi nødt til at lave en lappeløsning og lime det sammen, så vi blev i stand til at gribte om ting og i det hele taget bevæge fingrene på gripperen. Endnu en forbedring kunne være at lave gripperen mindre, hvilket ville gøre den lettere og mere praktisk, men dette ville betyde at komponenterne inde i gripperen også skulle være mindre.

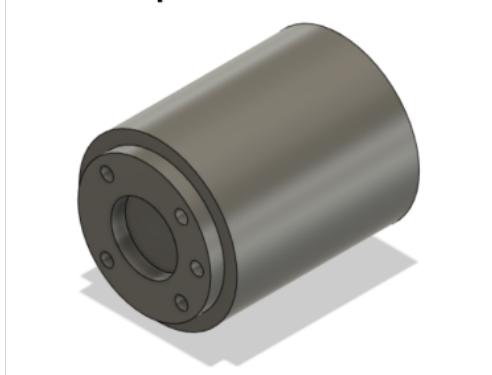
5 Konklusion

Vi har igennem arbejdet med projektet udviklet en velfungerende gripper i Fusion 360, hvor der i gripperens kabinet er gjort plads til dens komponenter inklusiv Raspberry Pi. Gripperen er designet med to fingre, konstrueret til at kunne bære en tom 33 cl sodavands-/øl dåse. Til at styre gripperen er der først færdigudviklet en URCap, bestående af en installation- og program node, gennem Universal Robots "URCap-starterpackage". Installation noden sørger for et UI til indtastning af port-nummer og IP-adresse, som bruges til oprettelse af XML-RPC-forbindelsen. Gennem program noden er der programmeret et UI med en knap til hhv. "grip" og "release", som kalder en metode, der laver et XML-RPC kald til en C++ server, som vi har kørende på vores Raspberry Pi. Når denne data modtages eksekverer C++ koden den relevante metode fra motorControl-klassen, alt efter om der fra URCap'ens side bliver kaldt efter et "grip" eller "release", og her vil gripperen lave den fysiske handling den bliver bedt om. Vi har testet grip-release sekvensen og kunne med succes samle en dåse op og placere den et andet sted. Yderligere er der skabt en TCP/IP forbindelse med hensigt på Industry 4.0. Her bliver sendt en kommasepareret string fra serveren, som fortæller antallet af grips og gripperens nuværende state. Her kan klientsiden gennem et UI overvåge disse data fra en ekstern enhed. Med udgangspunkt i problemformuleringen, kan det til sidst konkluderes, at projektet i stor grad er løst efter hensigten, hvor en af de eneste mangler må siges at være en fysisk test af gripperen monteret på en UR5-robotarm.

6 Appendix

A Liste over alle individuelle dele samt printetider

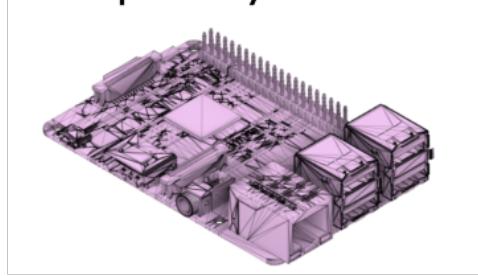
UR Tip



Limit Switch



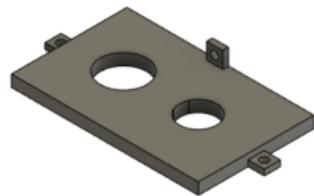
Raspberry Pi



DC Motor

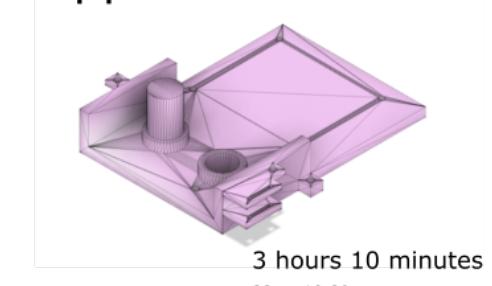


Gripper Holder Cap



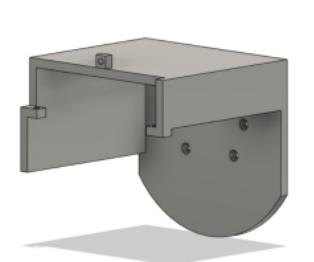
56 minutes
10g - 3.34m

Gripper Holder

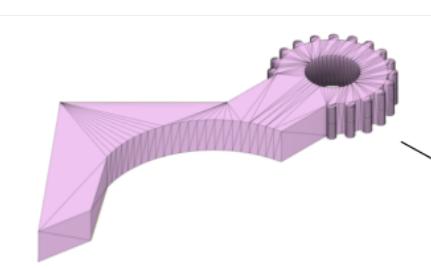


3 hours 10 minutes
32g - 10.89m

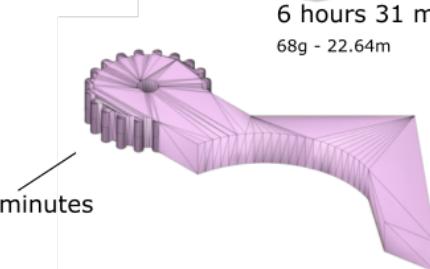
Gripper Case



6 hours 31 minutes
68g - 22.64m



2 hours 14 minutes
20g - 5.54m



Gripper Finger Left

Gripper Finger Right

Referenceliste

- [1] Elfa Distrelec. *L298N - Motordrev, Multiwatt, 4A, ST.* URL: <https://octopart.com/hg37-300-ab-00-copal-74220667>.
- [2] Octopart. *Nidec Copal - HG37-300-AB-00.* URL: <https://www.elfadistrelec.dk/da/motordrev-multiwatt-4a-st-1298n/p/17313229?>
- [3] Universal Robots. *Principle of URCaps integration in PolyScope.* URL: <https://plus.universal-robots.com/urcap-basics/principle-of-urcaps-in-polyscope/?fbclid=IwAR2Q9BZRYbwmzUEXRq2n5Lid0yQvPqBdwx1DpC3%20xNXn5dZmkoiuN-gT2phI>.
- [4] Universal Robots. *SCRIPTMANUAL - CB-SERIES - SW3.15.* URL: <https://www.universal-robots.com/download/manuals-cb-series/script/script-manual-cb-series-sw315/?fbclid=IwAR3D-1YsBRNbRW2RVIinfSJGKrQDWpz-k5BPBpR4AV7Ob89XgCSrPDLJjtE>.
- [5] Universal Robots. *URCap API documentation.* URL: <https://plus.universal-robots.com/urcap-basics/api-reference-docs/?fbclid=IwAR1NYCO%20SOsGVm51vpPUzFh2Rna-TY03z3dcjhkLK28ulo57NHUs6eYI8wY>.
- [6] Universal Robots. *URCap API Overview.* URL: <https://plus.universal-robots.com/urcap-basics/api-overview/?fbclid=IwAR2kBln1LzQudDD72kPhvCfTQqZf68UP3r1mOc3Hu01JQsat1uckFW8e8>.
- [7] Something'sGoneWrong! *DIY DC Motor Speed Control (PWM) // H-Bridge Circuit Tutorial.* URL: <https://youtu.be/yk7Z6NxMQmY>.
- [8] Elektor Store. *Raspberry Pi 3 B+.* URL: <https://www.elektor.com/raspberry-pi-3-b-plus>.