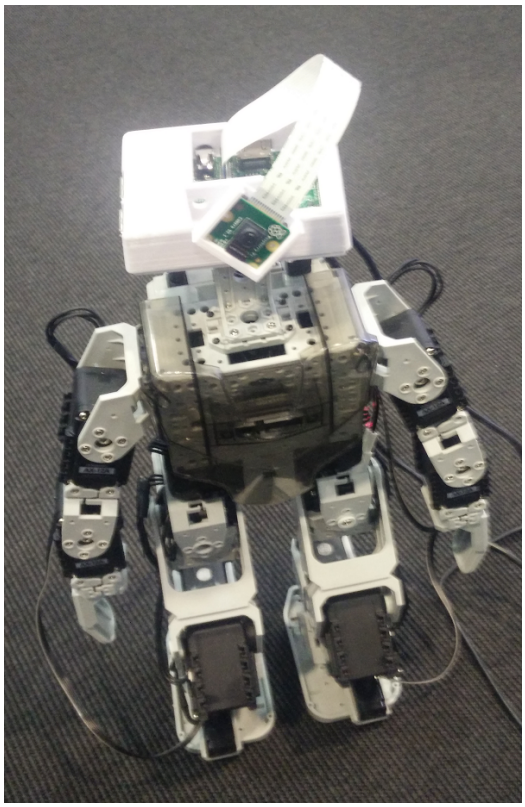
CHALMERS
UNIVERSITY OF TECHNOLOGY



Extern styrning och modularisering av styrning till Bioloid robot

Examensarbete inom Data- och Informationsteknik

Sebastian Lind
Tobias Laving

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2017

Examensarbete

Extern styrning och modularisering av styrning till Bioloid robot

Tobias Laving
Sebastian Lind

CHALMERS
UNIVERSITY OF TECHNOLOGY

Institutionen för Data- och Informationsteknik
Högskoleingenjör inom Datateknik
Chalmers Tekniska Högskola
Göteborgs universitet
Göteborg, Sverige 2017

Extern styrning och modularisering av styrning till Bioloid robot
Sebastian Lind
Tobias Laving

Examinator: Peter Ludin

Examensarbete 2017
Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola / Göteborgs Universitet
SE-412 96 Göteborg
Telefon +46 31 772 1000

Omslag: Visar bild på en sammansatt enhet bestående av en Raspberry pi 3 och en Bioloid robot.

Typeset in LaTeX
Printed by [Name of printing company]
Göteborg, Sverige 2017

Tobias Laving
Sebastian Lind
Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola

Sammanfattning

Detta examensarbete är utfört på uppdrag av Cybercom och handlar om att utveckla en enhet som kan hantera ljud och bild. Förvalda objekt ska kunna kännas igen av enheten samt ska en användare kunna ge röstkommandon. Enheten består av två mindre enheter, en Raspberry Pi 3 och en Bioloid robot. Bioloid robot är uppbyggd likt en människa och kan gå på ett horisontellt plan. Enheter ska kommer kommunicera med varandra via en seriell kommunikation genom en USB-kabel.

För att hantera ljud och bild har Raspberry Pi försetts med en kamera och en USB-mikrofon. Utveckling på denna enhet har resulterat i tre olika moduler som är vitala för den kombinerade enhet. En modul ansvara för bildhanteringen och använder OpenCV samt Viola-Jones metoden för att känna igen utvalda objekt. En annan modul hanterar röstkommandon med hjälp av CMU pocketsphinx. Den tredje modulen ansvar för all kommunikation mellan enheterna. Utvecklingen på Raspberry Pi har skett i C++ medan C har använts till Bioloid robot.

För att visa den kombinerade enhetens förmågor har en demonstration utvecklats. I demonstrationen har alla delar av projektet används för att påverka den kombinerade enhetens beteende. Projektet och demonstrationerna har utvecklats på ett sådant sätt att fortsatt utveckling förhoppningsvis uppmuntrats.

Tobias Laving
Sebastian Lind
Departement of Computer Engineering
Chalmers University of Technology

Abstract

This thesis carried out on behalf of Cybercom and is about development of a unit which can process sound and images. Predetermined objects are going to be recognized and a user can give voice commands. The unit is made of two lesser units, a Raspberry pi 3 and a Bioloid Robot. The robot is built as a human body and can walk on an horizontal plane. These two units will be able to communicate through serial communication via a USB cable.

To achieve sound and image processing the Raspberry Pi has been provided with a camera and a USB microphone. The functionalities have been divided in three different modules. One module is responsible for image processing with OpenCV and the use of the Viola-Jones method to recognize selected objects. Another one manage voice commands with help of CMU pocketsphinx. The third module deals with the serial communication. Development on the Raspberry Pi has been done in C++ while the Bioloid robot uses C code.

To show the ability of the combined unit a demonstration was developed. The demo is using all of the projects resulting resources to affect the units actions. The project and demo have been developed in a way that hopefully encourage further development.

Förord

Vi är tacksamma till Cybercom som gav oss möjligthet till detta examensarbete. Vi vill också tacka våra medarbetare som hjälpte oss med tidigare erfarenheter och information, speciellt Abdulla Al Chalati som hjälpte oss oerhört mycket med hur projektet skulle föras vidare men även vår fantastiska handledare Sakib Sistek.

Tobias Laving, Göteborg, 05 2017
Sebastian Lind, Göteborg, 05 2017

Innehåll

1

Introduktion

I detta kapitel beskrivs bakgrunden till examensarbetet samt dess syfte och mål. Projektets krav och avgränsningar kommer även presenteras.

1.1 Bakgrund

Intresset för robotar har funnits sedan antiken, både i form av arbetare och nöje[14]. Den första programmerbara roboten i dagens benämning skapades redan 1956 av George Devol [14]. Robotar används idag främst för att automatisera arbetssysslor men intresset för robotar som efterliknar människan har funnits i hundratals år [14]. Idag kan även hobbyister och mindre företag bidra till utvecklingen. Cybercom är ett sådant företag.

Cybercom är ett innovativt konsultbolag inom IT som vill utforska och utveckla ny teknik inom diverse områden för att visa sin kompetens samt ha tillgång till ett projekt stock att visa framtida kunder. Robotik, speciellt på hobbyist nivå, är ett område som har växt snabbt de senaste åren och Cybercom vill vara del i den utvecklingen. Cybercom har haft tidigare projekt inom området och vill fortsätta utvecklingen.

Detta projektet kommer fortsätta arbetet på en Bioloid robot typ A, vilket är en färdigbyggd robot med två armar och två ben som styrs med försedda motorer. Roboten har även tre sensorer placerade på dess kropp som kan användas för att mäta avstånd till omgivningen. Med roboten kommer grundläggande program som låter den gå, balansera och ändra riktning.

1.1.1 Cybercoms tidigare projekt

Cybercom har tidigare haft två projekt kring en Bioloid robot vilket har tillfört detta projekt med kod och information i form av rapporter och personlig handledning. Detta accelererade inlärningen och gav detta projekt en grund att stå på.

Ett av dessa projekt skapade kod som låter roboten läsa och utföra hela rörelse-

mönster från det inbyggda flashminnet. De hade dock begränsad tid vilket ledde till att de byggde ett projekt som skall vara lätt att ta över. Efter projektet blev det tydlig att en sensor var defekt.

Det andra projekt resulterade i kod som tillåter grundlig seriell kommunikation mellan robot och godtycklig enhet.

1.2 Syfte

Syften med projektet är att utveckla en enhet som består av en Bioloid robot och en Raspberry Pi 3. Enheten skall kunna behandla omvärlden och agera därefter.

Cybercom vill fortsätta utvecklingen av roboten och vill att detta projekt skall skapa en välstrukturerad grund för vidareutveckling.

1.3 Mål

Målet med projektet är att etablera ett samarbete mellan en Bioloid robot och en Raspberry Pi 3. Den externa enheten skall kunna behandla information från omvärlden genom bild- och ljudbehandling. Efter behandlingen skall den externa enheten styra roboten efter programmerat beteende. För att uppnå en bra plattform för vidare utveckling skall arbetet ha ett fokus på modularisering.

1.4 Avgränsningar

- Begränsad utveckling av redan implementerade styr funktioner som gå, ändra riktning och balansera.
- Ingen utveckling av robotens hårdvara.
- Roboten ska endast kunna arbeta i samma plan som den befinner sig.
- Roboten skall inte kunna lyfta och bära objekt.
- Roboten ska inte fungera utan koppling till fast strömkälla.
- Den externa enheten ska inte fungera utan koppling till fast strömkälla.
- Mjukvaran skall inte utvecklas till Windows operativsystem.

2

Teknisk bakgrund

Detta kapitlet behandlar och förklarar mjukvaran samt hårdvaran som behövs för att utföra projektet och hur dessa använts. Dessutom diskuteras teorier och vetenskapliga metoder som har utnyttjats under projektet.

2.1 Hårdvara

2.1.1 Bioloid Premium Robot Kit

Bioloid Premium Robot Kit är ett kit som är utvecklat för hobbyiester, robottävlingar och skolprojekt. Kan användas för både avancerade projekt med erfarna programmerare och nybörjare som vill lära sig programmering. Roboten består av Dynamixel AX-12A ställdon, IR sensor, längdmättnings sensor samt en kontrollenhet vid namn CM-530. Ställdonen kan kopplas om till diverse former med stor frihet. Den vanligaste formen är människoliknande, vilket används i detta projekt. Formen går att se i A.3. [17]

2.1.1.1 Dynamixel AX-12A

Dynamixel AX-12A är ett servoställdon som opererar mellan nio till tolv volt. Servoställdona kan seriekopplas valfritt för att ge roboten maximal frihet i uppbyggnad. Servoställdorna kan avläsa sin hastighet, temperatur, axelposition och sin egna spänning. Kontroll algoritmen som hanterar axelposition kan dessutom ställas in individuellt för varje enskild servoställdon. Servon har en inbyggd mikrokontroll som hanterar all sensor och positionshantering.[21]

2.1.1.2 CM-530

CM-530 är en kontrollenhet som inkluderar en CPU samt ett kommunikationskort som kan hantera kommunikation till och från kontrollenhet seriellt. Seriell kommuni-

kation kommer behandlas mer i detalj senare i detta kapitel 2.5. Enheten har också LED lampor, ingångar för mottagare och är kompatibel med AX-12A servos.[22]

2.1.2 Sensorerna

Det finns tre sensorer på roboten som kan känna av om något är framför dem. Två av sensorerna sitter på vardera fot och är IR-sensorer och den sista på bröstet och är en dynamisk rörelse sensor.

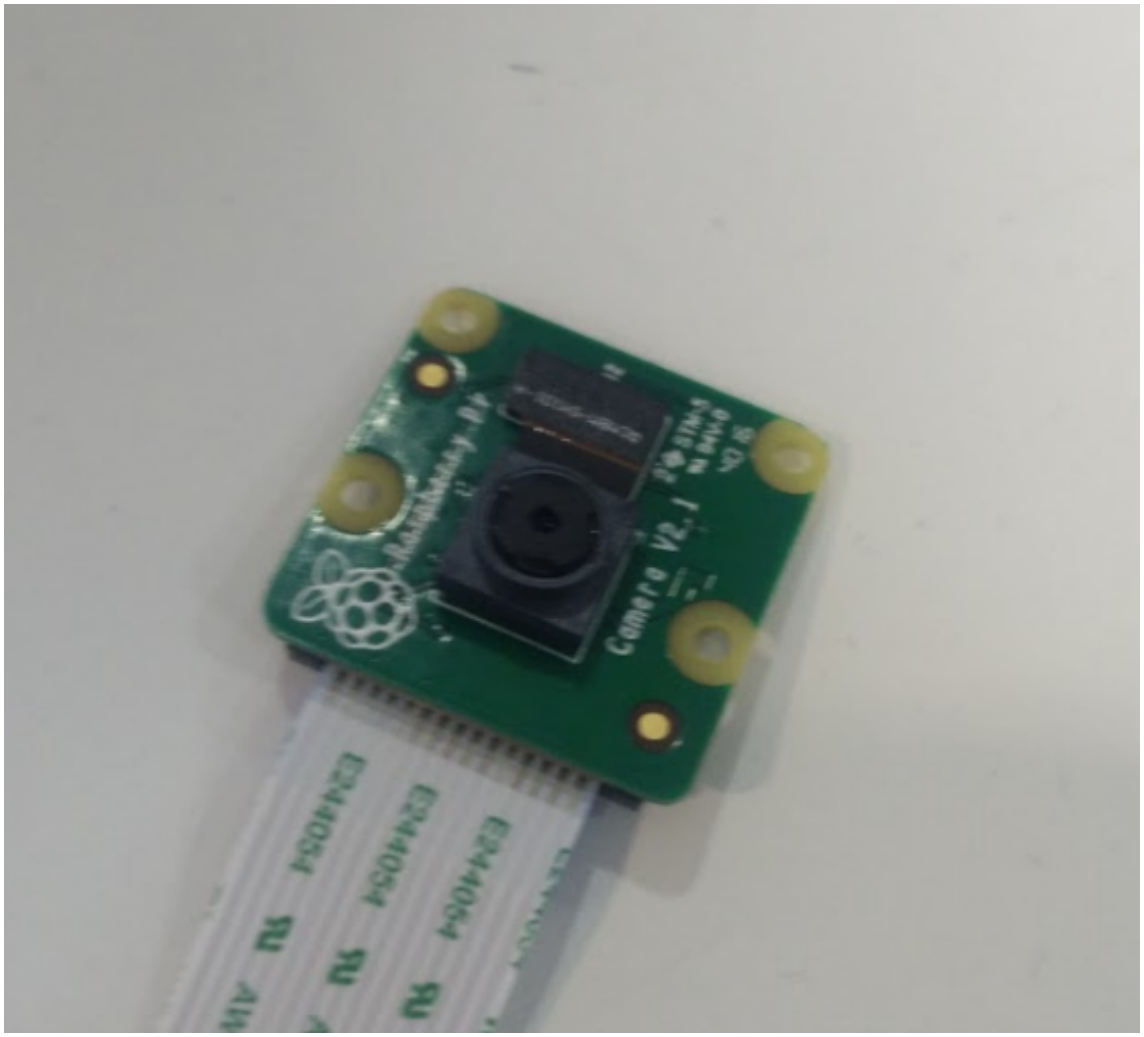
2.1.3 Raspberry Pi 3

Raspberry Pi 3 är en enkortsdator och är den tredje utgåvan av Raspberry Pi. Enheten har en storlek likt ett kreditkort och används främst i mindre system. Mer information går att hitta på tillverkarens hemsida.[15]

Figur 2.1: Raspberry Pi 3

2.1.4 Raspberry Pi Camera module V2

Raspberry Pi Camera module V2 är en kamera som är designad för att lätt kunna installeras och användas av en Raspberry Pi. Kameran kan spela in med en full hd upplösning i 30 bilder per sekund.



Figur 2.2: Raspberry Pi Camera module V2

2.2 Mjukvara

2.2.1 RoboPlus

RoboPlus är ett program som underlättar utveckling till en Bioloid roboten i form av ett flertal mindre applikationer. Programmet är framtaget av Robotis och ingår i Bioloid premium kit[18]. Programmets delar förklaras nedan.

2.2.1.1 RoboPlus Manager

RoboPlus Manager används för att testa hårdvara exempelvis sensorer och drivaxlar. Applikationen kan kolla vilka adresser de diverse delarna använder samt nollställa koden som kör på roboten.[19]

2.2.1.2 RoboPlus Motion

Innehåller verktyg som används för att testa och implementera rörelser. Kan även visa vilka del moment dessa rörelser utförs i, samt hastigheten för varje moment. Applikationen kan även modifiera redan implementerade rörelser.

2.2.1.3 RoboPlus Terminal

RoboPlus Terminal används för att föra över binärafiler till CM530 via seriell kommunikation. Terminalen kan även användas för att skicka och ta emot meddelanden mellan robot och dator. All kommunikation sker seriellt över USB.[20]

2.2.2 Eclipse Neon 2 C/C++

Eclipse är en utvecklingsmiljö med öppen källkod med stöd för flertal programmeringspråk. Eclipse Neon är den senaste stora uppdateringen av Eclipse. Det finns olika paket inom Eclipse Neon som är framtagna för olika varianter av språk.

2.2.3 WinARM

WinARM är ett bibliotek som används för att kompilera C-koden så den blir körbar på mikrokontroller genom att göra om det till en hexfil. Biblioteket är en samling av GNU verktyg för ARM MCU som är inspirerat av WinAVR projektet men mycket enklare och är inte beroende av cygwin eller mingw-miljö. [4]

2.2.4 Raspbian

Raspbian är ett Linux operativsystem som är anpassat för att användas på en Raspberry Pi. Raspbian kommer med förinstallerade programmeringsspråk som Java, Python, Scratch, Sonic Pi och många fler. Raspbian är utvecklat av utvecklarna som gör Raspberry Pi. [16]

2.2.5 Vim

Vim är en textredigerare som tillåter effektiv redigering direkt i en terminal. Vim är användbart för programmering, speciellt i de fall när ett operativsystem saknar ett grafiskt användargränssnitt. [26]

2.2.6 Atom

Atom är en texteditor som marknadsför sig med ”Atom is a texteditor that’s modern, approachable, yet hackable to the core—a tool you can customize to do anything but also use productively without ever touching a config file.”[5]

2.2.7 Git

Git är ett program som används för versionshantering av kod. Programmet kan hantera allt ifrån små personliga projekt till större projekt med hastighet och säkerhet. Med git kan flera användare hålla sig uppdaterade mot samma kodbas.[25]

2.2.8 OpenCV

OpenCV ett högpresterande bibliotek med öppen källkod som är riktat mot dator seende och maskinlärning. Tack vare den öppna källkoden och en bred användarebas så är biblioteket mycket väldokumenterat.[13]

2.2.9 Google Cloud Speech Api

Google Cloud Speech Api är en Googles Cloud tjänst som låter användare snabbt ladda upp ljudfiler för att sedan köra röstigenkänning och returnera resultat. Resultatet innehåller olika alternativ av vad tjänsten anser är de mest troliga orden eller meningarna.[10]

2.2.10 CMU Sphinx

CMU Sphinx är en samling verktyg med öppen källkod byggd för röstigenkänning. CMU Sphinx och dess verktyg är designade för att fungera en stor mängd hårdvara med olika begränsningar.[9]

2.2.11 Makefil för roboten

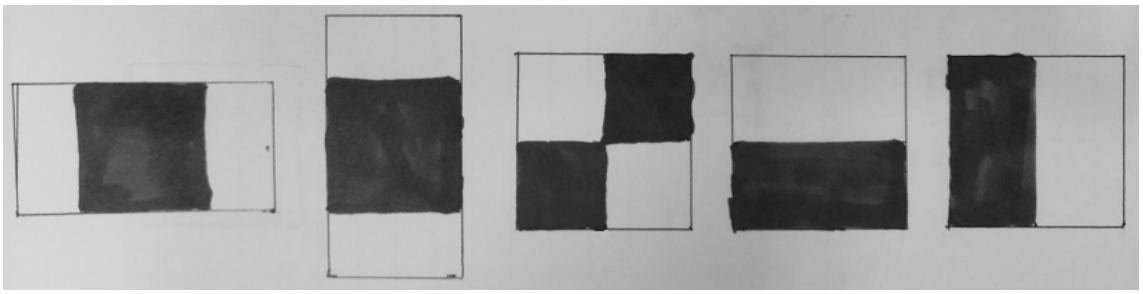
För att föra över kod till robotens kontrollenhet via RoboPlus Terminal behöver koden vara lagrad i en binär hexfil. Detta kan göras genom inställningar i makefilen som låter kompilatorn veta vilket format projektet skall byggas till.

2.3 Teori

2.3.1 Bildigenkänning

2.3.1.1 Haar-feature

Haar-features kan används av datasystem för att beräkna och jämföra skillnaden i ljusstyrka mellan två områden. Bilden nedan demonstrerar hur Haar-features kan se ut:

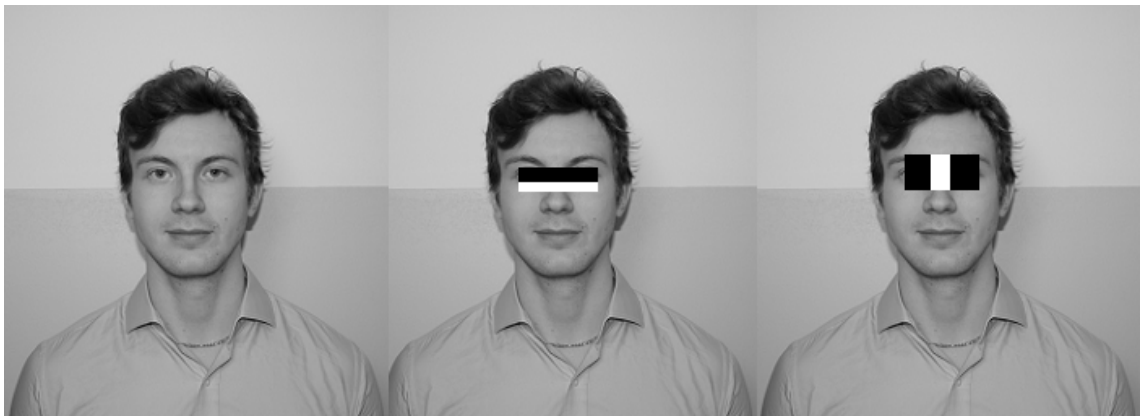


Figur 2.3: Olika Haar-features

Datasystemet kan ”placera” dessa rektanglar i en för att jämföra skillnaderna av bilden som är under rektangeln. Datasystem jämför då de delar av bilden som är under den svarta delen med de som är under den vita. Genom denna jämförelse kan system avgöra om de olika områdena har en eftersökt kontrast mellan sig.

2.3.1.2 Viola–Jones metoden

Viola–Jones metoden för bildigenkänning utvecklades 2001 och går mycket snabbare att köra än tidigare metoder samtidigt som den ger liknande resultat[27]. Metoden gav upphov till en ny start inom utvecklingen av bildigenkännings program. Metoden går ut på att jämnföra delbilder av en gråskalad bild med en så kallad cascade classifier. En Cascade classifier kan ses som en serie av Haar-features. Dessa bygger tillsammans upp vilka färgskillnader som finns i ett objekt. I figur 2.4 demonstreras detta genom att placera två olika haar-features i ett ansikte:



Figur 2.4: Haar-featues i ett ansikte

De två haar-features används för att testa om området under ögonen är ljusare än själva ögat samt övre del av näsan. En annan feature kollar om ögonen är mörkare än övre delen av näsan.

Metoden i sin helhet kontrollerar om en bild passerar alla features i en cascade classifier för att kunna säkerställa om det är ett eftersökt objekt.

2.3.1.3 Framtagning Cascade classifier

Viola–Jones metoden behöver classifiers anpassade för valda eftersökta objekt. Detta är omöjligt att göra effektivt för hand och därför har algoritmer tagits fram för att låta en dator ta fram dem. Detta görs genom en träningsprocess som kan ta flera dagar av körning beroende på parametrar och hårdvara.

Träningsprocessen tar bilder på objekten, så kallade positiva, samt bilder utan objekten, så kallade negativa, för att sedan skapa en serie av Haar-features som kan hittas i objekten. Denna serie anpassas sedan för att ge felaktiga träffar på de negativa bilderna. Efter tillräcklig träning är målet att producera en classifier som bara hittar rätt objekt.

Med träningens upplägg så kommer en classifier alltid bli bättre med fler bilder att tillgå. När denna teknik används av dagens industri brukar tusentals bilder användas under träningen.

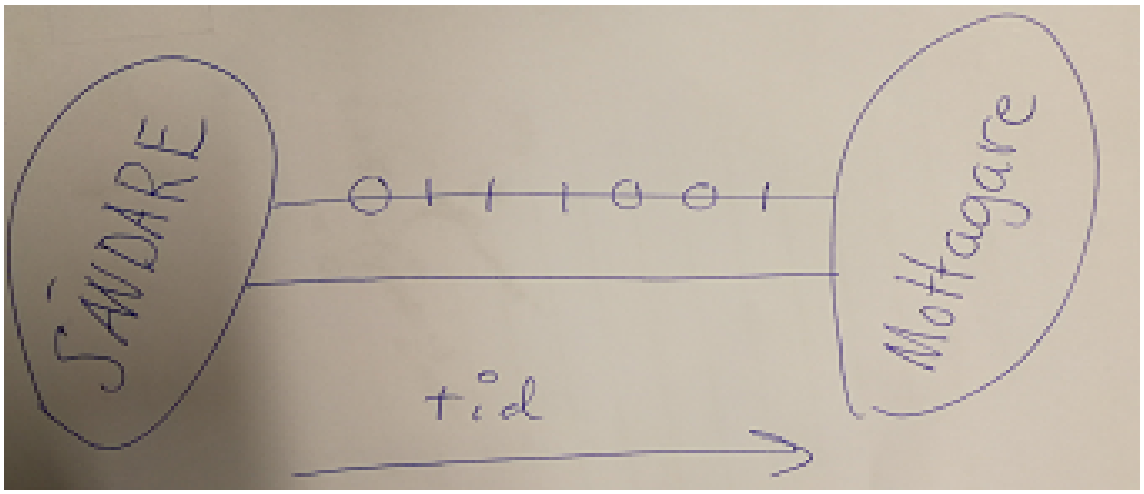
2.3.2 Taligenkänning

Den vanligaste metoden för att känna igen ord är att ta ljudvågorna och dela in dem i uttalanden mellan perioder av tystnad. Dessa uttalanden kan sedan analyseras individuellt och jämföras mot en databas av ljudvågor. Datorn kan sedan ta fram vilken ljudvåg i databasen som bäst överensstämmer med uttalandet.

Med en stor databas blir oddsen för att hitta korrekt ord stor, men körtiden kan istället bli lång. För att göra detta effektivt används ett flertal olika metoder, bland annat representeras uttalanden i kombinationer av siffror. För mer läsning refereras [11].

2.3.3 Seriell kommunikation

Seriell kommunikation är när olika enheter kommunicerar med varandra genom information som skickas en bit i taget. Eftersom informationen är uppdelad är det viktigt att alla parter är medvetna om vilket kommunikationsprotokoll som används.



Figur 2.5: Visualisering av seriell kommunikation

Seriell kommunikation kan göras över sladd men används också i trådlös kommunikation. Över sladd kan kommunikationen teknisk sett göras över en ledare men vanligtvis har sladden ledare för båda riktningar samt ledare för kontrollsignaler. Det finns sladdar och utgångar dedikerade till seriell kommunikation men det kan även göras över USB. [1]

3

Metod

Här beskrivs det upplägg och arbetsgång som valdes att användas för att uppnå önskat resultat samt vilket material som används.

3.1 Förstudie och planering

Innan projektets början ämnar gruppen att göra en förstudie vilket skall resultera i en planeringsrapport samt förståelse kring problemet. Det kommer även göras en överskådlig planering över de veckor som arbetet kommer utföras under.

Planeringen som resulterade finns att hitta i appendix A.1.

3.2 Upplägg

Arbetet i projektet kommer utföras vecka för vecka, deluppgift för deluppgift. För att uppnå detta kommer projektet ha en överliggande planering samt möten inför varje deluppgift. Under dessa möten skall uppgiften diskuteras för att hitta dess krav och hur arbetet skall fördelas. Med detta upplägg kommer arbetet enkelt kunna delas upp mellan medlemmarna, göras på distans och enkelt följas upp. Varje vecka kommer även ett möte hållas mellan projektmedlemmar samt företagshandledare. Under dessa möten kommer resultat och eventuella problem presenteras. Kontinuerliga möten kommer även hållas med handledare från Chalmers för att ge institutionen ett inblick i projektet.

3.3 Arbetsgång

Projektet ämnar att montera en extern enhet på roboten och sedan tillföra funktionalitet med hjälp av en kamera, en mikrofon samt kod. Som extern enhet tänker gruppen använda en extern enhet som kör ett Linux operativsystem.

Projektgruppen ämnar skriva kod i språket C++ i sådan stor utsträckning som är möjligt, med tanke på dess användningsområde i inbyggda system och för att minimera antalet programspråk. Vid utveckling kommer Linux operativsystem användas för att projektet lätt skall kunna föras över till den externa enheten.

Utvecklingsmiljön Roboplus kommer att användas till mindre grad för att skriva och testa kod till roboten. Eclipse Neon 2 kommer användas för att utveckla mot roboten medan valfri textredaktör kommer användas under utveckling mot Raspberry Pi. Atom och Vim är sådan textredaktörer och lär användas mycket.

Projektgruppen planerar att arbeta modulärt och bygga upp de olika delarna individuellt i sådan stor utsträckning som gruppen ser möjligt. Fördelar med att lägga upp arbetsgången på detta sätt är att delarna blir oberoende av varandra. Om en del skapar problem kan den förbises under en period medans andra delar av enheten utvecklas. Detta görs också för att kunna utveckla två oberoende komponenter samtidigt och inte begränsa arbetet till enbart parprogrammering. Nackdelar med detta arbetssätt blir att vid sammanfogning kan de olika delarna vara inkompatibla om det finns otillräcklig planering.

För att uppnå ett modulärt arbetssätt kommer olika delar och moduler placeras i egna mappar som innehåller all nödvändig kod för respektive modul.

Under projektets gång kommer projektet använda versionshantering genom git. Olika så kallade git grenar kommer användas för utveckling av moduler. En git gren låter användaren skapa en ny instans av kodbasen och ändra denna efter behov. När utvecklingen är klar i en gren kan utvecklarna sedan inkludera valda skillnader till den huvudsakliga kodbasen.

3.4 Material

- Bioloid Robot
- RoboPlus
  - RoboPlus Terminal
  - RoboPlus Manager
  - RoboPlus Motion
- Extern enhet med linux - Raspberry pi 3 med Raspbian
- Kamera - Raspberry pi Camera Module V2
- Mikrofon - valfri USB mikrofon
- 3D skrivare

4

Genomförande

I detta kapitel presenteras utvecklandet av de olika modulerna i kronologisk ordning. Utvecklingen av bildhantering använder OpenCV och Viola-Jones metoden. Tar upp hur roboten utvecklades och vilka demonstrationer som har skapats. I alla delar diskuteras det varför de olika metoderna och valen har gjorts samt vilka andra alternativ som undersöktes och varför dessa valdes bort. Tar även upp problem som har uppkommit under arbetet och hur dessa löstes.

4.1 Kravspecifikation

Detta är kraven för den MVP som projektet medlemmarna har kommit fram till och arbetat mot:

- Roboten ska kunna utöva bestämda rörelser.
- Roboten ska kunna läsa av objekt i sin närhet för att undvika kollision.
- Den externa enheten ska kunna ta emot röstkommandon
- Den externa enheten ska känna igen förbestämda objekt
- Den externa enheten ska kunna kommunicera seriellt med Roboten och vice-versa

4.2 Upplägg

Under projektets gång har gruppen arbetat vecka till vecka, deluppgift till deluppgift. De olika deluppgifterna har placerats på en whiteboardtavla i form av post-it lappar för att ge översikt. Projektet hade en överliggande planering med etapper som i sin tur har blivit uppdelade i mindre delar.

Varje vecka har även ett möte hållits mellan projektmedlemmar och en handledare vald av Cybercom. Under dessa möten kunde gruppen presentera resultat och even-

tuella problem. Gruppen har även haft kontakt med en tilldelad handledare från Chalmers och mött denna under projektets gång.

4.3 Arbetsgång

Eftersom projektet har involverat utveckling på två olika plattformar, en Bioloid robot och en Raspberry Pi 3, har arbetet delats in i två delar. All kod har skrivits i C eller C++ men delar har även testats genom bash scripts. Projektets första mål var att etablera kommunikation mellan de två enheterna. Eftersom ett tidigare projekt inom Cybercom har testat möjligheter att kommunicera till roboten över USB genom seriell Kommunikation har detta projekt fortsatt i samma spår.

4.4 Bioloid Robot

Arbetet började med en mindre förstudie på hur föregående projekt har arbetats med för att kunna återanvända funktionalitet och kod. Det visade sig att mjukvaran hos båda projekten grundar sig i den kod Robotis själva ger ut från deras officiella hemsida. Koden kan hämtas här. [2]

Ett problem som blev tydligt under förstudien var att båda projekten utförde samma saker i liknande former och delade namn på variabler med olika ändamål. Till exempel använde ett av projekten en funktion för utskrivning med C standard. Detta leder till att roboten försöker skriva till en terminal som inte finns på ett mikrokontrollkort CM-530.

Biblioteket win-ARM gör att mjukvara kan köras på ett mikrokontrollkort, Eclipse Neo 2 ger då felmeddelanden om koden försöker göra utskrivningar till dess terminal. CM-530 har ingen terminal och efter och vet inte då vad den ska göra med utskrivningarna.

Detta problem löstes genom att låta all kommunikationen från mikrokontrollerkortet ske över en seriell kommunikation. RoboPlus Terminal kunde då användas som mottagare för utskrivningar vilket för lättare utvecklingen.

För att sammanföra de två tidigare projekten utnyttjades kompilatorns möjlighet att rapportera problem. Nödvändig kod blev överförd till ett nytt projekten i intervaller, tills det att projektet kunde byggas och kompileras utan problem. Under denna utveckling bytes många namn ut på olika funktioner och variabler som utförde liknande uppgifter. Detta var nödvändigt då namnen krockade.

Kod blev dessutom borttagen, speciellt i motions.c där det olika värden returnerades utifrån hur det gick att använda rörelserna från dess adresser. Då all styrning av enheten skall ske i den externa enheten minimerades logiken därtill även felhantering som skede i motions.c.

4.4.1 Seriell Kommunikation

Den seriella kommunikationen mellan kontrollkortet CM-530 och Raspberry Pi sker med genom att skicka en byte åt vardera håll. Ett tidigare projekt har arbetat med att hantera den seriella kommunikationen till CM-530. Vilket gjorde att inget större arbete behövdes utföras kring detta då majoriteten var arbetet var utvecklat det som behövdes ändras var mindre små detaljer i kod så som metodnamn och adresser.

4.4.2 Rörelser med RoboPlus

Robotis ger ut ett färdigt rörelsepaket med diverse grundrörelser som utnyttjades i projektet. Detta rörelsepaket fördes över till roboten med RoboPlus Manager och sparas på ett internt flashminne. Genom att om installera rörelsepaketet går det enkelt att veta vilka rörelser som tillhörde vilka minnesadresser. Efter överföringen testades de olika rörelserna, speciellt de som enheten behöver för att uppnå projektets slutgiltiga mål.

Mer specifikt är dessa rörelser de som behövdes för att enheten skall kunna röra sig horisontellt. Vid tester observerades att rörelsen som heter ”Rap Chest” inte fungerade korrekt. Där ”Rap chest” är rörelsen som markerar att enheten har uppnått sitt mål. Rörelsen går ut på att roboten bankar på sitt bröst.

Problemet med rörelsen var att tyngdpunkten förändrades för snabbt framåt vilket resulterade att den föll. Vid undersökning av rörelsens delmoment låg de sista av dessa på en lägre hastighet medans dem första låg på en mycket snabbare hastighet. Det problem löste sig med hjälp av RoboPlus motion där man kunde manuellt ändra hastigheterna individuellt för delmomenten.

4.4.3 Sensorer

Vid början av projektet fanns inte information om vilken sensor som var defekt. Dessutom användes inte sensorn på bröstet i tidigare projekt. För att undersöka vilken sensor som var defekt samt hitta adressen till den okända sensorn testades hårdvaran med RoboPlus Manger.

Med programmet kan värden läsas av i realtid från sensorerna och visar de adresser som dessa är kopplade på. Det gick då att avgöra vilken sensorn som var defekt och vilken adress som den okända sensor är kopplad till.

Trots att sensorerna fungerade via RoboPlus Manger, uppstod problem när de användes i projektets kod. Problemet var att värdena blev annorlunda jämfört med vad de tidigare projekts rapport utgav.

I ett försök att få ut rätt värden och lösa problemet skrevs koden om och utvecklades ifrån Robotis grundkod istället. Detta gav dock samma resultat.

Vid fortsatt undersökning kom insikten att värdena hanteras hexadecimalt som standard. Tidigare projekt har konverterat dessa till decimaltal vilket var orsaken till problemet.

I den kod som Robotis ger ut finns det kod för att läsa av från sensorerna på fötterna.[3] Adresserna i koden modifierades för att ackumulera ytterligare en sensor.

4.5 Extern enhet - Raspberry Pi 3

På Raspberry Pi installerades kamera modulen efter medföljande instruktioner. Enheten är även kopplad till ett USB hörlurar med mikrofon. Hela enheten går att se i appendix A.2.

Som operativsystem används Raspbian 4.4 vilket installerades med hjälp av instruktioner på tillverkarens hemsida[15].

För att skriva kod till Raspberry Pi användes huvudsakligen en linux dator med Atom som textredaktör. Arbetet kontrollerades och testades på datorn för att sedan föras över till Raspberryn genom git. Vid mindre ändringar användes Vim för att redigera direkt på Raspberryn.

4.5.1 Projektstruktur

Arbetet på Raspberry Pi byggdes i en projektmapp med flera undermappar, där varje undermapp innehåller all nödvändig kod för respektive modul. Denna uppbyggnad gjorde det enkelt att byta ut modulernas innehåll samtidigt som projektets andra delar bibehåller sin funktionalitet.

Till varje modul skrevs även ett körbart program som kan testa modulens funktionalitet. Genom att introducera de lokala körbara filerna för varje modul är det enkelt att skriva och testa individuella delar. Detta förenklade utvecklingen och gav en bra överblick över de olika funktionerna.

Genom denna uppbyggnad är förhoppningen att det blir enkelt för framtida utvecklare att sätta sig in i hur de olika resurserna fungerar.

4.5.2 Makefiler

Projektet använder makefiler skrivna för hand. CMake[8] övervägdes för att kunna generera snabba och anpassade makefiler men idén övergavs. Egna makefiler ansågs snabbare och enklare att implementera istället för att lära sig ett nytt verktyg.

Makefilerna har genomgått ett antal iterationer allt eftersom projektet har växt för att anpassa sig efter dess storlek. Från början hanterade en huvudsaklig makefil i

huvudmappen all konstruktion. Under utvecklingen placerades sedan en makefil i alla undermappar. Dessa fick hantera sina individuella resurser och uppdelningen gjorde det enklare att få en översikt.

4.5.3 Moduler

4.5.3.1 Comm - Seriell kommunikation

Denna modul hanterar kommunikationen mellan enheterna på Raspberryns sida. Då tidigare projekt på Cybercom har lagt en grund för seriell kommunikation fortsatte detta projekt i samma spår.

Arbetet började med efterforskningar kring seriell kommunikation i C++ vilket visade att många liknande projekt med öppen källkod finns tillgängliga. Ett av dessa projekt är RS-232 [6]. RS-232 innehåller funktioner för att öppna portar för kommunikation och sedan skicka samt ta emot meddelanden över dessa.

Originellt var tanken att skriva om koden för att anpassa den till projektet men det insågs snabbt att RS-232 är utfört i sådan detalj att litet kunde tillföras. Delar av projektet har dock kod specifikt för Windows operativsystem, vilket har tagits bort på grund av projektets avgränsningar.

4.5.3.2 Audio - Taligenkänning

Denna modul sköter röststyrning till projektet. Denna del är den del som har gått igenom flest iterationer då olika resurser för taligenkänning har testats.

Då Raspberry Pi inte har ett internt ljudkort används en mikrofon som är kopplad genom USB. Ljud enheter över USB har vanligtvis inbyggt ljud kretsar som kan omvandla de analoga ljudsignalerna till digitala.

Efter de initiala efterforskningarna användes ett mindre projekt vid namn Voce[23]. Voce är ett projekt som använder CMU Sphinx4 and FreeTTS internt för röstigenkänning respektive röstsyntes. Voce ett väldokumenterat och fungerande projekt men då det är skrivet i Java är körtiden lång i jämförelse med C++. Projektet tillåter även liten konfiguration av de interna delarna vilket betyder att miljön som kör programmet måste konfigureras för att kringgå eventuella problem.

Trots att Voce är skrivet i Java har skaparen gjort lösningar till C++. Denna lösningen kör en Java instans genom C++ vilket kan köra Java kod.

Ett tidigt problem var att Voce använder systemets standard mikrofon. Eftersom Raspberryn inte har ett internt ljudkort som kan hantera ljudingångar krävdes konfigurationer av ett externt ljudkort till enkortsdatorn.

Operativsystem Raspbian tillåter väldigt stor möjlighet för konfiguration av sina separata delar, detta leder dock ofta till problem för oerfarna användare då mängden möjligheter blir för många. Detta blev mycket tydligt när externa ljudenheter skulle konfigureras till Raspberryn.

Det största problemet var kopplat till hur den senaste upplagan av Raspbian hanterar ljudenheter. Operativsystem har nyligen ändrat hur dessa används vilket gjorde mycket information ut daterat. Efter mycket efterforskning kunde Raspberryn till slut konfigureras rätt.

När alla konfigurationer slutligen fungerade presterade dock Voce inte efter förväntningarna. Orsaken är troligen kopplat till Java och hur C++ kan köra Java kod. Att köra en Java instans i C++ är resurskrävande vilket reflekterades i resultaten.

Efter ytterligare forskning testades Googles Cloud Speech Api, vilket är en Google tjänst som gör taligenkänningen åt användare. Googles tjänst har mycket bra träffsäkerhet, dessutom görs all bearbetning externt vilket lättar arbetet för projektets Raspberry Pi.

Efter initiala tester framstod det att tjänsten inte ökade hastigheten vilket är ett stort fokus. Trots god träffsäkerhet är hastigheten likande den hos en intern lösning på Raspberryn. C/C++ har dessutom inget inbyggt gränssnitt för att använda internet baserade Api:er. Datormoln tjänster innebär också ett krav på ständig uppkoppling samt en risk då tjänsten kan avvecklas. På grund av dessa anledningar valdes tjänsten bort.

Eftersom Google och Voce inte mötte projektets krav och mål så testades istället CMU pocketsphinx. Pocketsphinx är del av samma verktygslåda som CMU sphinx men designad med mindre system i åtanke. Återigen var det problem att konfigurera Raspberry Pi men till skillnad mot Voce kunde konfigurationen göras i verktygen snarare än operativsystemet. CMU pocketsphinx fungerade efter projektens förhoppningar och används som modulens huvudsakliga resurs.

4.5.3.3 Image - Bildigenkänning

Denna modul hanterar bildigenkänning och behandlar bilder som levereras av en Raspberry Camera V2. Till denna modul valdes OpenCV som grundsten att utgå ifrån. Valet gjordes då gruppens handledare på företagets sida samt ena gruppmedlemmen hade tidigare erfarenhet med dess resurser. Dessutom finns det mängder med andra projekt som använder OpenCV på Raspberry Pi 3 tillgängliga på internet vilket kan användas som hjälpmedel.

Installationen av OpenCV gjordes enkelt genom att följa instruktioner på deras hemsida. [13]

OpenCV har egna testprogram och exempel kod för att låta en ny utvecklare komma igång. Delar av detta exempel kod användes som grund för detta projekts utveckling.

Att sedan skriva egna program som använde OpenCV resurser skedde utan större problem.

Bildigenkänning genomförs med Viola-Jones metoden [27] vilket är implementerat genom en funktion internt i OpenCV.

Arbetet på bildigenkänningen har gått igenom några iterationer då det bästa tillvägagångssätt skulle hittas. I början sköttes all kamera initiering internt i modulen, men det resulterade i mycket repeterad kod då alla separata funktioner kör samma uppstart.

För att kring gå detta placerades initiering av kameran utanför modulen vilket resulterade i en mer generell användning av funktionerna. På detta sätt kan de olika funktionerna användas på alla sorters bilder, hur bilderna producerats bestäms längre upp i kedjan.

Med detta upplägg kan man dock argumentera att modulens arbetsuppgifter utförs utanför modulen, vilket är något som vill undvikas. Av denna anledning började arbetet på att skriva en funktion internt i modulen som tar godtycklig funktion som argument och kör denna med kamerans bilder. Arbetet levererade dock inga klara resultat.

När implementationen av bildigenkänningen var klar låg den största utmaningen i att konfigurera parametrar för snabbare körtid. De olika parametrarna lades i en extern konfigurations fil för att underlätta ändring och tester.

4.5.4 Tools

Under utvecklingen så tillfördes kod för att hantera import av konfigurationsfiler. Då flera moduler kan ta hjälp av detta skapades en egen mapp för liknande funktionalitet. INIReader är namnet på det kodstycke som är placerad i mappen. Det är en C++ klass som kan läsa en textfil och extraherar valda värden. Klassen tillåter kommenterar i textfilen vilket underlättar förståelse för de olika värdena och hur de påverkar applikationen.

INIReader är del av ett projekt som kan hittas på Github[7] och är skapat av Ben Hoyt. Projektet bygger på en implementation av filhantering skriven i C med INIReader som ett extra tillägg.

För att minimera antal filer i projektet placerades all nödvändig kod i en fil. Extra funktionalitet var dessutom implementerad för att bättre hantera kommentarer i filerna.

4.5.5 Cascade training

Med OpenCV medföljer ett program samt tillhörande instruktioner som kan träna Haar-cascades, vilket används i bildigenkännings processen. I början användes en personlig dator för tränings processen men sedan placerades träningen på en dator som Cybercom bidrog med.

Programmet behöver en stor mängd bilder med och utan objekten som eftersökts. Bilder på objekten, kallade positiva, behöver ofta manipulerats för hand så att så lite av omgivningen är kvar som möjligt. Negativa bilder behöver inte behandlas men skall absolut inte innehålla det eftersökta objektet.

Träningen genomfördes med hjälp av ett projekt som kan hittas på Github[12]. Projektet innehåller instruktioner och olika skripts för att förenkla förberedelsen och användningen av programmet. 40 positiva bilder blev tagna och manipulerade för hand och kunde sedan användes för att generera ett total på 1000 genom ett skript. 600 negativa bilder användes och skapades genom att filma sekvenser och använda VLC för att spara skärmdumpar från filmen med bestämd frekvens.

Träningen är indelad i 20 steg och kan startas från det steg den var på senast ifall träningen avbryts. Träningen tog cirka 3 dygn på båda datorerna, datorn på Cybercom behövdes dock startas om med jämna intervaller då körtiden blev längre och längre av okänd anledning.

4.6 Kombinerad enhet

Underkapitlet som beskriver hur ihop förandet av de två enheterna, Raspberry Pi och Bioloid robot, utfördes och vilka andra delar som behövdes.

4.6.1 Hållare av Raspberry Pi

För att göra en enhet av Raspberry Pi och roboten skapades en hållare till Raspberryn som kan monteras på roboten. Hållaren har även plats för att placera en kamera på ett stabilt sätt. För att bygga en hållare med precision användes en 3D skrivare. Det skrevs ut fyra olika delar, två för behållaren av Raspberryn, en för kameran och en för att fästa hållaren till roboten. Dessa var sedan monterade tillsammans med hjälp av lim och hakar.

Arbetet började från ritningar som kan hittas på Thingiverse [24] vilket är en sida som låter användare ladda upp och dela 3D ritningar. Vid första försöket att använda 3D skrivaren framstod det problem som var kopplat till materialet, kallat ”Black PETG”. Materialet fastnade inte korrekt på värmeplattan som objekten skrivs ut på. Istället bildades en stor klump massa vid utgångspipan i skrivaren.

Efterforskning visade att värmen på de olika delarna i skrivaren kunde ha varit roten till problemet. Efter ytligare försök med ändrade inställningar såsom förvärma skrivarens olika delar samt byte av material lyckades skrivaren placera objekt på värmeplattan.

Det nya materialet hette ”White PLA” och med detta lyckades de första delarna skrivas ut utan problem. Det är osäkert om vad som vilka ändringarna som löste problemet men med de korrekta konfigurationerna kunde utskrivningen slutföras.

Det uppstod dock ytterligare problem relaterad till hållaren som består av två delar, en underdel och en överdel. Originellt var båda delar i samma ritning men detta resulterade igen i en massa vid utgångspipan. Problemet var löst genom att skriva ut dessa delar individuellt genom att ändra ritningen.

4.6.2 Demonstrationer

Slutsteget i projektet var att sammanföra de två enheterna, en Raspberry pi 3 och en Bioloid robot. För att demonstrera enhetens kapacitet och förmågor skrevs olika funktioner som i något mån behandlar omgivningen.

En av dessa funktioner har som uppgift att få enheten att vrida sig till dess att den hittar ett objekt vald av en användare. Valet och starten av demonstrationen görs genom taligenkänning. För att uppnå detta programmerades enheten till att vrida sig ett steg i taget till dess att kameran registrerar eftersökt objekt. Skulle enheten inte hitta ett objekt efter det att den har roterat ett varv anser enheten att objekten inte finns i dess direkta omgivning och funktionen skall avslutas.

Först togs mätningar på hur många steg enheten behöver för att rotera ett helt varv samt tiden för dessa rörelser. På detta sätt kan Raspberry Pi sedan avsluta funktionen när rätt antal steg är utförda.

I första iterationen användes fördröjningar för att kontrollera antal steg enheten har tagit. Ett problem som uppstod var relaterat till hur den externa enheten hanterar bild igenkänningen. Då bildigenkänningen är skriven för sekventiell hantering hindrade fördröjningarna dess funktionalitet. Detta resulterade i att Raspberry Pi skickade felaktiga signaler till roboten, till exempel att objektet är till vänster när det faktiska objektet är till höger.

Problemet löstes genom att istället använda tidhållning. Med ytterligare tester framgick det att roboten tar cirka 17 sekunder att rotera 360 grader. En funktion kunde då skrivas som jämför Raspberryn:s interna tiden med tiden det tar för roboten att vända sig.

Om ett objekt har registrerats returnerar funktionen värdet sant, annars falskt, vilket sparas i programmet. Skulle enheten hitta ett objekt skall roboten börja röra sig mot det registrerade föremålet. Eftersom enheten slutar rotera när den hittar ett objekt antas det vara framför enheten.

Enheten börjar då röra sig framåt till dess att någon av robotens sensor registrerar ett föremål. Detta gjordes genom att skicka ett värde till roboten som får den att gå framåt och sedan vänta på ett svar från roboten som indikerar att något är framför den. Roboten kan visa att den har hittat något med en rörelse. Denna rörelse är ”Rap chest”.

Skulle enheten inte hitta eftersökta föremålet efter att den har roterat ska den istället gå omkring och leta. För att uppnå detta så programmerades enheten till att gå framåt till dess att den stöter på ett hinder. Ett tal blir då slumpmässigt valt och avgör sedan vilket håll enheten skall börja leta. Detta görs med en slumpmetod som finns i ett standard C++ bibliotek.

Förhinder uppstod när roboten skulle förflytta sig för att undvika ett hinder. Robotens sensorer ger inte utslag på ett avstånd som låter enheten vända sig utan att stöta i hindret. För att lösa detta programmerades enheten att först förflytta sig bakåt innan den roterar 90 grader i den bestämda riktningen.

Ett nytt problem uppstod vid andra iterationen av programmet då enheten ständigt trodde att något var framför den. Problemet var troligtvis relaterat till att roboten utvecklades så den konstant skall skicka information till den externa enheten angående sina sensor värden.

Det resulterade i att Raspberryn samlar denna information i en buffert. Då Programmet återkommer till att förflytta roboten framåt läser Raspberryn från bufferten att det finns något framför den.

Problemet löstes genom att ändra hur mjukvaran på roboten fungerar. Istället för en konstant informationsflöde skickades informationen över seriella länken först när en förändring av sensor värdena skedde. Detta upphörde den konstant informationsflödet och en buffert byggdes aldrig upp i rasbperryn.

5

Resultat

Detta kapitel behandlar vad projektet resulterade i och hur den kombinerade enheten fungerar. Beskriver även vad de separata enheterna kan göra individuellt.

5.1 Slutresultat

Slutprodukten är en kombinerad enhet som kan behandla omvärlden genom bild- och röstigenkänning och agera därefter. De två enheterna, Raspberry Pi 3 och Bioloid robot, kommunicerar genom seriell kommunikation där de båda kan skicka och ta emot data. Raspberry Pi behandlar omgivningen med en kamera samt en mikrofon. Den kan sedan bestämma hur enheten skall reagera med tanke på sin omgivning. Med kameran kan enheten leta efter föremål samtidigt som roboten hanterar avståndet till föremål med sensorer. Hur samarbetet fungerar mellan dessa enheter visas i olika demonstrationer.

5.2 Robot

Vid uppstart initierar roboten de delar som behövs för inläsning av hårdvaran och den seriella kommunikationen. Efter start inväntar roboten kommandon från den externa enheten parallellt som den läser av de sensorerna som är installerade.

Skulle den externa enheten styra roboten att utföra en manöver kommer denna utföras tills det att roboten ombeds sluta. På detta sätt utföras alla rörelser från en noll punkt och kommer inte avbryta en annan rörelse oavsiktligt.

5.2.1 Kommunikationen

De två funktionerna som hanterar den seriella kommunikationen är:

TxDBytePC - Skickar en byte över den seriella kommunikationen från Bioloid robot till den externa enheten som är kopplad via USB.

stdgetchar - Läser in en byte över den seriella kommunikationen och returnerar värdet som sedan kan sparas i programmet.

Nedan visas de rörelser som är väsentliga för projektets mål. Trots detta finns fler rörelser programmerade på internminnet. Rörelserna utförs genom att skicka följande ASCII tecken över den seriella kommunikationen:

- Framåt, vid 'w'
- Rotera åt höger, vid 'd'
- Rotera år vänster, vid 'a'
- Bakåt, vid 's'
- Stanna, vid 'b'
- Rap Chest, vid 'c'
- Stå upp, vid 'u'
- Sitt ner, vid 'g'

5.2.2 Rörelserna

Roboten använder följande funktioner för att utföra sina rörelser:

executeMotion - Utför en rörelse lagrat på det interna minnet. På minnet är en adress är kopplad till en rörelse. Adressen har ett numeriskt värde som parameter och funktionen packar senare upp rörelsen som är kopplad till detta värde.

ExecutemotionIfIdel - Funkar likt executeMotion fast använder en buffert istället för att rörelsen packas upp direkt. ExecutemotionIfIdel väntar på att funktionen executeMotionSequence ska användas och utför rörelserna om roboten inte redan utför någon annan rörelse.

executeMotionSequence - Utför alla rörelser i bufferten.

checkSensor - Läser av sensorerna i realtid och om någon ger utslag så skickas en byte över till den externa enheten.

inputMotion - Funktion som har byten som läses in av stdgetchar som in-parameter. Lägger denna rörelsen i form av en siffra till ExecutemotionIfIdel:s buffer.

5.2.3 Programstruktur

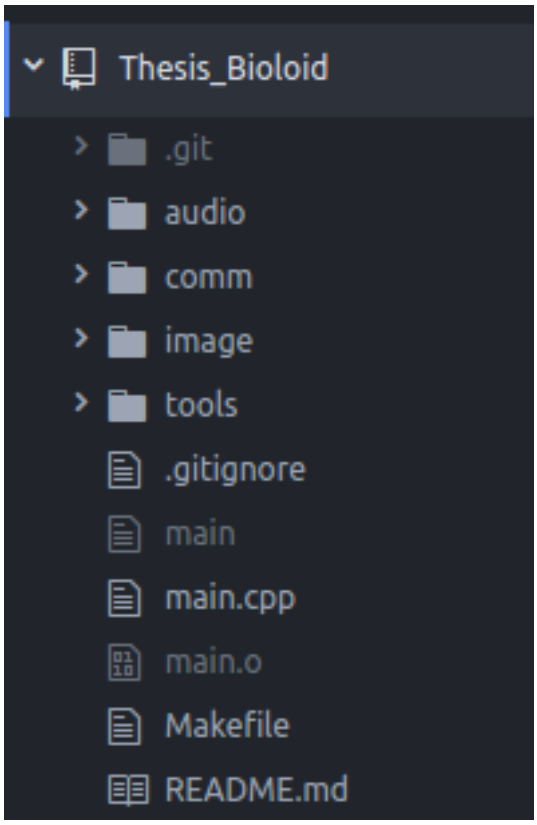
Programstrukturen upplagd på ett sätt som skall förenkla förståelse. Huvudmappen innehåller en makefile som kan bygga en hexfil från projektets kod. Utöver dessa

filer finns också två undermappar där projektets kod har placerats.

5.3 Extern enhet - Raspberry Pi 3

5.3.1 Projektstruktur

Projektet är byggt i den struktur som visas till under. Huvudmappen, Thesis_Bioloid, innehåller flera undermappar. Dessa undermappar är audio för taligenkänning, comm för kommunikation, image för bildigenkänning samt tools för generella verktyg och olika konfigurations filer. Strukturen är designad för att vara lätt förståelig och ge användaren en överblick över projektets helhet. För större förståelse kan även README filen rådgöras. Förhoppningsvis ger denna uppbyggnad nya utvecklare en enklare väg till att förstå programmet.



Figur 5.1: Projektstruktur på Raspberry Pi

5.3.2 Makefiler

I projektets huvudmapp finns en makefile som kan bygga alla delar av projektet efter behov. Varje modul har egen makefile som kan bygga sin egen modul samt en tillhörande körfil. Genom att använda ”make” i huvudmappen kommer projektet

rekursivt gå igenom och bygga de delar som har ändrats sen senaste make. Alla makefiler har en ”clean” funktion för att rensa körfiler och länkade objekt. I huvudmappen kan användaren kalla på ”make cleanall” för att rensa alla körfiler och länkade objekt i hela projektet. Makefilerna är dokumenterade och minimalistiska för att underlätta förståelse.

5.3.3 Moduler

5.3.3.1 Image - Bildigenkänning

Denna modul hanterar allt bild relaterat. Modulen har har ett flertal funktioner som kan testas individuellt genom dess körbara fil. Dessa tester är:

- Detection test
- Orientation test
- Number of detections test

De olika testen kan enkelt väljas genom program argument vid start. Alla demonstrationer är baserade på detektion genom haar-cascades som genomgås mer i detalj tidigare kapitel2.4. Eftersom programmet är gjort för att köras på en Raspbeery Pi är hastighet en viktig egenskap och alla tester har möjligheter att skriva ut detektions hastighet samt hastigheten för funktionens körning. På detta sätt blir det enkelt för utvecklare att få en översikt på prestandan.

5.3.3.2 Comm - Kommunikation

Denna modul hanterar all kommunikation mellan enheterna. Modulen har har ett flertal funktioner som demonstreras genom dess körbara fil.

Den körbara filen låter användare skicka meddelande över USB genom en terminal och är främst skriven för att underlätta utveckling och testning av roboten.

Modulen använder seriell kommunikation över USB och de viktigaste funktioner går att se nedan:

```
int RS232_OpenComport(int, int, const char *);
int RS232_PollComport(int, unsigned char *, int);
int RS232_SendByte(int, unsigned char);
int RS232_SendBuf(int, unsigned char *, int);
void RS232_CloseComport(int);
```

Figur 5.2: Del av RS232.h

De olika funktionerna har beskrivande namn och användandet har ett logiskt följe. RS232_OpenComport öppnar kommunikationen och måste utföras först. När porten är öppnad kan RS232_PollComport läsa från porten medan de resterande funktioner är självbeskrivande.

5.3.3.3 Audio - Taligenkänning

Denna modul hanterar taligenkänning. Modulen bygger på CMU pocketsphinx. Modulen kan testas genom dess körbara fil som går igenom de nödvändiga stegen för att ta emot och känna igen röster. Dessa demonstreras nedan:

```
6  int main(int argc, char *argv[])
7  {
8    string s;
9    int pos;
10   bool printM = true;
11   audio_init("lib/2005.lm","lib/2005.dic");
12   thread listen(audio_listen);
13   while(1)
14   {
15     if(printM)
16     {
17       cout<<"Listening.."<<endl;
18       printM = false;
19     }
20     if(audio_getCommandsSize() > 0){
21       printM = true;
22       cout<<"Heard this:"<<endl;
23
24       s="";
25       s=audio_popCommand();
26       cout<<"Popped command: "<<s<<endl;
27       cout<<"Command: "<<audio_parseCommand("ROBOT",s)<<endl;
28     }
29   }
30   listen.join();
31   return 0;
32 }
```

Figur 5.3: En bild på audio modulens main funktion

De nödvändiga steg för att lyssna efter meddelanden är:

- Rad 11, initiera modulen genom att ge namn på ordlistan den skall använda.

- Rad 12, sätt audio_listen på en egen tråd. Funktionen går in en oändlig loop och bör köras i bakgrunden.
- Rad 20, kolla om lyssnaren har registrerat något.
- Rad 25, ta ut meddelanden ur lyssnaren.

Efter dessa steg kan meddelandet hanteras efter användarens behag. Funktionen audio_parseCommand tar ett nyckelord och en sträng och returnerar allt efter nyckelordet, vilket förenklar röststyrning.

I Audio mappen finns även undermappar som innehåller resurser för att använda Voce och Google Speech Api. Trots att de inte används finns de kvar för att ge in inblick i hur modulen kan formas.

5.3.4 Tools - Verktygslåda

Denna mapp innehåller kod som kan importera konfigurations filer och extrahera valda värden. Koden används av alla delar av projektet.

5.4 Demonstrationer

Projektet har resulterat i ett samarbete mellan en Bioloid robot och en Raspberry pi. Dessutom kan den externa enheten behandla omgivningen genom ljud och bild. Detta demonstreras genom olika funktioner som utnyttjar projektets olika moduler för ett utföra ett programmerat beteende.

Demonstrationerna startas genom röstkommandon av en användre. Ett exempel är ”Robot find controller” vilket gör att enheten börjar rotera 360 grader eller till dess att kameran på den externa enheten känner igen ett föremål.

Skulle enheten känna igen ett objekt förflyttar sig enheten mot objektet till dess att objektet är framför den. När objektet är framför enheten utförs ”Rap chest”.

Om objektet inte hittas börjar enheten leta genom att gå tills den stöter på ett hinder. Skulle enheten stötta på något backar den ett steg och börjar sedan om proceduren genom att börja rotera igen.

Bilden, figur 5.4, nedan visar hur den huvudsakliga demonstrationen är strukturerad och visar vilka steg programmet tar samt vilka värden som påverkar robotens beteende. Innan demostationen startas är den seriella kommunikationen öppnad med hjälp av ”RS232_OpenComport”.

```
121  void findObject(string cascadef){
122    bool object = false;
123    do{
124      object = turnTo(cascadef);
125      sleep(3);
126
127      walkForward();
128
129      send_buffer[0] = 'u';
130      RS232_SendBuf(comport, send_buffer, SEND_CHARS);
131      sleep(3);
132
133      if(object == true){
134        send_buffer[0] = 'c';
135        RS232_SendBuf(comport, send_buffer, 1);
136        break;
137      }
138
139      avoidObstacle();
140
141    }while(!object);
142
143  }
```

Figur 5.4: Demonstration

- Rad 124, Kallar på funktionen som gör att roboten vänder sig till rätt objekt. Hittas objekt returneras sant annars falskt.
- Rad 127, Kallar på en funktion som gör att enheten går framåt tills den stöter på ett föremål.
- rad 129 och 130, Skickar ett värde till roboten så att den ska ställa sig upp korrekt, görs för att roboten efter förflyttningen kan vara i ett framåt lutat läge.
- rad 131, Raspberry Pi väntar på att rörelsen utförs av roboten.
- rad 133, Undersöker om det sparade värdet är sant eller ej. Om det är sant så utförs ”Rap chest”.
- rad 136, Är det sparade värdet är sant avslutas main loopen.
- rad 139, Försöker undvika hinder om sådan finns.
- rad 141, Börjar om proceduren.

6

Slutsats

I detta kapitel kommer de slutsatser och diskussionspunkter som projektgruppen har kommit fram till presenteras. Kapitlet kommer börja med en kort resumé av projektet för att sammanfatta projektets resultat.

6.1 Resumé

Detta projekt hade som syfte att utveckla en enhet bestående av en Bioloid robot och en Raspberry Pi 3. Enheten skulle kunna behandla omvärlden och agera därefter.

För att uppfylla syftet har projektgruppen monterat en Raspbery Pi 3 på en Biolid robot och sedan skapat en seriell kommunikation mellan dem. Raspberry Pi har sedan försetts med en kamera och en USB-mikrofon. Projektet har utvecklats i C/C++ och använt ett flertal bibliotek med öppen källkod. För bildhanteringen har OpenCV och Viola-Jones metoden använts. Taligenkänning har uppnåtts med hjälp av CMU pocketsphinx. Utvecklingen på Bioloid robot utgick ifrån två tidigare projekt som fokuserade på seriell kommunikation respektive förflyttning av roboten.

Tack vare träffsäkerheten hos igenkänning av föremål samt ljud kan den kombinerade enheten behandla omgivningen. Efter behandling kan Raspberry Pi använda seriella kommunikationen för att styra roboten efter programmerat beteende.

6.2 Kritisk diskussion

Ett problem med utvecklingen inom bildigenkänning var att den tyvärr var långsam då maskinlärning användes. Att använda haar-cascades ger bra resultat till priset av långa träningsperioder. Den långsamma processen beror på av att man behöver ett externt program som jobbar under en längre tid, cirka 3 dygn på dagens datorer. Den slutliga produkten blir dock väldigt användbar då den inte avgränsas till att endast fungera i en miljö.

Arbetet på Bioloid robot började med att sammanföra de två tidigare projekten. Detta visade sig vara ett dåligt tillvägagångssätt att börja utvecklingen. Med bris-

tande förstudie så krävdes det ofta nya studier på hur projekten var uppbyggda för att lösa problem. Ett mer optimalt alternativ hade varit att börja på ett helt nytt projekt och använda dessa två som inspiration och information.

Nackdelar med det nuvarande resultatet är främst relaterat till den tid det tar för roboten att utföra sina rörelser. Mjukvaran på den externa enheten körs i en snabb hastighet vilket leder till att den kan skicka flera signaler än roboten hinner bearbeta. Därför måste Raspberry Pi endast skicka ut en signal och sedan låta roboten bearbeta denna innan Raspberryn kan skicka ut en ny signal. Detta leder till att Raspberry Pi måste vänta väldigt ofta och kommer aldrig upp till sin potential och maximala effektivitet.

Utöver detta kan enheten för tillfället inte utföra några konkreta arbetsuppgifter som är till nytta. Projektet har främst utvecklat verktyg för att förenkla vidareutveckling av projektet. Ett sätt att fortsätta detta projekt är att modulesiera ytterligare genom att tillsätta ytterligare enheter. En enhet skulle till exempel hantera allt relaterad till bildigenkänning. På detta sätt kan körtiden optimeras samtidigt som designen blir mer framtidssäker då olika delar lätt kan bytas ut.

6.2.1 Miljö och hållbarhet perspektiv

Utvecklingspotentialen för robotar med bildigenkänning är ofantligt stor för att hjälpa miljön både på en global och- samhällesnivå. På en liten skala inom snar framtid kan det användas till maskiner som till exempel städar parker eller andra öppna ytor. På global skala kan bildigenkänningen användas för källsortering vilket låter robotar effektivisera processen i alla delar av världen.

6.3 Slutsats

Samarbetet kan efterliknas till en biologisk kropp där alla organ har egna egenskaper och tillför med olika fördelar till kroppens helhet. Detta projekt har producerat en enhet som kan efterlikna en kropp där den externa enheten efterliknas med en hjärna medan robotens interna processor agerar som ett nervsystem. Hjärnan har dessutom tillgång till ögon i form av en kamera och öron i form av en mikrofon.

Framtiden för robotik, och kanske även för detta projekt, ligger troligen i kombinationen av individuellt starka enheter. Tillsammans kan de med deras styrkor bygga ett samarbete som bryggar deras individuella nackdelar.

Litteratur

[1] URL: http://www.rejas.se/fritis/datorkommunikation/chap_serpar.html (hämtad 2017-06-07).

[2] URL: http://support.robotis.com/en/software/embeded_c/cm530/example_stm.htm (hämtad 2017-06-07).

[3] URL: http://support.robotis.com/en/software/embeded_c/cm530/example/ir_sensor_530.htm (hämtad 2017-06-07).

[4] Admin. "Quick start using WinARM". I: (). URL: http://www.zembedded.com/setting-up-eclipse-and-code-sourcery-lite-for-stm32-discovery-development/ (hämtad 2017-05-10).

[5] Atom. URL: https://atom.io/ (hämtad 2017-06-06).

[6] Teunis van Beelen. URL: http://www.teuniz.net/RS-232/# (hämtad 2017-06-06).

[7] Benhoyt. URL: https://github.com/benhoyt/inih (hämtad 2017-06-06).

[8] Cmake. URL: https://cmake.org/ (hämtad 2017-06-06).

[9] CMUSphinx. URL: https://cmusphinx.github.io/wiki/tutorialconcepts/ (hämtad 2017-06-06).

[10] Google. URL: https://cloud.google.com/speech/ (hämtad 2017-06-06).

[11] X D Huang m. fl. "Briefly Noted Hidden Markov Models for Speech Recognition Conceptual Information Retrieval: A Case Study in Adaptive Partial Parsing". I: (). URL: http://www.anthology.aclweb.org/J/J93/J93-1016.pdf (hämtad 2017-06-08).

[12] MrNugget. URL: https://github.com/mrnugget/opencv-haar-classifier-training (hämtad 2017-06-06).

[13] OpenCV. URL: http://opencv.org/ (hämtad 2017-06-06).

[14] University of Ottawa. "History of Robotics". I: (2003). URL: https://pdfs.semanticscholar.org/4f5f/1f60b8484afc46a380cc75e3f688c6354940.pdf (hämtad 2017-06-08).

[15] Raspberry. URL: https://www.raspberrypi.org (hämtad 2017-06-06).

[16] raspberrypi. URL: https://www.raspberrypi.org/downloads/raspbian/ (hämtad 2017-06-07).

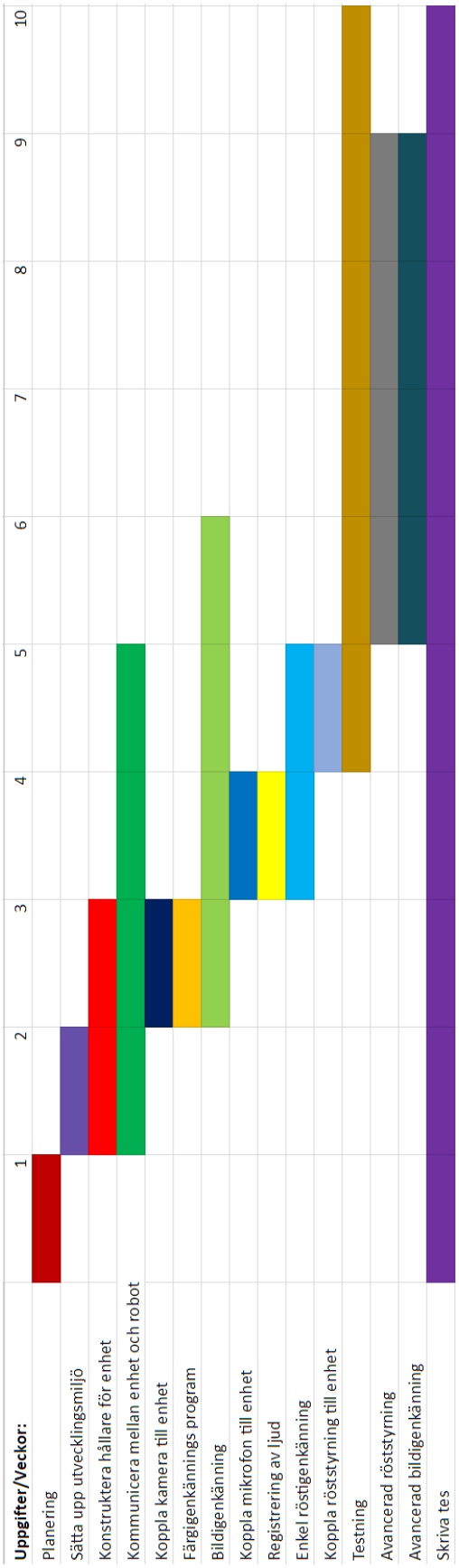
[17] Robotis. URL: http://en.robotis.com/index/product.php?cate_code=121010 (hämtad 2017-06-06).

[18] Robotis. URL: http://support.robotis.com/en/home.htm (hämtad 2017-06-07).

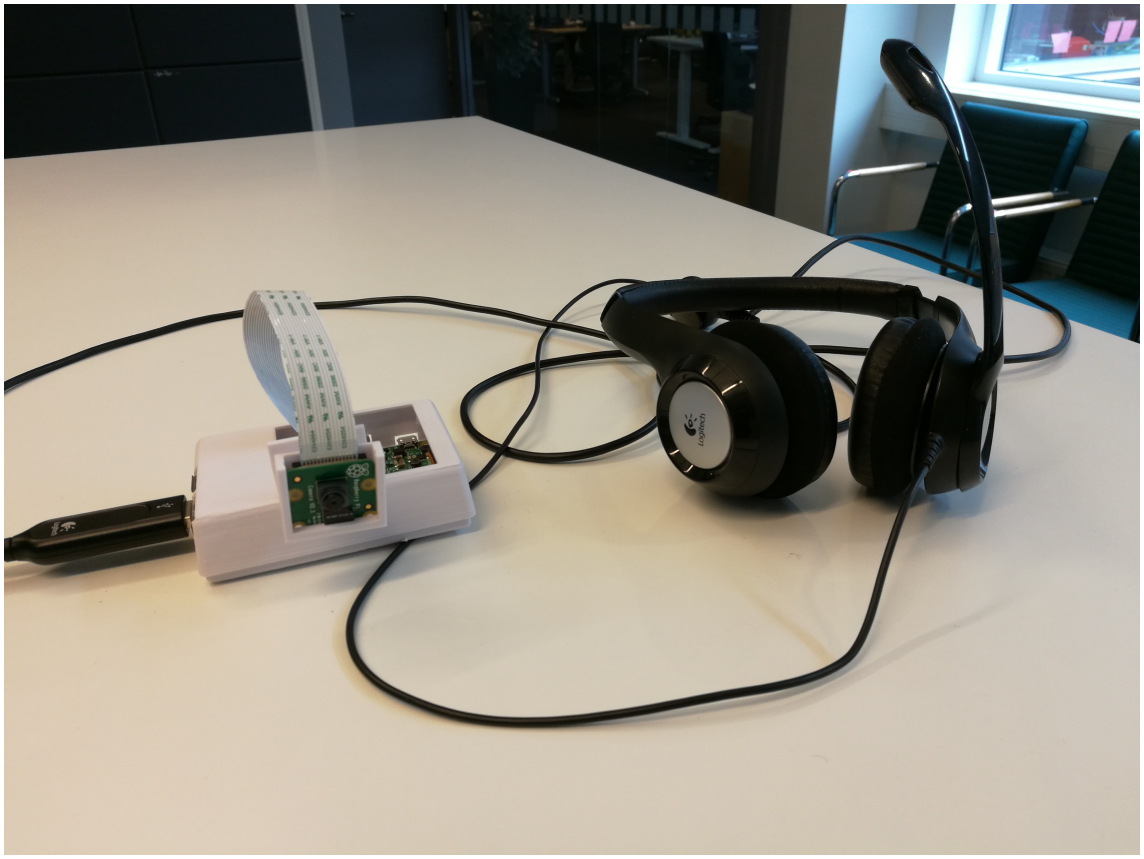
[19] Robotis. URL: http://support.robotis.com/en/software/roboplus/roboplus_manager_main.htm (hämtad 2017-06-06).
[20] Robotis. URL: http://support.robotis.com/en/software/roboplus/roboplus_terminal_main.htm (hämtad 2017-06-01).
[21] Robotshop. URL: http://www.robotshop.com/en/dynamixel-ax-12a-smart-servo-serial.html (hämtad 2017-06-06).
[22] Robotshop. URL: http://www.robotshop.com/en/robotis-bioloid-cm-530-controller.html (hämtad 2017-06-06).
[23] Tyler Streeter. URL: http://voce.sourceforge.net/ (hämtad 2017-06-06).
[24] Thingiverse. URL: https://www.thingiverse.com (hämtad 2017-06-06).
[25] Linus Torvalds. URL: https://git-scm.com/ (hämtad 2017-06-06).
[26] Vim. URL: http://www.vim.org/about.php (hämtad 2017-05-06).
[27] Paul Viola och Michael J Jones. "Robust Real-time Object Detection". I: (2001). URL: http://mame.myds.me/bitsavers/pdf/dec/tech_reports/CRL-2001-1.pdf (hämtad 2017-06-08).

A

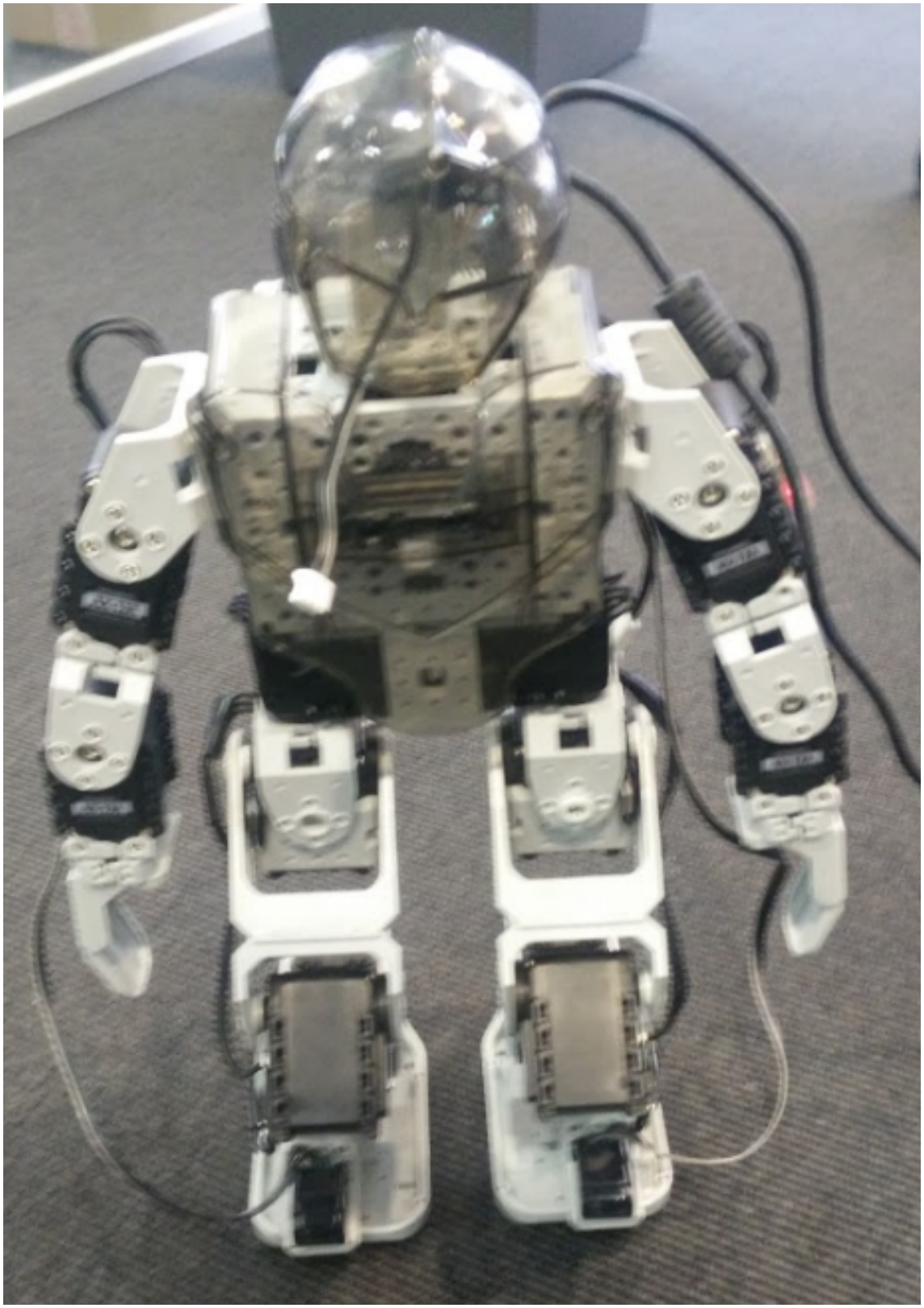
Appendix 1



Figur A.1: Tidsplan



Figur A.2: Extern enhet



Figur A.3: Bioloid Robot