Teknisk dokumentation för robotbil

Daniel Häggmyr
MindRoad

13. augusti 2015
v1.0

Version	Datum	Författare	Utförda ändringar
1.0	2015-08-11	DH	Första version

Innehållsförteckning

1 Introduktion

Detta projekt gjordes på MindRoad under sommaren 2015, med syfte att ta fram en robot med tillhörande mjukvara som ska kunna visas upp på mässor och liknande tillställningar. Resultatet blev en robot som kan skicka en videoström och styras manuellt över nätverk eller automatiskt följa ett objekt baserat på färg, samt datormjukvara som kan ansluta till denna.

Detta dokument tar i detalj upp de olika enheterna i roboten, alla komponenter som ingår och hur mjukvaran och hårdvaran är uppbyggd och fungerar. I slutet av dokumentet tas även projektets avgränsningar upp samt förslag på en del vidareutvecklingar som kan göras.

2 Översikt

Roboten består av tre delsystem: en Raspberry Pi, en mikrokontroller och klientmjukvara. Det här avsnittet tar upp en översikt av dessa system, hur de kommunicerar med varandra samt vilken hårdvara som finns i roboten.

2.1 Komponentlista

De komponenter som ingår i roboten är:

1st Raspberry Pi Model B+ med låda
1st Raspberry Pi Camera med låda och bandkabel
1st USB Wifi-adapter
1st Arduino Nano-kompatibel mikrokontroller
1st mini-USB-kabel
1st micro-USB-kabel
1st batteripack för mobiltelefoner
1st batterihållare för 4st AA-batterier med på/av-knapp
4st motorer med hjul
1st H-brygga
1st statuslampa
1st plexiglaschassi

2.2 Blockschema

Bild 1 nedan visar ett blockschema över de tre olika delsystemen i roboten. Blockschemat visar även kameran och elektroniken som block, för att få en helhet i hur roboten hänger ihop.

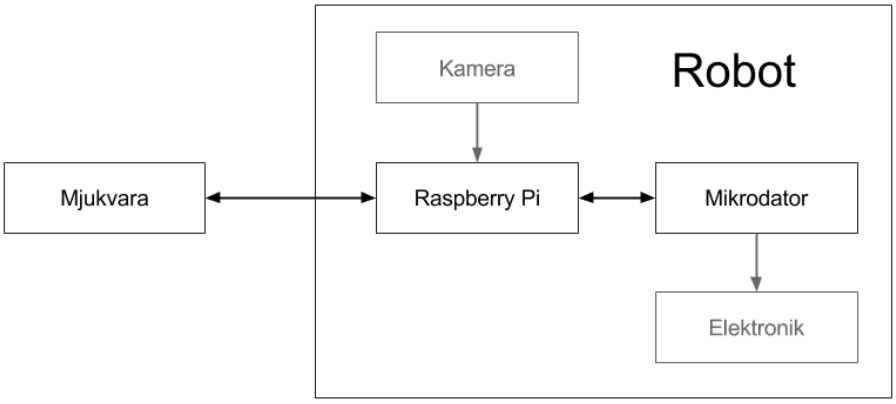

Bild 1: Blockschema över systemet

2.3 Raspberry Pi

Raspberry Pin är av modell B+ och kör en version av Linux-distributionen Raspbian. Under körning drivs den av ett portabelt batteripack. Robotprogramvaran är skriven i C och använder bland annat bildhanteringsbiblioteket OpenCV, mer detaljer om detta finns i sektion 3.2. Programmet körs automatiskt vid uppstart av Pin så att användaren inte ska behöva en skärm eller tangentbord för att starta roboten. Pin kommunicerar med mjukvaran över Wifi och med mikrokontrollern genom en serieport över USB. Den läser även kontinuerligt av kameran under körning.

2.4 Mikrokontroller

Mikrokontrollern är en Arduino Nano-kompatibel enhet som styr statuslampan och motorerna. Den startar automatiskt när den får ström genom USB, men utför ingenting innan den får instruktioner över serieporten. Mikrokontrollern skickar styrsignaler till elektroniken för höger och vänster motorpar som kan styras individuellt, och hastigheten regleras med hjälp av pulsbreddsmodulering. Mikrokontrollern styr även robotens statuslampa, som är en RGB-lysdiod. Varje färg kan individuellt sättas på och stängas av, dock så kan de inte styras med pulsbreddsmodulering.

2.5 Elektronik

Elektroniken i roboten består, förutom Pin och mikrokontrollern, av ett plexiglaschassi med fyra motorer och hjul, en H-brygga för motorstyrning och en statuslampa.

H-bryggan används för att driva motorerna, och möjliggör styrning av de två hjulparen i båda riktningarna. Denna krets får ström separat från de övriga komponenterna, då den kräver alldels för mycket ström för att Pin ska klara

av att driva hjulen. Den drivs av fyra stycken AA-batterier som sitter i en batterihållare i bakkant på robotens ovansida. Bredvid denna hållare sitter en strömbrytare som sätter på eller stänger av strömmen till H-bryggan. När H-bryggan har ström visas detta genom att en röd lysdiod tänds på kortet.

Statuslampan består av en ytmonterad RGB-diod och ett motstånd på ett kretskort. Den har fyra kontakter, en för spänning och en för varje färg. För att få dioden att lysa i en färg ska man ansluta motsvarande kontakt till jord. Mikrokontrollern kan i teorin aktivera alla tre färger samtidigt, men i praktiken används endast en åt gången.

2.6 Klientmjukvara

Klientmjukvaran är skriven i C++ i en Windows-miljö, och fungerar för tillfället enbart på Windows-datorer. Mjukvaran använder bland annat OpenCV, men detta behöver ej vara installerat på klientdatorn eftersom alla nödvändinga DLL-filer medföljer till programmet.

I mjukvaran kan man ansluta till roboten, se videoströmmen den skickar samt fjärrstyra den på olika sätt. Programvaran kommunicerar med roboten över Wifi och tar emot video samt skickar instruktioner. Mer information om hur kommunikationen ser ut finns i avsnitt 6.

2.7 Protokoll

I detta projekt har två egenskrivna protokoll används för kommunikationen mellan klienten och Pin över Wifi samt mellan Pin och mikrokontrollern över en serieport. Utformningen på dessa protokoll är väldigt snarlika och skiljer sig bara nämnvärt genom deras olika instruktionsset. Roboten skickar även en videoström till datorn över wifi, detta sker dock separat från dessa protokoll som finns beskrivna i avsnitt 6.

3 Delsystem - Raspberry Pi

Detta avsnitt beskriver Raspberry Pi-delsystemet i mer detalj. Delsystemet består av en Raspberry Pi B+ med låda, en Raspberry Pi Camera Module med låda, en bandkabel för att ansluta de två samt en Wifi-enhet.

3.1 Funktionalitet

Detta delsystem har som uppgift att vara ”hjärnan” i roboten. Dess viktigaste uppgifter är att kontinuerligt skicka bilder från kameran till klientmjukvaran samt att ta emot rörelseinstruktionern från mjukvaran och skicka rörelseinformation till mikrokontrollern. Det ska även filtrera bilderna från kameran efter färg och kunna spåra ett färgat objekt, och säga till mikrokontrollern hur den ska styra motorerna för att följa objektet. Det ska även se till att roboten stannar ifall något i nätverkskommunikationen går fel.

3.2 Mjukvara

Mjukvaran för detta delsystem är skrivet i C och förlitar sig till stor del på två bibliotek, OpenCV samt stödbiblioteket raspicam_cv för att få kameran att fungera med OpenCV. Nätverkskommunikation och serieportskommunikation sköts av två egenskrivna bibliotek, vars instruktioner kommer beskrivas mer detaljerat i avsnitt 6.

Under utveckling, när programmet startas manuellt på Pin, finns det två inparametrar man kan använda för att eventuellt underlätta processen. När programmet är kompilerat kan man starta programmet med "./robot", förutsatt att man står i programmets rotmapp. I standardläge utan flaggor startas programmet utan en lokal videoström och med timern för tappad kontakt aktiv. Dessa kan styras med två inparametrar, där den första styr om videoströmmen är på eller av och den andra om timern är aktiverad eller ej. En etta som inparameter sätter motsvarande funktion aktiv och en nolla funktionen som inaktiv. För att starta programmet med lokal video aktiv ska kommandot "./robot 1" köras, och för att starta programmet med video på och timer avstängd ska kommandot "./robot 1 0" köras.

3.2.1 Mjukvarukrav

För att programmet ska fungera måste bibilioteken OpenCV, Userland och raspicam_cv vara installerade och korrekt länkade. OpenCv kan laddas ner på `opencv.org`, Userland på `https://github.com/raspberrypi/userland` och raspicam_cv kan laddas ned på `github.com/robidouille/robidouille/tree/master/raspicam_cv`. Dessa bibliotek är redan nedladdade och länkade på Raspberry Pin, och om möjligt så rekommenderas att fortsätta utveckla på samma konfiguration eftersom det tar lång tid och är någorlunda komplicerat att installera och länka OpenCV. Biblioteket

Länkning görs genom en Cmake-fil, där sökvägarna till alla bibliotek och filer är inlagda. När kommandot *cmake* sedan körs skapas en Makefile, som sedan kan körs med kommandot *make*. Detta skapar en exekverbar fil vid namn *robot*.

3.2.2 Bildhantering och inläsning

Biblioteket OpenCV fungerar väldigt bra för att läsa in bilder från tex. webbkameror, men har inget stöd för kameramodulen som användes i det här projektet. Detta beror delvis på att kameran använder en speciallösning och en särskild kontakt på Raspberry Pi-kortet. Det fanns dock ett stödbibliotek tillgängligt online som läste in bilder från kameran och konverterade dem till ett OpenCV-kompatibelt format. Detta bibliotek var dock skrivet i C och använde en något gammal bildstruktur, $IplImage$, istället för det nyare $CvImage$. Detta gjorde det dock möjligt att använda kameran vilket annars hade varit mycket svårt, till priset av att det inte gick att använda de nyaste strukturerna.

För färgigenkänningen används huvudsakligen två funktioner, $cvCvtColor$ och $cvInRangeS$. Den första av dessa används för att konvertera mellan olika bildformat, i det här fallet från RGB till HSV (Hue, Saturation, Value - Nyans,

Mättnad, Intensitet). Att ha en bild i HSV-läge innebär att den första kanalen innehåller nyansinformation, den andra mättnadsinformation och den tredje intensiteten, till skillnad från RGB där de tre kanalerna innehåller rött, grönt och blått. Detta gör det mycket enklare att filtera på färg, eftersom nyansen på ett objekt oftast är konstant medan nyansen och mättnaden ändras.

Den andra funktionen, $cvInRangeS$, filtrerar en bild genom sätta alla pixlar inom ett intervall till vita, och de som faller utanför till svarta. I detta program används fasta intervallvärden på mättnad och intensitet, och ett omfång runt ett användarinställt nyansvärde.

3.2.3 Programstruktur

När programmet startas initaliseras först nätverket, serieporten och kameran. Sedan skapas en separat tråd vars enda uppgift är att kontinuerligt skicka bilder till klientprogrammet. Bilden som ska skickas ligger i en separat variabel och skrivs över i slutet av varje loop i huvudtråden. Eftersom inga semaforer eller likande har använts finns en möjlighet att tråden försöker skicka bilden samtidigt som den skrivs till men då effekten av detta blir att användaren får så kallad "tearing", att övre halvan av bilden är uppdaterad och inte den undre, ansågs andra uppgifter som viktigare att implementera.

Programmets huvudloop, efter initialiseringar och trådstart, ser i stora drag ut så här i pseudokod:

```
//Fetch and convert images
image = getImage();
hsv_image = cvCvtColor(image, HSV);
tresh_image = cvInRangeS(hsv_image, lower_bound, upper_bound);

if instructionReceived()
    executeInstruction();
else if timeoutExceeded()
    stopRobot();
    waitForContact();

//Get moments
moments = getMoments(tresh_image);

//Get area and center of mass
area = getCentralMoment(moments);
pos = getCenterOfMass(moments);
radius = getRadius(area);

//Draw circle on image
drawCircle(pos, radius);

if automaticEnabled()
```

```
    calculateMovement(area, pos);

image_to_be_sent = selectImageToBeSent();
```

Särskilt funktionen *executeInstruction* utgör en stor del av koden, eftersom den utför de instruktioner som finnns specificerade i avsnitt 6.2. Beroende på instruktion kan den bland annat ändra interna inställningar, tex om automatiskt läge är aktiverat eller inte, ändra vilken bild som ska sändas och skicka rörelseinstruktioner till mikrokontrollern.

Även *calculateMovement* utgör en viktig del, då den använder en enkel PD-reglering för att räkna ut vilka styrvärden som ska skickas till mikrokontrollern beroende på objektets area och position. De olika reglervärdena är utvalda genom experimenterande tills de ansågs vara bra nog.

instructionReceived undersöker om det finns någon data tillgänglig på den socket som används för nätverkskommunikation. Om sådan finns undersöker den om det är en giltig instruktion (se avsnitt 6.2 för information om protokollet), och i så fall fortsätter programmet med att utföra denna instruktion.

4 Delsystem - Mikrokontroller

Detta delsystem utgörs av en mikrokontroller som är Arduino Nano-kompatibel, en statuslampa och en H-brygga för motorstyrning. Detta avsnitt kommer gå igenom hårdvaran och mjukvaran.

4.1 Hårdvara

Mikrokontrollern är en ATmega328P med en förinstallerad Arduino bootloader. Denna sitter monterad på ett blått kretskort med bland annat ett serieportschip och en mini-USB-port. Dessa saker i kombination med varandra gör att den kan programmeras av den officiella Arduino-programvaran, samt kommunicera med en dator genom en serieport över USB.

Statuslampan består av en ytmonterad RGB-diod, ett ytmonterat motstånd samt fyra kontakter. Dioden har en gemensam anod, så +5V ligger konstant på en av dessa kontakter. De tre andra kontakterna är katoder till de tre olika färgerna, och de är kopplade till tre olika pinnar på mikrokontrollern. Genom att skriva en logisk nolla till en pinne sluts kretsen och dioden lyser med den valda färgen. I programmet används endast en färg åt gången, men det är möjligt för dioden att lysa med upp till alla tre samtidigt.

H-bryggan behöver en extern strömkälla i form av fyra AA-batterier för att fungera, en hållare samt strömknapp för dessa är monterad på robotens bakdel. På varje sida av H-bryggan finns två utgångar som hjulparen kopplats till. Varje hjulpar är parallellkopplat till utgångarna. Det finns även fyra insignaler på H-bryggans framsida, där varje par styr varsitt hjulpar. Genom att skriva logiskt 01, 10 och 00 till varje par kan man få dem att åka åt båda riktningar eller stå stilla.

4.2 Mjukvarukrav

För att mikrokontrollern ska kunna programmeras krävs en Arduino-kompatibel mjukvara, tex. den officiella Arduino-programvaran som finns till flera plattformar. Det krävs även drivrutiner för serieporten på mikrokontrollerns kretskort. Dessa bör installeras automatiskt under Linux, och finns bifogade till Windows i filen *arduino_driver.zip*.

4.3 Funktionalitet

Mikrokontrollern tar emot instruktioner från Pin enligt protokollet beskrivet i avsnitt 6.3 och antingen flyttar roboten eller byter färg på statuslampan. Genom pulsbreddsmodulering till motorerna kan roboten fås att åka i olika hastigheter.

4.4 Programstruktur

Programkoden är skriven i den officiella Arduino-programvaran, som i princip använder C. Huvudloopen är i det närmaste trivial, då den konstant kontrollerar om det finns någon instruktion mottagen och i så fall exekverar denna, och skrivs därför inte med i den här rapporten.

De flesta instruktioner som ska exekveras är även de oftast triviala, eftersom de i de allra flesta fall går ut på att skriva de värden de fick i instruktionen till rätt utgångar på mikrokontrollern. När roboten ska flyttas tar den hänsyn till hastigheten framåt eller bakåt samt utslaget i sidled. Eftersom hjulparen är stumma och inte kan svänga modifieras därför hastigheten på hjulparen istället. När roboten svänger höger rör sig vänster hjulpar snabbare än höger och tvärtom vid vänstersväng. När roboten står stilla kan den svänga på stället genom att röra de olika hjulparen åt olika håll.

5 Delsystem - Klientmjukvara

Klientprogrammet är skrivet i C++ och fungerar i en Windows-miljö. Det förutsätter att en nätverksanslutning finns.

5.1 Hårdvara

Hårdvaran som krävs är en dator med Windows, samt en nätverksanslutning. Denna kan vara med kabel eller trådlös så länge det går att ansluta till roboten.

Vidare kräver mjukvaran ingen särskild hårdvara, men den har möjlighet att ansluta till en Xbox 360-kontroll om en sådan skulle finnas att tillgå.

5.2 Mjukvarukrav

För att kompilera programmet krävs att biblioteket OpenCV är installerat och länkat till Visual Studio-projektet. OpenCV finns att tillgå, med installationsguider, på `opencv.org`. På hemsidan finns även guider för hur man ställer in

och länkar OpenCV till Visual Studio, så hur det görs tas inte upp i det här dokumentet.

5.3 Funktionalitet

Mjukvarans huvudfunktioner är att visa en videoström från roboten samt att tillåta fjärrstyrning av denna. Användaren kan styra roboten med hjälp av tangentbordets piltangenter, bestämma vilken typ av videoström som ska visas med hjälp av knappar i mjukvaran, ställa i vilken färg som ska spåras samt känslighet på spårningen samt byta mellan automatiskt och manuellt läge. Mjukvaran kan även stänga av roboten.

5.4 Programstruktur

Programmet börjar med att användaren kan mata in en ny IP-adress till roboten om så önskas. Efter detta stängs kommandoradfönstret, variabelinitialiseringar görs, huvudfönstret öppnas och nätverket initaliseras. Därefter börjar huvudloopen, där en iteration beskrivs översiktligt nedan i pseudokod:

```
if windowClosed()
    exitProgram();

if isConnnected()
    if softTimeoutExceeded()
        sendPing(); //Let robot know the connection is active

    image = getImageIfExists()
    if isOneChannelImage(image)
        convertToThreeChannels(image);

else if isConnecting()
    tryToConnect();
    if isConnected()
        resetSettingsToDefault();

if controllerConnected
readController();

drawGUI();
copyVideoToGUI();
showGUI();

readKeyboard();
executeKeypresses();

if inManualMode()
```

```
if controllerConnected()
    sendControllerMovement();
else
    sendKeyboardMovement();
```

Eftersom både roboten och klienten antar att anslutningen bryts om ingen data mottagits efter tre sekunder har en intern mjuk timeout satts så att klienten skickar en instruktion efer 1,5 sekunder om inget annat skickats under den tiden. Detta meddelande fyller inget annat syfte än att låta roboten veta att anslutningen fortfarande är aktiv. Varje gång en instruktion skickas från klienten nollställs en timer, och i varje iteration av huvudloopen kontrolleras denna timer. Om den överstiger 1500ms skickas ett ping-meddelande till roboten.

Det grafiska gränssnittet innehåller ett antal knappar, som hanteras separat från denna huvudloop. Vid varje musinteraktion med huvudfönstret anropas en funktion, som beroende på vilken typ av musinteraktion som sker och muspekarens position utför vissa uppgifter.

Om ett dubbelklick sker och muspekaren är inom videområdet skickas den aktuella pixelpositionen i videon till roboten, som sedan ställer in färgspårningen till det värde denna pixel har.

Om ett enkelklick sker och muspekaren är över en knapp tolkas detta som ett knapptryck och relevant kod för knappen exekveras. Eftersom alla knappar är implementerade på det här sättet är det alltid denna funktion som kallas och som utför koden, förutom vid fönsterstängning.

5.5 Gränssnitt

Gränssnittet i detta program fungerar genom att ladda in en fast bakgrundsbild, se bild 2, i en OpenCV-struktur (*IplImage*) och sedan använda bibliotekets inbyggda funktioner för att skriva relevanta element över bilden. För de olika orangea indikatorerna för bildläge och styrläge ritas en ljusare orange färg ut över de indikatorer som är aktiva. Status- och gamepadindikatorerna skrivs vid behov över med en gul (endast status) eller grön rektangel. När roboten är ansluten skrivs videofältet över med den senaste bilden som mottagits från roboten.

Som beskrivits i avsnitt 5.4 kontrolleras muspekarens position i fönstret vid klick eller dubbelklick, och om den befinner sig över en knapp vid enkelklick räknas detta som ett knapptryck. Höjd, placering och bredd för alla knappar och andra GUI-element finns i filen *gui.h* och används av programmet för att kontrollera var muspekaren finns. Detta gör att man relativt enkelt kan ändra layouten på programmet genom att skapa en ny bakgrundsbild och uppdatera informationen i *gui.h* om så skulle önskas.

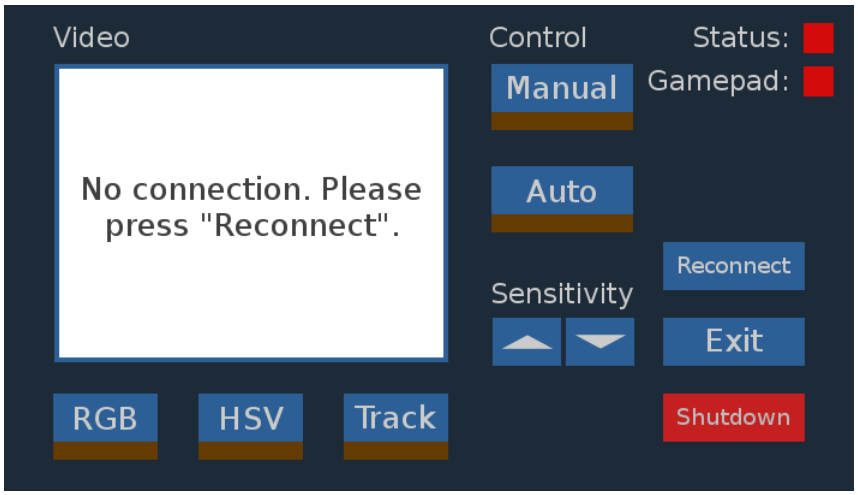

Bild 2: Det grafiska gränssnittets bakgrundsbild

6 Kommunikation

Kommunikationen mellan de olika systemen sker i huvudsak över två protokoll, som finns dokumenterade i detta avsnitt. Det sker även videoströmning separat från dessa protokoll.

All nätverkskommunikation mellan klienten och roboten sker över UDP. Detta är främst för att roboten kontinuerligt ska kunna skicka videodata utan att behöva vänta på en bekräftelse att klienten tagit emot denna. Detta innebär dock att det inte heller finns några garantier för att instruktionerna från klienten kommer fram, men under testning har detta inte visat sig vara några större problem så länge anslutningen är bra.

Kommunikationen mellan roboten och mikrokontrollern sker genom en serieport över USB. Då detta sker över en fysisk anslutning har den uppträtt robust under testning.

6.1 Videoströmning

Videoströmningen sker genom att roboten kontinuerligt skickar ut bilder på nätverket, ca 30 bilder per sekund. Dessa bilder skickas kompletta och okomprimerade över UDP, och tas emot av klienten. De skickade bilderna har storleken $160 * 120$ pixlar, och skalas på klientsidan upp till $320 * 240$ pixlar.

Alla bilder lagras i OpenCV-strukturen *IplImage*. Den innehåller bland annat information om bildens storlek, antal färger och antal färgkanaler. Bilddatan ligger sparad som en databuffer, med storleken bredd∗höjd∗kanaler bytes. Denna buffer är 57600 bytes stor, vilket turligt nog ryms i ett ensamt UDP-paket.

Istället för att skicka en hel $IplImage$ skickas endast denna buffer, och klienten har en "tom" bild med samma storlek och antal kanaler som den mottagna buffern kopieras till. Eftersom bilden i Track-läge bara har en kanal är buffern endast 19200 bytes stor när detta bildläge är aktiverat, och går inte att använda till en bild med tre kanaler då den väntar sig en större buffer. När bilden mottas kontrolleras antal mottagna bytes, och om det motsvarar storleken på en enkanalig bild används en tom enkanalig hjälpbild, som blir tilldelad databuffern som sedan blir konverterad till en motsvarande trekanalig bild som sedan används för uppritning på skärmen.

6.2 Nätverksprotokoll

Klientprogrammet kommunicerar med roboten genom ett egenskrivet protokoll över UDP. I detta protokoll kan en instruktion vara 16 byte långt, med en header och upp till 15 byte data per instruktion. Varje instruktion måste bestå av en header och mellan 0-15 databytes.

6.2.1 Headerstruktur

Headern beskriver vilken instruktion det är samt hur många databyte den innehåller. De första fyra bitarna i instruktionen beskriver instruktionstypen och de fyra sista bitarna antalet databytes. Detta innebär att man kan ha upp till 15 instruktioner där vardera kan ha upp till 15 databytes. På grund av denna struktur kan man med fördel skriva en header på hexadecimal form, där den första siffran betecknar instruktionstypen och den andra antal databytes. Till exempel kan headern för instruktion nr 10 med 5 databytes skrivas som A5 hexadecimalt.

6.2.2 Instruktionstyper

I tabell 1 nedanför listas alla instruktionstyper med deras (decimala) nummer, antal databytes och vad de utför för funktion.

ID	Namn	Data	Beskrivning
0	Stopp	0	Stoppar roboten
1	Kontakt	0	För att etablera kontakt mellan robot och klient. När roboten mottar denna signal skickas ett svarsmeddelande tillbaka för att indikera att kontakten fungerar.
2	Styrriktning	2	Innehåller vilka riktningar roboten ska röra sig i och med vilken hastighet. De två bitarna är signerade X- och Y-värden (X framåt/bakåt, Y höger/vänster)
3	Musposition	2	Innehåller positionsdata om var i bilden ett musklick görs. Den första byten innehåller x-värdet och den sista y-värdet. $1 \leq x \leq 160, 1 \leq y \leq 120$
4	Knappar	1	Skickas vid ett knapptryck i programmet. Innehåller information om vilken knapp som blivit nedtryckt (se tabell 2)
5	Ping	0	Används för att hålla kontakt med roboten
6	Bildläge	1	Ställer in om bilden ska visas i RGB-läge (0), HSV-läge (1) eller färgfiltrering (2).
7	Avstängning	0	Stänger av roboten.
$8 - 15$	Oanvända	-	Oanvända funktionsplatser. Användning av dessa kommer stoppa roboten.

Tabell 1: Detaljerad information om instruktionerna

Nedan följer tabellen för de olika knapparnas ID.

ID	Namn	Beskrivning
0	Manuellt	Sätter roboten i manuellt läge
1	Automatiskt	Sätter roboten i automatiskt läge.
2	Känslighet upp	Ökar känsligheten på färgigenkänningen
3	Känslighet ner	Minskar känsligheten på färgigenkänningen
$4 - 254$	Oanvända	Oanvända knappar.

Tabell 2: Detaljerad information om instruktionerna

6.3 Serieprotokoll

Raspberry Pin kommunicerar med mikrokontrollern genom ett egenskrivet protokoll över en serieport. I detta protokoll kan en instruktion vara 16 byte långt,

med en header och upp till 15 byte data per instruktion. Varje instruktion måste bestå av en header och mellan 0-15 databytes. Efter varje mottagen instruktion skickar mikrokontrollern ett svar som indikerar att instruktionen kommit fram korrekt eller vad som annars är fel.

6.3.1 Headerstruktur

Headern beskriver vilken instruktion det är samt hur många databyte den innehåller. De första fyra bitarna i instruktionen beskriver instruktionstypen och de fyra sista bitarna antalet databytes. Detta innebär att man kan ha upp till 15 instruktioner där vardera kan ha upp till 15 databytes. På grund av denna struktur kan man med fördel skriva en header på hexadecimal form, där den första siffran betecknar instruktionstypen och den andra antal databytes. Till exempel kan headern för instruktion nr 10 med 5 databytes skrivas som A5 hexadecimalt.

6.3.2 Instruktionstyper

I tabell 3 nedanför listas alla instruktionstyper med deras (decimala) nummer, antal databytes och vad de utför för funktion.

ID	Namn	Byte	Beskrivning
0	Stopp	0	Stoppar roboten och nollställer inställda riktningar
1	Hastighet	1	Anger hastigheten roboten ska förflytta sig med. Används i samband med de övriga instruktionerna. Datavärde mellan 0-255.
2	Framåt	0	Sätter färdriktning till framåt och flyttar roboten framåt med tidigare satt hastighet
3	Bakåt	0	Sätter färdriktning till bakåt och flyttar roboten bakåt med tidigare satt hastighet
4	Höger	1	Förflyttar roboten åt höger i tidigare satt hastighet och riktning. Databyten anger styrkraften mellan 0-255.
5	Vänster	1	Förflyttar roboten åt vänster i tidigare satt hastighet och riktning. Databyten anger styrkraften mellan 0-255.
6	Rotera hö.	0	Stannar roboten och roterar på plats åt höger i tidigare satt hastighet
7	Rotera vä.	0	Stannar roboten och roterar på plats åt vänster i tidigare satt hastighet
8	LED	1	Ändrar RGB-diodens färger. De sista tre bitarna i databyten styr färgerna i ordningen RGB. 1 tänder, 0 stänger av.
$9 - 15$	Oanvända	-	Oanvända funktionsplatser. Användning av dessa kommer stoppa roboten.

Tabell 3: Detaljerad information om instruktionerna

7 Instruktionssvar

Svaren från styrenheten består av endast en byte, som skickar en siffra binärt efter en mottagen instruktion. Denna siffra kan antingen betyda ”OK” eller beskriva vilket fel som uppstått. Eftersom en hel byte används finns det stöd för upp till 256 olika felkoder, men endast ett fåtal av dessa används.

ID	Beskrivning
0	Korrekt instruktion mottagen
1	Felaktig header mottagen
2	Felaktig data mottagen
3	För många byte data mottagna
4	För få byte data mottagna
$5 - 255$	Oanvända

Tabell 4: Detaljerad information om instruktionssvaren

8 Avgränsningar

I detta projekt har en del avgränsningar gjorts. Alla krav med prioritet 1 i kravspecifikationen är visserligen uppfyllda och även vissa med prioritet 2, men trots det har vissa avgränsningar fått göras.

Regleringen för att följa efter en boll har utvecklats tills den är ”bra nog”, eftersom det då var mer prioriterat att gå vidare med andra prioritet 1-krav. Även nätverkskommunikationen fick avgränsas, det gjordes ett försök att sända instruktioner och video på olika portar och över olika protokoll, TCP för instruktioner och UDP för video. Detta visade sig dock vara svårare än väntat och eftersom instruktioner förvånansvärt sällan tappades när endast UDP användes bedömdes det viktigare att gå vidare med andra krav.

Avgränsningar gjordes även för färgfiltreringen och bolligenkänningen. I nuläget antar roboten att den följer en boll med perfekt filtrering och räknar ut position och radie baserat på det. Detta för att det inte fanns tid nog att göra det bättre, till exempel genom att leta efter det största sammanhängade området i bilden och räkna på det så att ”skräpdata” minimeras. En annan möjlighet var att försöka identifiera runda objekt i bilden, men det visade sig vara för krävande för Raspberry Pin.

9 Vidareutveckling

Roboten har stor potential för vidareutveckling och förbättring. Varje delsystem är modulärt, så det är fullt möjligt att byta ut varje delsystem så länge det nya systemet följer kommunikationsprotokollet. Om man skulle byta ut motorerna eller dylikt borde dock regleringen ses över.

9.1 Förbättringar

En viktig förbättring som starkt rekommenderas att göras vid tillfälle är att se till att nätverkskommunikationen går över både TCP och UDP, på olika portar. Videon fungerar bra över UDP på grund av dess natur, det är inte kritiskt att allt går fram och det gör ingenting om några bildrutor tappas. Instruktionerna som

skickas bör dock ha en garanti att de faktiskt kommer fram, och borde därför skickas över TCP. Man skulle även på ett säkrare sätt kunna skicka information från roboten utöver videon. Ett försök gjordes med detta över UDP, men det var sällan informationen kom fram eftersom den blev tvungen att "samsas" med videon.

Även regleringen har utrymme för förbättring, och styrningen i allmänhet. När roboten åker i full fart och svänger sker detta genom att ett hjulpar saktas ned, vilket gör att robotens fart samtidigt minskar. Ett förslag på hur man kan förbättra detta är att inte låta roboten åka med full fart, så att det finns utrymme att justera hastigheten uppåt vid svängning.

Färg- och bollidentifieringen har stora utrymmen för förbättring. Till exempel kan man finjustera vilka områden som filtreras för att uppnå ett mer optimalt resultat, eller försöka identifiera det största sammanhängande området efter filtrering och enbart räkna på detta. Det skulle även gå att istället försöka använda kantdetektering för att hitta runda objekt. Ett försök gjordes med OpenCVs färdiga funktioner för cirkeligenkänning, men det visade sig att Raspberry Pin var för långsam för att hantera detta tillräckligt snabbt.

Nätverksadaptern som finns i roboten ska ha möjlighet till att skapa ett P2P-nätverk som sedan andra enheter kan ansluta till. Det var dock problem när detta skulle implementeras, det verkade som att drivrutinerna på något sätt var låsta. Detta är också något som skulle vara väldigt bra att ordna, eftersom man då kan slippa att behöva ansluta genom en router.

9.2 Tillägg

Det är fullt möjligt lägga till fler komponenter till roboten och mjukvaran, eller skapa ny mjukvara till andra plattformar.

Till exempel skulle mjukvara till Android kunna göras, eftersom OpenCV-biblioteket finns tillgängligt till Android. Man skulle då till exempel kunna styra roboten genom att luta telefonen i olika riktningar, eller bara genom en virtuell joystick på skärmen.

Eftersom Raspberry Pin har utgångar för ljud är det fullt möjligt att koppla in en högtalare, så att roboten kan spela upp olika ljud. Det är även möjligt att koppla in en mikrofon och installera röstigenkänningsmjukvara, så att den kan bli röststyrd.

Det skulle också vara möjligt att lägga till en Twitter- eller Instagramfunktion till roboten, så att man tex. på mässor kan ta bilder och ladda upp dem direkt till Twitter eller Instagram med ett knapptryck.

En annan möjlighet är att koppla in en andra kamera och ansluta roboten till en Oculus Rift, så man kan åka omkring och se videon i 3D.