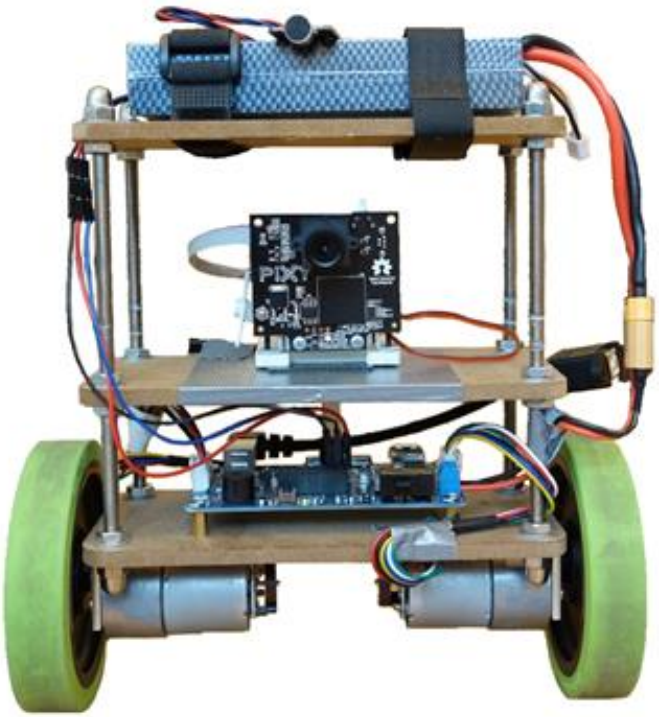
CHALMERS



Balanserande fotbollsspelande robotar

Kandidatarbete för SSYX02-16-13

Sara Boström
Edvin Eriksson Johansson
Fredrik Kjellberg
Rickard Larsson
Klas Lundgren
My Resare

Institutionen för Signaler och system
Chalmers tekniska högskola
Göteborg, Sverige 2016

Sammanfattning

Användningen av automatiserade robotar ökar i vårt digitala samhälle och robotfotboll är ett område där automatiserade robotar utvecklas. I utvecklandet av fotbollsspelande robotar ingår komplexa uppgifter i kodning, testning och felsökning. Det här projektet hade syftet att utveckla ett lag om tre fotbollsspelande robotar, med utgångspunkt i den färdigutvecklade hårdvaran Balanduino för balansering och CMU-kameran Pixy för seende och objektidentifiering. Hårdvaran programmerades med en beslutstagande enhet för att möjliggöra en intelligent fotbollsspelare som tar beslut beroende på situation. Robotarna testade sina färdigheter och beslutfattningsförmåga i en fotbollsmatch mot tre robotar från ett liknande projekt. Algoritmerna för robotarnas rörelser, färdigheter och beslutstagande enhet implementerades framgångsrikt. Det finns rum för förbättring av robotarna men matchen påvisade möjligheten att använda den här typen av robot och algoritmer i en riktig robotfotbollsturnering.

Abstract

The use of automated robots is increasing in our digital society and robot football is a field in which automated robots are researched and developed. The development of football playing robots includes complex tasks in coding, testing and debugging. The purpose of this project was to develop a team of three football playing robots. It started out with the predeveloped hardware Balanduino to enable balance and the CMU camera Pixy to enable vision and object recognition. The hardware was programmed with a decision making unit to make the robot an intelligent football player able to choose an action depending on the situation. The robots tested their skills and decision making in a football match against three similar robots from a similar project. The algorithms for the different movements, skills and decision making unit was implemented successfully. Improvements can still be made but the football game showed the potential for this type of robot and algorithms to be used in a real robot football tournament.

Innehåll

1 Inledning

Detta projekt går ut på att bygga ett fotbollspelande robotlag. Projektet grundar sig i att automation är ett snabbt växande område inom teknik. I bakgrunden nämns hur intresset syns i samhället och hur man försöker sprida vetenskapen. Specifikationer för projektet definieras med hjälp av spelregler och avgränsningar, sedan finns en tydlig uppgift att utföra.

1.1 Bakgrund

Tekniken som används inom robotfotboll sträcker sig utanför ramarna för sporten. Robotar börjar bli allt vanligare både på arbetsplatser och i hemmet eftersom de kan utföra uppgifter som är besvärliga, svåra eller rent av farliga för människor, [1]. För att kunna utföra dessa uppgifter autonomt måste roboten kunna interagera med sin omgivning, i många fall görs detta med hjälp av objektidentifiering, [2]. Med ett ökande intresse för robotar och en växande marknad är intresset att förbättra sig inom området stort. Detta tyder på att autonoma robotar med objektidentifiering, likt robotar inom robotfotboll, kommer att ha stort inflytande i framtiden och att det är viktigt att fortsätta arbeta med utvecklingen av kunskaperna inom området.

Det var med målet att väcka ett intresse hos lekmän för teknologin och vetenskapen bakom robotar som FIRA, *Federation of International Robot-soccer Association*, startade en turnering i robotfotboll år 1996, [3]. Det finns olika turneringar där många länder engagerar sig i ett flertal olika klasser. Det finns klasser för humanoida robotar, det vill säga robotar som är människoliknande, simulering av robotar samt fullt eller delvis autonoma robotar. Solc och Honzík, [4], skriver om sitt bidrag till klassen MiroSot (Mlcro Robot Soccer Tournament), som är ett exempel på en klass för autonoma robotar i storleksordningen liten.

Mycket forskning bedrivs inom ämnet, vilket grundar sig i många olika tekniska och vetenskapliga problem. Det kräver reglersystem, mekaniska och elektriska lösningar, något form av synsystem samt kod för beteende, [4]. Enligt Chen och Liu, [5], behöver en smart fotbollsspelande robots beteende vara uppbyggt på tre egenskaper: färdigheter såsom att kunna sparka, kunskap om regler och en beslutsfattande enhet. Den beslutsfattande enheten är den viktigaste egenskapen för att den avgör vilka av färdigheterna som ska användas och vilka regler som gäller i olika situationer. Pratomo et al. [6], har fokuserat på att ta fram algoritmer för positionering och att undvika objekt, två användbara och viktiga färdigheter för en fotbollsspelande robot.

Detta studentarbete kommer att behandla en autonom robot som använder sig av reglerteknikens klassiska princip den inverterande pendeln samt objektidentifiering med en kamera. Projektet kommer att bygga vidare på ett tidigare kandidatarbete från år 2015, [7], som hade målsättningen att konstruera en tvåhjulig fotbollspelande robot från grunden. Deras robot kunde dock inte balansera när projektet var klart. Därför kommer detta projekt att fortsätta där det tidigare avslutade genom att vidareutveckla en Balanduino, [8], en självbalanserande robot med öppen källkod, [9]. Denna ska integreras med en CMUCam5-kamera av

modellen Pixy, [10], som också har öppen källkod. Roboten kommer falla under FIRA:s kategori för autonoma robotar i medelstorlek, RoboSot.

1.2 Syfte och Mål

Syftet med projektet är att skapa tre fotbollsspelande robotar genom att vidareutveckla Balanduino-plattformen och att utrusta robotarna med varsin Pixy-kamera för objektidentifiering.

Målet med projektet är att de tre fotbollsspelande robotarna ska vara fullt autonoma med undantag av en fjärrkontrollerad på/av- styrning. En fotbollsmatch ska anordnas för att presentera resultatet och verifiera att kravspecifikationerna har uppnåtts. Motståndarlaget ska konstrueras i ett annat kandidatarbete som utförs parallellt av en annan grupp.

1.3 Problem och Uppgift

För att möjliggöra att robotarna ska klara av att spela fotboll autonomt utan alltför mycket avbrott behöver framförallt följande problem lösas.

En fotbollsspelande robot måste klara av att:

- *Utföra grundläggande rörelser*
  Detta innefattar att balansera, köra framåt och bakåt samt att rotera.

- *Tolka information och ta beslut därefter*
  Roboten kan tolka och lagra information om objekt som den har i sitt synfält. Objekten den ska kunna känna igen är boll, motspelare, medspelare, mål och planens kanter. Den ska veta positionerna för objekten relativt sig själv och baserat på det ta beslut om vad den ska göra.

- *Göra mål*
  För att göra mål behöver roboten kunna räkna ut hur den ska ta sig till skottläge. När den är i skottläge måste den kunna ta sig fram till bollen och sparka på den.

- *Tillämpa självständigt spel*
  Roboten behöver vara relativt självgående. Den ska med andra ord inte krocka med några andra objekt på planen eller planens kant och den ska jobba hem när den behöver försvara sitt eget mål.

1.4 Spelregler

Reglerna har sin utgångspunkt i FIRA:s regler [11] för robotfotboll i klassen RoboSot [12]. I synnerhet ska projektet följa dessa regler:

- Robotarna får inte överskrida storleken 35 cm $\times$ 35 cm (längd $\times$ bredd). Höjden är obegränsad.

- Robotarna ska vara helt autonoma och utan central styrning.
- Robotarna får kommunicera med varandra. Enda yttre kommunikationen som får ske ska vara i syfte att starta eller stoppa robotarna.
- Fysisk kontakt med motståndarrobotarna med avsikt att tackla är inte tillåtet.
- Varje lag består av tre spelare, varav en är designerad som målvakt.

Utöver dessa regler kom grupperna överens om ett antal gemensamma regler som skiljer sig från FIRA:s regler:

- Bollen ska vara orange och cirka 15 cm i diameter.
- Robotarna får inte ha några utstickade atrapper med syfte att hålla fast bollen.
- Spark sker genom att robotarna 'knuffar' på bollen. Ej med någon annan mekanism.
- Målen ska vara färgkodade med måtten 100 cm $\times$ 40 cm (längd $\times$ bredd).
- Spelarna på planen ska ha 'lagtröjor' som täcker så mycket av roboten som möjligt.
- Det ska finnas en 10 cm hög sarg runt planen markerad med färgkod som endast syns på nära håll. Sargen ska ha rundade hörn för att bollen inte ska fastna i hörnen.
- Spelplanen kommer inte byggas enligt FIRA:s regler om mått, den kommer vara mer anpassningsbar för att stämma överens med området som finns att tillgå vid slutpresentationen.
- Om en robot välter ska den tas av planen direkt och får sättas tillbaka in i spel efter minst 10 s på en plats där den ej stör pågående spel.

Vid eventuella oklarheter och för saker ej specificerade ovan gäller FIRA:s regler för RoboSot.

1.5 Avgränsningar

Projektet avgränsas till att utgå från en redan konstruerad robot (Balanduino) med befintlig balanseringsoptimering och en kamera (Pixy) med färdig mjukvara för objektidentifiering. Detta görs för att lägga fokus på fotbollsspelandet istället för att bygga allt från grunden.

FIRA:s regler tillåter intern kommunikation mellan robotarna. Detta används dock inte i projektet eftersom det inte hittades några bra och enkla lösningar för det samt att det inte var nödvändigt för att få till ett fotbollsspel.

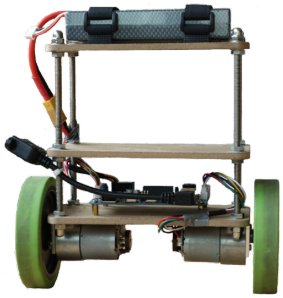
Budgeten för projektet, se appendix B, är 5000 SEK exklusive tre Balanduino som tillhör institutionen för Signaler och system på Chalmers tekniska högskola.

2 Robotkonstruktion

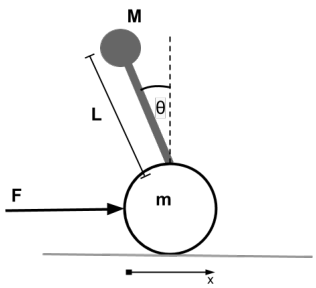
Hårdvaran består av redan utvecklade produkter som har kombinerats och modifierats för att uppfylla mål och syfte. Arbetet är till största del mjukvarubaserat men en introduktion till hårdvaran är nödvändig för att förstå hur produkten har utvecklats.

2.1 Balanduino

Balanduino, se figur 1, är en balanserande robot som bygger på fysiken i en inverterad pendel. Den inverterade pendeln, se figur 2, är ett klassiskt problem inom reglertekniken där pendelns masscentrum är ovanför fästpunkten vilket gör den instabil.



Figur 1: Balanduino-plattformen från TKJ Electronics



Figur 2: En inverterad pendel

För att balansera pendeln måste fästpunktens position anpassas i förhållande till pendelns vinkel, så att pendeln hålls upprätt. Detta görs med hjälp av re-

gulatorer som bestäms när ett uttryck för hur de fysikaliska storheterna i en inverterad pendel förhåller sig till varandra. Detta förhållande tas fram genom att alla ingående krafter ritas ut efter Newtons 2:a lag. Sedan uttrycks krafterna i mån av massa, acceleration och övriga parametrar i figur 2, detta ger

$$-mg \cdot sin(\theta) = m\ddot{x} \cdot cos(\theta) - mL\ddot{\theta} \tag{1}$$

$$F + mL\ddot{\theta} \cdot cos(\theta) - mL\dot{\theta}^2 \cdot sin(\theta) = (m + M)\ddot{x} \tag{2}$$

där variablerna är från figur 2 och där pendelns massa, m, och vagnens massa, M, anges i $[kg]$. Vinkeln θ anges i $[rad]$. Pendelns längd, L, anges i $[m]$ och dess vinkelhastighet, $\dot{\theta}$, anges i $[rad/s]$ och dess acceleration, $\ddot{\theta}$, anges i $[rad/s^2]$, g är jordens tyngdacceleration och anges precis som vagnaccelerationen $\ddot{x}$ i $[m/s^2]$. Balanduinon använder en PID-regulator där insignalen är kraften F som behövs för att reglera utsignalen θ till 0 radianer.

Balanduino får vinkelhastigheten och kraften som regulatorn är designad efter via en 6-axlig IMU (Inertial Measurement Unit). Kraften mäts via en accelerometer som består av tre axlar och vinkelhastigheten mäts med ett 3-axligt gyroskop. Ett Kalmanfilter används för att kontinuerligt uppskatta robotens vinkel gentemot marken utifrån mätvärdena från IMU:n. Kalmanfiltret är ett rekursivt filter som uppskattar tillståndet hos ett dynamiskt system utifrån en mängd inkompletta mätningar för att bland annat minska påverkan av mätbrus.

På undersidan av Balanduinon sitter en rotationskodare på varje hjul som räknar antalet magnetiska pulser och stegar uppåt eller nedåt beroende på åt vilket håll robotens hjul roterar.

2.2 Arduino

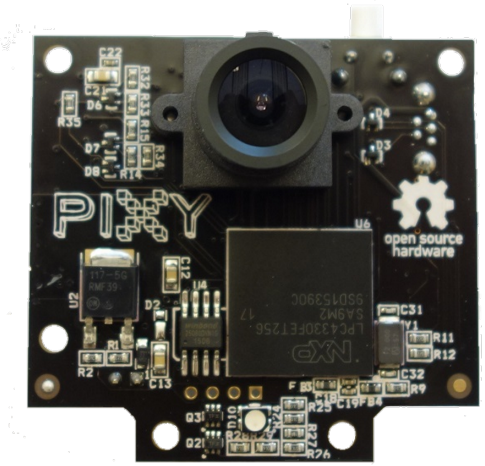
Balanduino är kompatibel med Arduino IDE (Integrated Development Environment) då dess mikrokontroller är en ATmega1284P från Atmel. Arduinoplattformen kan beskrivas som en enkel dator med öppen källkod. Det är ett kretskort med integrerade kretsar, ingångar och utgångar för matningsspänning och jord. Det finns både analoga och digitala ingångar och utgångar. Arduino används med fördel i autonoma robotar eftersom beräkningar behöver utföras i realtid och en Arduino har den nödvändiga beräkningsförmågan och är i en passande storlek för att få plats på en mindre robot.

2.3 CMUcam5 Pixy

För att identifiera objekt används en CMU-kamera, [13], kallad Pixy som visas i figur 3. Förkortningen CMU kommer ifrån namnet Carnegie Mellon University som var det universitet som utvecklade den första modellen av denna kameratyp. Syftet var att möjliggöra enklare kommunikation till robotar genom att konstruera en kamera som integreras med en mikrokontroller. En CMU-kamera är definitionen av en kameraklass utrustade med en processor som sköter bildbehandling internt. Denna interna mikroprocessor stödjer enkel bildbehandling

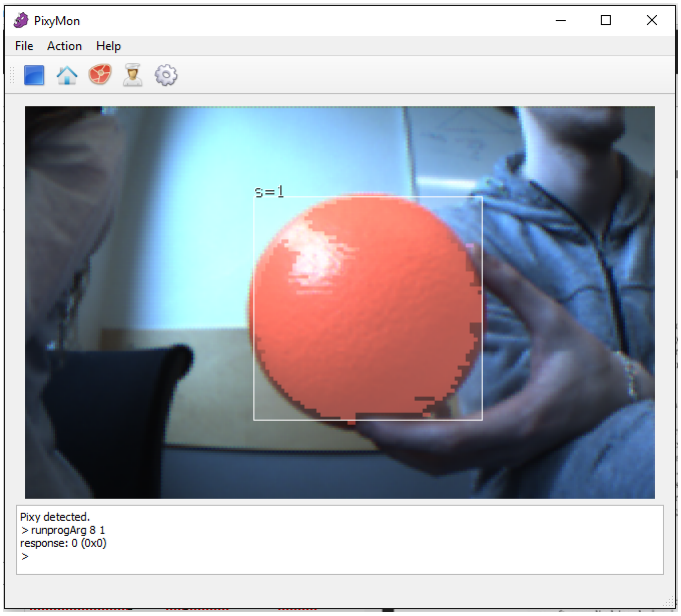
och så kallad *blob tracking*. Med det menas att områden i synfältet med liknande egenskaper i till exempel ljusstyrka eller färger kan förknippas till ett objekt.

Kameran har ett synfält på 75°, [14], och kan lära sig att identifiera objekt i både olika färgspektrum och geometriska former. Den kan identifiera sju färger: rött, orange, gult, grönt, blått, indigo och violett. Detta används för identifiera och urskilja olika objekt på spelplanen.



Figur 3: En bild på CMUCam5 Pixy.

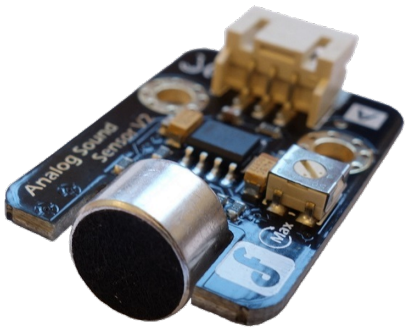
Pixy har ett tillhörande konfigureringsverktyg, PixyMon. I PixyMon kan olika parametrar som ljusstyrka, färgkänslighet och antal block justeras för att hitta ett objekt så effektivt som möjligt. PixyMon gör även att det är enkelt att lära Pixy-kameran nya objekt i olika ljusförhållanden. I figur 4 syns tydligt hur Pixy-kameran har identifierat en orange boll och tilldelat den en *signatur*, s = 1. Genom att ange signaturer för olika objekt kan man således urskilja och identifiera dem. Om objektet består av två eller fler färger rapporter det med det oktala talsystemet som en så kallad ColorCode och får en tvåsiffrig signatur.



Figur 4: En bild från gränssnittet i PixyMon där bollen har tilldelats en signatur.

2.4 Mikrofon

Eftersom det är nödvändigt att starta roboten för att den ska kunna balansera behövs en fjärrkontrollerad på/av-funktion för att samtliga robotar ska gå ur sitt viloläge samtidigt så att en match kan ske under kontrollerade former. För att lägga till denna funktion används en mikrofon. Mikrofonen är en analog ljudsensor från DFROBOT, [15], som mäter ljudintensiteten och skickar enhetslös data till Balanduinon. Om ljudintensiteten uppnår en viss nivå aktiveras styrfunktionen hos roboten och den går ut sitt viloläge.

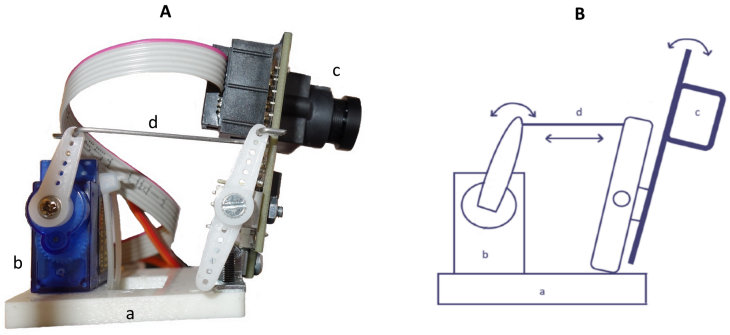


Figur 5: En bild på mikrofonen från DFROBOT.

2.5 Kameraställning för Pixy

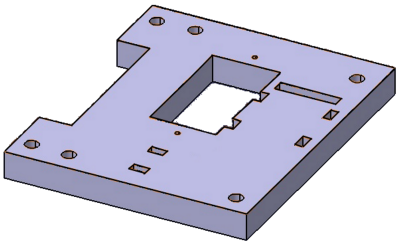
För att utöka kamerans begränsade synfält och för att underlätta objektidentifieringen när objekten är nära, är roboten utrustad med en ställning som rör sig i vertikal led. Anordningen består av en servomotor monterad på en platta med en länkarm mellan servot och kameran, se figur 6.

När ett objekt på golvet kommer nära roboten skymmer själva roboten objektet. När det sker aktiveras servot och kameran sänks ner vilket gör så att objektet alltid har samma bredd enligt Pixy. Detta gör att roboten aldrig kommer att tappa bort ett objekt även på nära håll.



Figur 6: **A:** En bild på ställningen och **B:** en 2D-ritning på ställningen med pilar som visar hur de olika komponenterna rör på sig, där a är plattan, b är servot, c är kameran och d är länkarmen.

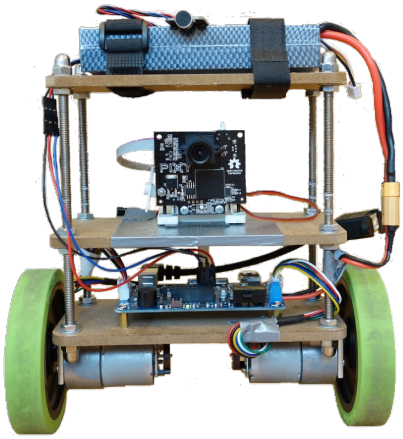
Kameraställningens platta, a i figur 6, är 3D-printad efter en egendesignad CAD-ritning och används för att optimera stabiliteten, se figur 7.



Figur 7: CAD-ritningen av plattan ritad i CATIA

2.6 Slutkonstruktion

För att konstruera den färdiga roboten krävs inga speciella verktyg eller metoder, då komponentera fungerar som självständiga moduler. Det som görs för att fästa kameran är att den skruvas fast i den 3D-printade plattan som monteras på Balanduinon med dubbelhäftande tejp och att Pixy kopplas in genom en medföljande Arduino kabel. Mikrofonen monteras högst upp på roboten för att minimera störningar i form av ljud från motorerna. Slutresultatet av detta är den färdiga roboten som visas i figur 8.



Figur 8: Den slutgiltliga roboten när alla komponenter är monterade.

3 Matematiska modeller

En viktig del i robotens beteende är mer avancerade beräkningar istället för enbart villkorsbaserad programmering. Dessa beräkningar görs främst för att beräkna avstånd mellan objekt och för att beräkna eventuella planerade banor på planen. Beräkningarna bygger på, för detta projekts syfte, framtagna och modifierade matematiska modeller.

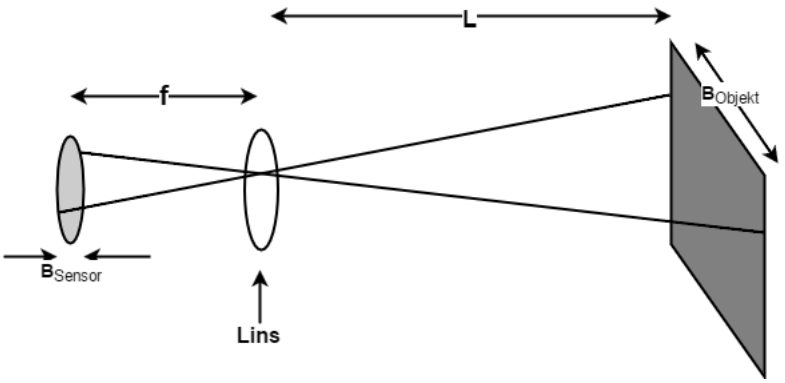
3.1 Avståndsberäkning mellan robot och objekt

Ett numeriskt värde på avstånd är av stor vikt vid många funktioner och beslutsfattande moment som roboten skall utföra. Avståndet som avses är avstånd från robot till olika objekt. Därav krävs en matematiskt modell för att beräkna avstånd som kan tillämpas på objekt av olika storlekar och utformning.

En matematiskt beräkningsmodell, [16], som tar hänsyn till kamerans brännvidd, sensorns storlek, objektets verkliga storlek och även storheter i pixlar som kameran producerar har använts. Avståndet beräknas enligt

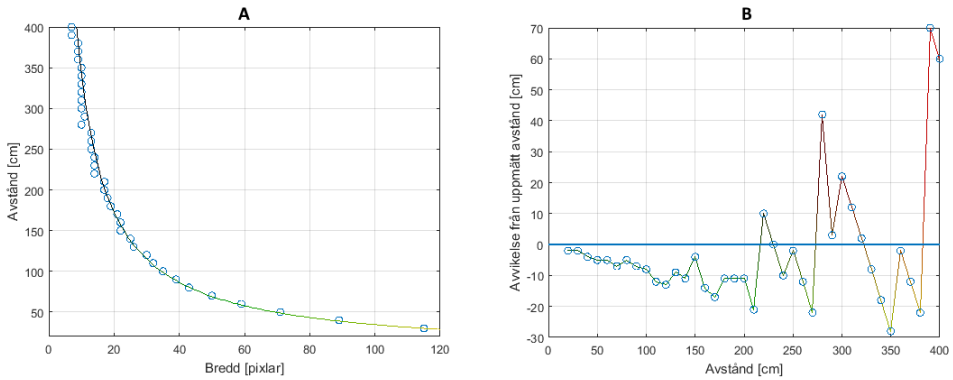
$$L = \frac{f \cdot B_{Objekt}}{B_{Sensor}} \cdot \frac{B_{Bild}}{B_{Pixlar}} \tag{3}$$

där avstånd L $[cm]$, brännvidd f, verklig bredd B_{Objekt} och sensorns bredd B_{Sensor} anges i $[mm]$ och bildens bredd B_{Bild} samt objektets bredd B_{Pixlar} anges i $[pixlar]$. Funktionens storheter finns illustrerade i figur 9.



Figur 9: Illustration av storheterna från ekvation 3.

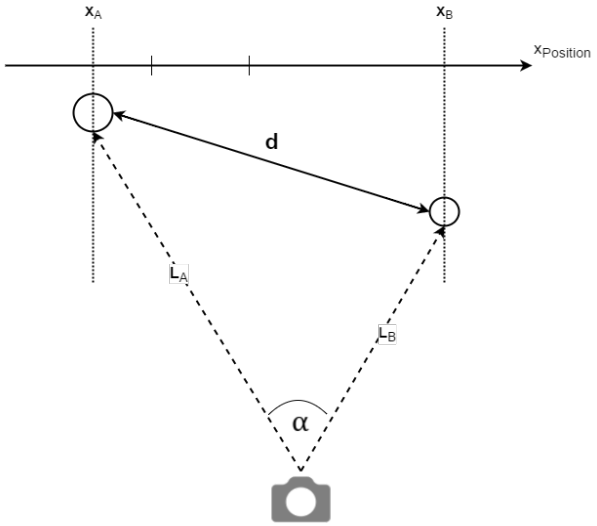
Denna teoretiska metod kontrollerades genom att ett antal punkter i det intressanta intervallet uppmättes för att sedan kurvanpassas manuellt, se figur 10. Som referens är cirklarna värden som mättes upp för hand. All data motsvarar avståndet till centrum av matchbollen med en diameter på 15 cm. Samma tester utfördes för medspelare, motspelare och målen för att säkerställa att teorin stämmer överens med verkligheten. Samtliga tester med resultat finns under kapitlet *Tester, Resultat och Analys.*



Figur 10: **A:** Avståndet till boll beräknat med ekvation 3 där cirklarna representerar mätvärden. **B:** Storleken på avvikelser från mätvärden.

3.2 Avståndsberäkning mellan objekt

Med avståndet mellan roboten och objektet till hands går det med hjälp av cosinussatsen att räkna ut avstånd mellan objekt som roboten har i sitt synfält. För att beräkna avståndet d $[cm]$ måste roboten hitta vinkeln α mellan objekten, se figur 11.



Figur 11: En figur som illustrerar utgångspunkten när avståndet mellan objekt, d, ska beräknas.

Pixy kan inte själv hitta vinkeln mellan objekt vilket medför att matematiska beräkningar krävs som baseras på att skala Pixy-kamerans kända synfält med antalet pixlar kameran ser. Eftersom Pixy har ett horisontellt synfält på 75° (vilket blir $\frac{5\pi}{12}$ radianer) och en bildbredd på 320 pixlar kan vinkeln mellan två objekt approximeras genom att räkna pixlarna och använda

$$\alpha = \mid X_A - X_B \mid \cdot \frac{75^\circ}{320 pixlar} \tag{4}$$

där X_A och X_B är objektens placering i horisontella planet på bilden, α $[radianer]$ är vinkeln. När α är beräknad kan man med hjälp av avstånd och trigonometriska satser identifiera fler användbara parametrar. Därefter beräknas avståndet d $[cm]$ med hjälp av cosinussatsen

$$d^2 = L_A^2 + L_B^2 - 2 \cdot L_A \cdot L_B \cdot cos(\alpha) \tag{5}$$

där L_A och L_B $[cm]$ är avståndet mellan kamera och objekt. Med hjälp av avståndet mellan objekten kan roboten fatta beslut utifrån hur nära andra robotar är till bollen eller hur nära bollen är målet.

3.3 Röra sig en bestämd färdväg

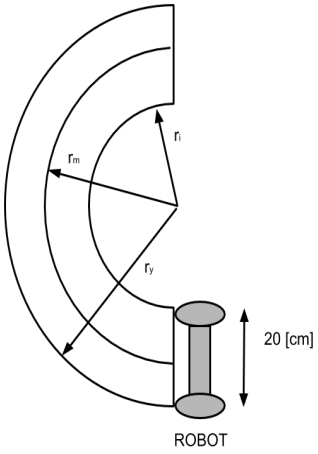
Vinkelgivarna på roboten möjliggör egenskapen för roboten att åka en bestämd distans och med en bestämd radie. Så som roboten är konstruerad motsvarar ett varv på hjulet ett utslag på 928 pulser i varje givare. Omkretsen på hjulen är 30.9211 [cm] vilket ger formeln för längden på färdvägen att motsvara

$$distance = \frac{pulses \cdot O}{P \cdot 2} \tag{6}$$

där *distance* är längden som roboten ska färdas [cm], *pulses* motsvarar antalet utslag på vinkelgivarna, O motsvarar hjulens omkrets [cm], P är antalet pulser rotationskodaren ger per varv och multiplikationen med två i nämnaren kommer från att båda vinkelgivarnas utslag kommer att summeras eftersom båda hjulen roterar åt samma håll vilket gör att ett varv totalt motsvarar 1856 pulser när roboten kör rakt framåt eller bakåt.

3.3.1 Åka i en bestämd radie

När roboten åker i en cirkelbåge kommer det yttre hjulet att färdas längre och det inre hjulet färdas en kortare sträcka. Om robotens mittpunkt motsvarar en radie som kallas r_m enligt figur 12 kan den yttre och inre radien beräknas med hjälp av robotens kända bredd 20 cm som är avståndet mellan hjulen.



Figur 12: Här visas olika radier som inner respektive ytterhjulet kommer färdas i då roboten åker en radie baserad på dess mittpunkt.

Ytterradien kan då beräknas till

$$r_y = r_m + 10 \tag{7}$$

där längderna anges i [cm] och 10 cm kommer från halva robotens bredd som blir avståndet mellan Balanduinons centrum till hjulet. På samma sätt kan innerradien beräknas enligt

$$r_i = r_m - 10 \tag{8}$$

För att kunna ta fram en skalfaktor som varje hjuls hastighet ska multipliceras med måste skillnaden i distans tas fram. Omkretsen på cirkeln som motsvarar hur långt den sidan av roboten har förflyttat sig när hjulet roterat ett varv ger att skalfaktorn för det yttre hjulet kommer att bli

$$k_y = \frac{r_y}{r_m} \tag{9}$$

där k_y är faktorn som det yttre hjulets hastighet kommer att multipliceras med. Denna funktion kan förenklas med vetskap om robotens bredd, till att endast bero på robotens medelpunktsradie

$$k_y = \frac{r_m + 10}{r_m} \tag{10}$$

där den bestämda radien r_m anges i [cm].

Samma metodik används för beräkning av skalfaktorn för innerhjulet enligt:

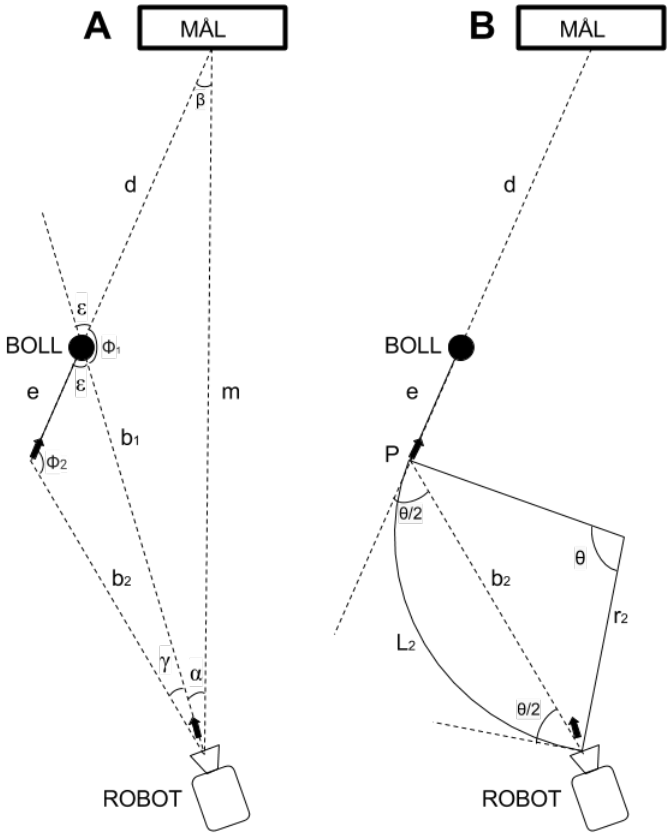
$$k_i = \frac{r_m - 10}{r_m} \tag{11}$$

där k_i är skalfaktorn för innerhjulets hastighet. En sträcka kan inte vara negativ men en negativ skalfaktorn motsvarar att hjulen roterar åt motsatt håll, roterar de med lika stor fart resulterar det i att roboten roterar på plats med radie 0 cm.

3.4 Ställa sig i skottposition

För att roboten ska ha någon chans att göra mål måste den kunna ställa sig i linje med boll och mål för att kunna sparka åt rätt håll. Om roboten inte bryr sig om att komma i rätt position kommer den att sprida sina skott över hela planen. För att reglera detta så att roboten ställer sig i en bättre position innan den skjuter kommer vinkeln mellan boll och mål relativt Balanduino-plattformens position att avgöra om den måste göra en ompositionering.

Med hjälp av avståndsberäkningar, trigonometri och cirkelns geometri kan en bana räknas ut som kommer ta roboten till en bestämd position och till en bestämd riktning. Den planerade positionen sätts till att vara en bit från bollen i den fortsatta linjen mellan mål och boll där roboten är riktad för ett skott mot mål vilket syns i figur 13. Anledningen till det bestämda avståndet e mellan bollen och den planerade positionen P är för att tillåta bollen att röra sig en aning under tiden roboten färdas den uträknade sträckan. Då kan roboten kolla att den befinner sig i ett tillräckligt bra läge efter sin rörelse för att sedan påbörja satsen inför skottrörelsen en bit från bollen. Vägen roboten färdas är baserad på att roboten roterar en bestämd vinkel från utgångsläget och sedan tar sig till rätt position genom att åka i en bestämd cirkelbåge.



Figur 13: **A:** Trianglar som behöver sättas upp för att beräkna viktiga storheter. **B:** Cirkelbågen som illustrerar den planerade färdvägen till punkten P.

Det framgår i figur 13 att vinklarna α, ϕ_1 och ϕ_2 är viktiga för att avgöra om denna matematiska modell kommer gå att använda. Vinkel α kommer att avgöra huruvida roboten kommer behöva röra sig åt höger eller vänster för att placera sig rätt. Spetsiga vinklar på ϕ_1 och ϕ_2 indikerar att bollen befinner sig närmare egen planhalva än roboten vilket kan medföra att roboten först behöver jobba hem, som det kallas, innan denna funktion kan användas för att placera sig i skottläge. Den högra triangeln i bilden baseras på objektens verkliga positioner. Den lite mindre vänstra triangeln i bilden skapas då målpunkten P flyttas en sträcka e från bollen.

Längden m är avståndet mellan robot och mål $[cm]$, längden b_1 är avståndet mellan robot och boll $[cm]$, båda dessa avstånd mäts av roboten med tidigare nämnd avståndsberäkning. Längden d är avståndet mellan boll och mål $[cm]$, beräknat enligt cosinussatsen som tidigare, se ekvation 5.

Vinkel α är uttagen tidigare av roboten enligt ekvation 4. Vinkel β är vinkeln mellan boll och robot relativt målet. Med dessa variabler förklarade kan nu sinussatsen för triangeln ställas upp enligt

$$\frac{\sin(\phi_1)}{m} = \frac{\sin(\beta)}{b_1} = \frac{\sin(\alpha)}{d} \tag{12}$$

vilket sedan ger ett känt värde för vinkeln ϕ_1 enligt

$$\phi_1 = \arcsin(\frac{m \cdot sin(\alpha)}{d}) \tag{13}$$

Dock kan sinussatsen ge två olika fall vilket måste tas hänsyn till i dessa beräkningar. Av anledningen att kamerans synfält på 75° medför att vinkeln α är en spetsig vinkel samt att vinkel β alltid är spetsig på grund av att bollen inte kan befinna sig bakom mål kommer alltid vinkel ϕ_1 att vara trubbig i de fall då både boll och mål är synliga samtidigt för roboten. Teorin för sinussatsens två fall tillämpas och den trubbiga vinkeln fås enligt

$$\phi_{1,trubbig} = \pi - \phi_{1,spetsig} \tag{14}$$

där vinklarna är angivna i radianer, $\phi_{1,spetsig}$ motsvarar den nyss beräknade ϕ_1 och $\phi_{1,trubbig}$ betecknar den verkliga vinkel som ska tas fram. Vinkel $\phi_{1,trubbig}$ kommer hädanefter endast betecknas som ϕ_1 för enkelhet. Enligt figur 13 framgår att vinkeln ϵ beräknas enligt

$$\epsilon = \pi - \phi_1 \tag{15}$$

där ϕ_1 och ϵ är vinklar angivna i radianer. Med två kända längder b_1 och e samt en känd vinkel ϵ i den vänstra triangeln kan nu avståndet b_2 till den tänkta punkten P beräknas med cosinussatsen enligt

$$b_2 = \sqrt{(b_1)^2 + (e)^2 - 2 \cdot b \cdot e \cdot \cos(\epsilon)} \tag{16}$$

där alltså avståndet e kan varieras beroende på hur långt ifrån bollens nuvarande position man vill placera sig i rätt vinkel. Längden b_2 motsvarar en korda för cirkelbågen som roboten kommer att färdas. Med cosinussatsen uppställd för den vänstra triangeln igen kan nu vinkeln ϕ_2 beräknas enligt

$$\phi_2 = \arccos(\frac{(e)^2 + (b_2)^2 - (b_1)^2}{2 \cdot e \cdot b_2}) \tag{17}$$

där samma antagande om att vinkeln ϕ_2 alltid är trubbig inte kan användas här. Om längden e sätts till ett längre avstånd eller om avstånden b_1 och b_2 är små kan det medföra att summan av vinklarna γ och α här kan bli en trubbig vinkel vilket i sin tur gör att ϕ_2 i den vänstra triangeln nu kan bli spetsig. Denna metodik har visat sig fungera i praktiken för sådana vinklar ϕ_2 som inte är allt för spetsiga.

b_2 betraktas som kordan för en cirkel som skär punkten där roboten befinner sig och punkten P dit roboten ska vilket skapar en cirkelbåge mellan dessa punkter. Vid skärningspunkten P tangerar cirkelbågen linjen mellan boll och

mål. Detta skapar en cirkelsektor med radien r_2 och vinkeln θ, se figur 13. Teorin för cirkelbågens geometri tillför modellen att roboten behöver rotera vinkeln $\theta/2$ för att positionera sig i rätt vinkel för utgångsläget, om den som utgångspunkt först tittade rakt mot boll.

För att komma till rätt slutvinkel kommer roboten dels behöva rotera tillbaka denna vinkel $\theta/2$ samt även rotera vinkeln ϵ för att hamna i önskad slutvinkel. Denna rotation görs när roboten åker längs cirkelbågen och algoritmen

$$\theta = \epsilon + \frac{\theta}{2} + \gamma \tag{18}$$

fås fram där vinkeln ϵ alltså motsvarar hur mycket roboten hade behövt vrida upp sig mot mål från början innan rotationen och θ är cirkelsektorns vinkel, i radianer. Lösning för θ ger

$$\theta = 2 \cdot \epsilon + 2 \cdot \gamma \tag{19}$$

vilket nu har gett oss tillräckligt med information om cirkelsektorn för att räkna ut radien r_2, och längden på sträckan L_2.

Teori för kordans längd ger sambandet

$$r_2 = \frac{b_2}{2 \cdot \sin(\frac{\theta}{2})} \tag{20}$$

vilket tillåter oss att räkna ut sträckan

$$L_2 = r_2 \cdot \theta \tag{21}$$

som är sträckan roboten kommer att färdas till bollen.

Denna matematiska modell har förklarats då robotens kamera är riktad mot bollen i utgångs-läget. Om de intressanta objekten inte befinner sig mitt i synfältet beräknas vinkeln mellan t.ex. boll och mitten av synfältet på samma sätt som vinkeln α tidigare och adderas till eller subtraheras från rotationen $\theta/2$ för att roboten ska positionera sig i rätt vinkel innan den börjar åka mot boll.

4 Mjukvara

I mjukvaran byggs robotens hjärna. Det är här strategin för att bete sig som en fotbollsspelare utformas. I mjukvaran implementeras också alla grundläggande funktioner som behövs för att spela fotboll såsom att förflytta sig, sparka bollen och identifiera objekt. De matematiska modellerna från avsnitt 3 används tillsammans med de grundläggande rörelserna för att utföra mer avancerade funktioner. Utgångspunkten för mjukvaran var den öppna källkoden som redan fanns för Balanduino och Pixy. Källkoden för projektet finns på GitHub, [17].

4.1 Klasserna i mjukvaran

De två viktigaste klasserna i programmet är *Motor* och *Controller*. *Motor* innehåller regulatorn och alla funktioner för integrering med hårdvaran, till exempel funktionen för att ta fram robotens vinkel utifrån mätvärdena från IMU:n och skicka utsignalen från regulatorn till motorerna. *Controller* fungerar som en autonom fjärrkontroll vars uppgift är att kontinuerligt räkna ut värdena på styrsignalerna som skickas till *Motor*. Klassen innehåller alla funktioner som tolkar informationen från pixyn och fattar beslut baserat på den för att kunna spela fotboll.

Controller och *Motor* kommunicerar via Motor-klassens funktion *steer* som tar antingen ett eller två kommandon och en värde för varje kommando som argument. Exempel på kommandon är *forward, backward, left, right* och *stop*. Steerfunktionen sätter sedan regulatorns styrsignaler utifrån argumenten. Utöver *Motor* och *Controller* finns de mindre klasserna: *Tools* och *EEPROM*. *Tools* innehåller funktioner för kommunikation via Arduinos serielport samt kalibrering av accelerometrarna och motorerna. *EEPROM* gör att bl a PID-parametrarna kan sparas i EEPROM-minnet på Arduinokortet och ändras via serielporten.

4.2 Regulator för balansering

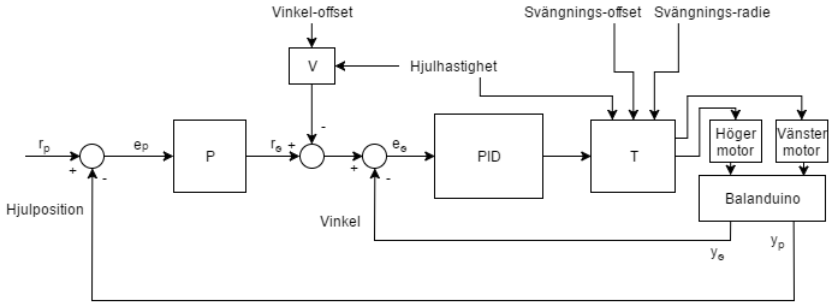
Robotens regulator klarar av att balansera roboten, köra frånåt, köra bakåt, rotera samt svänga med en bestämd radie. Den är nästan oförändrad jämfört med regulatorn som följer med Balanduino-plattformen med undantag av tillägget som gör att den kan svänga med en bestämd radie. En översikt av regulatorn finns i figur 14.

Regulatorn består av en inre och en yttre regulator. Den inre regulatorn är en PID-regulator som tar en vinkel som referensvärde. Den yttre regulatorn är en P-regulator som tar en målposition som referensvärde och den används endast när roboten står still eller bromsar. En ny målposition sätts när ett stopp-kommando skickas till regulatorn. När roboten ska köra framåt eller bakåt används bara den inre regulatorn.

Förutom referensvärdena tar regulatorn också in robotens uppmätta vinkel, vinkel-offset, svängnings-offset, svängnings-radie, hjulposition och hjulhastighet. Vinkel-offseten används för att köra framåt eller bakåt. Den skalas beroende av

hjulhastigheten och subtraheras sedan från referensvinkeln. Svängnings-offseten adderas respektive subtraheras till utsignalen till vänster respektive höger motor för att få roboten att svänga. Svängnings-offseten skalas också beroende på robotens hastighet. När svängnings-radien används räknas en skalfaktor ut, se uträkning i avsnitt 3.3, för höger och vänster hjul som multipliceras med utsignalen till respektive motor. Svängnings-offseten och svängnings-radien kan inte användas på samma gång ifall man vill ha förutsägbart beteende. Om värden sätts till noll har de ingen påverkan på utsignalen.

Hjulpositionen och hjulhastigheten kan mätas eftersom Balanduino har en rotationskodare vid varje motor som genererar pulser när motorn snurrar. Antalet pulser som motorerna har gått sparas i två räknare, en för varje rotationskodare. Hjulpositionen blir summan av pulsern i båda räknarna. Hjulhastigheten är antalet pulser per 100 millisekunder. När ett hjul snurrar ett varv motsvarar det plus eller minus 1856 pulser för det hjulets räknare, beroende på vilket håll hjulet snurrar.



Figur 14: Översikt av Balanduinons regulator. I V skalas vinkel-offseten med hjulhastigheten och i T räknas utsignalen för höger respektive vänster motor ut.

4.2.1 Förbättrad bromsning

När ett stopp-kommando skickas till regulatorn genom steer-funktionen finns en fördröjning på en halv sekund innan positionsregleringen slås på. Detta gör att roboten bromsar mjukare och klarar av att stanna från högre hastigheter än om positionsregleringen slås på direkt.

4.3 Kommunikation mellan Balanduino och Pixy

Balanduino och Pixy kommunicerar genom SPI (Serial Peripheral Interface) protokollet med hjälp av Arduinos bibliotek *SPI.h*. Pixy kommer med ett Arduino API (Application Programming Interface), som definieras i *Pixy.h*. Detta API används av Balanduino för att hämta information från Pixy. Det fungerar genom att Balanduino kör funktionen *getBlocks* som returnerar antalet objekt som Pixy ser och dessutom uppdaterar en array som innehåller övrig information om alla objekt såsom position på bilden, storlek på bilden, signatur och även vinkel ifall objektet är en färgkod.

4.3.1 Spara information om objekt med *getSignatureIndexes*

Informationen om alla objekt som Pixy ser finns i arrayen *pixy.blocks*. Problemet är att objekten i *pixy.blocks* sorteras med avseende på storlek i pixlar från störst till minst. Det gör det jobbigt att få ut informationen för ett visst objekt, t.ex. bollen, eftersom man inte vet vilket index objektet har i *pixy.blocks*.

Vad *getSignatureIndexes* gör är att den loopar igenom *pixy.blocks* och jämför signaturen på varje objekt i arrayen med de förprogrammerade signaturerna för objekten. Om signaturerna matchar sparas objektens index i *pixy.blocks* i *objectIndex* som är en array med sex platser, där varje plats innehåller informationen för varje typ av objekt. Nedan finns koden som definierar vilket objekt som har vilken plats i *objectIndex*:

```
//Obektens plats i objectIndex
//Objekt, Index
const int BALL = 0; //Boll
const int GOAL1 = 1; //Eget mål
const int GOAL2 = 2; //Motståndarmål
const int PLAYER1  = 3; //Medspelare
const int PLAYER2 = 4; //Motspelare
const int EDGE = 5; //Sarg
```

Det finns också en array *objectDistance* där avståndet [*cm*] till varje objekt sparas. Den funkar i övrigt på samma sätt som *objectIndex*. Avståndet räknas ut enligt formeln i avsnitt 3.1. Här används *objectIndex* och *objectDistance* för att plocka ut information om ett objekt i *pixy.blocks*:

```
//Användning av objectIndex och objectDistance
//Exempel: Ta reda på bollens bredd
int objectIndex[6];
int objectDistance[6];
.
.
widthBall = pixy.blocks[objectIndex[BALL]].width;
distanceBall = objectDistance[BALL];
```

Ifall det skulle finnas flera objekt av samma typ sparas bara informationen om objektet som är närmast. Om inget objekt av en viss typ syns sätts värdet på dess plats i båda arrayerna till -1.

För att jämföra signaturer behöver *getSignatureIndexes* känna till hur objekten lärdes till Pixy, alltså vilka signaturer de fick, och för att räkna ut avstånd behöver funktionen känna till den verkliga storleken på objekten. Värden på detta kan ställas in i filen *Configuration.h*.

4.4 Fjärrstyrning med hjälp av mikrofon

Mikrofonen ger ett värde på ljudintensiteten vilket läses av genom en av de analoga in-portarna på Balanduino. I koden samplas in-värdet från mikrofonen med frekvensen 10 Hz. En räknare håller koll på hur många gånger ett värde över gränsvärdet på ljudintensiteten mäts upp. Om räknaren kommer upp till

två växlas robotens tillstånd från av till på och vice versa. Denna räknare medför att ljudet måste pågå en bestämd tid för att programmet ska starta. Detta gör att programmet klarar av eventuella spikar från mikrofonen eller plötsliga höga och kortvariga ljud under själva matchen. Att räknaren måste räkna upp till två innebär att ljudet måste pågå i minst 200 ms för att roboten ska gå ur sitt viloläge. Detta medför att en kort störning från omgivningen inte kommer att aktivera styrfunktionen då en kontrollerad signal krävs.

Det finns också en räknare som håller koll på hur många gånger värdet från mikrofonen är under gränsvärdet. Den mäter alltså längden på tystnaden mellan två ljud. Ifall den räknaren kommer upp till fyra återställs den andra räknaren.

Den andra räknaren gör att roboten inte reagerar på två höga ljud ifall det är en period av tystnad som är längre än 400 ms mellan dem.

Variabeln som innehåller robotens tillstånd läses av i programmets huvudloop innan alla kontroll-offseten till regulatorn beräknas. Om variabeln är sann kallas den vanliga styrfunktionen, annars kallas robotens stoppfunktion. När tillståndet växlas till att vara på kommer roboten börja om med sin taktik. Den fortsätter alltså inte där programmet var innan tillståndet växlades till av.

4.5 Huvudprogrammet och *doTask*-funktionen

Huvudprogrammet har samma struktur som ett vanligt Arduinoprogram. Det består av två funktioner: *setup()* och *loop*, där *setup* görs en gång när programmet startar och *loop* görs om och om igen efter första *setup*.

Pseudokod för huvudprogrammet:

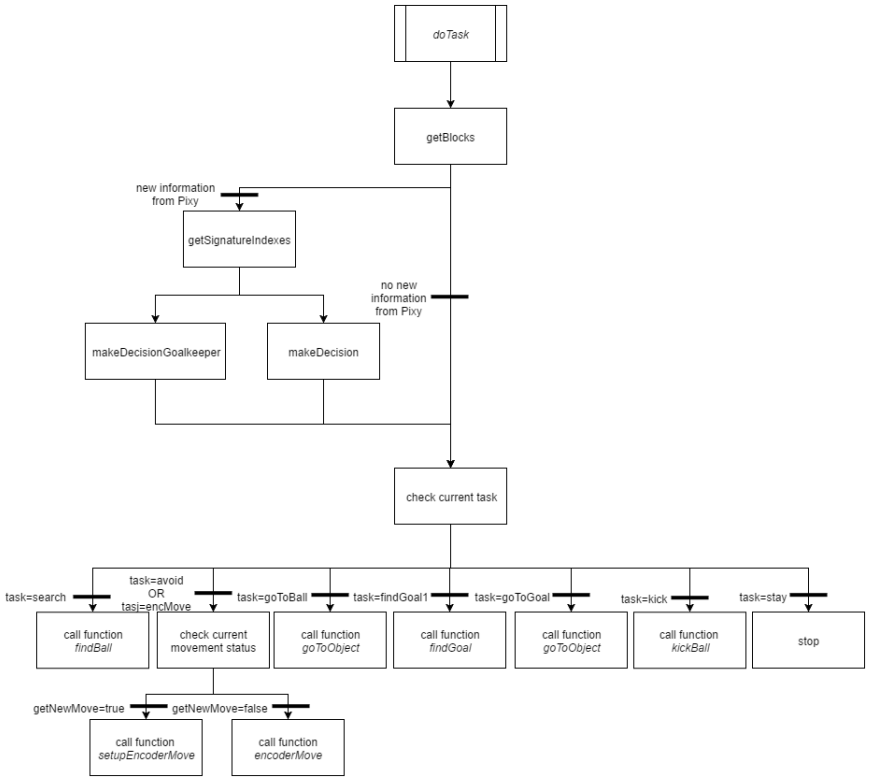
```
setup():
    Initiera seriell kommunikation
    Initiera rotationskodare, motorer och IMU
    Initiera Pixy
    Kalibrera gyro och återställ motorer

loop():
    Läs av värdet från mikrofonen
    och sätt robotens tillstånd till på eller av
    Om roboten är på:
        Räkna ut styrsignaler till regulatorn (doTask)
    Annars, om roboten är av:
        Kalla på robotens stoppfunktion
        Återställ värden i Controller
    Räkna ut robotens vinkel
    Kör regulatorn och skicka utsignalen till motorerna
    Updatera värdena från rotationskodarna
    Skicka och ta emot data från seriellporten
```

Uppdateringen av regulatorns referensvärde och utsignal måste ske med en hög frekvens för att roboten ska kunna balansera väl. Det gör att programkoden

som räknar ut regulatorns styrsignaler behöver vara så effektiv som möjligt, den får inte ta för lång tid att utföra. Nya styrsignaler måste räknas ut för varje iteration i huvudloopen, både baserat på vad Pixy ser och på vad roboten gjorde i föregående iteration. Denna koordinering sker i *doTask*. Funktionen anropas en gång per iteration och man kan säga att den binder samman alla funktioner i *Controller*.

Det som Pixy ser uppdateras med en uppdateringsfrekvens på 50Hz medan reglerloopen gås igenom betydligt fortare. I *doTask* finns därför en timer som håller koll på tiden sedan Pixy senast såg ett objekt. Utan den här timern hinner informationen hos Pixy inte alltid uppdateras när Balanduino försöker hämta den och Pixy rapporterar istället att den inte ser några objekt, trots att den gör det. På grund av att uppdateringsfrekvensen är 50 Hz väljs timern till 25 ms, roboten missade ibland att detektera objekt när gränsen var exakt 20 ms.

Centralt för *doTask* är variabeln *task* som sparar robotens nuvarande uppgift mellan iterationerna. Variabeln *task* är en instans av uppräkningstypen *Task* och kan anta åtta olika värden som motsvarar varsin uppgift: *search*, *goToBall*, *findGoal1*, *goToGoal*, *kick*, *avoid*, *encMove* och *stay*. Ett flödesschema för *doTask* finns i figur 15.



Figur 15: Algoritm som beskriver funktionen *doTask*.

Funktionen kallar först på *getBlocks* för att uppdatera informationen från Pixy. Sedan kallas *getSignatureIndexes* som sparar informationen på ett sätt som gör

det lättare att använda den i övriga funktionen. Efter det anropas *makeDecision* eller *makeDecisionGoalkeeper*, beroende på om roboten är offensiv spelare eller målvakt, som räknar ut vad nästa *task* ska vara, baserat på informationen från Pixy och föregående *task*. Det är i dessa funktioner som strategin finns. Efter det kallar *doTask* på funktionen som räknar ut regulatorns styrsignaler för respektive uppgift.

4.6 Viktiga stödfunktioner

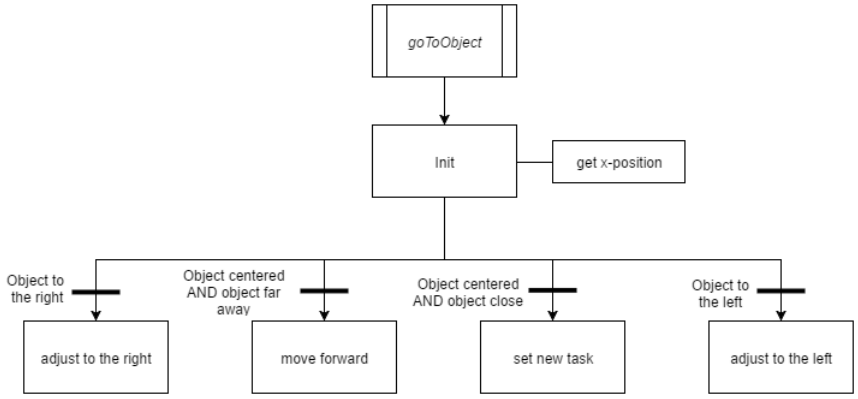
För att kunna utföra alla uppgifter som kan sättas i *doTask* behövs ett antal stödfunktioner som definierar alla rörelser. Till exempel att söka efter boll, sparka bollen och utföra en tidigare uträknad sekvens av rörelser.

4.6.1 Söka efter boll och mål med *findBall* respektive *findGoal*

Funktionen *findBall* använder sig av bollens senaste kända x-position för att roboten ska rotera åt det håll där bollen senast försvann ur kamerans synfält. För att målvakten ska kunna hitta sitt egna mål används *findGoal* som fungerar på samma sätt som *findBall*, men utifrån det egna målets senaste kända x-position.

4.6.2 Köra fram till objekt med *goToObject*

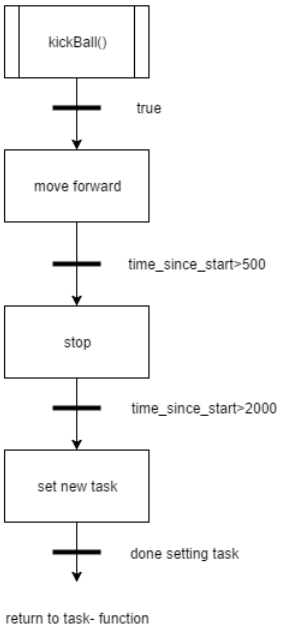
När *goToObject* kallas på finns ett objekt redan i synfältet. I initieringen av *goToObject* sparas x-positionen för det objektet som ges av Pixy. X-positionen är objektets geometriska centrum i x-led och hjälper funktionen att hålla objektet centrerat i synfältet. Funktionen använder avståndet till objektet för att avgöra om roboten ska åka framåt eller om den är framme vid objektet. Om roboten är framme vid objektet byter den uppgift till *stay*, eller *kick* ifall objektet är bollen. Funktionen beskrivs i figur 16.



Figur 16: Algoritm som beskriver funktionen *goToObject*

4.6.3 Sparkrörelse med hjälp av *kickBall*

Sparkrörelsen är tidsbaserad, se figur 17. I 500 millisekunder kör roboten i Balanduino-plattformens maxhastighet rakt fram, därefter kör den framåt i hastighet 0 i 1500 millisekunder innan den skickar kommandot *stop*. Att inte skicka *stop* direkt betyder att positionsregleringen fördröjs, vilket gör att den stannar mjukare och inte tappar regleringen eller faller omkull. Fördröjningen behövs också för att bollen ska hinna komma iväg en bit från roboten efter sparken. Utan fördröjningen skulle roboten göra en ny spark direkt eftersom bollen fortfarande skulle vara så nära och det skulle resultera i att roboten bara åkte i full fart framåt istället för att faktiskt sparka bollen.



Figur 17: Algoritm för *kickBall*

4.6.4 Pulsbaserad rörelse med *encoderMove*

En pulsbaserad rörelse kan vara att köra en bestämd sträcka med en viss radie eller att stillastående rotera ett visst antal grader. Detta är möjligt eftersom det går att räkna ut avståndet som ett hjul har gått, se avsnitt 3.3. Roboten behöver två funktioner för att uföra en pulsbaserad rörelse: *encoderMove* och *setupEncoderMove*. En rörelse definieras med en *MoveInstruction* och flera rörelser kan dessutom läggas in i en FIFO(First In First Out)-kö av typen *QueueList<MoveInstruction>*, vilket gör att en sekvens av rörelser kan utföras. En variabel *getNewMove* håller koll på om roboten är mitt i en rörelse eller om den ska påbörja en ny rörelse. Om *getNewMove* är *true* körs *setupEncoderMove*, annars körs *encoderMove*. För att en pulsbaserad rörelse ska utföras måste uppgiften i *doTask* vara *encMove* eller *avoid*.

4.6.4.1 *MoveInstruction*

En *MoveInstruction* innehåller informationen om en rörelse. Det finns två olika typer av rörelser: *line* och *spin*. Med *line* kör roboten framåt eller bakåt i en rak linje eller i en cirkelbåge. Med *spin* roterar roboten på plats. Informationen som finns i en *MoveInstruction* är: typen av rörelse, sträckan som roboten ska förflytta sig [*cm*], radien på cirkeln [*cm*], hastigheten på rörelsen och hur många grader rotationen ska vara. Ett positivt tal för radien och vinkeln anger rotation åt höger, ett positivt tal för längden motsvarar en längd framåt. Hastigheten anges i en skala 1-50 där 50 motsvarar robotens maxhastighet.

4.6.4.2 *QueueList<MoveInstruction>*

Typen *QueueList* fås genom att lägga till Arduino-biblioteket QueueList och inkludera *QueueList.h*. För att lägga in rörelser i kön gör man:

```
QueueList<MoveInstructionQueue> moveInstructionQueue;
.
.
MoveInstruction m1 = MoveInstruction(line,100,0,25);
MoveInstrcution m2 = MoveInstruction(spin,30,180);
MoveInstruction m3 = MoveInstruction(line,200,50,20);
moveInstructionQueue.push(m1);
moveInstructionQueue.push(m2);
moveInstructionQueue.push(m3);
```

Koden ovan lägger in tre rörelser i kön: Kör 100 cm rakt fram med hastighet 25, snurra 180° höger med hastighet 30, kör 200 cm i en cirkel åt höger med radie 50 cm och hastighet 25.

4.6.4.3 *setupEncoderMove*

För att *encoderMove* ska kunna mäta avståndet som roboten har kört behöver den veta vad robotens position var när rörelsen började. Om kön innehåller rörelser sparar *setupEncoderMove* startpositionen för varje hjul i två variabler och sätter sedan *getNewMove* till *false*. Om kön är tom betyder det att alla rörelser är klara och då sätter funktionen *task* till *search*.

4.6.4.4 *encoderMove*

När robotens startposition är satt kan *encoderMove* köras. Funktionen mäter skillnaden mellan startpositionen och nuvarande positionen och jämför den med värdet på sträckan alternativt vinkeln som lades in i *MoveInstruction*. När rörelsen är klar tar funktionen ut den ur kön och sätter *getNewMove* till *true*.

Beroende på typen av rörelse och om värdena är positiva eller negativa sätter *encoderMove* olika styrsignaler. Om typen är *line* kör roboten rakt fram när *distance* är positiv, när *distance* är negativ kör den båkåt. Den skickar dessutom

radien på rörelsen till regulatorn. Om typen är *spin* roterar roboten åt höger när *degree* är positiv, när *degree* är negativ roterar den åt vänster.

4.6.5 Ta sig till skottläge med *calculateTrajectory*

Funktionen *calculateTrajectory* räknar hur roboten måste köra för att ställa sig i linje med bollen och målet. Den räknar ut två rörelser, en rotation och en cirkelbåge. Hur dessa räknas ut beskrivs i avsnitt 3.4. Funktionen ser också till att värdena för rörelserna får rätt tecken beroende på var bollen och målet befinner sig relativt roboten. Rörelserna sätts sedan in i kön med rörelser. Funktionen tömmer också kön innan nya rörelser sätts in.

4.6.6 Jobba hem med *calculateGoHome*

Om roboten ser sitt eget mål och bollen samtidigt innebär det att den är på fel sida om bollen och behöver jobba hem, som det kallas, för att skydda sitt eget mål. Funktionen *calculateGoHome* räknar ut hur roboten ska åka utifrån var målet är och hur långt bort bollen är. Den räknar ut en rotation som ska svänga upp roboten mot sitt eget mål och den räknar ut en sträcka så att roboten åker en meter förbi bollen. Rörelserna sätts sedan in i kön med rörelser. Funktionen tömmer också kön innan nya rörelser sätts in.

4.6.7 Undvika hinder med hjälp av *calculateAvoidObject*

Som nämndes i inledningen är att undvika hinder en viktig funktion för en fotbollsspelande robot. I och med att det är många objekt, rörliga och konstanta, som befinner sig på planen samtidigt finns det stora risker för kollision. Dels behövs kollision undvikas med motspelare då det ingår i FIRA:s regler att inte tackla motspelare, dels blir den balanserande roboten instabil vid kollision och tappar tid för att balansera upp sig själv igen.

Funktionen *calculateAvoidObject* löser uppgiften genom att vända sig bort från det objekt som är närmast mot det håll bollen är. Det görs genom att först räkna ut vilket objekt av de objekten som Pixy ser som är närmast. Funktionen räknar sedan ut en rotation för att vrida roboten bort från objektet och en liten cirkelbåge för att ta sig en bit framåt. Om bollen är inom synfältet svänger roboten 75° bort från det närmsta objektet mot det håll bollen är. Om bollen inte är inom synfältet svänger roboten istället 120° mot det håll där bollen försvann ur synfältet och kör i en cirkelbåge med radien 30 cm och vinkeln 45°. Rörelserna sätts sedan in i kön med rörelser. Funktionen tömmer också kön innan nya rörelser sätts in.

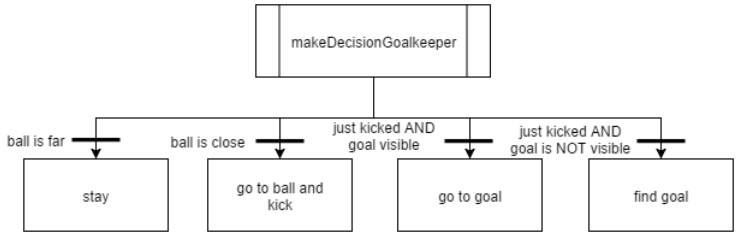
4.7 Strategi beroende på roll och *makeDecision*

Robotarnas spelstrategi baserar sig på vilken roll de har på spelplanen. Två roller finns: målvakt och offensiv spelare. Strategin för målvakten finns i *makeDecisionGoalkepper* och strategin för offensiv spelare finns i *makeDecision*. Båda

funktionerna tar antalet objekt som Pixy ser och nuvarande *task* som argument och returnerar en ny *task*. Den nya uppgiften räknas ut dels beroende på vad Pixy ser och dels beroende på vilken uppgift som roboten håller på att utföra, till exempel kan sparken eller en pulsbaserad rörelse bara avbrytas ifall roboten behöver undvika ett objekt.

4.7.1 Målvakt

Målvaktens huvudfunktion är att vakta målet och sparka bort bollar som kommer för nära, se figur 18. Implementeringen av detta görs genom att kalla på funktionen *goToObject* som beskrivs ovan, detta görs när bollen är 80 cm ifrån målvakten. För att se till att målvakten håller sig nära målet söker den rätt på sitt eget mål efter en spark, även detta görs med hjälp av *goToObject*.



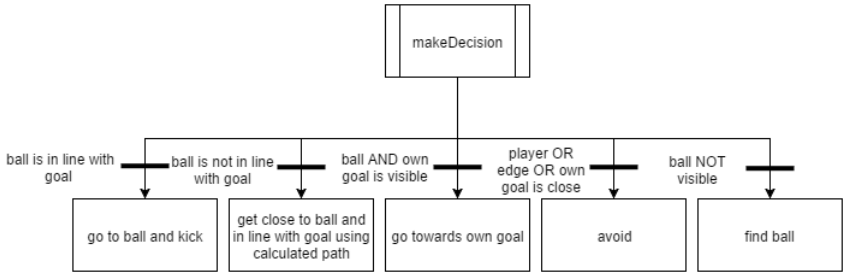
Figur 18: Algoritm som beskriver funktionen *makeDecisionGoalkeeper*

4.7.2 Offensiv robot

Prioriteringen för varje offensiv robot är att sparka bollen i mål, se figur 19. Om roboten inte ser bollen roterar den tills den har hittat bollen.

När bollen och motståndarmålet syns och är i linje med varandra kallas *goToObject* för att köra fram till bollen och skjuta den i mål. Om bollen och målet inte är i linje med varandra tar roboten sig till skottläge med hjälp av *calculateTrajectory*. När bollen och det egna målet syns jobbar roboten hem med hjälp av *calculateGoHome*. Om roboten ser bollen men inget av målen används *goToObject* för att åka mot bollen.

Ifall roboten skulle komma för nära ett objekt som inte är bollen undviker den det objektet med hjälp av *calculateAvoidObject*. Uppgiften *avoid* har en högre prioritet än alla andra uppgifter vilket gör att en rörelse kan avbrytas ifall roboten behöver undvika ett objekt.



Figur 19: Algoritm som beskriver funktionen *makeDecision*

5 Tester, resultat och analys

Resultaten har åstadkommits genom tester för tre olika delar i robotens funktionalitet: rörelser, objektidentifiering och strategiska funktioner. I detta kapitel kommer testerna för varje del av robotens struktur att beskrivas. Därefter kommer resultaten av testerna att presenteras med en efterföljande analys. Efter testerna av robotarnas funktionalitet kommer matchens resultat att redovisas och analyseras.

5.1 Robotens rörelser

För att verifiera att fotbollsspelarna kan utföra grundläggande rörelser har tester utförts för att verifiera kraven som tidigare satts upp i kravspecifikationen, se appendix A. Specifikationen omfattar olika typer av krav, dels ska robotarna kunna utföra rörelserna snabbt och dels ska de kunna göra det med en viss precision och säkerhet.

5.1.1 Hastighetstester

För att testa de uppsatta kraven om hastighet vid körning rakt fram kördes robotarna en viss sträcka. Tiden det tog mättes upp vilket gjorde att en medelhastighet kunde räknas ut. Vid alla tester av maxhastighet var start- och slutpunkt för tidtagning punkter där roboten hade kommit upp i sin maxhastighet respektive inte hade börjat bromsa. Två olika hastighetsresultat beräknades. I det ena fallet behövde roboten endast balansera under körning och i det andra skulle roboten även kunna stanna utan att ramla. Tester av maxhastigheter utfördes för raksträcka, för cirklar med olika radier samt för rotation på plats.

- *Test 1: Maxhastighetstest, tillåtet med fall vid inbromsning*
  Mäter tiden det tar för roboten att köra 5 m rakt fram för att beräkna robotens maxhastighet. Roboten måste klara av att stå upp under hela sträckan men tillåts ramla vid inbromsning.

- *Test 2: Maxhastighetstest, ej tillåtet med fall vid inbromsning med positionsreglering*
  Mäter tiden det tar för roboten att köra 2 m rakt fram för att sedan beräkna robotens maxhastighet. Roboten ska klara av att stå upp under inbromsningen. Positionsreglering är påslagen under inbromsning.

- *Test 3: Maxhastighetstest, ej tillåtet med fall vid inbromsning utan positionsreglering*
  Mäter robotens maxhastighet med distansen 5 m, roboten ska kunna stå upp vid inbromsning med positionsreglering avslagen.

- *Test 4: Maxhastighetstest, olika radier*

Mäter robotens maxhastighet under körning av cirklar i olika radier.

- *Test 5: Maxhastighetstest för rotation*
  Mäter hur lång tid det tar för roboten att rotera 5 varv på plats för att få fram en maximal rotationshastighet.

5.1.1.1 Resultat hastighetstester

Nedan följer resultaten av hastighetstesterna *Test 1 till 5*.

Tabell 1: Tabell för maxhastighetstest, *Test 1*:
5 m rakt fram, fall vid inbromsning tillåtet.

Försök	**Tid [s]**	**Hastighet [m/s]**
1	3,47	1.44
2	3,36	1.49
3	3,55	1.41
4	3,47	1.44
5	3,50	1.43
Medel maxhastighet:		**1.44**
Standardavvikelse:		**0.0293**

Medel maxhastighet i tabell 1 motsvarar 94% av robotens maximala hastighet enligt begränsningen som finns i koden.

Tabell 2: Tabell för maxhastighetstest, *Test 2*:
2 m rakt fram, fall vid inbromsning ej tillåten, positionsreglering påslagen vid inbromsning.

Försök	**Tid [s]**	**Hastighet [m/s]**
1	4,05	0.49
2	3,97	0.50
3	3,95	0.51
4	4,01	0.50
5	3,99	0.50
Medel maxhastighet:		**0.50**
Standardavvikelse:		**0.0048**

Här klarade roboten att stå upp vid 3 av de 5 försöken. *Medel maxhastighet* i tabell 2 motsvarade 40% av robotens teoretiska maxhastighet enligt begränsningen som finns i koden.

Tabell 3: Tabell för maxhastighetstest, *Test 3*:
5 m rakt fram, fall vid inbromsning ej tillåten, positionsreglering avslagen vid inbromsning

Försök	**Tid [s]**	**Hastighet [m/s]**
1	5,59	0.89
2	5,58	0.90
3	5,69	0.88
4	5,54	0.90
5	5,57	0.90
Medel maxhastighet:		**0.89**
Standardavvikelse:		**0.00899**

Medel maxhastighet i tabell 3 motsvarar 60% av robotens teoretiska maxhastighet enligt begränsningen som finns i koden. Denna variant av bromsmetod med positionsregleringen avslagen visar sig här möjliggöra inbromsning från en högre hastighet och anses fördelaktig. Bromsning utan positionsreglering har använts i samtliga nästkommande hastighetstester.

Tabell 4: Tabell för maxhastighetstest, *Test 4*:
körning i olika radier, fall vid inbromsning ej tillåten, positionsreglering avslagen vid inbromsning

Radie [cm]	**Tid [s]**	**Hastighet [m/s]**
100	8.60	0.73
100	8.63	0.73
100	8.54	0.74
Medel maxhastighet för radie 100 cm:		**0.73**
Standardavvikelse:		**0.0039**
50	4.52	0.70
50	4.40	0.71
50	4.31	0.73
Medel maxhastighet för radie 50 cm:		**0.71**
Standardavvikelse:		**0.01697**
30	2.64	0.71
30	2.68	0.70
30	2.68	0.70
Medel maxhastighet för radie 30 cm:		**0.71**
Standardavvikelse:		**0.00615**

Tabell 5: Tabell för maxhastighetstest, *Test 5*:
Tidtagning för rotation 5 varv för att mäta maximal rotationshastighet. Fall vid inbromsning ej tillåten, positionsreglering avslagen.

Försök	**Tid [s]**	**Hastighet [varv/s]**
1	7,78	0,64
2	7,91	0,63
3	7,86	0,64
Medel maxhastighet:		**0,64**
Standardavvikelse:		**0,00533**

Denna rotationshastighet motsvarar 100% av robotens maximala rotationshastighet enligt begränsningarna som finns i koden.

5.1.1.2 Analys av hastighetstester

Det framgår från hastighetstesterna att roboten kan köra med betydligt högre hastighet beroende på hur roboten stannar. Med positionsregleringen påslagen under inbromsningen är medelhastigheten 0.50 m/s jämfört med 0.89 m/s när positionsregleringen är avslagen. Detta motsvarar en ökning på 78% som tyder på att en bromsfunktion har stor betydelse och gynnar robotens snabbhet under matchen.

Standardavvikelsen från samtliga tester är låg vilket tyder på att motorerna genererar en jämn hastighet. Detta tyder på att roboten kommer genomföra rörelsebaserade funktioner med en jämn hastighet under matchen. Det framgår inte heller någon minskad maxhastighet för små radier, roboten kommer alltså inte behöva sakta in innan den ska köra i en skarp kurva. Anledningen att roboten kan rotera i maximal hastighet är troligtvis för att dess masscentrum inte förflyttar sig och därmed inte skapar stora störningar.

5.1.2 Precisionstester rörelse

Robotens precision vid rörelser är viktiga för att roboten ska kunna placera sig i enlighet med de beräknade matematiska modellerna. Tester här gick ut på att ge roboten ett kommando om att köra en bestämd sträcka från en viss utgångspunkt på ett visst sätt och sedan stanna på den önskade punkten. Precisionstester utfördes för rak förflyttning, förflyttning längs en cirkelbåge och för stillastående rotation.

- *Test 6: Precisionsrörelsetest för raksträcka.*
  Testet gick ut på att roboten fick kommandot att köra rakt en sträcka om 5 m, därefter mättes hur långt ifrån önskade punkten roboten stannade. För att roboten inte skulle vika av i sidled kalibrerades motorerna inför testet.
- *Test 7: Precisionsrörelsetest för radie.*

Roboten fick kommando att åka ett helt cirkelvarv med en given radie om 50 cm. Mätningar skedde för hur exakt roboten kunde åka i den önskade radien samt hur väl den klarade att åka ett helt cirkelvarv.

- *Test 8: Precisionsrörelsetest för rotation.*
  Testet gick ut på att ge roboten en bestämd vinkel den skulle rotera, skillnaden mellan det önskade värdet och verkliga värdet mättes.

5.1.2.1 Resultat precisionstester rörelse

Nedan följer resultaten för precisiontesterna *Test 6 till 8*

Tabell 6: Precisionstester rörelser raksträcka, *Test 6*: samtliga försök för en raksträcka om 5 m

Försök	**Verklig sträcka [cm]**	**Procentuell avvikelse** [%]
1	497 [cm]	0.6%
2	495 [cm]	1%
3	497 [cm]	0.6%
Medel procentuell avståndsavvikelse:		**0.73**%
Standardavvikelse:		**0.23**

Tabell 7: Precisionstester rörelser radie, *Test 7*: samtliga försök för körning med en radie om 50 cm, längdavvikelsen redovisar hur väl roboten klarade att köra ett helt cirkelvarv.

Försök	**Verklig radie [cm]**	**Längdavvikelse [cm]**
1	49,5 [cm]	-10 [cm]
2	49 [cm]	-15 [cm]
3	49 [cm]	-15 [cm]
Medel procentuell radieavvikelse:		**1.67** %
Standardavvikelse:		**0.00577**
Medel procentuell längdavvikelse [%]**:**		**4.24** %
Standardavvikelse:		**0.0092**

Tabell 8: Precisionstester rörelser, *Test 8*: Mätning för hur exakt roboten kan genomföra olika rotationsvinklar

Önskad rotation [°]	**Verklig rotation** [°]	**Procentuell avvikelse** [%]
1800 ° (5 varv)	1755 °	2.5 %
1800 ° (5 varv)	1755 °	2.5 %
1800 ° (5 varv)	1755 °	2.5 %
90 ° (1/4 varv)	95 °	5.6 %
90 ° (1/4 varv)	93 °	3.3 %
90 ° (1/4 varv)	93 °	3.3 %
Medel procentuell rotationsavvikelse:		**1.27** %
Standardavvikelse:		**1.347**

5.1.2.2 Analys av resultat precisionstester rörelse

Resultaten tyder på att roboten är konsekvent i sitt sätt att röra på sig. De flesta försök här har resulterat i en kortare sträcka, snävare radie eller en mindre vinkel än önskat. De fel som uppstår beror troligtvis inte på tillfälliga mätbrus utan är konsekventa fel som kan bero på approximationer i matematiken eller bero på skevheter i hårdvaran för rotationskodarna på roboten. Kalibreringen av motorerna har visat sig vara viktiga för dessa tester så att roboten kör rakt och har gjorts regelbundet under testernas gång. Dessa resultat visar att rörelserna är tillräckligt exakta för att roboten inte ska avvika nämnvärt från de körkomandon som roboten ska utföra under matchen för att ta sig till önskad position.

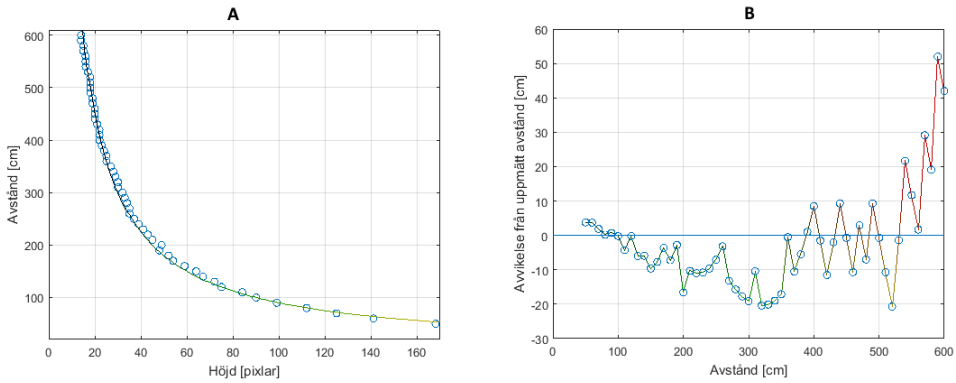
5.2 Objektidentifikation

Detta kapitel syftar på att visa hur kameran uppfattade omgivningen och hur robotens uppfattning skiljer sig från verkligheten. Testerna som utfördes är tester för hur långt avstånd som olika objekt kunde uppfattas på och hur väl avståndsberäkningen stämde med verkligheten.

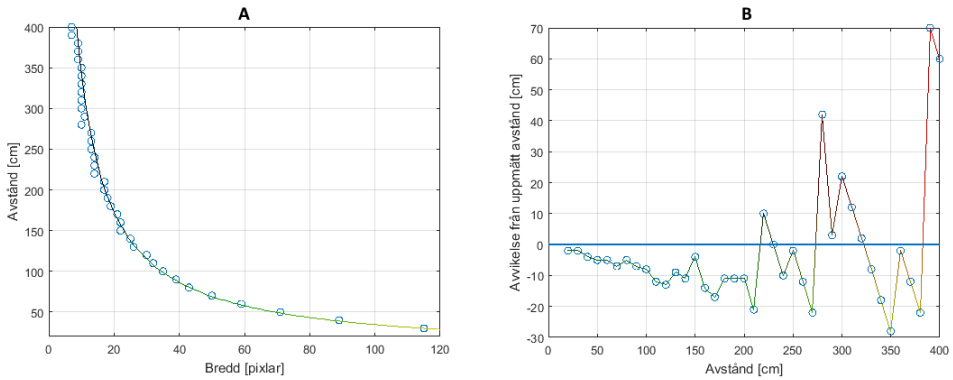
- *Test 9: Kameratest avståndsbedömning för boll.*
- *Test 10: Kameratest avståndsberäkning för mål.*
- *Test 11: Kameratest avståndsberäkning för andra robotar.*
- *Test 12: Kameratest avståndsbedömning för färgkoderna som placerats på sargen.*

5.2.1 Resultat av tester objektidentifikation

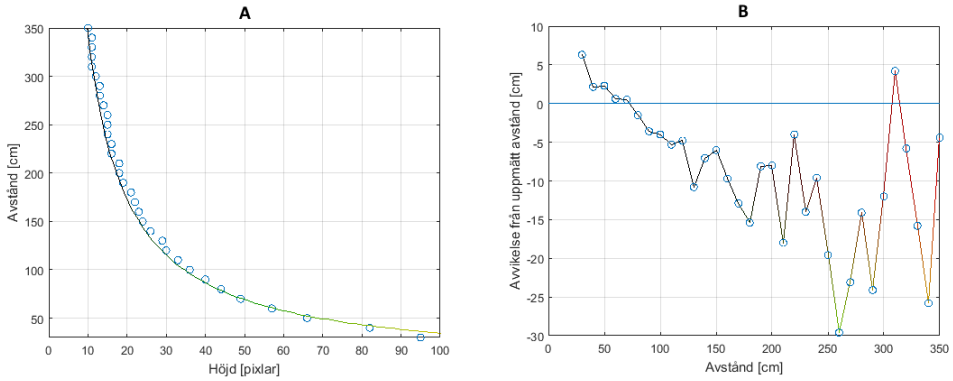
Nedan följer resultat från objektidentifikationstester *Test 9 till 12*



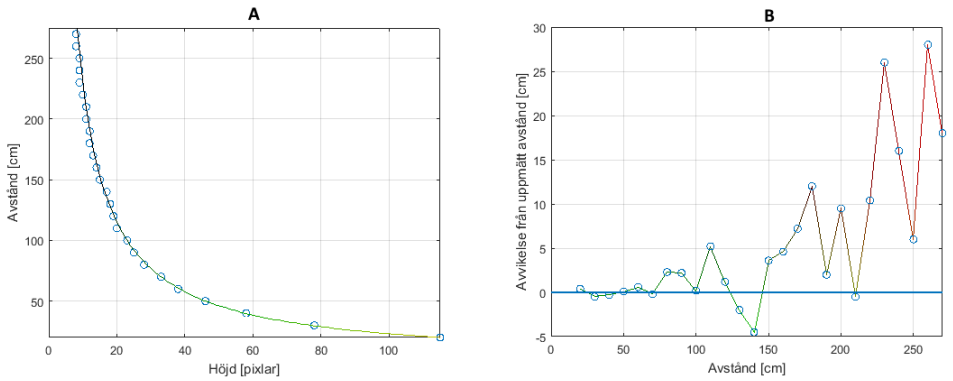
Figur 20: *Test 9* **A:** Avståndet till boll beräknat med ekvation 3 där cirklarna representerar mätvärden. **B:** Storleken på avvikelser från verkliga avståndet.



Figur 21: *Test 10* **A:** Avståndet till mål beräknat med ekvation 3 där cirklarna representerar mätvärden. **B:** Storleken på avvikelser från verkliga avståndet.



Figur 22: *Test 11* **A:** Avståndet till robot beräknat med ekvation 3 där cirklarna representerar mätvärden. **B:** Storleken på avvikelser från verkliga avståndet.



Figur 23: *Test 12* **A:** Avståndet till sarg beräknat med ekvation 3 där cirklarna representerar mätvärden. **B:** Storleken på avvikelser från verkliga avståndet.

5.2.2 Analys av resultat tester objektidentifikation

Färgkoderna som fästes på sargen i *Test 12* var små eftersom det inte var nödvändigt för roboten att uppfatta sargen på särskilt långt avstånd. Detta innebär att stora avvikelser i mätningen av sargen när den är långt borta inte påverkar robotens funktion nämnvärt. När avståndet till sargen var kort var mätfelet litet.

I testerna för boll, mål och lagtröjor märks en trend att beräkningsfelet var mindre när objektet befann sig närmare kameran. När objektet befann sig längre från kameran uppfattades ofta avtåndet till objektet som något kortare än det verkliga uppmätta avståndet. När objektet befann sig väldigt långt bort från kameran uppfattades ofta objektet att ligga längre bort än vad det gjorde i verkligheten, speciellt märkbart i *Test 10* för mål och *Test 12* för sargen.

5.3 Funktionstester

I följande tester verifieras robotens utförande av funktionerna som behövs för att spela fotboll. Detta omfattar tester för uppsökning av bollen, att utföra sparkrörelsen och att göra mål. Andra funktioner som har testats är om målvakten fungerar väl i sin taktik för att försvara målet. Här har även ett test av funktionen *avoid* genomförts för att se om roboten kan undvika att köra in i hinder.

- *Test 13: Funktionstest uppsöka boll.*
  Här testades hur väl roboten kan hitta och söka upp boll samt genomföra en spark på bollen. Bollen placerades ut i rummet, roboten placerades ut på ett annat ställe i rummet och sedan skulle den uppsöka boll. Ibland sparkades bollen undan innan roboten kommit fram till bollen och roboten skulle då söka upp bollen åt det håll den såg bollen senast. Roboten försökte hela tiden ta sig till boll och sparka iväg den rakt framåt. En lyckad uppsökning och spark noterades om roboten hittade boll, åkte fram till den, träffade bollen med sin spark och kunde stå upp efter genomförd spark. Testet repeterades till att omfatta 10 försök.
- *Test 14: Funktionstest för skottfunktionen.*
  Här testades hur bra de offensiva robotarna är på att göra mål. Bollen placerades ut stillastående någonstans mellan robot och mål så att både boll och mål fanns med i robotens synfält. Därefter försökte roboten sparka bollen i mål och antalet lyckade försök noterades. Testet repeterades till att omfatta 10 försök.
- *Test 15: Funktionstest för att testa om roboten kan undvika objekt som dyker upp i dess väg.*
  Testet gick ut på att testa om roboten kan avbryta sin pågående funktion när ett objekt dyker upp i dess väg. Detta testades genom att placera en motspelare i robotens väg när den försöker placera sig för att göra mål. Roboten skulle då klara av att avbryta sin placering och istället undvika motspelaren. Om roboten undvek att köra in i den

andra roboten registrerades ett positivt resultat. Testet upprepades tills det omfattade 10 försök.

- *Test 16: Funktionstest målvaktens strategi och utförande.*
  Testet undersöker hur ofta målvakten efter avspark kunde uppsöka sitt eget mål och placera sig framför det. Denna egenskap är viktig att konsekvent lyckas med för att målvakten ska kunna återgå till att skydda mål efter de gånger den har sparkat bort bollen från mål. Vidare i testet utsattes målvakten för lägen då den borde välja att åka ifrån sin position och sparka iväg bollen. Målvakten utsattes för 10 lägen där bollen kom tillräckligt nära för att målvakten borde åka fram och sparka iväg bollen. Om roboten mötte upp bollen och sparkade iväg den samt efteråt kunde ta sig tillbaka till en position framför målet registrerades ett positivt utfall av testet.

5.3.1 Resultat av tester funktioner

- Funktionstest uppsöka boll, *Test 13*: 10 av 10 försök lyckades.

Tabell 9: *Test 14*: Funktionstest skottfunktion, bollen placerades ut stilla på olika platser i rummet framför mål. Roboten med startpunkt där både boll och mål fanns med i synfältet. Om roboten lyckades göra mål noterades testet som lyckat.

Försök	**Mål [Ja/Nej]**
1	Ja
2	Ja
3	Ja
4	Ja
5	Ja
6	Nej
7	Nej
8	Ja
9	Nej
10	Ja

- Funktionstest *avoid*-funktionen, *Test 15*: 10 av 10 försök lyckades.
- Funktionstest målvaktsfunktionen, *Test: 16*: 10 av 10 försök lyckades.

5.3.2 Analys av resultat av tester funktioner

I *Test 13* fungerade funktionen precis som den var tänkt. Roboten lyckades alltid uppsöka bollen utan problem.

I *Test 14* ansågs en boll som träffat stolpe, träffat sarg eller om roboten ramlat som ett genomfört försök. Ett försök kan vara en kombination av flera skottrörelser. Denna funktion körs om och om igen på de offensiva robotarna under matchen så ett noterat misslyckat försök här kan i verkligheten ha resulterat i en studs som skapat ett nytt bra läge, där roboten sedan har god chans att

göra mål under matchen. I de fall då roboten missat mål har anledningen varit att roboten inte lyckats placera sig i rätt läge vid första beräkningen, vilket har gjort att den sedan vid nästkommande beräkning varit placerad precis bakom bollen. Då har den rört sig i en väldigt snäv radie som varit svår för roboten att kontrollera och resulterat i en spark åt helt fel håll. Under första beräkningen har roboten någon gång stått bra placerad och det har funnits möjlighet för roboten att avbryta sin radie för att gå över till en rak skottfunktion, denna egenskap har inte triggats på rätt ställe i detta test och är vad som borde utvecklas mera efter detta resultat. Radien som roboten åker mot boll med har oftare varit för snäva än för vid vilket tyder på någon form av skev approximation i den applicerade matematiken.

I *Test 15* fanns endast ett objekt på planen som roboten kunde köra in i som också var stillastående. I matchen fanns det många fler objekt samtidigt på planen som roboten kunde krocka in i samt att andra spelarrobotar även förflyttade sig. Detta borde motsvara bra hur roboten beter sig i matchen då den är på väg att krocka in i sargen. När det gäller hur stor chansen är att roboten krockar in i andra robotar ökar risken för kollision när andra objekt på planen förflyttar sig.

I *Test 16* ställde sig roboten bra i mål vid första placeringen. Efter ett antal sparkar kunde det hända att roboten uppsökte sin position från en större vinkel och kunde då placera sig mer mot kanten av sitt mål. Om bollen hamnade mellan målvakt och mål finns det i denna funktion inget som hämtar ut bollen från sitt egna mål, alltså är det viktigt att målvakten inte går upp och möter boll för ofta då det riskerar att målvakten hamnar på fel sida boll.

5.4 Match

Ett av målen med projektet var att kunna spela en fotbollsmatch mot ett annat robotlag. Matchen som spelades sattes upp med så nära tillämpning av de uppsatta reglerna som var praktiskt möjligt. Våra robotar var i början väldigt statiska eftersom de stängde av sig själva via mikrofonen när de började pipa på grund av balansrubbning.

Mikrofonen stängdes av i andra halvlek och då kunde man se mer konkreta resultat. De offensiva robotarna kunde ställa sig i skottposition, anledningen till att skott inte alltid genomfördes var att bollen rörde på sig mycket vilket var väntat.

Robotarna kunde undvika sargen så länge inte roboten hade påbörjat ett skott i närheten av kanten. Robotarna lyckades inte att undvika andra spelare eftersom Pixy hade svårt att identifiera objekt under matchförhållanden. En bugg i programmet när det gällde att undvika objekt var anledningen till detta. Målvakten fungerade väldigt bra och lyckades göra två räddningar. Målvakten var okej när det gällde att placera sig mellan boll och mål.

5.4.1 Matchresultat

I resultatet för matchen betraktas *Grupp 1* som projektgruppen för detta arbete.

Tabell 10: Tabell för matchresultat och matchstatistik.

Fotbollsmatch	**Projektlaget**	**Motståndarlaget**
Resultat	2	1
Självmål	0	1
Skott	17	13
Skott mot mål	8	4
Skott rakt på mål	4	2
Skott rakt på eget mål	1	1
Räddningar	2	3
Tacklingar på motståndare	4	2
Tacklingar på medspelare	3	3
Omkullkörningar	24	27

5.4.2 Analys av matchresultatet

Robotarnas utförande i matchen påvisade att det var fördelaktigt att ha robotar som står upp och på något sätt, även långsamt, alltid jobbar mot att få bollen mot andra lagets planhalva och deras mål. Det visade sig även fördelaktigt att ha en väldigt defensiv målvakt som står och försvarar mål. Något som visade sig vara väldigt problematiskt var att få Pixy att känna igen flera färgkoder. Det som hände var att färgkoderna flöt ihop till en enda stor signatur. Det märktes också att alla färger inte fungerar lika bra under samma ljus då andra spelare inte kunde upptäckas av kameran.

6 Diskussion

I projektet har vi fått ta ställning till en mängd alternativ varje gång vi implementerat en ny funktion. I diskussionen går vi igenom hur våra val har påverkat projektet, saker som kunde göras annorlunda och hur man kan bygga vidare på projektet. Projektet blir egentligen aldrig fulländat eftersom man alltid kan förbättra roboten oavsett hur bra den är.

6.1 Val av utgångspunkt Balanduinon

Att arbetet valdes att baseras på den redan befintliga Balanduino-plattformen har varit till stor fördel och bidragit till projektets resultat. Detta eftersom fokus i arbetet har kunnat läggas på att bygga taktiken för att spela fotboll istället för att försöka få roboten att balansera.

6.2 Arbetsprocessen

Under arbetets gång har det visat sig gått snabbare än planerat att tillföra roboten funktioner som använder sig av enstaka olika komponenter som att hitta bollen eller åka på ett visst sätt. Däremot har programmeringen snabbt blivit komplicerad då roboten ska klara av uppgifter som är en kombination av mindre separata uppgifter, som att sparka bollen i mål eller att undvika andra robotar samtidigt som den färdas mot sitt mål.

Detta har märkts tydligt när nya funktioner tänkt tillföras. Mycket tid har lagts på att få roboten att börja eller sluta utföra en rörelse vid den tänkta start- eller stopsignalen. När väl roboten har börjat göra den tänkta uppgiften vid rätt tillfälle har finjusteringsprocessen gått snabbt och smidigt.

6.3 Pixy

Att använda sig av Pixy-kameran för avståndsmätning har fungerat tillräckligt bra för att inte behöva studera andra alternativ i stor utsträckning. Laser hade kostat för mycket för att hålla sig inom ramen för detta projekt och en sonar eller radar skulle troligtvis få in för mycket störningar för att ge ett mer exakt resultat.

6.3.1 Tiltning av kameran

Det gick inte lika lätt att integrera tilt-funktionen till kameran som det var tänkt. När servot skulle kopplas ihop med Pixy-kamerans ICSP-uttag vid användandet av Balanduino-plattformen betedde sig servot inte alls som planerat. Dels roterade servot direkt vid koppling utan att programmet egentligen kördes, dels reagerade det annorlunda när objekt visades framför kameran. Detta berodde antagligen på att strömförsörjningen genom ICSP-uttaget inte räckte till för att strömsätta både Pixy och servot. När Pixy och servot strömsattes genom mini-usb uttaget på Pixy fungerade tiltningen mycket bättre. Men eftersom det inte skulle gå att springa efter roboten med en usb-sladd ikopplad i Pixy sattes kameran i ett fast läge under matchen. Pixy har även ett uttag för att koppla

in en extern strömkälla mellan 6-10 volt. Detta alternativet hann vi dock inte med att testa.

6.3.2 Ljuspåverkan

Kameran har visat sig väldigt känslig för olika ljusförhållanden och påverkar robotens betenden beroende på rummets miljöer. Eftersom många funktioner beror på hur stort objektet uppfattas som av kameran så har objekt fått läras om för varje ny miljö. Skarpt ljus ovanifrån har gjort en överexponerad del uppe på bollen och en för mörk miljö har gjort att bollens yta inte definierats väl och objekt lättare flyter ihop med varandra eller misstas med omgivningen.

6.3.3 Färger olika lätta att upptäcka

Pixy har visat sig ha olika lätt för sig att upptäcka olika färger i rummet beroende på ljussättningen. Till exempel. visade sig gult ofta vara mycket svårare att få bra täckning på jämfört med färgen blå, men vid kombinationen av de båda i en färgkodning uppfattades båda färgerna bra. Detta medförde att målen bestämdes vara olika färgkoder istället för att vara enfärgade. När detta infördes fick inte ena laget fördel i matchen genom att kunna se mål bättre från ett längre avstånd.

6.3.4 Pixy glömmer bort vad man lärt den

Det hände ofta när man satt och arbetade med roboten att kameran helt plötsligt inte kände igen några objekt på planen alls. Detta hände oftare ju fler olika färger och objekt man för tillfället hade lärt Pixy. Orsaken till detta har inte hittats.

6.3.5 3D-printningen

Att 3D-printa stödet till Pixy-kameran gav möjligheten att designa det för att passa kamerans tiltning perfekt samt designa plats för ett eventuellt framtida servo för panorering samt för att integrera fler ingenjörsområden i projektet. Denna del av projektet har fungerat väldigt väl även om plattan modifierats i efterhand då alla hål blivit för små i printningsprocessen.

6.4 Ta sig till skottposition

Funktionen för att ställa sig i skottposition fungerar okej men inte felfritt. Dels har roboten en tendens att bete sig olika beroende på vilken sida av boll den utgår från. När roboten placerades till vänster om boll i *Test 15* körde roboten ofta fram till bollen med en för snäv radie och när den utgick från ett läge till höger om boll kör den med en för stor radie. Detta är ett problem som inte lösts helt men uppstår troligtvis när vinklar i beräkningarna blir negativa. Roboten positionerar sig inte korrekt på avståndet e från bollen. Detta har lösts genom att korrigera de beräknade körkommandona med olika faktorer för att stämma bättre överrens med praktiken.

6.5 Balansering och stabilitet

Våra robotar uppvisade ett instabilt beteende under matchen mot den andra gruppen. Detta kapitel diskuterar därför olika lösningar som kan minska detta beteende. Det var dock svårt att medvetet framkalla det instabila beteendet vilket gjorde det svårt att veta vad som var fel.

6.5.1 Bromsfunktion

Den simpla bromsfunktionen som implementerades gjorde att roboten ramlade mer sällan. Den tog dock inte bort det instabila beteendet i alla situationer.

6.5.2 Kalmanfilter

Kalmanfiltret som används beräknar vinkeln roboten står. De tre värdena hos kalmanfiltret har under projektet ändrats för att testa vilka värden som är optimala. Värdena som var från början visade sig fungera bättre än några värden vi ändrade till. Om kalmanfiltret ändå skulle vara en orsak till instabiliteten eller en lösning på det, kan det tänkas att det har med själva utformningen av det och vilka värden som skickas in i filtret. Kanske behöver dessa värden hanteras först innan de behandlas av filtret.

6.6 Vidareutveckling

Här presenteras tänkta förslag på hur detta fotbollslag skulle kunna videreutvecklas för att bili ännu bättre på att spela fotboll.

6.6.1 Fjärrkontrollerad av- och påfunktion

På grund av att robotarna pep när de började tappa balansen så gick det inte att använda mikrofon för att starta och stänga av. Vi valde att pröva mikrofon eftersom det skulle bidra med en verklighetsenlig effekt om en domare skulle starta matchen med en visselpipa. Problemet skulle annars lätt kunna

6.6.2 Kommunikation

En viktig del ur FIRA:s reglemente som kunde ha prioriterats mer genom detta projekt har varit kommunikationen mellan robotarna. Området har inte prioriterats under arbetet då det har gått att skapa ett fungerande fotbollsspelande lag där besluten ligger hos varje robot för sig beroende på vad den sig runtomkring sig och hur de andra robotarna är placerade på planen i förhållande till mål och sarg. Ett avancerat passningsspel ansågs bidra till ökad speldulighet först om det togs till en ganska hög nivå som i projektets begränsade tidsplan var svårt att hinna med. Beslut huruvida roboten skall undvika andra med- eller motspelare, placera sig på rätt sida bollen samt att målvakten passar bollen till de offensiva robotarna på måfå ansågs vara tillräcklig interaktion för att uppfylla kravet som är grunden till att få ett välfungerande fotbollslag som samarbetar. Samarbete kan alltså så ske utan kommunikation med hjälp av vad robotarna ser.

Här skulle det finnas stora möjligheter att bygga upp betydligt mer avancerade spelscheman som bygger mer på kommunikation mellan robotarna. Denna kommunikation skulle kunna vara att robotarna talar om för varandra var de befinner sig, vilket läge de har och hur medspelarna bör agera. Här skulle en typ av sändare och mottagare för en frekvens kring 433 Mhz, som finns att köpa för Arduino, vara ett sätt att möjliggöra att robotar kan säga till varandra om de är fria till mål och vill få en pass eller liknande utan att den mottagande lagspelaren ser den andra.

Balanduino stödjer kommunikation via Bluetooth men det ansågs för instabilt och svårt att parkoppla för att vara värt att testa. Här testades kommunikation via LED-lampor vilket visade sig vara i princip omöjliga att upptäcka med Pixy. Att använda sig av mekaniska signaler som att höja en liten pompom krävde införskaffande av flera ytterligare mekaniska delar som troligtvis fortfarande skulle vara för små för att kunna uppfattas på tillräckligt långt håll. Den slutsatsen baserar vi på att de bollar i samma storlek som skulle användas till lagtröjor, visade sig vara för små för Pixy att upptäcka på mer än en meter.

6.6.3 Färdvägsplanering och förbättrade rörelseförmågor

Det finns möjligheter att arbeta vidare till viss del med rörelseförmågorna och färdplaneringen för att snabbare kunna placera sig rätt på planen eller smidigare röra sig i närheten av andra fotbollsspelare eller objekt. Detta skulle bidra till att man blir snabbare än motståndarna men bara till viss del eftersom att maxhastigheten på dessa robotar nås relativt lätt.

Roboten har svårigheter att träffa en boll i rörelse med sin spark-funktion. Detta beror på att sparkfunktionen endast åker rakt fram i hög hastighet och korrigerar inte rörelsemönstret utifrån var bollen är. Förbättringar skulle kunna göras så att roboten tar hänsyn till hur bollen rör sig. Dessa förbättringar skulle kunna ta hjälp av tiltfunktionen som finns till för att se hela bollen även när bollen är rakt framför roboten. Med hjälp av att komma närmare bollen än roboten gör skulle sparken kunna triggas senare och därmed minska mängden sparkar som missar bollen när den är i rörelse.

Roboten kan inte heller ta emot en boll. Det det svårt att passa bollen mellan robotarna. Att ta emot en boll skulle kunna lösas genom att roboten "fångar" bollen genom att röra sig bakåt i bromsande hastighet gentemot bollens hastighet precis när sammanstötningen skulle ha skett.

6.6.4 Vridning av synfältet för bättre uppfattning av omgivningen

I detta arbete valdes att fokusera på tiltningen av kameran för att kunna se bollen bra när det är väldigt korta avstånd till den och bollen annars försvinner ut ur undre delen av synfältet. Möjligheten till panorering är planerad genom att det 3D-printade stödet har platser och infästning för ytterligare ett servo som kan vrida plattformen kameran är fäst på. Detta skulle bidra till att roboten kan identifiera objekt runt omkring sig samtidigt som den färdas åt ett annat

håll men detta blev för komplicerat med ytterligare vinklar att ta hänsyn till för att vara värt att applicera i tidsramen inför fotbollsmatchen.

6.6.5 Målvakt med smart beräkning av bollbana

Målvakten är i dagsläget programmerad att till största del försöka skydda mål genom att ställa sig mellan bollen och målet och med jämna mellanrum verifiera att den står i rätt läge jämfört med målet som är bakom sig. Nästa steg i denna uppgift skulle vara att lägga till algoritmer för att beräkna en framtida bollbana på bollar som är på väg mot mål. Då skulle målvakten i förväg börja sin omplacering och därmed hinna ställa sig i rätt position tidigare och då hinna vrida sig med den bredare sidan som täcker större delen av målet.

Här skulle det även vara till stor hjälp att kunna vrida kameran snabbt för att verifiera att målvakten inte tappat bort målet bakom sig utan att behöva vända sig om.

7 Slutsats

Algoritmerna för de olika färdigheterna för att spela fotboll implementerades och robotarna spelade en fungerande fotbollsmatch där de framgångsrikt tog beslut om vilken färdighet som skulle användas beroende på situation. Resultatet har påvisat en möjlighet för hårdvaran och algoritmerna att användas i en riktig fotbollsturnering. Den fjärrkontrollerade på/av- styrningen fungerade inte felfritt på grund av att mikrofonen ibland reagerade på ljud från omgivningen, vilket ledde till att robotarna stängde av sig vid olämpliga tillfällen. Robotarna hade också en del problem med att de föll omkull, åkte in i sargen eller åkte på motspelare på spelplanen. På grund av dessa problem finns därför mycket som kan byggas vidare på och förbättras. En kamera med bättre och mindre känslig objektidentifiering samt åtgärder för stabilitet skulle ge ett mer flytande spel.

Referenser

[1] RobotWorx. (2016) Benefits of robots. Hämtat: 2016-02-04. [Online]. Available: https://www.robots.com/articles/viewing/benefits-of-robots

[2] L. Hardesty. (2015) Object recognition for robots. Hämtat: 2016-01-29. [Online]. Available: http://news.mit.edu/2015/object-recognition-robots-0724

[3] J.-H. Kim. (2016) About fira. Hämtat: 2016-01-29. [Online]. Available: http://fira.net/contents/sub01/sub01_1.asp

[4] F. Solc and B. Honzik, "Modelling and control of a soccer robot," in *Advanced Motion Control, 2002. 7th International Workshop on*, 2002, pp. 506–509.

[5] K.-Y. Chen and A. Liu, "A design method for incorporating multidisciplinary requirements for developing a robot soccer player," in *Multimedia Software Engineering, 2002. Proceedings. Fourth International Symposium on*, 2002, pp. 25–32.

[6] A. Pratomo, A. Prabuwono, M. Zakaria, K. Omar, M. Nordin, S. Sahran, S. Abdullah, and A. Heryanto, "Position and obstacle avoidance algorithm in robot soccer," *Journal of Computer Sciences*, vol. 6, no. 2, 2010. [Online]. Available: http://dx.doi.org/10.3844/jcssp.2010.173.179

[7] L. Jodensvi, V. Johansson, C. Lanfelt, and S. Löfström. (2015) One robot to roll them all- construction of a self-balancing, two-wheeled, football playing robot. Hämtat: 2016-04-05. [Online]. Available: http://publications.lib.chalmers.se/records/fulltext/219277/219277.pdf

[8] T. Electronics. (2015) About balanduino. Hämtat: 2016-01-29. [Online]. Available: http://balanduino.net/about-balanduino

[9] K. Lauszus. (2016) Code for balanduino. Hämtat: 2016-01-29. [Online]. Available: https://github.com/TKJElectronics/Balanduino

[10] C. Labs. (2016) Pixy cmucam5. Hämtat: 2016-01-29. [Online]. Available: http://charmedlabs.com/default/pixy-cmucam5/

[11] J.-H. Kim and D. Stonier. (2006) Fira robosot competition. Hämtat: 2016-02-04. [Online]. Available: http://www.fira.net/contents/data/RoboSot.pdf

[12] J.-H. Kim, D. Stonier, S. Choi, and N. Kuppuswamy. (2006) Fira robosot competition - version 1.0.0. Hämtat: 2016-01-29. [Online]. Available: http://fira.net/contents/data/RoboSot.pdf

[13] CMUcam. (2015) Cmucam. Hämtat: 2016-01-29. [Online]. Available: http://www.cmucam.org/

[14] C. Labs. (2016) Pixy cmucam5. Hämtat: 2016-01-29. [Online]. Available: http://www.cmucam.org/projects/cmucam5/wiki/Can_I_replace_the_lens_on_Pixy

[15] DFRobot. (2015) Analog sound sensor sku: Dfr0034. Hämtat: 2016-04-05. [Online]. Available: http://www.dfrobot.com/wiki/index.php/Analog_Sound_Sensor_SKU:_DFR0034

[16] J. K. Daftary. (2012) Calculate the distance to object from camera. Hämtat: 2016-03-11. [Online]. Available: http://www.techjini.com/blog/2012/12/24/calculate-the-distance-to-object-from-camera/

[17] F. Kjellberg, M. Resare, E. E. Johansson, S. Boström, K. Lundgren, and R. Larsson. (2016) Robotfotboll. Hämtat: 2016-04-05. [Online]. Available:

https://github.com/FreddanK/Robotfotboll

A Kravspecifikation

Nummer	Kriterier	Krav/Önskemål	Viktning	Verifieringsmetod	Målvärde	Uppfyller mål	Faktiskt mätvärde
Kategori:	**Rörelser**						
1	Roboten kan åka framåt och bakåt	K		Mäta tiden det tar att åka 5m	1 m/s	Nej	0.89 m/s
2	Roboten kan rotera på plats	K		Mäta tiden det tar att rotera 5 varv	0.5 varv/s	Ja	0.64 varv/s
3	Roboten kan åka i cirkelbåge	Ö	3	Markera cirkeln med tejp och mät tiden det tar för roboten att åka runt hela	0.5 m/s	Ja	0.71 m/s
4	Roboten kan utföra sparkrörelse	K		Verifiering, utan att ramla	Ja	Ja	Ja
	Objektidentifiering						
5	Kameran kan identifiera bollI	K		Avstånd mäts	3m	Ja	4.6m
6	Kameran kan identifiera fotbollsmålen	K		Avstånd mäts	6m	Ja	6.3m
7	Kameran kan särskilja fotbollsmålen	K		Verifiering	Ja	Ja	Ja
8	Kameran kan identifiera medspelare	K		Avstånd mäts	2m	Ja	3,5m
9	Kameran kan identifiera motspelare	K		Avstånd mäts	2m	Ja	3,5m
10	Kameran kan identifiera planens gränser	K		Avstånd mäts	1m	Ja	2,7m
11	Kameran kan bedöma avstånd till objekt	Ö	5	Verkligt avstånd mäts, jämförs med teori	Inom 10%	Ja	7%
	Kommunikation						
12	Roboten kan skicka information till en annan robot	Ö	2	Verifiering	Ja	Nej	Nej
13	Roboten kan ta emot information från en annan robot	Ö	2	Verifiering	Ja	Nej	Nej
	Taktik						
14	Roboten kan ta egna enkla taktiska beslut efter vad den ser i närheten	K		Verifiering, kan undvika hinder	Ja	Ja	Ja
15	Defensiva roboten kan skydda mål	K		Verifiering, roboten kan ställa sig framför mål	Ja	Ja	Ja
16	Defensiva roboten kan passa upp bollen till de offensiva robotarna	Ö	4	Sparka upp bollen mot zon 2 eller zon 3	Ja	Nej	Nej
	Funktioner						
17	Roboten kan uppsöka bollen	K		6m avstånd vid start, mät tiden för uppsökning	10s	Ja	8s
18	Roboten kan följa bollen på ett visst avstånd	K		Hålla bollen framför roboten på 20 cm avstånd och röra sig bort från roboten	Ja	Ja	Ja
19	Roboten kan dribbla bollen	Ö	1	Verifiering, kan dribbla med bollen	Ja	Nej	Nej
20	Roboten kan sparka bollen	K		Mät sträckan den kan sparka bollen	4m	Nej	3m
21	Roboten kan sparka bollen mot mål	K		Test hur ofta roboten sparkar bollen i mål	6 av 10	Ja	7 av 10
	Kamera-stödet						
22	Kan fästa en Pixy-kamera	K		Konstruktion	Ja	Ja	Ja
23	Tillåter kameran att tilta med servo	Ö	4	Konstruktion	Ja	Ja	Ja
24	Tillåter kameran att panorera med servo	Ö	1	Konstruktion	Ja	Nej	Nej

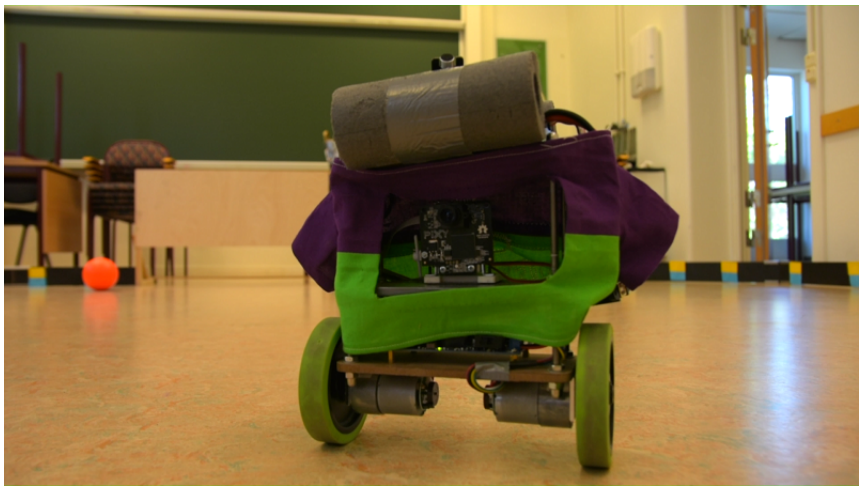
Figur 24: Kravspecifikationen uppsatt för denna produkt

B Budget

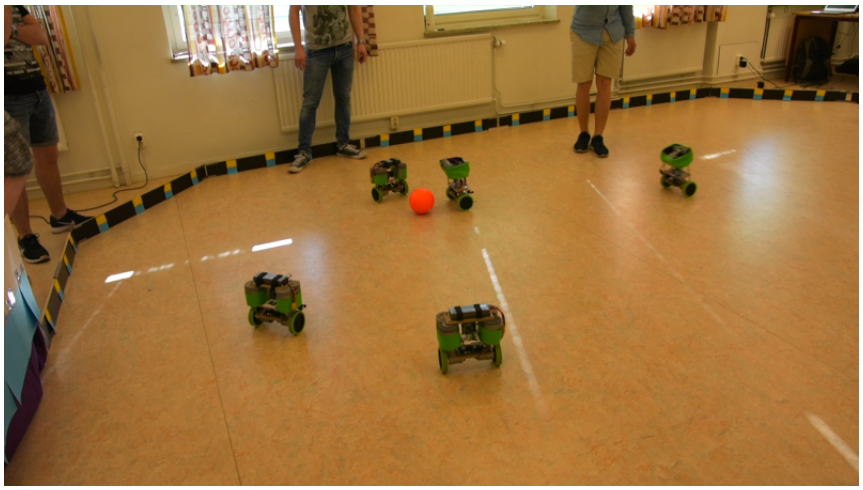
Komponent	**Antal**	**Total kostnad**
Balanduino	5	0
Kamera: Pixy CMUcam5	3	1 500 kr
Boll	1	100 kr
Mikrofoner	3	200 kr
Servo	3	200 kr
Övrigt	1	350 kr
Totalt		**2 350 kr**

C Matchbilder

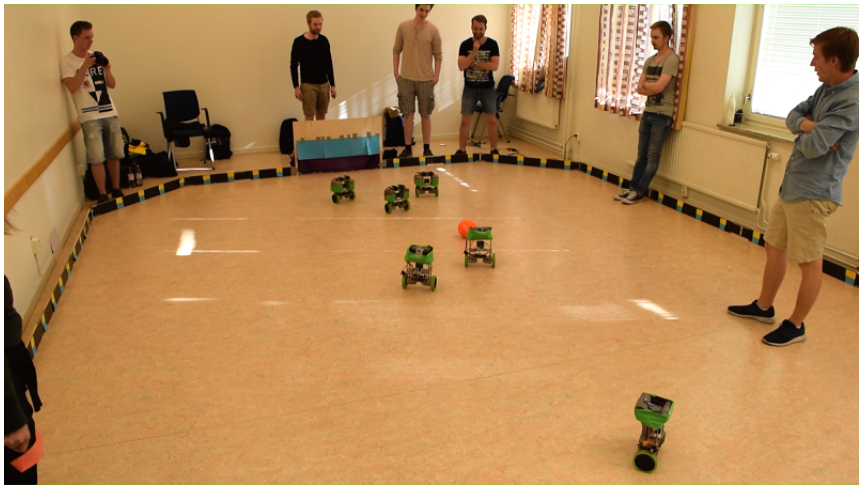
Figur 25: Robot med lagtröja sedd bakifrån



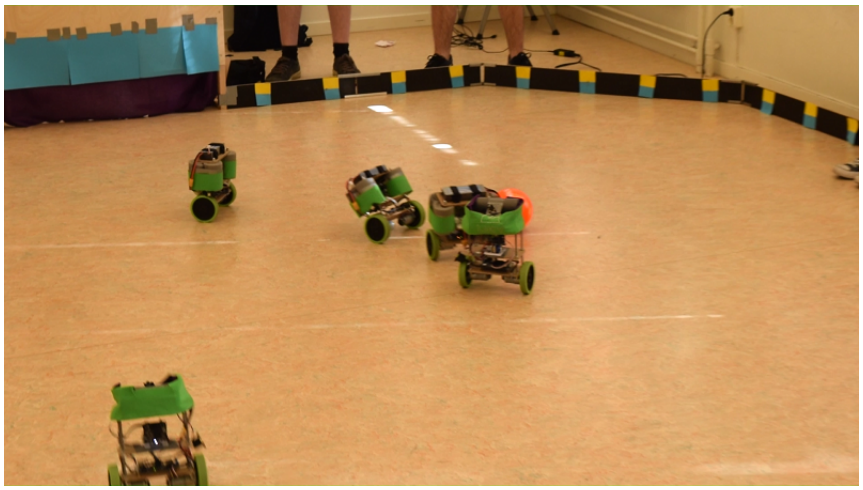
Figur 26: Robot med lagtröja sedd framifrån



Figur 27: Matchbild 1



Figur 28: Matchbild 2



Figur 29: Matchbild 3