CHALMERS

Autonoma fotbollsrobotar

Kandidatarbete för SSYX02-17-10

Simon Bastås
Anders Elfverson
Erik Henning Larsson
Måns Lerjefors
Rebecca Lippel
Sten Elling Tingstad Jacobsen

Institutionen för Signaler och system
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2017

Abstract

This is a report on the development and construction of a system for two-wheeled autonomous robots with the goal of playing football. The robots shall under given conditions be able to balance, perform movements and make decisions in order to conduct a football game. The hardware used is the Ardunio-based robot Balanduino. In addition to the robot, the camera Pixy is used for object identification and the Bluetooth modules HC-05 for wireless data transfer. Wireless transfer of data is required as the Pixy-camera is mounted above the playfield. The main focus of the project was to develop software for movements, decision making and communication. The robots can perform basic functions such as go to a determined position, shoot the ball against the goal and avoid collision. This shows that the system has potential to play a game of football. To develop the game the software needs to be expanded with more functions and tactics.

Sammanfattning

Denna rapport behandlar utveckling och konstruktion av ett system för tvåhjuliga autonoma robotar med målet att spela fotboll. Robotarna skall under givna förutsättningar klara av att balansera, utföra rörelser samt fatta beslut för att kunna genomföra en fotbollsmatch. Hårdvaran som används är den Arduino-baserade roboten Balanduino. Utöver roboten används kameran Pixy för objektidentifering och Bluetooth-modulerna HC-05 för trådlös dataöverföring. Trådlös överföring av data krävs då Pixy-kameran är monterad ovanför spelplanen. Projektets huvudfokus var att utveckla mjukvara för rörelser, beslutsfattning samt kommunikation. Mjukvara för balansering ingick i Balanduinos öppna källkod. Robotarna kan utföra grundläggande funktioner så som att åka till en bestämd position, skjuta bollen mot målet samt undvika kollision. Detta visar att systemet har potential att spela en fotbollsmatch. För att utveckla spelet behöver mjukvaran utvidgas med fler funktioner och taktiker.

Innehåll

1 Inledning

Robotar har de senaste åren blivit alltmer vanliga. De övertar arbetsuppgifter från människor som är både farliga, monotona och slitsamma. Dessa robotar syns inte bara i industrin utan även i människors vardag. Begrepp som autonoma robotar är numera välkänt i och med produkter som robotgräsklippare och robotdammsugare. Produkterna är populära tidsbesparande uppfinningar som finns i många hem. Intresset för autonoma robotar har ökat under 2000-talet [1], speciellt inom bilindustrin där mer eller mindre självstyrande bilar är en stor del av utvecklingen för många företag. Utvecklingen tyder på att området autonoma robotar kommer ha en betydande roll i framtiden.

De första fotbollsspelande robotarna byggdes 1992 av datorvetenskapsprofessorn Alan Mackworth och hans forskargrupp [2]. Robotarna styrdes med radiostyrning och fick data från en kamera i taket. De hade två olika utformningar på robotarna. Den första robottypen liknade en monstertruck där dess taktik gick ut på att om vartannat försvara mål samt åka mot boll och försöka göra mål. Detta upprepande beteende var inte särskilt önskvärt. Den andra typen var en racerbilsliknande robotmodell med mer avancerad taktik. Denna kunde evaluera situationen när den var stillastående och bestämde då om den skulle skjuta, försvara mål eller utföra någon annan aktivitet. Dock gjorde detta att om situationen ändrades från dess att roboten börjat röra sig så uppfattade inte roboten förändringen förrän nästa gång den stannade [3].

Sedan dess har det skett en stor utveckling inom området. Det finns idag två huvudsakliga organisationer inom robotfotboll, FIRA och Robocup, som har bidragit till en stor del av denna utveckling. Båda har sedan starten, vid mitten av 90-talet, anordnat årliga robotfotbollsturneringar [4] [5]. Gemensamt för både FIRA och Robocup är att de drivs i syfte att höja intresset kring teknik, robotar och artificiell intelligens. Detta genom att sammanföra studenter och forskare med kunskap inom mekatronik, elektronik, bildbehandling och kommunikation för att tävla i robotfotboll [6] [7]. Fotbollsmatcherna är indelade i olika klasser så som människoliknande robotar, att spela en datorsimulerad fotbollsmatch eller en klass där alla tävlande använder samma typ av robot [8] [9]. Robocup har som mål att ett robotfotbollslag skall vinna en match mot ett officiellt fotbollslag bestående av människor innan 2050.

Det har tidigare gjorts kandidatarbeten inom robotfotboll. År 2015 byggdes en tvåhjulig robot som inte lyckades balansera [10]. Året efter gjordes två kandidatarbeten i robotfotboll där de använde en färdigbyggd robot kallad Balanduino där kod för balansering ingick. De spelade en simpel fotbollsmatch mot varandra med ett blandat resultat [11] [12].

Detta kandidatarbetet, liksom föregående år, baseras på Balanduino-roboten. Det är en robot med två hjul som kan liknas vid en inverterad pendel, vilket är ett välkänt reglertekniskt problem. Utöver det reglertekniska arbetet, består också en stor del av projektet i att implementera ett sensorsystem för att robotarna skall kunna navigera i sin omgivning. Resultat och diskussion från tidigare års arbeten används som utgångspunkt för att utveckla projektet.

1.1 Syfte

Syftet med projektet är att utveckla robotar som kan spela en fotbollsmatch autonomt på en specifik spelplan. Detta skall ske genom att robotarna fattar egna beslut baserat på situationen. Varje individuell robot måste därför kunna utföra beräkningar på indata för att ett spel skall kunna uppstå.

1.2 Problembeskrivning

Robotarna skall klara vissa grundläggande situationer för att fotbollsspel skall kunna uppstå. De skall kunna åka till en given position, undvika att åka in i sargen samt att kollidera med andra robotar. Baserat på situationen skall roboten dessutom kunna ta offensiva eller defensiva beslut där den antingen skall försöka göra mål eller försvara sitt eget mål. För att robotarna skall klara dessa situationer behöver tre huvudproblem lösas. Hur roboten skall manövrera, hur objekt identifieras samt hur roboten skall kunna ta rätt beslut i olika situationer.

1.2.1 Rörelser

Robotarna skall kunna balansera medan de åker framåt, bakåt eller roterar. De skall även kunna färdas givna sträckor och rotera ett uträknat antal grader.

1.2.2 Objektidentifiering

En sensor behöver kunna identifiera flera objekt samtidigt samt spara objektens position. De objekt som sensorn behöver känna igen är boll, robotar och spelplan. Information om objekten måste kunna överföras till robotarna.

1.2.3 Beslutsfattning

Roboten behöver kunna ta olika beslut beroende på informationen den får från sensorn. Baserat på avstånd mellan roboten och olika objekt skall roboten till exempel försvara, försöka göra mål eller undvika att krocka.

1.3 Avgränsningar

Då kameran som valts för objektidentifiering är ljuskänslig krävs ett konstant ljusförhållande, därav avgränsas fotbollsspelandet till ett specifikt rum. Projekt avser alltså inte att utveckla robotar som kan spela fotboll i flera olika miljöer.

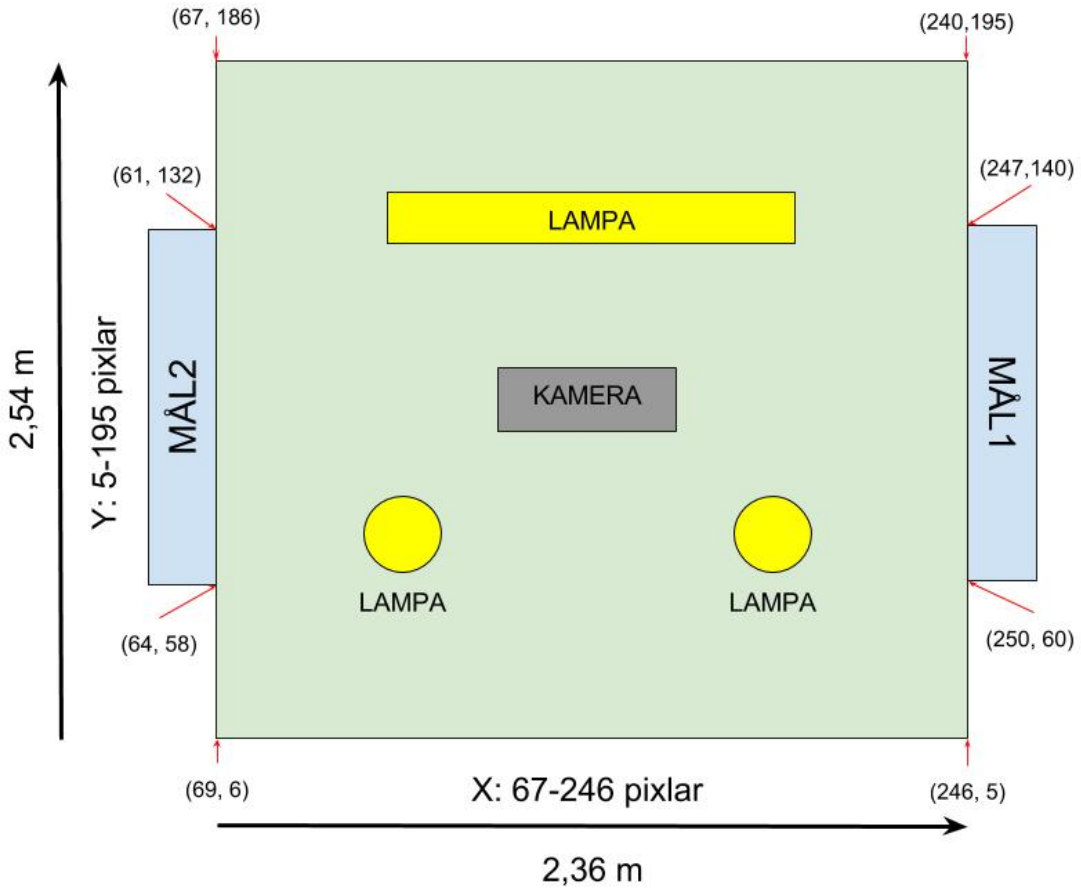
För att fokusera på mjukvaruutveckling och ett kommunikationssystem byggs inte egna robotar från grunden utan Balandunio används. Utöver detta utvecklas ingen kod för objektidentifiering utan en kamera med färdig mjukvara har valts.

2 Konstruktion

I detta kapitel beskrivs konstruktionen utav systemet vilket innefattar kamera, spelplan samt robotkonstruktion. Valet av lösning grundades i förra årets kandidatarbeten. Båda projekten hade problem med varierande ljusförhållanden eftersom de ej hade en fast spelplan. Detta ledde till att man ständigt behövde göra förändringar i inställningarna för deras objektidentifierande kamera [11] [12]. Av den anledningen användes en fast spelplan där ett konstant ljusförhållande kunde skapas. En bestämd spelplan gav också möjligheten till en ny kamerakonfiguration. Istället för att ha en kamera på varje Balanduino, som var fallet föregående år, så valdes det att ha en kamera monterad i taket. Den stora fördelen med en lösning av det här slaget är att man får en mer övergripande bild av spelplanen. En kamera i taket innebär alltså att kameran inte ständigt behöver leta efter nya objekt eftersom samtliga objekt ständigt är inom kamerans synfält. Det resulterar även i att kamerans information måste skickas trådlöst till robotarna. Hela konstruktionen av fotbollsspelets alla komponenter beskrivs i följande avsnitt.

2.1 Spelplanen

Spelplanen är uppsatt i ett rum där sarger och två mål placerats ut. I taket har en kamera monterats tillsammans med en Bluetooth-modul som är sammankopplade genom en Arduinio. Två Fresnel-lampor har monterats i taket för att få mer och jämnare ljus i rummet då kameran behöver goda ljusförhållanden. I rummet ligger det en matt mörkblå matta över hela golvet. Mattan motverkar reflektioner som kan störa ljusförhållandet på spelplanen. I figur 1 kan man se en ritning över spelplanen. Viktiga koordinater så som spelplanens hörnkoordinater samt koordinaterna för målens stolpar är utmarkerade. Dessa kommer användas i programmeringen.



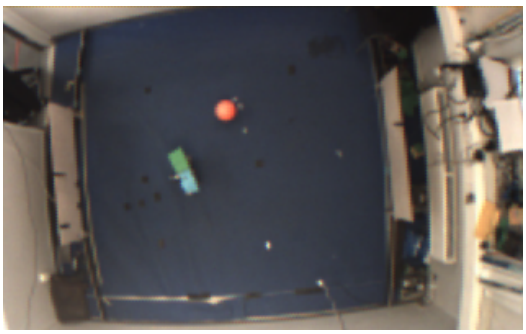
Figur 1: Fotbollsplanen sedd ovanifrån. Kamera och lampor är monterade i taket. Målen är placerade på spelplanens kant. Författarnas egna bild.

2.1.1 Pixy

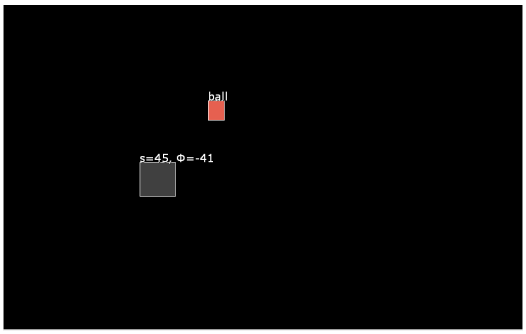
För att identifiera olika objekt har kameran Pixy använts, i figur 2 kan en sådan ses. Pixy består utav en kamera samt en integrerad microkontroller som klarar att processera enkel bildbehandling. Dess enkla implementering gör den lämplig för projektet [14]. Pixy-kameran kan identifiera sju olika färgsignaturer; röd, orange, gul, grön, cyan, blå och lila. Detta gör att Pixy kan identifiera 7 unika objekt med de grundläggande färgsignaturerna. Genom att kombinera två eller flera färger kan man dock få Pixy att identifiera ännu fler unika objekt, detta kallas färgkodning. Med färgkodning får man även data om vid vilken vinkel ett objekt befinner sig i förhållande till kamerans referensvinkeln. Andra parametrar man får ut från Pixy är vid vilken x- och y-position objektens mittpunkt befinner sig, samt objektens bredd och längd. Dessa parametrar anges i enheten pixel. Upplösningen på kameran är 320×200 pixlar. Pixy-kameran tar 50 bilder per sekund vilket gör att kameran är tillräckligt snabb för att till exempel identifiera en studsande boll [15].

Figur 2: Figuren visar en Pixy kamera. Från [13], Återgiven med tillstånd.

I figur 3 visas spelplanen från Pixy-kamrans yv och i figur 4 visas Pixy-kamerans identifikationsläget.

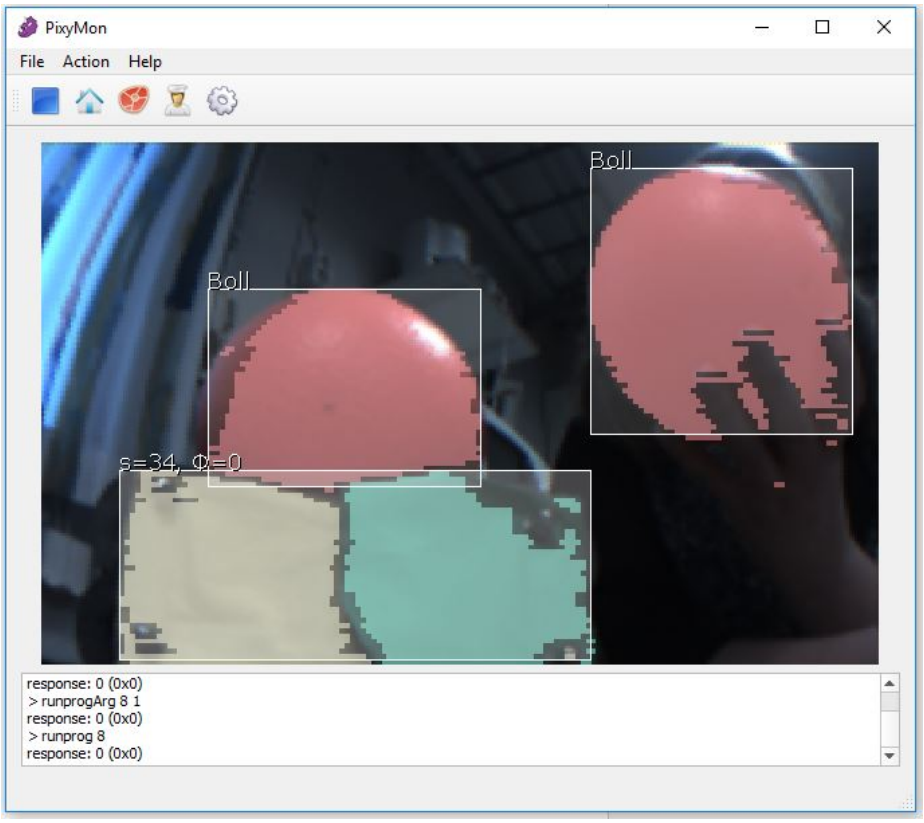


Figur 3: Bild från Pixy i matchsituation. Författarnas egna bild.



Figur 4: Pixyn identiferade objekt i matchsitaution. Författarnas egna bild.

Mjukvaran PixyMon används för att enkelt ställa in och optimera kamerans färg och ljusförhållanden för att optimera identifikationen av objekten i fotbollsspelet [15]. Figur 5 visar hur programmet PixyMon ser ut när en Pixy-kamera är inkopplad. Pixy-kameran i denna figur identifierar tre olika objekt, ett via färgkodning och två via färgsignatur.



Figur 5: Vy av PixyMon där de oranga bollarna är identifierade via färgsignatur och en robot via färgkodning. Författarnas egna bild.

2.1.2 Selecon Acclaim Fresnel 650W

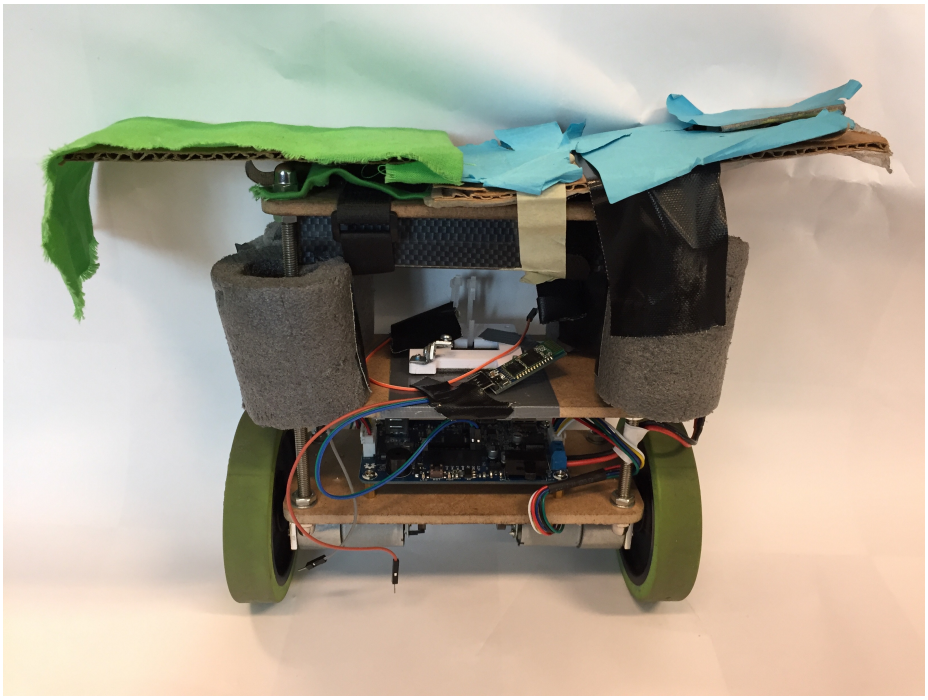
Ett jämnt och starkt ljus är viktigt för att Pixy-kameran skall fungera bra då mörka eller blänkande ytor kan leda till att kameran uppfattar en felaktig nyans. Två Fresnel-lampor används för att bilda ett jämnt ljusflöde i hela rummet så objekten ser lika ut för kameran i olika delar av rummet. Genom att ha flera ljuskällor minskar man eventuella skuggor, från till exempel boll och robotar, som även de hade kunnat påverka objektidentifieringen. I figur 6 kan en Fresnel-lampa ses.

Lamporna har en ställbar spridningsvinkel mellan $6°$-$60°$ och en ljusstyrka på 14500 lumen var. På lamporna finns spadar som är till för att skära av ljuset så lamporna endast lyser upp det önskade området. Till lamporna används en dimmer för att reglera ljusstyrkan till en nivå som skapar ett jämnt ljusförhållande.

Figur 6: Selecon Acclaim Fresnel. Återgiven med tillstånd.

2.2 Robotkonstruktion

Robotarna, som skall agera fotbollsspelare, baseras på en Balanduino-sats. På varje robot har en Bluetooth-modul kopplats på för att kunna ta emot information från Pixy. För att Pixy-kameran i taket skall kunna identifiera de olika robotarna har tygstycken fästs ovanpå roboten. Även skumplast har monterats på robotarna för att dämpa krockar och minska skador. I figur 7 kan den färdiga robotkonstruktionen ses.



Figur 7: Den färdiga robotkonstruktionen. Författarnas egna bild.

2.2.1 Balanduino

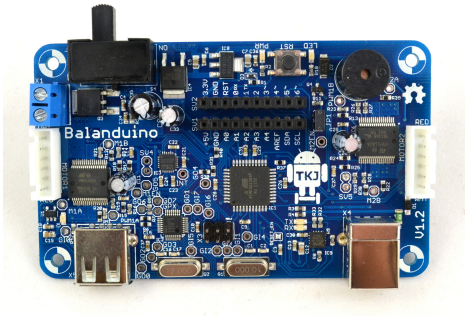
Balanduino-satsen är tillverkad av TKJ Electronics och en bild på en ihopbyggd Balanduino-sats kan ses i figur 8. Roboten kan liknas vid en inverterad pendel, vilket är ett instabilt system som aktivt behöver regleras för att inte välta. Denna sats kommer med ett Arduino-baserat kretskort med en Atmel 8-bit ATmega644A AVR microcontroller med en klockfrekvens på 8 Mhz. I figur 9 kan ett sådant Arduino-kort beskådas. På kretskortet sitter även en IMU (Inertial Measurment Unit) vilken består av en 3-axlig accelerometer för att mäta vinkel och ett 3-axligt gyroskop för att mäta vinkelacceleration. Dessa används för att beräkna robotens lutning och kompensera styrsignalen efter det. Kretskortet har även digitala och analoga in- och utgångar för att styra externa komponenter.

Två 12 V DC-motorer är fästa undertill och kopplade till varsin rotationskodare som kan användas för att beräkna hur långt Balanduinon har förflyttat sig. Rotationskodarna avger 64 pulser per varv, och med motorns utväxling på 30:1 leder detta till 1920 pulser per hel rotation utav ett hjul [16].

Inkluderat i Balanduino-satsen finns även öppen källkod tillgänglig via TKJs GitHub att hämta för att få roboten att balansera samt funktioner för att styra roboten via en extern kontroll eller en smarttelefon [17]. Denna kod används som grund för projektet. Koden för balansering innefattar en förinställd PID-regulator samt ett implementerat Kalman-filter. Kalman-filtrets funktion är att kompensera för osäkerheter i mätdata, då analog elektronik ger upphov till olika typer av brus [18]. Elektroniskt brus tenderar att vara normalfördelat, vilket innebär att det är jämnt fördelat kring den faktiska signalen, som är en förutsättning för filtrets funktionalitet [19]. Vid direkt mätning utav accelerometern och gyroskopet visas osäkra värden. Kalman-filtret uppskattar mätvärdet baserat på tidigare tillstånd samt aktuella mätningar. Detta resulterar i mer pålitlig data som behövs för att kunna balansera.

Koden modifieras för att kunna lägga in de funktioner som behövs för att spela en fotbollsmatch. Det innefattar kod för kommunikation, rörelser samt beslutstagande.

Figur 8: En Balanduino robot. Från [20], Återgiven med tillstånd.



Figur 9: Ett Balanduino kretskort. Från [20], Återgiven med tillstånd.

2.2.2 Spelaridentifiering

Tyget, som ovansidan av varje robot är inklätt i, har två olika färger. Detta för att kunna använda Pixys färgkodnignsfunktion. Tyg har valts då det är ett icke reflekterande material, detta är en nödvändig materialegenskap då kameran inte klarar av att känna igen objektet om stora förändringar i färgernas nyans sker. Varje robot har en unik kombination av färger för att Pixy-kameran skall kunna skilja robotarna åt. I figur 10 kan två robotars olika färgkodning ses.

Figur 10: Exempel på färgkodning på två olika robotar. Författarnas egna bild.

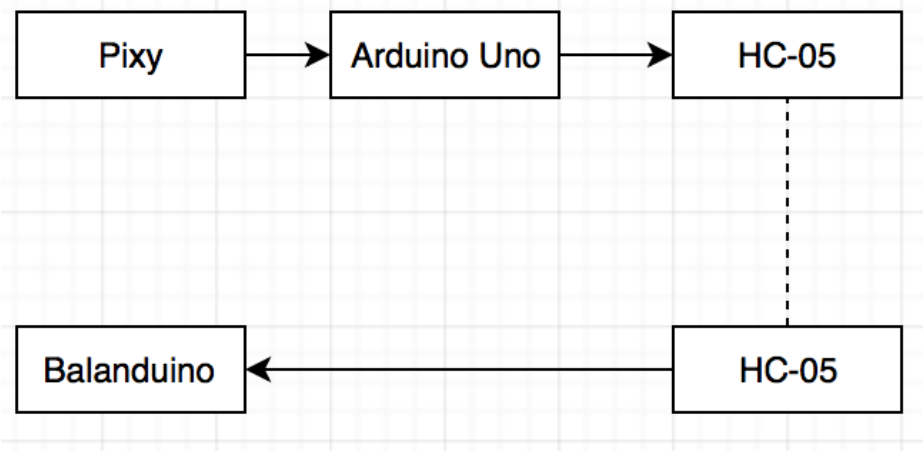
2.3 Kommunikation mellan Pixy-kameran och robotarna

För att robotarna skall kunna få information från Pixy-kameran har en Bluetooth-lösning valts. Modulen som har valts för detta projektet är anpassad för Arduino och heter HC-05. HC-05, som visas i figur 11, kan både skicka och ta emot Bluetooth-signaler.

Figur 11: En HC-05 Bluetooth-modul. Författarnas egna bild.

Då Bluetooth-enheterna parkopplas måste varje robot ha en dedikerad sändare som är kopplad till Pixy-kameran. Sändaren förmedlar information om positioner och vinklar till sin parkopplade enhet som är placerad på roboten. För ett fall där man spelar med två robotar måste man totalt ha fyra Bluetooth-enheter.

Bluetooth-modulen har en egen dator och datan lagras på den tills den överförs via seriell kommunikation till Balanduino. Ett flödesschema som visar hur information skickas från Pixy-kameran till Balanduino visas i figur 12.



Figur 12: Flödesschema över Bluetooth-kommunikation där heladragna pilar betyder att det är seriell kommunikation och streckad linjer Bluetooth-kommunikation.

3 Matematiska modeller

För att kunna implementera metoder för robotarnas rörelser i mjukvaran krävs modeller för hur roboten rör sig. Nedanför beskrivs de modeller och ekvationer som används i, eller varit grund för, programmeringen som behövs för ett fungerande spel.

3.1 Längdberäkning för Balanduino

Med datan från rotationskodarna räknas det önskade avståndet ut. Hjulens diameter är 9,85 cm. Omkretsen av hjulen beräknas då till 30,92 cm med hjälp av följande samband:

$$O = \pi \cdot d \tag{1}$$

Längden roboten har färdats räknas ut med följande formel:

$$L = \frac{O \cdot P_{uppmatt}}{P_{varv}} \tag{2}$$

Där L är längden roboten färdats i centimeter, O är hjulens omkrets i centimeter, $P_{uppmatt}$ är medelvärdet av antalet pulser som avläses från båda rotationskodarna. P_{varv} är en konstant med värdet 1920 vilket motsvarar antalet pulser en rotationskodare ger på ett varv. Således blir formeln en procentsats av hur många varv hjulen roterat multiplicerat med omkretsen för hjulen.

3.2 Rotationsberäkning för Balanduino

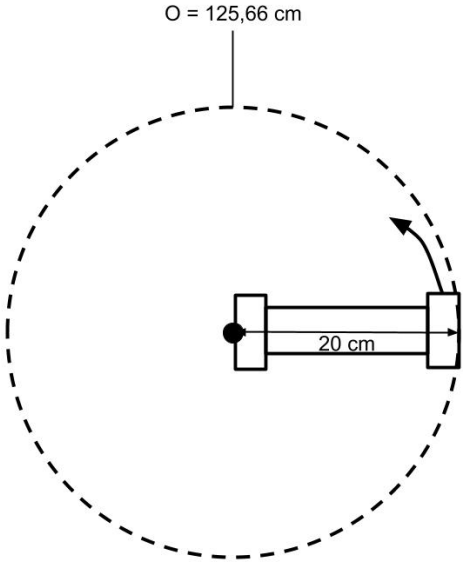
Utöver att åka rakt fram en viss sträcka är det nödvändigt för robotarna att kunna rotera ett givet antal grader. Robotarna har en hjulbas på 20 cm, vilket gör att cirkeln som skapas då roboten roterar runt ett hjul är 40 cm i diameter. Cirkelns omkrets beräknas med ekvation 1 till 125,66 cm. Då hjulens omkrets är 30,92 cm krävs det 4,06 hjulvarv för att rotera $360°$. Eftersom ett varv motsvarar 1920 pulser på rotationsavkodaren går det 7803 pulser på $360°$, eller 21,67 pulser per grad. Hur mycket roboten roterat beräknas med

$$\theta = \frac{P_V + P_H}{21,67} \tag{3}$$

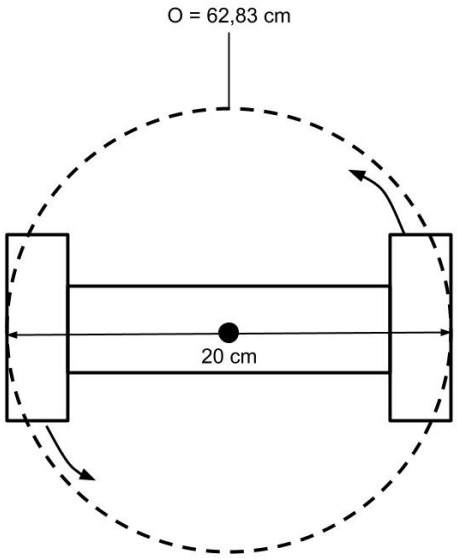
Där P_V är det vänstra hjulets rotationskodare och P_H det högra hjulets rotationskodare. Båda hjulens rotationskodare jämförs för att roboten roterar runt olika jämviktspunkter. Däremot blir den totala sträckan de båda hjulen färdats alltid 125,66 cm vid rotation $360°$ enligt

$$\begin{cases} y = (40 - x) \\ S = x \cdot \pi + y \cdot \pi \end{cases}$$

där x motsvarar radie från ena hjulet till jämviktspunkten och y det andra. S är sträckan som roboten förflyttats vid rotation. Figur 13 och Figur 14 illustrerar de beskrivna ekvationerna.



Figur 13: Robotens högra hjul roterar längs den streckade cirkeln medan det vänstra hjulet står stilla i cirkelns mittpunkt. När det högra hjulet roterat $360°$ har cirkeln ritats en gång. Cirkelns omkrets är 125,66 cm vilket leder till att hjulen sammanlagt har roterat 125,66 cm. Författarnas egna bild.



Figur 14: Robotens högra och vänstra hjul roterar längs den streckade cirkeln medan robotens mittpunkt roterar kring cirkelns mittpunkt. När båda hjulen roterat $360°$ har cirkeln ritats två gånger. Cirkelns omkrets är 62,83 cm men eftersom bägge hjulen färdats denna strecka blir den totala sträckan som hjulen tillsammans roterat 125,66 cm. Författarnas egna bild.

3.3 Olika rotation per kvadrant

När vinkeln mellan roboten och önskad position beräknas måste man ta hänsyn till vilken del av spelplanen som roboten befinner sig i relativt sin målposition. Vinkeln, betecknad θ, mellan önskad position och robotens position beräknas med hjälp av arcus tangens av differensen i x- och y-led mellan dessa positioner enligt ekvation 4, denna vinkeln visas i figur 15. Roboten är tänkt att befinna sig i mitten av figur 15 och önskad position i någon av kvadranterna.

$$\theta = \arctan \frac{|posRobot_x - posGoTo_x|}{|posRobot_y - posGoTo_y|} \tag{4}$$

För att bestämma den vinkel som roboten skall rotera måste θ från ekvation 4 göras om till en vinkel i robotens koordinatsystem, som kan ha en vinkel mellan $0°$ till $360°$.Vinkeln γ är den önskade positionens vinkel från $0°$ och ges av följande ekvationer:

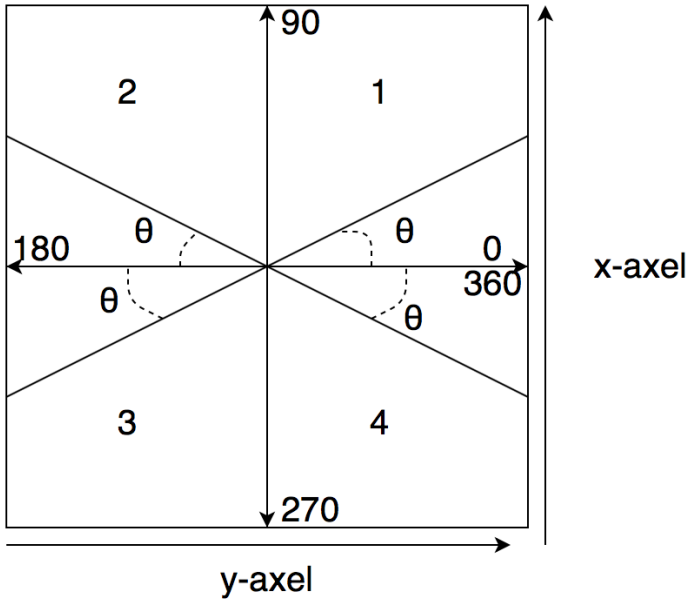
$$\gamma_{kvad1} = \theta \tag{5}$$

$$\gamma_{kvad2} = 180° - \theta \tag{6}$$

$$\gamma_{kvad3} = 180° + \theta \tag{7}$$

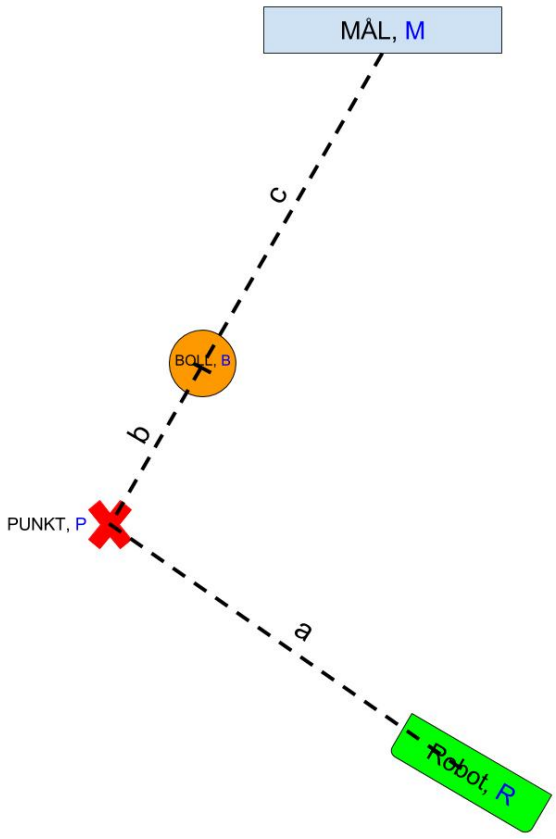
$$\gamma_{kvad4} = 360° - \theta \tag{8}$$

Beroende på i vilken kvadrant den önskade positionen är får man olika uträkningar på vinkeln från $0°$ till positionen i figur 15 enligt ekvation 5,6,7 och 8. Den vinkel som roboten skall rotera är robotens vinkel subtraherat med γ.



Figur 15: Figuren visar spelplanen från ett fågelperspektiv där x- och y-axeln visas i bilden. De inre pilarna visar i hur robotens kordinatsystem är roterat. Dessutom illusteras θ från ekvation 4. Författarnas egna bild.

3.4 Förflyttning till skottposition



Figur 16: Förflyttning av roboten till lämplig skottposition. Författarnas egna bild.

För att en robot skall kunna göra mål måste den först ta sig till en bra skottposition. En bra skottposition P avser att roboten befinner sig på längden b bakom bollen samt att skottpositionen P är i linje med boll och mål. Sträckan b används för att accelerera till en högre hastighet för att motsvara en skottrörelse.
För att positionera sig korrekt används de relevanta objektens position som fås av Pixykameran. Med den informationen beräknas avstånd, vinklar och riktningsvektorer. Slutligen kan en bana beräknas som tar roboten till en önskad slutposition och slutvinkel, detta ses i figur 16.
Eftersom vi får positionerna för både bollen och roboten från kameran blir det få beräkningar som krävs. Målets position som alltid är densamma har därför fasta koordinater.

Först beräknas vektorn från målet till bollen enligt

$$\overrightarrow{MB} = \begin{bmatrix} x_{boll} \\ y_{boll} \end{bmatrix} - \begin{bmatrix} x_{mal} \\ y_{mal} \end{bmatrix} = \begin{bmatrix} x_{MB} \\ y_{MB} \end{bmatrix}$$

och

$$\widehat{MB} = \frac{1}{\sqrt{x_{MB}^2 + y_{MB}^2}} \begin{bmatrix} x_{MP} \\ y_{MB} \end{bmatrix}$$

där $\widehat{MB}$ är en enhetsvektor och är alltså riktningen av $\overrightarrow{MB}$. Med riktningsvektorn och bollens position kan den önskade positionen P räknas ut via

$$P = \begin{bmatrix} x_{boll} \\ y_{boll} \end{bmatrix} + b \times \frac{1}{\sqrt{x_{MB}^2 + y_{MP}^2}} \begin{bmatrix} x_{MB} \\ y_{MB} \end{bmatrix}$$

där b är det önskade avståndet från bollen. Avståndet mellan robotens position och position P tas fram med ekvation 9 och vinkeln mellan dem fås med ekvation 4. Sedan används kvadrantberäkningarna från kapitel 3.3 för att bestämma den korrekta vinkeln.

$$\overrightarrow{PR} = \begin{bmatrix} x_{pos} \\ y_{pos} \end{bmatrix} - \begin{bmatrix} x_{robot} \\ y_{robot} \end{bmatrix} = \begin{bmatrix} x_{PR} \\ y_{PR} \end{bmatrix} \tag{9}$$

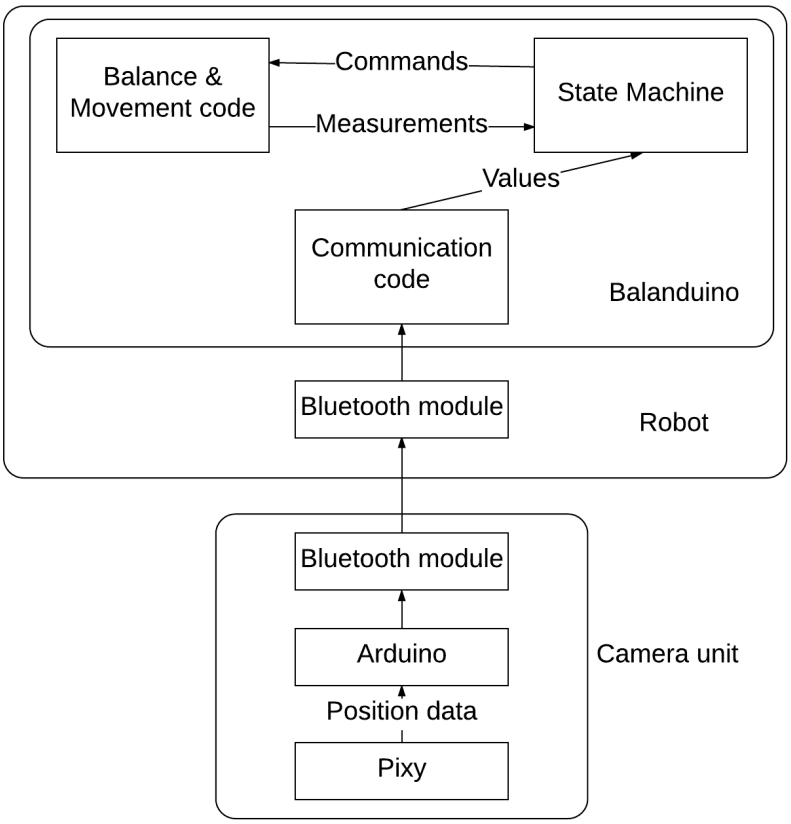
Dessa beräknade avstånd och vinklar är det som behövs för att roboten skall kunna ta sig till en bra skottposition.

4 Mjukvara

I detta kapitel beskrivs den mjukvara som används i projektet. Kapitlet beskriver de funktioner som krävs för att robotarna skall kunna spela fotboll. De funktioner som beskrivs är beslutsfattningssystemet, rörelsefunktioner, objektidentifieringen, reglering samt den trådlösa kommunikationen. Alla ekvationer från de matematiska modeller som beskrivs i kaptitel 3 används för att tillämpa dessa funktioner.

4.1 Överblick av mjukvaran i systemet

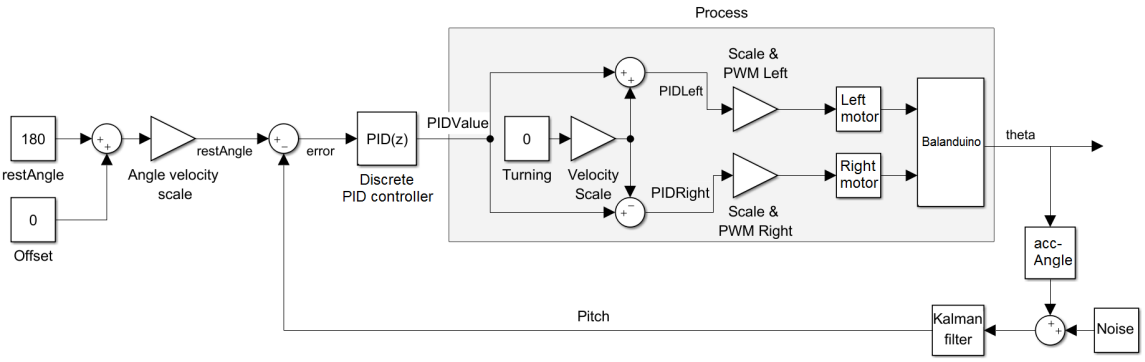
Mjukvaran i systemet illustreras i figur 17. Systemet kan delas upp i en kameraenhet och en robot. Kameraenheten består av en Pixy med standardmjukvara, en Arduino som fungerar som en databuffert mellan Pixy och Bluetooth-modulen. Bluetooth-modulen har mjukvara som skickar data ner till Bluetooth-modulen på roboten. Bluetooth-modulen på roboten har mjukvara som tar emot data från kameraenheten och skickar den vidare till Balanduino-kretskortet. På Balanduino-kretskortet finns mjukvara som sköter mottagandet av data, Communication code i figuren. Communication code samlar den inkomna datan i variabler som beteende-algoritmen, State Machine, kan använda. State Machine tar med hjälp av datan från Pixy samt mätdata från balanseringsmjukvaran, Balance & Movement code, fram vilka komandon som ska skickas till balanseringsmjukvaran. Balance & Movement code tar de komandon den får från State Machine och omsätter de till spänning över motorerna, den tar även hand om mätvärden från sensorer vid motorerna och gör den datan tillgänglig för StateMachine.



Figur 17: Illustration av vart all mjukvara i systemet befinner sig och hur de olika delarna relaterar till varandra. Författarnas egna bild.

4.2 Reglering

För att hålla roboten stående med dess instabila system stående krävs aktiv reglering av utsignalen. Systemet i sin helhet kräver reglering utav vinkel och position, alternativt vinkel och hastighet, för att balansera utan att förflyttas. Vid ett helt lodrätt tillstånd i teorin, med försummad friktion och luftmotstånd, kan den fortfarande röra sig med en konstant hastighet. Därav krävs en kaskadkopplad lösning, där en PID-regulator utan lågpassfiltrering hanterar vinkel och en P-regulator som hanterar position eller hastighet.



Figur 18: Blockschema över reglering där man önskar ett stillastående tillstånd. Författarnas egna bild.

PID-regulatorn i figur 18 beskrivs enligt följande ekvationer

$$P = error \cdot K_P$$

$$I = I_{-1} + K_I \cdot error \cdot h$$

$$D = \frac{K_D \cdot (error - error_{-1})}{h}$$

$$PIDValue = P + I + D$$

Systemet regleras kring en referensvinkel, $restAngle$, som vid stillastående är satt till $180°$ för att hålla roboten vertikal. Avläsningen av vinkeln, $accAngle$, samt vinkelaccelerationen behandlas utav Kalmanfiltret och ger den uppskattade vinkeln $Pitch$ vilken är mer pålitlig. $Pitch$ subtraheras från $restAngle$ och kvarvarande felet, $error$, skickas igenom regulatorn vilket resulterar i en styrsignal, $PIDValue$, som skalas om till ett procentuellt värde mellan 0 till 100 och därefter pulsbreddsmoduleras till motorerna. Stegtiden h uppdateras vid varje iteration utav avläsningen från IMU och är uppmätt till cirka 4 ms, vilket betyder att styrsignalen kan uppdateras 250 gånger per sekund.

För att få roboten att röra sig framåt eller bakåt sätts en förskjutning av referensvinkeln, $offset$. Detta ger en lutning och därav en tyngdpunkt som ligger utanför hjulaxeln. Den förflyttade tyngdpunkten måste motorerna konstant kompensera för genom att accelerera i lutningens riktning därav uppkommer en hastighet. $Offset$ skalas dessutom baserat på hastigheten där en högre hastighet leder till lägre förskjutning för att nå en konstant hastighet. Systemet är begränsat till en maximal förändring på önskad lutning på 1° per loop.

För att framkalla en rotation adderas och subtraheras ett önskat värde $turning$ från styrsignalen för respektive motor. Detta för att ge upphov till olika hastigheter på hjulen vilket leder till en svängande rörelse. Värdet $turning$ skalas även det av hastigheten för att kunna svänga stabilt.

Den yttre loopen är endast aktiv då man begär ett stillastående tillstånd. När positionsregleringen slås på kan positionen sparas. Roboten kan sen justera vinkeln för att stanna vid denna position. Denna funktion används ej då det tar för lång tid att justera så att roboten hamnar på exakt rätt position. Istället approximeras bromssträckan och tas hänsyn till vid beräkningen för hur långt roboten skall åka. Detta innebär att vi får ett visst fel på några cm men det kan kompenseras för i nästkommande rörelseberäkning.

4.3 Trådlös kommunikation

Kommunikationslösningen som tagits fram bygger på att en rad olika datorer snabbt kan få fram ett medelande utan betydande informationsförluster. Det är nödvändigt att ha flera datorer då varken Pixy eller Balanduino själva har Bluetooth-komponenter. Kommunikationen sker trådlöst eftersom att kameraenheten inte sitter på roboten.

Arduino Uno agerar som en central enhet för Pixy-kameran och Bluetooth-modulerna i taket. På denna finns den kod som hämtar data från Pixy och skickar datan via Bluetooth till Balanduino. Färdiga funktioner används för att hämta datan från Pixy med hjälp av ett bibliotek som Pixy-tillverkarna skrivit för Arduino. Den data som hämtas är en sträng innehållande vilket objekt informationen rör, objektets position i x- och y-led samt vinkel om objektet är en robot. Denna sträng skickas sedan till Bluetooth-enheten med hjälp av funktionen $print$ som ingår i Arduinobiblioteket. Enheten skickar sedan denna data till robotarna vilket tar cirka 80 ms. Detta gör att avläsningen på Balanduinon måste synkroniseras med när en data-sträng skickas från Arduino. Detta görs med en inbyggd funktion som heter $available$ som kollar om kommunikationen är tillgänglig, om den är det värden läsas av.

4.4 Huvudprogrammet på Balanduino

Huvudprogrammet på Balandruinokretsen utgår från två olika funktioner, *setup* och *loop*. Först exekveras *setup* 1 gång och sedan itereras *loop* tills programmet stängs av. *Loop* kallar på funktioner som balanserar roboten, styr robotens beteende samt sköter kommunikationen med Bluetooth-modulen.

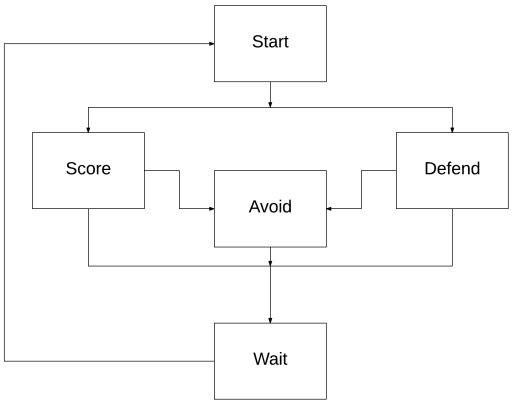
```
setup():
  Initiera rotationskodare
  Initiera motor
  Initiera IMU
  Kalibrera gyro
  Återställ motorer
loop():
  Om det har gått 80 millisekunder
      Hämta data från Bluetooth
  Kör tillståndsmaskinen, ta beslut baserat på tillgänglig
information.
  Beräkna robotens vinkel
  Beräkna styrsignalen baserat på målvärden från tillståndsmaskinen
  Skicka styrsignalen till motorerna
  Uppdatera värderna från rotationskodarna
```

Figur 19: Pseudokod för huvudprogrammet

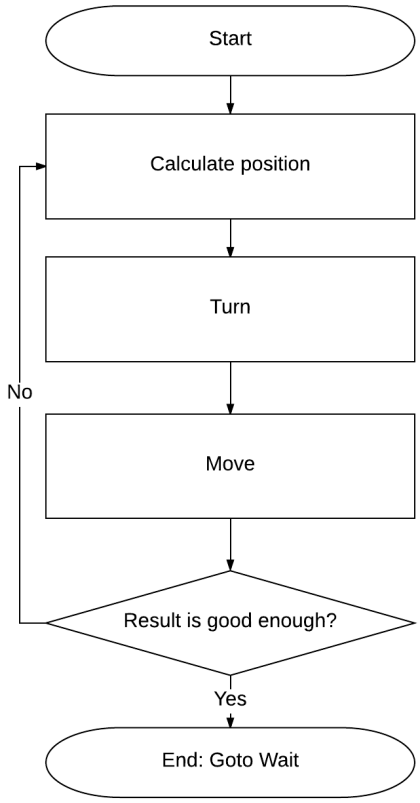
4.4.1 Beteendealgoritmen - State-machine

För att organisera beteendealgoritmen användes en tillståndsmaskins-lösning. Detta innebär att olika tillstånd har introducerats och en variabel bevakar vilket tillstånd som är aktuellt. När ett tillstånd är aktuellt exekveras den kod som är associerad med tillståndet samt en del test som undersöker ifall ett annat tillstånd borde bli aktuellt. Robotens olika taktiska och rörelsemässiga beteenden delas upp i uppgifter som tilldelades tillstånd som sedan byts emellan när dessa olika krav uppfyllts.

Koden innehåller flera olika parallella tillståndsmaskiner som exekveras tillsammans. En för övergripande taktiska beslut, planState. En för grundläggande rörelsekod, moveState, samt en för varje taktiskt beslut, till exempel scoreState. De sistnämnda aktiveras av den övergripande taktiska tillståndsmaskinen och de går igenom delmoment i respektive aktiviteter. PlanState och scoreState illustreras i figur 20 samt 21.
När roboten befinner sig i dessa tillstånd exekveras andra delar av koden, underliggande tillståndsmaskiner, som sköter specifika delmoment till uppgiften. Dessa ser till att roboten vet hur långt den har kommit med uppgiften och innehåller tester som ser till att varje delmoment klaras tillräckligt väl. I de flesta fall är roboten programmerad att försöka börja om ifall ett delmoment inte klarats av ordentligt.
Roboten kan ta en offensiv eller defensiv roll beroende på hur situationen. Om roboten befinner sig närmare bollen än motståndaren skall roboten ta sig till en bra skottposition och skjuta bollen mot mål. Om roboten däremot befinner sig längre bort från bollen än motståndaren skall den ta en defensiv roll och försvara sitt eget mål.



Figur 20: Illustration av de olika tillstånd som bestämmer beteendealgoritmens övergripande beteende, dessa kallas för ”planStates’ och består av Score, Defend och Avoid. Start är en beslutfattnings kod som bestämmer vad ”planeState” ska sättas till Defend eller Score. Avoid kollas kontinuerligt och ”planState” ändras till Avoid om roboten är för nära en annan robot eller kanten på spelplanen. Författarnas egna bild.



Figur 21: Illustration av aktionsflödet som följer beslutet ”Score”. (ScoreState) Författarnas egna bild.

Eftersom projektets spelplan är liten appliceras bara spel en mot en och därför finns det ej några dedikerade offensiva eller defensiva spelare. Robotarna anpassar sig istället efter situationen och tar defensiva alternativt offensiva beslut.

4.4.2 Informationshämtningens påverkan på balanseringen

För att hämta data krävs att roboten läser den seriella kommunikationen från den mottagande Bluetooth-modulen till Balanduino-kretskortet. Medan avläsningen exekveras kommer systemet ej att regleras då Balanduino-kretskortet har en enkärnig processor som ej klarar att köra parallella program. För att detta inte skall påverka funktionaliteten behöver man veta hur länge man kan köra balanseringskoden innan man bör byta till informationshämtning och vice versa. Tiden för respektive funktion är framtagen med iterativ testning, där kravet var att roboten skulle balansera stabilt samtidigt som den mottog data tillräckligt ofta. Testningen resulterade i att balanseringskoden skall köras i 80 ms då hämtning av information tar cirka 86 ms.

4.5 Grundläggande hjälpfunktioner

I mjukvaran finns ett antal grundläggande funktioner som används för att beräkna banor samt utföra enkla rörelser. Förutom funktionen $steer$ är funktionerna programmerade av projektgruppen.

4.5.1 Steer

Inkluderat i den medföljande Balanduinokoden med är funktionen $steer$ för att de grundläggande rörelserna. Funktionen tar emot en riktning och ett värde som bestämmer den förskjutna vinkeln $offset$ samt $turning$ som beskrevs i kapitel 4.2.

4.5.2 MoveInstruction

Hjälpmetod som ser till att roboten färdas framåt en given sträcka enligt ekvation 2. $MoveInstruction$ nyttjas av alla andra metoder och använder sig i sin tur utav den inbyggda funktion $steer$ för att förflytta sig. En variant av metoden har skapats som når en högre hastighet, den används när roboten skall skjuta bollen mot mål för att simulera ett skott.

4.5.3 TurnInstruction

Hjälpmetod som ser till att roboten roterar ett givet antal grader. Använder Balanduinons rotationskodare för att beräkna hur långt den har roterat enligt ekvation 3. Beräknar åt vilken riktning som minst rotation krävs för önskad vinkel. Likt $MoveInstruction$ används denna metod av alla andra metoder.

4.5.4 CalculateTurn

Hjälpmetod som givet data från kameran beräknar den vinkel roboten skall rotera för att ställa sig rakt mot ett objekt. Kompenserar även för vilken kvadrant det givna objektet befinner sig i relativt roboten. Placeringen av objektet relativt roboten påverkar åt vilket håll roboten skall rotera samt hur många extra grader den skall rotera enligt avsnitt 3.3.

4.5.5 CalculateScore

En metod som gör alla beräkningar enligt avsnitt 3.4 när beslut tagits om att roboten skall skjuta bollen mot mål. Använder sedan TurnInstruction och MoveInstruction två gånger var för att först ta roboten till en bra skottposition, se figur 16, och sedan skjuta bollen mot mål.

4.5.6 AvoidObject

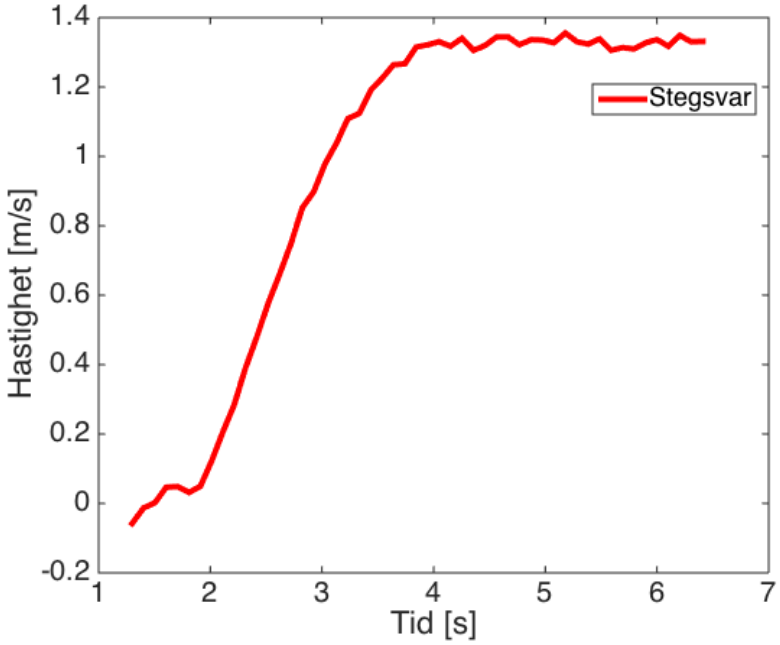
När roboten håller på att utföra en rörelse kontrollerar roboten en gång per loop om den håller på att krocka med sargen eller en annan robot. Om roboten kommer för nära sargen eller en annan robot så avbryts rörelsen.

5 Tester och resultat

I detta kapitel beskrivs tester som genomförts samt deras resultat. Resultaten från testerna har senare använts för att föra projektet framåt.

5.1 Stegsvar på hastigheten

För att ta reda på hur snabbt Balanduino kommer upp i önskad hastighet har Balanduinos stegsvar studerats. Stegsvaret erhölls genom att kontinuerligt mäta Balanduinons hastighet medan den accelererar upp till en önskad hastighet. Resultatet visas i figur 22.



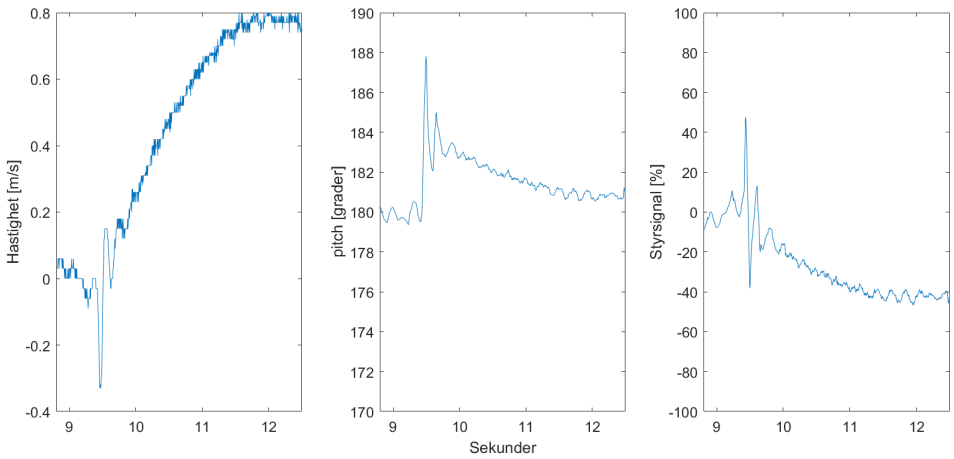
Figur 22: Stegsvar för 25°vinkelförskjutning. Författarnas egna bild.

Ur grafen beräknades accelerationen till 0,75 m/s^2, och stigtiden 1,32s. Båda är av intresse när beslut skall tas kring huruvida roboten exempelvis hinner bryta en bollbana. Testet visar även att systemet stabiliseras kring en konstant hastighet på 1,3 m/s vilket visar att systemet är stabilt. Då rummet är litet kommer endast konstant hastighet erhållas under korta perioder.

5.2 Jämförelse mellan regulatordesigner

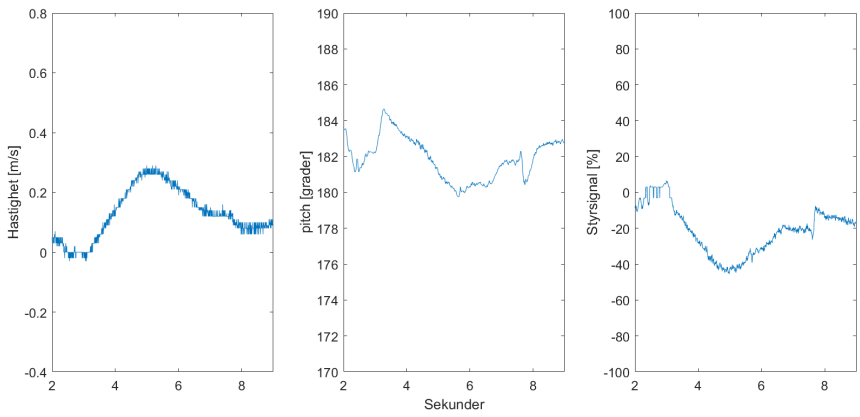
Som nämns tidigare i rapporten har TKJ Electronics en öppen källkod för balanseringen som modifieras och används i projektet. Systemet anses fungera väl med god balansering och är även förhållandevis snabbt. Balanseringskoden består av en PID-regulator, där regulatorkonstanterna är framtagna genom iterativ testning[16]. Med anledning av det ovetenskapliga tillvägagångssättet för att få fram dessa regulatorkonstanter gjordes ett jämförande test för att mäta prestanda. Testet jämförde TKJs implementation, figur 23, och en kaskadkopplad reglerdesign, figur 24, från en kurs i reglerteknik vilken istället regleras kring en referenshastighet. Beräkningarna av regulatorkonstanterna för den senare går att finna i appendix. Om utfallet av testet skulle visa en markant förbättring skulle tiden som krävs för att implementera en ny design vara motiverad.

Testet jämför stegsvar för hastigheterna, uppskattad vinkel samt beräknad styrsignal mellan de två lösningarna. De viktigaste faktorerna är acceleration, stabilitet samt att systemet reagerar konsekvent.



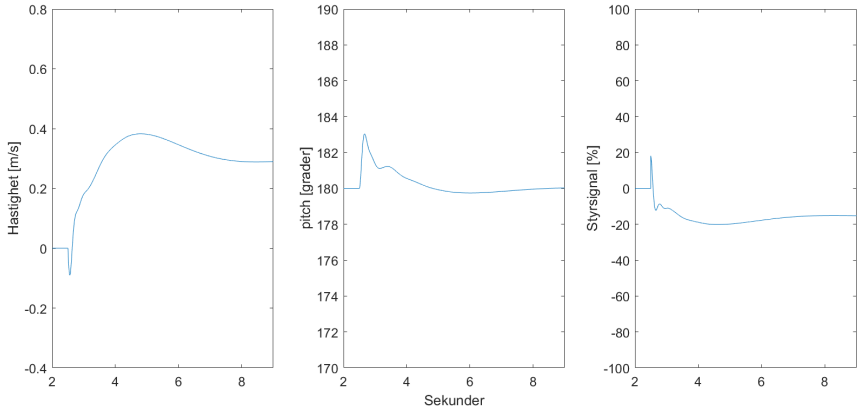
Figur 23: Stegsvar av hastighet, uppskattad vinkel samt styrsignal för TKJs regulator. Författarnas egna bild.

Som synes i figur 23 resulterar indata på $25°$ vinkelförskjutning i en hastighet beräknad till ungefär 0,8 m/s, stigtiden beräknades till 1,75 sekunder och vikelförskjutningen hamnade mellan $2°$-$8°$. På grund utav de itererande begränsningarna samt skalning med hastighet, som beskrivs i avsnitt 4.2, syns här tydligt att den önskade förskjutningen ej nås. Systemet håller sig stabilt med en jämn accelerationskurva upp till maxhastigheten där styrsignalen är väl inom ramen för vad motorerna kan producera.



Figur 24: Stegsvar av hastighet, uppskattad vinkel samt styrsignal för designad regulator. Författarnas egna bild.

Figur 24 visar grafer från den kaskadkopplade regulatorn. Denna implementation regleras kring en referenshastighet. Högsta referenshastighet som fungerade i praktiken var 0,3 m/s, vid högre hastigheter än så välte roboten omgående. Stigtiden från pålagt steg till maxhastighet beräknades till 2,03 sekunder vilket är en snarlik snabbhet på de båda systemen. Här syns tydlig problematik i implementationen då slutvärdet istället hamnade runt 0,13 m/s vilket inte alls följer stegsvaret i den simulerade modellen som visas i figur 25. Systemet ansågs vara för oberäkneligt för att kunna användas.



Figur 25: Stegsvar av hastighet, uppskattad vinkel samt styrsignal för simulerad beräknad regulator. Författarnas egna bild.

Simuleringen visar ett betydligt långsammare, men stabilare system än det i figur 24. Testerna blev ej helt jämförbara då den kaskadkopplade designen inte kom upp i samma hastighet. Efter att testerna har jämförts valdes TKJ Electronics implementation då den var stabilare och ingen tid behövdes lägga på att implementera den designen.

5.3 Pixlar till centimeter

Pixy-kameran identifierar punkter på spelplanen. Dessa kan sedan användas för att beräkna avståndet mellan två punkter eller två objekt. Detta avstånd kommer vara i enheten pixel medan Balanduino använder enheten centimeter för avstånd. Detta gör att ett förhållande mellan pixel och centimeter behöver tas fram för att kommunikationen från Pixy till robot skall vara meningsfull.

För att ta fram ett sådant mått utfördes följande test. Två objekt placerades längs samma uppmätta linje på spelplanen och med Pixy-kameran uppmättes avståndet i pixlar. Testet upprepades för flera olika platser på planen för att kontrollera om kameralinsen har konvexa attribut som påverkar beräkningarna.

Test	Pos obj2	Pos obj1	Avstånd[pixlar]	Avstånd[cm]	Ratio [cm/pixel]
1	224	53	171	227,6	1,33
2	190	7	183	254,1	1,38
3	190	2	188	254,1	1,35
4	225	135	90	120,0	1,33

Tabell 1: Mätvärden från testet

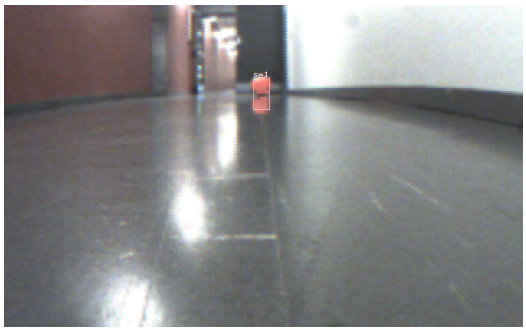
Testerna visar att förhållandet mellan centimeter och pixel i medelvärde är 1,35 cm/pixel. Någon större skillnad beroende på vart vid spelplanen testet utfördes märktes inte. Förhållandet kommer användas i metoder och ekvationer där avstånd mellan objekt kommer användas av Balanduino. Om Pixy-kameran anser att mitten av ett objekt egentligen inte är mitten på objektet, uppstår ett mätfel. Detta då det uppmätta avståndet i centimeter mellan punkterna blir felaktigt och därmed blir tabellvärdenas förhållande felaktig.

5.4 Pixy prestandatest

Pixy-kameran sitter monterad på 3 meters höjd ovanför mittpunkten av spelplanen. Avståndet ut till hörnen från mittpunkten är 1,6 meter vilket innebär att det längsta avståndet kameran behöver se är 3,4 m.
För att testa Pixy-kamerans prestanda mättes en rak linje upp i en korridor, se figur 26, där varje halvmeter upp till 4 meter markerades. Pixy-kameran placerades i ena änden av linjen och den boll som används i projektet flyttades successivt längs den uppmätta linjen. Efter att bollen placerats och man genom PixyMon bekräftat att Pixy sett objektet flyttades det ytterligare en halvmeter bort från Pixy kameran. Detta upprepades tills det var 4 meter mellan Pixy och objektet, då man nu var garanterad att Pixy skulle klara av att identifiera objekt på 3,4 meter. Resultat av testet syns i tabell 2.

Avstånd	Identifierad av Pixy
100cm	Ja
150cm	Ja
200cm	Ja
250cm	Ja
300cm	Ja
350cm	Ja
400cm	Ja

Tabell 2: Mätdata



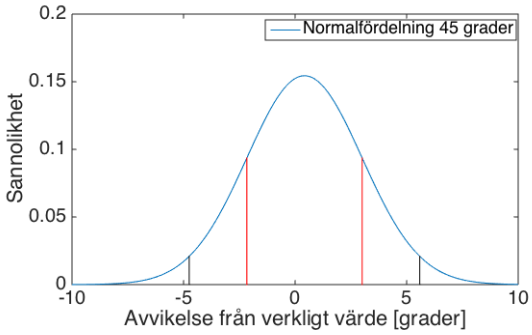
Figur 26: Bild av testet 300 cm avstånd. Författarnas egna bild.

Resultatet visar att kameran klarar av prestandakravet på att identifiera objekt på 3,4 meter. Eventuella mätfel på den uppmätta sträckan bidrar inte till någon osäkerhet i hurvida kameran klarar att identifiera objekt på 3,4 meters håll då man med säkerhet vet att kameran klarar att identifiera objekt upp till 4 meter.

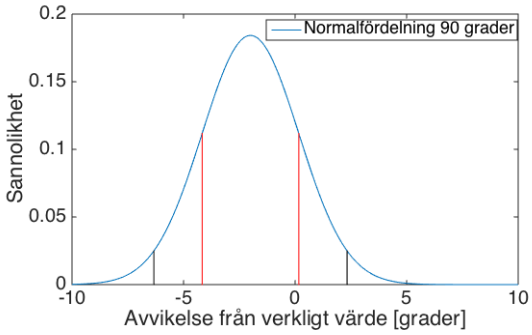
5.5 Vinkelprecision Pixy

Robotens vinkel identifierar Pixy med hjälp av sitt färgkodningsläge. För att testa hur precist denna vinkel identifieras av Pixy-kameran utfördes ett prestandatest. Vinklarna $45°$ och $90°$, har mätts upp manuellt varpå Balanduinon placerats i dessa vinklar. Att två olika vinklar valdes var för att undersöka om det fanns en försämrad precision kring vinklar skilda från $0°$, $90°$, $180°$ och $270°$. Pixy-kameran har sedan mätt vinkeln 150 gånger vardera i de båda fallen.

Av mätvärdena har standardavvikelsen för de båda fallen kunnat räknats ut till $\sigma_{45} = 2,58$ och $\sigma_{90} = 2,16$. Medelvärdet för felet blev $\mu_{45} = 0,42$ och $\mu_{90} = -2,00$. Med antagandet att felet är normalfördelat, har också två normalfördelningskurvor tagits fram, se figur 27 och 28. De röda linjerna indikerar avståndet $\pm\sigma$ från medelvärdet μ, och de svarta linjerna avståndet $\pm 2\sigma$ från medelvärdet. Området som innesluts av de röda linjerna omfattar $68,3\%$ av fallen, medan området som innesluts av de svarta linjerna omfattar $95,5\%$ av fallen.



Figur 27: Normalfördelning över mätfel för 45°, Författarnas egna bild.



Figur 28: Normalfördelning över mätfel för 90°, Författarnas egna bild.

Att medelvärdet inte blivit exakt noll beror troligtvis på osäkerheten i den manuella mätningen utav vinkeln. Bortsett från detta, fås en god uppfattning av sannolikheten för felaktiga avläsningar. Ur normalfördelningskurvorna går det också att se att det är sämre vinkelavläsning vid icke räta vinklar. Eftersom roboten sällan kommer befinna sig i räta vinklar blir därför normalfördelningen för $45°$ av större intresse. Baserat på testet så kommer alltså mätvärdena att variera med $\pm 2\sigma_{45}$ med en sannolikhet på $95,5\%$. Ett extremfall med en värsta tänkbara avläsning på $2\sigma_{45}$ och en färdsträcka på 2 m, vilket i princip är hela spelplanen, skulle innebära att roboten skulle åka till en punkt på $2\sin(2\sigma_{45}) = 0,09$ m vid sidan av bollens mittpunkt. Eftersom det är 10cm från robotens mittpunkt till ett hjul, så innebär det att roboten även i ett sådant scenario skulle träffa bollen.

5.6 Precisionstest Balanduino

När Balanduino skall åka från punkt A till punkt B är det viktigt att den hamnar just på punkt B. För att testa längdprecisionen hos Balanduinon mättes olika längder upp. Längden roboten skall åka har programmerats in med ekvation 2. En fast hastighet har använts då den kommer vara grund för alla rörelser under matchspel.

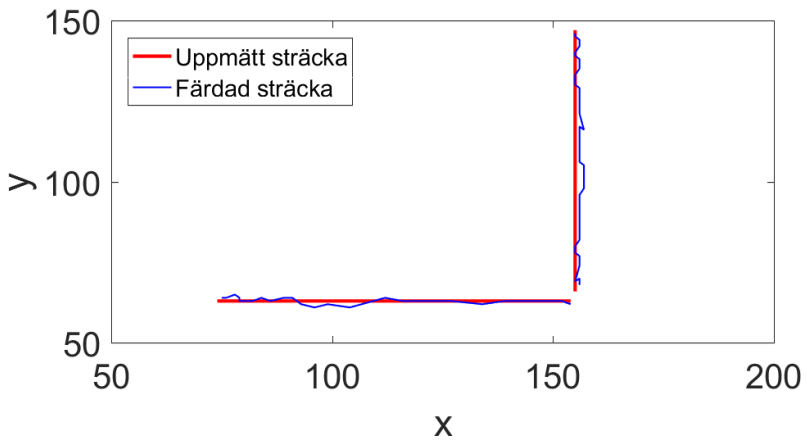
Önskad sträcka	**Färdad sträcka**	**Fel**
50 cm	71 cm	21 cm
70 cm	90 cm	20 cm
80 cm	100 cm	20 cm
100 cm	118 cm	18 cm
130 cm	150 cm	20 cm

Tabell 3: Tabell över korrekt sträcka och åkt sträcka.

I testerna syntes det tydligt att roboten börjar bromsa vid det korrekta avståndet och att det sedan följer en bromssträcka. Testet visar att roboten har ett linjärt beteende med en konstant stoppsträcka som i genomsnitt är 20 cm. I programmeringen tas hänsyn till denna stoppsträckan för att få roboten att stanna vid önskad position i de fall där roboten skall färdas längre sträckor. Skall roboten färdas sträckor som är kortare än själva stoppsträckan behövs det ej ta hänsyn till någon stoppsträcka då hastigheten är så pass låg.

5.7 Dynamiskt rörelsetest

En linje som är rak i x-led och en annan linje som är rak i y-led mättes upp med hjälp av Pixy-kameran. Längs linjerna drogs en boll samtidigt som bollens mittposition mättes. Genom att jämföra den uppmätta linjen och bollens bana kan man få reda på hur stor positionsavvikelse som Pixy-kameran skapar. I figur 29 nedan kan de uppmätta linjerna och bollens bana ses.

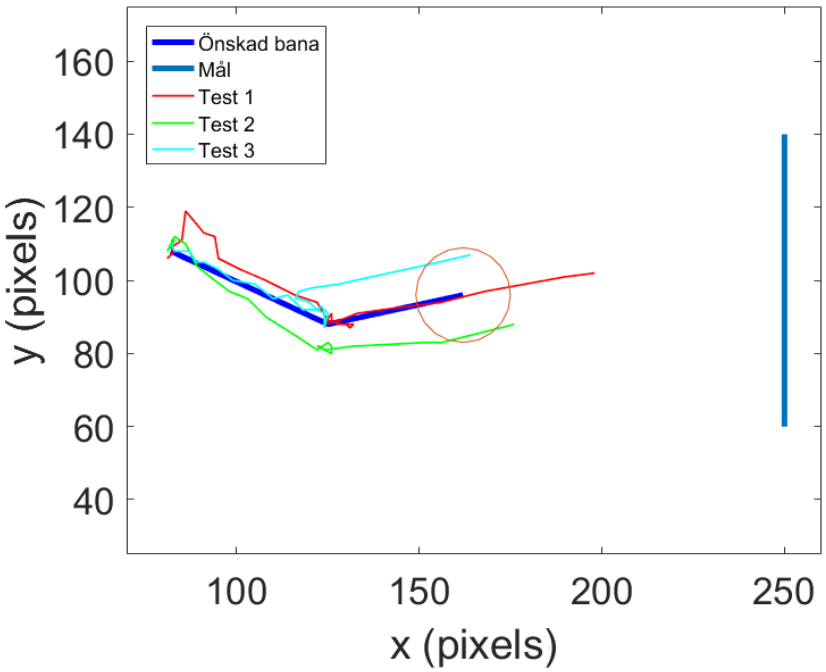


Figur 29: Bilden visar en av Pixy uppmätt linje, röd, samt en bollens bana, blå. Författarnas egna bild.

Att döma av denna grafen är Pixy-kameran bra på att avgöra vid vilken pixel ett objekt befinner sig på. Avvikelsen är några pixlar medan objektet som användas vid testet hade en storlek på 10x10 pixlar, vilket gör att mätfelet inte är då stort i förhållande till objektets storlek. Viss avvikelse kan bero på att det var svårt att röra bollen exakt längs den uppmätta linjen.

5.8 Test av skottbana

Testet gick ut på att undersöka hur väl roboten klarade att följa en förutbestämd bana för att ta sig till bollen och skjuta mot mål. Roboten började i en specifik startposition varpå den roterade, körde fram och roterade igen tills den stod i en bra skottposition med bollen mellan sig själv och målet. Sedan accelererade roboten mot bollen och sköt den mot mål. Testet upprepades tre gånger och robotens körda banor samt den förutbestämda banan går att se i figur 30 nedan.



Figur 30: Bilden visar robotens rörelse i de tre testen samt den förutbestämda banan. Författarnas egna bild.

Programmeringen för att få roboten att röra sig i denna förutbestämda bana bygger på de avlästa värdena från rotationskodarna. Dessa värden ger information om hur långt roboten förflyttats och hur mycket den har roterat.

Om roboten inte lyckades balanserade stabilt i startpositionen så ledde det till problem då roboten kunde avvika från sin förutbestämda startvinkel. Det var svårt att få roboten att stå still och det blir därför en felkälla till varför roboten inte följer den förbestämda linjen bättre. Ytterligare en felkälla är den marginella skillnaden i vinkel som roboten placerades ut på startpositionen med.

5.9 Test av överföringstid för kommunikationen

För att testa överföringshastigheten på Bluetooth-modulerna används en räknare i koden som beräknar den tid som gått sen programmet startade. I testet skrevs två tider ut, tidpunkten då hämtning av data via Bluetooth började och tidpunkten när hämtningen var färdig. Skillnaden mellan de två tidpunkterna blev överföringhastighet som systemet har. Nedan i tabell 4 kan överföringstiderna från testet observeras.

Test nr	**Överföringstid**
1	87 ms
2	86 ms
3	85 ms
4	89 ms
5	84 ms
6	87 ms

Tabell 4: Tabell över överföringshastigheter

Resultatet av testet visar att det i snitt tar 86 ms att hämta information om ett objekt via Bluetooth. Detta innebär att man kan uppdatera ett objekts position ungefär 6 gånger per sekund vilket är tillräckligt bra för att en fotbollsmatch skall kunna spelas flytande.

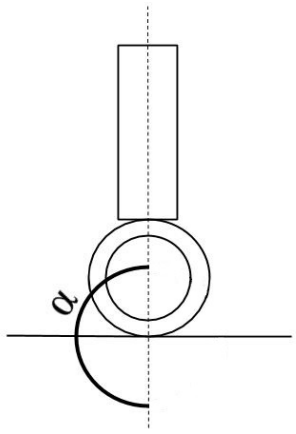
5.10 Test av looptid

Den kod som används av robotarna körs konstant i en loop. I loopen uppdateras bland annat variabler som används för att balansera roboten. För att ta reda på om variablerna uppdateras tillräckligt frekvent för att roboten skall kunna balansera stabilt, behöver ett test av looptiden göras.

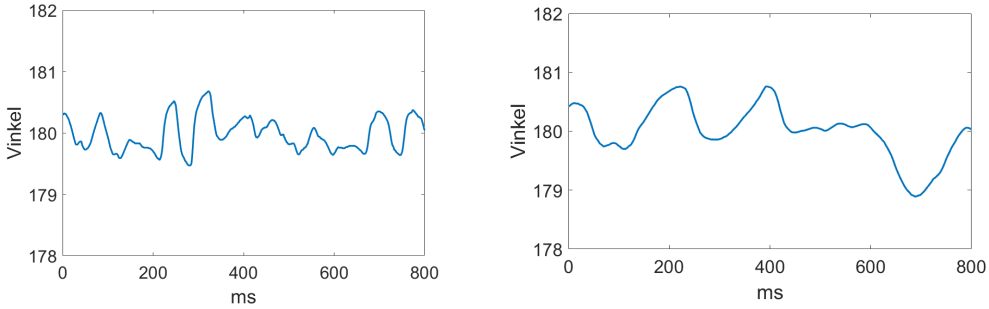
I koden läggs en räknare till som beräknar den tid som passerat sedan programmet startades. Koden skriver ut den tiden som passerat sedan den senast loopade. Testet visar att loopen tar 3-4 ms, vilket är tillräckligt för att roboten skall kunna balansera bra.

5.11 Test av Bluetoothpåverkan

Att utföra datahämtningen via Bluetooth gör att balanseringen inte uppdateras under en kort tid, se avsnitt 4.4.2. Detta testet gjordes för att ta reda på om denna paus i uppdateringen påverkar balanseringen negativt och i så fall i vilken utsträckning det påverkar balanseringen. Testet gick ut på att mäta robotens vinkel α, se figur 31, i två fall. I det första fallet skulle roboten enbart balansera och i det andra fallet skulle roboten hämta data via Bluetooth samtidigt som den balanserade. Information om robotens vinkel α ficks genom att koppla robotens Arduino till en dator. Vinkeln vid olika tidpunkter för de båda testen kan ses i figuren nedan.



Figur 31: Vinkeln α är vinkeln som mätts i detta test. Författarnas egna bild.

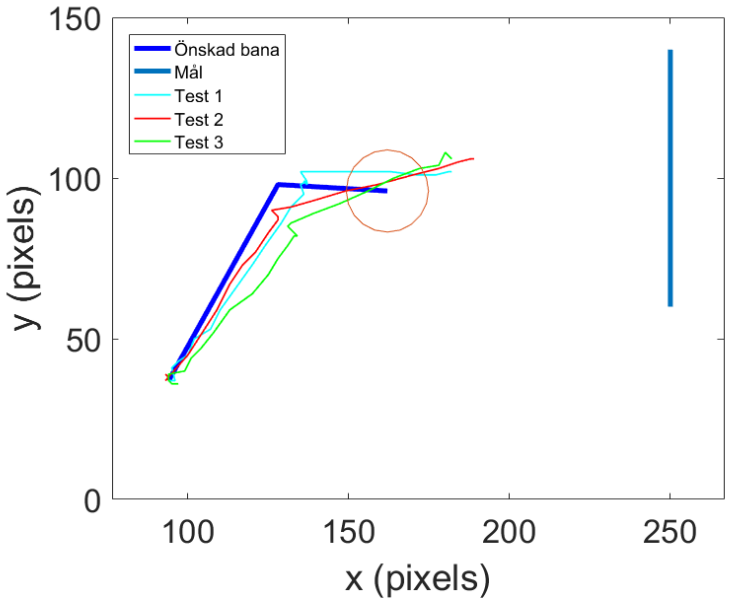


Figur 32: Till vänster: robotens vinkel utan påverkan av Bluetooth. Till höger: robotens vinkel med påverkan av Bluetooth.

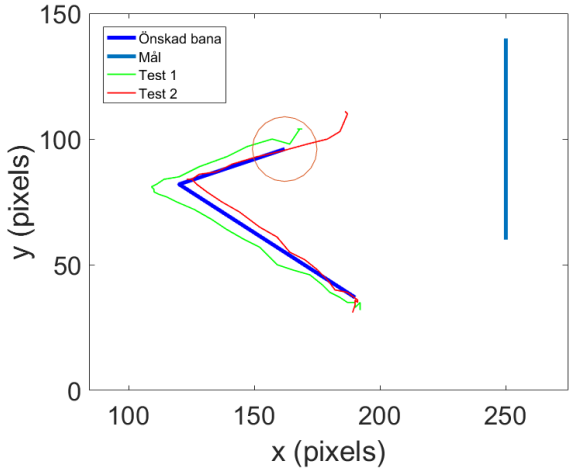
Resultatet visar att det inte är någon väsentlig skillnad på hur väl roboten balanserar när den hämtar data via Bluetooth och när den inte gör det. I båda fallen varierar vinkeln med ungefär $\pm 1^\circ$ från upprätt läge. Dock har det observerats vid plötsliga störningar att stabiliteten varit betydligt sämre då överföring av data har skett.

5.12 Skottest med hela systemet

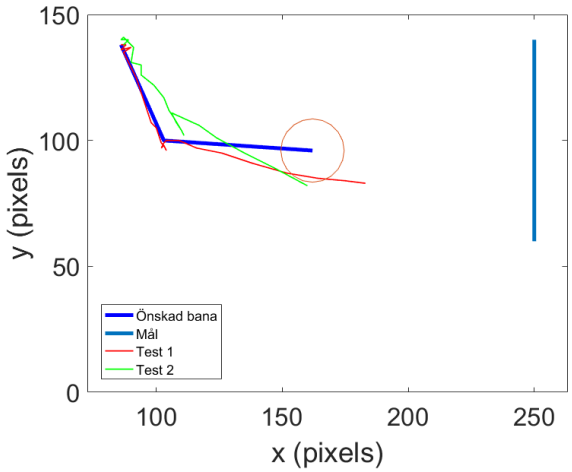
För att testa hela systemet, det vill säga kompatibiliteten mellan Balanduino, Pixy, mjukvara samt kommunikation, testades skottfunktionen. Roboten skall då via Bluetooth få information om bollens position, sin egen position och sin vinkel. Med den datan skall roboten sedan beräkna en bana för att ta sig till en bra skottposition och sedan skjuta bollen i mål. Testet har upprepats från tre olika startpositioner där roboten placerats. Bollen placeras på en fast position. Testet har återupprepas flera gånger på varje position. I graferna nedan visar hur roboten har åkt och uppmätta banan.



Figur 33: Figuren visar 3 olika tester där roboten skall ställa sig i en skottposition och sedan skjuta bollen i mål. Testerna representerar robotens färdade sträcka och cirkeln representerar området där bollen träffas. I samtliga test gjorde roboten mål.



Figur 34: Figuren visar 2 olika tester där roboten skall ställa sig i en skottposition och sedan skjuta bollen i mål. Testerna representerar robotens färdade sträcka och cirkeln representerar området där bollen träffas. I samtliga test gjorde roboten mål.



Figur 35: Figuren visar 2 olika tester där roboten skall ställa sig i en skottposition och sedan skjuta bollen i mål. Testerna representerar robotens färdade sträcka och cirkeln representerar området där bollen träffas. I samtliga test gjorde roboten mål.

Testet visar att systemet funkar bra och roboten träffade bollen varje gång. Dessutom lyckades roboten göra mål på alla dessa tester. Figurerna 33-35 visar dock att den ej följer banan perfekt utan roterar felaktigt ibland och åker därför lite ur kurs. Detta fel är även större än testet i kapitel 5.8 då det används förprogrammerade vinklar och positioner på roboten. Det extra felet kommer främst från felmarginalen i Pixy-kamerans vinkelavläsning.

6 Diskussion

Kapitlet ämnar att beskriva varför olika beslut tagits under projektet samt hur de har påverkat projektet. I avsnittet diskuteras också hur välfungerande slutprodukten blev, vad som skulle kunna ha gjorts annorlunda samt hur projektet kan utvecklas i framtida arbeten.

6.1 Pixy

Att använda Pixy-kameran har löst de grundläggande problemen med objektidentifiering. Mjukvaran PixyMon har gjort att kameran har varit enkel att arbeta med. Pixy-kamerans problem från föregående arbete var kända för gruppen och förhoppningen var att bättre ljusförhållande samt att hålla sig till en fast spelplan skulle lösa identifieringsproblemen. Det har fungerat bra med ljuset och den fasta spelplanen. Pixy-kameran har oftast lyckats hitta objekten efter att inställningarna och förutsättningarna förbättras.

Däremot har den vissa brister i prestandan. Pixy-kamerans vinkelmätning är instabil och fluktuerande som testet i avsnitt 5.5 visar. Detta har medfört att det blivit problematiskt att beräkna robotens riktning vilket resulterat i att roboten ibland roterat fel antal grader.

Ett annat problem med Pixy-kameran var att den ibland inte sparade de inställningar man gjort, vilket orsakade onödigt extra arbete. Varför den inte sparar inställningarna är fortfarande oklart. Eftersom Pixy löser de största problemen lades ingen avsevärd tid på att undersöka andra alternativ, i framtiden hade det antagligen gjorts för att förbättra prestandan.

6.2 Kommunikation

Under projektet har mycket arbete lagts på att överföra information trådlöst från Pixy-kameran till robotarna. Först beslutades det om att använda WiFi. En av de största fördelarna med WiFi är att en modul kan agera server och skicka data till flera klienter samtidigt till skillnad från Bluetooth som bara kan skicka till en klient i taget.

Det tog väldigt lång tid att få fram ett fungerande WiFi-system då ingen i gruppen hade någon kunskap av trådlös kommunikation innan projektet. När det till slut fungerade tog det alldeles för lång tid att skicka data via WiFi för att en fotbollsmatch skulle kunna spelas. Protokollet HTTP testades men var för långsamt då det totalt tog processen cirka 4 sekunder. Det fanns idéer om att använda andra protokoll för att överföra data men det hade troligen tagit för lång tid att implementera.

När WiFi ej fungerade för projektets syfte testades istället Bluetooth. Anledningen till att Bluetooth ej användes från början var att en Bluetooth-modul endast kan parkopplas med en annan. Framtagningen av en lösning med Bluetooth gick mycket snabbare. De fanns kod som kunde återanvändas från WiFi-lösningen och bra guider online.

Med Bluetooth fick vi ner överföringstiden till 86 millisekunder per objekt. Den överföringshastigheten innebär att roboten kan få information om ett objekts position upp till 6 gånger per sekund vilket var tillräckligt för att spela fotboll. Om man hade spelat med 4 robotar samtidigt hade uppdateringsfrekvensen varit ca 1 Hz. Det innebär att det hade varit svårt att få robotarna att spela fotboll då de troligtvist hade åkt till fel positioner och krockat med varandra. För ett större system hade det varit bättre att undersöka ett annat kommunikationssytem som är snabbare och kan skicka till flera enheter samtidigt.

Det har varit problem med att Bluetooth-modulerna som suttit på robotarna har gått sönder under projektet. Möjligen beror det på skador när roboten krockar med något föremål. En annan potentiell anledning kan vara att komponenterna hanterats ofta, där elektrostatiska urladdningar eller vanligt handhavande kan ha skadat modulerna. Dessa orsaker misstänks dels då Bluetooth-modulen som varit placerad i taket aldrig behövts bytas ut. Varje gång en komponent gått sönder har den varit väldigt varm, vilket tyder på någon form av kortslutning.

Att få den trådlösa kommunikationen att fungera tog väldigt lång tid att utveckla. Det gjorde att hela systemet var klart sent i projektet. Detta ledde till att programmeringen utav mjukvaran drog ut på tiden då vitala delar inte kunde testas tidigare. Hade Bluetooth valts som lösning från första början hade antagligen mjukvaran varit mer utvecklad och robotarna hade möjligtvis spelat fotboll bättre.

6.3 Balanduino

Att utgå ifrån Balanduino-satsen istället för att bygga en egen produkt har öppnat upp för möjligheterna att utveckla andra områden så som kommunikation, taktik med mera. Till balanduino finns som tidigare nämnt öppen källkod att tillgå där balansering och viss styrning fungerar vilket underlättat mycket vid implementeringen utav våra funktioner. Öppen källkod som sådan är uppskattat då det ger möjlighet för olika människor med olika kunskaper att förbättra eller förändra koden efter sina egna önskemål. Att Balanduino är baserat på Arduino är positivt på grund av det enkla gränssnittet samt att de använder programmeringsspråket C/C++ då finns mycket information om detta.

Svårigheterna med Balanduino-satsen var att viss del av koden var svårförstådd med vaga beskrivningar. Detta har lett till att det tagit lång tid för att skapa sig tillräcklig förståelse för att kunna skriva samt göra förbättringar av funktioner.

6.3.1 Modifikationer utav reglering

Att få ett instabilt system stabilt visade sig svårt då modellen för processen blir väldigt komplicerad. I roboten används manuellt testade värden vilket resulterat i god balansering med små variationer kring referensvinkeln. Dock är systemet aningen långsamt vid start samt stopp. Det har simulerats och undersökts huruvida olika implementationer utav den inre regulatorn ger bättre prestanda i form utav acceleration och stabilitet. TKJs lösning fungerar dock bra och robotarna kan spela fotboll medan de balanserade med denna regulatorkonfiguration.

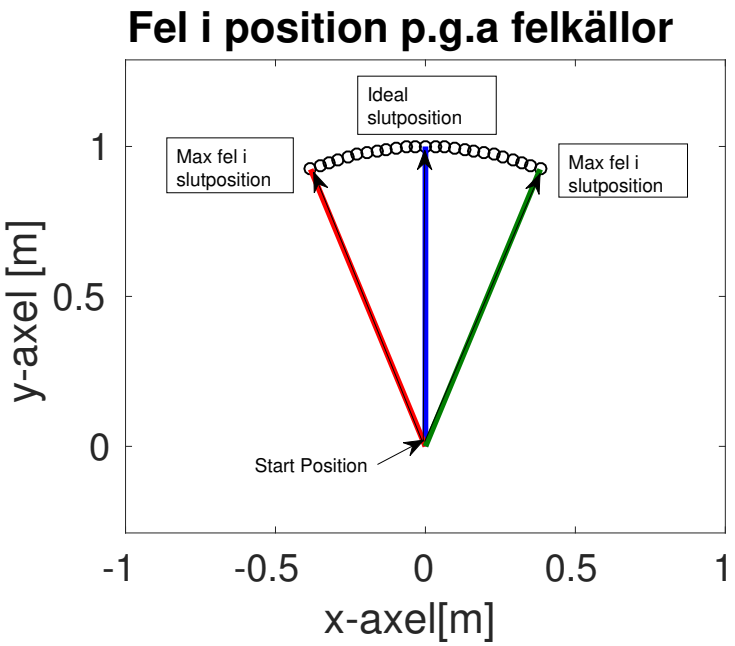
6.3.2 Balanserade Robot

Att använda en balanserande robot på två hjul är ur ett reglertekniskt perspektiv en intressant uppgift. För ett snabbt och precist fotbollsspel är det däremot inte optimalt. En stabil robot på fyra hjul är en enklare konstruktion då roboten slipper balansera. Man skulle då undvika att robotar faller samt att de skulle klara att accelerera och bromsa snabbare. En sådan konstruktion bör bidra till ett snabbare och mjukare fotbollsspel. Detta under förutsättning att dessa robotar kan rotera lika bra som Balanduino, vilket är en av Balanduinos fördelar.

6.4 Exekveringstider

Testet i avsnitt 5.10 visar att de tar 3-4 ms att loopa genom systemet. Det innebär att värdet från rotationskodarna uppdateras med samma intervall. Värdet på rotationskodarna ökar med 44 pulser mellan varje avläsning då roboten färdas i den hastighet som används i projektet. Detta kan resultera i att det beräknade avståndet kan överstigas med maximalt 43 pulser. Då 1 cm motsvarar 62 pulser är denna osäkerhet väldigt liten i sammanhanget. Däremot adderas värdena från båda avkodarna vid rotation i ekvation 3. Där blir det maximala felet 88 pulser som motsvarar 3,2°. Det maximala felet kan bidra till problem när roboten roterar. Felet bidrar till att roboten kan missa sin önskade position med maximalt 2 sin(3)=0,1m om roboten färdas 2 meter vilket är ungefär den längsta sträcka på spelplanen.

6.5 Felkällor i positionsprecision



Figur 36: Figuren visar hur mycket fel i position roboten kan hamna på grund av felkällor. Cirklarna representerar positioner där roboten kan hamna. Roboten kan hamn mellan 0-24 cm fel från den ideala slutpositonen. Men sannolikheten för att detta skall inträffa är mycket liten, som test 5.5 visar på. Författarnas egna bild.

Det finns flera felkällor i systemet då roboten skall åka till en position. Pixy-kameran kan mäta ut robotens vinkelfel som testet i avsnitt 5.5 visar. Den kan även identifiera robotens mittpunkt med några pixlar fel, se test 5.7. Detta skulle också påverka den beräknade vinkeln mellan roboten och andra positioner. Det färdade avståndet kan dessutom variera på grund av intervallet i pulsavläsningen som beskrivs i diskusionsavsnitt 6.4 ovan. Till sist har det noterats att roboten kan driva ur kurs när den skall åka rakt fram längre sträckor, bland annat på grund av olikheter i motorerna. Bromssträckan för Balanduino har visat sig vara relativt lik på olika avstånd och det har tagits hänsyn till den i programmeringen men det är inte helt säkert att den alltid är lika lång.

De största felkällorna är vinkelfelet från Pixy samt exekveringstiden. Om man summerar de värsta fallen av de felkällorna kan roboten hamna 24cm fel bredvid dess önskade position enligt figur 36. Felkällorna har bidragit till att roboten ibland kommit så mycket fel att den ibland misslyckats att skjuta bollen mot mål. Under testningen av systemet har det dock visat sig att roboten i de flesta fall klarat av sin uppgift trots alla felkällor. Storleken på objekten tillsammans med den lilla spelplanen bidrar till att felkällorna ej haft större påverkan. Hade spelplanen varit större hade bättre precision kunnat bli nödvändigt.

6.6 Förmåga att spela fotboll

En robot kan i dagsläget förflytta sig från en position till en annan med viss felmarginal. Den kan söka upp en boll, placera sig i skjutposition och sedan skjuta bollen i mål. Den kan inta en defensiv position och de kan undvika spelplanens kanter och andra robotar i viss mån. Man kan säga att den sammanfattningsvis klarar av de grundläggande funktioner som krävs för att spela fotboll. För att en robot skulle kunna spela en match skulle mjukvaran behöva vidareutvecklas så att ovan nämnda funktioner sammanfogas till ett helt spel. För att en match med två spelare skulle kunna spelas behöver, utöver det som nämndes ovan, koden ställas om så att robotarna inte agerar som att de är samma robot. Detta vore en högst rimlig uppgift om det fanns mer tid.

6.7 Vidareutveckling

I detta stycke presenteras förslag och tankar på hur projektet kan utvecklas för att förbättra fotbollsspelandet.

6.7.1 Utökning av spelplanen

Rummet som används som spelplan är inte stort nog för att spela en match med fler än två robotar. För att göra spelet mer avancerat med matcher mellan flera robotar skulle en större spelplan behövas. Med fler robotar kan ett spel med mera komplexa funktioner tillämpas så som passningar och positionsoptimering i grupp. Robotarna kan också ha bestämda roller som till exempel anfallare, back eller målvakt.

Ett problem som uppstår om man förstorar spelplanen är att den befintliga kameran ej kommer täcka hela spelplanen med tillräckligt hög upplösning. Integration av flera kameror som tillsammans skapar en bild över hela spelplanen alternativt en annan kamera som klarar av att se hela spelplanen skulle kunna lösa det problemet.

6.7.2 Startfunktion

I dagsläget behöver varje robot startas manuellt och när den startat börjar den direkt att spela fotboll. För att roboten inte skall börja spela fotboll förrän matchen startar kan man implementera en startfunktion. Roboten skulle då stå stilla och balansera medan den väntar på en startsignal som startar själva matchen. Exempel på vad för typ av signal som kan starta matchen är via Bluetooth-uppkopplingen, en mikrofon alternativt en IR-sensor med en fjärrkontroll.

6.7.3 Kommunikation mellan robotar

Det hade varit givande för spelet om robotarna kunde kommunicera med varandra under matchen. Kommunikationen hade kunnat ske via en centraldator eller genom att robotarna kan kommunicera direkt med varandra. Huvudsyftet med att införa kommunikation mellan robotarna är för att det möjliggjort ett avancerat spel med mer taktik. I systemet som finns nu skulle Arduino-enheten i taket kunna agera centraldator och skicka instruktioner till robotarna.

6.7.4 Skottfunktion

Att genomföra ett skott kräver i dagsläget att roboten kör på bollen med hög hastighet och sätter den i rörelse. Den funktionen fungerar bra men den höga hastigheten innebär också en lång bromssträcka. För att få samma kraft i skottet utan att ha en lång stoppsträcka kan en sparkmekanism konstrueras. Konstruktionen skulle kunna vara hävarmsliknande för att få bollen att lyfta från marken. En annan utveckling av skottfunktionen skulle vara att roboten taktisk väljer vart den skall sikta på mål baserat på målvaktens position. Den nuvarande skottpositionen är endast uttagen så att roboten träffar mål.

6.7.5 Stängd loop

Nuvarande lösningen tar beslut kring vilken position den vill åka till och åker sedan dit oavsett om förutsättningarna ändras eller ej. För att få roboten att oftare hamna på rätt position eller ändra sin rörelse bör man implementera ett system som bygger på en stängd loop. En stängd loop uppdaterar objektens position hela tiden och korrigerar sitt val efter det. En sådan implementering hade lett till ett spel där antalet misstag minskas då robotarna ej kommer färdas till positioner som kräver ytterligare justering. Dessutom skulle bollens bana kunna beräknas för att lägga till taktik för passningar, att ta emot boll och liknande.

7 Slutsats

Projektet har resulterat i ett system med en bra grund för att spela fotboll. Kamerans position och den trådlösa kommunikationen ger mjukvaran goda möjligheter för ett funktionellt fotbollspel. De funktioner som implementerats visar på att systemet är välfungerande som grund för att en mängd olika typer av funktionaliteter skulle kunna läggas till för att få ett fullgott fotbollsspel.

Med mer tid till mjukvaruutveckling finns det stor potential att få fram ett välfungerande spel där robotar kan spela en fotbollsmatch med mycket funktionalitet, utan att behöva vidareutveckla hårdvara och kringmaterial något nämnvärt.

Balanseringen har fungerat bra och varit stabil. De problem som utstått med stabiliteten har kommit som en följd av kompromissen mellan dataöverföring och uppdaterad styrsignal.

Referenser

[1] "Teknikbarometern oktober 2015," HUI Research, 2015. [Online]. Available: http://www.hui.se/BinaryLoader.axd?OwnerID=e01d819f-7730-4baa-9f5e-3948113680f7&OwnerType=0&PropertyName=EmbeddedFile_29bdba7e-0bba-49af-8b7c-59f9a80d209b&FileName=Teknikbarometern_oktober+2015.pdf&Attachment=True

[2] P. Fazli, "Research," 2013. [Online]. Available: http://www.cs.ubc.ca/~mack/Research.htm

[3] P. Fazli, "Robot soccer," 2013. [Online]. Available: http://www.cs.ubc.ca/~mack/RobotSoccer.htm

[4] "A brief history of robocup," Robocup. [Online]. Available: http://www.robocup.org/a_brief_history_of_robocup

[5] "Brief history," FIRA. [Online]. Available: http://www.fira.net/contents/sub01/sub01_3.asp

[6] "Overview," FIRA. [Online]. Available: http://www.fira.net/contents/sub02/sub02_1.asp

[7] "Objective," Robocup. [Online]. Available: http://www.robocup.org/objective

[8] "Robo Soccer," FIRA. [Online]. Available: http://www.fira.net/contents/sub03/sub03_1.asp

[9] "RoboCupSoccer," Robocup. [Online]. Available: http://www.robocup.org/domains/1

[10] L. Jodensvi, V. Johansson, C. Lanfelt, and S. Löfström, "One robot to roll them all," 2015. [Online]. Available: http://publications.lib.chalmers.se/records/fulltext/219277/219277.pdf

[11] S. Boström, E. E. Johansson, F. Kjellberg, R. Larsson, K. Lundgren, and M. Resare, "Balanserande fotbollspelande robotar," 2016. [Online]. Available: http://publications.lib.chalmers.se/records/fulltext/240592/240592.pdf

[12] J. Andersson, J. Alexandersson, J. Kammerland, S. Sundell, and T. Bergström, "Utveckling av autonoma robotar för fotbollsspel," 2016. [Online]. Available: http://publications.lib.chalmers.se/records/fulltext/246634/246634.pdf

[13] Store open electronics. [Online]. Available: store.open-electronics.org/PIXYCAM

[14] "Cmucam: Open source programmable embedded color vision sensors." [Online]. Available: http://www.cmucam.org/

[15] "Pixy (cmucam5): a fast, easy-to-use vision sensor," Kickstarter. [Online]. Available: https://www.kickstarter.com/projects/254449872/pixy-cmucam5-a-fast-easy-to-use-vision-sensor

[16] T. K. Jespersen, "privat kommunikation," 2017-03-29.

[17] T. Electronics. [Online]. Available: https://github.com/TKJElectronics/Balanduino

[18] D. Simon, "Kalman filtering." [Online]. Available: http://academic.csuohio.edu/simond/courses/eec644/kalman.pdf

[19] T. K. Jespersen, “A practical approach to kalman filter and how to implement it.” [Online]. Available: http://blog.tkjelectronics.dk/2012/09/a-practical-approach-to-kalman-filter-and-how-to-implement-it/
[20] TKJ Electronics. [Online]. Available: http://www.balanduino.com/gallery

A Appendix

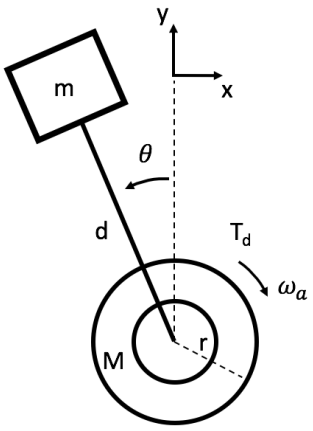
WiFi-lösningen

För att robotarna skulle kunna få information från Pixy kameran hade en WiFi lösning utvecklats. WiFi-modulerna som används heter ESP8266-01 och de är integrerade på ett utvecklingskort med ut- och ingångar samt en micro-USB ingång.

En av WiFi-modulerna var placerad i taket och kopplad till Pixy kameran. Denna agerade central enhet och förmedlade information om positioner och vinklar till alla spelare. På varje robot har en WiFi-modul kopplats till Arduinon för att den på så vis skall kunna ta emot den informationen som Pixy kameran skickat. Modulen i taket agerar som en server där datan från Pixy finns tillgänglig för klienterna.

Reglerberäkningar

Beräkningarna är ifrån maskintekniks tre inlämningsuppgifter i reglerteknik. I figur 37 kan en förenklad modell av Balanduino-roboten ses. Massan är indelad i två delar, en övre massa m och en undre massa M. Avståndet mellan den övre delen och hjulaxeln benämns med d och hjulradien med r. I tabell 5 går värdena på dessa konstanter att utläsas, tillsammans med värdena för de elektriska konstanterna i motorn. T_d är det drivande momentet, vinkelhastigheten ω_a, och θ är robotens lutning från vertikalexaln.



Figur 37: Modell över Balanduino. Författarnas egna bild.

Balanduino-konstanter		
Massa för Balanduinons övre del, batteri och hylla	m	380g
Massa för Balanduinons nedre del, övrig massa	M	970g
Avstånd mellan övre del och hjulaxel	d	180mm
Hjulradie	r	49mm
Strömomvandlingsfaktor	K_m	0,155 Nm/A
Resistans	R_a	2,4Ω

Tabell 5: Tabell över konstanter, förkortning, och värde

För att förenkla uttrycken, och eftersom tröghetsmomentet och rotationsenergin inte kommer ha någon större inverkan på systemet vid låga hastigheter så försummas dessa.
Enligt Newtons andra lag fås :

$$\sum F_{m,y} = m\ddot{y}_m \tag{10}$$

$$\sum F_{m,x} = m\ddot{x}_m \tag{11}$$

Positionen för massan m fås genom:

$$y_m = r + d\cos(\theta) \tag{12}$$

$$x_m = x - d\sin(\theta) \tag{13}$$

Alltså ges andraderivatan enligt följande:

$$\ddot{y}_m = \frac{d^2}{dt^2}(r + d\cos\theta)) = -d\ddot{\theta}\sin(\theta) - d\dot{\theta}^2\cos(\theta) \tag{14}$$

$$\ddot{x}_m = \frac{d^2}{dt^2}(x - d\sin(\theta)) = \ddot{x} - d\ddot{\theta}\cos(\theta) + d\dot{\theta}^2 sin(\theta) \tag{15}$$

Krafterna som verkar på m:

$$\sum F_{y,m} = -mg + mg\cos^2(\theta) \tag{16}$$

$$\sum F_{x,m} = mg\sin(\theta)\cos(\theta) \tag{17}$$

Vid kraftjämvikt kan alltså följande samband göras:

$$-d\ddot{\theta}\sin(\theta) - d\dot{\theta}^2\cos(\theta) = -g + g\cos^2(\theta) \tag{18}$$

$$\ddot{x} - d\ddot{\theta}\cos(\theta) + d\dot{\theta}^2 sin(\theta) = -g\sin(\theta)\cos(\theta) \tag{19}$$

Efter att ha multiplicerat det första jämviktssambandet med $\sin(\theta)$, och det andra jämviktssambandet med $\cos(\theta)$, för att slutligen addera det båda ekvationerna med varandra erhålls differntialekvationen nedan:

$$\ddot{x}\cos(\theta) - d\ddot{\theta} = -g\sin(\theta) \tag{20}$$

För modellens undre massa, M, kan krafterna i x-led beskrivas som följande:

$$\sum F_{M,x} = \frac{T_d}{r} + mg\sin(\theta)\cos(\theta) \tag{21}$$

Där T_d är det totala drivande momentet från DC-motorerna
Genom sambanden nedan

$$m\ddot{x}_m = -mg\sin(\theta)\cos(\theta) \tag{22}$$

$$\ddot{x}_m = \ddot{x} - d\ddot{\theta}\cos(\theta) + d\dot{\theta}^2\sin(\theta) \tag{23}$$

Kan följande rörelsesamband för den undre massan, M skrivas:

$$M\ddot{x} + m\ddot{x} - md\ddot{\theta}\cos(\theta) + md\dot{\theta}^2\sin(\theta) = \frac{T_d}{r} \tag{24}$$

Följande rörelseekvationer för hela systemet ges då av nedan differentialekvationer:

$$(m+M)\dot{v} - md\cos(\theta)\ddot{\theta} + md\sin(\theta)\dot{\theta}^2 = \frac{T_d}{r} \tag{25}$$

$$\cos(\theta)\dot{v} - d\ddot{\theta} + g\sin\theta = 0 \tag{26}$$

Modell för DC-motorn enligt Kirchoffs lag, där L_a är induktansen

$$-u(t) + R_a i_a(t) + L_a \frac{d}{dt} i_a(t) + u_m(t) = 0 \tag{27}$$

Där spänningen u_m är proportionell mot vinkelhastigeheten $\omega_a(t)$ enligt:

$$u_m(t) = 0.3078\omega_a(t) \tag{28}$$

Och det drivande momente T_d är proportionellt mot strömmen i_a enligt:

$$T_d(t) = 0.155 i_a(t) \tag{29}$$

Eftersom det är den långsamma dynamiken som dominerar dynamiken i ett system, vilket i motorfallet är mekaniken. Induktansen L_a kan därför försummas. Då balanseringsjämförelsen kommer utföras stillastående kommer motorns vinkelhastighet ω_a vara liten, och $K_u\omega_a$ sätts därför också till noll. Den förenklade DC-motorsmodellen blir då:

$$-u + R_a i_a = 0 \tag{30}$$

Detta tillsammans med att $T_d = K_m i_a$ resulterar i den förenklada olinjära modellen nedan:
Förenklad olinjär modell

$$(m+M)\dot{v} - md\cos(\theta)\ddot{\theta} + md\sin(\theta)\dot{\theta}^2 = \frac{K_m}{rR_a}u \tag{31}$$

$$\cos(\theta)\dot{v} - d\ddot{\theta} + g\sin(\theta) = 0 \tag{32}$$

Linjärisering kring arbetspunkten $\theta = 0, \dot{\theta} = 0, v = 0$

$$\begin{bmatrix} \Delta\dot{z_1} \\ \Delta\dot{z_2} \\ \Delta\dot{z_3} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ \frac{g}{d}(1+\frac{m}{M}) & 0 & 0 \\ \frac{gm}{M} & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta z_1 \\ \Delta z_2 \\ \Delta z_3 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{K_m}{rR_aMd} \\ \frac{K_m}{rR_aM} \end{bmatrix} \Delta u \tag{33}$$

$$\Delta y = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta z_1 \\ \Delta z_2 \\ \Delta z_3 \end{bmatrix} \tag{34}$$

Vilket ger överföringsekvationen för balansering

$$G_{inre}(s) = C(sI - A)^{-1}B = \frac{\frac{K_m}{rR_aMd}}{s^2 - \frac{g}{d}(1+\frac{m}{M})} \tag{35}$$

Regulatorn för detta väljs till en PID-regulator med filterkonstant T_f

$$F_{inre}(s) = K_p + \frac{K_i}{s} + \frac{sK_d}{1 + sT_f} \tag{36}$$

Beräkning av PID-konstanter i det inre återkopplat systemet. Polerna ges då av rötterna till den karakteristiska ekvationen:

$$1 + F_{inre}(s)G_{inre}(s) = 0 \tag{37}$$

Alltså

$$s^4 + \frac{1}{T_f}s^3 + \frac{7.55(K_d + K_pT_f) - 75.85T_f}{T_f}s^2 + \frac{7.55(K_p + K_iT_f) - 75.85}{Tf}s + \frac{7.55K_i}{T_f} = 0 \tag{38}$$

Identifiering gör att polerna kan räknas ut genom följande:

$$(s + a_1)(s + a_2)(s + a_3)(s + a_4) = 0 \tag{39}$$

$$T_f = \frac{1}{a_1 + a_2 + a_3 + a_4} \tag{40}$$

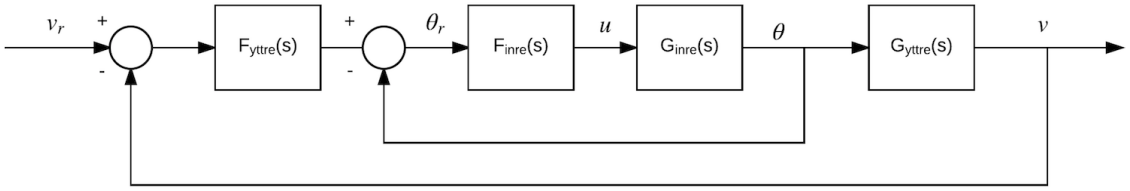
$$K_i = \frac{a_1a_2a_3a_4T_f}{7.55} \tag{41}$$

$$K_p = \frac{(a_4(a_1a_2 + a_3(a_1 + a_2)) + a_1a_2a_3)T_f + 75.85}{7.55} - K_iT_f \tag{42}$$

$$K_d = \frac{a_1a_2 + a_4(a_1 + a_2 + a_3) + a_3(a_1 + a_2)T_f + 75.85T_f}{7.55} - K_pT_f \tag{43}$$

Med de, enligt inlämningsuppgift 3 i reglerteknik, föreslagna polplacering på $-5 \pm j11$ och $-7,5 \pm j16$ erhålls följande värden: $K_p = 28.53, K_i = 241.53, K_d = 2.48, T_f = 0.04$

För den yttre hastighetsreglerande loopen väljs en PI-regulator. Då beräkningarna baseras på reglertekniksinlämningarna så används samma dimensioneringmetod för PI-regulatorn. Kraven som sätts på regulatorn är att fasmarginalen skall vara $\phi_m = 60^\circ$ och överkorsningsfrekvensen $\omega_c = 0.75\frac{rad}{s}$. Det hela reglerande systemet kan då beskrivas enligt figur 38 nedan



Figur 38: Modell över reglersystemet. Författarnas egna bild.

Den process som PI-regulatorn skall reglera blir således:

$$G_{tot} = \frac{F_{inre}G_{inre}}{1 + F_{inre}G_{inre}}G_{yttre} \tag{44}$$

Där G_{yttre} härleds från ekvation 20 med approximeringen att $\cos(\theta) = 1$ och $\sin(\theta) = 0$ för små vinklar, θ. Den yttre överföringsfunktionen blir då:

$$G_{yttre} = \frac{ds^2 - g}{s} \tag{45}$$

Fasvridning och amplituden för G_{tot} avläses ur ett bodediagram för överföringsfunktionen. Följande villkor kan då användas för att lösa ut de yttre regulatorparametrarna:

$$|F_{yttre}(j\omega_c)| = \frac{1}{|G_{tot}(j\omega_c)|} \tag{46}$$

$$\arg(F_{yttre}(j\omega_c)) = -180 + \phi_m - \arg(G_{tot}(j\omega_c)) \tag{47}$$

PI-parametrarna kan då beräknas till $K_p = 0.06846$ och $K_i = 0.03183$