

# Fear the Sphere

Planeringsstyrning för Sphero-robotar baserat på  
Augmented Reality och positionsspårning med  
kamera

**Jakob Bertlin**  
**Annie Lantz**  
**Jakob Andersson**  
**Yrsa Lifvergren**  
**Anton Pålsson**  
**Anton Sterner**  
**Johan Sundgren**

Examinator: Karljohan Lundin Palmerius

# Sammanfattning

Den sfäriska roboten Sphero 2.0 släpptes för allmänheten av företaget Sphero år 2013. Denna rapport beskriver ett medietekniskt kandidatarbete om hur robotik kan kombineras med AR (*Augmented Reality*) för att skapa en innovativ applikation för Sphero-robotar. Det är inte helt ovanligt att positionsbestämning används i samband med AR, men att kombinera detta med planeringsstyrning för Sphero-robotar är ovanligt. Användaren ska kunna styra roboten med surfplattan endast genom att peka på önskad slutposition på surfplattans skärm. Målet är att styrningen av roboten ska kunna tillämpas i ett spel samt visas upp på Visualiseringsscenter C i Norrköping under workshops inom robotik och programmering.

Projektgruppen på sju personer har under projektets gång arbetat agilt enligt en metod som liknar Scrum. Till sin hjälp har gruppen haft en UX-grupp från kursen TNM100 ”Formella metoder för användarupplevelse” samt två handledare för stöd och rådgivning. Under arbetets gång har även kundmöten genomförts för att hålla kunden uppdaterad samt se till att gruppen arbetar mot kundens krav på produkten.

Systemet består av flera separata moduler som är sammankopplade. Surfplatta, Sphero-robot och positionsspårning via kamera kommunicerar med varandra genom en server, skriven i C++, där majoriteten av beräkningsarbetet sker. Surfplattans mjukvara är baserad på spelmotorn **Unity** med in-sticksmodulen **Vuforia**, som erbjuder positionsspårning av surfplattan via markörer. Positionerna som användaren vill flytta roboten till väljs genom fingertryckning på surfplattans skärm och skickas till servern via trådlöst nätverk. I taket ovanför spelplanen sitter en kamera som spårar var roboten befinner sig i realtid. Robotens position på spelplanen och positionen skickad från surfplattan jämförs och servern beräknar hur roboten ska röra sig till önskad position. Spårningskameran uppdaterar robotens position kontinuerligt och när roboten befinner sig på korrekt plats ger servern ett nytt kommando som säger åt roboten att stanna eller ta sig till nästa angivna punkt.

Arbetet resulterade i en applikation som hanterar planeringsstyrning för Sphero-robotar. Styrningen baserar sig på spårning av surfplatta via AR med tredjepartsprogrammet **Vuforia SDK (Software Development Kit)** och även spårning av robotarna från en kamera placerad i taket baserat på **OpenCV**. All kommunikation mellan surfplatta och robot hanteras över en server. Spelaren väljer robot genom att trycka på skärmen och sedan välja till vilka punkter den ska förflyttas.

Med hjälp av en tydlig återkoppling i gränssnittet uppmanas användaren att stegevis lära sig styra Sphero-robotarna. Vid de användartester som genomfördes framgick att planeringsstyrningen var intuitiv och lättförståelig. Detta var något som gruppen strävat efter att uppnå, oberoende tidigare erfarenheter av planerad styrning hos användaren.

# Innehåll

<b>Sammanfattning</b>	<b>i</b>
<b>Figurer</b>	<b>v</b>
<b>Tabeller</b>	<b>vii</b>
<b>Konventioner, förkortningar och termer</b>	<b>viii</b>
<b>1 Inledning</b>	<b>1</b>
1.1 Bakgrund . . . . .	1
1.2 Syfte och Mål . . . . .	1
1.3 Krav . . . . .	2
1.4 Frågeställningar . . . . .	2
1.5 Avgränsningar . . . . .	2
<b>2 Relaterat arbete</b>	<b>3</b>
2.1 Sharky The Beaver . . . . .	3
2.2 Pokémon GO . . . . .	4
<b>3 Systemutveckling</b>	<b>5</b>
3.1 Utvecklingsmetodik & sprintplanering . . . . .	5
3.2 Parprogrammering . . . . .	6
3.3 Mötesrutiner . . . . .	6
3.4 Versionshantering . . . . .	6
3.5 Testning . . . . .	7
3.5.1 Integrationstester . . . . .	7
3.5.2 Användartester . . . . .	7
<b>4 System</b>	<b>8</b>
4.1 Målplattform . . . . .	8
4.2 Systemarkitektur . . . . .	8
4.3 Implementation av AR med Vuforia SDK . . . . .	9

4.3.1	Spårning med Vuforia <i>ImageTarget</i>	10
4.3.2	Positionsspårning	11
4.4	Spårning med OpenCV	12
4.4.1	<i>Blobtracking</i>	12
4.4.2	Separering av objekt med färg	13
4.4.3	Kamerakalibrering och koordinatomvandling	14
4.5	Server	16
4.6	Klient	18
4.7	Sphero 2.0 och SpheroRAW	18
4.7.1	Kommunikation med Sphero	19
4.7.2	Sphero-robotens riktning	19
4.8	Trådning	21
4.9	Återkoppling	21
4.10	Begränsningar	22
<b>5</b>	<b>Resultat</b>	<b>23</b>
<b>6</b>	<b>Analys och diskussion</b>	<b>28</b>
6.1	Metod	28
6.1.1	Versionshantering	28
6.2	Server och klient	29
6.3	Sphero 2.0	29
6.4	Multitrådad programmering	29
6.5	Spårning med webbkamera	30
6.6	Problem med modulbaserade system	30
6.7	Källkritik	31
6.8	Utvärdering av resultatet	31
6.9	Användartester	32
6.10	Samhällsaspekter	32
<b>7</b>	<b>Slutsatser</b>	<b>33</b>
7.1	Frågeställningar	33
7.2	Konsekvenser för berörd målgrupp	34
7.3	Framtida arbete	34
<b>Litteraturförteckning</b>		<b>36</b>
<b>A Involverade parter</b>		<b>39</b>
<b>B Användartester</b>		<b>40</b>

B.1 Frågor . . . . .	40
B.2 Användarnas svar . . . . .	40

# Figurer

3.1	Gantt-schema över projektets arbete . . . . .	5
4.1	Systemarkitekturen över hela systemet . . . . .	9
4.2	Prototyp-applikation för att rendera en kub mot spårningsmarkering . . . . .	9
4.3	Exempel på <i>ImageTarget</i> (markör) med få attribut . . . . .	10
4.4	Exempel på <i>ImageTarget</i> (markör) med många attribut . . . . .	10
4.5	Projektgruppens <i>ImageTarget</i> . . . . .	11
4.6	Applikations-prototyp för att konvertera användarinput . . . . .	12
4.7	Den stora bilden visar vad kameran ser, de två rutorna ”red image” och ”purple image” är de två trösklade bilderna. . . . .	13
4.8	Schackbräde som används vid kamerakalibrering. . . . .	14
4.9	Projektionsvektor $\vec{V}$ och dess skärning av planet. . . . .	15
4.10	Placering av Sphero-roboten på spelplanen vid kalibrering av ny bas sett från webbkameran. . . . .	16
4.11	Flödesschema för server . . . . .	17
4.12	Flödesschema för klient . . . . .	18
4.13	Vit Sphero 2.0 bredvid dess laddstation [28] . . . . .	19
4.14	Uträkning av vinkel mellan Sphero-roboten och dess mål . . . . .	20
4.15	Beskrivning av hur Sphero-roboten tolkar olika vinklar . . . . .	20
4.16	Återkoppling i form av pinnar samt timer till vänster och pilar till höger. . . . .	21
5.1	Skärmdump från den färdiga applikationen som visar skärmen för att ansluta till servern. . . . .	23
5.2	Skärmdump från den färdiga applikationen som visar startmenyn. . . . .	24
5.3	Information för användaren om hur applikationen används. . . . .	24
5.4	Skärmdump från den färdiga applikationen som visar startvyn för styrningen. . . . .	25
5.5	Skärmdump från den färdiga applikationen som visar vald Sphero-robot och givna koordinater för röd robot. . . . .	26
5.6	Skärmdump från den färdiga applikationen som visar vald Sphero-robot och givna koordinater för lila robot. . . . .	26
5.7	Utvecklare i projektgruppen demonstrerar färdiga applikationen. . . . .	27
6.1	Skärmdump från GitHub med fördelningen av programmeringsspråk i projektet. . . . .	31

B.1 Fördelningen på svaren i fråga 6. . . . .	42
---	----

# Tabeller

1	Konventioner	viii
2	Förkortningar	viii
3.1	Beskrivning av sprintar och milstolpar	5
A.1	Organisation och ansvarsområden	39

# Konventioner, förkortningar och termer

Tabell 1: Konventioner

Utseende	Förklaring	Exempel
<i>Kursiv stil</i>	Engelska uttryck och/eller teknisk term	Kommunicerar via <i>Bluetooth</i> ...
<b>Fet stil</b>	Programvara	Gjordes i <b>Unity</b> ...

Tabell 2: Förkortningar

Förkortningar	Förklaringar
API	<i>Application Programming Interface</i>
AR	<i>Augmented Reality</i>
BGR	Blå, grön, röd (färgrymd)
HSV	<i>Hue, Saturation, Value</i> (färgrymd)
QA	<i>Quality Assurance</i>
RGB	Röd, grön, blå (färgrymd)
SDK	<i>Software Development Kit</i>
TCP	<i>Transmission Control Protocol</i>
UDP	<i>User Datagram Protocol</i>
UX	<i>User Experience</i>

# Kapitel 1

## Inledning

Utevecklingen inom dagens teknik går snabbt framåt. Föremål som gemene man drömde om för bara 30 år sedan är nu verklighet och finns i en stor del av dagens hem. Människan bygger och anförtror sitt liv runt alla dagens smarta föremål såsom telefoner, klockor och datorer vilket skapar, om inte nästintill kräver, den ständiga utveckling som dagens teknik omringar människan med.

En teknik som visat en otrolig utveckling de senaste åren är AR (*Augmented Reality*). AR är en teknik som ger möjligheten förstärka en upplevelse genom att rendera 3D-objekt gentemot den verkliga världen. Vad som är speciellt med AR är dess breda utvecklingsområde och hur det kan tillämpas till så många olika projekt. Denna rapport kommer beskriva ett medietekniskt kandidatarbete om hur AR och robotik kan kombineras för att skapa en innovativ applikation för Sphero-robotar.

### 1.1 Bakgrund

År 2011 introducerade företaget Sphero (tidigare Orbotix) en prototyp av vad som senare skulle bli Sphero 1.0 för besökarna på CES (Consumer Electronics Show) i Las Vegas. Vilket startade ett intresse för vad som ansågs som en innovativ radiostyrd robot [1]. Den nya versionen Sphero 2.0, som används i detta projekt, släpptes för allmänheten år 2013. Tillsammans med denna kom även möjligheten att som utvecklare modifiera roboten genom att antingen använda Sphero:s egna språk OrbBasic eller olika experimentella API:n (*Application Programming Interface*). Att kunna modifiera roboten skapar möjligheter som bara utvecklarens fantasi sätter gränser för och det finns en rad olika exempel på applikationer och fritidsprojekt som involverar Sphero-roboten. Utav dessa applikationer som undersöks tycks kombinationen av Sphero-robot och AR vara mer sällsynt. Dessutom verkar de flesta applikationerna bygga på styrspaken som den officiella Sphero-applikationen använder sig av. En stor del av detta projekt ligger i att utveckla ett annorlunda sätt att styra roboten.

Inom AR-applikationer är det inte helt ovanligt att positionsbestämning av enheten används i samband med spårning och det har många olika användningsområden för spel och applikationer. Men att kombinera positionsbestämning inom AR för att skapa en planeringsstyrning för Sphero-robotar är tills nu ett utforskat område.

### 1.2 Syfte och Mål

Syftet med projektet är att utveckla en applikation där användaren styr robotar på en avgränsad spelplan. Robotarna styrs genom ett gränssnitt baserat på AR för surfplattor där användaren ser roboten på surfplattan med hjälp av surfplattans kamera. Istället för realtidsstyrning ska planerad styrning

användas. Med planerad styrning menas att användaren pekar ut en eller flera önskade positioner på skärmen, sedan kommer roboten att besöka alla de angivna punkterna. Användaren kan inte påverka redan givna kommandon, utan måste planera robotens färdväg i förväg. Målet är att styrningen av roboten ska kunna tillämpas till en spelidé i mån av tid samt visas upp på Visualiseringscenter C i Norrköping under workshops inom robotik och programmering.

## 1.3 Krav

De grundläggande punkter som skulle uppfyllas för en önskad fungerande slutprodukt var följande:

- Minst två spelare ska kunna spela samtidigt.
- Spelaren ska kunna styra två robotar parallellt.
- Planeringsstyrning ska användas istället för realtidsstyrning av robotarna.

Kunden hade som önskemål att två spelare med varsin surfplatta skulle kunna styra två robotar var och sedan på något vis spela mot varandra. Tanken med den planerade styrningen, istället för realtidsstyrning, var att se hur lättförståeligt och intuitivt det är för användaren att välja mellan och styra robotarna.

## 1.4 Frågeställningar

Nedan listas de fyra frågeställningarna som behandlas och besvaras i rapporten.

- Hur bör den planerade styrningen för Sphero-robotarna utformas tillsammans med AR för att göras så intuitiv som möjligt och ge en bra användarupplevelse?
- Vilka alternativ finns det för kommunikation mellan enheterna?
- Hur ska robotarna åtskiljas vid spårning av flera robotar samtidigt?
- Hur ska en robots positionskoordinater på en bild omvandlas till världskoordinater på spelplanen?

## 1.5 Avgränsningar

Systemet är plattformsberoende då servern är utvecklad för **Windows** och målplattformen för applikationen är **Android**-surfplattor.

Spelplanens storlek ( $1.5\text{m} \times 2.5\text{m}$ ) är anpassad för att en stor del av ytan ska ses genom surfplattans kamera utan att användaren ska behöva flytta runt alldeles för mycket. Dessutom kan planen inte vara mycket större då kamerans i taket ska kunna se och spåra robotarna på hela ytan.

# Kapitel 2

## Relaterat arbete

Att hitta relaterade projekt som liknar detta projekt var näst intill omöjligt. Spel och applikationer för Sphero-robotar och fleranvändarspel för AR har båda existerat under några år men dock på skilda håll. Utvecklingsnivån på många av de applikationer som fanns till Sphero-roboten var låg och ofta fritidsprojekt. Anledningen till detta kan vara att många av de API som Sphero-roboten erbjuder är väldigt experimentella och ibland utdaterade. Dessutom är programvaran som roboten använder sig av väldigt ostabil.

Däremot ser utvecklingsmöjligheterna och marknaden helt annorlunda ut för AR. Det finns många olika ramverk och tredjepartsprogram att använda och de flesta är väldigt väldokumenterade med många användare. Ett av målen med detta projekt var att implementera en planeringsstyrning till roboten med hjälp av AR. För att kunna göra detta möjligt bestämdes det att arbeta med **Vuforia**. Redan 2011 släpptes den första versionen av **Vuforia**:s programvara vilket tillåter privatpersoner att utveckla AR-applikationer till både **Android** och **iOS** helt gratis. Under efterföljande år har **Vuforia** utvecklats vidare till att bli ett stabilt, väldokumenterat och välanvänt tredjepartsprogram för att utveckla AR-applikationer både hemma och hos företag. Över 35 000 applikationer använder **Vuforia** för att driva sin AR [2]. **Vuforia** valdes till stor del på grund av samarbetet med spelmotorn **Unity** vilket gjorde det mycket enkelt att implementera och komma igång. 2013 skrev Alejandro Ibañez och Josep Figueras en masteruppsats där hela **Vuforia**:s programvara, uppbyggnad och tillämpning analyserades. Dessutom beskriver Ibañez och Figueras ett bra samarbete med **Unity** [3].

Exempel på applikationer med AR kombinerat med Sphero-roboten samt fleranvändarspel för AR är *Sharky The Beaver* [4] respektive *Pokémon Go*, vilka presenteras nedan.

### 2.1 Sharky The Beaver

Företaget som lanserade Sphero-roboten lyckades 2012 kombinera roboten och AR genom att introducera alla Sphero-ägare till *Sharky The Beaver*. Genom att spåra roboten renderas en figur i form av en bäver på roboten som följer alla dess positioner och rörelser. Detta öppnade upp för förändringar i det traditionella skapandet inom AR som vanligtvis följde att renderingen skedde på en fast placeras markering. Detta kombinerades med ett spel där användaren kunde kasta virtuella muffins mot roboten och på så sätt interagera med robotens animering.

## 2.2 Pokémon GO

Sommaren 2016 släpptes en beta-version av ett spel från spelutvecklarna Niantic av vad som skulle bli en viral succé. Mobilapplikationen Pokémon GO fick snabbt över en miljon användare världen runt och satte ett ordentligt avtryck i applikationsbranschen med sitt fleranvändarspel för AR. Det skapade ett intresse att utveckla fleranvändarspel som baserar spelidén på AR.

# Kapitel 3

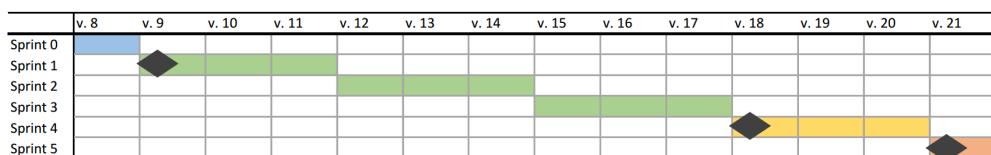
## Systemutveckling

I detta kapitel beskrivs vilken arbetsprocess som tillämpats, hur gruppen organiserats och de rutiner som funnits under projektets gång. De verktyg som används presenteras också i detta kapitel.

### 3.1 Utvecklingsmetodik & sprintplanering

Projektgruppens ambition var att arbeta efter den agila utvecklingsprocessen Scrum [5]. I vissa perioder tillämpades dock inte Scrum-ramverket till fullo. Parprogrammering användes i delar av projektet där utvecklarna kände att de behövde diskutera idéer och lösningar mer ingående med varandra.

För att nå slutprodukten delades arbetet upp i sprintar som var tre veckor långa vardera. Den ursprungliga planeringen för projektet finns representerat i figur 3.1. De färgade områdena representerar en sprint och de vridna rektanglarna representerar milstolpar. Tabell 3.1 beskriver alla sprintar och milstolpar.



Figur 3.1: Gantt-schema över projektets arbete

Tabell 3.1: Beskrivning av sprintar och milstolpar

Sprint 0	Efterforskning och planering
Sprint 1-3	Arbete på uppgifter ur produktbacklogg
Sprint 4	Förfining av produkten
Sprint 5	Avslutande ändringar
Milstolpe 1	En klar planering och en generell kunskap om området
Milstolpe 2	En fungerande produkt ska finnas
Milstolpe 3	En levererbar produkt ska finnas

Sprint 0 tillägnades planering och efterforskning för projektet där hela gruppen satt tillsammans och diskuterade vad som skulle göras och hur det skulle utföras. Efterkommande sprintar var till en början planerade som följer: sprint 1; arbeta med varje enskild komponent och få det att fungera i bästa

möjliga mån. Vidare i sprint 2 så planerades det att gruppen skulle integrera samtliga moduler och ha en fungerande prototyp för att därefter utveckla spellogiken med dess gränssnitt. Vid första sprintens slut var dock systemets komponenter långt ifrån klara. Integrationen av modulerna fick därför skjutas upp på obestämd tid tills de enskilda komponenterna fungerade enligt uppsatta mål. Det var först i slutet av sprint 3 som gruppen kunde börja integrera modulerna och få fram en fungerande prototyp. I sprint 4 lades därför tiden på testning och utökning av funktionalitet för styrning av Sphero-roboten via surfplattan. En del förenklingar fick därmed göras och gruppen fick gå ifrån kravet på att slutapplikationen skulle bli ett spel och istället fokusera på att implementera en planeringsstyrning.

Gruppen var från början överens om att först fokusera på att få applikationen att fungera för en användare. Därefter skulle systemet utvecklas till att fungera för flera användare.

## 3.2 Parprogrammering

Då gruppen lyckades integrera de enskilda komponenterna till ett system påbörjades implementationen av planeringsstyrningen. Gruppen valde då att parprogrammera för att kunna diskutera olika lösningar med varandra. Eftersom utvecklarna i tidigare sprintar hade utvecklat grunderna för komponenterna på eget håll var detta ett effektivt sätt att undvika missförstånd och oklarheter i koden och systemet.

## 3.3 Mötesrutiner

Varje dag inleddes med ett, max 15 minuter långt, scrum-möte där var och en i teamet gick igenom vad de arbetade med under gården, vad dagens uppgift var samt om något hade försvarat arbetet för utvecklaren. I början av sprint 0 genomfördes en sprintplanering där utvecklarna, utifrån produktbackloggen, bestämde i stora drag vad som skulle arbetas med. Därefter delades varje uppgift upp och sammansättades i organisationsverktyget **Trello** [6].

Utöver detta har gruppen haft två officiella möten med kunden för diskutera systemets krav. Handledningsmöten med examinator hölls vid behov både i helgrupp och mindre möten med en till två gruppmedlemmar för rådgivning vid diverse problem eller frågor. Under projektets gång hade gruppen två avstämningsmöten med examinator för att uppdatera om projektets utveckling.

Kontinuerliga möten med UX-gruppen (*User Experience*, en grupp från kursen TNM100, “Formella metoder för användarupplevelse”) hölls under projektets gång för att stämma av hur arbetet gick. De hjälpte också till med användartester av den planerade styrningen. I ett tidigt stadio fick även projektgruppen förslag på spelidéer.

## 3.4 Versionshantering

För versionshantering av den gemensamma koden användes verktyget **GitHub** [7]. Varje delgrupp använde i ett tidigt stadium egna *repositories* som arbetet sparades på. Först när alla moduler i systemet var testade var för sig och bedömdes som färdiga skapades ett gemensamt *repository*.

För att bibehålla struktur och synkronisera arbetet mellan utvecklarna krävdes det att varje enskild utvecklare höll sin kod väldokumenterad med kommentarer och väl valda variabelnamn.

## 3.5 Testning

Genom kontinuerlig testning kunde projektgruppen se till att produkten klarade de krav som var ställda av kunden från början. Detta resulterade i att utvecklarna kunde garantera QA (*Quality Assurance*), alltså att produkten håller en tillräckligt hög kvalitétsnivå.

### 3.5.1 Integrationstester

Systemet som utvecklats är helt modulbaserat och följer arkitekturen som beskrivs i avsnitt 4.2. Modulbaserade system kräver att alla moduler som implementeras in i systemet fungerar med de övriga delarna.

För att undersöka om systemet fortfarande fungerade som det skulle efter implementering av en ny modul har alltid en prototyp-applikation gått igenom ett mindre integrationstest. Testet anpassades beroende på vilken modul som testades.

Ett exempel på ett integrationstest var sammankopplingen av Sphero-styrningen med servern. Programmet som Sphero-styrningen låg i användes som bas där servern implementerades som separat funktion. Testningen gick sedan ut på att samtidigt kunna köra Sphero-styrningen parallellt med servern som tar emot paket, utan att de två delarna interagerade med varandra. När detta fungerade så förbereddes nästa steg som innebar att Sphero-styrningen skulle agera utifrån datan som hämtats från servern. I början implementerades en enkel rörelse för Sphero-roboten oavsett vad servern hade för data, som sedan byggdes ut att agera mer utifrån vad för typ av data som servern tagit emot. I varje integrationstest samarbetade minst två personer med varandra.

### 3.5.2 Användartester

Mot slutet av projektet genomfördes användartester med tio utomstående personer där användarna först fick följa instruktioner och sedan testa gränssnittet fritt. Därefter fick testpersonen svara på frågor angående upplevelsen. Detta gjordes för att utvecklarna skulle kunna utvärdera gränssnittet och undersöka huruvida det var lättförståeligt för någon som inte utvecklat systemet. Vilka frågor som besvarades samt några konstruktiva svar från de anonyma testpersonerna finns i bilaga B.

# Kapitel 4

## System

Det är många olika komponenter som ska lyckas kommunicera för att applikationen ska fungera. Detta kapitel täcker hur modulerna i systemet fungerar var för sig samt tillsammans som ett helt system.

### 4.1 Målplattform

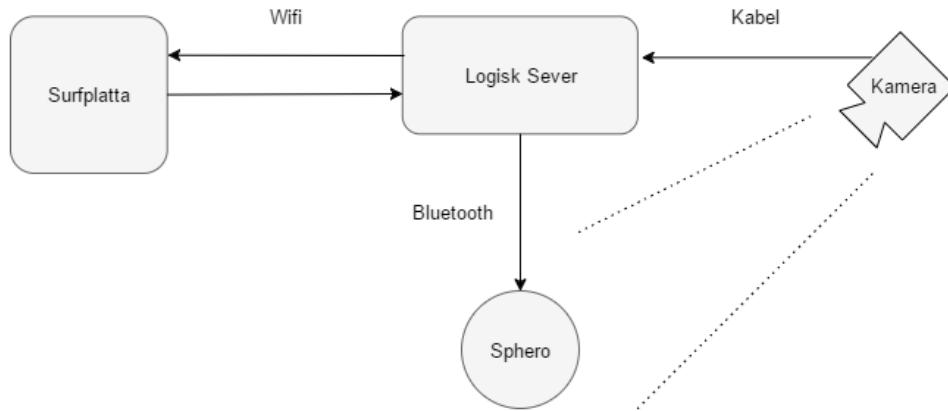
Målplattformen för applikationen är surfplattor med operativsystemet **Android**. All kommunikation sker via en server utvecklad för **Windows** då det är den plattform utvecklarna hade tillgång till samt mest erfarenhet av. Kommunikationen mellan surfplatta och server sker via trådlöst nätverk och via *Bluetooth* mellan server och robot. Planeringsstyrningen i surfplattan använder sig av AR via tredje-partprogrammet **Vuforia** och spelmotorn **Unity**. Spårningen av robotarnas position på spelplanen har implementeras med biblioteket **OpenCV** som innehåller metoder för att spåra ett, eller flera, objekts rörelser i realtidsvideo utan att markörer används.

### 4.2 Systemarkitektur

Systemet är uppbyggt grundläggande enligt den arkitektur som visas i figur 4.1. Denna arkitektur bestämdes under den första sprinten och har använts som referens under hela projektets gång. De moduler som ingår är; en surfplatta, en server, en webbkamera och två Sphero-robotar.

Webbkameran i taket används för att spåra robotens position därför att robotens egna positionsangivelse inte är tillräckligt noggrann och tillförlitlig.

Då användaren väljer nästa position för Sphero-roboten med en tryckning på surfplattans skärm skickas koordinaterna för den positionen via trådlöst nätverk till logikservern. De angivna koordinaterna jämförs med robotens position, vilken hämtas från spårningskameran. Vinkel och hastighet räknas ut i servern och skickas sedan till roboten. Roboten rör sig sedan längs den givna vinkeln medan spårningskameran i taket spårar dess position i realtid. Hastigheten för roboten minskas då roboten är inom en radie av 50 centimeter från önskad position. När roboten befinner sig inom en 30 centimeters radie anses den vara tillräckligt nära målet och ett stopp-kommando skickas.

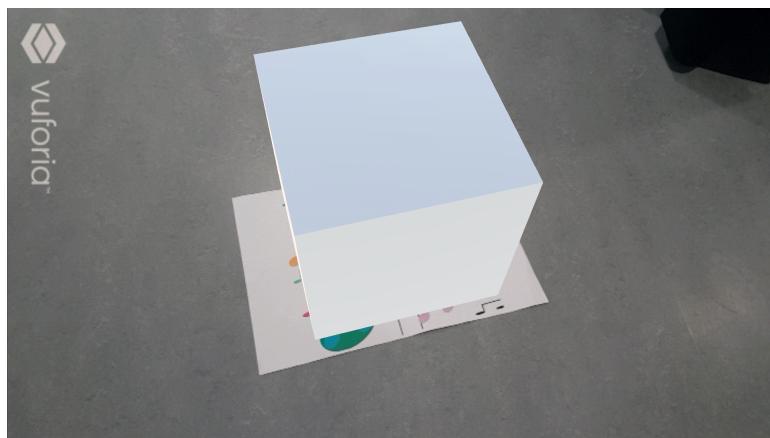


Figur 4.1: Systemarkitekturen över hela systemet

### 4.3 Implementation av AR med Vuforia SDK

För att styra Sphero-robotarna skulle enligt kraven en planeringsstyrning baserad på AR tillämpas. Detta innebär att användaren trycker på en punkt på surfplattans skärm vilket ska ge robotens nästa position. Markörspårning via surfplattans kamera krävs för att hålla reda på surfplattans position gentemot spelplanen och i förlängningen var på spelplanen användaren trycker. Olika alternativ för AR och markörspårning granskades och projektgruppen valde att använda spelmotorn **Unity** med **Vuforia SDK** (*Software Development Kit*) som insticksmodul, vilka har metoder för att underlättा utveckling mot **Android**. **Unity** beskrivs vidare i avsnitt 4.6.

Att komma igång med **Vuforia** i **Unity** var en enkel process då det finns många guider och videoer att tillgå på internet som går igenom grunderna. Redan en timme från start, helt utan förkunskaper, hade en prototyp-applikation skapats för att kunna rendera en kub mot spårningsmarkören, se figur 4.2. Att sedan gå vidare från det var betydligt mer komplicerat och krävde en stor mängd efterforskning för att komma fram till hur positionsangivningen skulle gå till.

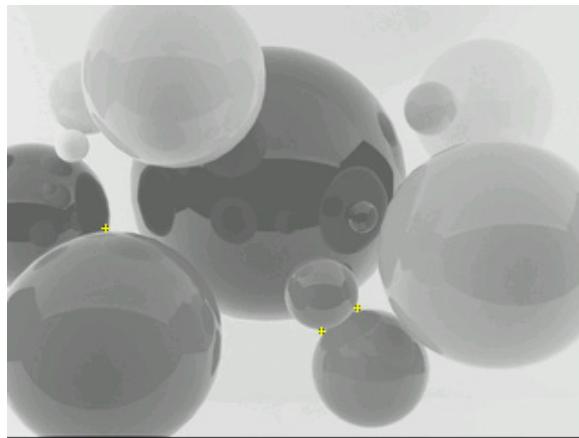


Figur 4.2: Prototyp-applikation för att rendera en kub mot spårningsmarkering

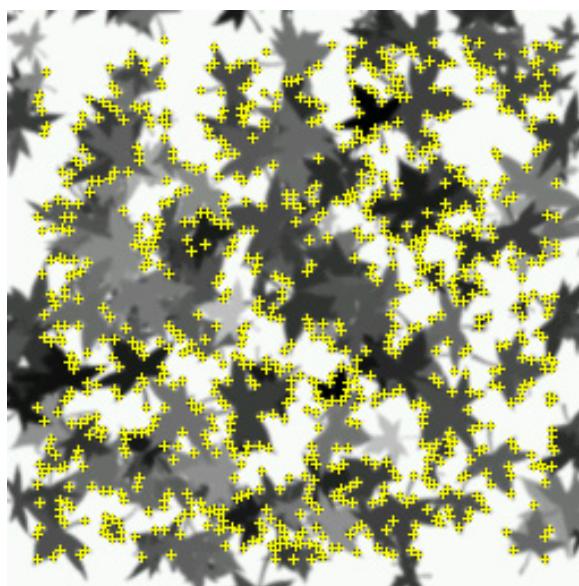
### 4.3.1 Spårning med Vuforia *ImageTarget*

**Vuforia** kan med hjälp av en kamera på enheten upptäcka och spåra intressepunkter i förutbestämda bilder (markörer) i realtidsvideo. Markörerna som används för spårning kallas i **Vuforia** för *ImageTarget* och kräver att bilden är uppbyggd med många särskilda attribut. Dessa attribut hittas i skärningspunkter som uppstår i bildens mönster och med hjälp av dessa kan kameran spåra bildens position och rendera objekt mot bilden. Det går att analysera hur bra bilden kan spåras genom **Vuforias Developer Portal**. Bilderna bedöms med en till fem stjärnor. Figur 4.3 visar en bild med få attribut vilket ger ett lågt betyg och som inte bör användas. Det är viktigt att bilden undviker repetitiva mönster då det sänker spårningens prestanda. En bild som innehåller många attribut och icke-repetitiva mönster kommer få ett högt betyg och kommer ge en stabil spårning [9], se figur 4.4.

När bilden ska importeras som en *ImageTarget* i **Unity** måste en databas i **Vuforias Developer Portal** skapas. Exakta mått på hur stor bilden är måste anges. Om måtten inte är rätt kan sambanden mellan spårningens intressepunkter bli inkorrekt. Sedan importeras databasen in i **Unity** och kan då användas till vidare utveckling av applikationen.



Figur 4.3: Exempel på *ImageTarget* (markör) med få attribut



Figur 4.4: Exempel på *ImageTarget* (markör) med många attribut

Markeringen som projektgruppen valde att använda är giraffer placerade i olika riktningar och färger. Att just giraffer valdes som motiv är inte kopplat till hur bra spårningen är. Mönstrets olika riktningar och icke-repetitiva mönster skapade en bra markör och gavs ett betyg på fem stjärnor i **Vuforia**s egna bedömning. Markören visas i figur 4.5.



Figur 4.5: Projektgruppens *ImageTarget*

### 4.3.2 Positionsspårning

För att kunna styra Sphero-roboten via surfplattan behöver den input som ges av användaren omvandlas till världskoordinater. Dessa koordinater ska sedan jämföras med de koordinater som ges av spårningen av robotarna från spårningskameran, se avsnitt 4.4. Mycket av arbetet med **Vuforia** gick till att lösa problemet med att omvandla den input som ges användaren till användbara koordinater för systemet. Metoden som används för att bestämma den givna positionen som användaren ger i världskoordinater kallas *Ray Casting*. När surfplattans kamera spårar given *ImageTarget* skapas ett osynligt plan gentemot markeringen med origo i mitten av bilden. Detta plan är då parallellt med ytan som roboten rör sig runt på. När planet skapats kan *Ray Casting* tillämpas. När användaren trycker på önskad position på skärmen skickas en stråle till den positionen i surfplattans kamera-riktning för att kolla om det finns något objekt att kollidera med. När strålen träffar det givna osynliga planeten kan avståndet från planeten till kameran tas fram och genom det kan punkten på planeten hämtas [10]. Koordinaterna för punkten som hämtas från användarens skärmtryckning skickas sedan till servern som diskuteras i 4.5.

Implementationen av *Ray Casting* för att hämta ut positionen baserades på funktioner i *Physics*-delen av **Unity**'s *Scripting API*, ett bibliotek som består av ett stort antal metoder för fysiska tillämpningar. En enkel prototyp för att hämta användarens önskade position visas i figur 4.6. Den gröna cirkeln representerar origo och den röda cirkeln visar positionen som valdes genom en skärmtryckning. Koordinaterna som skickas till servern skrivs ut där de visar att röda punkten är placerad cirka 43 cm i x-riktning och 3 cm i z-riktning från origo samt att avståndet från kameran till punkten är cirka 83 cm. Vanligtvis diskuteras (x,y) när koordinater nämns men i **Unity** går y-axeln rakt upp från marken, därför nämns koordinaterna i (x,z) istället. När koordinaterna sedan skickats till servern omvandlas de till (x,y) för att kunna hanteras av servern.



Figur 4.6: Applikations-prototyp för att konvertera användarinput

## 4.4 Spårning med OpenCV

Det finns många olika metoder för att spåra ett objekt i bilder och video. Det som främst skiljer metoderna åt är om de spårar en markör eller ej. En markör är en bild som placeras på ett objekt för att kunna urskilja objektet från omgivningen. För spårning av Sphero-robotarna är det inte möjligt att använda markörer eftersom markören inte alltid skulle vara synlig, därför valdes biblioteket **OpenCV** som innehåller metoder för att spåra objekt i bilder och video. Det är det mest kompletta öppna biblioteket inom datorseende, med stöd för program skrivna i C, C++, Python och Java. Kameran som användes för spårning av robotarna är en webbkamera av märket Logitech.

### 4.4.1 Blobtracking

För att hitta ett objekt på en bild måste det finnas specifikationer för vad det är för objekt som ska identifieras. *Blobtracking* syftar på att objektet som spåras är cirkulärt. *SimpleBlobDetector* [11] är en färdig metod, som inkluderas i **OpenCV**, för att spåra objekt. Metoden kan upptäcka och följa objekt utifrån ett antal olika parametrar [12]:

- Färg (egentligen intensitet, då det är en gråskalebild som används i detektorn)
- Storlek (area, antal pixlar i bilden)
- Cirkuläritet
- Konvexitet
- Tröghetsmoment (hur mycket objekten är utsträckta på skärmen, beror på objektens hastighet och kamerans uppdateringsfrekvens)

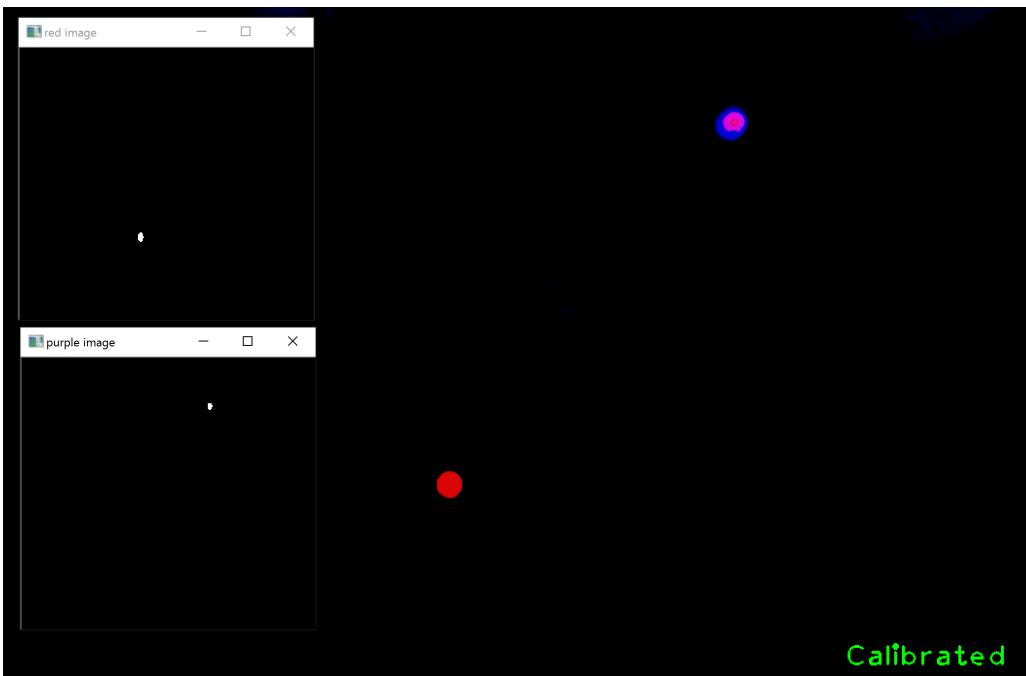
Alla dessa parametrar behöver inte vara aktiva samtidigt, exempelvis kan objekt spåras endast efter cirkuläritet och helt ignorera de andra parametrarna. Positionerna av alla objekt som hittas i bilden lagras i en vektor. De koordinater som lagras i positionsvektorn är de koordinater objektet har på bilden. I de fall detektorn hittar fler än ett objekt, kommer de ligga i vektorn i den ordning de hittas i bilden. Detektorn startar sin sökning i bildkoordinaterna  $(u, v) = (0, 0)$ , vilket motsvarar pixeln i det

övre vänstra hörnet av en bild. Därefter läser den av bilden rad för rad. Det betyder att om två objekt byter plats i höjdled i bilden, kommer deras koordinater också byta plats i positionsvektorn. Det gör det svårt att hålla isär objekten och att med säkerhet kunna identifiera dem. För att lösa detta problem utökas detektorn med ytterligare funktionalitet, vilket beskrivs under i 4.4.2.

#### 4.4.2 Separering av objekt med färg

För att flera Sphero-robotar ska kunna spåras samtidigt måste de hållas isär så att inte koordinaterna blandas ihop. I detta projekt utnyttjades att Sphero-robotarna har inbyggda RGB-dioder (röd, grön blå) som kan få roboten att lysa i olika färger. Kameran kunde inte uppfatta robotarnas färg när kameras standardinställningar användes, därför ändrades inställningarna (*gain*, *saturation*, *brightness*, *exposure*, *contrast*) för att tydliggöra färgerna från robotens dioder. Eftersom *SimpleBlobDetector* inte kan skilja på objekt utifrån färg, implementerades färgmasker för att få fram bilder där endast färger inom ett visst omfång syntes. För varje robot skapades en mask för den valda färgen på roboten, de färger som inte var inom rätt färgomfång filtrerades bort. Maskerna skapades genom att först omvandla färgerna från **OpenCVs** standard BGR-färgrymd (blå, grön, röd) till HSV-färgrymd [13]. HSV står för *hue* (nyans), *saturation* (mättnad) och *value* (intensitet). Ett intervall av HSV-värden definierades för varje färg på robotarna som skulle spåras. Koden som användes för att identifiera färgerna i HSV-värden är skriven av Kyle Hounslow [14]. För spårning av två Sphero-robotar valdes färgerna röd och lila.

Bilden trösklades med de olika färgintervallen, vilket resulterade i bilder där det enda som syntes var just den färgen som eftersöktes, se figur 4.7. För att fylla ut ”hål” i objekten användes bildbehandlingsmetoden *morphological closing* [15], vilket är *dilation* följt av *erosion*. De trösklade bilderna användes sedan i separata detektorer för att särskilja robotarnas positioner. Positionen som detektorn ger är endast objektets position i bildkoordinater. För att dessa koordinater ska kunna användas av de andra komponenterna i applikationen måste de transformeras till världskoordinater.

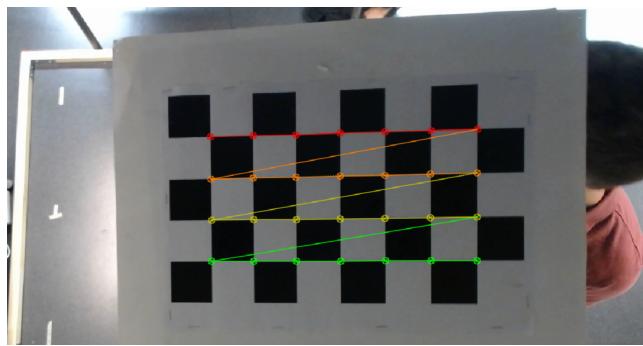


Figur 4.7: Den stora bilden visar vad kameran ser, de två rutorna ”red image” och ”purple image” är de två trösklade bilderna.

### 4.4.3 Kamerakalibrering och koordinatomvandling

När användaren har tryckt på en position på surfplattan ska återkoppling från spårningen ske för att visa var roboten befinner sig i förhållande till den position som användaren angivit. Robotens färdriktning räknas ut i servern med hjälp av dessa två positioner. För att detta ska fungera måste koordinatsystemen för surfplattan och spårningen av roboten stämma överens. Dessa två koordinatsystem ska alltså utgå från samma punkt på spelplanen. Ur kamerabilderna kan endast robotarnas bildkoordinater hämtas, men det som är intressant är robotarnas position på spelplanen. Därför måste bildkoordinaterna transformeras till koordinater på spelplanen (värlskoordinater).

För att skapa ett korrekt värlskoordinatsystem via webbkameran kalibrerades kameran med ett schackbrädemönster för att få fram kamerans *extrinsic*- och *intrinsic*-parametrar [16]. Dessa parametrar beskriver kamerans egenskaper och är nödvändiga att känna till vid transformationen från koordinater på bilden till värlskoordinater på spelplanen. Vid kalibrering av kameran togs 20 bilder. Efter kalibrering sparas kalibreringsdata i en XML-fil som kan läsas in vid varje uppstart av programmet. Om kameran flyttas till en ny position måste kalibrering av kameran ske på nytt. Kodén för kalibrering som användes är en något modifierad version av exempelkod som följer med i **OpenCV** [17]. Under kalibreringen flyttas schackbrädet runt i kamerans synfält och roterades i olika riktningar. I figur 4.8 ses schackbrädet och de spårade intressepunkterna. Med hjälp av schackbrädets position och rotation i dessa bilder uppskattades kamerans *intrinsic*-parametrar. Matrisen med *Intrinsic*-parametrarna innehåller kamerans brännvidd, bildens mittpunkt och bildsensorsns skevhetskoefficient mellan x- och y-axel, vilken oftast är noll. Denna matris brukar kallas för kameramatrisen [18].



Figur 4.8: Schackbräde som används vid kamerakalibrering.

I den första bilden i kalibreringssekvensen var schackbrädet placerat på golvet i mitten av spelplanen. Ur denna bils *extrinsic*-parametrar hämtades schackbrädets rotation och translation i x-, y- och z-led i förhållande till kamerans position, vilket i sedan kan användas för att beräkna spelplanens normalvektor. *Extrinsic*-parametrarna till en bild i kalibreringen innehåller sex olika koordinater; tre för rotationen och tre för translationen av schackmönstret i förhållande till kamerans position. En rotation anges ofta som en  $3 \times 3$ -matris, men **OpenCV** ger en  $1 \times 3$ -vektor. Därför räknades vektorom till en matris för att kunna användas i koordinatomvandlingen, med en metod som kallas *Rodrigues-rotation*.

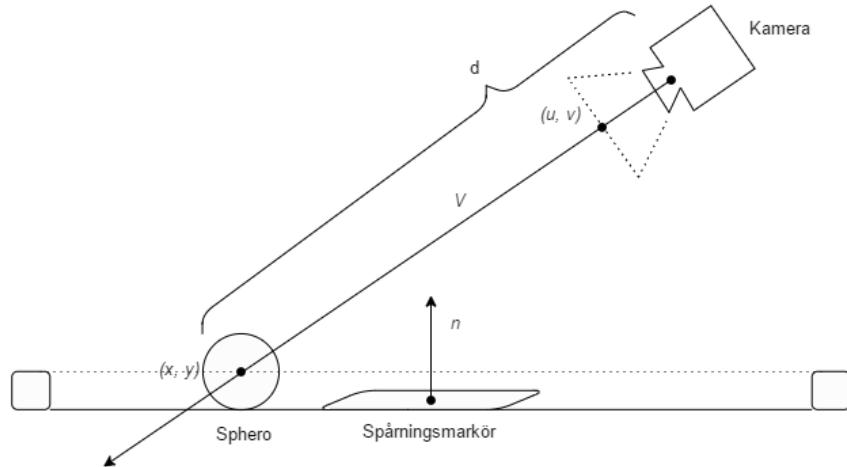
Rotationskoordinaterna roterades i en *Rodrigues-rotation*, vilket resulterar i en  $3 \times 3$ -matris. I matrisen inverterades y- och z-koordinaterna, detta för att normalen i nästa steg ska peka upp ur spelplanen. Matrisen roterades sedan i en kvarternion [19] och multipliceras med en vektor  $(0, 0, 1)$  för att få normalen till spelplanen. Dessa uträkningar har implementerats med hjälp av ett kodbibliotek för linjär algebra skrivet av Karljohan Lundin Palmerius, vilket är en del av *KJs Algebra 3D Package* [20].

En ny koordinatbas för spelplanen skapades för att axlarna i värlskoordinatsystemet ska stämma överens med spelplanens utformning. För att kunna beräkna en ny bas måste bildkoordinaterna först

omvandlas till världskoordinater. För att göra det beräknades en projektionsvektor  $\vec{V}$  och dess längd  $d$  ut enligt (4.1) respektive (4.2). Projektionsvektorn är en linje som går från kameran, genom en bildkoordinat  $(u, v)$  i kamerans *frustum* och vidare till en punkt  $(x, y)$  på spelplanen. Figur 4.9 visar kamerans projektionsvektor  $\vec{V}$  och dess skärning med planet vid Sphero-robotens position. Projektionsvektorn beräknades med hjälp av *intrinsic*-parametrarna från kalibreringen, där  $\vec{V}$  är projektionsvektorn från kameran till robotens position,  $u$  och  $v$  är Sphero-robotens position i bildkoordinater,  $f_x$  är kamerans brännvidd i pixlar och  $c_x$  tillsammans med  $c_y$  representerar bildens mittpunkt [16]. Kamerans position (translationskoordinaterna) är  $p_k$ ,  $n$  är normalen och  $\vec{V}$  är projektionsvektorn [21].

$$\vec{V} = \left( \frac{u - c_x}{f_x}, \frac{v - c_y}{f_x}, 1 \right) \quad (4.1)$$

$$d = \frac{p_k \cdot n}{\vec{V} \cdot n} \quad (4.2)$$

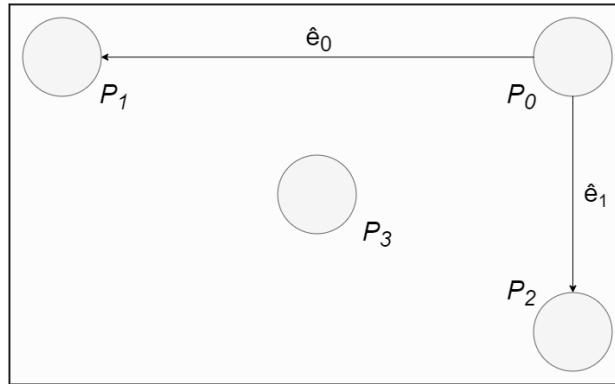


Figur 4.9: Projektionsvektor  $\vec{V}$  och dess skärning av planet.

Projektionsvektorn normaliseras och därefter beräknades punkten  $p$  där projektionsvektorn och normalen korsas.  $p$  är punkten där roboten som spåras befinner sig i världskoordinater, för att beräkna  $p$  multiplicerades  $\vec{V}$  med  $d$ .

När kameran kalibrerats skapades en ny bas för att göra ett koordinatsystem på golvet med origo i mitten av spelplanen. Genom att placera en av Sphero-robotarna på tre olika punkter ( $p_0, p_1, p_2$ ) i spelplanen beräknades två basvektorer till det nya koordinatsystemet (4.3). Roboten placerades också i mitten av spelplanen för att få positionen av önskat origo ( $p_3$ ), se figur 4.10. Detta origo sammanfaller med det origo som används av **Vuforia** i surfplattan. Det enda som skiljer koordinatsystemen åt är en skalfaktor, då surfplattan ger koordinater mätt i meter och kameraspårningen ger koordinater mätt i millimeter.

$$\begin{cases} \vec{e}_0 = \frac{\vec{p}_1 - \vec{p}_0}{|\vec{p}_1 - \vec{p}_0|} \\ \vec{e}_1 = \frac{\vec{p}_2 - \vec{p}_0}{|\vec{p}_2 - \vec{p}_0|} \end{cases} \quad (4.3)$$



Figur 4.10: Placering av Sphero-roboten på spelplanen vid kalibrering av ny bas sett från webbkameran.

När webbkameran har identifierat en robot med hjälp av *Blobtracking* i punkt  $p$  subtraheras  $p_3$  (positionen för origo) från  $p$ , (4.4), innan  $p$  multipliceras med basvektorerna för att få fram koordinaterna  $(x, y)$  i den nya basen, (4.5).

$$\vec{p} = \vec{p} - \vec{p}_3 \quad (4.4)$$

$$\begin{cases} x = \vec{p} \cdot \vec{e}_0 \\ y = \vec{p} \cdot \vec{e}_1 \end{cases} \quad (4.5)$$

De uträknade koordinaterna  $(x, y)$  skickas vidare till servern för att ge återkoppling till roboten om dess position.

## 4.5 Server

Projektgruppen har utvecklat en server som följer en klient-server-modell. Klient-server innebär att flera enheter ingår i ett *WiFi*-nätverk där en av enheterna agerar server [22]. Servern tar emot och skickar paket till klienterna som utgörs av surfplatta och Sphero-robotar. I systemet finns det fyra stycken olika enheter, bestående av Sphero-robotar, surfplatta, spårningskamera och server. Dessa tre olika enheter kommunicerar med en server som sköter majoriteten av de beräkningar som är nödvändiga för att enheter ska röra sig enhetligt över spelplanen.

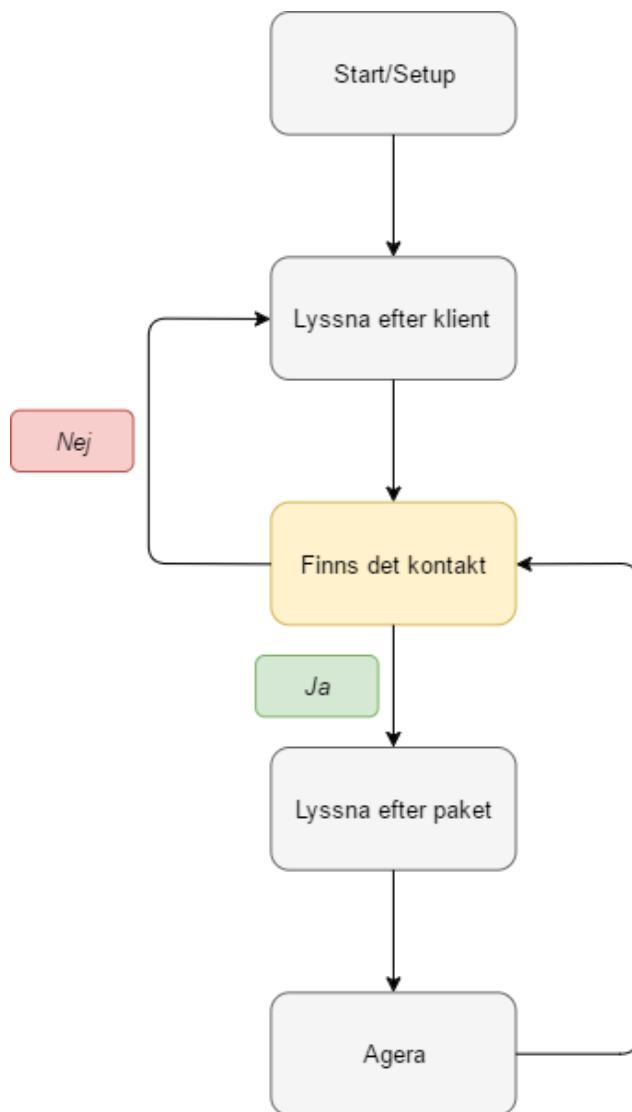
Servern kommunicerar genom *.NET-sockets* och är huvudsakligen skriven i C++. Ramverket .NET är utvecklat av Microsoft och är utvecklat för att underlätta för utvecklare att utveckla applikationer för Windowsplattformar [23]. I projektet användes ramverket till utvecklingen av nätverkskommunikation mellan surfplatta och server.

Servern kommunicerar med klienterna genom protokollet TCP (*Transmission Control Protocol*). Fördelarna med TCP är att de datapaket som skickas med hjälp av protokollet garanterat kommer fram till rätt mottagare, i rätt ordning och att data i paketet inte är korrupt. En av nackdelarna är att på grund av protokollets robusta metod blir servern långsammare på att skicka och ta emot paket jämfört med andra protokoll. Alternativa protokoll för kommunikation mellan server och klient är UDP (*User Datagram Protocol*) eller att utveckla *Raw sockets*. UDP är snabbare än TCP men det beror på att många av de kontroller som gör TCP så robust slopas till förmån av snabbare kommunikation. Kommunikation genom UDP garanterar varken att paket som skickas kommer fram eller att de kommer

fram i den ordning som de skickas i. *Raw sockets* innebär att utvecklaren själv väljer vad som ska ingå i protokollet vilket ger utvecklaren större möjlighet att anpassa paketen av data och hur de hanteras av servern, efter systemet [24].

Datapaketet skickas genom buffrade strömmar där paketen består av strängar med koordinater. Koordinaterna särskiljs med ett blanksteg. Innan paketen skickas serialiseras innehållet till en serie av bytes. Ordningen av bytes är enligt standarden *Network Byte Ordering* och när paketen tas emot av servern sker en omvänt serialisering, det vill säga bytes konverteras tillbaka till en sträng med koordinater [24].

Kommunikationen mellan servern och dess klienter illustreras i figur 4.11. Servern startas och börjar att lyssna efter klienter. När en klient skickar en förbindelseförfrågan som accepteras av servern öppnas en ström för kommunikation och servern börjar lyssna efter paket. Därefter agerar servern på olika sätt beroende på vilket paket som tagits emot från klienten. Bryts kontakten med klienten går server tillbaka till att lyssna efter en ny klient.

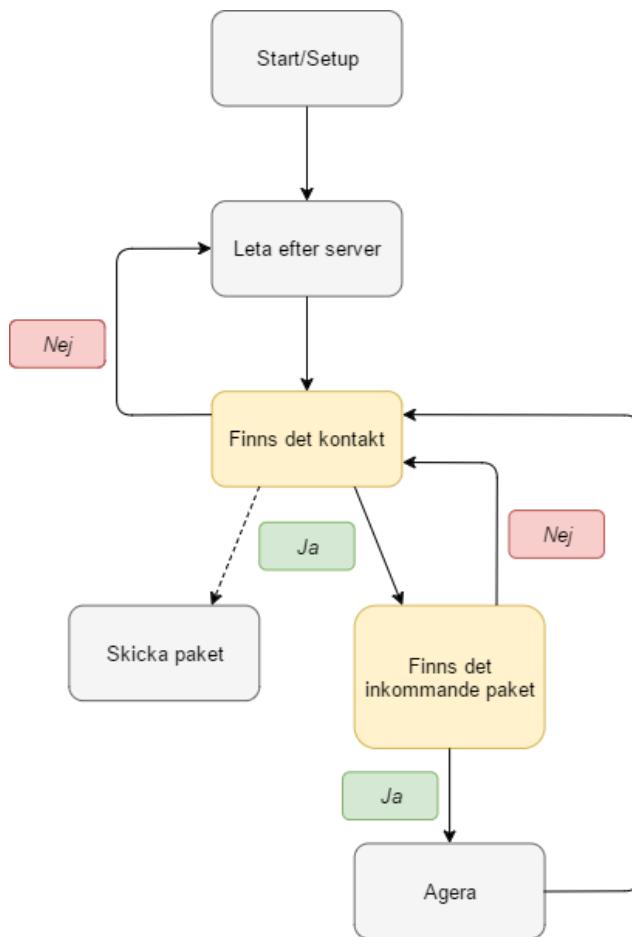


Figur 4.11: Flödesschema för server

## 4.6 Klient

Applikationen för surfplattan är uppbyggd i **Unity**. **Unity** är en multiplattform-spelmotor utvecklad av företaget Unity Technologies och stöder programmeringsspråken C# och JavaScript. Eftersom servern använder .NET som vanligtvis skrivs i C# och applikationen ska kunna köras på **Android** passar **Unity** bra som utvecklingsmiljö. I **Unity** skapas scener som kopplas ihop med olika scripts. Scenen är det som visas för användaren och scripten är funktionerna som körs i bakgrunden.

Nätverksdelens händelseförflopp i klienten illustreras i figur 4.12. Efter start skickar klienten en förbindelseförfrågan till servern. Därefter påbörjas en loop där den buffrade strömmen kollas och agerar därefter. Om klienten skickar ett nytt paket till servern skapas en ny processtråd som avslutas när handlingen är klar.



Figur 4.12: Flödesschema för klient

## 4.7 Sphero 2.0 och SpheroRAW

Sphero 2.0 är en rund robot täckt av ett skal i plast. Inne i skalet finns två stycken motorer som driver och har möjlighet att snurra skalet vilket gör att roboten kan röra sig framåt. Den är utrustad med accelerometer och gyroskop som används för att Roboten alltid ska kunna orientera sig i rummet och veta i vilken riktning den pekar i realtid [26]. Sphero 2.0 finns i en rad olika utseenden men den klassiskt vita är vad som används i detta projekt och exempel på en sådan illustreras i 4.13. [27]



Figur 4.13: Vit Sphero 2.0 bredvid dess laddstation [28]

### 4.7.1 Kommunikation med Sphero

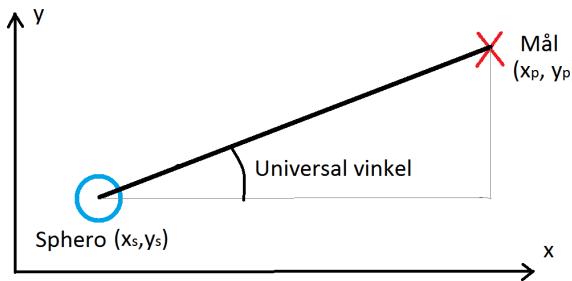
Sphero-roboten kommunicerar med enkla kommandon över ett standard *Bluetooth* protokoll. I vanliga fall används en telefon av **Android** eller **iOS** med Spheros officiella applikation för att styra robotten. Det finns även möjlighet att öppna en *Bluetooth*-port till en Sphero-robot från vilken enhet som helst. Detta utnyttjas i projektet vilket har lett till användningen av C++ biblioteket *SpheroRAW*. *SpheroRAW* är ett C++ bibliotek med öppen licens och källkod skapat av Paul Freund [29]. Biblioteket är gjort för att kommunicera med en Sphero-robot över *Bluetooth* från framförallt en **Windows**-enhets. Till skillnad från mobilapplikationen till Sphero-roboten så är *SpheroRAW* en lågnivåkommunikation, menat att skapa sina egna beteenden till en Sphero. Detta innebär en rad möjligheter men likväl många hinder. Till exempel måste kommandon till Sphero-roboten skickas i specifika ordningar för att mjukvaran inte ska krascha och starta om. Något som även måste hanteras manuellt är en instruktionskö, som maximalt kan innehålla 256 bytes av information. Robotten kraschar om instruktionsköns minnesmängd överskrids, därfor sker en rensning av instruktionsköen med jämna mellanrum.

För att styra robotten i godtycklig riktning används ett enkelt *roll*-kommando med tre argument; hastighet, vinkel och beteende. De två första argumenten beskriver hur fort sphero-roboten ska röra sig respektive vilken rörelseriktning i grader. Tredje och sista argumentet beskrivs av siffrorna 0-2 och motsvarar: bromsning (0), standardrörelse (1) respektive tvingad rörelse (2) [30]. Vid bromsning använder Sphero-roboten all sin potentiella kraft för att omedelbart stanna en eventuellt pågående rörelse. Standardrörelsen innebär att Sphero-roboten försöker göra en rörelse med så hög precision som möjligt i riktningen som har angivits. Anges istället en tvingad rörelse försöker Sphero-roboten röra sig mot riktningen så fort som möjligt oavsett tidigare hastigheter eller riktningar.

### 4.7.2 Sphero-robotens riktning

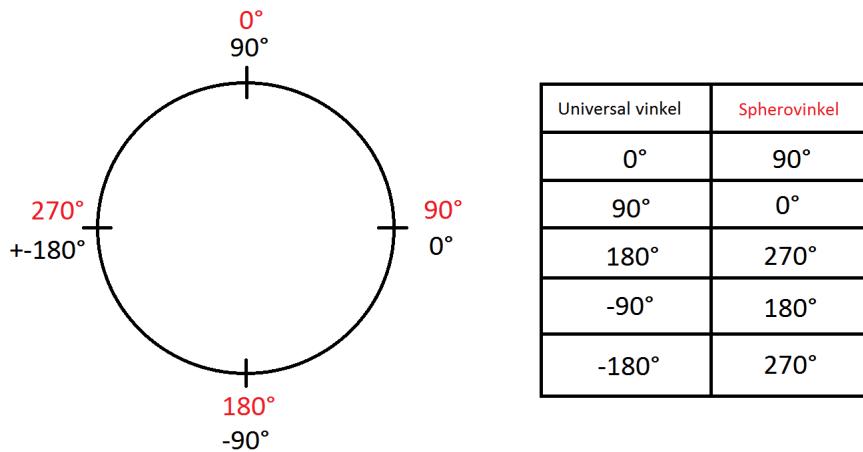
Som beskrivet i avsnitt 4.4 spåras Sphero-roboten av en webbkamera i taket som sparar dess x- och y-position på spelplanen. Vidare fås även ett spelardefinierat mål från surfplattan. För att Sphero-roboten ska färdas i rätt riktning beräknas vinkeln mellan robotten och målet. Riktningen beräknas från definitionen av skalärprodukten som syns i 4.6, där vinkeln  $V$  är vad som efterfrågas. Vidare illustration av problemet demonstreras även i figur 4.14.

$$(x_s, y_s) \cdot (x_p, y_p) = |(x_s, y_s)| |(x_p, y_p)| \cos(V) \quad (4.6)$$



Figur 4.14: Uträkning av vinkel mellan Sphero-roboten och dess mål

Sphero-roboten använder inte det standardiserade systemet med vinklar mellan  $[-180, 180]$  där vinkeln utgår från noll grader på positiva x-axeln. Istället så har Sphero-roboten noll grader i sin positiva y-riktning och vinkeln varierar mellan  $[0, 360]$  [30]. Skillnaden illustreras i figur 4.15.



Figur 4.15: Beskrivning av hur Sphero-roboten tolkar olika vinklar

Det går inte heller att direkt konvertera från universala vinklar till Sphero-vinklar eftersom att en Sphero-robot utgår från sin egna y-axel som nollvinkel. Dessutom anser den sin y-axel vara den riktning som Sphero-roboten vaknar i efter att ha legat i standby. Detta innebär att innan något *roll*-kommando kan skickas så måste det utföras ett test för att upptäcka vilken vinkel Sphero-robotens y-axel är på. Det görs genom att röra roboten längs sin egen y-axel (noll grader) och därefter räkna ut vinkeln mellan den och den globala y-axeln. Testet utförs vid uppstart av programmet, direkt efter att servern har kopplat till roboten. Denna vinkel används sedan som *offset* för att skicka roboten i den korrekta, avsedda riktningen.

## 4.8 Trådning

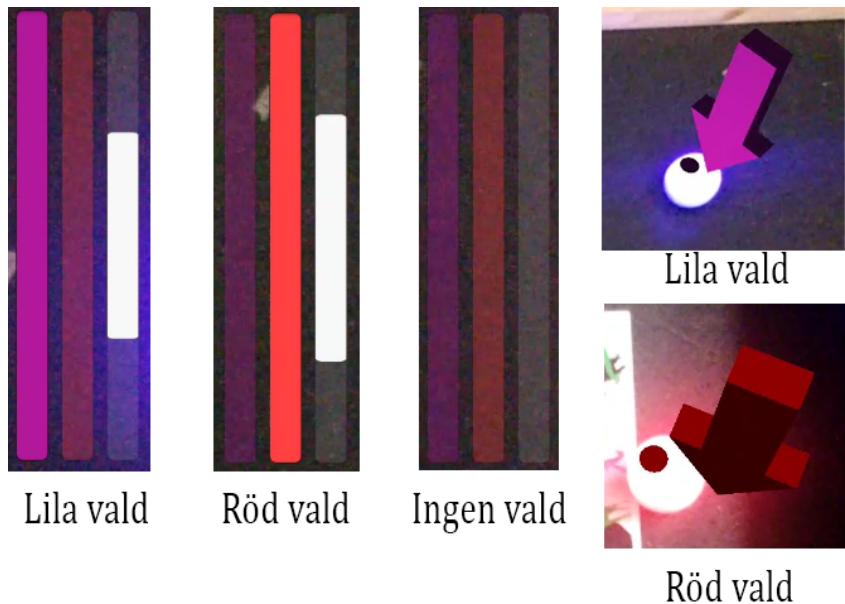
Eftersom den slutgiltiga produkten innehåller flera separata beräkningstunga delar på serversidan så var det naturliga valet att använda multitrådning. Det kan exekveras upp till fem stycken trådar parallellt på servern. Delarna som använder sig av trådning är som följer: nätverkskommunikation, spårning med webbkamera, styrning av upp till två stycken spheros på varsin tråd och slutligen så körs en huvudtråd.

## 4.9 Återkoppling

För att ge användaren en lättförståelig applikation var ett krav att någon visuell återkoppling skulle implementeras. Fyra olika sorters återkopplingar implementerades, två olika vid vald robot, en vid utplacering av nya positioner för roboten samt en timer som räknar ned från senaste klick tills koordinaterna skickas.

Då en robot väljs kommer en av två pinnar, placerad på vänster sida av skärmen, lysa upp i samma färg som den robot som valdes (se till vänster i figur 4.16). För att göra det ännu tydligare vilken robot som valdes renderas en stor pil i samma färg som vald robot ovanför som pekar ned mot den (visas till höger i figur 4.16). Pilens modellerades i **3DSMax** vilket är ett modelleringsprogram skapat av företaget Autodesk.

Till sist implementerades en timer (den vita stapeln i figur 4.16) vilket visar hur långt det är kvar tills valda koordinater skickas till roboten. Efter varje tryck som användaren gör startar timern om. Det tar det två sekunder innan timern har räknat ner till noll och surfplattan skickar vidare koordinaterna till servern.



Figur 4.16: Återkoppling i form av pinnar samt timer till vänster och pilar till höger.

## 4.10 Begränsningar

*Bluetooth*-räckvidden för roboten är begränsad vilket gör att enheten som är ansluten till roboten bör finnas inom en radie på max 10 meter för att Sphero-roboten inte ska tappa kontakten med servern.

Både robotarna och surfplattorna har begränsad batteritid. Tre timmars laddning ger en timmes körtid för en Sphero-robot. Detta innebär att om de inte laddas med jämn mellanrum kan det inte garanteras att användaren kan köra applikationen närsomhelst då båda robotarna används samtidigt och det inte finns några fler.

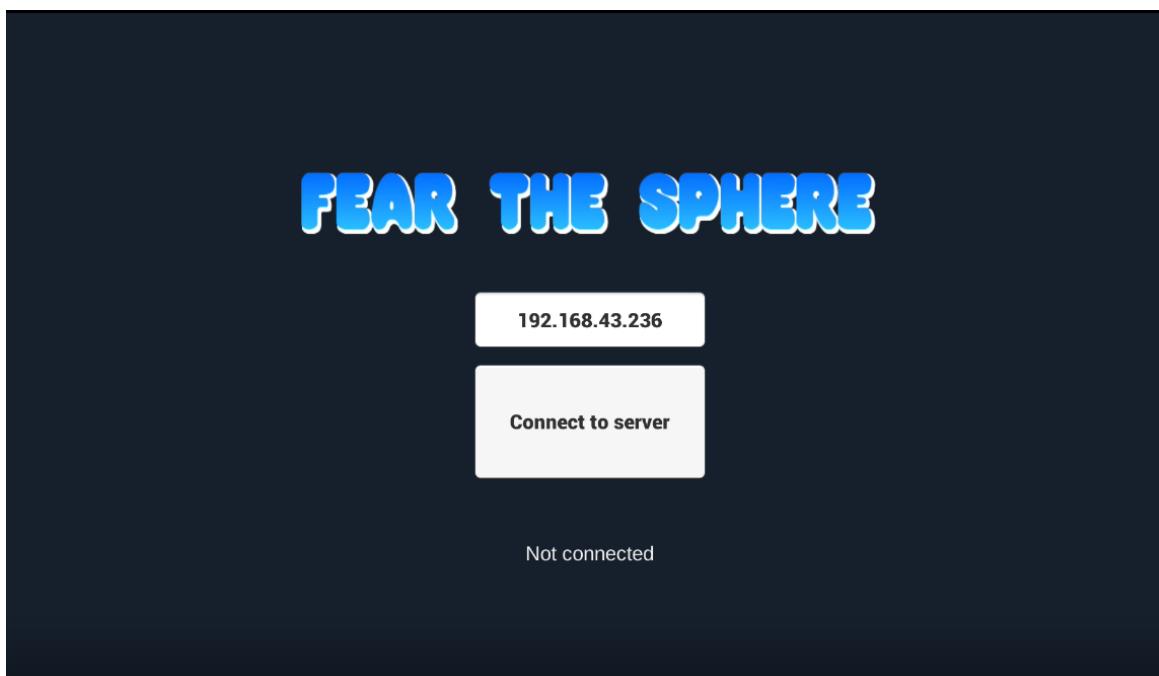
Det krävs att surfplattan är placerad så att den inte tappar spårningen mot markören, ifall det skulle ske kommer inte användaren kunna placera ut några nya punkter. Om surfplattan skulle förlora spårningen medan användaren håller på att att placera ut punkter kommer den avbryta och skicka iväg de koordinater som valdes innan spårningen tappades. Slutligen behöver spårningen från surfplattan mycket ljus för att fungera optimalt, samtidigt som kameran i taket inte kan ta in för mycket ljus från omgivningen för att kunna se och spåra robotarnas dioder tydligt.

# Kapitel 5

## Resultat

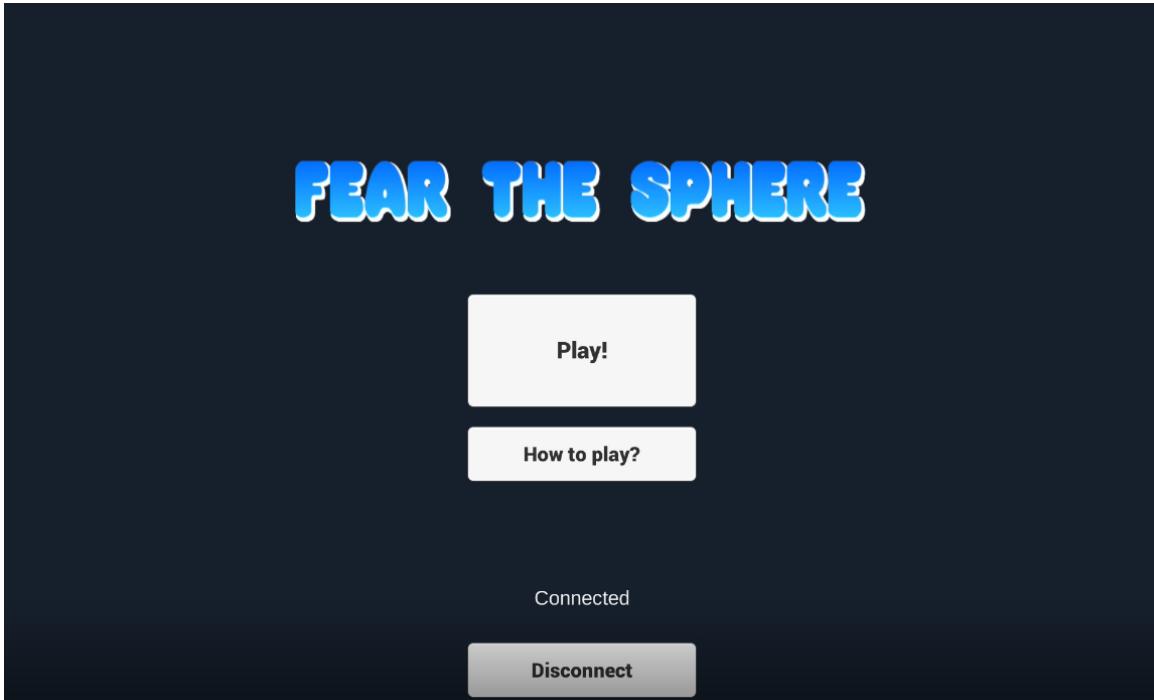
Resultatet av projektet är en applikation som hanterar planeringsstyrning för Sphero-robotar och sköter all kommunikation via en server. Genom att spåra en markering i markhöjd med surfplattans kamera kan användaren skicka nya positioner för roboten via skärmtryckningar till önskad position. Därefter sker all kommunikation via en server. En kamera placerad i taket spårar robotarnas position och ser till att roboten åker till önskad position.

Figur 5.1 visar hur användargränssnittet ser ut efter applikationen startas. Användaren skriver in en serverns IP-adress och trycker sedan på knappen med texten ”Connect to server” för att komma vidare till startmenyn som visas i figur 5.2.

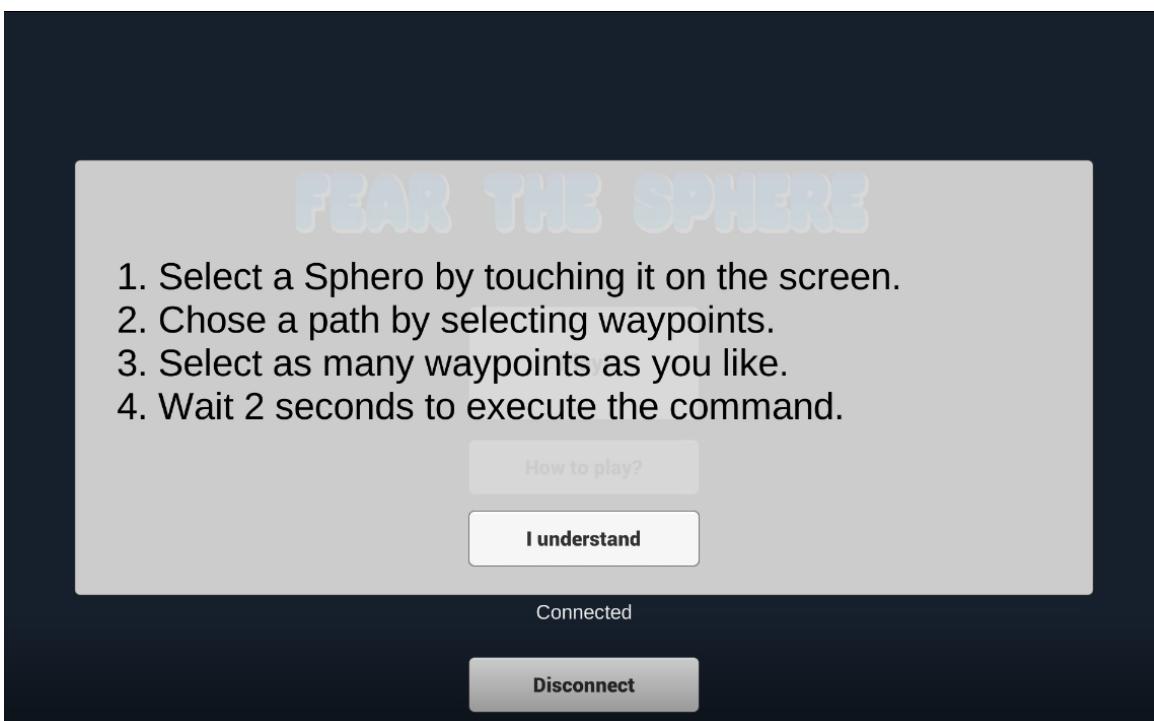


Figur 5.1: Skärmdump från den färdiga applikationen som visar skärmen för att ansluta till servern.

Från användargränssnittet som visas i figur 5.2 kan användaren gå vidare direkt till styrningen för robotarna och genom att trycka på knappen med texten ”How to play?” kan användaren få information om hur robotarna ska styras. Informationen som användaren får visas i figur 5.3.



Figur 5.2: Skärmdump från den färdiga applikationen som visar startmenyn.



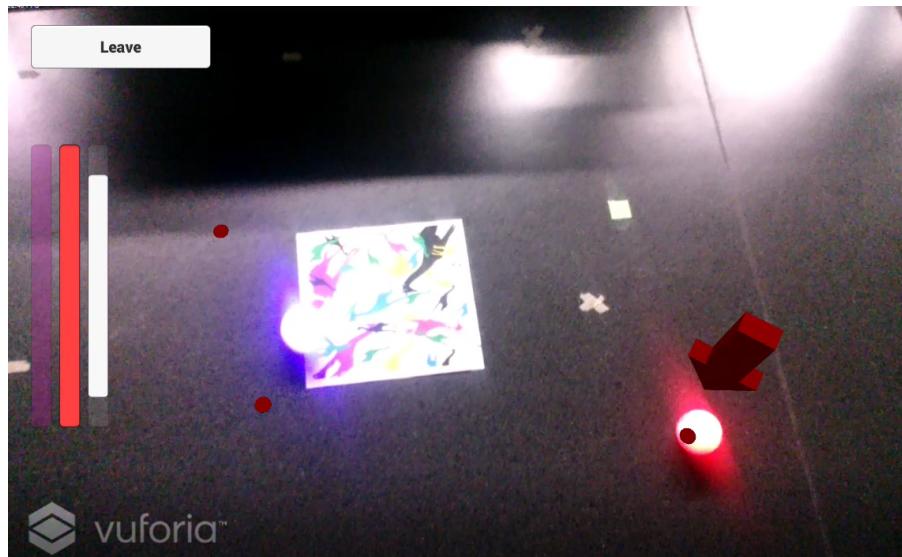
Figur 5.3: Information för användaren om hur applikationen används.

När användaren är redo att styra robotarna trycker den på knappen med texten "Play!" och kommer vidare till vyn som visas i figur 5.4. Användaren måste först se till att surfplattan hittar spårningsmarkören innan robotarna kan börja styras. Under tiden spårningsmarkören inte är registrerad av surfplattan visas texten NO TRACKING" i mitten av skärmen.

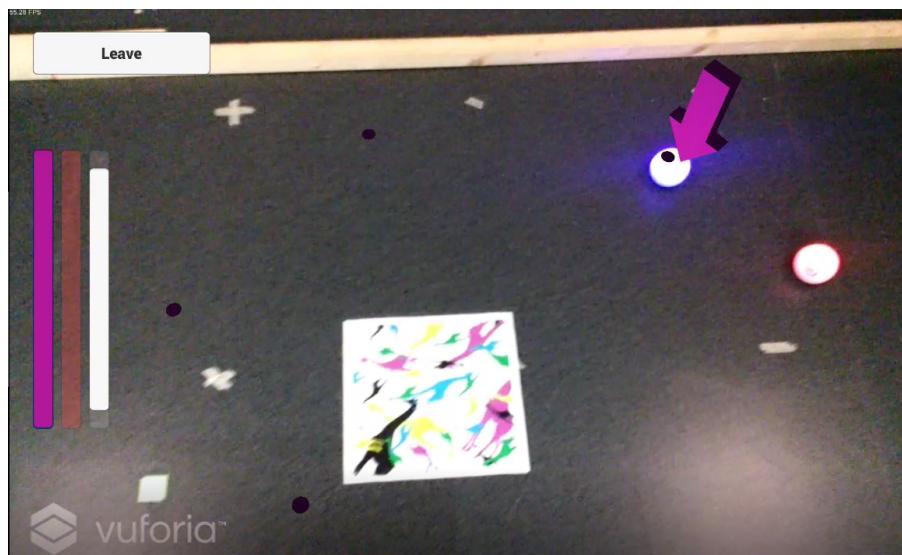


Figur 5.4: Skärmdump från den färdiga applikationen som visar startvyn för styrningen.

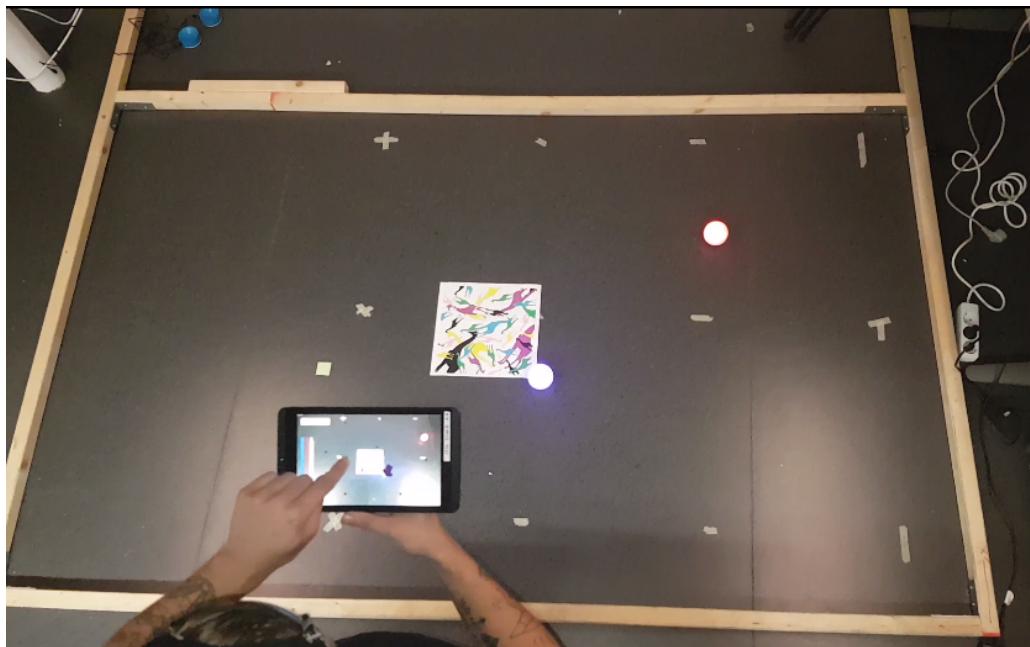
När spårningsmarkören är registrerad väljer användaren en robot genom att trycka på den, sedan trycker användaren ut önskade positioner för den valda roboten. Pilarna i figur 5.5 och figur 5.6 visar vald robot, de röda och lila prickarna beskriver användarens valda positioner att skicka till servern. De tre staplarna på vänster sida beskriver också vald robot (lila eller röd) och en tidtagare på två sekunder som börjar om efter varje ny position användaren placerar ut. När tidtagaren nått sitt slut skickas koordinaterna till servern för att sedan skickas till roboten. I figur 5.7 demonstreras det färdiga resultatet.



Figur 5.5: Skärmdump från den färdiga applikationen som visar vald Sphero-robot och givna koordinater för röd robot.



Figur 5.6: Skärmdump från den färdiga applikationen som visar vald Sphero-robot och givna koordinater för lila robot.



Figur 5.7: Utvecklare i projektgruppen demonstrerar färdiga applikationen.

# Kapitel 6

## Analys och diskussion

I detta kapitel diskuteras och analyseras utvecklingsprocessen, resultatet och vad som kunde göras annorlunda för att avsluta projektet med ett bättre resultat.

### 6.1 Metod

I princip alla områden av detta projekt var okända områden för samtliga medlemmar. Det var därför svårt att uppskatta hur lång tid en viss uppgift skulle ta att genomföra. Gruppen satte för höga mål i sprint 1 vilket resulterade i att sprintmålen inte uppnåddes. De efterföljande sprintarna innehöll därför mycket jobb som släpat efter. Efter den första sprinten kom gruppen till insikt att planering för en hel sprint var i princip omöjlig att genomföra, och gruppen började istället att planera uppgifterna med kortare framförhållning. Det innebar att inte planera de dagliga uppgifterna veckor i förväg, utan att sätta upp ett större mål och sedan uppdatera uppgifterna dag för dag. Vid de dagliga *scrum*-mötena diskuterade gruppen hur problem kunde lösas, och det var ofta på de mötena nya utmaningar och uppgifter konkretiseras. I och med att informationssökning kring de problem som uppstod tog upp en stor del av tiden, kom gruppen fram till att det inte var någon större idé att gå strikt efter den projektplan som fanns. Det skulle inte ha gett gruppen högre effektivitet att planera arbetet mer ingående eller för en längre tid framåt, då varje dag gav nya problem att lösa. Onödig tid hade då gått till att analysera alla små steg av utvecklingen innan gruppen hade nån egentlig kunskap om hur problemen skulle lösas. I detta skede ansåg projektgruppen att den bästa lösningen var att flytta milstolparna för projektet då det var svårt att uppskatta hur lång tid varje sprint skulle kräva.

Det är en av grundprinciperna i agilt arbete att vara flexibel och kunna ändra riktning i arbetet om det krävs. Gruppen kanske har tagit denna princip alltför bokstavligt och haft för lite struktur i arbetet.

Då gruppen till viss del tappade sprintarna var möten med utvärdering och återblick något som snarare skedde sporadiskt när gruppen kände att det behövdes. De dagliga *scrum*-mötena var gruppen däremot noggranna med för att alla skulle ha en bra överblick på projektet samt vad som behövde göras och vad som redan var klart.

#### 6.1.1 Versionshantering

En gemensam versionshantering för hela gruppen prioriterades inte till en början, då alla moduler i projektet är så pass separerade funktionsmässigt att det var svårt för en gruppmedlem som inte var insatt att bidra med kod. Detta gjorde att gruppen satte upp en gemensam versionshantering som alla kunde bidra till först när alla moduler skulle integreras.

## 6.2 Server och klient

Att ta fram en server som kunde fylla kundens krav var en stor utmaning och skapade mycket förseningar för projektgruppen. Enligt planeringen skulle en kontakt mellan server och klient finnas efter sprint 1 som en grund att bygga vidare på i kommande sprintar. Ett antal olika tekniker prövades innan en fungerande server utvecklades. Första server-prototypen skapades i C++ med **Windows Socket API**, förkortat Winsock eller WSA. Ytterligare en simpel klient togs fram och kommunikation mellan två datorer kunde ske över ett nätverk. Problemet med detta var när klienten skulle implementeras i surfplattan då gruppen antingen var tvungna att skriva hela klienten i Java med Android Studios eller i C# med Unity. Detta kom med för stora utmaningar och efter många misslyckade försök att skicka paket mellan parterna kasseras Winsock API:t och en ny server-prototyp skapades. Den nya servern och klienten gjordes istället i Unity och med Unitys egna nätverks-API. Detta var enklare då båda parterna var skrivna med samma programmeringsspråk. Kommunikation över nätverk mellan två datorer fungerade bra men när klienten kördes på en surfplatta fick den ingen kontakt. Det blev ingen klarhet vad som förhindrade kontakten eller hur man kunde gå runt de och därfor skapades ännu en ny server-prototyp. Denna gången skriven med .NET. Vid sprint 3 fick projektgruppen fram en fungerande grund med kommunikation mellan dator och surfplatta med .NET.

## 6.3 Sphero 2.0

Sphero-roboten har varit källan till många problem som projektet stött på. Bland de första felet som uppstod var att roboten kraschade eller dog om den fick för många kommandon. Detta lösades genom att applicera en algoritm som rensar kommando-kön ofta för att undvika att Sphero-robotens minnesmängd överskrider. Det fungerade med den implementerade planeringsstyrningen då den inte låter användaren skicka en stor mängd olika kommandon till roboten, via servern, utan bara koordinater till de punkter den ska köra till.

Gruppen har haft problem med att implementera hur robotarna ska stanna. Målet var att den ska sakta in och åka längsammare ju närmare den önskade positionen den befinner sig. Roboten får stanna-kommandot när den befinner sig på rätt position men den är ändå för långsam på att stanna vilket resulterar i att den stannar för sent. Ett försök att åtgärda detta gjordes genom att ge stoppkommandot om roboten befinner sig inom tre decimeter från den angivna punkten. Detta bidrar till att det med största sannolikhet kommer att finnas ett fel i färdvägen eller på positionen där den stannar. När Sphero-roboten beräknar hur den ska åka för att komma till nästa position är den tillåten att ha en felvinkel på fem grader i körriktningen och kommer inte korrigera förrän felvinkeln överstiger det.

Roboten dör ibland under körning. Det är oklart om det är på grund av att den kraschar trots rensning av instruktionskön eller om det är för att den har varit vaken för länge utan att få ett nytt kommando. Detta försökte åtgärdas genom att servern kan återansluta till roboten under körning. Detta fungerade dock inte helt felfritt, för att starta sfären igen måste roboten lyftas upp och det gjorde att dess vinkel blev fel. Om vinkeln är fel kommer inte roboten att åka åt rätt håll och därmed inte komma fram till den angivna punkten. Tidsbegränsningen på detta projekt innebar att robotens styrning inte kunde finslipas mer.

## 6.4 Multitrådad programmering

Trådningen användes i första hand på grund av den prestandaökning som det gav, särskilt när det gäller att spåra föremål med kameran samt nätverkskommunikationen. Det användes även i styrningen av robotarna där koden består av många funktioner där systemet väntar i upp till 200 ms innan den utför

nästa kommando, detta på grund av att Sphero-roboten kraschar om den får för många kommandon under kort tid. Om trådningen inte hade använts på dessa delar hade systemet spenderat mycket tid på att endast vänta. Detta hade kunnat lösas på annat sätt men på grund av att gruppen redan implementerat trådar på både nätverk och kamera var det den enklaste vägen att gå och arbete kunde då riktas på andra viktigare områden.

Problem som kan uppstå när många trådar skapas är till exempel att två parallella trådar vill skriva till samma variabel samtidigt. I detta projekt är denna påverkan minimal då det är få eller inga variabler som hanteras i olika trådar samtidigt. Ett undantag är nätverkskommunikationen som används av nästintill varje tråd. Det krävde extra mycket översyn för att en tråd inte skulle skriva över ett paket som en annan tråd ville skicka.

## 6.5 Spårning med webbkamera

Spårning av objekt med kamera är ett område som ingen i gruppen hade tidigare erfarenheter av. Startsträckan blev väldigt lång innan ett beslut kunde tas kring vilken metod som var bäst att använda. Till en början lades mycket tid på efterforskningar och experiment med kodexempel från **OpenCV** och bloggar kring användandet av OpenCV, bland annat *Learn OpenCV* [31].

I projektet används en Logitech-webbkamera. Gruppen undersökte även om en Kinectkamera var ett bra alternativ. Det som hade varit fördelaktigt med en Kinectkamera är att den mäter djupet i bilden. Då gruppen redan hade börjat implementera kod i **OpenCV** visade det sig att det skulle bli krångligt att använda en Kinectkamera. Biblioteket **OpenNI** för Kinect skulle behövas installeras och **OpenCV** byggas om för att det skulle fungera. Gruppen hade redan haft en del problem med att få **OpenCV** att fungera och kände att det var bättre att använda sig av en kamera som redan fungerade istället för att lägga tid på att få Kinectkameran att fungera.

I avsnitt 4.4.2 beskrivs hur objekt separerades med hjälp av dess färg. Denna lösning är inte helt utan begränsningar. Kamerans standardinställningar har svårt att se vilken färg robotarna lyser i och måste därmed ställas om för att enbart se robotarna, den övriga belysningen i rummet kan därför inte vara för stark. Det ger då konsekvenser för spårningen av markören i **Vuforia**, som behöver vara väl upplyst för att inte tappas bort av kameran i surfplattan. Det blir därför en balansakt att ställa in ljuset i rummet så att båda spårningssystemen fungerar samtidigt.

Ett annat område som gav problem var valet av färg på Sphero-robotarna. Först valdes röd och grön färg på robotarna vid kontroll av två stycken samtidigt. Den gröna färgen var enligt spårningskameran för lik det blåaktiga skenet från surfplattans skärm, vilket gjorde att spårningen ibland plockade upp surfplattans position också. Det löstes genom att ställa in den ena Spherons dioder till en lila färg. Om fler robotar ska anslutas i framtiden måste testning ske för att hitta ytterligare färger som kan åtskiljas från varandra, alternativt ändra metod för att särskilja robotarna.

## 6.6 Problem med modulbaserade system

Vid utveckling av systemet märktes snabbt att det stora problemet för att komma vidare med projektet var att det var modulbaserat. Själva ryggraden för systemet är den server som innehöll de flesta mäder och det var också den del som var mest komplicerad, speciellt då ingen av gruppmedlemmarna hade förkunskaper om många ämnen. Detta resulterade i att servern tog längst tid att implementera. Utvecklarna hamnade i en position där de stod stilla utvecklingsmässigt. Detta ledde till att under mittperioden av projektet stod utvecklingsprocessen i vissa delar still tills problemet med servern löstes. I och med detta påverkades sprint-planeringen negativt, som nämns ovan.

Att hitta en lösning till detta problem kan vara problematiskt då det är svårt att helt utan förkunskap veta hur lång tid varje modul kommer ta att implementera. I ett projekt där utvecklingsteamet erhåller spetskompetens inom de olika ämnena som täcks kan en mer kompetent planering göras och då undvika att utvecklingsprocessen någon gång stannar.

I och med ett så stort projekt innehållande många moduler och programmeringsspråk (se figur 6.1). så kunde gruppen inte gå igenom samlig kod i projektet tillsammans och sätta sig in i den som det var tänkt från början då det hade varit väldigt tidskrävande. Däremot tillämpades en väldigt stor del parprogrammering där utvecklarna kunde diskutera fram olika lösningar på problemen tillsammans.



Figur 6.1: Skärmdump från GitHub med fördelningen av programmeringsspråk i projektet.

## 6.7 Källkritik

Källorna i detta projekt har i första hand baserats på utgivarnas egna officiella dokumentation. Ett exempel på detta är **Unity** och **Vuforia** som har omfattande dokumentation om sina respektive mjukvaror. När det gäller sådana källor har gruppen var att se dessa som trovärdiga på grund av antagandet att ingen vet bättre än utgivarna av mjukvaran själva. **OpenCV** har däremot bristande dokumentation när det gäller vissa delar av biblioteket. Här var utvecklarna ofta tvungna att leta efter alternativa källor som kunde förklara hur metoder fungerade och hur de implementeras.

Det positiva med att arbeta med agil utveckling är att utvecklarna hela tiden arbetar mot att skapa fungerande mjukvara efter varje inkrement. I ett projekt som detta så innebär implementation av olika bibliotek även mycket *trial and error*, vilket gör att när en källa hittas så testas den oftast i programmet för att se om den fungerar eller inte. Problem som gruppen stötte på var också ofta vanliga och programmeringsrelaterade. Det finns många hemidor där dessa problem lösas och snabba svar ges av användare på sidan, exempel på sådana sidor är **Stack Overflow** [32]. Om resultatet då blir som förväntat vid testning kan källan antas vara trovärdig.

## 6.8 Utvärdering av resultatet

Kunden önskade från början att det skulle vara möjligt för två användare att styra varsin två Sphero-robotar samtidigt. Det innebär att två surfplattor och fyra robotar skulle kommunicera med servern samtidigt. Gruppen fokuserade på att få applikationen att fungera för en användare först; med en surfplatta och två robotar, för att senare kunna lägga till en eller fler användare. Det tog längre tid att komma så långt än vad gruppen trodde så de hann inte utöka implementeringen till två användare.

Målet var att styrningen skulle implementeras för att skapa ett spel. En av idéerna var att det skulle renderas diamanter i två olika färger och sedan skulle användaren fånga diamanten med den färgen som matchade Sphero-robotens färg och på så sätt få poäng. På grund av tidsbrist hann grunden bara läggas för detta spel och ingen fungerande prototyp hann skapas. Istället prioriterades en intuitiv återkoppling för byte mellan robotar för en användare för att då ha en bättre grund att senare kunna utveckla ett spel om intresse skulle finnas.

Trots att resultatet inte helt följer de krav som gavs av kunden från början är gruppen mycket nöjda med vad som åstadkommits under projektets tid. Då alla delar i projektet var helt främmande för gruppmedlemmarna har mycket ny kunskap erhållits. Framförallt har gruppen stött på stora hinder på vägen, lärt sig hur dessa tas itu med och sedan går vidare från det.

## 6.9 Användartester

Testpersonerna som deltog i användartesterna var i åldern 21 - 27 och ger därmed en något missvisande bild av resultatet då målgruppen (besökare på Visualiseringsscenter C) berör alla åldrar och ska vara intuitivt för såväl yngre barn som äldre vuxna. Svaren som angavs var för gruppen väldigt givande. Även om få av de som deltog hade tidigare erfarenheter utav planerad styrning upplevde merparten av testpersonerna att st var intuitivt. Samtliga testpersoner fann även bytet mellan robotar lättförståeligt. De ändringar som gjordes i gränssnittet utifrån användartesterna var bland annat att den gröna bollen som först indikerade vilken robot som var vald byttes ut till en roterande pil i samma färg som roboten. De prickar som visar valda *way points* var redan i samma färg som vald Sphero-robot men detta tydliggjordes genom att lägga till en lampa i **Unity**-scenen för bättre kontrast. Slutligen lades en timer till i vyn som visade hur lång tid det var kvar tills koordinaterna automatiskt skickades för den valda Sphero-roboten. Alltså en nedräkning på 2 sekunder från senast tryckta position. Utöver detta så ändrades det så att de prickar som visas då koordinaterna trycks ut är gråa så länge ingen robot är vald för att minska risken för förvirring och öka tydligheten att dessa klick inte registreras.

## 6.10 Samhällsaspekter

Produkten utvecklades hela tiden med tanken att den skulle användas av Visualiseringsscenter C under workshops och liknande. Därför har komponenterna som bygger upp systemet inte på något sätt granskats gällande kostnadseffektivitet eller hur de skulle kunna göras smidigare. Att göra detta till en produkt för gemene man skulle kräva en hel ombyggnad av systemarkitekturen. Det stora problemet med den aktuella systemarkitekturen är att att den inte går att göras helt mobil då spårningskameran i taket måste finnas för att ge återkoppling för robotens position. Därför skulle en produkt gjord för allmänheten då både vara osmidig, svår att hantera tekniskt och framförallt mycket dyr.

# Kapitel 7

## Slutsatser

Här presenteras de slutsatser som projektgruppen har kommit fram till. De frågeställningar som ställdes i början av projektet besvaras, konsekvenser för den berörda målgruppen och hur projektet skulle kunna utvecklas diskuteras.

### 7.1 Frågeställningar

Nedan besvaras de frågeställningar som togs upp i inledningskapitlet.

- Hur bör den planerade styrningen för Sphero-robotarna utformas tillsammans med AR för att göras så intuitiv som möjligt och ge en bra användarupplevelse?

En tydlig återkoppling, i form av renderade prickar på surfplattan, där användaren just har tryckt ut en position ger användaren chansen att se robotens valda väg då styrningen inte sker i realtid. Detta tillsammans med en stor snurrande pil i samma färg som vald robot indikerar vilken robot som är vald att styra för tillfället. Utöver pilen finns även staplar på sidan av skärmen som lyser i samma färg som den valda Sphero-roboten. Den tydliga återkopplingen minskar risken för förvirring vid spelandet och gör användarupplevelsen bättre, något som bekräftades vid användartesterna som genomfördes.

- Vilka alternativ finns det för kommunikation mellan enheterna?

Kommunikationen mellan surfplatta och server sker genom en *WiFi*-uppkoppling. De alternativa metoderna för kommunikation mellan surfplatta och server innehåller *Bluetooth* eller att surfplattan är kopplad genom en USB-kabel. Att använda kabel är möjligt men högst opraktiskt med tanke på att användaren ska kunna röra sig mer eller mindre fritt kring spelplanen. Vad gäller *Bluetooth* är det ett rimligare alternativ men här blir det snarare en fråga om hastighet och vad som är enklare att implementera. Valet att låta surfplattan kommunicera genom en *WiFi*-uppkoppling motiverades helt enkelt av att det är ett snabbare kommunikationsmedel och enklare att implementera.

Olika protokoll utvärderades men projektgruppen beslutade att protokollet TCP skulle passa projektets systemarkitektur bäst. TCP garanterar att de paket av data som skickas kommer fram, att de kommer fram i rätt ordning och att datan inte är korrupt, vilket gjorde TCP som det självklara valet för projektet. När koordinater skickas från surfplatta till server som sedan skickar vidare till Sphero-roboten är det viktigt att paketet inte förloras någonstans på vägen. Om detta skedde skulle spelet inte att upplevas responsivt och användaren skulle behöva skicka om sina koordinater. Andra protokoll såsom UDP eller *Raw Sockets* går att anpassa så att även de kontrollerar att paketen som skickas kommer fram. Projektgruppen beslutade att det inte var aktuellt att modifiera dessa typer av protokoll

då det redan fanns ett protokoll som uppfyllde kraven för systemet. Systemet kräver ett robust och stabilt protokoll framför ett snabbt men oberäkneligt protokoll för att skicka paket av data. Därför kommunicerar servern genom TCP.

När det kommer till Sphero-roboten kommunicerar den enbart via *Bluetooth*. Här hade alltså gruppen inget alternativ vad gäller kommunikationen mellan server och Sphero-robot.

- Hur ska robotarna åtskiljas från varandra vid spårning av flera robotar samtidigt?

Varje Sphero-robot innehåller RGB-dioder som aktiveras vid start och varje robot tilldelas en unik färg. Spårningskamerans bildinställningar ändras för att öka kontrast och färgfyllnad så att färgerna syns tydligt. En mask appliceras på bilden för att separera färgerna och göra den ursprungliga bilden till flera bilder där endast den sökta färgen syns i varje bild. Vid spårning av två robotar får två bilder och objekt i bilderna förstärks med hjälp av *morphological closing* för att objekten på bilden ska få en tydlig rund form. Varje bild har en egen detektor för att upptäcka roboten och få ut dess koordinater. Dessa koordinater skickas sedan vidare till servern. Färgerna på robotarna valdes så att maskerna inte skulle vara alltför lika varandra för att undvika förväxling samt för att spårningskameran i taket inte skulle råka spåra surfplattans skärm.

- Hur ska en robots positionskoordinater på en bild omvandlas till världskoordinater på spelplanen?

Kameran kalibreras först en gång med ett schackbrädemönster för att räkna ut dess *extrinsic*- och *intrinsic*-parametrar, vilket ger kamerans egenskaper. Ur *extrinsic*-parametrarna hämtas spelplanens position och rotation i förhållande till kamerans position. Med denna information beräknas sedan spelplanens normalvektor. Skärningen mellan en projektionsvektor från kameran och spelplanens normalvektor ger en punkt  $(x, y)$  i världskoordinater.

Utifrån tre hörnpunkter på spelplanen beräknas två basvektorer med samma orientering som spelplanen. De uppmätta koordinaterna vid spelplanens mittpunkt subtraheras från de beräknade världskoordinaterna. Resultatet multipliceras med de två basvektorerna för att få koordinaterna i ett koordinatsystem som har sitt origo i spelplanens mittpunkt.

## 7.2 Konsekvenser för berörd målgrupp

Målet med slutprodukten var att den skulle visas upp på Visualiseringsscenter C. Där skulle det kunna inspirera barn och ungdomar till att börja programmera och kanske vidare utbilda sig till ingenjörer. Att visa vad en grupp utvecklare med programmeringsvana kan skapa helt utan förkunskap inom områdena kan bevisa att det bara är fantasin som sätter gränser.

Till framläggningen för detta projekt har en video förberetts. Den kommer att publiceras på YouTube och möjligtvis på Sphero-relaterade forum för att inspirera andra till att utveckla nya applikationer och projekt till Sphero-robotar. Detta skulle påvisa möjligheterna som skapas genom att använda Sphero-API:et och att det då finns värde i att se till att deras API:er är aktuella.

## 7.3 Framtida arbete

I nuläget måste användaren själv kalibrera både spårningskamera och en ny bas till det gemensamma koordinatsystemet. Spårningskameran måste alltid kalibreras när den sätts upp på en ny plats men det skulle vara möjligt att låta roboten kalibrera den nya basen automatiskt i början av första styrningen.

Då resultatet inte blev ett spel som först var planerat finns det stort utrymme att utveckla vidare denna applikation. Det som just nu är färdigt är styrsystemet och kommunikationen för roboten. Detta kan appliceras på nästintill vilken spelidé som helst för en användare som baserar sig på AR. Om man vill fortsätta utveckla det arbete som gjorts under projekttiden för att låta flera användare koppla upp sina robotar måste servern ändras för att hantera kommunikation för flera användare. Spårningen av robotarna måste också kunna hantera mer än två färger. För tillfället är det även hårdkodat vilken robot som är vilken. En funktionalitet som önskas är att användaren bör kunna ha en översikt över de robotar som servern kan ansluta till.

# Litteraturförteckning

- [1] Mike Schramm, *Hands-on with Sphero at the CES 2011*, Engadget, 2011-01-08, hämtad: 2017-05-10  
<https://www.engadget.com/2011/01/08/hands-on-with-sphero-at-the-ces-2011/>
- [2] PTC inc, *The World's Most Widely Deployed AR Platform* hämtad: 2017-05-09  
<https://www.vuforia.com/>
- [3] Alexandro Simonetti Ibañez and Josep Paredes Figueras, *Vuforia v1.5 SDK: Analysis and evaluation of capabilities*, 2013-03-19, hämtad: 2017-05-08  
<https://upcommons.upc.edu/bitstream/handle/2099.1/17769/memoria.pdf>
- [4] Romain Dillet, *Sphero's New Augmented Reality App Allows You To Walk A Beaver Around Your House*, Tech Crunch, 2012-11-16, hämtad: 2017-05-10  
<https://techcrunch.com/2012/11/16/spheros-new-augmented-reality-app-allows-you-to-walk-a-beaver-around-your-home/>
- [5] Ken Schwaber och Jeff Sutherland, *Scrumguiden*, hämtad: 2017-02-22  
<http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-SE.pdf>
- [6] Trello, Inc. *Trello*, hämtad: 2017-05-15  
<https://trello.com/>
- [7] GitHub, Inc. *GitHub*, hämtad: 2017-05-15  
<https://github.com/>
- [8] Sphero, *API Quick Reference* hämtad: 2017-05-26  
<https://sdk.sphero.com/api-reference/api-quick-reference/>
- [9] PTC inc, *Natural Features and Image Ratings*, hämtad: 2017-05-08  
<https://library.vuforia.com/articles/Solution/Natural-Features-and-Ratings>
- [10] Unity Technologies, *Physics.Raycast*, hämtad: 2017-05-08  
<https://docs.unity3d.com/ScriptReference/Physics.Raycast.html>
- [11] OpenCV, *cv::SimpleBlobDetector Class Reference*, hämtad: 2017-05-10  
[http://docs.opencv.org/trunk/d0/d7a/classcv\\_1\\_1SimpleBlobDetector.html](http://docs.opencv.org/trunk/d0/d7a/classcv_1_1SimpleBlobDetector.html)
- [12] Satya Mallick, *Blob Detection Using OpenCV ( Python, C++ )*, hämtad: 2017-05-10  
<https://www.learnopencv.com/blob-detection-using-opencv-python-c/>

- [13] Wikipedia, *HSL and HSV*, hämtad: 2017-05-09  
[https://en.wikipedia.org/wiki/HSL\\_and\\_HSV](https://en.wikipedia.org/wiki/HSL_and_HSV)
- [14] Kyle Hounslow, *AutoColorFilter*, hämtad: 2017-05-10  
<https://github.com/kylehounslow/opencv-tuts/tree/master/auto-colour-filter>
- [15] Wikipedia, *Mathematical morphology*, hämtad: 2017-05-09  
[https://en.wikipedia.org/wiki/Mathematical\\_morphology](https://en.wikipedia.org/wiki/Mathematical_morphology)
- [16] Wikipedia, *Camera Resectioning*, hämtad: 2017-05-10  
[https://en.wikipedia.org/wiki/Camera\\_resectioning](https://en.wikipedia.org/wiki/Camera_resectioning)
- [17] OpenCV, *Camera calibration with OpenCV*, hämtad: 2017-05-10  
[http://docs.opencv.org/3.1.0/d4/d94/tutorial\\_camera\\_calibration.html](http://docs.opencv.org/3.1.0/d4/d94/tutorial_camera_calibration.html)
- [18] Wikipedia, *Camera matrix*, hämtad: 2017-05-10  
[https://en.wikipedia.org/wiki/Camera\\_matrix](https://en.wikipedia.org/wiki/Camera_matrix)
- [19] Wikipedia, *Quaternions and spatial rotation*, hämtad: 2017-05-10  
[https://en.wikipedia.org/wiki/Quaternions\\_and\\_spatial\\_rotation](https://en.wikipedia.org/wiki/Quaternions_and_spatial_rotation)
- [20] KJ Lundin Palmerius, *KJs Algebra 3D Package*, hämtad: 2017-05-21  
<http://weber.itn.liu.se/~karlu20/div/A3D/>
- [21] Wikipedia, *Line-plane Intersection*, hämtad: 2017-05-10  
[https://en.wikipedia.org/wiki/Line-plane\\_intersection](https://en.wikipedia.org/wiki/Line-plane_intersection)
- [22] Margaret Rouse, *client/server* hämtad: 2017-05-09  
<http://searchnetworking.techtarget.com/definition/client-server>
- [23] Microsoft, *.NET Framework Guide* hämtad: 2017-05-09  
<https://docs.microsoft.com/en-us/dotnet/articles/framework/>
- [24] Brian Hall, *Beej's Guide to Network Programming* hämtad: 2017-05-09  
[http://beej.us/guide/bgnet/output/print/bgnet\\_A4\\_2.pdf](http://beej.us/guide/bgnet/output/print/bgnet_A4_2.pdf)
- [25] PTC inc, *Sphero* hämtad: 2017-05-09  
<http://www.sphero.com/sphero>
- [26] PTC inc, *Sphero Reference API* hämtad: 2017-05-09  
<https://sdk.sphero.com/api-reference/api-quick-reference/>
- [27] Wikipedia, *Sphero specifications* hämtad: 2017-05-30  
<https://en.wikipedia.org/wiki/Sphero>
- [28] Think Geek, *Image of sphero and charger* hämtad: 2017-05-31  
<http://www.thinkgeek.com/product/ed94/>
- [29] PTC inc, *SpheroRAW Low Level Sphero C++ API* hämtad: 2017-05-09  
<https://github.com/PaulFreund/SpheroRAW>
- [30] PTC inc, *Low level API documentation* hämtad: 2017-05-10  
[https://github.com/orbotix/DeveloperResources/raw/master/docs/Sphero\\_API\\_1.50.pdf](https://github.com/orbotix/DeveloperResources/raw/master/docs/Sphero_API_1.50.pdf)

- [31] Satya Mallick, *Learn OpenCV*, hämtad: 2017-05-10  
<https://www.learnopencv.com/>
- [32] Stack Exchange, Inc, *Stackoverflow*, hämtad: 2017-05-11  
<http://stackoverflow.com/company/about>

# Bilaga A

## Involverade parter

Tabell A.1: Organisation och ansvarsområden

Namn	Ansvarsområde	Individuellt arbete
Jakob Andersson	Testansvarig	Utveckling av server samt kommunikation mellan server och surfplatta.
Jakob Bertlin	Produktägare	Ansvarar för kommunikation med Sphero, utformingen av planeringsstyrningen och intregration av de färdiga modulerna (styrning, kamera, nätverk) till ett sluttgiltigt program på serversidan.
Annie Lantz	Kontaktansvarig	Planeringsstyrningen samt rapportskelett. Ansvarat för och genomfört användartester.
Yrsa Lifvergren	Dokumentationsansvarig	Utveckling och testning av spårning via webbkamera, Blobtracking, färgrymd och kalibrering av ny bas.
Anton Pålsson	Kodansvarig	Ansvarar för kommunikation med server samt klient till surfplattan.
Anton Sterner	Scrummästare	Utveckling och testning av spårning via webbkamera, kamerakalibrering och koordinattransformationer.
Johan Sundgren	Materialansvarig	Ansvar för implementationen av AR samt positionsangivning för robot.

Handledare och kunder har varit Ali Samini och Karljohan Lundin Palmerius.

# Bilaga B

## Användartester

Testpersonerna är anonyma i sina svar. Det enda som angavs var ålder för att gruppen skulle få en uppfattning om spridningen på användarna.

### B.1 Frågor

Nedan listas de frågor som ställdes till testpersonerna.

1. Har du någon tidigare erfarenhet av planerad styrning?
2. Hur upplevde du den planerade styrningen?
3. Var något oklart gällande styrningen?
4. Vad tyckte du om återkopplingen som gavs i gränssnittet i form av prickar på skärmen?
5. Hur tydligt var det att kommandon skickas automatiskt efter 2 sekunder? Hade du hellre sett att användaren själv fick skicka iväg en rad kommandon manuellt?
6. Hur tydligt var det att flera punkter kunde tryckas ut i rad?
7. Var det lätt att förstå hur bytet mellan vilken robot som skulle röra sig skulle ske?
8. Vad tyckte du om återkopplingen, som gavs i gränssnittet, som visade vilken robot som var vald?
9. Hur användarvänligt upplevde du gränssnittet?
10. Var det något du saknade i gränssnittet?

### B.2 Användarnas svar

Nedan listas några av testpersonernas svar som ansågs konstruktiva för gruppen. Därmed så är inte samtliga svar presenterade eller alla testpersoner representerade. Användarna var i åldern 21 till 27 och åtta av tio hade inga tidigare erfarenheter av planerad styrning.

#### Svar på fråga 2:

- Något långsam.
- Ganska exakt.
- Ganska smidig och pricksäker.
- Det var lite svårt att markera spheros när de befann sig långt ifrån centrum av banan. Precisionen av styrningen var helt ok, det hände att spherosen inte riktigt hamnade där man tänkte sig.

- Den fungerade bra, sfären rullade väldigt likt den utsatta banan.
- Så länge man höll koll på att sätta ut punkter snabbt nog fungerade det bra.

**Svar på fråga 3:**

- Att man inte kunde avmarkera vilken robot man ville styra efter att man hade valt den. Man var tvungen att flytta den minst en gång.
- Nej, det var intuitivt hur styrningen gick till.
- Det tog ett tag tills jag förstod att jag var tvungen att välja sfär för varje ny styrning.
- Jag var lite snabb att välja den andra Spheron innan den andra hade åkt iväg. Men å andra sidan det stod detta klart och tydligt i instruktionerna. Jag var lite för ivrig.

**Svar på fråga 4:**

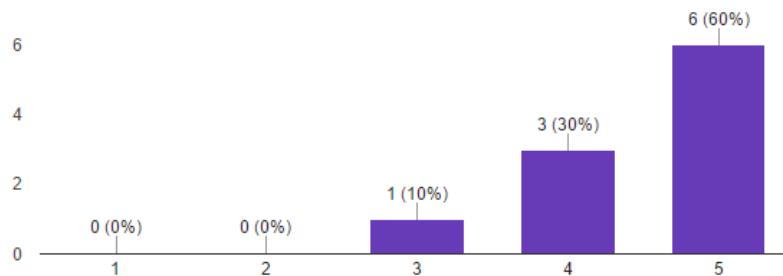
- Det var bra, hade dock sett att det fanns en visuell återkopplings på vart roboten åkte medans den flyttade på sig. Tänk typ dragna linjer mellan de punkterna som användaren sätter ut.
- Prickarna syntes tydligt vilket var bra, men det var lite oklart när spheron inte ville åka då pricken var för nära kanten av banan. Då provade jag trycka några gånger igen innan jag förstod att den kunde bete sig konstigt om den skulle för nära kanten. Något i gränssnittet som varnade eller förtydligade detta skulle vara bra.
- Prickarna visade tydligt hur rutten skulle bli, men det hade nog varit ännu tydligare med exempelvis en linje mellan prickarna. Tror även att det sattes ut punkter även om ingen sfär var vald vilket kan bli lite förvirrande då det inte resulterar i något.

**Svar på fråga 5:**

- Jag hade hellre sett att roboten behandlade varje punkt allt eftersom användaren tryckte. Dvs att roboten inte behöver vänta på att användaren har matat in alla punkter den ska åka till. Alltså typ en dynamisk kö (först in, först ut) med punkter som roboten åker till.
- Efter instruktionerna så var det tydligt, hade det inte förklarats där hade det inte varit klart. Det skulle kunna vara en bra valbar funktion, om användaren vill stryka hel manuellt eller automatiskt typ. Tror att manuell styrning av kommandon skulle kunna uppskattas i vissa lägen tror jag.
- I med att det var ganska mycket delay, hade det nog varit bättre om användaren själv kunde skicka iväg kommandon manuellt kanske bäst om man hade fått välja hela banan först och sedan toucha för att starta direkt.
- Feedback på hur länge det är kvar innan kommandot skickas hade uppskattats.
- Det fungerade bra, men hade varit tydligare med en mätare som tickade ned två sekunder tills kommandon skulle skickas iväg.
- Ja, möjligtvis. Det hade varit lättare att i lugn och ro sätta ut vägen och sedan skicka kommandot, även om jag gillar att den rör sig automatiskt. Båda funkar nog!
- Tror det är lättare om användare får välja när alla kommandon ska köras.

- När det väl framgått (en extra påminnelse från Annie) så var det tydligt och inga problem att vänta två sekunder.

**Svar på fråga 6:** Användaren gavs här möjligheten att på en femgradig skala (1 = inte så klart och 5 = superklart) välja det alternativ som passade hens upplevelse bäst. Resultatet presenteras nedan i figur B.1



Figur B.1: Fördelningen på svaren i fråga 6.

#### Svar på fråga 7:

- Ja, förutom att jag tänkte att jag kunde välja att byta robot istället för att skapa en plan för den valda.
- Ja, men det var lite oklart exakt när man kunde börja styra den andra robotten efter man styrt den första, men efter några styrningar förstod man.
- Ja, dock kunde inte en sfär väljas förrän ett aktivt kommando skickats iväg. Det skulle va tänkbart att om användaren har en sfär vald och då klickar på en annan sfär så byts den aktiva sfären till den istället för att lägga ut rutt.

#### Svar på fråga 8:

- Ganska bra, jag hade dock hellre sett att det ritades ut en cirkel runt robotten för att indikera vald robot. Tänk att cirkeln ligger parallellt med golvplanet.
- Jag förstod enkelt vilken robot jag valt.
- Det var tydligt vilken robot som var vald.
- Inte helt tydligt, denna markering kunde vara i samma färg som vald sphero.
- Väldigt tydlig, dock lite oklart med att belysningen i unityscenen fick markeringarna att se olika ut beroende på position.
- Bra, kanske man skulle återskapa två sfärer istället för två streck.

#### Svar på fråga 9:

- Första menyn, "connect skärmensamt instruktionsskärmen var bra.
- Det var bra, tydligt med enkla val. Inga onödiga saker som störde.

- Tydligt om man läst instruktionerna, annars hade man nog försökt trycka iväg båda sfärerna samtidigt.

**Svar på fråga 10:**

- Ordning för punkterna i rutten.
- Nedräknare för de två sekunderna.
- Lite starkare färger på punkterna och i samma färger som sfärerna.