

Metiri

En kamerabaserad linjal

Pontus Arnesson
Johan Forslund
Vera Fristedt Andersson
Fredrik Johansson
Fredrik Norrstig

Examinator: Daniel Jönsson

Sammanfattning

Den här projektrapporten beskriver utvecklingen av Android-applikationen *Metiri* som är ett arbete i kursen *TNM094, Medietekniskt kandidatarbete*. Metiri är det latinska ordet för ”mätt” och beskriver produktens syfte. Applikationen är ett mobilt mätverktyg som använder förstärkt verklighet för att mäta plana ytor i realtid. Det finns även möjlighet att mäta plana ytor i lagrade miljöer.

Metiri har utvecklats med hjälp av plattformen *ARCore*, 3D-API:et *OpenGL ES* och databasbiblioteket *Room*. Arbetsgruppen har under hela projektets gång arbetat agilt enligt utvecklingsmetodiken *Scrum*. Rapporten beskriver arbetsprocessen ingående och även de problem som arbetsgruppen har stött på. Metiri har under hela projektets gång utvecklats med användaren i fokus.

Innehåll

Sammanfattning	i
Figurer	v
Tabeller	vi
1 Inledning	1
1.1 Bakgrund	1
1.1.1 Kundens krav	1
1.2 Syfte	1
1.3 Frågeställningar	2
1.4 Avgränsningar	2
2 Relaterat arbete	3
2.1 Testning av iOS Mätverktyg	3
2.1.1 Fördelar med applikationen	3
2.1.2 Nackdelar med applikationen	4
2.2 Testning av AR Ruler App	5
2.2.1 Fördelar med applikationen	5
2.2.2 Nackdelar med applikationen	6
3 Teknik och Teori	7
3.1 Programutveckling för Android	7
3.2 3D-grafik för Android	7
3.3 ARCore som API	8
3.3.1 Virtuella objekt	8
3.3.2 Tillstånd	8
3.4 Tekniken bakom ARCore	9
4 Utvecklingsprocessen	10
4.1 Agil utveckling med Scrum	10
4.2 Tidsplan	11

4.2.1	Sprintplanering	11
4.2.2	Milstolpar och leverabler	12
4.2.3	Avstämningssmöten	12
4.3	Kravhantering	12
4.4	Versionshantering	12
4.5	Kvalitetssäkring och testningsprinciper	13
4.6	Mötesprinciper och rutiner	14
4.7	Dokumentation	14
4.8	Ansvarsfördelning	14
5	Teknisk beskrivning	16
5.1	Målplattform	16
5.2	Utvecklingsmiljö	16
5.3	Systembegränsningar	17
5.4	System-arkitektur och programdesign	17
5.5	OpenGL / Sceneform	18
6	Redogörelse för arbetet	19
6.1	Förarbete	19
6.2	Grafiskt gränssnitt	19
6.2.1	Funktionalitet	19
6.2.2	Implementation	21
6.3	Mätningsfunktionalitet	22
6.3.1	Lista med mätpunkter	22
6.3.2	Träffkontroll	22
6.3.3	Avståndsberäkning	22
6.4	3D-grafik	23
6.4.1	Kameratransformation utan rotation	23
6.4.2	3D-objekt i applikationen	24
6.4.3	Borttoning	27
6.4.4	Snap	27
6.5	Spara bilder till lagringsutrymme	28
6.6	Användartester	28
6.7	Jämförelser av mätprecision	29
6.8	Spara rum	29
6.8.1	Spara till databas	30
6.8.2	Använda sparad data	31
6.9	Hjälpmmedel	34

6.9.1	Onboarding	34
6.9.2	Hjälpillustration för att detektera ytor	34
6.9.3	Hjälp- och Inställningssida	34
6.9.4	Felmeddelanden	35
6.9.5	Aktivitetsindikator	37
7	Fallgropar	38
7.1	Spara till kamerarulle	38
7.2	Spara ner miljö	38
7.3	Använda samma lista i två trådar	39
7.4	Hjälpillustration för att detektera ytor	39
8	Resultat	40
8.1	Mätning i realtid	40
8.2	Mätning i lagrad miljö	40
8.3	Användartester	40
8.4	Jämförelse av mätprecision	41
9	Analys och diskussion	44
9.1	Grafiskt gränssnitt	44
9.2	Mätningsfunktionalitet	44
9.3	3D-grafik	45
9.4	Spara bilder till lagringsutrymme	45
9.5	Användartester	45
9.6	Jämförelse av mätprecision	45
9.7	Databashantering	46
9.8	Hjälpmittel	46
9.8.1	Onboarding	46
9.8.2	GIF	46
9.8.3	Hjälp- och inställningssida	47
9.9	Resultat	47
9.10	Arbetet i ett vidare sammanhang	47
10	Slutsatser	48
Litteraturförteckning		50
A	Användartester	52
B	Individuella bidrag	59

Figurer

2.1	iOS mobilapplikation Mätverktyg	4
2.2	Mobilapplikationen AR Ruler App	5
3.1	Diagram över SLAM-processen	9
4.1	Arbetsprocessen för sprintcyklarna	10
4.2	Gantt-schema som visar övergripande tidsplan	11
4.3	Översikt över hur Trello kan se ut under en sprint	13
4.4	Versionshanteringsflöde med två grenar	13
6.1	Ursprunglig skiss av applikationens gränssnitt	20
6.2	Mobilapplikationens grundläggande gränssnitt	21
6.3	Hårkors som visar planets rotation.	25
6.4	Cirkel som består av 16 likformiga trianglar.	25
6.5	Logiken bakom uppriötningen av linjernas bredd.	26
6.6	Vad användaren ser när applikationen frågar om tillåtelse för att lagra filer på telefonen.	28
6.7	En sparad bild redo för mätningar. De små rektanglarna visualiseras hittade plan där punkter kan placeras ut.	31
6.8	Aktiviteten där användaren väljer vilket rum som ska användas för mätning.	32
6.9	Förstorningsglas syns längst upp till höger i bild. Fingret (som inte syns) är placerat vid termosken.	33
6.10	Mobilapplikationens onboarding	35
6.11	GIF:en som instruerar hur användaren ska röra enheten.	36
6.12	Applikationens hjälpsida	36
6.13	Indikatorn som visas när en bild sparas till kamerarullen.	37
8.1	Mätning i realtid	41
8.2	Mätning i sparad miljö	42
8.3	Mätning av ett LiU-kort	43

Tabeller

4.1 Förlaring av de olika sprintarna	11
5.1 De mobiltelfoner som används under utvecklingen	17
6.1 Databastabell över sparade rum	30
6.2 Databastabell över den sparade miljön	30
8.1 Jämförelse av mätprecision för olika applikationer på ett objekt med bredden 31,4 cm	42

Kapitel 1

Inledning

Detta kapitel introducerar projektet och beskriver dess bakgrund, syfte, frågeställningar samt avgränsningar.

1.1 Bakgrund

Människor förlitar sig mer och mer på sina telefoner. Därför finns det flera verktyg som en telefon fördelaktigt kan ersätta med hjälp av applikationer. Ett av dessa verktyg är måttbandet. Via en mobilkamera bör det finnas möjlighet att kunna mäta avstånd med hjälp av dagens teknologi. Applikationer som gör just detta finns redan, men de är inte tillräckligt precisa i sina mått eller tillräckligt lätt att använda för att dra ordentlig nytta av dem. I detta projekt undersöks hur *Augmented Reality* (förstärkt verklighet på svenska, vidare förkortat AR) kan användas för att möjliggöra mätningar i en mobilapplikation. *Android*-plattformen *ARCore* eller *iOS*-plattformen *ARKit* möjliggör användning av AR i mobilapplikationer.

1.1.1 Kundens krav

Kundens grundläggande krav på projektet är att skapa en applikation för mätningar i omvärlden. Denna applikation ska köras på mobila enheter och använda sig av AR. Utöver dessa krav ska applikationen vara en förbättring av liknande applikationer på något sätt. Detta innebär till exempel att precisionen kan förbättras eller att fler användbara funktioner kan erbjudas än vad konkurrenterna har. Något som kunden saknar i alla existerande mätverktygs-applikationer är möjligheten att mäta i en sparad miljö.

1.2 Syfte

Syftet med projektet är att undersöka hur mätfunktionalitet i en mobilapplikation kan skapas med hjälp av *ARCore*. Liknande system analyseras för att identifiera dess svagheter och för att veta vad som bör förbättras i detta projekt. Resultatet kommer att testas och jämföras med redan etablerade applikationer. Det kommer även undersökas hur data för en omgivning kan sparas till en databas för mätning vid senare tillfälle.

1.3 Frågeställningar

Rapporten ska besvara följande frågeställningar:

- Hur kan den data som ARCore identifierar lagras för att göra mätningar vid ett senare tillfälle?

När ARCore analyserar omgivningen identifieras plana ytor och distinkta punkter i rummet. I denna frågeställning undersöks hur denna data – tillsammans med en kamerabild – kan lagras, och användas vid ett senare tillfälle för att göra mätningar.

- Hur kan ett gränssnitt utformas så att användaren upplever gränssnittet som intuitivt i en AR-applikation där det är nödvändigt att placera ut objekt?

Många människor är ovana med att använda AR-applikationer vilket gör det viktigt att applikationen är tydlig att förstå. I denna applikation ska användaren placera ut mätpunkter i omvälvden. För att detta ska vara intuitivt undersöks hur gränssnittet ska utformas på bästa sätt för att förenkla användandet.

- Vilket 3D-API för mobiltelefoner är bäst utformat för att rita grundläggande geometrier i både AR och icke-AR?

Denna applikation har två olika lägen, i det ena läget mäter användaren miljön i realtid och i det andra läget kan miljön mätas vid ett senare tillfälle. Det första läget använder AR-teknik medan det andra läget inte gör det. Därför eftersöks det bäst utformade API:et för att kunna rita grundläggande geometrier i båda dessa lägen.

1.4 Avgränsningar

Mobilapplikationen kommer att utvecklas för Android-enheter som stödjer ARCore. ARCore saknar objektkänning och på grund av detta kommer det endast vara möjligt att mäta plana ytor i mobilapplikationen.

Produktens användare antas ha tidigare erfarenhet kring användning av smartphones samt användning av mobila applikationer.

Det förväntas att rapportens läsare har liknande förkunskaper som en tredjeårs civilingenjörsstudent inom medieteknik. Det inkluderar grundläggande kunskaper inom OpenGL, databashantering och multitrådning.

Kapitel 2

Relaterat arbete

Det finns ett antal olika mätapplikationer på marknaden. Sedan uppkomsten av ARCore och ARKit har de ökat i mängd eftersom dessa plattformar ger relativt enkla metoder för att implementera mätfunktionalitet. Inför projektet eftersöktes applikationer som kan mäta i lagrade miljöer, vilket var ett utav kundens önskemål. Gruppen kunde inte finna någon applikation som klarar detta. Däremot jämfördes andra applikationer som erbjuder mätfunktionalitet. Gruppen valde att jämföra en mätapplikation som används på iOS-enheter och en som används på Android-enheter. Det relaterade arbetet som studerades beskrivs i detta kapitel.

2.1 Testning av iOS Mätverktyg

Apple har utvecklat en mobilapplikation vid namn Mätverktyg som ingår i alla Apple-enheter med operativsystemet iOS 12.0 eller högre [1]. Figur 2.1 visar hur applikationen ser ut vid användning.

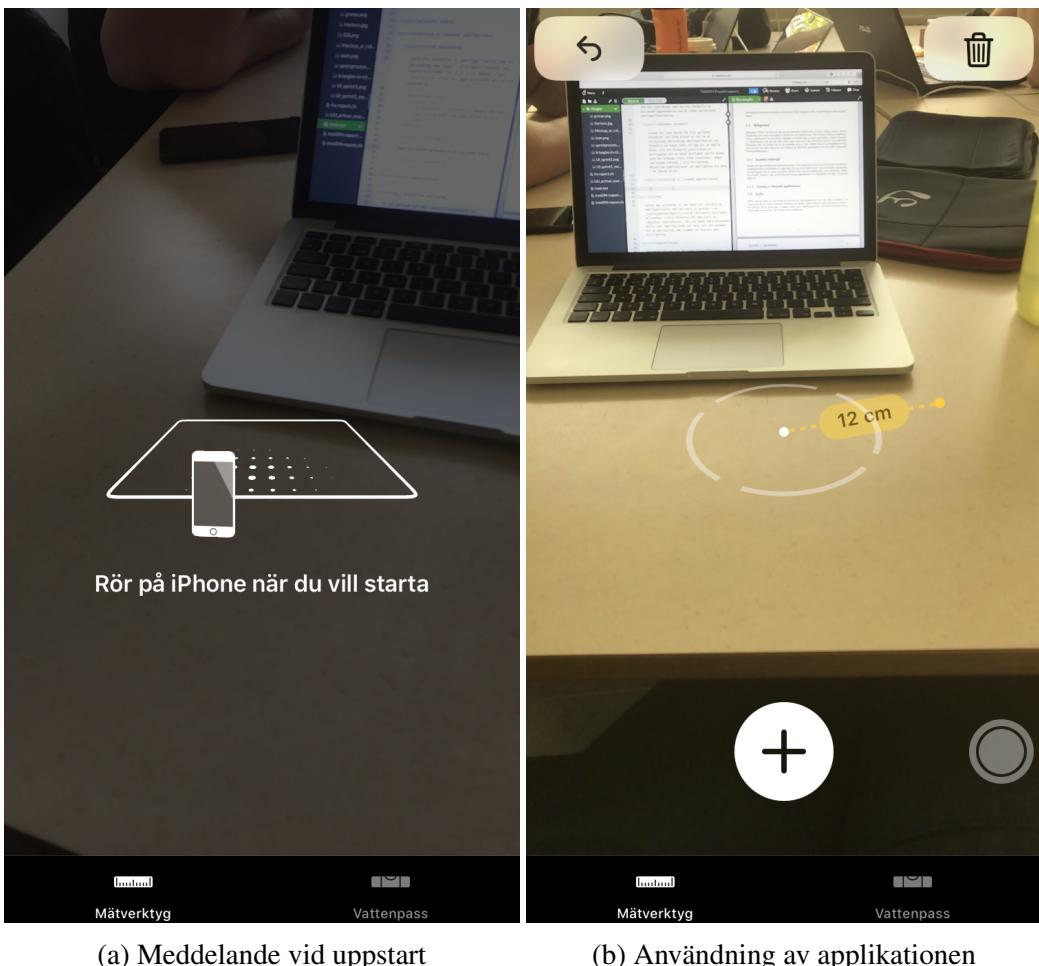
2.1.1 Fördelar med applikationen

Applikationens gränssnitt är stilrent och det framgår tydligt att användarens fokus ska riktas mot kameran. Applikationen hjälper användaren att komma igång vid start med hjälp av en GIF (*Graphics Interchange Format*) som visar en mobil som rör sig fram och tillbaka med mobilkameran riktad mot en yta, se Figur 2.1a. Med hjälp av ikonerna vid toppen av skärmen är det möjligt att förstå att det går att ångra sitt senaste steg samt att det är möjligt att slänga något. Största knappen är knappen med ett additions-tecken i mitten vilket tydligt gör att den knappen är viktig.

Applikationen ger återkoppling vid användning genom att visa felmeddelanden då den inte lyckas hitta ytor att mäta. Ytterligare återkoppling ges då ytor inte hittas genom att göra additions-knappen mer genomskinlig och grå vilket tyder på att den inte är klickbar och den streckade cirkeln i mitten försvisser. Återkoppling sker dessutom när ett plan hittas på så sätt att den streckade cirkeln i mitten roteras sådan att den ligger längs med det hittade planets yta.

När det redan finns utsatta punkter i miljön och punkten i mitten av skärmen kommer nära en av dessa redan utsatta punkter, förflyttas mittpunkten till den utsatta punkten och telefonen vibrerar vilket ger haptisk återkoppling. Denna förflyttning till en närliggande punkt kallas *snapping*.

Det är möjligt att mäta areor i Mätverktyg. Genom att dra fyra sammakopplade streck, kan applikationen beräkna arean hos ytan. Applikationen kan också detektera hörn och mittpunkter hos objekt som den kan snappa till.



Figur 2.1: iOS mobilapplikation Mätverktyg

2.1.2 Nackdelar med applikationen

Om man inte har läst något om Mätverktygs-applikationen innan, kan det vara svårt att förstå hur den fungerar första gången man använder den. Det finns ingen hjälp-sida eller liknande som kan hjälpa användaren att bekanta sig med applikationen innan användning.

Knappen som är placerad till höger om additionsknappen kan vara svår att förstå. Den sparar ner kameravyn med utsatta mätningar till mobilens kamerarulle. Knappen har samma utseende som en Apple-enhets foto-knapp har vid bildtagning, men det är inte ett allmänt igenkänt ikonutseende bland teknikanvändare.

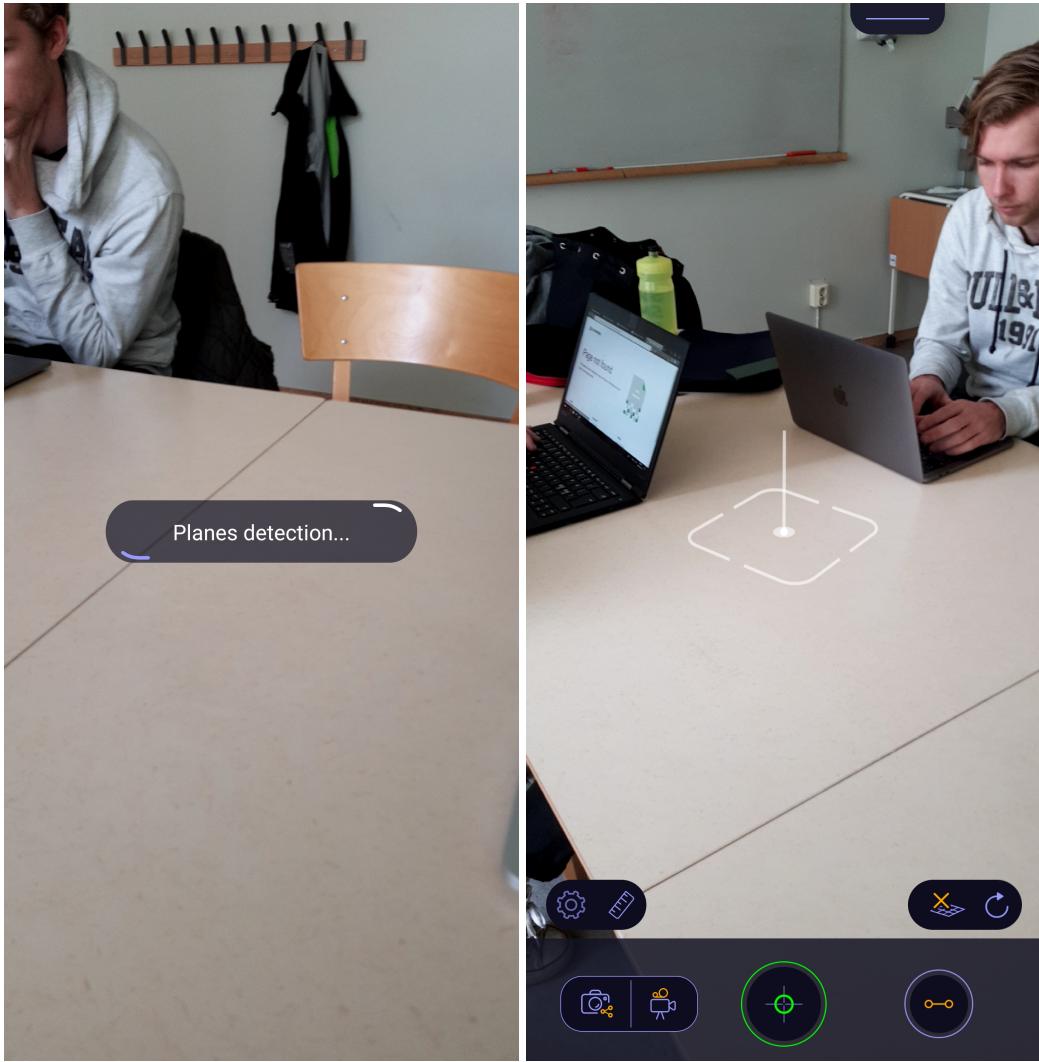
Det är inte möjligt att göra mätningar på olika platser i miljön. Alla mätningar måste vara sammankopplade, annars försvinner de gamla så fort en ny mätning görs.

Mätningarna saknar millimeterprecision vilket gör att mätning av små objekt får stora fel. Då man exempelvis mäter ett 0.5 cm långt objekt får man resultatet 1 cm, alltså en felsmärginal på 100% i detta fall.

Knappen som gör det möjligt att ångra utplacering av en punkt är en knapp som alla i gruppen upplever vara en väl använd knapp. Som användare är det skönt att ha knappar som ofta används i botten av skärmen för att underlätta för tummen att nå dem vid användning.

2.2 Testning av AR Ruler App

Mobilapplikationen AR Ruler App finns för både Android- och iOS-telefoner och är en av de mätverktygsapplikationer i *Google Play Store* som har flest nerladdningar [2]. Applikationens utseende är synligt i Figur 2.2.



(a) Meddelande vid plandetektion

(b) Användning av applikationen

Figur 2.2: Mobilapplikationen AR Ruler App

2.2.1 Fördelar med applikationen

Vid applikationens start kommer det alltid upp en videogenomgång som visar vilka möjligheter som finns med applikationen. Det är möjligt att hoppa över denna introduktionsvideo om man inte är intresserad.

Det finns möjlighet att mäta avstånd, areor, vinklar och volymer hos flera olika typer av geometrier. Cirklar, linjer, cylindrar och rektanglar är några av de geometrier som är möjliga att rita ut.

I mitten av skärmen finns en kvadrat med rundade hörn som lutar längs med planet som har hittats. Detta ger användaren möjlighet att se att planet som applikationen har identifierat faktiskt är korrekt plan.

2.2.2 Nackdelar med applikationen

Introduktionsvideon visar vad som är möjligt med applikationen men inte hur man använder själva applikationen. Det finns dessutom ingen hjälp-sida att hitta vid användning av applikationen om man inte förstår hur den fungerar.

Applikationen har inte ett intuitivt gränssnitt eftersom det används ikoner som inte är allmänt igenkända, se Figur 2.2b. Strecket och punkten i mitten av skärmen markerar ut den plats där en ny mätpunkt kommer sättas ut om man väljer att göra det. Denna figur ser ut som en pil med en sugpropp och är inte heller särskilt intuitiv vad gäller dess betydelse. Gruppen upplever dessutom applikationens utseende som rörigt då det finns för många knappar placerade på en liten yta.

Det är inte möjligt att göra någonting när applikationen letar efter plan och det framgår inte hur man ska göra för att hitta ett plan, se Figur 2.2a. Man måste aktivt välja ett särskilt plan för att kunna mäta på det och det är inte möjligt att ha mätningar på olika plan samtidigt.

Den större knappen på skärmen uppfattas som den viktiga knappen eftersom den sticker ut från de andra. Däremot har denna knapp olika betydelse beroende på vilken funktion man är inne på. Om man trycker på inställningsknappen blir den stora knappen till en ”stäng sida”-knapp vilket kan förvirra användaren gällande dess funktionalitet.

Kapitel 3

Teknik och Teori

Detta kapitel beskriver grundläggande och väsentlig teori som används för att skapa applikationen. Eftersom applikationen är skapad för Android beskrivs här grundläggande Android-utveckling och hur 3D-grafik kan användas för detta operativsystem. Dessutom redogörs för hur ARCore fungerar och är uppbyggt. Detta är viktig information för att förstå hur applikationen fungerar och varför vissa val har gjorts i utvecklingen. Här redovisas inga motiveringar.

3.1 Programutveckling för Android

Programutveckling för Android är en process som omfattar utveckling av applikationer till Android-enheter. Utvecklingen skrivs vanligtvis i *Java* och *XML*. Java är ett objektorienterat programmerings-språk och XML är ett märkspråk. Generellt sätt skrivs applikationens logik i Java medan designen skrivs i XML.

Android-applikationer består av aktiviteter där varje aktivitet är en unik sida i applikationen. Det grafiska gränssnittet i dessa sidor byggs upp av XML-taggar som kan ses som applikationens byggstenar, där varje tagg representerar en Android-komponent. De mest basala komponenterna är *Layout* och *View*. Layout fungerar som osynliga fält som ger sidan struktur. Innanför dessa fält används bland annat View-komponenter som en bas för andra komponenter. Varje View ansvarar för uppritning av dess innehåll samt hantering av händelser. Exempelvis är View en basklass till *Button* som är en enkel knapp. Det är även möjligt att detektera när användaren trycker på en View-komponent med hjälp av funktionen *setOnClickListener()*. Denna funktion tar in en *callback*-funktion som körs när användaren trycker på View-komponenten.

3.2 3D-grafik för Android

All grafik i applikationen är skriven i *OpenGL ES*. Detta är en delmängd av *OpenGL* som är avsedd att användas för mobila enheter. Sedan version 2.0 av OpenGL ES är det möjligt att programmera *shaders* [3], vilket tillåter utvecklaren att nyttja kraften i mobiltelefonens grafikkort. Detta har gjort det möjligt att skapa kraftfullare 3D-applikationer än vad som tidigare var möjligt.

För att integrera OpenGL ES i Android används klassen *GLSurfaceView*, vilket är en speciell typ av View där 3D-grafik kan ritas. För att rendera grafiken och för att hantera livscykeln för OpenGL kompletteras denna klass med gränssnittet *Renderer*. Detta gränssnitt kräver att följande funktioner implementeras:

- *onSurfaceCreated()* – Körs när GLSurfaceView skapas. I denna funktion är det lämpligt att initialisera de grafiska objekt som senare ska ritas.
- *onDrawFrame()* – Körs varje gång systemet ritar om OpenGL-miljön. Fungerar som en *render-loop*.
- *onSurfaceChanged()* – Körs när geometrin hos GLSurfaceView förändras, exempelvis om användaren byter mobiltelefonens orientering från porträtt till landskap.

Dessa funktioner körs på en egen tråd i applikationen. Detta innebär att en Android-applikation som integrerar OpenGL ES alltid kommer att ha minst två trådar. En tråd som hanterar det grafiska gränsnittet (vidare benämnt UI-tråden) och en tråd som hanterar OpenGL-renderingen.

3.3 ARCore som API

2018 släppte *Google* sin AR-plattform ARCore. ARCore är uppbyggt av ett antal klasser som tillsammans används för att skapa en AR-applikation. En del av dessa klasser beskrivs här.

3.3.1 Virtuella objekt

För att kunna placera ut virtuella objekt i den verkliga världen används klassen *Anchor*. Denna klass beskriver en statisk position i den verkliga världen. Genom att låta 3D-objekt följa denna position upplevs objektet att vara placerat i den verkliga världen.

I ARCore är det enbart möjligt att placera objekt på plana ytor. När ARCore analyserar omgivningen letar plattformen efter just sådana ytor. De upptäckta ytorna representeras med klassen *Plane* i ARCore. Plattformen är begränsad till att enbart identifiera tvådimensionella ytor.

Vidare används klassen *Camera* för att ge information om mobiltelefonens kamera och dess position i omvälvärlden. Klassen är implementerad sådan att kamerans ursprungsposition beskriver omvälvärldens origo, vidare kallat världsorigo. Denna klass har funktioner för att hämta perspektiv- och kameramatris. Dessa matriser kan sedan användas i OpenGL-scenen för att rendera objekt på korrekt position med korrekt perspektiv.

För att representera transformationen för virtuella objekt i ARCore används klassen *Pose*, där varje objekts transformation beskrivs i världskoordinater. Detta inkluderar objektets translation och rotation. Alla de klasser som beskrivits ovan (*Anchor*, *Plane*, *Camera*) har vars en *Pose* i ARCore.

3.3.2 Tillstånd

För att hämta applikations tillstånd vid ett givet ögonblick används klassen *Frame*. Denna klass innehåller även en funktion för att detektera om det finns något *Plane* vid givna skärmkoordinater. Det är genom *Frame* som en instans av *Camera* skapas.

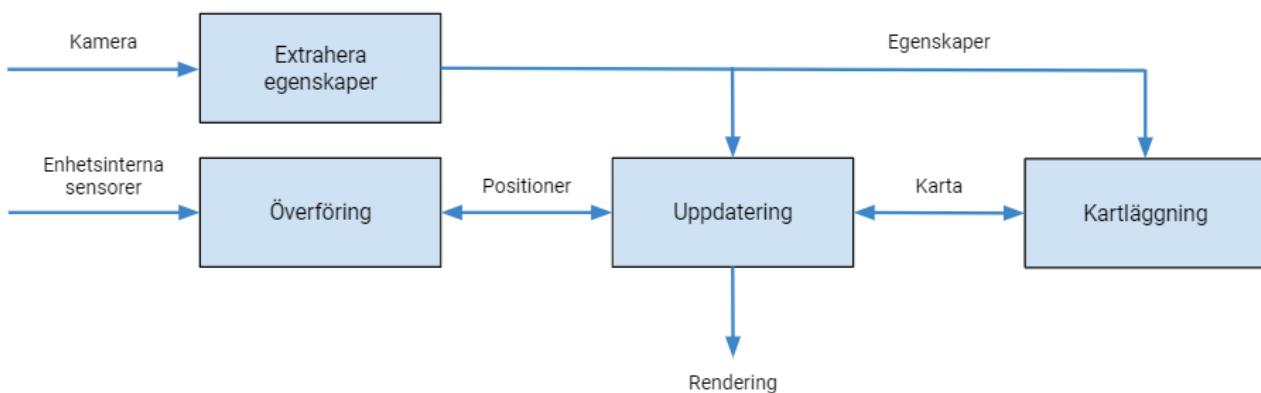
Längst upp i hierarkin för ARCore finns klassen *Session*. Denna klass hanterar all den data som är relaterad till ARCore. *Session* hanterar även livscykeln för ARCore och är huvudklassen för att kommunicera med ARCore's API. Alla instanser av *Frame* skapas genom denna klass. En annaniktig funktion i denna klass är att kommunicera till OpenGL så att GLSurfaceViews bakgrundstextur tilldelas av mobilkamerans bild.

3.4 Tekniken bakom ARCore

De viktigaste teknikerna som ARCore använder sig av är rörelsespårning, miljöförståelse och ljusuppskattning. Rörelsespårning gör att telefonen kan spåra och veta var den befinner sig relativt till den verkliga världen. Miljöförståelse gör att telefonen kan detektera plan, både dess storlek och position. Ljusuppskattning tillåter telefonen att dynamiskt uppskatta vilken ljusnivå omgivningen har.

För att systemet ska vara användbart är det viktigt att användaren kan röra på sig utan att AR-upplevelsen bryts. Detta möjliggörs med en avancerad teknik som förkortas *SLAM* [4], kort för *Simultaneous Localization and Mapping*. Med användning av denna teknik kan systemet lokalisera distinkta punkter relativt till omgivningen, samtidigt som systemet med hjälp av dessa punkter bygger upp en karta över den miljö som enheten befinner sig i. För att lyckas med detta krävs visuell data från kameran och sensordata från mobiltelefonens *IMU*. IMU är ett samlingsnamn för enhetens accelerometer, gyroskop och ibland magnetometer. Ett simplifierat diagram över SLAM-processen visas i Figur 3.1.

De distinkta punkter som systemet lokaliseringen brukar kallas för *feature points*. När dessa punkter kombineras med användarens rörelsedata kan systemet beräkna kamerans *pose*. Kamerans pose beskriver kamerans orientering samt position relativt till omgivningen. Med hjälp av denna pose kan systemet placera ut virtuella objekt i rummet som ser verklighetstrogn ut oberoende av användarens position i rummet.



Figur 3.1: Diagram över SLAM-processen

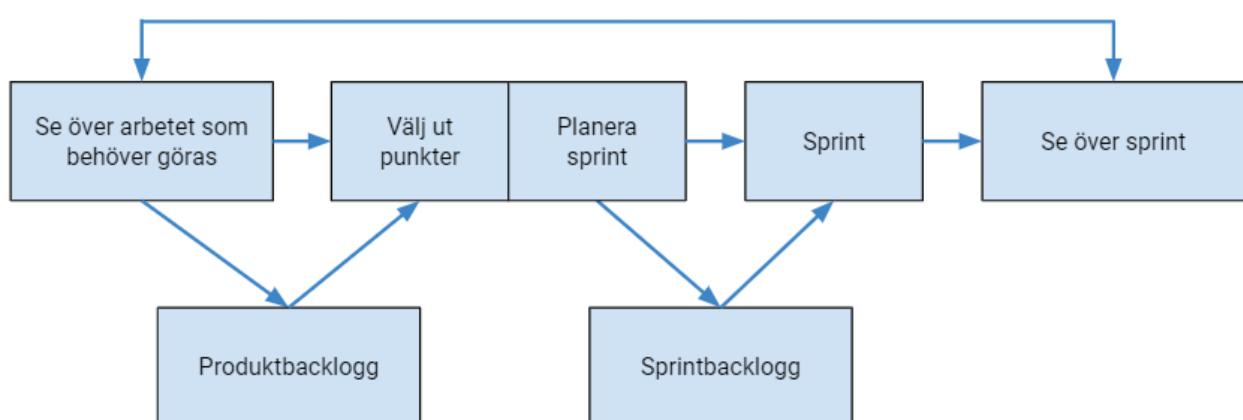
Kapitel 4

Utvecklingsprocessen

Här redovisas arbetslagets utvecklingsprocess. Utvecklingsprocessen innehåller de delar som styrt arbetslaget genom projektets gång. Detta är de delar som har fått projektet att fungera smidigt och som har gjort att gruppen har kunnat arbeta organiserat. Individuella bidrag inom gruppen kan ses i Bilaga B.

4.1 Agil utveckling med Scrum

En väl utbredd och agil utvecklingsmetodik är *Scrum* och det är den metodik som projektet har nyttjat. Anledningen till detta är att det arbetssätt som Scrum följer gör det möjligt att enkelt hantera eventuella ändringar och gör att alla i projektet är lika involverade. Det ger även gruppen en översikt över varandras arbete och hur projektet ligger till [5, 6]. Arbetslaget utgick från en produktbacklogg som innehöll uppgifter, fortsättningsvis benämnt poster, som behövde slutföras under utvecklingen av mobilapplikationen. Varje post skrevs kortfattad och specifik så att den kunde avklaras under en sprint. Produktbackloggen kunde exempelvis innehålla krav för applikationen samt definitioner av de funktioner som skulle implementeras. Arbetet delades upp i sprints som pågick under en till två veckor. Inför varje sprint valde gruppen ut ett antal poster från produktbackloggen som placerades i en sprintbacklogg. Sprintbackloggen innehöll de poster som skulle utföras under sprinten listade i prioriteringsordning. Se Figur 4.1 för sprintcyklernas arbetsprocess.



Figur 4.1: Arbetsprocessen för sprintcyklarna

4.2 Tidsplan

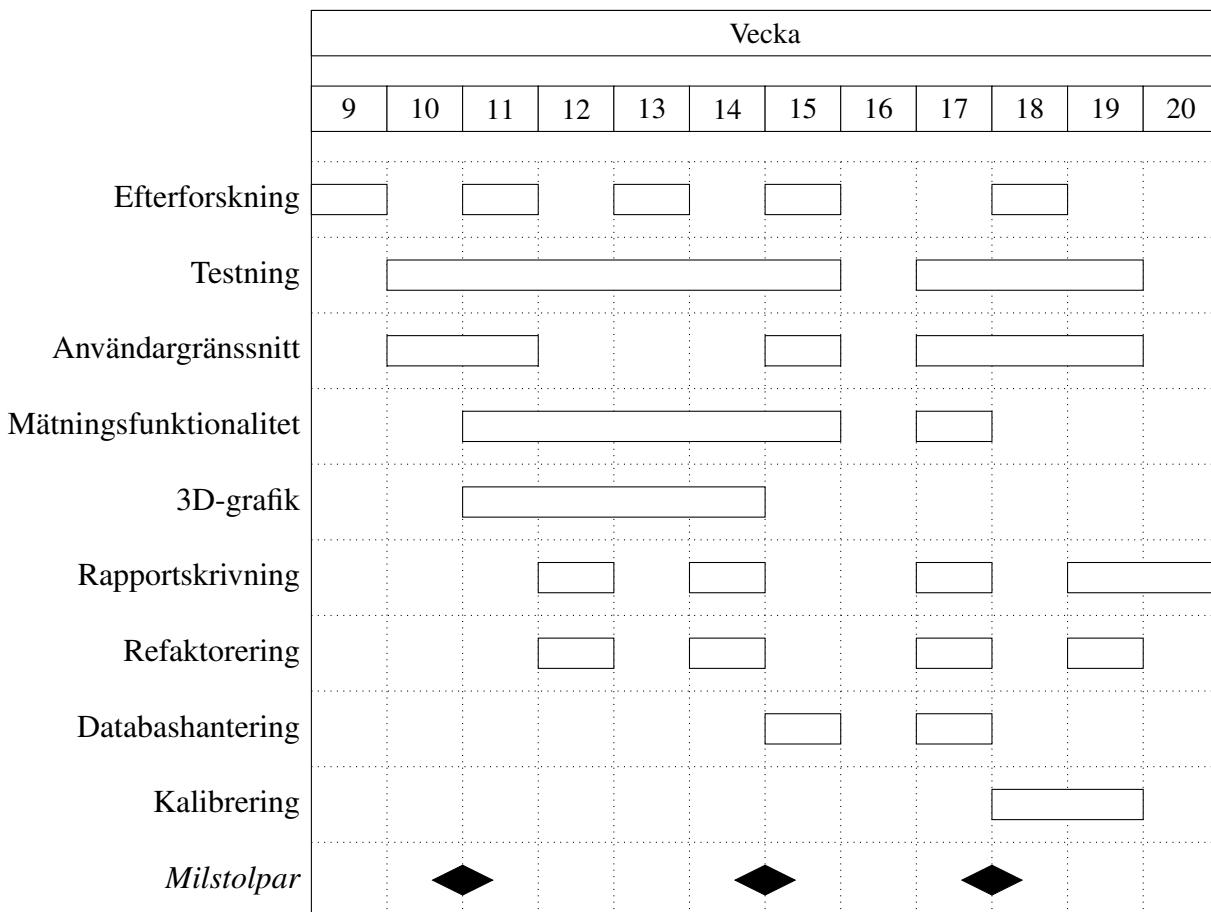
Här diskuteras hur projektets delmål planerades utefter utvecklingsmetodiken *Scrum* och hur tidsplanen följdes under arbetets gång.

4.2.1 Sprintplanering

Sprintarna planerades till att vara en till två veckor långa och ha en kortfattad beskrivning om vad som skulle implementeras. Den ursprungliga sprintplaneringen finns i Tabell 4.1 och medföljande tidsplan, representerad av ett *Gantt*-schemat, i Figur 4.2. Under vecka 16 var arbetslaget lediga på grund av tentamensförberedelser.

Tabell 4.1: Förlägning av de olika sprintarna

Sprint	Beskrivning	Vecka
0	Förstudie och uppsättning av utvecklingsmiljön	9
1	Grundläggande användargränssnitt	10
2	Grundläggande ARCore och utplacering av punkter	11-12
3	Beräkna avstånd mellan punkter	13-14
4	Databaslagring och mätning	15-17
5	Kalibrering av precision med hjälp av referensobjekt	18-19
6	Slutförande	20



Figur 4.2: Gantt-schema som visar övergripande tidsplan

Tidsplanen och sprintarnas innehåll agerade som en vägledning under utvecklingsprocessen och sprintarna planerades för att uppfylla milstolparna.

Under utvecklingen togs beslutet att utesluta kalibreringen av precision för att ägna mer tid åt databasimplementation. Detta beslut togs i samförstånd med kund. Denna ändring ledde till att sprint 4 förlängdes från 2 veckor till 4 veckor.

Tidsplanen innefattar inte tiden för underhåll av produkten som pågår efter att produkten har levererats.

4.2.2 Milstolpar och leverabler

- Milstolpe 1 - Applikationen ska kunna startas i en mobiltelefon med ett användargränssnitt som är utrustat med en kamera-vy och nödvändiga illustrationer för att en användare ska kunna använda applikationens framtida funktioner. Detta ska fungera som en första prototyp av applikationens utseende.
- Milstolpe 2 - Applikationen ska kunna mäta objekt i realtid. Detta innebär att funktionaliteten för att göra mätningar i realtid ska vara klar och integrerad med användargränssnittet. Det här målet är centralt för applikationen och dess framtida utveckling. Detta fungerar som en ny version av applikationen och den kan även börja användas av kunden.
- Milstolpe 3 - Applikationen ska kunna mäta objekt från lagrad data, detta innebär dels att applikationen måste kunna spara mätdata. Men även att applikationen ska kunna hämta denna data för mätning. Därför består detta mål av att implementera en databas som kommunicerar med applikationen. För att uppnå detta mål krävs också att arbetslaget löser problemet med att spara mätdata tillsammans med en bild. Även detta lanseras som en ny version som är klar för användning.

4.2.3 Avstämningsmöten

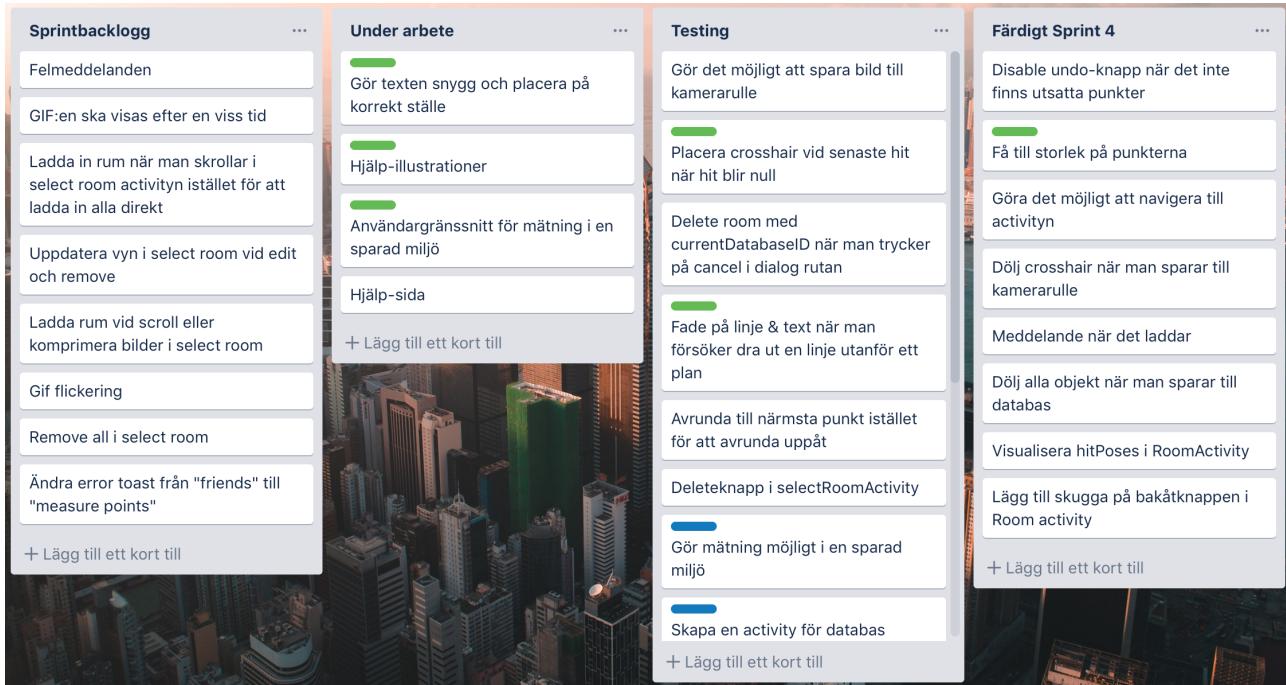
Avstämningsmöten med kunden (även gruppens mentor för projektet) skedde vecka 7, vecka 11 och vecka 16 för att uppdatera kunden om projektets status och för att få vägledning inför större strategiska beslut. En diskussion fördes om vad arbetslaget ville arbeta med och vad mentorn ansåg vara möjligt att göra och beslut togs därefter.

4.3 Kravhantering

För gruppens hantering av produkt- och sprintbackloggen användes tjänsten *Trello* [7]. Trello hjälpte till att organisera arbetet och gav en överblick för hur projektet låg till med hjälp av tavlor som bestod av listor som var fyllda med kort. Trello gjorde det möjligt att dra och släppa kort mellan listor och på så sätt visa utvecklingens framsteg. Vart och ett av korten representerade en post från produktbackloggen som placeras i följande stadier beroende på hur långt gruppen hade arbetat med posten: att göra, gör, testning och klart. Varje stadie representerades av en lista.

4.4 Versionshantering

Olika versioner av systemets kod hanterades med versionshanteringssystemet *Git* [8]. För att förenkla hanteringen mellan datorer användes även webbtjänsten *GitHub* för att göra det möjligt att lagra koden



Figur 4.3: Översikt över hur Trello kan se ut under en sprint

på internet [9].

Projektet delades upp i olika grenar, vardera gren innehöll kod för ett visst stadie av applikationen. Huvudgrenen innehöll den senaste fungerande versionen av applikationen. Dit skickades enbart kod som hade testats och blivit godkänd av minst två personer i gruppen. När ny funktionalitet skulle implementeras skapades en ny lokal gren som utgick från huvudgrenen. Detta gjordes för att undvika att icke-fungerande kod skickades till huvudgrenen. När den nya grenen skulle sammanfogas med huvudgrenen sparades grenen först på Github och sedan gjordes en förfrågan som granskades och godkändes av en annan person i gruppen. Ett exempel på detta flöde redovisas i Figur 4.4.



Figur 4.4: Versionshanteringsflöde med två grenar

4.5 Kvalitetssäkring och testningsprinciper

Gruppen har arbetat med kodgranskning för att kontrollera kod. Detta gjordes i samband med varje sammanfogning av grenar som beskrivs i sektion 4.4. Syftet med detta var att utvecklarna kunde hitta buggar i varandras kod och att utvecklarna fick större insikt i systemet som helhet. Under utvecklingsfasen har koden gemensamt refaktorerats i samband med varje avslutad sprint för att förhindra ostrukturerad kod och för att uppdatera varje gruppmedlem om hur koden fungerar.

4.6 Mötesprinciper och rutiner

Då gruppen har arbetat enligt Scrum-metoden har ett möte på maximalt 15 minuter skett i början av varje arbetsdag. På detta möte fick alla gruppmedlemmar kortfattat beskriva vad de har arbetat med och vad de ska fortsätta arbeta på. Efter att alla i gruppen haft ordet har det funnits möjlighet att ställa frågor och föra diskussioner om det som sagts.

En sprintplanering har skett inför varje ny sprint med en tidsbegränsning på fyra timmar per två veckors arbete, en tidsbegränsning som *Scrumguiden* rekommenderar [5]. Arbetslaget har samarbetat i sprintplaneringen och har gemensamt diskuterat fram en plan inför nästkommande sprint. Sprintplaneringen har bestämt vad som ska uppnås och hur det ska uppnås.

Efter varje avslutad sprint skedde en sprintgranskning som har varit tidsbegränsad till två timmar per två veckors arbete, ytterligare en tidsbegränsning som *Scrumguiden* rekommenderar. Utvecklarna har då diskuterat resultatet av sprinten. Efter det har produktbackloggen justerats för att ge ett värdefullt underlag inför planeringen av nästkommande sprint.

Mellan varje sprintgranskning och sprintplanering har det skett en sprintåterblick som har varit tidsbegränsad till en timme per två veckors arbete, även här per rekommendation av *Scrumguiden*. Syftet med sprintåterblicken var att ge arbetsgruppen ett tillfälle att granska sig själva och skapa en förbättlingsplan som kan genomföras under nästkommande sprint.

4.7 Dokumentation

Under kundmöte, sprintplanering, sprintgranskning och sprintåterblick har dokumentationsansvarig fört anteckningar om det som sagts och bestämts. Dessa anteckningar har publicerats på gruppens gemensamma *Google Drive* för att vara tillgängliga för hela gruppen.

Under projektets gång har relevanta källor sparats ned och vardera gruppmedlem har dokumenterat sin arbetsprocess med exempelvis anteckningar och skärmumpar. Denna information har också sparats till den gemensamma *Google Drive*:en och det har främst varit dokumentansvariges uppgift att detta följts.

Dokumentation av kod har skett löpande genom projektet. I koden har funktionsanrop förklarats tydligt med kortfattade kommentarer. Genom logiska val av variabel- och funktionsnamn har koden blivit förståelig utan användning av överflödig dokumentation.

4.8 Ansvarsfördelning

Arbetsgruppen har haft olika ansvarsområden enligt följande:

- En roterande scrummästare (Fredrik Norrstig/Pontus Arnesson) som har haft ansvar för att se till att Scrum-processen följts och som väglett arbetslaget genom en effektiv användning av Scrum.
- En produktägare (Vera Fristedt Andersson) som har varit ansvarig för hantering av produktbackloggen. Produktägaren har sett till att maximera värdet av produkten och utvecklarnas arbete samt skött kontakten med kunden.
- En designansvarig (Pontus Arnesson) som har haft huvudansvar för designen av applikationens utseende.

- En testansvarig (Fredrik Norrstig) som har haft huvudansvar för att se till att tester av applikationen har skett regelbundet under arbetets gång.
- En dokumentationsansvarig (Fredrik Johansson) som har fört anteckningar på möten och sett till att alla kollegor sparar ner eventuella källor och dokumenterar sin arbetsprocess.
- En versionsansvarig (Johan Forslund) som har sett till att kollegorna har skött versionsuppdateringen och att den har gått rätt till.

Kapitel 5

Teknisk beskrivning

Beskrivning av de tekniska detaljer som har legat till grund för utförandet av projektet. Kapitlet behandlar de verktyg och system som används och vilka begränsningar som har medföljt med dessa. Denna information framställdes redan innan implementationen påbörjades för att förbereda gruppen och för att få en överblick över hur systemet skulle utvecklas.

5.1 Målplattform

Det finns ett antal olika plattformar att välja mellan vad gäller AR-funktionalitet. Under en lång tid har dock de flesta sådana plattformar baserats på markörspårning, vilket innebär att användaren behöver placera ut en fysisk markör för att systemet ska kunna placera ut och spåra virtuella objekt. Detta är dock inte hållbart i denna applikation eftersom användaren ska kunna ta fram sin telefon när som helst och göra mätningar i en godtycklig miljö. Istället behövs en plattform som kan identifiera omvärlden utan att behöva fysiska markörer.

ARCore uppfyller det behov som nämns ovan genom att använda den SLAM-teknik som beskrivs i sektion 3.4. Även Apple har släppt en liknande plattform (ARKit) som bygger på samma teknik. ARCore och ARKit kan anses likvärdiga, men i detta projekt har ARCore valts som plattform då den stödjer fler mobila enheter i och med att Android används som operativsystem. Detta medför att produkten kan lanseras bredare än om applikationen hade skapats för IOS [10].

Alltså kan applikationen enbart användas på mobiltelefoner som har stöd för ARCore. Detta begränsar mobiltelefonens operativsystem till Android.

5.2 Utvecklingsmiljö

Android Studio är en utvecklingsmiljö skapad av Google för Android-utveckling. Eftersom Google har utvecklat både Android och Android Studio ger det en sömlös utvecklingsmiljö med de kompilatorer, pakethanterare och hjälpverktyg som behövs.

Android studio erbjuder även en mobilemulator. En mobilemulator är en mjukvara som är avsedd för att efterlikna hårdvaran hos en mobiltelefon. Detta effektiviseringar testing av applikationen eftersom applikationen ej behöver överföras till en mobilenhet inför varje test.

5.3 Systembegränsningar

Applikationens AR-funktionalitet begränsas till det som är implementerat i ARCore. Därmed begränsas applikationens förmåga att detektera plan av:

- Det verkliga rummets ljussättning. En välupplyst yta krävs för att ARCore ska kunna detektera den.
- Texturen hos det underlag som mätningen sker på. För att applikationen ska kunna hitta nycelpunkter krävs det att en viss variation finns i ytans textur.
- Hurvida underlaget är plant eller ej.

ARCore stöds enbart på relativt moderna enheter då ARCore fortfarande är relativt nytt. På ARCores hemsida finns en lista över dessa enheter [11].

De tester som har utförts på applikationen har begränsats till de enheter som redovisas i Tabell 5.1.

Tabell 5.1: De mobiltelfoner som används under utvecklingen

Enhet	Operativsystem
Sony Xperia XZ1	Android 9.0, Pie
Oneplus 5T	Android 9.0, Pie
Oneplus 6T	Android 9.0, Pie
Samsung Galaxy S8	Android 9.0, Pie
Google Nexus 6	Android 8.1, Oreo

5.4 System-arkitektur och programdesign

På en överskådlig nivå kan systemet separeras i tre olika delar:

- Grafiskt gränssnitt
- AR
- Databashantering

Applikationens grafiska gränssnitt representeras av olika aktiviteter, se sektion 3.1. Metiri använder följande aktiviteter:

- *MainActivity*
- *RoomActivity*
- *SelectRoomActivity*
- *SettingsActivity*
- *HelpActivity*

Dessa aktiviteter är i sin tur kopplade till varsin XML-fil. Utöver 2D-grafiken från XML-filen använder sig MainActivity och RoomActivity också av 3D-ritade objekt. Dessa objekt visas på skärmen med hjälp av specifika renderare. Genom att skriva en separat renderare för varje typ av objekt kan denna kod lätt återanvändas i applikationen, vilket förhindrar onödig kopiering av kod. En vidare beskrivning av de olika renderarna redogörs i Kapitel 6.

Databashanteringen ansvarar för att lagra den data som har producerats när systemet har analyserat rummet. Databasen är lokal vilket innebär att mobiltelefonen inte behöver vara uppkopplad mot internet vid användning. Detta är en fördel eftersom onödig dataförbrukning kan undvikas. Dessutom ger det applikationen bättre prestanda eftersom den inte behöver vänta på svar från någon extern server.

All kod som hanterar logik är skriven i Java, medan all kod som hanterar det grafiska gränssnittet är skriven i XML. Det är även möjligt att använda *Kotlin* som programmeringsspråk för Android-utveckling, men eftersom den mesta dokumentationen om ARCore är skriven för Java valdes Java istället.

5.5 OpenGL / Sceneform

Ett viktigt val för projektet var huruvida OpenGL eller *Sceneform* skulle användas för att rendera 3D-objekt i miljön. *Sceneform* är ett tillägg till ARCore som erbjuder ett högnivå-API för att skapa scen-grafer och rendera objekt i AR-applikationen. På så sätt kan utvecklarna undvika att behöva hantera OpenGL och de svårigheter som medföljer.

Trots dessa fördelar valdes ändå OpenGL till detta projekt. Det finns två anledningar till detta. Den första är att applikationen enbart innehåller enkla geometrier och de flesta objekt saknar textur. Därför är det överflödigt att importera *Sceneform* som mestadels är tänkt för att rendera modellerade objekt. OpenGL är därför prestandamässigt en bättre lösning. Den andra och viktigaste motiveringen till att OpenGL valdes är på grund av att användaren ska kunna utföra mätningar i en sparad miljö. När dessa mätningar utförs är ARCore inte längre aktivt. Detta skulle innehålla ett stort problem eftersom *Sceneform* specifikt är utformat för att användas med ARCore. Därför var det nödvändigt att använda OpenGL så att användaren ska kunna utföra mätningar i en sparad miljö.

Kapitel 6

Redogörelse för arbetet

Detta kapitel beskriver hur projektet har utförts. Detta innefattar hela processen från skapande av grafiskt gränssnitt till implementering av 3D-grafik och logik för systemet. Besluten som har tagits är väl motiverade och utvecklingsprocessen beskrivs ingående. Notera att delarna nedan inte är beskrivna i kronologisk ordning, utan istället är placerade i en ordning som gör att förståelsen successivt byggs upp.

6.1 Förarbete

Innan implementationen av applikationen påbörjades bekantade sig gruppen med Android-utveckling. Detta gjordes eftersom gruppen saknade tidigare erfarenhet av denna utvecklingsform. Gruppen behövde även bekanta sig med ARCore. Det gjordes genom att studera ett demo-projekt vid namn *HelloAR* [12] som innehåller grundläggande AR-funktionalitet.

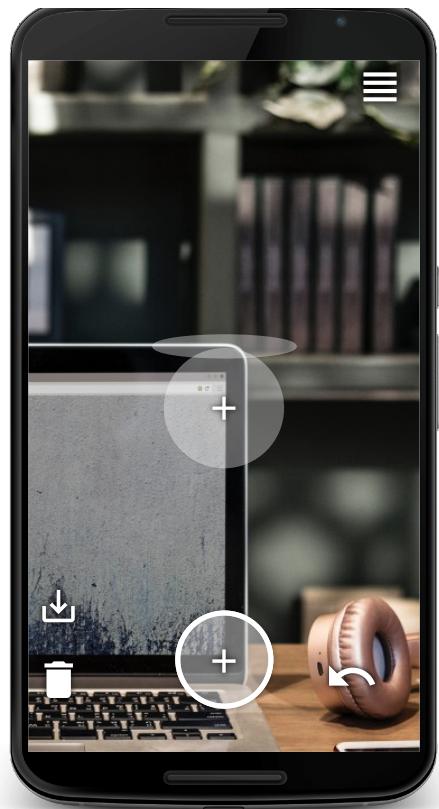
För att underlätta utvecklingen gjorde gruppen individuella skisser för applikationens design. Skisserna tydde på att gruppen hade en gemensam vision gällande applikationens utseende. Genom att kombinera skisserna togs designen, som illustreras i Figur 6.1, fram.

6.2 Grafiskt gränssnitt

Det är viktigt att ha ett väl utformat grafiskt gränssnitt i en applikation för att underlätta användarupplevelsen. En applikations användning framgår tydligt och utan oklarheter om den har ett bra gränssnitt. I detta avsnitt beskrivs gränssnittets funktionalitet och hur arbetsgruppen gick tillväga för att implementera det.

6.2.1 Funktionalitet

Funktionalitet är förmågan att utföra de funktioner som en produkt är konstruerad för. Detta avsnitt redogör för de funktioner som gränssnittet erbjuder användaren och varför gränssnittet ser ut som det gör.



Figur 6.1: Ursprunglig skiss av applikationens gränssnitt

Startsida

Applikationens gränssnitt på startsidan består av en kameravy, två huvudsakliga knappar och två mindre knappar, vilket kan ses i Figur 6.2a. Det är detta gränssnitt som utgör applikationens grund och som visar på applikationens syfte.

Då applikationen är avsedd för att mäta avstånd i användarens omgivning, tar kameravyn upp hela skärmytan. Det är denna kameravy som representerar omgivningen och via denna vy ska användaren kunna mäta avstånd. Detta bidrar till att applikationens huvudsakliga användning framgår extra tydligt. I mitten av kameravyn visas ett hårkors som visar för användaren var mätpunkten kommer att placeras om hen trycker på additionsknappen. Henryson, Billinghurst och Ollila [13] jämförde effektiviteten mellan att använda ett hårkors som sikte och att låta användaren trycka på skärmen för att placera ut objekt i en AR-miljö. Tekniken där ett hårkors användes visade sig vara betydligt snabbare.

Anledningen till att det finns två större knappar och två mindre är för att användaren ska rikta sin uppmärksamhet mot de två större knapparna i första hand. Via dessa två knappar ska användaren kunna lista ut hur hen ska använda applikationen. Meny-knappen tyder på att det finns mer funktionalitet att hitta än det som syns vid första anblick. En meny-knapp tyder också på att det finns hjälp att hitta om användaren inte förstår sig på applikationen.

Meny-sida

Menysidan visar fem olika knappar vilket kan ses i 6.2b. "Help" navigerar användaren till en hjälpsida, "Settings" navigerar användaren till en inställningssida, "Remove all points" raderar alla utsatta mätpunkter, "Save room" sparar ner kameravyn som ett rum till en databas och "Open room" hämtar sparade rum från databasen. Eftersom ikonerna för knapparna "Save room" och "Open room" inte förklarar sig själva, behövdes det även en kort text bredvid för att beskriva dessa funktioner. För att



(a) Startskärm

(b) Skärm vid tryck av meny-knappen

Figur 6.2: Mobilapplikationens grundläggande gränssnitt

uppnå ett stilrent och enhetligt utseende adderades därmed text till alla knappar i menyn.

6.2.2 Implementation

Det första som behövde göras innan gränssnittet kunde implementeras var att integrera ARCore med mobilkameran. För att ARCore ska kunna fungera krävs det att kameravyn renderas utav OpenGL. För att göra detta skapades en renderare som enbart hanterar kameravyn i applikationen. Denna renderare bygger upp en rektangel med hjälp av två trianglar och ritar ut rektangeln på grafikytan. Sedan användes Session-klassen från ARCore som ger tillgång till kamerans bild som en textur. Texturen applicerades sedan på rektangeln. Genom att utföra detta i varje bildruta visas kamerans bild som en bakgrund i applikationen.

Parallelt med detta letade gruppen upp ikoner som passade till de knappar som skulle implementeras. För att uppnå ett konsekvent utseende på knapparna, valdes ett gemensamt tema på ikonerna. Därmed hämtades i stort sett alla ikoner från hemsidan *Material Design* [14]. I de fall då Material Design saknade passande ikoner användes hemsidan *Iconfinder* [15] istället.

6.3 Mätningsfunktionalitet

Kravet för att möjliggöra mätningar i applikationen omfattar följande: Att skapa en struktur som hanterar data om mätpunkters position relativt till varandra. Att kontrollera möjligheten att placera ut mätpunkter på vald yta. Att beräkna avstånd mellan utplacerade punkter. Slutligen genom att kombinera dessa krav implementerades applikationens mätfunktionalitet.

6.3.1 Lista med mätpunkter

För att ha kontroll över de utsatta mätpunkterna skapades en klass som innehåller följande:

- En lista som innehåller mätpunkterna.
- Möjligheten att skapa nya punkter.
- Möjligheten att radera punkter.
- Funktionalitet för att koppla ihop mätpunkter med varandra.

Klassen innehåller en lista av Anchor-objekt, Anchor beskrivs kortfattat i sektion 3.3.1. Vardera objekt i listan kan ha upp till två vänner. Mellan ett objekt och dess vänner kan mätningar göras. Eftersom linjer ritas ut mellan de punkter där en mätning sker har varje objekt en restriktion av maximalt två vänner. Restriktionen finns för att undgå uppkomsten av spindelnätsliknande kluster av linjer.

Klassens lista kan maximalt innehålla 20 objekt. Detta görs för att hålla AR-miljön stilren samt för att begränsa kravet på antal beräkningar i applikationen.

6.3.2 Träffkontroll

När användaren trycker på additionsknappen så anropas en funktion för att testa om det finns ett Plane vid hårkorsets-koordinater. Denna funktion returnerar antingen det Plane som är parallellt under hårkorset, eller en odefinierad yta om den inte träffar något Plane. Genom att anropa denna funktion i varje bildruta har applikationen alltid tillgång till den senaste träffytan. När fler än två punkter angetts sker avståndsberäkning mellan punkterna.

6.3.3 Avståndsberäkning

Avståndet beräknas med hjälp av en funktion som tar in två Anchors som parametrar. Varje Anchor innehåller koordinater vilket gör det möjligt att använda en tillämpning av Pythagoras sats (6.1) för att beräkna avståndet mellan två Anchors.

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} \quad (6.1)$$

Eftersom funktionen används på flera ställen i applikationen placerades den i en hjälpklass för att kunna återanvändas.

6.4 3D-grafik

Varje objekt som ritas till grafikytan använder en specifik renderare för respektive objekt. Både MainActivity och RoomActivity har en grafikyta och implementerar Renderer-gränssnittet, vilket innebär att de innehåller funktionerna onSurfaceCreated() och onDrawFrame().

Alla objekt utgår från den vanliga Modell-Vy-Projektion-strukturen där modellmatrisen beskriver objektets transformation, vymatrisen beskriver kamerans transformation och projektionsmatrisen beskriver hur objektet ska projiceras. Processen för att rita ut objekt varierar beroende på objektets typ. Exempelvis använder vissa objekt orthografisk projektion medan andra objekt använder perspektivprojektion. Mer om hur de olika objekten implementeras beskrivs i Sektion 6.4.2.

6.4.1 Kameratransformation utan rotation

Cirklar, text och linjer är två-dimensionella objekt utritade i en tre-dimensionell miljö. Om dessa objekt skulle rotera 90° upplevs de som osynliga. Det innebär att objekten alltid måste vara vända mot kameran för att de alltid ska synas.

Vanligtvis när kameran roterar runt ett objekt påverkas objektets geometri av rotationen i vymatrisen. Genom att nollställa rotationen i den matris som skapats efter att modell- och vymatrisen multiplicerats kan objektet vändas mot kameran. Detta genomfördes genom att först studera hur en homogen transformationsmatris är uppbyggd, se (6.2).

$$M = \begin{bmatrix} xx & xy & xz & xw \\ yx & yy & yz & yw \\ zx & zy & zz & zw \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6.2)$$

Denna matris är sammansatt av en 3×3 rotationsmatris betecknad R , en 3×1 translationsmatris betecknad T och de homogena värdena, se (6.3).

$$M = \begin{bmatrix} R & T \\ (0, 0, 0) & 1 \end{bmatrix} \quad (6.3)$$

Genom att applicera en uniform skalning med skalningsfaktorn d samt att nollställa rotationen får matrisen i (6.4).

$$M = \begin{bmatrix} d & 0 & 0 & T.x \\ 0 & d & 0 & T.y \\ 0 & 0 & d & T.z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6.4)$$

En ekvation för att skapa matrisen M beskrivs i (6.5).

$$M = I \cdot d + V_r \quad (6.5)$$

I är enhetsmatrisen och V_r är en matris som enbart innehåller 0:or förutom i sista kolumnen där

objekts translation är, enligt (6.6).

$$V_r = \begin{bmatrix} 0 & 0 & 0 & T.x \\ 0 & 0 & 0 & T.y \\ 0 & 0 & 0 & T.z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6.6)$$

Detta resultat applicerades i shadern för cirklarna och linjerna

6.4.2 3D-objekt i applikationen

Applikationen består av ett antal enkla geometrier för att visualisera mätningen för användaren. Varje sådant objekt är direkt skapat i OpenGL. Det finns öppna bibliotek som ger funktionalitet för att rita enkla geometrier, exempelvis *PLIB* [16], men för att få full kontroll över objekten valde gruppen att rita geometrierna direkt i OpenGL.

Hårkors

Det hårkors som används i applikationen skapas i renderaren *CrosshairRenderer* där ortografisk projektion används så att håkorset upplevs ligga på skärmen och alltid behålla samma storlek. Detta hårkors byggs upp av tre plan bestående av sex trianglar där dessa plan tilldelas samma rotation som planet de ligger på. Anledningen till att håkorset skapas med tre plan istället för endast två är att det blev en flimrande mittpunkt där de två planen korsade varandra eftersom pixlarna då renderas ut i varandra. Anledningen till att rotationen adderades till håkorset var för att visa för användaren om någon mätbar yta har hittats, och i vilken riktning detta plan ligger. Detta sker genom att skicka med planets rotationkvaternion [17] och skapa en rotationsmatris av denna, som sedan multipliceras med vymatrisen. Detta medför att håkorset endast kan ritas om träffkontrollen, beskrivs i sektion 6.3.2, har hittat en definierad yta. Slutligen multipliceras modellens matris med de tidigare matriserna för att transformera objektet.

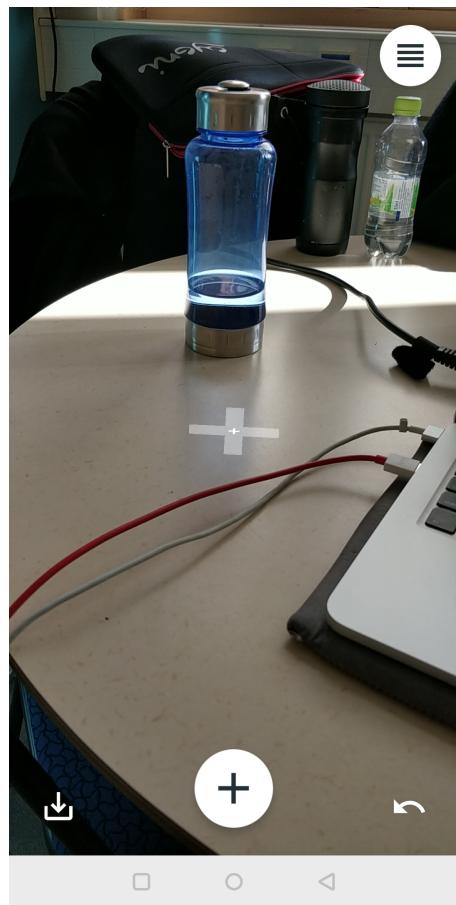
Det skapades två olika objekt av detta hårkors, ett större och ett mindre där båda placerades i mitten av skärmen. Det lilla håkorset fick en opacitet som var helt ogenomskinlig medan det stora har en opacitet på ungefär 50 procent. Detta gjordes för att ha kvar samma precision från tidigare genom det lilla håkorset, men samtidigt uppfatta planets rotation bättre med det stora. De två hårkorsen roterade efter planets yta visas i Fig. 6.3

Mätpunkter

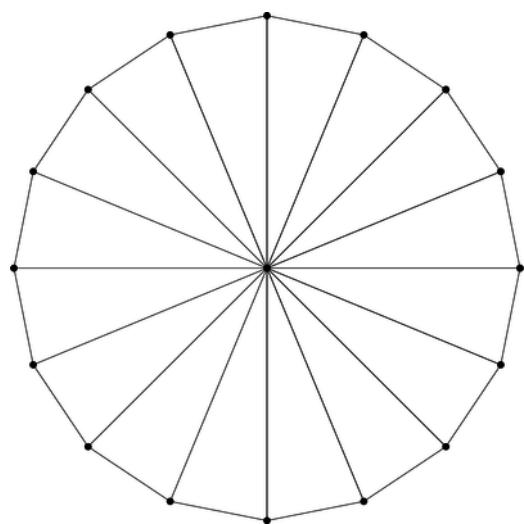
När användaren ska mäta mellan två punkter placeras en cirkel ut i respektive punkt. För att bygga upp en cirkel krävs ett antal trianglar, där antalet bestämmer hur rund cirkeln blir. Ett större antal trianglar kräver dock mer processerkraft. I Figur 6.4 visas en cirkel som är uppbyggd av 16 trianglar, vilket ger en figur som upplevs vara rund.

Eftersom cirklarna i denna applikation är relativt små var det tillräckligt att använda 16 antal trianglar utan att cirklarna upplevs kantiga. Ett matematiskt uttryck för de diskreta punkter som definierar en triangelbaserad cirkel (se Figur 6.4 som referens) redovisas i Ekvation 6.7

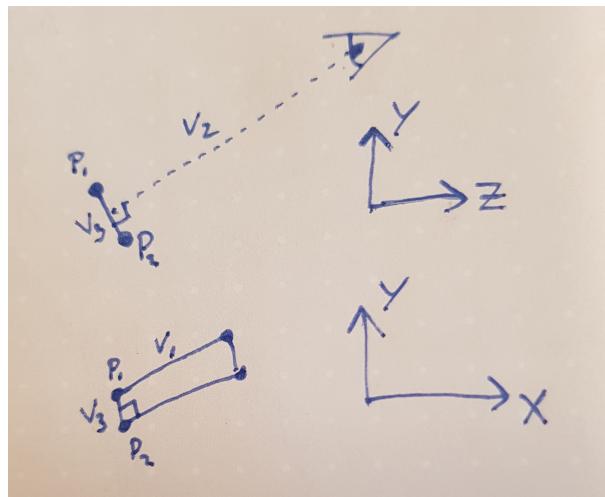
$$(x_k, y_k) = \sum_{k=0}^n (x + (r \cos(\frac{k2\pi}{n})), y + (r \sin(\frac{k2\pi}{n}))) \quad (6.7)$$



Figur 6.3: Hårkors som visar planets rotation.



Figur 6.4: Cirkel som består av 16 likformiga trianglar.



Figur 6.5: Logiken bakom uppritningen av linjernas bredd.

där n är antalet trianglar och r är cirkelns radie. Detta uttryck applicerades i cirkel-renderaren där x och y i Ekvation 6.7 representerar 3D-koordinaterna x och y . Alla z -koordinater sattes till noll eftersom cirkeln i applikationen är två-dimensionella. Detta resulterade i n antal vertex som lagrades i en buffer. Dessa vertex ligger på cirkelns kontur. Eftersom cirkeln ska ritas med trianglar adderas även en vertex i punkten $(0,0)$ vilket är punkten i mitten av cirkeln, se Figur 6.4.

Linjer

För att sammankoppla mätpunkter implementerades linjer mellan punkterna. Vardera linje består av ett plan som är uppbyggt av två trianglar. För att ge planen en korrekt position tilldelades två av dess fyra hörnpunkter koordinater utifrån mätpunkterna eller hårkorsets position. Detta görs kontinuerligt i renderings-loopen för att kompensera för kamerans rörelse. Dessa koordinater hämtas utifrån världskoordinaten för att säkerställa att de använder sig av samma koordinatsystem. Genom att tilldela planen en konstant bredd gavs illusionen av att planen såg ut som linjer. Detta möjliggjordes av Ekvation 6.8 och 6.9. Ekvationerna beräknar de två resterande hörnpunkternas position utifrån de två redan tilldelade positionerna. Detta innebär att planen alltid kan behålla formen av en rektangel. Genom att rikta dessa rektanglarnas yta mot kameran kan linjernas bredd bibehållas. Detta förlopp illustreras i Figur 6.5.

$$v_3 = v_1 \times v_2 \quad (6.8)$$

v_3 är vektorn för linjens bredd, v_1 är vektorn för linjens längd och v_2 är vektorn mellan kameran och linjen.

$$p_2 = p_1 \pm b\hat{v}_3 \quad (6.9)$$

p_1 och p_2 är två av linjens vertex-punkter, b är en konstant som beskriver bredden och \hat{v}_3 är en normaliserad vektor för linjens bredd.

Text

Mättexten är den text som visas ovanför varje utritad linje och visar hur långt det är emellan två punkter. OpenGL har ingen inbyggd funktionalitet för att rita text. Istället har textrenderingsbibliote-

ket *Texample* [18] används. Några modifikationer har gjorts i biblioteket för att ändra hur texten skrivs ut, till exempel att texten alltid ska vara roterad mot kameran.

6.4.3 Borttoning

Eftersom hårkorset endast kan ritas på plan försvann hårkorset abrupt då det rörde sig från ett plan till en odefinierad yta. Denna effekt var oönskad och det påverkade även hårkorsets sammankopplade linjer och text. Därför implementerades en metod som utgör en borttoningseffekt. Metoden tilldelar hårkorset sin senaste definierade position samtidigt som opaciteten minskar linjärt medan träffkontrollens resultat är en odefinierad yta. Detta pågår antingen tills att träffkontrollen hittar en ny definierad yta eller om opaciteten når noll. Metoden kan appliceras på hårkors, linjer och text.

6.4.4 Snap

För att undvika behovet av att placera ut punkter precis bredvid varandra för att sammanlänka två mätstreck implementerades en snap-funktion. Denna funktion implementerades med ett antal saker i åtanke:

- Visuell representation av att linjen förflyttas
- Visuell representation av att hårkorset förflyttas
- Funktionalitet så att beräkningarna blir korrekta och rätt mätpunkt används
- Ge användaren återkoppling i form av haptisk återkoppling

Alla ovanstående punkter körs då träffkontrollens avstånd till en utsatt mätpunkt är mindre än ett visst värde, häданefter snapavstånd.

Vid snapping sker en konvertering mellan världskoordinater och skärmkoordinater, vidare benämnt koordinatomvandling. Denna process innebär att hämta en mätpunkts pose och göra om den till en matris. Därefter multipliceras denna matris med vymatrisen och projektionsmatrisen. Den resulterande matrisen bestående av *clip coordinates* [19] behövde sedan göras om till *NDC-coordinates*, vidare benämnt NDC-koordinater. Dessa NDC-koordinater är 2D-koordinater mellan -1 och 1 där (-1,-1) är skärmens nedre vänstra hörn och (1,1) är skärmens översta högra hörn. Efter detta görs NDC-koordinaterna om till skärmens pixelkoordinater (x,y) genom (6.10) och (6.11).

$$x = w * ((x_N + 1.0) / 2.0) \quad (6.10)$$

w är skärmens bredd och x_N är x-koordinaten i NDC-form.

$$y = h * ((1.0 - y_N) / 2.0) \quad (6.11)$$

h är skärmens höjd och y_N är y-koordinaten i NDC-form.

Ett exempel på detta är när x-komponenten är -1 blir resultatet 0 pixlar, vilket skulle vara längst till vänster på skärmen. Skulle x-komponenten vara 1 blir resultatet w vilket är skärmens bredd i antal pixlar.

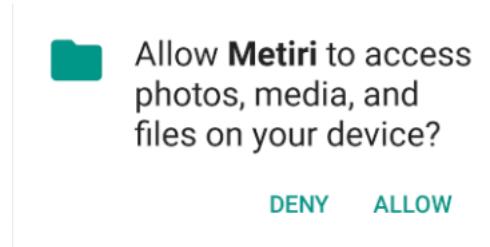
Snapavståndet som beräknas använder koordinatomvandling för att avståndet inte ska bero på världskoordinater utan endast på skärmkoordinater. Detta görs eftersom vid användning av världskoordinater

vid snapping blir punkter långt borta svåra att träffa. Medan punkter som ligger nära får ett för långt snapavstånd relativt till skärmen.

När träffkontrollen är i snapavstånd translateras hårkorset till skärmkoordinaterna för den närliggande mätpunktens pose och enheten vibrerar i fem millisekunder. Om en linje ritades till hårkorset innan snapingen sätts linjens slutpunkt till den närliggande mätpunkten istället. Vid knapptryck på additionsknappen beräknas även all mätfunktionalitet från den mätpunkt snappingen skedde.

6.5 Spara bilder till lagringsutrymme

Användaren har möjlighet att spara ned applikationens kameravy med utsatta mätningar till sin telefons kamerabibliotek. Bilden som sparas får inte med några av knapparna som ligger framför kameravyn i applikationen. Sedan version 6.0 av Android måste applikationen fråga om tillåtelse för att kunna lagra filer [20]. Därför skapades en hjälpklass i applikationen som används för att fråga användaren om tillåtelse då användaren trycker på spara-knappen för första gången, se Figur 6.6.



Figur 6.6: Vad användaren ser när applikationen frågar om tillåtelse för att lagra filer på telefonen.

För att spara bilden läses alla pixlar från grafikytan och skrivs till en bitmap. Funktionalitet för detta är inbyggt i OpenGL ES och fungerar enbart om det körs i OpenGL-tråden. Detta kan alltså inte användas i UI-tråden. Detta leder till en komplikation eftersom när användaren trycker på knappen för att spara körs applikationen just då på UI-tråden. Därför sätts istället en flagga i applikationen som beskriver att användaren har tryckt på spara-knappen. Därmed kan applikationen spara bilden nästa gång som onDrawFrame() kallas på genom att kolla om flaggan är satt.

För att ge användaren mer information om bilden sparas den med dagens datum (och tid) som namn på filen. Bilden komprimeras med *JPEG*-kompression, vilket gör att den inte tar upp lika mycket plats på telefonen som om bilden hade sparats okomprimerad. Syftet med att bilden komprimeras är att applikationen inte ska ta upp onödigt mycket lagringsutrymme på mobiltelefonen.

Denna funktionalitet är även nödvändig när bilder ska sparas till databasen. Skillnaden då är att bilderna enbart innehåller bakgrundsbilden (kameravyn) eftersom mätpunkter och linjer inte ska synas om användaren vill göra mätningar i efterhand. Detta genomförs genom att läsa pixlarna från grafikytan innan alla cirklar, linjer och text ritas ut på skärmen. I övrigt följs samma princip som beskrivits ovan.

6.6 Användartester

Använtartester (se Bilaga A) gjordes cirka två tredjedelar in i projektets gång. Syftet med användartesterna var att ge viktig information som gruppen kunde dra nytta av i det fortsatta arbetet. Använtartesterna genomfördes genom att deltagarna fick ett antal uppgifter de skulle utföra, utan vidare förklaringar. För varje uppgift antecknades en ungefärlig tid samt korta kommentarer om hur upp-

giften utfördes. Efter alla uppgifter var klara fick deltagarna ge egna synpunkter och förbättringar på applikationen.

De uppgifter som skulle utföras av testdeltagarna var:

- Mäta en sida på ett bord
- Ta bort senaste punkten
- Mät valfritt objekt och spara mätningen till mobilens kamerarulle
- Ta bort alla punkter
- Spara miljön till databas

Dessa uppgifter var mer eller mindre svåra för användarna att utföra. Det framgick tydligt vad som var en självklarhet och vad som behövde tydliggöras ytterligare i applikationen.

En del av svårigheterna som uppstod för deltagarna i testerna var förutspådda då applikationen inte var långt gången i det stadiet. Dock fanns det några synpunkter som kom fram som gruppen inte hade tänkt på innan. Ett exempel på detta var att när användaren ska spara en bild till kamerarullen eller för senare mätning. Då riktades enheten ofta ner mot bordet/marken för att leta upp knappen som utför funktionen, för att sedan klicka när denna var funnen. Detta medför att skärmdumpen som tas blir oanvändbar eftersom bilden som producerats inte innehåller någonting användaren önskar.

Sammanfattningsvis saknade de flesta användare en snapfunktion och möjlighet att kunna hitta hjälp eller förklaringar till sådant som var oklart (hjälp-sidan saknade innehåll när användartesterna gjordes). Många hade även problem med att detektera plan, förstå hur utplacering av punkter fungerade och att navigera till databasfunktionaliteten.

6.7 Jämförelser av mätprecision

I slutfasen av projektet jämfördes Metiris mätprecision med liknande applikationer. Dessa jämförelser genomfördes med hjälp av en specifik procedur.

Proceduren innebar att mäta ett objekt från olika avstånd. Mätresultatet antecknades och jämförelseproceduren gjordes om från nya avstånd. Mätresultaten gjordes i millimeterprecision. Detta utfördes fem gånger för vardera applikation för att få ett så rättvist resultat som möjligt.

För att minska procedurens felkälla användes samma objekt vid mätningen. Dessutom mättes avståndet genom att föra telefonen horisontellt längs en bordskant för att säkerställa att rörelsebanan var densamma för vardera test.

6.8 Spara rum

Eftersom användaren ska kunna spara ner miljöer och göra mätningar i efterhand krävs infrastruktur för att hantera denna data. I denna applikation används biblioteket *Room*. Room ger ett abstraktionslager över *SQLite* (som redan finns implementerat i Android). I detta projekt valdes Room eftersom det kräver minimal grundkod för att sätta upp, till skillnad från om *SQLite* används utan Room.

Applikationen består av en databas med två tabeller. Se exempel i Tabell 6.1 och 6.2 för en överblick över strukturen. I sektion 6.8.1 beskrivs hur denna data tas fram.

Tabell 6.1: Databastabell över sparade rum

id	name	saveDate	imagePath	viewMatrix	projMatrix
1	Vardagsrum	1557580393515	/data/user/0/...	(JSON-data)	(JSON-data)
2	Sovrum	1557582847522	/data/user/0/...	(JSON-data)	(JSON-data)
2	Kök	1557583113516	/data/user/0/...	(JSON-data)	(JSON-data)

Tabell 6.2: Databastabell över den sparade miljön

id	x	y	z	i	j	roomId
1	-0.83795...	-0.96184...	-1.72384...	0	0	1
2	-0.83925...	-0.96823...	-1.72953...	0	10	1
3	-0.83153...	-0.96235...	-1.72958...	0	20	1
...

Tabell 6.1 visar hur varje rum lagras. *saveDate* beskriver tidpunkten då rummet sparades i Unix-tid. Detta är vanligt förekommande i databaser och används främst för att det förenklar sorteringen. Själva bilddatan lagras inte i databasen, istället innehåller *imagePath* en adress till den plats i lagringsutrymmet där bilden finns. Detta görs för att inte fylla upp databasen med för mycket data, vilket kan göra att den blir långsammare. *viewMatrix* och *projMatrix* innehåller matriser för vy- respektive projektionsmatris. Eftersom denna data innehåller många värden lagras de i JSON-format, vilket gör att all denna data kan lagras på samma plats.

I Tabell 6.2 redovisas hur applikationen lagrar varje rums miljö. *x*, *y* och *z* är koordinaterna för varje punkt där ARCore har hittat en yta. *i* och *j* är skärmkoordinaterna för dessa punkter. De två tabellerna har en *One-To-Many*-relation, vilket innebär att varje rad i Tabell 6.2 är kopplad till en rad i Tabell 6.1. Denna relation lagras i *roomId*. Detta görs för att lätt kunna hämta alla datapunkter som tillhör ett specifikt rum. Databasen är även konfigurerad sådan att om användaren raderar ett rum raderas även alla datapunkter som är kopplade till rummet.

6.8.1 Spara till databas

Användaren kan spara ned ett rum till databasen och göra mätningar i efterhand. Exempelvis ska användaren kunna spara ned en bild på sitt vardagsrum och vid ett senare tillfälle kunna göra mätningar i bilden trots att användaren inte är i rummet. Detta görs genom att applikationen tar en skärmdump på kamerans nuvarande vy och lagrar i mobiltelefonen. Tillsammans med denna bild sparas all data som ARCore har samlat in medan användaren har använt applikationen. Detta inkluderar information om alla Plane som identifierats, samt hur vy- och projektionsmatrisen såg ut vid den tidpunkt då användaren sparade rummet.

När användaren väljer att spara ett rum till databasen sker ett antal olika steg. Det första steget är att extrahera vy- och projektionsmatris från kamerans nuvarande tillstånd. Därefter sparas ett nytt rum till databasen och nuvarande tid sparas undan i en variabel eftersom den behövs senare. Nästa steg är att iterera genom pixlarna på skärmen med ett visst avstånd mellan varje pixel. Efter att ha testat olika värden bestämde gruppen att 10 pixlars avstånd är rimligt för att få med tillräckligt många pixlar.

I varje iteration görs en träffkontroll med nuvarande pixel-värden. Om träffkontroll-funktionen returnerar ett plan sparas planets rotation tillfälligt undan eftersom det behövs senare i processen.

Nästa steg – som också sker i varje iteration – är att rita ut små rektanglar på grafikytan i varje position på skärmen där träffkontrollen-funktionen fick en träff. Detta görs för att användaren ska kunna se var i bilden som applikationen har data för att göra mätningar. Utan dessa rektanglar skulle användaren

blint behöva testa sig fram när denne gör mätningar. För att rita rektanglarna används en renderare som fungerar på samma sätt som CrosshairRenderer, med den skillnaden att geometrin är en rektangel istället för ett hårkors. Det är i detta steg som den sparade rotation som nyss nämndes kommer till användning. Rektanglarna roteras med denna rotation för att rektanglarna ska ligga längs med den verkliga ytan, se Figur 6.7.



Figur 6.7: En sparad bild redo för mätningar. De små rektanglarna visualiseras hittade plan där punkter kan placeras ut.

När detta är gjort sparas nuvarande träff till tabellen i Tabell 6.2.

Efter att alla pixlar har genomgåtts igenom sparas en bild till lagringsutrymmet på det sätt som beskrivs i Sektion 6.5. Bildens namn blir den tidpunkt som sparades ned i början av processen. Sista steget är därför att användaren att namnge rummet och sedan uppdatera databasen med denna information.

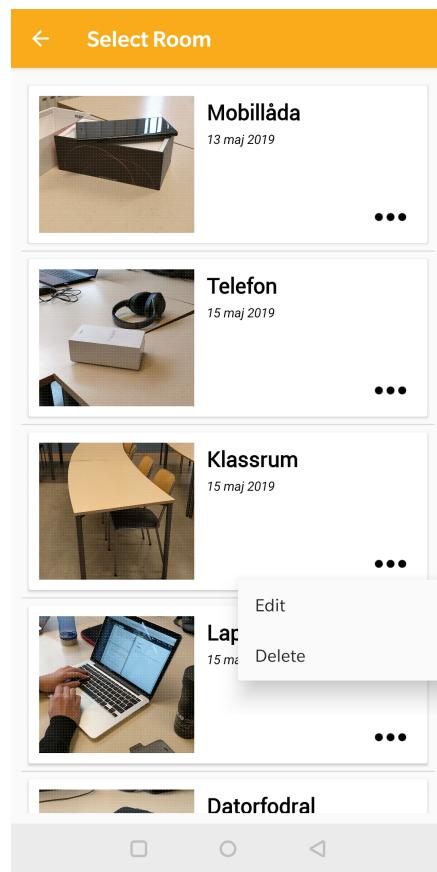
6.8.2 Använda sparad data

I applikationen finns funktioner för att hämta data från databasen. Eftersom dessa operationer är asynkrona körs funktionerna på en ny tråd. Detta görs för att UI-tråden inte ska frysa medan applikationen väntar på svar från databasen. Den nya tråden skapas med användning av klassen *AsyncTask* som finns inbyggd i Android. *AsyncTask* ger två funktioner som är användbara vid asynkront arbete:

- *doInBackground()* – Den kod som skrivs i denna funktion kommer att exekveras på en separat tråd. I denna funktion gör applikationen relevanta SQL-förfrågningar, exempelvis att hämta alla sparade rum från databasen. Funktionen returnerar den data som har hämtats från databasen.
- *onPostExecute()* – Denna funktion körs när databashämtningen är klar. Koden som skrivs i denna funktion körs på UI-tråden. Funktionen tar in den data som *doInBackground()* returnerade. På så vis har funktionen tillgång till data från databasen.

Välja rum från lista

För att komma åt och använda den sparade datan kan användaren välja från en lista med alla sparade rum. Detta görs i SelectRoomActivity som användaren kommer till via ”open room”-knappen i menyn på startsidan. Här läses viss information in från databasen och visas för användaren i form av bild, namn och datum. Detta kan ses i 6.8



Figur 6.8: Aktiviteten där användaren väljer vilket rum som ska användas för mätning.

Denna aktivitet är uppbyggd av en kort-baserad design. Detta innebär att en lista med objekt skapas där varje objekt ser ut som ett kort för att få en tydlig och stilren uppdelning

Information från databasen läses i denna aktivitet in till vardera kort separat genom att ha en brygga mellan databasen och varje vy i listan. Detta sker i bakgrunden tills all data som ska visas laddats klart. Vid knapptryck på ett kort kommer användaren till det rum som klickades och kan sedan mäta i detta rum.

I varje kort finns en ”mer”-knapp som representeras av tre prickar. Vid knapptryck på denna kommer en lista med två val där användaren kan välja att ändra rummets namn, eller ta bort rummet från databasen, se Figur 6.8. Detta gjordes med en popupmeny där dessa två val hämtar ID:et på rummet vars knapp klickades, och utför en uppdatering respektive borttagning av rummet med det ID:et.

Göra mätningar i rummet

När användaren har valt ett rum från listan öppnas en ny aktivitet: RoomActivity. Denna aktivitet använder ID:et som valdes i SelectRoomActivity och hämtar all miljödata (se Tabell 6.2) som tillhör det rummet. Aktiviteten har även en lista över alla punkter som användaren har placerat ut, se Sektion 6.3.1.

Även i denna aktivitet används en GLSurfaceView för att skapa en grafikyta. Den sparade bilden som är adresserad i databasen läses in från fil och ritas som en textur på grafikytan. OpenGL initialiseras så att vy- och projektmatrimer från databasen används vid varje ritning. Detta görs så att alla objekt ritas ut i samma koordinatsystem som applikationen gjorde när användaren valde att spara bilden.

Om användaren trycker någonstans på skärmen anropas en funktion som tar in ett *MotionEvent* som parameter. *MotionEvent* är inbyggd i Android och från den kan applikationen extrahera var på skärmen (i skärmkoordinater) som användaren tryckte. Därefter görs en sökning i tabellen med miljödata för att se om det finns någon punkt med dessa skärmkoordinater. Om sådant är fallet skapas en Anchor med de koordinater (x, y, z) som finns i databasen för dessa skärmkoordinater, och läggs in i listan över mätpunkter. Därefter ritar applikationens renderare ut cirklarna, strecken och texten så att användaren kan se mätresultatet. Detta sker på samma sätt som i reallidsmätningen.

Eftersom användaren trycker på skärmen för att sätta ut mätpunkter är det svårt att få bra precision. Fingrets storlek är betydligt större än det pixelområde där mätpunkten placeras. För att lösa detta har ett förstorningsglas utvecklats till denna aktivitet. Medan användaren för fingret över skärmen visas en ruta där ett område runt fingret förstoras, se Figur 6.9.



Figur 6.9: Förstorningsglas syns längst upp till höger i bild. Fingret (som inte syns) är placerat vid termosen.

Detta implementerades genom ett antal steg. Först ritas en kvadrat ut genom att kombinera två triang-

lar. Därefter används funktionen `glReadPixels()` som beskrivs i Avsnitt 6.5 för att läsa pixlarna runt det område där fingret är placerat. Efter att pixlarna har lästs in till en bitmap ritas de som en textur på den kvadratiska ytan. Eftersom den kvadratiska ytan är större än det område som `glReadPixels()` läser ges effekten av att området runt fingret förstoras.

6.9 Hjälpmittel

För att ge användaren en bra upplevelse krävs hjälpmittel och återkoppling. Därför utvecklades metoder som guidar användaren på ett enkelt och stilrent sätt som är anpassade till de situationer där problem kan uppstå.

6.9.1 Onboarding

Efter att ha genomfört en del användartester (se avsnitt 6.6) framgick det att vissa hade svårt att förstå hur vissa funktioner används. Gruppen tänkte att en lösning på detta problem kunde vara att ta hjälp av *onboarding*. Onboarding är en slags introduktion som användaren genomgår första gången applikationen startas. Genom att använda onboarding kan de viktigaste funktionerna visas vilket underlättar användarens förståelse för applikationen.

Eftersom en del användare inte förstod hur de skulle sätta ut punkter i miljön kände gruppen att det var viktigt att förtydliga detta. Särskilt eftersom utplacering av punkter utgör applikationens grund. Utan denna möjlighet försvisser applikationens syfte helt. Första onboarding-skärmen implementerades därmed för att förtydliga huvudknappens funktionalitet (se Figur 6.10a).

Andra onboarding-skärmen (se Figur 6.10b) användes för att förtydliga att Metiri gör det möjligt att spara kameravyer som det går att mäta i vid senare tillfällen. Skärmen förtydligar även att det finns en meny i applikationen som erbjuder ytterligare funktioner.

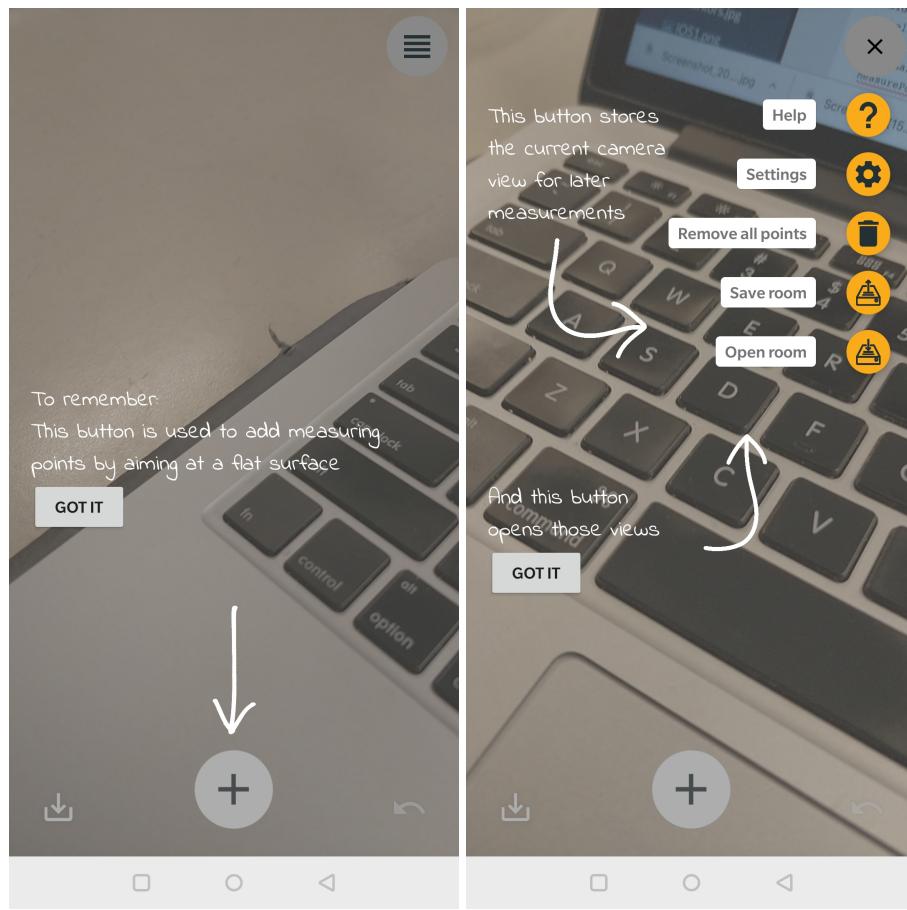
6.9.2 Hjälpillustration för att detektera ytor

För att ARCore ska detektera ytor krävs en viss rörelse av den mobila enheten. En bra strategi för att samla in så mycket data som möjligt av miljön är att göra en cirkulär rörelse med enheten över ytan som ska mäts. Denna rörelse är svår att beskriva kortfattat och tydligt med text. Eftersom huvudsidan ska vara så stilren som möjligt skapades en GIF som går igenom det önskade mönstret som enheten ska röra sig i. GIF:en visas varje gång applikationen startas vilket gör det lättare för användaren att förstå hur hen ska göra för att komma igång med applikationen.

GIF:ens grundläggande utseende skapades först i *Adobe Illustrator* som en Illustrator-fil. Den visar en mobiltelefon som rör sig framför en plan yta med ett objekt liggande ovanpå. Filen exporterades till *Adobe After Effects* för att skapa rörelse. Mobiltelefonens rörelse modifierades med hjälp av *keyframes* som gjorde att enheten kunde röra sig i en önskad bana framför ytan. Slutligen exporterades en video från *Adobe After Effects* med genomskinlig bakgrund till *Adobe Photoshop* där en GIF-fil kunde skapas. En statisk skärmdump där telefonen är halvvägs genom sin loop kan ses i Figur 6.11.

6.9.3 Hjälp- och Inställningssida

Metiri har en hjälpsida, se Figur 6.12, med FAQ (*Frequently Asked Questions*). Här besvaras frågor som användaren kan tänkas ha.



Figur 6.10: Mobilapplikationens onboarding

Det finns även en inställningssida som visas i menyn. Sidan är för närvarande tom, eftersom den fick prioriteras ner under projektets gång. Här är det tänkt att användaren ska kunna ställa in önskad mått enhet, exempelvis centimeter eller tum, samt ha möjlighet att stänga av vibrationerna i applikationen.

6.9.4 Felmeddelanden

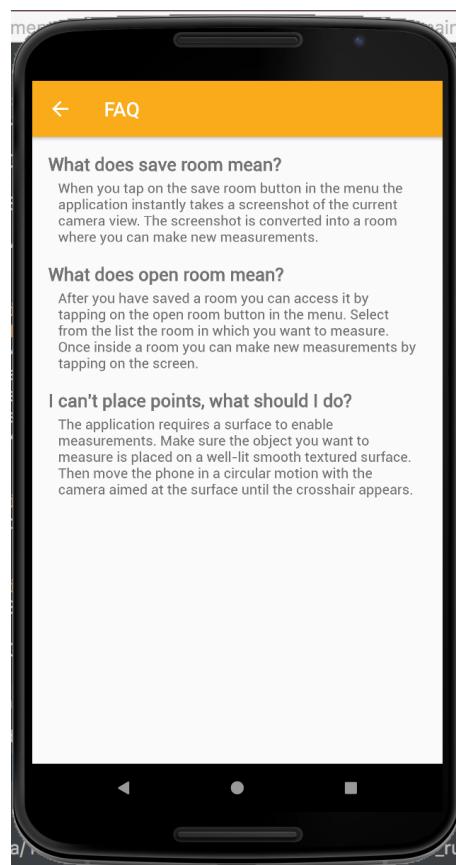
Eftersom användaren inte är medveten om applikationens interna begränsningar krävs felmeddelanden för att kunna uppdatera användaren om särskilda handlingar som inte är möjliga. Utan återkoppling kan användaren uppleva applikationen som frustrerande eftersom hen omöjligt kan veta varför applikationen inte beter sig som användaren förväntar sig. Användaren får upp felmeddelande då hen försöker bryta någon av följande begränsningar:

- Maximalt antal mätpunkter som kan sättas ut är 20 stycken.
- En mätpunkt kan vara kopplad till maximalt två linjer.
- En mätpunkt kan inte kopplas till sig själv.

Ett meddelande kommer även upp när användaren har tagit bort alla mätpunkter i miljön.



Figur 6.11: GIF:en som instruerar hur användaren ska röra enheten.



Figur 6.12: Applikationens hjälpsida

6.9.5 Aktivitetsindikator

Metiri innehåller ett antal tidskrävande funktioner, exempelvis sparning av bilder. Detta resulterar i att applikationens kamerabild fastnar då dessa operationer körs. Detta kan upplevas som irriterande för användaren. Därför implementerades en indikator som ger användaren återkoppling när tidskrävande operationer körs. Indikatorn illustreras i Figur 6.13.



Figur 6.13: Indikatorn som visas när en bild sparas till kamerarullen.

Kapitel 7

Fallgropar

Detta kapitel går igenom problem som gruppen stötte på under utvecklingsprocessen. Detta bidrar till förståelse för vilka alternativ som finns för att skapa en mätapplikation och vilka sådana som bör undvikas.

7.1 Spara till kamerarulle

De flesta exempel som finns på nätet som visar hur bilder sparas till mobilens kamerabibliotek använder Java-funktionen *MediaStore.Images.Media.insertImage()*. Android slutade ge support för denna funktion i nivå Q av sitt API och därför går det inte att använda denna. Istället användes den funktion som beskrivs i sektion [6.5](#).

Varje gång en bild sparades ner till kamerarullen krävdes det att enheten startades om. Utan denna omstart uppdaterades inte galleriets gränssnitt med den nya bilden. Android krävde att applikationen explicit talade om för galleriet att en ny bild har lagts till.

Lösningen på det här problemet gick att implementera på två sätt. Det ena sättet gick ut på att låta galleriet söka igenom alla dess bilder för att sedan uppdatera gränssnittet. Denna lösningen är prestandakrävande ifall galleriet innehåller en stor mängd bilder som måste undersökas. Den andra lösningen innebär att man talar om för galleriet att enbart uppdatera den ny tillagda bilden. Det görs med hjälp av bildens adress som applikationen har tillgång till. Den sistnämnda lösningen är mindre prestandakrävande då den är oberoende av andra objekt i galleriet. Det är även denna lösning som används i Metiri.

7.2 Spara ner miljö

Innan den slutgiltiga lösningen för att mäta i sparade miljöer implementerades, var tanken att gå en annan väg. Idéen var att ta Session-objektet från ARCore och spara ner det för att sedan, tillsammans med en skärmdump, kunna återskapa den tidigare miljön. Eftersom ARCore knappt har några setters var det inte möjligt att ge tidigare miljödata till plattformen. Tanken var då att serialisera Session-objektet och sedan deserialisera det vid ett senare tillfälle. På så sätt skulle tillståndet bevaras. Två olika verktyg användes för att försöka utföra serialiseringen: *JSON* och *Jackson*. Det som visade sig vara ett problem är att klasser i Java behöver implementera gränssnittet *Serializable* för att kunna serialiseras, vilket beskrivs av Opyrchal och Prakash [21]. Eftersom ARCore har låst källkod kunde detta inte implementeras i Session-klassen. Ett ytterligare försök var att skapa en egen klass som ärver från Session-klassen och implementera Serializable på den klassen. Detta fungerade ej eftersom de

klasser som en klass ärver från även måste vara `Serializable`. Det är därmed omöjligt att spara ner Session-objektet och återanvända det vid en annan körning av applikationen.

7.3 Använda samma lista i två trådar

Först uppstod ett problem när listan med mätpunkter användes i `RoomActivity`. När användaren trycker på skärmen läggs en ny punkt in i listan. Samma lista itereras även i `onDrawFrame()` för att rita ut alla mätpunkter. Detta betyder att listan används i två olika trådar. UI-tråden när användaren trycker på skärmen och OpenGL-tråden när punkterna ska ritas ut. När en lista i Java uppdateras i en tråd samtidigt som den itereras i en annan tråd kan systemet krascha med ett `ConcurrentModificationException`.

Lösningen till detta var att konvertera listan till en synkroniserad lista med hjälp av Java-funktionen `Collections.synchronizedList()`. Friesen [22] beskriver att nyckelordet `synchronized` i Java kan användas för att enbart tillåta att en tråd i taget kan interagera med en synkroniserad lista. Syntax för detta redovisas nedan.

```
1     measurePointsList = Collections
2                     .synchronizedList(measurePointsList);
3
4     synchronized (measurePointsList) {
5         for (mp : measurePointsList) {
6             ...
7         }
8     }
```

7.4 Hjälpillustration för att detektera ytor

Innan den slutgiltiga GIF:en gjordes skapades en statisk *SVG* som var tänkt att användas, vilket är en filtyp som Android Studio också stödjer. Denna animerades dock genom att ändra källkoden för filen, där translationen av telefonobjektet i bilden ändrades utefter en bana skapad av bezierkurvor. När den animerade SVG-filen sedan skulle importeras till Android studio var dock detta inte kompatibelt utan bara statiska SVG-filer var möjligt att använda.

Kapitel 8

Resultat

I detta kapitel presenteras slutprodukten Metiri. Metiri har uppfyllt kundens krav och möjliggör mätning i realtid samt i en lagrad miljö. I detta kapitel redogörs även för applikationens mätningsprecision jämfört med liknande applikationer.

8.1 Mätning i realtid

Mobilapplikationens huvudsakliga syfte är att en användare ska kunna mäta sin omgivning i realtid med hjälp av AR. Genom att sätta ut mätpunkter på plana ytor kan avståndet mellan punkterna beräknas enligt metoden som beskrivs i sektion 6.3.3. Figur 8.1 visar hur mätning i realtid ser ut på telefonens skärm. Mätpunkter sätts ut genom att rikta kameran mot ytan, se till att hårrorset lägger sig längs med ytan, och trycka på additionsknappen i botten av skärmen.

När en användare mäter i realtid finns det möjlighet att sätta ut maximalt 20 punkter åt gången. Det går att ta bort den senast utsatta punkten och att ta bort alla utsatta punkter. Det finns också möjlighet att spara applikationens kameravy med punkter och mätningar, utan att få med knapparna, till mobilens kamerarulle.

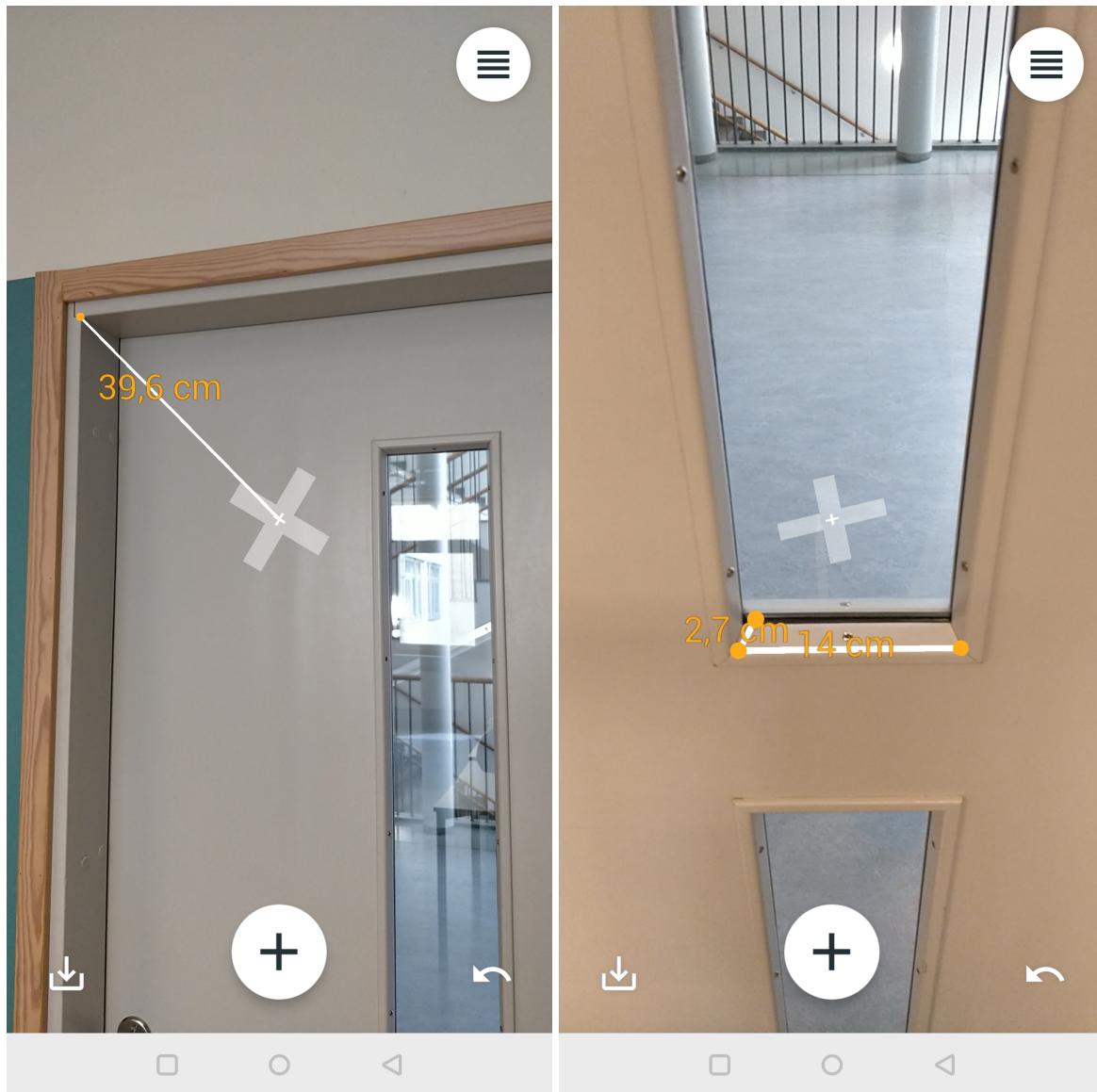
8.2 Mätning i lagrad miljö

Mobilapplikationen gör det möjligt för en användare att spara en miljö i en lokal databas för senare mätning. Här sätts mätpunkter ut med hjälp av fingertryck. För att se var fingertrycket sker kommer ett förstoringsglas upp i högra hörnet som visar var i den sparade miljön användaren pekar. Figur 8.2 visar hur mätning i en sparad miljö ser ut på telefonens skärm.

När en användare mäter i en lagrad miljö finns det möjlighet att sätta ut maximalt 20 punkter åt gången. Det går att ta bort den senast utsatta punkten och det går att spara applikationens kameravy med punkter och mätningar, utan att få med knapparna, till mobilens kamerarulle. Det är endast möjligt att sätta ut mätpunkter på de områden som täcks av vita prickar då det är dessa ytor som ARCore han analysera innan miljön sparades ner till databasen.

8.3 Användartester

Resultatet av användartesterna (Bilaga A) var att även om applikationen inte hade alla nödvändiga funktioner, ansågs gränssnittet på förstasidan som enkelt och tydligt. För användare som inte hade



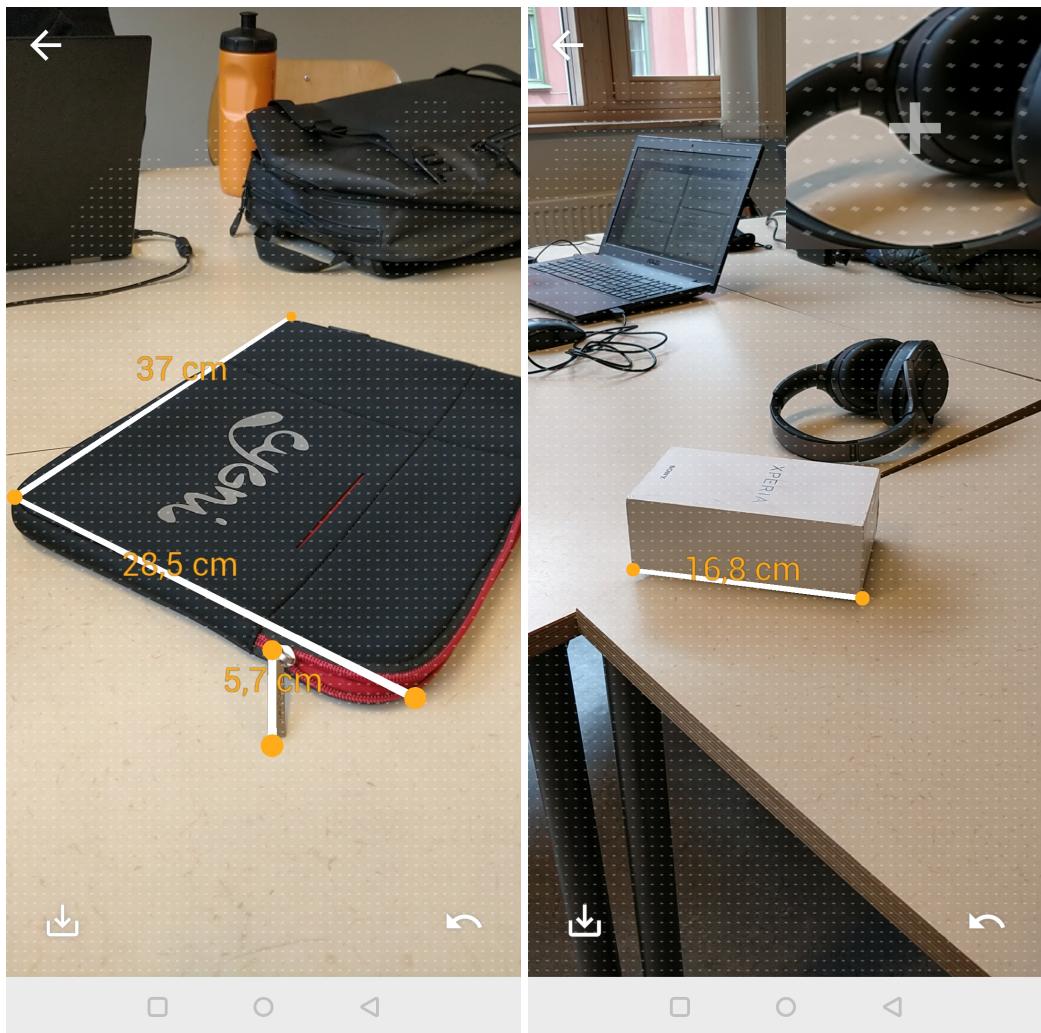
Figur 8.1: Mätning i realtid

använt en liknande applikation tidigare och som inte hade hög teknisk färdighet var det i vissa fall problematiskt att göra en mätning. Detta resulterade i att mer fokus lades på att göra det mer lättförståeligt för användaren att göra enkla mätningar, samt förtydliga de mer svår förståeliga funktionerna.

8.4 Jämförelse av mätprecision

Metiris mätresultat stämmer till viss del överens med verkligheten. Vid en mätning av ett LiU-kort med måtten 8,5 cm x 5,3 cm gav Metiri resultatet 8 cm x 5 cm, se Figur 8.3.

Jämförelser av mätprecision som genomfördes med liknande applikationer kan ses i tabell 8.1. Värdena i tabellen är från mätningar av en Macbook Pro, där resultatet är ett genomsnitt av alla mätningar som gjordes med de olika applikationerna.



(a) Flera mätningar

(b) Mätning och förstoringsglas

Figur 8.2: Mätning i sparad miljö

Tabell 8.1: Jämförelse av mätprecision för olika applikationer på ett objekt med bredden 31,4 cm

Applikation	Avstånd 25 cm	Avstånd 50 cm
Metiri	32 cm	31,9 cm
iOS Mätverktyg	31 cm	30,8 cm
AR Ruler app	31 cm	32 cm



Figur 8.3: Mätning av ett LiU-kort

Kapitel 9

Analys och diskussion

Detta kapitel analyserar och diskuterar resultatet utifrån kundens krav och förbättringsmöjligheter. Generellt har arbetsprocessen fungerat bra och kombinationen mellan ARCore och OpenGL visade sig vara lämpliga för att uppnå de mål som sattes för projektet. Nedan diskuteras mer specifikt de olika delarna i projektet.

9.1 Grafiskt gränssnitt

Det framgick via användartesterna (se avsnitt 6.6) att de ikoner som finns i applikationen tydligt förklarar sina funktioner utan missförstånd. Flera användare uppskattade dessutom det minimalistiska utseendet med få knappar på startsidan. De tyckte att applikationens användning och syfte tydligt framgick.

Användartesterna framhövde också ett problem som ofta uppstod. När användarna skulle spara applikationens kameravy till mobilens kamerarulle, riktade de ofta ner kameran i knäet innan de tryckte på spara-knappen. På grund av tidsbrist hann gruppen inte implementera en lösning för detta. En lösning skulle kunna vara att ha en nedräkning till när bilden faktiskt tas alternativt ha en text som säger åt användaren att rikta kameran mot önskad vy och sen trycka på en "ta bild"-knapp.

En annan förbättringsmöjlighet skulle kunna vara att kombinera spara till kamerabibliotek med spara till databas i en och samma knapp. Gruppen upplever att dessa två funktioner är lika frekvent använda men har inte kommit på en bra lösning för hur det skulle kunna göras på ett snyggt och användarvänligt sätt.

Eftersom hårkorsets snap beräknas med hjälp av skärmens pixlar medför det att snap-känsligheten kommer att variera mellan telefoner med olika skärmplösningar. Detta är något som skulle åtgärdas genom att ta tillhän mobiltelefonens skärmupplösning när beräkningarna sker.

9.2 Mätningsfunktionalitet

Metiri erbjuder inte användare möjlighet att vrida på skärmen till horisontellt läge för tillfället. Flera användare upplevde att ett horisontellt läge hade passat bättre vid mätningar och gruppen hade gärna sett att den möjligheten fanns. Tyvärr fanns det inte tillräckligt med tid för att implementera ett gränssnitt som kan skifta till horisontellt läge.

9.3 3D-grafik

Mättexten skulle kunna synas bättre i applikationen. För närvarande smälter texten ibland ihop med mätpunkterna eftersom de har samma färg. En tjockare kant kring texten skulle kunna vara en lösning till detta problem. De delar av mättexten som ligger framför mätslinjerna syns heller inte särskilt bra då den orange färg som gruppen valt inte sticker ut tillräckligt på vit bakgrund. En lösning till detta skulle kunna vara att byta färg på text och punkter, alternativt färg på linjer.

Det faktum att OpenGL ES var en nödvändighet för detta projekt visade sig vara tidskrävande. Det har lagts mycket tid på att lösa 3D-tekniska problem. Vid ett projekt som *enbart* använder ARCore för att hantera miljön (särskilt om avancerade 3D-objekt krävs) är det förmodligen mer lämpligt att använda Sceneform som 3D-API.

9.4 Spara bilder till lagringsutrymme

Att spara bilder till lagringsutrymme tar tid i applikationen men eftersom återkoppling ges till användaren gällande detta kan eventuell frustration undvikas. Att det tar tid är något som gäller applikationens prestanda och det är något som inte har varit i fokus under detta projekt. Främst eftersom ingen i gruppen har några större kunskaper om hur man arbetar med prestandaförbättringar i en AR-applikation.

9.5 Användartester

Något som gruppen hade valt att satsa mer på, om projektet hade gjorts om, hade varit användartester. Det hade varit nyttigt med fler och mer utförliga användartester för att kunna se mönster i hur användarna upplever Metiri. Med fler användartester hade valen gällande applikationens funktionalitet och utseende dessutom kunnat styrkas mer.

Gruppen hade också velat göra nya användartester i slutet av utvecklingsprocessen när applikationen var klar. Då hade valen som togs utefter användartesterna, som gjordes två tredjedelar in i utvecklingsprocessen, varit möjliga att stödja genom att se om användarna denna gång hade lättare att förstå applikationen. När användartesterna i avsnitt 6.6 gjordes hade bland annat databasdelen inte utvecklats. Alltså vet gruppen inte hur användare uppfattar denna del utan har bara haft sig själva att utgå från under utvecklingen av detta. Det hade därmed varit gynnsamt om det fanns möjlighet att finslipa applikationen ytterligare utefter användartester kring slutprodukten.

9.6 Jämförelse av mätprecision

De applikationer som det har genomförts tester på använder sig av antingen ARCore eller ARKit. Det innebär att mätsresultaten mellan de applikationer som använder sig av samma plattform inte skiljer sig avsevärt.

Varken Mätverktyg eller AR Ruler app stödjer millimeterprecision. Detta medför att mätningar av små objekt, exempelvis tangenter på ett tangentbord, inte blir lika exakta som för Metiri. Dock är det så pass stor felmarginal vid mätningar av större objekt, att centimeterprecision då skulle anses som fullt tillräckligt.

Mätsresultaten kan även skilja sig mellan olika enheter eftersom varje kamera har unika intrinsics. Inledningsvis var tanken att motverka mätningsfel genom att utnyttja detta faktum men det exkluderades

eftersom det inte är möjligt att kalibrera ARCore.

Resultaten av mätningarna är sammanfattningsvis relativt exakta för alla tre applikationer. Dessa applikationer kan dock inte helt ersätta måttband eller andra traditionella mätverktyg då precisionen inte alltid är tillräckligt pålitlig.

9.7 Databashantering

Mätningar i sparade rum är ett koncept som inte har funnits tillgängligt sedan tidigare. Detta innebar att gruppen var tvungna att vara intuitiva under denna implementering. Detta resulterade i att implementeringen blev väldigt tidskrävande då ett flertal idéer visade sig vara otillräckliga. Den slutgiltiga metoden som visade sig fungera som beskrivs i sektion 6.8.2. Även om denna metod fungerar är den bara i begynnelsestadiet. Här finns alltså mycket förbättringsmöjligheter.

9.8 Hjälpmmedel

9.8.1 Onboarding

Gruppen ville helst ha progressiv onboarding i applikationen. Progressiv onboarding innebär att första gången applikationen startas tvingas användaren att interagera med applikationens funktioner innan onboardingen försvinner [23]. Då skulle man exempelvis kunna instruera användaren att sätta ut några mätpunkter så att hen får uppleva hur det faktiskt görs istället för att enbart läsa en kort instruktion. Man skulle också kunna instruera användaren att spara ner en första miljö till databasen så att hen bekantar sig med denna funktion och lär sig förstå vad det innebär istället för att behöva gissa sig fram. Då onboardingen utvecklades i slutet av projektet och på grund av brist på exempel gällande progressiv onboarding på internet, beslutade gruppen sig för att köra på en mer enkel variant som beskrivs i avsnitt 6.9.1.

Metiris onboarding skulle behövd ytterligare arbete. Onboarding-skärmarna är inte anpassade för olika skärmstorlekar och därmed hamnar text och pilar helt fel på vissa telefoner. Andra onboarding-skärmen, som är synlig i Figur 6.10b, ser inte ens bra ut på den skärmstorlek som den är utvecklad för. Det är inte tydligt vilka av knapparna som pilarna faktiskt pekar på. På grund av tidbrist och att ingen i gruppen är särskilt kunniga gällande XML, prioriterades detta bort men det skulle behöva justeras.

9.8.2 GIF

GIF:en blinkar till när den visas i applikationen vilket kan uppfattas som irriterande vid användning. Den hade ännu inte implementerats då användartesterna gjordes och därmed saknar gruppen information gällande hur den uppfattas av användare. Den kanske kan behöva kompletteras med en kort text som exempelvis säger Rör telefonen för att hitta ytor"eller liknande för att användarna verkligen ska förstå vad de ska göra. Det skulle även kunna vara så att användare hellre ser att GIF:en är placerad i mitten av skärmen istället för övre delen av skärmen. Gruppen upplevde det som störande när GIF:en skymde hålkorset då den var placerad i mitten och tog därfor beslutet att placera den i övre delen.

9.8.3 Hjälp- och inställningssida

Hjälpsidan borde vara mer utförlig. Det hade varit hjälpsamt med figurer och ritningar för att förklara sådant som är svårt att beskriva med ord och det hade behövts fler frågor och svar.

Inställningssidan har som tidigare nämnts inte implementerats än. Gruppen tror framförallt att det hade uppskattats av användare om det gick att ändra måttenhet så att applikationen blir mer internationellt användbar. Möjlighet att stänga av vibrationer upplevs också som fördelaktigt. Ytterligare inställningar skulle kunna vara att exempelvis ändra färg på mätpunkter, text, iconer i menyn, etc. från orange till något annat.

9.9 Resultat

Den felmarginal som kan uppstå hos mätningar i applikationen kan bero på att när en mätning gjordes sattes mätpunkterna inte ut exakt där de var tänkta att placeras. Annars kan det bero på att ARCore inte fick tillräckligt med tid på sig att analysera omgivningen innan mätningen gjordes.

Sammanfattningsvis stämmer det slutgiltiga resultatet bra överens med de förväntningar som både gruppen och kund haft på projektet.

9.10 Arbetet i ett vidare sammanhang

Eftersom att applikationen ska göra det möjligt för användare att spara ner miljöer i en databas för att kunna mäta i dem vid ett senare tillfälle kan det ställa till med integritetsproblem. Det kan hända att personer råkar fastna på bild som inte vill vara med. När användaren väljer att radera en post i databasen raderas enbart miljödatan, inte kamerabilden. Detta kan leda till att användaren råkar ta en bild på något olämpligt och sedan inte kan ta bort bilden. Det enda sättet att radera en sådan bild i nuvarande läge är att rensa all data som tillhör applikationen, men detta är något som användaren inte vet om. Vid vidareutveckling av applikationen kommer problemet att åtgärdas.

Gruppen har inte undersökt hur applikationens färger påverkar färgblinda. Det hade behövts undersökas vilka färgkombinationer som orsakar förvirring för människor som har problem med färgseendet och anpassa applikationen efter detta.

För att inkludera fler användare skulle Metiri också kunna utvecklas för iOS-enheter.

Under användning av Metiri kan användaren bli uppslukad av applikationen vilket leder till att hen inte är medveten om sin omgivning och kan råka ut för fara. Detta är något som utvecklarna kan försöka ta hänsyn till vid vidareutveckling genom att exempelvis meddela användaren om att hen har använt applikationen länge, alternativt meddela användaren att de behöver se upp om de backar medan de använder applikationen.

Kapitel 10

Slutsatser

Syftet med projektet var att utveckla en mobilapplikation som kan sätta ut punkter i en AR-miljö med hjälp av ARCore, för att sedan mäta avståndet mellan dem. Metiri gör just detta och erbjuder dessutom möjlighet att spara ner en miljö för mätning vid ett senare tillfälle - något som efterfrågades av kunden vid projektets start. Applikationen har under hela projektets gång utvecklats med användaren i fokus och mätresultaten stämmer överens med de förutsättningar som ARCore gav.

Hur kan den data som ARCore identifierar lagras för att göra mätningar vid ett senare tillfälle?

Det är två begränsningar som förhindrar att det ska vara möjligt att göra mätningar direkt i ARCore vid ett senare tillfälle. Den första begränsningen beror på att Session-klassen i ARCore inte har några setters, vilket gör det omöjligt att ge ARCore information vid ett senare tillfälle. Den andra begränsningen beror på att Session-klassen inte är serialiserbar, vilket gör det omöjligt att lagra ett session-objekt för att användas vid ett senare tillfälle.

Den lösning som istället utvecklades i detta projekt är att göra mätningarna i efterhand utan att använda ARCore. Detta görs genom att vid ett givet tillfälle scanna igenom omgivningen och spara ned denna data i en databas tillsammans med en skärmdump av nuvarande kameravy. Även kamera- och perspektivmatriser sparar i databasen. Därefter kan applikationen utföra mätningar genom att detektera vilka skärmkoordinater som användaren trycker på och sedan använda datan från databasen för att utföra mätningarna. Eftersom kamera- och perspektivmatriser har lagrats kan 3D-grafiken renderas korrekt i den statiska bilden med hjälp av OpenGL.

Hur kan ett gränssnitt utformas så att användaren upplever gränssnittet som intuitivt i en AR-applikation där det är nödvändigt att placera ut objekt?

För att skapa ett intuitivt gränssnitt åt Metiri, utgick gruppen från de fördelar och nackdelar som identifierades hos gränssnitten i Mätverktyg och AR Ruler App, se avsnitt 2.1 & 2.2. Mätverktygs gränssnitt upplevdes till största del som väldigt intuitivt och väldesignat. Det som applikationen saknade var självförklarande iconer och att det inte fanns någon hjälp för användaren att hitta. Två väl använda knappar var dessutom placerad i övre delen av skärmen vilket minskar användarvänligheten. Vad gäller AR Ruler App upplevde gruppen dess gränssnitt som rörigt och icke-intuitivt. Mätverktyg blev alltså den app vars gränssnitt gruppen hämtade inspiration ifrån och nackdelarna gällande både applikationers gränssnitt hölls i åtanke under hela utvecklingsprocessen.

Det resulterande gränssnittet visade sig vara både enkelt och tydligt enligt de användare som utförde användartester på applikationen (se avsnitt 8.3). Gränssnittet framhäver vad applikationens huvudsyfte är, det är tydligt vad alla knappar innebär och de mest använda knapparna är placerade i botten av skärmen för att ge lättare åtkomst åt användaren. Metiris gränssnitt har alltså uppfyllt kundens krav gällande att förbättra en redan existerande mätverktygs-applikation eftersom det är en förbättring på både Mätverktyg och AR Ruler App.

Det finns olika sätt att låta användare placera ut objekt (mätpunkter) i en AR-miljö. I Avsnitt 6.2.1 redogörs att ett hårkors i mitten av skärmen är den bästa lösningen.

Vilket 3D-API för mobiltelefoner är bäst utformat för att rita grundläggande geometrier i både AR och icke-AR?

Vid användning av ARCore finns det två alternativ för att rendera 3D-objekt i applikationen. Antingen görs detta direkt med OpenGL eller så används ARCore-tillägget Sceneform. Däremot visar det sig att Sceneform är utformat för att användas exklusivt med ARCore, vilket gör att en applikation som kombinerar AR och icke-AR ej bör använda detta API, se sektion 5.5. Istället behöver OpenGL användas för att kunna rendera i båda delarna av applikationen. Detta medför även fördelen att applikation får bättre prestanda eftersom Sceneform inte behöver inkluderas i applikationen.

För applikationer som ej kombinerar AR och icke-AR bör Sceneform ses som ett starkt alternativ. Det tar tid att utveckla applikationer med OpenGL eftersom den inbyggda funktionaliteten är begränsad. Bara att rita grundläggande geometrier så som cirklar och linjer kräver eftertanke och mycket kod. Däremot kan detta förenklas genom att använda ett färdigt bibliotek för att rita geometrier, exempelvis PLIB. Eftersom Sceneform redan erbjuder denna funktionalitet kan arbetsprocessen effektiveras genom att använda detta API.

Litteraturförteckning

- [1] Apple, *Mätverktyg*, hämtad: 2019-05-15
<https://itunes.apple.com/se/app/m%C3%A4tverktyg/id1383426740?mt=8>
- [2] Google, *AR Ruler App*, hämtad: 2019-05-15
<https://play.google.com/store/apps/details?id=com.grymala.aruler&hl=sv>
- [3] The Khronos Group Inc, *OpenGL ES Overview*, hämtad: 2019-05-14
<https://www.khronos.org/opengles/>
- [4] C. Wang, C. Thorpe, *Simultaneous localization and mapping with detection and tracking of moving objects*, IEEE
- [5] Ken Schwaber & Jeff Sutherland, *Scrumguiden*, 2013
<https://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-SE.pdf>
- [6] Ian Sommerville, *Software Engineering, 10th Edition, Global Edition*, Pearson 2016
- [7] Trello
<https://trello.com>
- [8] Git
<https://git-scm.com>
- [9] GitHub
<https://github.com>
- [10] StatCounter, *Mobile Operating System Market Share Worldwide*, hämtad: 2019-06-03
<http://gs.statcounter.com/os-market-share/mobile/worldwide>
- [11] Google, *ARCore supported devices*, hämtad: 2019-05-15
<https://developers.google.com/ar/discover/supported-devices>
- [12] Google, *Hello ARCore*, hämtad 2019-05-15
<https://github.com/google-ar/arcore-android-sdk/tree/master/samples/>
- [13] Henrysson. A, Billinghurst. M, Ollila. M, *Virtual object manipulation using a mobile phone*, Proceedings of the 2005 international conference on Augmented tele-existence
- [14] Material Design, *Icons*, hämtad: 2019-03-04
<https://material.io/tools/icons/?style=baseline>
- [15] Iconfinder, *Iconfinder*, hämtad: 2019-03-05
<https://www.iconfinder.com/>
- [16] Sourceforge, *A Simple Geometry library for OpenGL*, hämtad: 2019-05-30
<http://plib.sourceforge.net/sg/index.html>

- [17] Kvaternion, hämtad: 2019-05-14
<http://emis.ams.org/proceedings/Varna/vol1/GEOM09.pdf>
- [18] Fractious, *Rendering text in OpenGl on Android*, hämtad: 2019-05-14
<http://fractiousg.blogspot.com/2012/04/rendering-text-in-opengl-on-android.html>
- [19] World to screen coordinates, hämtad: 2019-05-15
http://www.songho.ca/opengl/gl_transform.html
- [20] Android, *App permissions best practices*, hämtad: 2019-05-16
<https://developer.android.com/training/permissions/usage-notes>
- [21] L. Opyrchal, A. Prakash, *Efficient object serialization in Java* , IEEE
- [22] Jeff Friesen, *Java threads and the concurrency utilities*, Apress
- [23] Medium, *Essential Onboarding Tactics - Part 1 Progressive Onboarding*, hämtad: 2019-05-15
<https://medium.com/welcome-aboard/essential-onboarding-tactics-part-1-progressive-onboarding-ad2847b83fbf>

Bilaga A

Användartester

Användartest 1

Ålder: 22 år.

Kön: Kvinnan.

Teknisk färdighet (1-5) : 5.

Denna försöksperson hade tidigare använt appen lite.

Uppgift 0 - Få upp crosshair (nämns inte):

Tid: 20 s.

Instruktioner: Inga.

Uppgift 1 - Mät en sida på ett bord:

Tid: 30 s.

Fritext: Hon fick inte upp hårkorset direkt men kunde sätta punkt, då tog hon bort punkten för att hitta hårkorset för att sedan mäta.

Uppgift 2 - Ta bort senaste punkten:

Tid: Direkt.

Fritext: Inga oklarheter.

Uppgift 3 - Mät valfritt objekt och spara din mätning till mobilen:

Tid: Direkt.

Fritext: Gjorde exakt som man ska.

Uppgift 4: Ta bort alla punkter:

Tid: Instant.

Fritext: Hon använde bakåtknappen för att radera alla punkter direkt.

Uppgift 5: Spara miljön till databas:

Tid: 10 s.

Fritext: Hon försökte använda "spara till kamerarulle"-knapp. Efter jag sagt att den sparade till kamerarulle öppnade hon menyn och hittade rätt.

Utvärdering:

- Utan någon info skulle hon inte veta vad "spara till databas" betyder eller gör.
- Bakåtknapp fungerar bra, remove all points är lite onödig. Kanske bara hålla in bakåt?
- Hade velat ha en "hjälpknapp", settings kanske bara för att ändra till feet tex.
- Tycker att det var enkla iconer där man förstod vad de gjorde. Inte så mycket på skärmen vilket var bra.
- Man behöver inte så mycket instruktioner och kan testa sig fram.

Användartest 2

Ålder: 21 år.

Kön: Man.

Teknisk färdighet (1-5) : 5.

Uppgift 0 - Få upp crosshair (nämns inte):

Tid: Direkt.

Instruktioner: Inga.

Uppgift 1 - Mät en sida på ett bord:

Tid: 20 s.

Fritext: Han fick upp hårrorsetet snabbt men vände till horisontellt läge. Allt gick som tänkt annars.

Uppgift 2 - Ta bort senaste punkten:

Tid: 15 s.

Fritext: Han försökte trycka "add" med hårrorsetet på punkten, efter han satt ut fler punkter kollade han efter fler möjligheter och hittade bakåtknappen. Tummen var i vägen samt att den stora knappen i mitten drog uppmärksamhet.

Uppgift 3 - Mät valfritt objekt och spara din mätning till mobilen:

Tid: 30 s.

Fritext: Han fick något annat plan än som var tänkt så mätningen blev felaktig. Tog bort punkterna och testade igen och då gick det, sen sparade snabbt till kameran.

Uppgift 4: Ta bort alla punkter:

Tid: Direkt.

Fritext: Han klickade flera gånger på bakåtknapp. Han höll dock på punkterna med hårrorsetet när han klickade bakåtknappen.

Uppgift 5: Spara miljön till databas:

Tid: 15 s.

Fritext: Spara till kamerarullen klickades först. När detta inte gick gick han in i menyn där han hittade knappen som han tyckte hade otydligt namn.

Utvärdering:

- Bra applikation om mätten är korrekta.
- Enkelt att förstå vad man ska göra.
- Tydliga signifiers.
- Snappen-funktionalitet var lite otydlig
- Bra feedback, toasts behövs troligtvis inte dock.
- Man skulle kunna ha att hårrorsetet väljer vilken punkt som ska bort!

Användartest 3

Ålder: 56 år.

Kön: Man.

Teknisk färdighet (1-5) : 4.

Uppgift 0 - Få upp crosshair (nämns inte):

Tid: Direkt.

Instruktioner: Inga.

Uppgift 1 - Mät en sida på ett bord:

Tid: 10 s.

Fritext: Allt gick smidigt. Han fastnade lite med snappen i sista punkten och avståndets text blev utanför skärmen så han fick flytta kameran för att hitta avståndet.

Uppgift 2 - Ta bort senaste punkten:

Tid: Direkt.

Fritext: Han använde bakåtknappen direkt.

Uppgift 3 - Mät valfritt objekt och spara din mätning till mobilen:

Tid: 10 s.

Fritext: Han fundera på vad som skulle mätas sedan mätte felfritt (han mätte mellan två vägg-plan) och sparade med rätt knapp.

Uppgift 4: Ta bort alla punkter:

Tid: Direkt.

Fritext: Han använde bakåtknapp.

Uppgift 5: Spara miljön till databas:

Tid: 5 s.

Fritext: Han gick in i menyn. Men efter att ha klickat på ladda upp ville han klicka något mer för att få någon typ av feedback. Till exempel en ok knapp och text.

Utvärdering:

- Är det en vit bakgrund vill han har svart kors (alternativt mer kontrast eller skugga).
- Han vill snappa till hörn och punkter i miljön. Alternativt zoom för bättre precision.
- När användaren ska spara till databas så vill han ha text där det står att man ska rikta kameran rätt, sedan en knapp där det står spara. Detta för att inte få fel bild (typ att man kollar ner i marken)
- Feedback var bildens hamnar.

Användartest 4

Ålder: 54 år.

Kön: Kvinna.

Teknisk färdighet (1-5) : 3.

Uppgift 0 - Få upp crosshair (nämns inte):

Tid: Direkt.

Instruktioner: Inga.

Uppgift 1 - Mät en sida på ett bord:

Tid: 20 s.

Fritext: Hon var inte säker på hur hon skulle göra så utforskade menyn sen satte ut med add.

Hon var ganska nära bordet men det gick att sätta ut punkt.

Uppgift 2 - Ta bort senaste punkten:

Tid: 5 s.

Fritext: Hon frågade om man skulle ångra sig och då tryckte hon rätt.

Uppgift 3 - Mät valfritt objekt och spara din mätning till mobilen:

Tid: 10 s.

Fritext: Hon mätte rätt men tvekade lite innan hon tryckte rätt sparaknapp. Däremot hade hon kameran ner i marken.

Uppgift 4: Ta bort alla punkter:

Tid: 10s.

Fritext: Hon använde bakåtknappen för att ta bort allt.

Uppgift 5: Spara miljön till databas:

Tid: 10 s.

Fritext: Hon var lite förvirrad vad som menades med frågan men gick in i menyn och tryckte rätt.

Utvärdering:

- Hon tycker att skulle vara mycket användbart med applikationen, visste inte att det fanns sådana i dagens läge.
- Inget var dåligt.

Användartest 5

Ålder: 56 år

Kön: Man

Teknisk färdighet (1-5) : 3

Uppgift 0 - Få upp crosshair (nämns inte):

Tid: Direkt.

Instruktioner: Inga.

Uppgift 1 - Mät en sida på ett bord:

Tid: 60 s.

Fritext: Han ville klicka på hårkorset på kanten av bordet flera gånger. Efter några försök att klicka och zooma vid hårkorset använde han rätt knapp.

Uppgift 2 - Ta bort senaste punkten:

Tid: 20 s.

Fritext: Gick in i menyn och hittade remove all points, klickade ut ur menyn och tryckte rätt.

Uppgift 3 - Mät valfritt objekt och spara din mätning till mobilen:

Tid: 200+ s.

Fritext: Applikationen kraschade och då fick han inte upp hårkorset. Problem uppstod med att hitta rätt plan och få punkterna att fastna. Testade att gå in i menyn och klickade först på databas sparande. Vid andra mätningen klickades spara till kamerarulle.

Uppgift 4: Ta bort alla punkter:

Tid: Direkt.

Fritext: Gick in i menyn då han sett ikonen innan och tryckte remove all points.

Uppgift 5: Spara miljön till databas:

Tid: Direkt.

Fritext: Eftersom han varit inne i menyn innan visste han att han skulle klicka upload to database. Däremot samlade han inte in data om miljön och hade kameran ner i bordet.

Utvärdering:

- Suddgummi-symbol för att ta bort ett visst streck.

Användartest 6

Ålder: 52.

Kön: Kvinna

Teknisk färdighet (1-5) : 3.

Uppgift 0 - Få upp crosshair (nämns inte):

Tid: Direkt.

Instruktioner: Inga.

Uppgift 1 - Mät en sida på ett bord:

Tid: 120 s.

Fritext: Vände till liggande orientering. Hon siktade på i mitten av sidan av bordet och antog att appen skulle mäta sidan direkt. Hon klickade på add som var tänkt men eftersom hon bara satt en punkt i mitten av bordet kom hon inte vidare. Hon försökte ändra men fick inte till det utan fick använda bakåtknappen för att ta bort allt. Hon satt en punkt i mitten av långsidan igen och fick trixa till hon förstod att hon skulle sätta en punkt i varje ände. Ville ofta flytta de utsatta punkterna med fingrarna.

Uppgift 2 - Ta bort senaste punkten:

Tid: Direkt.

Fritext: Inga problem.

Uppgift 3 - Mät valfritt objekt och spara din mätning till mobilen:

Tid: 220 s.

Fritext: Sparandet gick väldigt bra. Hon använde rätt kapp direkt. Däremot var det svårt att mäta telefonboxen på bordet för att hon var väldigt nära och enheten hittade inte korrekt plan. Detta gav förvirring då punkterna inte stannade på rätt position så det tog lång tid att så till korrekt placering.

Uppgift 4: Ta bort alla punkter:

Tid: Direkt.

Fritext: Använde bakåtknappen för att ta bort allt.

Uppgift 5: Spara miljön till databas:

Tid: 10 s.

Fritext: Ville spara med "spara till kamerarulle". När detta inte gick satte hon punkt och testade att trycka samma knapp igen. Till slut när det nämnts att metoden var fel och det kanske fanns något annat sätt gick hon in i menyn och hittade rätt.

Utvärdering:

- Spara var solkort.
- Det var svårt att få punkterna att fastna.
- Det kanske ska vara skillnad på startpunkt och slutpunkt.
- En liten kort film kan vara bra för att visa hur man ska göra.
- Vill ha möjlighet att ha liggande.
- Dra punkter!
- Vill mäta objekt direkt med ett knapptryck, typ rektanglar.

Användartest 7

Denna användare har använt Apples egna mätverktyg tidigare.

Ålder: 22 år.

Kön: Man.

Teknisk färdighet (1-5) : 5.

Uppgift 0 - Få upp crosshair (nämns inte):

Tid: Direkt.

Fritext: Inga.

Uppgift 1 - Mät en sida på ett bord:

Tid: Direkt.

Fritext: Gick utan problem.

Uppgift 2 - Ta bort senaste punkten:

Tid: Direkt.

Fritext: Inga problem.

Uppgift 3 - Mät valfritt objekt och spara din mätning till mobilen:

Tid: 10 s.

Fritext: Håarkorset försvann lite när han skulle mäta en yta om inte var plan men det kom tillbaka och objektet kunde ändå mätas skapligt.

Uppgift 4: Ta bort alla punkter:

Tid: Direkt.

Fritext: Använde bakåtknappen för att ta bort allt.

Uppgift 5: Spara miljön till databas:

Tid: 10 s.

Fritext: Han gick in i menyn och gick ur igen. Efter han tänkt lite gick han in igen och tryckte rätt.

Utvärdering:

- Inget strul att använda applikationen.
- Vill ha snapfunktion.

Bilaga B

Individuella bidrag

Johan Forslund *Versionsansvarig*

Ansvarade för versionshanteringen, implementerade databasen, skapade funktionalitet för att mäta i sparat rum, utvecklade 3D-grafik (mätpunkter, text, förstorningsglas), löste vissa problem relaterade till linjär algebra, hanterade trådning och asynkrona problem.

Fredrik Johansson *Dokumentansvarig*

Förde anteckningar under möten, ansvarade för gruppens dokumentation av kod, implementerade 3D-grafik och funktionalitet för linjer samt borttoningseffekter, implementerade meny och navigering mellan aktiviteter, utvecklade gränssnitt och aktivitetsindikatorer.

Pontus Arnesson *Scrum-mästare och designansvarig*

Ansvarade för att möten hölls enligt Scrum, skapade aktiviteten SelectRoomActivity med dess funktioner och gränssnitt, skapade och implementerade den rörliga gif-filen, skapade iconer och bilder till gränssnittet, implementerade 3d-grafik för hårkors och dess funktionalitet.

Fredrik Norrstig *Scrum-mästare och testningsansvarig*

Ansvarade för att möten hölls enligt Scrum och såg till att tester gjordes. Var med och implementerade logiken av hur punkter sätts ut i miljön och dess implementation. Implementerade snap-funktion av linjen och hårkors, mest tekniska delen och lite grafiskt. Implementation av hur avstånd räknas ut.

Vera Fristedt Andersson *Produktägare*

Ansvarade för kundkontakt, hantering av produktbackloggen och att produktens värde maximerades. Organiserade även gruppens arbete och såg till att tidsplanen hölls. Arbetade främst med implementation av mätpunkterna, mättexten, onboardingen, spara bild till kamerarulle och hur koden kring mätpunkterna skulle byggas upp.