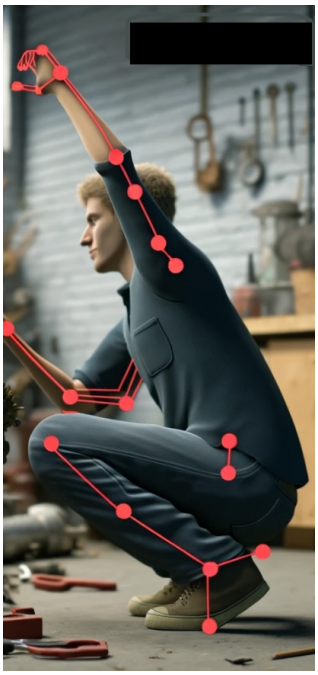


Mobilapplikation för videoinspelning med användning av standardbibliotek för maskininlärning

Balansering och ergonomiska hänsyn vid monteringsband

Examensarbete inom högskoleprogrammet Datateknik

Odai Alrahem
Per Tenggren

INSTITUTIONEN FÖR DATA- OCH INFORMATIONSTEKNIK

CHALMERS TEKNISKA HÖGSKOLA

GÖTEBORGS UNIVERSITET

Göteborg 2024

www.chalmers.se

Examensarbete 2024

Mobilapplikation för videoinspelning med användning av standardbibliotek för maskininlärning

Balansering och ergonomiska hänsyn vid monteringsband

ODAI ALRAHEM
PER TENGGREN

GÖTEBORGS UNIVERSITET

CHALMERS

Institutionen för data- och informationsteknik
Chalmers Tekniska Högskola
Göteborgs Universitet
Göteborg 2024

Mobilapplikation för videoinspelning med användning av standardbibliotek för maskininlärning
Balansering och ergonomiska hänsyn vid monteringsband
ODAI ALRAHEM
PER TENGGREN

Handledare: Jonas Almström Duregård, Institutionen för data- och informationsteknik
Examinator: Lars Svensson, Institutionen för data- och informationsteknik

Examensarbete 2024
Institutionen för Data- och informationsteknik
Chalmers Tekniska Högskola
SE-412 96 Göteborg
Telefon +46 31 772 1000

Omslagsbild: Omslagsbilden visar ett arbetsmoment där olika kroppsdelar är utmärkta. Bilden är genererad i ChatGPT 4.0.

Skriven i LaTeX
Göteborg 2024

Mobilapplikation för videoinspelning med användning av standardbibliotek för maskininlärning
Balansering och ergonomiska hänsyn vid monteringsband

ODAI ALRAHEM
PER TENGGREN
Institutionen för Data- och informationsteknik
Chalmers Tekniska Högskola
Göteborgs Universitet

Sammanfattning

Detta examensarbete syftar till att undersöka om standardbibliotek för maskininlärning kan användas i mobilapplikationer vars syfte är att hjälpa till vid linjebalansering och tidsstudier. Studien är ett samarbete med Solme AB.

Arbetet utvärderar olika maskininlärningsbibliotek som kan användas för ansiktsidentifiering för anonymisering av inspelade personer. Vidare ska kroppsrörelseidentifikation användas för att identifiera ergonomiskt utmanande positioner. Objektidentifikation används för att identifiera specifika delmoment i arbetsprocessen. Syftet ska också att utvärdera om det var möjligt att utveckla applikationen med gemensam kod för iOS och Android.

Resultaten visar att tekniker för maskininlärning kan implementeras framgångsrikt i mobila miljöer, men att kraven på en plattformsoberoende kod och användande av standardbibliotek begränsar funktionaliteten avseende hastighet, och möjligheten att via objektidentifikation identifiera olika delmoment. Avslutningsvis diskuteras framtida utveckling av applikationen samt miljö och etiska överväganden.

Abstract

This degree project report investigates whether standard library for machine learning can be used in mobile applications designed to assist with line balancing and time studies. The study is conducted in collaboration with Solme AB.

The work evaluates various machine learning libraries for facial recognition aimed at anonymizing recorded individuals, body movement identification to detect ergonomically challenging positions, and object identification to recognize specific subtasks. Furthermore, the aim was to assess the feasibility of developing the application with a common codebase for both iOS and Android.

The results indicate that machine learning techniques can be successfully implemented in mobile environments. However, the requirements for cross-platform code and standard libraries limit functionality in terms of speed and the ability to identify different sub-tasks through object identification.

Finally, the future development of the application, as well as environmental and ethical considerations, are discussed.

Nyckelord: mobilapplikation, maskininlärning, plattformsoberoende, objektdetektering, kroppsrörelsedetektering, linjebalansering

Förord

Detta examensarbete har utförts som en del av högskoleingenjörsprogrammet Datateknik vid Chalmers Tekniska Högskola och har gett oss en unik chans att applicera teoretiska kunskaper i praktiken, inom området mobilapplikationsutveckling och maskininlärning.

Vi vill först framföra ett stort tack till vår handledare, Jonas Almström Duregård, vars vägledning och expertkunskaper har varit ovärderliga genom hela processen. Hans engagemang och konstruktiva feedback har varit avgörande för att vi skulle kunna navigera projektets utmaningar och nå våra mål. Tack också till vår examinator Lars Svensson och vår oppositionsgrupp Alexander Däldborg och Erik Palm för värdefull återkoppling.

Ett särskilt tack till Niklas Axelsson, vår handledare hos Solme AB, som med sitt engagemang och sina insikter har bidragit till vår professionella utveckling. Niklas har försett oss med de verktyg som behövdes för att effektivt hantera komplexa tekniska frågor och möjliggjort en djupare förståelse för den praktiska tillämpningen av våra studier.

Vi är också tacksamma för den hjälp vi fått från Oskar Ljung, Marcus Rosell och Claes Rosell, som generöst delade med sig av sin kunskap och hjälpte oss att se det större sammanhanget i projektet. Deras råd och stöd har varit ovärderliga.

Ett tack även till alla på Solme AB för ett fantastiskt samarbete. Er öppenhet och stöd har varit fundamentalt för att ge oss en realistisk inblick i branschens utmaningar och möjligheter.

Slutligen, ett varmt tack till våra familjer och vänner för deras ständiga stöd och uppmuntran. Utan er hade denna resa varit mycket svårare.

Odai Alrahem och Per Tenggren, Göteborg, maj 2024

Akronymer

Nedan återfinns en lista över de beteckningar som används i examensarbetet ordnade i alfabetisk ordning:

API	Application Programming Interface (Applikationsprogrammeringsgränssnitt)
AWS S3	Amazon Web Services Simple Storage Service
CNN	Convolutional Neural Networks
codec	Coder Decoder
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment (Integrerad utvecklingsmiljö)
JPEG	Joint Photographic Experts Group
JSON	JavaScript Object Notation
ML	Machine Learning
REST	Representational State Transfer
SSD	Single Shot MultiBox Detector

Innehåll

Figurer

Tabeller

1
Inledning

1.1 Bakgrund

Uppdragsgivaren Solme AB, tillhandahåller en mjukvarulösning, AVIX, som deras kunder bland annat använder för metod- och tidsstudier, processdokumentation och så kallad linjebalansering.

AVIX är "...ett komplett systemstöd för att förbättra företags lönsamhet genom att knyta samman och effektivisera det produktionstekniska arbetet inom produktionsoptimering, produktionseffektivitet, produktutveckling och producerbarhet" [1].

Kundernas analysarbete sker idag huvudsakligen genom att arbetsmomenten dokumenteras med hjälp av en traditionell videoutrustning där resultatet sedan vidareförädlas i AVIX. Solmes kunder efterfrågar möjlighet att utnyttja vanliga mobiltelefoners videoinspelningsmöjlighet och att kunna använda mer automatisering av olika moment genom artificiell intelligens (AI).

Den tekniska utvecklingen har gett tillgång till relativt enkla mobiltelefoner med avancerade videoupptagningsmöjligheter. Företaget är därför intresserade av att undersöka om videoupptagningen istället kan ske med en mobiltelefon som i sin programvara dessutom förbereder en del av analysarbetet genom att markera olika händelser.

1.2 Syfte

Undersöka om det går att utveckla en mobilapplikation som använder en vanlig mobiltelefon för att ta upp en video över ett antal arbetsmoment. Vidare ska olika delar av videon markeras genom att applikationen använder sig av standardbibliotek för maskininlärning.

1.3 Mål och förväntat resultat

Målet är att ta fram ett principbevis (proof of concept") för att det är möjligt att använda tillgängliga standardbibliotek för maskinlärning för att skapa en videoapplikation för mobiltelefoner med videoinspelningsmöjligheter, där olika moment i en arbetsprocess markeras.

Förväntat resultat är en fungerande applikation för Android och iOS som kan spela in en video på en person som utför olika arbetsmoment. Projektet ska svara på dessa frågor:

Är det möjligt att skapa en applikation som fungerar för både iOS och Android med nedanstående funktionalitet?

- Är det möjligt att använda standardbibliotek för att:
  - a) skapa oskarp mask över ansikte,
  - b) markera rörelser med så kallad rörelsedetektion
  - c) särskilja monteringscykler genom att identifiera förutbestämda objekt
- Är det möjligt att markera ovanstående i videon i realtid eller behövs ett särskilt moment på en sparad video?
- Är det möjligt att överföra video och information om rörelser och monteringscykler till AVIX.

1.4 Avgränsningar

Följande avgränsningar är gjorda som en konsekvens av uppdragsgivarens uppdrag och för att projektet ska kunna slutföras inom avsatt tid:

- Applikationen ska endast att utnyttja befintliga standardbibliotek och inte använda sig av maskinlärning som måste tränas av kunden eller uppdragsgivaren.
- Applikationen kommer att utvecklas så att en enda rörelse identifieras, exempelvis händerna över axelhöjd.
- På samma sätt kommer endast ett objekt att identifieras för att särskilja monteringscykler.

2
Teknisk bakgrund

Här beskrivs de tekniska förutsättningarna för att besvara de inledande frågeställningarna.

2.1 Applikation för iOS och Android

För att utveckla en applikation som fungerar på både iOS och Android finns det två huvudsakliga tillvägagångssätt. Antingen utvecklar man en separat applikation för varje plattform, så kallad plattformsberoende (native) utveckling, eller så väljer man att utveckla en applikation som fungerar på båda plattformarna med samma kodbas, så kallad plattformsoberoende (cross-plattform) utveckling [2].

Applikationen utvecklas sedan i en integrerad utvecklingsmiljö (IDE) som innehåller en redigerare för källkod, en kompilerare som översätter programmeringsspråkets källkod till maskinkod och en debugger för att hitta eventuella fel i programkoden.

2.2 Dart och Flutter

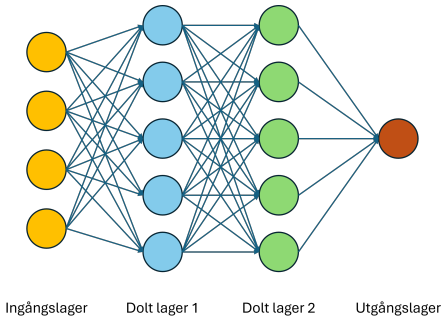
Dart är ett programmeringsspråk [3] som utvecklades av Google och är designat för att bygga strukturerade webb-, server- och mobilapplikationer. Dart är ett objektorienterat, klassbaserat språk som framförallt används tillsammans med Flutter [4], Googles UI-ramverk för att utveckla applikationer för iOS, Android, webb och desktop från en enda kodbas.

2.3 Standardbibliotek

Standardbibliotek för maskininlärning (ML) i mobiltelefoner är programvarubibliotek som tillhandahåller verktyg, modeller och applikationsprogrammeringsgränssnitt (API) för att utveckla och implementera ML-funktioner direkt på mobila enheter. Dessa bibliotek är utformade för att vara optimerade för mobil hårdvara och operativsystem, och de möjliggör utvecklingen av ML-drivna applikationer som kan köras offline och i realtid på mobilenheten. Exempel på standardbibliotek är TensorFlow Lite, Google ML Kit och Core ML. Dessa beskrivs närmare längre fram i rapporten.

2.3.1 Convolutional Neural Network

Ett Convolutional Neural Network (CNN) är en typ av artificiellt neuralt nätverk [5] som är särskilt bra på att hantera bilddata. Den fungerar genom att använda olika lager som var och ett har specifika uppgifter och stegvis extraheras och sammanfattas mönster från en bild. Resultatet används sedan för att klassificera eller analysera. Den principiella uppbyggnaden av CNN finns i Fig. 2.1.



Figur 2.1: Principiell och förenklad uppbyggnad av CNN

Ingångslageret tar emot den råa bilden som input, ofta representerad som en matris av pixelvärden. I faltningslager används filter (eller kärnor) för att extrahera lokala mönster från bilden. Filtrerna sveper över bilden och skapar faltningsoperationer som genererar aktiveringskartor eller ”feature maps”. Varje filter kan identifiera olika mönster, som kanter eller texturer.

Efter faltningslagren tillämpas en icke-linjär aktiveringsfunktion, vanligtvis ReLU (Rectified Linear Unit), för att introducera icke-linearitet i modellen vilket hjälper nätverket att lära sig komplexa mönster.

För att minska dimensionerna på aktiveringskartorna och därmed minska beräkningskostnader och överanpassning, används poolinglager (ofta max-pooling). Poolinglagret tar ut det maximala värdet från en region i aktiveringskartan, vilket bevarar viktiga egenskaper samtidigt som dimensionerna reduceras.

De reducerade och bearbetade aktiveringskartorna plattas ut till en vektor och skickas genom ett eller flera fullt anslutna lager. Dessa lager fungerar som vanliga neurala nätverk och här görs slutliga beslutsfattanden.

Det sista lagret i en CNN är ett fullt anslutet lager med en aktiveringsfunktion som softmax (för klassificeringsuppgifter) som genererar sannolikhetsfördelningen över de möjliga klasserna.

2.3.2 Single Shot MultiBox Detector

Single Shot MultiBox Detector (SSD) är en algoritm för objektdetektering [6] som fungerar i ett enda framåtpass av det neurala nätverket, vilket gör att den snabbt kan upptäcka objekt i bilder. Till skillnad från metoder som involverar separata steg för att generera förslag på områden som innehåller objekt och därefter klassificera dem, förutsäger SSD samtidigt flera begränsningsrutor och klassannolikheter för objekt i dessa rutor på en gång. Detta integrerade tillvägagångssätt gör det möjligt

för SSD att uppnå höga detekteringshastigheter och gör den särskilt väl lämpad för mobilapplikationer.

2.3.3 FFmpegKit

FFmpegKit är ett multimediaramverk [7] med öppen källkod som kan avkoda, koda, omkoda, muxa (sammanföra flera signaler till en kanal), demuxa (dela upp sammaförda signaler i de ursprungliga signalerna), streama, filtrera och hantera många typer av media. Den stöder ett brett utbud av filformat och så kallade codecs (coder/decoder), vilket är användbart inom video- och ljudbearbetningsindustrin. De funktioner som används i detta arbete är möjligheten att dela upp en videofil i ett antal bildrutor i form av JPEG-filer samt att sätta samman ett antal bildrutor till en videofil.

2.3.4 JavaScript Object Notation

JavaScript Object Notation (JSON) är ett språkoberoende format som är designat för att underlätta datautbyte mellan olika system. Det är textbaserat och enkelt att tolka både för datorer och människor, vilket gör det mycket användbart för applikationsintegration [8]. JSON-formatet har sitt ursprung i JavaScript, men dess användning sträcker sig långt utöver JavaScript-utveckling och stöds av de flesta moderna programmeringsspråk.

JSON byggs upp av två delar:

- Nyckel/värdepar: Dessa representeras vanligtvis som objekt i programmeringsspråk, där varje nyckel (en sträng) mappas till ett värde. I JSON omsluts dessa par av klammerparenteser och separeras av kommatecken, medan nyckel och värde separeras av kolon.
- Ordnade listor av värden/arrays.

Ett värde inom JSON kan vara en sträng, ett tal, ett booleskt värde, null, ett objekt (en samling av nyckel/värdepar), eller en array (en lista av värden). JSON tillåter att objekt och arrays nästlas inuti varandra, vilket gör det möjligt att representera komplexa datastrukturer.

2.4 Amazon Web Services Simple Storage Service

Amazon Web Services Simple Storage Service (AWS S3, eller bara S3) är en objektslagringstjänst [9] som erbjuder skalbarhet, datatillgänglighet, säkerhet och prestanda. Den ger användare möjlighet att lagra och hämta vilken mängd data som helst, från vilken plats som helst på internet. Användare kan lagra dokument, bilder, videor och mer i så kallade ”buckets” (hinkar), som fungerar som grundläggande behållare för data. S3 är utformad för att erbjuda hög hållbarhet och har omfattande säkerhets- och efterlevnadsfunktioner som möjliggör skapandet av anpassade regelverk. Tjänsten används ofta för säkerhetskopiering och återställning, datalagringsarkiv, webbplatser och mobilapplikationer samt för att stödja olika dataanalyser.

3
Metod

I metodavsnittet beskrivs de moment som planerades för att kunna svara på de frågor som ställdes inledningsvis.

3.1 Inledande litteraturstudier

Genom en inledande litteraturstudie togs underlag fram för att bestämma vilken utvecklingsmiljö som var lämpligast för applikationen samt vilket standardbibliotek som var mest passande att använda.

Vid valet av utvecklingsmiljö utvärderades olika alternativ huvudsakligen baserat på utvecklingskostnader, komplexiteten i att underhålla kodbasen och tiden till marknad. På en övergripande nivå stod valet mellan att utveckla den mobila applikationen med separata kodbaser för iOS och Android, eller med en gemensam kodbas.

I nästa steg undersöktes vilka standardbibliotek som fanns tillgängliga. Dessa utvärderades utifrån eventuella begränsningar, hur väl de fungerar med de möjliga utvecklingsmiljöerna (IDE) samt hur väl de kunde bidra till att besvara de inledningsvis ställda frågorna.

3.2 Realtid eller i efterhand

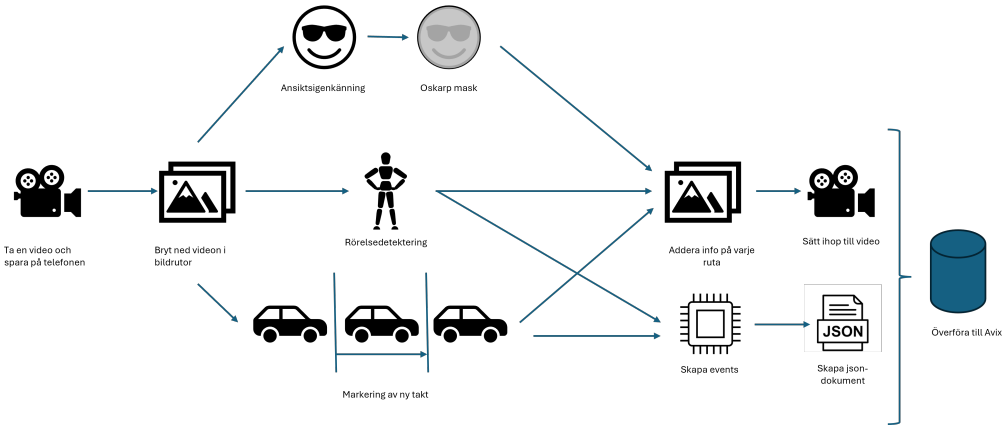
Applikationen kunde antingen lägga på ansiktsmaskering, rörelse- och objektdetektion direkt i realtid, eller genomföra dessa operationer i efterhand på en inspelad videofil. Här övervägdes olika parametrar för att fatta beslut om vilken väg som skulle väljas:

- Vilket programstöd finns för att hantera operationerna i realtid, både vad gäller det programmeringsspråk som används såväl som de modeller som implementeras.
- En bedömning av hur operationerna påverkar mobiltelefonens övriga funktioner under tiden samt hur lång tid det tar att genomföra de olika operationerna.
- Finns det fördelar för uppdragsgivaren att välja hantering i realtid eller i efterhand?

3.3 Genomförandemodell

För att kunna lösa uppgiften delades projektet in i ett antal delmoment som beskrivs i Fig. 3.1 nedan. En video spelades in på mobiltelefonen varpå en oskarp mask lades på ansikten, rörelser detekterades samt nya objekt identifierades. Som beskrivits ovan kunde detta antingen ske i realtid eller i efterhand på inspelad video. I det senare fallet behövde videon först delas upp i enstaka bildrutor för att sättas ihop till en video efter det att samtliga delmomenten genomförts.

Parallellt sparades information om kroppsställningsdetektion och objektdetektion i ett gemensamt JSON-dokument med tidskoder och beskrivning av händelserna.



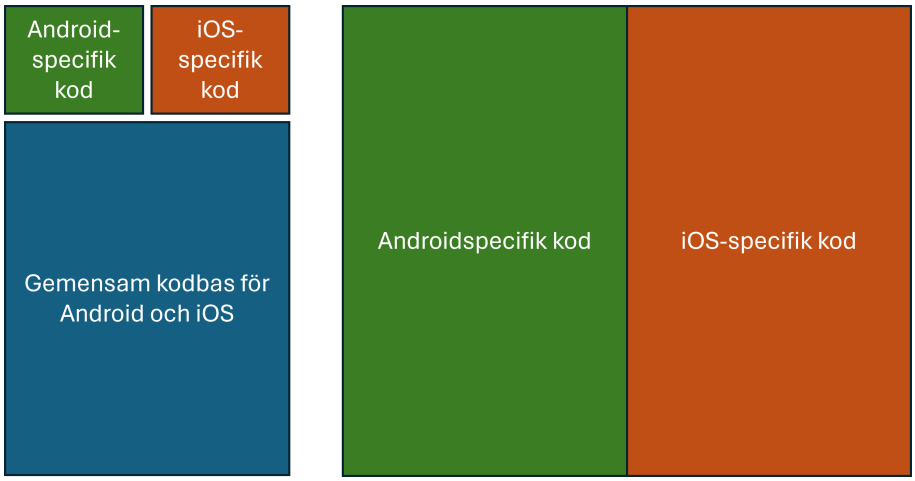
Figur 3.1: Principiell uppbyggnad av systemet

Genomförandet skulle komma att resultera i en eller två separata kodbasstrukturer, beroende på om koden utvecklades plattformsoberoende eller plattformsspecifikt vilket visas i Fig. 3.2. Plattformsoberoende innebar att så mycket av kodbasen som möjligt skulle utvecklas för både iOS och Android och där ett minimum av plattformsspecifik kod används. Plattformsspecifik kod syftade till att optimera koden för respektive operativsystem och här skulle resultatet bli två i princip helt separata kodstrukturer.
Som genomförandemetod valdes vattenfallsmodellen [10] eftersom projektets olika delar var få och väldefinierade. Färdigställd kod sparades på ett gemensamt github-konto.

3.4 Användning av standardbibliotek för ML

För att kunna anonymisera ansikten i videon behövde en metod skapas som först identifierade ansikten och därefter lade en oskarp mask över dem. Det var nödvändigt att först avgöra vilken algoritm som skulle användas för att detektera ansikten och därefter vilken metod som skulle användas för att lägga på en oskarp mask.

I nästa steg implementerades kroppsställningsdetektion, där syftet var att identifiera människokroppar i inspelad video och därefter extrahera specifika punkter på kropparna, såsom armbågar, knän och handleder. Dessa punkter, även kallade nyckelpunkter"eller landmärken", hade x- och y-koordinater som representerade deras



Figur 3.2: Principiell uppbyggnad av systemet vid plattformsoberoende respektive plattformsspecifik hantering av kodbas

position i den digitala bilden eller videon. Koordinaterna kunde sedan användas för att analysera kroppsställningen och rörelserna hos en person samt bedöma om olika rörelser kunde anses vara ergonomiskt utmanande. Ett exempel på en sådan rörelse var om man arbetar med händerna ovanför axlarna.

Avslutningsvis behövde gruppen lösa hur olika delmoment i ett löpande band skulle kunna särskiljas, exempelvis när monteringen av en viss del påbörjas och när den är klar. Metodmässigt implementerades denna funktion så att applikationen skulle kunna identifiera när olika objekt dyker upp i videon. Den ML-algoritm som valdes för detta ändamål skulle kunna ställas in för att identifiera ett visst antal förbestämda objekt, exempelvis en bil, en skruvdragare eller en hammare. En alternativ lösning som diskuterades var att låta personen som spelar in videon manuellt ange när ett delmoment startar eller när en ny monteringscykel börjar.

3.5 Skapa JSON-fil och överföring till AVIX

Ett JSON-dokument med tidskoder och beskrivning av olika händelser skulle tas fram. Filen med informationen skulle genereras av klasser som hanterade kroppställnings- och objektdetektion.

Användaren skulle kunna lista samtliga video- och JSON-filer, markera vilka som skulle överföras och därefter överföra dessa via AWS S3, vilket var ett önskemål från uppdragsgivaren, till en server kopplat till AVIX.

3.6 Testning

När applikationen var färdigställd testades den dels för att säkerställa att alla delar fungerade på en Android-telefon, dels enligt ett framtaget testschema. Testschemat inkluderade videofiler av olika längd och med varierande antal bilder per sekund, varpå olika storheter, såsom filens storlek, noterades. Om filen genererades i efterhand, noterades även tidsåtgången för genereringen av videofilen.

4
Genomförande

I detta kapitel utvecklas beskrivning av hur projektet genomfördes utifrån den metodmässiga diskussionen i föregående kapitel.

4.1 Inledande litteraturstudier

För att utvärdera vilken IDE som skulle användas behövde det först avgöras om applikationen skulle utvecklas plattformsspecifikt eller plattformsoberoende. I Tabell 4.1. nedan listas för- och nackdelar mellan de två principerna [11].

Aspekt	**Plattformsspecifik**	**Plattformsoberoende**
Prestanda	Högre, optimerad för specifika plattformar.	Kan vara lägre, särskilt för resurskrävande applikationer.
Tillgång till APIer	Full tillgång till operativsystemets APIer och funktioner.	Begränsad tillgång, beroende på ramverket.
Användarupplevelse	Kan skräddarsys för varje plattforms designriktlinjer.	Kan vara utmanande att matcha plattformsspecifik UX/UI.
Kostnad	Högre, kräver separata utvecklingsteam eller kunskap.	Lägre, en enda kodbas för flera plattformar.
Tid till Marknaden	Längre, separata kodbaser för varje plattform.	Snabbare, tack vare delad kodbas.
Underhåll	Mer tidskrävande och komplicerat med flera kodbaser.	Enklare, en enda kodbas för alla plattformar.
Återanvändbar Kod	Begränsad mellan plattformarna.	Hög, stora delar av koden kan återanvändas över plattformar.
Funktionalitet	Ingen begränsning, kan utnyttja plattformens fulla potential.	Vissa avancerade funktioner kan vara svåra att implementera.
Beroende	Beroende av operativsystemets utvecklingscykler.	Beroende av tredjepartsramverkets uppdateringar och stöd för nya funktioner.
Anpassningsbarhet	Hög, möjlighet till djupgående anpassning och optimering.	Lägre, vissa anpassningar kan vara begränsade av ramverket.

Tabell 4.1: Jämförelse mellan plattformsspecifik och plattformsoberoende utveckling

Av ovanstående sammanställning noterades att en plattformsoberoende hantering

hade fördelarna att utvecklingskostnaden är lägre, underhållet enklare och tid till marknad kortare. Detta var fördelaktigt utifrån antagandet att applikationen inte kommer att vara en huvudprodukt, utan snarare ett tillägg till företagets tjänstekatalog. I det fortsatta arbetet utgick gruppen därför ifrån principen att så långt som möjligt arbeta med en gemensam kodbas.

I Tabell 4.2 beskrivs för- och nackdelar med olika val av program att utveckla applikationen i [12] [13]. Eftersom målet var en applikation med endast en kodbas, noterades att Flutter hade fördelar vad gäller gemensam kodbas och prestanda.

Utvecklings-miljö	**Fördelar**	**Nackdelar**
Xamarin	C# och .NET ekosystemet, vilket kan vara fördelaktigt för de som redan är bekanta med dessa. Stöd för både Android och iOS vilket minskar mängden plattformsspecifik kod. Bra tillgång till enhetshårdvara för funktioner som videohantering.	Prestandan kan vara sämre jämfört med native kod, speciellt för grafikintensiva appar. Mindre gemenskap och stöd jämfört med andra plattformar.
React Native	Möjlighet att använda JavaScript, vilket är populärt och har en stor utvecklargemenskap. Snabbladdning underlättar snabbare utvecklingscykler. Komponentbaserad arkitektur som kan underlätta återanvändning av kod.	Prestandaproblem för mer komplexa och grafikintensiva appar. Behovet av native moduler för avancerad funktionalitet kan komplicera utvecklingen.
Flutter	Hög prestanda tack vare Dart och kompilering till native kod. Rik uppsättning inbyggda widgets och ett kraftfullt UI-ramverk. Stöd för både Android och iOS från en enda kodbas.	Dart är mindre känt och har en mindre utvecklargemenskap jämfört med JavaScript. Större appstorlekar jämfört med andra lösningar.

Tabell 4.2: Jämförelse mellan olika utvecklingsmiljöer

Att utveckla en applikation som kan maskera ansikten, lägga till rörelsedetektion och detektera objekt kräver användning av specialiserade bibliotek eller ramverk. Eftersom dessa uppgifter ofta utförs med hjälp av maskininlärningsmodeller, bör man välja bibliotek som har bra stöd för maskininlärning och bildbehandling. I Tabell 4.3 har vi ställt upp för och nackdelar med några allmänt förekommande bibliotek [14].

Bibliotek/Ramverk	Fördelar	Nackdelar
TensorFlow Lite	- Optimerad för mobila enheter. - Stort utbud av förtränade modeller. - Aktivt community och stöd - Kan användas med både Flutter och React Native.	- Kräver kunskap om TensorFlow och maskininlärning. - Kan vara resurskrävande på mobila enheter.
ML Kit	- Enkelt att komma igång med. - Integrerar väl med Firebase som ärGoogles utvecklingsplattform för mobil- och webbapplikationer. - Erbjuder både på API på enheten och molnbaserad. - Bra dokumentation och stöd.	- Begränsad till Googles egen plattform och modeller. - Kan vara mindre flexibelt jämfört med TensorFlow Lite.
OpenCV	- Kraftfullt för bild- och videobearbetning. - Stort utbud av funktioner för datorseende. - Stöder realtidsbearbetning. - Kan användas med crossplatform ramverk.	- Större inlärningströskel. - Inte lika optimerad för mobilanvändning som ML Kit.
Core ML (iOS)	- Optimerad för iOS-enheter. - Enkel integration med iOS-appar. - Stödjer en mängd olika modeller - Bra prestanda och effektivitet.	- Endast för iOS, vilket innebär att en separat lösning behövs för Android. - Mindre community jämfört med TensorFlow.

Tabell 4.3: Jämförelse av bibliotek och ramverk för cross-platform apputveckling

I ovanstående sammanställning noterades att Core ML innebar att målsättningen att ha en gemensam kodbas för applikationen måste överges. Av de tre biblioteken som återstod valdes ML Kit i första hand eftersom detta bibliotek hade modeller som täckte de behov av ansikts-, kroppsställnings- och objektdetektion som projektet krävde.

4.2 Realtid eller i efterhand

För att avgöra om applikationen skulle utvecklas så att beräkningar görs i realtid eller i efterhand gjordes följande överväganden:

- Några fördelar med att genomföra alla beräkningar i realtid har inte kunnat identifieras.
- Enklare ML-modeller behöver ingen internetuppkoppling för att kunna utföras.

Framtida utveckling kan dock komma att kräva internetuppkoppling. Därför finns det en fördel att redan från början separera inspelningen av videon och de beräkningar som kräver internetuppkoppling.

- Att i realtid spela in en video samtidigt som applikationen ska utföra ansikts-, kroppsställnings- och objektdetektion ställer höga krav på den mobiltelefon som ska användas.
- Av prestandaskäl kommer det att vara nödvändigt att minska antalet bilder per sekund för att systemet ska hinna med alla beräkningar varför videouppspelningen kan bli hackig.

De två sista punkterna avseende applikationens prestanda undersöks i de tester som beskrivs längre fram i rapporten.

4.3 Hantering av videofil

För att kunna hantera inspelad videofil delades videon upp i ett antal bildrutor genom att använda FFmpegKit. Innan uppdelningen sker, kontrolleras att videofilen existerar. I koden kan också anges hur många bilder per sekund som ska genereras. Bilderna sparas i ett tillfälligt bibliotek i formatet JPEG. Efter detta kan man arbeta vidare med de klasser som anaonymiseras via oskarp mask, identifierar kroppsställning och detekterar objekt.

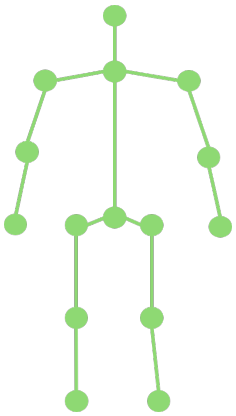
En FaceDetector-instans skapas inledningsvis med syftet att anonymisera ansikten med en oskarp mask. Därefter fylls en lista på med sökvägar till alla bilderrutor. Listan gås igenom och för varje bild analyseras den genom via ”Face Detection” i ML Kit. Om ett ansikte identifieras presenteras det som en rektangel med en viss position och storlek. Rektangeln klipps ur bilden, på urklippet appliceras gaussisk oskärpa varpå urklippet läggs tillbaka i bildfilen och bilden läggs tillbaka i det ursprungliga biblioteket. Slutligen, när klassen inte längre behövs, kan ansiktsdetektorn frigöras och stängas för att frigöra resurser via metoden ”dispose”. Efter det att ansikten anonymiseras vidtar kroppsställnings- och objektdetektion.

För att identifiera olika nyckelpunkters x- och y-koordinater användes ML Kit Pose Detection. Den använda modellen skapar för varje bildruta specifika nyckelpunkter med x- och y-värden i ett koordinatsystem. Ett exempel visas i Fig. 4.1 Dessa punkter används sedan för att identifiera utvalda rörelser.

Olika rörelser, som bestäms genom ett regelverk, sparas som händelser i applikationen och definieras som förändringar i koordinater. Händelsen ”Arbeta över axelhöjd” skapas exempelvis genom att beräkna om någon av händernas koordinater befinner sig över någon av axlarnas koordinater. På samma sätt kan en händelse, arbetar böjd, skapas om någon av handkoordinaterna befinner sig under knäkoordinaterna. Händelser och tidpunkter sparas undan i en JSON-fil.

För att identifiera objekt användes ML Kit Object Detection". När ett förbestämt objekt identifieras skapas en händelse. Händelsen och tidskoden sparas i ett JSON-dokument. Vilka objekt som kan detekteras bestäms av den förtränade modellen.

Efter att dessa metoder har avslutats, ska de bildrutor som applikationen har bearbetat sättas samman till en video.



Figur 4.1: Exempel på identifierade nyckelpunkter

Figur 4.2: Exempel på identifiering av objekt på löpande band

4.4 Sätta samman till video

Metoden börjar med att säkerställa att det finns bildrutor att återsamla. Dessa bildrutor antas vara JPEG-filer i en temporär mapp och har en del av filnamnet som indikerar att de är ”frames”. Därefter itererar metoden över listan med sökvägar till bildfilerna och namnger filerna för att säkerställa att de har en konsekvent och ordnad numrering. Detta är viktigt för att FFmpeg korrekt ska kunna tolka ordningsföljden av bildrutorna när videon återskapas. Med hjälp av ett FFmpeg-kommando skapas en videofil från de sorterade bildrutorna. I kommando anges exempelvis vilken bildfrekvens som ska genereras samt vilket format videofilen ska ha. I detta projekt skapas en mpeg4-fil. Slutligen rensas den temporära mappen för att frigöra utrymme och undvika läckage av resurser.

4.5 Skapande av en JSON-fil och överföring till AVIX

En enkel JSON-fil skapas med information om händelser och vid vilken tid skapades. I detta fall sparas vilken bildruta som händelsen inträffat i. Information om händelser och tid lades på i samma klasser som identifierade kroppsrörelser och objekt, Ett exempel på enkel JSON-fil återfinns i Fig. 4.3.

```
{
  "start_timestamp": "2024-03-28T12:00:00",
  "end_timestamp": "2024-03-28T12:30:00",
  "events": [
    {
      "name": "movement_X",
      "start_timestamp": "2024-03-28T12:05:00",
      "end_timestamp": "2024-03-28T12:10:00"
    }
  ]
}

```

Figur 4.3: Exempel på hur information om olika händelser och moment förs över jämte tidsmarkörer i ett JSON dokument

Resultatet när applikationen arbetat igenom ovanstående moment består av två delar, en sammanställd video och ett JSON-dokument med tidskoder kopplade till händelser från rörelse- och objektdetekteringen. I applikationen ges användaren möjlighet att markera fil som denne vill föra över till AVIX. Överföring sker sedan med AWS S3.

4.6 Testning

Den färdiga applikationen testades i en Android surfplatta. Tre videofilmer genererades om 5 sekunder, 30 sekunder och 45 sekunder. För var och en av dessa testades tre olika inställningar på bildfrekvenser: 5, 10 och 20 bilder per sekund. Antal sekunder att generera den slutliga videon noterades liksom slutlig videofils storlek.

Vid körning av applikationen sparades resultatet undan i en textfil med LaTex-kod för att underlätta införande av testresultat i rapporten.

5
Resultat

I detta kapitel redovisas resultatet av de olika delarna av projektet.

5.1 Resultat från litteraturstudierna

I planeringsrapporten sågs behov av att genomföra litteratursökningar inom ergonomi. När det gäller ergonomidelen hade dock företaget redan specificerad vilka rörelser som skulle noteras.

Från litteraturstudierna bestämde gruppen att använda Flutter och Dart för att ta fram applikationen och att sikta på en gemensam kodbas för iOS och Android.

I Flutter finns det dock plattformsspecifika inställningar som måste göras, exempelvis olika beroenden (dependencies) som behöver hanteras. Externa bibliotek och paket som är gemensamma för Android och iOS definieras i en gemensam inställningsfil. Inställningar som är plattformsspecifika hanteras på olika sätt i iOS och Android.

För att generera applikationen till Android krävs programmet Android Studio och för iOS behöver man en Mac-dator. Detta beror på att iOS-utvecklingsverktyg, inklusive Xcode och dess simulatorer, endast körs på MacOS. Då vi inte hade tillgång till en Mac dator färdigställdes applikationen endast för Android. En översiktlig beskrivning av hur en anpassning till iOS behöver göras finns i Appendix. A.

Som program att utveckla applikationen i valde vi Android Studio som är baserat på IntelliJ IDEA [15] och ger stöd för programmeringsspråken Java, Kotlin och med hjälp av ett tillägg, även Dart.

I den inledande litteraturstudien diskuterades också möjliga val av standardbibliotek där valde vi att använda Google ML-kit eftersom det i detta bibliotek fanns funktioner för ansiktsigenkänning, kroppsrörelse och objektdetektion. Dessa var också relativt enkla att implementera och gav önskade grundfunktioner.

Valda funktioner ur Google ML-kit

- Face detection
- Pose detection
- Object detection

Vi använder också FFmpegKit för att dela upp ursprunglig videofil i bildfiler och sätta samman dessa efter bearbetning till en slutlig videofil.

5.2 Realtid eller i efterhand

Gruppen försökte skapa en applikation som genomförde de olika beräkningarna i realtid, men det visar sig att i Dart och med ML-kit som valts i projektet, så var det inte möjligt att i realtid identifiera ansikten och anonymisera dessa med en oskarp mask. Den lösning som är möjlig är att först dela upp videon i enskilda bilder, därefter identifiera ansikten och skapa oskarp mask för att till slut sätta samman bilderna till en ny video. Vilket i princip är samma sak som att hantera beräkningarna i efterhand.

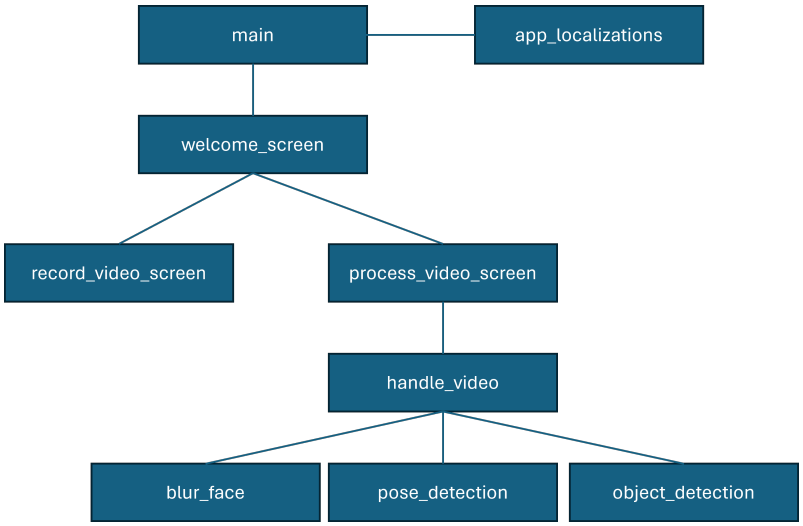
För att lösa hantering i realtid hade gruppen behövt utveckla plattformsspecifika lösningar vilket valts bort. Men det finns också andra skäl att gruppen valde att hantera inspelad video i efterhand.

När vi genomförde tester noteras att tiden det tar att generera en fil i efterhand är längre än längden på själva videofilen. Av prestandaskäl var det därför nödvändigt att bygga applikationen i flera steg där användaren först spelar in en video, och efter detta processer videofil tillsammans med JSON-fil.

Eftersom det inte fanns någon uppenbar fördel för uppdragsgivaren av att skapa en applikation som i realtid genomförde alla beräkningar fanns det heller ingen anledning att utmana ansatsen att först skapa en videofil och därefter ge användaren möjlighet att processa denna.

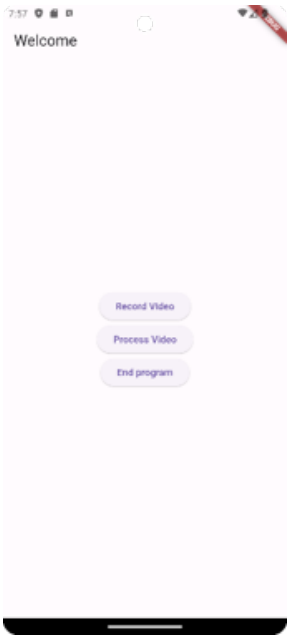
5.3 Beskrivning av framtagen applikation

Den slutliga applikationen byggs upp av ett antal klasser och klassdiagram återfinns i Fig. 5.1. För att hantera språkstöd körs app_localization igång samtidigt som huvudskärmen skapas i welcome_screen.



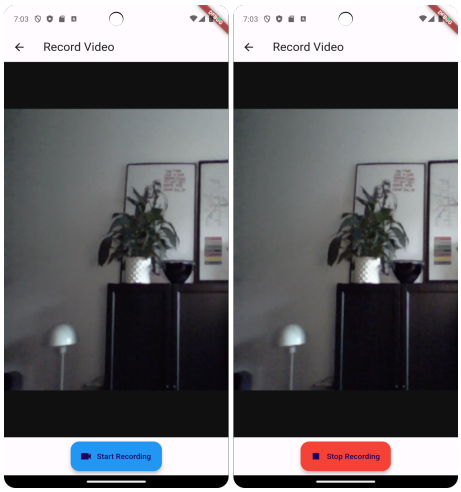
Figur 5.1: Klassdiagram för applikationen

Välkomstskärmen med de initiala valen spela in video (Record Video) och hantera video (Process Video).



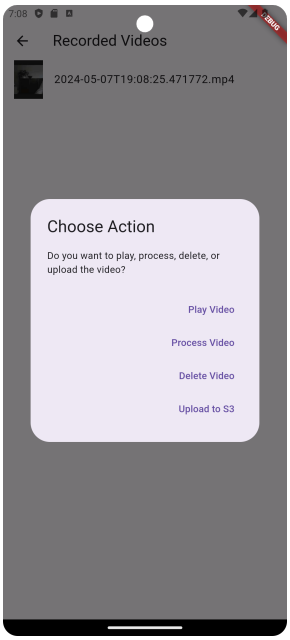
Figur 5.2: Välkomstskärmens olika valmöjligheter

Inspelning av videofilm sker i record_video_screen. Normal industristandard är 24 frames per sekund [16]. Hur användaren spelar in och stoppar inspelning visas i Fig. 5.3.



Figur 5.3: Här visas inspelningsfönstret där användaren först kan starta inspelningen genom att klicka på blå knapp, och sedan avsluta inspelningen via röd knapp

När användaren sedan vill processa inspelad videofilm görs det genom process_video_screen och de val användaren har visas i Fig. 5.4.



Figur 5.4: Efter det att användaren spelat in en video finns ett antal olika val, spela upp video, processa video, radera video eller ladda upp till Avix via AWS S3

5.4 Hantering av videofilen

De olika stegen som applikationen går igenom styrs av klassen handle_video. Först delas inspelad video upp i enskilda bildrutor. Applikationen använder FFmpeg för att extrahera bildrutor från inmatningsvideofilen. Rutorna sparas som JPEG-bilder i en specificerad katalog (outputDirectory). I processen kan antalet bilder per sekund som tas med specificeras. När videofilen är uppdelad kan ansikten identifieras och anonymiseras. Varje bildruta får ett eget nummer som anger ordningsföljden av bilder. Detta nummer används senare som tidskod.

Anonymisering sker i klassen blur_face. För att identifiera ansikten användes ML-kits ”face-detection” funktion som levererade koordinater och storlek på en box som omsluter identifierat ansikte. I Flutter används sedan en image-funktion som kan klippa ut den identifierade boxen, förse den med gaussisk oskärpa och sedan klistra in boxen i den ursprungliga filen igen. Resultatet blir en oskarp rektangel som följer ansiktet. När ansiktet är anonymiserat kan förinställda rörelser identifieras.

Identifiering av kroppsrörelser sker i pose_detection genom att ett antal nyckelpunkter/landmärken på kroppen identifieras. Dessa som finns beskrivna i Appendix B. Från dessa kan man i varje bildruta sätta upp regler för olika kroppsrörelser. Exempelvis kan man, om y-koordinaten för höger eller vänster hand är större än y-koordinaten för höger eller vänster axel skapa händelsen att man arbetar ovanför axlarna. Händelsen tillsammans med bildrutans ordningsnummer sparas sedan undan i JSON-dokumentet.

I applikationen har endast en rörelse implementerats, men det är enkelt att definiera flera rörelser som ska noteras. Händelsen Ärbeta över axelhöjdskapas exempelvis genom att beräkna om någon av händernas y-koordinater befinner sig över någon av axlarnas y-koordinater. På samma sätt kan en händelse skapas om någon av hand-

koordinaterna befinner sig under knäkoordinaterna om personen arbetar böjd. När olika kroppsställningar har identifierats ska nya delmoment noteras av applikationen genom att förinställda objekt detekteras.

Som tidigare redogjorts valde vi att använda object_detection i vår applikation för identifiera ett förvalt objekt. I det standardbibliotek vi använde ML-kit object detection, kan ett antal vardagligt förekommande objekt identifieras. I vår applikation har vi valt att notera ifall en blomma visas. Men i en mer utvecklad version kan man välja andra objekt för att markera att en ny monteringscykel påbörjats. Efter det att eventuella objekt detekteras ska de olika bildrutorna sättas samman till en videofil.

Detta är det sista steget där assembleFinalVideo-funktionen sätter ihop de bearbetade bildrutorna tillbaka till en video. Funktionen innebär att alla bearbetade filer med bildrutor från katalogen samlas in, sorteras och kopieras till en tillfällig katalog för att säkerställa att de är sekventiellt ordnade. Via ett FFmpeg-kommando kombineras dessa bilder sedan till en video, med angivande av bildhastighet och videocodecinställningar. Utdata sparas till en slutlig videofil. Efter sammansättning av videon raderas den tillfälliga katalogen som används för att hålla de sekventiella bildrutorna för att frigöra utrymme.

5.5 JSON-fil och överföring till AVIX via S3

JSON-filen skapas när användaren ger kommandot att processa en inspelad videofil. Olika händelser med angivande vilken ruta de uppstår i samt typ av händelse skapas i respektive klass.

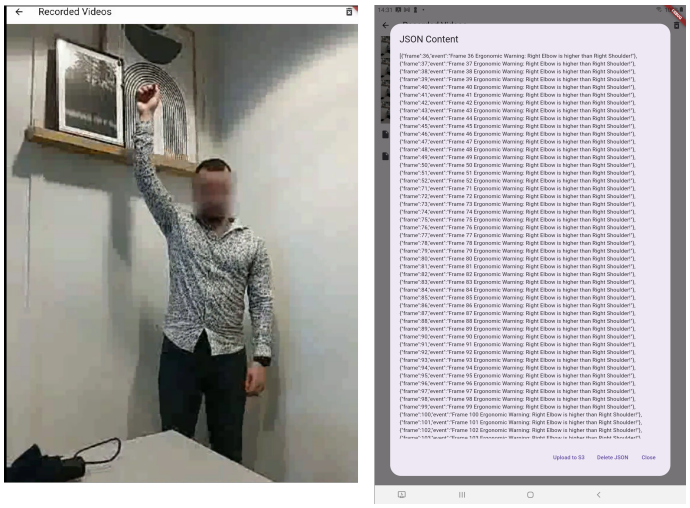
Metoden ”uploadVideo” tar en filadress som parameter och använder asynkron programmering för att hantera uppladdningen utan att blockera övriga funktioner. Den konfigurerar först en anslutning till S3-lagringen med nödvändiga autentiseringsuppgifter och en serveradress. Funktionen försöker sedan ladda upp filen till en specifik hink och objektnyckel. Om uppladdningen lyckas, visas ett bekräftelsemeddelande, men vid fel visas ett felmeddelande.

I applikationens kod är accessnycklar med mera, hårdkodade i applikationen. För att undvika säkerhetsrisker bör dock en skarp version hantera detta via så kallade miljövariabler.

5.6 Tester

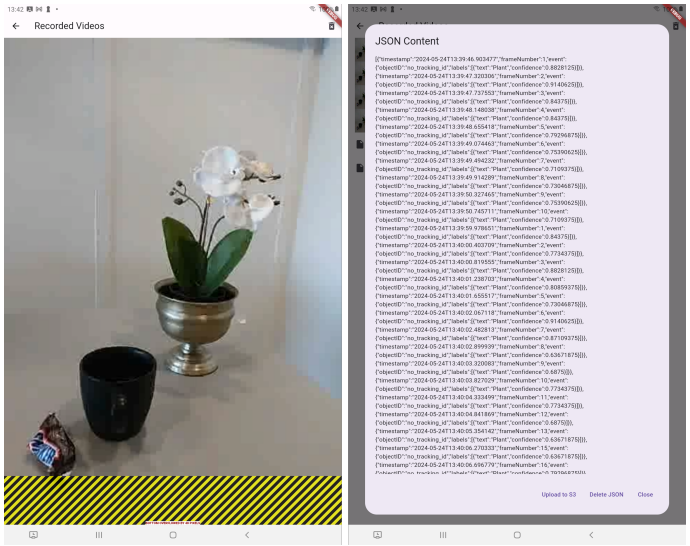
Applikationens funktionalitet verifierades i en ”androidpadda”. Det vill säga gruppen kontrollerade att en video kunde spelas in samt att applikationen identifierade kroppspositioner och objekt på ett korrekt sätt. Efter detta testkördes tre olika videofiler om 6, 32 respektive 42 sekunder och för vardera användes inställningarna 5, 10 respektive 20 bilder per sekund (fps) för att dela upp inspelad videofil i bildrutor.

I Fig. 5.5 nedan ser vi hur applikationen korrekt hanterar en person vars högra armbåge är över höger axel.



Figur 5.5: På bilden återfinns ett utsnitt ur en inspelad video där en person lyfter armen över huvudet och tillsammans en bild av den händelser som skapats i JSON-filen

Vidare i Fig. 5.6 noteras att applikationen också korrekt klarar av att identifierar en blomma som var det objekt som identifieras i applikationens kod. Även här noteras att applikationen korrekt adderar detta till JSON-filen.



Figur 5.6: På bilden återfinns ett utsnitt ur en inspelad video med en blomma (plant) och jämte bilden de händelser som skapats i JSON-filen

Först testades en originalfil som var 6 sekunder lång där resultatet återfinns i Tabell 5.1. Eftersom det var en kort originalvideo så är skillnaderna i filstorlek inte noterbara. Vad som däremot kan noteras är skillnaden i den tid det tog att generera en slutfil, där det gick dubbelt så fort att generera en fil med 20 bilder per sekund jämfört med 10 bilder per sekund. En hypotes som inte hann testas är att detta beror på interna processer i paddan som ”stör” applikationen.

fps	Tid att köra programmet	Filens storlek (MB)	Videofilens tid
5	00:00:21.932	0.21	00:00:06.020
10	00:00:41.171	0.21	00:00:06.020
20	00:01:20.648	0.23	00:00:06.020

Tabell 5.1: Resultat av videobearbetning med olika fps-inställningar, det vill säga 5, 10 och 20 bildrutor per sekund. Till höger ses den ursprungliga filens längd. Filens slutliga storlek ses i kolumnen ”Filens storlek” och den tid det tog för applikationen att generera en slutlig fil återfinns under ”Tid att köra programmet”.

Nästa fil som testades var en videofil som i original var 32 sekunder lång. Resultatet ses nedan i Tabell. 5.2. Det mest noterbara här är den tid det tar att generera en slutlig videofil. Vid inställningen 20 bilder per sekund tog det mer än sex gånger så lång tid att generera den slutliga filen i förhållande till originalfilens längd.

fps	Tid att köra programmet	Filens storlek (MB)	Videofilens tid
5	00:03:01.296	0.83	00:00:32.040
10	00:06:02.547	0.97	00:00:32.040
20	00:12:45.226	1.45	00:00:32.035

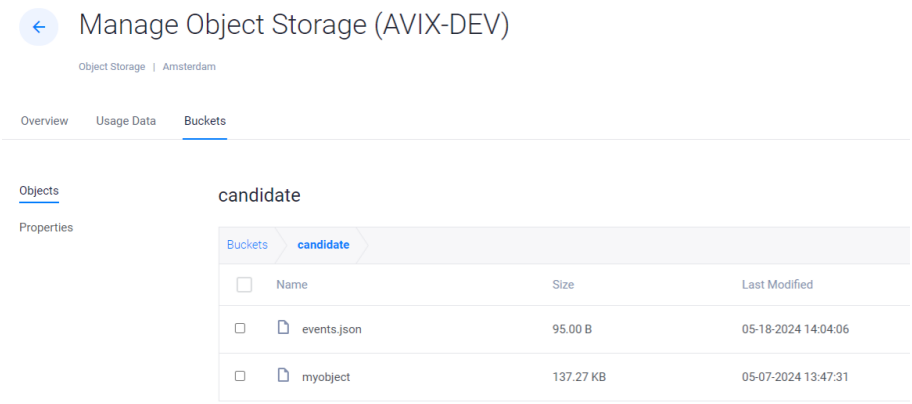
Tabell 5.2: Resultat av videobearbetning av en 32 sekunder lång videofil. Här ser vi att ju fler bilder per sekund vi har som inställning, desto längre tid tar det att generera en slutlig fil.

Det sista testet gjordes på en fil som var något längre, 42 sekunder. Resultatet i Tabell. 5.3 nedan visar liknande förhållande som i föregående test. Nämligen att det tar lång tid att generera en resulterade videofil från en relativt kort originalfil. Dock kan man notera att det gick något snabbare att hantera en 42 sekunder lång fil jämfört med den 32 sekunder långa videofil som användes i föregående test. Även här kan en hypotes ställas upp att detta beror på interna processer i använd padda.

fps	Tid att köra programmet	Filens storlek (MB)	Videofilens tid
5	00:02:31.956	1.05	00:00:42.020
10	00:04:54.275	1.09	00:00:42.020
20	00:09:50.848	1.15	00:00:42.025

Tabell 5.3: Resultat av videobearbetning av en videofil som i original är 42 sekunder lång. Noterbart är att denna fil gick snabbare att hantera och tiden att generera en slutlig video är något kortare än filen som var 32 sekunder lång i original

Avslutningsvis verifierades att skapad videofil och JSON-fil laddats över till AVIX via S3 på ett korrekt sätt vilket visas i Figur 5.7 nedan.



Figur 5.7: Här ser vi hur videofil och JSON-fil laddats över till AVIX via AWS S3

Fyra stycken resultat från ovanstående tester

- Anonymisering av ansikte, identifiering av kroppsrörelser och objekt, generering av slutlig videofil samt uppladdning till S3 fungerar i applikationen.
- Generellt sett tar det relativt lång tid att generera en slutlig video från en original video om strax under en minut lång. Särskilt om man eftersträvar godtagbar videokvalitet vilket 20 bilder per sekund kan antas ha.
- Storleken på den resulterande videofilen påverkas inte nämnvärt av antalet bilder per sekund i den slutliga videon. Om sambandet är linjärt skulle en 30 minuter lång video resultera i en 30 MB stor slutlig videofil vilket inte borde vara ett problem att lagra i en modern mobiltelefon.
- Någonting påverkar den tid det tar att generera slutlig videofil utöver applikationens olika programsteg. Det ses dels i skillnaden som uppstår i det första testet, men också i skillnaden mellan andra och tredje testet.

6
Diskussion och slutsatser

Här diskuteras resultatet och erfarenheter från genomförande av projektet. Sist resoneras kring etiska och miljömässiga överväganden.

6.1 Valda metoder

Beslutet att utveckla en applikation med en gemensam kodbas var som vi ser det korrekt utifrån att applikationen ska fungera som ett hjälpmedel i en befintlig tjänst som företaget tillhandahåller. Det valet innebar dock att viss funktionalitet inte gick att implementera. Exempelvis bättre hantering av anonymiseringen av ansikten. Nu läggs en rektangel över ansiktet med gaussisk oskärpa. Det hade varit mer estetiskt tilltalande om oskärpan följde ansiktsformen.

När det gäller valet av standardbibliotek stötte vi på problem med att implementera TensorFlow Lite. Eftersom ML-kit gav tillräckligt stöd för utveckling av applikationen valdes detta. Det visade sig också att paketet var enklare att installera.

Att inte välja en agil projektmodell fungerade bra i det här projektet då vi uppfattade att uppdragsgivaren hade en klar bild av vad slutprodukten skulle vara, samt att vi i planeringsrapporten arbetat igenom en utvecklingsprocess som vi följde i princip helt och hållet. Några delmoment utvecklades iterativt. Exempelvis tog vi först fram en metod som delade upp originalfilen, därefter la vi på kod för att anonymisera ansiktet och sammanställa slutlig videofil. Efter detta adderades kod för kroppsrörelsedetektion och sist koden för objektdetektering. I varje steg verifierades att applikationen fungerade med installerad funktionalitet.

6.2 Genomförande och resultat

Inledningsvis ställdes följande frågor som projektet skulle besvara:

- Är det möjligt att skapa en applikation som fungerar för både iOS och Android med nedanstående funktionalitet?
- Är det möjligt satt använda standardbibliotek för att:
  - a) skapa oskarp mask över ansikte,
  - b) markera rörelser med så kallad rörelsedetektion
  - c) särskilja monteringscykler genom att identifiera förutbestämda objekt
- Är det möjligt att markera ovanstående i videon i realtid eller behövs ett särskilt moment på en sparad video?

- Är det möjligt att överföra video och information om rörelser och monteringscykler till AVIX

I princip kan gruppen efter genomfört projekt svara ja på alla dessa frågor, utom särskilja monteringscykler via objektdetektion. Vi lyckades förvisso implementera objektdetektering, men de objekt som kan detekteras i standarduppsättningen av ML-kit räcker inte för den funktion som uppdragsgivaren eftersträvar. Detta är den största begränsningen i den slutliga applikationen utifrån uppdragsgivarens inledande önskemål.

För att få en fullt fungerande applikation med denna funktion behöver man träna egna modeller via exempelvis TensorFlow. Detta är dock ett tidskrävande arbete och en förutsättning är att man får tillgång till kundernas inspelade videofiler. Alternativt kan man nöja sig med att låta den som spelar in videon manuellt markera delmoment och ny monteringscykel.

När det gäller kroppsrörelsedetektion så fungerar rutinen bra och det går att lägga till regler för fler ergonomiskt utmanande positioner med samma metodik som vi gjort. Det vill säga sätta upp regler för hur olika kroppsdelars koordinater förhåller sig till varandra.

Som vi beskrivit ovan så ser vi ingen direkt fördel med att generera slutresultatet i realtid. Genom att låta applikationen hantera de olika uppgifterna i efterhand minskar förmodligen kraven på den mobiltelefon som används, vilket ökar användbarheten.

6.3 Förbättringar och framtida utveckling

Följande områden för framtida utveckling har identifierats:

- Applikationen behöver utvecklas färdig även för användning under IOS.
- Det är tydligt att applikationens kod behöver optimeras främst för att ta ned tiden det tar att generera en längre video. Detta är viktigt eftersom inspelningar enligt uppgift från uppdragsgivaren kan vara uppemot 30 minuter långa.
- I vår kod behandlas anonymisering av ansikte, kroppsrörelsedetektion, objektdetektion och sammanslagning av slutlig videofil seriellt. Det är möjligt att en del av dessa metoder skulle kunna exekveras parallellt för att få ned den tid det tar att hantera en videofil.
- Vidare kan man testa andra standardbibliotek som exempelvis TensorFlow Lite eller se om koden kan effektiviseras genom att utveckla delar plattformsspecifikt. Vidare undersökningar och tester behövs för att kunna veta vilka andra val som kan förbättra applikationen.
- Ytterligare en väg kan vara att processa videon i en extern miljö, men denna eventuella lösning har legat utanför projektets syfte.
- Den process vi valt att använda objektdetektering för att identifiera nya monteringscykler och delmoment levererar inte önskat resultat i den meningen att de objekt som kan detekteras inte motsvarar de objekt som skulle behöva observeras i skarp produktion. Anledningen till detta är att exempelvis en kaross på en bil inte nödvändigtvis identifieras som en bil. Vi ser två sätt att hantera detta, antingen tränar man själv upp applikationen på de olika fall som

AVIX hanterar eller så inför man en möjlighet för den operatör som spelar in originalvideo att markera olika delar manuellt
- Integriteten för dem som blir inspelade kan behöva ses över, dels vad gäller tillstånd att låta sig spelas in, säkerhet i hantering av videofiler vad avser lagring och spridning samt säkerhet vid överföring till AVIX.

6.4 Lärdomar

Gruppen la ner mycket tid på att ta fram en heltäckande planeringsrapport och de olika delmomenten följdes under hela applikationsutvecklingen. Det gjorde arbetet relativt enkelt.

Det är väldigt tydligt vilken trade-off man som utvecklare gör när man väljer att utveckla en applikation i en gemensam kodbas. Ett exempel på detta är bildbehandlingsmöjligheterna i som beskrivits tidigare. För att komma runt detta måste i dagsläget dessa funktioner utvecklas i andra plattformsspecifika programmeringsspråk, Kotlin för Android och Swift för iOS.

De förtränade modellerna som har använts har begränsningar, särskilt när det gäller att identifiera objekt. Uppdragsgivaren behöver kunna identifiera olika delmoment och monteringscykler mer flexibelt. Detta innebär att användningen av standardbibliotek för maskininlärning inte är tillräcklig.

6.5 Etiska och miljömässiga överväganden

Applikationen kan användas till att förbättra ergonomin vid manuella monteringslinjer och det ansluter till Mål 8 i FNs Globala mål: “Verka för varaktig, inkluderande och hållbar ekonomisk tillväxt, full och produktiv sysselsättning med anständiga arbetsvillkor för alla”.

Om applikationen används till att effektivisera det manuella arbetet kan det leda till att arbetstillfällen försvinner. Men samtidigt kan det innebära färre förslitningsskador bland de som utför arbetet samt effektivare arbetssätt vilket kan leda till säkrare anställningar för övriga anställda. Ur ett konsekvensetiskt förhållningssätt skulle förändringen därför vara motiverad.

Eftersom personer spelas in under arbete är det viktigt att de är medvetna om hur materialet kommer att användas, hur det lagras, vem som har tillgång till det, och hur det raderas. En del av detta är inbyggt i Android, då användaren måste godkänna att applikationen använder mobilens videokamera. Funktionen som anonymiserar ansiktet på den inspelade personen är också en viktig del av detta.

Avslutningsvis. Applikationen använder ML-kit vilket är ett standardbibliotek som kan användas fritt kommersiellt. Lämplig copyrighttext om detta bör läggas in i den slutliga koden för applikationen.

Litteraturförteckning

[1] AVIX Suite. (2024) AVIX översikt. [Hämtad: [2024-05-26]]. [Online]. Available: https://www.avix.eu/sv/anvandningsomraden/avix-oversikt/

[2] R. F. Bușer, "Native or cross-platform mobile applications development," *Young Economists Journal / Revista Tinerilor Economisti*, vol. 20, no. 40, pp. 55–62, Apr. 2023.

[3] Dart Team. Dart Overview. https://dart.dev/overview. Hämtad: [2024-05-05].

[4] Flutter Team. Flutter FAQ. https://docs.flutter.dev/resources/faq. Hämtad: [2024-05-05].

[5] S. S. Skiena, *The Data Science Design Manual*, ser. Texts in Computer Science. Cham, Switzerland: Springer, 2017.

[6] I. Condés, E. Perdices, J. Fernández-Conde, and J. Cañas, "Robust person identification and following in a mobile robot based on deep learning and optical tracking," *Electronics (Switzerland)*, vol. 12, no. 21, November 2023, page 4.

[7] Flutter Community. ffmpeg_kit_flutter. https://pub.dev/packages/ffmpeg_kit_flutter. Hämtad: [2024-05-01].

[8] JSON.org, "JSON (JavaScript Object Notation)," https://www.json.org/json-sv.html, 2023, hämtad [2024-04-03].

[9] Amazon Web Services. Amazon Simple Storage Service (S3). https://aws.amazon.com/pm/serv-s3/. [Hämtad: [2024-05-09]].

[10] E. M. M. Alzeyani and C. Szabó, "A study on the methodology of software project management used by students whether they are using an agile or waterfall methodology," in *2022 20th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, Nov 2022, pp. 22–27.

[11] CircleCI, "Native vs. Cross-Platform Mobile Development," https://circleci.com/blog/native-vs-cross-platform-mobile-dev/, 2023, [Hämtad 2024-04-12].

[12] A. Bădică and R. F. Bușe, "Cross-platform development tools: React-native versus flutter," *Young Economists Journal / Revista Tinerilor Economisti*, vol. 20, no. 40, pp. 107–115, April 2023.

[13] P. Nawrocki, K. Wrona, M. Marczak, and B. Sniezynski, "A comparison of native and cross-platform frameworks for mobile applications," *Computer*, vol. 54, no. 3, pp. 18–27, March 2021.

[14] M. G. Sarwar Murshed, C. Murphy, D. Hou, N. Khan, G. Ananthanarayanan, and F. Hussain, "Resource-constrained IoT devices: Machine learning at the edge," *ACM Computing Surveys*, vol. 54, no. 8, pp. 22–25, November 2022.

[15] JetBrains. Discover IntelliJ IDEA. https://www.jetbrains.com/help/idea/discover-intellij-idea.html. Hämtad: [2024-05-05].

[16] Dacast. (2024) Understanding fps values: The advanced guide to video frame rates. https://www.dacast.com/blog/frame-rate-fps/. [Hämtad: [2024-05-21]].

A

Åtgärder för att få applikationen att fungera i iOS - översiktlig beskrivning

A.1 Lägg till iOS-specifika behörigheter

iOS kräver specifika behörigheter för filåtkomst och kameraanvändning som måste definieras i `Info.plist`-filen. Man bör lägga till följande i sin `Info.plist` för att tillåta applikationen att använda dokumentkatalogen och kamera:

```
<key>NSCameraUsageDescription</key>
<string>This app needs access to the camera to take photos.</string>
<key>NSPhotoLibraryUsageDescription</key>
<string>This app needs access to the photo library for saving and managing your
    photos.</string>
<key>NSMicrophoneUsageDescription</key>
<string>This app needs access to microphone for capturing audio with video.</
    string>
<key>NSDocumentsFolderUsageDescription</key>
<string>This app needs access to documents to read and write files.</string>
<key>UIBackgroundModes</key>
<array>
    <string>fetch</string>
    <string>processing</string>
</array>
```

A.2 Använd rätt bibliotek för iOS

`flutter_ffmpeg`-paketen har olika varianter beroende på vilka funktioner man behöver. För att säkerställa att du har rätt stöd för både Android och iOS bör man välja en variant som stöder både plattformarna effektivt. Det kan vara bra att använda `full` eller `full-gpl` versionen för att få tillgång till alla codecs och funktioner:

```
dependencies:
  flutter_ffmpeg:
    git:
      url: https://github.com/tanersener/flutter-ffmpeg.git
      ref: full-gpl      # Använd 'full' eller 'full-gpl' för maximal
    funktionalitet
```

A.3 Anpassa koden för iOS-specifika funktioner

När man arbetar med filer på iOS, måste mqn säkerställa att alla filvägar och hanteringar är kompatibla med iOS:s säkerhetsbegränsningar. iOS-applikationer har striktare regler för filåtkomst, så man bör verifiera att sina sökvägar och metodanrop är korrekta och tillåtna på plattformen. Se över eventuella filåtkomstproblem speciellt när man använder externa bibliotek som FFmpeg.

A.4 Debugging och testning på en verklig enhet

iOS-simulatorer stöder inte alla funktioner som video- och audiobehandling, så det är viktigt att man testar sin applikation på en verklig enhet för att säkerställa att allt fungerar som det ska, speciellt för mediahantering.

A.5 Säkerställ att koden hanterar iOS-unika fall

Ibland kan vissa koddelar eller biblioteksanrop behöva anpassas för iOS. Till exempel kan vissa FFmpeg-kommandon som fungerar på Android behöva modifieras för iOS på grund av olika stödda codecs eller funktioner.

B

Nyckelpunkter i kroppställningsdetektion

```
leftShoulder = pose.getPoseLandmark(PoseLandmark.LEFT)
rightShoulder = pose.getPoseLandmark(PoseLandmark.RIGHT)
leftElbow = pose.getPoseLandmark(PoseLandmark.LEFT)
rightElbow = pose.getPoseLandmark(PoseLandmark.RIGHT)
leftWrist = pose.getPoseLandmark(PoseLandmark.LEFT)
rightWrist = pose.getPoseLandmark(PoseLandmark.RIGHT)
leftHip = pose.getPoseLandmark(PoseLandmark.LEFT)
rightHip = pose.getPoseLandmark(PoseLandmark.RIGHT)
leftKnee = pose.getPoseLandmark(PoseLandmark.LEFT)
rightKnee = pose.getPoseLandmark(PoseLandmark.RIGHT)
leftAnkle = pose.getPoseLandmark(PoseLandmark.LEFT)
rightAnkle = pose.getPoseLandmark(PoseLandmark.RIGHT)
leftPinky = pose.getPoseLandmark(PoseLandmark.LEFT)
rightPinky = pose.getPoseLandmark(PoseLandmark.RIGHT)
leftIndex = pose.getPoseLandmark(PoseLandmark.LEFT)
rightIndex = pose.getPoseLandmark(PoseLandmark.RIGHT)
leftThumb = pose.getPoseLandmark(PoseLandmark.LEFT)
rightThumb = pose.getPoseLandmark(PoseLandmark.RIGHT)
leftHeel = pose.getPoseLandmark(PoseLandmark.LEFT)
rightHeel = pose.getPoseLandmark(PoseLandmark.RIGHT)
leftFootIndex = pose.getPoseLandmark(PoseLandmark.LEFT)
rightFootIndex = pose.getPoseLandmark(PoseLandmark.RIGHT)
nose = pose.getPoseLandmark(PoseLandmark.NOSE)
leftEyeInner = pose.getPoseLandmark(PoseLandmark.LEFT)
leftEye = pose.getPoseLandmark(PoseLandmark.LEFT)
leftEyeOuter = pose.getPoseLandmark(PoseLandmark.LEFT)
rightEyeInner = pose.getPoseLandmark(PoseLandmark.RIGHT)
rightEye = pose.getPoseLandmark(PoseLandmark.RIGHT)
rightEyeOuter = pose.getPoseLandmark(PoseLandmark.RIGHT)
leftEar = pose.getPoseLandmark(PoseLandmark.LEFT)
rightEar = pose.getPoseLandmark(PoseLandmark.RIGHT)
leftMouth = pose.getPoseLandmark(PoseLandmark.LEFT)
rightMouth = pose.getPoseLandmark(PoseLandmark.RIGHT)
```

INSTITUTIONEN FÖR DATA- OCH INFORMATIONSTEKNIK
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige
www.chalmers.se

GÖTEBORGS
UNIVERSITET

CHALMERS